SF-seq

Jake VanCampen

Wed Oct 4 21:47:43 2017

SF-seq read quality score distributions

I worked with the following sets of paired-end reads representing two splicing factor treatments from an mRNA alternative splicing experiment: 2_2B_control, and 29_4E_fox.

```
2_2B_control_S2_L008_R1_001.fastq.gz
2_2B_control_S2_L008_R2_001.fastq.gz
29_4E_fox_S21_L008_R1_001.fastq.gz
29_4E_fox_S21_L008_R2_001.fastq.gz
```

To determine quality score distributions for each read the program FastQC was loaded on the high performace computer along with it's dependencies: `ml easybuild intel/2017a FastQC`

In a working directory called SF-seq, a new directory was created for the output of FastQC. FastQC was then run using the following options on all four files.

```
$ fastqc --noextract -o fastqc_out -t 4 \
2_2B_control_S2_L008_R1_001.fastq.gz 2_2B_control_S2_L008_R2_001.fastq.gz \
29_4E_fox_S21_L008_R1_001.fastq.gz 29_4E_fox_S21_L008_R2_001.fastq.gz
```

Results

Quality score distributions for the forward and reverse reads as well as per-base N content of both treatments were extracted from the FastQC output and compared to results from my quality score distribution plots.

The quality score distributions for the treatment `29_4E_fox_S21` show high average quality scores (approching 40) for read one with lower quality scores at the begining and slowly tapering towards the end of the read (Figure 1). Read two shows a lower quality score across all the bases, not exceeding 38, with lowest quality at the start and end of the read (Figures 2). The percent N-content across the bases for reads one and two show the highest percent N-content at the first base position for both reads, and undetectable N-content over the remaining base positions (Figures 3, 4). The detectable N-content at the first base position explains the low quality start in the quality score distributuions because an 'N' at that position will decrease the average quality score at that base position. The plots for the treatment `2_2B_control` show a similar pattern where read two shows a lower average quality score than read one, and the low quality score at the begining of the read is reflected in the percent N-content of each read at base position one (Figure 5-8).

I then ran my quality score distribution script `qscore_dist2.py` on the files using the high performance computer:

```
# load the necessary modules
ml zlib/1.2.11 python3/3.6.1

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/2_2B_control_S2_L008_R1_001.fastq

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/2_2B_control_S2_L008_R2_001.fastq

# run qscore_dist2.py on all files
```

```
time ./qscore_dist2.py -f ~/SF-seq/data/29_4E_fox_S21_L008_R1_001.fastq

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/29_4E_fox_S21_L008_R2_001.fastq
```

The error file from the script that ran `qscore_dist2.py` held the time of each run, showing that each run took less than 20 minutes, while the total runtime was just over one hour. This is slower than the total runtime of FastQC (2m 46s). FastQC analysis included the multi-threading option -t 4 which ran the analysis for each file on a separate thread, likely decreasing runtime compared to my script. Additionally, FastQC is written in Java and may have more efficient data structures than my script written in Python, potentially resulting in a faster runtime for this analysis.

```
$ cat QD.err

The following have been reloaded with a version change:
  1) zlib/1.2.8 => zlib/1.2.11



real    18m24.967s
user    18m21.792s
sys 0m1.324s

real    18m28.796s
user    18m26.856s
sys 0m1.334s

real    15m24.851s
user    15m23.165s
sys 0m1.163s

real    15m16.293s
user    15m14.693s
sys 0m1.070s
```

The quality score distribution plots from my script are shown in Figures 9-12. These distributuions show similar patterns as the results from the FastQC output. Read two for both treatments shows a lower average quality score for more base positions than in read one, and the quality score drop-off toward the begining and end of the read is similar to the results found in the FastQC output as expected.

Adaptor trimming comparison

The sequences must be filtered to trim known adapter sequences from each read so that only the insert sequences remain. With the common occurance of PCR primer contamination, PCR primers are also important to remove. There are a variety of programs that accomplish these tasks, some loaded on Talapas include: `process_shortreads` (part of the Stacks pipeline), `Trimmomatic`, and `cutadapt`. The programs are notably different in a few ways. First, Trimmomatic is written in Java, cutadapt is a python package available through the Python Package Index, and Process shortreads is written in C++ and Perl. There are numerous differences for dealing with paired end reads using the mentioned programs. Trimmomatic takes forward and reverse reads, and outputs four files: both paired and unpaired forward and reverse reads. Paired forward and reverse reads have to be complementary, while unpaired forward and reverse read files include reads where either the forward or reverse did not meet the quality threshold. Process_shortread by default discards low quality reads unless their retention is specified as an option. Cutadapt by defualt must take pairs of reads, and will throw out the pair if quality is low, outputting adapter-trimmed read pairs to two forward and reverse read files. Demultiplexing is not supported in cutadapt where it is supported in Trimmomatic and Process_shortreads. This analysis was carried out using process_shortreads because it contains all functionality necessary for this analysis with concise runtime options. The following scripts were run to properly trim adapter sequences from sequenced reads:

```
process_shortreads -1 /home/jvancamp/SF-seq/2_2B_control_S2_L008_R1_001.fastq.gz
-2 /home/jvancamp/SF-seq/2_2B_control_S2_L008_R2_001.fastq.gz -o /home/jvancamp/
SF-seq/2B_control
--adapter_1 AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT


process_shortreads -1 /home/jvancamp/SF-seq/29_4E_fox_S21_L008_R1_001.fastq.gz
-2 /home/jvancamp/SF-seq/29_4E_fox_S21_L008_R2_001.fastq.gz -o /home/jvancamp/
SF-seq/4E_fox
--adapter_1 AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT
```

The results for each treatment are shown below:

```
File 29_4E_fox_S21_L008_R1_001.fastq.gz
Total Sequences 9654866
Reads containing adapter sequence   37408
Ambiguous Barcodes  0
Low Quality 0
Trimmed Reads   17749
Orphaned Paired-ends    0
Retained Reads  9635207

File 2_2B_control_S2_L008_R1_001.fastq.gz
Total Sequences 11661330
Reads containing adapter sequence   48613
Ambiguous Barcodes  0
Low Quality 0
Trimmed Reads   26409
Orphaned Paired-ends    0
Retained Reads  11639126
```

These results show 0.18% of `29_4E_fox_S21` reads were trimmed, and 0.23% of `2_2B_control_S2` reads were trimmed. A frequency table of read lengths for each treatmnet were generated using a similar manner of the following Unix command:

```
$ zcat 29_4E_fox_S21_L008_R1_001.1.fq.gz | awk '{if (NR%4==2) print length}' |\
sort | uniq -c > 4E_R1_lengths.txt
```

The files contained leading spaces which were removed with more Unix:

```
→for file in *.txt; do cat $file | awk '{$1=$1};1' > tmp && mv tmp $file; done
```

Read Length Distributions

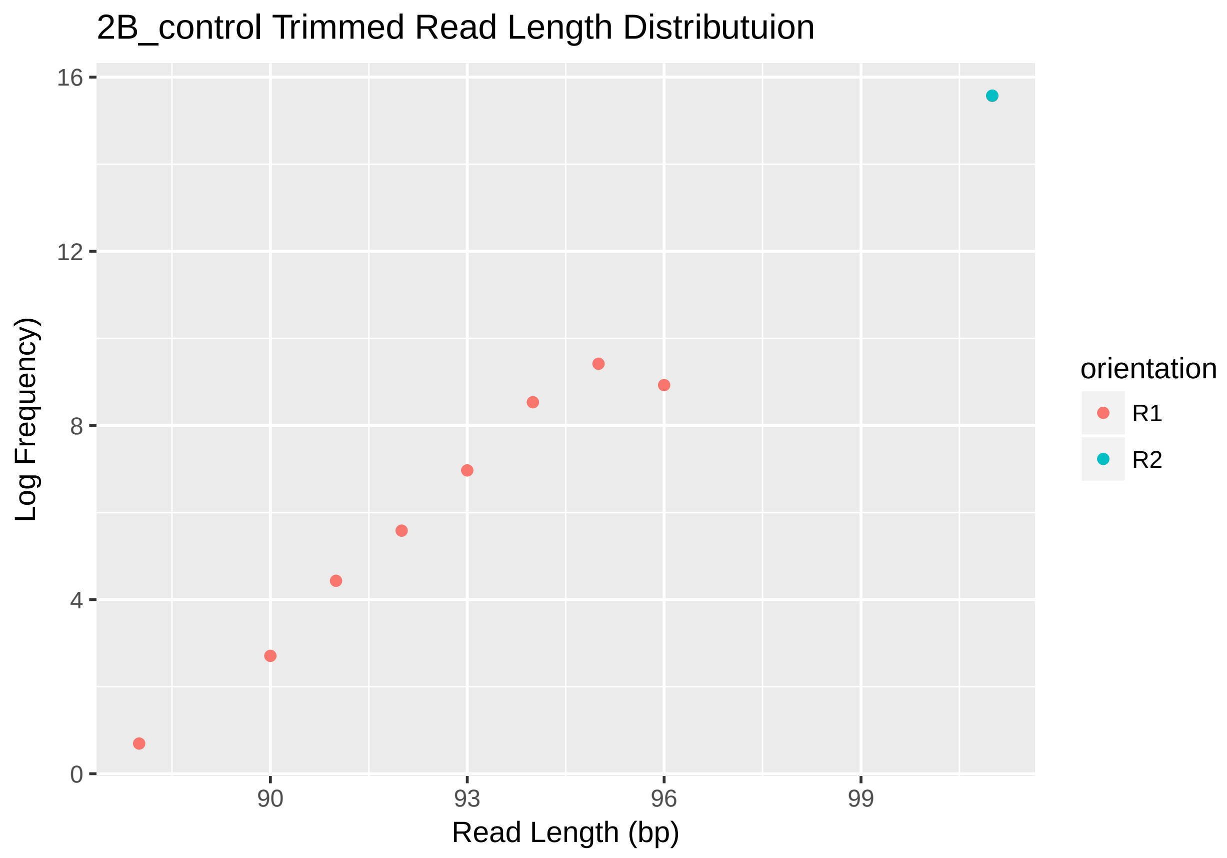
Distributions of the frequency of both forward and reverse read lengths were plotted for both the `2B_control` and `4E_fox` treatment. First the data were transformed in R from Unix-generated frequency tables.

```r
library(tidyverse)

setwd('/Users/JakeVanCampen/Documents/SF_seq/')

R1_2B <- read.csv('length_dist/2B_R1_lengths.txt', sep = ' ', header = F)
```

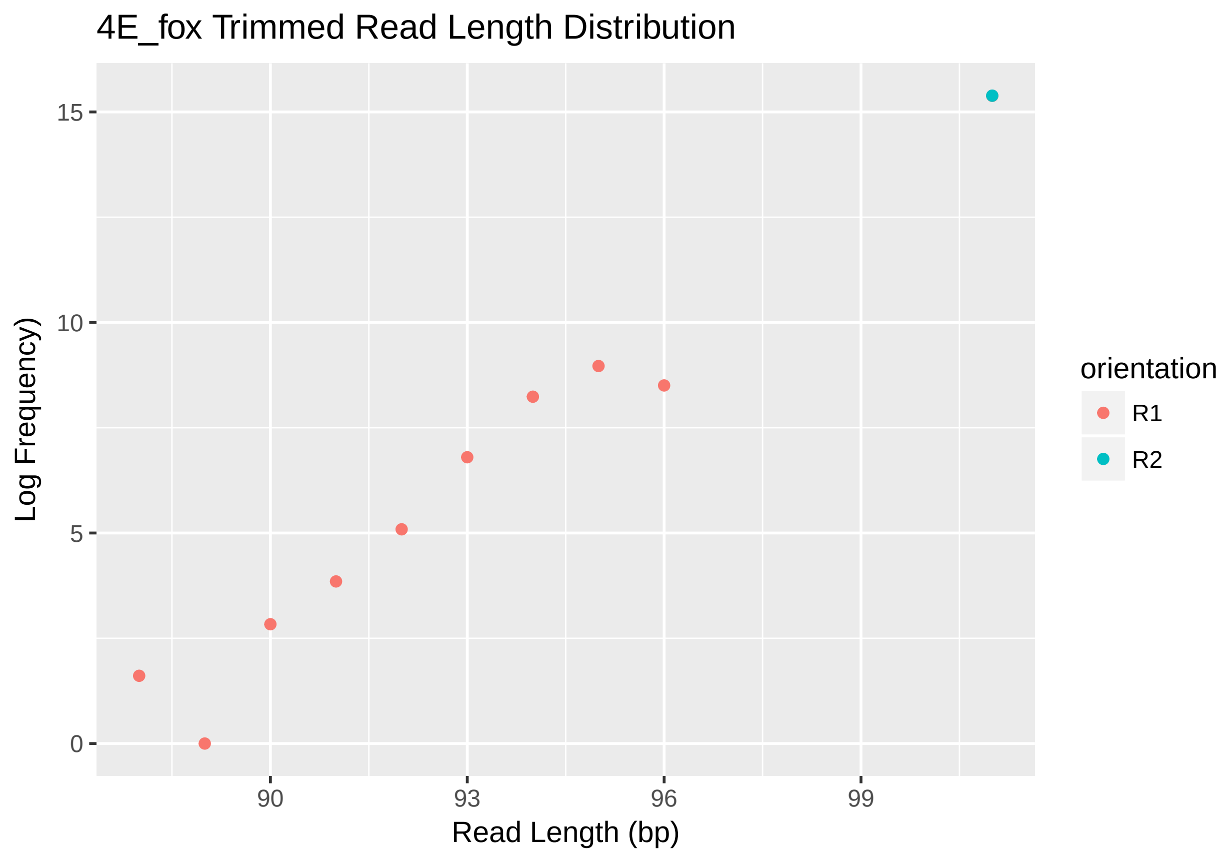
```r
read <- rep('R1', nrow(R1_2B))
R1_2B <- cbind(R1_2B,read)

R2_2B <- read.csv('length_dist/2B_R2_lengths.txt', sep = ' ',header = F)
read <- rep('R2', nrow(R2_2B))
R2_2B <- cbind(R2_2B, read)

data_2B <- rbind(R1_2B, R2_2B)
colnames(data_2B) <- c('frequency', 'length', 'orientation')


# 4E_fox length distributions
R1_4E <- read.csv('length_dist/4E_R1_lengths.txt', sep = ' ',header = F)
read <- rep('R1', nrow(R1_4E))
R1_4E <- cbind(R1_4E,read)

R2_4E <- read.csv('length_dist/4E_R2_lengths.txt', sep = ' ',header = F)
read <- rep('R2', nrow(R2_4E))
R2_4E <- cbind(R2_4E,read)

data_4E <- rbind(R1_4E,R2_4E)
colnames(data_4E) <- c('frequency', 'length', 'orientation')
```

Then the distributions for each treatment were plotted with ggplot:

```r
dist_2B <-  ggplot(data = data_2B,
                 mapping = aes(
                   x = length, y = log(frequency), color = orientation)) +
  geom_point() +
  labs(title = '2B_control Trimmed Read Length Distributuion',
       x = 'Read Length (bp)',
       y = 'Log Frequency)')

print(dist_2B)
```



```r
dist_4E <- ggplot(data = data_4E,
                  mapping = aes(x = length,
                                y = log(frequency),
                                color = orientation)) +
  geom_point() +
  labs(title = '4E_fox Trimmed Read Length Distribution',
       x = 'Read Length (bp)',
       y = 'Log Frequency)')

print(dist_4E)
```



These plots show a wider distribution of trimmed read lengths for R1 compared to R2 for both 2B_control and 4E_fox treatments. The majority of reads are seen at read length 101 for R1, while all are seen at length 101 for R2. To confirm the percent of reads that were trimmed for each treatment the following calculations were performed in R.

2B_control:

```r
total_2B <- sum(data_2B$frequency)
percent_trimmed <- sum(data_2B$frequency[data_2B$length<100])/total_2B*100
percent_untrimmed <- 100-percent_trimmed
print(paste('percent_trimmed:',percent_trimmed))
```

```
## [1] "percent_trimmed: 0.22672059480025"
```

```r
print(paste('percent_untrimmed:', percent_untrimmed))
```

```
## [1] "percent_untrimmed: 99.7732794051997"
```

4E_fox:

```r
total_4E <- sum(data_4E$frequency)
percent_trimmed <- sum(data_4E$frequency[data_4E$length<100])/total_4E*100
percent_untrimmed <- 100-percent_trimmed
print(paste('percent_trimmed:',percent_trimmed))
```

```
## [1] "percent_trimmed: 0.183785562433532"
```

```r
print(paste('percent_untrimmed:',percent_untrimmed))
```

```
## [1] "percent_untrimmed: 99.8162144375665"
```

These results highlight that the differences in the read length distributions for both treatments (R2 compared to R1) accounts for less than 1% of the total reads. Fragment analyzer traces show average library size was around 400bp for 2B_control and 375 for 4E_fox (Figures 13,14). For adapter sequences to appear in the sequenced reads, fragments of size less than ~250bp need to be sequenced. Fragmnents of these sizes account for a very small proportion of the fragmnets as shown in Figures 13 and 14, therfore it makes sense that adapter contamination occurs in a small fraction of the reads.

rRNA reads and strand-specificity

Murine non-coding RNA sequences were downloaded from the ensemble 90 database, and a gmap database was built from the ncRNA sequences using the following GMAP commands:

```
# load gsnap modules
ml intel/2017a GMAP-GSNAP

# build gmap db in specified directory
gmap_build -d mouse_rRNA -D /home/jvancamp/SF-seq -g
/home/jvancamp/SF-seq/Mus_musculus.GRCm38.ncrna.fa.gz
```

The reads were then aligned to ncRNA database to determine the proportion of reads containing potential rRNA sequences. The following scripts were used to align the reads with GSNAP.

```
# load gsnap/gmap
ml intel/2017a GMAP-GSNAP

# run GSNAP to align reads
gsnap -d mouse_rRNA -D ~/SF-seq -t 16 -A sam -N 1 --gunzip \
--split-output=2B --no-sam-headers \
~/SF-seq/2B_control/2_2B_control_S2_L008_R1_001.1.fq.gz \
~/SF-seq/2B_control/2_2B_control_S2_L008_R2_001.2.fq.gz


# run GSNAP to align reads
gsnap --gunzip -t 16 --split-output=4E -A sam -N 1 -d mouse_rRNA -D \
~/SF-seq --no-sam-headers \
~/SF-seq/4E_fox/29_4E_fox_S21_L008_R1_001.1.fq.gz \
~/SF-seq/4E_fox/29_4E_fox_S21_L008_R2_001.2.fq.gz
```

A summary file of the number of reads for each mapping category was generated with a simple unix command. Data were transformed in R, producing a table of the following alignment results:

```r
library(stringr)
library(pander)

align_2B <- read.csv('/Users/JakeVanCampen/Documents/SF_seq/align/2B_align.txt',
                     sep = ' ', header = F)

align_4E <- read.csv('/Users/JakeVanCampen/Documents/SF_seq/align/4E_align.txt',
                     sep = ' ', header = F)

# report the proportion of mapping reads
concordant_2B <- align_2B[grepl('(concordant)', align_2B$V2),]
nonmapping_2B <- sum(align_2B[!grepl('(concordant)', align_2B$V2),]$V1)
mapping_2B <- sum(concordant_2B$V1)
```

```r
concordant_4E <- align_4E[grepl('(concordant)', align_4E$V2),]
nonmapping_4E <- sum(align_4E[!grepl('(concordant)', align_4E$V2),]$V1)
mapping_4E <- sum(concordant_4E$V1)

prop_mapped_2B <- mapping_2B/(sum(align_2B$V1))*100
prop_mapped_4E <- mapping_4E/(sum(align_4E$V1))*100


row_2B <- c(mapping_2B, nonmapping_2B, prop_mapped_2B)
row_4E <- c(mapping_4E, nonmapping_4E, prop_mapped_4E)
col_treat <- c('2B_control', '4E_fox')

alignment <- rbind(row_2B,row_4E)
alignment <- as.tibble(cbind(alignment,col_treat))
colnames(alignment) <- c('mapping','nonmapping','percent_mapped','treatment')

alignment <- alignment %>% select(treatment,mapping,nonmapping,percent_mapped)
alignment <- type_convert(alignment)

pander(type_convert(alignment))
```

Parsed with column specification: cols(treatment = col_character())

treatment	mapping	nonmapping	percent_mapped
2B_control	438312	11298299	3.735
4E_fox	673330	9201122	6.819

This table shows the number of reads mapping and not mapping to mouse ncRNA for each treatment, and represents that there are a potential 3.73% of reads from ncRNAs for 2B_control, and 6.8% of reads from ncRNAs for 4E_fox. I have no logical hypothesis for the comparison of these numbers given that the variation in rRNA present in sequencing libraries is likely due to systematic error in library prep between the treatments.

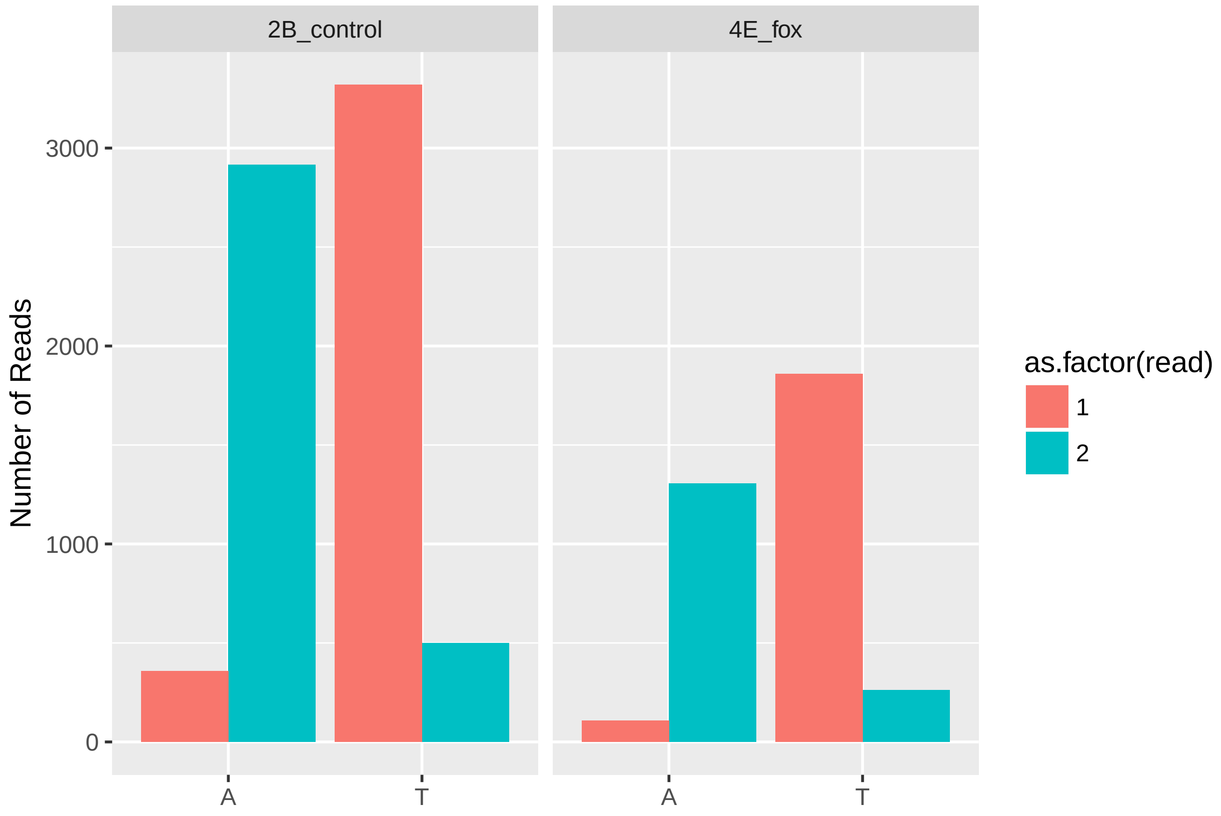
Finally, to demonstrate the 'strandedness' of the libraries. I carried out the following analysis using Unix commands:

```
[jvancamp@ln1 SF-seq]$ zcat 2_2B_control_S2_L008_R1_001.fastq.gz | \
grep 'AAAAAAAAAAAAAAAAAAAAAAAAAAA$' | wc -l
358
[jvancamp@ln1 SF-seq]$ zcat 2_2B_control_S2_L008_R2_001.fastq.gz | \
grep 'AAAAAAAAAAAAAAAAAAAAAAAAAAA$' | wc -l
2915
[jvancamp@ln1 SF-seq]$ zcat 2_2B_control_S2_L008_R2_001.fastq.gz | \
grep 'TTTTTTTTTTTTTTTTTTTTTTTTTTT$' | wc -l
500
[jvancamp@ln1 SF-seq]$ zcat 2_2B_control_S2_L008_R1_001.fastq.gz | \
grep 'TTTTTTTTTTTTTTTTTTTTTTTTTTT$' | wc -l
3319
[jvancamp@ln1 SF-seq]$ zcat 29_4E_fox_S21_L008_R1_001.fastq.gz | \
grep 'TTTTTTTTTTTTTTTTTTTTTTTTTTT$' | wc -l
1858
[jvancamp@ln1 SF-seq]$ zcat 29_4E_fox_S21_L008_R1_001.fastq.gz | \
grep 'AAAAAAAAAAAAAAAAAAAAAAAAAAA$' | wc -l
```

```
107
[jvancamp@ln1 SF-seq]$ zcat 29_4E_fox_S21_L008_R2_001.fastq.gz | \
grep 'AAAAAAAAAAAAAAAAAAAAAAAAAAA$' | wc -l
1305
[jvancamp@ln1 SF-seq]$ zcat 29_4E_fox_S21_L008_R2_001.fastq.gz | \
grep 'TTTTTTTTTTTTTTTTTTTTTTTTTTT$' | wc -l
262
```

Plot the results in ggplot:

```r
read <- rep(c("1","2"),4)
trt <- c(rep('2B_control', 4),rep('4E_fox',4))
poly <- rep(c(rep('A',2),rep('T',2)),2)
num_reads <- c(358, 2915, 3319, 500, 107, 1305, 1858, 262)

strd_table <- type_convert(as.tibble(cbind(read, trt, poly, num_reads)))

ggplot(strd_table, aes(poly, num_reads, fill = as.factor(read))) +
  geom_col(position = position_dodge()) +
  facet_grid(~trt)+
  labs(x = '', y = 'Number of Reads')
```



The barchart shows the number of reads containing poly A and poly T tails for reads 1 and 2 corresponding to each treatment. These results show that read 1 has substantially more reads with a poly-T tail compared to read 2 for both treatments, and that read 2 has substantially more reads with poly-A tails compared to read 1 for both treatments. This makes sense because it is expected that read 1 will be the reverse compliment of the RNA strand (poly-T tail), and read 2 will represent the RNA strand (poly-A tail). This shows the libraries are stranded, and there are many reads with poly-A, poly-T tails because they poly-A selection was carried out during library prep.

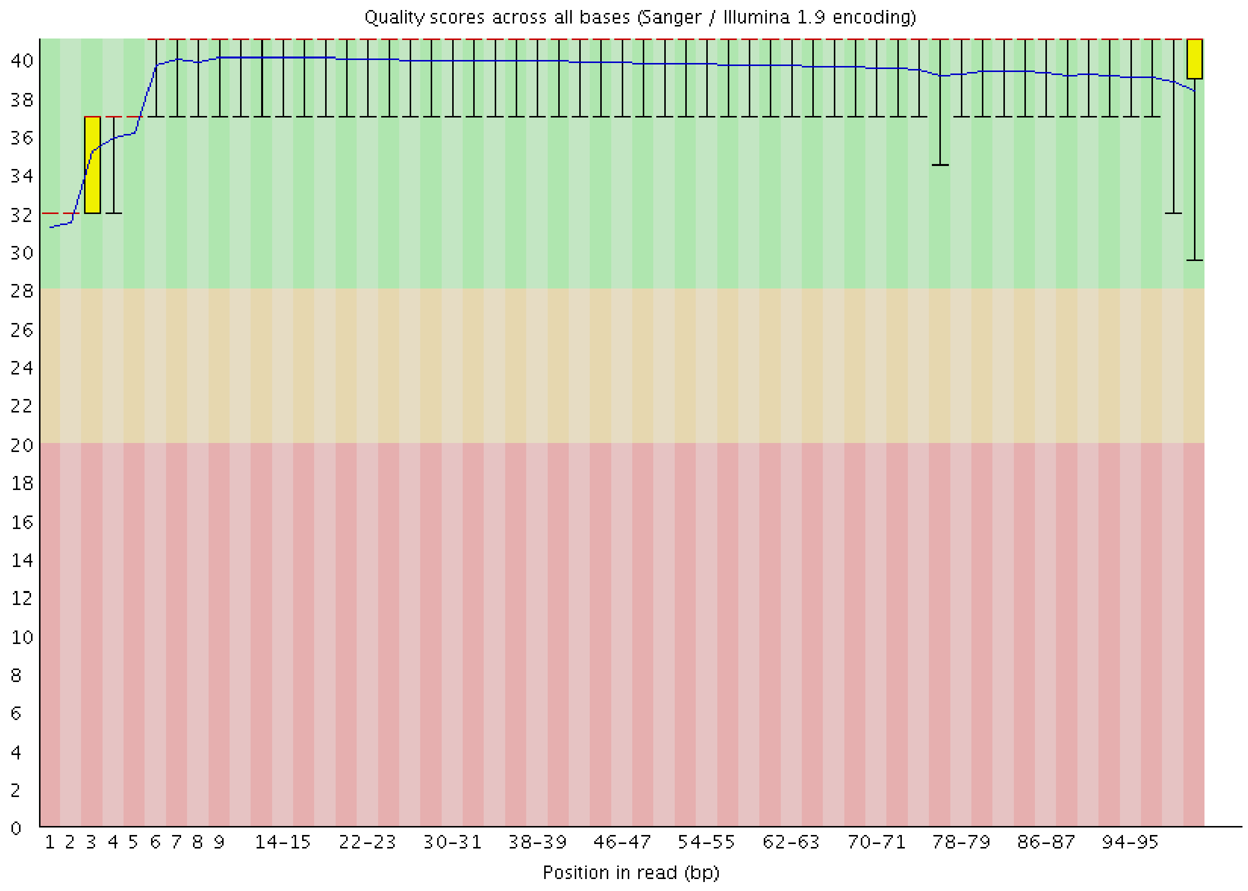


Figure 1: 29_4E_fox_S21 Read1 QScore distribution

Figures

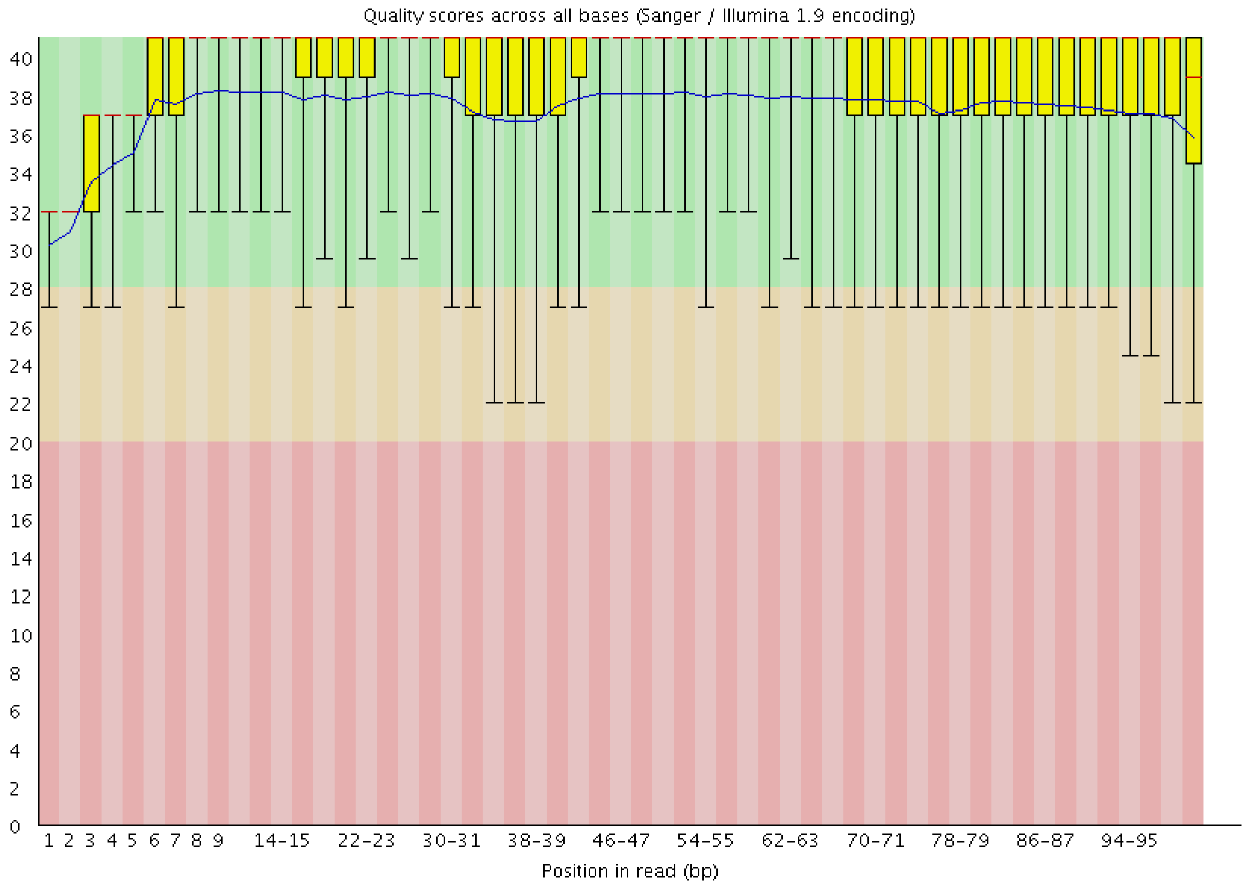


Figure 2: 29_4E_fox_S2 Read2 QScore distribution

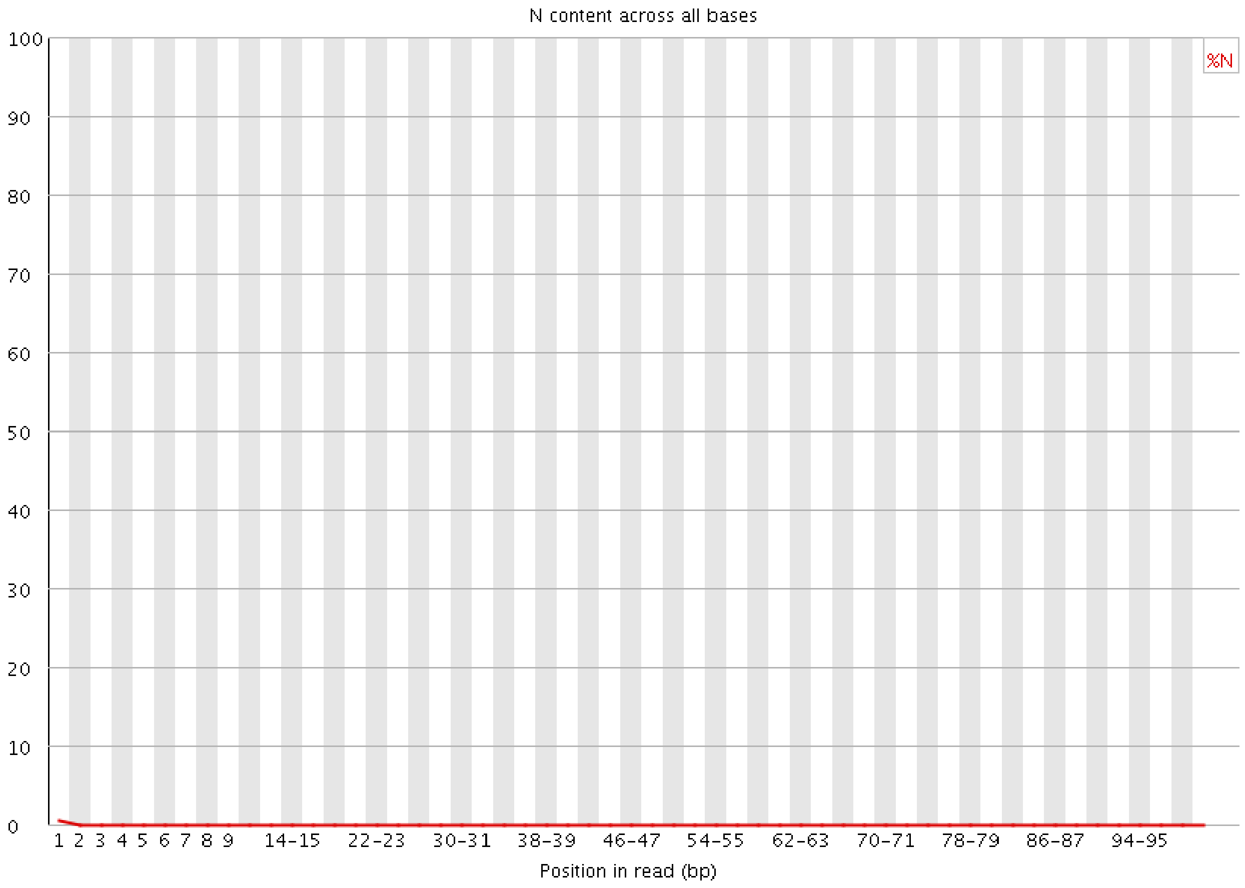


Figure 3: 29_4E_fox_S2 Read 1 Per-base N-content

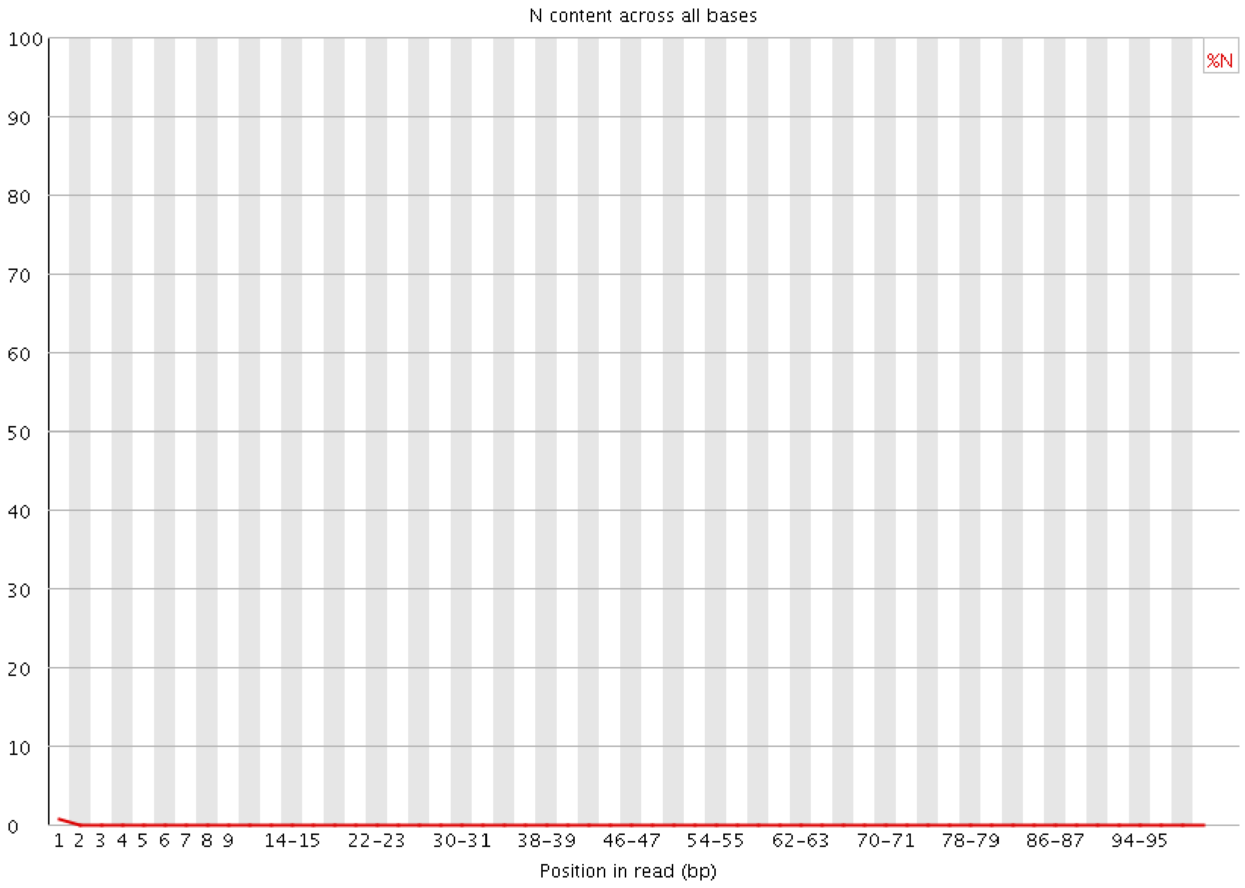


Figure 4: 29_4E_fox_S2 Read 2 Per-base N-content

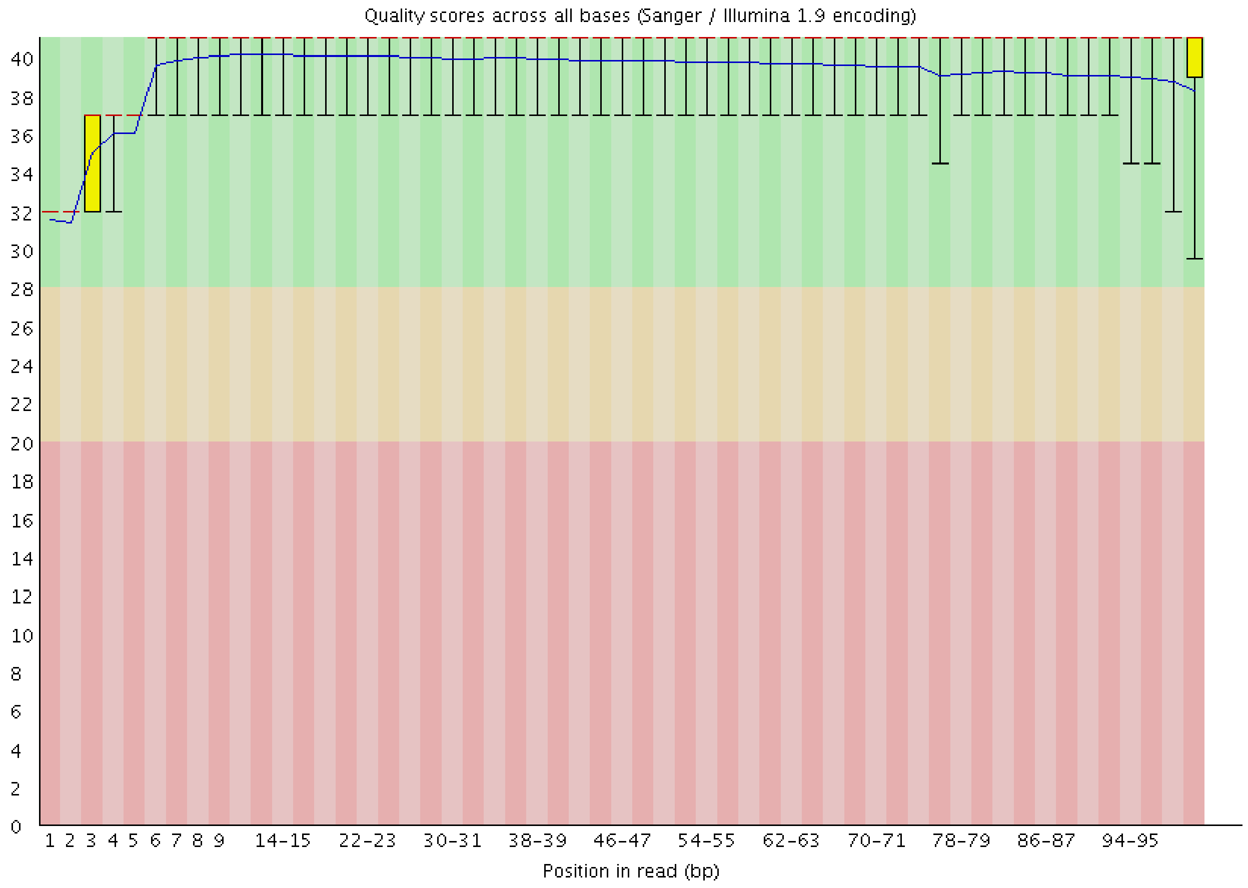


Figure 5: 2_2B_control_S2 Read1 QScore distribution

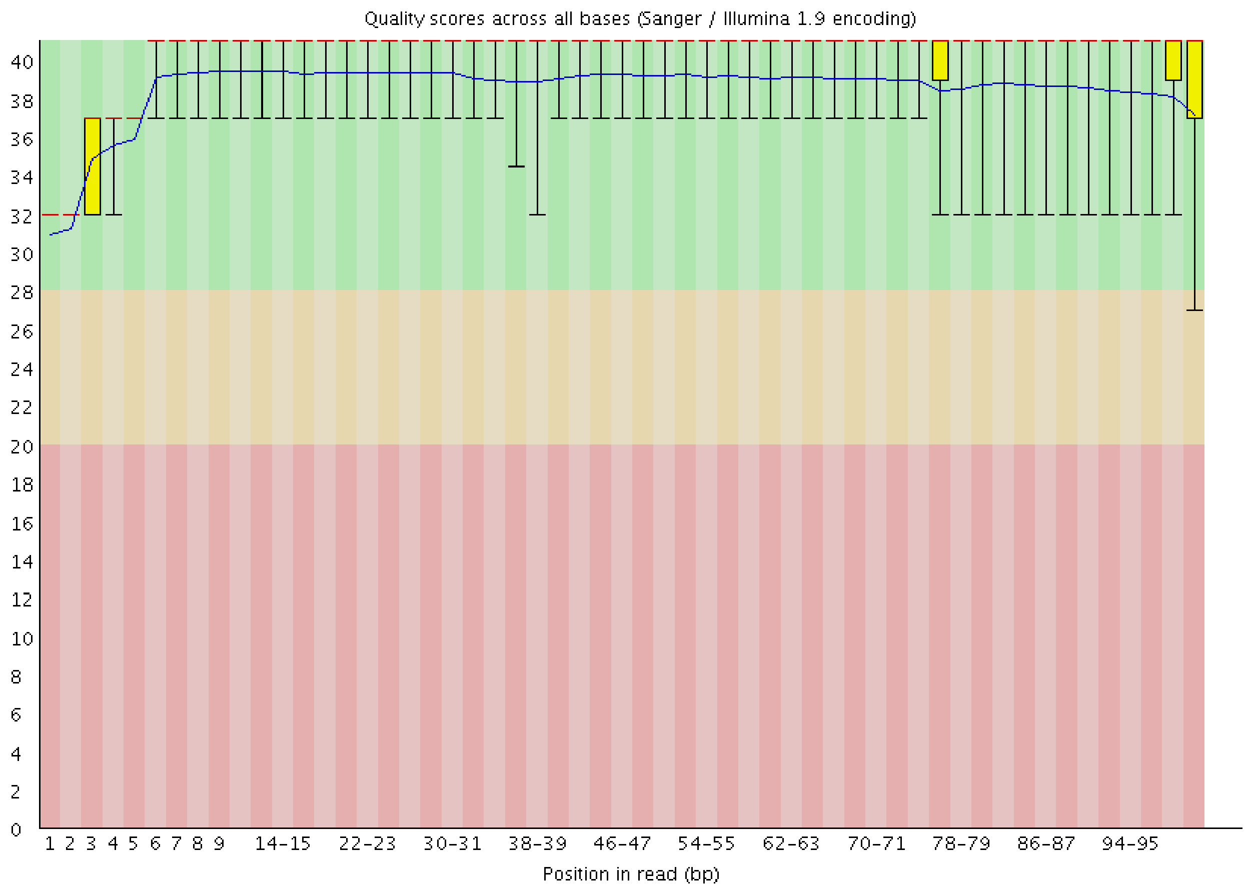


Figure 6: 2_2B_control_S2 Read2 QScore distribution

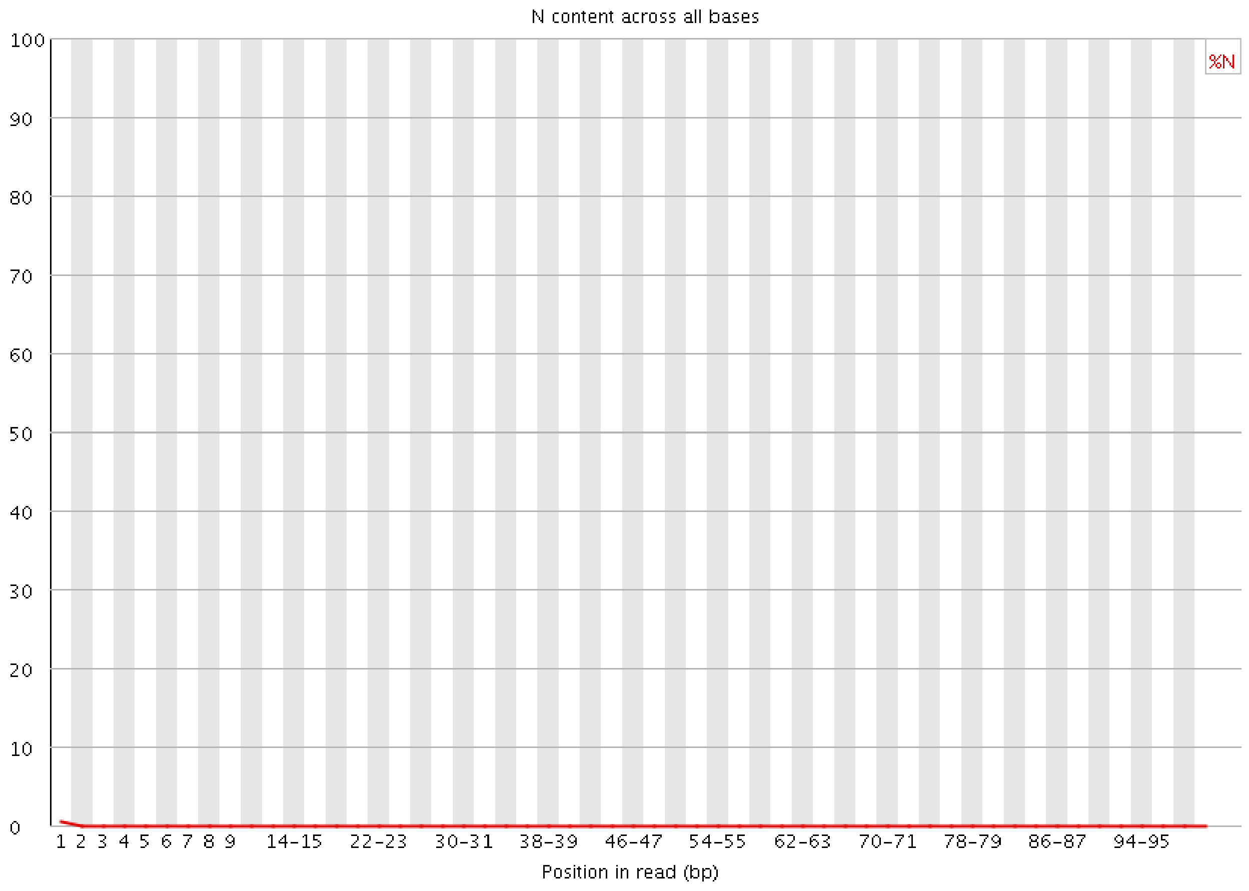


Figure 7: 2_2B_control_S2 Per-base N-content

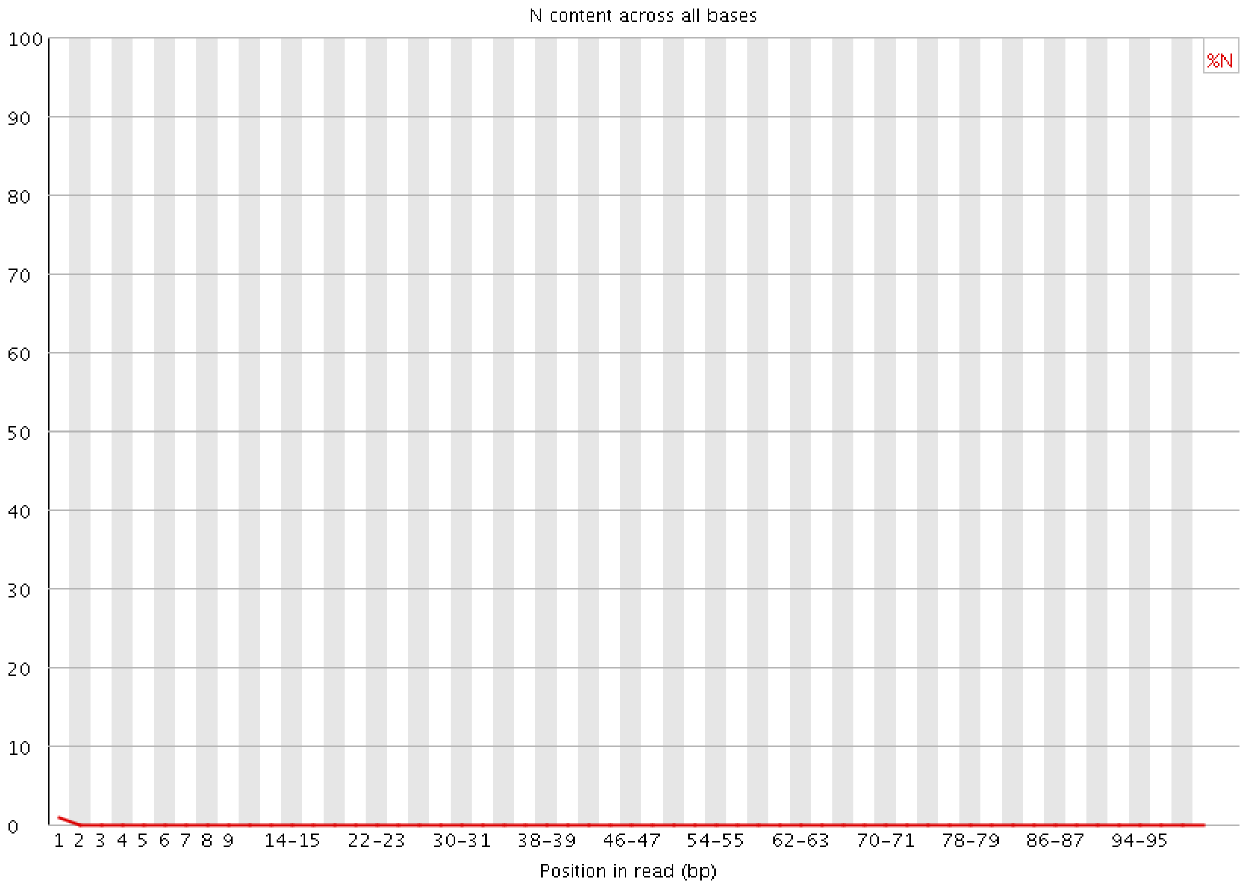


Figure 8: 2_2B_control_S2 Read2 Per-base N-content

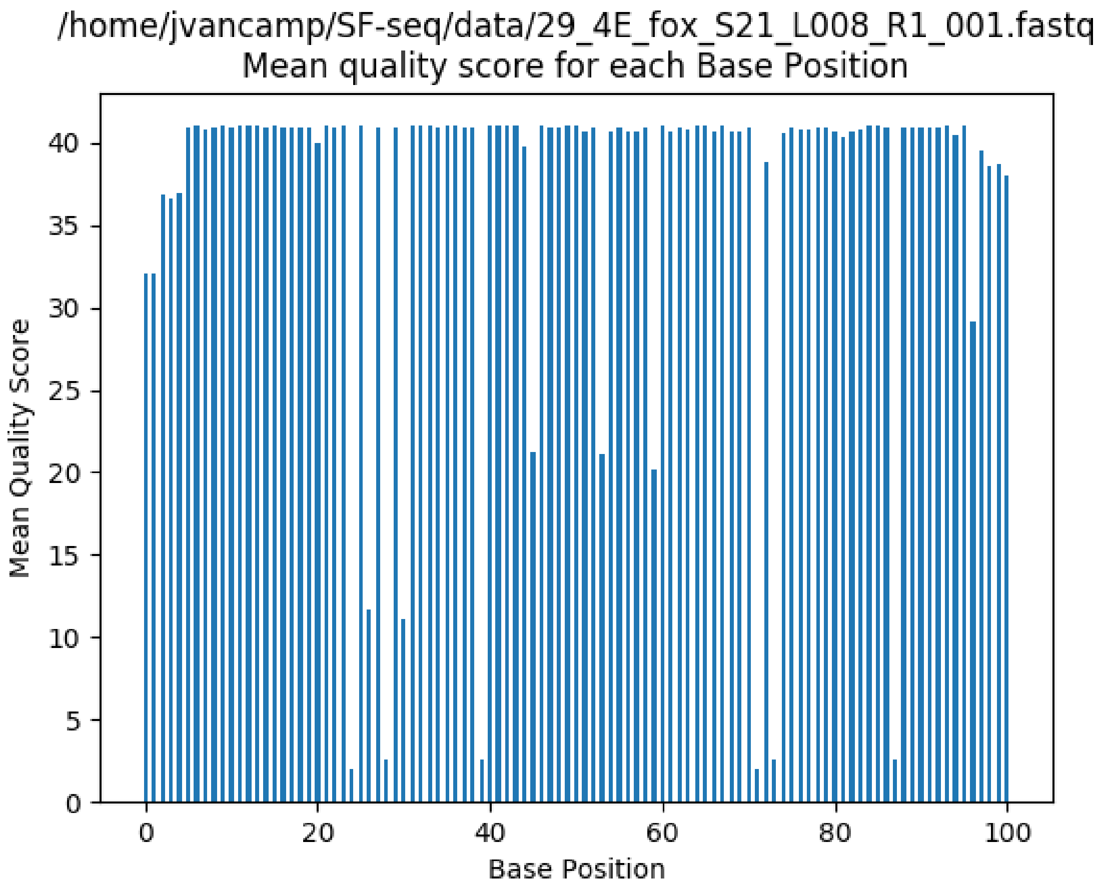


Figure 9: 29_4E_fox_S21 Read1 QScore distribution (My Script)

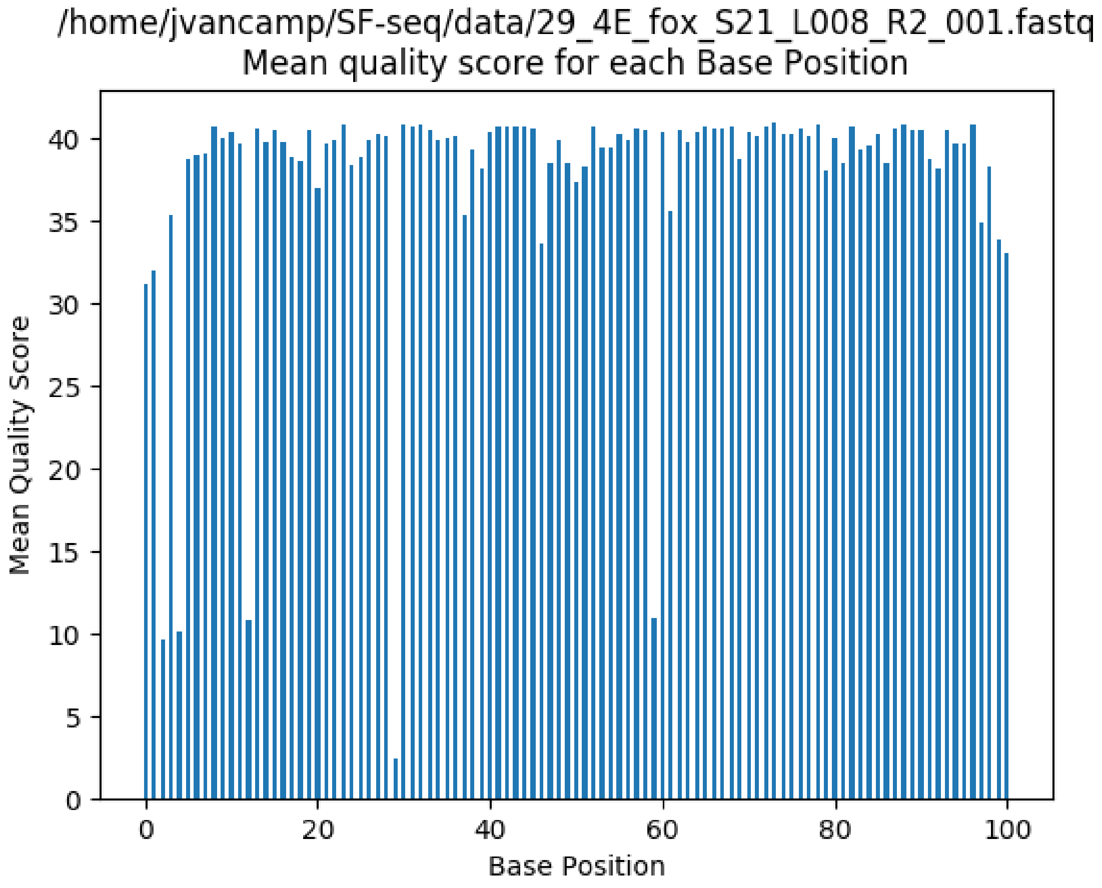


Figure 10: 29_4E_fox_S21 Read2 QScore distribution (My Script)

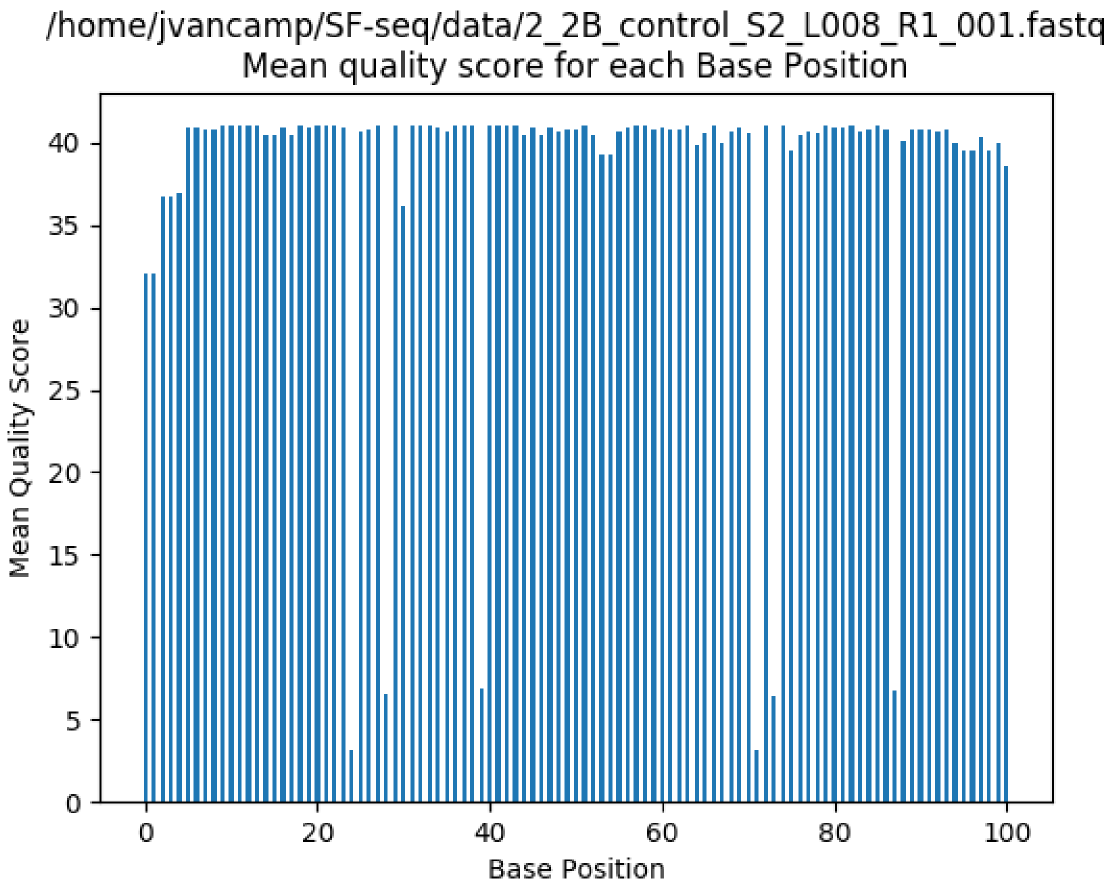


Figure 11: 2__2B__control__S2 Read1 QScore distribution (My Script)

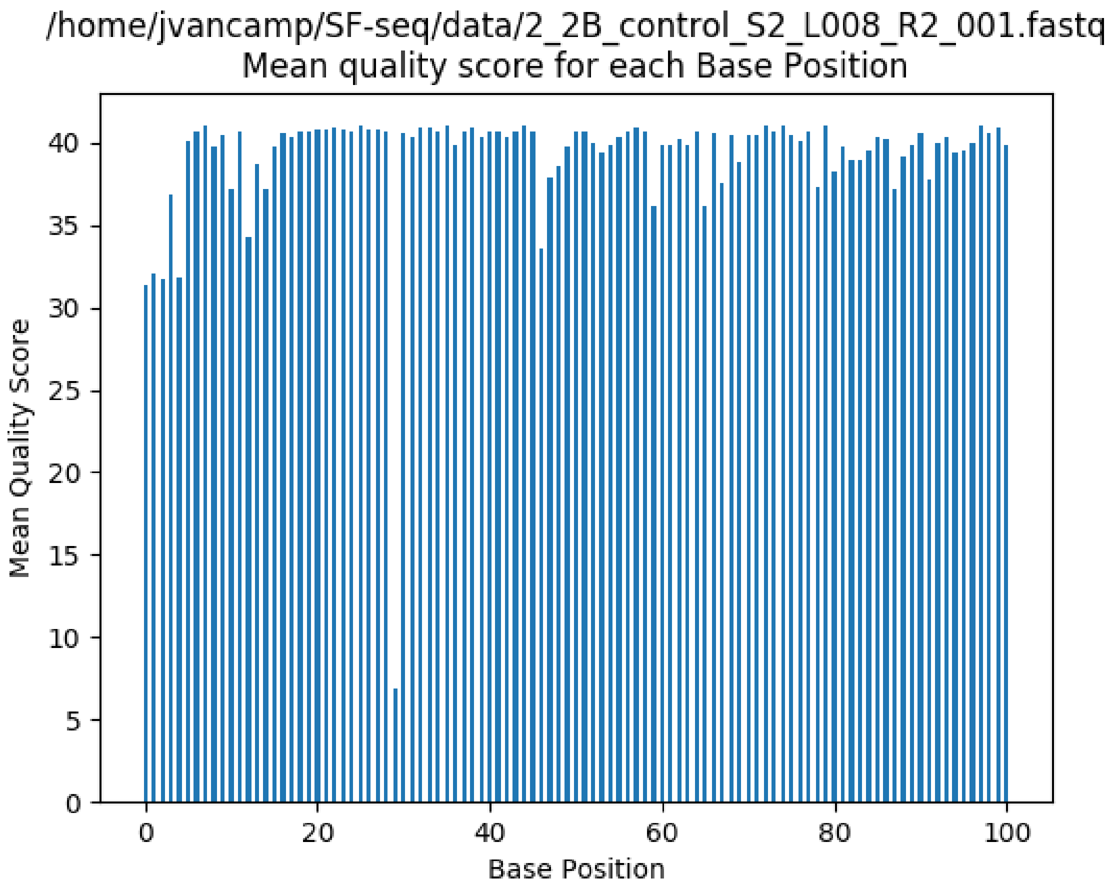
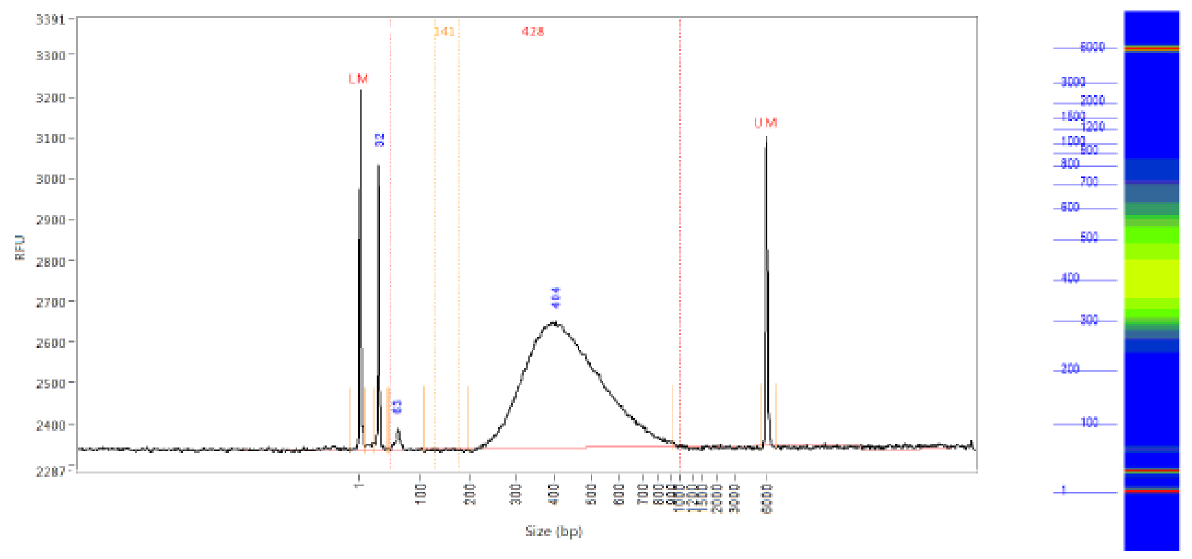


Figure 12: 2__2B__control__S2 Read2 QScore distribution (My Script)

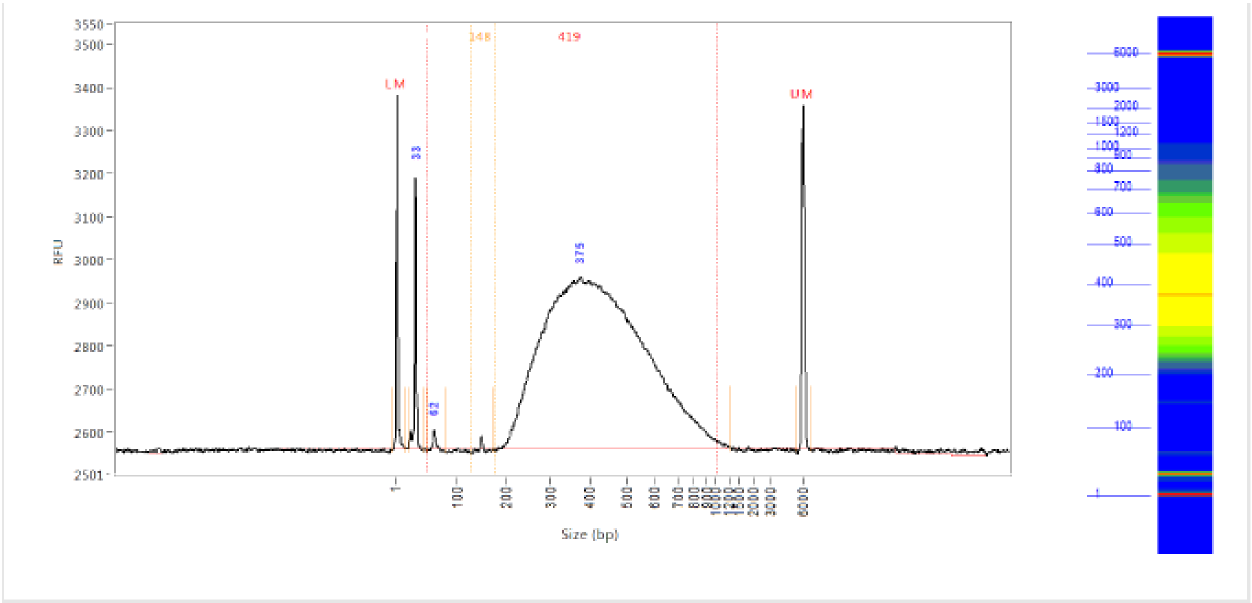


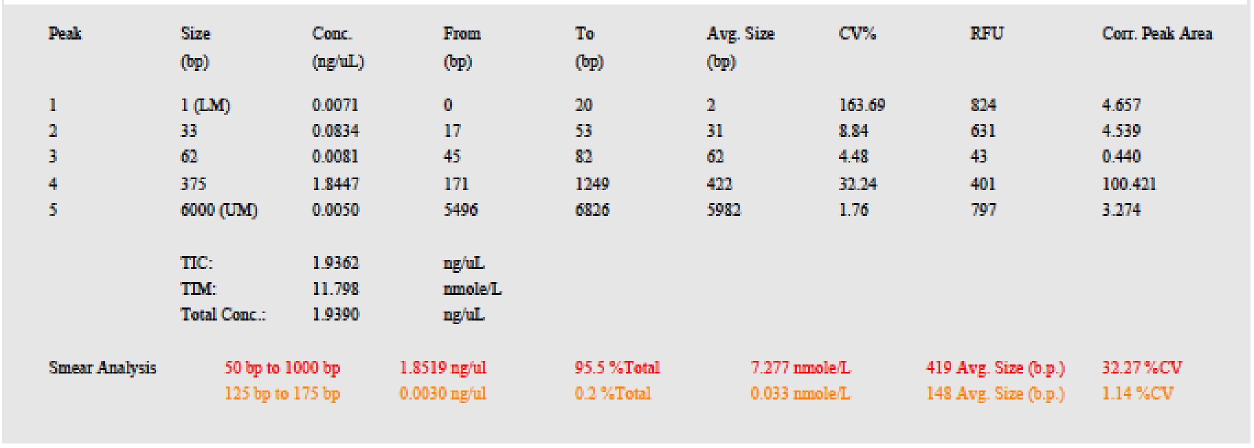
Peak	Size (bp)	Conc. (ng/uL)	From (bp)	To (bp)	Avg. Size (bp)	CV%	RFU	Corr. Peak Area
1	1 (LM)	0.0104	0	10	1	159.89	876	4.910
2	32	0.1112	27	44	32	4.63	694	4.386
3	63	0.0190	48	106	62	6.40	49	0.751
4	404	1.4179	193	923	432	25.11	308	55.946
5	6000 (UM)	0.0064	5521	6876	5986	1.75	755	3.048

TIC:	1.5481	ng/uL
TIM:	11.576	nmole/L
Total Conc.:	1.5590	ng/uL

Smear Analysis						
	50 bp to 1000 bp	1.4378 ng/ul	92.2 %Total	5.533 nmole/L	428 Avg. Size (b.p.)	27.22 %CV
	125 bp to 175 bp	0.0000 ng/ul	0.0 %Total	0.000 nmole/L	141 Avg. Size (b.p.)	10.27 %CV

Figure 13: 2B Fragment Analysis Trace





Peak	Size (bp)	Conc. (ng/uL)	From (bp)	To (bp)	Avg. Size (bp)	CV%	RFU	Corr. Peak Area
1	1 (LM)	0.0071	0	20	2	163.69	824	4.657
2	33	0.0834	17	53	31	8.84	631	4.539
3	62	0.0081	45	82	62	4.48	43	0.440
4	375	1.8447	171	1249	422	32.24	401	100.421
5	6000 (UM)	0.0050	5496	6826	5982	1.76	797	3.274
	TIC:	1.9362	ng/uL					
	TIM:	11.798	nmole/L					
	Total Conc.:	1.9390	ng/uL					

Smear Analysis	50 bp to 1000 bp	1.8519 ng/ul	95.5 %Total	7.277 nmole/L	419 Avg. Size (b.p.)	32.27 %CV
	125 bp to 175 bp	0.0030 ng/ul	0.2 %Total	0.033 nmole/L	148 Avg. Size (b.p.)	1.14 %CV

Figure 14: 4E Fragment Analysis Trace