Benchmark of error rate inference from UMI-tagged data and PCR error model

Mikhail Shugay

November 6, 2016

Observed error rate and minor-based error model (MBEM) estimates

Load data from 2 independent experiments with 10 different PCR assays. Data was ran with different quality and UMI size threshold.

```
library(plyr)
library(ggplot2)
library(RColorBrewer)

df <- data.frame()

for (q in c(20, 25, 30)) {
  for (m in c(8, 16, 32)) {
    for (proj in c("73", "82")) {
      for (sample in c("encyclo", "kappa-hf-taq", "phusion", "sd-hs", "snp-detect",
                       "taq-hs", "tersus", "tersus-snp-buff", "truseq", "velox")) {
        .df <- read.table(
            paste("data",
              paste(paste(q, m, paste("polerr", proj, sep=""), sep="_"), sample,
                    "variant.caller.txt", sep = "."),
                    sep ="/"),
            header=T, sep="\t", stringsAsFactors = F)
        .df$q <- q
        .df$m <- m
        .df$proj <- proj
        .df$sample <- sample
        df <- rbind(df, .df)
      }
    }
  }
}

df <- subset(df, count > 0 &
               !grepl("D", mutation) & !grepl("I", mutation) & global.est == 0 &
               coverage > 0)

df$mut.split <- sapply(df$mutation, function(x) strsplit(as.character(x), "[S:>]"))
df$mutation.pos <- as.integer(sapply(df$mut.split, function(x) x[2]))
df$mutation.from <- sapply(df$mut.split, function(x) x[3])
df$mutation.to <- sapply(df$mut.split, function(x) x[4])
df$mut.split  <- NULL
```

We next compare minor-based error model (MBEM) predictions against observed PCR error rate. MBEM computes PCR error rate for n cycles as following

$$\phi^{(n)} = \pi\rho + (1-\pi)\,\rho'$$

where π is the fraction of MIGs with an error occupying a substantial fraction of reads, ρ is the overall fraction of erroneous reads in these MIGs and ρ' is the hidden fraction of errors in molecules for which no error was detected.

For computing π we count MIGs that contain a fraction of errors m/l that can not be explained by sequencing error rate p_{seq} under selected quality threshold Q alone, that is

$$F_{binom}\left(M \geq m | l, p_{seq}\right) < p_0$$

$$p_{seq} = \frac{1}{3}10^{-Q/10}$$

where l is the number of reads in MIG (MIG size) and m is the count of a given minor variant. Note that we also compute false-discovery rate FDR under the specified P-value threshold p_0 and substitute π with $\pi * (1 - FDR)$.

The hidden fraction of errors is estimated from Poisson distribution as $\pi = 1 - \exp(-\rho'\hat{l})$ where $\hat{l}$ is the geometric mean of MIG sizes. Finally,

$$\phi^{(n)} = \pi\rho - \frac{(1-\pi)\log(1-\pi)}{\hat{l}}$$

Error rate per PCR cycle is computed as

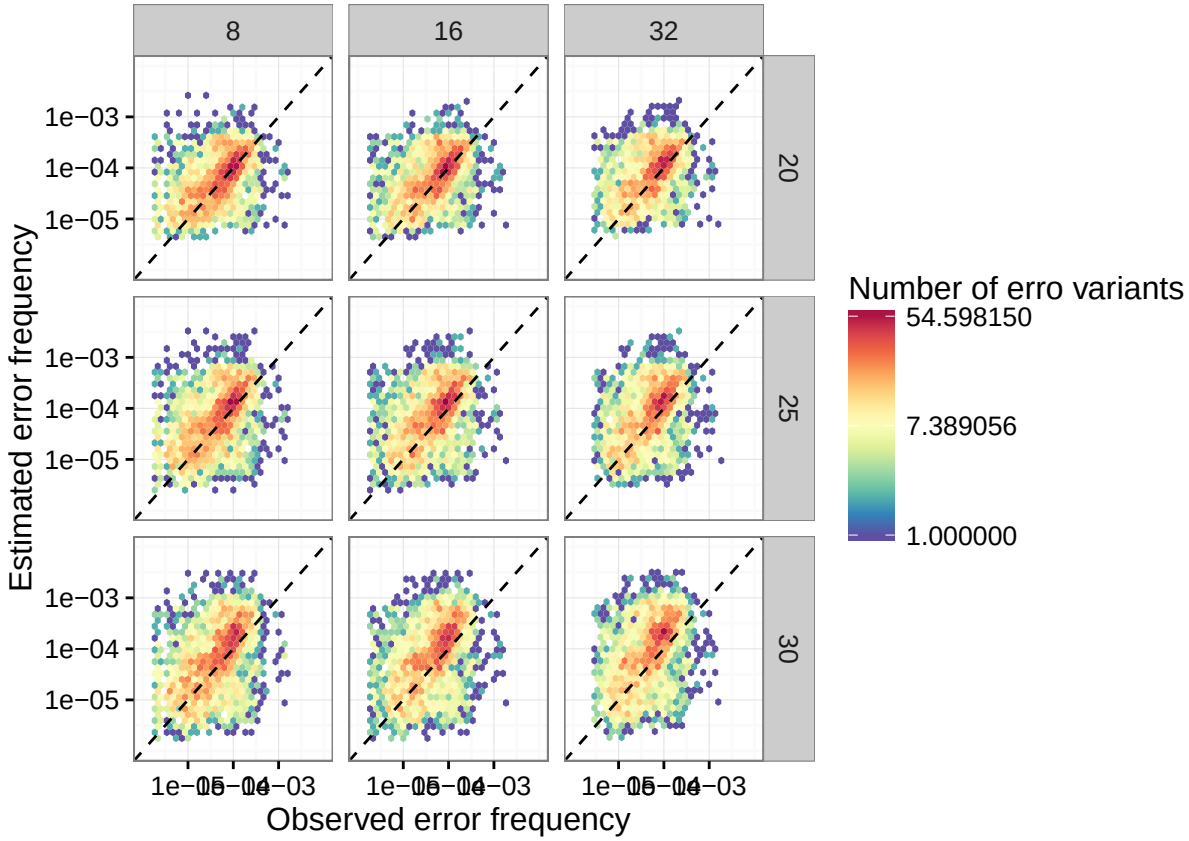
$$\phi = \frac{\phi^{(n)}(1+\lambda)}{n\lambda}$$

where $1 + \lambda$ is PCR efficiency.

In order to apply this model to datasets where only errors introduced at first PCR cycle that become dominant due to PRC stochastics can become major variants, one need to multiply ϕ by error propagation probability $P_{prop} = (1-\lambda)\,\lambda^2$.

Note that this is not the case for this dataset: same polymerase was used during linear PCR to attach UMI and at subsequent PCR cycles. Thus per cycle error rate computed from errors within MIGs should nicely reflect observed error rate.

```
rf <- colorRampPalette(rev(brewer.pal(11, 'Spectral')))
r <- rf(32)

ggplot(df, aes(x=freq, y=error.rate)) +
  stat_binhex() +
  geom_abline(intercept = 0, slope=1, linetype="dashed") +
  scale_x_log10("Observed error frequency", limits=c(1e-6, 1e-2), breaks=c(1e-5,1e-4,1e-3)) +
  scale_y_log10("Estimated error frequency", limits=c(1e-6, 1e-2), breaks=c(1e-5,1e-4,1e-3)) +
  facet_grid(q~m) +
  scale_fill_gradientn("Number of erro variants", colors=r, trans="log") +
  theme_bw()
```




```
a <- aov(log(freq) ~ I(log(error.rate)) + q + m + proj, df)
summary(a)
```

```
##                        Df Sum Sq Mean Sq  F value   Pr(>F)
## I(log(error.rate))      1   7606    7606 5769.797  < 2e-16 ***
## q                       1     50      50   37.988 7.21e-10 ***
## m                       1     33      33   25.189 5.23e-07 ***
## proj                    1      1       1    0.776    0.378
## Residuals           30074  39643       1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
for (mm in unique(df$m)) {
  for (qq in unique(df$q)) {
    print(paste("m =", mm , "q =", qq, with(subset(df, q == qq & m == mm),
                                           cor(freq, error.rate, method = "spearman"))))
  }
}
```

```
## [1] "m = 8 q = 20 0.439187907278551"
## [1] "m = 8 q = 25 0.440171986104134"
## [1] "m = 8 q = 30 0.437514996064882"
## [1] "m = 16 q = 20 0.419952825360284"
## [1] "m = 16 q = 25 0.416393783658744"
## [1] "m = 16 q = 30 0.420338819999655"
## [1] "m = 32 q = 20 0.38017513895925"
## [1] "m = 32 q = 25 0.399270957195644"
## [1] "m = 32 q = 30 0.40434020987666"
```

Choosing statistical model for PCR error calling

Log-linear model for error rate prediction

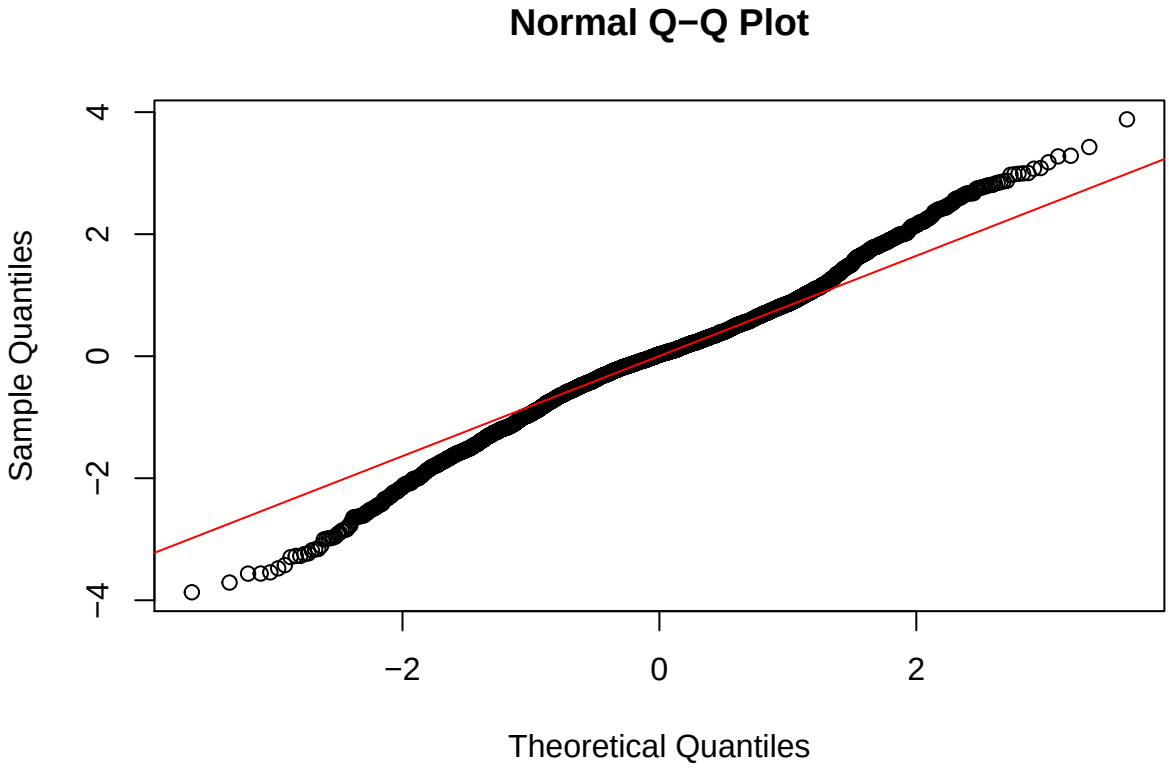
We first perform a fitting of observed error rate under log transformation.

- We get a good fit for log frequencies with slope ~ 1.
- Residuals are normally distributed
- Note that natural logarithm is used here so that analytic calculations we'll perform further will.

```
df.1 <- subset(df, q == 25 & m == 8)

fit <- lm(log(freq) ~ I(log(error.rate)) - 1, data = df.1)
summary(fit)
```

```
##
## Call:
## lm(formula = log(freq) ~ I(log(error.rate)) - 1, data = df.1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8169 -0.9201 -0.0979  0.7079  5.5450
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## I(log(error.rate)) 1.034687   0.002479   417.4   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.476 on 3666 degrees of freedom
## Multiple R-squared:  0.9794, Adjusted R-squared:  0.9794
## F-statistic: 1.742e+05 on 1 and 3666 DF,  p-value: < 2.2e-16
```

```
# Note that there is a very small standard error for slope which
# is close to 1. So we will only with noise terms
# that we'll show to be normally distributed

# Parameters of residual distribution

df.1$res <- log(df.1$freq) - log(df.1$error.rate)

mu.res <- mean(df.1$res)
sd.res <- sd(df.1$res)

# Note that as we used regression-through-origin (RTO),
# the residuals will have non-zero mean and can be correlated
# with predictor

# Residuals are distributed normally

qqnorm(scale(df.1$res))
qqline(scale(df.1$res), distribution = function(p) qnorm(p, mean = 0, sd = 1),
       col = 2)
```




```
shapiro.test(df.1$res)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  df.1$res
## W = 0.98877, p-value < 2.2e-16
```

So far we have shown that

$$\log f_i = \log \phi_i + \epsilon_i$$
$$\epsilon_i \sim \mathcal{N}\left(\mu_\epsilon, \sigma_\epsilon^2\right)$$

where f_i is the observed rate of a given error, ϕ_i is its MBEM estimate and ϵ_i are normally distributed noise terms. Thus,

$$f \sim \mathcal{LN}\left(\log \phi + \mu_\epsilon, \sigma_\epsilon^2\right)$$

In our setting the number of observed MIGs n for each error should be nicely described by Poisson distribution as the coverage $N > 1000$ and error rates are $f \sim 10^{-5}$. As we indirectly estimate our error rates the probability distribution of n is computed as follows,

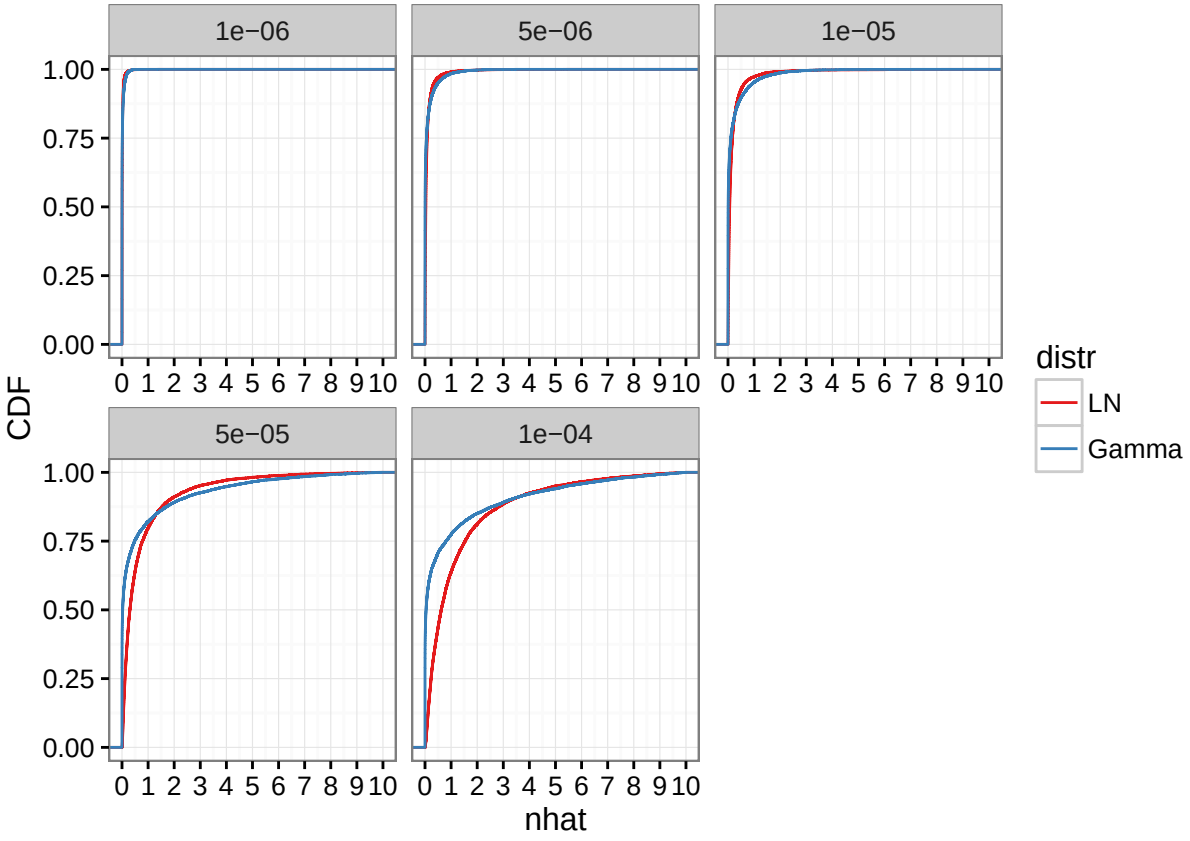
$$\hat{n} = fN \sim \mathcal{LN}\left(\log \phi N + \mu_\epsilon, \sigma_\epsilon^2\right)$$
$$P(n|\phi, N, \mu_\epsilon, \sigma_\epsilon) = \int P_{poisson}(n|\hat{n}) P(\hat{n}|\phi, N, \mu_\epsilon, \sigma_\epsilon) d\hat{n}$$

Comparing Gamma and Log-Normal distributions

As Gamma but not Log-Normal distribution is the conjugate prior for the Poisson distribution, we'll next show that for our range of parameters these distributions are similar when the mean and variance are fixed to the same value, which will allow us to simplify all calculation by switching to the former.

```
N <- 10000
phi <- c(1e-6, 5e-6, 1e-5, 5e-5, 1e-4)
phi.mu <- mu.res + log(phi) + log(N)
phi.sigma <- rep(sd.res, length(phi))

phi.mean <- exp(phi.mu + phi.sigma^2 / 2)
phi.var <- (exp(phi.sigma^2) - 1) * exp(2 * phi.mu + phi.sigma^2)

phi.alpha <- phi.mean * phi.mean / phi.var
phi.beta <- phi.mean / phi.var

df.distr.compare <- data.frame()

for (i in 1:length(phi)) {
  df.distr.compare <- rbind(df.distr.compare,
                            data.frame(phi = phi[i], distr="LN", value=rlnorm(10000, phi.mu[i], phi.sig
  df.distr.compare <- rbind(df.distr.compare,
                            data.frame(phi = phi[i], distr="Gamma", value=rgamma(10000, phi.alpha[i], p
}

ggplot(df.distr.compare, aes(x=value, color=distr)) + stat_ecdf() +
  facet_wrap(~phi, scales = "free_x") +
  ylab("CDF") +
  scale_x_continuous("nhat", limits=c(0, 10), breaks=0:10) +
  scale_color_brewer(palette = "Set1") +
  theme_bw()
```

```
## Warning: Removed 1026 rows containing non-finite values (stat_ecdf).
```



We therefore assume that

$$\hat{n} \sim Gamma\,(\alpha, \beta)$$
$$\alpha = (e^{\sigma_\epsilon^2} - 1)^{-1}$$
$$\beta = \alpha(\phi N)^{-1} e^{-\mu_\epsilon - \sigma_\epsilon^2/2}$$

so

$$n \sim NegBinom(r, p)$$
$$r = (e^{\sigma_\epsilon^2} - 1)^{-1}$$
$$p = (1 + r(\phi N)^{-1} e^{-\mu_\epsilon - \sigma_\epsilon^2/2})^{-1}$$

Computing Q-scores for errors using a composite Log Normal + Beta Binomial model and comparing them with real Q-scores.

Beta Binomial model parameters are estimated using mean and sd of fitting performed in previous section

Beta Binomial P-value is computed for variant **counts** in they are <= 5, Log Normal model for **frequencies** is computed otherwise

Q score distributions are compared using QQ plot

The threshold of 5 chose empirically, decreasing and increasing them will under- and over-estimate Q-scores respectively

```r
# Neg-binomial parameters

df.1$r <- 1 / (exp(sd.res^2) - 1)
df.1$p <- 1 / (1 + df.1$r * exp(-mu.res - exp(sd.res^2) / 2) / df.1$coverage / df.1$error.rate)

# Model P-values

df.1$pval <- with(df.1, 1.0 - pnbinom(count, size = r, prob = 1 - p) +
                    0.5 * dnbinom(count, size = r, prob = 1 - p))


ggplot(df.1, aes(x=pval)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```