puma User Guide

R. D. Pearson, X. Liu, M. Rattray, M. Milo, N. D. Lawrence, G. Sanguinetti

May 31, 2009

1 Abstract

Most analyses of Affymetrix GeneChip data are based on point estimates of expression levels and ignore the uncertainty of such estimates. By propagating uncertainty to downstream analyses we can improve results from microarray analyses. For the first time, the *puma* package makes a suite of uncertainty propagation methods available to a general audience. *puma* also offers improvements in terms of scope and speed of execution over previously available uncertainty propagation methods. Included are summarisation, differential expression detection, clustering and PCA methods, together with useful plotting functions.

2 Citing *puma*

The *puma* package is based on a large body of methodological research. Citing *puma* in publications will usually involve citing one or more of the methodology papers (1; 2; 3; 4; 5; 6) that the software is based on as well as citing the software package itself. For the methodology papers, see http://www.bioinf.manchester.ac.uk/resources/puma/. *puma* makes use of the donlp2() function (7) by Peter Spellucci. The use of donlp2() must be acknowledged in any publication which contains results obtained with puma or parts of it. Citation of the author's name and netlib-source is suitable. The software itself as well as the extension of PPLR to the multi-factorial case (the `pumaDE` function) can be cited as:

puma: a Bioconductor package for Propagating Uncertainty in Microarray Analysis (2007) Pearson et al. In preparation

3 Introduction

Microarrays provide a practical method for measuring the expression level of thousands of genes simultaneously. This technology is associated with many significant sources of experimental uncertainty, which must be considered in order to make confident inferences from the data. Affymetrix GeneChip arrays have multiple probes associated with each target. The probe-set can be used to measure the target concentration and this measurement is then used in the downstream analysis to achieve the biological aims of the experiment, e.g. to detect significant differential expression between conditions, or for the visualisation, clustering or supervised classification of data.

Most currently popular methods for the probe-level analysis of Affymetrix arrays (e.g. RMA, MAS5.0) only provide a single point estimate that summarises the target concentration. Yet the probe-set also contains much useful information about the uncertainty associated with this measurement. By using probabilistic methods for probe-level analysis it is possible to associate gene expression levels with credibility intervals that quantify the measurement uncertainty associated with the estimate of target concentration within a sample. This within-sample variance is a very significant source of uncertainty in microarray experiments, especially for relatively weakly expressed genes, and we argue that this information should not be discarded after the probe-level analysis. Indeed, we provide a number of examples were the inclusion of this information gives improved results on benchmark data sets when compared with more traditional methods which do not make use of this information.

PUMA is an acronym for Propagating Uncertainty in Microarray Analysis. The *puma* package is a suite of analysis methods for Affymetrix GeneChip data. It includes functions to:

1. Calculate expression levels and confidence measures for those levels from raw CEL file data.
2. Combine uncertainty information from replicate arrays
3. Determine differential expression between conditions, or between more complex contrasts such as interaction terms
4. Cluster data taking the expression-level uncertainty into account
5. Perform a noise-propagation version of principal components analysis (PCA)

There are a number of other Bioconductor packages which can be used to perform the various stages of analysis highlighted above. The *affy* package gives access to a number of methods for calculating expression levels from raw CEL file data. The *limma* package provides well-proven methods for determination of differentially expressed genes. Other packages give access to clustering and PCA methods. In keeping with the Bioconductor philosophy, we aim to reuse as much code as possible. In many cases, however, we offer techniques that can be seen as alternatives to techniques available in other packages. Where this is the case, we have attempted to provide tools to enable the user to easily compare the different methods.

We believe that the best method for learning new techniques is to use them. As such, the majority of this user manual (Section 4) is given over to case studies which highlight different aspects of the package. The case studies include the scripts required to recreate the results shown. At present there is just one case study (based on data from the *estrogen* package), but others will soon be included.

One of the most popular packages within Bioconductor is *limma*. Because many users of the *puma* package are already likely to be familiar with *limma*, we have written a special section (Section 5), highlighting the similarities and differences between the two packages. While this section might help experienced *limma* users get up to speed with *puma* more quickly, it is not required reading, particularly for those with little or no experience of *limma*.

The main benefit of using the propagation of uncertainty in microarray analysis is the potential of improved end results. However, this improvement does come at the cost of increased computational demand, particularly that of the time required to run the various algorithms. The key algorithms are, however, parallelisable, and we have built this parallel functionality into the package. Users that have access to a computer cluster, or even a number of machines on a network, can make use of this functionality. Details of how this should be set up are given in Section 6. This section can be skipped by those who will be running *puma* on a single machine only.

We have chosen to leave details of individual functions out of this vignette, though comprehensive details can be found in the online help for each function.

This software package uses the optimization program donlp2 (7).

4 Introductory example analysis - estrogen

In this section we introduce the main functions of the *puma* package by applying them to the data from the *estrogen* package

4.1 Installing the *puma* package

The recommended way to install *puma* is to use the `biocLite` function available from the bioconductor website. Installing in this way should ensure that all appropriate dependencies are met.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("puma")
```

4.2 Loading in the data

The first step in any *puma* analysis is to load the package. Start *R*, and then type the following command:

```
> library(puma)
```

The next step in a typical analysis is to load in data from Affymetrix CEL files, using the `ReadAffy` function from the *affy* package. *puma* makes extensive use of phenotype data, which maps arrays to the condition or conditions of the biological samples from which the RNA hybridised to the array was extracted. It is recommended that this phenotype information is supplied at the time the CEL files are loaded. If the phenotype information is stored in the `AffyBatch` object in this way, it will then be made available for all further analyses.

The easiest way to supply phenotype information is in a text file that is loaded using the `phenotype` parameter of the `ReadAffy` function (see *affy* documentation or Case Studies within this document for more information). The phenotype text file that comes with the *estrogen* package is unfortunately not in the form required by `ReadAffy`, and so we will add phenotype information to the `AffyBatch` object directly using the `pData` method.

The data used in this example are also available in the *pumadata* package. As an alternative to loading data from CEL files for this example, simply type `library(pumadata)` then `data(affybatch.estrogen)` at the R prompt.

```
> datadir <- file.path(.find.package("estrogen"),"extdata")
> estrogenFilenames <- c("low10-1.cel", "low10-2.cel"
+     , "high10-1.cel", "high10-2.cel", "low48-1.cel"
+     , "low48-2.cel", "high48-1.cel", "high48-2.cel")
> affybatch.estrogen <- ReadAffy(
+     filenames=estrogenFilenames
+ ,   celfile.path=datadir
+ )
> pData(affybatch.estrogen) <- data.frame(
+     "estrogen"=c("absent","absent","present","present"
+         ,"absent","absent","present","present")
+ ,   "time.h"=c("10","10","10","10","48","48","48","48")
+ ,   row.names=rownames(pData(affybatch.estrogen))
+ )

> show(affybatch.estrogen)

The downloaded packages are in
        /var/folders/FQ/FQNbVutOGOaRYLbY7uRNEU+++TI/-Tmp-//Rtmp2Cv5TG/downloaded_packag
AffyBatch object
size of arrays=640x640 features (9 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=8
number of genes=12625
annotation=hgu95av2
notes=
```

Here we can see that `affybatch.estrogen` has 8 arrays, each with 12,625 probesets.

```
> pData(affybatch.estrogen)

             estrogen time.h
low10-1.cel    absent     10
low10-2.cel    absent     10
high10-1.cel  present     10
high10-2.cel  present     10
low48-1.cel    absent     48
low48-2.cel    absent     48
high48-1.cel  present     48
high48-2.cel  present     48
```

We can see from this phenotype data that this experiment has 2 factors (estrogen and time.h), each of which has two levels (absent vs present, and 10 vs 48), hence this is a 2x2 factorial experiment. For each combination of levels we have two replicates, making a total of 2x2x2 = 8 arrays.

4.3 Determining expression levels

We will first use multi-mgMOS to create an expression set object from our raw data. This step is similar to using other summarisation methods such as MAS5.0 or RMA, and for comparison purposes we will also create an expression set object from our raw data using RMA. Note that the following lines of code are likely to take a significant amount of time to run, so if you in hurry simply type `data(eset_estrogen_mmgmos)` and `data(eset_estrogen_rma)` at the command prompt.

```
> eset_estrogen_mmgmos <- mmgmos(affybatch.estrogen)
> eset_estrogen_rma <- rma(affybatch.estrogen)
```

Unlike many other methods, multi-mgMOS provides information about the expected uncertainty in the expression level, as well as a point estimate of the expression level.

```
> exprs(eset_estrogen_mmgmos)[1,]
```

```
 low10-1.cel  low10-2.cel high10-1.cel high10-2.cel
    7.044149     7.006220     6.387901     6.900364
 low48-1.cel  low48-2.cel high48-1.cel high48-2.cel
   10.117206     9.937288    10.696670    10.154695
```

```
> assayDataElement(eset_estrogen_mmgmos,"se.exprs")[1,]
```

```
 low10-1.cel  low10-2.cel high10-1.cel high10-2.cel
   0.5585693    0.5785603    0.7373298    0.6430139
 low48-1.cel  low48-2.cel high48-1.cel high48-2.cel
   0.2002344    0.2190787    0.1582523    0.2098834
```

Here we can see the expression levels, and standard errors of those expression levels, for the first probe set of the `affybatch.estrogen` data set.

4.4 Determining gross differences between arrays

A useful first step in any microarray analysis is to look for gross differences between arrays. This can give an early indication of whether arrays are grouping according to the different factors being tested. This can also help to identify outlying arrays, which might indicate problems, and might lead an analyst to remove some arrays from further analysis.

Principal components analysis (PCA) is often used for determining such gross differences. *puma* has a variant of PCA called Propagating Uncertinaty in Microarray Analysis Principal Components Analysis (pumaPCA) which can make use of the uncertainty in the expression levels determined by multi-mgMOS. Again, note that the following example can take some time to run, so to speed things up, simply type `data(pumapca_estrogen)` at the R prompt.

```
> pumapca_estrogen <- pumaPCA(eset_estrogen_mmgmos)
```

For comparison purposes, we will run standard PCA on the expression set created using RMA.

```
> pca_estrogen <- prcomp(t(exprs(eset_estrogen_rma)))
```

```
> par(mfrow=c(1,2))
> plot(pumapca_estrogen,legend1pos="right",legend2pos="top",main="pumaPCA")
> plot(
+         pca_estrogen$x
+ ,        xlab="Component 1"
+ ,        ylab="Component 2"
+ ,        pch=unclass(as.factor(pData(eset_estrogen_rma)[,1]))
+ ,        col=unclass(as.factor(pData(eset_estrogen_rma)[,2]))
+ ,        main="Standard PCA"
+ )
```

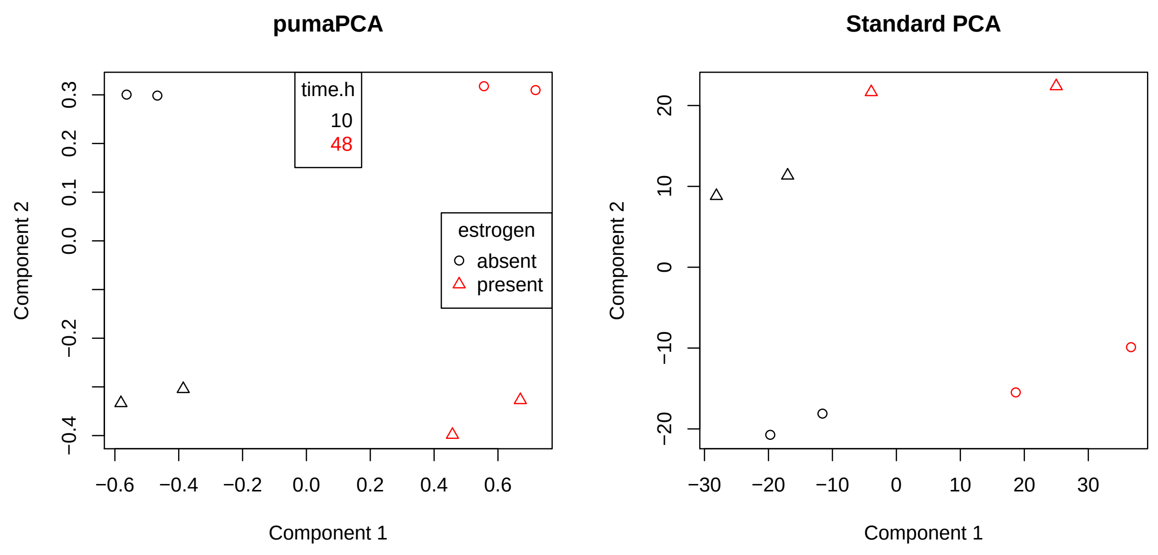

Figure 1: First two components after applying `pumapca` and `prcomp` to the `estrogen` data set processed by multi-mgMOS and RMA respectively.

It can be seen from Figure 1 that the first component appears to be separating the arrays by time, whereas the second component appears to be separating the arrays by presence or absence of estrogen. Note that grouping of the replicates is much tighter with multi-mgMOS/pumaPCA. With RMA/PCA, one of the absent.48 arrays appears to be closer to one of the absent.10 arrays than the other absent.48 array. This is not the case with multi-mgMOS/pumaPCA.

Before carrying out any further analysis, it is generally advisable to check the distributions of expression values created by your summarisation method. Like PCA analysis, this can help in identifying problem arrays. It can also inform whether further normalisation needs to be carried out. One way of determining distributions is by using box plots.

```
> par(mfrow=c(1,2))
> boxplot(data.frame(exprs(eset_estrogen_mmgmos)),main="mmgMOS - No norm")
> boxplot(data.frame(exprs(eset_estrogen_rma)),main="Standard RMA")
```

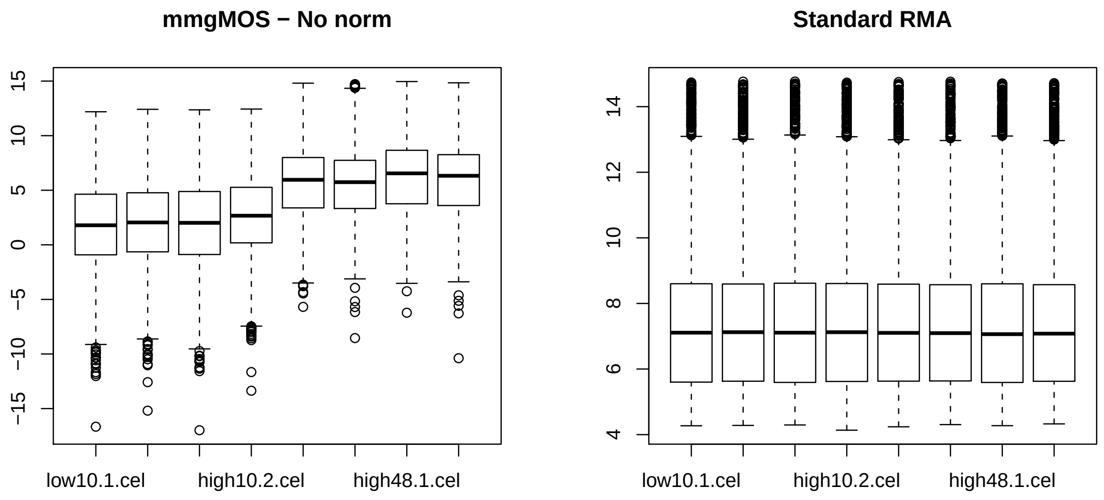

Figure 2: Box plots for `estrogen` data set processed by multi-mgMOS and RMA respectively.

From Figure 2 we can see that the expression levels of the time=10 arrays are generally lower than those of the time=48 arrays, when summarised using multi-mgMOS. Note that we do not see this with RMA because the quantile normalisation used in RMA will remove such differences. If we intend to look for genes which are differentially expressed between time 10 and 48, we will first need to normalise the mmgmos results.

```
> eset_estrogen_mmgmos_normd <- pumaNormalize(eset_estrogen_mmgmos)
> boxplot(data.frame(exprs(eset_estrogen_mmgmos_normd))
+     , main="mmgMOS - median scaling")
```

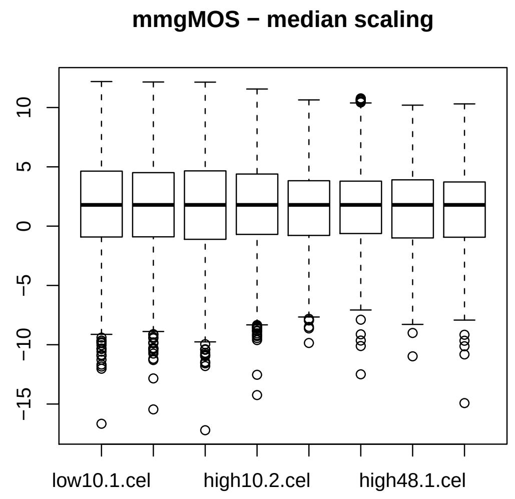

Figure 3: Box plot for `estrogen` data set processed by multi-mgMOS and normalisation using global median scaling.

Figure 3 shows the data after normalisation. We can now see that the distributions of expression levels are similar across arrays.

4.5 Identifying differentially expressed (DE) genes

There are many different methods available for identifying differentially expressed genes. *puma* incorporates the Probability of Positive Log Ratio (PPLR) method (5). The PPLR method can make use of the information about uncertainty in expression levels provided by multi-mgMOS. This proceeds in two stages. Firstly, the expression level information from the different replicates of each condition is combined to give a single expression level (and standard error of this expression level) for each condition. Note that the following code can take a long time to run. The end result is available as part of the *pumadata* package, so the following line can be replaced with `data(eset_estrogen_comb)`.

```
> eset_estrogen_comb <- pumaComb(eset_estrogen_mmgmos_normd)
```

Note that because this is a 2 x 2 factorial experiment, there are a number of contrasts that could potentially be of interest. *puma* will automatically calculate contrasts which are likely to be of interest for the particular design of your data set. For example, the following command shows which contrasts *puma* will calculate for this data set

```
> colnames(createContrastMatrix(eset_estrogen_comb))

[1] "present.10_vs_absent.10"
[2] "absent.48_vs_absent.10"
[3] "present.48_vs_present.10"
[4] "present.48_vs_absent.48"
[5] "estrogen_absent_vs_present"
[6] "time.h_10_vs_48"
[7] "Int__estrogen_absent.present_vs_time.h_10.48"
```

Here we can see that there are seven contrasts of potential interest. The first four are simple comparisons of two conditions. The next two are comparisons between the two levels of one of the factors. These are often referred to as "main effects". The final contrast is known as an "interaction effect".

Don't worry if you are not familiar with factorial experiments and the previous paragraph seems confusing. The techniques of the *puma* package were originally developed for simple experiments where two different conditions are compared, and this will probably be how most people will use *puma*. For such comparisons there will be just one contrast of interest, namely "condition A vs condition B".

To identify genes that are differentially expressed between the different conditions use the `pumaDE` function. For the sake of comparison, we will also determine genes that are differentially expressed using a more well-known method, namely using the *limma* package on results from the RMA algorithm.

```
> pumaDERes <- pumaDE(eset_estrogen_comb)
> limmaRes <- calculateLimma(eset_estrogen_rma)
```

The results of these commands are ranked gene lists. Suppose we are particularly interested in the interaction term. We saw above that this was the seventh contrast identified by *puma*. The following commands will identify the gene deemed to be most likely to be differentially expressed due to the interaction term by our two methods

```
> topLimmaIntGene <- topGenes(limmaRes, contrast=7)
> toppumaDEIntGene <- topGenes(pumaDERes, contrast=7)
```

Let's look first at the gene determined by RMA/limma to be most likely to be differentially expressed due to the interaction term

```
> plotErrorBars(eset_estrogen_rma, topLimmaIntGene)
```

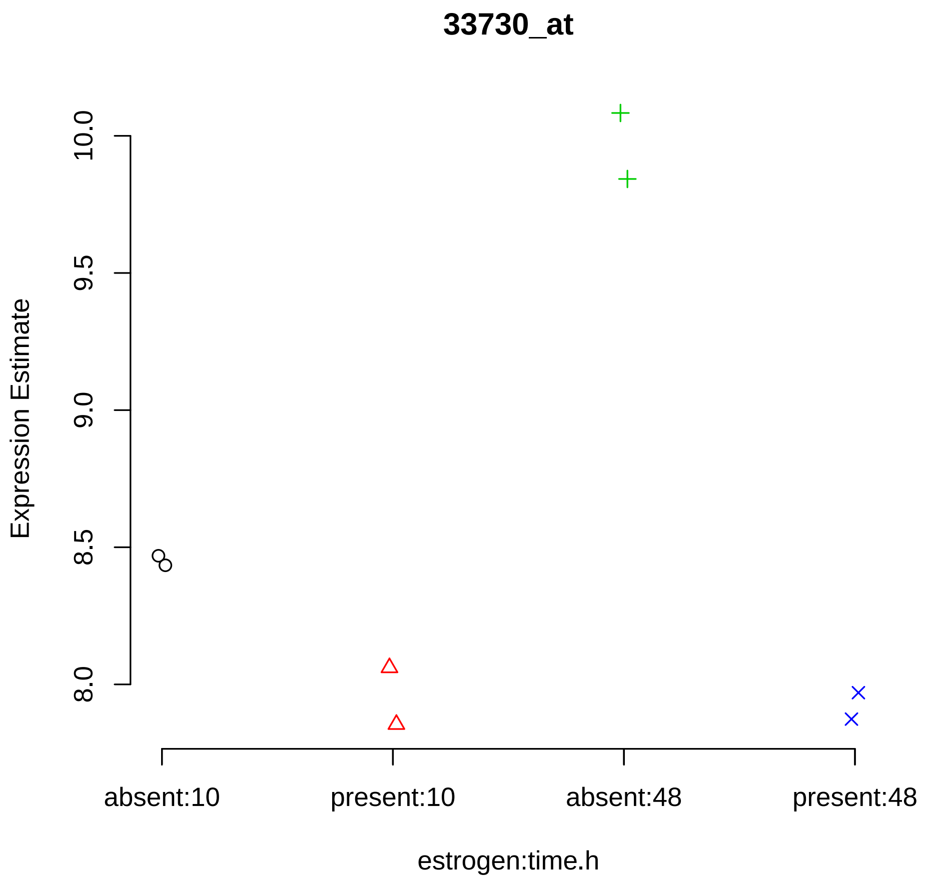

Figure 4: Expression levels (as calculated by RMA) for the gene most likely to be differentially expressed due to the interaction term in the `estrogen` data set by RMA/limma

The gene shown in Figure 4 would appear to be a good candidate for a DE gene. There seems to be an increase in the expression of this gene due to the combination of the estrogen=absent and time=48 conditions. The within condition variance (i.e. between replicates) appears to be low, so it would seem that the effect we are seeing is real.

We will now look at this same gene, but showing both the expression level, and, crucially, the error bars of the expression levels, as determined by multi-mgMOS.

```
> plotErrorBars(eset_estrogen_mmgmos_normd, topLimmaIntGene)
```

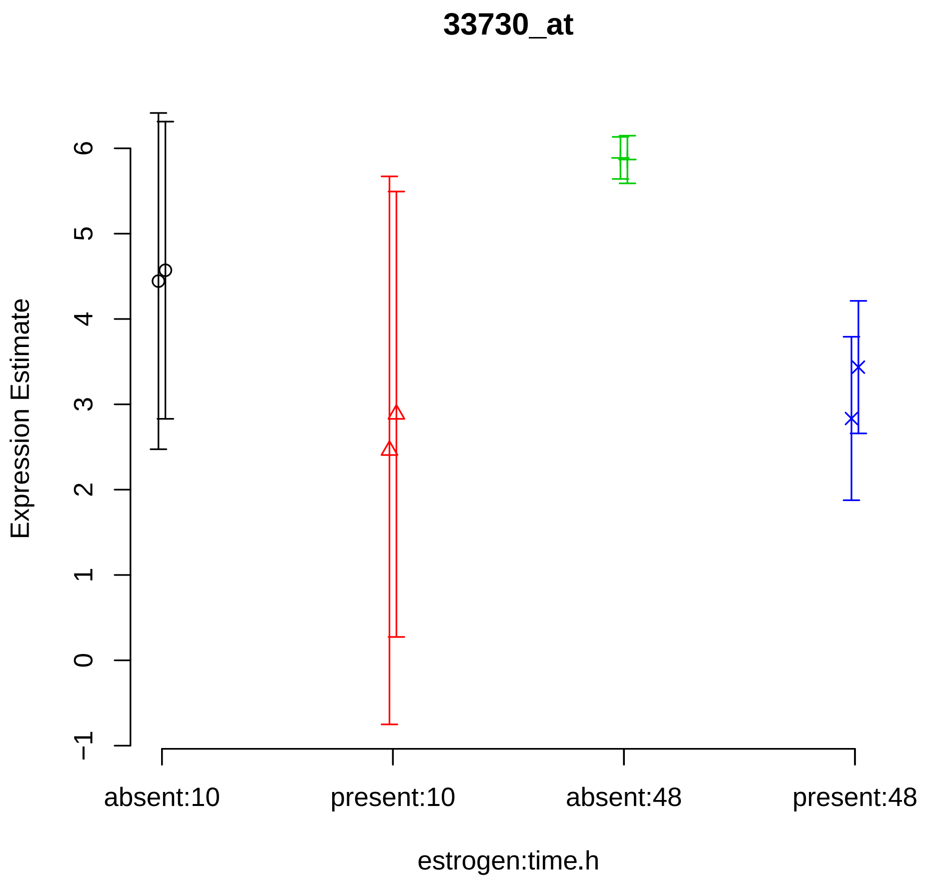

Figure 5: Expression levels and error bars (as calculated by multi-mgMOS) for the gene determined most likely to be differentially expressed due to the interaction term in the `estrogen` data set by RMA/limma

Figure 5 tells a somewhat different story from that shown in figure 4. Again, we see that the expected expression level for the absent:48 condition is higher than for other conditions. Also, we again see that the within condition variance of expected expression level is low (the two replicates within each condition have roughly the same value). However, from figure 5 we can now see that we actually have very little confidence in the expression level estimates (the error bars are large), particularly for the time=10 arrays. Indeed the error bars of absent:10 and present:10 both overlap with those of absent:48, indicating that the effect previously seen might actually be an artifact.

```
> plotErrorBars(eset_estrogen_mmgmos_normd, toppumaDEIntGene)
```

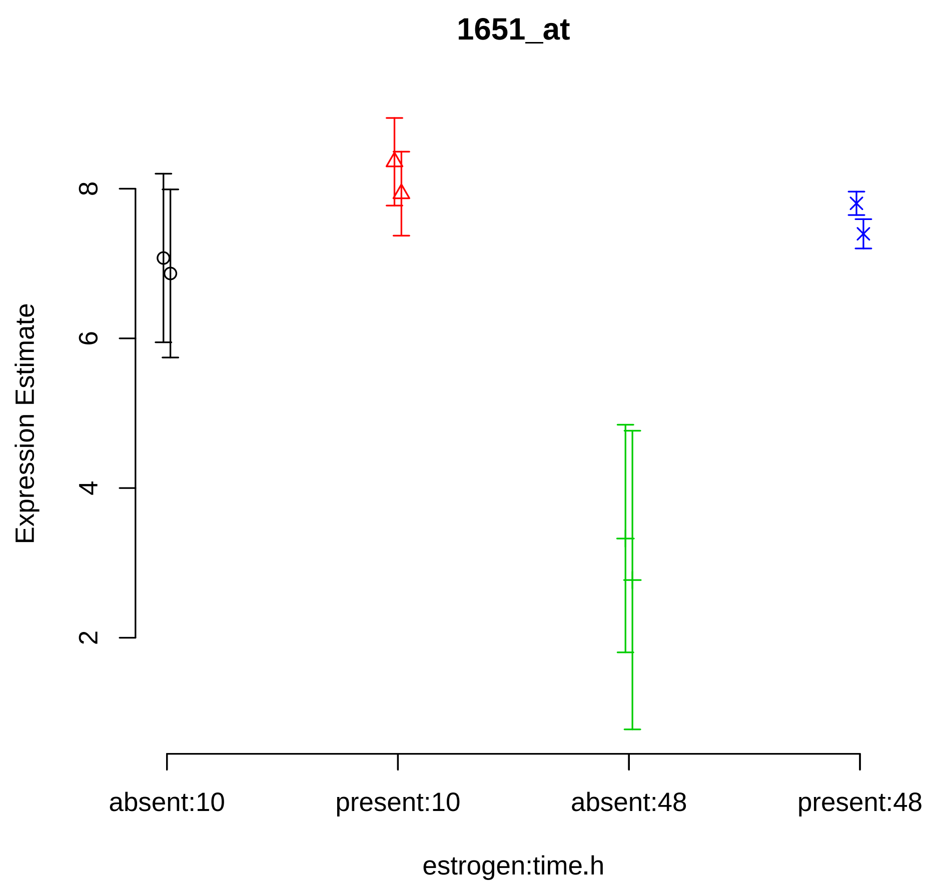

Figure 6: Expression levels and error bars (as calculated by multi-mgMOS) for the gene determined most likely to be differentially expressed due to the interaction term in the `estrogen` data set by mmgmos/pumaDE

Finally, figure 6 shows the gene determined by multi-mgMOS/PPLR to be most likely to be differentially expressed due to the interaction term. For this gene, there appears to be lower expression of this gene due to the combination of the estrogen=absent and time=48 conditions. Unlike with the gene shown in 5, however, there is no overlap in the error bars between the genes in this condition, and those in other conditions. Hence, this would appear to be a better candidate for a DE gene.

5 *puma* for *limma* users

puma and *limma* both have the same primary goal: to identify differentially expressed genes. Given that many potential users of *puma* will already be familiar with *limma*, we have consciously attempted to incorporate many of the features of *limma*. Most importantly we have made the way models are specified in *puma*, through the creation of design and contrast matrices, very similar to way this is done in *limma*. Indeed, if you have already created design and contrast matrices in *limma*, these same matrices can be used as arguments to the `pumaComb` and `pumaDE` functions.

One of the big differences between the two packages is the ability to automatically create design and contrast matrices within *puma*, based on the phenotype data supplied with the raw data. We believe that these automatically created matrices will be sufficient for the large majority of analyses, including factorial designs with up to three different factors. It is even possible, through the use of the `createDesignMatrix` and `createContrastMatrix` functions, to automatically create these matrices using *puma*, but then use them in a *limma* analysis. More details on the automatic creation of design and contrast matrices is given in Appendix A.

One type of analysis that cannot currently be performed within *puma*, but that is available in *limma*, is the detection of genes which are differentially expressed in at least one out of three or more different conditions (see e.g. Section 8.6 of the *limma* user manual). Factors with more than two levels can be analysed within *puma*, but only at present by doing pairwise comparisons of the different levels. The authors are currently working on extending the functionality of *puma* to incorporate the detection of genes differentially expressed in at least one level of multi-level factors.

puma is currently only applicable to Affymetrix GeneChip arrays, unlike *limma*, which is applicable to a wide range of arrays. This is due to the calculation of expression level uncertainties within multi-mgMOS from the PM and MM probes which are specific to GeneChip arrays.

6 Parallel processing with *puma*

The most time-consuming step in a typical *puma* analysis is running the `pumaComb` function. This function, however, operates on a probe set by probe set basis, and therefore it is possible to divide the full set of probe sets into a number of different "chunks", and process each chunk separately on separate machines, hence significantly speeding up the function.

This parallel processing capability has been built in to the *puma* package, making use of functionality from the R package *snow*. The *snow* package itself has been designed to run on the following three underlying technologies: MPI, PVM and socket connections. However, the *puma* package has only been tested using MPI, and this is the recommended technology to use. Parallel processing in *puma* has also only been tested to date on a Sun GridEngine cluster. The steps to set up *puma* on such an architecture are as follows:

1. Download the latest version of lam-mpi from http://www.lam-mpi.org/
2. Install lam-mpi following the instructions available at http://www.lam-mpi.org/
3. Create a text file called `hostfile`, the first line of which has the IP address of the master node of your cluster, and subsequent line of which have the IP addresses of each node you wish to use for processing
4. From the command line type the command `lamboot hostfile`. If this is successful you should see a message saying

   ```
   LAM 7.1.2/MPI 2 C++/ROMIO - Indiana University
   ```

   (or similar)
5. Install R and the *puma* package on each node of the cluster (note this will often simply involve running R CMD INSTALL on the master node)
6. Install the R packages *snow* and *Rmpi* on each node of the cluster

The function `pumaComb` should automatically run in parallel if the `lamboot` command was successful, and *puma* , *snow* and *Rmpi* are all installed on each node of the cluster. By default the function will use all available nodes.

If you want to override the default parallel behaviour of `pumaComb`, you can set up your own cluster which will subsequently be used by the function. This cluster has to be named `cl`. To run a cluster with, e.g. four nodes, run the following code:

```
> library(Rmpi)
> library(snow)
> cl <- makeCluster(4)
> clusterEvalQ(cl, library(puma))
```

Note that it is important to use the variable name `cl` to hold the `makeCluster` object as *puma* checks for a variable of this name. The argument to `makeCluster` (here 4) should be the number of nodes on which you want the processing to run (usually the same as the number of nodes included in the `hostfile` file, though can also be less).

Running `pumaComb` in parallel should generally give a speed up almost linear in terms of the number of nodes (e.g. with four nodes you should expect the function to complete in about a quarter of the time as if using just one node).

7 Session info

This vignette was created using the following:

```
> sessionInfo()

R version 2.9.0 (2009-04-17)
i386-apple-darwin8.11.1

locale:
C

attached base packages:
[1] stats     graphics  grDevices datasets  utils
[6] methods   base

other attached packages:
[1] limma_2.18.0      hgu95av2cdf_2.4.0 pumadata_1.0.1
[4] puma_1.10.0       affy_1.22.0       Biobase_2.4.1

loaded via a namespace (and not attached):
[1] affyio_1.12.0        preprocessCore_1.6.0
[3] tools_2.9.0
```

A Automatic creation of design and contrast matrices

The *puma* package has been designed to be as easy to use as possible, while not compromising on power and flexibility. One of the most difficult tasks for many users, particularly those new to microarray analysis, or statistical analysis in general, is setting up design and contrast matrices. The *puma* package will automatically create such matrices, and we believe the way this is done will suffice for most users' needs.

It is important to recognise that the automatic creation of design and contrast matrices will only happen if appropriate information about the levels of each factor is available for each array in the experimental design. This data should be held in an `AnnotatedDataFrame` class. The easiest way of doing this is to ensure that the `affybatch` object holding the raw CEL file data has an appropriate phenoData slot. This information will then be passed through to any `ExpressionSet` object created, for example through the use of `mmgmos`. The `phenoData` slot of an `ExpressionSet` object can also be manipulated directly if necessary.

Design and contrast matrices are dependent on the experimental design. The simplest experimental designs have just one factor, and hence the `phenoData` slot will have a matrix with just one column. In this case, each unique value in that column will be treated as a distinct level of the factor, and hence `pumaComb` will group arrays according to these levels. If there are just two levels of the factor, e.g. A and B, the contrast matrix will also be very simple, with the only contrast of interest being A vs B. For factors with more than two levels, a contrast matrix will be created which reflects all possible combinations of levels. For example, if we have three levels A, B and C, the contrasts of interest will be A vs B, A vs C and B vs C. In addition, from *puma* version 1.2.1, the following additional contrasts will be created: A vs other (i.e. A vs B & C), B vs other and C vs other.

If we now consider the case of two or more factors, things become more complicated. There are now two cases to be considered: factorial experiments, and non-factorial experiments. A factorial experiment is one where all the combinations of the levels of each factor are tested by at least one array (though ideally we would have a number of biological replicates for each combination of factor levels). The `estrogen` case study (Section 4) is an example of a factorial experiment. A non-factorial experiment is one where at least one combination of levels is not tested. If we treat the example used in the `puma-package` help page as a two-factor experiment (with factors "level" and "batch"), we can see that this is not a factorial experiment as we have no array to test the conditions "level=ten" and "batch=B". We will treat the factorial and non-factorial cases separately in the following sections.

A.1 Factorial experiments

For factorial experiments, the design matrix will use all columns from the `phenoData` slot. This will mean that `combineRepliactes` will group arrays according to a combination of the levels of all the factors.

A.2 Non-factorial designs

For non-factorial designed experiments, we will simply ignore columns (right to left) from the `phenoData` slot until we have a factorial design or a single factor. We can see this in the example used in the `puma-package` help page. Here we have ignored the "batch" factor, and modelled the experiment as a single-factor experiment (with that single factor being "level").

A.3 Further help

There are examples of the automated creation of design and contrast matrices for increasingly complex experimental designs in the help pages for `createDesignMatrix` and `createContrastMatrix`.

References

[1] Milo,M., Niranjan,M., Holley,M.C., Rattray,M. and Lawrence,N.D. (2004) A probabilistic approach for summarising oligonucleotide gene expression data. Technical report available upon request.

[2] Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2005) A tractable probabilistic model for Affymetrix probe-level analysis across multiple chips. Bioinformatics, 21:3637-3644.

[3] Sanguinetti,G., Milo,M., Rattray,M. and Lawrence, N.D. (2005) Accounting for probe-level noise in principal component analysis of microarray data. Bioinformatics, 21:3748-3754.

[4] Rattray,M., Liu,X., Sanguinetti,G., Milo,M. and Lawrence,N.D. (2006) Propagating uncertainty in Microarray data analysis. Briefings in Bioinformatics, 7:37-47.

[5] Liu,X., Milo,M., Lawrence,N.D. and Rattray,M. (2006) Probe-level measurement error improves accuracy in detecting differential gene expression. Bioinformatics, 22:2107-2113.

[6] Liu,X., Lin,K.K., Andersen,B., and Rattray,M. (2006) Including probe-level uncertainty in model-based gene expression clustering. BMC Bioinformatics, 8(98).

[7] Peter Spellucci. DONLP2 code and accompanying documentation. Electronically available via http://plato.la.asu.edu/donlp2.html.