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

Mikhail Shugay

November 6, 2016

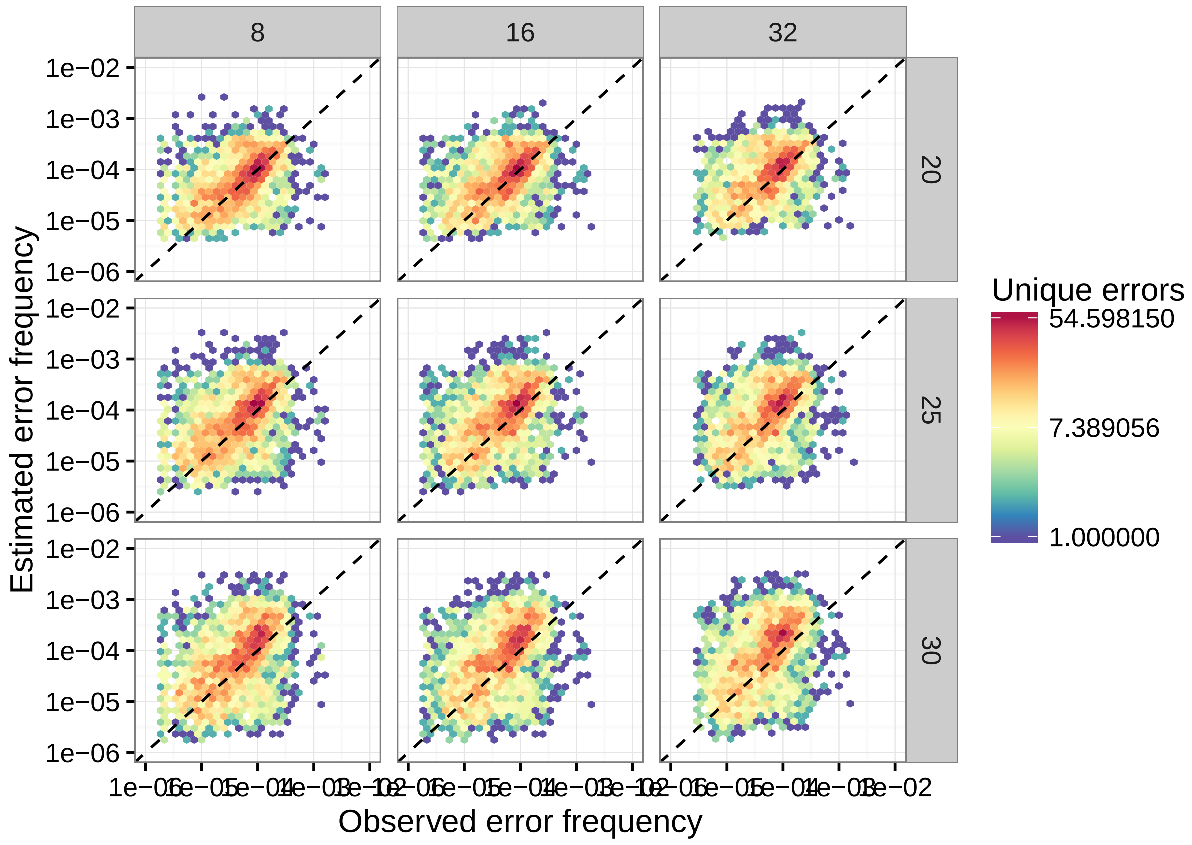
Load data from 2 independent experiments with 10 different PCR assays. Data was ran with different quality and UMI size threshold:

```r
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
```

Minor-based error model fitting for observed PCR error rate:

> Note that 0-order model is used here. 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-6, 1e-5, 1e-4, 1e-3, 1e-2)) +
  scale_y_log10("Estimated error frequency", limits=c(1e-6, 1e-2),
                breaks=c(1e-6, 1e-5, 1e-4, 1e-3, 1e-2)) +
  facet_grid(q~m) +
  scale_fill_gradientn("Unique errors", colors=r, trans="log") +
  theme_bw()
```



```
a <- aov(log10(freq) ~ I(log10(error.rate)) + q + m + proj, df)
summary(a)
```

```
##                         Df Sum Sq Mean Sq  F value   Pr(>F)
## I(log10(error.rate))     1   1435  1434.5 5769.797  < 2e-16 ***
## q                        1      9     9.4   37.988 7.21e-10 ***
## m                        1      6     6.3   25.189 5.23e-07 ***
## proj                     1      0     0.2    0.776    0.378
## Residuals            30074   7477     0.2
## ---
## 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"
```

Detailed fitting of observed error rate under log transformation.

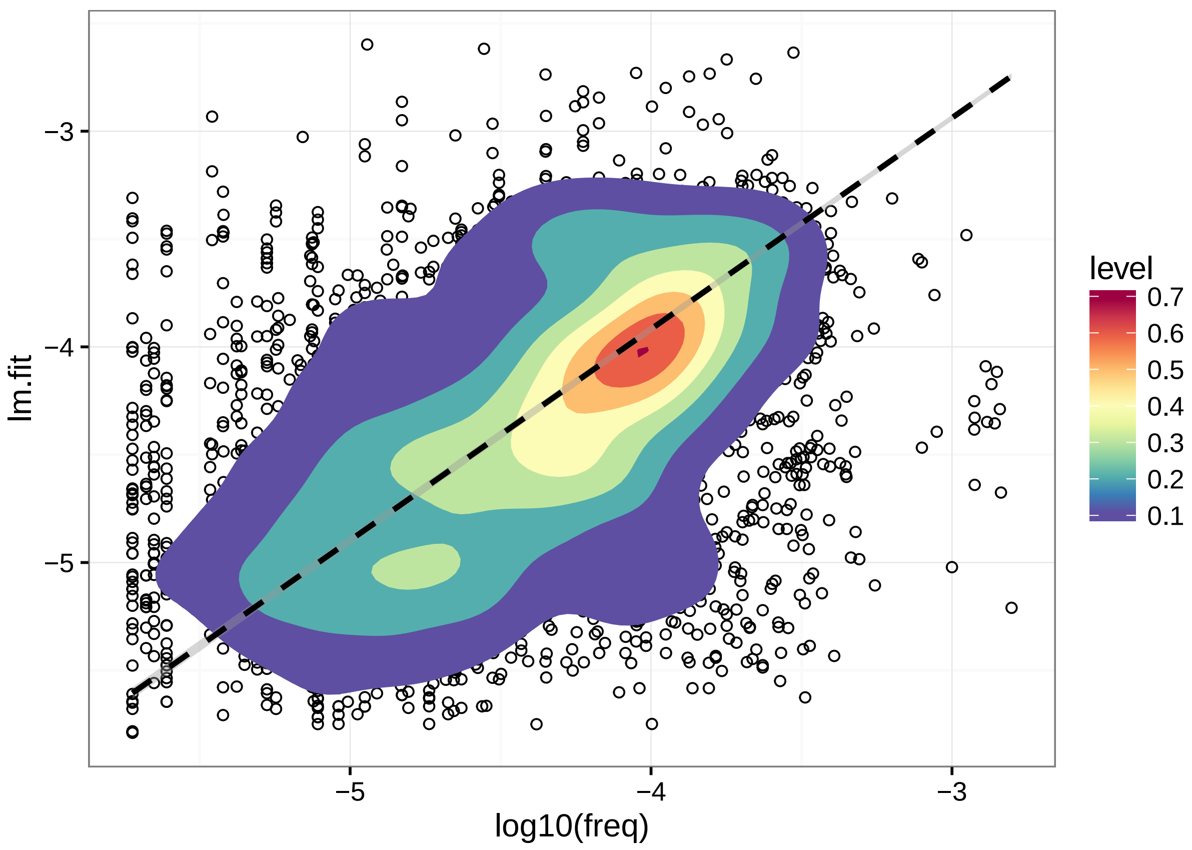
We get a good fit for log frequencies with slope ~ 1.

Residuals are not correlated with error rate and are normally distributed

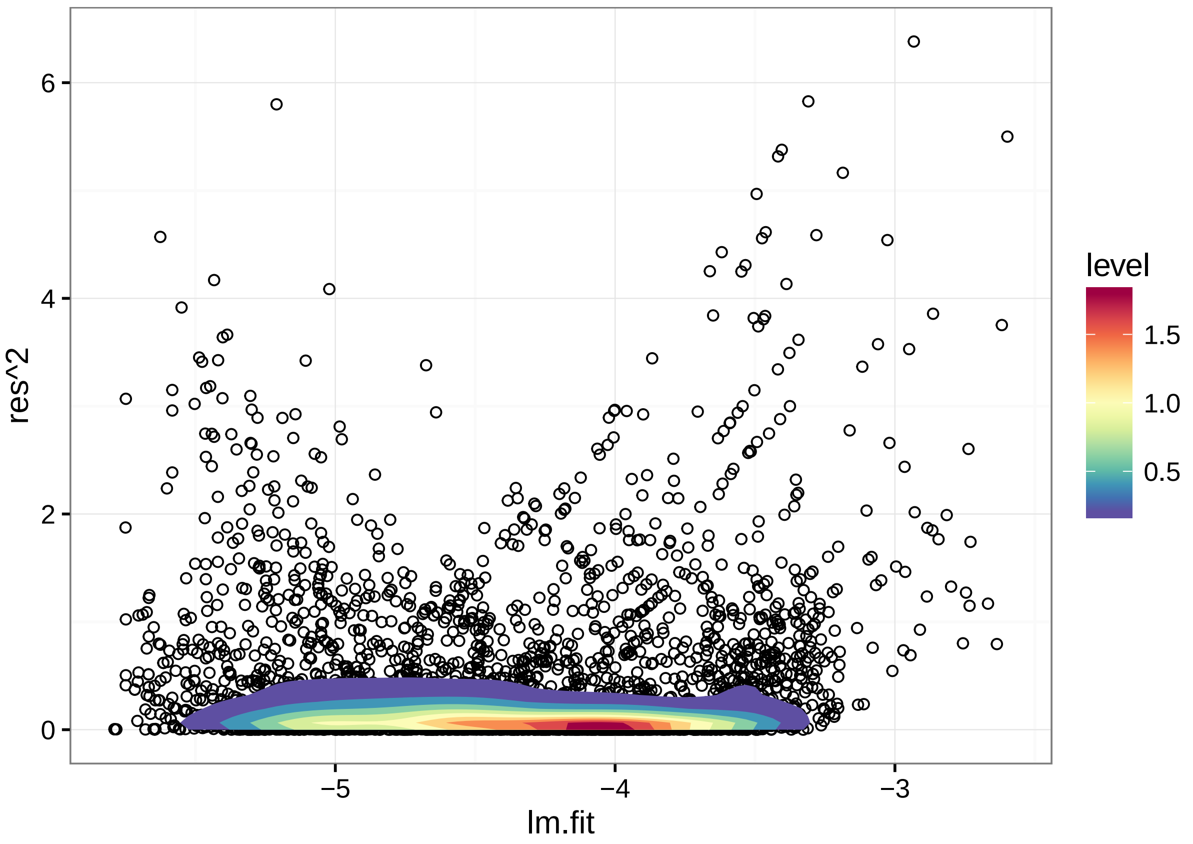
```
df.1 <- subset(df, q == 25 & m == 8)

fit <- lm(log10(freq) ~ I(log10(error.rate)) - 1, data = df.1)
summary(fit)
```

```
##
## Call:
## lm(formula = log10(freq) ~ I(log10(error.rate)) - 1, data = df.1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.52623 -0.39962 -0.04251  0.30744  2.40817
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## I(log10(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: 0.6409 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
```

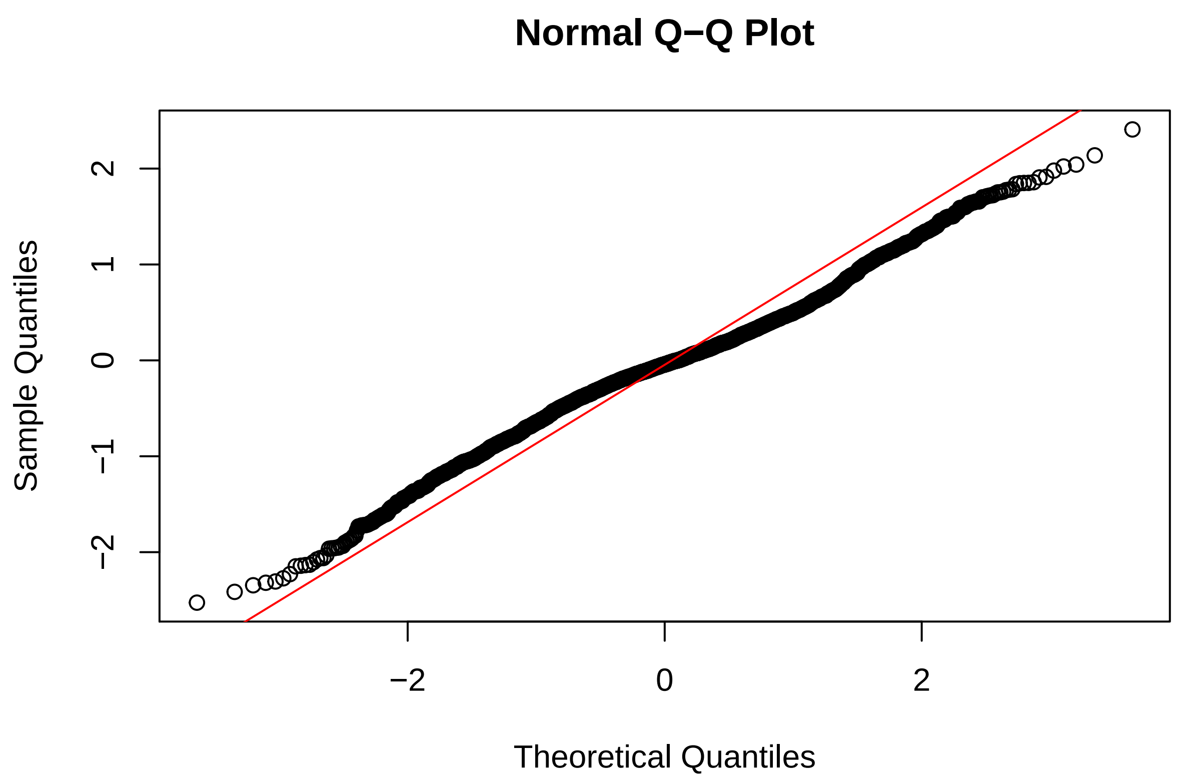
```
df.1$lm.fit <- fitted.values(fit)
df.1$res <- residuals(fit)

ggplot(df.1, aes(x=log10(freq), y=lm.fit)) +
  geom_point(shape=21) +
  stat_density2d(aes(fill=..level..), geom="polygon") +
  geom_smooth(method="lm", formula=y ~ x - 1, linetype="dashed", color="black") +
  scale_fill_gradientn(colors=r) + theme_bw()
```



```
# No correlation between residual sd and MBEM error rate

ggplot(df.1, aes(x=lm.fit, y=res^2)) +
  geom_point(shape=21) +
  stat_density2d(aes(fill=..level..), geom="polygon") +
  scale_fill_gradientn(colors=r) + theme_bw()
```



```r
# Residuals are distributed normally

mdl.res.sd <- sd(df.1$res)

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



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

```
##
##  Shapiro-Wilk normality test
##
## data:  df.1$res
## W = 0.98917, p-value = 3.907e-16
```

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

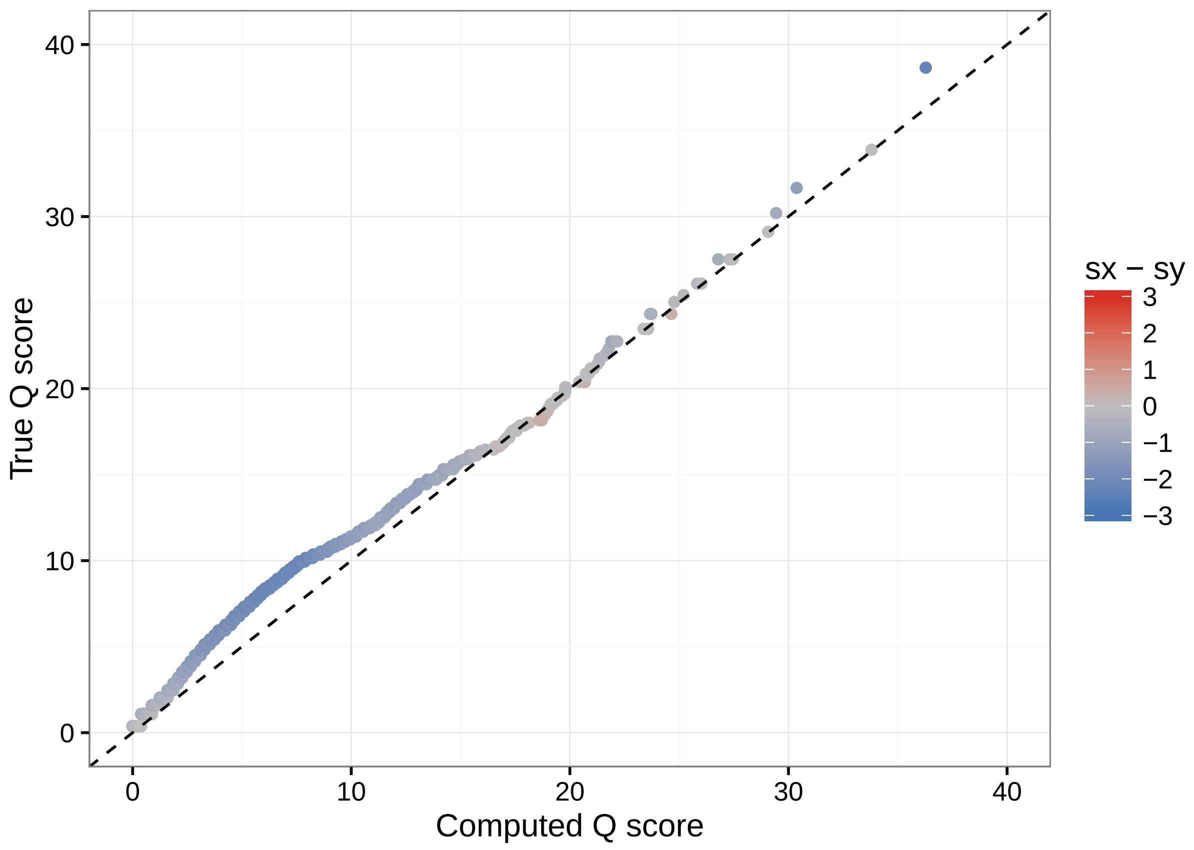
> Beta Binomial model parameters are estimated using mean and sd of fitting performed in previous section
>
> Beta Binomial P-value is computed for variant **counts** in they are <= 5, Log Normal model for **frequencies** is computed otherwise
>
> Q score distributions are compared using QQ plot
>
> The threshold of 5 chose empirically, decreasing and increasing them will under- and over-estimate Q-scores respectively

```r
# source("https://bioconductor.org/biocLite.R")
# biocLite("Biobase")
# install.packages("TailRank", repos="http://R-Forge.R-project.org")
library(TailRank)
```

```
## Loading required package: oompaBase
```

```r
# Beta-binomial parameters

er.m <- df.1$error.rate * exp(mdl.res.sd^2 / 2)
er.v <- df.1$error.rate ^ 2 * exp(mdl.res.sd^2) * (exp(mdl.res.sd^2) - 1)
```

```r
df.1$alpha <- er.m * (er.m * (1 - er.m) / er.v - 1)
df.1$beta <- (1 - er.m) * (er.m * (1 - er.m) / er.v - 1)

# Model P-values

df.1$pval <- with(df.1, mapply(function(x, y, a, b, z)
                ifelse(x <= 5, 1.0 - pbb(x, N = y, u = a, v = b) +
                         0.5 * dbb(x, N = y, u = a, v = b),
                       1.0 - pnorm(log10(x / y), mean = log10(z), sd = mdl.res.sd)),
                count, coverage, alpha, beta, error.rate))

# Real P-values

df.count.summary <- ddply(df.1, .(count), summarize, weight = length(count))

df.1$pval.true <- with(df.count.summary,
                     sapply(df.1$count,
                       function(x)
                         sum(ifelse(x > count,
                                    0, ifelse(x == count, 0.5 * weight, weight)))) /
                       sum(weight))

# For qqplot in ggplot

sx <- sort(-10*log10(df.1$pval))
sy <- sort(-10*log10(df.1$pval.true))
lenx <- length(sx)
leny <- length(sy)
if (leny < lenx)sx <- approx(1L:lenx, sx, n = leny)$y
if (leny > lenx)sy <- approx(1L:leny, sy, n = lenx)$y

ggplot(data.frame(q=sx, q_true=sy), aes(x=q, y=q_true, color=sx-sy)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "black", linetype="dashed") +
  scale_x_continuous("Computed Q score", limits=c(0, 40)) +
  scale_y_continuous("True Q score", limits=c(0, 40)) +
  scale_color_gradient2(low = "#4575b4", mid="grey", high="#d73027", midpoint=0,
                        limits=c(-3,3)) +
  theme_bw()
```



True Q score
Computed Q score
0
10
20
30
40
sx − sy
3
2
1
0
−1
−2
−3