HowTo 'BGX'

Anne-Mette K. Hein, Imperial College London

Sept. 2005

1 Introduction

This note describes how to use 'bgx'. 'bgx' is 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 C++ program. These are described below. Both ways make use of probe level GeneChip data. To obtain the data in a format usable by 'bgx' you must have the GeneChip .CEL files and you must use the R-package 'affy' to process the CEL-files. This is also described below.

'bgx' produces samples of posterior distributions of each parameter in the 'bgx' model. When you are analysing many genes and many arrays there are a very large number of parameters in the model (2*'number of probes'+2*'number of genes' + a few more). Consequently the full output can be enormous. To accomodate this Varying degrees of detail in the output of a 'bgx' analysis can be chosen (see below). A number of R-functions are supplied that provide useful summaries of the output files in terms of plots or tables. There are many other sumamries and plots that maybe of interest and can be produced from the bgx output. The provided functions maybe used as inspiration for handling and making your own functions of choice for the 'bgx'-output. The available functions are described below.

2 Reading CEL files in to R

First you must read your CEL-files into R. This can be done as follows:

1. Start R
2. download the 'affy' library from the 'bioconductor'-website (http://www.bioconductor.org). You can do this by using the 'packages' dropdown menu, and choosing 'install packages from bioconductor'.
3. load the affy library:

```
library(affy)
```

4. Read in your CEL-files. You can e.g. do this by changing your working directory (using 'change dir' in the 'file' dropdown menu) to the directory where you keep the CEL-files you want to analyse and then use the 'ReadAffy' R function:

   mydata=ReadAffy()

   The CEL-files in your working directory are then read in to an affy-batch object which you have called 'mydata'. They will be read in in alphabetical order. Alternatively you can specify which CEL files are to be read in explicitly by specifying the path to them and their name as in: ReadAffy(files="path/to/filex/filex.CEL","path/to/filey/filey.CEL",...,"path/to/filew/filew.CEL") where 'path/to/file' specifies the directory and subdirectories where the CEL files are kept, and 'filex.CEL' is the name of a CEL file.

3 Running 'bgx' through R

R-functions that allow you to run 'bgx' through R is provided in the file 'bgx.R' which is kept in the 'R' sub-directory of the bgx directory. The functions make use of a .dll file, 'bgx.dll', which has been created from the C++ code and is provided in the directory bgx/bin. You need not concern yourself with this, but you must keep the structure of the bgx-directory as it is, in order that the R-functions can find the files they need. To run 'bgx' from R on your CEL-files you should proceed as follows:

1. First read in you CEL-files in R (go through steps 1 to 4 of 'Reading in CEL files to R'.)

2. change your working directory to be 'bgx'.

3. source in the bgx-R functions in the 'bgx.R' file:

   source("R/bgx.R")

4. You have now available the R-functions written in the file bgx.R. To run bgx you need to use the 'bgx' function. A call of the 'bgx'-function may look as follows

   bgx(mydata,run="torture",samplesets=c(2,2),genes=myGenes,genesToWatch=NULL,serial=FALSE)

   Here 'mydata' must be an affy-batch object (as the one you obtained when you read in your CEL files using ReadAffy()). The parameter 'run' specifies the run-length and detail of output and must take one

of the values "torture", "diagnostic", "full", "long", "confirm" or "explore". To see what each of these signify have a look in the file bgx.R or type 'bgx' in you R-window and hit the enter-key and look at the bgx-function code scroll over the R-window. The parameter 'samplesets' specifies the number of replicates in each condition. If specified as 'c(2,2)' it will treat the first two arrays read into R as replicates under condition 1, and the next two as replicates under condition 2. Note that you have to make sure that the CEL-files are read in by ReadAffy so that all condition 1 files are read in first, followed by all condition 2 files, etc. (see the preceding section for how to read in CEL-files). You can have an unequal number of replicates, and just one replicate under a condition, and you can analyse more than two conditions. E.g. samplesets=c(1), samplesets=c(2), samplesets=c(1,1) and samplesets=c(2,1,4) are valid options. The parameter 'genes' allows you to specify a subset of genes to be analysed. If 'genes' is not specified all genes will be analysed. Example subsets of genes are: c(1:10), c(3:104), seq(1, 100, by=10), c(3,35,88). In the R-language these correspond to (1,2,$\cdots$,10), (3,4,$\cdots$,104), (1,11,21,$\cdots$,91) and (3,35,88). The parameter genesToWatch allows you to specify a subset of genes for which samples from the full posterior distributions of ALL related parameters (including S_{gj} and H_{gj} are collected. As this creates a very large amount of output it should be left as NULL in general, and if used should be used for only a few genes as the amount of amount generated is extensive. The implementation of functions to produce standard plots that use the extensive output is in progress. The parameter 'serial' allows each of the CEL files in the 'mydata' affy-batch object to be analysed separately as one replicate under one condition, one by one.

4 Running 'bgx' as a C++ standalone program

The C++ 'bgx' program takes five input files: 'infile', 'PM.txt', 'MM.txt', 'SS.txt' and 'PS.txt'. These are produced by running the R-function 'standalone.bgx'.

1. First read in you CEL-files in R (go through steps 1 to 4 of 'Reading CEL files in to R'.)

2. source in the R-functions in the file 'bgx.R' (go through steps 2 and 3 in 'Running 'bgx' through R').

3. run the standalone.bgx function. E.g.:

   >standalone.bgx(mydata,run="torture",samplesets=NULL,genes=NULL,prefix="")

   The parameters are similar to those of the bgx-function. The required files for use with the C++ standalone program are produced. For further description see the README file in the bgx-directory.

5 How to use the bgx output

'bgx' produces a number of files, depending on the value of the parameter 'run'. The file 'bgxPosteriorPlots.R in the 'R' directory contains a number of R-functions that use output files which are produced when the output level of the run is "diagnostic", that is, when the 'run' parameter is set to "torture", "experiment", "full" or "long". Some of the functions may also be used with less extensive output files. A description of each of the available functions are given below. Many other plots and tables may be of interest and produced from the output files by creating your functions. WHEN RUNNING THESE R-FUNCTIONS YOUR WORKING DIRECTORY MUST BE THE DIRECTORY CONTAINING THE OUTPUT FILES.

densityPlotsLogPMsandMMs The function plots kernel-density plots of the sets of log(PM), of log(MM) values and of log(PM-MM) values (with PM>MM only). One curve is produced for each array. The function has two parameters: 'data' and 'genes'. Data should be an affy-batch object, genes the list of genes whose probes should be included ('genes=NULL' means all genes). Example call:

densityPlotsLogPMsandMMs(mydata,genes=c(1:1000))

densityPlotsLogSandHandmu Plots kernel density plots for $\log(S+1)$, $\log(H+1)$ and μ. For $\log(S+1)$ and $\log(H+1)$: one per array, for μ: one per condition. Condition specific line types are used. There are no parameters to this function. Example call:

densityPlotsLogSandHandmu()

muMAplots Plots mean(μ_{c_i})-mean(μ_{c_j}) against 0.5*(mean(μ_{c_i})+mean(μ_{c_j})) for all pairs of conditions (c_i, c_j), with $c_j = 1, ...,$numberConditions and $c_i = c_j + 1, ...,$numberConditions. A loess curve is fitted and plotted to each plot. There are no parameters to this function. Example call:

muMAplots()

createMuSummaryTable This function producing a table with summaries of the analysis. The table has one row per gene and the following columns:

row 1: the gene number (obtained when reading CEL files in R)

row 2: an indicator for whether the gene was included in the 'genesToWatch' list. 1: if yes, 0: if no (default)

rows 3 and 4: mean and sd of μ_1

... ...

rows 2+2*C-1 and 2+2*C: mean and sd of μ_C (C: number conditions analysed)

rows 2+2*C+1 to 2+2*C+8: mean($\mu_2-\mu_1$), sd($\mu_2-\mu_1$), mean($\mu_2-\mu_1$)/sd($\mu_2-\mu_1$), rank(abs(mean($\mu_2-\mu_1$)/sd($\mu_2-\mu_1$))), mean($\mu_2-\mu_1$-loess)/sd($\mu_2-\mu_1$),rank(abs(mean($\mu_2-\mu_1$-loess)/sd($\mu_2-\mu_1$))), loess, Posterior Probability of $\mu_2-\mu_1$-loess¿0

... ...

rows 2+2*C+numberPairs*8-7 to 2+2*C+numberPairs*8: as above but for the last pair.

Here 'loess' is the loess fitted curve to the MA plot of mean(μ_{cond_2}) on mean(μ_{cond_1}), and 'last pair' is the last pair when the pairs of conditions are ordered as: (1,2), (1,3), ... , (1,C), (2,3),(2,4), ... , (2,C), ..., (C-1,C). There are no parameters to this function. Example call:

createMuSummaryTable

makePostDistrMuBGXindex produces a table displaying quantiles of the posterior distribution for the bgx gene expression index for a given condition. The table has the format: row(=gene) by column(=quantile). Quantiles are: 2.5, 5, 10, 25, 33, 50, 66, 75, 90, 95, 97.5 %. The function takes one parameter: the condition. Example call: the following produces a table containing quantiles of the posterior distribution of gene expression for the genes under condition 2:

makePostDistrMuBGXindex(2)

makePostDistrMuBGXPairDiff : the function produces a table displaying quantiles of the posterior distribution of differential expression between two conditions (cond1-cond2) after subtraction of the loess normalisation curve. This curve is obtained from the MA plot for the

mean μ values. The conditions, cond1 and cond2, are parameters to the function. The table format is as for the function 'makePostDistrMuBGXindex'. The following example function call creates a table displaying the posterior distributions of the differences in expression between condition 3 and 1 after subtraction of the loess normalisation curve:

makePostDistrMuBGXPairDiff(3,1)

makeMuMAPlotCond2againstCond1 : The function creates two plots: an MAplot of condy against condx and a normalised MAplot of condy against condx using the mean μ values. The function takes two parameters condx and condy. NOTE: TO RUN THIS FUNCTION YOU MUST FIRST HAVE RUN THE FUNCTION createMuSummaryTable(). Example function call:

makeMuMAPlotCond2againstCond1(3,2)

addkhighestRankedPointsToMuMAplot THIS function is meant to be used after an MA plot has been produced using the function makeMuMAPlotCond2againstCond1. It will (re)plot the points for the k highest ranked genes in the existing MA plot. It takes five parameters: condx and condy which specifies the conditions. 'ranking' which must be "ratio" or "PPPD" and specifies the criterion on which the genes are to be ranked: "ratio"=(E(diff)-loess)/SD(diff) or "PPPD"=**p**osterior **p**robability of **p**ositive **d**ifference. 'K' which specifies the number of points the be plotted (e.g. for the k=10 highest ranked genes) and 'colour' which specifies the colour of the points to be added. Example function calls are:

makeMuMAPlotCond2againstCond1(1,2)

addkhighestRankedPointsToMuMAplot(1,2,"ratio",100,red)

addkhighestRankedPointsToMuMAplot(1,2,"ratio",10,yellow)

plotCredibilityIntervalforDiff The function has five parameters: 'condx', 'condy', 'size', 'first', 'last' and 'ranking'. The function plots stacked credibility intervals of the differences in expression between condition 'condx' and 'condy' (with the loess curve subtracted) of genes 'first' to 'last' when genes are ordered using 'ranking'. NOTE! TO RUN THIS FUNCTION YOU MUST FIRST HAVE RUN THE FUNCTION createMuSummaryTable(). The width of the credibility interval is given

by 'size'. Possible values are: 1, 2, 3, 4, 5, 6 and 7 corresponding to: 2.5-97.5, 5-95, 10-90, 15-85, 20-80, 25-75, 33-67%. Numbers of genes with credibility intervals for difference in expression that do not cover zero are denoted in the right hand margin in red. To find the gene names look in the corresponding rows (row number: gene number+1) in the geneNames.cvs table. Example function call which will plot the credibility interval of size '3' of the differences in expression between condition 1 and 2 for the genes that are ranked as the 50 to 100 highest using the posterior probability of positive difference as a ranking criterion.

plotCredibilityIntervalforDiff(1,2,3,50,100,"PPPD")

Example

```
library(affy)
mydata=ReadAffy()
help1=c(3035,4090,4040,3732,4486,7176,4363,4818,9283,7298,4439,5096,6686)
help2=c(5219,12212,9096,4794,3944,4925,9222,8847,5586,5317,193,7162,7061)
help3=c(3957,2998,4478,7481,22278,22279:22285,22225,22242,22248,22295)
spikeGeneNumbers=c(help1,help2,help3)
mygenes=c(seq(1,22300,by=100),spikeGeneNumbers)
source("R/bgx.R")
bgx(mydata,genes=mygenes,samplesets=c(3,3))
source("R/bgxPosteriorPlots.R")
densityPlotsLogPMsandMMs(mydata,mygenes)
densityPlotsLogSandHandmu()
muMAplots()
muMAplotsAfterLoess()
createMuSummaryTable()
makePostDistrMuBGXindex(2)
makePostDistrMuBGXPairDiff(1,2)
addkhighestRankedPointsToMuMAplot(1,2,"ratio",50,"blue")
addkhighestRankedPointsToMuMAplot(1,2,"ratio",40,"lightblue")
addkhighestRankedPointsToMuMAplot(1,2,"ratio",30,"green")
addkhighestRankedPointsToMuMAplot(1,2,"ratio",20,"yellow")
addkhighestRankedPointsToMuMAplot(1,2,"ratio",10,"red")
```