Apophenia

Thu Jul 2 2015 12:33:07

Contents

1 Welcome

Apophenia is an open statistical library for working with data sets and statistical models. It provides functions on the same level as those of the typical stats package (such as OLS, Probit, or singular value decomposition) but gives the user more flexibility to be creative in model-building. The core functions are written in C, but experience has shown them to be easy to bind to in Python/Julia/Perl/Ruby/&c.

It is written to scale well, to comfortably work with gigabyte data sets, million-step simulations, or computationally-intensive agent-based models.

The goods

The library has been growing and improving since 2005, and has been downloaded well over 1e4 times. To date, it has over two hundred functions and macros to facilitate statistical computing, such as:

- OLS and family, discrete choice models like Probit and Logit, kernel density estimators, and other common models.
- Functions for transforming models (like Normal $$ truncated Normal) and combining models (produce the cross-product of that truncated Normal with three others, or use Bayesian updating to combine that cross-product prior with an OLS likelihood to produce a posterior distribution over the OLS parameters).
- Database querying and maintenance utilities.
- Data manipulation tools for splitting, stacking, sorting, and otherwise shunting data sets.
- Moments, percentiles, and other basic stats utilities.
- t-tests, F-tests, et cetera.
- Several optimization methods available for your own new models.
- It does *not* re-implement basic matrix operations or build yet another database engine. Instead, it builds upon the excellent `GNU Scientific` and `SQLite` libraries. MySQL/mariaDB is also supported.

For the full list of macros, functions, and prebuilt models, check the `index`.

`Download Apophenia here`.

Most users will want to download the latest packaged version linked from the `Download Apophenia here` header.

Those who would like to work on a cutting-edge copy of the source code can get the latest version by cutting and pasting the following onto the command line. If you follow this route, be sure to read the development README in the `apophenia` directory this command will create.

```
git clone https://github.com/b-k/apophenia.git
```

The documentation

To start off, have a look at this Gentle Introduction to the library.

`The outline` gives a more detailed narrative.

The `index` lists every function in the library, with detailed reference information. Notice that the header to every page has a link to the outline and the index.

To really go in depth, download or pick up a copy of `Modeling with Data`, which discusses general methods for doing statistics in C with the GSL and SQLite, as well as Apophenia itself. *`A Useful Algebraic`*

System of Statistical Models (PDF) discusses some of the theoretical structures underlying the library.

There is a wiki with some convenience functions, tips, and so on.

Notable features Much of what Apophenia does can be done in any typical statistics package. The apop_data element is much like an R data frame, for example, and there is nothing special about being able to invert a matrix or take the product of two matrices with a single function call (apop_matrix_inverse and apop_dot, respectively). Even more advanced features like Loess smoothing (apop_loess) and the Fisher Exact Test (apop_test_fisher_exact) are not especially Apophenia-specific. But here are some things that are noteworthy.

- It's a C library! You can build applications using Apophenia for the data-processing back-end of your program, and not worry about the overhead associated with scripting languages. For example, it is currently used in production for certain aspects of processing for the U.S. Census Bureau's American Community Survey. And the numeric routines in your favorite scripting language typically have a back-end in plain C; perhaps Apophenia can facilitate writing your next one.
- The apop_model object allows for consistent treatment of distributions, regressions, simulations, machine learning models, and who knows what other sorts of models you can dream up. By transforming and combining existing models, it is easy to build complex models from simple sub-models.
- For example, the apop_update function does Bayesian updating on any two well-formed models. If they are on the table of conjugates, that is correctly handled, and if they are not, an appropriate variant of MCMC produces an empirical distribution. The output is yet another model, from which you can make random draws, or which you can use as a prior for another round of Bayesian updating. Outside of Bayesian updating, the apop_model_metropolis function is good for approximating other complex models.
- The maximum likelihood system combines several subsystems into one form: it will do a few flavors of conjugate gradient search, Nelder-Mead Simplex, Newton's Method, or Simulated Annealing. You pick the method by a setting attached to your model. If you want to use a method that requires derivatives and you don't have a closed-form derivative, the ML subsystem will estimate a numerical gradient for you. If you would like to do EM-style maximization (all but the first parameter are fixed, that parameter is optimized, then all but the second parameter are fixed, that parameter is optimized, ..., looping through dimensions until the change in objective across cycles is less than `eps`), add a settings group specifying the tolerance at which the cycle should stop: `Apop_settings_add_group(your_model, apop_mle, .dim_cycle_tolerance=eps)`.
- The Iterative Proportional Fitting algorithm, apop_rake, is best-in-breed, designed to handle large, sparse matrices.

Contribute!

- Develop a new model object.
- Contribute your favorite statistical routine.
- Package Apophenia into an RPM, portage, cygwin package.
- Report bugs or suggest features.
- Write bindings for your preferred language. For example, here are a `Perl wrapper` and early versions of `a Julia wrapper` and `an R wrapper` which you could expand upon.

If you're interested, `write to the maintainer` (Ben Klemens), or join the `GitHub` project.

2 Getting started

If you are entirely new to Apophenia, have a look at the Gentle Introduction here.

As well as the information in this outline, there is a separate page covering the details of setting up a computing environment and another page with some sample code for your perusal.

References and extensions

2.1 A quick overview

This is a "gentle introduction" to the Apophenia library. It is intended to give you some initial bearings on the typical workflow and the concepts and tricks that the manual pages assume you are familiar with.

If you want to install Apophenia now so you can try the samples on this page, see the Setting up page.

An outline of this overview:

- Apophenia fills a space between traditional C libraries and stats packages.
- The apop_data structure represents a data set (of course). Data sets are inherently complex, but there are many functions that act on apop_data sets to make life easier.
- The apop_model encapsulates the sort of actions one would take with a model, like estimating model parameters or predicting values based on new inputs.
- Databases are great, and a perfect fit for the sort of paradigm here. Apophenia provides functions to make it easy to jump between database tables and apop_data sets.

The opening example

Setting aside the more advanced applications and model-building tasks, let us begin with the workflow of a typical fitting-a-model project using Apophenia's tools:

- Read the raw data into the database using apop_text_to_db.
- Use SQL queries handled by apop_query to massage the data as needed.
- Use apop_query_to_data to pull some of the data into an in-memory apop_data set.
- Call a model estimation such as

  ```
  apop_estimate (data_set, apop_ols)
  ```

  or

  ```
  apop_estimate (data_set, apop_probit)
  ```

  to fit parameters to the data. This will return an apop_model with parameter estimates.
- Interrogate the returned estimate, by dumping it to the screen with apop_model_print, sending its parameters and variance-covariance matrices to additional tests (the `estimate` step runs a few for you), or send the model's output to be input to another model.

Here is an example of most of the above steps which you can compile and run, to be discussed in detail below.

The program relies on the U.S. Census's American Community Survey public use microdata for DC 2008, which you can get from the command line via:

```
wget https://raw.github.com/rodri363/tea/master/demo/ss08pdc.csv
```

or by pointing your browser to that address and saving the file.

The program:

```
#include <apop.h>

int main(){
    apop_text_to_db(.text_file="ss08pdc.csv", .tabname="dc");
    apop_data *data = apop_query_to_data("select log(pincp+10) as log_income,
       agep, sex "
                    "from dc where agep+ pincp+sex is not null and pincp>=0");
    apop_model *est = apop_estimate(data, apop_ols);
    apop_model_print(est);
}
```

If you saved the code to `census.c` and don't have a Makefile or other build system, then you can compile it with

```
gcc census.c -std=gnu99 -lapophenia -lgsl -lgslcblas -lsqlite3 -o census
```

or

```
clang census.c -lapophenia -lgsl -lgslcblas -lsqlite3 -o census
```

and then run it with `./census`. This compile line will work on any system with all the requisite tools, but for full-time work with this or any other C library, you will probably want to write a Makefile.

The results are unremarkable—age has a positive effect on income, and sex (1=male, 2=female) does has a negative effect—but it does give us some lines of code to dissect.

The first two lines in `main()` make use of a database. I'll discuss the value of the database step more at the end of this page, but for now, note that there are several functions, apop_query and apop_query_to_data being the ones you will most frequently be using, that will allow you to talk to and pull data from either an SQLite or mySQL/mariaDB database. The database is a natural place to do data processing like renaming variables, selecting subsets, and transforming values.

Designated initializers

Like this line,

```
apop_text_to_db(.text_file="data", .tabname="d");
```

many Apophenia functions accept named, optional arguments. To give another example, the apop_data set has the usual row and column numbers, but also row and column names. So you should be able to refer to a cell by any combination of name or number; for the data set you read in above, which has column names, all of the following work:

```
x = apop_data_get(data, 2, 3); //observation 2, column 3
x = apop_data_get(data, .row=2, .colname="sex"); // same
apop_data_set(data, 2, 3, 1);
apop_data_set(data, .colname="sex", .row=2, .val=1);
```

Default values mean that the apop_data_get, apop_data_set, and apop_data_ptr functions handle matrices, vectors, and scalars sensibly:

```
//Let v be a hundred-element vector:
apop_data *v = apop_data_alloc(100);
```

```
[fill with data here]
double x1 = apop_data_get(v, 10);
apop_data_set(v, 2, .val=x1);

//A 100x1 matrix behaves like a vector
apop_data *m = apop_data_alloc(100, 1);
[fill with data here]
double m1 = apop_data_get(v, 1);

//let s be a scalar stored in a 1x1 apop_data set:
apop_data *v = apop_data_alloc(1);
double *scalar = apop_data_ptr(s);
```

These conveniences may be new to users of less user-friendly C libraries, but it it fully conforms to the C standard (ISO/IEC 9899:2011). See the Designated initializers page for details.

2.1.1 apop_data

A lot of real-world data processing is about quotidian annoyances about text versus numeric data or dealing with missing values, and the apop_data set and its many support functions are intended to make data processing in C easy. Some users of Apophenia use the library only for its apop_data set and associated functions. See Data sets for extensive notes on using the structure.

The structure includes seven parts:

- a vector,
- a matrix,
- a grid of text elements,
- a vector of weights,
- names for everything: row names, a vector name, matrix column names, text names,
- a link to a second page of data, and
- an error marker

This is not a generic and abstract ideal, but is the sort of mess that real-world data sets look like. For example, here is some data for a weighted OLS regression. It includes an outcome variable in the vector, dependent variables in the matrix and text grid, replicate weights, and column names in bold labeling the variables:

Rowname	Vector	Matrix			Text		Weights
	Outcome	**Age**	**Weight (kg)**	**Height (cm)**	**Sex**	**State**	
"Steven"	1	32	65	175	Male	Alaska	1
"Sandra"	0	41	61	165	Female	Alabama	3.2
"Joe"	1	40	73	181	Male	Alabama	2.4

Apophenia's functions generally assume that one row across all of these elements describes a single observation or data point.

See above for some examples of getting and setting individual elements.

Also, apop_data_get, apop_data_set, and apop_data_ptr consider the vector to be the -1st column, so using the data set in the figure, `apop_data_get(sample_set, .row=0, .col=-1) == 1`.

Reading in data

As per the example above, use apop_text_to_data or apop_text_to_db and then apop_query_to_data.

Subsets

There are many macros to get views of subsets of the data. Each generates a disposable wrapper around the base data: once the variable goes out of scope, the wrapper disappears, but modifications made to the data in the view are modifications to the base data itself.

```
#include <apop.h>

int main(){
    apop_table_exists("data", 'd');
    apop_data *d = apop_text_to_data("data");

    //tally row zero of the data set's matrix by viewing it as a vector:
    gsl_vector *one_row = Apop_rv(d, 0);
    double sigma = apop_vector_sum(one_row);
    printf("Sum of row zero: %g\n", sigma);
    assert(sigma==14);

    //view column zero as a vector; take its mean
    double mu = apop_vector_mean(Apop_cv(d, 0));
    printf("Mean of col zero: %g\n", mu);
    assert(fabs(mu - 19./6)<1e-5);

    //get a sub-data set (with names) of two rows beginning at row 3; print to screen
    apop_data *six_elmts = Apop_rs(d, 3, 2);
    apop_data_print(six_elmts);
}
```

All of these slicing routines are macros, because they generate several background variables in the current scope (something a function can't do). Traditional custom is to put macro names in all caps, like `APOP_DATA_ROWS`, which to modern sensibilities looks like yelling. The custom has a logic: there are ways to hang yourself with macros, so it is worth distinguishing them typographically. Apophenia tones it down by capitalizing only the first letter.

Basic manipulations

See Data sets for a list of many other manipulations of data sets, such as apop_data_listwise_delete for quick-and-dirty removal of observations with `NaN`s, apop_data_split / apop_data_stack, or apop_data_sort to sort all elements by a single column.

Apply and map

If you have an operation of the form *for each element of my data set, call this function*, then you can use apop_map to do it. You could basically do everything you can do with an apply/map function via a `for` loop, but the apply/map approach is clearer and more fun. Also, if you set OpenMP's `omp_set_num_threads(N)` for any `N` greater than 1 (the default on most systems is the number of CPU cores), then the work of mapping will be split across multiple CPU threads. See Map/apply for a number of examples.

Text

String handling in C usually requires some tedious pointer and memory handling, but the functions to put strings into the text grid in the apop_data structure and get them out again will do the pointer shunting for you. The apop_text_alloc function is really a realloc function: you can use it to resize the text grid as necessary. The apop_text_set function will write a single string to the grid, though you may be using apop_query_to_text or apop_query_to_mixed_data to read in an entire data set at once. Functions that act on entire data sets, like apop_data_rm_rows, handle the text part as well.

The text grid for `your_data` has `your_data->textsize[0]` rows and `your_data->textsize[1]` columns. If you are using only the functions to this point, then empty elements are a blank string (`""`), not `NULL`. For reading individual elements, refer to the (i, j)th text element via `your_data->text[i][j]`.

Errors

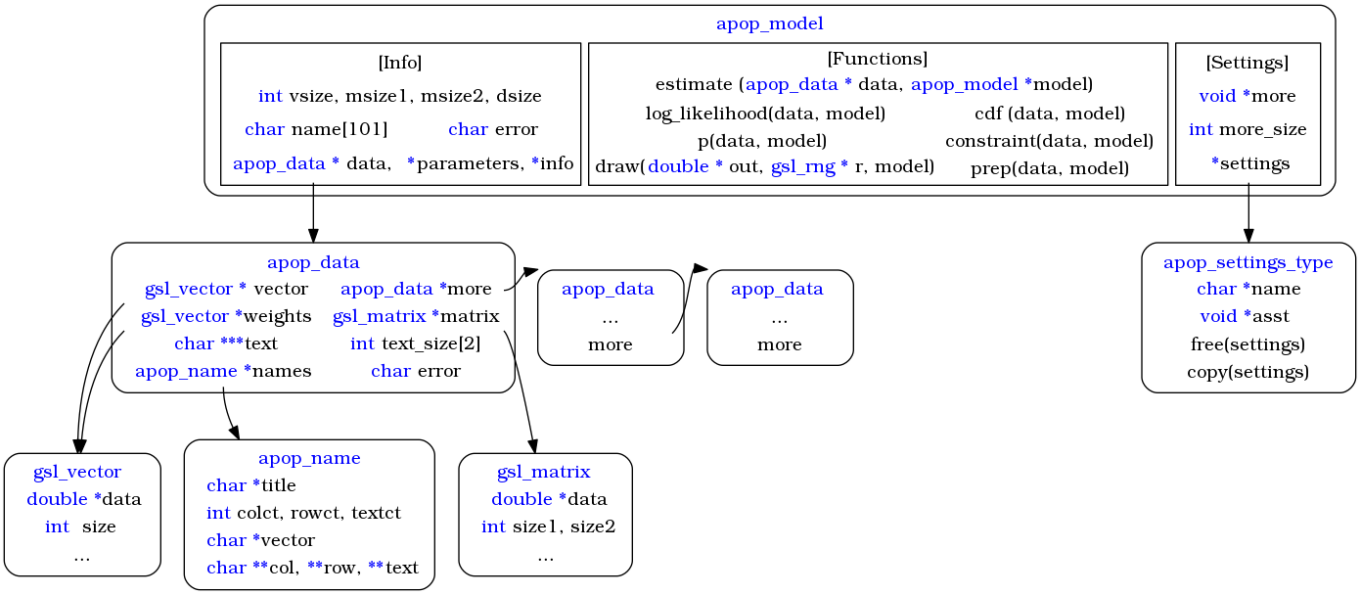
Many functions will set the `error` element of the apop_data structure being operated on if anything goes

wrong. You can use this to halt the program or take corrective action:

```
apop_data *the_data = apop_query_to_data("select * from d");
Apop_stopif(!the_data || the_data->error, exit(1), 0, "Trouble querying the data");
```

The whole structure

Here is a diagram of all of Apophenia's structures and how they relate. It is taken from this cheat sheet on general C and SQL use (2 page PDF).



All of the elements of the apop_data structure are laid out at middle-left. You have already met the vector, matrix, weights, and text grid.

The diagram shows the apop_name structure, which has received little mention so far because names basically take care of themselves. A query will bring in column names (and row names if you set `apop_opts.db_name_column`), or use apop_data_add_names to add names to your data set and apop_name_stack to copy from one data set to another.

The apop_data structure has a `more` element, for when your data is best expressed in more than one page of data. Use apop_data_add_page, apop_data_rm_page, and apop_data_get_page. Output routines will sometimes append an extra page of auxiliary information to a data set, such as pages named `<Covariance>` or `<Factors>`. The angle-brackets indicate a page that describes the data set but is not a part of it (so an MLE search would ignore that page, for example).

Now let us move up the structure diagram to the apop_model structure.

2.1.2 apop_model

Even restricting ourselves to the most basic operations, there are a lot of things that we want to do with our models: use a data set to estimate the parameters of a model (like the mean and variance of a Normal distribution), or draw random numbers, or show the expected value, or show the expected value of one part of the data given fixed values for the rest of it. The apop_model is intended to encapsulate most of these desires into one object, so that models can easily be swapped around, modified to create new models, compared, and so on.

From the figure above, you can see that the apop_model structure includes a number of informational items, key being the `parameters`, `data`, and `info` elements; a list of settings to be discussed below; and a set of procedures for many operations. Its contents are not (entirely) arbitrary: the overall intent and the theoretical basis for what is and is not included in an apop_model are described in this U.S. Census Bureau research report.

There are helper functions that will allow you to avoid dealing with the model internals. For example, the apop_estimate helper function means you never have to look at the model's `estimate` method (if it even has one), and you will simply pass the model to a function, as with the above form:

```
apop_model *est = apop_estimate(data, apop_ols);
```

- Apophenia ships with a broad set of models, like apop_ols, apop_dirichlet, apop_loess, and apop_pmf (probability mass function); see the full list on the models documentation page. You would fit any of them using apop_estimate call, with the appropriate model as the second input.
- The models that ship with Apophenia, like apop_ols, include the procedures and some metadata, but are of course not yet estimated using a data set (i.e., `data == NULL`, `parameters == NULL`). The line above generated a new model, `est`, which is identical to the base OLS model but has estimated parameters (and covariances, and basic hypothesis tests, a log likelihood, AIC_c, BIC, et cetera), and a `data` pointer to the apop_data set used for estimation.
- You will mostly use the models by passing them as inputs to functions like apop_estimate, apop_draw, or apop_predict; more examples below. Other than apop_estimate, most require a parameterized model like `est`. After all, it doesn't make sense to draw from a Normal distribution until its mean and standard deviation are specified.
- If you know what the parameters should be, for most models use apop_model_set_parameters. E.g.

```
apop_model *std_normal = apop_model_set_parameters(apop_normal, 0, 1);
apop_data *a_thousand_normals = apop_model_draws(std_normal, 1000);

apop_model *poisson = apop_model_set_parameters(apop_poisson, 1.5);
apop_data *a_thousand_waits = apop_model_draws(poisson, 1000);
```

- You can use apop_model_print to print the various elements to screen.
- You can combine and transform models with functions such as apop_model_fix_params, apop_model_coordinate_transform, or apop_model_mixture. Each of these functions produce a new model, which can be estimated, re-combined, or otherwise used like any other model.

```
//A helper function to check whether a data point is nonnegative
double over_zero(apop_data *in, apop_model *m){ return apop_data_get(in) > 0; }

//Generate a truncated Normal distribution by adding a data constraint:
apop_model *truncated_normal= apop_model_dconstrain(.base_model=apop_normal,
                                                    .constraint=over_zero);

//Get the cross product of that and a free Normal.
apop_model *cross = apop_model_cross(apop_normal, truncated_normal);

//Given assumed data, estimate the parameters of the cross product.
apop_model *xest = apop_estimate(assumed_data, cross);

//Assuming more data, use the fitted cross product as the prior for a Normal distribution.
apop_model *posterior = apop_update(moredata, .prior=xest, .likelihood=apop_normal);

//Assuming more data, use the posterior as the prior for another updating round.
apop_model *post2 = apop_update(moredata2, .prior=posterior, .likelihood=apop_normal);
```

- Writing your own models won't be covered in this introduction, but it can be easy to copy and modify the procedures of an existing model to fit your needs. When in doubt, delete a procedure, because any procedures that are missing will have defaults filled when used by functions like apop_estimate (which uses apop_maximum_likelihood) or apop_cdf (which uses integration via random draws). See Writing new models for details.
- There's a simple rule of thumb for remembering the order of the arguments to most of Apophenia's functions, including apop_estimate : the data always comes first.

Settings

How many bins are in a histogram? At what tolerance does the maximum likelihood search end? What are the models being combined in an apop_mixture distribution?

Apophenia organizes settings in *settings groups*, which are then attached to models. In the following snippet demonstrating Bayesian updating, we specify a Beta distribution prior. If the likelihood function were a Binomial distribution, apop_update knows the closed-form posterior for a Beta-Binomial pair, but in this case, with a PMF as a likelihood, it will have to run Markov chain Monte Carlo. The apop_mcmc_settings group attached to the prior specifies details of how the run should work.

For a likelihood, we generate an empirical distribution—a PMF—from an input data set, via `apop_estimate(your_data, apop_pmf)`. When we call apop_update on the last line, it already has all of the above info on hand.

```
apop_model *beta = apop_model_set_parameters(apop_beta, 0.5, 0.25);
Apop_settings_add_group(beta, apop_mcmc, .burnin = 0.2, .periods =1e5);
apop_model *my_pmf = apop_estimate(your_data, apop_pmf);
apop_model *posterior = apop_update(.prior= beta, .likelihood = my_pmf);
```

Databases and models

Returning to the introductory example, you saw that (1) the library expects you to keep your data in a database, pulling out the data as needed, and (2) that the workflow is built around apop_model structures.

Starting with (2), if a stats package has something called a *model*, then it is probably of the form Y = [an additive function of **X**], such as $y = x_1 + \log(x_2) + x_3^2$. Trying new models means trying different functional forms for the right-hand side, such as including x_1 in some cases and excluding it in others. Conversely, Apophenia is designed to facilitate trying new models in the broader sense of switching out a linear model for a hierarchical, or a Bayesian model for a simulation. A formula syntax makes little sense over such a broad range of models.

As a result, the right-hand side is not part of the apop_model. Instead, the data is assumed to be correctly formatted, scaled, or logged before being passed to the model. This is where part (1), the database, comes in, because it provides a proxy for the sort of formula specification language above:

```
apop_data *testme= apop_query_to_data("select y, x1, log(x2), pow(x3, 2) from
       data");
apop_model *est = apop_estimate(testme, apop_ols);
```

Generating factors and dummies is also considered data prep, not model internals. See apop_data_to_dummies and apop_data_to_factors.

Now that you have `est`, an estimated model, you can interrogate it. This is where Apophenia and its encapsulated model objects shine, because you can do more than just admire the parameter estimates on the screen: you can take your estimated data set and fill in or generate new data, use it as an input to the parent distribution of a hierarchical model, et cetera. Some simple examples:

```
 //If you have a new data set with missing elements (represented by NaN), you can fill in predicted values:
apop_predict(new_data_set, est);
apop_data_print(new_data_set);
```

```
 //Fill a matrix with random draws.
apop_data *d = apop_model_draws(est, .count=1000);

 //How does the AIC_c for this model compare to that of est2?
printf("AIC_c=%g\n", apop_data_get(est->info, .rowname="AIC_c")
                        - apop_data_get(est2->info, .rowname="AIC_c"));
```

2.1.3 Conclusion

This introduction has shown you the apop_data set and some of the functions associated, which might be useful even if you aren't formally doing statistical work but do have to deal with data with real-world elements like column names and mixed numeric/text values. You've seen how Apophenia encapsulates many of a model's characteristics into a single apop_model object, which you can send with data to functions like apop_estimate, apop_predict, or apop_draw. Once you've got your data in the right form, you can use this to simply estimate model parameters, or as an input to later analysis.

What's next?

- Check out the system for hypothesis testing, both with traditional known distributions (using apop_test for dealing with Normal-, t-, χ^2-distributed statistics); and for the parameters of any model; in Tests & diagnostics.
- Try your own hand at putting new models into the apop_model framework, as discussed in Writing new models.
- For example, have a look at this blog and its subsequent posts, which wrap a microsimulation into an apop_model, so that its parameters can be estimated and confidence intervals set around them.
- See the Optimization page for discussion of the many features the optimization system has. It allows you to use a diverse set of search types on constrained or unconstrained models.
- Skim through the full list of macros and functions—there are hundreds—to get a sense of what else Apophenia offers.

2.2 Setting up

2.2.1 The supporting cast

To use Apophenia, you will need to have a working C compiler, the GSL (v1.7 or higher) and SQLite installed. mySQL/mariaDB is optional.

- Some readers may be unfamiliar with modern package managers and common methods for setting up a C development environment; see Appendix O of *Modeling with Data* for an introduction.
- Other pages in this documentation have a few more notes for Windows users, including MinGW users.
- Install the basics using your package manager. E.g., try

```
sudo apt-get install make gcc libgsl0-dev libsqlite3-dev
```

or

```
sudo yum install make gcc gsl-devel libsqlite3x-devel
```

- Download Apophenia here.
- Once you have the library downloaded, compile it using

```
tar xvzf apop*tgz && cd apophenia-0.999
./configure && make && make check && sudo make install
```

If you decide not to keep the library on your system, run `sudo make uninstall` from the source directory to remove it.

- If you need to install packages in your home directory because you don't have root permissions, see the Not root? page.
- A Makefile will help immensely when you want to compile your program.
- You can verify that your setup works by trying some sample programs.

Windows

2.2.2 Not root?

If you aren't root, then the common procedure for installing a library is to create a subdirectory in your home directory in which to install packages. The key is the `-prefix` addition to the `./configure` command.

```
export MY_LIBS = myroot   #choose a directory name to be created in your home directory.
mkdir $HOME/$MY_LIBS

# From Apophenia's package directory:
./configure --prefix $HOME/$MY_LIBS
make
make install   #Now you don't have to be root.

# Adjust your paths so the compiler and the OS can find the library.
# These are environment variables, and they are usually set in the
# shell's startup files. I assume you are using bash here.

echo "export PATH=$HOME/$MY_LIBS/include:\$PATH" >> ~/.bashrc
echo "export CPATH=$HOME/$MY_LIBS/include:\$CPATH" >> ~/.bashrc
echo "export LIBRARY_PATH=$HOME/$MY_LIBS:\$LIBRARY_PATH" >> ~/.bashrc
echo "export LD_LIBRARY_PATH=$HOME/$MY_LIBS:\$LD_LIBRARY_PATH" >> ~/.bashrc
```

Once you have created this local root directory, you can use it to install as many new libraries as desired, and your paths will already be set up to find them.

2.2.3 Makefile

Instead of giving lengthy compiler commands at the command prompt, you can use a Makefile to do most of the work. How to:

- Copy and paste the following into a file named `makefile`.
- Change the first line to the name of your program (e.g., if you have written `census.c`, then the first line will read `PROGNAME=census`).
- If your program has multiple `.c` files, add a corresponding `.o` to the currently blank `objects` variable, e.g. `objects=sample2.o sample3.o`

- One you have a Makefile in the directory, simply type `make` at the command prompt to generate the executable.

```
PROGNAME = your_program_name_here
objects =
CFLAGS = -g -Wall -O3
LDLIBS = -lapophenia -lgsl -lgslcblas -lsqlite3

$(PROGNAME): $(objects)
```

- If your system has `pkg-config`, then you can use it for a slightly more robust and readable makefile. Replace the above C and link flags with:

  ```
  CFLAGS = -g -Wall `pkg-config --cflags apophenia` -O3
  LDLIBS = `pkg-config --libs apophenia`
  ```

  The `pkg-config` program will then fill in the appropriate directories and libraries. Pkg-config knows Apophenia depends on the GSL and database libraries, so you need only list the most-dependent library.

- The `-O3` flag is optional, asking the compiler to run its highest level of optimization (for speed).

- GCC users may need the `-std=gnu99` or `-std=gnu11` flag to use post-1989 C standards.

- Order matters in the linking list: the files a package depends on should be listed after the package. E.g., since sample.c depends on Apophenia, `gcc sample.c -lapophenia` will work, while `gcc -lapophenia sample.c` is likely to give you errors. Similarly, list `-lapophenia` before `-lgsl`, which comes before `-lgslcblas`.

2.2.4 Windows

MinGW users, see that page.

If you have a choice, Cygwin is strongly recommended. The setup program is very self-explanatory. As a warning, it will probably take up >300MB on your system. You should install at least the following programs:

- autoconf/automake
- binutils
- gcc
- gdb
- gnuplot – for plotting data
- groff – needed for the man program, below
- gsl – the engine that powers Apophenia
- less – to read text files
- libtool – needed for compiling programs
- make
- man – for reading help files
- more – not as good as less but still good to have

- sqlite3 – a simple database engine, a requisite for Apophenia

If you are missing anything else, the program will probably tell you. The following are not necessary but are good to have on hand as long as you are going to be using Unix and programming.

- git – to partake in the versioning system
- emacs – steep learning curve, but people love it
- ghostscript (for reading .ps/.pdf files)
- openssh – needed for git
- perl, python, ruby – these are other languages that you might also be interested in
- tetex – write up your documentation using the nicest-looking formatter around
- X11 – a windowing system

X-Window will give you a nicer environment in which to work. After you start Cygwin, type `startx` to bring up a more usable, nice-looking terminal (and the ability to do a few thousand other things which are beyond the scope of this documentation). Once you have Cygwin installed and a good terminal running, you can follow along with the remainder of the discussion without modification.

Some older versions of Cygwin have a `search.h` file which doesn't include the function `lsearch()`. If this is the case on your system, you will have to update your Cygwin installation.

Finally, windows compilers often spit out lines like:

```
Info: resolving _gsl_rng_taus by linking to __imp__gsl_rng_taus (auto-import)
```

These lines are indeed just information, and not errors. Feel free to ignore them.

[Thanks to Andrew Felton and Derrick Higgins for their Cygwin debugging efforts.]

MinGW

2.2.5 MinGW

Minimalist GNU for Windows is indeed minimalist: it is not a full POSIX subsystem, and provides no package manager. Therefore, you will have to make some adjustments and install the dependencies yourself.

Matt P. Dziubinski successfully used Apophenia via MinGW; here are his instructions (with edits by BK):

- get libregex (the ZIP file) from: http://sourceforge.net/project/showfiles.php?group_id=204414&package_id=306189
- get libintl (three ZIP files) from: http://gnuwin32.sourceforge.net/packages/libintl.htm . download "Binaries", "Dependencies", "Developer files"
- follow "libintl" steps from: http://kayalang.org/download/compiling/windows
- Modify `Makefile`, adding -lpthread to AM_CFLAGS (removing -pthread) and -lregex to AM_CFLAGS and LIBS
- Now compile the main library:

  ```
  make
  ```

- Finally, put one more expected directory in place and install:

```
mkdir -p -- "/usr/local/Lib/site-packages"
make install
```

- You will get the usual warning about library paths, and may have to take the specified action:

```
----------------------------------------------------------------------
Libraries have been installed in:
  /usr/local/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
  - add LIBDIR to the `PATH' environment variable
    during execution
  - add LIBDIR to the `LD_RUN_PATH' environment variable
    during linking
  - use the `-LLIBDIR' linker flag

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
```

2.3 Some examples

Here are a few pieces of sample code for testing your installation or to give you a sense of what code with Apophenia's tools looks like.

Two data streams

The sample program here is intended to show how one would integrate Apophenia into an existing program. For example, say that you are running a simulation of two different treatments, or say that two sensors are posting data at regular intervals. The goal is to gather the data in an organized form, and then ask questions of the resulting data set. Below, a thousand draws are made from the two processes and put into a database. Then, the data is pulled out, some simple statistics are compiled, and the data is written to a text file for inspection outside of the program. This program will compile cleanly with the sample Makefile.

```
#include <apop.h>

//Your processes are probably a bit more complex.
double process_one(gsl_rng *r){
    return gsl_rng_uniform(r) * gsl_rng_uniform(r) ;
}

double process_two(gsl_rng *r){
    return gsl_rng_uniform(r);
}

int main(){
    gsl_rng *r = apop_rng_alloc(123);

    //create the database and the data table.
    apop_db_open("runs.db");
    apop_table_exists("samples", 'd'); //If the table already exists, delete it.
    apop_query("create table samples(iteration, process, value); begin;");

    //populate the data table with runs.
    for (int i=0; i<1000; i++){
        double p1 = process_one(r);
        double p2 = process_two(r);
        apop_query("insert into samples values(%i, %i, %g);", i, 1, p1);
```

```
        apop_query("insert into samples values(%i, %i, %g);", i, 2, p2);
    }
    apop_query("commit;"); //the begin-commit wrapper saves writes to the drive.

    //pull the data from the database, converting it into a table along the way.
    apop_data *m  = apop_db_to_crosstab("samples", "iteration","process", "
      value");

    gsl_vector *v1 = Apop_cv(m, 0); //get vector views of the two table columns.
    gsl_vector *v2 = Apop_cv(m, 1);

    //Output a table of means/variances, and t-test results.
    printf("\t   mean\t\t   var\n");
    printf("process 1: %f\t%f\n", apop_mean(v1), apop_var(v1));
    printf("process 2: %f\t%f\n\n", apop_mean(v2), apop_var(v2));
    printf("t test\n");
    apop_data_show(apop_t_test(v1, v2));
    apop_data_print(m, "the_data.txt");
}
```

Run a regression

See A quick overview for an example of loading a data set and running a simple regression.

A sequence of t-tests

In The section on map/apply, a new t-test on every row, with all operations acting on entire rows rather than individual data points:

```
#include <apop.h>

double row_offset;

void offset_rng(double *v){*v = gsl_rng_uniform(apop_rng_get_thread()) + row_offset;}
double find_tstat(gsl_vector *in){ return apop_mean(in)/sqrt(apop_var(in));}
double conf(double in, void *df){ return gsl_cdf_tdist_P(in, *(int *)df);}

//apop_vector_mean is a macro, so we can't point a pointer to it.
double mu(gsl_vector *in){ return apop_vector_mean(in);}

int main(){
    apop_data *d = apop_data_alloc(10, 100);
    gsl_rng *r = apop_rng_alloc(3242);
    for (int i=0; i< 10; i++){
        row_offset = gsl_rng_uniform(r)*2 -1; //declared and used above.
        apop_vector_apply(Apop_rv(d, i), offset_rng);
    }

    size_t df = d->matrix->size2-1;
    apop_data *means = apop_map(d, .fn_v = mu, .part ='r');
    apop_data *tstats = apop_map(d, .fn_v = find_tstat, .part ='r');
    apop_data *confidences = apop_map(tstats, .fn_dp = conf, .param = &df);

    printf("means:\n"); apop_data_show(means);
    printf("\nt stats:\n"); apop_data_show(tstats);
    printf("\nconfidences:\n"); apop_data_show(confidences);

    //Some sanity checks, for Apophenia's test suite.
    for (int i=0; i< 10; i++){
        //sign of mean == sign of t stat.
        assert(apop_data_get(means, i, -1) * apop_data_get(tstats, i, -1) >=0);

        //inverse of P-value should be the t statistic.
        assert(fabs(gsl_cdf_tdist_Pinv(apop_data_get(confidences, i, -1),100)
                    - apop_data_get(tstats, i, -1) < 1e-3));
    }
}
```

In the documentation for apop_query_to_text, a program to list all the tables in an SQLite database.

```
#include <apop.h>

void print_table_list(char *db_file){
    apop_db_open(db_file);
    apop_data *tab_list= apop_query_to_text("select name "
                    "from sqlite_master where type=='table'");
    for(int i=0; i< tab_list->textsize[0]; i++)
        printf("%s\n", tab_list->text[i][0]);
}

int main(int argc, char **argv){
    if (argc == 1){
        printf("Give me a database name, and I will print out "
               "the list of tables contained therein.\n");
        return 0;
    }
    print_table_list(argv[1]);
}
```

Marginal distribution

A demonstration of fixing parameters to create a marginal distribution, via apop_model_fix_params

```
#include <apop.h>

int main(){
    size_t ct = 5e4;

    //set up the model & params
    apop_data *params = apop_data_falloc((2,2,2), 8,  1, 0.5,
                                                  2,  0.5, 1);
    apop_model *pvm = apop_model_copy(apop_multivariate_normal);
    pvm->parameters = apop_data_copy(params);
    pvm->dsize = 2;
    apop_data *d = apop_model_draws(pvm, ct);

    //set up and estimate a model with fixed covariance matrix but free means
    gsl_vector_set_all(pvm->parameters->vector, GSL_NAN);
    apop_model *mep1 = apop_model_fix_params(pvm);
    apop_model *e1 = apop_estimate(d, mep1);

    //compare results
    printf("original params: ");
    apop_vector_print(params->vector);
    printf("estimated params: ");
    apop_vector_print(e1->parameters->vector);
    assert(apop_vector_distance(params->vector, e1->parameters->vector)<1e-2);
}
```

2.4 References and extensions

2.4.1 The book version

Apophenia co-evolved with *Modeling with Data: Tools and Techniques for Statistical Computing*. You can read about the book, or download a free PDF copy of the full text, at modelingwithdata.org.

As with many computer programs, the preferred manner of citing Apophenia is to cite its related book. Here is a BibTeX-formatted entry giving the relevant information:

```
@book{klemens:modeling,
    title = "Modeling with Data: Tools and Techniques for Statistical Computing",
    author="Ben Klemens",
```

```
    year=2008,
    publisher="Princeton University Press"
}
```

The rationale for the apop_model struct, based on an algebraic system of models, is detailed in a U.S. Census Bureau research report:

```
@techreport{klemens:algebra,
    title = "A Useful Algebraic System of Statistical Models",
    author="Ben Klemens",
    month=jul,
    year=2014,
    institution="U.S.\ Census Bureau",
    number="06"
}
```

2.4.2 How do I write extensions?

The system is written to not require a registration or initialization step to add a new model or other such parts. Just write your code and `include` it like any other C code. A new apop_model has to conform to some rules if it is to play well with apop_estimate, apop_draw, and so forth. See the notes at Writing new models. Once your new model or function is working, please post the code or a link to the code on the Apophenia wiki.

C, SQL and coding utilities

2.4.3 Further references

For your convenience, here are links to some other libraries you are probably using.

- The standard C library
- The GSL documentation, and its index
- SQL understood by SQLite

2.4.4 C, SQL and coding utilities

Learning C

Modeling with Data has a full tutorial for C, oriented at users of standard stats packages. More nuts-and-bolts tutorials are in abundance. Some people find pointers to be especially difficult; fortunately, there's a claymation cartoon which clarifies everything.

Header aggregation

There is only one header. Put

```
#include <apop.h>
```

at the top of your file, and you're done. Everything declared in that file starts with `apop_` or `Apop_`. It also includes `assert.h`, `math.h`, `signal.h`, and `string.h`.

Linking

You will need to link to the Apophenia library, which involves adding the `-lapophenia` flag to your compiler. Apophenia depends on SQLite3 and the GNU Scientific Library (which depends on a BLAS), so you will probably need something like:

```
gcc sample.c -lapophenia -lsqlite3 -lgsl -lgslcblas -o run_me -g -Wall -O3
```

Your best bet is to encapsulate this mess in a Makefile. Even if you are using an IDE and its command-line management tools, see the Makefile page for notes on useful flags.

Standards compliance

To the best of our abilities, Apophenia complies to the C standard (ISO/IEC 9899:2011). As well as relying on the GSL and SQLite, it uses some POSIX function calls, such as `strcasecmp` and `popen`.

Designated initializers

2.4.4.1 Errors, logging, debugging and stopping

The `error` element

The apop_data set and the apop_model both include an element named `error`. It is normally `0`, indicating no (known) error.

For example, apop_data_copy detects allocation errors and some circular links (when `Data->more == Data`) and fails in those cases. You could thus use the function with a form like

```
apop_data *d = apop_text_to_data("indata");
apop_data *cp = apop_data_copy(d);
if (cp->error) {printf("Couldn't copy the input data; failing.\n"); return 1;}
```

There is sometimes (but not always) benefit to handling specific error codes, which are listed in the documentation of those functions that set the `error` element. E.g.,

```
apop_data *d = apop_text_to_data("indata");
apop_data *cp = apop_data_copy(d);
if (cp->error == 'a') {printf("Couldn't allocate space for the copy; failing.\n"); return 1;}
if (cp->error == 'c') {printf("Circular link in the data set; failing.\n"); return 2;}
```

The end of Appendix O of *Modeling with Data* offers some GDB macros which can make dealing with Apophenia from the GDB command line much more pleasant. As discussed below, it also helps to set `apop_opts.stop_on_warning='v'` or `'w'` when running under the debugger.

2.4.4.2 Verbosity level and logging

The global variable `apop_opts.verbose` determines how many notifications and warnings get printed by Apophenia's warning mechanism:

-1: turn off logging, print nothing (ill-advised)
0: notify only of failures and clear danger
1: warn of technically correct but odd situations that might indicate, e.g., numeric instability
2: debugging-type information; print queries
3: give me everything, such as the state of the data at each iteration of a loop.

These levels are of course subjective, but should give you some idea of where to place the verbosity level. The default is 1.

The messages are printed to the `FILE*` handle at `apop_opts.log_file`. If this is blank (which happens at startup), then this is set to `stderr`. This is the typical behavior for a console program. Use

```
apop_opts.log_file = fopen("mylog", "w");
```

to write to the `mylog` file instead of `stderr`.

As well as the error and warning messages, some functions can also print diagnostics, using the Apop_notify macro. For example, apop_query and friends will print the query sent to the database engine iff apop_↩ opts.verbose >=2 (which is useful when building complex queries). The diagnostics attempt to follow the same verbosity scale as the warning messages.

2.4.4.3 Stopping

By default, warnings and errors never halt processing. It is up to the calling function to decide whether to stop.

When running the program under a debugger, this is an annoyance: we want to stop as soon as a problem turns up.

The global variable apop_opts.stop_on_warning changes when the system halts:

'n': never halt. If you were using Apophenia to support a user-friendly GUI, for example, you would use this mode.
The default: if the variable is '\0' (the default), halt on severe errors, continue on all warnings.
'v': If the verbosity level of the warning is such that the warning would print to screen, then halt; if the warning message would be filtered out by your verbosity level, continue.
'w': Halt on all errors or warnings, including those below your verbosity threshold.

See the documentation for individual functions for details on how each reports errors to the caller and the level at which warnings are posted.

2.4.4.4 Legible output

The output routines handle four sinks for your output. There is a global variable that you can use for small projects where all data will go to the same place.

```
apop_opts.output_type = 'f'; //named file
apop_opts.output_type = 'p'; //a pipe or already-opened file
apop_opts.output_type = 'd'; //the database
```

You can also set the output type, the name of the output file or table, and other options via arguments to individual calls to output functions. See apop_prep_output for the list of options.

C makes minimal distinction between pipes and files, so you can set a pipe or file as output and send all output there until further notice:

```
apop_opts.output_type = 'p';
apop_opts.output_pipe = popen("gnuplot", "w");
apop_plot_lattice(...); //see https://github.com/b-k/Apophenia/wiki/gnuplot_snippets
fclose(apop_opts.output_pipe);
apop_opts.output_pipe = fopen("newfile", "w");
apop_data_print(set1);
fprintf(apop_opts.output_pipe, "\nNow set 2:\n");
apop_data_print(set2);
```

Continuing the example, you can always override the global data with a specific request:

```
apop_vector_print(v, "vectorfile"); //put vectors in a separate file
apop_matrix_print(m, "matrix_table", .output_type = 'd'); //write to the db
apop_matrix_print(m, .output_pipe = stdout);  //now show the same matrix on screen
```

I will first look to the input file name, then the input pipe, then the global output_pipe, in that order, to determine to where I should write. Some combinations (like output type = 'd' and only a pipe) don't make sense, and I'll try to warn you about those.

What if you have too much output and would like to use a pager, like less or more? In C and POSIX terminology, you're asking to pipe your output to a paging program. Here is the form:

```
FILE *lesspipe = popen("less", "w");
assert(lesspipe);
apop_data_print(your_data_set, .output_pipe=lesspipe);
pclose(lesspipe);
```

`popen` will search your usual program path for `less`, so you don't have to give a full path.

- apop_data_print
- apop_matrix_print
- apop_vector_print

2.4.4.5 About SQL, the syntax for querying databases

For a reference, your best bet is the `Structured Query Language reference` for SQLite. For a tutorial; there is an abundance of `tutorials online`. Here is a nice blog `entry` about complementaries between SQL and matrix manipulation packages.

Apophenia currently supports two database engines: SQLite and mySQL/mariaDB. SQLite is the default, because it is simpler and generally more easygoing than mySQL, and supports in-memory databases.

The global `apop_opts.db_engine` is initially `NULL`, indicating no preference for a database engine. You can explicitly set it:

```
apop_opts.db_engine='s' //use SQLite
apop_opts.db_engine='m' //use mySQL/mariaDB
```

If `apop_opts.db_engine` is still `NUL` on your first database operation, then I will check for an environment variable `APOP_DB_ENGINE`, and set `apop_opts.db_engine='m'` if it is found and matches (case insensitive) `mariadb` or `mysql`.

```
export APOP_DB_ENGINE=mariadb
apop_text_to_db indata mtab db_for_maria

unset APOP_DB_ENGINE
apop_text_to_db indata stab db_for_sqlite.db
```

Write apop_data sets to the database using apop_data_print, with `.output_type='d'`.

- Column names are inserted if there are any. If there are, all dots are converted to underscores. Otherwise, the columns will be named `c1`, `c2`, `c3`, &c.
- If apop_opts.db_name_column is not blank (the default is `"row_name"`), then a so-named column is created, and the row names are placed there.
- If there are weights, they will be the last column of the table, and the column will be named `weights`.
- If the table does not exist, create it. Use apop_data_print `(data, "tabname", .output_type='d', .output_append='w')` to overwrite an existing table or with `.output_append='a'` to append. Appending is the default. Or, call apop_table_exists `("tabname", 'd')` to ensure that the table is removed ahead of time.
- If your data set has zero data (i.e., is just a list of column names or is entirely blank), apop_data_print returns without creating anything in the database.
- Especially if you are using a pre-2007 version of SQLite, there may be a speed gain to wrapping the call to this function in a begin/commit pair:

```
apop_query("begin;");
apop_data_print(dataset, .output_name="dbtab", .output_type='d');
apop_query("commit;");
```

Finally, Apophenia provides a few nonstandard SQL functions to facilitate math via database; see Database moments (plus pow()!).

2.4.4.6 Threading

Apophenia uses OpenMP for threading. You generally do not need to know how OpenMP works to use Apophenia, and many points of work will thread without your doing anything.

- All functions strive to be thread-safe. Part of how this is achieved is that static variables are marked as thread-local or atomic, as per the C standard. There still exist compilers that can't implement thread-local or atomic variables, in which case your safest bet is to set OMP's thread count to one as below (or get a new compiler).

- Some functions modify their inputs. It is up to you to use those functions in a thread-safe manner. The apop_matrix_realloc handles states and global variables correctly in a threaded environment, but if you have two threads resizing the same `gsl_matrix` at the same time, you're going to have problems.

- There are few compilers that don't support OpenMP. When compiling on such a system all work will be single-threaded.

- Set the maximum number of threads to `N` with the environment variable

```
export OMP_NUM_THREADS N
```

or the C function

```
#include <omp.h>
omp_set_num_threads(N);
```

Use one of these methods with `N=1` if you want a single-threaded program. You can return later to using all available threads via `omp_set_num_threads(omp_get_num_procs())`.

- apop_map and friends distribute their `for` loop over the input apop_data set across multiple threads. Therefore, be careful to send thread-unsafe functions to it only after calling `omp_set_num_threads(1)`.

- There are a few functions, like apop_model_draws, that rely on apop_map, and therefore also thread by default.

- The function apop_rng_get_thread retrieves a statically-stored RNG specific to a given thread. Therefore, if you use that function in the place of a `gsl_rng`, you can parallelize functions that make random draws.

- apop_rng_get_thread allocates its store of threads using `apop_opts.rng_seed`, then incrementing that seed by one. You thus probably have threads with seeds 479901, 479902, 479903, ... [If you have a better way to do it, please feel free to modify the code to implement your improvement and submit a pull request on Github.]

See this tutorial on C threading if you would like to know more, or are unsure about whether your functions are thread-safe or not.

2.4.4.7 Designated initializers

Functions so marked in this documentation use standard C designated initializers and compound literals to allow you to omit, call by name, or change the order of inputs. The following examples are all equivalent.

The standard format:

```
apop_text_to_db("infile.txt", "intable", 0, 1, NULL);
```

Omitted arguments are left at their default vaules:

```
apop_text_to_db("infile.txt", "intable");
```

You can use the variable's name, if you forget its ordering:

```
apop_text_to_db("infile.txt", "intable", .has_col_name=1, .has_row_name=0);
```

If an un-named element follows a named element, then that value is given to the next variable in the standard ordering:

```
apop_text_to_db("infile.txt", "intable", .has_col_name=1, NULL);
```

- There may be cases where you can not use this form (it relies on a macro, which may not be available). You can always call the underlying function directly, by adding `_base` to the name and giving all arguments:

```
apop_text_to_db_base("infile.txt", "intable", 0, 1, NULL);
```

- If one of the optional elements is an RNG and you do not provide one, I use one from apop_rng_get_thread.

3 An outline of the library

The narrative in this section goes into greater detail on how to use the components of Apophenia. You are encouraged to read A quick overview first.

This overview begins with the apop_data set, which is the central data structure used throughout the system. Section Databases covers the use of the database interface, because there are a lot of things that a database will do better than a matrix structure like the apop_data struct.

Section Models covers statistical models, in the form of the apop_model structure. This part of the system is built upon the apop_data set to hold parameters, statistics, data sets, and so on.

Histosec covers probability mass functions, which are statistical models built directly around a data set, where the chance of drawing a given observation is proportional to how often that observation appears in the source data. There are many situations where one would want to treat a data set as a probability distribution, such as using apop_kl_divergence to find the information loss from an observed data set to a theoretical model fit to that data.

Section Tests & diagnostics covers traditional hypothesis testing, beginning with common statistics that take an apop_data set or two as input, and continuing on to generalized hypothesis testing for any apop_model.

Because estimation in the apop_model relies heavily on maximum likelihood estimation, Apophenia's optimizer subsystem is extensive. Optimization offers some additional notes on optimization and how it can be used in non-statistical contexts.

Histosec

Assorted

3.1 Data sets

The apop_data structure represents a data set. It joins together a `gsl_vector`, a `gsl_matrix`, an apop_name, and a table of strings. It tries to be lightweight, so you can use it everywhere you would use a `gsl_matrix` or a `gsl_vector`.

Here is a diagram showing a sample data set with all of the elements in place. Together, they represent a data set where each row is an observation, which includes both numeric and text values, and where each row/column may be named.

Rowname	Vector	Matrix			Text		Weights
	Outcome	**Age**	**Weight (kg)**	**Height (cm)**	**Sex**	**State**	
"Steven"	1	32	65	175	Male	Alaska	1
"Sandra"	0	41	61	165	Female	Alabama	3.2
"Joe"	1	40	73	181	Male	Alabama	2.4

In a regression, the vector would be the dependent variable, and the other columns (after factor-izing the text) the independent variables. Or think of the apop_data set as a partitioned matrix, where the vector is column -1, and the first column of the matrix is column zero. Here is some sample code to print the vector and matrix, starting at column -1 (but you can use apop_data_print to do this).

```
for (int j = 0; j< data->matrix->size1; j++){
    printf("%s\t", apop_name_get(data->names, j, 'r'));
    for (int i = -1; i< data->matrix->size2; i++)
        printf("%g\t", apop_data_get(data, j, i));
    printf("\n");
}
```

Most functions assume that each row represents one observation, so the data vector, data matrix, and text have the same row count: `data->vector->size==data->matrix->size1` and `data->vector->size==*data->textsize`. This means that the apop_name structure doesn't have separate `vector_names`, `row_names`, or `text_row_names` elements: the `rownames` are assumed to apply for all.

See below for notes on managing the `text` element and the row/column names.

3.1.1 Pages

The apop_data set includes a `more` pointer, which will typically be `NULL`, but may point to another apop_data set. This is intended for a main data set and a second or third page with auxiliary information, such as estimated parameters on the front page and their covariance matrix on page two, or predicted data on the front page and a set of prediction intervals on page two.

The `more` pointer is not intended for a linked list for millions of data points. In such situations, you can often improve efficiency by restructuring your data to use a single table (perhaps via apop_data_pack and apop_data_unpack).

Most functions, such as apop_data_copy and apop_data_free, will handle all the pages of information. For example, an optimization search over multi-page parameter sets would search the space given by all pages.

Pages may also be appended as output or auxiliary information, such as covariances, and an MLE would not search over these elements. Any page with a name in XML-ish brackets, such as `<Covariance>`, is considered information about the data, not data itself, and therefore ignored by search routines, missing data routines, et cetera. This is achieved by a rule in apop_data_pack and apop_data_unpack.

Here is a toy example that establishes a baseline data set, adds a page, modifies it, and then later retrieves it.

```
apop_data *d = apop_data_alloc(10, 10, 10); //the base data set, a 10-item vector +
```

```
        10x10 matrix
apop_data *a_new_page = apop_data_add_page(d, apop_data_alloc(2,2), "new 2 x 2 page");
gsl_vector_set_all(a_new_page->matrix, 3);

//later:
apop_data *retrieved = apop_data_get_page(d, "new", 'r'); //'r'=search via
        regex, not literal match.
apop_data_print(retrieved); //print a 2x2 grid of 3s.
```

3.1.2 Functions for using apop_data sets

There are a great many functions to collate, copy, merge, sort, prune, and otherwise manipulate the apop_data structure and its components.

- apop_data_add_named_elmt
- apop_data_copy
- apop_data_fill
- apop_data_memcpy
- apop_data_pack
- apop_data_rm_columns
- apop_data_sort
- apop_data_split
- apop_data_stack
- apop_data_transpose : transpose matrices (square or not) and text grids
- apop_data_unpack
- apop_matrix_copy
- apop_matrix_realloc
- apop_matrix_stack
- apop_text_set
- apop_text_paste
- apop_text_to_data
- apop_vector_copy
- apop_vector_fill
- apop_vector_stack
- apop_vector_realloc
- apop_vector_unique_elements

Apophenia builds upon the GSL, but it would be inappropriate to redundantly replicate the GSL's documentation here. Meanwhile, here are prototypes for a few common functions. The GSL's naming scheme is very consistent, so a simple reminder of the function name may be sufficient to indicate how they are used.

- `gsl_matrix_swap_rows (gsl_matrix * m, size_t i, size_t j)`
- `gsl_matrix_swap_columns (gsl_matrix * m, size_t i, size_t j)`
- `gsl_matrix_swap_rowcol (gsl_matrix * m, size_t i, size_t j)`
- `gsl_matrix_transpose_memcpy (gsl_matrix * dest, const gsl_matrix * src)`
- `gsl_matrix_transpose (gsl_matrix * m)` : square matrices only
- `gsl_matrix_set_all (gsl_matrix * m, double x)`
- `gsl_matrix_set_zero (gsl_matrix * m)`
- `gsl_matrix_set_identity (gsl_matrix * m)`
- `gsl_matrix_memcpy (gsl_matrix * dest, const gsl_matrix * src)`
- `void gsl_vector_set_all (gsl_vector * v, double x)`
- `void gsl_vector_set_zero (gsl_vector * v)`
- `int gsl_vector_set_basis (gsl_vector * v, size_t i)`: set all elements to zero, but set item i to one.
- `gsl_vector_reverse (gsl_vector * v)`: reverse the order of your vector's elements
- `gsl_vector_ptr` and `gsl_matrix_ptr`. To increment an element in a vector use, e.g., `*gsl_vector_ptr(v, 7) += 3;` or `(*gsl_vector_ptr(v, 7))++`.
- `gsl_vector_memcpy (gsl_vector * dest, const gsl_vector * src)`

3.1.2.1 Reading from text files

The apop_text_to_data() function takes in the name of a text file with a grid of data in (comma|tab|pipe|whatever)-delimited format and reads it to a matrix. If there are names in the text file, they are copied in to the data set. See Input text file formatting for the full range and details of what can be read in.

If you have any columns of text, then you will need to read in via the database: use apop_text_to_db() to convert your text file to a database table, do any database-appropriate cleaning of the input data, then use apop_query_to_data() or apop_query_to_mixed_data() to pull the data to an apop_data set.

Input text file formatting

3.1.3 Alloc/free

You may not need to use these functions often, given that apop_query_to_data, apop_text_to_data, and many transformation functions will auto-allocate apop_data sets for you.

The apop_data_alloc function allocates a vector, a matrix, or both. After this call, the structure will have blank names, `NULL` `text` element, and `NULL` `weights`. See Name handling for discussion of filling the names. Use apop_text_alloc to allocate the `text` grid. The `weights` are a simple `gsl_vector`, so allocate a 100-unit weights vector via `allocated_data_set->weights = gsl_vector_alloc(100)`.

Examples of use can be found throughout the documentation; for example, see A quick overview.

- apop_data_alloc
- apop_data_calloc

- apop_data_free
- apop_text_alloc : allocate or resize the text part of an apop_data set.
- apop_text_free

See also:

- gsl_matrix * gsl_matrix_alloc (size_t n1, size_t n2)
- gsl_matrix * gsl_matrix_calloc (size_t n1, size_t n2)
- void gsl_matrix_free (gsl_matrix * m)
- gsl_vector * gsl_vector_alloc (size_t n)
- gsl_vector * gsl_vector_calloc (size_t n)
- void gsl_vector_free (gsl_vector * v)

3.1.4 Using views

There are several macros for the common task of viewing a single row or column of a apop_data set.

```
apop_data *d = apop_query_to_data("select obs1, obs2, obs3 from a_table");

//Get a column using its name. Note that the generated view, ov, is the
//last item named in the call to the macro.
Apop_col_t(d, "obs1", ov);
double obs1_sum = apop_vector_sum(ov);

//Get row zero of the data set's matrix as a vector; get its sum
double row_zero_sum = apop_vector_sum(Apop_rv(d, 0));

//Get a row or rows as a standalone one-row apop_data set
apop_data_print(Apop_r(d, 0));

//ten rows starting at row 3:
apop_data *d10 = Apop_rs(d, 3, 10);
apop_data_print(d10);

//Column zero's sum
gsl_vector *cv = Apop_cv(d, 0);
double col_zero_sum = apop_vector_sum(cv);
//or one one line:
double col_zero_sum = apop_vector_sum(Apop_cv(d, 0));

//Pull a 10x5 submatrix, whose origin element is the (2,3)rd
//element of the parent data set's matrix

double sub_sum = apop_matrix_sum(Apop_subm(d, 2,3, 10,5));
```

Because these macros can be used as arguments to a function, these macros have abbreviated names to save line space.

- Apop_r : get row as one-observation apop_data set
- Apop_c : get column as apop_data set
- Apop_cv : get column as gsl_vector
- Apop_rv : get row as gsl_vector

- Apop_cs : get columns as apop_data set
- Apop_rs : get rows as apop_data set
- Apop_mcv : matrix column as vector
- Apop_mrv : matrix row as vector
- Apop_subm : get submatrix of a gsl_matrix

A second set of macros have a slightly different syntax, taking the name of the object to be declared as the last argument. These can not be used as expressions such as function arguments.

- Apop_col_t
- Apop_row_t
- Apop_col_tv
- Apop_row_tv

The view is an automatic variable, not a pointer, and therefore disappears at the end of the scope in which it is declared. If you want to retain the data after the function exits, copy it to another vector:

```
return apop_vector_copy(Apop_rv(d, 2)); //return a gsl_vector copy of row 2
```

Curly braces always delimit scope, not just at the end of a function. When program evaluation exits a given block, all variables in that block are erased. Here is some sample code that won't work:

```
apop_data *outdata;
if (get_odd){
    outdata = Apop_r(data, 1);
} else {
    outdata = Apop_r(data, 0);
}
apop_data_print(outdata); //breaks: outdata points to out-of-scope variables.
```

For this if/then statement, there are two sets of local variables generated: one for the `if` block, and one for the `else` block. By the last line, neither exists. You can get around the problem here by making sure to not put the macro declaring new variables in a block. E.g.:

```
apop_data *outdata = Apop_r(data, get_odd ? 1 : 0);
apop_data_print(outdata);
```

3.1.5 Set/get

First, some examples:

```
apop_data *d = apop_data_alloc(10, 10, 10);
apop_name_add(d->names, "Zeroth row", 'r');
apop_name_add(d->names, "Zeroth col", 'c');

//set cell at row=8 col=0 to value=27
apop_data_set(d, 8, 0, .val=27);
assert(apop_data_get(d, 8, .colname="Zeroth") == 27);
double *x = apop_data_ptr(d, .col=7, .rowname="Zeroth");
*x = 270;
assert(apop_data_get(d, 0, 7) == 270);
```

```
// This is invalid---the value doesn't follow the colname. Use .val=5.
// apop_data_set(d, .row = 3, .colname="Column 8", 5);

// This is OK, to set (3, 8) to 5:
apop_data_set(d, 3, 8, 5);


//apop_data set holding a scalar:
apop_data *s = apop_data_alloc(1);
apop_data_set(s, .val=12);
assert(apop_data_get(s) == 12);

//apop_data set holding a vector:
apop_data *v = apop_data_alloc(12);
for (int i=0; i< 12; i++) apop_data_set(s, i, .val=i*10);
assert(apop_data_get(s,3) == 30);

//This is a common form from pulling from a list of named scalars,
//produced via apop_data_add_named_elmt
double AIC = apop_data_get(your_model->info, .rowname="AIC");
```

- The versions that take a column/row name use apop_name_find for the search; see notes there on the name matching rules.
- For those that take a column number, column -1 is the vector element.
- For those that take a column name, I will search the vector last—if I don't find the name among the matrix columns, but the name matches the vector name, I return column -1.
- If you give me both a `.row` and a `.rowname`, I go with the name; similarly for `.col` and `.colname`.
- You can give the name of a page, e.g.

  ```
  double AIC = apop_data_get(data, .rowname="AIC", .col=-1, .page="<Info>");
  ```

- Numeric values default to zero, which is how the examples above that treated the apop_data set as a vector or scalar could do so relatively gracefully. So `apop_data_get(dataset, 1)` gets item (1, 0) from the matrix element of `dataset`. But as a do-what-I-mean exception, if there is no matrix element but there is a vector, then this form will get vector element 1. Relying on this DWIM exception is useful iff you can guarantee that a data set will have only a vector or a matrix but not both. Otherwise, be explicit and use `apop_data_get(dataset, 1, -1)`.

The apop_data_ptr function follows the lead of `gsl_vector_ptr` and `gsl_matrix_ptr`, and like those functions, returns a pointer to the appropriate `double`. For example, to increment the (3,7)th element of an apop_data set:

```
(*apop_data_ptr(dataset, 3, 7))++;
```

- apop_data_get
- apop_data_set
- apop_data_ptr : returns a pointer to the element.
- apop_data_get_page : retrieve a named page from a data set. If you only need a few items, you can specify a page name to `apop_data_(get|set|ptr)`.

  `See also:`

- `double gsl_matrix_get (const gsl_matrix * m, size_t i, size_t j)`

- double gsl_vector_get (const gsl_vector * v, size_t i)
- void gsl_matrix_set (gsl_matrix * m, size_t i, size_t j, double x)
- void gsl_vector_set (gsl_vector * v, size_t i, double x)
- double * gsl_matrix_ptr (gsl_matrix * m, size_t i, size_t j)
- double * gsl_vector_ptr (gsl_vector * v, size_t i)
- const double * gsl_matrix_const_ptr (const gsl_matrix * m, size_t i, size_t j)
- const double * gsl_vector_const_ptr (const gsl_vector * v, size_t i)
- gsl_matrix_get_row (gsl_vector * v, const gsl_matrix * m, size_t i)
- gsl_matrix_get_col (gsl_vector * v, const gsl_matrix * m, size_t j)
- gsl_matrix_set_row (gsl_matrix * m, size_t i, const gsl_vector * v)
- gsl_matrix_set_col (gsl_matrix * m, size_t j, const gsl_vector * v)

3.1.6 Map/apply

These functions allow you to send each element of a vector or matrix to a function, either producing a new matrix (map) or transforming the original (apply). The ..._sum functions return the sum of the mapped output.

There are two types, which were developed at different times. The apop_map and apop_map_sum functions use variadic function inputs to cover a lot of different types of process depending on the inputs. Other functions with types in their names, like apop_matrix_map and apop_vector_apply, may be easier to use in some cases. They use the same routines internally, so use whichever type is convenient.

You can do many things quickly with these functions.

Get the sum of squares of a vector's elements:

```
  //given apop_data *dataset and gsl_vector *v:
double sum_of_squares = apop_map_sum(dataset, gsl_pow_2);
double sum_of_sqvares = apop_vector_map_sum(v, gsl_pow_2);
```

Create an index vector $[0, 1, 2, ...]$.

```
double index(double in, int index){return index;}
apop_data *d = apop_map(apop_data_alloc(100), .fn_di=index, .inplace='y');
```

Given your log likelihood function, which acts on a apop_data set with only one row, and a data set where each row of the matrix is an observation, find the total log likelihood via:

```
static double your_log_likelihood_fn(apop_data * in)
     {[your math goes here]}

double total_ll = apop_map_sum(dataset, .fn_r=your_log_likelihood_fn);
```

How many missing elements are there in your data matrix?

```
static double nan_check(const double in){ return isnan(in);}

int missing_ct = apop_map_sum(in, nan_check, .part='m');
```

Get the mean of the not-NaN elements of a data set:

```
static double no_nan_val(const double in){ return isnan(in)? 0 : in;}
static double not_nan_check(const double in){ return !isnan(in);}

static double apop_mean_no_nans(apop_data *in){
    return apop_map_sum(in, no_nan_val)/apop_map_sum(in, not_nan_check);
}
```

The following program randomly generates a data set where each row is a list of numbers with a different mean. It then finds the t statistic for each row, and the confidence with which we reject the claim that the statistic is less than or equal to zero.

Notice how the older apop_vector_apply uses file-global variables to pass information into the functions, while the apop_map uses a pointer to send parameters to the functions.

```
#include <apop.h>

double row_offset;

void offset_rng(double *v){*v = gsl_rng_uniform(apop_rng_get_thread()) + row_offset;}
double find_tstat(gsl_vector *in){ return apop_mean(in)/sqrt(apop_var(in));}
double conf(double in, void *df){ return gsl_cdf_tdist_P(in, *(int *)df);}

//apop_vector_mean is a macro, so we can't point a pointer to it.
double mu(gsl_vector *in){ return apop_vector_mean(in);}

int main(){
    apop_data *d = apop_data_alloc(10, 100);
    gsl_rng *r = apop_rng_alloc(3242);
    for (int i=0; i< 10; i++){
        row_offset = gsl_rng_uniform(r)*2 -1; //declared and used above.
        apop_vector_apply(Apop_rv(d, i), offset_rng);
    }

    size_t df = d->matrix->size2-1;
    apop_data *means = apop_map(d, .fn_v = mu, .part ='r');
    apop_data *tstats = apop_map(d, .fn_v = find_tstat, .part ='r');
    apop_data *confidences = apop_map(tstats, .fn_dp = conf, .param = &df);

    printf("means:\n"); apop_data_show(means);
    printf("\nt stats:\n"); apop_data_show(tstats);
    printf("\nconfidences:\n"); apop_data_show(confidences);

    //Some sanity checks, for Apophenia's test suite.
    for (int i=0; i< 10; i++){
        //sign of mean == sign of t stat.
        assert(apop_data_get(means, i, -1) * apop_data_get(tstats, i, -1) >=0);

        //inverse of P-value should be the t statistic.
        assert(fabs(gsl_cdf_tdist_Pinv(apop_data_get(confidences, i, -1),100)
                    - apop_data_get(tstats, i, -1) < 1e-3));
    }
}
```

One more toy example demonstrating the use of apop_map and apop_map_sum :

```
#include <apop.h>
/* This sample code sets the elements of a data set's vector to one
   if the index is even.  Then, via the weights vector, it adds up
   the even indices.

   There is really no need to use the weights vector; this code
   snippet is an element of Apophenia's test suite, and goes the long
```

```
   way to test that the weights are correctly handled. */

double set_vector_to_even(apop_data * r, int index){
    apop_data_set(r, 0, -1, .val=1-(index %2));
    return 0;
}

double set_weight_to_index(apop_data * r, int index){
    gsl_vector_set(r->weights, 0, index);
    return 0;
}

double weight_given_even(apop_data *r){
    return gsl_vector_get(r->vector, 0) ? gsl_vector_get(r->weights, 0) : 0;
}

int main(){
    apop_data *d = apop_data_alloc(100);
    d->weights = gsl_vector_alloc(100);
    apop_map(d, .fn_ri=set_vector_to_even, .inplace='v'); //'v=void. Throw out return values.
    apop_map(d, .fn_ri=set_weight_to_index, .inplace='v');
    double sum = apop_map_sum(d, .fn_r = weight_given_even);
    assert(sum == 49*25*2);
}
```

- If the number of threads is greater than one, then the matrix will be broken into chunks and each sent to a different thread. Notice that the GSL is generally threadsafe, and SQLite is threadsafe conditional on several commonsense caveats that you'll find in the SQLite documentation. See apop_rng_get_thread() to use the GSL's RNGs in a threaded environment.

- The ...sum functions are convenience functions that call ...map and then add up the contents. Thus, you will need to have adequate memory for the allocation of the temp matrix/vector.

- apop_map
- apop_map_sum
- apop_matrix_apply
- apop_matrix_map
- apop_matrix_map_all_sum
- apop_matrix_map_sum
- apop_vector_apply
- apop_vector_map
- apop_vector_map_sum

3.1.7 Basic Math

- apop_vector_exp : exponentiate every element of a vector
- apop_vector_log : take the natural log of every element of a vector
- apop_vector_log10 : take the log (base 10) of every element of a vector
- apop_vector_distance : find the distance between two vectors via various metrics

- apop_vector_normalize : scale/shift a matrix to have mean zero, sum to one, have a range of exactly $[0, 1]$, et cetera
- apop_vector_entropy : calculate the entropy of a vector of frequencies or probabilities

See also:

- `int gsl_matrix_add (gsl_matrix * a, const gsl_matrix * b)`
- `int gsl_matrix_sub (gsl_matrix * a, const gsl_matrix * b)`
- `int gsl_matrix_mul_elements (gsl_matrix * a, const gsl_matrix * b)`
- `int gsl_matrix_div_elements (gsl_matrix * a, const gsl_matrix * b)`
- `int gsl_matrix_scale (gsl_matrix * a, const double x)`
- `int gsl_matrix_add_constant (gsl_matrix * a, const double x)`
- `gsl_vector_add (gsl_vector * a, const gsl_vector * b)`
- `gsl_vector_sub (gsl_vector * a, const gsl_vector * b)`
- `gsl_vector_mul (gsl_vector * a, const gsl_vector * b)`
- `gsl_vector_div (gsl_vector * a, const gsl_vector * b)`
- `gsl_vector_scale (gsl_vector * a, const double x)`
- `gsl_vector_add_constant (gsl_vector * a, const double x)`

3.1.8 Matrix math

- apop_dot : matrix · matrix, matrix · vector, or vector · matrix
- apop_matrix_determinant
- apop_matrix_inverse
- apop_det_and_inv : find determinant and inverse at the same time

See the GSL documentation for myriad further options.

3.1.9 Summary stats

- apop_data_summarize
- apop_vector_moving_average
- apop_vector_percentiles
- apop_vector_bounded

See also:

- `double gsl_matrix_max (const gsl_matrix * m)`
- `double gsl_matrix_min (const gsl_matrix * m)`

- void gsl_matrix_minmax (const gsl_matrix * m, double * min_out, double * max_out)
- void gsl_matrix_max_index (const gsl_matrix * m, size_t * imax, size_t * jmax)
- void gsl_matrix_min_index (const gsl_matrix * m, size_t * imin, size_t * jmin)
- void gsl_matrix_minmax_index (const gsl_matrix * m, size_t * imin, size_t * jmin, size_t * imax, size_t * jmax)
- gsl_vector_max (const gsl_vector * v)
- gsl_vector_min (const gsl_vector * v)
- gsl_vector_minmax (const gsl_vector * v, double * min_out, double * max_↩ out)
- gsl_vector_max_index (const gsl_vector * v)
- gsl_vector_min_index (const gsl_vector * v)
- gsl_vector_minmax_index (const gsl_vector * v, size_t * imin, size_t * imax)

3.1.10 Moments

For most of these, you can add a weights vector for weighted mean/var/cov/..., such as `apop_vector_↩ mean(d->vector, .weights=d->weights)`

- apop_mean : the first three with short names operate on a vector.
- apop_sum
- apop_var
- apop_matrix_sum
- apop_data_correlation
- apop_data_covariance
- apop_data_summarize
- apop_matrix_mean
- apop_matrix_mean_and_var
- apop_vector_correlation
- apop_vector_cov
- apop_vector_kurtosis
- apop_vector_kurtosis_pop
- apop_vector_mean
- apop_vector_skew
- apop_vector_skew_pop
- apop_vector_sum
- apop_vector_var
- apop_vector_var_m

3.1.11 Conversion among types

There are no functions provided to convert from apop_data to the constituent elements, because you don't need a function.

If you need an individual element, you can use its pointer to retrieve it:

```
apop_data *d = apop_query_to_mixed_data("vmmw", "select result, age, "
                                        "income, replicate_weight from data");
double avg_result = apop_vector_mean(d->vector, .weights=d->weights);
```

In the other direction, you can use compound literals to wrap an apop_data struct around a loose vector or matrix:

```
//Given:
gsl_vector *v;
gsl_matrix *m;

// Then this form wraps the elements into automatically-allocated apop_data structs.

apop_data *dv = &(apop_data){.vector=v};
apop_data *dm = &(apop_data){.matrix=m};

apop_data *v_dot_m = apop_dot(dv, dm);

//Here is a macro to hide C's ugliness:
#define As_data(...) (&(apop_data){__VA_ARGS__})

apop_data *v_dot_m2 = apop_dot(As_data(.vector=v), As_data(.matrix=m));

//The wrapped object is an automatically-allocated structure pointing to the
//original data. If it needs to persist or be separate from the original,
//make a copy:
apop_data *dm_copy = apop_data_copy(As_data(.vector=v, .matrix=m));
```

- apop_array_to_vector : `double*` → `gsl_vector`
- apop_data_fill : `double*` → apop_data
- apop_data_falloc : macro to allocate and fill a apop_data set
- apop_text_to_data : delimited text file → apop_data
- apop_text_to_db : delimited text file → database table
- apop_vector_to_matrix

3.1.12 Name handling

If you generate your data set via apop_text_to_data or from the database via apop_query_to_data (or apop_query_to_text or apop_query_to_mixed_data) then column names appear as expected. Set `apop_opts.db_name_column` to the name of a column in your query result to use that column name for row names.

Sample uses, given apop_data set `d`:

```
int row_name_count = d->names->rowct
int col_name_count = d->names->colct
int text_name_count = d->names->textct

//Manually add names in sequence:
```

```
apop_name_add(d->names, "the vector", 'v');
apop_name_add(d->names, "row 0", 'r');
apop_name_add(d->names, "row 1", 'r');
apop_name_add(d->names, "row 2", 'r');
apop_name_add(d->names, "numeric column 0", 'c');
apop_name_add(d->names, "text column 0", 't');
apop_name_add(d->names, "The name of the data set.", 'h');

//or append several names at once
apop_data_add_names(d, 'c', "numeric column 1", "numeric column 2", "numeric column 3");

//point to element i from the row/col/text names:

char *rowname_i = d->names->row[i];
char *colname_i = d->names->col[i];
char *textname_i = d->names->text[i];

//The vector also has a name:
char *vname = d->names->vector;
```

- apop_name_add : add one name
- apop_data_add_names : add a sequence of names at once
- apop_name_stack : copy the contents of one name list to another
- apop_name_find : find the row/col number for a given name.
- apop_name_print : print the apop_name struct, for diagnostic purposes.

3.1.13 Text data

The apop_data set includes a grid of strings, named `text`, for holding text data.

Text should be encoded in UTF-8. ASCII is a subset of UTF-8, so that's OK too.

There are a few simple forms for handling the `text` element of an `apop_data` set.

- Use apop_text_alloc to allocate the block of text. It is actually a realloc function, which you can use to resize an existing block without leaks. See the example below.
- Use apop_text_set to write text elements. It replaces any existing text in the given slot without memory leaks.
- The number of rows of text data in `tdata` is `tdata->textsize[0]`; the number of columns is `tdata->textsize[1]`.
- Refer to individual elements using the usual 2-D array notation, `tdata->text[row][col]`.
- `x[0]` can always be written as `*x`, which may save some typing. The number of rows is `*tdata->textsize`. If you have a single column of text data (i.e., all data is in column zero), then item `i` is `*tdata->text[i]`. If you know you have exactly one cell of text, then its value is `**tdata->text`.
- After apop_text_alloc, all elements are the empty string `""`, which you can check via

  ```
  if (!strlen(dataset->text[i][j])) printf("<blank>")
  //or
  if (!*dataset->text[i][j]) printf("<blank>")
  ```

  For the sake of efficiency when dealing with large, sparse data sets, all blank cells point to *the same* static empty string, meaning that freeing cells must be done with care. Your best bet is to rely on apop_text_set, apop_text_alloc, and apop_text_free to do the memory management for you.

Here is a sample program that uses these forms, plus a few text-handling functions.

```
#include <apop.h>

int main(){
    apop_query("create table data (name, city, state);"
            "insert into data values ('Mike Mills', 'Rockville', 'MD');"
            "insert into data values ('Bill Berry', 'Athens', 'GA');"
            "insert into data values ('Michael Stipe', 'Decatur', 'GA');");
    apop_data *tdata = apop_query_to_text("select name, city, state from data");
    printf("Customer #1: %s\n\n", *tdata->text[0]);

    printf("The data, via apop_data_print:\n");
    apop_data_print(tdata);

    //the text alloc can be used as a text realloc:
    apop_text_alloc(tdata, 1+tdata->textsize[0], tdata->textsize[1]);
    apop_text_set(tdata, *tdata->textsize-1, 0, "Peter Buck");
    apop_text_set(tdata, *tdata->textsize-1, 1, "Berkeley");
    apop_text_set(tdata, *tdata->textsize-1, 2, "CA");

    printf("\n\nAugmented data, printed via for loop:\n");
    for (int i=0; i< tdata->textsize[0]; i++){
        for (int j=0; j< tdata->textsize[1]; j++)
            printf("%s\t", tdata->text[i][j]);
        printf("\n");
    }

    apop_data *states = apop_text_unique_elements(tdata, 2);
    char *states_as_list = apop_text_paste(states, .between=", ");
    printf("\n States covered: %s\n", states_as_list);
}
```

- apop_data_transpose() : also transposes the text data. Say that you use `dataset = apop_query_to_text("select onecolumn from data");` then you have a sequence of strings, `d->text[0][0], d->text[1][0], ....` After `apop_data dt = apop_data_transpose(dataset)`, you will have a single list of strings, `dt->text[0]`, which is often useful as input to list-of-strings handling functions.

- apop_query_to_text
- apop_text_alloc : allocate or resize the text part of an apop_data set.
- apop_text_set : replace a single cell of the text grid with new text.
- apop_text_paste : convert a table of strings into one long string.
- apop_text_unique_elements : get a sorted list of unique elements for one column of text.
- apop_text_free : you may never need this, because apop_data_free calls it.
- apop_regex : friendlier front-end for POSIX-standard regular expression searching; pulls matches into an apop_data set.
- apop_text_unique_elements

3.1.13.1 Generating factors

Factor is jargon for a numbered category. Number-crunching programs prefer integers over text, so we need a function to produce a one-to-one mapping from text categories into numeric factors.

A *dummy* is a variable that is either one or zero, depending on membership in a given group. Some methods (typically when the variable is an input or independent variable in a regression) prefer dummies; some

methods (typically for outcome or dependent variables) prefer factors. The functions that generate factors and dummies will add an informational page to your apop_data set with a name like <categories for your_column> listing the conversion from the artificial numeric factor to the original data. Use apop_data_get_factor_names to get a pointer to that page.

You can use the factor table to translate from numeric categories back to text (though you probably have the original text column in your data anyway).

Having the factor list in an auxiliary table makes it easy to ensure that multiple apop_data sets use the same single categorization scheme. Generate factors in the first set, then copy the factor list to the second, then run apop_data_to_factors on the second:

```
apop_data_to_factors(d1);
d2->more = apop_data_copy(apop_data_get_factor_names(d1));
apop_data_to_factors(d2);
```

See the documentation for apop_logit for a sample linear model using a factor dependent variable and dummy independent variable.

- apop_data_to_dummies
- apop_data_to_factors
- apop_data_get_factor_names

3.1.14 Input text file formatting

This reference section describes the assumptions made by apop_text_to_db and apop_text_to_data.

Each row of the file will be converted to one record in the database or one row in the matrix. Values on one row are separated by delimiters. Fixed-width input is also OK; see below.

By default, the delimiters are set to "|,\t", meaning that a pipe, comma, or tab will delimit separate entries. To change the default, use an argument to apop_text_to_db or apop_text_to_data like .delimiters="\t" or .delimiters="|".

The input text file must be UTF-8 or traditional ASCII encoding. Delimiters must be ASCII characters. If your data is in another encoding, try the POSIX-standard iconv program to filter the data to UTF-8.

- The character after a backslash is read as a normal character, even if it is a delimiter, #, or ". \li If a field contains several such special characters, surround it by \c "s. The surrounding marks are stripped and the text read verbatim.
- Text does not need to be delimited by quotes (unless there are special characters). If a text field is quote-delimited, I'll strip them. E.g., "Males, 30-40", is an OK column name, as is "Males named \"Joe\"".
- Everything after an unprotected # is taken to be comments and ignored.
- Blank lines (empty or consisting only of white space) are also ignored.
- If you are reading into the gsl_matrix element of an apop_data set, all text fields are taken as zeros. You will be warned of such substitutions unless you set apop_opts.verbose==0 beforehand. For mixed text/numeric data, try using apop_text_to_db and then apop_query_to_mixed_data.
- There are often two delimiters in a row, e.g., "23, 32,, 12". When it's two commas like this, the user typically means that there is a missing value and the system should insert a NAN; when it is two tabs in a row, this is typically just a formatting glitch. Thus, if there are multiple delimiters in a row, I check whether the second (and subsequent) is a space or a tab; if it is, then it is ignored, and if it is any other delimiter (including the end of the line) then a NaN is inserted.

If this rule doesn't work for your situation, you can explicitly insert a note that there is a missing data point. E.g., try:

```
perl -pi.bak -e 's/,,/,NaN,/g' data_file
```

If you have missing data delimiters, you will need to set apop_opts.nan_string to text that matches the given format. E.g.,

```
//Apophenia's default NaN string, matching NaN, nan, or NAN, but not Nancy:
apop_opts.nan_string = "NaN";
//Popular alternatives:
apop_opts.nan_string = "Missing";
apop_opts.nan_string = ".";

//Or, turn off nan-string checking entirely with:
apop_opts.nan_string = NULL;
```

SQLite stores these NaN-type values internally as `NULL`; that means that functions like apop_query_to_data will convert both your `nan_string` string and `NULL` to `NaN`.

- The system uses the standards for C's `atof()` function for floating-point numbers: INFINITY, -INFINITY, and NaN work as expected.
- If there are row names and column names, then the input will not be perfectly square: there should be no first entry in the sequence of column names like `row names`. That is, for a 100x100 data set with row and column names, there are 100 names in the top row, and 101 entries in each subsequent row (name plus 100 data points).
- White space before or after a field is ignored. So `1, 2,3, 4 , 5, " six ",7` is eqivalent to `1,2,3,4,5," six ",7`.
- NUL characters (`'\0'`) are treated as white space, so if your fields have NULs as padding, you should have no problem. NULs inside of a string terminates the string as it always does in C.
- Fixed-width formats are supported (for plain ASCII encoding only), but you have to provide a list of field ending positions. For example, given

  ```
  NUMLEOL
  123AABB
  456CCDD
  ```

  and `.field_ends=(int[]){3, 5, 7}`, we have three columns, named NUM, LE, and OL. The names can be read from the first row by setting `.has_row_names='y'`.

3.2 Databases

These are convenience functions to handle interaction with SQLite or mySQL/mariaDB. They open one and only one database, and handle most of the interaction therewith for you.

You will probably first use apop_text_to_db to pull data into the database, then apop_query to clean the data in the database, and finally apop_query_to_data to pull some subset of the data out for analysis.

- In all cases, your query may be in `printf` form. For example:

  ```
  char tabname[] = "demographics";
  char colname[] = "heights";
  int min_height = 175;
  apop_query("select %s from %s where %s > %i", colname, tabname, colname, min_height);
  ```

See the Database moments (plus pow()!) section below for not-SQL-standard math functions that you can use when sending queries from Apophenia, such as `pow`, `stddev`, or `sqrt`.

- apop_text_to_db : Read a text file on disk into the database. Data analysis projects often start with a call to this.
- apop_data_print : If you include the argument `.output_type='d'`, this prints your apop_data set to the database.
- apop_query : Manipulate the database, return nothing (e.g., insert rows or create table).
- apop_db_open : Optional, for when you want to use a database on disk.
- apop_db_close : A useful (and in some cases, optional) companion to apop_db_open.
- apop_table_exists : Check to make sure you aren't reinventing or destroying data. Also, a clean way to drop a table.

- Apophenia reserves the right to insert temp tables into the opened database. They will all have names beginning with `apop_`, so the reader is advised to not generate tables with such names, and is free to ignore or delete any such tables that turn up.
- If you need to deal with two databases, use SQL's `attach database`. By default with SQLite, Apophenia opens an in-memory database handle. It is a sensible workflow to use the faster in-memory database as the primary database, and then attach an on-disk database to read in data and write final output tables.

3.2.1 Extracting data from the database

- apop_db_to_crosstab : take up to three columns in the database (row, column, value) and produce a table of values.
- apop_query_to_data
- apop_query_to_float
- apop_query_to_mixed_data
- apop_query_to_text
- apop_query_to_vector

3.2.2 Writing data to the database

See the print functions at Legible output. E.g.

```
apop_data_print(yourdata, .output_type='d', .output_name="dbtab");
```

3.2.3 Command-line utilities

A few functions have proven to be useful enough to be worth breaking out into their own programs, for use in scripts or other data analysis from the command line:

- The `apop_text_to_db` command line utility is a wrapper for the apop_text_to_db command.
- The `apop_db_to_crosstab` function is a wrapper for the apop_db_to_crosstab function.

3.2.4 Database moments (plus pow()!)

SQLite lets users define new functions for use in queries, and Apophenia uses this facility to define a few common functions.

- `select ran() from table` will produce a new random number between zero and one for every row of the input table, using `gsl_rng_uniform`.

- The SQL standard includes the `count(x)` and `avg(x)` aggregators, but statisticians are usually interested in higher moments as well—at least the variance. Therefore, SQL queries using the Apophenia library may include any of these moments:

```
select count(x), stddev(x), avg(x), var(x), variance(x), skew(x), kurt(x), kurtosis(x),
std(x), stddev_samp(x), stddev_pop(x), var_samp(x), var_pop(x)
from table
group by whatever
```

`var` and `variance`; `kurt` and `kurtosis` do the same thing; choose the one that sounds better to you. Kurtosis is the fourth central moment by itself, not adjusted by subtracting three or dividing by variance squared. `var`, `var_samp`, `stddev` and `stddev_samp` give sample variance/standard deviation; `variance`, `var_pop`, `std` and `stddev_pop` give population standard deviation. The plethora of variants are for my↩ SQL compatibility.

- The var/skew/kurtosis functions calculate sample moments. If you want the second population moment, multiply the variance by $(n-1)/n$; for the third population moment, multiply the skew by $(n-1)(n-2)/n^2$. The equation for the unbiased sample kurtosis as calculated in Appendix M of *Modeling with Data* is not quite as easy to adjust.

- Also provided: wrapper functions for standard math library functions—`sqrt(x)`, `pow(x,y)`, `exp(x)`, `log(x)`, and trig functions. They call the standard math library function of the same name to calculate $\sqrt{x}$, x^y, e^x, $\ln(x)$, $\sin(x)$, $\arcsin(x)$, et cetera. For example:

```
select sqrt(x), pow(x,0.5), exp(x), log(x), log10(x),
    sin(x), cos(x), tan(x), asin(x), acos(x), atan(x)
from table
```

- The `ran()` function calls `gsl_rng_uniform` to produce a uniform draw between zero and one. It uses the stock of RNGs from apop_rng_get_thread.

Here is a test script using many of the above.

```
#include <apop.h>

#define Diff(L, R) assert(fabs((L)-(R))<1e-4));
#define Diff2(L, R) assert(fabs((L)-(R))<1e-3));
#define getrow(rowname) apop_data_get(row, .colname=#rowname)

double test_all(apop_data *row){
    Diff(gsl_pow_2(getrow(root)), getrow(rr))
    Diff(getrow(ln), getrow(L10)*log(10))
    Diff(getrow(rr), getrow(rragain))
    Diff(getrow(one), 1)
    return 0;
}

int main(){
```

```
    apop_opts.db_engine='s'; //SQLite only.

    //create a table with two rows.
    //We didn't explicitly open a db with apop_db_open,
    //so this will be an in-memory SQLite db.
    apop_query("create table a(b); "
               "insert into a values(1); "
               "insert into a values(1); "

                "create table randoms as "
                "select ran() as rr "
                /* join to create 2^13=8192 rows*/
                "from a,a,a,a,a,a,a,a,a,a,a,a,a;");
    apop_data *d = apop_query_to_data(
            "select rr, sqrt(rr) as root, "
            "log(rr) as ln, log10(rr) as L10, "
            "exp(log(rr)) as rragain, "
            "pow(sin(rr),2)+pow(cos(rr),2) as one "
            "from randoms");
    apop_map(d, .fn_r=test_all);

    //the pop variance of a Uniform[0,1]=1/12; kurtosis=1/80.
    Apop_col_tv(d, "rr", rrow);
    Diff(apop_var(rrow)*8191./8192., 1/12. );
    Diff(apop_vector_kurtosis(rrow)*8191./8192., 1/80.);//approx.

    Diff(apop_query_to_float("select stddev(rr) from randoms"),
                sqrt(1/12.)*8192./8191);


    //compare the std dev of a uniform as reported by the
    //database routine, the matrix routine, and math.
    apop_query("create table atab (a numeric)");
    for (int i=0; i< 2e5; i++)
        apop_query("insert into atab values(ran())");
    apop_query("create table powa as "
            "select a, pow(a, 2) as sq, pow(a, 0.5) as sqrt "
            "from atab");

    double db_pop_stddev = apop_query_to_float("select stddev_pop(a) from powa");
    d = apop_query_to_data("select * from powa");
    //get the full covariance matrix, but just use the (0,0)th elmt.
    apop_data *cov = apop_data_covariance(d);
    double matrix_pop_stddev = sqrt(apop_data_get(cov)*(d->matrix->size1/(d->matrix->size1-1.)
      ));
    Diff(db_pop_stddev, matrix_pop_stddev);
    double actual_stddev = sqrt(2*gsl_pow_3(.5)/3);
    Diff2(db_pop_stddev, actual_stddev);

    float sq_mean = apop_query_to_float("select avg(sq) from powa");
    float actual_sq_mean = 1./3;
    Diff2(sq_mean, actual_sq_mean);

    float sqrt_mean = apop_query_to_float("select avg(sqrt) from powa");
    float actual_sqrt_mean = 2./3;
    Diff2(sqrt_mean, actual_sqrt_mean);
}
```

3.3 Models

See apop_model for an overview of the intent and basic use of the apop_model struct.

This segment goes into greater detail on the use of existing apop_model objects. If you need to write a new model, see Writing new models.

The `estimate` function will estimate the parameters of your model. Just prep the data, select a model, and

produce an estimate:

```
apop_data *data = apop_query_to_data("select outcome, in1, in2, in3 from dataset
      ");
apop_model *the_estimate = apop_estimate(data, apop_probit);
apop_model_print(the_estimate);
```

Along the way to estimating the parameters, most models also find covariance estimates for the parameters, calculate statistics like log likelihood, and so on, which the final print statement will show.

The `apop_probit` model that ships with Apophenia is unparameterized: `apop_probit->parameters==NULL`. The output from the estimation, `the_estimate`, has the same form as `apop_probit`, but `the_estimate->parameters` has a meaningful value.

Apophenia ships with many well-known models for your immediate use, including probability distributions, such as the apop_normal, apop_poisson, or apop_beta models. The data is assumed to have been drawn from a given distribution and the question is only what distributional parameters best fit. For example, given that the data is Normally distributed, find μ and σ via `apop_estimate(your_data, apop_normal)`.

There are also linear models like apop_ols, apop_probit, and apop_logit. As in the example, they are on equal footing with the distributions, so nothing keeps you from making random draws from an estimated linear model.

- If you send a data set with the `weights` vector filled, apop_ols estimates Weighted OLS.
- If the dependent variable has more than two categories, the apop_probit and apop_logit models estimate a multinomial logit or probit.
- There are separate apop_normal and apop_multivariate_normal functions because the parameter formats are slightly different: the univariate Normal keeps both μ and σ in the vector element of the parameters; the multivariate version uses the vector for the vector of means and the matrix for the Σ matrix. The univariate version is so heavily used that it merits a special-case model.

See the Models page for a list of models shipped with Apophenia, including popular favorites like apop_beta, apop_binomial, apop_iv (instrumental variables), apop_kernel_density, apop_loess, apop_lognormal, apop_pmf (see Empirical distributions and PMFs (probability mass functions) below), and apop_poisson.

Simulation models seem to not fit this form, but you will see below that if you can write an objective function for the `p` method of the model, you can use the above tools. Notably, you can estimate parameters via maximum likelihood and then give confidence intervals around those parameters.

More estimation output

In the apop_model returned by apop_estimate, you will find:

- The actual parameter estimates are in an apop_data set at `your_model->parameters`.
- A pointer to the apop_data set used for estimation, named `data`.
- Scalar statistics of the model listed in the output model's `info` group, which may include some hypothesis tests, a list of expected values, log likelihood, AIC, AIC_c, BIC, et cetera. These can be retrieved via a form like

  ```
  apop_data_get(your_model->info, .rowname="log likelihood");
  //or
  apop_data_get(your_model->info, .rowname="AIC");
  ```

  If those are not necessary, adding to your model an apop_parts_wanted_settings group with its default values (see below on settings groups) signals to the model that you want only the parameters and to not waste possibly significant CPU time on covariances, expected values, et cetera. See the apop_parts_wanted_settings documentation for examples and further refinements.

- In many cases, covariances of the parameters as a page appended to the parameters; retrieve via

```
apop_data *cov = apop_data_get_page(your_model->parameters, "<Covariance>");
```

- Typically for regression-type models, the table of expected values (typically including expected value, actual value, and residual) is a page stapled to the main info page. Retrieve via:

```
apop_data *predict = apop_data_get_page(your_model->info, "<Predicted>");
```

See individual model documentation for what is provided by any given model.

Post-estimation uses

But we expect much more from a model than estimating parameters from data.

Continuing the above example where we got an estimated Probit model named `the_estimate`, we can interrogate the estimate in various familiar ways:

```
apop_data *expected_value = apop_predict(NULL, the_estimate);

double density_under =  apop_cdf(expected_value, the_estimate);

apop_data *draws = apop_model_draws(the_estimate, .count=1000);
```

Data format for regression-type models

3.3.1 Parameterizing or initializing a model

The models that ship with Apophenia have the requisite procedures for estimation, making draws, and so on, but have `parameters==NULL` and `settings==NULL`. The model is thus, for many purposes, incomplete, and you will need to take some action to complete the model. As per the examples to follow, there are several possibilities:

- Estimate it! Almost all models can be sent with a data set as an argument to the `apop_estimate` function. The input model is unchanged, but the output model has parameters and settings in place.
- If your model has a fixed number of numeric parameters, then you can set them with apop_model_set_parameters.
- If your model has a variable number of parameters, you can directly set the `parameters` element via apop_data_falloc. For most purposes, you will also need to set the `msize1`, `msize2`, `vsize`, and `dsize` elements to the size you want. See the example below.
- Some models have disparate, non-numeric settings rather than a simple matrix of parameters. For example, an kernel density estimate needs a model as a kernel and a base data set, which can be set via apop_model_set_settings.

Here is an example that shows the options for parameterizing a model. After each parameterization, 20 draws are made and written to a file named draws-[modelname].

```
#include <apop.h>

#define print_draws(mm) apop_data_print(apop_model_draws(mm, 20),\
                                        .output_name= "draws-" #mm);

int main(){
    apop_model *uniform_20 = apop_model_set_parameters(apop_uniform, 0, 20);
    apop_data *d = apop_model_draws(uniform_20, 10);
```

```
    //Estimate a Normal distribution from the data:
    apop_model *N = apop_estimate(d, apop_normal);
    print_draws(N);

    //estimate a one-dimensional multivariate Normal from the data:
    apop_model *mvN = apop_estimate(d, apop_multivariate_normal);
    print_draws(mvN);

    //fixed parameter list:
    apop_model *std_normal = apop_model_set_parameters(apop_normal, 0, 1);
    print_draws(std_normal);

    //variable-size parameter list:
    apop_model *std_multinormal = apop_model_copy(apop_multivariate_normal);
    std_multinormal->msize1 =
    std_multinormal->msize2 =
    std_multinormal->vsize =
    std_multinormal->dsize = 3;
    std_multinormal->parameters = apop_data_falloc((3, 3, 3),
                                1,  1, 0, 0,
                                1,  0, 1, 0,
                                1,  0, 0, 1);
    print_draws(std_multinormal);

    //estimate a KDE using the defaults:
    apop_model *k = apop_estimate(d, apop_kernel_density);
    print_draws(k);

    /*A KDE estimation consists of filling an apop_kernel_density_settings group,
      so we can set it to use a Normal(, 2) kernel via: */
    apop_model *k2 = apop_model_set_settings(apop_kernel_density,
                        .base_data=d,
                        .kernel = apop_model_set_parameters(apop_normal, 0, 2));
    print_draws(k2);
}
```

3.3.2 Filtering & updating

The model structure makes it easy to generate new models that are variants of prior models. Bayesian updating, for example, takes in one apop_model that we call the prior, one apop_model that we call a likelihood, and outputs an apop_model that we call the posterior. One can produce complex models using simpler transformations as well. For example, apop_model_fix_params will set the free parameters of an input model to a fixed value, thus producing a model with fewer parameters. To transform a Normal(μ, σ) into a one-parameter Normal(μ, 1):

```
apop_model *N_sigma1 = apop_model_fix_params(apop_model_set_parameters(apop_normal, NAN, 1));
```

This can be used anywhere the original Normal distribution can be. To give another example, if we need to truncate the distribution in the data space:

```
//The constraint function.
double over_zero(apop_data *in, apop_model *m){
    return apop_data_get(in) > 0;
}

apop_model *trunc = apop_model_dconstrain(.base_model=N_sigma1,
                                          .constraint=over_zero);
```

Chaining together simpler transformations is an easy method to produce models of arbitrary detail. In the following example:

- Nature generated data using a mixture of three Poisson distributions, with $\lambda = 2.8$, 2.0, and 1.3. The resulting model is generated using apop_model_mixture.

- Not knowing the true distribution, the analyst models the data with a single Poisson (λ) distribution with a prior on λ. The prior selected is a truncated Normal(2, 1), generated by sending the stock apop_normal model to the data-space constraint function apop_dconstrain.
- The apop_update function takes three arguments: the data set, which comes from draws from the mixture, the prior, and the likelihood. It produces an output model which, in this case, is a P↩ MF describing a distribution over λ, because a truncated Normal and a Poisson are not conjugate distributions. Knowing that it is a PMF, the `->data` element holds a set of draws from the posterior.
- The analyst would like to present an approximation to the posterior in a simpler form, and so finds the parameters μ and σ of the Normal distribution that is closest to that posterior.

Here is a program—almost a single line of code—that builds the final approximation to the posterior model from the subcomponents, including draws from Nature and the analyst's prior and likelihood:

```
#include <apop.h>

// For defining the bounds the data-constraining function
// needs to enforce.
double greater_than_zero(apop_data *d, apop_model *m){
    return apop_data_get(d) > 0;
}

int main(){
    apop_model_print (
        apop_estimate(
             apop_update(
                apop_model_draws(
                    apop_model_mixture(
                        apop_model_set_parameters(apop_poisson, 2.8),
                        apop_model_set_parameters(apop_poisson, 2.0),
                        apop_model_set_parameters(apop_poisson, 1.3)
                    ),
                    1e4
                ),
                apop_model_dconstrain(
                    .base_model=apop_model_set_parameters(apop_normal, 2, 1),
                    .constraint=greater_than_zero
                ),
                apop_poisson
            )->data,
            apop_normal
        )
    );
}
```

3.3.3 Model methods

- apop_estimate : estimate the parameters of the model with data.
- apop_predict : the expected value function.
- apop_draw : random draws from an estimated model.
- apop_p : the probability of a given data set given the model.
- apop_log_likelihood : the log of apop_p
- apop_score : the derivative of apop_log_likelihood
- apop_model_print : write model components to the screen or a file
- apop_model_copy : duplicate a model

- apop_model_set_parameters : Use this to convert a Normal(μ, σ) with unknown μ and σ into a Normal(0, 1), for example.
- apop_model_free
- apop_model_clear , apop_prep : remove the parameters from a parameterized model. Used infrequently.
- apop_model_draws : many random draws from an estimated model.
- apop_update : Bayesian updating
- apop_model_coordinate_transform : apply an invertible transformation to the data space
- apop_model_dconstrain : constrain the data space of a model to a subspace. E.g., truncate a Normal distribution so $x > 0$.
- apop_model_fix_params : hold some parameters constant
- apop_model_mixture : a linear combination of models
- apop_model_cross : If (d_1) has a Normal (μ, σ) distribution and d_2 has an independent Poisson (λ) distribution, then (d_1, d_2) has an `apop_model_cross(apop_normal, apop_poisson)` distribution with parameters (μ, σ, λ).

3.3.4 Settings groups

Describing a statistical, agent-based, social, or physical model in a standardized form is difficult because every model has significantly different settings. An MLE requires a method of search (conjugate gradient, simplex, simulated annealing), and a histogram needs the number of bins to be filled with data.

So, the apop_model includes a single list which can hold an arbitrary number of settings groups, like the search specifications for finding the maximum likelihood, a histogram for making random draws, and options about the model type.

Settings groups are automatically initialized with default values when needed. If the defaults do no harm, then you don't need to think about these settings groups at all.

Here is an example where a settings group is worth tweaking: the apop_parts_wanted_settings group indicates which parts of the auxiliary data you want.

```
1 apop_model *m = apop_model_copy(apop_ols);
2 Apop_settings_add_group(m, apop_parts_wanted, .covariance='y');
3 apop_model *est = apop_estimate(data, m);
```

Line one establishes the baseline form of the model. Line two adds a settings group of type apop_parts_wanted_settings to the model. By default other auxiliary items, like the expected values, are set to `'n'` when using this group, so this specifies that we want covariance and only covariance. Having stated our preferences, line three does the estimation we want.

Notice that the `_settings` ending to the settings group's name isn't written—macros make it happen. The remaining arguments to `Apop_settings_add_group` (if any) follow the Designated initializers syntax of the form `.setting=value`.

There is an apop_model_copy_set macro that adds a settings group when it is first copied, joining up lines one and two above:

```
apop_model *m = apop_model_copy_set(apop_ols, apop_parts_wanted, .
      covariance='y');
```

Settings groups are copied with the model, which facilitates chaining estimations. Continuing the above example, you could re-estimate to get the predicted values and covariance via:

```
Apop_settings_set(est, apop_parts_wanted, predicted, 'y');
apop_model *est2 = apop_estimate(data, est);
```

Maximum likelihood search has many settings that could be modified, and so provides another common example of using settings groups:

```
apop_model *the_estimate = apop_estimate(data, apop_probit);

//Redo the Probit's MLE search using Newton's Method:
Apop_settings_add_group(the_estimate, apop_mle, .verbose='y',
                        .tolerance=1e-4, .method="Newton");
apop_model *re_est = apop_estimate(data, the_estimate);
```

To clarify the distinction between parameters and settings, note that parameters are estimated from the data, often via a maximum likelihood search. In an ML search, the method of search, the number of bins in a histogram, or the number of steps in a simulation would be held fixed as the search iterates over possible parameters (and if these settings do change, then that is a meta-model that could be encapsulated into another apop_model). As a consequence, parameters are always numeric, while settings may be any type.

- Apop_settings_set, for modifying a single setting, doesn't use the designated initializers format.
- Because the settings groups are buried within the model, debugging them can be a pain. Here is a documented macro for `gdb` that will help you pull a settings group out of a model for your inspection, to cut and paste into your `.gdbinit`. It shouldn't be too difficult to modify this macro for other debuggers.

```
define get_group
    set $group = ($arg1_settings *) apop_settings_get_grp( $arg0, "$arg1", 0 )
    p *$group
end
document get_group
Gets a settings group from a model.
Give the model name and the name of the group, like
get_group my_model apop_mle
and I will set a gdb variable named $group that points to that model,
which you can use like any other pointer. For example, print the contents with
p *$group
The contents of $group are printed to the screen as visible output to this macro.
end
```

For using a model, that's all of what you need to know. For details on writing a new settings group, see Writing new settings groups .

- Apop_settings_add_group
- Apop_settings_set
- Apop_settings_get : get a single element from a settings group.
- Apop_settings_get_group : get the whole settings group.

3.3.5 Data format for regression-type models

Regression-type estimations typically require a constant column. That is, the 0th column of the data is a constant (one), so the parameter β_0 is slightly special in corresponding to a constant rather than a variable.

Some stats packages implicitly assume a constant column, which the user never sees. This violates the principle of transparency upon which Apophenia is based. Given a data matrix X with the estimated parameters β, if the model asserts that the product $X\beta$ has meaning, then you should be able to easily calculate that product. With a ones column, a dot product is one line: `apop_dot(x, your_est->parameters)`; without a ones column, one would basically have to construct one (using `gsl_matrix_set_all` and `apop_data_stack`).

Each regression-type estimation has one dependent variable and several independent. In the end, we want the dependent variable to be in the vector element. Removing a column from a `gsl_matrix` and adjusting all subsequent columns is relatively difficult, because (like most structs built with the aim of very efficient processing) the struct depends on an equal spacing in memory between each element.

The automatic case

We can resolve both the need for a ones column and for having the dependent column in the vector at the same time. Given a data set with no vector element and the dependent variable in the first column of the matrix, we can copy the dependent variable into the vector and then replace the first column of the matrix with ones. The result fits all of the above expectations.

You as a user merely have to send in a apop_data set with `NULL` vector and a dependent column in the first column. If the data is coming from the database, then the query is natural:

```
apop_data *regression_data = apop_query_to_data("select depvar, indyvar1,
       indyvar2, indyvar3 from dataset");
apop_model_print(apop_estimate(regression_data, apop_ols));
```

The already-prepped case

If your data has a vector element, then the prep routines won't change anything. If you don't want to use a constant column, or your data has already been prepped by an estimation, then this is what you want.

```
apop_data *regression_data = apop_query_to_mixed_data("vmmm", "select
       depvar, indyvar1, indvar2, indvar3 from dataset");
apop_model_print(apop_estimate(regression_data, apop_logit));
```

3.4 Tests & diagnostics

Here is the model for all hypothesis testing within Apophenia:

- Calculate a statistic.
- Describe the distribution of that statistic.
- Work out how much of the distribution is (above|below|closer to zero than) the statistic.

There are a handful of named tests that produce a known statistic and then compare to a known distribution, like apop_test_kolmogorov or apop_test_fisher_exact. For traditional distributions (Normal, t, χ^2), use the apop_test convenience function.

In especially common cases, like the parameters from an OLS regression, the commonly-associated t test is included as part of the estimation output, typically as a row in the `info` element of the output apop_model.

- apop_test

- apop_paired_t_test
- apop_f_test
- apop_t_test
- apop_test_anova_independence
- apop_test_fisher_exact
- apop_test_kolmogorov
- apop_estimate_coefficient_of_determination
- apop_estimate_r_squared

See also these Monte Carlo methods:

- apop_bootstrap_cov
- apop_jackknife_cov

To give another example of testing, here is a function that was briefly a part of Apophenia, but seemed a bit out of place. Here it is as a sample:

```
// Input: any vector, which will be normalized in place. Output: 1 - the p-value
// for a chi-squared test to answer the question, "with what confidence can I
// reject the hypothesis that the variance of my data is zero?"

double apop_test_chi_squared_var_not_zero(gsl_vector *in){
    Apop_stopif(!in, return NAN, 0, "input vector is NULL. Doing nothing.");
    apop_vector_normalize(in, .normalization_type='s');
    double sum=apop_vector_map_sum(in, gsl_pow_2);
    return gsl_cdf_chisq_P(sum, in->size);
}
```

Or, consider the Rao statistic, $\frac{\partial}{\partial\beta}\log L(\beta)' I^{-1}(\beta)\frac{\partial}{\partial\beta}\log L(\beta)$ where L is a model's likelihood function and I its information matrix. In code:

```
apop_data * infoinv = apop_model_numerical_covariance(data,
      your_model);
apop_data * score = &(apop_data*){.vector=apop_numerical_gradient(data, your_model)};
apop_data * stat = apop_dot(apop_dot(score, infoinv), score);
```

Given the correct assumptions, this is $\sim \chi^2_m$, where m is the dimension of β, so the odds of a Type I error given the model is:

```
double p_value = apop_test(stat, "chi squared", beta->size);
```

Generalized parameter tests

But if your model is not from the textbook, then you have the tools to apply the above three-step process to the parameters of any apop_model.

- Model parameters are a statistic, and you know that `apop_estimate(your_data, your_model)` will output a model with a `parameters` element.
- apop_parameter_model will return an apop_model describing the distribution of these parameters.

- We now have the two ingredients to send to apop_cdf, which takes in a model and a data point and returns the area under the data point.

Defaults for the parameter models are filled in via bootstrapping or resampling, meaning that if your model's parameters are decidedly off the Normal path, you can still test claims about the parameters.

The introductory example in A quick overview ran a standard OLS regression, whose output includes some standard hypothesis tests; to conclude, let us go the long way and replicate those results via the general apop_parameter_model mechanism. The results here will of course be identical, but the more general mechanism can be used in situations where the standard models don't apply.

The first part of this program is identical to the introductory program, using `ss08pdc.csv` if you have downloaded it as per the instructions in A quick overview, or a simple sample data set if not. The second half executes the three steps uses many of the above features: one of the inputs to apop_parameter_model (which row of the parameter set to use) is sent by adding a settings group, we pull that row into a separate data set using Apop_r, and we set its vector value by referring to it as the -1st element.

```
#include <apop.h>
#include <unistd.h>

int main(void){
    char *datafile = (access("ss08pdc.csv", R_OK)!=-1) ? "ss08pdc.csv" : "data";
    apop_text_to_db(.text_file=datafile, .tabname="dc");
    apop_data *data = apop_query_to_data("select log(pincp+10), agep, sex "
                                    "from dc where agep+ pincp+sex is not null and pincp>=0");
    apop_model *est = apop_estimate(data, apop_ols);
    apop_model_show(est);

    Apop_settings_add_group(est, apop_pm, .index =1);
    apop_model *first_param_distribution = apop_parameter_model(data, est);

    Apop_row(est->parameters, 1, param);
    double area_under_p = apop_cdf(param, first_param_distribution);

    apop_data_set(param, 0, -1, .val=0);
    double area_under_zero = apop_cdf(param, first_param_distribution);
    printf("reject the null for agep with %g percent confidence.\n",
                                  100*(2*fabs(area_under_p-area_under_zero)));
}
```

Note that the procedure did not assume the model parameters had a certain form. It queried the model for the distribution of parameter `agep`, and if the model didn't have a closed-form answer then a distribution via bootstrap would be provided. Then that model was queried for its CDF. [The procedure does assume a symmetric distribution. Fixing this is left as an exercise for the reader.] For a model like OLS, this is entirely overkill, which is why OLS provides the basic hypothesis tests automatically. But for models where the distribution of parameters is unknown or has no closed-form solution, this may be the only recourse.

3.5 Optimization

This section includes some notes on the maximum likelihood routine. As in the section on writing models above, if a model has a `p` or `log_likelihood` method but no `estimate` method, then calling `apop_estimate(your_data, your_model)` executes the default estimation routine of maximum likelihood.

If you are a not a statistician, then there are a few things you will need to keep in mind:

- Physicists, pure mathematicians, and the GSL minimize; economists, statisticians, and Apophenia maximize. If you are doing a minimization, be sure that your function returns minus the objective function's value.

- The overall setup is about estimating the parameters of a model with data. The user provides a data set and an unparameterized model, and the system tries parameterized models until one of them is found to be optimal. The data is fixed. The optimization tries a series of parameterized models, searching for the one that is most likely. In a non-stats setting, you may have `NULL` data.

- Because the unit of analysis is a parameterized model, not just parameters, you need to have an apop_model wrapping your objective function.

This example, to be discussed in detail below, optimizes Rosenbrock's banana function, $(1-x)^2 + s(y-x^2)^2$, where the scaling factor s is fixed ahead of time, say at 100.

```
#include <apop.h>

typedef struct {
    double scaling;
} coeff_struct;

long double banana (double *params, coeff_struct *in){
    return (gsl_pow_2(1-params[0])
                + in->scaling*gsl_pow_2(params[1]-gsl_pow_2(params[0])));
}

long double ll (apop_data *d, apop_model *in){
    return - banana(in->parameters->vector->data, in->more);
}

int main(){
    coeff_struct co = {.scaling=100};
    apop_model *b = &(apop_model) {"aBananas!", .log_likelihood= ll,
                    .vsize=2, .more = &co, .more_size=sizeof(coeff_struct)};
    Apop_model_add_group(b, apop_mle, .verbose='y', .method="NM simplex");
    Apop_model_add_group(b, apop_parts_wanted);
    apop_model *e1 = apop_estimate(NULL, b);
    apop_model_print(e1);

    //for printing the path below
    apop_data *bfgs_path = NULL;
    Apop_settings_set(b, apop_mle, path, &bfgs_path);

    Apop_settings_set(b, apop_mle, method, "BFGS cg");
    apop_model *e2 = apop_estimate(NULL, b);
    apop_model_print(e2);

    apop_data_show(bfgs_path);

    gsl_vector *one = apop_vector_fill(gsl_vector_alloc(2), 1, 1);
    assert(apop_vector_distance(e1->parameters->vector, one) < 1e-2);
    assert(apop_vector_distance(e2->parameters->vector, one) < 1e-2);
}
```

The `banana` function returns a single number to be minimized. You will need to write an apop_model to send to the optimizer, which is a two step process: write a log likelihood function wrapping the real objective function (`ll`), and a model that uses that log likelihood (`b`).

- The `.vsize=2` part of the declaration of `b` on the second line of `main()` specified that the model's parameters are a vector of size two. That is, the list of `doubles` to send to `banana` is set in `in->parameters->vector->data`.

- The `more` element of the apop_model structure is designed to hold any arbitrary structure of size `more_size`, which is useful for models that require additional constants or other settings, like the `coeff_struct` here. See Writing new settings groups for more on handling model settings.

- Statisticians want the covariance and basic tests about the parameters. This line shuts off all auxiliary calculations:

  ```
  Apop_settings_add_group(your_model, apop_parts_wanted);
  ```

  See the documentation for apop_parts_wanted_settings for details about how this works. It can also offer quite the speedup: especially for high-dimensional problems, finding the covariance matrix without any additional information can take dozens of evaluations of the objective function for each evaluation that is part of the search itself.

- MLEs have an especially large number of parameter tweaks that could be made; see the apop_mle_settings page.

- As a useful diagnostic, you can add a `NULL` apop_data set to the MLE settings in the `.path` slot, and it will be allocated and filled with the sequence of points tried by the optimizer.

- The program has some extras above and beyond the necessary: it uses two methods (notice how easy it is to re-run an estimation with an alternate method, but the syntax for modifying a setting differs from the initialization syntax) and checks that the results are accurate.

3.5.1 Setting Constraints

The problem is that the parameters of a function must not take on certain values, either because the function is undefined for those values or because parameters with certain values would not fit the real-world problem.

If you give the optimizer an unconstrained likelihood function plus a separate constraint function, apop_maximum_likelihood will combine them to a function that is continuous at the constraint boundary, but which is guaranteed to never have an optimum outside of the constraint.

A constraint function must do three things:

- If the constraint does not bind (i.e. the parameter values are OK), then it must return zero.

- If the constraint does bind, it must return a penalty, that indicates how far off the parameter is from meeting the constraint.

- If the constraint does bind, it must set a return vector that the likelihood function can take as a valid input. The penalty at this returned value must be zero.

The idea is that if the constraint returns zero, the log likelihood function will return the log likelihood as usual, and if not, it will return the log likelihood at the constraint's return vector minus the penalty. To give a concrete example, here is a constraint function that will ensure that both parameters of a two-dimensional input are both greater than zero, and that their sum is greater than two. As with the constraints for many of the models that ship with Apophenia, it is a wrapper for apop_linear_constraint.

```
static long double greater_than_zero_constraint(apop_data *data, apop_model *v){
    static apop_data *constraint = NULL;
    if (!constraint) constraint= apop_data_falloc((3,3,2), 0,  1, 0,   //0 < 1x + 0y
                                                           0,  0, 1,   //0 < 0x + 1y
                                                           2,  1, 1);  //2 < 1x + 1y
    return apop_linear_constraint(v->parameters->vector, constraint, 1e-3);
}
```

- apop_linear_constraint()

3.5.2 Notes on simulated annealing

For convex optimizations, methods like conjugate gradient search work well, and for relatively smooth optimizations, the Nelder-Mead simplex algorithm is a good choice. For situations where the surface being searched may have several local optima and be otherwise badly behaved, there is simulated annealing.

Simulated annealing is a controlled random walk. As with the other methods, the system tries a new point, and if it is better, switches. Initially, the system is allowed to make large jumps, and then with each iteration, the jumps get smaller, eventually converging. Also, there is some decreasing probability that if the new point is less likely, it will still be chosen. Simulated annealing is best for situations where there may be multiple local optima. Early in the random walk, the system can readily jump from one to another; later it will fine-tune its way toward the optimum. The number of points tested is determined by the parameters of the simulated colling program, not the values returned by the likelihood function. If you know your function is globally convex (as are most standard probability functions), then this method is overkill.

3.5.3 Useful functions

- apop_estimate_restart : Restarting an MLE with different settings can improve results.
- apop_maximum_likelihood : Rarely called directly. If a model has no `estimate` element, call apop_estimate to prep the model and run an MLE.
- apop_model_numerical_covariance
- apop_numerical_gradient

3.6 Assorted

Some functions for missing data:

- apop_data_listwise_delete
- apop_ml_impute

A few more descriptive methods:

- apop_matrix_pca : Principal component analysis
- apop_anova : One-way or two-way ANOVA tables
- apop_rake : Iterative proportional fitting on large, sparse tables

General utilities:

- Apop_stopif : Apophenia's error-handling and warning-printing macro.
- apop_opts : the global options
- apop_system : a printf-style wrapper around the standard `system` function.

A few more math utilities:

- apop_matrix_is_positive_semidefinite
- apop_matrix_to_positive_semidefinite

- apop_generalized_harmonic
- apop_multivariate_gamma
- apop_multivariate_lngamma
- apop_rng_alloc

4 Empirical distributions and PMFs (probability mass functions)

The apop_pmf model wraps an apop_data set so it can be read as an empirical model, with a likelihood function (equal to the associated weight for observed values and zero for unobserved values), a random number generator (which simply makes weighted random draws from the data), and so on. Setting it up is a model estimation from data like any other, done via apop_estimate(`your_data`, apop_pmf).

You have the option of cleaning up the data before turning it into a PMF. For example...

```
apop_data_pmf_compress(your_data);          //remove duplicates
apop_data_sort(your_data);
apop_vector_normalize(your_data->weights);  //weights sum to one
apop_model *a_pmf = apop_estimate(your_data, apop_pmf);
```

These are largely optional.

- The CDF is calculated based on the percent of the weights between the zeroth row of the PMF and the row specified. This generally makes more sense after apop_data_sort.
- Compression produces a corresponding improvement in efficiency when first calculating CDFs, but is otherwise not necessary.
- Sorting or normalizing is not necessary for making draws or getting a likelihood or log likelihood.

It is the `weights` vector that holds the density represented by each row; the rest of the row represents the coordinates of that density. If the input data set has no `weights` segment, then I assume that all rows have equal weight.

For a PMF model, the `parameters` are `NULL`, and the `data` itself is used for calculation. Therefore, modifying the data post-estimation can break some internal settings set during estimation. If you modify the data, throw away any existing PMFs (via apop_model_free) and re-estimate a new one.

4.1 Comparing histograms

Using apop_data_pmf_compress puts the data into one bin for each unique value in the data set. You may instead want bins of fixed with, in the style of a histogram, which you can get via apop_data_to_bins. It requires a bin specification. If you send a `NULL` binspec, then the offset is zero and the bin size is big enough to ensure that there are $\sqrt{N}$ bins from minimum to maximum. The binspec will be added as a page to the data set, named "<binspec>". See the apop_data_to_bins documentation on how to write a custom bin spec.

There are a few ways of testing the claim that one distribution equals another, typically an empirical PMF versus a smooth theoretical distribution. In both cases, you will need two distributions based on the same binspec.

For example, if you do not have a prior binspec in mind, then you can use the one generated by the first call to the histogram binning function to make sure that the second data set is in sync:

```
apop_data_to_bins(first_set, NULL);
apop_data_to_bins(second_set, apop_data_get_page(first_set, "<binspec>")
      );
```

You can use apop_test_kolmogorov or apop_histograms_test_goodness_of_fit to generate the appropriate statistics from the pairs of bins.

Kernel density estimation will produce a smoothed PDF. See apop_kernel_density for details. Or, use apop_vector_moving_average for a simpler smoothing method.

- apop_data_pmf_compress() : merge together redundant rows in a data set before calling apop_estimate(your_data, apop_pmf); optional.
- apop_vector_moving_average() : smooth a vector (e.g., your_pmf->data->weights) via moving average.
- apop_histograms_test_goodness_of_fit() : goodness-of-fit via χ^2 statistic
- apop_test_kolmogorov() : goodness-of-fit via Kolmogorov-Smirnov statistic
- apop_kl_divergence() : measure the information loss from one (typically empirical) distribution to another distribution.

5 Writing new models

The apop_model is intended to provide a consistent expression of *any* model that (implicitly or explicitly) expresses a likelihood of data given parameters, including traditional linear models, textbook distributions, Bayesian hierarchies, microsimulations, and any combination of the above. The unifying feature is that all of the models act over some data space and some parameter space (in some cases one or both is the empty set), and can assign a likelihood for a fixed pair of parameters and data given the model. This is a very broad requirement, often used in the statistical literature. For discussion of the theoretical structures, see *A Useful Algebraic System of Statistical Models* (PDF).

This page is about writing new models from scratch, beginning with basic models and on up to models with arbitrary internal settings, specific methods of Bayesian updating using your model as a prior or likelihood, and so on. I assume you have already read Models on using models and have tried a few things with the canned models that come with Apophenia, so you already know how a user handles basic estimation, adding a settings group, and so on.

This page includes:

- A walkthrough of writing a new model from scratch.
- Writing new settings groups, covering the writing of *ad hoc* structures to hold model- or method-specific details, like the number of periods for burning in an MCMC run or the number of bins in a histogram.
- Registering new methods in vtables, covering the means of writing special-case routines for functions that are not part of the apop_model itself, including the score or conjugate prior/likelihood pairs for apop_update.
- The data elements, a detailed list of the requirements for the non-function elements of an apop_model.
- Methods, a detailed list of requirements for the function elements of an apop_model.

5.1 A walkthrough

Users are encouraged to always use models via the helper functions, like apop_estimate or apop_cdf. The helper functions do some boilerplate error checking, and call defaults as needed. For example, if your model has a `log_likelihood` method but no `p` method, then apop_p will use exp(`log_likelihood`). If you don't give an `estimate` method, then `apop_estimate` will call apop_maximum_likelihood.

So the game in writing a new model is to write just enough internal methods to give the helper functions what they need. In the not-uncommon best case, all you need to do is write a log likelihood function or an RNG.

Here is how one would set up a model that could be estimated using maximum likelihood:

- Write a likelihood function. Its header will look like this:

  ```
  long double new_log_likelihood(apop_data *data, apop_model *m);
  ```

  where `data` is the input data, and `m` is the parametrized model (i.e. your model with a `parameters` element already filled in by the caller). This function will return the value of the log likelihood function at the given parameters.

- Write the object:

```
apop_model *your_new_model = &(apop_model){"The Me distribution",
            .vsize=n0, .msize1=n1, .msize2=n2, .dsize=nd,
            .log_likelihood = new_log_likelihood };
```

- The first element is the `.name`, a human-language name for your model.
- The `vsize`, `msize1`, and `msize2` elements specify the shape of the parameter set. For example, if there are three numbers in the vector, then set `.vsize=3` and omit the matrix sizes. The default model prep routine will call `new_est->parameters = apop_data_alloc(vsize, msize1, msize2)`.
- The `dsize` element is the size of one random draw from your model.
- It's common to have [the number of columns in your data set] parameters; this count will be filled in if you specify `-1` for `vsize`, `msize(1|2)`, or `dsize`. If the allocation is exceptional in a different way, then you will need to allocate parameters by writing a custom `prep` method for the model.
- Is this a constrained optimization? Add a `.constraint` element for those too. See Setting Constraints for more.

You already have more than enough that something like this will work (the `dsize` is used for random draws):

```
apop_model *estimated = apop_estimate(your_data, your_new_model);
```

Once that baseline works, you can fill in other elements of the apop_model as needed.

For example, if you are using a maximum likelihood method to estimate parameters, you can get much faster estimates and better covariance estimates by specifying the dlog likelihood function (aka the score):

```
void apop_new_dlog_likelihood(apop_data *d, gsl_vector *gradient, apop_model *m){
    //do algebra here to find df/dp0, df/dp1, df/dp2....
    gsl_vector_set(gradient, 0, d_0);
    gsl_vector_set(gradient, 1, d_1);
}
```

The score is not part of the model object, but is registered (see below) using

```
apop_score_insert(apop_new_dlog_likelihood, your_new_model);
```

5.1.1 Threading

Many procedures in Apophenia use OpenMP to thread operations, so assume your functions are running in a threaded environment. If a method can not be threaded, wrap it in an OpenMP critical region. E.g.,

```
void apop_new_dlog_likelihood(apop_data *d, gsl_vector *gradient, apop_model *m){
    #pragma omp critical (newdlog)
    {
        //un-threadable algebra here
    }
    gsl_vector_set(gradient, 0, d_0);
    gsl_vector_set(gradient, 1, d_1);
}
```

5.2 Writing new settings groups

Your model may need additional settings or auxiliary information to function, which would require associating a model-specific struct with the model. A method associated with a model that uses such a struct usually begins with calls like

```
long double ysg_ll(apop_data *d, apop_model *m){
    ysg_settings *sets = apop_settings_get(m, ysg);

    ...
}
```

These model-specific structs are handled as expected by apop_model_copy and apop_model_free, and many functions that modify or transform an apop_model try to handle settings groups as expected. This section describes how to build a settings group so all these automatic steps happen as expected, and your methods can reliably retrieve settings as needed.

But before getting into the detail of how to make model-specific groups of settings work, note that there's a lightweight method of storing sundry settings, so in many cases you can bypass all of the following. The apop_model structure has a `void` pointer named `more` which you can use to point to a model-specific struct. If `more_size` is larger than zero (i.e. you set it to `your_model.more_size=sizeof(your_struct)`), then it will be copied via `memcpy` by apop_model_copy, and freed by apop_model_free. Apophenia's routines will never impinge on this item, so do what you wish with it.

The remainder of this subsection describes the information you'll have to provide to make use of the conveniences described to this point: initialization of defaults, smarter copying and freeing, and adding to an arbitrarily long list of settings groups attached to a model. You will need four items: a typedef for the structure itself, plus init, copy, and free functions. This is the sort of boilerplate that will be familiar to users of object-oriented languages in the style of C++ or Java, but it's really a list of arbitrarily-typed elements, which makes this feel more like LISP. [And being a reimplementation of an existing feature of LISP, this section will be macro-heavy.]

The settings struct will likely go into a header file, so here is a sample header for a new settings group named `ysg_settings`, with a dataset, two sizes, and a reference counter. `ysg` stands for Your Settings Group; replace that substring with your preferred name in every instance to follow.

```
typedef struct {
    int size1, size2;
    char *refs;
    apop_data *dataset;
} ysg_settings;

Apop_settings_declarations(ysg)
```

The first item is a familiar structure definition. The last line is a macro that declares the init, copy, and free functions discussed below. This is everything you would need in a header file, should you need one. These

are just declarations; we'll write the actual init/copy/free functions below.

The structure itself gets the full name, `ysg_settings`. Everything else is a macro keyed on `ysg`, without the `_settings` part. Because of these macros, your struct name must end in `_settings`.

If you have an especially simple structure, then you can generate the three functions with these three macros in your `.c` file:

```
Apop_settings_init(ysg, )
Apop_settings_copy(ysg, )
Apop_settings_free(ysg, )
```

These macros generate appropriate functions to do what you'd expect: allocating the main structure, copying one struct to another, freeing the main structure. The spaces after the commas indicate that in these cases no special code gets added to the functions that these macros expand into.

You'll never call the generated functions directly; they are called by Apop_settings_add_group, apop_↩ model_free, and other model or settings-group handling functions.

Now that initializing/copying/freeing of the structure itself is handled, the remainder of this section will be about how to add instructions for the structure internals, like data that is pointed to by the structure elements.

- For the allocate function, use the above form if everything in your code defaults to zero/`NULL`. Otherwise, you will need a new line declaring a default for every element in your structure. There is a macro to help with this too. These macros will define for your use a structure named `in`, and an output pointer-to-struct named `out`. Continuing the above example:

  ```
  Apop_settings_init (ysg,
      Apop_stopif(!in.size1, return NULL, 0, "I need you to give me a value for size1.");
      Apop_varad_set(size2, 10);
      Apop_varad_set(dataset, apop_data_alloc(out->size1, out->size2));
      Apop_varad_set(refs, malloc(sizeof(int)));
      *refs=1;
  )
  ```

  Now, `Apop_settings_add(a_model, ysg, .size1=100)` would set up a group with a 100-by-10 data set, and set the reference counter allocated and to one.

- Some functions do extensive internal copying, so you will need a copy function even if your code has no explicit calls to apop_model_copy. The default above simply copies every element in the structure. Pointers are copied, giving you two pointers pointing to the same data. We have to be careful to prevent double-freeing later.

  ```
  //The elements of the set to copy are all copied by the function's boilerplate,
  //and then make one additional modification:
  Apop_settings_copy (ysg,
      #pragma omp critical (ysg_refs)
          (*refs)++;
  )
  ```

- The struct itself is freed by boilerplate code, but add code in the free function to free data pointed to by pointers in the main structure. The macro defines a pointer-to-struct named `in` for your use. Continuing the example:

  ```
  Apop_settings_free (ysg,
      #pragma omp critical (ysg_refs)
          if (!(--in->refs)) {
              free(in->dataset);
              free(in->refs);
          }
  )
  ```

With those three macros in place and the header as above, Apophenia will treat your settings group like any other, and users can use Apop_settings_add_group to populate it and attach it to any model.

5.3 Registering new methods in vtables

The settings groups are for adding arbitrary model-specific nouns; vtables are for adding arbitrary model-specific verbs.

Many functions (e.g., entropy, the dlog likelihood, Bayesian updating) have special cases for well-known models like the Normal distribution. Any function may maintain a registry of models and associated special-case procedures, aka a vtable.

Lookups happen based on a hash that takes into account the elements of the model that will be used in the calculation. For example, the `apop_update_hash` takes in two models and calculates the hash based on the address of the prior's `draw` method and the likelihood's `log_likelihood` or `p` method. Thus, a vtable lookup for new models that re-use the same methods (at the same addresses in memory) will still find the same special-case function.

If you need to deregister the function, use the associated deregister function, e.g. `apop_update_vtable_drop(apop_beta, apop_binomial)`. You can guarantee that a method will not be re-added by following up the `_drop` with, e.g., `apop_update_vtable_add(NULL, apop_beta, apop_binomial)`.

The steps for adding a function to an existing vtable:

- See apop_update, apop_score, apop_predict, apop_model_print, and apop_parameter_model for examples and procedure-specific details.
- Write a function following the given type definition, as listed in the function's documentation.
- Use the associated `_vtable_add` function to add the function and associate it with the given model. For example, to add a Beta-binomial routine named `betabinom` to the registry of Bayesian updating routines, use `apop_update_vtable_add(betabinom, apop_beta, apop_binomial)`.
- Place a call to `..._vtable_add` in the `prep` method of the given model, thus ensuring that the auxiliary functions are registered after the first time the model is sent to apop_estimate.

The easiest way to set up a new vtable is to copy/paste/modify an existing one. Briefly:

- See the existing setups in the vtables portion of `apop.h`.
- Cut/paste one and do a search and replace to change the name to match your desired use.
- Set the typedef to describe the functions that get added to the vtable.
- Rewrite the hash function to check the part of the inputs that interest you. For example, the update vtable associates functions with the `draw`, `log_likelihood`, and `methods` of the model. A model where these elements are identical will still match even if other elements are different.

5.4 The data elements

The remainder of this section covers the detailed expectations regarding the elements of the apop_model structure. I begin with the data (non-function) elements, and then cover the method (function) elements. Some of the following will be requirements for all models and some will be advice to authors; I use the accepted definitions of `"must", "shall", "may"` and related words.

5.4.1 Data

- Each row of the `data` element is treated as a single observation by many functions. For example, apop_bootstrap_cov depends on each row being an iid observation to function correctly. Calculating

the Bayesian Information Criterion (BIC) requires knowing the number of observations in the data, and assumes that row count==observation count. For complex data, the apop_data_pack and apop_data_unpack functions can help with this.

- Some functions (bootstrap again, or many uses of apop_kl_divergence) use apop_draw to use your model's RNG (or a default) to draw a value, write it to the matrix element of the data set, and then move on to an estimation or other step. In this case, the data sent in will be entirely in the `->matrix` element of the apop_data set sent to model methods. Your `likelihood`, `p`, `cdf`, and `estimate` routines must accept data as a single row of the matrix of the apop_data set for such functions to work. They may accept other formats. Tip: you can use apop_data_pack and apop_data_unpack to convert a structured set to a single row and back again.

- Your routines may accept other data formats, as per contract with the user. For example, regression-type functions use a function named `ols_shuffle` to convert a matrix where the first column is the dependent variable to a data set with dependent variable in the vector and a column of ones in the first matrix column.

5.4.2 Parameters, vsize, msize1, msize2

- The sizes will be used by the `prep` method of the model; see below. Given the model `m` and its elements `m.vsize`, `m.msize1`, `m.msize2`, functions that need to allocate a parameter set will do so via `apop_data_alloc(m.vsize, m.msize1, m.msize2)`.

5.4.3 Info

- The first page, which should be named `<info>`, is typically a list of scalars. Nothing is guaranteed, but the elements may include:

- AIC: `Aikake Information Criterion`

- AIC_c: AIC with a finite sample correction. "*Generally, we advocate the use of AIC_c when the ratio n/K is small (say < 40)*" [Kenneth P. Burnham, David R. Anderson: *Model Selection and Multi-Model Inference*, p 66, emphasis in original.]

- BIC: `Bayesian Information Criterion`

- R squared

- R squared adj

- log likelihood

- status [0=OK, nozero=other].

For those elements that require a count of input data, the calculations assume each row in the input apop_data set is a single datum.

Get these via, e.g., `apop_data_get(your_model->info, .rowname="log likelihood")`. When writing for any arbitrary function, be prepared to handle `NaN`, indicating that the element is not calculated or saved in the info page by the given model.

For OLS-type estimations, each row corresponds to the row in the original data. For filling in of missing data, the elements may appear anywhere, so the row/col indices are essential.

5.4.4 settings, more

In object-oriented jargon, settings groups are the private elements of the data set, to be pulled out in certain contexts, and ignored in all others. Therefore, there are no rules about internal use. The `more` element of the apop_model provides a lightweight means of attaching an arbitrary struct to a model. See Writing new settings groups above for details.

- As many settings groups of different types as desired can be added to a single apop_model.
- One apop_model can not hold two settings groups of the same type. Re-additions cause the removal of the previous version of the group.
- If the `more` pointer points to a structure or value (let it be `ss`), then `more_size` must be set to `sizeof(ss)`.

5.5 Methods

5.5.1 p, log_likelihood

- Function headers look like `long double your_p_or_ll(apop_data *d, apop_model *params)`.
- The inputs are an apop_data set and an apop_model, which should include the elements needed to fully estimate the probability/likelihood (probably a filled `->parameters` element, possibly a settings group added by the user).
- Assume that the parameters have been set, by users via apop_estimate or apop_model_set_parameters, or by apop_maximum_likelihood by its search algorithms. If the parameters are necessary, the function shall check that the parameters are not `NULL` and set the model's `error` element to `'p'` if they are missing.
- Return `NaN` on errors. If an error in the input model is found, the function may set the input model's `error` element to an appropriate `char` value.
- If your model includes both `log_likelihood` and `p` methods, it must be the case that `log(p(d, m))` equals `log_likelihood(d, m)` for all `d` and `m`. This implies that `p` must return a value ≥ 0. Note that apop_maximum_likelihood will accept functions where `p` returns a negative value, but diagonstics that depend on log likelihood like AIC will return NaN.
- If observations are assumed to be iid, you may be able to use apop_map_sum to write the core of the log likelihood function.

5.5.2 prep

- Function header looks like `void your_prep(apop_data *data, apop_model *params)`.
- Re-prepping a model after it has already been prepped shall have no effect. Where there is ambiguity with the other requirements, this takes precedence.
- The model's `data` pointer shall be set to point to the input data.
- The `info` element shall be allocated and its title set to `<Info>`.
- If `vsize`, `msize1`, or `msize2` are -1, then the prep function shall set them to the width of the input data.
- If `dsize` is -1, then the prep function shall set it to the width of the input data.

- If the `parameters` element is not allocated, the function shall allocate it via `apop_data_alloc(vsize, msize1, msize2)` (or equivalent).
- The default is apop_model_clear. It does all of the above.
- The input data may be modified by the prep routine. For example, the apop_ols prep routine shuffles a single input matrix as described above under `data`, and the apop_pmf prep routine calls apop_data_pmf_compress on the input data.
- The prep routine may initialize any desired settings groups. Unless otherwise stated, these should not be removed if they are already there, so that users can override defaults by adding a settings group before starting an estimation.
- If any functions associated with the model need to be added to a vtable (see above), the registration shall happen here. Registration may also happen elsewhere.

5.5.3 estimate

- Function header looks like `void your_estimate(apop_data *data, apop_model *params)`. It modifies the input model, and returns nothing. Note that this is different from the wrapper function, apop_estimate, which makes a copy of its input model, preps it, and then calls the `estimate` function with the prepeped copy.
- Assume that the prep routine has already been run. Notably, this means that parameters have been allocated.
- Assume that the `parameters` hold garbage (as in a `malloc` without a subsequent assignment to the `malloc`-ed space).
- The function shall set the `parameters` of the input model. For consistency with other models, the estimate should be the maximum likelihood estimate, unless otherwise documented.
- Additional settings may be set.
- The model's <Info> page may be filled with statistics, as discussed at infosubsec. For scalars like log likelihood and AIC, use apop_data_add_named_elmt.
- Data should not be modified by the `estimate` routine; any changes to the data made by `estimate` must be documented.
- The default called by apop_estimate is apop_maximum_likelihood.
- If errors occur during processing, set the model's `error` element to a single character. Documentation should include the list of error characters and their meaning.

5.5.4 draw

- Function header looks like `void your_draw(double *out*, *gsl_rng* r,` apop_model `*params)`
- Assume that model `paramters` are set, via apop_estimate or apop_model_set_parameters. The author of the draw method should check that `parameters` are not `NULL` if needed and fill the output with NaNs if necessary parameters are not set.
- Caller inputs a pointer-to-`double` of length `dsize`; user is expected to make sure that there is adequate space. Caller also inputs a `gsl_rng`, already allocated (probably via apop_rng_alloc, possibly from apop_rng_get_thread).

- The function shall fill the space pointed to by the input pointer with a random draw from the data space, where the likelihood of any given observation is proportional to its likelihood as given by the `p` method. Data shall be reduced to a single vector via apop_data_pack if it is not already a single vector.

5.5.5 cdf

- Function header looks like `long double your_cdf(apop_data *d, apop_model *params).`
- Assume that `parameters` are set, via apop_estimate or apop_model_set_parameters. The author of the CDF method should check that `parameters` are not `NULL` and return NaN if necessary parameters are not set.
- The CDF method must accept data as a single row of data in the `matrix` of the input apop_data set (as per a draw produced using the `draw` method). May accept other formats.
- Returns the percentage of the likelihood function $\leq$ the first row of the input data. The definition of $\leq$ is chosen by the model author.
- If one is not already present, an `apop_cdf_settings` group may be added to the model to store temp data. See the apop_cdf function for details.

5.5.6 constraint

- Function header looks like `long double your_constraint(apop_data *data, apop_model *params).`
- Assume that `parameters` are set, via apop_estimate, apop_model_set_parameters, or the internals of an MLE search. The author of the constraint method should check that `parameters` are not `NULL` and return NaN if necessary parameters are not set.
- See apop_linear_constraint for a useful basis and/or example. Many constraints can be written as wrappers for this function.
- If the constraint is met, then return zero.
- If the constraint fails, then (1) move the `parameters` in the input model to a constraint-satisfying value, and (2) return the distance between the input parameters and what you've moved the parameters to. The choice of within-bounds parameters and distance function is left to the author of the constraint function.

6 Module Index

6.1 Modules

Here is a list of all modules:

7 Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

8 Module Documentation

8.1 Models

Detailed Description

This section is a detailed description of the stock models that ship with Apophenia. It is a reference. For an explanation of what to do with an apop_model, see Models.

The primary questions one has about a model in practice are what format the input data should take and what to expect of an estimated output.

Generally, the input data consists of an apop_data set where each row is a single observation. Details beyond that are listed below.

The output after running apop_estimate to produce a fitted model are generally found in three places: the vector of the output parameter set, its matrix, or a new settings group. The basic intuition is that if the parameters are always a short list of scalars, they are in the vector; if there exists a situation where they could take matrix form, the parameters will be in the matrix; if they require more structure than that, they will be a settings group.

If the basic structure of the apop_data set is unfamiliar to you, see Data sets, which will discuss the basic means of getting data out of a struct. For example, the estimated apop_normal distribution has the mean in position zero of the vector and the standard deviation in position one, so they could be extracted as follows:

```
apop_data *d = apop_text_to_data("sample data from before")
apop_model *out = apop_estimate(d, apop_normal);
double mu = apop_data_get(out>parameters, 0);
double sigma = apop_data_get(out>parameters, 1);

//What is the p-value of test whose null hypothesis is that =3.3?
printf ("pval=%g\n", apop_test(3.3, "normal", mu, sigma);
```

See Models for discussion of how to pull settings groups using Apop_settings_get (for one item) or apop_settings_get_group (for a full settings group).

8.1.1 Model Documentation

8.1.1.1 apop_bernoulli

The Bernoulli model: A single random draw with probability p.

Name `Bernoulli distribution`

Input format The Bernoulli parameter p is the percentage of non-zero values in the matrix. Its variance is $p(1-p)$.

Post-estimate data Unchanged.

Post-estimate parameters p is the only element in the vector (e.g., get its value via `double p = apop_data_get(outmodel->parameters);`). A <Covariance> page has the variance of p in the (0,0)th element of the matrix.

postestimate_info Reports `log likelihood`.

RNG Returns zero or one.

8.1.1.2 apop_beta

The beta distribution has two parameters and is restricted to data between zero and one. You may also find apop_beta_from_mean_var to be useful.

Name `Beta distribution`

Input format Any arrangement of scalar values.

Parameter format A vector, v[0]= α; v[1]= β

Post-estimate data Unchanged.

postestimate_info Reports `log likelihood`.

RNG Produces a scalar $\in [0, 1]$.

8.1.1.3 apop_binomial

The multi-draw generalization of the Bernoulli, or the two-bin special case of the Multinomial distribution.

It is implemented as an alias of the apop_multinomial model, except that it has an explicit CDF, we know it has two parameters, and its draw method returns a scalar. I.e., `.vsize==2` and `.dsize==1`.

Input format Each row of the matrix is one observation, consisting of two elements. The number of draws of type zero (sometimes read as 'misses' or 'failures') are in column zero, the number of draws of type one ('hits', 'successes') in column one.

Parameter format a vector, v[0]= n; v[1]= p_1. Thus, p_0 isn't written down; see apop_multinomial for further discussion. If you input $v[1] > 1$ and `apop_opts.verbose >=1`, the log likelihood function will throw a warning. Post-estimate, will have a `<Covariance>` page with the covariance matrix for the ps (n effectively has no variance).

Post-estimate data Unchanged.

RNG The RNG returns a single number representing the success count, not a vector of length two giving both the failure bin and success bin. This is notable because it differs from the input data format, but it tends to be what people expect from a Binomial RNG. For draws with both dimensions (or situations where draws are fed back into the model), use an apop_multinomial model with `.vsize =2`.

8.1.1.4 apop_coordinate_transform

Apply a coordinate transformation of the data to produce a distribution over the transformed data space. This is sometimes called a Jacobian transformation.

Here is an example that replicates the Lognormal distribution. Note the use of apop_model_copy_set to set up a model with the given settings.

```
/* A Lognormal distribution is a transform of the Normal distribution, where
 the data space of the Normal is exponentiated. Thus, to get back to the original data space, take the log
       of the current data.
 */
#include <apop.h>
#define Diff(a, b) assert(fabs((a)-(b)) < 1e-2);

//Use this function to produce test data below.
apop_data *draw_exponentiated_normal(double mu, double sigma, double draws){
    apop_model *n01 = apop_model_set_parameters(apop_normal, mu, sigma);
    apop_data *d = apop_data_alloc(draws);
    gsl_rng *r = apop_rng_alloc(13);
    for (int i=0; i< draws; i++) apop_draw(gsl_vector_ptr(d->vector,i), r, n01);
    apop_vector_exp(d->vector);
    return d;
}

// The transformed-to-base function and its derivative for the Jacobian:
apop_data *rev(apop_data *in){ return apop_map(in, .fn_d=log, .part='a'); }

/*The derivative of the transformed-to-base function. */
double inv(double in){return 1./in;}
double rev_j(apop_data *in){ return fabs(apop_map_sum(in, .fn_d=inv, .part='a')); }

int main(){
    apop_model *ct = apop_model_coordinate_transform(
                        .transformed_to_base= rev, .jacobian_to_base=rev_j,
                        .base_model=apop_normal);
    //Apop_model_add_group(ct, apop_parts_wanted);//Speed up the MLE.
```

```
    //make fake data
    double mu=2, sigma=1;
    apop_data *d = draw_exponentiated_normal(mu, sigma, 2e5);

    //If we correctly replicated a Lognormal, mu and sigma will be right:
    apop_model *est = apop_estimate(d, ct);
    apop_model_free(ct);
    Diff(apop_data_get(est->parameters, 0), mu);
    Diff(apop_data_get(est->parameters, 1), sigma);

    /*The K-L divergence between our Lognormal and the stock Lognormal
      should be small. Try it with both the original params and the estimated ones. */
    apop_model *ln = apop_model_set_parameters(apop_lognormal, mu, sigma);
    apop_model *ln2 = apop_model_copy(apop_lognormal);
    ln2->parameters = est->parameters;
    Diff(apop_kl_divergence(ln, ln2,.draw_ct=1000), 0);
    Diff(apop_kl_divergence(ln, est,.draw_ct=1000), 0);
}
```

Name `Fill me`

Input format The input data is sent to the first model, so use the input format for that model.

Post-estimate data Unchanged.

Settings apop_coordinate_transform_settings

8.1.1.5 apop_cross

A cross product of models. Generate via apop_model_cross .

For the case when you need to bundle two uncorrelated models into one larger model. For example, the prior for a multivariate normal (whose parameters are a vector of means and a covariance matrix) is a Multivariate Normal-Wishart pair.

Name `Cross product of models`

Input format There are two means of handling the input format. If the settings group attached to the data set has a non-`NULL` `splitpage` element, then append the second data set as an additional page to the first data set, and name the second set with the name you listed in `splitpage`; see the example.

If `splitpage` is `NULL`, then I will send the same data set to both models.

Parameter format currently `NULL`; check the sub-models for their parameters.

Post-estimate data Unchanged.

Settings apop_cross_settings

8.1.1.6 apop_dconstrain

A model that constrains the base model to within some data constraint. E.g., truncate $P(d)$ to zero for all d outside of a given constraint. Generate using apop_model_dconstrain .

The log likelihood works by using the `base_model` log likelihood, and then scaling it based on the part of the base model's density that is within the constraint. If you have an easy means of specifying what that density is, please do, as in the example. If you do not, the log likelihood will calculate it by making `draw_ct` random draws from the base model and checking whether they are in or out of the constraint. Because this default method is stochastic, there is some loss of precision, and conjugate gradient methods may get confused.

Here is an example that makes a few draws and estimations from data-constrained models. Note the use of apop_model_set_settings to prepare the constrained models.

Name `Data-constrained model`

Input format That of the base model.

Parameter format That of the base model. In fact, the `parameters` element is a pointer to the base model `parameters`, so both are modified simultaneously.

Post-estimate data Unchanged.

RNG Draw from the base model; if the draw is outside the constraint, throw it out and try again.

Settings apop_dconstrain_settings

Examples

```
#include <apop.h>

//The constraint function.
double over_zero(apop_data *in, apop_model *m){
    return apop_data_get(in) > 0;
}

//The optional scaling function.
double in_bounds(apop_model *m){
    double z = 0;
    gsl_vector_view vv = gsl_vector_view_array(&z, 1);
    return 1- apop_cdf(&((apop_data){.vector=&vv.vector}), m);
}

int main(){
    /*Set up a Normal distribution, with data truncated to be nonnegative.
      This version doesn't use the in_bounds function above, and so the
      default scaling function is used.*/
    gsl_rng *r = apop_rng_alloc(213);
    apop_model *norm = apop_model_set_parameters(apop_normal, 1.2, 0.8);
    apop_model *trunc = apop_model_set_settings(apop_dconstrain,
                            .base_model=apop_model_copy(norm),
                            .constraint=over_zero, .draw_ct=5e4, .rng=r);

    //make draws. Currently, you need to prep the model first.
    apop_prep(NULL, trunc);
    apop_data *d = apop_model_draws(trunc, 1e5);

    //Estimate the parameters given the just-produced data:
    apop_model *est = apop_estimate(d, trunc);
    apop_model_print(est);
    assert(apop_vector_distance(est->parameters->vector, norm->parameters->vector)<1e-1
      );

    //Generate a data set that is truncated at zero using alternate means
    apop_data *normald = apop_model_draws(apop_model_set_parameters(apop_normal, 0, 1), 5e4);
    for (int i=0; i< normald->matrix->size1; i++){
        double *d = apop_data_ptr(normald, i);
        if (*d < 0) *d *= -1;
    }

    //this time, use an unparameterized model, and the in_bounds fn
    apop_model *re_trunc = apop_model_set_settings(apop_dconstrain,
                            .base_model=apop_normal,
                            .constraint=over_zero, .scaling=in_bounds);

    apop_model *re_est = apop_estimate(normald, re_trunc);
    apop_model_print(re_est);
    assert(apop_vector_distance(re_est->parameters->vector,
                apop_vector_fill(gsl_vector_alloc(2), 0, 1))<1e-1);
    apop_model_free(trunc);
}
```

8.1.1.7 apop_dirichlet

A multivariate generalization of the Beta distribution.

Name `Dirichlet distribution`

Input format Each row of your data matrix is a single observation.

Parameter format The estimated parameters are in the output model's `parameters->vector`. The size of the model is determined by the width of your input data set, so later RNG draws, &c will match in size.

Post-estimate data Unchanged.

RNG A call to `gsl_ran_dirichlet`. Output format is identical to the input data format.

8.1.1.8 apop_exponential

The Exponential distribution.

$Z(\mu, k) = \sum_k 1/\mu e^{-k/\mu}$
$lnZ(\mu, k) = \sum_k -\ln(\mu) - k/\mu$
$dlnZ(\mu, k)/d\mu = \sum_k -1/\mu + k/(\mu^2)$

Some write the function as: $Z(C, k) = \ln C C^{-k}$. If you prefer this form, just convert your parameter via $\mu = \frac{1}{\ln C}$ (and convert back from the parameters this function gives you via $C = \exp(1/\mu)$).

Name `Exponential distribution`

Input format One scalar observation per row (in the `matrix` or `vector`). See also apop_data_rank_compress for means of dealing with one more input data format.

Parameter format μ is in the zeroth element of the vector.

Post-estimate data Unchanged.

RNG Just a wrapper for `gsl_ran_exponential`.

CDF Returns a scalar draw.

8.1.1.9 apop_gamma

$G(x, a, b) = \frac{1}{(\Gamma(a)b^a)} x^{a-1} e^{-x/b}$

$lnG(x, a, b) = -ln\Gamma(a) - alnb + (a-1)ln(x) + -x/b$

$dlnG/da = -\psi(a) - lnb + ln(x)$ (also, $dln\gamma = \psi$)

$dlnG/db = -a/b + x/(b^2)$

Name `Gamma distribution`

Input format A scalar, in the `vector` or `matrix` elements of the input apop_data set.
See also apop_data_rank_compress for means of dealing with one more input data format.

Parameter format First two elements of the vector are $$ and $$.

Post-estimate data Unchanged.

RNG A wrapper for `gsl_ran_gamma`, which returns a scalar.
See the notes for apop_exponential on a popular alternate form.

8.1.1.10 apop_improper_uniform

The improper uniform returns $P(x) = 1$ for every value of x, all the time (and thus, log likelihood(x)=0). It has zero parameters.

- See also the apop_uniform model.

Name `Improper uniform distribution`

Input format Ignored.

Parameter format `NULL`

Post-estimate data Unchanged.

Post-estimate parameters `NULL`

RNG The `draw` function makes no sense, and therefore sets the value in `*out` to `NAN`, returns 1, and prints a warning if `apop_opts.verbose >=1`.

CDF Half of the distribution is less than every given point, so the CDF always returns 0.5. One could perhaps make an argument that this should really be infinity, but a half is more in the spirit of the distribution's use to represent a lack of information.

8.1.1.11 apop_iv

Instrumental variable regression

Operates much like the apop_ols model, but the input parameters also need to have a table of substitutions (like the addition of the `.instruments` setting in the example below).

Which columns substitute where can be specified in your choice of two ways. The first is to use the vector element of the apop_data set to list the column numbers to be substituted (the dependent variable is zero; first independent column is one), and then one column for each item to substitute.

The second method, if the vector of the instrument apop_data set is `NULL`, is to use the column names to find the matching columns in the base data to substitute. This is generally more robust and/or convenient.

- If the `instruments` data set is `NULL` or empty, I'll just run OLS.
- The apop_lm_settings group has a `destroy_data` setting. If you set that to `'y'`, I will overwrite the column in place, saving the trouble of copying the entire data set.

Name `instrumental variables`

Input format See the discussion on the apop_ols page regarding its prep routine. See above regarding the `.instruments` elment of the attached apop_lm_settings group.

Prep_routine See the discussion on the apop_ols page regarding its prep routine.

Parameter format As per apop_ols

Post-estimate data Unchanged.

Examples
```
/* Instrumental variables are often used to deal with variables measured with noise, so
   this example produces a data set with a column of noisy data, and a separate instrument
   measured with greater precision, then sets up and runs an instrumental variable regression.

   To guarantee that the base data set has noise and the instrument is cleaner, the
   procedure first generates the clean data set, then copies the first column to the
   instrument set, then the add_noise function inserts Gaussian noise into the base
   data set. Once the base set and the instrument set have been generated, the setup for
   the IV consists of adding the relevant names and using Apop_model_add_group to add a
   lm (linear model) settings group with an .instrument=instrument_data element.

   In fact, the example sets up a sequence of IV regressions, with more noise each
   time.
   */

   #include <apop.h>
   #define Diff(L, R, eps) Apop_stopif(fabs((L)-(R))>=(eps)), return, 0, "%g is too different \
           from %g (abitrary limit=%g).", (double)(L), (double)(R), eps);

   int datalen =1e4;

   //generate a vector that is the original vector + noise
   void add_noise(gsl_vector *in, gsl_rng *r, double size){
       apop_model *nnoise = apop_model_set_parameters(apop_normal, 0, size)
```

```
      ;
    apop_data *nd = apop_model_draws(nnoise, in->size);
    gsl_vector_add(in, Apop_cv(nd, 0));
    /*for (int i=0; i< in->size; i++){
        double noise;
        apop_draw(&noise, r, nnoise);
        *gsl_vector_ptr(in, i) += noise;
    }*/
    apop_data_free(nd);
    apop_model_free(nnoise);
}

void test_for_unbiased_parameter_estimates(apop_model *m, double tolerance){
        Diff(apop_data_get(m->parameters, 0, -1), -1.4, tolerance);
        Diff(apop_data_get(m->parameters, 1, -1), 2.3, tolerance);
}

int main(){
    gsl_rng *r = apop_rng_alloc(234);

    apop_data *data = apop_data_alloc(datalen, 2);
    for(int i=0; i< datalen; i++){
        apop_data_set(data, i, 1, 100*(gsl_rng_uniform(r)-0.5));
        apop_data_set(data, i, 0, -1.4 + apop_data_get(data,i,1)*2.3);
    }
    apop_name_add(data->names, "dependent", 'c');
    apop_name_add(data->names, "independent", 'c');
    apop_model *oest = apop_estimate(data, apop_ols);
    apop_model_show(oest);

    //the data with no noise will be the instrument.
    gsl_vector *col1 = Apop_cv(data, 1);
    apop_data *instrument_data = apop_data_alloc(data->matrix->size1, 1);
    gsl_vector_memcpy(Apop_cv(instrument_data, 0), col1);
    apop_name_add(instrument_data->names, "independent", 'c');
    Apop_model_add_group(apop_iv, apop_lm, .instruments = instrument_data);

    //Now add noise to the base data four times, and estimate four IVs.
    int tries = 4;
    apop_model *ests[tries];
    for (int nscale=0; nscale<tries; nscale++){
        add_noise(col1, r, nscale==0 ? 0 : pow(10, nscale-tries));
        ests[nscale] = apop_estimate(data, apop_iv);
        if (nscale==tries-1){ //print the one with the largest error.
            printf("\nnow IV:\n");
            apop_model_show(ests[nscale]);
        }
    }

    /* Now test. The parameter estimates are unbiased.
       As we add more noise, the covariances expand.
       Test that the ratio of one covariance matrix to the next
       is less than one, though these are typically very much
       smaller than one (as the noise is an order of magnitude
       larger in each case), and the ratios will be identical
       for each j, k below. */
    test_for_unbiased_parameter_estimates(ests[0], 1e-6);
    for (int i=1; i<tries; i++){
        test_for_unbiased_parameter_estimates(ests[i], 1e-3);

        gsl_matrix *cov = apop_data_get_page(ests[i-1]->parameters, "<Covariance>")->
      matrix;
        gsl_matrix *cov2 = apop_data_get_page(ests[i]->parameters, "<Covariance>")->
      matrix;
        gsl_matrix_div_elements(cov, cov2);
        for (int j =0; j< 2; j++)
            for (int k =0; k< 2; k++)
                assert(gsl_matrix_get(cov, j, k) < 1);
```

```
    }
}
```

8.1.1.12 apop_kernel_density

The kernel density smoothing of a PMF or histogram.

At each point along the histogram, put a distribution (default: Normal(0,1)) on top of the point. Sum all of these distributions to form the output distribution.

Setting up a kernel density consists of setting up a model with the base data and the information about the kernel model around each point. This can be done using the apop_model_set_settings function to get a copy of the base apop_kernel_density model and add a apop_kernel_density_settings group with the appropriate information; see the `main` function of the example below.

Name `kernel density estimate`

Input format One observation per line. Each row in turn will be passed through to the elements of `kernelbase` and optional `set_params` function, so follow the format of the base model.

Parameter format None

Post-estimate data Unchanged.

RNG Randomly selects a data point, then randomly draws from that sub-distribution. Returns 0 on success, 1 if unable to pick a sub-distribution (meaning the weights over the distributions are somehow broken), and 2 if unable to draw from the sub-distribution.

CDF Sums the CDF to the given point of all the sub-distributions.

Settings apop_kernel_density_settings, including:

- `data` a data set, which, if not `NULL` and `base_pmf` is `NULL`, will be converted to an apop_pmf model.
- `base_pmf` This is the preferred format for input data. It is the histogram to be smoothed.
- `kernelbase` The kernel to use for smoothing, with all parameters set and a `p` method. Popular favorites are apop_normal and apop_uniform.
- `set_params` A function that takes in a single number and the model, and sets the parameters accordingly. The function will call this for every point in the data set. Here is the default, which is used if this is `NULL`. It simply sets the first element of the model's parameter vector to the input number; this is appropriate for a Normal distribution, where we want to center the distribution on each data point in turn.

```
1 static void apop_set_first_param(apop_data *in, apop_model *m){
2     apop_data_set(m->parameters, .val= apop_data_get(in));
3 }
```

See the sample code for for a Uniform[0,1] recentered around the first element of the PMF matrix.

Examples This example sets up and uses KDEs based on Normal and Uniform distributions.

```
/* This program draws ten random data points, and then produces two kernel density
estimates: one based on the Normal distribution and one based on the Uniform.

It produces three outputs:
--stderr shows the random draws
--kerneldata is a file written with plot data for both KDEs
--stdout shows instructions to gnuplot, so you can pipe:
./kernel | gnuplot -persist

Most of the code is taken up by the plot() and draw_some_data() functions, which are
straightforward. Notice how plot() pulls the values of the probability distributions
at each point along the scale.

The set_uniform_edges function sets the max and min of a Uniform distribution so that the
given point is at the center of the distribution.
```

```
The first KDE uses the defaults, which are based on a Normal distribution with std dev 1;
the second explicitly sets the .kernel and .set_fn for a Uniform.
*/

#include <apop.h>

void set_uniform_edges(apop_data * r, apop_model *unif){
    apop_data_set(unif->parameters, 0, -1, r->matrix->data[0]-0.5);
    apop_data_set(unif->parameters, 1, -1, r->matrix->data[0]+0.5);
}

void plot(apop_model *k, apop_model *k2){
    apop_data *onept = apop_data_alloc(1,1);
    FILE *outtab = fopen("kerneldata", "w");
    for (double i=0; i<20; i+=0.01){
        apop_data_set(onept, .val=i);
        fprintf(outtab, "%g %g %g\n", i, apop_p(onept, k), apop_p(onept, k2));
    }
    fclose(outtab);
    printf("plot 'kerneldata' using 1:2\n"
           "replot 'kerneldata' using 1:3\n");
}

apop_data *draw_some_data(){
    apop_model *uniform_0_20 = apop_model_set_parameters(apop_uniform, 0, 20);
    apop_data *d = apop_model_draws(uniform_0_20, 10);
    apop_data_print(apop_data_sort(d), .output_pipe=stderr);
    return d;
}

int main(){
    apop_data *d = draw_some_data();
    apop_model *k = apop_estimate(d, apop_kernel_density);
    apop_model *k2 = apop_model_set_settings(apop_kernel_density,
                                             .base_data=d,
                                             .set_fn = set_uniform_edges,
                                             .kernel = apop_uniform);
    plot(k, k2);
}
```

8.1.1.13 apop_loess

Regression via loess smoothing

This uses a somewhat black-box routine, first written by Chamberlain, Devlin, Grosse, and Shyu in 1988, to fit a smoothed series of quadratic curves to the input data, thus producing a curve more closely fitting than a simple regression would.

The curve is basically impossible to describe using a short list of parameters, so the representation is in the form of the `predicted` vector of the `expected` data set; see below.

From the 1992 manual for the package: "The method we will use to fit local regression models is called *loess*, which is short for local regression, and was chosen as the name since a loess is a deposit of fine clay or silt along a river valley, and thus is a surface of sorts. The word comes from the German löss, and is pronounced löíss.''

Name `Loess smoothing`

Input format The data is basically OLS-like: the first column of the data is the dependent variable to be explained; subsequent variables are the independent explanatory variables. Thus, your input data can either have a dependent vector plus explanatory matrix, or a matrix where the first column is the dependent variable.

Unlike with OLS, I won't move your original data, and I won't add a **1**, because that's not really the loess custom. You can of course set up your data that way if you like.

If your data set has a weights vector, I'll use it.

In any case, all data is copied into the model's apop_loess_settings. The code is primarily FORTRAN code from 1988 converted to C; the data thus has to be converted into a relatively obsolete internal format.

Parameter format Unused.

Post-estimate data Unchanged.

Post-estimate parameters None.

Predict Fills in the zeroth column (ignoring and overwriting any data there), and adds an additional page to the input apop_data set named "<Confidence>" with a lower and upper CI for each point.

8.1.1.14 apop_logit

Apophenia makes no distinction between the bivariate logit and the multinomial logit. This does both.

The likelihood of choosing item j is: $e^{x\beta_j}/(\sum_i e^{x\beta_i})$

so the log likelihood is $x\beta_j - ln(\sum_i e^{x\beta_i})$

Name `Logit`

Input format The first column of the data matrix this model expects is zeros, ones, ..., enumerating the factors; to get there, try apop_data_to_factors; if you forget to run it, I'll run it on the first data column for you. The remaining columns are values of the independent variables. Thus, the model will return [(data columns)-1] ×[(option count)-1] parameters. Column names list factors in the dependent variables; row names list the independent variables.

Prep_routine You will probably want to convert some column of your data into factors, via apop_data_to_factors. If you do, then that adds a page of factors to your data set (and of course adjusts the data itself). If I find a factor page, I will use that info; if not, then I will run apop_data_to_factors on the first column (the vector if there is one, else the first column of the matrix.)

Also, if there is no vector, then I will move the first column of the matrix, and replace that matrix column with a constant column of ones, just like with OLS.

Parameter format As above.

Post-estimate data Unchanged.

RNG Much like the apop_ols RNG, qv. Returns the category drawn.

Here is an artifical example which clarifies the simplest use of the model:

```
#include <apop.h>
#include <unistd.h>

char *testfile = "logit_test_data";

//generate a fake data set.
//Notice how the first column is the outcome, just as with standard regression.
void write_data(){
    FILE *f = fopen(testfile, "w");
    fprintf(f, "\
        outcome,A, B \n\
        0, 0, 0     \n\
        1, 1, 1     \n\
        1, .7, .5   \n\
        1, .7, .3   \n\
        1, .3, .7   \n\
        \n\
        1, .5, .5   \n\
        0, .4, .4   \n\
        0, .3, .4   \n\
        1, .1, .3   \n\
        1, .3, .1   ");
    fclose(f);
```

```
}

int main(){
    write_data();
    apop_data *d = apop_text_to_data(testfile);
    Apop_model_add_group(apop_logit, apop_mle, .tolerance=1e-5);
    apop_model *est = apop_estimate(d, apop_logit);
    unlink(testfile);

    /* Apophenia's test suite checks that this code produces
       values close to canned values. As a human, you probably
       just want to print the results to the screen. */
    apop_model_show(est);

    assert(fabs(apop_data_get(est->parameters, .rowname="1")- -1.155026) < 1e-6);
    assert(fabs(apop_data_get(est->parameters, .rowname="A")- 4.039903) < 1e-6);
    assert(fabs(apop_data_get(est->parameters, .rowname="B")- 1.494694) < 1e-6);
}
```

Here is an example using data from a U.S. Congressional vote, including one text variable that has to be converted to factors, and one to convert to dummies. A loop then calculates the customary p-values.

```
// See http://modelingwithdata.org/arch/00000160.htm for context and analysis.
#include <apop.h>

int main(){
    //read the data to db, get the desired columns,
    //prep the two categorical variables
    apop_text_to_db("amash_vote_analysis.csv", .tabname="amash");
    apop_data *d = apop_query_to_mixed_data("mmmtt", "select 0,
       ideology,log(contribs+10) as contribs, vote, party from amash");
    apop_data_to_factors(d); //0th text col -> 0th matrix col
    apop_data_to_dummies(d, .col=1, .type='t', .append='y');

    //Estimate a logit model, get covariances,
    //calculate p values under popular Normality assumptions
    Apop_model_add_group(apop_logit, apop_parts_wanted, .covariance='y');
    apop_model *out = apop_estimate(d, apop_logit);
    apop_model_show(out);
    for (int i=0; i< out->parameters->matrix->size1; i++){
        printf("%s pval:\t%g\n",out->parameters->names->row[i],
        apop_test(apop_data_get(out->parameters, i), "normal", 0, sqrt(apop_data_get(out->parameters->mor
    }
}
```

8.1.1.15 apop_lognormal

The log likelihood function for lognormal distributions:

$f = exp(-(ln(x) - \mu)^2/(2\sigma^2))/(x\sigma\sqrt{2\pi})$

$lnf = -(ln(x) - \mu)^2/(2\sigma^2) - ln(x) - ln(\sigma\sqrt{2\pi})$

Name `Lognormal distribution`

Input format A scalar in the the matrix or vector element of the input apop_data set.

Parameter format Zeroth vector element is the mean of the logged data set; first is the standard deviation of the logged data set.

Post-estimate data Unchanged.

postestimate_info Reports `log likelihood`.

RNG An Apophenia wrapper for the GSL's Normal RNG, exponentiated.

8.1.1.16 apop_mixture

The mixture model transformation: a linear combination of multiple models.

Use apop_model_mixture to produce one of these models. In the examples below, some are generated from unparameterized input models with a form like

```
apop_model *mf = apop_model_mixture(apop_model_copy(apop_normal), apop_model_copy(apop_normal));
Apop_model_add_group(mf, apop_mle, .starting_pt=(double[]){50, 5, 80, 5},
                                   .step_size=3, .tolerance=1e-6);
apop_model_show(apop_estimate(dd, mf));
```

One can also skip the estimation and use already-parameterized models as input to apop_model_mixture, e.g.:

```
apop_model *r_ed = apop_model_mixture(apop_model_set_parameters(apop_normal, 54.6, 5.87),
                        apop_model_set_parameters(apop_normal, 80.1, 5.87));
apop_data *wts = apop_data_falloc((2), 0.36, 0.64);
Apop_settings_add(r_ed, apop_mixture, weights, wts->vector);
printf("LL=%g\n", apop_log_likelihood(dd, r_ed));
```

Notice that the weights vector has to be added after the call to apop_model_mixture. If none is given, then equal weights are assigned to all components of the mixture.

One can think of the estimation as a missing-data problem: each data point originated in one distribution or the other, and if we knew with certainty which data point came from which distribution, then the estimation problem would be trivial: just generate the subsets and call `apop_estimate(dataset1, model1)`, ..., `apop_estimate(datasetn, modeln)` separately. But the assignment of which element goes where is unknown information, which we guess at using an expectation-maximization algorithm. The standard algorithm starts with an initial set of parameters for the models, and assigns each data point to its most likely model. It then re-estimates the model parameters using their subsets. The standard algorithm, see e.g. this PDF, repeats until it arrives at an optimum.

Thus, the log likelihood method for this model includes a step that allocates each data point to its most likely model, and calculates the log likelihood of each observation using its most likely model. [It would be a valuable extension to extend this to not-conditionally IID models. Commit `1ac0dd44` in the repository had some notes on this, now removed.] As a side-effect, it calculates the odds of drawing from each model (the vector). Following the above-linked paper, the probability for a given observation under the mixture model is its probability under the most likely model weighted by the previously calculated λ for the given model.

Apohenia modifies this routine slightly because it uses the same maximum likelihood back-end that most other apop_models use for estimation. The ML search algorithm provides model parameters, then the LL method allocates observations and reports a LL to the search algorithm, then the search algorithm uses its usual rules to step to the next candidate set of parameters. This provides slightly more flexibility in the search.

Estimations of mixture distributions can be sensitive to initial conditions. You are encouraged to try a sequence of random starting points for your model parameters. Some authors recommend plotting the data and eyeballing a guess as to the model parameters.

- A kernel density is a mixture of a large number of homogeneous models, where each is typically centered around a point in your data. For such situations, apop_kernel_density will be easier to use.

Name `Mixture of models`

Input format The same data gets sent to each of the component models of the mixture. Each row is an observation, and the estimation routine assumes that models are conditionally IID (i.e., having chosen what component of the mixture the observation comes from, its likelihood can be calculated independently of all other observations).

Parameter format The parameters are broken out in a readable form in the settings group, so your best bet is to use those. See the sample code for usage.

The `parameter` element is a single vector piling up all elements, beginning with the first $n-1$ weights, followed by an `apop_data_pack` of each model's parameters in sequence. Because all elements are in a single vector, one could run a maximum likelihood search for all components (including the weights) at once. The `log_likehood`, `estimate`, and other methods unpack this vector into its component parts for you.

Post-estimate data Unchanged.

RNG Uses the weights to select a component model, then makes a draw from that component. The model's `dsize` (draw size) element is set when you set up the model in the model's `prep` method (automatically called by apop_estimate, or call it directly) iff all component models have the same `dsize`.

Settings apop_mixture_settings

Examples The first example uses a text file `faith.data`, in the `tests` directory of the distribution.

```
#include <apop.h>

/* This replacement for apop_model_print(in) demonstrates retrieval of the useful
settings: the weights () and list of estimated models. It is here only for
demonstration purposes---it is what apop_model_print(your_mix) will do.
*/
void show_mix(apop_model *in){
    apop_mixture_settings *ms = Apop_settings_get_group(in, apop_mixture);
    printf("The weights:\n");
    apop_vector_print(ms->weights);
    printf("\nThe models:\n");
    for (apop_model **m = ms->model_list; *m; m++) //model_list is a NULL-terminated
       list.
        apop_model_print(*m, stdout);
}

int main(){
    apop_text_to_db("faith.data", "ff");
    apop_data *dd = apop_query_to_data("select waiting from ff");
    apop_model *mf = apop_model_mixture(apop_model_copy(apop_normal), apop_model_copy(apop_normal));

    /* The process is famously sensitive to starting points. Try many random points, or
       eyeball the distribution's plot and guess at the starting values. */
    Apop_model_add_group(mf, apop_mle, .starting_pt=(double[]){50, 5, 80, 5},
                                       .step_size=3, .tolerance=1e-6);
    apop_model *mfe = apop_estimate(dd, mf);
    apop_model_print(mfe, stdout);
    printf("LL=%g\n", apop_log_likelihood(dd, mfe));


    printf("\n\nValues calculated in the source paper, for comparison.\n");
    apop_model *r_ed = apop_model_mixture(
                        apop_model_set_parameters(apop_normal, 54.61364, 5.869089
      ),
                        apop_model_set_parameters(apop_normal, 80.09031, 5.869089
      ));
    apop_data *wts = apop_data_falloc((2), 0.3608498, 0.6391502);
    Apop_settings_add(r_ed, apop_mixture, weights, wts->vector);
    show_mix(r_ed);
    printf("LL=%g\n", apop_log_likelihood(dd, r_ed));
}
```

This example begins with a fixed mixture distribution, and makes assertions about the characteristics of draws from it.

```
#include <apop.h>

/*
Use apop_model_mixture to generate a hump-filled distribution, then find
```

```
the most likely data points and check that they are near the humps.
*/

//Produce a 2-D multivariate normal model with unit covariance and given mean
apop_model *produce_fixed_mvn(double x, double y){
    apop_model *out = apop_model_copy(apop_multivariate_normal);
    out->parameters = apop_data_falloc((2, 2, 2),
                        x, 1, 0,
                        y, 0, 1);
    out->dsize = 2;
    return out;
}

int main(){
    //here's a mean/covariance matrix for a standard multivariate normal.
    apop_model *many_humps = apop_model_mixture(
                        produce_fixed_mvn(5, 6),
                        produce_fixed_mvn(-5, -4),
                        produce_fixed_mvn(0, 1));
    apop_prep(NULL, many_humps);

    int len = 100000;
    apop_data *d = apop_model_draws(many_humps, len);

    gsl_vector *first = Apop_cv(d, 0);
    printf("mu=%g\n", apop_mean(first));
    assert(fabs(apop_mean(first)- 0) < 5e-2);

    gsl_vector *second = Apop_cv(d, 1);
    printf("mu=%g\n", apop_mean(second));
    assert(fabs(apop_mean(second)- 1) < 5e-2);

/*  Abuse the ML imputation routine to search for the input value with the highest
    log likelihood. Do the search via simulated annealing. */

    apop_data *x = apop_data_alloc(1,2);
    gsl_matrix_set_all(x->matrix, NAN);

    apop_opts.stop_on_warning='v';
    Apop_settings_add_group(many_humps, apop_mle, .n_tries=20, .iters_fixed_T=10, .k
      =3,  .method="annealing");
    apop_ml_impute(x, many_humps);

    printf("Optimum found at:\n");
    apop_data_show(x);
    assert(fabs(apop_data_get(x, .col=0)- 0) + fabs(apop_data_get(x, .col=1) - 1)
       < 1e-2);
}
```

8.1.1.17 apop_multinomial

The n–option generalization of the Binomial distribution.

Name `Binomial distribution`

Input format Each row of the matrix is one observation: a set of draws from a single bin. The number of draws of type zero are in column zero, the number of draws of type one in column one, et cetera.

- You may have a set of several Bernoulli-type draws, which could be summed together to form a single Binomial draw. The apop_data_to_dummies function (using the `.keep_first='y'` option), to split a single column of numbers into a sequence of columns, may help with this.

Parameter format The parameters are kept in the vector element of the apop_model parameters element. `parameters->vector->data[0]`==n; `parameters->vector->data[1...]`==p_1....

The numeraire is bin zero, meaning that p_0 is not explicitly listed, but is $p_0 = 1 - \sum_{i=1}^{k-1} p_i$, where k is the number of bins. Conveniently enough, the zeroth element of the parameters vector holds

n, and so a full probability vector can easily be produced by overwriting that first element. For example:

```
1 apop_model *estimated = apop_estimate(your_data, apop_multinomial);
2 int n = apop_data_get(estimated->parameters);
3 apop_data_set(estimated->parameters, .val=1 - (apop_sum(estimated->parameters)-n));
```

And now the parameter vector is a proper list of probabilities.

- Because an observation is a single row, the number of bins, k is set to equal the length of the first row (counting both vector and matrix elements, as appropriate). The covariance matrix will be $k \times k$.
- Each row should sum to N, the number of draws. The estimation routine doesn't check this, but instead uses the average sum across all rows.

Post-estimate data Unchanged.

postestimate_info Reports `log likelihood`.

RNG Returns a single vector of length k, the result of an imaginary tossing of N balls into k urns, with the given probabilities.

8.1.1.18 apop_multivariate_normal

This is the multivariate generalization of the Normal distribution.

Name `Multivariate normal distribution`

Input format Each row of the matrix is an observation.

Parameter format An apop_data set whose vector element is the vector of means, and whose matrix is the covariances.

If you had only one dimension, the mean would be a vector of size one, and the covariance matrix a 1×1 matrix. This differs from the setup for apop_normal, which outputs a single vector with μ in element zero and σ in element one.

After estimation, the <Covariance> page gives the covariance matrix of the means.

Post-estimate data Unchanged.

postestimate_info Reports `log likelihood`.

RNG From Devroye (1986), p 565.

8.1.1.19 apop_normal

You know it, it's your attractor in the limit, it's the Gaussian distribution.

$N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-x^2/2\sigma^2)$

$\ln N(\mu, \sigma^2) = (-(x-\mu)^2/2\sigma^2) - \ln(2\pi\sigma^2)/2$

$d \ln N(\mu, \sigma^2)/d\mu = (x-\mu)/\sigma^2$

$d \ln N(\mu, \sigma^2)/d\sigma^2 = ((x-\mu)^2/2(\sigma^2)^2) - 1/2\sigma^2$

See also the apop_multivariate_normal.

Name `Normal distribution`

Input format A scalar, in the `vector` or `matrix` elements of the input apop_data set.

Parameter format Parameter zero (in the vector) is the mean, parmeter one is the standard deviation (i.e., the square root of the variance). After estimation, a page is added named <Covariance> with the 2 $\times$ 2 covariance matrix for these two parameters.

Post-estimate data Unchanged.

postestimate_info Reports the log likelihood.

Predict `apop_predict(NULL, estimated_normal_model)` returns the expected value. The `->more` element holds an apop_data set with the title `<Covariance>`, whose matrix holds the covariance of the mean.

RNG A wrapper for the GSL's Normal RNG.

Settings None.

8.1.1.20 apop_ols

Ordinary least squares. Weighted least squares is also handled by this model.

Name `Ordinary Least Squares`

Input format See the notes on the prep routine.

If you provide weights in `your_input_data->weights`, then I will use them appropriately. That is, the apop_ols model really implements Weighted Least Squares, but in most cases `weights==NULL` and the math reduces to the special case of Ordinary Least Squares.

Prep_routine If your input data has no `vector` element, then column zero of the matrix is taken to be the dependent variable. This routine moves the dependent variable to the `vector`, and replaces column zero with a column of all ones, indicating a constant term. This is the norm for OLS, and is probably what you want. The easiest way to generate data for this sort of process is via a query like `apop_query_to_matrix("select depvar, independent_var1, independent_var2 from dataset")`.

If your data has a `vector` element, then the prep routines won't try to force something to be there. That is, nothing will be moved, and no constant column generated. If you don't want to use a constant column, or your data has already been prepped by an estimation, then this is what you want. See apop_query_to_mixed_data for an easy way to generate a data set like this via queries.

Parameter format A vector of OLS coefficients. Coefficient zero refers to the constant column, if any. The `vector` of the output will therefore be of size `data->size2`.

The estimation routine appends a page named `<Covariance>`, giving the covariance matrix for the estimated parameters (not the data itself).

Post-estimate data You can specify whether the data is modified with an apop_lm_settings group. Else, left unchanged.

postestimate_parameter_model For the mean, a noncentral t distribution (apop_t_distribution).

postestimate_info Reports log likelihood, and runs apop_estimate_coefficient_of_determination to add R^2-type information (SSE, SSR, &c) to the info page.

Residuals: I add a page named `<Predicted>`, with three columns. The first column is the dependent variable from the input data. Let our model be $Y = \beta X + \epsilon$. Then the second column is the predicted values: βX, and the third column is the residuals: ϵ. The third column is therefore always the first minus the second.

Given your estimate `est`, the zeroth element is one of

```
apop_data_get(est->info, .page= "Predicted", .row=0, .colname="observed"),
apop_data_get(est->info, .page= "Predicted", .row=0, .colname="predicted")
or
apop_data_get(est->info, .page= "Predicted", .row=0, .colname="residual").
```

RNG Linear models are typically only partially defined probability models. For OLS, we know that $P(Y|X\beta) \sim \mathcal{N}(X\beta, \sigma)$, because this is an assumption about the error process, but we don't know much of anything about the distribution of X.

The apop_lm_settings group includes an apop_model element named `input_distribution`. This is the distribution of the independent/predictor/X columns of the data set.

The default is that `input_distribution = apop_improper_uniform` , meaning that $P(X) = 1$ for all X. So $P(Y, X) = P(Y|X)P(X) = P(Y|X)$. This seems to be how many people

use linear models: the X values are taken as certain (as with actually observed data) and the only question is the odds of the dependent variable. If that's what you're looking for, just leave the default. This is sufficient for getting log likelihoods under the typical assumption that the observed data has probability one.

But you can't draw from an improper uniform. So if you draw from a linear model with a default `input_distribution`, then you'll get an error.
Alternatively, you may know something about the distribution of the input data. For example, the data model may simply be a PMF from the actual data:

```
1 apop_settings_set(your_model, apop_lm, input_distribution, apop_estimate(inset, apop_pmf));
```

Now, random draws are taken from the input data, and the dependent variable value calculated via $X\beta + \epsilon$, where X is the drawn value, β the previously-estimated parameters and ϵ is a Normally-distributed random draw. Or change the PMF to any other appropriate distribution, such as a apop_multivariate_normal, or an apop_pmf filled in with more data, or perhaps something from http://en.wikipedia.org/wiki/Errors-in-variables_models , as desired.

Examples A quick overview opens with a sample program using OLS. For quick reference, here is the program, but see that page for a full discussion.

```
#include <apop.h>

int main(){
    apop_text_to_db(.text_file="data", .tabname="d");
    apop_data *data = apop_query_to_data("select * from d");
    apop_model *est = apop_estimate(data, apop_ols);
    apop_model_print(est);
}
```

8.1.1.21 apop_pmf

A probability mass function is commonly known as a histogram, or still more commonly, a bar chart. It indicates that at a given coordinate, there is a given mass.

Each row of the PMF's data set holds the coordinates, and the *weights vector* holds the mass at the given point. This is in contrast to the crosstab format, where the location is simply given by the position of the data point in the grid.

For example, here is a typical crosstab:

	col 0	col 1	col 2
row 0	0	8.1	3.2
row 1	0	0	2.2
row 2	0	7.3	1.2

Here it is as a sparse listing:

dimension 1	dimension 2	value
0	1	8.1
0	2	3.2
1	2	2.2
2	1	7.3
2	2	1.2

The `apop_pmf` internally represents data in this manner, with the dimensions in the `matrix`, `vector`, and `text` element of the data set, and the cell values are held in the `weights` element (not the vector).

If your data is in a crosstab (with observation coordinates in the matrix element for 2-D data or the vector for 1-D data), then use apop_crosstab_to_db to make the conversion. See also the wiki for another crosstab-to-PMF function.

If your data is already in the sparse listing format (which is probably the case for 3- or more dimensional data), then estimate the model via:

```
1 apop_model *my_pmf = apop_estimate(in_data, apop_pmf);
```

- If the `weights` element is `NULL`, then I assume that all rows of the data set are equally probable.
- If the `weights` are present but sum to a not-finite value, the model's `error` element is set to `'w'` when the estimation is run, and a warning printed.

Name `PDF or sparse matrix`

Input format One observation per row, with coordinates in the `vector`, `matrix`, and/or `text`, and the density at that point in the `weights`. If `weights==NULL`, all observations are equiprobable.

Parameter format None. The list of observations and their weights are in the `data` set, not the `parameters`.

Post-estimate data The data you sent in is linked to (not copied).

Post-estimate parameters Still `NULL`.

RNG Return the data in a random row of the PMF's data set. If there is a weights vector, I will use that to make draws; else all rows are equiprobable.

- If you set `draw_index` to `'y'`, e.g.,

```
1 Apop_settings_add(your_model, apop_pmf, draw_index, 'y');
```

then I will return the row number of the draw, not the data in that row. Because apop_draw only returns numeric data, this is the only meaningful way to make draws from text data.

- The first time you draw from a PMF with uneven weights, I will generate a vector tallying the cumulative mass. Subsequent draws will have no computational overhead. Because the vector is built using the data on the first call to this or the `cdf` method, do not rearrange or modify the data after the first call. I.e., if you choose to use apop_data_sort or apop_data_pmf_compress on your data, do it before the first draw or CDF calculation.

m->error='f'	There is zero or NaN density in the CMF. I set the model's `error` element to `'f'` and set `out=NAN`.
m->error='a'	Allocation error. I set the model's `error` element to `'a'` and set `out=NAN`. Maybe try apop_data_pmf_compress first?

CDF *Assuming the data is sorted in a meaningful manner*, find the total mass up to a given data point.

That is, a CDF only makes sense if the data space is totally ordered. The sorting you define using apop_data_sort defines that ordering.

- The input data should have the same number of columns as the data set used to construct the PMF. I use only the first row.
- If the observation is not found in the data, return zero.
- The first time you get a CDF from from a data set with uneven weights, I will generate a vector tallying the cumulative mass. Subsequent draws will have no computational overhead. Because the vector is built using the data on the first call to this or the `cdf` method, do not rearrange or modify the data after the first call. I.e., if you choose to use apop_data_sort or apop_data_sort on your data, do it before the first draw or CDF calculation.

Settings apop_pmf_settings

8.1.1.22 apop_poisson

$p(k) = \frac{\mu^k}{k!}\exp(-\mu)$.

Name `Poisson distribution`

Input format One scalar observation per row (in the `matrix` or `vector`).

Parameter format One parameter, the zeroth element of the vector (`double mu = apop_data_get(estimated_model->parameters)`).

Post-estimate data Unchanged.

Post-estimate parameters Unless you decline it by adding the apop_parts_wanted_settings group, I will also give you the variance of the parameter, via bootstrap, stored in a page named <Covariance>.

postestimate_info Reports `log likelihood`.

RNG A wrapper for `gsl_ran_poisson`. Sets a single scalar.

8.1.1.23 apop_probit

Apophenia makes no distinction between the Bivariate Probit and the Multinomial Probit. This one does both.

Name `Probit`

Input format The first column of the data matrix this model expects is zeros, ones, ..., enumerating the factors; see the prep routine. The remaining columns are values of the independent variables. Thus, the model will return [(data columns)-1] ×[(option count)-1] parameters. Column names are options; row names are input variables.

Prep_routine The initial column of data should be a set of factors, set up via apop_data_to_factors. If I find a factor page, I will use that info; if not, then I will run apop_data_to_factors on the left-most column (the vector if there is one, else the first column of the matrix.)

Also, if there is no vector, then I will move the first column of the matrix, and replace that matrix column with a constant column of ones, just like with OLS.

Parameter format As above

Post-estimate data Unchanged.

RNG See apop_ols; this one is similar but produces a category number instead of OLS's continuous draw.

8.1.1.24 apop_t_distribution

The t distribution, primarily for descriptive purposes.

If you want to test a hypothesis, you probably don't need this, and should instead use apop_test.

In that world, the t distribution is parameter free. The data are assumed to be normalized to be based on a mean zero, variance one process, you get the degrees of freedom from the size of the data, and the distribution is thus fixed.

For modeling purposes, more could be done. For example, the t-distribution is a favorite proxy for Normal-like situations where there are fat tails relative to the Normal (i.e., high kurtosis). Or, you may just prefer not to take the step of normalizing your data—one could easily rewrite the theorems underlying the t-distribution without the normalizations.

In such a case, the researcher would not want to fix the df, because df indicates the fatness of the tails, which has some optimal value given the data. Thus, there are two modes of use for these distributions:

- Parameterized, testing style: the degrees of freedom are determined from the data, and all necessary normalizations are assumed. Thus, this code—

```
1 apop_data *t_for_testing = apop_estimate(data, apop_t)
```

—will return exactly the type of t-distribution one would use for testing.

- By removing the `estimate` method—

```
apop_model *spare_t = apop_model_copy(apop_t);
spare_t->estimate = NULL;
apop_model *best_fitting_t = apop_estimate(your_data, spare_t);
```

—I will find the best *df* via maximum likelihood, which may be desirable for to find the best-fitting model for descriptive purposes.

Name `t distribution`

Input format Unordered list of scalars in the matrix and/or vector.

Parameter format Three scalars in the `vector` element:

```
double mu=apop_data_get(estimated_model->parameters, 0)
double sigma=apop_data_get(estimated_model->parameters, 1)
double df=apop_data_get(estimated_model->parameters, 2)
```

Post-estimate data Unchanged.

8.1.1.25 apop_uniform

This is the two-parameter version of the Uniform, expressing a uniform distribution over [a, b].

The MLE of this distribution is simply a = min(your data); b = max(your data). Often useful for the RNG, such as when you have a Uniform prior model.

Name `Uniform distribution`

Input format One scalar observation per row (in the `matrix` or `vector`).

Parameter format Zeroth vector element is a, the min; element one is b, the max.

Post-estimate data Unchanged.

postestimate_info Reports `log likelihood`.

8.1.1.26 apop_yule

$$Y(x,b) = (b-1)\gamma(b)\gamma(k)/\gamma(k+b)$$

$$\ln Y(x,b) = \ln(b-1) + ln\gamma(b) + \ln\gamma(k) - \ln\gamma(k+b)$$

$$d\ln Y/db = 1/(b-1) + \psi(b) - \psi(k+b)$$

Name `Yule distribution`

Input format One scalar observation per row (in the `matrix` or `vector`). See also apop_data_rank_compress for means of dealing with one more input data format.

Parameter format One element in the parameter set's vector.

Post-estimate data Unchanged.

RNG From `Devroye (1986)`, p 553.

Settings MLE-type: apop_mle_settings, apop_parts_wanted_settings

8.1.1.27 apop_zipf

Wikipedia has notes on the `Zipf distribution`.

$$Z(a) = \frac{1}{\zeta(a) * i^a}$$

$$lnZ(a) = -(\log(\zeta(a)) + a\log(i))$$

Name `Zipf distribution`

Input format One scalar observation per row (in the `matrix` or `vector`). See also apop_data_rank_compress for means of dealing with one more input data format.

See also apop_data_rank_compress for means of dealing with one more input data format.

Parameter format One item in the parameter set's vector.

Post-estimate data Unchanged.

RNG Returns an ordinal ranking, starting from 1.
From Devroye (1986), Chapter 10, p 551.

Settings apop_mle_settings

8.2 Public functions, structs, and types

Data Structures

- struct apop_arms_settings
- struct apop_cdf_settings
- struct apop_composition_settings
- struct apop_coordinate_transform_settings
- struct apop_cross_settings
- struct apop_data
- struct apop_dconstrain_settings
- struct apop_kernel_density_settings
- struct apop_lm_settings
- struct apop_loess_settings
- struct apop_mcmc_proposal_s
- struct apop_mcmc_settings
- struct apop_mixture_settings
- struct apop_mle_settings
- struct apop_model
- struct apop_name
- struct apop_opts_type
- struct apop_parts_wanted_settings
- struct apop_pm_settings
- struct apop_pmf_settings
- struct apop_settings_type

Macros

- #define **apop_ANOVA**
- #define Apop_c(d, col)
- #define Apop_col_t(d, colname, outd)
- #define Apop_col_tv(m, col, v)
- #define Apop_cs(d, colnum, len)
- #define Apop_cv(data_to_view, col)
- #define apop_data_add_names(dataset, type, ...)
- #define apop_data_falloc(sizes, ...)
- #define apop_data_fill(adfin, ...)
- #define apop_data_free(freeme)
- #define apop_data_prune_columns(in, ...)
- #define **apop_errorlevel**
- #define apop_estimate_r_squared(in)
- #define **apop_F_distribution**
- #define **apop_F_test**
- #define apop_gaussian
- #define **apop_IV**
- #define **apop_line_to_vector**
- #define Apop_mcv(matrix_to_view, col)
- #define apop_mean
- #define **apop_ml_imputation**(d, m)
- #define **apop_model_coordinate_transform**(...)
- #define apop_model_copy_set(model, type, ...)

- #define apop_model_cross(...)
- #define **apop_model_dcompose**(...)
- #define apop_model_dconstrain(...)
- #define apop_model_mixture(...)
- #define apop_model_set_parameters(in, ...)
- #define Apop_model_set_settings(model, ...)
- #define **apop_model_set_settings**
- #define Apop_mrv(matrix_to_view, row)
- #define Apop_notify(verbosity, ...)
- #define **apop_OLS**
- #define **apop_PMF**
- #define Apop_r(d, rownum)
- #define apop_rng_get_thread(thread_in)
- #define Apop_row_t(d, rowname, outd)
- #define Apop_row_tv(m, row, v)
- #define Apop_rs(d, rownum, len)
- #define Apop_rv(data_to_view, row)
- #define Apop_settings_add_group(model, type, ...)
- #define Apop_settings_copy(name, ...)
- #define Apop_settings_declarations(ysg)
- #define Apop_settings_free(name, ...)
- #define Apop_settings_get(model, type, setting)
- #define Apop_settings_get_group(m, type)
- #define Apop_settings_init(name, ...)
- #define Apop_settings_rm_group(m, type)
- #define Apop_settings_set(model, type, setting, data)
- #define Apop_stopif(test, onfail, level, ...)
- #define Apop_subm(matrix_to_view, srow, scol, nrows, ncols)
- #define apop_sum
- #define **apop_test_ANOVA_independence**(d)
- #define **apop_text_add**
- #define apop_text_fill(dataset, ...)
- #define apop_var
- #define **Apop_varad_set**(var, value)
- #define apop_vector_fill(avfin, ...)

Functions

- apop_data * apop_anova (char *table, char *data, char *grouping1, char *grouping2)
- int apop_arms_draw (double *out, gsl_rng *r, apop_model *m)
- gsl_vector * apop_array_to_vector (double *in, int size)
- apop_model * apop_beta_from_mean_var (double m, double v)
- apop_data * apop_bootstrap_cov (apop_data *data, apop_model *model, gsl_rng *rng, int iterations, char keep_boots, char ignore_nans, apop_data **boot_store)
- double apop_cdf (apop_data *d, apop_model *m)
- void apop_crosstab_to_db (apop_data *in, char *tabname, char *row_col_name, char *col_col_↩ name, char *data_col_name)
- void apop_data_add_named_elmt (apop_data *d, char *name, double val)
- void **apop_data_add_names_base** (apop_data *d, const char type, char const **names)
- apop_data * apop_data_add_page (apop_data *dataset, apop_data *newpage, const char *title)
- apop_data * apop_data_alloc (const size_t size1, const size_t size2, const int size3)

- apop_data * apop_data_calloc (const size_t size1, const size_t size2, const int size3)
- apop_data * apop_data_copy (const apop_data *in)
- apop_data * apop_data_correlation (const apop_data *in)
- apop_data * apop_data_covariance (const apop_data *in)
- apop_data * **apop_data_fill_base** (apop_data *in, double[])
- char apop_data_free_base (apop_data *freeme)
- double apop_data_get (const apop_data *data, size_t row, int col, const char *rowname, const char *colname, const char *page)
- apop_data * apop_data_get_factor_names (apop_data *data, int col, char type)
- apop_data * apop_data_get_page (const apop_data *data, const char *title, const char match)
- apop_data * apop_data_listwise_delete (apop_data *d, char inplace)
- void apop_data_memcpy (apop_data *out, const apop_data *in)
- gsl_vector * apop_data_pack (const apop_data *in, gsl_vector *out, char more_pages, char use_↩ info_pages)
- apop_data * apop_data_pmf_compress (apop_data *in)
- void apop_data_print (const apop_data *data, Output_declares)
- void **apop_data_print** (const apop_data *data, char const *output_name, FILE *output_pipe, char output_type, char output_append)
- apop_data * apop_data_prune_columns_base (apop_data *d, char **colnames)
- double * apop_data_ptr (apop_data *data, int row, int col, const char *rowname, const char *colname, const char *page)
- apop_data * apop_data_rank_compress (apop_data *in, int min_bins)
- apop_data * apop_data_rank_expand (apop_data *in)
- void apop_data_rm_columns (apop_data *d, int *drop)
- apop_data * apop_data_rm_page (apop_data *data, const char *title, const char free_p)
- apop_data * apop_data_rm_rows (apop_data *in, int *drop, int(*do_drop)(apop_data *, void *), void *drop_parameter)
- int apop_data_set (apop_data *data, size_t row, int col, const double val, const char *rowname, const char *colname, const char *page)
- void **apop_data_show** (const apop_data *data)
- apop_data * apop_data_sort (apop_data *data, apop_data *sort_order, char asc, char inplace, double *col_order)
- apop_data ** apop_data_split (apop_data *in, int splitpoint, char r_or_c)
- apop_data * apop_data_stack (apop_data *m1, apop_data *m2, char posn, char inplace)
- apop_data * apop_data_summarize (apop_data *data)
- apop_data * apop_data_to_bins (apop_data const *indata, apop_data const *binspec, int bin_↩ count, char close_top_bin)
- int **apop_data_to_db** (const apop_data *set, const char *tabname, char)
- apop_data * apop_data_to_dummies (apop_data *d, int col, char type, int keep_first, char append, char remove)
- apop_data * apop_data_to_factors (apop_data *data, char intype, int incol, int outcol)
- apop_data * apop_data_transpose (apop_data *in, char transpose_text, char inplace)
- void apop_data_unpack (const gsl_vector *in, apop_data *d, char use_info_pages)
- int apop_db_close (char vacuum)
- int apop_db_open (char const *filename)
- apop_data * apop_db_to_crosstab (char const *tabname, char const *row, char const *col, char const *data, char is_aggregate)
- double apop_det_and_inv (const gsl_matrix *in, gsl_matrix **out, int calc_det, int calc_inv)
- apop_data * apop_dot (const apop_data *d1, const apop_data *d2, char form1, char form2)
- int apop_draw (double *out, gsl_rng *r, apop_model *m)
- apop_model * apop_estimate (apop_data *d, apop_model *m)

- apop_data * apop_estimate_coefficient_of_determination (apop_model *)
- void **apop_estimate_parameter_tests** (apop_model *est)
- apop_model * apop_estimate_restart (apop_model *e, apop_model *copy, char *starting_pt, double boundary)
- apop_data * apop_f_test (apop_model *est, apop_data *contrast)
- long double apop_generalized_harmonic (int N, double s)
- apop_data * apop_histograms_test_goodness_of_fit (apop_model *h0, apop_model *h1)
- apop_data * apop_jackknife_cov (apop_data *data, apop_model *model)
- long double apop_kl_divergence (apop_model *from, apop_model *to, int draw_ct, gsl_rng *rng)
- long double apop_linear_constraint (gsl_vector *beta, apop_data *constraint, double margin)
- double apop_log_likelihood (apop_data *d, apop_model *m)
- apop_data * apop_map (apop_data *in, apop_fn_d *fn_d, apop_fn_v *fn_v, apop_fn_r *fn_r, apop_fn_dp *fn_dp, apop_fn_vp *fn_vp, apop_fn_rp *fn_rp, apop_fn_dpi *fn_dpi, apop_fn_vpi *fn_vpi, apop_fn_rpi *fn_rpi, apop_fn_di *fn_di, apop_fn_vi *fn_vi, apop_fn_ri *fn_ri, void *param, int inplace, char part, int all_pages)
- apop_data * **apop_map** (apop_data *in, double(*fn_d)(double), double(*fn_v)(gsl_vector *), double(*fn_r)(apop_data *), double(*fn_dp)(double, void *), double(*fn_vp)(gsl_vector *, void *), double(*fn_rp)(apop_data *, void *), double(*fn_dpi)(double, void *, int), double(*fn_vpi)(gsl_vector *, void *, int), double(*fn_rpi)(apop_data *, void *, int), double(*fn_di)(double, int), double(*fn_vi)(gsl_vector *, int), double(*fn_ri)(apop_data *, int), void *param, int inplace, char part, int all_pages)
- double apop_map_sum (apop_data *in, apop_fn_d *fn_d, apop_fn_v *fn_v, apop_fn_r *fn_r, apop_fn_dp *fn_dp, apop_fn_vp *fn_vp, apop_fn_rp *fn_rp, apop_fn_dpi *fn_dpi, apop_fn_vpi *fn_vpi, apop_fn_rpi *fn_rpi, apop_fn_di *fn_di, apop_fn_vi *fn_vi, apop_fn_ri *fn_ri, void *param, char part, int all_pages)
- double **apop_map_sum** (apop_data *in, double(*fn_d)(double), double(*fn_v)(gsl_vector *), double(*fn_r)(apop_data *), double(*fn_dp)(double, void *), double(*fn_vp)(gsl_vector *, void *), double(*fn_rp)(apop_data *, void *), double(*fn_dpi)(double, void *, int), double(*fn_vpi)(gsl_vector *, void *, int), double(*fn_rpi)(apop_data *, void *, int), double(*fn_di)(double, int), double(*fn_vi)(gsl_vector *, int), double(*fn_ri)(apop_data *, int), void *param, char part, int all_pages)
- void apop_matrix_apply (gsl_matrix *m, void(*fn)(gsl_vector *))
- void apop_matrix_apply_all (gsl_matrix *in, void(*fn)(double *))
- gsl_matrix * apop_matrix_copy (const gsl_matrix *in)
- double apop_matrix_determinant (const gsl_matrix *in)
- gsl_matrix * apop_matrix_inverse (const gsl_matrix *in)
- int apop_matrix_is_positive_semidefinite (gsl_matrix *m, char semi)
- gsl_vector * apop_matrix_map (const gsl_matrix *m, double(*fn)(gsl_vector *))
- gsl_matrix * apop_matrix_map_all (const gsl_matrix *in, double(*fn)(double))
- double apop_matrix_map_all_sum (const gsl_matrix *in, double(*fn)(double))
- double apop_matrix_map_sum (const gsl_matrix *in, double(*fn)(gsl_vector *))
- double apop_matrix_mean (const gsl_matrix *data)
- void apop_matrix_mean_and_var (const gsl_matrix *data, double *mean, double *var)
- apop_data * apop_matrix_pca (gsl_matrix *data, int const dimensions_we_want)
- void apop_matrix_print (const gsl_matrix *data, Output_declares)
- void **apop_matrix_print** (const gsl_matrix *data, char const *output_name, FILE *output_pipe, char output_type, char output_append)
- gsl_matrix * apop_matrix_realloc (gsl_matrix *m, size_t newheight, size_t newwidth)
- void **apop_matrix_show** (const gsl_matrix *data)
- gsl_matrix * apop_matrix_stack (gsl_matrix *m1, gsl_matrix const *m2, char posn, char inplace)
- long double apop_matrix_sum (const gsl_matrix *m)
- double apop_matrix_to_positive_semidefinite (gsl_matrix *m)

- void apop_maximum_likelihood (apop_data *data, apop_model *dist)
- apop_model * apop_ml_impute (apop_data *d, apop_model *meanvar)
- apop_model * apop_model_clear (apop_data *data, apop_model *model)
- apop_model * apop_model_copy (apop_model *in)
- apop_model * **apop_model_cross_base** (apop_model *mlist[])
- apop_data * apop_model_draws (apop_model *model, int count, apop_data *draws)
- long double apop_model_entropy (apop_model *in, int draws)
- apop_model * apop_model_fix_params (apop_model *model_in)
- apop_model * apop_model_fix_params_get_base (apop_model *model_in)
- void apop_model_free (apop_model *free_me)
- apop_data * apop_model_hessian (apop_data *data, apop_model *model, double delta)
- apop_model * apop_model_metropolis (apop_data *d, gsl_rng *rng, apop_model *m)
- int apop_model_metropolis_draw (double *out, gsl_rng *rng, apop_model *model)
- apop_model * **apop_model_mixture_base** (apop_model **inlist)
- apop_data * apop_model_numerical_covariance (apop_data *data, apop_model *model, double delta)
- void apop_model_print (apop_model *model, FILE *output_pipe)
- apop_model * **apop_model_set_parameters_base** (apop_model *in, double ap[])
- void **apop_model_show** (apop_model *print_me)
- apop_model * apop_model_to_pmf (apop_model *model, apop_data *binspec, long int draws, int bin_count)
- long double apop_multivariate_gamma (double a, int p)
- long double apop_multivariate_lngamma (double a, int p)
- int apop_name_add (apop_name *n, char const *add_me, char type)
- apop_name * apop_name_alloc (void)
- apop_name * apop_name_copy (apop_name *in)
- int apop_name_find (const apop_name *n, const char *findme, const char type)
- void apop_name_free (apop_name *free_me)
- void apop_name_print (apop_name *n)
- void apop_name_stack (apop_name *n1, apop_name *nadd, char type1, char typeadd)
- gsl_vector * apop_numerical_gradient (apop_data *data, apop_model *model, double delta)
- double apop_p (apop_data *d, apop_model *m)
- apop_data * apop_paired_t_test (gsl_vector *a, gsl_vector *b)
- apop_model * apop_parameter_model (apop_data *d, apop_model *m)
- apop_data * apop_predict (apop_data *d, apop_model *m)
- void apop_prep (apop_data *d, apop_model *m)
- int apop_prep_output (char const *output_name, FILE **output_pipe, char *output_type, char *output_append)
- int apop_query (const char *q,...)
- apop_data * apop_query_to_data (const char *fmt,...)
- double apop_query_to_float (const char *fmt,...)
- apop_data * apop_query_to_mixed_data (const char *typelist, const char *fmt,...)
- apop_data * apop_query_to_text (const char *fmt,...)
- gsl_vector * apop_query_to_vector (const char *fmt,...)
- apop_data * apop_rake (char const *margin_table, char *const *var_list, int var_ct, char *const *contrasts, int contrast_ct, char const *structural_zeros, int max_iterations, double tolerance, char const *count_col, char const *init_table, char const *init_count_col, double nudge)
- int apop_regex (const char *string, const char *regex, apop_data **substrings, const char use_case)
- gsl_rng * apop_rng_alloc (int seed)
- gsl_rng * **apop_rng_get_thread_base** (int thread)
- double apop_rng_GHgB3 (gsl_rng *r, double *a)

- void apop_score (apop_data *d, gsl_vector *out, apop_model *m)
- int apop_system (const char *fmt,...)
- apop_data * apop_t_test (gsl_vector *a, gsl_vector *b)
- int apop_table_exists (char const *name, char remove)
- double apop_test (double statistic, char *distribution, double p1, double p2, char tail)
- apop_data * apop_test_anova_independence (apop_data *d)
- apop_data * apop_test_fisher_exact (apop_data *intab)
- apop_data * apop_test_kolmogorov (apop_model *m1, apop_model *m2)
- apop_data * apop_text_alloc (apop_data *in, const size_t row, const size_t col)
- apop_data * **apop_text_fill_base** (apop_data *data, char *text[])
- void apop_text_free (char ***freeme, int rows, int cols)
- char * apop_text_paste (apop_data const *strings, char *between, char *before, char *after, char *between_cols, int(*prune)(apop_data *, int, int, void *), void *prune_parameter)
- int apop_text_set (apop_data *in, const size_t row, const size_t col, const char *fmt,...)
- apop_data * apop_text_to_data (char const *text_file, int has_row_names, int has_col_names, int const *field_ends, char const *delimiters)
- int apop_text_to_db (char const *text_file, char *tabname, int has_row_names, int has_col_names, char **field_names, int const *field_ends, apop_data *field_params, char *table_params, char const *delimiters, char if_table_exists)
- apop_data * apop_text_unique_elements (const apop_data *d, size_t col)
- apop_model * apop_update (apop_data *data, apop_model *prior, apop_model *likelihood, gsl_rng *rng)
- void apop_vector_apply (gsl_vector *v, void(*fn)(double *))
- int apop_vector_bounded (const gsl_vector *in, long double max)
- gsl_vector * apop_vector_copy (const gsl_vector *in)
- double apop_vector_correlation (const gsl_vector *ina, const gsl_vector *inb, const gsl_vector *weights)
- double apop_vector_cov (gsl_vector const *v1, gsl_vector const *v2, gsl_vector const *weights)
- double apop_vector_distance (const gsl_vector *ina, const gsl_vector *inb, const char metric, const double norm)
- long double apop_vector_entropy (gsl_vector *in)
- void apop_vector_exp (gsl_vector *v)
- gsl_vector * **apop_vector_fill_base** (gsl_vector *in, double[])
- double apop_vector_kurtosis (const gsl_vector *in)
- double apop_vector_kurtosis_pop (gsl_vector const *v, gsl_vector const *weights)
- void apop_vector_log (gsl_vector *v)
- void apop_vector_log10 (gsl_vector *v)
- gsl_vector * apop_vector_map (const gsl_vector *v, double(*fn)(double))
- double apop_vector_map_sum (const gsl_vector *in, double(*fn)(double))
- double apop_vector_mean (gsl_vector const *v, gsl_vector const *weights)
- gsl_vector * apop_vector_moving_average (gsl_vector *, size_t)
- void apop_vector_normalize (gsl_vector *in, gsl_vector **out, const char normalization_type)
- double * apop_vector_percentiles (gsl_vector *data, char rounding)
- void apop_vector_print (gsl_vector *data, Output_declares)
- void **apop_vector_print** (gsl_vector *data, char const *output_name, FILE *output_pipe, char output_type, char output_append)
- gsl_vector * apop_vector_realloc (gsl_vector *v, size_t newheight)
- void **apop_vector_show** (const gsl_vector *data)
- double apop_vector_skew (const gsl_vector *in)
- double apop_vector_skew_pop (gsl_vector const *v, gsl_vector const *weights)
- gsl_vector * apop_vector_stack (gsl_vector *v1, gsl_vector const *v2, char inplace)

- long double apop_vector_sum (const gsl_vector *in)
- gsl_matrix * apop_vector_to_matrix (const gsl_vector *in, char row_col)
- gsl_vector * apop_vector_unique_elements (const gsl_vector *v)
- double apop_vector_var (gsl_vector const *v, gsl_vector const *weights)
- double apop_vector_var_m (const gsl_vector *in, const double mean)

Variables

- apop_model * **apop_bernoulli**
- apop_model * **apop_beta**
- apop_model * **apop_binomial**
- apop_model * **apop_chi_squared**
- apop_model * **apop_composition**
- apop_model * **apop_coordinate_transform**
- apop_model * **apop_cross**
- apop_model * **apop_dconstrain**
- apop_model * **apop_dirichlet**
- apop_model * **apop_exponential**
- apop_model * **apop_f_distribution**
- apop_model * **apop_gamma**
- apop_model * **apop_improper_uniform**
- apop_model * **apop_iv**
- apop_model * **apop_kernel_density**
- apop_model * **apop_loess**
- apop_model * **apop_logit**
- apop_model * **apop_lognormal**
- apop_model * **apop_mixture**
- apop_model * **apop_multinomial**
- apop_model * **apop_multivariate_normal**
- apop_model * **apop_normal**
- apop_model * **apop_ols**
- apop_opts_type apop_opts
- apop_opts_type **apop_opts**
- apop_model * **apop_pmf**
- apop_model * **apop_poisson**
- apop_model * **apop_probit**
- apop_model * **apop_t_distribution**
- apop_model * **apop_uniform**
- apop_model * **apop_wls**
- apop_model * **apop_yule**
- apop_model * **apop_zipf**

Detailed Description

8.2.1 Macro Definition Documentation

8.2.1.1 #define Apop_c(d, col)

A macro to generate a temporary one-column view of apop_data set `d`, pulling out only column `col`. After this call, `outd` will be a pointer to this temporary view, that you can use as you would any apop_data set.

See also

Apop_cs, Apop_cv, Apop_col_tv, Apop_col_t, Apop_mcv

8.2.1.2 #define Apop_col_t(d, colname, outd)

After this call, `v` will hold a view of the apop_data set `m`. The view will consist only of a `gsl_vector` view of the column of the apop_data set `m` with name `col_name`. Unlike Apop_c, the second argument is a column name, that I'll look up using apop_name_find, and the third is the name of the view to be generated.

See also

Apop_cs, Apop_c, Apop_cv, Apop_col_tv, Apop_mcv

8.2.1.3 #define Apop_col_tv(m, col, v)

After this call, `v` will hold a `gsl_vector` view of the apop_data set `m`. The view will consist only of the column with name `col_name`. Unlike Apop_cv, the second argument is a column name, that I'll look up using apop_name_find, and the third is the name of the view to be generated.

See also

Apop_cs, Apop_c, Apop_cv, Apop_col_t, Apop_mcv

8.2.1.4 #define Apop_cs(d, colnum, len)

A macro to generate a temporary view of apop_data set `d` including only certain columns, beginning at column `col` and having length `len`.

The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See also

Apop_c, Apop_cv, Apop_col_tv, Apop_col_t, Apop_mcv

8.2.1.5 #define Apop_cv(data_to_view, col)

A macro to generate a temporary one-column view of the matrix in an apop_data set `d`, pulling out only column `col`. The view is a `gsl_vector` set.

As usual, column -1 is the vector element of the apop_data set.

```
1 gsl_vector *v = Apop_cv(your_data, i);
2
3 for (int i=0; i< your_data->matrix->size2; i++)
4     printf("_%i = %g\n", i, apop_vector_sum(Apop_c(your_data, i)));
```

The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See also

Apop_cs, Apop_c, Apop_col_tv, Apop_col_t, Apop_mcv

8.2.1.6 #define apop_data_add_names(dataset, type, ...)

Add a list of names to a data set.

- Use this with a list of names that you type in yourself, like

```
1 apop_data_add_names(mydata, 'c', "age", "sex", "height");
```

  Notice the lack of curly braces around the list.

- You may have an array of names, probably autogenerated, that you would like to add. In this case, make certain that the last element of the array is `NULL`, and call the base function:

```
1 char **[] colnames = {"age", "sex", "height", NULL};
2 apop_data_add_names_base(mydata, 'c', colnames);
```

  But if you forget the `NULL` marker, this has good odds of segfaulting. You may prefer to use a `for` loop that inserts each name in turn using apop_name_add.

See also

apop_name_add, although apop_data_add_names will be more useful in most cases.

8.2.1.7 #define apop_data_falloc(sizes, ...)

Allocate a data set and fill it with values. Put the data set dimensions (one, two, or three dimensions as per apop_data_alloc) in parens, then the data (as per apop_data_fill). E.g.:

```
1 apop_data *identity2 = apop_data_falloc((2,2),
2                          1, 0,
3                          0, 1);
4
5 apop_data *count_vector = apop_data_falloc((5), 0, 1, 2, 3, 4);
```

If you forget the parens, you will get an obscure error during compilation.

- This is a simple macro wrapping apop_data_fill and apop_data_alloc, because they appear together so often. The second example expands to:

```
1 apop_data *count_vector = apop_data_fill(apop_data_alloc(5), 0, 1, 2, 3, 4);
```

8.2.1.8 #define apop_data_fill(adfin, ...)

Fill a pre-allocated data set with values.

adfin	An apop_data set (that you have already allocated).
...	A series of at least as many floating-point values as there are blanks in the data set.

Returns

A pointer to the same data set that was input.

- I need as many arguments as the size of the data set, and can't count them for you. Too many will be ignored; too few will produce unpredictable results, which may include padding your matrix with garbage or a simple segfault.

- Underlying this function is a base function that takes a single list, as opposed to the set of unassociated numbers sent to apop_data_fill. See the example below for a comparison.

- This function assumes that if the apop_data set has both `vector` and `matrix`, then `vector->size==matrix->size1`.

- See also apop_data_falloc to allocate and fill on one line. E.g., to generate a unit vector for three dimensions:

```
1 apop_data *unit_vector = apop_data_falloc((3), 1, 1, 1);
```

An example, using both a loose list of numbers and an array.

```
#include <apop.h>

void with_fixed_numbers(){
    apop_data *a =apop_data_alloc(2,2,2);
    double    eight   = 8.0;
    apop_data_fill(a, 8, 2.2, eight/2,
                      0, 6.0, eight);
    apop_data_show(a);
}

void with_a_list(){
  apop_data *a =apop_data_alloc(2,2,2);
  double    eight   = 8.0;
  double list[] = {8, 2.2, eight/2,
                   0, 6.0, eight};
    apop_data_fill_base(a, list);
    apop_data_show(a);
}

int main(){
    with_fixed_numbers();
    printf("-----\n");
    with_a_list();
}
```

See also

apop_text_fill, apop_data_falloc, apop_data_unpack

8.2.1.9 #define apop_data_free(freeme)

Free an apop_data structure.

- As with `free()`, it is safe to send in a `NULL` pointer (in which case the function does nothing).

- If the `more` pointer is not `NULL`, I will free the pointed-to data set first. If you don't want to free data sets down the chain, set `more=NULL` before calling this.

- This is actually a macro (that calls apop_data_free_base). It sets `freeme` to `NULL` when it's done, because there's nothing safe you can do with the freed location, and you can later safely test conditions like `if (data) ...`.

8.2.1.10 #define apop_data_prune_columns(in, ...)

Keep only the columns of a data set that you name.

in	The data set to prune.
...	A list of names to retain (i.e. the columns that shouldn't be pruned out). For example, if you have run apop_data_summarize, you have columns for several statistics, but may care about only one or two; see the example.

For example:

```
#include <apop.h>

// This sample produces a dummy times table, gets a summary, and prunes the summary table.
int main(){
    int i, j;
    apop_data *d = apop_data_alloc(0, 10, 4);
    for (i=0; i< 10; i++)
        for (j=0; j< 4; j++)
            apop_data_set(d, i, j, i*j);
    apop_data *summary = apop_data_summarize(d);
    apop_data_prune_columns(summary, "mean", "median");
    assert(apop_name_find(summary->names, "mean", 'c')!=-2);
    assert(apop_name_find(summary->names, "median", 'c')!=-2);
    assert(apop_name_find(summary->names, "max", 'c')==-2); //not found
    assert(apop_name_find(summary->names, "variance", 'c')==-2); //not found
    assert(apop_data_get(summary, .row=0, .colname="mean")==0);
    assert(apop_data_get(summary, .row=1, .colname="median")==4);
    assert(apop_data_get(summary, .row=2, .colname="median")==8);
    apop_data_show(summary);
}
```

- I use a case-insensitive search to find your column.
- If your name multiple columns, I'll only give you the first.
- If I can't find a column matching one of your strings, I throw an error to the screen and continue.
- This is a macro calling apop_data_prune_columns_base. It packages your list of columns into a list of strings, adds a `NULL` string at the end, and calls that function.

8.2.1.11 #define apop_estimate_r_squared(in)

A synonym for apop_estimate_coefficient_of_determination, q.v.

8.2.1.12 #define apop_gaussian

Alias for the apop_normal distribution, qv.

8.2.1.13 #define Apop_mcv(matrix_to_view, col)

Get a vector view of a single column of a gsl_matrix.

matrix_to_vew	A gsl_matrix.
row	An integer giving the column to be viewed.

Returns

A `gsl_vector` view of the given column. The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

```
1 gsl_matrix *m = apop_query_to_data("select col1, col2, col3 from data")->matrix;
2 printf("The correlation coefficient between columns two "
3        "and three is %g.\n", apop_vector_correlation(Apop_mcv(m, 2), Apop_mcv(m, 3)));
```

See also

Apop_r, Apop_cv

8.2.1.14 #define apop_mean

Returns the mean of the elements of the vector v.

v	A gsl_vector.

8.2.1.15 #define apop_model_copy_set(model, type, ...)

Copy a model and add a settings group. Useful for models that require a settings group to function. See Apop_settings_add_group.

Returns

A pointer to the newly-prepped model.

8.2.1.16 #define apop_model_cross(...)

Generate a model consisting of the cross product of several independent models. The output apop_model is a copy of apop_cross; see that model's documentation for details.

- If you input only one model, return a copy of that model; print a warning iff `apop_opts.verbose >= 2`.

error=='n'	First model input is `NULL`.

Examples:

```
#include <apop.h>

/* In this initial example, build a cross product of two Normal(2,.1) distributions.
Make 10,000 draws from it.

Then, build a cross product of two unparameterized Normals and estimate the parameters
of the combined model; check that they match the (2, .1) we started with.
*/
void cross_normals(){
    double mu = 2;
    double sigma = .1;
    apop_model *n1 = apop_model_set_parameters(apop_normal, mu, sigma);
    apop_model *n2 = apop_model_copy(n1);
    apop_model *two_independent_normals = apop_model_cross(n1, n2);
    //
    //We don't use it, but the cross product of three is just as easy:
    apop_model *n3 = apop_model_copy(n1);
    apop_model *three_independent_normals = apop_model_cross(n1, n2, n3);

    apop_data *draws = apop_model_draws(two_independent_normals, .count=10000);

    //The unparameterized cross product:
    apop_model *two_n = apop_model_cross(
                    apop_model_copy(apop_normal),
                    apop_model_copy(apop_normal)
                    );
    apop_model *estimated_norms = apop_estimate(draws, two_n);

    apop_model_print(estimated_norms);
    apop_data *estp1 = Apop_settings_get(estimated_norms, apop_cross, model1)->parameters;
```

```
    apop_data *estp2 = Apop_settings_get(estimated_norms, apop_cross, model2)->parameters;
    assert(fabs(apop_data_get(estp1, 0) - mu)    < 1e-3);
    assert(fabs(apop_data_get(estp2, 0) - mu)    < 1e-3);
    assert(fabs(apop_data_get(estp1, 1) - sigma) < 1e-3);
    assert(fabs(apop_data_get(estp2, 1) - sigma) < 1e-3);
}

//bind together a Poisson and a Normal
void norm_cross_poisson(){
    apop_model *m1 = apop_model_set_parameters(apop_poisson, 3);
    apop_model *m2 = apop_model_set_parameters(apop_normal, -5, 1);
    apop_model *mm = apop_model_cross(m1, m2);
    int len = 1e5;
    apop_data *draws = apop_model_draws(mm, len);
    for (int i=0; i< len; i++){
        Apop_row_v(draws, i, onev);
        assert((int)onev->data[0] == onev->data[0]);
        assert(onev->data[1]<0);
    }

    /*The rest of the test script recovers the parameters.
    Input data to an apop_cross model can take two formats. In cross_normals, the
    draws are in a single matrix. Here, the data for the Poisson (col 0 of the draws)
    will be put in an apop_data set, and the data for the Normal (col 1 of the draws)
    on a second page appended to the first. Then, set the .splitpage element of the
    apop_cross settings group to the name of the second page.
    */
    apop_data *comeback = apop_data_alloc();
    comeback->vector = apop_vector_copy(Apop_cv(draws, 0));
    apop_data_add_page(comeback, apop_data_alloc(), "p2");
    comeback->more->vector = apop_vector_copy(Apop_cv(draws, 1));

    //set up the un-parameterized crossed model, including
    //the name at which to split the data set
    apop_model *estme = apop_model_cross(apop_model_copy(apop_poisson), apop_model_copy(apop_normal));
    Apop_settings_add(estme, apop_cross, splitpage, "p2");
    apop_model *ested = apop_estimate(comeback, estme);

    //test that the parameters are as promised.
    apop_model *m1back = apop_settings_get(ested, apop_cross, model1);
    apop_model *m2back = apop_settings_get(ested, apop_cross, model2);
    assert(fabs(apop_data_get(m1back->parameters, .col=-1) - 3) < 5e-1);
    assert(fabs(apop_data_get(m2back->parameters, .col=-1) - -5) < 5e-1);
    assert(fabs(apop_data_get(m2back->parameters, .col=-1, .row=1) - 1) < 5e-1);

    //You can cross as many models as you'd like.
    apop_model *m3 = apop_model_set_parameters(apop_poisson, 8);
    apop_model *mmm = apop_model_cross(m1, m2, m3);
    apop_data *sum = apop_data_summarize(apop_model_draws(mmm, 1e5));
    assert(fabs(apop_data_get(sum, .row=0, .colname="mean") - 3) < 2e-2);
    assert(fabs(apop_data_get(sum, .row=1, .colname="mean") - -5) < 2e-2);
    assert(fabs(apop_data_get(sum, .row=2, .colname="mean") - 8) < 4e-2);
    assert(apop_data_get(sum, .row=0, .colname="median") == 3);
    assert(apop_data_get(sum, .row=2, .colname="median") == 8);
}

int main(){
    cross_normals();
    norm_cross_poisson();
}
```

8.2.1.17 #define apop_model_dconstrain(...)

Build an `apop_dconstrain` model, q.v., which applies a data constraint to the data set. For example, this is how one would truncate a model to have data above zero.

Returns

An apop_model that is a copy of apop_dconstrain and is appropriately set up.

- Uses the apop_dconstrain_settings group. This macro takes elements of that struct as inputs.
- This function uses the Designated initializers syntax for inputs.

8.2.1.18 #define apop_model_mixture(...)

Produce a model as a linear combination of other models. See the documentation for the apop_mixture model.

...	A list of models, either all parameterized or all unparameterized. See examples in the apop_mixture documentation.

8.2.1.19 #define apop_model_set_parameters(in, ...)

Take in an unparameterized apop_model and return a new apop_model with the given parameters. For example, if you need a N(0,1) quickly:

```
1 apop_model *std_normal = apop_model_set_parameters(apop_normal, 0, 1);
```

This doesn't take in data, so it won't work with models that take the number of parameters from the data, and it will only set the vector of the model's parameter apop_data set. This is most standard models. If you have a situation where these options are out, you could

- manually set Set .vsize and/or .msize1 and .msize2 first, then call this function, or
- prep the model via something like `apop_model *new = apop_model_copy(in); apop_prep(your_data, new);` (because apop_prep is required to correctly allocate `new->parameters` to conform to your data).

in	An unparameterized model, like apop_normal or apop_poisson.
...	The list of parameters.

Returns

A copy of the input model, with parameters set.

out->error=='d'	dimension error: you gave me a model with an indeterminate number of parameters. See notes above. Set .vsize or .msize1 and .msize2 first, then call this function, or use `apop_model *new = apop_model_copy(in); apop_prep(your_data, new);` and then call this .

See also

apop_data_fill

8.2.1.20 #define Apop_model_set_settings(model, ...)

This is the complement to apop_model_set_parameters, for those models that are set up by adding settings group, rather than filling in a list of parameters.

For example, the apop_kernel_density model is built by adding a apop_kernel_density_settings group. From the example on the apop_kernel_density page:

```
apop_model *k2 = apop_model_set_settings(apop_kernel_density,
                    .base_data=d,
                    .set_fn = set_uniform_edges,
                    .kernel = apop_uniform);
```

The name of the model and the settings group to be built must match, which is the case for many model transformations, including apop_dconstrain and apop_cross. If the names do not match, use apop_model_copy_set.

8.2.1.21 #define Apop_mrv(matrix_to_view, row)

Get a vector view of a single row of a gsl_matrix.

matrix_to_vew	A gsl_matrix.
row	An integer giving the row to be viewed.

Returns

A `gsl_vector` view of the given row. The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See apop_vector_correlation for an example of use.

See also

Apop_r, Apop_rv

8.2.1.22 #define Apop_notify(verbosity, ...)

Notify the user of errors, warning, or debug info.

writes to apop_opts.log_file, which is a `FILE` handle. The default is `stderr`, but use `fopen` to attach to a file.

verbosity	At what verbosity level should the user be warned? E.g., if level==2, then print iff apop_opts.verbosity >= 2.
...	The message to write to the log (presuming the verbosity level is high enough). This can be a printf-style format with following arguments, e.g., `apop_notify(0, "Beta is currently %g", beta)`.

8.2.1.23 #define Apop_r(d, rownum)

A macro to generate a temporary one-row view of apop_data set `d`, pulling out only row `row`. The view is also an apop_data set, with names and other decorations.

```
//pull a single row
apop_data *v = Apop_r(your_data, 7);

//or loop through a sequence of one-row data sets.
apop_model *std = apop_model_set_parameters(apop_normal, 0, 1);
for (int i=0; i< your_data->matrix->size1; i++)
    printf("Std Normal CDF up to observation %i is %g\n",
                        i, apop_cdf(Apop_r(your_data, i), std));
```

The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See also

Apop_rs, Apop_row_v, Apop_row_tv, Apop_row_t, Apop_mrv

8.2.1.24 #define apop_rng_get_thread(thread_in)

The `gsl_rng` is not itself thread-safe, in the sense that it can not be used simultaneously by multiple threads. However, if each thread has its own `gsl_rng`, then each will safely operate independently.

Thus, Apophenia keeps an internal store of RNGs for use by threaded functions. If the input to this function, `thread`, is greater than any previous input, then the array of `gsl_rng`s is extended to length `thread`, and each element extended using `++apop_opts.rng_seed` (i.e., the seed is incremented before use).

This function can be used anywhere a `gsl_rng` would be used.

thread_in	The number of the RNG to retrieve, starting at zero (which is how OpenMP numbers its threads). If -1, I'll look up the current thread (via `omp_get_thread_num`) for you.

See threading for additional notes. In most cases, you want to use `apop_rng_get_thread(-1)`.

Returns

The appropriate RNG, initialized if necessary.

8.2.1.25 #define Apop_row_t(d, rowname, outd)

After this call, `v` will hold an apop_data view of an apop_data set `m`. The view will consist only of the row with name `row_name`. Unlike Apop_r, the second argument is a row name, that I'll look up using apop_name_find, and the third is the name of the view to be generated.

See also

Apop_rs, Apop_r, Apop_rv, Apop_row_tv, Apop_mrv

8.2.1.26 #define Apop_row_tv(m, row, v)

After this call, `v` will hold a `gsl_vector` view of an apop_data set `m`. The view will consist only of the row with name `row_name`. Unlike Apop_rv, the second argument is a row name, that I'll look up using apop_name_find, and the third is the name of the view to be generated.

See also

Apop_rs, Apop_r, Apop_rv, Apop_row_t, Apop_mrv

8.2.1.27 #define Apop_rs(d, rownum, len)

A macro to generate a temporary view of apop_data set `d` pulling only certain rows, beginning at row `row` and having height `len`.

The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See also

Apop_r, Apop_rv, Apop_row_tv, Apop_row_t, Apop_mrv

8.2.1.28 #define Apop_rv(data_to_view, row)

A macro to generate a temporary one-row view of the matrix in an apop_data set `d`, pulling out only row `row`. The view is a `gsl_vector` set.

```
1 gsl_vector *v = Apop_rv(your_data, i);
2
3 for (int i=0; i< your_data->matrix->size1; i++)
4     printf("_%i = %g\n", i, apop_vector_sum(Apop_r(your_data, i)));
```

The view is automatically allocated, and disappears as soon as the program leaves the scope in which it is declared.

See also

Apop_r, Apop_rv, Apop_row_tv, Apop_row_t, Apop_mrv

8.2.1.29 #define Apop_settings_add_group(model, type, ...)

Add a settings group. The first two arguments (the model you are attaching to and the settings group name) are mandatory, and then you can use the Designated initializers syntax to specify default values (if any).

Returns

A pointer to the newly-prepped group.

See Settings groups or Optimization for examples.

- If a settings group of the given type is already attached to the model, the previous version is removed. Use Apop_settings_get to check whether a group of the given type is already attached to a model, and Apop_settings_set to modify an existing group.

8.2.1.30 #define Apop_settings_copy(name, ...)

A convenience macro for declaring the copy function for a new settings group. See Writing new settings groups for details and an example.

8.2.1.31 #define Apop_settings_declarations(ysg)

Put this in your header file to declare the init, copy, and free functions for ysg_settings. Of course, these functions will also have to be defined in a .c file using Apop_settings_init, Apop_settings_copy, and Apop_settings_free.

8.2.1.32 #define Apop_settings_free(name, ...)

A convenience macro for declaring the delete function for a new settings group. See Writing new settings groups for details and an example.

8.2.1.33 #define Apop_settings_get(model, type, setting)

Retrieves a setting from a model. See Apop_settings_get_group to pull the entire group.

model	An apop_model.
type	A string giving the type of the settings group you are retrieving, without the `_settings` ending. E.g., for an apop_mle_settings group, use `apop_mle`.
setting	The struct element you want to retrieve.

8.2.1.34 #define Apop_settings_get_group(m, type)

Retrieves a settings group from a model. See Apop_settings_get to just pull a single item from within the settings group.

This macro returns NULL if a group of type `type_settings` isn't found attached to model `m`, so you can easily put it in a conditional like

```
1 if (!apop_settings_get_group(m, "apop_ols")) ...
```

m	An apop_model
type	A string giving the type of the settings group you are retrieving. E.g., for an apop_mle_settings group, use only `apop_mle`.

Returns

A void pointer to the desired struct (or `NULL` if not found).

8.2.1.35 #define Apop_settings_init(name, ...)

A convenience macro for declaring the initialization function for a new settings group. See Writing new settings groups for details and an example.

8.2.1.36 #define Apop_settings_rm_group(m, type)

Removes a settings group from a model's list.

- If the so-named group is not found, do nothing.

8.2.1.37 #define Apop_settings_set(model, type, setting, data)

Modifies a single element of a settings group to the given value.

- If `model==NULL`, fails silently.
- If `model!=NULL` but the given settings group is not found attached to the model, set `model->error='s'`.

8.2.1.38 #define Apop_stopif(test, onfail, level, ...)

Execute an action and print a message to the current `FILE` handle held by `apop_opts.log_file` (default: `stderr`).

test	The expression that, if true, triggers the action.
onfail	If the assertion fails, do this. E.g., `out->error='x'; return GSL_NAN`. Notice that it is OK to include several lines of semicolon-separated code here, but if you have a lot to do, the most readable option may be `goto outro`, plus an appropriately-labeled section at the end of your function.
level	Print the warning message only if apop_opts.verbose is greater than or equal to this. Zero usually works, but for minor infractions use one, or for more verbose debugging output use 2.
...	The error message in printf form, plus any arguments to be inserted into the printf string. I'll provide the function name and a carriage return.

Some examples:

```
1 //the typical case, stopping function execution:
2 Apop_stopif(isnan(x), return NAN, 0, "x is NAN; failing");
3
4 //Mark a flag, go to a cleanup step
5 Apop_stopif(x < 0, needs_cleanup=1; goto cleanup, 0, "x is %g; cleaning up and exiting.", x);
6
7 //Print a diagnostic iff <tt>apop_opts.verbose>=1</tt> and continue
8 Apop_stopif(x < 0,  , 1, "warning: x is %g.", x);
```

- If `apop_opts.stop_on_warning` is nonzero and not `'v'`, then a failed test halts via `abort()`, even if the `apop_opts.verbose` level is set so that the warning message doesn't print to screen. Use this when running via debugger.

- If `apop_opts.stop_on_warning` is `'v'`, then a failed test halts via `abort()` iff the verbosity level is high enough to print the error.

8.2.1.39 #define Apop_subm(matrix_to_view, srow, scol, nrows, ncols)

Generate a view of a submatrix within a `gsl_matrix`. Like Apop_r, et al., the view is an automatically-allocated variable that is lost once the program flow leaves the scope in which it is declared.

data_to_view	The root matrix
srow	the first row (in the root matrix) of the top of the submatrix
scol	the first column (in the root matrix) of the left edge of the submatrix
nrows	number of rows in the submatrix
ncols	number of columns in the submatrix

Returns

An automatically-allocated view of type `gsl_matrix`.

8.2.1.40 #define apop_sum

An alias for apop_vector_sum. Returns the sum of the data in the given vector.

8.2.1.41 #define apop_text_fill(dataset, ...)

Fill the text part of an already-allocated apop_data set with a list of strings.

dataset	A data set that you already prepared with apop_text_alloc.
...	A list of strings. The first row is filled first, then the second, and so on to the end of the text grid.

- If an element is `NULL`, write `apop_opts.nan_string` at that point. You may prefer to use `""` to express a blank.
- If you provide more or fewer strings than are needed to fill the text grid and `apop_opts.verbose >=1`, I print a warning and continue to the end of the text grid or data set, whichever is shorter.
- If the data set is `NULL`, I return `NULL`. If you provide a `NULL` data set but a non-NULL list of text elements, and `apop_opts.verbose >=1`, I print a warning and return `NULL`.
- Remember that the C preprocessor concatenates two adjacent strings into one. Here is an attempt to fill a 2×3 grid:

```
1 apop_data *one23 = apop_text_fill(apop_text_alloc(NULL, 2, 3),
2                                   "one", "two", "three"   //missing comma!
3                                   "two", "four", "six");
```

  The preprocessor will join `"three" "two"` to form `"threetwo"`, leaving you with only five strings.

- If you have a `NULL`-delimited array of strings (not just a loose list as above), then use `apop_text_fill_base`.

8.2.1.42 #define apop_var

An alias for apop_vector_var. Returns the variance of the data in the given vector.

8.2.1.43 #define apop_vector_fill(avfin, ...)

Fill a pre-allocated `gsl_vector` with values.

See apop_data_alloc for a relevant example. See also apop_matrix_alloc.

Warning: I need as many arguments as the size of the vector, and can't count them for you. Too many will be ignored; too few will produce unpredictable results, which may include padding your vector with garbage or a simple segfault.

avfin	A `gsl_vector` (that you have already allocated).
...	A series of exactly as many values as there are spaces in the vector.

Returns

A pointer to the same vector that was input.

8.2.2 Function Documentation

8.2.2.1 **apop_data*** apop_anova (char * table, char * data, char * grouping1, char * grouping2)

This function produces a traditional one- or two-way ANOVA table. It works from data in an SQL table, using queries of a form like `select data from table group by grouping1, grouping2`.

table	The table to be queried. Anything that can go in an SQL `from` clause is OK, so this can be a plain table name or a temp table specification like `(select ... )`, with parens.
data	The name of the column holding the count or other such data
grouping1	The name of the first column by which to group data
grouping2	If this is `NULL`, then the function will return a one-way ANOVA. Otherwise, the name of the second column by which to group data in a two-way ANOVA.

8.2.2.2 int apop_arms_draw (double * out, gsl_rng * r, **apop_model** * m)

Adaptive rejection Metropolis sampling, to make random draws from a univariate distribution.

The author, Wally Gilks, explains on http://www.amsta.leeds.ac.uk/~wally.gilks/adaptive.↩rejection/web_page/Welcome.html , that "ARS works by constructing an envelope function of the log of the target density, which is then used in rejection sampling (see, for example, Ripley, 1987). Whenever a point is rejected by ARS, the envelope is updated to correspond more closely to the true log density, thereby reducing the chance of rejecting subsequent points. Fewer ARS rejection steps implies fewer point-evaluations of the log density."

- It accepts only functions with univariate inputs. I.e., it will put a single value into a 1x1 apop_data set, and then evaluate the log likelihood at that point. For multivariate situations, see apop_model↩_metropolis.

- It is currently the default for the apop_draw function given a univariate model, so you can just call that if you prefer.

- There are a great number of parameters, in the `apop_arms_settings` structure. The structure also holds a history of the points tested to date. That means that the system will be more accurate as more draws are made. It also means that if the parameters change, or you use apop_model_copy, you should call `Apop_settings_rm_group(your_model, apop_arms)` to clear the model of points that are not valid for a different situation.

8.2.2.3 gsl_vector* apop_array_to_vector (double * in, int size)

Copies a one-dimensional array to a `gsl_vector`. The input array is undisturbed.

in	An array of `doubles`. (No default. Must not be `NULL`);
size	How long `line` is. If this is zero or omitted, I'll guess using the `sizeof(line)/sizeof(line[0])` trick, which will work for most arrays allocated using `double []` and won't work for those allocated using `double *`. (default = auto-guess)

Returns

A `gsl_vector`, allocated and filled with a copy of (not a pointer to) the input data.

- If you send in a `NULL` vector, you get a `NULL` pointer in return. I warn you of this if `apop_opts.verbosity >=1` .
- This function uses the Designated initializers syntax for inputs.

See also

apop_data_falloc

8.2.2.4 **apop_model*** apop_beta_from_mean_var (double m, double v)

The Beta distribution is useful for modeling because it is bounded between zero and one, and can be either unimodal (if the variance is low) or bimodal (if the variance is high), and can have either a slant toward the bottom or top of the range (depending on the mean).

The distribution has two parameters, typically named α and β, which can be difficult to interpret. However, there is a one-to-one mapping between (alpha, beta) pairs and (mean, variance) pairs. Since we have good intuition about the meaning of means and variances, this function takes in a mean and variance, calculates alpha and beta behind the scenes, and returns the appropriate Beta distribution.

m	The mean the Beta distribution should have. Notice that m is in [0,1].
v	The variance which the Beta distribution should have. It is in (0, 1/12), where (1/12) is the variance of a Uniform(0,1) distribution. Funny things happen with variance near 1/12 and mean far from 1/2.

Returns

Returns an apop_model produced by copying the `apop_beta` model and setting its parameters appropriately.

out->error=='r'	Range error: mean is not within [0, 1].

8.2.2.5 **apop_data*** apop_bootstrap_cov (**apop_data** * data, **apop_model** * model, gsl_rng * rng, int iterations, char keep_boots, char ignore_nans, **apop_data** ** boot_store)

Give me a data set and a model, and I'll give you the bootstrapped covariance matrix of the parameter estimates.

data	The data set. An `apop_data` set where each row is a single data point. (No default)
model	An apop_model, whose `estimate` method will be used here. (No default)
iterations	How many bootstrap draws should I make? (default: 1,000)
rng	An RNG that you have initialized, probably with `apop_rng_alloc`. (Default: an RNG from apop_rng_get_thread)

keep_boots	Deprecated; use `boot_store`.

boot_store	If not `NULL`, put the list of drawn parameter values here, with one parameter set per row. Sample use: `apop_data *boots; apop_bootstrap_cov(data, model, .boot_store=&boots); apop_data_print(boots);` They are packed via apop_data_pack, so use apop_data_unpack if needed. (Default: 'n')
	
	`apop_data *boot_output = apop_bootstrap_cov(your_data, your_model, .keep_boots='y');`
	`apop_data *boot_stats = apop_data_get_page(boot_output, "<bootstrapped statistics>");`
	
	`printf("The statistics calculated on the 28th iteration:\n");`
	`gsl_vector *row_27 = Apop_rv(boot_stats, 27);`
	`apop_data_print(apop_data_unpack(row_27));`

ignore_nans	If `'y'` and any of the elements in the estimation return `NaN`, then I will throw out that draw and try again. If `'n'`, then I will write that set of statistics to the list, `NaN` and all. I keep count of throw-aways; if there are more than `iterations` elements thrown out, then I throw an error and return with estimates using data I have so far. That is, I assume that `NaNs` are rare edge cases; if they are as common as good data, you might want to rethink how you are using the bootstrap mechanism. (Default: 'n')

Returns

An apop_data set whose matrix element is the estimated covariance matrix of the parameters.

out->error=='n'	`NULL` input data.
out->error=='N'	`too` many NaNs.

- This function uses the Designated initializers syntax for inputs.

This example is a sort of demonstration of the Central Limit Theorem. The model is a simulation, where each call to the estimation routine produces the mean/std dev of a set of draws from a Uniform Distribution. Because the simulation takes no inputs, apop_bootstrap_cov simply re-runs the simulation and calculates a sequence of mean/std dev pairs, and reports the covariance of that generated data set.

```
#include <apop.h>

// Find the /  of a set of 10 draws from a Uniform(-1, 1)
void sim_step(apop_data *none, apop_model *m){
    int sub_draws = 20;
    static apop_model *unif;
    if (!unif) unif = apop_model_set_parameters(apop_uniform, -1, 1);
    apop_data *draws= apop_model_draws(unif, sub_draws);

    apop_data_set(m->parameters, 0, .val=apop_mean(Apop_cv(draws, 0)));
    apop_data_set(m->parameters, 1, .val=sqrt(apop_var(Apop_cv(draws, 0))));
    apop_data_add_names(m->parameters, 'r', "", "");
    apop_data_free(draws);
}

apop_model *clt_sim = &(apop_model){.name="CLT simulation", .vsize=2, .estimate=
      sim_step};

int main(){
    apop_data *boots;
    apop_data * boot_cov = apop_bootstrap_cov(NULL, clt_sim, .iterations=1000, .
      boot_store=&boots);
    apop_data_print(boot_cov);
    apop_data *means = Apop_c(boots, 0);

    printf("\nStats via Normal model:\n");
    apop_data *np = apop_estimate(means, apop_normal)->parameters;
```

```
    np->more = NULL; //rm covariance of statistics.
    apop_data_print(np);

    // from the Normal should == sqrt(cov(_boot))
    assert(fabs(sqrt(apop_data_get(boot_cov,0,0)) - apop_data_get(np, 1)) < 1e-4)
      ;
}
```

See also

apop_jackknife_cov

8.2.2.6 double apop_cdf (**apop_data** * d, **apop_model** * m)

Input a one-row data point/vector and a model; returns the area of the model's PDF beneath the given point.

By default, make random draws from the PDF and return the percentage of those draws beneath or equal to the given point. Many models have closed-form solutions that make no use of random draws.

See also apop_cdf_settings, which is the structure used to store draws already made (which means the second, third, ... calls to this function will take much less time than the first), the `gsl_rng`, and the number of draws to be made. These are handled without your involvement, but if you would like to change the number of draws from the default, add this group before calling apop_cdf :

```
1 Apop_model_add_group(your_model, apop_cdf, .draws=1e5, .rng=my_rng);
2 double cdf_value = apop_cdf(your_data_point, your_model);
```

- Only the first row of the input apop_data set is used. Note that if you need to view row 20 of a data set as a one-row data set, use Apop_r.

Here are many examples using common, mostly symmetric distributions.

```
#include <apop.h>

int main(){
    //Set up an apop_data set with only one number.
    //Most of these functions will only look at the first data point encountered.
    apop_data *onept = apop_data_falloc((1), 23);

    apop_model *norm = apop_model_set_parameters(apop_normal, 23, 138.8)
      ;
    double val = apop_cdf(onept, norm);
    assert(fabs(val - 0.5) < 1e-4);

    double tolerance = 1e-8;
    //Macroizing the sample routine above:
    #define model_val_cdf(model, value, cdf_result) {   \
        apop_data_set(onept, .val=(value));             \
        assert(fabs((apop_cdf(onept, model))-(cdf_result))< tolerance);   \
    }

    apop_model *uni = apop_model_set_parameters(apop_uniform, 20, 26);
    model_val_cdf(uni, 0, 0);
    model_val_cdf(uni, 20, 0);
    model_val_cdf(uni, 21, 1./6);
    model_val_cdf(uni, 23, 0.5);
    model_val_cdf(uni, 25, 5./6);
    model_val_cdf(uni, 26, 1);
    model_val_cdf(uni, 260, 1);

    //Improper uniform always returns 1/2.
```

```
    model_val_cdf(apop_improper_uniform, 0, 0.5);
    model_val_cdf(apop_improper_uniform, 228, 0.5);
    model_val_cdf(apop_improper_uniform, INFINITY, 0.5);

    apop_model *binom = apop_model_set_parameters(apop_binomial, 2001, 0.5);
    model_val_cdf(binom, 0, 0);
    model_val_cdf(binom, 1000, .5);
    model_val_cdf(binom, 2000, 1);

    apop_model *bernie = apop_model_set_parameters(apop_bernoulli, 0.75);
    //p(0)=.25; p(1)=.75; that determines the CDF.
    //Notice that the CDF's integral is over a closed interval.
    model_val_cdf(bernie, -1, 0);
    model_val_cdf(bernie, 0, 0.25);
    model_val_cdf(bernie, 0.1, 0.25);
    model_val_cdf(bernie, .99, 0.25);
    model_val_cdf(bernie, 1, 1);
    model_val_cdf(bernie, INFINITY, 1);

    //alpha=beta -> symmetry
    apop_model *beta = apop_model_set_parameters(apop_beta, 2, 2);
    model_val_cdf(beta, -INFINITY, 0);
    model_val_cdf(beta, 0.5, 0.5);
    model_val_cdf(beta, INFINITY, 1);

    //This beta distribution -> uniform
    apop_model *beta_uni = apop_model_set_parameters(apop_beta, 1, 1);
    model_val_cdf(beta_uni, 0, 0);
    model_val_cdf(beta_uni, 1./6, 1./6);
    model_val_cdf(beta_uni, 0.5, 0.5);
    model_val_cdf(beta_uni, 1, 1);


    beta_uni->cdf = NULL; //With no closed-form CDF; make random draws to estimate the CDF.
    Apop_model_add_group(beta_uni, apop_cdf, .draws=1e6); //extra draws to improve accuracy, but we
       have to lower our tolerance anyway.
    tolerance=1e-3;
    model_val_cdf(beta_uni, 0, 0);
    model_val_cdf(beta_uni, 1./6, 1./6);
    model_val_cdf(beta_uni, 0.5, 0.5);
    model_val_cdf(beta_uni, 1, 1);


    //sum of three symmetric distributions: still symmetric.
    apop_model *sum_of_three = apop_model_mixture(beta, apop_improper_uniform, beta_uni);
    model_val_cdf(sum_of_three, 0.5, 0.5);


    apop_data *threepts = apop_data_falloc((3,1), -1, 0, 1);
    apop_model *kernels = apop_estimate(threepts, apop_kernel_density);
    model_val_cdf(kernels, -5, 0);
    model_val_cdf(kernels, 0, 0.5);
    model_val_cdf(kernels, 10, 1);
}
```

8.2.2.7 void apop_crosstab_to_db (**apop_data** * in, char * tabname, char * row_col_name, char * col_col_name, char * data_col_name)

See apop_db_to_crosstab for the storyline; this is the complement, which takes a crosstab and writes its values to the database.

For example, I would take

	c0	c1
r0	2	3

r1	0	4

and do the following writes to the database:

```
1 insert into your_table values ('r0', 'c0', 2);
2 insert into your_table values ('r0', 'c1', 3);
3 insert into your_table values ('r1', 'c0', 3);
4 insert into your_table values ('r1', 'c1', 4);
```

- If your data set does not have names (or not enough names), I will use the scheme above, filling in names of the form `r0`, `r1`, ... `c0`, `c1`, ... Text columns get their own names, `t0`, `t1`.

- This function handles only the matrix and text.

8.2.2.8 void apop_data_add_named_elmt (**apop_data** * d, char * name, double val)

A convenience function to add a named element to a data set. Many of Apophenia's testing procedures use this to easily produce a column of named parameters. It is public as a convenience.

d	The apop_data structure. Must not be `NULL`, but may be blank (as per allocation via apop_data_alloc `( )`).
name	The name to add
val	the value to add to the set.

- I use the position of the last non-empty row name to know where to put the value. If there are two names in the data set, then I will put the new name in the third name slot and the data in the third slot in the vector. If you use this function from start to finish in building your list, then you'll be fine.

- If the vector is too short (or `NULL`), I will call apop_vector_realloc internally to make space.

- This fits well with the defaults for apop_data_get. An example:

```
1 apop_data *list = apop_data_alloc();
2 apop_data_add_named_elmt(list, "height", 165);
3 apop_data_add_named_elmt(list, "weight", 60);
4
5 double height = apop_data_get(list, .rowname="height");
6
7 //or
8 #define Lookup(dataset, key) apop_data_get(dataset, .rowname=#key)
9 height = Lookup(list, height);
```

8.2.2.9 **apop_data*** apop_data_add_page (**apop_data** * dataset, **apop_data** * newpage, const char * title)

Add a page to an apop_data set. It gets a name so you can find it later.

dataset	The input data set, to which a page will be added.
newpage	The page to append
title	The name of the new page.

Returns

The new page. I post a warning if I am appending or appending to a `NULL` data set and `apop_↩ opts.verbose >=1` .

- See Pages for further notes.

8.2.2.10 **apop_data*** apop_data_alloc (const size_t size1, const size_t size2, const int size3)

Allocate an apop_data structure.

- The typical case is three arguments, like `apop_data_alloc(2,3,4)`: vector size, matrix rows, matrix cols. If the first argument is zero, you get a `NULL` vector.
- Two arguments, `apop_data_alloc(2,3)`, would allocate just a matrix, leaving the vector `NULL`.
- One argument, `apop_data_alloc(2)`, would allocate just a vector, leaving the matrix `NULL`.
- Zero arguments, `apop_data_alloc()`, will produce a basically blank set, with `out->matrix` and `out->vector` set to `NULL`.

For allocating the text part, see apop_text_alloc.

The `weights` vector is set to `NULL`. If you need it, allocate it via

```
1 d->weights = gsl_vector_alloc(row_ct);
```

Returns

The apop_data structure, allocated and ready to be populated with data.

out->error=='a'	Allocation error. The matrix, vector, or names couldn't be `malloc`ed, which probably means that you requested a very large data set.

- An apop_data struct, by itself, is about 72 bytes. If I can't allocate that much memory, I return `NULL`. But if even this much fails, your computer may be on fire and you should go put it out.
- This function uses the Designated initializers syntax for inputs.

See also

apop_data_calloc

8.2.2.11 **apop_data*** apop_data_calloc (const size_t size1, const size_t size2, const int size3)

Allocate a apop_data structure, to be filled with data; set everything in the allocated portion to zero. See apop_data_alloc for details.

Returns

The apop_data structure, allocated and zeroed out.

out->error=='a'	allocation error; probably out of memory. - This function uses the Designated initializers syntax for inputs.

See also

apop_data_alloc

8.2.2.12 **apop_data*** apop_data_copy (const **apop_data** * in)

Copy one apop_data structure to another. That is, all data is duplicated.

Basically a front-end for apop_data_memcpy for those who prefer this sort of syntax.

If the data set has a `more` pointer, that will be followed and subsequent pages copied as well.

in	the input data

Returns

a structure that this function will allocate and fill. If input is NULL, then this will be NULL.

out.error='a'	Allocation error.
out.error='c'	Cyclic link: `D->more == D` (may be later in the chain, e.g., `D->more->more = D->more`) You'll have only a partial copy.
out.error='d'	Dimension error; should never happen.
out.error='p'	Missing part error; should never happen.

- If the input data set has an error, then I will copy it anyway, including the error flag (which might be overwritten). I print a warning if the verbosity level is `>=1`.

8.2.2.13 **apop_data*** apop_data_correlation (const **apop_data** * in)

Returns the matrix of correlation coefficients $(\sigma_{xy}^2/(\sigma_x\sigma_y))$ relating each column with each other.

in	A data matrix: rows are observations, columns are variables. If you give me a weights vector, I'll use it.

Returns

Returns the square variance/covariance matrix with dimensions equal to the number of input columns.

out->error='a'	Allocation error.

8.2.2.14 **apop_data*** apop_data_covariance (const **apop_data** * in)

Returns the sample variance/covariance matrix relating each column of the matrix to each other column.

in	An apop_data set. If the weights vector is set, I'll take it into account.

- This is the sample covariance—dividing by $n-1$, not n. If you need the population variance, use

```
1 apop_data *popcov = apop_data_covariance(indata);
2 int size=indata->matrix->size1;
3 gsl_matrix_scale(popcov->matrix, size/(size-1.));
```

Returns

Returns an apop_data set the variance/covariance matrix.

out->error='a'	Allocation error.

8.2.2.15 char apop_data_free_base (**apop_data** * freeme)

Free the elements of the given apop_data set and then the apop_data set itself. Intended to be used by apop_data_free, a macro that calls this to free elements, then sets the value to `NULL`.

- apop_data_free is a macro that calls this function and, on success, sets the input pointer to `NULL`. For typical cases, that's slightly more useful than this function.

freeme.error='c'	Circular linking is against the rules. If `freeme->more == freeme`, then I set `freeme.error='c'` and return. If you send in a structure like A -> B -> B, then both data sets A and B will be marked.

Returns

`0` on OK, `'c'` on error.

8.2.2.16 double apop_data_get (const **apop_data** * data, size_t row, int col, const char * rowname, const char * colname, const char * page)

Returns the data element at the given point.

In case of error (probably that you asked for a data point out of bounds), returns `NAN`. See the set/get page for details and examples.

data	The data set. Must not be `NULL`.
row	The row number of the desired element. If `rowname==NULL`, default is zero.
col	The column number of the desired element. -1 indicates the vector. If `colname==NULL`, default is zero if the `->matrix` element is not `NULL` and -1 if the `->matrix` element is `NULL` and the `->vector` element is not.
rowname	The row name of the desired element. If `NULL`, use the row number.
colname	The column name of the desired element. If `NULL`, use the column number.
page	The case-insensitive name of the page on which the element is found. If `NULL`, use first page.

Returns

The value at the given location.

8.2.2.17 **apop_data*** apop_data_get_factor_names (**apop_data** * data, int col, char type)

Factor names are stored in an auxiliary table with a name like `"<categories for your_var>"`. Producing this name is annoying (and prevents us from eventually making it human-language independent), so use this function to get the list of factor names.

data	The data set. (No default, must not be `NULL`)
col	The column in the main data set whose name I'll use to check for the factor name list. Vector==-1. (default=0)
type	If you are referring to a text column, use 't'. (default='d')

Returns

A pointer to the page in the data set with the given factor names.

- This function uses the Designated initializers syntax for inputs.

8.2.2.18 **apop_data*** apop_data_get_page (const **apop_data** * data, const char * title, const char match)

It's good form to get a page from your data set by name, because you may not know the order for the pages, and the stepping through makes for dull code anyway (`apop_data *page = dataset; while (page->more) page= page->more;`).

data	The apop_data set to use. No default; if NULL, gives a warning if apop_opts.↩ verbose >=1 and returns NULL.
title	The name of the page to retrieve. Default="<Info>", which is the name of the page of additional estimation information returned by estimation routines (log likelihood, status, AIC, BIC, confidence intervals, ...).
match	If 'c', case-insensitive match (via strcasecmp); if 'e', exact match, if 'r' regular expression substring search (via apop_regex). Default='c'.

Returns

The page whose title matches what you gave me. If I don't find a match, return NULL.

- This function uses the Designated initializers syntax for inputs.

8.2.2.19 **apop_data*** apop_data_listwise_delete (**apop_data** * d, char inplace)

If there is an NaN anywhere in the row of data (including the matrix, the vector, the weights, and the text) then delete the row from the data set.

- If every row has a NaN, then this returns NULL.
- If apop_opts.nan_string is not NULL, then I will make case-insensitive comparisons to the text elements to check for bad data as well.
- If inplace = 'y', then I'll free each element of the input data set and refill it with the pruned elements. I'll still take up (up to) twice the size of the data set in memory during the function. If every row has a NaN, then your apop_data set will end up with NULL vector, matrix, ... if inplace = 'n', then the original data set is left where it was, though internal elements may be moved.
- I only look at the first page of data (i.e. the more element is ignored).
- Listwise deletion is often not a statistically valid means of dealing with missing data. It is typically better to impute the data (preferably multiple times). See apop_ml_impute for a less-invalid means, or Tea for survey imputation for heavy-duty survey editing and imputation.
- This function uses the Designated initializers syntax for inputs.

d	The data, with NaNs
inplace	If 'y', clear out the pointer-to-apop_data that you sent in and refill with the pruned data. If 'n', leave the set alone and return a new data set. Default='n'.

Returns

A (potentially shorter) copy of the data set, without NaNs. If inplace=='y', a pointer to the input, which was shortened in place. If the entire data set is cleared out, then this will be NULL.

See also

apop_data_rm_rows

8.2.2.20 void apop_data_memcpy (**apop_data** * out, const **apop_data** * in)

Copy one apop_data structure to another.

This function does not allocate the output structure or the vector, matrix, text, or weights elements—I assume you have already done this and got the dimensions right. I will assert that there is at least enough room in the destination for your data, and fail if the copy would write more elements than there are bins.

- If you want space allocated or are unsure about dimensions, use apop_data_copy.
- If both `in` and `out` have a `more` pointer, also copy subsequent page(s).
- You can use the subsetting macros, Apop_r, Apop_rs, Apop_c, and so on, to copy within a data set:

```
1 //Copy the contents of row i of mydata to row j.
2 apop_data *fromrow = Apop_r(mydata, i);
3 apop_data *torow = Apop_r(mydata, j);
4 apop_data_memcpy(torow, fromrow);
5
6 // or just
7 apop_data_memcpy(Apop_r(mydata, i), Apop_r(mydata, j));
```

out	A structure that this function will fill. Must be preallocated with the appropriate sizes.
in	The input data.

out.error='d'	Dimension error.
out.error='p'	Part missing; e.g., in->matrix exists but out->matrix doesn't.

8.2.2.21 gsl_vector* apop_data_pack (const **apop_data** * in, gsl_vector * out, char more_pages, char use_info_pages)

This function takes in an apop_data set and writes it as a single column of numbers, outputting a `gsl_vector`. It is valid to use the `out_vector->data` element as an array of `doubles` of size `out_vector->data->size` (i.e. its `stride==1`).

The complement is `apop_data_unpack`. I.e.,

```
1 apop_data_unpack(apop_data_pack(in_data), data_copy)
```

will return the original data set (stripped of text and names).

in	an `apop_data` set. No default; if `NULL`, return `NULL`.
out	If this is not `NULL`, then put the output here. The dimensions must match exactly. If `NULL`, then allocate a new data set. Default = `NULL`.
more_pages	If `'y'`, then follow the `->more` pointer to fill subsequent pages; else fill only the first page. Informational pages will still be ignored, unless you set `.use_info_pages='y'` as well. Default = `'y'`.
use_info_pages	Pages in XML-style brackets, such as `<Covariance>` will be ignored unless you set `.use_info_pages='y'`. Be sure that this is set to the same thing when you both pack and unpack. Default: `'n'`.

Returns

A `gsl_vector` with the vector data (if any), then each row of data (if any), then the weights (if any), then the same for subsequent pages (if any `&& .more_pages=='y'`). If `out` is not `NULL`, then this is `out`.

NULL	If you give me a vector as input, and its size is not correct, returns `NULL`.

- This function uses the Designated initializers syntax for inputs.

8.2.2.22 **apop_data*** apop_data_pmf_compress (**apop_data** * in)

Say that you have added a long list of observations to a single apop_data set, meaning that each row has weight one. There are a huge number of duplicates, perhaps because there are a handful of types that keep repeating:

Vector value	Text name	Weights
12	Dozen	1
1	Single	1
2	Pair	1
2	Pair	1
1	Single	1
1	Single	1
2	Pair	1
2	Pair	1

Use this function to reduce this to a set of distinct values, with their weights adjusted accordingly:

Vector value	Text name	Weights
12	Dozen	1
1	Single	3
2	Pair	4

in	An apop_data set that may have duplicate rows. As above, the data may be in text and/or numeric formats.

Returns

Your input is changed in place, via apop_data_rm_rows, so use apop_data_copy before calling this function if you need to retain the original format. For your convenience, this function returns a pointer to your original data, which has now been pruned. If there is a `weights` vector, I will add those weights together as duplicates are merged. If there is no `weights` vector, I will create one, which is initially set to one for all values, and then aggregated as above.

8.2.2.23 void apop_data_print (const **apop_data** * data, Output_declares)

Print an apop_data set to a file, the database, or the screen, as determined by the .output_type.

- See apop_prep_output for more on how printing settings are set.
- See Legible output for more details and examples.
- See About SQL, the syntax for querying databases for notes on writing an apop_data set to the database.
- This function uses the Designated initializers syntax for inputs.

8.2.2.24 **apop_data*** apop_data_prune_columns_base (**apop_data** * d, char ** colnames)

Keep only the columns of a data set that you name. This is the function called internally by the apop_data_prune_columns macro. In most cases, you'll want to use that macro. An example of the two uses demonstrating the difference:

```
1 apop_data_prune_columns(d, "mean", "median");
2
3 char *list[] = {"mean", "median", NULL};
4 apop_data_prune_columns_base(d, list);
```

d	The data set to prune.
colnames	A NULL-terminated list of names to retain.

Returns

A pointer to the input data set, now pruned.

See also

apop_data_rm_columns

8.2.2.25 double* apop_data_ptr (**apop_data** * data, int row, int col, const char * rowname, const char * colname, const char * page)

Get a pointer to an element of an apop_data set.

- If a `NULL` vector or matrix (as the case may be), or the row/column you requested is outside bounds, return `NULL`.
- See the set/get page for details.

data	The data set. Must not be `NULL`.
row	The row number of the desired element. If `rowname==NULL`, default is zero.
col	The column number of the desired element. -1 indicates the vector. If `colname==NULL`, default is zero.
rowname	The row name of the desired element. If `NULL`, use the row number.
colname	The column name of the desired element. If `NULL`, use the column number.
page	The case-insensitive name of the page on which the element is found. If `NULL`, use first page.

Returns

A pointer to the element.

8.2.2.26 **apop_data*** apop_data_rank_compress (**apop_data** * in, int min_bins)

One often finds data where the column indicates the value of the data point. There may be two columns, and a mark in the first indicates a miss while a mark in the second is a hit. Or say that we have the following list of observations:

```
1 2 3 3 2 1 1 2 1 1 2 1 1
```

Then we could write this as:

```
1 0  1  2  3
2 ----------
3 0  6  4  2
```

because there are six 1s observed, four 2s observed, and two 3s observed. We call this rank format, because 1 (or zero) is typically the most common, 2 is second most common, et cetera.

This function takes in a list of observations, and aggregates them into a single row in rank format.

- For the complement, see apop_data_rank_expand.

- See also apop_data_to_factors to convert real numbers or text into a matrix of categories.

in	The input apop_data set. If `NULL`, return `NULL`.
min_bins	If this is omitted, the number of bins is simply the largest number found. So if there are bins {0, 1, 2} and your data set happens to consist of `0 0 1 1 0`, then I won't know to generate results with three bins where the last bin has a count of zero. Set `.min_bins=2` to ensure that bin is included.

```
/* A round trip: generate Zipf-distributed draws, summarize them to a single list of
rankings, then expand the rankings to a list of single entries. The sorted list at the end
of this should be identical to the (sorted) original list. */
#include <apop.h>

int main(){
    gsl_rng *r = apop_rng_alloc(2342);
    int i, length = 1e4;
    apop_model *a_zipf = apop_model_set_parameters(apop_zipf, 3.2);
    apop_data *draws = apop_data_alloc(length);
    for (i=0; i< length; i++)
        apop_draw(apop_data_ptr(draws, i, -1), r, a_zipf);
    apop_data *by_rankings = apop_data_rank_compress(draws);
    //The first row of the matrix is suitable for plotting.
    //apop_data_show(by_rankings);
    assert(apop_matrix_sum(by_rankings->matrix) == length);

    apop_data *re_expanded = apop_data_rank_expand(by_rankings);
    gsl_sort_vector(draws->vector);
    gsl_sort_vector(re_expanded->vector);
    assert(apop_vector_distance(draws->vector, re_expanded->vector) < 1e-5);
}
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.27 **apop_data*** apop_data_rank_expand (**apop_data** * in)

The complement to this is apop_data_rank_compress; see that function's documentation for the story and an example.

This function takes in a data set where the zeroth column includes the count(s) of times that zero was observed, the first gives the count(s) of times that one was observed, et cetera. It outputs a data set whose vector element includes a list that has exactly the given frequency of zeros, ones, et cetera.

8.2.2.28 void apop_data_rm_columns (**apop_data** * d, int * drop)

Remove the columns of the apop_data set corresponding to a nonzero value in the `drop` vector.

- The returned data structure looks like it was modified in place, but the data matrix and the names are duplicated before being pared down, so if your data is taking up more than half of your memory, this may not work.

d	The apop_data structure to be pared down.
drop	An array of ints. If use[7]==1, then column seven will be cut from the output. A reminder: `calloc(in->size2 , sizeof(int))` will fill your array with zeros on allocation, and `memset(use, 1, in->size2 * sizeof(int))` will quickly fill an array of ints with nonzero values. apop_data_rm_rows

8.2.2.29 **apop_data*** apop_data_rm_page (**apop_data** * data, const char * title, const char free_p)

Remove the first page from an apop_data set that matches a given name.

data	The input data set, from which a page will be removed. No default. If `NULL`, maybe print a warning (see below).
title	The case-insensitive name of the page to remove. Default: `"<Info>"`
free_p	If `'y'`, then apop_data_free the page. Default: `'y'`.

Returns

If not freed, a pointer to the apop_data page that I just pulled out. Thus, you can use this to pull a single page from a data set. I set that page's `more` pointer to `NULL`, to minimize any confusion about more-than-linear linked list topologies. If `free_p=='y'` (the default) or the page is not found, return `NULL`.

- I don't check the first page, so there's no concern that the head of your list of pages will move. Again, the intent of the `->more` pointer in the apop_data set is not to fully implement a linked list, but primarily to allow you to staple auxiliary information to a main data set.
- If I don't find the page you want, I return NULL, and maybe print a warning; see below.
- For the two above cases where a warning may be printed, if the page is to be returned and `apop_opts.verbose >= 1` , print a warning. If the page is to be freed and `apop_opts.verbose >= 2` , print a warning.
- The remaining `more` pointers in the apop_data set are adjusted accordingly.

8.2.2.30 **apop_data*** apop_data_rm_rows (**apop_data** * in, int * drop, apop_fn_ir do_drop, void * drop_parameter)

Remove the rows set to one in the `drop` vector or for which the `do_drop` function returns one.

in	the apop_data structure to be pared down
drop	a vector with as many elements as the max of the vector, matrix, or text parts of `in`, with a one marking those rows to be removed.

do_drop	A function that returns one for rows to drop and zero for rows to not drop. A sample function:
	`1 int your_drop_function(apop_data *onerow, void *extra_param){`
	`2  return gsl_isnan(apop_data_get(onerow)) ||`
	`3  !strcmp(onerow->text[0][0], "Uninteresting data point");`
	`4 }`
	apop_data_rm_rows will use Apop_r to get a subview of the input data set of height one, and send that subview to this function (and since arguments typically default to zero, you don't have to write out things like apop_data_get `(onerow, .row=0, .col=0)`, which can help to keep things readable).

drop_parameter	If your `do_drop` function requires additional input, put it here and it will be passed through.

Returns

Returns a pointer to the input data set, now pruned.

- If all the rows are to be removed, then you will wind up with the same apop_data set, with `NULL` `vector`, `matrix`, `weight`, and text. Therefore, you may wish to check for `NULL` elements after use. I remove rownames, but leave the other names, in case you want to add new data rows.
- The typical use is to provide only a list or only a function. If both are `NULL`, I return without doing anything, and print a warning if `apop_opts.verbose >=2`. If you provide both, I will drop the row if either the vector has a one in that row's position, or if the function returns a nonzero value.
- This function uses the Designated initializers syntax for inputs.

See also

apop_data_listwise_delete, apop_data_rm_columns

8.2.2.31 int apop_data_set (**apop_data** * data, size_t row, int col, const double val, const char * colname, const char * rowname, const char * page)

Set a data element. See the set/get page for details and examples.

Returns

0=OK, -1=error: couldn't find row/column name, or you asked for a location outside the vector/matrix bounds.

- The error codes for out-of-bounds errors are thread-safe iff you are have a C11-compliant compiler (thanks to the `_Thread_local` keyword) or a version of GCC with the `__thread` extension enabled.
- Set weights via `gsl_vector_set(your_data->weights, row, val);`.
- Set text elements via apop_text_set.

data	The data set. Must not be `NULL`.
row	The row number of the desired element. If `rowname==NULL`, default is zero.
col	The column number of the desired element. -1 indicates the vector. If `colname==NULL`, default is zero.
rowname	The row name of the desired element. If `NULL`, use the row number.
colname	The column name of the desired element. If `NULL`, use the column number.
page	The case-insensitive name of the page on which the element is found. If `NULL`, use first page.
val	The value to give the point.

- This function uses the Designated initializers syntax for inputs.

8.2.2.32 **apop_data*** apop_data_sort (**apop_data** * data, **apop_data** * sort_order, char asc, char inplace, double * col_order)

Sort an apop_data set on an arbitrary sequence of columns.

The `sort_order` set is a one-row data set that should look like the data set being sorted. The easiest way to generate it is to use Apop_r to pull one row of the table, then copy and fill it. For each column you want used in the sort, assign a ranking giving whether the column should be sorted first, second, ... Columns you don't want used in the sorting should be set to `NAN`. Ties are broken by the earlier element in the default order (see below).

E.g., to sort by the last column of a five-column matrix first, then the next-to-last column, then the next-to-next-to-last, then by the first text column, then by the second text column:

```
1 apop_data *sort_order = apop_data_copy(Apop_r(data, 0));
2 sort_order->vector = NULL; //so it will be skipped.
3 Apop_data_fill(sort_order, NAN, NAN, 3, 2, 1);
4 apop_text_set(sort_order, 0, 0, "4");
5 apop_text_set(sort_order, 0, 1, "5");
6 apop_data_sort(data, sort_order);
```

To determine which columns are sorted at which step, I use only comparisons, not the actual numeric values. For example, (1, 2, 3) and (-1.32, 0, 27) work identically. For text, I use `atof` to convert the your text to a number, as in the example above that set text values of `"4"` and `"5"`. A blank string, NaN numeric value, or NULL element in the apop_data set means that column will not be sorted.

- Strings are sorted case-insensitively, using `strcasecmp`. [exercise for the reader: modify the source to use Glib's locale-correct string sorting.]

- The setup generates a lexicographic sort using the columns you specify. If you would like a different sort order, such as Euclidian distance to the origin, you can generate a new column expressing your preferred metric, and then sorting on that. See the example below.

data	The data set to be sorted. If `NULL`, this function is a no-op that returns `NULL`.
sort_order	An apop_data set describing the order in which columns are used for sorting, as above. If `NULL`, then sort by the vector, then each matrix column, then text, then weights, then row names.
inplace	If 'n', make a copy, else sort in place. (default: 'y').
asc	If 'a', ascending; if 'd', descending. This is applied to all columns; column-by-column application is to do. (default: 'a').
col_order	For internal use only. In your call, it should be `NULL`; you can leave this off your function call entirely and the Designated initializers syntax will takes care of it for you.

Returns

A pointer to the sorted data set. If `inplace=='y'` (the default), then this is the same as the input set.

A few examples:

```
#include <apop.h>
#include <unistd.h>
#ifdef Testing
#include "sort_tests.c" //For Apophenia's test suite, some tedious checks that the sorts worked
#endif

//get_distance is for the sort-by-Euclidian distance example below.
double get_distance(gsl_vector *v) {return apop_vector_distance(v);}

int main(){
    apop_text_to_db("amash_vote_analysis.csv");
```

```
    apop_data *d = apop_query_to_mixed_data("mntmtm", "select
       1,id,party,contribs/1000.0,vote,ideology from amash_vote_analysis ");

    //use the default order of columns for sorting
    apop_data *sorted = apop_data_sort(d, .inplace='n');
#ifndef Testing
    apop_data_print(sorted);
#else
    check_sorting1(sorted);
#endif

    //set up a specific column order
    apop_data *perm = apop_data_copy(Apop_r(d, 0));
    perm->vector = NULL;
    apop_data_fill(perm, 5, 3, 4);
    apop_text_set(perm, 0, 0, "2");
    apop_text_set(perm, 0, 1, "1");

    apop_data_sort(d, perm);
#ifndef Testing
    apop_data_print(d);
#else
    check_sorting2(d);
#endif

    //sort a list of names
    apop_data *blank = apop_data_alloc();
    apop_data_add_names(blank, 'r', "C", "E", "A");
    apop_data_sort(blank);
    assert(*blank->names->row[0] == 'A');
    assert(*blank->names->row[1] == 'C');
    assert(*blank->names->row[2] == 'E');

    //take each row of the matrix as a vector; store the Euclidian distance to the origin in the vector;
    //sort in descending order.
    apop_data *rowvectors = apop_text_to_data("test_data");
    apop_map(rowvectors, .fn_v=get_distance, .part='r', .inplace='y');
    apop_data *arow = apop_data_copy(Apop_r(rowvectors, 0));
    arow->matrix=NULL; //sort only by the distance vector
    apop_data_sort(rowvectors, arow, .asc='d');
#ifndef Testing
    apop_data_print(rowvectors);
#else
    double prev = INFINITY;
    for (int i=0; i< rowvectors->vector->size; i++){
        double this = apop_data_get(rowvectors, i, -1);
        assert(this < prev);
        prev = this;
    }
#endif
}
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.33 **apop_data**** apop_data_split (**apop_data** * in, int splitpoint, char r_or_c)

Split one input apop_data structure into two.

For the opposite operation, see apop_data_stack.

in	The apop_data structure to split
splitpoint	The index of what will be the first row/column of the second data set. E.g., if this is -1 and `r_or_c=='c'`, then the whole data set will be in the second data set; if this is the length of the matrix then the whole data set will be in the first data set. Another way to put it is that for values between zero and the matrix's size, `splitpoint` will equal the number of rows/columns in the first matrix.
r_or_c	If this is 'r' or 'R', then put some rows in the first data set and some in the second; of 'c' or 'C', split columns into first and second data sets.

Returns

An array of two apop_data sets. If one is empty then a `NULL` pointer will be returned in that position. For example, for a data set of 50 rows, `apop_data **out = apop_data_split(data, 100, 'r')` sets `out[0] = apop_data_copy(data)` and `out[1] = NULL`.

- When splitting at a row, the text is also split.
- The `more` pointer is ignored.
- The `apop_data->vector` is taken to be the -1st element of the matrix.
- Weights will be preserved. If splitting by rows, then the top and bottom parts of the weights vector will be assigned to the top and bottom parts of the main data set. If splitting by columns, identical copies of the weights vector will be assigned to both parts.
- Data is copied, so you may want to call `apop_data_free(in)` after this.

8.2.2.34 **apop_data*** apop_data_stack (**apop_data** * m1, **apop_data** * m2, char posn, char inplace)

Put the first data set either on top of or to the left of the second data set.

For the opposite operation, see apop_data_split.

m1	the upper/rightmost data set (default = `NULL`)
m2	the second data set (default = `NULL`)
posn	If 'r', stack rows of m1 above rows of m2 if 'c', stack columns of m1 to left of m2's (default = 'r')
inplace	If `'y'`, use apop_matrix_realloc and apop_vector_realloc to modify `m1` in place. Otherwise, allocate a new apop_data set, leaving `m1` undisturbed. (default='n')

Returns

The stacked data, either in a new apop_data set or `m1`

out->error=='a'	Allocation error.
out->error=='d'	Dimension error; couldn't make a complete copy.

- The function returns a new data set, meaning that until you apop_data_free() the original data sets, you will be taking up twice as much memory.
- If m1 or m2 are `NULL`, returns a copy of the other element, and if both are `NULL`, returns `NULL`. If `m2` is `NULL` and `inplace` is `'y'`, returns the original `m1` pointer unmodified.

- Text is handled as you'd expect: If 'r', one set of text is stacked on top of the other [number of columns must match]; if 'c', one set of text is set next to the other [number of rows must match].
- `more` is ignored.
- If stacking rows on rows, the output vector is the input vectors stacked accordingly. If stacking columns by columns, the output vector is just a copy of the vector of `m1` and `m2->vector` doesn't appear in the output at all.
- The same rules for dealing with the vector(s) hold for the vector(s) of weights.
- Names are a copy of the names for `m1`, with the names for `m2` appended to the row or column list, as appropriate.
- This function uses the Designated initializers syntax for inputs.

8.2.2.35 **apop_data*** apop_data_summarize (**apop_data** * indata)

Put summary information about the columns of a table (mean, std dev, variance, min, median, max) in a table.

indata	The table to be summarized. An apop_data structure. May have a `weights` element.

Returns

An apop_data structure with one row for each column in the original table, and a column for each summary statistic.

out->error='a'	Allocation error.

- This function gives more columns than you probably want; use apop_data_prune_columns to pick the ones you want to see.
- See apop_data_prune_columns for an example.

8.2.2.36 **apop_data*** apop_data_to_bins (**apop_data** const * indata, **apop_data** const * binspec, int bin_count, char close_top_bin)

Create a histogram from data by putting data into bins of fixed width. Your input apop_data set may be multidimensional, and may include both vector and matrix parts, and the bins output will have corresponding dimension.

indata	The input data that will be binned, one observation per row. This is copied and the copy will be modified. (No default)

binspec	This is an apop_data set with the same number of columns as `indata`. If you want a fixed size for the bins, then the first row of the bin spec is the bin width for each column. This allows you to specify a width for each dimension, or specify the same size for all with something like:
	`1 apop_data *binspec = apop_data_copy(Apop_r(indata, 0));`
	`2 gsl_matrix_set_all(binspec->matrix, 10); //bins of size 10 for all dim.s`
	`3 apop_data_to_bins(indata, binspec);`
	The presumption is that the first bin starts at zero in all cases. You can add a second row to the spec to give the offset for each dimension. (default: NULL)

bin_count	If you don't provide a bin spec, I'll provide this many evenly-sized bins to cover the data set. (Default: $\sqrt{N}$)
close_top_bin	Normally, a bin covers the range from the point equal to its minimum to points strictly less than the minimum plus the width. if `'y'`, then the top bin includes points less than or equal to the upper bound. This solves the problem of displaying histograms where the top bin is just one point. (default: `'y'` if `binspec==NULL`, else `'n'`)

Returns

A pointer to an apop_data set with the same dimension as your input data. Each cell is an integer giving the bin number into which the cell falls.

- If no binspec and no binlist, then a grid with offset equal to the min of the column, and bin size such that it takes $\sqrt{N}$ bins to cover the range to the max element.
- The text segment is not binned. The `more` pointer, if any, is not followed.
- Given `NULL` input, return `NULL` output. Print a warning if `apop_opts.verbose >= 2`.

Iff you didn't give me a binspec, then I attach one to the output set as a page named `<binspec>`. This means that you can snap a second data set to the same grid using

```
1 apop_data_to_bins(first_set, NULL);
2 apop_data_to_bins(second_set, apop_data_get_page(first_set, "<binspec>"));
```

- If you want to plot the output, it may help to run it through apop_data_pmf_compress to produce a vector of bin weights.

Here is a sample program highlighting apop_data_to_bins and apop_data_pmf_compress .

```
#define _GNU_SOURCE
#include <apop.h>

#define printdata(dataset)            \
        printf("\n-----------\n\n"); \
        apop_data_print(dataset);

int main(){
    apop_data *d = apop_text_alloc(apop_data_alloc(6), 6, 1);
    apop_data_fill(d,   1,   2,   3,   3,   1,   2);
    apop_text_fill(d,  "A", "A", "A", "A", "A", "B");

    asprintf(&d->names->title, "Original data set");
    printdata(d);

        //binned, where bin ends are equidistant but not necessarily in the data
    apop_data *binned = apop_data_to_bins(d);
    asprintf(&binned->names->title, "Post binning");
    printdata(binned);
    assert(fabs(//equal distance between bins
              (apop_data_get(binned, 1) - apop_data_get(binned, 0))
            - (apop_data_get(binned, 2) - apop_data_get(binned, 1))) < 1e-5);

        //compressed, where the data is as in the original, but weights
        //are redone to accommodate repeated observations.
    apop_data_pmf_compress(d);
    asprintf(&d->names->title, "Post compression");
    printdata(d);
    assert(apop_sum(d->weights)==6);

    apop_model *d_as_pmf = apop_estimate(d, apop_pmf);
```

```
    apop_data *firstrow = Apop_r(d, 0); //1A
    assert(fabs(apop_p(firstrow, d_as_pmf) - 2./6 < 1e-5));
}
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.37 **apop_data*** apop_data_to_dummies (**apop_data** * d, int col, char type, int keep_first, char append, char remove)

A utility to make a matrix of dummy variables. You give me a single vector that lists the category number for each item, and I'll produce a matrix with a single one in each row in the column specified.

After that, you have to decide what to do with the new matrix and the original data column.

- You can manually join the dummy data set with your main data, e.g.:

```
1 apop_data *dummies  = apop_data_to_dummies(main_regression_vars, .col=8, .type='t');
2 apop_data_stack(main_regression_vars, dummies, 'c', .inplace='y');
```

- The `.remove='y'` option specifies that I should use apop_data_rm_columns to remove the column used to generate the dummies. Implemented only for `type=='d'`.

- By specifying `.append='y'` or `.append='e'` I will run the above two lines for you. Your apop_data pointer will not change, but its `matrix` element will be reallocated (via apop_data_stack).

- By specifying `.append='i'`, I will place the matrix of dummies in place, immediately after the data column you had specified. You will probably use this with `.remove='y'` to replace the single column with the new set of dummy columns. Bear in mind that if there are two or more dummy columns, adding columns will change subsequent column numbers; use apop_name_find to find columns instead of giving an explicit column number.

- If `.append='i'` and you asked for a text column, I will append to the end of the table, which is equivalent to `append='e'`.

d	The data set with the column to be dummified (No default.)
col	The column number to be transformed; -1==vector (default = 0)
type	'd'==data column, 't'==text column. (default = 't')
keep_first	If `'n'`, return a matrix where each row has a one in the (column specified *minus one*). That is, the zeroth category is dropped, the first category has an entry in column zero, et cetera. If you don't know why this is useful, then this is what you need. If you know what you're doing and need something special, set this to `'y'` and the first category won't be dropped. (default = `'n'`)
append	If `'e'` or `'y'`, append the dummy grid to the end of the original data matrix. If `'i'`, insert in place, immediately after the original data column. (default = `'n'`)
remove	If `'y'`, remove the original data or text column. (default = `'n'`)

Returns

An apop_data set whose `matrix` element is the one-zero matrix of dummies. If you used `.append`, then this is the main matrix. Also, I add a page named `"\<categories for your_var\>"` giving a reference table of names and column numbers (where `your_var` is the appropriate column heading).

out->error=='a'	allocation error
out->error=='d'	dimension error

- Use apop_data_get_factor_names to get the list of category names.
- NaNs (if any) appear at the end of the sort order.
- See Generating factors for further discussion.
- See the documentation for apop_logit for a sample linear model using this function.
- This function uses the Designated initializers syntax for inputs.

See also

apop_data_to_factors

8.2.2.38 **apop_data*** apop_data_to_factors (**apop_data** * data, char intype, int incol, int outcol)

Convert a column of text or numbers into a column of numeric factors, which you can use for a multinomial probit/logit, for example.

If you don't run this on your data first, apop_probit and apop_logit default to running it on the vector or (if no vector) zeroth column of the matrix of the input apop_data set, because those models need a list of the unique values of the dependent variable.

data	The data set to be modified in place. (No default. If `NULL`, returns `NULL` and a warning)
intype	If `'t'`, then `incol` refers to text, if `'d'`, refers to the vector or matrix. (default = `'t'`)
incol	The column in the text that will be converted. -1 is the vector. (default = 0)
outcol	The column in the data set where the numeric factors will be written (-1 means the vector). (default = 0)

For example:

```
1 apop_data *d  = apop_query_to_mixed_data("mmt", "select 0, year, color from data");
2 apop_data_to_factors(d);
```

Notice that the query pulled a column of zeros for the sake of saving room for the factors. It reads column zero of the text, and writes it to column zero of the matrix.

Another example:

```
1 apop_data *d  = apop_query_to_data("mmt", "select type, year from data");
2 apop_data_to_factors(d, .intype='d', .incol=0, .outcol=0);
```

Here, the `type` column is converted to sequential integer factors and those factors overwrite the original data. Since a reference table is added as a second page of the apop_data set, you can recover the original values as needed.

Returns

A table of the factors used in the code. This is an `apop_data` set with only one column of text. Also, I add a page named `"<categories for your_var>"` giving a reference table of names and column numbers (where `your_var` is the appropriate column heading) use apop_data_get_factor_names to retrieve that table.

out->error=='a'	allocation error.
out->error=='d'	dimension error.

- If the vector or matrix you wanted to write to is `NULL`, I will allocate it for you.
- See Generating factors for further discussion.
- See the documentation for apop_logit for a sample linear model using this function.
- This function uses the Designated initializers syntax for inputs.

See also

apop_data_to_dummies

8.2.2.39 **apop_data*** apop_data_transpose (**apop_data** * in, char transpose_text, char inplace)

Transpose the matrix and text elements of the input data set, including the row/column names.

The vector and weights elements of the input data set are completely ignored (but see also apop_vector_to_matrix, which can convert a vector to a 1 X N matrix.) If copying, these other elements won't be present; if `.inplace='y'`, it is up to you to handle these not-transposed elements correctly.

in	The input apop_data set. If `NULL`, I return `NULL`. (default: `NULL`)
transpose_text	If `'y'`, then also transpose the text element. (default: `'y'`)
inplace	If `'y'`, transpose the input in place; if `'n'`, produce a transposed copy, leaving the original untouched. Due to how `gsl_matrix_transpose_memcpy` works, a copy will still be made, then copied to the original location. (default: `'y'`)

Returns

If `inplace=='n'`, a newly alloced apop_data set, with the appropriately transposed matrix and/or text. The vector and weights elements will be `NULL`. If `transpose_text='n'`, then the text element of the output set will also be `NULL`.
if `inplace=='y'`, a pointer to the original data set, with matrix and (if `transpose_text='y'`, text) transposed and vector and weights left in place untouched.

- Row names are written to column names of the output matrix, text, or both (whichever is not empty in the input).
- If only the matrix or only the text have names, then the one set of names is written to the row names of the output.
- If both matrix column names and text column names are present, text column names are lost.
- if you have a `gsl_matrix` with no names or text, you may prefer to use `gsl_matrix_transpose_memcpy`.
- This function uses the Designated initializers syntax for inputs.

8.2.2.40 void apop_data_unpack (const gsl_vector * in, **apop_data** * d, char use_info_pages)

This is the complement to apop_data_pack, qv. It writes the `gsl_vector` produced by that function back to the apop_data set you provide. It overwrites the data in the vector and matrix elements and, if present, the `weights` (and that's it, so names or text are as before).

in	A `gsl_vector` of the form produced by apop_data_pack. No default; must not be `NULL`.
d	That data set to be filled. Must be allocated to the correct size. No default; must not be `NULL`.
use_info_pages	Pages in XML-style brackets, such as `<Covariance>` will be ignored unless you set `.use_info_pages='y'`. Be sure that this is set to the same thing when you both pack and unpack. (Default: `'n'`).

- If I get to the end of the first page of the `apop_data` set and have more entries in the vector to unpack, and the data to fill has a `more` element, then I will continue into subsequent pages.
- This function uses the Designated initializers syntax for inputs.

8.2.2.41 int apop_db_close (char vacuum)

Closes the database on disk. If you opened the database with `apop_db_open(NULL)`, then this is basically optional.

vacuum	'v': vacuum—do clean-up to minimize the size of the database on disk. 'q': Don't bother; just close the database. (default = 'q')

Returns

0 on OK, nonzero on error.

- This function uses the Designated initializers syntax for inputs.

8.2.2.42 int apop_db_open (char const * filename)

If you want to use a database on the hard drive instead of memory, then call this once and only once before using any other database utilities.

With SQLite, if you want a disposable database which you won't use after the program ends, don't bother with this function.

The trade-offs between an on-disk database and an in-memory db are as one would expect: memory is faster, but the database is destroyed when the program exits.

MySQL users: either set the environment variable APOP_DB_ENGINE=mysql or set `apop_opts.db_engine = 'm'`.

The Apophenia package assumes you are only using a single database at a time. You can use the SQL `attach` function to load other databases, or see `this blog post` for further suggestions and sample code.

When you are done doing your database manipulations, call apop_db_close if writing to disk.

filename	The name of a file on the hard drive on which to store the database. If `NULL`, then the database will be kept in memory (in which case, the other database functions will call this function for you and you don't need to bother).

- See About SQL, the syntax for querying databases for mroe notes on using databases.

Returns

0: everything OK
1: database did not open.

8.2.2.43 **apop_data*** apop_db_to_crosstab (char const * tabname, char const * row, char const * col, char const * data, char is_aggregate)

Give the name of a table in the database, and optional names of three of its columns: the x-dimension, the y-dimension, and the data. The output is a 2D matrix with rows indexed by 'row' and cols by 'col' and the cells filled with the entry in the 'data' column.

tabname	The database table I'm querying. Anything that will work inside a `from` clause is OK, such as a subquery in parens. (no default; must not be `NULL`)
row	The column of the data set that will indicate the rows of the output crosstab (no default; must not be `NULL`)
col	The column of the data set that will indicate the columns of the output crosstab (no default; must not be `NULL`)
data	The column of the data set holding the data for the cells of the crosstab (default: `count(*)`)
is_aggregate	Set to `'y'` if the `data` is a function like `count(*)` or `sum(col)`. That is, set to `'y'` if querying this would require a `group by` clause. (default: if I find an end-paren in `datacol`, `'y'`; else `'n'`.)

- If the query to get data to fill the table (select row, col, data from tabname) returns an empty data set, then I will return a `NULL` data set and if `apop_opts.verbosity >= 1` print a warning.

out->error='n'	Name not found error.
out->error='q'	Query returned an empty table (which might mean that it just failed).

- The simplest use is to get a tally of how often (r1, r2) appears in the data via `apop_db_to_crosstab("datatab", "r1", "r2")`.
- If you want a 1-D crosstab, omit the other dimension. Or omit both to get a grand tally of your statistic for the entire table.
- There is a commnad-line tool, `apop_db_to_crosstab` that calls this function.
- This function uses the Designated initializers syntax for inputs.

8.2.2.44 double apop_det_and_inv (const gsl_matrix * in, gsl_matrix ** out, int calc_det, int calc_inv)

Calculate the determinant of a matrix, its inverse, or both, via LU decomposition. The `in` matrix is not destroyed in the process.

See also

apop_matrix_determinant, apop_matrix_inverse

in	The matrix to be inverted/determined.
out	If you want an inverse, this is where to place the matrix to be filled with the inverse. Will be allocated by the function.
calc_det	0: Do not calculate the determinant. 1: Do.
calc_inv	0: Do not calculate the inverse. 1: Do.

Returns

If `calc_det == 1`, then return the determinant. Otherwise, just returns zero. If `calc_inv!=0`, then `*out` is pointed to the matrix inverse. In case of difficulty, I will set `*out=NULL` and return `NaN`.

8.2.2.45 **apop_data*** apop_dot (const **apop_data** * d1, const **apop_data** * d2, char form1, char form2)

A convenience function for dot products, which requires less prep and typing than the `gsl_cblas_dgexx` functions.

It makes use of the semi-overloading of the apop_data structure. `d1` may be a vector or a matrix, and the same for `d2`, so this function can do vector dot matrix, matrix dot matrix, and so on. If `d1` includes both a vector and a matrix, then later parameters will indicate which to use.

d1	the left part of $d1 \cdot d2$
d2	the right part of $d1 \cdot d2$
form1	't' or 'p': transpose or prime `d1->matrix`, or, if `d1->matrix` is `NULL`, read `d1->vector` as a row vector. 'n' or 0: use matrix if present; no transpose. (the default) 'v': ignore the matrix and use the vector.
form2	As above, with `d2`.

Returns

an apop_data set. If two matrices come in, the vector element is `NULL` and the matrix has the dot product; if either or both are vectors, the vector has the output and the matrix is `NULL`.

out->error='a'	Allocation error.
out->error='d'	dimension-matching error.
out->error='m'	GSL math error.
NULL	If you ask me to take the dot product of NULL, I return NULL.

- Some systems auto-transpose non-conforming matrices. You input a 3×5 and a 3×5 matrix, and the system assumes that you meant to transpose the second, producing a $(3 \times 5) \cdot (5 \times 3) \to (3 \times 3)$ output. Apophenia does not do this. First, it's ambiguous whether the output should be 3×3 or 5×5. Second, your next run might have three observations, and two 3×3 matrices don't require transposition; auto-transposition thus creates situations where bugs can pop up on only some iterations of a loop.

- For a vector $\cdot$ a matrix, the vector is always treated as a row vector, meaning that a (3×1) dot a (3×4) matrix is correct, and produces a (1×4) vector. For a matrix $\cdot$ a vector, the vector is always treated as a column vector. Requests for transposing the vector are ignored in both cases.

- As a corrollary to the above rule, a vector dot a vector always produces a scalar, which will be put in the zeroth element of the output vector; see the example.

- If you want to multiply an $N \times 1$ vector $\cdot$ a $1 \times N$ vector to produce an $N \times N$ matrix, then use apop_vector_to_matrix to turn your vectors into matrices; see the example.

- A note for readers of *Modeling with Data*: the awkward instructions on using this function on p 130 are now obsolete, thanks to the designated initializer syntax for function calls. Notably, in the case where `d1` is a vector and `d2` a matrix, then `apop_dot(d1,d2,'t')` won't work, because `'t'` now refers to `d1`. Instead use `apop_dot(d1,d2,.form2='t')` or `apop_dot(d1,d2,0, 't')`

- This function uses the Designated initializers syntax for inputs.

Sample code:

```
/* A demonstration of dot products and various useful
   transformations among types. */

#include <apop.h>

double eps=1e-3;//slow to converge series-->large tolerance.
#define Diff(L, R) Apop_assert(fabs((L)-(R))<(eps)), "%g is too different from %g (abitrary limit=%g).",
       (double)(L), (double)(R), eps);

int main(){
    int len = 3000;
    gsl_vector *v = gsl_vector_alloc(len);
    for (double i=0; i< len; i++) gsl_vector_set(v, i, 1./(i+1));
    double square;
    gsl_blas_ddot(v, v, &square);
    printf("1 + (1/2)^2 + (1/3)^2 + ...= %g\n", square);

    double pi_over_six = gsl_pow_2(M_PI)/6.;
    Diff(square, pi_over_six);

    /* Now using apop_dot, in a few forms.
       First, vector-as-data dot itself.
       If one of the inputs is a vector,
       apop_dot puts the output in a vector-as-data:*/
    apop_data *v_as_data = &(apop_data){.vector=v};
    apop_data *vdotv = apop_dot(v_as_data, v_as_data);
    Diff(gsl_vector_get(vdotv->vector, 0), pi_over_six);

    /* Wrap matrix in an apop_data set. */
    gsl_matrix *v_as_matrix = apop_vector_to_matrix(v);
    apop_data dm = (apop_data){.matrix=v_as_matrix};

    // (1 X len) vector dot (len X 1) matrix --- produce a scalar (one item vector).
    apop_data *mdotv = apop_dot(v_as_data, &dm);
    double scalarval = apop_data_get(mdotv);
    Diff(scalarval, pi_over_six);

    //(len X 1) dot (len X 1) --- bad dimensions.
    apop_opts.verbose=-1; //don't print an error.
    apop_data *mdotv2 = apop_dot(&dm, v_as_data);
    apop_opts.verbose=0; //back to safety.
    assert(mdotv2->error);

    // If we want (len X 1) dot (1 X len) --> (len X len),
    // use apop_vector_to_matrix.
    apop_data dmr = (apop_data){.matrix=apop_vector_to_matrix(v, .
      row_col='r')};
    apop_data *product_matrix = apop_dot(&dm, &dmr);
    //The trace is the sum of squares:
    gsl_vector_view trace = gsl_matrix_diagonal(product_matrix->matrix);
    double tracesum = apop_sum(&trace.vector);
    Diff(tracesum, pi_over_six);

    apop_data_free(product_matrix);
    gsl_matrix_free(dmr.matrix);
}
```

8.2.2.46 int apop_draw (double * out, gsl_rng * r, **apop_model** * m)

Draw from a model.

out	An already-allocated array of `doubles` to be filled by the draw method. It must have size `m->dsize`.
r	A `gsl_rng`, probably allocated via apop_rng_alloc. Optional; if `NULL`, then I will call apop_rng_get_thread for an RNG.
m	The model from which to make draws.

- If the model has its own `draw` method, then this function will call it.
- Else, if the model is univariate, use apop_arms_draw to generate random draws.
- Else, if the model is multivariate, use apop_model_metropolis to generate random draws.
- This makes a single draw of the given size. See apop_model_draws to fill a matrix with draws.

Returns

Zero on success; nozero on failure. `out[0]` is probably `NAN` on failure.

8.2.2.47 **apop_model**∗ apop_estimate (**apop_data** ∗ d, **apop_model** ∗ m)

Estimate the parameters of a model given data.

This function copies the input model, preps it (see apop_prep), and calls `m.estimate(d, m)` (which users are encouraged to never call directly). If your model has no `estimate` method, then call `apop_maximum_likelihood(d, m)`, with the default MLE settings.

d	The data
m	The model

Returns

A pointer to an output model, which typically matches the input model but has its `parameters` element filled in.

8.2.2.48 **apop_data**∗ apop_estimate_coefficient_of_determination (**apop_model** ∗ m)

Also known as R^2. Let Y be the dependent variable, ϵ the residual, n the number of data points, and k the number of independent vars (including the constant). Returns an apop_data set with the following entries (in the vector element):

- $SST \equiv \sum (Y_i - \bar{Y})^2$
- $SSE \equiv \sum \epsilon^2$
- $R^2 \equiv 1 - \frac{SSE}{SST}$
- $R^2_{adj} \equiv R^2 - \frac{(k-1)}{(n-k-1)}(1 - R^2)$

Internally allocates (and frees) a vector the size of your data set.

Returns

A 5×1 apop_data table with the following fields:

- "R squared"
- "R squared adj"
- "SSE"
- "SST"
- "SSR"

If the output is in `sss`, use `apop_data_get(sss, .rowname="SSE")` to get the SSE, and so on for the other items.

m	A model. I use the pointer to the data set used for estimation and the info page named `"<Predicted>"`. The Predicted page should include observed, expected, and residual columns, which I use to generate the sums of squared errors and residuals, et cetera. All generalized linear models produce a page with this name and of this form, as do a host of other models. Nothing keeps you from finding the R^2 of, say, a kernel smooth; it is up to you to determine whether such a thing is appropriate to your given models and situation.

- `apop_estimate(yourdata, apop_ols)` does this automatically
- If I don't find a `"<Predicted>"` page, print an error (iff `apop_opts.verbose >=0`) and return `NULL`.
- The number of observations equals the number of rows in the Predicted page
- The number of independent variables, needed only for the adjusted R^2, is from the number of columns in the main data set's matrix (i.e. the first page; i.e. the set of parameters if this is the `parameters` output from a model estimation).
- If your data (first page again) has a `weights` vector, I will find weighted SSE, SST, and SSR (and calculate the R^2s using those values).

8.2.2.49 **apop_model*** apop_estimate_restart (**apop_model** * e, **apop_model** * copy, char * starting_pt, double boundary)

Maximum likelihod searches are not guaranteed to find a global optimum, and it can be difficult to tune a search such that it covers a wide space, but also accurately hones in on the optimum. In both cases, one could restart the search using a different starting point or different parameters.

The simplest use of this function is to restart a model at the latest parameter estimates.

```
1 apop_model *m = apop_estimate(data, model_using_an_MLE_search);
2 for (int i=0; i< 10; i++)
3     m = apop_estimate_restart(m);
4 apop_data_show(m);
```

By adding a line to reduce the tolerance each round [e.g., `Apop_settings_set(m, apop_mle, tolerance, pow(10,-i))`], you can start broad and hone in on a precise optimum.

You may have a new estimation method, such as first doing a coarse simulated annealing search, then a fine conjugate gradient search. When reading this example, recall that the form for adding a new settings group differs from the form for modifying existing settings:

```
1 Apop_model_add_settings(your_base_model, apop_mle, .method=APOP_SIMAN);
2 apop_model *m = apop_estimate(data, your_base_model);
3 Apop_settings_set(m, apop_mle, method, APOP_CG_PR);
4 m = apop_estimate_restart(m);
5 apop_data_show(m);
```

Only one estimate is returned, either the one you sent in or a new one. The loser (which may be the one you sent in) is freed, to prevent memory leaks.

e	An apop_model that is the output from a prior MLE estimation. (No default, must not be NULL.)
copy	Another not-yet-parametrized model that will be re-estimated with (1) the same data and (2) a starting_pt as per the next setting (probably to the parameters of e). If this is NULL, then copy e. (Default = NULL)
starting_pt	"ep"=last estimate of the first model (i.e., its current parameter estimates) "es"= starting point originally used by the first model "np"=current parameters of the new (second) model "ns"=starting point specified by the new model's MLE settings. (default = "ep")
boundary	I test whether the starting point you give me has magintude greater than this bound, so I can warn you if there's divergence in your sequence of re-estimations. (default: 1e8)

Returns

If the new estimated parameters include any NaNs/Infs, then the old estimate is returned (even if the old estimate included NaNs/Infs). Otherwise, the estimate with the largest log likelihood is returned.

- This function uses the Designated initializers syntax for inputs.

8.2.2.50 **apop_data*** apop_f_test (**apop_model** * est, **apop_data** * contrast)

Runs an F-test specified by q and c. See the chapter on hypothesis testing in Modeling With Data, p 309, which will tell you that:

$$\frac{N-K}{q}\frac{(\mathbf{Q}'\hat{\beta}-\mathbf{c})'[\mathbf{Q}'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{Q}]^{-1}(\mathbf{Q}'\hat{\beta}-\mathbf{c})}{\mathbf{u}'\mathbf{u}} \sim F_{q,N-K},$$

and that's what this function is based on.

est	An apop_model that you have already calculated. (No default)
contrast	An apop_data set whose matrix represents **Q** and whose vector represents **c**. Each row represents a hypothesis. (Defaults: if matrix is NULL, it is set to the identity matrix with the top row missing. If the vector is NULL, it is set to a zero matrix of length equal to the height of the contrast matrix. Thus, if the entire apop_data set is NULL or omitted, we are testing the hypothesis that all but β_1 are zero.)

Returns

An apop_data set with a few variants on the confidence with which we can reject the joint hypothesis.

out->error='a'	Allocation error.
out->error='d'	dimension-matching error.
out->error='i'	matrix inversion error.
out->error='m'	GSL math error.

- There are two approaches to an F-test: the ANOVA approach, which is typically built around the claim that all effects but the mean are zero; and the more general regression form, which allows for any set of linear claims about the data. If you send a NULL contrast set, I will generate the set of linear contrasts that are equivalent to the ANOVA-type approach. This is why the top row of the default **Q** matrix is missing: there is no hypothesis test about the coefficient for the constant term. See the example below.

- This function uses the Designated initializers syntax for inputs.

```
#include <apop.h>

#define Diff(L, R, eps) {double left=(L), right=(R); Apop_stopif(isnan(left-right) ||
       fabs((left)-(right))>(eps), abort(), 0, "%g is too different from %g (abitrary limit=%g).", (double)(left)
       eps);}

void test_f(apop_model *est){
    apop_data *rsq  = apop_estimate_coefficient_of_determination
      (est);
    apop_data *constr= apop_data_calloc(est->parameters->vector->size-1, est->
      parameters->vector->size);
    int i;
    for (i=1; i< est->parameters->vector->size; i++)
        apop_data_set(constr, i-1, i, 1);
    apop_data *ftab = apop_F_test(est, constr);
    apop_data *ftab2 = apop_F_test(est, NULL);
    //apop_data_show(ftab);
    //apop_data_show(ftab2);
    double n = est->data->matrix->size1;
    double K = est->parameters->vector->size-1;
    double r = apop_data_get(rsq, .rowname="R squared");
    double f = apop_data_get(ftab, .rowname="F statistic");
    double f2 = apop_data_get(ftab2, .rowname="F statistic");
    Diff (f , r*(n-K)/((1-r)*K) , 1e-3);
    Diff (f2 , r*(n-K)/((1-r)*K) , 1e-3);
}

int main(){
    apop_data *d = apop_text_to_data("test_data2");
    apop_model *an_ols_model = apop_model_copy(apop_ols);
    Apop_model_add_group(an_ols_model, apop_lm, .want_expected_value= 1);
    apop_model *e  = apop_estimate(d, an_ols_model);
    test_f(e);
}
```

8.2.2.51 long double apop_generalized_harmonic (int N, double s)

Calculate $\sum_{n=1}^{N} \frac{1}{n^s}$

- There are no doubt efficient shortcuts do doing this, but I use brute force. [Though Knuth's Art of Programming v1 doesn't offer anything, which is strong indication of nonexistence.] To speed things along, I save the results so that they can just be looked up should you request the same calculation.

- If `N` is zero or negative, return NaN. Notify the user if `apop_opts.verbosity >=0`

For example:

```
#include <apop.h>

int main(){
    double out = apop_generalized_harmonic(270, 0.0);
     assert (out == 270);
     out  = apop_generalized_harmonic(370, -1.0);
     assert (out == 370*371/2);
     out  = apop_generalized_harmonic(12, -1.0);
     assert (out == 12*13/2);
}
```

8.2.2.52 apop_data* apop_histograms_test_goodness_of_fit (apop_model * observed, apop_model * expected)

Test the goodness-of-fit between two apop_pmf models.

Let o_i be the ith observed bin and e_i the expected value of that bin; then under typical assumptions, $\sum_i^N (o_i - e_i)^2/e_i \sim \chi^2_{N-1}$.

If you send two histograms, I assume that the histograms are synced: for PMFs, you've used apop_data_to_bins to generate two histograms using the same binspec, or you've used apop_data_pmf_compress to guarantee that each observation value appears exactly once in each data set.

In any case, all values in the `observed` set must appear in the `expected` set with nonzero weight; otherwise this will return a χ^2 statistic of `GSL_POSINF`, indicating that it is impossible for the `observed` data to have been drawn from the `expected` distribution.

- If an observation row has weight zero, I skip it. if `apop_opts.verbose >=1` I will show a warning.

8.2.2.53 **apop_data*** apop_jackknife_cov (**apop_data** * in, **apop_model** * model)

Give me a data set and a model, and I'll give you the jackknifed covariance matrix of the model parameters.

The basic algorithm for the jackknife (glossing over the details): create a sequence of data sets, each with exactly one observation removed, and then produce a new set of parameter estimates using that slightly shortened data set. Then, find the covariance matrix of the derived parameters.

- Jackknife or bootstrap? As a broad rule of thumb, the jackknife works best on models that are closer to linear. The worse a linear approximation does (at the given data), the worse the jackknife approximates the variance.

in	The data set. An apop_data set where each row is a single data point.
model	An apop_model, that will be used internally by apop_estimate.

out->error=='n'	`NULL` input data.

Returns

An apop_data set whose matrix element is the estimated covariance matrix of the parameters.

See also

For example:

```
#include <apop.h>

int main(){
    int draw_ct = 1000;
    apop_model *m = apop_model_set_parameters(apop_normal, 1, 3);
    double sigma = apop_data_get(m->parameters, 1);
    apop_data *d = apop_model_draws(m, draw_ct);
    apop_data *out = apop_jackknife_cov(d, m);
    double error = fabs(apop_data_get(out, 0,0)-gsl_pow_2(sigma)/draw_ct) //var(mu)
                + fabs(apop_data_get(out, 1,1)-gsl_pow_2(sigma)/(2*draw_ct))//var(sigma)
                +fabs(apop_data_get(out, 0,1)) +fabs(apop_data_get(out, 1,0));//
      cov(mu,sigma); should be 0.
    apop_data_free(d);
    apop_data_free(out);
    assert(error < 1e-2);//Not very accurate.
}
```

8.2.2.54 long double apop_kl_divergence (**apop_model** * from, **apop_model** * to, int draw_ct, gsl_rng * rng)

Kullback-Leibler divergence.

This measure of the divergence of one distribution from another has the form $D(p,q) = \sum_i \ln(p_i/q_i)p_i$. Notice that it is not a distance, because there is an asymmetry between p and q, so one can expect that $D(p,q) \neq D(q,p)$.

from	the p in the above formula. (No default; must not be `NULL`)
to	the q in the above formula. (No default; must not be `NULL`)
draw_ct	If I do the calculation via random draws, how many? (Default = 1e5)
rng	A `gsl_rng`. If `NULL` or number of threads is greater than 1, I'll take care of the RNG; see apop_rng_get_thread. (Default = `NULL`)

This function can take empirical histogram-type models (apop_pmf) or continuous models like apop_loess or apop_normal.

If there is a PMF (I'll try `from` first, under the presumption that you are measuring the divergence of a fitted model from an observed data distribution), then I'll step through it for the points in the summation.

- If you have two empirical distributions in the form of apop_pmf, they must be synced: if $p_i > 0$ but $q_i = 0$, then the function returns `GSL_NEGINF`. If `apop_opts.verbose >=1` I print a message as well.

If neither distribution is a PMF, then I'll take `draw_ct` random draws from `from` and evaluate at those points.

- Set `apop_opts.verbose = 3` for observation-by-observation info.

- This function uses the Designated initializers syntax for inputs.

8.2.2.55 long double apop_linear_constraint (gsl_vector * beta, **apop_data** * constraint, double margin)

This is designed to be called from within the constraint method of your apop_model. Just write the constraint vector+matrix and this will do the rest. See Setting Constraints for detailed discussion.

beta	The proposed vector about to be tested. No default, must not be `NULL`.

constraint	A vector/matrix pair [v | m1 m2 ... mn] where each row is interpreted as a less-than inequality: $v < m1x1 + m2x2 + ... + mnxn$. For example, say your constraints are $3 < 2x + 4y - 7z$ and y is positive, i.e. $0 < y$. Allocate and fill the matrix representing these two constraints via:
	`1 apop_data *constr = apop_data_falloc((2,2,3), 3, 2, 4, 7,`
	`2 0, 0, 1, 0);`
	. Default: each elements is greater than zero. For three parameters this would be equivalent to setting
	`1 apop_data *constr = apop_data_falloc((3,3,3), 0, 1, 0, 0,`
	`2 0, 0, 1, 0,`
	`3 0, 0, 0, 1);`

margin	If zero, then this is a >= constraint, otherwise I will return a point this amount within the borders. You could try `GSL_DBL_EPSILON`, which is the smallest value a `double` can hold, or something like 1e-3. Default = 0.

Returns

The penalty: the distance between beta and the closest point that meets the constraints. If the constraint is met, the penalty is zero. If the constraint is not met, this `beta` is shifted by `margin` (Euclidean distance) to meet the constraints.

- If your apop_data has more structure than a vector, try apop_data_pack to pack it into a vector. This is what apop_maximum_likelihood does.
- The function doesn't check for odd cases like coplanar constraints.
- This function uses the Designated initializers syntax for inputs.

8.2.2.56 double apop_log_likelihood (**apop_data** * d, **apop_model** * m)

Find the log likelihood of a data/parametrized model pair.

d	The data
m	The parametrized model, which must have either a `log_likelihood` or a `p` method.

8.2.2.57 **apop_data*** apop_map (**apop_data** * in, apop_fn_d * fn_d, apop_fn_v * fn_v, apop_fn_r * fn_r, apop_fn_dp * fn_dp, apop_fn_vp * fn_vp, apop_fn_rp * fn_rp, apop_fn_dpi * fn_dpi, apop_fn_vpi * fn_vpi, apop_fn_rpi * fn_rpi, apop_fn_di * fn_di, apop_fn_vi * fn_vi, apop_fn_ri * fn_ri, void * param, int inplace, char part, int all_pages)

Apply a function to every element of a data set, matrix or vector; or, apply a vector-taking function to every row or column of a matrix.

Your function could take any combination of a `gsl_vector`, a `double`, an apop_data, a parameter set, and the position of the element in the vector or matrix. As such, the function takes twelve function inputs, one for each combination of vector/matrix, params/no params, index/no index. Fortunately, because this function uses the Designated initializers syntax for inputs, you will specify only one.

For example, here is a function that will cut off each element of the input data to between $(-1, +1)$. It takes in a lone `double` and a parameter in a `void*`, so it gets sent to apop_map via `.fn_dp=cutoff`.

```
1 double cutoff(double in, void *limit_in){
2     double *limit = limit_in;
3     return GSL_MAX(-*limit, GSL_MIN(*limit, in));
4 }
5
6 double param = 1;
7 apop_map(your_data, .fn_dp=cutoff, .param=&param, .inplace='y');
```

fn_v	A function of the form `double your_fn(gsl_vector *in)`
fn_d	A function of the form `double your_fn(double in)`
fn_r	A function of the form `double your_fn(apop_data *in)`
fn_vp	A function of the form `double your_fn(gsl_vector *in, void *param)`
fn_dp	A function of the form `double your_fn(double in, void *param)`
fn_rp	A function of the form `double your_fn(apop_data *in, void *param)`
fn_vpi	A function of the form `double your_fn(gsl_vector *in, void *param, int index)`

fn_dpi	A function of the form `double your_fn(double in, void *param, int index)`
fn_rpi	A function of the form `double your_fn(apop_data *in, void *param, int index)`
fn_vi	A function of the form `double your_fn(gsl_vector *in, int index)`
fn_di	A function of the form `double your_fn(double in, int index)`
fn_ri	A function of the form `double your_fn(apop_data *in, int index)`
in	The input data set. If `NULL`, I'll return `NULL` immediately.
param	A pointer to the parameters to be passed to those function forms taking a `*param`.
part	Which part of the apop_data struct should I use? 'v'==Just the vector 'm'==Every element of the matrix, in turn 'a'==Both 'v' and 'm' 'r'==Apply a function `gsl_vector` → `double` to each row of the matrix 'c'==Apply a function `gsl_vector` → `double` to each column of the matrix Default is 'a', but notice that I'll ignore a `NULL` vector or matrix, so if your data set has only a vector or only a matrix, that's what I'll use.
all_pages	If `'y'`, then follow the `more` pointer to subsequent pages. If `'n'`, handle only the first page of data. Default: `'n'`.
inplace	If 'n' (the default), generate a new apop_data set for output, which will contain the mapped values (and the names from the original set). If 'y', modify in place. The `double` → `double` versions, `'v'`, `'m'`, and `'a'`, write to exactly the same location as before. The `gsl_vector` → `double` versions, `'r'`, and `'c'`, will write to the vector. Be careful: if you are writing in place and there is already a vector there, then the original vector is lost. If 'v' (as in void), return `NULL`. (Default = 'n')

out->error='p'	missing or mismatched parts error, such as `NULL` matrix when you sent a function acting on the matrix element.

- The function forms with `r` in them, like `fn_ri`, are row-by-row. I'll use Apop_r to get each row in turn, and send it to the function. The first implication is that your function should be expecting a apop_data set with exactly one row in it. The second is that `part` is ignored: it only makes sense to go row-by-row.

- For these `r` functions, if you set `inplace='y'`, then you will be modifying your input data set, row by row; if you set `inplace='n'`, then I will return an apop_data set whose `vector` element is as long as your data set (i.e., as long as the longest of your text, vector, or matrix parts).

- If you set `omp_set_num_threads(n)` using $n > 1$, split the data set into as many chunks as you specify and process them simultaneously. You need to watch out for the usual hang-ups about multi-threaded programming, but if your data is iid, and each row's processing is independent of the others, you should have no problems. Bear in mind that generating threads takes some small overhead, so simple cases like adding a few hundred numbers will actually be slower when threading.

- See Map/apply for many more examples and notes.

See also

apop_map_sum

8.2.2.58 double apop_map_sum (**apop_data** * in, apop_fn_d * fn_d, apop_fn_v * fn_v, apop_fn_r * fn_r, apop_fn_dp * fn_dp, apop_fn_vp * fn_vp, apop_fn_rp * fn_rp, apop_fn_dpi * fn_dpi, apop_fn_vpi * fn_vpi, apop_fn_rpi * fn_rpi, apop_fn_di * fn_di, apop_fn_vi * fn_vi, apop_fn_ri * fn_ri, void * param, char part, int all_pages)

A function that effectively calls apop_map and returns the sum of the resulting elements. Thus, this function returns a `double`. See the apop_map page for details of the inputs, which are the same here, except that `inplace` doesn't make sense—this function will always just add up the input function outputs.

- I don't copy the input data to send to your input function. Therefore, if your function modifies its inputs as a side-effect, your data set will be modified as this function runs.
- The sum of zero elements is zero, so that is what is returned if the input apop_data set is `NULL`. If `apop_opts.verbose >= 2` print a warning.
- See Map/apply for many more examples and notes.
- This function uses the Designated initializers syntax for inputs.

8.2.2.59 void apop_matrix_apply (gsl_matrix * m, void(*)(gsl_vector *) fn)

Apply a function to every row of a matrix. The function that you input takes in a `gsl_vector` and returns nothing. `apop_matrix_apply` will produce a vector view of each row, and send each row to your function.

m	The matrix
fn	A function of the form `void fn(gsl_vector* in)` which may modify the data at the `in` pointer in place.

- If the matrix is `NULL`, this is a no-op and returns immediately.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.60 void apop_matrix_apply_all (gsl_matrix * in, void(*)(double *) fn)

Applies a function to every element in a matrix (as opposed to every row)

in	The matrix whose elements will be inputs to the function
fn	A function with a form like `void f(double *in)` which may modify the data at the `in` pointer in place.

- If the matrix is `NULL`, this is a no-op and returns immediately.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.61 gsl_matrix* apop_matrix_copy (const gsl_matrix * in)

Copy one `gsl_matrix` to another. That is, all data are duplicated. Unlike `gsl_matrix_memcpy`, this function allocates and returns the destination, so you can use it like this:

```
1 gsl_matrix *a_copy = apop_matrix_copy(original);
```

in	the input data

Returns

A structure that this function will allocate and fill. If `gsl_matrix_alloc` fails, returns `NULL`.

8.2.2.62 double apop_matrix_determinant (const gsl_matrix * in)

Find the determinant of a matrix. The `in` matrix is not destroyed in the process.

See also apop_matrix_inverse , or apop_det_and_inv to do both at once.

in	The matrix to be determined.

Returns

The determinant.

8.2.2.63 gsl_matrix* apop_matrix_inverse (const gsl_matrix * in)

Inverts a matrix. The `in` matrix is not destroyed in the process. You may want to call apop_matrix_↩ determinant first to check that your input is invertible, or use apop_det_and_inv to do both at once.

in	The matrix to be inverted.

Returns

Its inverse.

8.2.2.64 int apop_matrix_is_positive_semidefinite (gsl_matrix * m, char semi)

Test whether the input matrix is positive semidefinite (PSD).

A covariance matrix will always be PSD, so this function can tell you whether your matrix is a valid covariance matrix.

Consider the 1x1 matrix in the upper left of the input, then the 2x2 matrix in the upper left, on up to the full matrix. If the matrix is PSD, then each of these has a positive determinant. This function thus calculates N determinants for an Nx N matrix.

m	The matrix to test. If `NULL`, I will return zero—not PSD.
semi	If anything but `'s'`, check for positive definite, not semidefinite. (default 's')

See also apop_matrix_to_positive_semidefinite, which will change the input to something PSD.

- This function uses the Designated initializers syntax for inputs.

8.2.2.65 gsl_vector* apop_matrix_map (const gsl_matrix * m, double(*)(gsl_vector *) fn)

Map a function onto every row of a matrix. The function that you input takes in a `gsl_vector` and returns a `double`. This function will produce a sequence of vector views of each row of the input matrix, and send each to your function. It will output a `gsl_vector` holding your function's output for each row.

m	The matrix
fn	A function of the form `double fn(gsl_vector* in)`

Returns

A `gsl_vector` with the corresponding value for each row.

- If you input a `NULL` matrix, I return `NULL`.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.66 gsl_matrix* apop_matrix_map_all (const gsl_matrix * in, double(*)(double) fn)

Maps a function to every element in a matrix (as opposed to every row).

in	The matrix whose elements will be inputs to the function
fn	A function with a form like `double f(double in).`

Returns

a matrix of the same size as the original, with the function applied.

- If you input a `NULL` matrix, I return `NULL`.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.67 double apop_matrix_map_all_sum (const gsl_matrix * in, double(*)(double) fn)

Like `apop_matrix_map_all`, but returns the sum of the resulting mapped function. For example, `apop_matrix_map_all_sum(v, isnan)` returns the number of elements of `m` that are `NaN`.

- If you input a `NULL` matrix, I return the sum of zero items: zero.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.68 double apop_matrix_map_sum (const gsl_matrix * in, double(*)(gsl_vector *) fn)

Like `apop_matrix_map`, but returns the sum of the resulting mapped vector. For example, let `log_like` be a function that returns the log likelihood of an input vector; then `apop_matrix_map_sum(m, log_like)` returns the total log likelihood of the rows of `m`.

- If you input a `NULL` matrix, I return the sum of zero items: zero.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.69 double apop_matrix_mean (const gsl_matrix * data)

Returns the mean of all elements of a matrix.

data	The matrix to be averaged. If `NULL`, return zero.

Returns

The mean of all cells of the matrix.

8.2.2.70 void apop_matrix_mean_and_var (const gsl_matrix * data, double * mean, double * var)

Returns the mean and population variance of all elements of a matrix.

- If `NULL`, return $\mu = 0, \sigma^2 = NaN$.
- Gives the population variance (sum of squares divided by N). If you want sample variance, multiply the result by $N/(N-1)$:

```
1 double mu, var;
2 apop_data *data= apop_query_to_data("select * from indata");
3 apop_matrix_mean_and_var(data->matrix, &mu, &var);
4 var *= (data->size1*data->size2)/(data->size1*data->size2-1.0);
```

data	the matrix to be averaged.
mean	where to put the mean to be calculated.
var	where to put the variance to be calculated.

8.2.2.71 **apop_data*** apop_matrix_pca (gsl_matrix * data, int const dimensions_we_want)

Principal component analysis: hand in a matrix and (optionally) a number of desired dimensions, and I'll return a data set where each column of the matrix is an eigenvector. The columns are sorted, so column zero has the greatest weight. The vector element of the data set gives the weights.

You may also specify the number of elements your principal component space should have. If this is equal to the rank of the space in which the input data lives, then the sum of weights will be one. If the dimensions desired is less than that (probably so you can prepare a plot), then the weights will be accordingly smaller, giving you an indication of how much variation these dimensions explain.

data	The input matrix. I modify int in place so that each column has mean zero. (No default. If `NULL`, return `NULL` and print a warning iff `apop_opts.verbose >= 1`.)
dimensions_↩ we_want	The singular value decomposition will return this many of the eigenvectors with the largest eigenvalues. (default: the size of the covariance matrix, i.e. `data->size2`)

Returns

Returns an apop_data set whose matrix is the principal component space. Each column of the returned matrix will be another eigenvector; the columns will be ordered by the eigenvalues.

The data set's vector will be the largest eigenvalues, scaled by the total of all eigenvalues (including those that were thrown out). The sum of these returned values will give you the percentage of variance explained by the factor analysis.

out->error=='a'	Allocation error.

8.2.2.72 void apop_matrix_print (const gsl_matrix * data, Output_declares)

Print a `gsl_matrix` to the screen, a file, a pipe, or a database table.

- See apop_prep_output for more on how printing settings are set.
- See also Legible output for more details and examples.
- This function uses the Designated initializers syntax for inputs.

8.2.2.73 gsl_matrix* apop_matrix_realloc (gsl_matrix * m, size_t newheight, size_t newwidth)

This function will resize a `gsl_matrix` to a new height or width.

Data in the matrix will be retained. If the new height or width is smaller than the old, then data in the later rows/columns will be cropped away (in a non–memory-leaking manner). If the new height or width is larger than the old, then new cells will be filled with garbage; it is your responsibility to zero out or otherwise fill new rows/columns before use.

- A large number of `realloc`s can take a noticeable amount of time. You are encouraged to determine the size of your data beforehand and avoid writing `for` loops that reallocate the matrix at every iteration.
- The `gsl_matrix` is a versatile struct that can represent submatrices and other cuts from parent data. Resizing a subset of a parent matrix makes no sense, so return `NULL` and print a warning if asked to resize a view of a matrix.

m	The already-allocated matrix to resize. If you give me `NULL`, this becomes equivalent to `gsl_matrix_alloc`
newheight,newwidth	The height and width you'd like the matrix to be.

Returns

m, now resized

8.2.2.74 gsl_matrix* apop_matrix_stack (gsl_matrix * m1, gsl_matrix const * m2, char posn, char inplace)

Put the first matrix either on top of or to the right of the second matrix. Returns a new matrix, meaning that at the end of this function, until you `gsl_matrix_free()` the original matrices, you will be taking up twice as much memory. Plan accordingly.

m1	the upper/rightmost matrix (default: `NULL`, in which case this copies `m2`)
m2	the second matrix (default: `NULL`, in which case `m1` is returned)
posn	If `'r'`, stack rows on top of other rows. If `'c'` stack columns next to columns. (default: `'r'`)
inplace	If `'y'`, use apop_matrix_realloc to modify `m1` in place; see the caveats on that function. Otherwise, allocate a new matrix, leaving `m1` undisturbed. (default: `'n'`)

Returns

the stacked data, either in a new matrix or a pointer to `m1`.

For example, here is a function to merge four matrices into a single two-part-by-two-part matrix. The original matrices are unchanged.

```
1 gsl_matrix *apop_stack_two_by_two(gsl_matrix *ul, gsl_matrix *ur, gsl_matrix *dl, gsl_matrix *dr){
2   gsl_matrix *output, *t;
3     output = apop_matrix_stack(ul, ur, 'c');
```

```
4     t = apop_matrix_stack(dl, dr, 'c');
5     apop_matrix_stack(output, t, 'r', .inplace='y');
6     gsl_matrix_free(t);
7     return output;
8 }
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.75 long double apop_matrix_sum (const gsl_matrix * m)

Returns the sum of the elements of a matrix. Occasionally convenient.

m	the matrix to be summed.

8.2.2.76 double apop_matrix_to_positive_semidefinite (gsl_matrix * m)

This function takes in a matrix and converts it in place to the 'closest' positive semidefinite matrix.

m	On input, any matrix; on output, a positive semidefinite matrix. If `NULL`, return `NaN` and print an error.

Returns

the distance between the original and new matrices.

- See also the test function apop_matrix_is_positive_semidefinite.
- This function can be used as the core of a model constraint.
- Adapted from the R Matrix package's nearPD, which is Copyright (2007) Jens Oehlschlägel [under the GPL].

8.2.2.77 void apop_maximum_likelihood (**apop_data** * data, **apop_model** * dist)

Find the likelihood-maximizing parameters of a model given data.

- I assume that apop_prep has been called on your model. The easiest way to guarantee this is to use apop_estimate, which calls this function if the input model has no `estimate` method.
- All of the settings are specified by adding a apop_mle_settings struct to your model, so see the many notes there. Notably, the default method is the Fletcher-Reeves conjugate gradient method, and if your model does not have a dlog likelihood function, then a numeric gradient will be calculated via apop_numerical_gradient. Add an apop_mle_settings group to your model to set tuning parameters or select other methods, including the Nelder-Mead simplex, simulated annealing, and root-finding.

data	An apop_data set.
dist	The apop_model object: apop_gamma, apop_probit, apop_zipf, &c. You can add an `apop_mle_settings` struct to it (`Apop_model_add_group(your_model, apop_mle, .verbose=1, .method="PR cg", and_so_on)`).

Returns

None, but the input model is modified to include the parameter estimates, &c.

- There is auxiliary info in the `->info` element of the post-estimation struct. Get elements via, e.g.:

```
apop_model *est = apop_estimate(your_data, apop_probit);


int status = apop_data_get(est->info, .rowname="status");
if (status)
    //trouble
else
    //optimum found
    apop_data_print(est->parameters); //Here are the estimated parameters
```

- During the search for an optimum, ctrl-C (SIGINT) will halt the search, and the function will return whatever parameters the search was on at the time.

8.2.2.78 **apop_model*** apop_ml_impute (**apop_data** * d, **apop_model** * mvn)

Impute the most likely data points to replace NaNs in the data, and insert them into the given data. That is, the data set is modified in place.

How it works: this uses the machinery for apop_model_fix_params. The only difference is that this searches over the data space and takes the parameter space as fixed, while basic fix params model searches parameters and takes data as fixed. So this function just does the necessary data-parameter switching to make that happen.

d	The data set. It comes in with NaNs and leaves entirely filled in.
mvn	A parametrized apop_model from which you expect the data was derived. if `NULL`, then I'll use the Multivariate Normal that best fits the data after listwise deletion.

Returns

An estimated apop_model. Also, the data input will be filled in and ready to use.

8.2.2.79 **apop_model*** apop_model_clear (**apop_data** * data, **apop_model** * model)

Set up the `parameters` and `info` elements of the `apop_model`:

At close, the input model has parameters of the correct size.

- This is the default action for apop_prep, and many models with a custom prep routine call apop_model_clear at the end. Also, apop_estimate calls this function internally, which means that you robably never have to call this function directly.
- If the model has already been prepped, this function should be a no-op.

data	If your params vary with the size of the data set, then the function needs a data set to calibrate against. Otherwise, it's OK to set this to `NULL`.
model	The model whose output elements will be modified.

Returns

A pointer to the same model, should you need it.

outmodel->error=='d'	dimension error.

8.2.2.80 **apop_model*** apop_model_copy (**apop_model** * in)

Outputs a copy of the apop_model input.

in	The model to be copied

Returns

A copy of the original. Includes copies of all settings groups, and the `parameters` (if not `NULL`, copied via apop_data_copy).

- If `in.more_size > 0` I `memcpy` the `more` pointer from the original data set.
- The data set at `in->data` is not copied, but is also pointed to.

out->error=='a'	Allocation error. In extreme cases, where there aren't even a few hundred bytes available, I will return `NULL`.
out->error=='s'	Error copying settings groups.
out->error=='p'	Error copying parameters or info page; the given apop_data struct may be `NULL` or may have its own `->error` element.

8.2.2.81 **apop_data*** apop_model_draws (**apop_model** * model, int count, **apop_data** * draws)

Make a set of random draws from a model and write them to an apop_data set.

model	The model from which draws will be made. Must already be prepared and/or estimated.
count	The number of draws to make. If `draw_matrix` is not `NULL`, then this is ignored and `count=draw_matrix->matrix->size1`. default=1000.
draws	If not `NULL`, a pre-allocated data set whose `matrix` element will be filled with draws.

Returns

An apop_data set with the matrix filled with `size` draws. If `draw_matrix!=NULL`, then return a pointer to it.

out->error=='m'	Input model isn't good for making draws: it is `NULL`, or `m->dsize=0`.
out->error=='s'	You gave me a `draws` matrix, but its size is less than the size of a single draw from the data, `model->dsize`.
out->error=='d'	Trouble drawing from the distribution for at least one row. That row is set to all `NAN`.

- Prints a warning if you send in a non-`NULL` apop_data set, but its `matrix` element is `NULL`, when `apop_opts.verbose>=1`.
- See also apop_draw, which makes a single draw.
- Random numbers are generated using RNGs from apop_rng_get_thread, qv.

Here is a two-line program to draw a different set of ten Standard Normals on every run (provided runs are more than a second apart):

```
#include <apop.h>
#include <time.h>

int main(){
    apop_opts.rng_seed = time(NULL);
    apop_data_print(
            apop_model_draws(
                apop_model_set_parameters(apop_normal, 0, 1),
                .count=10,
            )
    );
}
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.82 long double apop_model_entropy (**apop_model** * in, int draws)

Calculate the entropy of a model: $\int -\ln(p(x))p(x)dx$, which is the expected value of $-\ln(p(x))$.

The default method is to make draws using apop_model_draws, then evaluate the log likelihood at those points using the model's `log_likelihood` method.

There are a number of routines for specific models, inlcuding the apop_normal and apop_pmf models.

- If you want the entropy of a data set, see apop_vector_entropy.
- The entropy is calculated using natural logs. If you prefer base-2 logs, just divide by ln(2): `apop_model_entropy(my_model)/log(2)`.

in	A parameterized apop_model. That is, you have already used apop_estimate or apop_model_set_parameters to estimate/set the model parameters.
draws	If using the default method of making random draws, how many random draws to make (default=1,000)

Sample code:

```
#include <apop.h>
#define Diff(left, right, eps) Apop_stopif(fabs((left)-(right))>(eps), \
        abort(), 0, "%g is too different from %g (abitrary limit=%g).", \
        (double)(left), (double)(right), eps)


/* The entropy function, like some other functions (including apop_update) has a lookup
 table for known models like the Normal distribution. If the input model has
 \c log_likelihood, \c p, and \c draw functions that are the ones found in \ref
 apop_nomrmal, then use a known calculation to report entropy; else report based on
 random draws from the model.

If we make a copy of the \ref apop_normal model and replace the log likelihood with
a new function that produces identical values, the lookup table will not find the
modified model, and the calculation via random draws will be done. Of course, the
final entropy as calculated using both methods should differ only by a small amount.
*/

long double mask(apop_data *d, apop_model *m){
    return apop_normal->log_likelihood(d, m);
}

int main(){
    for (double i=0.1; i< 10; i+=.2){
        apop_model *n = apop_model_set_parameters(apop_normal, 8, i);
        long double v= apop_model_entropy(n);
        n->log_likelihood = mask;
```

```
        long double w= apop_model_entropy(n, 50000);
        Diff(v, w, 5e-2);
    }
}
```

8.2.2.83 **apop_model*** apop_model_fix_params (**apop_model** * model_in)

Produce a model based on another model, but with some of the parameters fixed at a given value.

You will send me the model whose parameters you want fixed, with the `parameters` element set as follows. For the fixed parameters, simply give the values to which they will be fixed. Set the free parameters to `NaN`.

For example, here is a Binomial distribution with a fixed $n = 30$ but p_1 allowed to float freely:

```
apop_model *bi30 = apop_model_fix_params(apop_model_set_parameters(apop_binomial, 30, NAN));
Apop_model_add_group(bi30, apop_mle, .starting_pt=(double[]){.5}); // The Binomial doesn't like the
                                                                   //  default starting point of 1.
apop_model *out = apop_estimate(your_data, bi30);
```

The output is an `apop_model` that can be estimated, Bayesian updated, et cetera.

- Rather than using this model, you may simply want a now-filled-in copy of the original model. Use apop_model_fix_params_get_base to retrieve the original model's parameters.
- The `estimate` method always uses an MLE, and it never calls the base model's `estimate` method.
- If the input model has an apop_mle_settings group attached, I'll use them for the `estimate` method. Otherwise, I'll set my own.
- If the parameter input has non-NaN values at the free parameters, then I'll use those as the starting point for any MLE search; the defaults for the variables without fixed values starts from **1** as usual.
- I do check the `more` pointer of the `parameters` for additional pages and `NaN`s on those pages.

Here is a sample program. It produces a few thousand draws from a Multivariate Normal distribution, and then tries to recover the means given a var/covar matrix fixed at the correct variance.

```
#include <apop.h>

int main(){
    size_t ct = 5e4;

    //set up the model & params
    apop_data *params = apop_data_falloc((2,2,2), 8,  1, 0.5,
                                                  2,  0.5, 1);
    apop_model *pvm = apop_model_copy(apop_multivariate_normal);
    pvm->parameters = apop_data_copy(params);
    pvm->dsize = 2;
    apop_data *d = apop_model_draws(pvm, ct);

    //set up and estimate a model with fixed covariance matrix but free means
    gsl_vector_set_all(pvm->parameters->vector, GSL_NAN);
    apop_model *mep1 = apop_model_fix_params(pvm);
    apop_model *e1 = apop_estimate(d, mep1);

    //compare results
    printf("original params: ");
    apop_vector_print(params->vector);
    printf("estimated params: ");
    apop_vector_print(e1->parameters->vector);
    assert(apop_vector_distance(params->vector, e1->parameters->vector)<1e-2);
}
```

model_in	The base model

Returns

a model that can be used like any other, with the given params fixed or free.

8.2.2.84 **apop_model*** apop_model_fix_params_get_base (**apop_model** * fixed_model)

The apop_model_fix_params function produces a model that has only the non-fixed parameters of the model. After estimation of the fixed-parameter model, this function fills the `parameters` element of the base model and returns a pointer to the base model.

8.2.2.85 void apop_model_free (**apop_model** * free_me)

Free an apop_model structure.

- The `parameters` and `settings` are freed. These are the elements that are copied by `apop_model_copy`.
- The `data` element is not freed, because the odds are you still need it.
- If `free_me->more_size` is positive, the function runs `free(free_me->more)`. But it has no idea what the `more` element contains; if it points to other structures (like an apop_data set), you need to free them before calling this function.
- If `free_me` is `NULL`, this does nothing.

free_me	A pointer to the model to be freed.

8.2.2.86 **apop_data*** apop_model_hessian (**apop_data** * data, **apop_model** * model, double delta)

Numerically estimate the matrix of second derivatives of the parameter values, via a series of re-evaluations at small differential steps. [Therefore, it may be expensive to do this for a very computationally-intensive model.]

data	The apop_data at which the model was estimated (default: `NULL`)
model	The apop_model, with parameters already estimated (no default, must not be `NULL`)
delta	the step size for the differentials. (default: 1e-3, but see below)

Returns

The matrix of estimated second derivatives at the given data and parameter values.

- If you do not set delta as an input, I first look for an apop_mle_settings group attached to the input model, and check that for a `delta` element. If that is also missing, use the default of 1e-3.
- This function uses the Designated initializers syntax for inputs.

8.2.2.87 **apop_model*** apop_model_metropolis (**apop_data** * d, gsl_rng * rng, **apop_model** * m)

Use `Metropolis-Hastings Markov chain Monte Carlo` to make draws from the given model.

The basic storyline is that draws are made from a proposal distribution, and the likelihood of your model given your data and the drawn parameters evaluated. At each step, a new set of proposal parameters are drawn, and if they are more likely than the previous set the new proposal is accepted as the next step, else with probability (prob of new params)/(prob of old params), they are accepted as the next step anyway. Otherwise the last accepted proposal is repeated.

The output is an apop_pmf model with a data set listing the draws that were accepted, including those repetitions. The output model is modified so that subsequent draws are one more step from the Markov chain, via apop_model_metropolis_draw.

d	The apop_data set used for evaluating the likelihood of a proposed parameter set.
rng	A `gsl_rng`, probably allocated via apop_rng_alloc. (Default: an RNG from apop_rng_get_thread)
m	The apop_model from which parameters are being drawn. (No default; must not be `NULL`)

Returns

A modified apop_pmf model representing the results of the search. It has a specialized `draw` method that returns another step from the Markov chain with each draw.

out->error='c'	Proposal was outside of a constraint; see below.

- If a proposal fails to meet the `constraint` element of the model you input, then the proposal is thrown out and a new one selected. By the default proposal distribution, this is not mathematically correct (it breaks detailed balance), and values near the constraint will be oversampled. The output model will have `outmodel->error=='c'`. It is up to you to decide whether the resulting distribution is good enough for your purposes or whether to take the time to write a custom proposal and step function to accommodate the constraint.

Attach an apop_mcmc_settings group to your model to specify the proposal distribution, burnin, and other details of the search. See the apop_mcmc_settings documentation for details.

- The default proposal includes an adaptive step: you specify a target accept rate (default: .35), and if the accept rate is currently higher the variance of the proposals is widened to explore more of the space; if the accept rate is currently lower the variance is narrowed to stay closer to the last accepted proposal. Technically, this breaks ergodicity of the Markov chain, but the consensus seems to be that this is not a serious problem. If it does concern you, you can set the `base_adapt_fn` in the apop_mcmc_settings group to a do-nothing function, or one that damps its adaptation as $n \to \infty$.
- If you have a univariate model, apop_arms_draw may be a suitable simpler alternative.
- Note the `gibbs_chunks` element of the apop_mcmc_settings group. If you set `gibbs_chunks='a'`, all parameters are drawn as a set, and accepted/rejected as a set. The variances are adapted at an identical rate. If you set `gibbs_chunks='i'`, then each scalar parameter is assigned its own proposal distribution, which is adapted at its own pace. With `gibbs_chunks='b'` (the default), then each of the vector, matrix, and weights of your model's parameters are drawn/accepted/adapted as a block (and so on to additional chunks if your model has `->more` pages). This works well for complex models which naturally break down into subsets of parameters.
- Each chunk counts as a step in the Markov chain. Therefore, if there are several chunks, you can expect chunks to repeat from step to step. If you want a draw after cycling through all chunks, try using apop_model_metropolis_draw, which has that behavior.
- If the likelihood model has `NULL` parameters, I will allocate them. That means you can use one of the stock models that ship with Apophenia. If I need to run the model's prep routine to get the size of the

parameters, then I will make a copy of the likelihood model, run prep, and then allocate parameters for that copy of a model.

- On exit, the `parameters` element of your likelihood model has the last accepted parameter proposal.
- If you set `apop_opts.verbose=2` or greater, I will report the accept rate of the M-H sampler. It is a common rule of thumb to select a proposal so that this is between 20% and 50%. Set `apop_opts.verbose=3` to see the stream of proposal points, their likelihoods, and the acceptance odds. You may want to set `apop_opts.log_file=fopen("yourlog", "w")` first.
- This function uses the Designated initializers syntax for inputs.

8.2.2.88 int apop_model_metropolis_draw (double * out, gsl_rng * rng, **apop_model** * model)

The draw method for models estimated via apop_model_metropolis.

That method produces an apop_pmf, typically with a few thousand draws from the model in a batch. If you want to get a single next step from the Markov chain, use this.

A Markov chain works by making a new draw and then accepting or rejecting the draw. If the draw is rejected, the last value is reported as the next step in the chain. Users sometimes mitigate this repetition by making a batch of draws (say, ten at a time) and using only the last.

If you run this without first running apop_model_metropolis, I will run it for you, meaning that there will be an initial burn-in period before the first draw that can be reported to you. That run is done using `model->data` as input.

out	An array of `doubles`, which will hold the draw, in the style of apop_draw.
rng	A `gsl_rng`, already initialized, probably via apop_rng_alloc.
model	A model which was probably already run through apop_model_metropolis.

Returns

On return, `out` is filled with the next step in the Markov chain. The `->data` element of the PMF model is extended to include the additional steps in the chain. If a proposal failed the model constraints, then return 1; else return 0. See the notes in the documentation for apop_model_metropolis.

- After pulling the attached settings group, the parent model is ignored. One expects that `base_model` in the mcmc settings group == the parent model.
- If your settings break the model parameters into several chunks, this function returns after stepping through all chunks.

8.2.2.89 **apop_data*** apop_model_numerical_covariance (**apop_data** * data, **apop_model** * model, double delta)

Produce the covariance matrix for the parameters of an estimated model via the derivative of the score function at the parameter. I.e., I find the second derivative via apop_model_hessian , and take the negation of the inverse.

I follow Efron and Hinkley in using the estimated information matrix—the value of the information matrix at the estimated value of the score—not the expected information matrix that is the integral over all possible data. See Pawitan 2001 (who cribbed a little off of Efron and Hinkley) or Klemens 2008 (who directly cribbed off of both) for further details.

data	The data by which your model was estimated

model	A model whose parameters have been estimated.
delta	The differential by which to step for sampling changes. (default: 1e-3, but see below)

Returns

A covariance matrix for the data. Also, if the data does not have a "<Covariance>" page, I'll set it to the result as well [i.e., I won't overwrite an existing covariance page].

- If you do not set delta as an input, I first look for an apop_mle_settings group attached to the input model, and check that for a `delta` element. If that is also missing, use the default of 1e-3.
- This function uses the Designated initializers syntax for inputs.

8.2.2.90 void apop_model_print (**apop_model** * model, FILE * output_pipe)

Print the results of an estimation for a human to look over.

model	The model whose information should be displayed (No default. If `NULL`, print `NULL`)

output_pipe	The output stream. Default: `stdout`. If you'd like something else, use `fopen`. E.g.:
	`1 FILE *out =fopen("outfile.txt", "w"); //or "a" to append.`
	`2 apop_model_print(the_model, out);`
	`3 fclose(out);  //optional in many cases.`

- The default prints the name, parameters, info, &c. but I check a vtable for alternate methods you define; see Registering new methods in vtables for details. The typedef new functions must conform to and the hash used for lookups are:

```
1 typedef void (*apop_model_print_type)(apop_model *params, FILE *out);
2 #define apop_model_print_hash(m1) ((m1)->log_likelihood ? (size_t)(m1)->log_likelihood : \
3             (m1)->p ? (size_t)(m1)->p*33 : \
4             (m1)->estimate ? (size_t)(m1)->estimate*33*33 : \
5             (m1)->draw ? (size_t)(m1)->draw*33*27  : \
6             (m1)->cdf ? (size_t)(m1)->cdf*27*27 : 27)
```

When building a special print method, all output should `fprintf` to the input `FILE*` handle. Apophenia's output routines also accept a file handle; e.g., if the file handle is named `out`, then if the `thismodel` print method uses `apop_data_print` to print the parameters, it must do so via a form like `apop_data_print(thismodel->parameters, .output_pipe=ap)`.

Your `print` method can use both by masking itself for a few lines:

```
1 void print_method(apop_model *in, FILE* ap){
2   void *temp = in->estimate;
3   in->estimate = NULL;
4   apop_model_print(in, ap);
5   in->estimate = temp;
6
7   printf("Additional info:\n");
8   ...
9 }
```

- Print methods are intended for human consumption and are subject to change.
- This function uses the Designated initializers syntax for inputs.

8.2.2.91 **apop_model*** apop_model_to_pmf (**apop_model** * model, **apop_data** * binspec, long int draws, int bin_count)

Make random draws from an apop_model, and bin them using a binspec in the style of apop_data_to_bins. If you have a data set that used the same binspec, you now have synced histograms, which you can plot or sensibly test hypotheses about.

binspec	A description of the bins in which to place the draws; see apop_data_to_bins. (default: as in apop_data_to_bins.)
model	The model to be drawn from. Because this function works via random draws, the model needs to have a `draw` method. (No default)
draws	The number of random draws to make. (arbitrary default = 10,000)
bin_count	If no bin spec, the number of bins to use (default: as per apop_data_to_bins, $\sqrt{(N)}$)

Returns

An apop_pmf model, with a new binned data set attached (which you may have to `apop_data_free(output_model->data)` to prevent memory leaks). The weights on the data set are normalized to sum to one.

- This function uses the Designated initializers syntax for inputs.

8.2.2.92 long double apop_multivariate_gamma (double a, int p)

The multivariate generalization of the Gamma distribution.

$$\Gamma_p(a) = \pi^{p(p-1)/4} \prod_{j=1}^{p} \Gamma\left[a + (1-j)/2\right].$$

Because $\Gamma(x)$ is undefined for $x \in \{0, -1, -2, ...\}$, this function returns `NAN` when $a + (1-j)/2$ takes on one of those values.

See also apop_multivariate_lngamma, which is more numerically stable in most cases.

8.2.2.93 long double apop_multivariate_lngamma (double a, int p)

The log of the multivariate generalization of the Gamma; see also apop_multivariate_gamma.

8.2.2.94 int apop_name_add (**apop_name** * n, char const * add_me, char type)

Adds a name to the apop_name structure. Puts it at the end of the given list.

n	An existing, allocated apop_name structure.
add_me	A string. If `NULL`, do nothing; return -1.
type	'r': add a row name 'c': add a matrix column name 't': add a text column name 'h': add a title (i.e., a header). 'v': add (or overwrite) the vector name

Returns

Returns the number of rows/cols/depvars after you have added the new one. But if `add_me` is `NULL`, return -1.

8.2.2.95 **apop_name*** apop_name_alloc (void)

Allocates a name structure

Returns

An allocated, empty name structure. In the very unlikely event that `malloc` fails, return `NULL`.

Because apop_data_alloc uses this to set up its output, you will rarely if ever need to call this function explicitly. You may want to use it if wrapping a `gsl_matrix` into an apop_data set. For example, to put a title on a vector:

```
1 apop_data *d = &(apop_data){.vector=your_vector, .names=apop_name_alloc()};
2 apop_name_add(d->names, "A column of numbers", 'v');
3 apop_data_print(d);
4
5 ...
6 apop_name_free(d->names); //but d itself is auto-allocated; no need to free it.
```

8.2.2.96 **apop_name*** apop_name_copy (**apop_name** * in)

Copy one apop_name structure to another. That is, all data is duplicated.

Used internally by apop_data_copy, but sometimes useful by itself. For example, say that we have an apop_data struct named `d` and a gsl_matrix of the same dimensions named `m`; we could give `m` the labels from `d` for printing:

```
1 apop_data *wrapped = &(apop_data){.matrix=m, .names=apop_name_copy(d)};
2 apop_data_print(wrapped);
3 apop_name_free(wrapped->names); //wrapped itself is auto-allocated; do not free.
```

in	The input names

Returns

A apop_name struct with copies of all input names.

8.2.2.97 int apop_name_find (const **apop_name** * n, const char * name, const char type)

Finds the position of an element in a list of names.

The function uses POSIX's `strcasecmp`, and so does case-insensitive search the way that function does.

n	the apop_name object to search.
name	the name you seek; see above.
type	`'c'` (=column), `'r'` (=row), or `'t'` (=text). Default is `'c'`.

Returns

The position of `findme`. If `'c'`, then this may be -1, meaning the vector name. If not found, returns -2. On error, e.g. `name==NULL`, returns -2.

8.2.2.98 void apop_name_free (**apop_name** * free_me)

Free the memory used by an apop_name structure.

8.2.2.99 void apop_name_print (**apop_name** * n)

Prints the given list of names to stdout. Useful for debugging.

n	The apop_name structure

8.2.2.100 void apop_name_stack (**apop_name** * n1, **apop_name** * nadd, char type1, char typeadd)

Append one list of names to another.

If the first list is empty, then this is a copy function.

n1	The first set of names (no default, must not be `NULL`)
nadd	The second set of names, which will be appended after the first. (no default. If `NULL`, a no-op.)
type1	Either 'c', 'r', 't', or 'v' stating whether you are merging the columns, rows, text, or vector. If 'v', then ignore `typeadd` and just overwrite the target vector name with the source name. (default: 'r')
typeadd	Either 'c', 'r', 't', or 'v' stating whether you are merging the columns, rows, or text. If 'v', then overwrite the target with the source vector name. (default: type1)

8.2.2.101 gsl_vector* apop_numerical_gradient (**apop_data** * data, **apop_model** * model, double delta)

A wrapper around the GSL's one-dimensional `gsl_deriv_central` to find a numeric differential for each dimension of the input apop_model's log likelihood (or `p` if `log_likelihood` is `NULL`).

data	The apop_data set to use for all evaluations.
model	The apop_model, expressing the function whose derivative is sought. The gradient is taken via small changes along the model parameters.
delta	The size of the differential. (default: 1e-3, but see below)

```
1 gsl_vector *gradient = apop_numerical_gradient(data, your_parametrized_model);
```

- If you do not set delta as an input, I first look for an apop_mle_settings group attached to the input model, and check that for a `delta` element. If that is also missing, use the default of 1e-3.
- This function uses the Designated initializers syntax for inputs.

8.2.2.102 double apop_p (**apop_data** * d, **apop_model** * m)

Find the probability of a data/parametrized model pair.

d	The data
m	The parametrized model, which must have either a `log_likelihood` or a `p` method.

8.2.2.103 **apop_data*** apop_paired_t_test (gsl_vector * a, gsl_vector * b)

Answers the question: with what confidence can I say that the mean difference between the two columns is zero?

If `apop_opts.verbose >=2`, then display some information, like the mean/var/count for both vectors and the t statistic, to stderr.

a	A column of data
b	A matched column of data

Returns

an apop_data set with the following elements: `mean left - right`: the difference in means; if positive, first vector has larger mean, and one-tailed test is testing $L > R$, else reverse if negative.
`t statistic`: used for the test
`df`: degrees of freedom
`p value, 1 tail`: the p-value for a one-tailed test that one vector mean is greater than the other.
`confidence, 1 tail`: 1- p value.
`p value, 2 tail`: the p-value for the two-tailed test that left mean = right mean.
`confidence, 2 tail`: 1-p value

See also

apop_t_test for an example, and for when the element-by-element difference between the vectors has no sensible interpretation.

8.2.2.104 **apop_model*** apop_parameter_model (**apop_data** * d, **apop_model** * m)

Get a model describing the distribution of the given parameter estimates.

For many models, the parameter estimates are well-known, such as the t-distribution of the parameters for OLS.

For models where the distribution of $\hat{p}$ is not known, if you give me data, I will return an apop_normal or apop_multivariate_normal model, using the parameter estimates as mean and apop_bootstrap_cov for the variances.

If you don't give me data, then I will assume that this is a stochastic model where re-running the model will produce different parameter estimates each time. In this case, I will run the model 1e4 times and return a apop_pmf model with the resulting parameter distributions.

Before calling this, I expect that you have already run apop_estimate to produce $\hat{p}$.

The apop_pm_settings structure dictates details of how the model is generated. For example, if you want only the distribution of the third parameter, and you know the distribution will be a PMF generated via random draws, then set settings and call the model via:

```
1 apop_model_group_add(your_model, apop_pm, .index =3, .draws=3e5);
2 apop_model *dist = apop_parameter_model(your_data, your_model);
```

Some useful parts of apop_pm_settings:

- `index` gives the position of the parameter (in apop_data_pack order) in which you are interested. Thus, if this is zero or more, then you will get a univariate output distribution describing a single parameter. If `index == -1`, then I will give you the multivariate distribution across all parameters. The default is zero (i.e. the univariate distribution of the zeroth parameter).
- `draws` If there is no closed-form solution and bootstrap is inappropriate, then the last resort is a large numbr of random draws of the model, summarized into a PMF. Default: 1,000 draws.
- `rng` If the method requires random draws, then use this. If you provide `NULL` and one is needed, I provide one for you via apop_rng_get_thread.

The default is via resampling as above, but special-case calculations for certain models are held in a vtable; see Registering new methods in vtables for details. The typedef new functions must conform to and the hash used for lookups are:

```
1 typedef apop_model* (*apop_parameter_model_type)(apop_data *, apop_model *);
2 #define apop_parameter_model_hash(m1) ((size_t)((m1).log_likelihood ? (m1).log_likelihood : (m1).p)*33 +
      (m1).estimate ? (size_t)(m1).estimate: 27)
```

8.2.2.105 **apop_data*** apop_predict (**apop_data** * d, **apop_model** * m)

A prediction supplies E(a missing value | original data, already-estimated parameters, and other supplied data elements).

For a regression, one would first estimate the parameters of the model, then supply a row of predictors **X**. The value of the dependent variable y is unknown, so the system would predict that value.

For a univariate model (i.e. a model in one-dimensional data space), there is only one variable to omit and fill in, so the prediction problem reduces to the expected value: E(a missing value | original data, already-estimated parameters). [In some models, this may not be the expected value, but is a best value for the missing item using some other meaning of 'best'.]

In other cases, prediction is the missing data problem: for three-dimensional data, you may supply the input (34, `NaN`, 12), and the parameterized model provides the most likely value of the middle parameter given the parameters and known data.

- If you give me a `NULL` data set, I will assume you want all values filled in, for most models with the expected value.

- If you give me data with `NaNs`, I will take those as the points to be predicted given the provided data.

If the model has no `predict` method, the default is to use the apop_ml_impute function to do the work. That function does a maximum-likelihood search for the best parameters.

Returns

If you gave me a non-`NULL` data set, I will return that, with the `NaNs` filled in. If `NULL` input, I will allocate an apop_data set and fill it with the expected values.

There may be a second page (i.e., a apop_data set attached to the `->more` pointer of the main) listing confidence and standard error information. See your specific model documentation for details.

- Special-case calculations for certain models are held in a vtable; see Registering new methods in vtables for details. The typedef new functions must conform to and the hash used for lookups are:

```
1 typedef apop_data * (*apop_predict_type)(apop_data *d, apop_model *params);
2 #define apop_predict_hash(m1) ((size_t)((m1).log_likelihood ? (m1).log_likelihood : (m1).p)*33 +
      (m1).estimate ? (size_t)(m1).estimate: 27)
```

8.2.2.106 void apop_prep (**apop_data** * d, **apop_model** * m)

Allocate and initialize the `parameters`, `info`, and other requisite parts of a apop_model.

Some models have associated prep routines that also attach settings groups to the model, and set up additional special-case functions in vtables.

- The input model is modified in place.

- If called repeatedly, subsequent calls to apop_prep are no-ops. Thus, a model can not be re-prepped using a new data set or other conditions.

- The default prep is to simply call apop_model_clear. If the input apop_model has a prep method, then that gets called instead.

8.2.2.107 int apop_prep_output (char const * output_name, FILE ** output_pipe, char * output_type, char * output_append)

If you're reading this, it is probably because you were referred by another function that uses this internally. You should never call this function directly, but do read this documentation.

There are four settings that affect how output happens, which can be set when you call the function that sent you to this documentation, e.g:

```
1 apop_data_print(your_data, .output_type ='f', .output_append = 'w');
```

output_name	The name of the output file, if any. For a database, the table to write.
output_pipe	If you have already opened a file and have a `FILE*` on hand, use this instead of giving the file name.
output_type	`'p'` = pipe, `'f'`= file, `'d'` = database
output_append	`'a'` = append (default), `'w'` = write over.

At the end, `output_name`, `output_pipe`, and `output_type` are all set. Notably, the local `output_pipe` will have the correct location for the calling function to `fprintf` to.

- See legi for more discussion.
- The default is output to stdout. For example,

```
1 apop_data_print(your_data);
2 //is equivalent to
3 apop_data_print(your_data, .output_type='p', .output_pipe=stdout);
```

- Tip: if writing to the database, you can get a major speed boost by wrapping the call in a begin/commit wrapper:

```
1 apop_query("begin;");
2 apop_data_print(your_data, .output_name="dbtab", .output_type='d');
3 apop_query("commit;");
```

8.2.2.108 int apop_query (const char * fmt, ...)

Send a query to the database that returns no data.

- As with functions like the `apop_query_to_data`, the query can include printf-style format specifiers, such as `apop_query("create table %s(id, name, age);", tablename)`.

fmt	A `printf`-style SQL query.

Returns

0 on success, 1 on failure.

8.2.2.109 **apop_data*** apop_query_to_data (const char * fmt, ...)

Queries the database and dumps the result into an apop_data set.

fmt	A `printf`-style SQL query.

Returns

If no rows are returned, NULL; else an apop_data set with the data in place. Most data will be in the matrix element of the output. Column names are appropriately placed. If apop_opts.db_name_column matches one of the fields in your query's output (default: row_names), then that column will be used for row names (and therefore will not appear in the matrix).

out->error=='q'	Query error. A valid query that returns no rows is not an error; in that case, you get NULL.

- The query can include printf-style format specifiers, such as `apop_query_to_data("select age from %s where id=%i;", tablename, id_number)`.
- Blanks in the database (i.e., NULLs) and elements that match apop_opts.nan_string are filled with NANs in the matrix.

8.2.2.110 double apop_query_to_float (const char * fmt, ...)

Queries the database, and dumps the result into a single double-precision floating point number.

- This calls apop_query_to_data and returns the (0,0)th element of the returned matrix. Thus, if your query returns multiple lines, you will get no warning, and the function will return the first in the list (which is not always well-defined; maybe use an `order by` clause in your query if you expect multiple lines).
- If `apop_opts.db_name_column` is set, then I'll ignore that column. It gets put into the names of the apop_data set, and then thrown away when I look at only the `gsl_matrix` element of that set.
- If the query produces a blank table, returns NAN, and if `apop_opts.verbose>=2`, prints an error.
- The query can include printf-style format specifiers, such as `apop_query_to_float("select age from %s where id=%i;", tablename, id_number)`.
- If the query produces an error, returns NAN, and if `apop_opts.verbose>=0`, prints an error. If you need to distinguish between blank tables, NaNs in the data, and query errors, use apop_query_to_data.

fmt	A `printf`-style SQL query.

Returns

A `double`, actually.

8.2.2.111 **apop_data*** apop_query_to_mixed_data (const char * typelist, const char * fmt, ...)

Query data to an `apop_data` set, but a mix of names, vectors, matrix elements, and text.

If you are querying to a matrix and maybe a name, use `apop_query_to_data` (and set apop_opts.db_name_column if desired). If querying only text, use apop_query_to_text. But if your data is a mix of text and numbers, use this.

The first argument is a character string consisting of the letters `nvmtw`, one for each column of the SQL output, indicating whether the column is a name, vector, matrix column, text column, or weight vector. You can have only one `n`, one `v`, and one `w`.

If the query produces more columns than there are elements in the column specification, then the remainder are dumped into the text section. If there are fewer columns produced than given in the spec, the additional elements will be allocated but not filled (i.e., they are uninitialized and will have garbage).

typelist	A string consisting of the letters `nvmtw`. For example, if your query columns should go into a text column, the vector, the weights, and two matrix columns, this would be "tvwmm".
fmt	A `printf`-style SQL query.

out->error=='d'	Dimension error. Your count of matrix parts didn't match what the query returned.
out->error=='q'	Query error. A valid query that returns no rows is not an error; in that case, you get `NULL`.

- apop_opts.db_name_column is ignored. Use the `'n'` character to indicate the output column with row names.
- As with the other `apop_query_to_...` functions, the query can include printf-style format specifiers, such as `apop_query_to_mixed_data("tv", "select name, age from`

8.2.2.112 **apop_data*** apop_query_to_text (const char * fmt, ...)

Dump the results of a query into an array of strings.

Returns

An apop_data structure with the `text` element filled.

fmt	A `printf`-style SQL query.

out->error=='q'	The database engine was unable to run the query (e.g., invalid SQL syntax). Again, a valid query that returns zero rows is not an error, and `NULL` is returned.
out->error=='d'	Database error.

- If `apop_opts.db_name_column` matches a column of the output table, then that column is used for row names, and therefore will not be included in the `text`.
- `query_output->text` is always a 2-D array of strings, even if the query returns a single column. In that case, use `returned_tab->text[i][0]` (or equivalently, `*returned_tab->text[i]`) to refer to row `i`.
- If an element in the database is `NULL`, the corresponding cell in the output table will be filled with the text given by `apop_opts.nan_string`. The default is `"NaN"`, but you can set `apop_opts.nan_string = "whatever you like"` to change the text to whatever you like.
- Returns `NULL` if your query is valid but returns zero rows.
- The query can include printf-style format specifiers, such as `apop_query_to_text("select name from %s where id=%i;", tablename, id_number)`.

For example, the following function will list the tables in an SQLite database (much like you could do from the command line using `sqlite3 dbname.db ".table"`).

```
#include <apop.h>

void print_table_list(char *db_file){
    apop_db_open(db_file);
    apop_data *tab_list= apop_query_to_text("select name "
                    "from sqlite_master where type=='table'");
```

```
    for(int i=0; i< tab_list->textsize[0]; i++)
        printf("%s\n", tab_list->text[i][0]);
}

int main(int argc, char **argv){
    if (argc == 1){
        printf("Give me a database name, and I will print out "
               "the list of tables contained therein.\n");
        return 0;
    }
    print_table_list(argv[1]);
}
```

8.2.2.113 gsl_vector* apop_query_to_vector (const char * fmt, ...)

Queries the database and dumps the first column of the result into a `gsl_vector`.

fmt	A `printf`-style SQL query.

Returns

A `gsl_vector` holding the first column of the returned matrix. Thus, if your query returns multiple lines, you will get no warning, and the function will return the first in the list.

out->error=='q'	Query error. A valid query that returns no rows is not an error; in that case, you get `NULL`.

- Uses apop_query_to_data internally, then throws away all but the first column of the matrix.
- If `apop_opts.db_name_column` is set, then I'll ignore that column. It gets put into the names of the apop_data set, and then thrown away when I look at only the `gsl_matrix` part of that set.
- If the query returns zero rows of data or no columns, the function returns `NULL`.
- The query can include printf-style format specifiers, such as `apop_query_to_vector("select age from %s where id=%i;", tablename, id_number)`.

8.2.2.114 **apop_data*** apop_rake (char const * margin_table, char *const * var_list, int var_ct, char *const * contrasts, int contrast_ct, char const * structural_zeros, int max_iterations, double tolerance, char const * count_col, char const * init_table, char const * init_count_col, double nudge)

Fit a log-linear model via iterative proportional fitting, aka raking.

Raking has many uses. The Modeling with Data blog presents a series of discussions of uses of raking, including some worked examples.

Or see Wikipedia for an overview of Log linear models, aka Poisson regressions. One approach toward log-linear modeling is a regression form; let there be four categories, A, B, C, and D, from which we can produce a model positing, for example, that cell count is a function of a form like $g_1(A) + g_2(BC) + g_3(CD)$. In this case, we would assign a separate coefficient to every possible value of A, every possible value of (B, C), and every value of (C, D). Raking is the technique that searches for that large set of parameters.

The combinations of categories that are considered to be relevant are called *contrasts*, after ANOVA terminology of the 1940s.

The other constraint on the search are structural zeros, which are values that you know can never be non-zero, due to field-specific facts about the variables. For example, U.S. Social Security payments are available only to those age 65 or older, so "age <65 and gets_soc_security=1" is a structural zero.

Because there is one parameter for every combination, there may be millions of parameters to estimate, so the search to find the most likely value requires some attention to technique. For over half a century, the consensus method for searching has been raking, which iteratively draws each category closer to the mean in a somewhat simple manner (this was first developed circa 1940 and had to be feasible by hand), but which is guaranteed to eventually arrive at the maximum likelihood estimate for all cells.

Another complication is that the table is invariably sparse. One can easily construct tables with millions of cells, but the corresponding data set may have only a few thousand observations.

This function uses the database to resolve the sparseness problem. It constructs a query requesting all combinations of categories the could possibly be non-zero after raking, given all of the above constraints. Then, raking is done using only that subset. This means that the work is done on a number of cells proportional to the number of data points, not to the full cross of all categories. Set `apop_opts.verbose` to 2 or greater to show the query on `stderr`.

- One could use raking to generate 'fully synthetic' data: start with observation-level data in a margin table. Begin the raking with a starting data set of all-ones. Then rake until the all-ones set transforms into something that conforms to the margins and (if any) structural zeros. You now have a data set which matches the marginal totals but does not use any other information from the observation-level data. If you do not specify an `.init_table`, then an all-ones default table will be used.

margin_table	The name of the table in the database to use for calculating the margins. The table should have one observation per row. (No default)
var_list	The full list of variables to search. A list of strings, e.g., `(char *[]){"var1", "var2", ..., "var15"}`
var_ct	The count of the full list of variables to search.
contrasts	The contrasts describing your model. Like the `var_list` input, a list of strings like `(char *[]){"var1", "var7", "var13"}` contrast is a pipe-delimited list of variable names. (No default)
contrast_ct	The number of contrasts in the list of contrasts. (No default)
structural_zeros	a SQL clause indicating combinations that can never take a nonzero value. This will go into a `where` clause, so anything you could put there is OK, e.g. "age <65 and gets_soc_security=1 or age <15 and married=1". Your margin data is not checked for structural zeros. (default: no structural zeros)
max_iterations	Number of rounds of raking at which the algorithm halts. (default: 1000)
tolerance	I calculate the change for each cell from round to round; if the largest cell change is smaller than this, I stop. (default: 1e-5)
count_col	This column gives the count of how many observations are represented by each row. If `NULL`, ech row represents one person. (default: `NULL`)
init_table	The default is to initially set all table elements to one and then rake from there. This is effectively the 'fully synthetic' approach, which uses only the information in the margins and derives the data set closest to the all-ones data set that is consistent with the margins. Care is taken to maintan sparsity in this case. If you specify an `init_table`, then I will get the initial cell counts from it. (default: the fully-synthetic approach, using a starting point of an all-ones grid.)
init_count_col	The column in `init_table` with the cell counts.
nudge	There is a common hack of adding a small value to every zero entry, because a zero entry will always scale to zero, while a small value could eventually scale to anything. Recall that this function works on sparse sets, so I first filter out those cells that could possibly have a nonzero value given the observations, then I add `nudge` to any zero cells within that subset.

Returns

An apop_data set where every row is a single combination of variable values and the `weights` vector gives the most likely value for each cell.

out->error='i'	Input was somehow wrong.
out->error='c'	Raking did not converge, reached max. iteration count.

- Set `apop_opts.verbose=3` to see the intermediate tables at the end of each round of raking.
- If you want all cells to have nonzero value, then you can do that via pre-processing:

```
1 apop_query("update data_table set count_col = 1e-3 where count_col = 0");
```

- This function is thread-safe. To make this happen, temp database tables are named using a number built with `omp_get_thread_num`.
- This function uses the Designated initializers syntax for inputs.

8.2.2.115 int apop_regex (const char * string, const char * regex, **apop_data** ** substrings, const char use_case)

Extract subsets from a string via regular expressions.

This function takes a regular expression and repeatedly applies it to an input string. It returns the count of matches, and optionally returns the matches themselves organized into the `text` grid of an apop_data set.

- There are three common flavors of regular expression: Basic, Extended, and Perl-compatible (BRE, ERE, PCRE). I use EREs, as per the specs of your C library, which should match POSIX's ERE specification.

For example, "p.val" will match "P value", "p.value", "p values" (and even "tempeval", so be careful).

If you give a non-`NULL` address in which to place a table of paren-delimited substrings, I'll return them as a row in the text element of the returned apop_data set. I'll return *all* the matches, filling the first row with substrings from the first application of your regex, then filling the next row with another set of matches (if any), and so on to the end of the string. Useful when parsing a list of items, for example.

string	The string to search (no default)
regex	The regular expression (no default)
substrings	Parens in the regex indicate that I should return matching substrings. Give me the *address* of an apop_data* set, and I will allocate and fill the text portion with matches. Default= `NULL`, meaning do not return substrings (even if parens exist in the regex). If no match, return an empty apop_data set, so `output->textsize[0]==0`.
use_case	Should I be case sensitive, `'y'` or `'n'`? (default = `'n'`, which is not the POSIX default.)

Returns

Count of matches found. 0 == no match. `substrings` may be allocated and filled if needed.

- If `apop_opts.stop_on_warning='n'` returns -1 on error (e.g., regex `NULL` or didn't compile).
- If `strings==NULL`, I return 0—no match—and if `substrings` is provided, set it to `NULL`.

- Here is the test function. Notice that the substring-pulling function call passes `&subs`, not plain `subs`.

```
#include <apop.h>
int main(){
    char string1[] = "Hello. I am a string.";
    assert(apop_regex(string1, "hell"));
    apop_data *subs;
    apop_regex(string1, "(e).*I.*(xxx)*(am)", .substrings = &subs);
    //apop_data_show(subs);
    assert(!strcmp(subs->text[0][0], "e"));
    assert(!strlen(subs->text[0][1])); //The non-match to (xx)* has a zero-length blank
    assert(!strcmp(subs->text[0][2], "am"));
    apop_data_free(subs);

    //Split a comma-delimited list, throwing out white space.
    //Notice that the regex includes only one instance of a non-comma blob
    //ending in a non-space followed by a comma, but the function keeps
    //applying it until the end of string.
    char string2[] = " one, two , three ,four";
    apop_regex(string2, " *([^,]*[^ ]) *(,|$) *", &subs);
    assert(!strcmp(*subs->text[0], "one"));
    assert(!strcmp(*subs->text[1], "two"));
    assert(!strcmp(*subs->text[2], "three"));
    assert(!strcmp(*subs->text[3], "four"));
    apop_data_free(subs);

    //Get a parenthetical. For EREs, \( \) match plain parens in the text.
    char string3[] = " one (but secretly, two)";
    apop_regex(string3, "(\\([^)]*\\))", &subs);
    assert(!strcmp(*subs->text[0], "(but secretly, two)"));
    apop_data_free(subs);

    //NULL input string ==> no-op.
    int match_count = apop_regex(NULL, " *([^,]*[^ ]) *(,|$) *", &subs);
    assert(!match_count);
    assert(!subs);
}
```

- Each set of matches will be one row of the output data. E.g., given the regex `([A-Za-z])([0-9])`, the column zero of `outdata` will hold letters, and column one will hold numbers. Use apop_data_transpose to reverse this so that the letters are in `outdata->text[0]` and numbers in `outdata->text[1]`.

8.2.2.116 gsl_rng* apop_rng_alloc (int seed)

Initialize a `gsl_rng`.

Uses the Tausworth routine.

seed	The seed. No need to get funny with it: 0, 1, and 2 will produce wholly different streams.

Returns

The RNG ready for your use.

- If you are confident that your code is debugged and would like a new stream of values every time your program runs (provided your runs are more than a second apart), seed with the time:

```
#include <apop.h>
#include <time.h>

int main(){
```

```
    apop_opts.rng_seed = time(NULL);
    apop_data_print(
            apop_model_draws(
                apop_model_set_parameters(apop_normal, 0, 1),
                .count=10,
            )
    );
}
```

8.2.2.117 double apop_rng_GHgB3 (gsl_rng * r, double * a)

RNG from a Generalized Hypergeometric type B3.

Devroye uses this as the base for many of his distribution-generators, including the Waring.

- If one of the inputs is <=0, error; return NaN and print a warning.

8.2.2.118 void apop_score (**apop_data** * d, gsl_vector * out, **apop_model** * m)

Find the vector of first derivatives (aka the gradient) of the log likelihood of a data/parametrized model pair.

On input, the model `m` must already be sufficiently prepped that the log likelihood can be evaluated; see p, log_likelihood for details.

On output, the `gsl_vector` input to the function will be filled with the gradients (or `NaN`s on errors). If the model parameters have a more complex shape than a simple vector, then the vector will be in `apop_data_pack` order; use `apop_data_unpack` to reformat to the preferred shape.

d	The apop_data set at which the score is being evaluated.
out	The score to be returned. I expect you to have allocated this already.
m	The parametrized model, which must have either a `log_likelihood` or a `p` method.

- The default is to use apop_numerical_gradient, but special-case calculations for certain models are held in a vtable; see Registering new methods in vtables for details. The typedef new functions must conform to and the hash used for lookups are:

```
1 typedef void (*apop_score_type)(apop_data *d, gsl_vector *gradient, apop_model *m);
2 #define apop_score_hash(m1) ((size_t)((m1).log_likelihood ? (m1).log_likelihood : (m1).p))
```

8.2.2.119 int apop_system (const char * fmt, ...)

Call `system()`, but with `printf`-style arguments. E.g.,

```
1 char filenames[] = "apop_asst.c apop_asst.o"
2 apop_system("ls -l %s", filenames);
```

Returns

The return value of the `system()` call.

8.2.2.120 **apop_data*** apop_t_test (gsl_vector * a, gsl_vector * b)

Answers the question: with what confidence can I say that the means of these two columns of data are different?

If `apop_opts.verbose` is >=1, then display some information to stdout, like the mean/var/count for both vectors and the t statistic.

a	one column of data
b	another column of data

Returns

an apop_data set with the following elements: `mean left - right`: the difference in means; if positive, first vector has larger mean, and one-tailed test is testing $L > R$, else reverse if negative.
`t statistic`: used for the test
`df`: degrees of freedom
`p value, 1 tail`: the p-value for a one-tailed test that one vector mean is greater than the other.
`confidence, 1 tail`: 1- p value.
`p value, 2 tail`: the p-value for the two-tailed test that left mean = right mean.
`confidence, 2 tail`: 1-p value

Example usage:

```
1 gsl_vector *L = apop_query_to_vector("select * from data where sex='M'");
2 gsl_vector *R = apop_query_to_vector("select * from data where sex='F'");
3 apop_data *test_out = apop_t_test(L, R);
4 printf("Reject the null hypothesis of no difference between M and F with %g%% confidence\n",
       apop_data_get(test_out, .rowname="confidence, 2 tail"));
```

See also

apop_paired_t_test, which answers the question: with what confidence can I say that the mean difference between the two columns is zero?

8.2.2.121 int apop_table_exists (char const * name, char remove)

Check for the existence of a table, and maybe delete it.

Recreating a table which already exists can cause errors, so it is good practice to check for existence first. Also, this is the stylish way to delete a table, since just calling `"drop table"` will give you an error if the table doesn't exist.

name	the table name (no default)
remove	'd' ==>delete table so it can be recreated in main. 'n' ==>no action. Return result so program can continue. (default)

Returns

0 = table does not exist
1 = table was found, and if remove=='d', has been deleted -1 = processing error

- In the SQLite engine, this function considers table views to be tables.
- This function uses the Designated initializers syntax for inputs.

8.2.2.122 double apop_test (double statistic, char * distribution, double p1, double p2, char tail)

This is a convenience function to do the lookup of a given statistic along a given distribution. You give me a statistic, its (hypothesized) distribution, and whether to use the upper tail, lower tail, or both. I will return the odds of a Type I error given the model—in statistician jargon, the p-value. [Type I error: odds of rejecting the null hypothesis when it is true.]

For example,

```
1 apop_test(1.3);
```

will return the density of the standard Normal distribution that is more than 1.3 from zero. If this function returns a small value, we can be confident that the statistic is significant. Or,

```
1 apop_test(1.3, "t", 10, .tail='u');
```

will give the appropriate odds for an upper-tailed test using the t-distribution with 10 degrees of freedom (e.g., a t-test of the null hypothesis that the statistic is less than or equal to zero).

Several more distributions are supported; see below.

- For a two-tailed test (the default), this returns the density outside the range. I'll only do this for symmetric distributions.
- For an upper-tail test ('u'), this returns the density above the cutoff
- For a lower-tail test ('l'), this returns the density below the cutoff

statistic	The scalar value to be tested.
distribution	The name of the distribution; see below.
p1	The first parameter for the distribution; see below.
p2	The second parameter for the distribution; see below.
tail	'u' = upper tail; 'l' = lower tail; anything else = two-tailed. (default = two-tailed)

Returns

The odds of a Type I error given the model (the p-value).

Here are the distributions you can use and their parameters.

`"normal"` or `"gaussian"`

- p1= μ, p2= σ
- default (0, 1)

`"lognormal"`

- p1= μ, p2= σ
- default (0, 1)
- Remember, μ and σ refer to the Normal one would get after exponentiation
- One-tailed tests only

`"uniform"`

- p1=lower edge, p2=upper edge
- default (0, 1)
- two-tailed tests are run relative to the center, (p1+p2)/2.

`"t"`

- p1=df
- no default

`"chi squared"`, `"chi"`, `"chisq"`:

- p1=df
- no default
- One-tailed tests only; default='u' (*p*-value for typical cases)

`"f"`

- p1=df1, p2=df2
- no default
- One-tailed tests only

- This function uses the Designated initializers syntax for inputs.

8.2.2.123 **apop_data*** apop_test_anova_independence (**apop_data** * d)

Run a Chi-squared test on an ANOVA table, i.e., an NxN table with the null hypothesis that all cells are equally likely.

d	The input data, which is a crosstab of various elements. They don't have to sum to one.

Returns

A apop_data set including elements named `"chi squared statistic"`, `"df"`, and `"p value"`. Retrieve via, e.g., `apop_data_get(out, .rowname="p value")`.

See also

apop_test_fisher_exact

8.2.2.124 **apop_data*** apop_test_fisher_exact (**apop_data** * intab)

Run the Fisher exact test on an input contingency table.

Returns

An apop_data set with two rows:
"probability of table": Probability of the observed table for fixed marginal totals.
"p value": Table p-value. The probability of a more extreme table, where 'extreme' is in a probabilistic sense.

- If there are processing errors, these values will be NaN.

out->error=='p'	Processing error in the test.

For example:

```
#include <apop.h>

int main() {
    /* This test is thanks to Nick Eriksson, who sent it to me in the form of a bug report. */
    apop_data * testdata = apop_data_falloc((2, 3),
                                30, 50, 45,
                                34, 12, 17 );
    apop_data * t2 = apop_test_fisher_exact(testdata);
    assert(fabs(apop_data_get(t2,.rowname="p value") - 0.0001761) < 1e-6);
}
```

8.2.2.125 **apop_data*** apop_test_kolmogorov (**apop_model** * m1, **apop_model** * m2)

Run the Kolmogorov-Smirnov test to determine whether two distributions are identical.

m1	A sorted PMF model. I.e., a model estimated via something like `apop_model *m1 = apop_estimate(apop_data_sort(input_data), apop_pmf);`
m2	Another apop_model. If it is a PMF, then I will use a two-sample test, which is different from the one-sample test used if this is not a PMF.

Returns

An apop_data set including the p-value from the Kolmogorov-Smirnov test that the two distributions are equal.

out->error='m'	Model error: `m1` is not an apop_pmf. I verify this by checking whether `m1->cdf == apop_pmf->cdf`.

- If you are using a apop_pmf model, the data set(s) must be sorted before you set up the model, as per the example below. See apop_data_sort and the discussion of CDFs in the apop_pmf documentation. If you don't do this, the test will almost certainly reject the null hypothesis that `m1` and `m2` are identical. A future version of Apophenia may implement a mechanism to allow this function to test for sorted data, but it currently can't.

Here is an example, which tests whether a set of draws from a Normal(0, 1) matches a sequence of Normal distributions with increasing mean.

```
#include <apop.h>
//This program finds the p-value of a K-S test between
//500 draws from a N(0, 1) and a N(x, 1), where x grows from 0 to 1.

apop_model * model_to_pmfs(apop_model *m1, int size){
    apop_data *outd1 = apop_model_draws(m1, size);
    return apop_estimate(apop_data_sort(outd1), apop_pmf);
}

int main(){
    apop_model *n1 = apop_model_set_parameters(apop_normal, 0, 1);
    apop_model *pmf1 = model_to_pmfs(n1, 5e2);
    apop_data *ktest;

    //first, there should be zero divergence between a PMF and itself:
    apop_model *pmf2 = apop_model_copy(pmf1);
    ktest = apop_test_kolmogorov(pmf1, pmf2);
    double pval = apop_data_get(ktest, .rowname="p value, 2 tail");
    assert(pval > .999);

    //as the mean m drifts, the pval for a comparison
    //between a N(0, 1) and N(m, 1) gets smaller.
    printf("mean\tpval\n");
```

```
    double prior_pval = 18;
    for(double i=0; i<= .6; i+=0.2){
        apop_model *n11 = apop_model_set_parameters(apop_normal, i, 1);
        ktest = apop_test_kolmogorov(pmf1, n11);
        apop_data_print(ktest, NULL);
        double pval = apop_data_get(ktest, .rowname="p value, 2 tail");
        assert(pval < prior_pval);
        printf("%g\t%g\n", i, pval);
        prior_pval = pval;
    }
    apop_model_free(pmf1);
}
```

8.2.2.126 **apop_data*** apop_text_alloc (**apop_data** * in, const size_t row, const size_t col)

This allocates or resizes the `text` element of an apop_data set.

If the `text` element already exists, then this is effectively a `realloc` function, reshaping to the size you specify.

in	An apop_data set. It's OK to send in `NULL`, in which case an apop_data set with `NULL` `matrix` and `vector` elements is returned.
row	the number of rows of text.
col	the number of columns of text.

Returns

A pointer to the relevant apop_data set. If the input was not `NULL`, then this is a repeat of the input pointer.

out->error=='a'	Allocation error.

8.2.2.127 void apop_text_free (char *** freeme, int rows, int cols)

Free a matrix of chars* (i.e., a char***). This is what `apop_data_free` uses internally to deallocate the `text` element of an apop_data set. You may never need to use it directly.

Sample usage:

```
1 apop_text_free(yourdata->text, yourdata->textsize[0], yourdata->textsize[1]);
```

8.2.2.128 char* apop_text_paste (**apop_data** const * strings, char * between, char * before, char * after, char * between_cols, apop_fn_riip prune, void * prune_parameter)

Join together the `text` grid of an apop_data set into a single string.

For example, say that we have a data set with some text: row 0 has `"a0"`, `"b0"`, `"c0"`; row 2 has `"a1"`, `"b1"`, `"c1"`; and so on. We would like to produce

```
1 insert into tab values ('a0', 'b0', 'c0');
2 insert into tab values ('a1', 'b1', 'c1');
3 ...
```

This could be sent to an SQL engine to copy the data to a database (but this is just an example for demonstration—use apop_data_print to write to a database table).

To construct this single string from the text grid, we would need to add:

- before the text, `Insert into tab values ('`.

- between each element on a row: `', '`
- between rows: `'); \ninsert into tab values('`
- at the tail end: `');'`

Thus, do the conversion via:

```
char *insert_string = apop_text_paste(indata,
    .before="Insert into tab values ('",
    .between="', '",
    .between_cols="'); \\ninsert into tab values(',
    .after="');'"
);
```

strings	An apop_data set with a grid of text to be combined into a single string
between	The text to put in between the rows of the table, such as ", ". (Default is a single space: " ")
before	The text to put at the head of the string. For the query example, this would be .before="select ". (Default: NULL)
after	The text to put at the tail of the string. For the query example, .after=" from data_table". (Default: NULL)
between_cols	The text to insert between columns of text. See below for an example (Default is set to equal .between)

prune	If you don't want to use the entire text set, you can provide a function to indicate which elements should be pruned out. Some examples:
	`//Just use column 3`
	`int is_not_col_3(apop_data *indata, int row, int col, void *ignore){`
	`  return col!=3;`
	`}`
	
	`//Jump over blanks as if they don't exist.`
	`int is_blank(apop_data *indata, int row, int col, void *ignore){`
	`  return strlen(indata->text[row][col])==0;`
	`}`

prune_parameter	A void pointer to pass to your prune function.

Returns

A single string with the elements of the strings table joined as per your specification. Allocated by the function, to be freed by you if desired.

- If the table of strings is NULL or has no text, the output string will have only the .before and .after parts with nothing in between.
- if apop_opts.verbose >=3, then print the pasted text to stderr.
- It is sometimes useful to use Apop_r and Apop_rs to get a view of only one or a few rows in conjunction with this function.
- This function uses the Designated initializers syntax for inputs.

This sample snippet generates the SQL for a query using a list of column names (where the query begins with select , ends with from datatab, and has commas

in between each element), re-processes the same list to produce the head of an HTML table, then produces the body of the table with the query result.

```
#include <apop.h>

int main(){
    apop_query("create table datatab(name, age, sex);"
                "insert into datatab values ('Alex', 23, 'm');"
                "insert into datatab values ('Alex', 32, 'f');"
                "insert into datatab values ('Michael', 41, 'f');"
                "insert into datatab values ('Michael', 14, 'm');");

    apop_data *cols = apop_text_alloc(NULL, 3, 1);
    apop_text_set(cols, 0, 0, "name");
    apop_text_set(cols, 1, 0, "age");
    apop_text_set(cols, 2, 0, "sex");
    char *query= apop_text_paste(cols, .before="select ", .between=", ");
    apop_data *d = apop_query_to_text("%s from datatab", query);
    char *html_head = apop_text_paste(cols, .before="<table><tr><td>",
                                .between="</td><td>", .after="</tr>\n<tr><td>");
    char *html_table = apop_text_paste(d, .before=html_head, .after="</td></tr></table>\n",
                                .between="</tr>\n<tr><td>", .between_cols="</td><td>");
    FILE *outfile = fopen("yourdata.html", "w");
    fprintf(outfile, "%s", html_table);
    fclose(outfile);
}
```

8.2.2.129 int apop_text_set (**apop_data** * in, const size_t row, const size_t col, const char * fmt, ...)

Add a string to the text element of an apop_data set. If you send me a NULL string, I will write the value of apop_opts.nan_string in the given slot. If there is already something in that slot, that string is freed, preventing memory leaks.

in	The apop_data set, that already has an allocated text element.
row	The row
col	The column
fmt	The text to write.
...	You can use a printf-style fmt and follow it with the usual variables to fill in.

Returns

0=OK, -1=error (probably out-of-bounds)

- UTF-8 or ASCII text is correctly handled.
- Apophenia follows a general rule of not reallocating behind your back: if your text matrix is currently of size (3,3) and you try to put an item in slot (4,4), then I display an error rather than reallocating the text matrix.
- The string added is a copy (via asprintf), not a pointer to the input(s).
- If there had been a string at the grid point you are writing to, the old one is freed to prevent leaks. Remember this if you had other pointers aliasing that string.
- If an element is NULL, write apop_opts.nan_string at that point. You may prefer to use "" to express a blank.
- apop_text_alloc will reallocate to a new size if you need. For example, this code will fill the diagonals of the text array with a message, resizing as it goes:

```
1 apop_data *list = (something already allocated.);
2 for (int n=0; n < 10; n++){
3     apop_text_alloc(list, n+1, n+1);
4     apop_text_set(list, n, n, "This is cell (%i, %i)", n, n);
5 }
```

8.2.2.130 **apop_data*** apop_text_to_data (char const * text_file, int has_row_names, int has_col_names, int const * field_ends, char const * delimiters)

Read a delimited or fixed-wisdth text file into the matrix element of an apop_data set.

See Input text file formatting.

See also apop_text_to_db, which handles text data, and may othewise be a perferable approach to data management.

text_file	= "-" The name of the text file to be read in. If "-" (the default), use stdin.
has_row_names	Does the lines of data have row names? 'y' =yes; 'n' =no (default: 'n')
has_col_names	Is the top line a list of column names? See Input text file formatting for notes on dimension (default: 'y')
field_ends	If fields have a fixed size, give the end of each field, e.g. .field_ends=(int[]){3, 8 11}. (default: NULL, indicating not fixed width)
delimiters	A string listing the characters that delimit fields. (default: "|,\t")

Returns

Returns an apop_data set.

out->error=='a'	allocation error
out->error=='t'	text-reading error

example: See apop_ols.

- This function uses the Designated initializers syntax for inputs.

8.2.2.131 int apop_text_to_db (char const * text_file, char * tabname, int has_row_names, int has_col_names, char ** field_names, int const * field_ends, **apop_data** * field_params, char * table_params, char const * delimiters, char if_table_exists)

Read a delimited or fixed-width text file into a database table. See Input text file formatting.

For purely numeric data, you may be able to bypass the database by using apop_text_to_data.

See the apop_ols page for an example that uses this function to read in sample data (also listed on that page).

Apophenia ships with an apop_text_to_db command-line utility, which is a wrapper for this function.

Especially if you are using a pre-2007 version of SQLite, there may be a speedup to putting this function in a begin/commit wrapper:

```
1 apop_query("begin;");
2 apop_data_print(dataset, .output_name="dbtab", .output_type='d');
3 apop_query("commit;");
```

text_file	The name of the text file to be read in. If "-", then read from STDIN. (default: "-")

tabname	The name to give the table in the database (default: `text_file` up to the first dot, e.g., `text_file=="pant_lengths.csv"` gives `tabname=="pant_lengths"`)
has_row_↩ names	Does the lines of data have row names? (default: 0)
has_col_names	Is the top line a list of column names? (default: 1)
field_names	The list of field names, which will be the columns for the table. If `has_col_names==1`, read the names from the file (and just set this to `NULL`). If has_col_names == 1 && field_names !=NULL, I'll use the field names. (default: NULL)
field_ends	If fields have a fixed size, give the end of each field, e.g. `.field_ends=(int[]){3, 8 11}`. (default: `NULL`, indicating not fixed width)
field_params	There is an implicit `create table` in setting up the database. If you want to add a type, constraint, or key, put that here. The relevant part of the input apop_data set is the `text` grid, which should be $N \times 2$. The first item in each row (`your_↩ params->text[n][0]`, for each n) is a regular expression to match against the variable names; the second item (`your_params->text[n][1]`) is the type, constraint, and/or key (i.e., what comes after the name in the `create` query). Not all variables need be mentioned; the default type if nothing matches is `numeric`. I go in order until I find a regex that matches the given field, so if you don't like the default, then set the last row to have name `.*`, which is a regex guaranteed to match anything that wasn't matched by an earlier row, and then set the associated type to your preferred default. See apop_regex on details of matching. (default: NULL)
table_params	There is an implicit `create table` in setting up the database. If you want to add a table constraint or key, such as `not null primary key (age, sex)`, put that here.
delimiters	A string listing the characters that delimit fields. default = `"|,\t"`
if_table_exists	What should I do if the table exists? `'n'` Do nothing; exit this function. (default) `'d'` Retain the table but delete all data; refill with the new data (i.e., call `"delete * from your_table"`). `'o'` Overwrite the table from scratch; deleting the previous table entirely. `'a'` Append new data to the existing table.

Returns

Returns the number of rows on success, -1 on error.

- This function uses the Designated initializers syntax for inputs.

8.2.2.132 **apop_data*** apop_text_unique_elements (const **apop_data** * d, size_t col)

Give me a column of text, and I'll give you a sorted list of the unique elements. This is basically running `select distinct * from datacolumn`, but without the aid of the database.

d	An apop_data set with a text component
col	The text column you want me to use.

Returns

An apop_data set with a single sorted column of text, where each unique text input appears once.

See also

apop_vector_unique_elements

8.2.2.133 **apop_model*** apop_update (**apop_data** * data, **apop_model** * prior, **apop_model** * likelihood, gsl_rng * rng)

Take in a prior and likelihood distribution, and output a posterior distribution.

- This function first checks a table of conjugate distributions for the pair you sent in. If the models are listed on the table, then the function returns a corresponding closed-form model with updated parameters.

- If the parameters aren't in the table of conjugate, and the prior distribution has a `p` or `log_likelihood` element, then use apop_model_metropolis to generate the posterior. If you expect MCMC to run, you may add an apop_mcmc_settings group to your prior to control the details of the search. See also the apop_model_metropolis documentation.

- If the prior does not have a `p` or `log_likelihood` but does have a `draw` element, then make draws from the prior and weight them by the `p` given by the likelihood distribution. This is not a rejection sampling method, so the burnin is ignored.

data	The input data, that will be used by the likelihood function (default = `NULL`.)
prior	The prior apop_model. If the system needs to estimate the posterior via MCMC, this needs to have a `log_likelihood` or `p` method. (No default, must not be `NULL`.)
likelihood	The likelihood apop_model. If the system needs to estimate the posterior via MCMC, this needs to have a `log_likelihood` or `p` method (ll preferred). (No default, must not be `NULL`.)
rng	A `gsl_rng`, already initialized (e.g., via apop_rng_alloc). (default: an RNG from apop_rng_get_thread)

Returns

an apop_model struct representing the posterior, with updated parameters.

- In all cases, the output is a apop_model that can be used as the input to this function, so you can chain Bayesian updating procedures.

- Here are the conjugate distributions currently defined:

Prior	Likelihood	Notes
Beta	Binomial	
Beta	Bernoulli	
Exponential	Gamma	Gamma likelihood represents the distribution of λ^{-1}, not plain λ
Normal	Normal	Assumes prior with fixed σ; updates distribution for μ
Gamma	Poisson	Uses sum and size of the data

Here is a test function that compares the output via conjugate table and via Metropolis-Hastings sampling:

```
#include <apop.h>

//For the test suite.
void distances(gsl_vector *v1, gsl_vector *v2, double tol){
    double error = apop_vector_distance(v1, v2, .metric='m');
    double updated_size = apop_vector_sum(v1);
    Apop_stopif(error/updated_size > tol, exit(1), 0, "The error is %g, which is too big.",
      error/updated_size);
```

```
}

int main(){
    double binom_start = 0.6;
    double beta_start_a = 0.3;
    double beta_start_b = 0.5;
    double n = 4000;
    //First, the easy estimation using the conjugate distribution table.
    apop_model *bin = apop_model_set_parameters(apop_binomial, n, binom_start);
    apop_model *beta = apop_model_set_parameters(apop_beta, beta_start_a, beta_start_b);
    apop_model *updated = apop_update(.prior= beta, .likelihood=bin);

    //Now estimate via MCMC.
    //Requires a one-parameter binomial, with n fixed,
    //and a data set of n data points with the right p.
    apop_model *bcopy = apop_model_set_parameters(apop_binomial, n, GSL_NAN);
    apop_data *bin_draws = apop_data_falloc((1,2), n*(1-binom_start), n*
      binom_start);
    bin = apop_model_fix_params(bcopy);
    Apop_settings_add_group(beta, apop_mcmc, .burnin=.2, .periods=1e5);

    apop_model *out_h = apop_update(bin_draws, beta, bin, NULL);
    apop_model *out_beta = apop_estimate(out_h->data, apop_beta);

    //Finally, we can compare the conjugate and Gibbs results:
    distances(updated->parameters->vector, out_beta->parameters->vector, 0.01);

    //The apop_update function used apop_model_metropolis to generate
    //a batch of draws, so the draw method for out_h is apop_model_metropolis_draw.
    //So, here we make more draws using metropolis, and compare the beta
    //distribution that fits to those draws to the beta distribution output above.
    int draws = 1.3e5;
    apop_data *d = apop_model_draws(out_h, draws);
    apop_model *drawn = apop_estimate(d, apop_beta);
    distances(updated->parameters->vector, drawn->parameters->vector, 0.02);
}
```

- The conjugate table is stored using a vtable; see Registering new methods in vtables for details. If you are writing a new vtable entry, the typedef new functions must conform to and the hash used for lookups are:

```
1 typedef apop_model *(*apop_update_type)(apop_data *, apop_model , apop_model);
2 #define apop_update_hash(m1, m2) ((size_t)(m1).draw + (size_t)((m2).log_likelihood ? (m2).log_likelihood :
      (m2).p)*33)
```

- This function uses the Designated initializers syntax for inputs.

8.2.2.134 void apop_vector_apply (gsl_vector * v, void(*)(double *) fn)

Apply a function to every row of a matrix. The function that you input takes in a `double*` and may modify the input value in place. This function will send a pointer to each element of your vector to your function.

v	The input vector
fn	A function of the form `void fn(double in)`

- If the vector is `NULL`, this is a no-op.
- See the map/apply page for details.

See also

apop_map

8.2.2.135 int apop_vector_bounded (const gsl_vector * in, long double max)

Test that all elements of a vector are within bounds, so you can preempt a procedure that is about to break on infinite or too-large values.

in	A `gsl_vector`
max	An upper and lower bound to the elements of the vector. (default: INFINITY)

Returns

1 if everything is bounded: not Inf, -Inf, or NaN, and $-\max < x < \max$;
0 otherwise.

- A `NULL` vector has no unbounded elements, so `NULL` input returns 1. You get a warning if `apop_opts.verbosity >=2`.
- This function uses the Designated initializers syntax for inputs.

8.2.2.136 gsl_vector* apop_vector_copy (const gsl_vector * in)

Copy one `gsl_vector` to another. That is, all data is duplicated. Unlike `gsl_vector_memcpy`, this function allocates and returns the destination, so you can use it like this:

```
1 gsl_vector *a_copy = apop_vector_copy(original);
```

in	The input vector

Returns

A structure that this function will allocate and fill. If `gsl_vector_alloc` fails, returns `NULL` and print a warning.

8.2.2.137 double apop_vector_correlation (const gsl_vector * ina, const gsl_vector * inb, const gsl_vector * weights)

Returns the correlation coefficient of two vectors: $\frac{\text{cov}(a,b)}{\sqrt{\text{var}(a)}\sqrt{\text{var}(b)}}$.

An example

```
1 gsl_matrix *m = apop_text_to_data("indata")->matrix;
2 printf("The correlation coefficient between rows two "
3        "and three is %g.\n", apop_vector_correlation(Apop_mrv(m, 2), Apop_mrv(m, 3)));
```

ina,inb	Two vectors of equal length (no default, must not be NULL)
weights	Replicate weights for the observations. (default: equal weights for all observations)

- This function uses the Designated initializers syntax for inputs.

8.2.2.138 double apop_vector_cov (const gsl_vector * v1, const gsl_vector * v2, const gsl_vector * weights)

Find the sample covariance of a pair of vectors, with an optional weighting. This only makes sense if the weightings are identical, so the function takes only one weighting vector for both.

v1,v2	The data vectors (no default; must not be `NULL`)
weights	The weight vector. (default equal weights for all elements)

Returns

The sample covariance

- This function uses the Designated initializers syntax for inputs.

8.2.2.139 double apop_vector_distance (const gsl_vector * ina, const gsl_vector * inb, const char metric, const double norm)

Returns the distance between two vectors, where distance is defined based on the third (optional) parameter:

- 'e' (the default): scalar distance (standard Euclidean metric) between two vectors. $\sqrt{\sum_i (a_i - b_i)^2}$, where i iterates over dimensions.
- 'm' Returns the Manhattan metric distance between two vectors: $\sum_i |a_i - b_i|$, where i iterates over dimensions.
- 'd' The discrete norm: if $a = b$, return zero, else return one.
- 's' The sup norm: find the dimension where $|a_i - b_i|$ is largest, return the distance along that one dimension.
- 'l' or 'L' The L_p norm, $\left(\sum_i |a_i - b_i|^2\right)^{1/p}$. The value of p is set by the fourth (optional) argument.

ina	First vector (No default, must not be NULL)
inb	Second vector (Default = zero)
metric	The type of metric, as above.
norm	If you are using an L_p norm, this is p. Must be strictly greater than zero. (default = 2)

- The defaults are such that

```
1 apop_vector_distance(v);
2 apop_vector_distance(v, .metric = 's');
3 apop_vector_distance(v, .metric = 'm');
```

  gives you the standard Euclidean length of v, its longest element, and its sum.

- This function uses the Designated initializers syntax for inputs.

```
#include <apop.h>

/* Test distance calculations using a 3-4-5 triangle */
int main(){
    gsl_vector *v1 = gsl_vector_alloc(2);
    gsl_vector *v2 = gsl_vector_alloc(2);
    apop_vector_fill(v1, 2, 2);
    apop_vector_fill(v2, 5, 6);

    assert(apop_vector_distance(v1, v1, 'd') == 0);
    assert(apop_vector_distance(v1, v2, 'd') == 1);
    assert(apop_vector_distance(v1, .metric='m') == 4);
    assert(apop_vector_distance(v2, .metric='s') == 6);
    assert(apop_vector_distance(v1,v2) == 5.); //the hypotenuse of the 3-4-5 triangle
    assert(apop_vector_distance(v1,v2, 'm') == 7.);
    assert(apop_vector_distance(v1,v2, 'L', 2) == 5.);  //L_2 norm == Euclidean
}
```

8.2.2.140 long double apop_vector_entropy (gsl_vector * in)

Given a vector representing a probability distribution of observations, calculate the entropy, $\sum_i -\ln(v_i)v_i$.

- You may input a vector giving frequencies (normalized to sum to one) or counts (arbitrary sum).

- The entropy of a data set depends only on the frequency with which elements are observed, not the value of the elements themselves. The apop_data_pmf_compress function will reduce an input apop_data set to one weighted line per observation, and the weights would determine the entropy:

```
1 apop_data *data = apop_text_to_data("indata");
2 apop_data_pmf_compress(data);
3 data_entropy = apop_vector_entropy(d->weights);
```

- The entropy is calculated using natural logs. To convert to base 2, divide by ln(2); see the example.
- The entropy of an empty data set (`NULL` or a total weight of zero) is zero. Print a warning when given `NULL` input and `apop_opts.verbose >=1`.
- If the input vector has negative elements, return `NaN`; print a warning when `apop_opts.verbose >= 0`.

Sample code:

```
#include <apop.h>

#define Diff(left, right, eps) Apop_stopif(fabs((left)-(right))>(eps), abort(), 0, "%g is too different
       from %g (abitrary limit=%g).", (double)(left), (double)(right), eps)

long double entropy_base_2(gsl_vector *x) {
    return apop_vector_entropy(x)/log(2);
}

int main(){
    apop_model *flip = apop_model_set_parameters(apop_bernoulli, .5);

    //zero data => entropy zero
    gsl_vector *v = gsl_vector_calloc(1);
    assert(apop_vector_entropy(v) == 0);

    //negative data => NaN
    gsl_vector_set(v, 0, -1);
    int v1 = apop_opts.verbose;
    apop_opts.verbose = -1;
    assert(isnan(apop_vector_entropy(v)));
    apop_opts.verbose = v1;

    //N equiprobable bins => entropy = log(N)
    v = apop_vector_realloc(v, 100);
    gsl_vector_set_all(v, 1./100);
    Diff(log(100), apop_vector_entropy(v), 1e-5);

    //Normalization is optional. You may send a vector of counts.
    gsl_vector_set_all(v, 1);
    Diff(log(100), apop_vector_entropy(v), 1e-5);

    //flip two coins.
    apop_data *coin_flips = apop_model_draws(flip, .count=10000);
    apop_data *c2         = apop_model_draws(flip, .count=10000);
    apop_data_stack(c2, coin_flips, 'c', .inplace='y');

    //entropy of one coin flip in base2 == 1
    apop_data_pmf_compress(coin_flips);
    Diff(entropy_base_2(coin_flips->weights), 1, 1e-3);

    //entropy of two coin flips in base2 == 2
    apop_data_pmf_compress(c2);
    Diff(entropy_base_2(c2->weights), 2, 1e-3);

    //flip three coins, via model cross products
    Diff(entropy_base_2(apop_data_pmf_compress(apop_model_draws(
            apop_model_cross(flip, flip, flip) ,.count=10000))->weights), 3, 1e-3);

    apop_data_free(coin_flips);
```

```
    apop_data_free(c2);
    gsl_vector_free(v);
}
```

8.2.2.141 void apop_vector_exp (gsl_vector * v)

Replace every vector element v_i with exp (v_i).

- If the input vector is `NULL`, do nothing.

8.2.2.142 double apop_vector_kurtosis (const gsl_vector * in)

Returns the sample fourth central moment of the data in the given vector. Corrections are made to produce an unbiased result as per Appendix M (PDF) of *Modeling with data*.

- This is an estimate of the fourth central moment without normalization. The kurtosis of a $\mathcal{N}(0,1)$ is $3\sigma^4$, not three, one, or zero.

  See also

  apop_vector_kurtosis_pop

8.2.2.143 double apop_vector_kurtosis_pop (gsl_vector const * v, gsl_vector const * weights)

Returns the population fourth central moment [$\sum_i (x_i - \mu)^4/n$)] of the data in the given vector, with an optional weighting.

v	The data vector
weights	The weight vector. If NULL, assume equal weights.

Returns

The weighted kurtosis.

- Some people like to normalize the fourth central moment by dividing by variance squared, or by subtracting three; those things are not done here, so you'll have to do them separately if desired.
- This function uses the Designated initializers syntax for inputs.

  See also

  apop_vector_kurtosis for the unbiased sample version.

8.2.2.144 void apop_vector_log (gsl_vector * v)

Replace every vector element v_i with ln (v_i).

- If the input vector is `NULL`, do nothing.

8.2.2.145 void apop_vector_log10 (gsl_vector * v)

Replace every vector element v_i with log $_{10}(v_i)$.

- If the input vector is `NULL`, do nothing.

8.2.2.146 gsl_vector* apop_vector_map (const gsl_vector * v, double(*)(double) fn)

Map a function onto every element of a vector. Thus function will send each element to the function you provide, and will output a `gsl_vector` holding your function's output for each row.

v	The input vector
fn	A function of the form `double fn(double in)`

Returns

A `gsl_vector` (allocated by this function) with the corresponding value for each row.

- If you input a `NULL` vector, I return `NULL`.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.147 double apop_vector_map_sum (const gsl_vector * in, double(*)(double) fn)

Returns the sum of the output of `apop_vector_map`. For example, `apop_vector_map_sum(v, isnan)` returns the count of elements of `v` that are `NaN`.

- If you input a `NULL` vector, I return the sum of zero items: zero.
- See the map/apply page for details.

See also

apop_map, apop_map_sum

8.2.2.148 double apop_vector_mean (gsl_vector const * v, gsl_vector const * weights)

Find the mean, weighted or unweighted.

v	The data vector
weights	The weight vector. Default: assume equal weights.

Returns

The weighted mean

- This function uses the Designated initializers syntax for inputs.

8.2.2.149 gsl_vector* apop_vector_moving_average (gsl_vector * v, size_t bandwidth)

Return a new vector that is the moving average of the input vector.

v	The input vector, unsmoothed
bandwidth	An integer ≥ 1 giving the number of elements to be averaged to produce one number.

Returns

A smoothed vector of size `v->size - (bandwidth/2)*2`.

8.2.2.150 void apop_vector_normalize (gsl_vector * in, gsl_vector ** out, const char normalization_type)

This function will normalize a vector, either such that it has mean zero and variance one, or ranges between zero and one, or sums to one.

in	A `gsl_vector` with the un-normalized data. `NULL` input gives `NULL` output. (No default)
out	If normalizing in place, `NULL`. If not, the address of a `gsl_vector*`. Do not allocate. (default = `NULL`.)
normalization_type	`'p'`: normalized vector will sum to one. E.g., start with a set of observations in bins, end with the percentage of observations in each bin. (the default) `'r'`: normalized vector will range between zero and one. Replace each X with (X-min) / (max - min). `'s'`: normalized vector will have mean zero and (sample) variance one. Replace each X with $(X - \mu)/\sigma$, where σ is the sample standard deviation. `'m'`: normalize to mean zero: Replace each X with $(X - \mu)$

Example

- This function uses the Designated initializers syntax for inputs.

8.2.2.151 double* apop_vector_percentiles (gsl_vector * data, char rounding)

Returns an array of size 101, where `returned_vector`[95] gives the value of the 95th percentile, for example. `Returned_vector`[100] is always the maximum value, and `returned_vector`[0] is always the min (regardless of rounding rule).

data	A `gsl_vector` with the data. (No default, must not be `NULL`.)
rounding	Either be `'u'`, `'d'`, or `'a'`. Unless your data is exactly a multiple of 101, some percentiles will be ambiguous. If `'u'`, then round up (use the next highest value); if `'d'`, round down to the next lowest value; if `'a'`, take the mean of the two nearest points. (Default = `'d'`.)

- If the rounding method is `'u'` or `'a'`, then you can say "5% or more of the sample is below returned_vector[5]"; if `'d'` or `'a'`, then you can say "5% or more of the sample is above returned_vector[5]".
- You may eventually want to `free()` the array returned by this function.
- This function uses the Designated initializers syntax for inputs.

8.2.2.152 void apop_vector_print (gsl_vector * data, Output_declares)

Print a vector to the screen, a file, a pipe, or the database.

- See apop_prep_output for more on how printing settings are set.
- For example, the default for apop_opts.output_delimiter is a tab, which puts the vector on one line, but `apop_opts.output_type="\n"` would print the vector vertically.
- See also Legible output for more details and examples.
- This function uses the Designated initializers syntax for inputs.

8.2.2.153 gsl_vector* apop_vector_realloc (gsl_vector * v, size_t newheight)

This function will resize a `gsl_vector` to a new length.

Data in the vector will be retained. If the new height is smaller than the old, then data at the end of the vector will be cropped away (in a non–memory-leaking manner). If the new height is larger than the old, then new cells will be filled with garbage; it is your responsibility to zero out or otherwise fill them before use.

- A large number of reallocs can take a noticeable amount of time. You are thus encouraged to make an effort to determine the size of your data and do one allocation, rather than writing for loops that resize a vector at every increment.

- The gsl_vector is a versatile struct that can represent subvectors, matrix columns and other cuts from parent data. Resizing a portion of a parent matrix makes no sense, so return NULL and print an error if asked to resize a view.

v	The already-allocated vector to resize. If you give me NULL, this is equivalent to gsl_vector_alloc
newheight	The height you'd like the vector to be.

Returns

v, now resized

8.2.2.154 double apop_vector_skew (const gsl_vector * in)

Returns an unbiased estimate of the sample skew of the data in the given vector.

8.2.2.155 double apop_vector_skew_pop (gsl_vector const * v, gsl_vector const * weights)

Returns the population skew ($\sum_i (x_i - \mu)^3/n$)) of the data in the given vector. Observations may be weighted.

v	The data vector
weights	The weight vector. Default: equal weights for all observations.

Returns

The weighted skew.

- Some people like to normalize the skew by dividing by (variance) $^{3/2}$; that's not done here, so you'll have to do so separately if need be.

- Apophenia tries to be smart about reading the weights. If weights sum to one, then the system uses w->size as the number of elements, and returns the usual sum over $n-1$. If weights > 1, then the system uses the total weights as n. Thus, you can use the weights as standard weightings or to represent elements that appear repeatedly.

8.2.2.156 gsl_vector* apop_vector_stack (gsl_vector * v1, gsl_vector const * v2, char inplace)

Put the first vector on top of the second vector.

v1	the upper vector (default=NULL, in which case this copies v2)
v2	the second vector (default=NULL, in which case nothing is added)
inplace	If 'y', use apop_vector_realloc to modify v1 in place; see the caveats on that function. Otherwise, allocate a new vector, leaving v1 undisturbed. (default='n')

Returns

the stacked data, either in a new vector or a pointer to v1.

- This function uses the Designated initializers syntax for inputs.

8.2.2.157 long double apop_vector_sum (const gsl_vector * in)

Returns the sum of the data in the given vector.

8.2.2.158 gsl_matrix* apop_vector_to_matrix (const gsl_vector * in, char row_col)

This function copies the data in a vector to a new one-column (or one-row) matrix and returns the newly-allocated and filled matrix.

For the reverse, try apop_data_pack.

in	a `gsl_vector` (No default. If `NULL`, I return `NULL`, with a warning if `apop_opts.verbose >=1`)
row_col	If `'r'`, then this will be a row (1 x N) instead of the default, a column (N x 1). (default: `'c'`)

Returns

a newly-allocated `gsl_matrix` with one column (or row).

- If you send in a `NULL` vector, you get a `NULL` pointer in return. I warn you of this if `apop_opts.verbosity >=2` .
- If `gsl_matrix_alloc` fails you get a `NULL` pointer in return.
- This function uses the Designated initializers syntax for inputs.

8.2.2.159 gsl_vector* apop_vector_unique_elements (const gsl_vector * v)

Give me a vector of numbers, and I'll give you a sorted list of the unique elements. This is basically running `select distinct datacol from data order by datacol`, but without the aid of the database.

v	a vector of items

Returns

a sorted vector of the distinct elements that appear in the input.

- NaNs (if any) appear at the end of the sort order.

See also

apop_text_unique_elements

8.2.2.160 double apop_vector_var (gsl_vector const * v, gsl_vector const * weights)

Find the sample variance of a vector, weighted or unweighted.

v	The data vector
weights	The weight vector. If NULL (the default), assume equal weights.

Returns

The weighted sample variance.

- This uses (n-1) in the denominator of the sum; i.e., it corrects for the bias introduced by using $\bar{x}$ instead of μ.

- Multiply the output by (n-1)/n if you need population variance.
- Apophenia tries to be smart about reading the weights. If weights sum to one, then the system uses `w->size` as the number of elements, and returns the usual sum over $n-1$. If weights > 1, then the system uses the total weights as n. Thus, you can use the weights as standard weightings or to represent elements that appear repeatedly.
- This function uses the Designated initializers syntax for inputs.

  See also

  apop_vector_var_m for the case where you already have the vector's mean.

8.2.2.161 double apop_vector_var_m (const gsl_vector * in, const double mean)

Returns the variance of the data in the given vector, given that you've already calculated the mean.

in	the vector in question
mean	the mean, which you've already calculated using apop_vector_mean.

See also

apop_vector_var

8.2.3 Variable Documentation

8.2.3.1 **apop_opts_type** apop_opts

Here are where the options are initially set. See the apop_opts_type documentation for details.

9 Data Structure Documentation

9.1 apop_arms_settings Struct Reference

Data Fields

- double convex
- char do_metro
- apop_model * model
- int neval
- int ninit
- int npoint
- arms_state * **state**
- double * xinit
- double xl
- double xprev
- double xr

Detailed Description

For use with apop_arms_draw, to perform derivative-free adaptive rejection sampling with metropolis step.

That function generates default values for this struct if you do not attach one to the model beforehand, via a form like `apop_model_add_group(your_model, apop_arms, .model=your_model, .xl=8, .xr =14);`. If you initialize it manually via apop_settings_add_group, the `model` element is mandatory; you'll get a run-time complaint if you forget it.

Field Documentation

9.1.0.1 double apop_arms_settings::convex

Adjustment for convexity

9.1.0.2 char apop_arms_settings::do_metro

Set to `'y'` if the metropolis step is required (i.e., if you're not sure if the function is log-concave).

9.1.0.3 **apop_model*** apop_arms_settings::model

The model from which to draw. Mandatory. Must have either a `log_likelihood` or `p` method.

9.1.0.4 int apop_arms_settings::neval

On exit, the number of function evaluations performed

9.1.0.5 int apop_arms_settings::ninit

The length of `xinit`.

9.1.0.6 int apop_arms_settings::npoint

Maximum number of envelope points. I `malloc` space for this many `doubles` at the outset. Default = 1e5.

9.1.0.7 double* apop_arms_settings::xinit

A `double*` giving starting values for x in ascending order, e.g., `(double *){1, 10, 100}`. . Default: -1, 0, 1. If this isn't `NULL`, I need at least three items, and the length in `ninit`.

9.1.0.8 double apop_arms_settings::xl

Left bound. If you don't give me one, I'll use min[min(xinit)/10, min(xinit)*10].

9.1.0.9 double apop_arms_settings::xprev

For internal use; please ignore. Previous value from Markov chain.

9.1.0.10 double apop_arms_settings::xr

Right bound. If you don't give me one, I'll use max[max(xinit)/10, max(xinit)*10].

9.2 apop_cdf_settings Struct Reference

Data Fields

- int draws
- gsl_matrix * draws_made
- int * draws_refcount
- gsl_rng * rng

Detailed Description

For use by apop_cdf when the CDF is generated via Monte Carlo methods.

Field Documentation

9.2.0.1 int apop_cdf_settings::draws

For random draw methods, how many draws? Default: 10,000.

9.2.0.2 gsl_matrix* apop_cdf_settings::draws_made

A store of random draws used to calcuate the CDF. Need only be generated once, and so stored here.

9.2.0.3 int* apop_cdf_settings::draws_refcount

For internal use.

9.2.0.4 gsl_rng* apop_cdf_settings::rng

For random draw methods. See apop_rng_get_thread on the default.

9.3 apop_composition_settings Struct Reference

Data Fields

- int **draw_ct**
- apop_model * **generator_m**

- apop_model * **ll_m**

9.4 apop_coordinate_transform_settings Struct Reference

Data Fields

- apop_model * base_model
- apop_data *(* base_to_transformed)(apop_data *)
- double(* jacobian_to_base)(apop_data *)
- apop_data *(* transformed_to_base)(apop_data *)

Field Documentation

9.4.0.1 **apop_model*** apop_coordinate_transform_settings::base_model

The pre-transformation model.

9.4.0.2 **apop_data***(* apop_coordinate_transform_settings::base_to_transformed) (**apop_data** *)

The function to transform the model from pre-transform space to post-transform space.

9.4.0.3 double(* apop_coordinate_transform_settings::jacobian_to_base) (**apop_data** *)

The derivative of the `transformed_to_base` function.

9.4.0.4 **apop_data***(* apop_coordinate_transform_settings::transformed_to_base) (**apop_data** *)

The function to transform from post-transform space back to pre-transform space. If this function does not exist, using a Jacobian-based transformation is probably not mathematically correct.

9.5 apop_cross_settings Struct Reference

Data Fields

- apop_model * model1
- apop_model * model2
- char * splitpage

Detailed Description

The settings to accompany the apop_cross model, representing the cross product of two models (or, via recursion, a list of models of arbitrary length).

Field Documentation

9.5.0.1 **apop_model*** apop_cross_settings::model1

The first model in the stack.

9.5.0.2 **apop_model*** apop_cross_settings::model2

The second model.

9.5.0.3 char* apop_cross_settings::splitpage

The name of the page at which to split the data. If `NULL`, I send the entire data set to both models as needed.

9.6 apop_data Struct Reference

Data Fields

- char **error**
- gsl_matrix * **matrix**
- struct apop_data * **more**
- apop_name * **names**
- char *** **text**
- size_t **textsize** [2]
- gsl_vector * **vector**
- gsl_vector * **weights**

Detailed Description

The apop_data structure represents a data set. See Data sets.

9.7 apop_dconstrain_settings Struct Reference

Data Fields

- apop_model * base_model
- double(* constraint)(apop_data *, apop_model *)
- int draw_ct
- gsl_rng * rng
- double(* scaling)(apop_model *)

Detailed Description

For use with the apop_dconstrain model. See its documentation for an example.

Field Documentation

9.7.0.1 **apop_model*** apop_dconstrain_settings::base_model

The model, before constraint.

9.7.0.2 double(* apop_dconstrain_settings::constraint) (**apop_data** *, **apop_model** *)

The constraint. Return 1 if the data is in the constraint; zero if out.

9.7.0.3 int apop_dconstrain_settings::draw_ct

How many draws to make for calculating the in-constraint model density via random draws. Current default: 1e4.

9.7.0.4 gsl_rng* apop_dconstrain_settings::rng

If you don't provide a `scaling` function, I calculate the in-constraint model density via random draws. If no `rng` is provided, I use a default RNG; see apop_rng_get_thread.

9.7.0.5 double(* apop_dconstrain_settings::scaling) (**apop_model** *)

Optional. Return the percent of the model density inside the constraint.

9.8 apop_kernel_density_settings Struct Reference

Data Fields

- apop_data * base_data
- apop_model * base_pmf
- apop_model * kernel
- int own_kernel
- int **own_pmf**
- void(* set_fn)(apop_data *, apop_model *)

Detailed Description

Settings for the apop_kernel_density model.

Field Documentation

9.8.0.1 **apop_data*** apop_kernel_density_settings::base_data

The data that will be smoothed by the KDE.

9.8.0.2 **apop_model*** apop_kernel_density_settings::base_pmf

I actually need the data in a apop_pmf. You can give that to me explicitly, or I can wrap the `.base_data` in a PMF.

9.8.0.3 **apop_model*** apop_kernel_density_settings::kernel

The distribution to be centered over each data point. Default, apop_normal with std dev 1.

9.8.0.4 int apop_kernel_density_settings::own_kernel

For internal use only.

9.8.0.5 void(* apop_kernel_density_settings::set_fn) (**apop_data** *, **apop_model** *)

The function I will use for each data point to center the kernel over each point. Default: set the upper-left element of the parameter set to the upper-left scalar in the data: `apop_data_set(m->parameters, .val= apop_data_get(in));`.

9.9 apop_lm_settings Struct Reference

Data Fields

- int destroy_data

- apop_model * input_distribution
- apop_data * instruments
- char want_cov
- char want_expected_value

Detailed Description

Settings for least-squares type models such as apop_ols or apop_iv

Field Documentation

9.9.0.1 int apop_lm_settings::destroy_data

If `'y'`, then the input data set may be normalized or otherwise mangled.

9.9.0.2 **apop_model*** apop_lm_settings::input_distribution

The distribution of $P(Y|X)$ is specified by the model holding this struct, but the distribution of X needs to be specified as well for any calculation of $P(Y)$. See the notes in the RNG section of the apop_ols documentation.

9.9.0.3 **apop_data*** apop_lm_settings::instruments

Use for the apop_iv regression, qv.

9.9.0.4 char apop_lm_settings::want_cov

Deprecated. Please use apop_parts_wanted_settings.

9.9.0.5 char apop_lm_settings::want_expected_value

Deprecated. Please use apop_parts_wanted_settings.

9.10 apop_loess_settings Struct Reference

Data Fields

- double ci_level
- apop_data * **data**
- struct loess_struct lo_s
- int want_predict_ci

Detailed Description

The code for the loess system is based on FORTRAN code from 1988, overhauled in 1992, linked in to Apophenia in 2009. The structure that does all the work, then, is a `loess_struct` that you should basically take as opaque.

The useful settings from that struct re-appear in the apop_loess_settings struct so you can set them directly, and then the settings init function will copy your preferences into the working struct.

The documentation for the elements is cut/pasted/modified from Cleveland, Grosse, and Shyu.

9.10.0.1 double apop_loess_settings::ci_level

If running a prediction, the level at which to calculate the confidence interval. default: 0.95

9.10.0.2 struct loess_struct apop_loess_settings::lo_s

`.data`: Mandatory. Your input data set.

`.lo_s.model.span`: smoothing parameter. Default is 0.75.

`.lo_s.model.degree`: overall degree of locally-fitted polynomial. 1 is locally-linear fitting and 2 is locally-quadratic fitting. Default is 2.

`.lo_s.normalize`: Should numeric predictors be normalized? If `'y'` - the default - the standard normalization is used. If `'n'`, no normalization is carried out.

.lo_s.model.parametric: for two or more numeric predictors, this argument specifies those variables that should be conditionally-parametric. The argument should be a logical vector of length `p`, specified in the order of the predictor group ordered in `x`. Default is a vector of 0's of length `p`.

.lo_s.model.drop_square: for cases with degree = 2, and with two or more numeric predictors, this argument specifies those numeric predictors whose squares should be dropped from the set of fitting variables. The method of specification is the same as for parametric. Default is a vector of 0's of length p.

.lo_s.model.family: the assumed distribution of the errors. The values may be `"gaussian"` or `"symmetric"`. The first value is the default. If the second value is specified, a robust fitting procedure is used.

`lo_s.control.surface`: determines whether the fitted surface is computed `"directly"` at all points or whether an `"interpolation"` method is used. The default, interpolation, is what most users should use unless special circumstances warrant.

`lo_s.control.statistics`: determines whether the statistical quantities are computed `"exactly"` or approximately, where `"approximate"` is the default. The former should only be used for testing the approximation in statistical development and is not meant for routine usage because computation time can be horrendous.

`lo_s.control.cell`: if interpolation is used to compute the surface, this argument specifies the maximum cell size of the k-d tree. Suppose k = floor(n∗cell∗span) where n is the number of observations. Then a cell is further divided if the number of observations within it is greater than or equal to k. default=0.2

`lo_s.control.trace_hat`: Options are `"approximate"`, `"exact"`, and `"wait.to.decide"`. When lo_s.control.surface is `"approximate"`, determines the computational method used to compute the trace of the hat matrix, which is used in the computation of the statistical quantities. If "exact", an exact computation is done; normally this goes quite fast on the fastest machines until n, the number of observations is 1000 or more, but for very slow machines, things can slow down at n = 300. If "wait.to.decide" is selected, then a default is chosen in loess(); the default is "exact" for n < 500 and "approximate" otherwise. If surface is "exact", an exact computation is always done for the trace. Set trace_hat to "approximate" for large dataset will substantially reduce the computation time.

`lo_s.model.iterations`: if family is `"symmetric"`, the number of iterations of the robust fitting method. Default is 0 for lo_s.model.family = gaussian; 4 for family=symmetric.

That's all you can set. Here are some output parameters:

`fitted_values:` fitted values of the local regression model

`fitted_residuals:` residuals of the local regression fit

`enp:` equivalent number of parameters.

`s:` estimate of the scale of the residuals.

`one_delta:` a statistical parameter used in the computation of standard errors.

`two_delta:` a statistical parameter used in the computation of standard errors.

`pseudovalues:` adjusted values of the response when robust estimation is used.

`trace_hat:` trace of the operator hat matrix.

`diagonal:` diagonal of the operator hat matrix.

`robust:` robustness weights for robust fitting.

`divisor:` normalization divisor for numeric predictors.

9.10.0.3 int apop_loess_settings::want_predict_ci

If `'y'` (the default), calculate the confidence bands for predicted values

9.11 apop_mcmc_proposal_s Struct Reference

Data Fields

- int **accept_count**
- int(* adapt_fn)(struct apop_mcmc_proposal_s *ps, struct apop_mcmc_settings *ms)
- apop_model * proposal
- int reject_count
- void(* step_fn)(double const *, struct apop_mcmc_proposal_s *, struct apop_mcmc_settings *)

Detailed Description

A proposal distribution for apop_mcmc_settings and its accompanying functions and information. By default, these will be apop_multivariate_normal models. The `step_fn` and `adapt_fn` have to be written around the model and your preferences. For the defaults, the step function recenters the mean of the distribution around the last accepted proposal, and the adapt function widens Σ for the Normal if the accept rate is too low; narrows it if the accept rate is too large.

You may provide an array of proposals. The length of the list of proposals must match the number of chunks, as per the `gibbs_chunks` setting in the apop_mcmc_settings group that the array of proposals is a part of. Each proposal must be initialized to include all elements, and the step and adapt functions probably have to be written anew for each type of model.

Field Documentation

9.11.0.1 int(* apop_mcmc_proposal_s::adapt_fn) (struct **apop_mcmc_proposal_s** *ps, struct **apop_mcmc_settings** *ms)

Called every step, to adapt the proposal distribution using information to this point in the chain.

9.11.0.2 **apop_model*** apop_mcmc_proposal_s::proposal

The distribution from which test parameters will be drawn. After getting the draw using the `draw` method of the proposal, the base model's `parameters` element is filled using apop_data_fill. If `NULL`, apop_model_metropolis will use a Multivariate Normal with the appropriate dimension, mean zero, and covariance matrix I. If not `NULL`, be sure to parameterize your model with an initial position.

9.11.0.3 int apop_mcmc_proposal_s::reject_count

If there are multiple apop_mcmc_proposal_s structs for multiple chunks, These count accepts/rejects for this chunk. The apop_mcmc_settings group has a total for the aggregate across all chunks.

9.11.0.4 void(* apop_mcmc_proposal_s::step_fn) (double const *, struct **apop_mcmc_proposal_s** *, struct **apop_mcmc_settings** *)

Modifies the parameters of the proposal distribution given a successful draw. Typically, this function writes the drawn data point to the parameter set. If the draw is a scalar, the default function sets the 0th element of the model's `parameter` set with the draw (works for the apop_normal and other models). If the draw has multiple dimensions, they are all copied to the parameter set, which must have the same size.

9.12 apop_mcmc_settings Struct Reference

Data Fields

- int accept_count
- int(* base_adapt_fn)(struct apop_mcmc_proposal_s *ps, struct apop_mcmc_settings *ms)
- apop_model * base_model
- void(* base_step_fn)(double const *, struct apop_mcmc_proposal_s *, struct apop_mcmc_settings *)
- int **block_count**
- size_t * block_starts
- double burnin
- apop_data * **data**
- char gibbs_chunks
- int histosegments
- double last_ll
- long int periods
- apop_model * pmf
- int proposal_count
- int proposal_is_cp
- apop_mcmc_proposal_s * proposals
- int reject_count
- char start_at
- double target_accept_rate

Detailed Description

Method settings for a model to be put through Bayesian updating.

Field Documentation

9.12.0.1 int apop_mcmc_settings::accept_count

After calling apop_model_metropolis, this will have the number of accepted proposals.

9.12.0.2 int(* apop_mcmc_settings::base_adapt_fn) (struct **apop_mcmc_proposal_s** *ps, struct **apop_mcmc_settings** *ms)

If a apop_mcmc_proposal_s has `NULL adapt_fn`, use this. If you don't want an adapt function, set this to a do-nothing function.

9.12.0.3 **apop_model*** apop_mcmc_settings::base_model

The model you provided with a `log_likelihood` or `p` element (which need not sum to one). You do not have to set this: if it is `NULL` on input to apop_model_metropolis, I will fill it in.

9.12.0.4 void(* apop_mcmc_settings::base_step_fn) (double const *, struct **apop_mcmc_proposal_s** *, struct **apop_mcmc_settings** *)

If an apop_mcmc_proposal_s struct has `NULL step_fn`, use this. If you don't want a step function, set this to a do-nothing function.

9.12.0.5 size_t* apop_mcmc_settings::block_starts

For internal use

9.12.0.6 double apop_mcmc_settings::burnin

What *percentage* of the periods should be ignored as initialization. That is, this is a number between zero and one.

9.12.0.7 char apop_mcmc_settings::gibbs_chunks

See the apop_model_metropolis documentation for discussion.

`'a'`: One step draws and accepts/rejects all parameters as a unit

`'b'`: draw in blocks: the vector is a block, the matrix is a separate block, the weights are a separate block, and so on through every page of the model parameters. Each block of parameters is drawn and accepted/rejected as a unit.

`'1'`: draw each parameter and accept/reject separately. One MCMC step consists of a set of draws for every parameter.

9.12.0.8 int apop_mcmc_settings::histosegments

If outputting a binned PMF, how many segments should it have?

9.12.0.9 double apop_mcmc_settings::last_ll

If you have already run MCMC, the last log likelihood in the chain.

9.12.0.10 long int apop_mcmc_settings::periods

For how many steps should the MCMC chain run?

9.12.0.11 **apop_model*** apop_mcmc_settings::pmf

If you have already run MCMC, I keep a pointer to the model so far here. Use apop_model_metropolis_draw to get one more draw.

9.12.0.12 int apop_mcmc_settings::proposal_count

The number of proposal sets; see `gibbs_chunks` below.

9.12.0.13 int apop_mcmc_settings::proposal_is_cp

For internal use.

9.12.0.14 **apop_mcmc_proposal_s*** apop_mcmc_settings::proposals

The list of proposals. You can probably use the default of adaptive multivariate normals. See the apop_↩ mcmc_proposal_s struct for details.

9.12.0.15 int apop_mcmc_settings::reject_count

After calling apop_model_metropolis, this will have the number of rejected proposals.

9.12.0.16 char apop_mcmc_settings::start_at

If `'1'` (the default), start with a first proposal of all 1s. Even when this is a far-from-useful starting point, MCMC typically does a good job of crawling to better spots early in the chain.
The default when this is unset is to start at the `parameters` of the apop_model sent in to apop_model↩ _metropolis.

9.12.0.17 double apop_mcmc_settings::target_accept_rate

The desired acceptance rate, for use by adaptive proposals. Default: .35

9.13 apop_mixture_settings Struct Reference

Data Fields

- apop_model * cmf
- int * cmf_refct
- int **model_count**
- apop_model ** model_list
- int * param_sizes
- gsl_vector * weights

Detailed Description

For mixture distributions, typically set up using apop_model_mixture. See apop_mixture for discussion. Please consider all elements but `model_list` and `weights` as private and subject to change. See the examples for use of these elements.

Field Documentation

9.13.0.1 **apop_model*** apop_mixture_settings::cmf

For internal use by the draw method.

9.13.0.2 int* apop_mixture_settings::cmf_refct

For internal use, so I can garbage-collect the CMF when needed.

9.13.0.3 **apop_model**** apop_mixture_settings::model_list

A `NULL-terminated` list of component models.

9.13.0.4 int* apop_mixture_settings::param_sizes

The number of parameters for each model. Useful for unpacking the params.

9.13.0.5 gsl_vector* apop_mixture_settings::weights

The likelihood of a draw from each component.

9.14 apop_mle_settings Struct Reference

Data Fields

- double **delta**
- double dim_cycle_tolerance
- int **iters_fixed_T**
- double **k**
- int max_iterations
- char * method
- double **mu_t**
- int **n_tries**
- apop_data ** path
- gsl_rng * **rng**
- double * starting_pt
- double step_size
- double **t_initial**
- double **t_min**
- double tolerance
- int verbose

Detailed Description

The settings for maximum likelihood estimation (including simulated annealing).

Field Documentation

9.14.0.1 double apop_mle_settings::dim_cycle_tolerance

If zero (the default), the usual procedure. If > 0, cycle across dimensions: fix all but the first dimension at the starting point, optimize only the first dim. Then fix the all but the second dim, and optimize the second dim. Continue through all dims, until the log likelihood at the outset of one cycle through the dimensions is within this amount of the previous cycle's log likelihood. There will be at least two cycles.

9.14.0.2 int apop_mle_settings::max_iterations

Ignored by simulated annealing. Other methods halt (and set the `"status"` element of the output estimate's info page) if they do this many iterations without finding an optimum.

9.14.0.3 char* apop_mle_settings::method

The method to be used for the optimization. All strings are case-insensitive.

String	Name	Notes

"NM simplex"	Nelder-Mead simplex	Does not use gradients at all. Can sometimes get stuck.
"FR cg"	Conjugate gradient (Fletcher-Reeves) (default)	CG methods use derivatives. The converge to the optimum of a quadratic function in one step; performance degrades as the objective digresses from quadratic.
"BFGS cg"	Broyden-Fletcher-Goldfarb-↩ Shanno conjugate gradient	
"PR cg"	Polak-Ribiere conjugate gradient	
"Annealing"	simulated annealing	Slow but works for objectives of arbitrary complexity, including stochastic objectives.
"Newton"	Newton's method	Search by finding a root of the derivative. Expects that gradient is reasonably well-behaved.
"Newton hybrid"	Newton's method/gradient descent hybrid	Find a root of the derivative via the Hybrid method If Newton proposes stepping outside of a certain interval, use an alternate method. See the GSL manual for discussion.
"Newton hybrid no scale"	Newton's method/gradient descent hybrid with spherical scale	As above, but use a simplified trust region.

9.14.0.4 **apop_data**** apop_mle_settings::path

If not `NULL`, record each vector tried by the optimizer as one row of this apop_data set. Each row of the `matrix` element holds the vector tried; the corresponding element in the `vector` is the evaluated value at that vector (after out-of-constraints penalties have been subtracted). A new apop_data set is allocated at the pointer you send in. This data set has no names; add them as desired. For a sample use, see Optimization.

9.14.0.5 double* apop_mle_settings::starting_pt

An array of doubles (e.g., `(double*){2,4,6,8}`) suggesting a starting point. If NULL, use an all-ones vector. If `startv` is a `gsl_vector` and is not a view of a matrix, use `.starting_pt=startv->data`.

9.14.0.6 double apop_mle_settings::step_size

The initial step size.

9.14.0.7 double apop_mle_settings::tolerance

The precision the minimizer uses in its stopping rule. Only vaguely related to the precision of the actual MLE.

9.14.0.8 int apop_mle_settings::verbose

Give status updates as we go. This is orthogonal to the `apop_opts.verbose` setting.

9.15 apop_model Struct Reference

Data Fields

- long double(* **cdf**)(apop_data *d, apop_model *params)
- long double(* **constraint**)(apop_data *data, apop_model *params)
- apop_data * **data**
- int(* **draw**)(double *out, gsl_rng *r, apop_model *params)
- int **dsize**
- char **error**
- void(* **estimate**)(apop_data *data, apop_model *params)
- apop_data * **info**
- long double(* **log_likelihood**)(apop_data *d, apop_model *params)
- void * **more**
- size_t **more_size**
- int **msize1**
- int **msize2**
- char **name** [101]
- long double(* **p**)(apop_data *d, apop_model *params)
- apop_data * **parameters**
- void(* **prep**)(apop_data *data, apop_model *params)
- apop_settings_type * **settings**
- int **vsize**

Detailed Description

The elements of the apop_model type, representing a statistical model. See Models and Writing new models for use and details.

A statistical model. See Models for details.

9.16 apop_name Struct Reference

Data Fields

- char ** **col**
- int **colct**
- char ** **row**
- int **rowct**
- char ** **text**
- int **textct**
- char * **title**
- char * **vector**

Detailed Description

This structure holds the names of the components of the apop_data set. You may never have to worry about it directly, because most operations on apop_data sets will take care of the names for you.

9.17 apop_opts_type Struct Reference

Data Fields

- char db_engine
- char * db_name_column
- char db_pass [101]
- char db_user [101]
- char input_delimiters [100]
- FILE * log_file
- char * nan_string
- char output_delimiter [100]
- int **rng_seed**
- char stop_on_warning
- int verbose
- float **version**

Detailed Description

The global options.

Field Documentation

9.17.0.1 char apop_opts_type::db_engine

If this is 'm', use mySQL, else use SQLite.

9.17.0.2 char* apop_opts_type::db_name_column

If not NULL or `""`, the name of the column in your tables that holds row names.

9.17.0.3 char apop_opts_type::db_pass[101]

Password for database login. Max 100 chars.

9.17.0.4 char apop_opts_type::db_user[101]

Username for database login. Max 100 chars.

9.17.0.5 char apop_opts_type::input_delimiters[100]

Deprecated. Please use per-function inputs to apop_text_to_db and apop_text_to_data. Default = "|,\t"

9.17.0.6 FILE* apop_opts_type::log_file

The file handle for the log. Defaults to `stderr`, but change it with, e.g., `apop_opts.log_file = fopen("outlog", "w");`

9.17.0.7 char* apop_opts_type::nan_string

The string used to indicate NaN. Default: `"NaN`. Comparisons are case-insensitive.

9.17.0.8 char apop_opts_type::output_delimiter[100]

The separator between elements of output tables. The default is "\t", but for LaTeX, use "&\t", or use "|" to get pipe-delimited output.

9.17.0.9 char apop_opts_type::stop_on_warning

See Errors, logging, debugging and stopping .

9.17.0.10 int apop_opts_type::verbose

Set this to zero for silent mode, one for errors and warnings. default = 0.

9.18 apop_parts_wanted_settings Struct Reference

Data Fields

- char **covariance**
- char **info**
- char **predicted**
- char **tests**

Detailed Description

The default is for the estimation routine to give some auxiliary information, such as a covariance matrix, predicted values, and common hypothesis tests. Some uses of a model depend on these items, but if they are a waste of time for your purposes, this settings group gives a quick way to bypass them all.

Adding this settings group to your model without changing any default values—

```
Apop_model_add_group(your_model, apop_parts_wanted);
```

—will turn off all of the auxiliary calculations covered, because the default value for all the switches is `'n'`, indicating that all elements are not wanted.

From there, you can change some of the default `'n'`s to `'y'`s to retain some but not all auxiliary elements. If you just want the parameters themselves and the covariance matrix:

```
Apop_model_add_group(your_model, apop_parts_wanted, .covariance='y');
```

- Not all models support this, although the models with especially compute-intensive auxiliary info do (e.g., the maximum likelihood estimation system). Check the model's documentation.

- Tests may depend on covariance, so `.covariance='n', .tests='y'` may be treated as `.covariance='y', .tests='y'`.

9.19 apop_pm_settings Struct Reference

Data Fields

- apop_model * **base**
- int **draws**
- int **index**
- gsl_rng * **rng**

Detailed Description

Settings for getting parameter models (i.e. the distribution of parameter estimates)

9.20 apop_pmf_settings Struct Reference

Data Fields

- gsl_vector * cmf
- int * cmf_refct
- char draw_index
- long double total_weight

Detailed Description

Settings to accompany the apop_pmf.

Field Documentation

9.20.0.1 gsl_vector* apop_pmf_settings::cmf

A cumulative mass function, for the purposes of making random draws.

9.20.0.2 int* apop_pmf_settings::cmf_refct

For internal use, so I can garbage-collect the CMF when needed.

9.20.0.3 char apop_pmf_settings::draw_index

If `'y'`, then draws from the PMF return the integer index of the row drawn. If `'n'` (the default), then return the data in the vector/matrix elements of the data set.

9.20.0.4 long double apop_pmf_settings::total_weight

Keep the total weight, in case the input weights aren't normalized to sum to one.

9.21 apop_settings_type Struct Reference

Data Fields

- void * **copy**
- void * **free**
- char **name** [101]
- unsigned long **name_hash**
- void * **setting_group**

9.22 coeff_struct Struct Reference

Data Fields

- double **scaling**