

# affyPLM: Methods for fitting probe level models to Affy data

Ben Bolstad  
bolstad@stat.berkeley.edu

November 25, 2003

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Fitting Probe level models</b>	<b>2</b>
2.1	What is a probelevel model and what is a PLMset? . . . . .	2
2.2	Using the software to fit probelevel models . . . . .	2
2.2.1	Some examples . . . . .	2
2.2.2	Some specific notes about specifying models in <code>fitPLM</code> . . . . .	5
2.2.3	Pre-processing . . . . .	5
<b>3</b>	<b>Quality diagnostics</b>	<b>5</b>
<b>4</b>	<b>How long will it take to run the model fitting procedures?</b>	<b>7</b>
4.1	Why is it so much slower than the <code>rma</code> function? . . . . .	8

## 1 Introduction

This document describes an R Ihaka and Gentleman (1996) package that has been written to fit robust probe level models to affy data.

After starting R, the package should be loaded using this will load affyPLM as well as the affy package and its dependencies.

## 2 Fitting Probe level models

### 2.1 What is a probelevel model and what is a PLMset?

A probelevel model is a linear model fit to probe data. In particular it is where we fit a model with probe level and chip level parameters for each probeset. It is easy to arrange the data for a probeset so that the rows are probes and the columns are chips. In this case our probe level parameters are usually a factor variable for each probe. The chip level parameters could be, a factor with a level for each array, factor variables grouping the chips into treatment groups or perhaps some sort of covariate (pH, temperature).

A **PLMset** is an R object that holds the results of a fitted probelevel model. Some of the items that are kept are parameter estimates, their standard errors and robust fit weights.

### 2.2 Using the software to fit probelevel models

#### 2.2.1 Some examples

The main function for fitting Probe level models is the function `fitPLM`. The most basic way to call the function is to call passing an **AffyBatch** objects, this will fit a linear model with an effect estimated for each chip.

```
> data(Dilution)
> Pset <- fitPLM(Dilution)
```

Background correcting

Normalizing

Fitting models

once you have a fitted model (this object is referred to as a **PLMset** object) you may view the chip level parameter estimates and the corresponding standard errors using accessor functions `coefs` and `se` respectively.

```
> coefs(Pset)[1:10, ]
```

	20A	20B	10A	10B
1000_at	7.712273	7.589297	7.604220	7.553855
1001_at	5.403363	5.246484	5.273807	5.361618
1002_f_at	6.001355	5.842824	5.875274	5.901889
1003_s_at	6.682987	6.369074	6.497950	6.402646
1004_at	6.265232	5.923743	6.150849	6.088767
1005_at	7.967263	7.947008	7.794284	7.874413
1006_at	5.039561	4.905057	5.009430	5.000676
1007_s_at	8.486376	8.370319	8.384898	8.341944
1008_f_at	7.602705	7.371384	7.741406	7.583359
1009_at	9.341384	9.374631	9.403627	9.334484

```
> se(Pset)[1:10, ]
```

	20A	20B	10A	10B
1000_at	0.03799718	0.03674898	0.03948395	0.03526662
1001_at	0.05850690	0.05890020	0.05918202	0.05606334
1002_f_at	0.06684831	0.06552934	0.06374240	0.06469234
1003_s_at	0.06114872	0.05960096	0.06161738	0.06088379
1004_at	0.05008092	0.04901174	0.04981303	0.04707207
1005_at	0.03306321	0.03260544	0.03615805	0.03332997
1006_at	0.06249281	0.05936004	0.05909235	0.05928336
1007_s_at	0.03226892	0.03122670	0.03342641	0.03164790
1008_f_at	0.03742417	0.03946399	0.03788032	0.03814021
1009_at	0.02842270	0.02861522	0.03055057	0.02848978

The `fitPLM` function also allows you to specify more complicated models. For instance we may wish to fit a robust linear model with an effect for liver dilution level and scanner (these are already defined in the `phenoData` slot of the `AffyBatch`, you can also give names of objects in the current environment)

```
> Pset <- fitPLM(Dilution, model = PM ~ -1 + probes + liver + scanner,
+               normalize = FALSE, background = FALSE)
```

Fitting models

```
> coefs(Pset)[1:2, ]
```

	liver_10	liver_20	scanner_2
1000_at	8.432918	8.924804	-0.6891075
1001_at	7.228003	7.597236	-0.6156121

By default all variables are treated as factors. The “-1 + probes” is handled specially by the c code (currently the probe effects are constrained by a sum to zero constraint). The first factor specified is unconstrained (in the above case that would be liver), all of the following factors are constrained with the endpoint constraint (ie in our case scanner).

It is also possible to fit models where some of the variables are treated as covariates rather than as factors.

```
> logliver <- log2(c(20, 20, 10, 10))
> Pset <- fitPLM(Dilution, model = PM ~ -1 + probes + logliver +
+               scanner, normalize = FALSE, background = FALSE, variable.type = c(logliver = "c"))
```

No default type given. Assuming default variable type is factor

Fitting models

```
> coefs(Pset)[1:2, ]
```

```

           logliver scanner_1 scanner_2
1000_at 0.4918856  6.798910  6.109802
1001_at 0.3692328  6.001438  5.385826

```

We can also fit models with intercept parameters

```

> data(affybatch.example)
> Pset <- fitPLM(affybatch.example, model = PM ~ probes + samples,
+   normalize = FALSE, background = FALSE)

```

Fitting models

```
> coefs.const(Pset)[1:2]
```

```

A28102_at AB000114_at
 7.039324   7.037748

```

```
> coefs(Pset)[1:2, ]
```

```

           20B          10A
A28102_at -0.03310827 -0.2537973
AB000114_at 0.09955687 -0.1887458

```

and with different constraints on the parameters.

```

> data(affybatch.example)
> Pset <- fitPLM(affybatch.example, model = PM ~ probes + samples,
+   normalize = FALSE, background = FALSE, constraint.type = c(samples = "contr.sum")

```

No default type given. Assuming default constraint type is contr.treatment

Fitting models

```
> coefs.const(Pset)[1:2]
```

```

A28102_at AB000114_at
 6.943689   7.008018

```

```
> coefs(Pset)[1:2, ]
```

```

           20A          20B
A28102_at 0.09563518 0.0625269
AB000114_at 0.02972966 0.1292865

```

### 2.2.2 Some specific notes about specifying models in fitPLM

If you do not specify a model when using fitPLM then the default model is used. The default model is `PM -1 + probes + samples`. The words `probes` and `samples`, for probe and array effects, are reserved words in the specification of a model. The default model will compute a parameter for each chip.

There are some specific rules that you must follow when specifying a model the first is that the response should always be `PM`, next specify the intercept and probe effect terms ie `-1 + probes`, then you may specify your chip level parameters. To put chip level parameter all you need to do is put the name in your model as we have done in the earlier examples with `liver`, `scanner` and `logliver`. These variables should either be defined in the `phenoData` slot of the `AffyBatch` holding your data or as appropriate vectors in your current R session, like the example with `logliver` above.

By default all parameters are treated as factor variables unless specified otherwise. The first chip level factor is always left unconstrained (unless an intercept term has been fitted), all the following chip level factors are by default constrained with the endpoint constraint, unless you specify otherwise. Note that the probe parameters are by default constrained using the sum to zero constraint, it is recommended you use this constraint, but it is possible to use an endpoint constraint by using `constraint.type`

To specify that a chip level parameter should be treated as a covariate rather than a factor, you use the `variable.type` parameter of `fitPLM`. The example using `logliver` demonstrates this.

### 2.2.3 Pre-processing

By default the `fitPLM` function will preprocess the data using the RMA Irizarry et al. (2003b), Irizarry et al. (2003a) preprocessing steps. In particular it uses the same background and normalization Bolstad et al. (2003) as the `rma` function of `affy`. It is possible to turn off either of these steps by specifying that `background` and/or `normalize` are `FALSE`.

## 3 Quality diagnostics

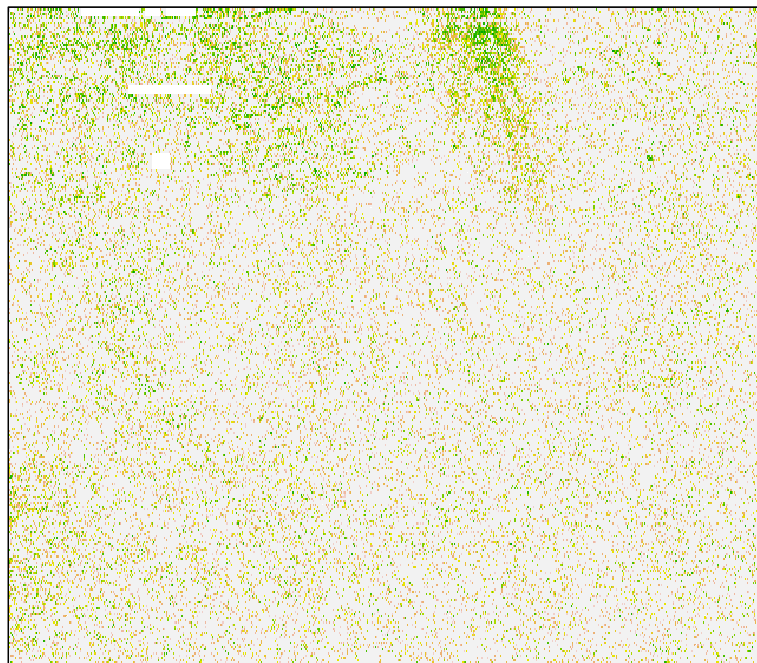
A pseudo chip image plot of the weights from the robust linear model fit is a useful quality diagnostic.

```
> Pset <- fitPLM(Dilution)
```

```
Background correcting  
Normalizing  
Fitting models
```

```
> image(Pset, which = 2)
```

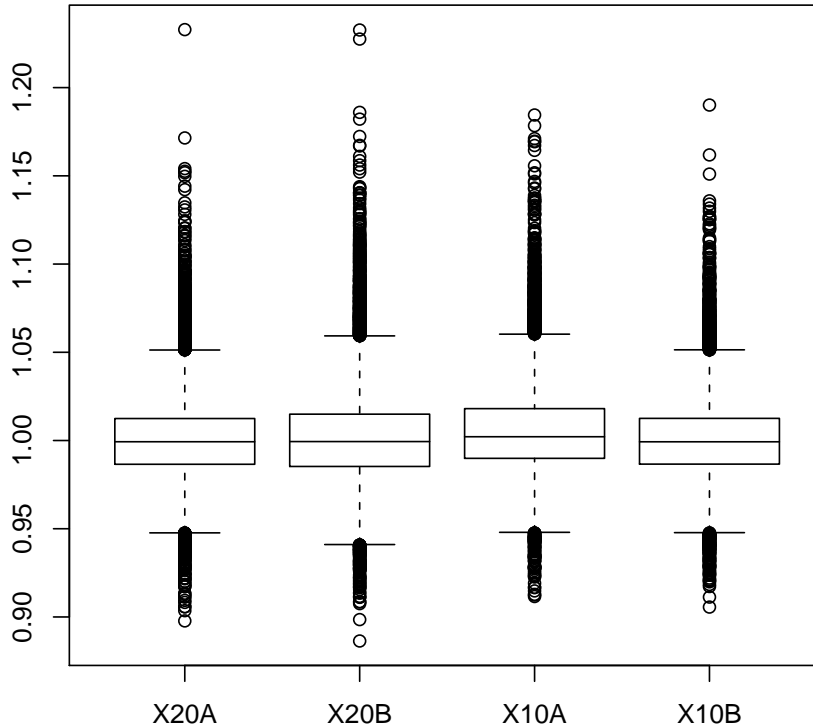
## 20B



Areas of low weight are greener, high weights (ie weight of 1) are light grey). The image plot will allow you to detect artifacts on your chip.

Another diagnostic, particularly useful when you fit a model with an effect for each array (the default model) is to boxplot the model standard errors (standardized across probesets so each has median 1). This is done by using the boxplot function on a **PLMset** object.

```
> boxplot(Pset)
```



Another possible diagnostic is to compute M's for each chip relative to a median (pseudo)chip. This is handled by the command `eval=FALSE` `Mbox(Pset)` which will produce boxplots of the M values by array. A discordant chip may show on such a plot.

## 4 How long will it take to run the model fitting procedures?

It may take considerable time to run the `fitPLM` function. The length of time it is going to take to run the model fitting procedure will depend on a number of factors including:

1. CPU speed
2. Memory size of the machine (RAM and VM)
3. Array type
4. Number of arrays

Component	Specs
OS	Red Hat Linux 8.0
kernel	2.4.20-ac2 with preemptive patch applied
processor	AMD Athlon Thunderbird 1.2 Ghz
RAM	1 GB
Swap	6 GB
R	R-1.7.0 (Development)
affy	1.1.8
affyPLM	0.4-14

Figure 1: Benchmarking Machine Specifications

## 5. Number of parameters in model

It is recommended that you run the `fitPLM` function only on machines with large amounts of RAM. If you have a large number of arrays the number of parameters in your model will have the greatest effect on runtime.

The `fitPLM` function has been tested using the `system.time` function. The specifications of the test machine are given in figure 1. The results are given in 2.

## 4.1 Why is it so much slower than the `rma` function?

The robust linear model fitting procedure uses IRLS (iteratively re-weighted least squares) which is going to be inherently slower than the median polish algorithm. In addition the `fitPLM` procedure produces standard error and weight estimates where as the `rma` function focuses only on producing expression estimates. If your goal is to compute expression estimates one for each probeset, each array it is probably better to use `rma` function. When you wish to fit a more general model `fitPLM` is the more appropriate choice.

## References

- B.M. Bolstad, R.A. Irizarry, M. Åstrand, and T.P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, 2003.
- Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- R. A. Irizarry, B. M. Bolstad, F. Collin, L. M. Cope, B. Hobbs, and T. P. Speed. Summaries of affymetrix genechip probe level data. *Nucleic Acids Research*, 2003a. To appear.

Number of Arrays	Number of Chip level parameters	Run time (seconds)
5	5	85
10	10	176
10	5	134
10	2	117
20	20	483
20	10	278
20	5	223
20	2	200
30	30	1099
30	10	407
30	5	322
30	2	272
40	40	2079
40	10	553
40	5	422
40	2	399
50	50	6112 (about 102 Minutes)
50	10	745
50	5	567
50	2	483

Figure 2: Runtimes in seconds

Rafael A. Irizarry, Bridget Hobbs, Francois Collin, Yasmin D. Beazer-Barclay, Kristen J. Antonellis, Uwe Scherf, and Terence P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 2003b. To appear.