# HowTo BGX

Ernest Turro
Imperial College London

April 25, 2007

## 1 Introduction

This vignette describes how to use *bgx*, a C++ implementation of a Bayesian hierarchical integrated approach to the modelling and analysis of Affymetrix GeneChip arrays. The model and methodology is described in Hein et al, 2005.

There are two ways to run *bgx*: (1) through R and (2) as a standalone binary. Both ways make use of probe level GeneChip data, which you must obtain as GeneChip CEL files.

## 2 Reading in the CEL files

When you load *bgx*, several required packages from the Bioconductor[1] project are automatically loaded.

```
> library(bgx)
```

The *affy* package allows you to read CEL files into an `AffyBatch` object. This can be achieved by changing your working directory to wherever the CEL files are stored and executing:

```
> aData <- ReadAffy()
```

This will read in the CEL files in alphabetical order and save the data in the `aData` object. Alternatively, you can specify the specific files you would like to read in by adding their paths to the argument list, for example:

```
> aData <- ReadAffy("CEL/choe/chipC-rep1.CEL", "CEL/choe/chipS-rep2.CEL")
```

[1] `http://bioconductor.org`

# 3 Running BGX through R

A basic execution of the program can be performed by simply passing an `AffyBatch` object as a single parameter to the `bgx` function and saving the result in an `ExpressionSet` object. The result will hold array-specific gene expression values and their corresponding standard errors in `assayData(eset)$exprs` and `assayData(eset)$se.exprs` respectively.

```
> eset <- bgx(aData)
```

A more elaborate scenario would involve splitting the arrays into a number of conditions using the *samplesets* argument[2]; specifying which genes to analyse with the *genes* argument; specifying whether to take into account probe affinity with *probeAff*; setting the number of burn-in and post burn-in runs with the *burnin* and *iter* arguments respectively; setting the set of parameters to save with the *output* argument[3]; and specifying where to save the runs with *rundir*. Execute `help(bgx)` in R for a full explanation of all the parameters.

As an example, let us analyse the `Dilution` data set and save the results in the current working directory (”.”):

```
> library(affydata)
> library(hgu95av2cdf)
> data(Dilution)
> eset <- bgx(Dilution, samplesets=c(2,2), probeAff=FALSE, burnin=2048, iter=8192,g
```

The `eset` object will contain gene expression information for each gene under each condition (not necessarily each array). You may obtain the gene expression measure using the `exprs` function. For instance:

```
> exprs(eset)[10:40,] # Shorthand for assayData(eset)\$exprs[10:40,]
```

```
         condition 1 condition 2
947_at       6.54882     6.23533
948_s_at     4.80074     4.45260
949_s_at     4.75990     4.54776
950_at       4.46098     4.23591
951_at       1.81786     2.75080
952_at       2.52115     2.29997
953_g_at     5.27994     4.87892
```

[2]Note that if your `AffyBatch` object contains information on the experimental design in the `phenoData` slot, you do not need to use the *samplesets* argument.

[3]*output* can be set to either ”minimal”, ”trace” or ”all”. See the documentation for an explanation of what these levels mean

```
954_s_at     6.36742     6.09736
955_at       6.62402     6.34845
956_at       7.01234     6.70974
957_at       4.62311     4.22943
958_s_at     5.53824     5.18073
959_at       1.61462     1.32862
960_g_at     5.22585     4.92409
961_at       1.61015     1.63794
962_at       2.23727     2.01095
963_at       4.59609     4.20270
964_at       4.28568     4.00659
965_at       2.14193     1.36846
966_at       4.45682     4.10201
967_g_at     4.89294     4.59490
968_i_at     3.08878     3.58380
969_s_at     4.74009     4.49441
970_r_at     6.31431     6.17404
971_s_at     2.83499     2.78558
973_at       4.38212     4.08772
974_at       1.70736     2.25615
975_at       4.38175     4.01188
976_s_at     2.73965     3.41333
977_s_at     4.89886     4.55122
978_at       3.03629     2.60538
```

Run `help(ExpressionSet)` in R for more information.

Note that *samplesets* should be set to an array specifying the number of replicates in each condition. If set to `(3,2)`, `bgx` will treat the first three arrays read into R as replicates under condition 1 and the next two as replicates under condition 2. You should make sure that all condition 1 files are read in first and all condition 2 files are read in second by `ReadAffy()`. You may check the order of the samples in your `AffyBatch` object by using the `sampleNames` function:

```
> sampleNames(Dilution)

[1] "20A" "20B" "10A" "10B"
```

# 4 Running BGX as a standalone binary

Occasionally it may be useful to run `bgx` as a standalone binary from the command line[4]. In this case, you should use the `standalone.bgx` function instead of the `bgx` function.

[4]You can compile it by tweaking 'src/Makefile.standalone' to your specifications and running 'make -f Makefile.standalone' from the 'src' directory.

It takes the same arguments as `bgx`, with the addition of *dirname*, which should specify where you would like to save the input files required by the standalone binary.

```
aData <- ReadAffy() # Read in 6 arrays across two conditions
                    # in alphabetical order
standalone.bgx(aData, samplesets=c(3,3), genes=c(1:650,1000:1200),
  burnin=16384, iter=65536, output="minimal",
  dirname="input-choe3replicates")
```

Once you have saved the input files, you should locate the binary, make sure it is executable[5], and pass the path to the newly created `infile.txt` file as a single argument. For example:

```
./bgx ../input-choe3replicates/infile.txt
```

# 5 Detailed analysis of the output

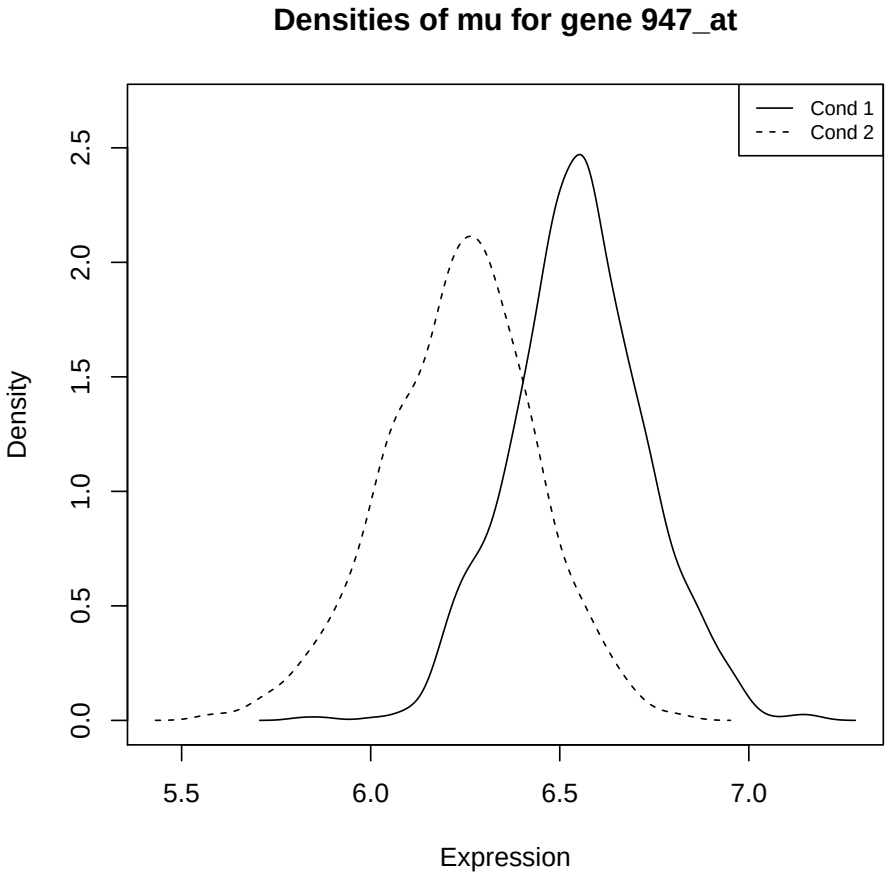
If you wish to analyse the output in detail, you should first read the output into a list as follows:

```
> bgxOutput <- readOutput.bgx("run.1")
```

You may then pass the `bgxOutput` object to any of several analysis functions. For instance, to view the gene expression distributions under the various conditions for gene 10, you could do:

```
> plotExpressionDensity(bgxOutput, gene=10)
```

[5]Under Unix-like environments, you can type `chmod +x bgx` at the command prompt to do this.




In order to get a list of ranked differential expression values, you could do:

```
> rankedGeneList <- rankByDE(bgxOutput)
> print(rankedGeneList[1:25,]) # print top 25 DEG
```

| | Position | DiffExpression |
|---|---|---|
| 956_at | 19 | 34.415585 |
| 941_at | 4 | 34.090607 |
| AFFX-HSAC07/X00351_5_at | 83 | 33.011335 |
| 947_at | 10 | 30.873117 |
| AFFX-HUMGAPDH/M33197_5_at | 89 | 30.510521 |
| 955_at | 18 | 27.775266 |
| AFFX-HUMGAPDH/M33197_M_at | 91 | 24.097000 |
| AFFX-HSAC07/X00351_M_at | 85 | 21.913618 |
| 954_s_at | 17 | 21.591047 |
| 953_g_at | 16 | 19.871139 |
| AFFX-HUMGAPDH/M33197_3_at | 87 | 19.282793 |
| AFFX-BioDn-3_at | 70 | 16.954794 |

| | | |
|---|---|---|
| 946_at | 9 | 16.395416 |
| 958_s_at | 21 | 15.611503 |
| AFFX-HUMISGF3A/M97935_3_at | 93 | 14.703255 |
| AFFX-HUMISGF3A/M97935_MB_at | 96 | 14.534467 |
| AFFX-HUMISGF3A/M97935_MA_at | 95 | 12.268148 |
| 957_at | 20 | 11.727123 |
| 960_g_at | 23 | 11.675859 |
| 977_s_at | 39 | 11.107148 |
| 993_at | 54 | 9.943220 |
| 963_at | 26 | 9.749545 |
| AFFX-HSAC07/X00351_3_at | 81 | 9.602072 |
| 969_s_at | 32 | 9.545740 |
| 982_at | 44 | 9.341472 |

Run `help(analysis.bgx)` for more detailed usage instructions on the analysis functions.