Umpire: The Ultimate Microarray Prediction, Inference, and Reality Engine

Jiexin Zhang and Kevin R. Coombes

14 July 2009

1 Introduction

Version 1.0 of the Ultimate Microarray Prediction, Inference, and Reality Engine (Umpire) is an R package that allows researchers to simulate complex, realistic microarray data. Statisticians and bioinformaticians who develop and improve methods to analze microarray data recognize that it is difficult to evaluate methods on real data where "ground truth" is unknown, and they frequently turn to simulations where they can control the true underlying structure. In many instances, however, the simulations that have been performed are rather simplistic. Often, genes are treated as independent, in spite of the elaborate correlation structures that give rise to networks and pathways in real biology. Differential epxression is frequently simulated using two homogeneous groups following nearly perfect normal distributions, with the amount of differential expression identical for all genes. The `Umpire` package, which is invoked by the command

```
> library(Umpire)
```

provides tools that allow users to simulate microarray data from a more realistic model.

2 The gene expression model

2.1 Engines

The fundamental object in `Umpire` is a "random-vector generator" (RVG), which is represented by the `Engine` class. Equivalently, each `Engine` object represents a specific multivariate distribution, from which random vectors can be generated using the generic `rand` method. In Version 1.0 of `Umpire`, we include three basic building blocks for these kinds of distributions: independent normal, independent log normal, and multivariate normal. The following example creates an object that will generate vectors of length 3.

```
> nGenes <- 3
> means <- rnorm(nGenes, 6, 1)
> sds <- 1/rgamma(nGenes, rate=14, shape=6)
> indn <- IndependentNormal(means, sds)
> summary(indn)
```

```
An IndependentNormal object, representing a vector
of length 3 of independent normal random variables.
```

```
> indn
```

```
An object of class "IndependentNormal"
Slot "mu":
[1] 4.817261 5.168326 4.630687

Slot "sigma":
[1] 3.839131 2.109463 1.206702
```

Now we generate five vectors from this distribution.

```
> x <- rand(indn, 5)
> x
```

```
         [,1]      [,2]      [,3]     [,4]     [,5]
[1,] 4.508216 11.083197 0.8589291 1.657415 9.542727
[2,] 4.330626  6.984256 5.8524106 4.963475 7.636607
[3,] 4.872833  3.611485 4.2672394 3.785379 1.902016
```

We use a similar method to create an object that generates independent log normal data.

```
> nGenes <- 4
> logmu <- rnorm(nGenes, 6, 1)
> logsigma <- 1/rgamma(nGenes, rate=14, shape=6)
> indLN <- IndependentLogNormal(logmu, logsigma)
> indLN
```

```
An object of class "IndependentLogNormal"
Slot "logmu":
[1] 4.666941 6.237990 5.137024 6.186839

Slot "logsigma":
[1] 2.144192 1.732211 2.085422 3.246242
```

In order to create a multivariate normal RVG, we must specify the mean vector and the covariance matrix. Here we start with the correlation matrix for a simple two-dimensional RVG.

```
> a <- runif(1)
> b <- sqrt(1-a^2)
> X <- matrix(c(a, b, -b, a), 2, 2)
```

Next, we choose random positive squared-eigenvalues.

```
> Lambda2 <- diag(rev(sort(rexp(2))), 2)
```

We combine these into a covariance matrix.

```
> Y <- t(X) %*% Lambda2 %*% X
```

Finally, we use the MVN constructor

```
> mvn <- MVN(c(0,0), Y)
```

and use it to generate five random vectors.

```
> x <- rand(mvn, 5)
```

A general `Engine` is a list of RVG components, like those just created from the `IndependentNormal` or `MVN` constructors. For example, we can create an RVG Engine with two components using the command:

```
> engine <- Engine(list(indn, mvn))
> summary(engine)

An Engine with 2 components.

> data <- rand(engine, 5)
> data

           [,1]       [,2]       [,3]      [,4]       [,5]
[1,] -2.9760465 -1.4301555  6.6720999  6.846802  6.1459152
[2,]  7.1909455  3.4706992  4.6732073  7.323242  2.2245688
[3,]  3.2033364  4.1869240  3.7385078  5.675392  5.1522911
[4,]  1.1687270 -0.4317227  1.1841320 -1.001320  1.0863994
[5,] -0.2010683  1.4866611 -0.8089795 -1.458363 -0.6368591
```

3 Additive and Multiplicative Noise

We model the observed signal, Y_{gi}, for gene g in sample i as:

$$Y_{gi} = S_{gi} * exp(H_{gi}) + E_{gi}$$

where

$$S_{gi} = \text{true biological signal}$$

$$H_{gi} = \text{multiplicative noise}$$

$$E_{gi} = \text{additive noise}$$

The noise model represents technical noise that is layered on top of any biological variability when measuring gene expression in a set of samples. For example, background noise is usually

additive to the signal, and the variation between the signal pixels, which is independent of the magnitude of the signal, is the representative multiplicative noise [**?**]. We modeled both additive and multiplicative noise as normal distribution:

$$E_{gi} \sim \text{Normal}(\nu, \tau)$$

$$H_{gi} \sim \text{Normal}(0, \phi)$$

Note that we allow the additive noise to include a bias term (ν) that may represent, for example, a low level of cross-hybridization provding some level of signal at all genes.

The noise model is represented in the `Umpire` package by the `NoiseModel` class. You create a `NoiseModel` object by supplying values for ν, τ, and ϕ.

```
> noise <- NoiseModel(30, 40, 0.10)
> noise

An object of class "NoiseModel"
Slot "additiveOffset":
[1] 30

Slot "additiveScale":
[1] 40

Slot "multiplicativeScale":
[1] 0.1
```

Use the `blur` function to add noise to a data matrix. For example,

```
> ndata <- blur(noise, data)
> summary(data)
```

```
       V1                 V2                 V3                 V4
 Min.   :-2.9760   Min.   :-1.4302   Min.   :-0.809   Min.   :-1.458
 1st Qu.:-0.2011   1st Qu.:-0.4317   1st Qu.: 1.184   1st Qu.:-1.001
 Median : 1.1687   Median : 1.4867   Median : 3.739   Median : 5.675
 Mean   : 1.6772   Mean   : 1.4565   Mean   : 3.092   Mean   : 3.477
 3rd Qu.: 3.2033   3rd Qu.: 3.4707   3rd Qu.: 4.673   3rd Qu.: 6.847
 Max.   : 7.1909   Max.   : 4.1869   Max.   : 6.672   Max.   : 7.323
       V5
 Min.   :-0.6369
 1st Qu.: 1.0864
 Median : 2.2246
 Mean   : 2.7945
 3rd Qu.: 5.1523
 Max.   : 6.1459
```

```
> summary(ndata)
```

```
       V1                V2                V3                V4                V5
 Min.   :-14.897   Min.   :-17.55   Min.   :-4.251   Min.   :-19.68   Min.   :-34.57
 1st Qu.:  6.581   1st Qu.: 13.12   1st Qu.:14.369   1st Qu.: -2.48   1st Qu.: 15.83
 Median :  9.120   Median : 19.13   Median :26.457   Median : 19.30   Median : 40.75
 Mean   : 19.185   Mean   : 38.10   Mean   :20.011   Mean   : 22.52   Mean   : 31.45
 3rd Qu.: 31.278   3rd Qu.: 78.23   3rd Qu.:30.883   3rd Qu.: 30.70   3rd Qu.: 65.20
 Max.   : 63.844   Max.   : 97.57   Max.   :32.597   Max.   : 84.77   Max.   : 70.07
```

4 Gene Expression

Let T_{gi} denote the expression of a transcriptionally active gene g in sample i. For most purposes, We allow T_{gi} to follow a log-normal distribution ($\log(T_g) \sim Normal(\mu_g, \sigma_g)$. In a class of samples, the mean expression of gene g on the log scale is denoted by μ_g and the standard deviation on the log scale is σ_g. Both μ_g and σ_g are properties of the gene itself and the sample class. Suppose, for example, in a class of samples, that on average, g_1 expresses at a higher level than g_2, and the variance of g_1 is smaller than g_2. Then, $\mu_{g_1} > \mu_{g_2}$ and $\sigma_{g_1} < \sigma_{g_2}$.

Within a given simulation, we typically place hyperdistributions on the log-normal parameters μ_g and σ_g. We take $\mu_g \sim Normal(\mu_o, sigma_0)$ to have a normal distribution with mean μ_0 and standard deviation σ_0. We take the σ_g to have an inverse gamma distribution with *rate* and *shape* parameters. Reasonable values for the hyperparameters can be estimated from real data. For instance, $\mu_0 = 6$ and $\sigma_0 = 1.5$ are typical values on the log scale of a microarray experiment using Affymetrix HG-U133A chip. The parameters for the inverse gamma distribution are determined by the method of moments from thwe desired mean and standard deviation; we have found that a mean of 0.65 and a standard deviation of 0.01 (for which rate= 28.11 and shape= 44.25) produce reasonable data.

Thus, we can create a simulation engine of this type by

```
> nGenes <- 4000
> mu0 <- 6
> sigma0 <- 1.5
> rate <- 28.11
> shape <- 44.25
> logmu <- rnorm(nGenes, mu0, sigma0)
> logsigma <- 1/rgamma(nGenes, rate=rate, shape=shape)
> indLN <- IndependentLogNormal(logmu, logsigma)
> engine <- Engine(list(indLN))
```

4.1 The Multi-hit Model of Cancer

The multiple hit theory of cancer was first proposed by Carl Nordling in 1953 [**?**] and extended by Alfred Knudson in 1971. The basic idea is that cancer can only result after multiple insults (mutations; hits) to the DNA of a cell.

4.2 Active and Inactive Genes

We model the true biological signal S_{gi}, for gene g in sample i, as a mixture:

$$S_{gi} \sim (1 - z_g) * \delta_0 + z_g * T_{gi}$$

In this model, δ_0 is a point mass at zero, z_g defines the activity state (1 = active, 0 = inactive), and T_{gi} is the expression of a transcriptionally active gene.By allowing for some genes to be transcriptionally inactive, this design takes into account that the transcriptional activity of most genes is conditional on the biological context. For example, tissue-specific genes, unlike housekeeping genes, only express in certain tissue samples. Activity is modeled in `Umpire` using a binomial distribution, $z_g \sim Binom(p_0)$. To create a simulation engine that inciorproates transcriptional activation, we write

```
> p0 <- 0.8
> engine <- EngineWithActivity(p0, list(indLN))
> summary(engine)

An Engine with 1 components.
Fraction of active genes 1
```

4.3 Correlated blocks of genes

Instead of simulating genes as independent entities, we consider blocks of correlated genes. Biologically, genes are usually interconnected in networks and pathways. Often, clustering methods are used to group genes into correlated blocks. Thus, it is natural to simulate microarray experiments from the perspective of blocks. Since the size of the blocks and degree of correlations among genes within a block depend on biological condition of samples, they need to be simulated within a reasonable range in order to study their effect on the microarray data analysis. As shown in Table 1, we allow the mean block size, bs, to range from 1 to 1000, and the sizes of gene blocks to vary around the pre-defined mean block size. To be more specific, the block size follow normal distribution with mean bs and standard deviation $0.3 * bs$. $bs = 1$ is a special case in which standard deviation of block size = 0. Thus, when $bs = 1$, there are no correlated blocks, which means all genes are independent of each other. On the other extreme, $bs = 1000$ demonstrates the situation in which the common theme is large networks involving many genes. We have also simulated blocks where the standard deviation = 0 for all mean block size, under which circumstance all blocks in a microarray experiments have the exactly same sizes. Comparing with variable block size, the setting of constant block size affects the variability of the parameters of interest. However, because we believe that the variable block size is more realistic, we will present in this paper only the results obtained from variable block size. Comparison between results from constant block size and from variable block size is shown in supplementary material.

As discussed in previous paragraph, we need to simulate the correlation among genes within a block. The correlation matrix for a block b, $cor.matrix_b$, has 1 on the diagnal and ρ_b or -ρ_b in off-diagnal places. We allow ρ to follow a beta distribution with parameters p and w: $Beta(pw, (1 -$

$p) * w$). With the setting of $p = 0.6$ and $w = 5$, most blocks are relatively highly correlated (mean of ρ is around 0.6). The portion of negatively correlated genes within a block is denoted by parameter *p.neg*. In the simplest set-up, we have all genes in the same block to have the same correlation ρ_b. Because ρ_b is always positive, this set-up means there is not negatively correlated genes within a block ($p.neg = 0$). In more complicated set-up, we allow the portion of negatively correlated genes within a block (*p.neg*) to be supported on [0,0.5]. Thus, we have *p.neg* = 0.5-abs(x-0.5) where x follow some beta distribution. Three scenarios were considered: (1) x beta(1,1), in which case the *p.neg* is uniformly distributed between 0 and 0.5; (2) x beta(5,5), in which case it is more likely that the *p.neg* is close to 0.5; (3) x beta(0.5,0.5), in which case it is more likely that the *p.neg* is close to 0. Figure **??**c shows the histogram of pair-wise correlations within 10000 genes and mean block size 50 when *p.neg* is uniformly distributed between 0 and 0.5. The distributions of pair-wise correlations for all four *p.neg* scenarios is shown in supplementary materials (fig:pneg)

We allow the log expression values of genes in a block to follow a multi-variate normal (MVN) distribution. The mean for the MVN object is defined by μ_g, and the covariance matrix is defined as the following:

$$cov.matrix[i, j] = cor.matrix[i, j] * \sigma_{g_i} * \sigma_{g_j}$$

where σ_{g_i} defines the standard deviation of gene i, which follows the inverse gamma distribution as described in previous section. *cor.matrix* denotes the correlation matrix as described in previous paragraph.

In previous section, we mentioned that some genes would be transcriptionally inactive under certain biological conditions. Instead of simulating this active status for each gene, we simulate the whole block of genes being transcriptionally active or inactive. This follows the argument that the whole pathway/network could be turned on or off under certain bioligical conditions. The active status of blocks for the microarray experiment follows a binomial distribution with parameter π which defines the portion of transcriptionally inactive blocks. When a block is turned off, z_g, the status indicator, is set to be 0 for all genes in this block, so that the true biological signals for these genes are zero.

4.4 Normal *vs* cancer samples

We simulate normal samples being a homogeneous population with *nGenes* genes and *nSamples* samples. We allow *nSamples* to vary from 10 to 100 in order to study the effect of number of independent observations on various test statistics. The same number of cancer samples are being generated with a portion of differentially expressed genes. We simulate differentially expressed genes in cancer samples by changing their mean expression values. Instead of changing individual genes, we perform this mean altering to blocks of genes in order to simulate the affect of cancer pathology on certain pathway/networks. *p.diff* is used to define the percentage of differentially expressed blocks which are then randomly selected from transcriptionally active blocks. We keep transcriptionally inactive blocks inactive in both normal and cancer samples in this setting. However, it is possible that an inactive block of genes in normal samples being turned on in cancer samples, or vise versa, when certain carcinogens work through pathways that are supposed to be off, or on, in normal samples. We will incorporate this level of complication in later implementations. The parameter

$diff.mean$ denotes the absolute changes of the mean expression values on log scale for a block of genes. $diff.mean$ follows a gamma distribution with parameter α and β (Figure **??**d). The α and β are both set be 10 so that the absolute fold change on log scale is 1 (thus 2 fold change on raw scale), and the long tail on the right hand side of the distribution indicates a few genes would have large fold changes. A gene in the changed block is randomly assigned to be up-regulated or down-regulated in cancer samples.

Using parameters described above and summarized in Table 1, Figure **??**e shows the distribution of the average of log expression values of 10000 genes from 10 simulated normal samples given the mean block size being 100. The bimodal profile is due to the fact that part of genes are transcriptionally inactive. Similarly, 10 cancer samples were generated with 10% differentially expressed blocks. Figure **??**g shows the log mean expression values of these 10000 genes in normal samples verse those in cancer samples. Red dots represent those genes that differentially expressed in cancer samples.

group	**parameter**	**value**	**description**
log mean of true biological signal μ_g	μ_0	6	mean
	σ_0	1.5	std
log std of true biological signal σ_g	$sMean$	0.65	mean
	$sVar$	0.01	variance
multiplicative noise H_{gi}	ϕ	0.1	std
additive noise E_{gi}	ν	30	mean
	τ	40	std
block correlation ρ	p	0.6	beta dist. parameter 1
	w	5	beta dist. parameter 2
diff.mean	α	10	gamma dist. parameter 1
	β	10	gamma dist. parameter 2
constant	π	0.3	portion of transcriptionally active blocks
	bs	1,5,10,50,100, 250,500,1000	number of genes per block size
	$nGenes$	10000	number of genes in microarray experiment
	$nSamples$	10,25,50,100	number of samples in each condition
	$p.diff$	0.1	portion of differentially expressed genes

Table 1: Parameters used in simulation

5 Appendix

This analysis was performed in the following directory:

```
> getwd()
```

```
[1] "/tmp/RtmpTJ0Tx7/Rbuild10ca10bc296d/Umpire/vignettes"
```

This analysis was performed in the following software environment:

```
> sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version 3.12

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

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

other attached packages:
[1] survival_3.7-0 Umpire_2.0.10  rmarkdown_2.27

loaded via a namespace (and not attached):
 [1] sass_0.4.9         utf8_1.2.4         generics_0.1.3     tidyr_1.3.1
 [5] rstatix_0.7.2      lattice_0.22-6     digest_0.6.36      magrittr_2.0.3
 [9] evaluate_0.24.0    grid_4.4.1         mvtnorm_1.2-5      fastmap_1.2.0
[13] Matrix_1.7-0       jsonlite_1.8.8     backports_1.5.0    mclust_6.1.1
[17] purrr_1.0.2        fansi_1.0.6        mc2d_0.2.1         BimodalIndex_1.1.9
[21] scales_1.3.0       jquerylib_0.1.4    abind_1.4-5        cli_3.6.3
[25] rlang_1.1.4        splines_4.4.1      munsell_0.5.1      cachem_1.1.0
[29] yaml_2.3.9         tools_4.4.1        ggsignif_0.6.4     dplyr_1.1.4
[33] colorspace_2.1-0   ggplot2_3.5.1      ggpubr_0.6.0       broom_1.0.6
[37] buildtools_1.0.0   vctrs_0.6.5        R6_2.5.1           lifecycle_1.0.4
[41] car_3.1-2          oompaBase_3.2.9    cluster_2.1.6      pkgconfig_2.0.3
```

```
[45] pillar_1.9.0       bslib_0.7.0        gtable_0.3.5       glue_1.7.0
[49] highr_0.11         xfun_0.45          tibble_3.2.1       tidyselect_1.2.1
[53] sys_3.4.2          knitr_1.48         htmltools_0.5.8.1  carData_3.0-5
[57] maketools_1.3.0    compiler_4.4.1
```