Authentication and copy number analysis of mouse cell lines

John Didion

2014-12-24

John Didion

2014-12-24 This vignette reproduces the analyses presented in the publication "SNP array profiling of mouse cell lines identifies their strains of origin and reveals cross-contamination and widespread aneuploidy" by John P Didion et al (BMC Genomics 2014, 15:847, doi:10.1186/1471-2164-15-847).

Note that, following publication of the article, we identified a control sample that was mislabeled. After correcting that error, our assay selects more markers for inbred (3,593 compared to 3,552) and outbred (1,665 compared to 1,652) samples and removed 7 markers that were in LD (compared to zero). These changes do not materially affect the results of any downstream analyses, therefore we did not publish a correction. However, the data files referenced in the publication (http://dx.doi.org/10.6084/m9.figshare.1185417) have been updated.

Getting started

Developing an assay from scratch requires the following steps:

1. Initialize the database.
2. Import the marker annotations. Marker annotations are specific to a genome build, so you first need to insert the genome information into the database.
3. Import the sample annotations.
4. Import the genotype data.
5. If you wish to create an assay that is based only on markers that are common between multiple arrays, define a new marker set that is the intersection of those arrays.
6. Call with appropriate arguments.

Working directory

First, we choose a directory to contain all input, output and temporary files.

```
wd <- "~/clasp"
if (file.access(wd, 2) < 0) {
    dir.create(wd)
}
setwd(wd)
```

Database

CLASP stores all data in a relational database. CLASP uses SQLite by default. Using a different database engine is not currently supported, but may be in a future version. A database is initialized by loading the schema that is included in the package. Or, if you have an existing database, you can obtain a connection to it.

```
library(clasp)
db.file <- "clasp.db"
db <- init.database(db.file)
```

Inputs

CLASP requires three types of inputs:

1. Genotype data: genotype calls (and also, optionally, raw hybridization intensity value). Developing an assay requires control genotypes from the expected background of the cell lines being tested. To run the assay, obviously you need genotypes from the cell lines you want to test.
2. Annotations: metadata describing the genotypes samples, and describing the markers on the array used to genotype the samples.
3. Reference data: to use the functions that predict contamination and chromosome copy numbers, a large panel of references are needed to determine the normal intensity distribution for each marker.

Developing the necessary reference data sets for new arrays is not supported in the current version of the software, but will be supported in a future version. In the meantime, please contact the author for assistance.

All necessary files required to develop and run assays for the MUGA and MegaMUGA arrays are available on FigShare. The MUGA (Mouse Universal

Genotyping Array) series of arrays are commercially available, and at least one company (Neogen) offers full-service genotyping (the authors do not have a financial relationship with any such companies).

All of CLASP's import functions will recognize whether you are providing a file name or a data frame. If you are expecting an import function to load your data from a file, it must be tab-delimited and contain a header.

Marker annotations

A marker annotation file has five columns:

1. Marker name (character).
2. Chromosome number (integer). CLASP only uses autosomal markers.
3. Physical position of the marker (integer).
4. Genetic position of the marker in centimorgans (numeric).
5. Alleles (character). String listing the two possible genotypes for this marker (e.g. AT).

Alleles are typically coded as nucleotides (A/C/G/T), but they may be coded however you wish.

Sample annotations

A sample annotation file has eight required columns:

1. Sample name (character).
2. Cell line name (character). Empty if this is a non-cell line sample.
3. Genetic background (character). E.g. name of mouse inbred strain or outbred stock.
4. Sample type (inbred, outbred, F1, wild).
5. Taxon (character). E.g "Mus musculus castaneus."
6. Person/lab/institution that provided the sample (character).
7. Description of the sample (character).
8. Whether the sample should be used for creation of synthetic (*in silico*) F1 samples.

In addition, if your samples include F1s for which you also have the parental genotypes, you should include an additional two columns describing the pedigree of the F1:

9. Mother name (character).
10. Father name (character).

Genotype data

Each row of a genotypes file gives the genotype (and, optionally, raw intensity values) for one sample at one marker. It has at least three columns:

1. Sample name (character).
2. Marker name (character).
3. Genotype call (character).

The format of intensity values depends on the array platform. E.g. Illumina arrays have a single intensity associated with each probe of a biallelic marker (X and Y).

For each marker, a genotype can have one of four values: one of the two specified for that marker (in column 5 of the marker annotation table), a "no-call" value (which defaults to "N" but you can specify an alternate value in the call to) or a heterozygous value (any value that is not one of the previous three is treated as a heterozygous call).

Importing data from the Didion et al. paper

We will use the package to download all of the genotype, annotation and reference files from FigShare.

```
library(rfigshare)

# This is the FigShare OAuth key registered for rfigshare.
auth = fs_auth(
    token="xdBjcKOiunwjiovwkfTF2QjGhROeLMw0y0nSCSgvg3YQxdBjcKOiunwjiovwkfTF2Q",
    token_secret="4mdM3pfekNGO16X4hsvZdg")

## Authentication successful

# Download all the files for project ID 1185417. Use a modified version of
# rfigshare's fs_download to only download files that don't already exist
article_id <- 1185417
details <- lapply(article_id, fs_details, mine=FALSE, session=auth,
    show_versions=FALSE, version=NULL)
urls <- unlist(sapply(details, function(output)
    unlist(lapply(output$files, function(f) f$download_url))))
filenames <- unlist(sapply(details, function(output)
    unlist(lapply(output$files, function(f) f$name))))
for (i in 1:length(urls)) {
```

```
    if (!file.exists(filenames[i])) {
        download.file(urls[i], destfile=filenames[i], method="internal")
    }
}
```

The files should now all be in our working directory.

```
list.files()
```

```
##  [1] "clasp.db"                   "dilution_series.RData"
##  [3] "MegaMUGA_genotypes.txt.gz" "MegaMUGA_markers.txt"
##  [5] "MegaMUGA_samples.txt"       "mm.genoCN.RData"
##  [7] "mm.norm.params.RData"       "mouse-cell-lines.Rmd"
##  [9] "MUGA_genotypes.txt.gz"      "MUGA_markers.txt"
## [11] "MUGA_samples.txt"           "muga.genoCN.RData"
## [13] "muga.norm.params.RData"
```

Data import

Genome

All of the marker annotations are based on mouse genome build 37. We create an entry for build 37 in the database.

```
genome.id <- import.genome("Mus musculus", "NCBI/37", db)
```

Markers

We will now define the MUGA and MegaMUGA arrays in the database, and at the same time import the set of marker annotations for each array. When we define an array, we specify the platform (Illumina, in this case). This tells the function to store the intensity values in the "Illumina" table in the database.

```
mm.array <- import.array("MegaMUGA", "Illumina", genome.id, "MegaMUGA_markers.txt", db,
    description="MegaMUGA markers")
knitr::kable(head(mm.array$marker.set$markers))
```

id	name	chromosome	position_bp	position_cm	alleles
1	UNC6	1	3000355	1.4995	TC
2	JAX00000010	1	3125499	1.5620	AG

id	name	chromosome	position_bp	position_cm	alleles
3	JAX00240603	1	3242877	1.6075	CT
4	JAX00240610	1	3256689	1.6111	CT
5	JAX00240613	1	3313481	1.6260	TC
6	JAX00240636	1	3379644	1.6433	CA

```
muga.array <- import.array("MUGA", "Illumina", genome.id, "MUGA_markers.txt", db,
    description="MUGA markers")
knitr::kable(head(muga.array$marker.set$markers))
```

id	name	chromosome	position_bp	position_cm	alleles
3	JAX00240603	1	3242877	1.6075	CT
75100	UNC010001397	1	3326920	1.6295	GT
9	UNC010515443	1	3658709	1.6480	GA
11	UNC010001943	1	3967211	1.6480	CA
16	UNC010515539	1	4420704	1.6480	AG
18	UNC010515556	1	4521110	1.6516	GA

For this assay, we want to be able to combine information from samples genotyped on the two different platforms, so we need to create a marker set that contains only those markers common to both platforms. There should be 6,212 markers in common.

```
merged.marker.set <- intersect.marker.sets(
    c(mm.array$marker.set$set.id, muga.array$marker.set$set.id),
    db, insert=T, name="Common", "Intersection between MUGA and MegaMUGA markers")
print(paste("There are", nrow(merged.marker.set$markers), "markers in common"))
```

```
## [1] "There are 6212 markers in common"
```

Samples

Now we'll import the sample annotations.

```
mm.sample.set <- import.sample.set("MegaMUGA Samples", db,
    "MegaMUGA samples from Didion et al 2014", "MegaMUGA_samples.txt", mm.array$array.id)
knitr::kable(head(mm.sample.set$samples))
```

id	name	cell_line	background	type	taxon	source	descript
1	MM_5073_129P1/ReJ	NA	129P1/ReJ	inbred	Mus (Mus) musculus	JAX	NA
2	MM_5115_129P3/J	NA	129P3/J	inbred	Mus (Mus) musculus	JAX	NA
3	MM_5032_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA
4	MM_5104_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA
5	MM_5083_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA
6	MM_5047_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA

```
muga.sample.set <- import.sample.set("MUGA Samples", db,
    "MUGA samples from Didion et al 2014", "MUGA_samples.txt", muga.array$array.id)
knitr::kable(head(muga.sample.set$samples))
```

id	name	cell_line	background	type	taxon	source
309	MUGA_1628_129P2/OlaHsd	NA	129P2/OlaHsd	inbred	Mus (Mus) musculus	HARLA
310	MUGA_38_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX
311	MUGA_16_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX
312	MUGA_1829_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX
313	MUGA_1916_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX
314	MUGA_155_129S6	NA	129S6/SvEvTac	inbred	Mus (Mus) musculus	F Pardo

For convenience, we can merge the two sample sets together. Since we are creating the sample set from sample IDs, the default return value is the new sample set ID, but we can set to also get back a table containing the full sample information.

```
merged.sample.set <- import.sample.set("Merged", db,
    "Merged MegaMUGA and MUGA samples from Didion et al 2014",
    c(mm.sample.set$samples$id, muga.sample.set$samples$id),
    fetch=TRUE)
print(paste("There are", nrow(merged.sample.set$samples), "samples in the merged set"))
```

```
## [1] "There are 620 samples in the merged set"
```

```
knitr::kable(head(merged.sample.set$samples))
```

id	name	cell_line	background	type	taxon	source	descript
1	MM_5073_129P1/ReJ	NA	129P1/ReJ	inbred	Mus (Mus) musculus	JAX	NA
2	MM_5115_129P3/J	NA	129P3/J	inbred	Mus (Mus) musculus	JAX	NA
3	MM_5032_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA
4	MM_5104_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA

id	name	cell_line	background	type	taxon	source	descript
5	MM_5083_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA
6	MM_5047_129S1/SvImJ	NA	129S1/SvImJ	inbred	Mus (Mus) musculus	JAX	NA

Genotypes

Finally, we'll import the genotype data. The input files have been gzipped to save space, since they can be several gigabytes. The import function will detect if the import file has a ".gz" extension and automatically decompress it.

```
import.genotypes("MegaMUGA_genotypes.txt.gz", db, platform="Illumina",
    marker.set.id=mm.array$marker.set$set.id, sample.set.id=mm.sample.set$set.id)
```

```
## [1] TRUE
```

```
import.genotypes("MUGA_genotypes.txt.gz", db, platform="Illumina",
    marker.set.id=muga.array$marker.set$set.id, sample.set.id=muga.sample.set$set.id)
```

```
## [1] TRUE
```

Assay development

Now that all of the data has been loaded into the database, we can develop the assay we will use to validate cell lines. Here we use the criteria as described in the paper: no HWE filters; pairwise markers with r^2 above 0.25 are in LD, but markers in LD with different SDPs are retained; and consistency checking between replicates. The new assay is inserted into the database.

```
assay <- develop.assay("Merged", merged.marker.set$set.id, db,
    sample.set=merged.sample.set$set.id, insert=TRUE, max.ld.r2=0.25,
    description="Assay for Didion et al 2014 samples using merged data",
      ob.name="MergedOutbred",
    ob.description="Outbred-specific assay for Didion et al 2014 data set")
```

```
## Loading markers...
## Loading samples...
## Loading genotypes...
## Identifying replicate samples...
## Testing replicates for genotype consistency...
## Filtering by call rate...
## Filtering by MAF...
```

```
## Computing diagnostic values...
## Filtering by pairwise LD...
## Computing pairwise sample distances...
## Getting diagnosic values and pairwise distances for final marker set...
## (Re)computing pairwise marker LD on final marker set...
## Inserting assay marker set in database...
## Preparing assay for outbred samples...
## Done!
```

Let's look at a summary of the assay.

```
print(summary(assay))
```

```
##
## Summary of assay Merged:
## ------------------------
## Markers: Inbred = 3586 ; Outbred = 1665
## Effective sample size: Inbred = 156 ; Outbred = 3
## Pairwise alignment score: Inbred = 0.503 ( 0.214 - 1 ) ; Outbred = 0.561 ( 0.203 - 0.952
## Mean marker diagnostic value: Inbred = 0.5
## Mean LD (r^2): 0.155
```

Executing the assay

We developed our assay using the reference samples in our data set. Now we can execute the assay on the cell line samples and see whether their expected and predicted backgrounds match.

```
result <- execute.assay(assay, db, error.rate=0.03)
```

```
## Loading cell line annotations and genotypes...
## Computing pairwise identity of cell lines...
## Imputing synthetic F1 samples...
## Comparing cell lines to control samples...
## Imputing synthetic F1s between outbred samples...
## Comparing cell lines to outbred samples...
## Computing primary match scores
## Computing secondary match scores for 39 cell lines
## Computing PIA values...
## Done!
```

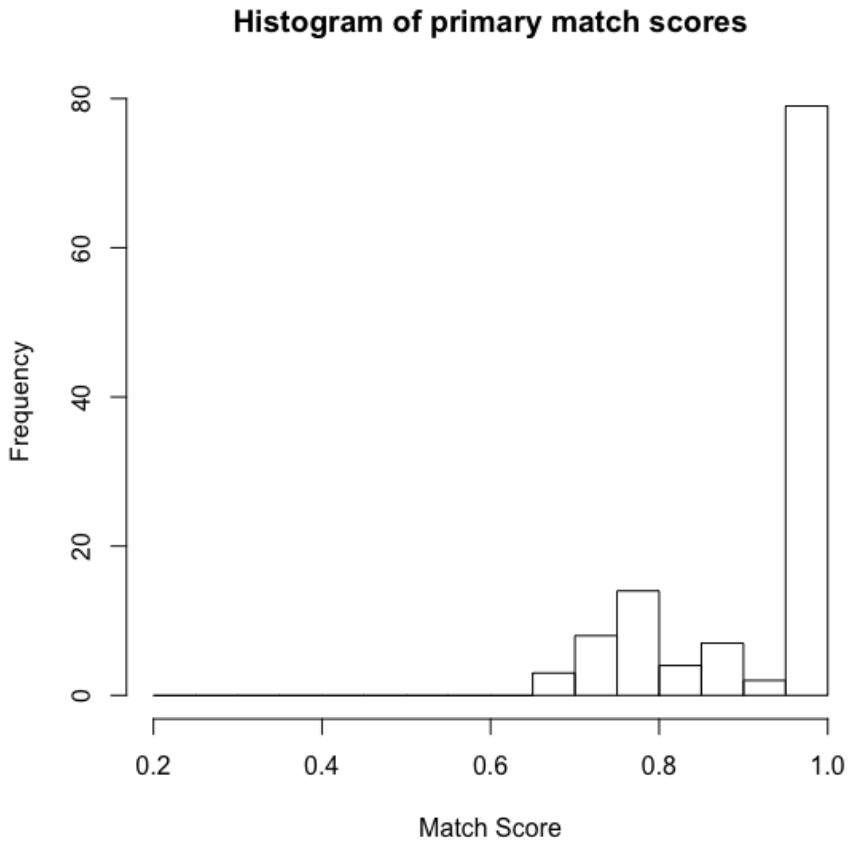
The main thin we'll be interested in is the results summary table, which lists the primary match, secondary match (if any), alignment scores and probability of incorrect assignment (PIA) for all of the cell line samples.

```
temp <- head(result$summary)
# shorten up the column names for better printing
colnames(temp)[4:5] <- c("sec.match", "sec.match.score")
knitr::kable(temp)
```

id	primary.match	primary.match.score	sec.match	sec.match.score
MM_5009_T30	129S2/SvPas	0.9690463	NA	1.8E-17
MM_4985_A962.2	129S2/SvPas	0.9799219	NA	1.8E-17
MM_5005_A972.2	129S2/SvPas	0.9785276	NA	1.8E-17
MM_5069_A987	129S2/SvPas	0.9969325	NA	1.8E-17
MM_5067_E99	129P2/OlaHsd	0.9972114	NA	1.2E-61
MM_5144_E101	129P2/OlaHsd	0.9994423	NA	1.2E-61

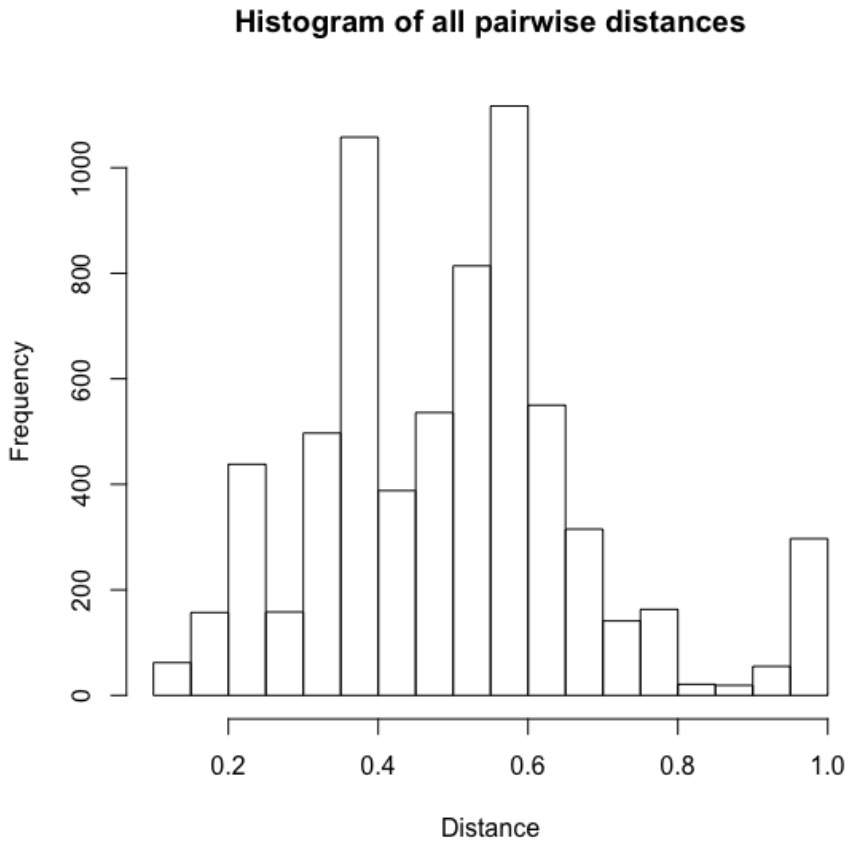
Let's look at a distribution of the primary alignment scores among our samples.

```
hist(result$summary$primary.match.score, breaks=seq(0.2,1,0.05), xlab="Match Score",
     main="Histogram of primary match scores")
```



There is also a distance matrix of the pairwise distances (i.e. number of markers with different genotypes) between all samples. Let's display this as a histogram.

```
hist(result$pairwise.matrix[upper.tri(result$pairwise.matrix)], xlab="Distance",
     main="Histogram of all pairwise distances")
```



Finally, we can look at the detailed results for each sample to see what the second best match was.

```
knitr::kable(result$details[[1]]$primary.matches, row.names=F,
    caption=paste("Best matches for ", result$samples[1,"name"], ":", sep=""))
```

Table 7: Best matches for MM_5009_Tc1:

idx	name	score
3	129S2/SvPas	0.9690463
81	129S1/SvImJ	0.9670943
456	SYNTH_MM_5141_129S2/SvPasxMUGA_1829_129S1/SvImJ	0.9665365
82	129S6/SvEvTac	0.9659788
457	SYNTH_MM_5141_129S2/SvPasxMUGA_155_129S6	0.9656999
4273	SYNTH_MUGA_1829_129S1/SvImJxMUGA_155_129S6	0.9637479

idx	name	score
661	SYNTH_MM_5051_129T2/SvEmsJxMUGA_1829_129S1/SvImJ	0.9592861
5	129T2/SvEmsJ	0.9590073
410	SYNTH_MM_5141_129S2/SvPasxMM_5051_129T2/SvEmsJ	0.9590073
4	129S4/SvJaeJ	0.9567764

Contamination and copy number estimation

Since we imported the raw intensity data along with the genotypes, we can also predict whether the cell line samples are contaminated and what are the copy numbers of their chromosomes.

Normalization of intensities

First we need to perform normalization on the intensity values and convert them into the standard metrics used for intensity-based analyses:

1. B allele frequency (BAF). For a given marker, the BAF of a given sample reflects the degree to which its intensity values match the expected values for samples known to have the B allele. Therefore, samples with BAF ~ 0 are expected to have the A allele, BAF ~ 0.5 heterozygous, and BAF ~ 1.0 B allele. BAFs that deviate from those three modes are indicative of contamination, i.e. the alleles not in the expected ratios. For example, at marker X, if sample 1 is AA and sample 2 is AB, and sample 1 is contaminated by sample 2, the allelic ratio will be somewhere between 1.0 (if the level of contamination is very low) and 0.25 (if the mixture is 50/50).
2. Log R Ratio (LRR). The log-transformed R value. At a given marker, the R value of a given sample is basically the Z-score of it's sum intensity (e.g. X+Y for Illumina arrays) relative to a reference distribution. In mathematical terms, if I is the sum intensity and I0 and d0 are the mean and standard deviation of the sum intensities across a large number of reference samples, then R = (I - I0) / d0.

The function handles transforming raw intensity data into BAF and LRR values. Since we set , it will do this using two different normalization steps, thresholded quantile normalization (tQN) and genomic wave correction. For details on these two methods, see:

1. Staaf J et al. "Normalization of Illumina Infinium whole-genome SNP data improves copy number estimates and allelic intensity ratios." BMC Bioinformatics 2008, 9:409.

2. Diskin SJ et al. “Adjustment of genomic waves in signal intensities from whole-genome SNP genotyping platforms.” Nucleic Acids Research 2008, 36:e126.

```
muga <- grep("MUGA", result$samples$name)
muga.norm.params <- load.objects("muga.norm.params.RData")
muga.metrics <- compute.metrics(muga.array$marker.set$set.id, muga.norm.params, db,
    cell.lines=result$samples[muga,"id"], platform="Illumina", normalize=TRUE)

mm <- grep("MM", result$samples$name)
mm.norm.params <- load.objects("mm.norm.params.RData")
mm.metrics <- compute.metrics(mm.array$marker.set$set.id, mm.norm.params, db,
    cell.lines=result$samples[mm,"id"], platform="Illumina", normalize=TRUE)
```

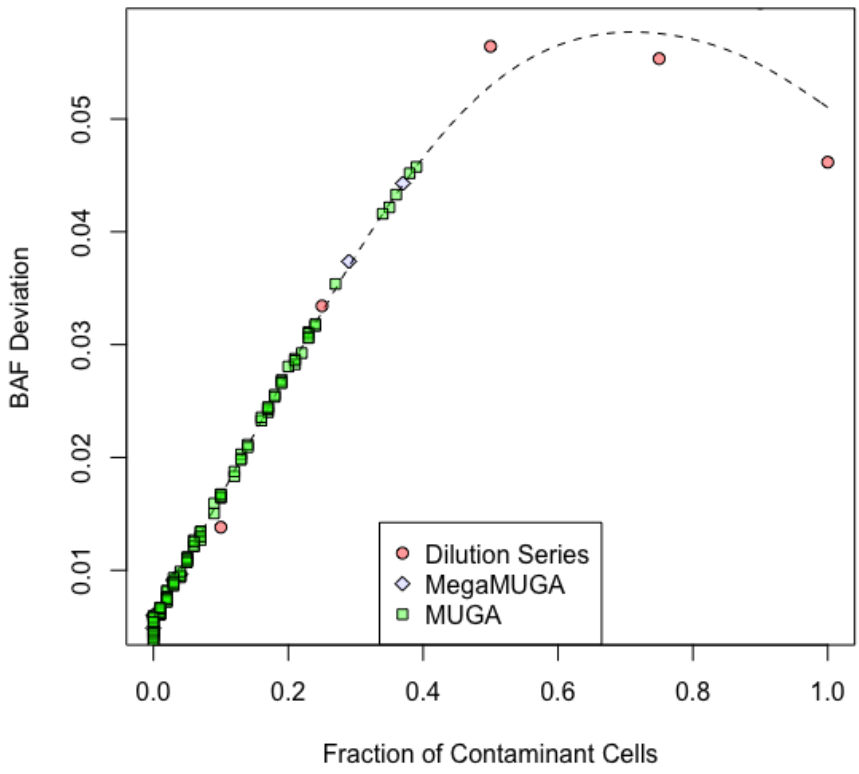
Estimating contamination

We estimate contamination by comparing BAF values of our samples to those derived from genotyping a dilution series between two genetically divergent reference samples. The two reference samples are combined in a series of relative concentrations, and we expect there to be an approximately linear relationship between the level of contaminant and the BAF values at markers where we know the two reference samples have different genotypes. From this we derive a linear model, and then we fit the BAF values from the cell line samples to that model to predict the degree of contamination.

```
dilution.series <- load.objects("dilution_series.RData")
muga.contamination <- estimate.contamination(muga.metrics, dilution.series,
    baf.thresholds=c(0.02, 0.46))
mm.contamination <- estimate.contamination(mm.metrics, dilution.series,
    baf.thresholds=c(0.02, 0.46))
```

First, we can look at how the BAF values from our cell line samples are fit to those from the dilution series.

```
plot(mm.contamination$ypred, col='black', lty=2, type='l',
    xlab="Fraction of Contaminant Cells", ylab="BAF Deviation")
points(mm.contamination$ds$x, mm.contamination$ds$y, bg=rgb.alpha('red', 0.4), pch=21)
points(mm.contamination$cl$x, mm.contamination$cl$y, bg=rgb.alpha('blue', 0.1), pch=23)
points(muga.contamination$cl$x, muga.contamination$cl$y, bg=rgb.alpha('green', 0.4), pch=22)
legend('bottom', c('Dilution Series','MegaMUGA','MUGA'),
    pt.bg=c(rgb.alpha('red', 0.4), rgb.alpha('blue',0.1), rgb.alpha('green',0.4)),
    pch=c(21,23,22))
```



Now let's look at what are predicted to be the most contaminated cell lines.

```
contamination <- as.data.frame(rbind(
    cbind(name=result$samples[muga,"name"], contamination=muga.contamination$contamination),
    cbind(name=result$samples[mm,"name"], contamination=mm.contamination$contamination)))
knitr::kable(head(contamination[order(contamination[,2], decreasing=T),], 10))
```

	name	contamination
573	MUGA_3793_Ehrlich-Lettre Ascites, Strain E	0.39
574	MUGA_3804_Ehrlich-Lettre Ascites, Strain E	0.38

	name	contamination
306	MM_7296_MM Feeder	0.37
550	MUGA_3889_BRUCE4	0.36
569	MUGA_3824_CMT1-2	0.35
616	MUGA_3895_RW4	0.34
305	MM_4972_RW4	0.29
620	MUGA_3892_YAMC	0.27
518	MUGA_724_15-caroli Korat	0.24
615	MUGA_3819_TL1	0.24

Estimating copy number

To estimate chromosomal copy numbers, we use the genoCN algorithm (Sun et al. "Integrated study of copy number states and genotype calls using high-density SNP arrays."" Nucleic Acids Research 2009, 37:5365–5377). There are a number of parameters to this algorithm that need to be selected based on the array platform being used. We have determined the best parameters for MUGA and MegaMUGA.

```
mm.genoCN <- load.objects("mm.genoCN.RData")
muga.genoCN <- load.objects("muga.genoCN.RData")
```

Now we'll run the algorithm. This is fairly quick for MUGA samples, but can take upwards of 2 minutes per sample for MegaMUGA.

```
mm.cn <- estimate.copy.number(result$samples[mm,], mm.metrics, mm.genoCN)
```

```
muga.cn <- estimate.copy.number(result$samples[muga,], muga.metrics, muga.genoCN)
```

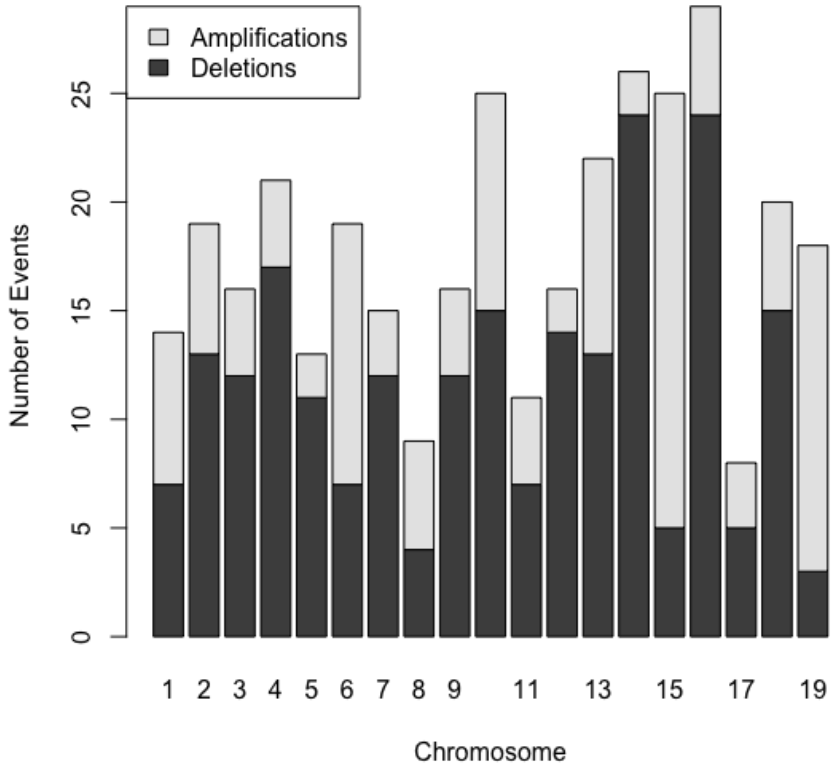
Now let's look at a the mean copy number for each chromosome and each sample. This gives us a gross estimate of the chromosomal copy number, but we would have to look more closely to know whether a whole chromosome has increased or decreased in copy number, or if instead there are substantial sub-chromosomal rearrangements. We determined that mean values < 1.5 were indicative of whole-chromosome loss and values > 2.1 were indicative of chromosome gains.

```
copy.number <- do.call(rbind, lapply(c(mm.cn, muga.cn), function(x) x$mean.cn))
rownames(copy.number) <- c(result$samples[mm,"name"], result$samples[muga,"name"])
knitr::kable(head(round(copy.number, 2)))
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
MM_1507_1	1.97	1.96	1.96	1.93	1.96	1.96	1.98	1.97	2.00	1.97	1.99	1.96	1.98	1.89	1.94	1.97	1.95	1.95	1.99
MM_1498_1	1.98	1.98	1.96	1.96	1.96	1.98	1.99	1.99	1.98	1.98	2.00	1.98	1.99	1.90	1.95	1.97	1.97	1.96	1.99
MM_1508_1	1.98	1.98	1.97	1.97	1.97	1.98	1.99	1.99	1.99	1.99	2.00	1.98	1.99	1.91	1.96	1.98	1.96	1.98	1.99
MM_1506_1	1.96	1.96	1.96	1.96	1.94	1.97	2.00	1.99	2.00	1.96	2.00	1.96	1.99	1.95	1.97	1.96	1.97	1.97	1.99
MM_1507_1	1.96	1.96	1.91	1.91	1.96	1.91	1.96	1.94	1.97	1.91	1.97	1.91	1.96	1.88	1.97	1.87	1.90	1.96	1.98
MM_1514_1	1.94	1.94	1.88	1.87	1.90	1.89	1.93	1.94	1.98	1.91	1.96	1.93	1.95	1.84	1.93	1.80	1.87	1.91	1.96

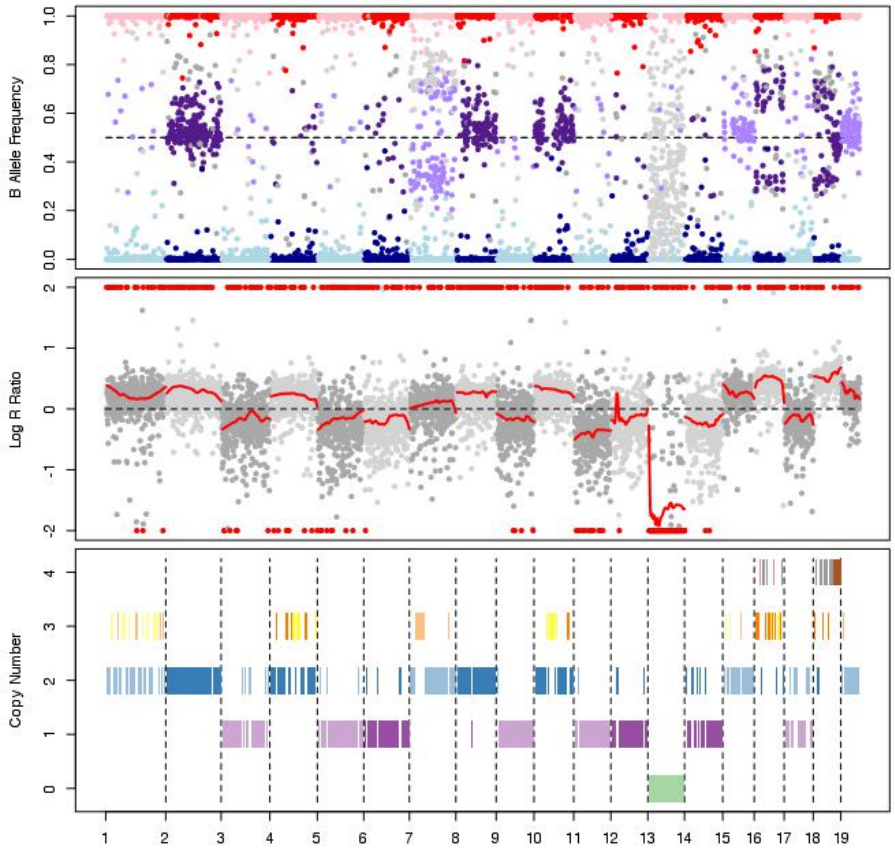
Let's now look at the number of gain and loss events per chromosome, according to the above thresholds.

```
events <- lapply(c(muga.cn, mm.cn), function(x)
    list(loss=which(x$mean.cn < 1.5), gain=which(x$mean.cn > 2.1)))
losses <- table(unlist(lapply(events, function(x) x$loss)))
gains <- table(unlist(lapply(events, function(x) x$gain)))
tab <- rbind(Deletions=losses[as.character(1:19)],
    Amplifications=gains[as.character(1:19)])
tab[is.na(tab)] <- 0
colnames(tab) <- 1:19
barplot(tab, xlab="Chromosome", ylab="Number of Events", legend=T,
    args.legend=list(x="topleft"))
```



It is enlightning to look at the BAF and and LRR values along with the copy number predictions for a sample with predicted aneuploidy. We'll pick an example for which we know the ploidy is highly irregular.

```
copy.number.genoCN.plot(598, muga.array$marker.set$id, muga.metrics[["598"]],
  muga.cn[["598"]], muga.norm.params, db)
```



Wrapping up

Our analysis is now complete. Let's save off all of our results for further examination.

```
save(assay, result, mm.contamination, muga.contamination, mm.cn, muga.cn, file="results.RDat
```

The last step is to close our connection to the database.

```
RSQLite::dbDisconnect(db)
```

```
## [1] TRUE
```