gbget tutorial

Giulio Bottazzi

December 27, 2014

Contents

1 Introduction to `gbget` **1**

2 Examples **2**

1 Introduction to `gbget`

The special utility `gbget` is used to perform basic manipulation of ASCII data files. It takes a file or a list of files containing tabular data and extract command line specified parts of these data. The result of the extraction can be further manipulated with a list of possible transformations.

Essentially, the user provides a list of *data specifications* at the command line. The program analyze each specification in turns and, according to it, extract and print data to standard output.

Each *data specification* has the following structure

```
filename[block](col-range,rows-range)<options>
```

where:

filename is the name of a regular (ASCII) file

block identify a given data-block inside the file; data-blocks are separated by 2 or more empty lines (format chosen for consistency with gnuplot specifications). Notice that comment lines (beginning with a "#" symbol) DO NOT count as empty lines for this purpose.

col-range the range of required columns, specified as `begin:end:skip`. Negative values are counted from the end. Default is `1:-1:1` i.e. all columns. If begin>end the columns are read in reversed order.

rows-range the range of required rows, specified as `begin:end:skip`. Negative values are counted from the end. Default is `1:-1:1` i.e. all rows. If begin>end the rows are read in reversed order.

<**options**> is a list of single letter options that identify successive transformations to be applied to data.

The list of options includes

t transpose the matrix

f flatten the data column-wise, reducing them to a single column

l take the log of all fields

d take the column-wise difference: substract column 1 from 2, column 2 from 3, etc.

D remove all lines containing at least one NAN entry

z remove the mean to each column

Z reduce each column to zscores, that is remove the mean and divide by the standard deviation

2 Examples

A few examples can help to understand the gbget syntax. Consider the file `test.dat` with the following content

```
10 20 30
11 21 31
12 22 32
13 23 33
14 24 34


15 25 35
16 26 36
17 27 37
18 28 38
19 29 39
```

i.e. two data blocks, separated by two blank lines, each made of three columns and 5 rows. This file should already exists in the `gbutils` source directory. If the support for zlib has been found on your system, in the examples below you can equivalently use the compressed file `test.dat.gz`.

Now if you type:

```
# gbget 'test.dat[1](1:2)'
```

(the ~'~ are normally required to protect the content of the string from shell expansion) you obtain

```
1.000000e+01  2.000000e+01
1.100000e+01  2.100000e+01
1.200000e+01  2.200000e+01
1.300000e+01  2.300000e+01
1.400000e+01  2.400000e+01
```

i.e. the first two column `(1:2)` of the first datablock `[1]`. By default, the output is in scientific notation, but more on this below.

If instead you type

```
# gbget 'test.dat[2](-2:,2:3)'
```

you obtain

```
2.600000e+01  3.600000e+01
2.700000e+01  3.700000e+01
```

i.e. the rows from 2 to 3 (included) of the last two columns of the second data block `[2]`. Notice that a negative entry in column or row specification means "count from the end".

You can also print *each second column* of the second block with

```
# gbget 'test.dat[2](::2)'
```

that gives you

```
1.500000e+01  3.500000e+01
1.600000e+01  3.600000e+01
1.700000e+01  3.700000e+01
1.800000e+01  3.800000e+01
1.900000e+01  3.900000e+01
```

or *each third row* of the whole file

```
# gbget 'test.dat(,::3)'
```

to have

```
1.000000e+01  2.000000e+01  3.000000e+01
1.300000e+01  2.300000e+01  3.300000e+01
1.600000e+01  2.600000e+01  3.600000e+01
1.900000e+01  2.900000e+01  3.900000e+01
```

If the initial and final positions in a slice specification are reversed, the columns or the rows are printed in reverse order. For instance

```
# gbget 'test.dat(,3:1)'
```

gives

```
1.200000e+01  2.200000e+01  3.200000e+01
1.100000e+01  2.100000e+01  3.100000e+01
1.000000e+01  2.000000e+01  3.000000e+01
```

Several different transformations can be applied to the chosen *matrix* of data. For instance, the matrix can be flattened column-wise, i.e. each column can be put after the previous one in a single, long, column using the flag `f`, or you can transpose it using the option `t`. Let see some examples. Consider

```
# gbget 'test.dat[1](,2:3)'
```

which gives

```
1.100000e+01  2.100000e+01  3.100000e+01
1.200000e+01  2.200000e+01  3.200000e+01
```

now you can flatten the output

```
# gbget 'test.dat[1](,2:3)f'
```

```
1.100000e+01
1.200000e+01
2.100000e+01
2.200000e+01
3.100000e+01
3.200000e+01
```

or transpose it

```
# gbget 'test.dat[1](,2:3)t'

1.100000e+01  1.200000e+01
2.100000e+01  2.200000e+01
3.100000e+01  3.200000e+01
```

Finally, the output format can be customized using options `-o` and `-e`. To fully exploit these options you need to know the syntax of the `printf` command implemented in the standard C libraries. However, the *type* of output can be easily chosen with a single flag. If instead of the scientific notation you prefer a fixed point notation for your output, you have to use the option `-o` with the value `%f`

```
# gbget 'test.dat[1](,2:3)t' -o ' %f'

11.000000 12.000000
21.000000 22.000000
31.000000 32.000000
```

while in order to have an *integer* output you need `-o` with the value `%d`

```
# gbget 'test.dat[1](,2:3)t' -o ' %d'

11 12
21 22
31 32
```

in this case, however, be aware of the truncations: gbget rounds a non-integer value down to the nearest integer!