affyPLM: Fitting Probe Level Models

Ben Bolstad
bmb@bmbolstad.com
http://bmbolstad.com

November 11, 2026

Contents

1 Introduction **2**

2 Fitting Probe Level Models **2**
2.1 What is a Probe Level Model and What is a `PLMset`? 2
2.2 Getting Started with the Default Model 2
2.3 Getting Full Control over `fitPLM` 4
2.3.1 Pre-processing 4
2.3.2 Controlling what is Returned in the *PLMset* 4
2.3.3 Controlling how the model is fit 5
2.4 Specifying models in `fitPLM` 6
2.4.1 Notation 7
2.4.2 RMA style PLM 8
2.4.3 PLM with chip-level factor and covariate variables 8
2.4.4 Probe intensity covariate PLM 9
2.4.5 PLM with both probe types as response variables 11
2.4.6 PLM with probe-effects estimated within levels of a chip-level factor variable 12
2.4.7 PLM with probe-effect estimated within probe.type 13
2.4.8 PLM without chip level effects 13
2.4.9 PLM with only probe-effects 14
2.4.10 Constraints 14

3 How long will it take to run the model fitting procedures? **15**

4 Dealing with the `PLMset` object **16**

A M-estimation: How `fitPLM` fits models **16**

1 Introduction

This majority of this document describes the `fitPLM` function. Other vignettes for affyPLM describe quality assessment tools and the `threestep` function for computing expression measures. After starting R, the package should be loaded using:

```
> library(affyPLM)
> require(affydata)
> data(Dilution)   # an example dataset provided by the affydata package
> ##FIXME: drop this after Dilution is updated
> Dilution = updateObject(Dilution)
> options(width=36)
```

this will load affyPLM as well as the affy package and its dependencies. The *Dilution* dataset will serve as an example dataset for this document.

2 Fitting Probe Level Models

2.1 What is a Probe Level Model and What is a `PLMset`?

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

A *PLMset* is an R object that holds the results of a fitted probe level model. Among the items stored are parameter estimates and corresponding standard errors, weights and residuals.

2.2 Getting Started with the Default Model

The main function for fitting PLM is the function `fitPLM`. The easiest way to call the function is to call it by passing an *AffyBatch* object without any other arguments, this will fit a linear model with an effect estimated for each chip and an effect for each probe. This can be accomplished using:

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

Once you have a fitted model stored in a *PLMset* object the chip level parameter estimates and the corresponding standard errors can be examined using the accessor functions `coefs` and `se` respectively. For example, to examine the parameter estimates for the first 5 probesets and their corresponding standard error estimates use:

```
> coefs(Pset)[1:5,]
```

```
              20A      20B
1000_at   7.712294 7.589361
1001_at   5.403364 5.246518
1002_f_at 6.001354 5.842818
1003_s_at 6.682994 6.369082
1004_at   6.265231 5.923743
              10A      10B
1000_at   7.604202 7.553845
1001_at   5.273806 5.361613
1002_f_at 5.875315 5.901890
1003_s_at 6.497949 6.402645
1004_at   6.150849 6.088775
```

```
> se(Pset)[1:5,]
```

```
                20A        20B
1000_at   0.03798884 0.03674159
1001_at   0.05850398 0.05889818
1002_f_at 0.06684381 0.06552410
1003_s_at 0.06114870 0.05960099
1004_at   0.05008044 0.04901129
                10A        10B
1000_at   0.03947771 0.03526104
1001_at   0.05917896 0.05606087
1002_f_at 0.06373685 0.06468718
1003_s_at 0.06161744 0.06088377
1004_at   0.04981253 0.04707150
```

Note that the default model is the RMA expression measure model. Specifically, the default model is

$$\log_2 \mathrm{PM}_{kij} = \beta_{kj} + \alpha_{ki} + \epsilon_{kij}$$

where β_{kj} is the $\log_2$ gene expression value on array j for probeset k and α_{ki} are probe effects. Note that to make the model identifiable the constraint $\sum_{i=1}^{I} \alpha_{ki} = 0$ is used. Thus, for this default model, the parameter estimates given are gene expression values.

2.3 Getting Full Control over `fitPLM`

While the default model is very useful and simple to use, the `fitPLM` function also provides the user with a great deal of control. Specifically, the user has the ability to change the preprocessing, how the model is fitted and what output is returned.

2.3.1 Pre-processing

By default, the `fitPLM` function will preprocess the data using the RMA preprocessing steps. In particular, it uses the same background and normalization as the `rma` function of the affy package. It is possible to turn off either of these preprocessing steps by specifying that `background` and/or `normalize` are `FALSE` in the call to `fitPLM`. The arguments `background.method` and `normalize.method` can be used to control which pre-processing methods are used. The same preprocessing methods, as described in the `threestep` vignette, may be used with the `fitPLM` function.

2.3.2 Controlling what is Returned in the *PLMset*

The *PLMset* that the `fitPLM` function returns contains a number of different quantities, some of which are always returned such as parameter estimates and standard error estimates and others which are more optional. The user can control whether weights, residuals, variance-covariance matrices and residual standard deviations are returned. By default, all of these items are returned, but in certain situations a user might find it useful to exclude certain items to save memory. Control is via the `output.param` argument which should be provided as a list. The default settings can be seen by typing

```
> verify.output.param()

$weights
[1] TRUE

$residuals
[1] TRUE

$varcov
[1] "none"

$resid.SE
[1] TRUE
```

Control of whether weights, residuals and residual standard deviations are returned is via logical variables. There are three options `varcov = "none"`, `varcov = "chiplevel"` and `varcov = "all"` for variance covariance matrices. These correspond to, not returning any variance estimates, only the portion of the variance covariance matrix related to

the chiplevel parameters or the entire variance covariance matrix respectively. When each probeset has a large number of probes (or there are large numbers of parameters in the model) the last option will return many large variance covariance matrices. The following code returns a `PLMset` with no weights or residuals stored:

```
> Pset <- fitPLM(Dilution,output.param=list(residuals=FALSE,weights=FALSE))
```

2.3.3 Controlling how the model is fit

`fitPLM` implements iteratively re-weighted least squares M-estimation regression. Control over how `fitPLM` carries out the model fitting procedure is given by the `model.param` argument. This value of this parameter should be a `list` of settings. In particular, these settings are the following:

- `trans.fn` which controls how the response variable is transformed. This value should be a string. By default `trans.fn="log2"`, but other possible options include: `"loge"` or `"ln"` to use the natural logarithm, `"log10"` for logarithm base 10, `"sqrt"` for square root and `"cuberoot"` to use a cubic root transformation.

- `se.type` which controls how the variance-covariance matrix is estimated in the M-estimation procedure. Possible values are `1`, `2`, `3` or `4`. See the Appendix for more details.

- `psi.type` is a string which selects how the weights are computed in the robust regression. By default `psi.type="Huber"`. Other possible options include `"fair"`, `"Cauchy"`, `"Geman-McClure"`, `"Welsch"`, `"Tukey"`, and `"Andrews"`. More details can be found in the Appendix.

- `psi.k` is a numerical tuning constant used by `psi.type`. The default values are dependent on the option chosen by `psi.type`. More details can be found in the Appendix.

- `max.its` controls the maximum number of iterations of IRLS that will be used in the model fitting procedure. By default `max.its=20`. Note, that this many iterations may not be needed if convergence occurs.

- `init.method` controls how the initial parameter estimates are derived. By default `init.method="ls"` ordinary least squares is used although `"Huber"` is also a possibility.

- `weights.chip` are input weights for each chip in the dataset. This parameter should be a vector of length number of arrays in the dataset. By default, every chip is given equal weight.

- `weights.probe` are input weights for each probe in the dataset. This parameter should be a vector of length number of probes in dataset (this length depends on the response variable in the model). By default, every probe has equal weight.

As an example, we use `model.param` to control the fitting procedure so that it is fit as a standard linear regression model (ie without robustness). This is accomplished by:

```
> Pset <- fitPLM(Dilution,model.param=list(max.its=0))
```

2.4 Specifying models in `fitPLM`

Although the default model is very useful, it is by no means the only model that can be fitted using `fitPLM`. In this section we describe many, but certainly not all the different types of models which may be fitted. In the example code presented here we will use the `subset` argument to restrict model fitting to the first 100 probesets for speed. In any real analysis model fitting would be carried out to all probesets on a particular array type.

2.4.1 Notation

i	Index for probes $i = 1, \ldots, I_k$
j	Index for arrays $j = 1, \ldots, J$
k	Index for probeset $k = 1, \ldots, K$
l	Index for probe type $l = 1, 2$ where 1 is PM and 2 is MM.
m	Index for level of primary treatment factor variable $m = 1, \ldots, M$
α_{ki}	probe effect parameter for probe i Included in the model by using `probes`
α_{kim}	probe effect parameter for probe i estimated only for arrays where primary treatment factor variable is level m Included in the model by using `treatment:probes`
α_{kil}	probe effect parameter for probe i estimated only for probes of type l Included in the model by using `probe.type:probes`
α_{kilm}	probe effect parameter for probe i estimated only for probes of type l where primary treatment factor variable is level m Included in the model by using `treatment:probe.type:probes`
β_{kj}	array (chip) effect. Included in the model by using `samples`
ϕ_{kl}	probe-type effect Included in the model by using `probe.types`
ϕ_{klm}	probe-type effect for probe type l estimated only for arrays where primary treatment factor variable is level m Included in the model by using `treatment:probe.types`
ϕ_{klj}	probe-type effect for probe type l estimated only for array j Included in the model by using `samples:probe.types`
θ	a vector of chip-level parameters
μ_k	an intercept parameter
γ_k	a slope parameter
y_{kijl}	a processed probe-intensity. Typically on $\log_2$ scale.
ϵ_{kijl}	an error term
x_j	measurements of chip-level factor and covariate variables for chip j In the model descriptions below we will use `treatment, trt.cov` for these terms. In practice these would be replaced with names of variables in the current R environment or the *phenoData* of the supplied *AffyBatch*.

Since we are focusing on models that are fitted in a probeset by probeset manner for brevity the subscript k will be omitted from further discussion. Note the terms `probes`, `samples` and `probe.types` are considered reserved words when specifying a model to `fitPLM`.

2.4.2 RMA style PLM

These are variations of the RMA model each consisting of models with chip and probe-effects . The first, `PM ~ -1 + samples + probes`, is the default model used when no model is specified in the `fitPLM` call.

Model	`fitPLM` syntax
$y_{ij1} = \beta_j + \alpha_i + \epsilon_{ij}$	`PM ~ -1 + samples + probes`
$y_{ij1} = \mu + \beta_j + \alpha_i + \epsilon_{ij}$	`PM ~ samples + probes`
$y_{ij1} = \beta_j + \epsilon_{ij}$	`PM ~ -1 + samples`
$y_{ij1} = \mu + \beta_j + \epsilon_{ij}$	`PM ~ samples`
$y_{ij2} = \beta_j + \alpha_i + \epsilon_{ij}$	`MM ~ -1 + samples + probes`
$y_{ij2} = \mu + \beta_j + \alpha_i + \epsilon_{ij}$	`MM ~ samples + probes`
$y_{ij2} = \beta_j + \epsilon_{ij}$	`MM ~ -1 + samples`
$y_{ij2} = \mu + \beta_j + \epsilon_{ij}$	`MM ~ samples`

2.4.3 PLM with chip-level factor and covariate variables

These models use treatment variables as an alternative to sample effects for the chip level factors.

Model	`fitPLM` syntax
$y_{ij1} = x_j^T\theta + \alpha_i + \epsilon_{ij}$	`PM ~ -1 + treatment + trt.cov + probes`
$y_{ij1} = x_j^T\theta + \epsilon_{ij}$	`PM ~ -1 + treatment + trt.cov`
$y_{ij2} = x_j^T\theta + \alpha_i + \epsilon_{ij}$	`MM ~ -1 + treatment + trt.cov + probes`
$y_{ij2} = x_j^T\theta + \epsilon_{ij}$	`MM ~ -1 + treatment + trt.cov`

For example to fit, a model with effects for both liver tissue concentration and scanner along with probe effects with MM as the response variable to the first 100 probesets of the Dilution dataset the following code would be used:

```
> Pset <- fitPLM(Dilution,  MM ~ -1 + liver + scanner + probes,subset = geneNames(Dilut
```

Examining the fitted chip-level parameter estimates for the first probeset via:

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

```
   liver_10    liver_20   scanner_2
 6.26484583  6.41754460 -0.07134098
```

shows that the treatment effect for scanner was constrained to make the model identifiable. `fitPLM` always leaves the first factor variable unconstrained if there is no intercept term. All other chip level factor variables are constrained. The parameter estimates for the probe effects can be examined as follows:

```
> coefs.probe(Pset)[1]
```

```
$`1000_at`
            Overall
probe_1   3.0216710
probe_2   0.2126189
probe_3  -2.0222307
probe_4  -0.3203435
probe_5  -2.0974766
probe_6   3.3955995
probe_7  -2.7692459
probe_8  -3.5822543
probe_9  -1.2662559
probe_10 -1.3146932
probe_11 -0.8544930
probe_12 -2.0119851
probe_13  3.0164205
probe_14  4.5947743
probe_15  1.8807170
```

To make a treat a variable as a covariate rather than a factor variable the `variable.type` argument may be used. For example, to fit a model with the logarithm of liver concentration treated as a covariate we could do the following:

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

Warning: No default variable type so assuming 'factor'

> coefs(Pset)[1,]

  logliver  scanner_1  scanner_2
0.04494461 7.47722447 7.44720674
```

2.4.4 Probe intensity covariate PLM

This class of models allows the inclusion of PM or MM probe intensities as covariate variables in the model. Note that the fitting methods currently used by `fitPLM` are robust, but not resistant (ie outliers in the response variable are dealt with, but outliers in explanatory variables are not).

Model	`fitPLM` syntax
$y_{ij1} = \gamma y_{ij2} + \beta_j + \alpha_i + \epsilon_{ij}$	`PM ~ -1 + MM + samples + probes`
$y_{ij1} = \gamma y_{ij2} + \mu + \beta_j + \alpha_i + \epsilon_{ij}$	`PM ~ MM + samples + probes`
$y_{ij1} = \gamma y_{ij2} + \beta_j + \epsilon_{ij}$	`PM ~ -1 + MM +samples`
$y_{ij1} = \gamma y_{ij2} + \mu + \beta_j + \epsilon_{ij}$	`PM ~ MM + samples`
$y_{ij2} = \gamma y_{ij1} + \beta_j + \alpha_i + \epsilon_{ij}$	`MM ~ -1 + PM + samples + probes`
$y_{ij2} = \gamma y_{ij1} + \mu + \beta_j + \alpha_i + \epsilon_{ij}$	`MM ~ PM + samples + probes`
$y_{ij2} = \gamma y_{ij1} + \beta_j + \epsilon_{ij}$	`MM ~ -1 + PM +samples`
$y_{ij2} = \gamma y_{ij1} + \mu + \beta_j + \epsilon_{ij}$	`MM ~ PM + samples`
$y_{ij1} = x_j^T\theta + \gamma y_{ij2} + \alpha_i + \epsilon_{ij}$	`PM ~ MM + treatment + trt.cov + probes`
$y_{ij1} = x_j^T\theta + \gamma y_{ij2} + \epsilon_{ij}$	`PM ~ MM + treatment + trt.cov`
$y_{ij2} = x_j^T\theta + \gamma y_{ij1} + \alpha_i + \epsilon_{ij}$	`MM ~ PM + treatment + trt.cov + probes`
$y_{ij2} = x_j^T\theta + \gamma y_{ij1} + \epsilon_{ij}$	`MM ~ PM + treatment + trt.cov`

To fit a model with an intercept term, MM covariate variable, sample and probe effects use the following code:

```
> Pset <- fitPLM(Dilution,  PM ~ MM + samples + probes,subset = geneNames(Dilution)[1:1
```

We can examine the various parameter estimates for the model fit to the first probeset using:

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

        20B         10A         10B
-0.01390033 -0.02593732 -0.05154342

> coefs.const(Pset)[1,]

Intercept        MM
6.9439916 0.1131364

> coefs.probe(Pset)[1]

$`1000_at`
             Overall
probe_1  -0.09248614
probe_2   0.05086713
probe_3  -2.05564830
probe_4   1.55090007
probe_5  -1.63841905
probe_6   1.05603587
probe_7  -3.40366080
probe_8  -1.07325873
probe_9   0.18670389
probe_10 -0.52200547
```

```
probe_11 -0.85784749
probe_12  0.73605865
probe_13  2.18822218
probe_14  2.96008104
probe_15  1.31332885
```

As can be seen by this example code intercept and covariate parameters are accessed using `coefs.const`.

2.4.5 PLM with both probe types as response variables

It is possible to fit a model that uses both PM and MM intensities as the response variable. This is done by specifying `PMMM` as the response term in the model. When both PM and MM intensities are used as the response, there is a special reserved term *probe.type* which may (optionally) be used as part of the model specification. This term designates that a probe type effect (ie whether PM or MM) should be included in the model.

Model	`fitPLM` syntax
$y_{ijl} = \beta_j + \phi_j + \alpha_i + \epsilon_{ijl}$	`PMMM ~ -1 + samples + probe.type + probes`
$y_{ijl} = \mu + \beta_j + \phi_j + \alpha_i + \epsilon_{ijl}$	`PMMM ~ samples + probe.type + probes`
$y_{ijl} = \beta_j + \phi_j + \epsilon_{ijl}$	`PMMM ~ -1 + samples+ probe.type`
$y_{ijl} = \mu + \beta_j + \phi_j + \epsilon_{ijl}$	`PMMM ~ samples+ probe.type`
$y_{ijl} = x_j^T\theta + \phi_j + \alpha_i + \epsilon_{ijl}$	`PMMM ~ treatment + trt.cov + probe.type + probes`
$y_{ijl} = x_j^T\theta + \phi_j + \epsilon_{ijl}$	`PMMM ~ treatment + trt.cov + probe.types`
$y_{ijl} = x_j^T\theta + \phi_j + \alpha_i + \epsilon_{ijl}$	`PMMM ~ treatment + trt.cov + probe.type + probes`
$y_{ijl} = x_j^T\theta + \phi_j + \epsilon_{ijl}$	`PMMM ~ treatment + trt.cov + probe.type`

For example to fit such a model with factor variables for liver RNA concentration, probe type and probe effects use:

```
> Pset <- fitPLM(Dilution,  PMMM ~ liver + probe.type + probes,subset = geneNames(Dilut
```

Examining the parameter estimates:

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

[1] 0.06796632

> coefs.const(Pset)[1,]

    Intercept probe.type_MM
     7.597301     -1.317395

> coefs.probe(Pset)[1]
```

```
$`1000_at`
            Overall
probe_1   1.6712982
probe_2   0.1412111
probe_3  -2.1775557
probe_4   0.6740782
probe_5  -2.0523876
probe_6   2.3925977
probe_7  -3.2392634
probe_8  -2.6055623
probe_9  -0.5632586
probe_10 -0.9691117
probe_11 -0.9129540
probe_12 -0.7431879
probe_13  2.7703955
probe_14  4.0347865
probe_15  1.7050728
```

shows that probe type estimates are also accessed by using `coefs.const`.

2.4.6 PLM with probe-effects estimated within levels of a chip-level factor variable

It is also possible to estimate separate probe-effects for each level of a chip-level factor variable.

Model	`fitPLM` syntax
$y_{ij1} = x_j^T\theta + \alpha_{im} + \epsilon_{ij1}$	`PM ~ treatment:probes + treatment + trt.cov`
$y_{ij1} = y_{ij2}\gamma + x_j^T\theta + \alpha_{im} + \epsilon_{ij1}$	`PM ~ MM + treatment + treatment:probes + trt.cov`
$y_{ijl} = x_j^T\theta + \phi_j + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ treatment + trt.cov + treatment:probes`

Fitting such a model with probe effects estimated within the levels of the liver variable is done with:

```
> Pset <- fitPLM(Dilution,  PM ~ -1 + liver + liver:probes,subset = geneNames(Dilution)
```

Examining the estimated probe-effects for the first probeset can be done via:

```
> coefs.probe(Pset)[1]
```

```
$`1000_at`
           liver_10:   liver_20:
probe_1   0.45658552  0.18539206
probe_2   0.09857862  0.03521846
probe_3  -2.32629757 -2.26742904
```

```
probe_4   1.56680474  1.44646328
probe_5  -1.79470843 -2.21774179
probe_6   1.41541418  1.46642950
probe_7  -3.73516433 -3.50762769
probe_8  -1.66741086 -1.29889473
probe_9  -0.02414340  0.07770227
probe_10 -0.77865227 -0.47421776
probe_11 -0.95612811 -0.97135598
probe_12  0.37531107  0.60090422
probe_13  2.46981266  2.57311873
probe_14  3.54667188  3.39711587
probe_15  1.63293065  1.42011704
```

2.4.7 PLM with probe-effect estimated within probe.type

Probe effects can also be estimated within probe type or within probe type for each level of the primary treatment factor variable.

Model	`fitPLM` syntax
$y_{ijl} = x_j^T\theta + \alpha_{il} + \epsilon_{ij1}$	`PMMM ~ treatment + trt.cov + probe.type:probes`
$y_{ijl} = x_j^T\theta + \alpha_{ilm} + \epsilon_{ij1}$	`PMMM ~ treatment + trt.cov + treatment:probe.type:probes`

As an example, use the following code to fit such models and then examine the possible

```
> Pset <- fitPLM(Dilution,   PMMM ~ -1 + liver + probe.type:probes,subset = geneNames(D
> coefs.probe(Pset)[1]

> Pset <- fitPLM(Dilution,   PMMM ~ -1 + liver + liver:probe.type:probes,subset = geneN
> coefs.probe(Pset)[1]
```

2.4.8 PLM without chip level effects

It is possible to fit models which do not have any chip-level variables at all. If this is the case, then the probe type effect takes precedence over any probe effects in the model. That is it will be unconstrained.

Model	`fitPLM` syntax
$y_{ijl} = \alpha_i + \phi_{jl} + \epsilon_{ijl}$	`PMMM ~ -1 + probe.type + probes`
$y_{ijl} = \mu + \phi_{jl} + \alpha_i + \epsilon_{ijl}$	`PMMM ~ probe.type + probes`
$y_{ijl} = \phi_l + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ -1 + probe.type + treatment:probes`
$y_{ijl} = \mu + \phi_l + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ probe.type + treatment:probes`
$y_{ijl} = \mu + \phi_{lj} + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ samples:probe.type + treatment:probes`
$y_{ijl} = \mu + \phi_{lm} + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ treatment:probe.type + treatment:probes`

2.4.9 PLM with only probe-effects

It is also possible to fit models where only probe effects alone are estimated.

Model	`fitPLM` syntax
$y_{ij1} = \alpha_i + \epsilon_{ij1}$	`PM ~ -1 + probes`
$y_{ij1} = \mu + \alpha_i + \epsilon_{ij1}$	`PM ~ probes`
$y_{ij1} = alpha_{im} + \epsilon_{ij1}$	`PM ~ -1 + treatment:probes`
$y_{ij1} = \mu + (\theta\alpha)_{im_j} + \epsilon_{ij1}$	`PM ~ treatment:probes`
$y_{ij2} = \alpha_i + \epsilon_{ij2}$	`MM ~ -1 + probes`
$y_{ij2} = \mu + \alpha_i + \epsilon_{ij2}$	`MM ~ probes`
$y_{ij2} = \alpha_{im} + \epsilon_{ij2}$	`MM ~ -1 + treatment:probes`
$y_{ij2} = \mu + \alpha_{im} + \epsilon_{ij2}$	`MM ~ treatment:probes`
$y_{ijl} = \alpha_i + \epsilon_{ijl}$	`PMMM ~ -1 + probes`
$y_{ijl} = \mu + \alpha_i + \epsilon_{ijl}$	`PMMM ~ probes`
$y_{ijl} = \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ -1 + treatment:probes`
$y_{ijl} = \mu + \alpha_{im} + \epsilon_{ijl}$	`PMMM ~ treatment:probes`

2.4.10 Constraints

These are the constraints that will be imposed to make the models identifiable (when needed):

Parameter	Constraints	Default
β_i	$\sum_{j=0}^{J} \beta_i = 0$ or $\beta_1 = 0$	$\beta_1 = 0$
ϕ_l	$\sum_{l=1}^{2} \phi_l = 0$ or $\phi_1 = 0$	$\phi_1 = 0$
ϕ_{lj}	$\sum_{l=1}^{2} \phi_{lj} = 0$ or $\phi_{1j} = 0$	$\phi_{1j} = 0$
ϕ_{lm}	$\sum_{l=1}^{2} \phi_{lm} = 0$ or $\phi_{1m} = 0$	$\phi_{1m} = 0$
α_i	$\sum_{i=0}^{I} \alpha_i = 0$ or $\alpha_1 = 0$	$\sum_{i=0}^{I} \alpha_i = 0$
α_{im}	$\sum_{i=0}^{I} \alpha_{im} = 0$ or $\alpha_{1m} = 0$	$\sum_{i=0}^{I} \alpha_{im} = 0$
α_{il}	$\sum_{i=0}^{I} \alpha_{il} = 0$ or $\alpha_{1l} = 0$	$\sum_{i=0}^{I} \alpha_{il} = 0$
α_{ilm}	$\sum_{i=0}^{I} \alpha_{ilm} = 0$ or $\alpha_{1lm} = 0$	$\sum_{i=0}^{I} \alpha_{ilm} = 0$

In general, there is a general hierarchy by which items are left unconstrained:

intercept > `treatment` > `sample` > `probe.type` > `probes`

the highest term in this hierarchy that is in a particular model is always left unconstrained, everything else will be constrained. So for example a model containing probe.type and probe effects will have the probe.type effects unconstrained and the probe effects constrained.

Constraints are controlled using the `constraint.type` argument which is a vector with named items should be either `"contr.sum"` or `"contr.treatment"`. The names for this vector should be names of items in the model.

```
> data(Dilution)
```

```
> ##FIXME: remove next line
> Dilution = updateObject(Dilution)
> Pset <- fitPLM(Dilution, model = PM ~ probes + samples,constraint.type=c(samples="con

Warning: No default constraint specified. Assuming 'contr.treatment'.

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

[1] 7.633951 5.368074

> coefs(Pset)[1:2,]

                20A         20B
1000_at  0.03532897 0.008747425
1001_at -0.02135238 0.033203170
                 10A
1000_at -0.005110253
1001_at -0.033062279
```

3 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
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.

4 Dealing with the `PLMset` object

As previously mentioned, the results of a call to `fitPLM` are stored in a `PLMset` object. There are a number of accessor functions that can be used to access values stored in a `PLMset` including:

- `coefs` and `se`: access chip-level factor/covariate parameter and standard error estimates.
- `coefs.probe` and `se.probe`: access probe effect parameter and standard error estimates.
- `coefs.const` and `se.const`: access intercept, MM covariate and probe type effect parameter and standard error estimates.
- `weights`: access final weights from M-estimation procedure. Note that you may optionally supply a vector of probeset names as a second parameter to get weights for only a subset of probes.
- `resids`: access residuals. Note that you may optionally supply a vector of probeset names as a second parameter to get residuals for only a subset of probes.
- `varcov`: access variance matrices.

A M-estimation: How `fitPLM` fits models

Suppose we wish to fit the following model

$$y_i = f(\mathbf{x}_i, \theta) + \epsilon_i \tag{1}$$

where y_i is a response variable, $\mathbf{x}_i$ is a vector of explanatory variables, and θ is a vector of parameters to be estimated. An estimator of θ is given by

$$\min_\theta \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i, \theta)\right)^2 \tag{2}$$

which is the known least squares estimator. In some situations, outliers in the response variable can have significant effect on the estimates of the parameters. To deal with potential problem we need a robust method of fitting the model. One such method is known as M-estimation. An M-estimator for this regression, taking into account scale, is the solution of

$$\min_\theta \sum_{i=1}^{N} \rho\left(\frac{y_i - f(\mathbf{x}_i, \theta)}{s}\right) \tag{3}$$

where ρ is a suitable function. Reasonable properties for ρ include symmetry $\rho(x) = \rho(-x)$, a minimum at $\rho(0) = 0$, positive $\rho(x) \geq 0\ \forall x$ and increasing as the absolute value of x increases, i.e. $\rho(x_i) \geq \rho(x_j)$ if $|x_i| > |x_j|$. Furthermore, there the need to estimate s, where s is a scale estimate. One approach is to estimate both s and θ using a system of equations. The approach that we use is to estimate s using the median absolute deviation (MAD) which provides a robust estimate of scale. The above equation leads to solving

$$\sum_{i=1}^{N} \psi\left(\frac{y_i - f(\mathbf{x}_i, \theta)}{s}\right) = 0. \tag{4}$$

where ψ is the derivative of ρ. Note that strictly speaking, for robustness, ψ should be bounded. Now define $r_i = \frac{y_i - f(x_i,\theta)}{s}$ and a weight function $w(r_i) = \frac{\psi(r_i)}{r_i}$. Then the previous equation can be rewritten as

$$\sum_{i=1}^{N} w(r_i)\, r_i = 0 \tag{5}$$

which is the same as the set of equations that would be obtained if we were solving the iteratively re-weighted least squares problem

$$\min \sum_{i}^{N} w\left(r_i^{(n-1)}\right) {r_i^{(n)}}^2 \tag{6}$$

where the superscript (n) represents the iteration number. More details about M-estimation can be found in Huber (1981). Tables 1 and 2 describe the various different weight functions and associated default constants provided by `fitPLM`.

B Variance Matrix and Standard error estimates for `fitPLM`

Huber (1981) gives three forms of asymptotic estimators for the variance-covariance matrix of parameter estimates $\hat{b}$.

$$\kappa^2 \frac{\sum \psi^2/(n-p)}{(\sum \psi'/n)^2} (X^T X)^{-1} \tag{7}$$

$$\kappa \frac{\sum \psi^2/(n-p)}{\sum \psi'/n} V^{-1} \tag{8}$$

$$\frac{1}{\kappa} \frac{\sum \psi^2}{n-p} V^{-1} \left(X^T X\right) V^{-1} \tag{9}$$

where

$$\kappa = 1 + \frac{p}{n} \frac{\mathrm{Var}(\psi')}{E\psi'} \tag{10}$$

Method	$\rho(x)$	$\psi(x)$	$w(x)$
`Huber`	$\begin{cases}\text{if } \lvert x\rvert \le k \\ \text{if } \lvert x\rvert > k\end{cases}$	$\begin{cases}x^2/2 \\ k\left(\lvert x\rvert - k/2\right)\end{cases}$	$\begin{cases}x \\ k\operatorname{sgn}(x)\end{cases}$	$\begin{cases}1 \\ \frac{k}{\lvert x\rvert}\end{cases}$
`fair`	$c^2\left(\frac{\lvert x\rvert}{c} - \log\left(1+\frac{\lvert x\rvert}{c}\right)\right)$	$\frac{x}{1+\frac{\lvert x\rvert}{c}}$	$\frac{1}{1+\frac{\lvert x\rvert}{c}}$
`Cauchy`	$\frac{c^2}{2}\log\left(1+(x/c)^2\right)$	$\frac{x}{1+(x/c)^2}$	$\frac{1}{1+(x/c)^2}$
`Geman-McClure`	$\frac{x^2/2}{1+x^2}$	$\frac{x}{(1+x^2)^2}$	$\frac{1}{(1+x^2)^2}$
`Welsch`	$\frac{c^2}{2}\left(1-\exp\left(-\left(\frac{x}{c}\right)^2\right)\right)$	$x\exp\left(-(x/c)^2\right)$	$\exp\left(-(x/c)^2\right)$
`Tukey`	$\begin{cases}\text{if } \lvert x\rvert \le c \\ \text{if } \lvert x\rvert > c\end{cases}$	$\begin{cases}\frac{c^2}{6}\left(1-\left(1-(x/c)^2\right)^3\right) \\ \frac{c^2}{6}\end{cases}$	$\begin{cases}x\left(1-(x/c)^2\right)^2 \\ 0\end{cases}$	$\begin{cases}\left(1-(x/c)^2\right)^2 \\ 0\end{cases}$
`Andrews`	$\begin{cases}\text{if } \lvert x\rvert \le k\pi \\ \text{if } \lvert x\rvert > k\pi\end{cases}$	$\begin{cases}k^2(1-\cos(x/k)) \\ 2k^2\end{cases}$	$\begin{cases}k\sin(x/k) \\ 0\end{cases}$	$\begin{cases}\frac{\sin(x/k)}{x/k} \\ 0\end{cases}$

Table 1: ρ, ψ and weight functions for some common M-estimators.

Method	**Tuning Constant**
`Huber`	1.345
`fair`	1.3998
`Cauchy`	2.3849
`Welsch`	2.9846
`Tukey`	4.6851
`Andrews`	1.339

Table 2: Default tuning constants (k or c) for M-estimation ρ, ψ and weight functions.

$$V = X^T \Psi' X \tag{11}$$

and Ψ' is a diagonal matrix of ψ' values. When using `fitPLM` these are selected using `se.type=1`, `se.type=2`, or `se.type=3` respectively. Treating the regression as a weighted least squares problem would give

$$\frac{\sum w(r_i) r_i^2}{n-p} \left(X^T W X\right)^{-1} \tag{12}$$

as the estimator for variance covariance matrix, where W is a diagonal matrix of weight values. This option is selected by using `se.type=4`.

References

B. M. Bolstad. *Low Level Analysis of High-Density Oligonucleotide Data: Background, Normalization and Summarization.* PhD thesis, University of California, Berkeley, 2004.

B. M. Bolstad, R. A. Irizarry, M. Astrand, 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, Jan 2003.

P. J. Huber. *Robust statistics.* John Wiley & Sons, Inc, New York, New York, 1981.

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 Res*, 31(4):e15, Feb 2003a.

R. A. Irizarry, B. Hobbs, F. Collin, Y. D. Beazer-Barclay, K. J. Antonellis, U. Scherf, and T. P. Speed. Exploration, normalization, and summaries of high density oligonucleot ide array probe level data. *Biostat*, 4(2):249–264, 2003b. URL `http://biostatistics.oupjournals.org/cgi/content/abstract/4/2/249`.