multiscan: Combining multiple laser scans of microarrays

Mizanur R. Khondoker*[†] Chris A. Glasbey [‡] Bruce J. Worton [§]

April 21, 2009

Contents

1 Introduction

The sensitivity level of microarray scanners is adjustable and plays a crucial role in getting reliable measurement of the fluorescence intensity. A change in scanner setting transforms the intensity measurements by a multiplicative constant. A scanner's sensitivity has to be raised to a certain

*Package maintainer, Email: Mizanur.Khondoker@ed.ac.uk

[†]Division of Pathway Medicine, The University of Edinburgh Medical School, The Chancellor's Building, 49 Little France Crescent, Edinburgh EH16 4SB, UK.

[‡]Biomathematics & Statistics Scotland, King's Buildings, Edinburgh, EH9 3JZ, Scotland, UK. Email:chris@bioss.ac.uk.

[§]School of Mathematics, University of Edinburgh, King's Buildings, Edinburgh, EH9 3JZ, Scotland, UK. Email: Bruce.Worton@ed.ac.uk.

level to ensure that the intensity levels of weakly expressed genes exceed the intrinsic noise level of the scanner and so become measurable. This may, however, cause another problem: signal censoring for highly expressed genes. Scanners cannot record pixel intensities above some software dependent threshold ($2^{16} - 1 = 65535$, for a 16-bit computer storage system), so highly expressed genes can have pixel values which are right censored at the largest possible value that the scanner software allows. It is not usually possible to find a scanner setting which is optimal for both weakly and highly expressed genes. So, it seems reasonable to consider multiple scanning of the same microarray at different scanner settings and estimate spot intensities from these combined data.

The `multiscan` package implements the method of Khondoker *et al.* (2006) for estimating gene expressions from multiple laser scans of hybridised microarrays. The method is based on a non-linear functional regression model with both additive and multiplicative error terms. Maximum likelihood estimation based on a Cauchy distribution is used to fit the model, which reduces the sampling variability in expression estimates and is able to estimate gene expressions taking account of outliers and the systematic bias caused by signal censoring of highly expressed genes.

The package contains a function `multiscan`, and a small data set `murine` for illustrating the use of the function. The function produces output of class `multiscan`. S3 print and plot methods are also defined for this new class.

After installation, the package can be loaded using

```
> library(multiscan)
```

2 Package documentation

2.1 Help files

After installing and loading the package, the help file with detailed information of the `multiscan` function, its usage, arguments and values, can be viewed by issuing the command `help(multiscan)` or `?multiscan` at the R command prompt.

2.2 Package vignettes

To view the package vignette (this document) you need to use

```
> vignette("multiscan")
```

and a PDF file will open in your PDF reader. For Windows, the package vignette can also be accessed through the R menu bar item *Vignettes*.

3 Data

The only required argument for the `multiscan` function is the `data` argument which should be a numeric matrix or data frame containing the intensity data of a single microarray scanned at multiple (two or more) scanner settings. The data frame can be created by reading text or csv files into R, using `read.table` or `read.csv` command.

The number of rows (`n`) of the data matrix should be equal to the number of spots/probes on the array, and the number of columns (`m`) equal to the number of scans. Replicated probes on the array are treated as individual spots. Columns can be given in any order, but will be arranged in order of scanner's sensitivity before fitting the model. The data should be on the raw scale, i.e., not transformed!

An example data set, `murine`, has been included as part of the package, which can be loaded using

```
> data(murine)
```

and the first few rows of the data can be viewed by typing

```
> murine[1:10, ]
```

```
       scan1     scan2     scan3     scan4
1  1145.1677 1778.3423 3138.7314  4516.025
2   296.0000  472.0000  869.0000  1374.000
3   467.0000  692.0000 1179.8000  1992.000
4   795.4694 1247.3893 2176.6643  3329.654
5  2084.2793 3334.1765 6084.2651 10014.496
6   525.0000  688.6923 1169.9434  1825.786
7   266.2000  384.1875  732.1111  1072.600
8   495.9074  742.3148 1313.9796  2009.833
9   565.9101  895.0319 1616.3943  2495.142
10 1160.0336 1832.7114 3259.6541  5295.662
```

To see a brief description of this example data use

```
> help(murine)
```

4 Example usage of `multiscan`

Suppose that we want to obtain estimates of gene expressions combining the 4 scans of the `murine` data set. This can be done using

```
> data(murine)
> fit <- multiscan(murine)
```

```
Estimating 1000 gene expressions using 4 scans of data
------------------------------------------------------------
Log-likelihood at initial parameters:  -21348.7187
------------------------------------------------------------
End of global iteration: 1, Log-likelihood:   -19542.94736
End of global iteration: 2, Log-likelihood:   -19373.08458
End of global iteration: 3, Log-likelihood:   -19351.64478
End of global iteration: 4, Log-likelihood:   -19349.44816
End of global iteration: 5, Log-likelihood:   -19348.63585
End of global iteration: 6, Log-likelihood:   -19348.63585
```

```
> fit

Call:  multiscan(data = murine)

Scanning effects:
   scan1      scan2      scan3      scan4
1.000000   1.566959   2.764289   4.328168

Scale parameters:
      Additive   Multiplicative               Nu
   5.241983306      0.006761393      0.390186429

Log-likelihood at convergence:
    Loglf
-19348.64
```

To view the components of the fitted model use

```
> str(fit)

List of 12
 $ call   : language multiscan(data = murine)
 $ beta   : num [1:4] 1 1.57 2.76 4.33
 $ scale  : num [1:3] 5.24198 0.00676 0.39019
 $ mu     : num [1:1000] 1137 313 460 792 2127 ...
 $ data   : num [1:1000, 1:4] 1145 296 467 795 2084 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
  .. ..$ : chr [1:4] "scan1" "scan2" "scan3" "scan4"
 $ fitted : num [1:1000, 1:4] 1137 313 460 792 2127 ...
 $ sdres  : num [1:1000, 1:4] 0.868 -3.042 1.077 0.421 -2.813 ...
 $ outerit: int 6
 $ gconv  : int 0
 $ convmu : int [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
 $ conv   : int 0
 $ loglf  : num -19349
 - attr(*, "class")= chr "multiscan"
```

The estimated gene expressions can be obtained by extracting the `mu` component of the fitted object, i.e., by doing

```
> gene.exprs <- fit$mu
```

A plot of the fitted model can be created by using the generic `plot` function. For example, the plot in Figure 1 was produced using

```
> plot(fit)
```

To view the description of the other optional arguments of the `multiscan` function please refer to the help file

```
> help(multiscan)
```

5 Model diagnostics

The standardised residuals of the fitted model on the input data are stored in the `sdres` component of the fitted `multiscan` object. These residuals can be plotted against the rank of the estimated expression values using

```
> op <- par(mfrow = c(2, 2))
> plot(fit, residual = TRUE)
> par(op)
```

The resulting plot is shown in Figure 2.

6 Computing time and memory usage

The function is computationally slow and memory-intensive. That is due to the nested iteration loops of the numerical optimization of the likelihood function involving a large number ($n+m+2$) of parameters. The optimization uses an alternating algorithm with the Nelder-Mead simplex method (Nelder and Mead, 1965) in the inner loops. The function `multiscan` directly uses the C function `nmmin`, the internal code used in the general-purpose optimization tool `optim`, for implementing the Nelder-Mead simplex method.

For large data sets with many tens of thousands of probes, it is recommended to consider first fitting the model using a random subset (e.g. 10,000 rows) of the data matrix, and then using the estimated scanning effects and scale parameters obtained as initial values for fitting the model to the full data set.

7 Alternative software

A web interface, created by David Nutter of Biomathematics & Statistics Scotland (BioSS), based on the original Fortran code written by Khondoker *et al.* (2006) is available at `http://www.bioss.ac.uk/ktshowcase/create.cgi`. Although it uses the same algorithm, results from the web interface may not be exactly identical to that of `multiscan` as it uses a different (non-free IMSL routine) implementation of the Nelder-Mead simplex method.

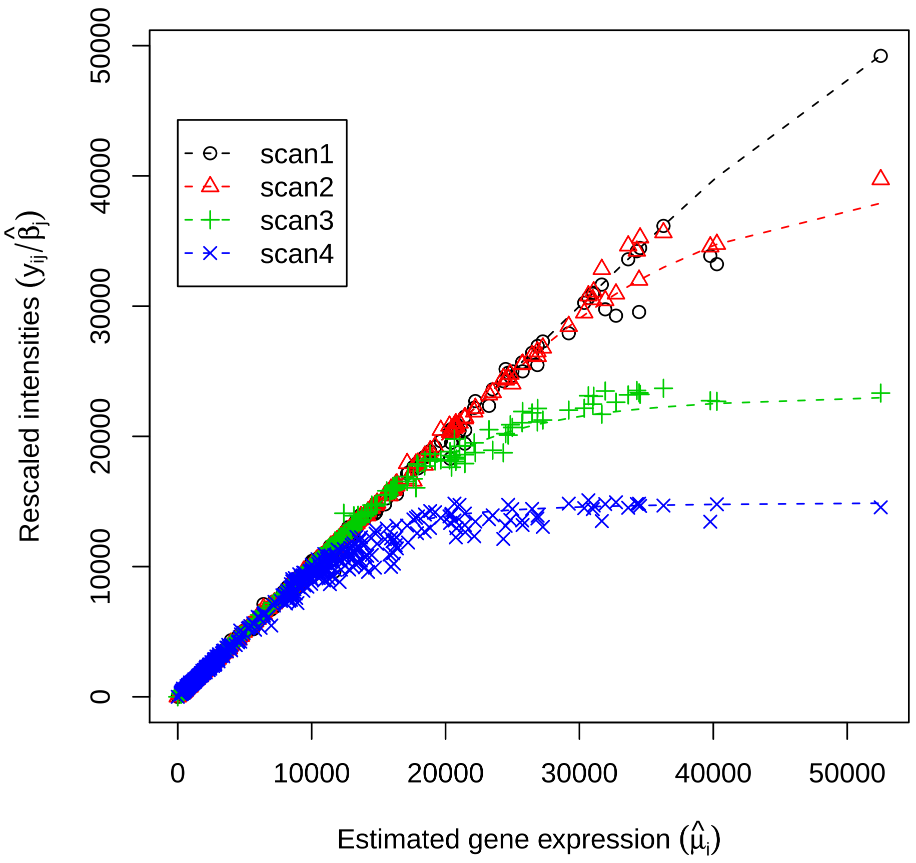

Figure 1: A plot of the rescaled intensities $(y_{ij}/\hat{\beta}_j)$ against the estimated gene expressions. The dashed lines represent the fitted model.

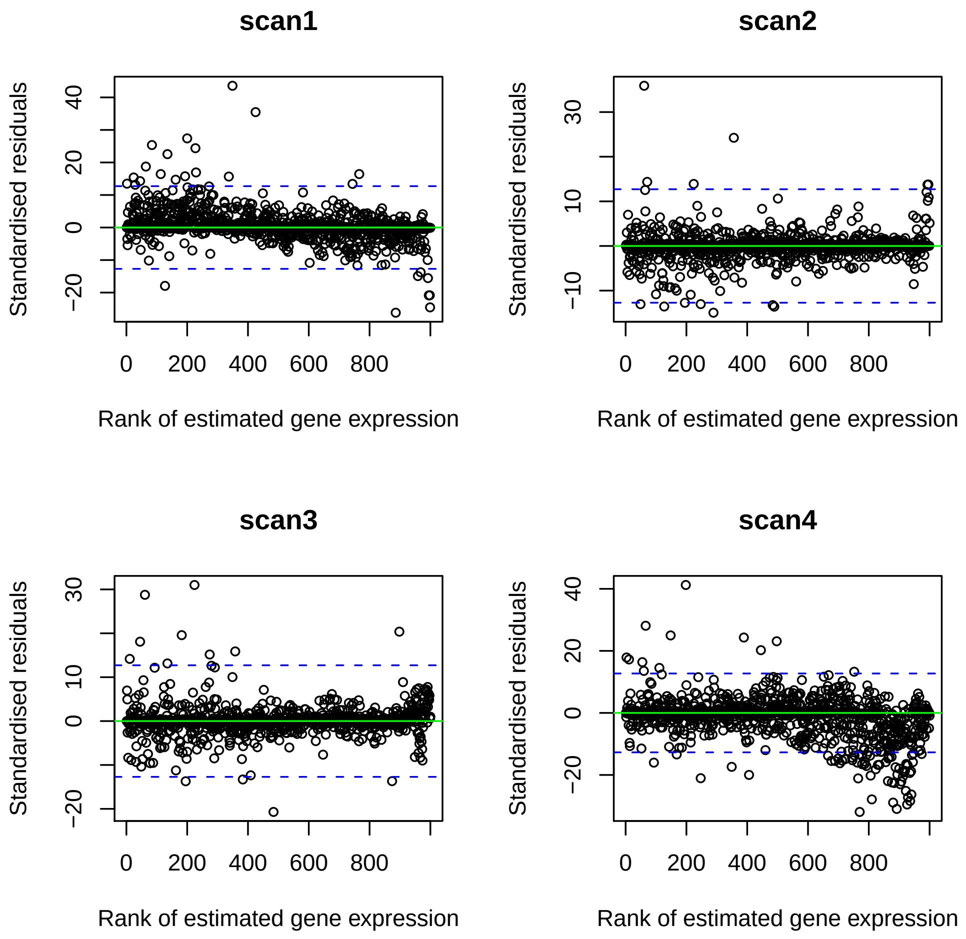

Figure 2: A plot of standardised residuals against the rank of estimated gene expressions. The dashed lines show 95% probability limits (±12.71).

8 Acknowledgments

We thank Marc Carlson, our colleague Thorsten Forster, and the anonymous reviewers for their useful comments on the earlier version of the package. The work of M.R.K. and C.A.G. in Khondoker *et al.* (2006) was supported by the Scottish Executive Environment and Rural Affairs Department (SEERAD).

9 References

Khondoker, M. R., Glasbey, C. A. and Worton, B. J. (2006). Statistical estimation of gene expression using multiple laser scans of microarrays. *Bioinformatics* **22**, 215–219.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal* **7**, 308–313.