

# affyContam: structured corruption of CEL-level data

VJ Carey `stvjc` at `channing.harvard.edu`

October 28, 2009

## 1 Introduction

Microarray quality assessment is a major concern of microarray analysts. This package provides some simple approaches to *in silico* creation of quality problems in CEL-level data to help evaluate performance of quality metrics.

## 2 A small dataset

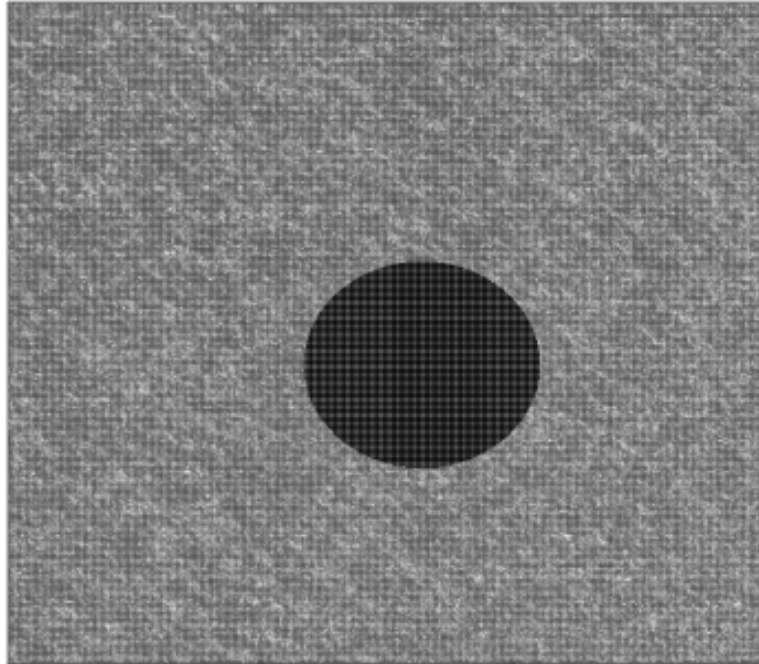
The *affydata* package includes a dataset called Dilution.

```
> library(affydata)
> data(Dilution)
> image(Dilution[, 1])
```

We can introduce a circular defect as follows:

```
> library(affyContam)
> dilc = setCircRegion(Dilution, chip = 1)
> image(dilc[, 1])
```

20A



Note that the extents of the region (which may be rectangular, see `setRectRegion`) can be varied using arguments to the `set*Region` functions. The distribution of intensity values used to reset the raw data can also be specified using the `valgen` parameter.

### 3 Systematic demonstration

The following code shows how one might explore several procedures for sensitivity to artifacts. It runs very slowly so all evaluation has been turned off, but `eval=FALSE` can be reset by interested users.

```
> library(affyMvout)
> library(affy)
> library(SpikeIn)
> data(SpikeIn133)
> library(mdqc)
> library(affyContam)
> library(limma)
```

```

> s12 = SpikeIn133[, 1:12]
> s12rma = rma(s12)
> mads = apply(exprs(s12rma), 1, mad)
> kp = which(mads > quantile(mads, 0.95))
> kppn = featureNames(s12rma)[kp]
> mostmr = c("203508_at", "204563_at", "204513_s_at", "204205_at",
+ "204959_at", "207655_s_at", "204836_at", "205291_at", "209795_at",
+ "207777_s_at", "204912_at", "205569_at", "207160_at", "205692_s_at",
+ "212827_at", "AFFX-LysX-3_at", "AFFX-PheX-3_at", "AFFX-ThrX-3_at")
> fullrun = function(abatch, arma, contFun, filtpn, targpn, chips = 1,
+ ...) {
+   dvec = (1:ncol(exprs(abatch)))
+   des = model.matrix(~dvec)
+   af1 = lmFit(arma[filtpn, ], des, method = "robust", maxit = 300)
+   eaf1 = eBayes(af1)
+   orig.tt = eaf1$t[targpn, 2]
+   cbat = contFun(abatch, chip = chips[1], ...)
+   if (length(chips) > 1) {
+     for (i in 2:(length(chips))) cbat = contFun(cbat, chip = chips[i],
+       ...)
+   }
+   crma = rma(cbat)[filtpn, ]
+   dvec = (1:ncol(exprs(abatch)))
+   des = model.matrix(~dvec)
+   cf1 = lmFit(crma, des, method = "robust", maxit = 300)
+   ecf1 = eBayes(cf1)
+   contam.tt = ecf1$t[targpn, 2]
+   caos = ArrayOutliers(cbat)
+   if (nrow(caos[[1]]) < 1) {
+     warning("no outliers by affyMvout")
+     return(list(aos = caos, md = mdqc(caos[[3]][, 2:10])))
+   }
+   todrop = as.numeric(rownames(caos[[1]]))
+   cbatf = cbat[, -todrop]
+   frma = rma(cbatf)[filtpn, ]
+   dvec = (1:ncol(exprs(abatch)))[-todrop]
+   des = model.matrix(~dvec)
+   f1 = lmFit(frma, des, method = "robust", maxit = 300)
+   ef1 = eBayes(f1)
+   repair.tt = ef1$t[targpn, 2]
+   md = mdqc(caos[[3]][, 2:10], robust = "MCD")
+   list(orig = orig.tt, contam = contam.tt, repair = repair.tt,

```

```
+         md = md, todrop = todrop)
+ }
```

```
> ff = fullrun(s12, s12rma, setCircRegion, kppn, mostmr, chips = 1:2)
```

Here are some contaminator procedures.

First, a modest sized circle with constant low intensity (40)

```
> scr.40 = function(x, chip = 1, center = c(150, 150), rad = 75,
+   vals = 30, valgen = NULL) {
+   cdfname = paste(annotation(x), "cdf", sep = "")
+   require(cdfname, character.only = TRUE, quietly = TRUE)
+   xext = seq(center[1] - rad, center[1] + rad)
+   yext = seq(center[2] - rad, center[2] + rad)
+   badco = expand.grid(xext, yext)
+   badco = badco[(badco[, 1] - center[1])^2 + (badco[, 2] -
+     center[2])^2 < rad^2, ]
+   indsbad = apply(badco, 1, function(x) xy2indices(x[1], x[2],
+     cdf = cdfname))
+   if (is.null(valgen))
+     exprs(x)[indsbad, chip] = vals
+   else exprs(x)[indsbad, chip] = valgen(length(indsbad))
+   x
+ }
```

A similar contamination procedure with constant high intensity (20000):

```
> scr.20k = function(x, chip = 1, center = c(500, 500), rad = 75,
+   vals = 30000, valgen = NULL) {
+   cdfname = paste(annotation(x), "cdf", sep = "")
+   require(cdfname, character.only = TRUE, quietly = TRUE)
+   xext = seq(center[1] - rad, center[1] + rad)
+   yext = seq(center[2] - rad, center[2] + rad)
+   badco = expand.grid(xext, yext)
+   badco = badco[(badco[, 1] - center[1])^2 + (badco[, 2] -
+     center[2])^2 < rad^2, ]
+   indsbad = apply(badco, 1, function(x) xy2indices(x[1], x[2],
+     cdf = cdfname))
+   if (is.null(valgen))
+     exprs(x)[indsbad, chip] = vals
+   else exprs(x)[indsbad, chip] = valgen(length(indsbad))
+   x
+ }
```

Second, a somewhat larger circle with rescaled variance:

```
> incvarCircRegion = function(x, chip = 1, center = c(150, 500),
+   rad = 100, fac = 3) {
+   tmp = fac * getCircRegion(x, chip, center, rad)
+   setCircRegion(x, chip, center, rad, vals = tmp)
+ }
```

Third, a large rectangular region with rescaled variance:

```
> incvarRectRegion = function(x, chip = 1, xinds = 350:700, yinds = 1:700,
+   fac = 3) {
+   tmp = fac * getRectRegion(x, chip, xinds, yinds)
+   setRectRegion(x, chip, xinds, yinds, vals = tmp)
+ }
```

```
> tryout = scr.40(s12)
> tryout = scr.20k(tryout)
> tryout = incvarCircRegion(tryout)
> fin = incvarRectRegion(tryout)
> png(file = "lkcomp.png")
> image(fin[, 1], main = "composite contamination")
> dev.off()
```

```
> d1b = fullrun(s12, s12rma, scr.40, kppn, mostmr, chips = 1)
> save(d1b, file = "d1b.rda")
> d2b = fullrun(s12, s12rma, scr.40, kppn, mostmr, chips = 1:2)
> save(d2b, file = "d2b.rda")
> d3b = fullrun(s12, s12rma, scr.40, kppn, mostmr, chips = c(1:2,
+   11))
> save(d3b, file = "d3b.rda")
> H1b = fullrun(s12, s12rma, scr.20k, kppn, mostmr, chips = 1)
> save(H1b, file = "H1b.rda")
> H2b = fullrun(s12, s12rma, scr.20k, kppn, mostmr, chips = 1:2)
> save(H2b, file = "H2b.rda")
> H3b = fullrun(s12, s12rma, scr.20k, kppn, mostmr, chips = c(1:2,
+   11))
> save(H3b, file = "H3b.rda")
> I1b = fullrun(s12, s12rma, incvarCircRegion, kppn, mostmr, chips = 1)
> save(I1b, file = "I1b.rda")
> I2b = fullrun(s12, s12rma, incvarCircRegion, kppn, mostmr, chips = 1:2)
> save(I2b, file = "I2b.rda")
> I3b = fullrun(s12, s12rma, incvarCircRegion, kppn, mostmr, chips = c(1:2,
+   11))
```

```
> save(I3b, file = "I3b.rda")
> R1b = fullrun(s12, s12rma, incvarRectRegion, kppn, mostmr, chips = 1)
> save(R1b, file = "R1b.rda")
> R2b = fullrun(s12, s12rma, incvarRectRegion, kppn, mostmr, chips = 1:2)
> save(R2b, file = "R2b.rda")
> R3b = fullrun(s12, s12rma, incvarRectRegion, kppn, mostmr, chips = c(1:2,
+ 11))
> save(R3b, file = "R3b.rda")
```