

DistAnalysis 1.0

DistAnalysis is a module that interacts with a PostgreSQL database calculating a parameter distribution or a functional relation. All analyses can be made specifying up to two different grouping conditions.

The module also offers the possibility to graph obtained curves with a PHP module.

Disclaimer This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

CHAPTER 1: Installation

DistAnalysis, to properly run, has the following requirements:

- a working PostgreSQL 7.1 installation (even if it should work also with a previous version)
- a working HTTP server (all test were made with Apache) with PHP4 correctly configured and interfaced with PostgreSQL (even if it should work also with PHP3)
- PHPlot 4.4.6 (<http://www.sourceforge.net/projects/phplot>)

To install the software edit Makefile.global and specify the directory in which PostgreSQL is installed and the database name into which you want to install the module.

To compile the module type:

```
make
```

To install it, run the following command as PostgreSQL superuser (typically postgres):

```
make install
```

The installation routine creates four tables (see below for details) and some functions into the target database.

If you want to uninstall it, simply type:

```
make uninstall
```

and all created items will be removed.

Note that, if you want to install the module on several databases, you have to repeat this procedure more times.

To install PHP scripts follow these steps:

- copy all php scripts into a web browse-able folder
- extract into the same folder phplot.php, phplot_data.php, rgb.inc.php and benjamingothic.ttf taken from PHPlot 4.4.6 distribution
- patch phplot.php with phplot.php.diff using the following command

```
patch phplot.php < phplot.php.diff
```

if the patching process fails, it could depend by an incorrect line terminating format (did you open phplot.php with a Windows editor ?). In this case try editing phplot.php with a Unix editor and forcing it to rewrite the file (for example inserting and deleting a space), so that newlines on the file can be converted from CR+LF to LF format

- the script will be accessible from the following address:

```
http://server/chosen_phplot_path/
```

where server and path depend on your installation

CHAPTER 2: a rapid introduction

To understand what DistAnalysis can do for you, let's have a look to some examples.

Ex.1: the first analysis

Suppose you have the table "students":

Name	Age	Year	Num_Exams	Course	Tutor
text	smallint	smallint	smallint	integer	integer

and that you want to analyze how parameter Age is distributed into the table. You can do it by calling a DistAnalysis helper function, `da_run(...)`, with, among other arguments, the following simple SQL statement:

```
select age as param from students
```

So, calling:

```
da_run('ex1', 'select age as param from students', ... )
```

you instruct the module to analyze that set of data and build a distribution image of that parameter.

The parameter to analyze must always be called 'param'. Argument 'ex1' identifies the distribution name and will be used later to fetch calculated data.

The module stores calculated informations in `d_info`, `d_data` and `d_cumul` tables.

- `d_info` contains all general informations about the distribution
- `d_data` contains the collection of points regarding the probability density function in the form (x,y)
- `d_cumul` contains the collection of points regarding the probability cumulative function in the form (x,y)

Additionally the module uses a fourth table, `f_data`, whose meaning will be clearer later:

- `f_data` contains the collection of points regarding functional relations in the form (x,y)

Now, to get the calculated distribution we have to address it using its name; we can fetch general informations about data firing the following statement:

```
select * from d_info where descr = 'ex1'
```

or get the probability density function points with the statement:

```
select x, y from d_data d inner join d_info i on d.id=i.id  
where i.descr = 'ex1';
```

Similarly, to get the cumulative function data, execute:

```
select x, y from d_cumul d inner join d_info i on d.id=i.id  
where i.descr = 'ex1';
```

Ex.2: grouping feature

Suppose now that we want to compare age distributions regarding students of different courses.

We could solve the problem executing the method previously illustrated many times as the number of different courses we have data for. Or we can use the grouping instruction:

```
da_run(      'ex2' ,
            'select age as param, course from students',
            'course',
            ...
        )
```

which separately analyzes students of different courses.

Note that you must always specify the grouping parameters ('course' in this case). Grouping parameters can only be integer.

You can specify up to two different grouping conditions (see Ex.3).

DistAnalysis stores general parameters of any distribution into `d_info` with different values in the grouping field `gp1` (values taken from the field `Course` in previous example). So, if you want to extract data about Physics Dept. (i.e. `Course=10`), consider the following query:

```
select x, y
  from d_data ('or d_cumul') d inner join d_info i
    on d.id = i.id
where
  i.descr = 'ex2' and
  i.gp1 = 10
```

the table's choice (`d_data` or `d_cumul`) depends whether you're looking for density or cumulative function.

Ex.3: a more complex example

Now consider the additional table "Tutors":

Tutor	Name	Sex
integer	text	char

and suppose you want to consider the Age distribution of students with female tutors that have passed more than 10 exams, also detailing Year and Course.

You can do the job executing:

```
da_run( 'ex3',
        'select age as param, year, course
        from
            students s inner join tutors t on
            s.tutor=t.tutor
        where
            s.num_exams > 10 and
            t.sex = ''F''
        ',
        'year, course',
        ...
    )
```

As usual all general informations are stored into `d_info` with different values of the grouping parameters `gp1` and `gp2` (now there's also `gp2` because there're 2 grouping conditions).

So, to extract the distribution regarding students at their 5th year in the Physics Dept. (Course=10), just hit:

```
Select x, y
from d_data ('or d_cumul') d inner join d_info i on
    d.id = i.id
where
    i.descr = 'ex3' and
    i.gp1 = 5 and
    i.gp2 = 10
```

Note that `gp1` refers to the first grouping parameter (Year) and `gp2` to the second (Course).

Ex.4: a functional relation

Suppose now that you want to analyze the functional relation between `age` and `num_exams`, detailing the results by Course. You can do the job by calling another function, `fa_run(...)`, whose external behavior is similar to `da_run`. Basically, it accepts a query and a grouping conditions as input and builds a set of points into table `f_data` which represents the link between the two parameters `x` and `y`.

In this example the code is:

```
fa_run( 'ex4',
        'select
          age as y,
          num_exams as x,
          course
        from students',
        'course',
        ...
      )
```

Note that input parameters must be called x and y.

To extract calculated data you can use the following query:

```
Select x, y
      from f_data f inner join d_info i on f.id = i.id
where
      i.descr = 'ex4' and
      i.gpl = param
```

where param is the course you're interested in. As usual you can use up to two different grouping conditions.

To understand how to do graphical analysis, see chapter 4.

CHAPTER 3: Reference Guide

The module is written in C and PL/pgSQL. The C code consists in helper functions built to manage a PostgreSQL aggregate function through which all distribution analyses are made.

Distribution analysis routine

da_run(

1. description text: description of the measure being analyzed. Data can be addressed using this value
2. nintervals integer: number of intervals to divide the domain into
3. query text: query to perform to extract data
4. groupby text: comma separated list of grouping fields (" if there's no grouping)
5. forbid1 float8: forbidden value. Don't include into distribution this value but count the sample
6. forbid2 float8: forbidden value
7. forbid3 float8: forbidden value
8. nsigma_confidence integer: limit analysis to $[-n*\sigma, +n*\sigma]$ (ie. if =3 approximation is $\sim 2.7E-03$ over hypothesis of normal distribution). 0 to disable filtering.
9. unit_descr text: unit measure description (i.e. sec, dB, mW,...)
10. gp_sql text: grouping sql query. This query is used to determine the group name
11. fzero boolean: filter zero-value probability density function points. This flag is used to avoid 'comb' effect on graphical representations.

)

Parameters 5, 6 and 7 are optional and you can omit them (da_run accepts from 8 up to 11 arguments). The reason because has been included support for forbidden values is that filtering out data with SQL source code doesn't correctly reports the total number of analyzed samples (including those discarded).

gp_sql is an optional SQL query that can be used to determine the name of a group starting from gp1 and gp2 values.

I.e.:

```
'select case when gp1_param=0 then 'a' else 'b' end'
```

It's internally executed from PHP module to differentiate functions belonging to the same experiment with different gp1 and gp2 parameters. Before executing the query gp1_param and gp2_param are substituted with their numerical values.

gp_sql can be a void string "".

Functional analysis routine

fa_run(

1. description text: description of the measure being analyzed. Data can be addressed using this value
2. query text: query to perform to extract data
3. groupby text: comma separated list of grouping fields (" if there's no grouping)
4. gp_sql text: grouping sql query. This query is used to determine the group name
5. unit_x text: unit measure description for x-axis data (i.e. sec, dB, mW, ..)
6. unit_y text: unit measure description for y-axis data (i.e. sec, dB, mW, ..)

)

Parameters' meaning is the same of what already seen for da_run.

Tables

d_info: contains all general informations about the distribution

id	integer primary key	data identifier
descr	varchar(32)	data name
mean	float8	mean of the parameter
sigma	float8	standard deviation of the parameter
nelems	integer	number of samples considered in the analysis
ntotelems	integer	total number of samples (including not valid ones)
noutrange	integer	number of samples outside +/- nsigma
nintrvs	integer	number of intervals
gp1	integer	first grouping identifier
gp2	integer	second grouping identifier
unit	varchar(7)	measure unit
unit_y	varchar(7)	y-axis measure unit (for functional relations)
nsigma	integer	prefiltering size (between +/- nsigma)
type	integer default 0	analysis type (0 = distrib 1 = func)
filterzero	boolean default false	filter zero value samples in d_data
sql_descr	text	SQL code to identify the group from gp1 and gp2
conf	float8	confidence interval (not yet used)
id_dist	integer	reference distribution id (not yet used)

d_data: contains probability density functions tabulated values

Field	Type	Description
id	integer references d_info on delete cascade	data identifier
x1	float8	starting range
x2	float8	ending range
p	float8	probability (calculated as frequency ratio)

d_cumul: contains cumulative probability functions tabulated values

Field	Type	Description
id	integer references d_info on delete cascade	data identifier
x	float8	data value
p	float8	cumulative probability

f_data: contains functional relation (x,y) tabulated values

Field	Type	Description
id	integer references d_info on delete cascade	data identifier
x	float8	x-axis data
y	float8	y-axis data

Note that d_data, f_data and d_cumul tables are linked with a cascaded referential integrity to d_info. So, if you delete a row from d_info, you will delete all associated rows in the other tables.

CHAPTER 4: Graphical analysis

DistAnalysis contains some PHP scripts that can be used for graphical distributions analysis. They run on top of any PHP working environment (typically an Apache web server with `mod_php` correctly configured). All graphical scripts were built starting from `phplot-4.4.6` by Afan Ottenheimer, with some changes to the main module (`phplot.php.diff`).

After you've installed the module, you can test it by issuing the following command:

```
make runtest
```

from inside test directory. The script creates a new database, `distanalysis_demo`, and a temporary inside it. Then it copies 10000 random samples for each of four remarkable distributions (normal, t-student, chi-square and gamma) and launch `da_run` routine to analyze data. Additionally, it commence a functional relation analysis for the relation $y = \sin(x)$.

After the script has done, we're ready to start using the graphic module. From inside any browser, open the address `http://server/path`, depending on where you put the scripts.

The first page contains a login request. You have to specify database, username and password. Once you've been logged in, you can choose the functions you want to graph selecting them from the listbox on the left. On all items there is a final (D) for distributions and (F) for functional relations. You cannot choose both functional relations and distributions at the same time.

To choose multiple items use CTRL or SHIFT keys. You can compare up to 8 graphs at the same time.

Other fields in the form let you customize the graph:

- Width and Height control the size of the output images
- X-window and Y-window are used to limit the analysis to a specific area
- Graph, X-Axis and Y-Axis titles let you override the default values
- Legend position can be used to move the legend over the graph
- Show URL flag can be used to view graphs' URL. This is useful to embed images in documents (see later description)

All settings can be left to their default values.

In substitution of the direct selection from the listbox, you can choose the functions you're interested in using a SQL query from inside the SQL query box. The query must return only the field id from table `d_info`.

For example, to select all distributions generated from previous test, use the following query:

```
select id from d_info where descr = 'demo'
```

The module automatically recognizes if that id corresponds to a distribution or a functional relation. This is useful to test the query to use before embedding it into an URL, as documented later. If your query returns both functional relations and distributions, the module will fire an error.

In addition to the normal usage, the module permits you to embed graphs inside documents, using an URL. To understand how it can be done, you should before have a look to the scripts structure.

The module is divided in four scripts:

- `index.php` the login page
- `choose_graph.php` main page from where you can choose functions
- `draw_graph.php` container for the graphs
- `fgraph.php` the script that produces all graphs

If you want to embed an URL inside a document you have to manually call `fgraph.php` with the right arguments. Here's a list of accepted parameters:

- `postgres_dbname` Required
- `postgres_username` Required
- `postgres_password` Required
- `sql` SQL code to fetch IDs of functions to graph. Required
- `type` Function type: 1=prob density, 2=cumulative prob, 3= functional relation. Required.
- `postgres_port` Optional, default = 5432
- `postgres_host` Optional, default = localhost
- `xsize` Graph width. Optional, default = 600
- `ysize` Graph height. Optional, default = 400
- `xstart` Start to graph data from xstart. Optional
- `xend` Stop to graph data at xend. Optional
- `ystart` Start to graph data from ystart. Optional
- `yend` Stop to graph data at yend. Optional
- `title_x` Use this x-axis title instead of the default one. Optional
- `title_y` Use this y-axis title instead of the default one. Optional
- `title_graph` Use this title for the graph instead of the default one. Optional
- `legend_pos` X,Y Position for the upper left legend corner (default = 90,70). Optional

For example, if you want to get probability density for the T-Student distribution, use the following link:

```
http://supporto//plot/fgraph.php?postgres_dbname=distanalysis_demo&postgres_username=username&postgres_password=password&sql=select%20id%20from%20d_info%20where%20id=3%20or%20id=4%20or%20id=1%20or%20id=2&type=2
```

where server, path, username and password depend on your installation.

CHAPTER 5: tips and tricks

Here are some useful suggestions regarding the usage of DistAnalysis.

- The PHP module will not work if any `d_info` row contains in `sql_descr` an invalid SQL instruction, because it uses the view `da_info`, which automatically runs that code. So check what's wrong in `sql_descr` if `choose_graph.php` returns an error.
- By default, PostgreSQL doesn't authenticate connection requests from localhost. This could create a security breach, because, by default, DistAnalysis connects to localhost. To solve it, edit your `pg_hba.conf` and switch 127.0.0.1 entry from trust to password.
- The distributions graphs are not so accurate if data samples are less than 20-30 times the number of intervals.
- When analyzing functional relations, especially on large tables with double precision `x` and `y` parameters, take care to round values to the precision you want, otherwise `f_data` becomes huge. You can use the routine `round(value, decimal_places)` for this scope.

Copyright (C) 2001 David Ciarniello, wiseminosse@inwind.it

This is distributed with NO WARRANTY and under the terms of the GNU GPL license.

If you use it - a cookie or some credit would be nice.

You can get a copy of the GNU GPL at <http://www.gnu.org/copyleft/gpl.html>

See <http://web.tiscalinet/wiseminosse> for later updates.