gbutils overview

Giulio Bottazzi

December 27, 2014

Contents

1 Brief description of programs

The programs in `gbutils` can be divided in four broad classes:

- Data Manipulation

- Data Transformation
- Descriptive Statistics
- Statistical Tests and Models

The basic operation is essentially the same for all programs: you feed the standard input of the program with data in ASCII format separated by spaces, tabs or newline character. In general, each input line is considered a record and the blank separated entries in each line are considered different fields. The exact way in which different records and fields are treated depends on the program and can vary accordingly to the options specified in the command line (see below).

After the program has read the data from standard input, it performs the required manipulations/analyses and prints the result to standard output, in the form of an ASCII file of newline separated records. Inside each output record, the fields are separated by spaces. Obviously, the meaning of the records and fields depend on the program.

1.1 Data Manipulation

These programs do not perform any analysis by themselves. Rather, they are provided as an help to prepare data for subsequent analysis. In particular, gbget is the only program that reads data from file and not from standard input. It can be used to extract data, according to a given pattern, from one or more files and send them, through a pipe `|`, to other utilities. This program possesses a rather complex set of options. See README.gbget for a tutorial on its use.

gbget extract data from a tabular input according to a specified pattern. It is possible to access more files at the same time, merge their contents and transpose or flatten the resulting table.

gbfun compute generic functions on data in a column-wise manner. The function can be applied to all the columns or defined in a recursive way.

gbgrid generate a grid (i.e. a matrix) of values according to a user specified function

gbboot generate bootstrapped sequences from data provided sample

gbrand generated i.i.d. pseudo random variates

gbenv provide information about the numeric environment and the internal settings of the package

1.2 Data transformation

These programs perform basic transformation on input data, which are often considered preliminary to further statistical analysis.

gbmave print moving statistics (average, variance, etc.) of input data

gbinterp compute the interpolation on a regular mesh of user provided points. It can also print first and second derivative of the interpolation.

gbfilternear filter near points in Ecuclidean metrics. Point whose distance is below a given threshold are removed.

1.3 Descriptive statistics

These utilities are useful in the representation and description of data. They encompass simple statistics and more "advanced" non parametric methods.

gbdist cumulative distribution of input data

gbstat simple descriptive statistics of input data

gbbin compute binned statistics

gbquant quantiles of the empirical distribution of input data

gbhisto histogram for univariate data. Choose between absolute frequencies, relative frequencies and empirical density

gbker kernel density estimate for univariate data. The type of kernel, the bandwidth and the computation method can be specified at the command line

gbnear density estimate via nearest neighbors method

gbker2d kernel density estimate for bivariate data

gbhisto2d histograms for bivariate data

gbgcorr Compute the correlation dimension of a time series with a Gaussian kernel.

gbacorr It computes the autocorrelogram or the cross-autocorrelogram of a series of observations. It reads the data column-wise.

gbxcorr Compute the cross-covariance and cross-correlation coefficients with and without the removal of the mean of two samples. It reads the data column-wise.

1.4 Statistical tests and models

The utilities provide statistical tests to compare different samples and non-parametric method to investigate relationship between paired (or in general compounded) observations.

gbtest various one and two samples statistical tests. When available, p-score significance is also provided.

gbmodes find the critical bandwidth for a kernel density estimate to generate a given number of modes and compute the associated p-value using smoothed bootstrap technique

gbbin the program takes couples of values X Y (separated by spaces), bins them with respect to the first variable and prints statistics of the second variables

gbkreg compute the kernel non-linear regression function

gbkreg2d compute the kernel non-linear regression function on three dimensional data

gblreg compute linear OLS regression

gbglreg compute generalized linear OLS regression

gbnlreg compute non linear regression using OLS, MAD or asymmetric MAD estimators

gbnlqreg compute non linear quantile regression

gbnlmult contemporaneous least square estimation of a system of non linear equations.

gbnlprobit estimate a non linear probit model on binary data

gbhill estimate different families of probability distribution on the extremal data using maximum likelihood.

For more information on a specific command, use the -h command line option.

Please, notice that all the programs work by loading the whole set of data in memory before computing the relevant statistics. In this respect, they are probably not suitable to be used on very large datasets.

2 Understanding Input/Output

All the commands of this package read input in ASCII format. The data should be separated by white characters (spaces or tabs) or newlines. Lines beginning with a fence symbol # are ignored. They are simply skipped by the input routine.

If support for the zlib has been included at compile time (see above) the input ASCII file can be gz-compressed.

A file can contain several blocks of data. Blocks are separated by two consecutive blank lines. In general, all operations are performed on the first block found in the datafile. The program 'gbget' can be used to extract one particular block (or set of blocks) from one file.

2.1 Sequential, tabular and compounded input

The utilities in this package use three different ways of reading data from input:

sequential In 'sequential' format a single dataset is internally build from the data input file. All the entries found on one column of input are read sequentially and put in the same dataset. Notice that the different lines must contain the same number of entries or NAN values are generated.

tabular In 'tabular' format, each column of the input is treated as a different dataset; the program will internally create a list of datasets, one for each column of input. The different entries on one line are then put inside different sets. Notice that, in this case, the number of fields in the first non-comment non-empty line decides the number of datasets. All subsequent input lines should contain the same number of fields (but see below).

compounded In 'compounded' format the program reads a fixed number of fields from each line. Each line is internally stored as an n-tuple (a couple or a triplet) and treated accordingly. Notice that if some line contains more fields than needed, the extra fields are ignored.

An example can clarify the difference between the 'sequential', 'tabular' and 'compounded' format. Suppose to have the following input datafile

```
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
```

using the 'sequential' input the unique dataset `{1.0,2.0,3.0,4.0,...,9.0}` is internally generated by the program. Notice the ordering: all the entries of one line are inserted in the internal dataset before the next line is red.

In 'tabular' format, the program builds instead three different datasets: `{1.0,4.0,7.0}`, `{2.0,5.0,8.0}` and `{3.0,6.0,9.0}` and use each set separately for its subsequent duties (topically reproducing the same statistical analysis for each set).

In 'compounded' format, assuming that the program accepts couples, the following array of ordered couples is generated `{(1.0,2.0),(4.0,5.0),(7.0,8.0)}`. Notice that this is a single dataset, made of couples of associated values.

When available, the "sequential" format is the default while the "tabular" format is activated with the option '-t'. See the 6 for the list of input format accepted by the different utilities.

2.2 Missing values and NaN management

When the conversion of an input entry to an internal floating point number cannot be performed, or when, on an input line, there are not enough values for the required "tabular" or "compounded" format, a NAN (not-a-number) value is generated. This approach is introduced to make possible the manipulation of files with an uneven number of entries in different columns or with "non numerical" values.

The following utilities automatically remove the `NaN` values from their input: `gbstat`, `gbdist` and `gbquant`.

The other utilities do handle `NaN` values as expected: if `NaN` values are present they typically return `NaN` output. In this case, the option `D` of the `gbget` utility is provided to remove all the lines containing `NaN` entries. This program can be used in a pipe like

```
...| gbget '()D' | ....
```

to treat the data before passing them to other `NaN`-sensitive utilities.

2.3 Radix and thousands separator

In addition to the radix symbol which separates the fractional and the integer part of the number, sometimes data are reported with a thousand separator symbol. For instance "one million" could be written "1,000,000.00". The character used to separate thousands and the fractional part are defined inside the C locale. Programs in the gbutils package can automatically recognize the locale settings and process these entries accordingly. Please use "gbenv" to see the definitions in use. Changing the locale typically amount simply to the redefinition of the LANG environment variable

```
# export LANG="en_US"
```

A list of the available locale can be obtained with the locale program

```
# locale -a
```

and the actual setting verified with

```
# locale
```

For more details refer to the locale documentation.

2.4 Output format and precision

In general, the output from the different programs is made of newline separated records of space separated fields of standard ASCII characters, which represent floating point numbers. The default format is scientific notation with a precision of six digits. The format and the precision can be changed using the environment variable `GB_OUT_FLOAT_FORMAT`. This variable can be set to any `printf` (the standard library C function) meaningful string. For instance with

```
# export GB_OUT_FLOAT_FORMAT="%.8e"
```

the precision is extended to eight digit. While with

```
# export GB_OUT_FLOAT_FORMAT="%.fe"
```

the scientific notation is replaced with a fixed-point notation. Please, refer to the `printf` documentation for further details.

There is also a second variable, `GB_OUT_EMPTY_FORMAT`, which can be used to tune the comment headings that many programs generate with the verbose option `-v`. Notice that it is automatically set to a value which is consistent with the float format chosen, so in general it is a good idea not to change it explicitly.

3 Numerical Error handling

The default behaviour of Gnu Scientific Library functions is to abort the execution of the program if a numeric error is produced. Some of these errors, especially underflow errors, are tolerable inside a computation. The 'gbutils' package provides a way of switching off the GSL error handling. It is sufficient to set the environment variable `GB_ERROR_HANDLER_OFF` using

```
# export GB_ERROR_HANDLER_OFF=
```

and all the programs will ignore numerical errors. This feature must be used carefully, after checking that the loss of precision implied by the presence of these errors can be considered tolerable for the actual computation one wants to perform. The default behaviour can be recovered using

```
# unset GB_ERROR_HANDLER_OFF
```

4 Binary format

THIS IS AN EXPERIMENTAL FEATURE

Like ASCII files, the binary files are structured as sequences of separate blocks. Each block is made of

- one `size_t` with number of columns C
- C `size_t` with the length of the rows, $R_1 \ldots R_C$
- the data stored sequentially column by column, for a total number equal to $R_1+R_2+\ldots+R_C$

This structure allows the storage of non matrices structures in binary format. If lengths are different, the missing values are replaced with NANs. This mimic the behaviour of ASCII data handling.

Notice that blocks are simply written one after the other. No particular separators are inserted between them.

Implementation: the option `-b` redefines the function used to read and/or write data.

This feature has been implemented for `gbget`, `gbmstat` and `gbfun`.

5 Graphic output

As previously mentioned, the output of many programs in the gbutils package, like gbhisto or gbker, is intended to be plotted and not directly read from the terminal. It is generally composed of records and fields of standard ASCII characters. This type of output can be displayed using the various plotting utilities commonly available in Unix systems. We shortly review below three possibilities.

5.1 GNU plotutils package

The plotutils package can be found here. It contains the program `graph` which generate a plot starting from input data. For example to obtain a plot of the kernel density of the data in file `datafile.dat` one can use

```
gbker < datafile.dat | graph -T x
```

where `-T x` choose an xwindow as output device.

5.2 Gnuplot interactive session

An alternative is to use the powerful plotting environment provided by gnuplot. The program can be found here.

From inside a gnuplot session, the previous kernel density can be obtained with

```
plot "< gbker < datafile.dat "
```

see Gnuplot documentations for details.

5.3 Gnuplot's plot from command line

As the example above shows, in order to directly plot the output of a command, inside gnuplot you need to put it inside a special string delimited by ˜"<˜ and ˜"˜. Moreover, all double quotes symbols ˜"˜ have to be escaped. These requirements can lead to cumbersome expressions when complicated commands are necessary. In any case, starting an interactive gnuplot session and writing the expression whose output should be plotted doesn't seem so attractive when one needs fast, simple plotting, for exploratory purposes. For these case the command `gbplot` is provided. This is a shell script that accept the data to be plotted as input and the directive on how to plot it on the command line.

The basic usage is as follows

```
gbplot [options] [plot|splot] <plotting options> < datafile
```

or

```
command pipe | gbplot [options] [plot|splot] <plotting options>
```

The command `plot` or `splot` are required. One can provide further plotting options by inserting them after these command. For example one can plot the kernel density estimate using

```
gbker < datafile.dat | gbplot plot
```

In this way the density is plotted using simple points. To use the fancier gnuplot's 'histeps' style use instead

```
gbker < datafile.dat | gbplot plot with histeps
```

The syntax of the plotting options is exactly the same that would be used inside gnuplot, after a the `plot` or `splot` command. For instance to specify a range for the x values use

```
gbker < datafile.dat | gbplot plot '[-1:1]' with histeps
```

It is also possible to obtain multiple plots of the data using the gnuplot special file name '`""`', as in

```
gbker < datafile.dat | gbplot plot 'w p , "" w l'
```

This command draws the kernel estimate two times: the first with points, the second with a line (as specified by the `w l` expression).

`gbplot` also possesses several options. They must be specified before the `plot` or `splot` command. To insert a title in the plot use the option `-t`

```
gbker < datafile.dat | gbplot -t Title plot with histeps
```

Terminal type and output file can be specified with the `-T` and `-o` options respectively. The command

```
gbker < datafile.dat | gbplot -T pdf -o output.pdf plot with histeps
```

produce a pdf version fo the plot and save it in 'output.pdf'.

Finally, if an interactive manipulation of plot parameters or data is required, you can use the option `-i`. This option opens an interactive gnuplot session, allowing for direct manipulation of plot settings and parameters

```
gbker < datafile.dat | gbplot -i plot with histeps
```

Once the session is closed, the output is saved in a file using a specific terminal if options `-o` and `-T` have been specified.

6 Programs summary table

Name	Input Type	External lib	NAN
gbget	c+	(matheval)	*
gbfun	c+	matheval	
gbgrid	no	matheval	
gbrand	no	gsl	
gbboot	s,t,c+	(gsl)	
gbenv	no		
gbmave	s,t		*
gbinterp	c,2	gsl	
gbfilternear	c+		
gbdist	s,t		*
gbstat	s,t		*
gbquant	s,t,c+		*
gbhisto	s,t		
gbker	s	gsl	
gbnear	s		
gbhisto2d	c2		
gbgcorr	s		
gbacorr	c1,c2		
gbxcorr	c2		
gbker2d	c2		
gbbin	c+		
gbtest	c+	(gsl)	*
gbmodes	s	gsl	
gbbin	t		
gbkreg	c2	gsl	
gbkreg2d	c3		
gblreg	c2	gsl	
gbglreg	c+	gsl	
gbnlreg	c+	gsl,matheval	
gbnlqreg	c+	gsl,matheval	
gbhill	s	gsl	
gbnlmult	c+	gsl,matheval	
gbnlprobit	c+	gsl,matheval	
gbnlpanel	c+	gsl,matheval	*
gbnlpolyit	c+	gsl,matheval	

Input Type: ’s’ sequential; ’t’ tabular; ’c’ compounded c2 read couples, c3 triplets, c+ a variable number of columns; 'no' no input required

External libs: gsl: Gnu Scientific Library matheval: GNU matheval library () means optional dependence (special features are available only if the library is found)

NAN: program automatically ignores NAN values in computations