# affyNFM: R implementation of a probe-level nested factorial model for Affymetrix data

John R. Stevens[*]

June 10, 2010

[*] Assistant Professor of Statistics, Department of Mathematics and Statistics, and Center for Integrated Biosystems, Utah State University
(`http://www.stat.usu.edu/~jrstevens`)

## 1 Introduction

The *affyNFM* R code provides tools to fit a probe-level nested factorial model for small-sample Affymetrix data. This model is presented and evaluated in the Stevens et al. (2010) manuscript "A Comparison of Probe-Level and Probeset Models for Small-Sample Gene Expression Data.". This tutorial vignette provides a guide for the use of these tools.

**Note:** If you use the *affyNFM* R code please cite Stevens et al. (2010).

## 2 Sample data

For purposes of illustration in this tutorial, we use the `spikein95` data provided with the *SpikeInSubset* R package:

```
> library(SpikeInSubset)
> data(spikein95)
```

In this `AffyBatch` object there are six arrays, with 16 (of 12,626) probesets spiked-in. The names of these probesets and the relative spike-in concentrations can be seen:

```
> pData(spikein95)
```

```
                  37777_at 684_at 1597_at 38734_at 39058_at 36311_at 36889_at
1521a99hpp_av06       0.00   0.25     0.5        1        2        4        8
1532a99hpp_av04       0.00   0.25     0.5        1        2        4        8
2353a99hpp_av08       0.00   0.25     0.5        1        2        4        8
1521b99hpp_av06       0.25   0.50     1.0        2        4        8       16
1532b99hpp_av04       0.25   0.50     1.0        2        4        8       16
2353b99hpp_av08r      0.25   0.50     1.0        2        4        8       16
                  1024_at 36202_at 36085_at 40322_at 407_at 1091_at 1708_at
1521a99hpp_av06        16       32       64      128   0.00     512    1024
1532a99hpp_av04        16       32       64      128   0.00     512    1024
2353a99hpp_av08        16       32       64      128   0.00     512    1024
1521b99hpp_av06        32       64      128      256   0.25    1024       0
1532b99hpp_av04        32       64      128      256   0.25    1024       0
2353b99hpp_av08r       32       64      128      256   0.25    1024       0
                  33818_at 546_at
1521a99hpp_av06        256     32
1532a99hpp_av04        256     32
2353a99hpp_av08        256     32
1521b99hpp_av06        512     64
1532b99hpp_av04        512     64
2353b99hpp_av08r       512     64
```

## 3 Sample affyNFM call

In this section we will use the sample data and apply the nested factorial model (NFM) to identify genes (or probesets) that are differentially expressed between the control and treatment conditions.

### 3.1 Create necessary objects

First load necessary libraries:

```
> library(affy)
> library(nlme)
> library(perm)
```

Next create the `AffyBatch` object, which we will call `use.abatch`. In practice this would most likely be accomplished using the `ReadAffy` function, but here (for the purposes of reproducible demonstration) we load an existing `AffyBatch` object.

```
> library(SpikeInSubset)
> data(spikein95)
> use.abatch <- spikein95
```

Next define control (`use.t1`) and treatment (`use.t2`) array indices. In our example, arrays 1, 2, and 3 are control, and arrays 4, 5, and 6 are treatment:

```
> use.t1 <- c(1, 2, 3)
> use.t2 <- c(4, 5, 6)
```

To save time (in this demonstration), we define a subset of genes to use by creating a vector (`use.gn`) of `geneNames`. We randomly select 100 gene names and add the 16 known spike-in genes. This subset of genes will be tested for differential expression. In practice, the subset may be identified using non-specific filtering (Hackstadt and Hess 2009).

```
> gn.spike <- colnames(pData(use.abatch))
> set.seed(1234)
> gn.others <- sample(geneNames(use.abatch), 100)
> use.gn <- c(gn.spike, gn.others)
```

The *affyNFM* tools involve computationally expensive iterative procedures, so it may be desirable to save the NFM results. We define a local directory (`use.wd`) to save these results. In practice, this would be a directory easily accessible to the user.

```
> use.wd <- "C:/folder"
```

We define a filename base (`use.filename`) to save the NFM results in the specified directory. Note that we do not include a file extenstion, as the ".csv" extension will be added automatically by the `affyNFM` function.

```
> use.filename <- "sample.results"
```

With the necessary objects now defined, we source in the *affyNFM* code:

```
> source("http://www.stat.usu.edu/~jrstevens/affyNFM.R")
```

This *affyNFM* code defines several functions, with two main functions of interest: `affyNFM` calculates the NFM F-statistics and `nfm.pvals` calculates the NFM permutation p-values.

## 3.2 F-statistic calculation

The main `affyNFM` function calculates the NFM F-statistic for each probeset, after RMA background correction and quantile normalization. To facilitate subsequent p-value calculation, the F-statistics are calculated for all possible (and non-redundant) permutations of treatment labels.

The arguments to this `affyNFM` function are as follows:

1. `abatch` – the (raw) `AffyBatch` object to be analyzed
2. `t1` – the array indices of the control samples
3. `t2` – the array indices of the treatment samples
4. `gn` – (optional) the subset of `geneNames` to be tested for differential expression using the NFM. If not provided, the full set of `geneNames` represented on the `abatch` object is used.
5. `wd` – (optional) the working directory specifying where to save the results of the NFM. If not provided (but `filename` is provided), the current working directory (`getwd()`) is used.
6. `filename` – (optional) The filename base of the results to be saved. If provided, the `affyNFM` function will create a `filename.csv` file in the `wd` directory. If not provided, the `affyNFM` function will return a `data.frame` object.
7. `progress` – (optional) A filename base where a progress report will be saved. If not specified (but `filename` is specified), then the filename base will be `progress_filename`. A ".csv" file will be created in the `wd` directory. This may be useful to monitor run-time.
8. `verbose` – (optional) A TRUE/FALSE logical indicating whether or not to send detailed progress to output. This is different from the file output controlled by the `progress` argument. The default is TRUE.
9. `start` – (optional) An integer specifying which iteration number to begin in the permutation F-calculation. This is useful in cases of restarting, for debugging purposes. If restart occurs, be sure to rename and save results for previous iterations (`filename`). The default value is 1.
10. `perms` – (optional) A TRUE/FALSE logical indicating whether or not to calculate the NFM F-statistics for all non-redundant permutations of treatment labels. The default value is TRUE.

A call to the `affyNFM` function creates a `data.frame` object with columns `gn` (for gene name), `F.original` (the F-statistic for the original treatment labels), and `F.2`, ..., `F.nperms`, where `nperms` is the number of non-redundant permutations of treatment labels. If the `filename` argument is specified, then this `data.frame` object will be saved as the `filename.csv` file in the `wd` directory. If the `filename` argument is not specified, then this `data.frame` object is returned. Because of the computational expense to create this `data.frame` object, it is recommended to specify the `filename` argument so that the results are saved to file.

### 3.2.1 Non-permutation approach

To save computational time, or if p-value calculation will not be necessary, it is possible to obtain only the F-statistics for the original treatment labels by specifying `perms=FALSE`:

```
> F.frame0 <- affyNFM(abatch = use.abatch, t1 = use.t1, t2 = use.t2,
+     gn = use.gn, perms = FALSE)
```

```
Thu Jun 10 12:09:20 2010 Performing background correction and quantile normalization...
Thu Jun 10 12:09:38 2010 Performing nfm  ...
Thu Jun 10 12:10:09 2010 Non-permutation F-statistic calculation complete.
```

```
> head(F.frame0)
```

```
        gn       F.original
1 37777_at 25.6765994741892
2   684_at 9.58904372054756
3  1597_at 5.78032489098403
4 38734_at 42.8220076748622
5 39058_at 38.1179172895468
6 36311_at 49.1593223340546
```

### 3.2.2 All-permutations approach, saving results to file

Here we calculate the F-statistics for all non-redundant permutations and save the results to file.

```
> affyNFM(abatch = use.abatch, t1 = use.t1, t2 = use.t2, gn = use.gn,
+     wd = use.wd, filename = use.filename)
```

```
Thu Jun 10 12:10:09 2010 Performing background correction and quantile normalization...
Thu Jun 10 12:10:24 2010 Performing nfm on iteration 1 of 10 ...
Thu Jun 10 12:10:51 2010 Performing nfm on iteration 2 of 10 ...
Thu Jun 10 12:11:19 2010 Performing nfm on iteration 3 of 10 ...
Thu Jun 10 12:11:45 2010 Performing nfm on iteration 4 of 10 ...
Thu Jun 10 12:12:11 2010 Performing nfm on iteration 5 of 10 ...
Thu Jun 10 12:12:39 2010 Performing nfm on iteration 6 of 10 ...
Thu Jun 10 12:13:05 2010 Performing nfm on iteration 7 of 10 ...
Thu Jun 10 12:13:31 2010 Performing nfm on iteration 8 of 10 ...
Thu Jun 10 12:13:57 2010 Performing nfm on iteration 9 of 10 ...
Thu Jun 10 12:14:23 2010 Performing nfm on iteration 10 of 10 ...
Thu Jun 10 12:14:50 2010 Empirical sampling distribution complete.
Results saved as sample.results.csv in directory C:/folder
```

Then the results are read back in to create the `data.frame` object `use.frame`:

```
> use.frame <- read.csv(paste(use.wd, "/", use.filename, ".csv",
+     sep = ""))
> head(use.frame)
```

```
        gn F.original         F.2        F.3        F.4       F.5       F.6
1 37777_at  25.676599 0.154893615 0.57092669 0.17835347 0.3033485 0.8860315
2   684_at   9.589044 0.302653914 0.15809471 0.17460500 0.5895994 0.3623032
3  1597_at   5.780325 0.003302107 0.05236453 0.37210233 2.7201865 1.2900620
4 38734_at  42.822008 0.579834873 0.08875088 0.04533029 1.7471837 0.5329230
5 39058_at  38.117917 1.055835677 0.55499753 0.17294685 0.4088933 0.1608452
6 36311_at  49.159322 1.810655086 0.37215615 0.55222583 1.1122678 0.1666232
         F.7       F.8        F.9      F.10
1 0.33771432 0.5961576 1.38366286 0.5462265
2 0.38901078 0.8853128 0.83794041 1.2396464
3 0.44301189 0.2581654 0.87816291 1.8921279
4 0.39969612 0.2843751 0.39302122 1.3883087
5 0.01048544 0.3890653 0.97783473 1.7450757
6 0.28174217 0.1727109 0.08822834 0.8243724
```

Because the results were saved to file, a progress report was also created. The contents of this progress report file can be used to monitor run-time:

```
> progress.frame <- read.csv(paste(use.wd, "/progress_", use.filename,
+     ".csv", sep = ""))
> progress.frame
```

```
                       time iteration pct.complete
1  Thu Jun 10 12:10:24 2010         0            0
```

```
2  Thu Jun 10 12:10:51 2010        1          10
3  Thu Jun 10 12:11:19 2010        2          20
4  Thu Jun 10 12:11:45 2010        3          30
5  Thu Jun 10 12:12:11 2010        4          40
6  Thu Jun 10 12:12:39 2010        5          50
7  Thu Jun 10 12:13:05 2010        6          60
8  Thu Jun 10 12:13:31 2010        7          70
9  Thu Jun 10 12:13:57 2010        8          80
10 Thu Jun 10 12:14:23 2010        9          90
11 Thu Jun 10 12:14:50 2010       10         100
```

### 3.2.3 All-permutations approach, not saving results to file

We could call the `affyNFM` function without saving results to file, by not specifying a `filename` argument. Here we also demonstrate supression of progress output (to the R terminal) by specifying `verbose=FALSE`:

```
> use.frame1 <- affyNFM(abatch = use.abatch, t1 = use.t1, t2 = use.t2,
+     gn = use.gn, verbose = FALSE)
> head(use.frame1)
```

```
        gn       F.original                 F.2               F.3
1 37777_at 25.6765994741892   0.154893614562845  0.570926690897756
2   684_at 9.58904372054756   0.302653913506587  0.158094707126131
3  1597_at 5.78032489098403 0.00330210745250836 0.0523645308950134
4 38734_at 42.8220076748622   0.579834873120239 0.0887508822531098
5 39058_at 38.1179172895468    1.05583567744497  0.554997530656564
6 36311_at 49.1593223340546    1.81065508632521  0.372156154233074
                F.4               F.5               F.6                F.7
1  0.178353470062144 0.303348528976957  0.88603154295087   0.33771431582125
2  0.174605001161088 0.589599377598369   0.3623031949798  0.389010777310083
3  0.372102334713257  2.72018645841031  1.29006201610316  0.443011887349268
4 0.0453302868590168  1.74718365560216 0.532922994618565  0.399696124974263
5  0.172946850738009 0.408893281997868 0.160845164164175 0.0104854439155456
6  0.552225827661248  1.11226775662473 0.166623208695875  0.281742167701483
               F.8               F.9              F.10
1 0.596157595706377  1.38366285762355 0.546226503089202
2 0.885312834222096 0.837940408668387  1.23964643799936
3 0.258165430067475 0.878162908410805  1.89212788535138
4  0.28437511173146 0.393021223160814  1.38830872048363
5 0.389065288100556 0.977834732317531  1.74507567806407
6  0.17271086919443 0.088228341202868 0.824372434122621
```

Note that this `use.frame1` object is equivalent to the previously defined `use.frame` object.

### 3.3 P-value calculation from permutation results

If NFM F-statistics were calculated for all non-redundant permutations of treatment labels, then a permutation p-value can be calculated for each gene by calling the `nfm.pvals` function. This function takes just one argument, a `data.frame` object in the same format as returned by the `affyNFM` function. The `nfm.pvals` function returns a `data.frame` object with named columns representing the gene name, original NFM F-statistic, and permutation p-value.

```
> use.frame <- read.csv(paste(use.wd, "/", use.filename, ".csv",
+     sep = ""))
> pframe <- nfm.pvals(use.frame)
> head(pframe)
```

```
        gn         F           p
1 37777_at 25.676599 0.009482759
2   684_at  9.589044 0.019827586
3  1597_at  5.780325 0.050000000
4 38734_at 42.822008 0.004310345
5 39058_at 38.117917 0.006896552
6 36311_at 49.159322 0.003448276
```

## 4 Assessment of sample analysis

For purposes of demonstration with these spike-in data, we can convert the p-values to q-values and check which of the spike-in probesets were identified as significantly differentially expressed by the NFM approach when controlling the FDR at 0.10. We first generate the necessary `data.frame` object (`g`). Note that in addition to the three columns returned by the `nfm.pvals` function, this object `g` has additional columns for the q-value (`q`), the control concentration (`C`), and the treatment concentration (`T`), for only the spike-in probesets.

```
> library(qvalue)
> pframe$q <- qvalue(p = pframe$p)$q
> t.spike <- is.element(pframe$gn, gn.spike)
> conc.ctl <- as.numeric(pData(use.abatch)[1, ])
> conc.trt <- as.numeric(pData(use.abatch)[4, ])
> f <- data.frame(gn = gn.spike, C = conc.ctl, T = conc.trt)
> g <- merge(f, pframe)
> head(g)

        gn      C    T          F           p          q
1  1024_at   16.0   32  40.306978 0.005172414 0.09765192
2  1091_at  512.0 1024  31.997610 0.008620690 0.09765192
3  1597_at    0.5    1   5.780325 0.050000000 0.35398820
4  1708_at 1024.0    0 812.866437 0.000862069 0.09765192
5 33818_at  256.0  512 170.847848 0.001724138 0.09765192
6 36085_at   64.0  128  35.654643 0.007758621 0.09765192
```

Now we generate Figure 1 to summarize the result. Note that similar figures are reported in Stevens et al. (2010).

## References

[1] Hackstadt, A.J. and Hess, A.M. (2009) "Filtering for Increased Power for Microarray Data Analysis," *BMC Bioinformatics*, 10:11.

[2] Stevens, J.R., Bell, J.L, Aston, K.I., and White, K.L. (2010) "A Comparison of Probe-Level and Probeset Models for Small-Sample Gene Expression Data," *BMC Bioinformatics*, 11:281.

```
> eps <- 0.1
> plot(g$C + eps, g$T + eps, log = "xy", pch = 1, cex = 20 * g$q,
+     xlab = "Control Concentration (pM)", ylab = "Treatment Concentration (pM)",
+     main = "Sample Analysis: spike-in q-values", col.axis = NA)
> t <- g$q <= 0.1
> points(g$C[t] + eps, g$T[t] + eps, pch = 16, cex = 20 * g$q[t],
+     col = "#08519C")
> axis(side = 1, labels = c(0, 1, 10, 100, 1000), at = c(0.1, 1,
+     10, 100, 1000))
> axis(side = 2, labels = c(0, 1, 10, 100, 1000), at = c(0.1, 1,
+     10, 100, 1000))
```

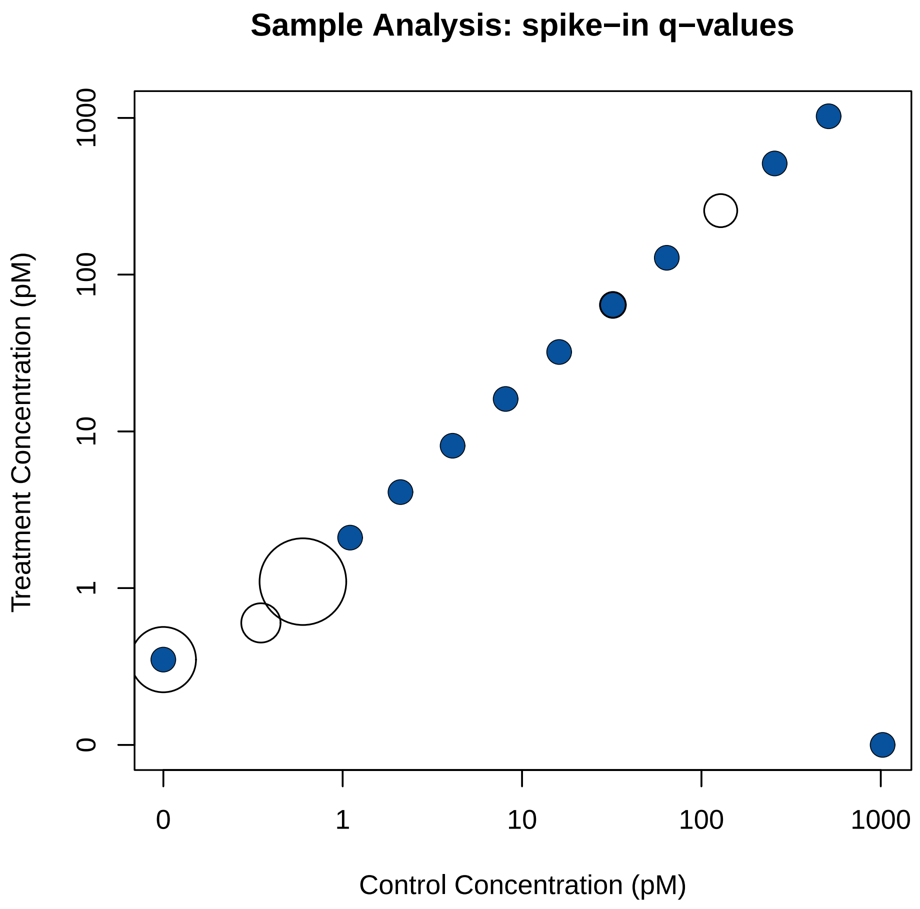

Figure 1: Bubble plot for the spike-in probesets in the sample analysis. The horizontal and vertical axes are the spike-in concentrations for the control and treatment conditions, with tick marks on the log scale. The size of the plotting character for each spike-in gene is proportional to the corresponding q-value (converted from NFM permutation p-value). Q-values less than 0.1 are represented as closed blue dots, while q-values greater than 0.1 are represented as open circles. Statistical significance (q-value $< 0.1$) is more common for genes with higher control and treatment concentrations.