Chapter 1

Code

For the simulations in this project, I used R; "a language and environment for statistical computing and graphics" [1]. I created a package with the functions needed to created the Kozachenko-Leonenko entropy estimator (KLEE), and then used this package to run simulations on samples from different statistical distributions to create the results in this paper.

To create my package `Entropy-Estimators`, I used two of Hadley Wickham's [2] packages; `devtools` and `roxygen2`, I also used `ggplot2` to plot the graphs in this paper. `Entropy-Estimators` also has 3 dependency packages (alongside the base R packages); `dplyr` for the manipulation of data, `FNN` for the kth nearest neighbour function, and `Rcpp` to create a C++ for loop for faster computation. I will outline the important code used for the simulations; however, the full package and a complete account of the code used can be found on my GitHub page *https://github.com/KarinaMarks/Entropy-Estimators.*

1.1 The Estimator

About KLEE

1.2 Exact Entropies

To consider the bias of the estimator, I had to find the exact value of entropy from a 1-dimensional normal, uniform and exponential distribution. The function written to return this for the normal distribution is `NormalEnt` with parameter `sd`, the standard deviation of the sample, we do not need the mean value for finding the entropy of the normal distribution. The function is defined as follows;

```
NormalEnt <- function(sd){
  (log(sqrt(2*pi*exp(1))*sd))
}
```

With `sd =1`, as is true in the samples considered here, we find the entropy to be given by;

```
> NormalEnt(sd=1)
[1] 1.418939
```

The function for the uniform distribution is `UniformEnt`, with parameters `min` and `max`, is defined as;

```
UniformEnt <- function(min, max){
  log(max - min)
}
```

Here we use `min=0` and `max=100` in the samples considered; thus we find the exact entropy to be given by;

```
> UniformEnt(min = 0, max = 100)
[1] 4.60517
```

Lastly, for the exponential distribution we have the function `ExpoEnt`, with only one parameter `rate`, defined below;

```
ExpoEnt <- function(rate){
  1 - log(rate)
}
```

In this paper we are using the exponential distribution with parameter `rate=1.5`, thus;

```
> ExpoEnt(rate = 1.5)
[1] 0.5945349
```

1.3 Simulations

*In this section I used the packages **readr** to save the data, **dplyr** for the manipulation of data and **Rcpp** for creating a fast loop over hundreds of iterations.*

I created functions `normalloop`, `uniformloop` and `expoloop`, in C++ which, for each sample size N creates M samples of that size, finds the estimator for sample and puts the result in a vector of length M. These functions are as follows;

```
cppFunction('
          NumericVector normalloop(int M, int N, int k){
            NumericVector est(M);
            NumericVector x(N);
            for (int i = 0; i < M; i++) {
            int sd=1;
            Function KLEE("KLEE");
            Function rnorm("rnorm");
            x=rnorm(N, sd=sd);
```

```
est[i]=as<double>(KLEE(x ,k=k));
}
return Rcpp::wrap(est);
}
')
```

for the normal distribution, and for the uniform distribution;

```
cppFunction('
          NumericVector uniformloop(int M, int N, int k,
            int min, int max){
            NumericVector est(M);
            NumericVector x(N);
            for (int i = 0; i < M; i++) {
            Function KLEE("KLEE");
            Function runif("runif");
            x=runif(N, min=min, max=max);
            est[i]=as<double>(KLEE(x ,k=k));
            }
            return Rcpp::wrap(est);
            }
            ')
```

Lastly for the exponential distribution;

```
cppFunction('
          NumericVector expoloop(int M, int N, int k,
            float rate){
            NumericVector est(M);
            NumericVector x(N);
            for (int i = 0; i < M; i++) {
            Function KLEE("KLEE");
            Function rexp("rexp");
            x=rexp(N, rate=rate);
            est[i]=as<double>(KLEE(x ,k=k));
            }
            return Rcpp::wrap(est);
            }
            ')
```

Using these functions I created each column of the tables, where each table is a different distribution, each column is a different value of $k \in \{1, 2, ..., 11\}$ and each row is a different sample size $N \in \{100, 200, 300, ..., 50000\}$. Below is how the column with $k = 1$ for the normal distribution was created, all other columns were done similarly;

```
# initalise the data frame with all sample sizes n
data.frame(n = seq(100, 50000, 100)) %>%
  # group by n to use summarise on each n
```

```
  dplyr::group_by(n) %>%
  # for each n the mean of the normalloop function is
      found, taken over 500 samples of size n
  summarise(Ent = mean(normalloop(M=500, N=n, k=1, rate
      =0.5), na.rm=TRUE))
```

1.4 Analysis

In this section I use the packages `ggplot2` for the graphs, `dplyr` for the data manipulation and `readr` to read in my csv data files.

Once I obtained all the simulated data, I found the modulus of the bias for each sample size N, each k and each distribution. I then selected the information for all k with $N = 100, 25000$ and 50000, to display in Tables **??**, **??** and **??**, this involved taking `Data` and subtracting either `NormalEnt(sd=1)`, `UniformEnt(min=0, max=100)` or `ExpoEnt(rate=0.5)` from the estimators, depending on the distribution.

Next, I plotted graphs for each k of the logarithm of the bias of the estimator $\hat{H}_{N,k}$ against the logarithm of the sample size N, shown in Figures **??**, **??**, **??**, **??**, **??** and **??**. I used the following code to do this, changing the `y` value to either `k1, k2, ..., k11` depending on which value of k I was plotting. Also the `data` would be read in from a different file for each distribution, the code below shows plotting the simulations from the normal distribution with $k = 1$.

```
# read in the data as a data frame
data <- as.data.frame(read_csv("./Data/data_normal.csv"))

# find the modulus of the bias for all n and k
data[-1] <- abs(data[-1] - NormalEnt(1))

# take the logarithm of everything
logdata <- log(data)

# the max and min x values
xmin <- min(logdata$n)
xmax <- max(logdata$n)

# the min and max y values
ymin <- -15 # this is because there are only 5 values
    smaller than -15
ymax <- ceiling(max(logdata[-1]))

# plot the graph for each k - here k=1
# defining the data
ggplot(data=logdata, aes(x=n, y=k1)) +
  # plotting the points
```

```
  geom_point(size=0.8) +
  # adding a linear regression line
  geom_smooth(method="lm") +
  # labelling the axis
  xlab("log(N)") +
  ylab("log|Bias(H)|") +
  # setting the axis limits
  xlim(c(xmin, xmax)) +
  ylim(c(ymin, ymax)) +
  # choosing the graph theme
  theme_minimal()
```

Additionally, I created a summary table of the useful information needed, containing the coefficients of the intercept ζ and the gradient $-a_k$ from the regression analysis, the coefficient of determination R^2 and the standard error σ also from the regression analysis. I also modified ζ and$-a_k$ to find both a_k and c_k. The code below shows how I did this for the normal distribution, and it is similar for the other two distributions, just changing the data inputted and the exact value of entropy used to find the bias.

```
# read in the data as a data frame
data <- as.data.frame(read_csv("./Data/data_normal.csv"))

# find the modulus of the bias for all n and k - removing
    the 1st column, n
data[-1] <- abs(data[-1] - NormalEnt(1))

# take the logarithm of everything
logdata <- log(data)

# initalise and empty df with everything in
Info <- data.frame(k = 1:11, ak = rep(0, 11),
                   zeta = rep(0, 11), powera = rep(0, 11),
                   ck = rep(0, 11), rsquared = rep(0, 11),
                   sigma = rep(0, 11))

# fill in data frame
for (k in 1:11){
  # find linear relationship of logarithm of bias against
      logrithm of n
  reg <- lm(logdata[[k+1]] ~ logdata$n)

  # the coeffs of log(bias)
  zeta <- round(reg$coefficients[["(Intercept)"]], 4)
  ak <- round(reg$coefficients[["logdata$n"]], 4)

  # the coeffs of normal bias
```

```r
  ck <- round(exp(reg$coefficients[["(Intercept)"]]), 4)
  powera <- -ak

  # find the R squared value
  rsquared <- summary(reg)$r.squared

  # find the standard error
  sigma <- summary(reg)$sigma

  # fill in the each row for k=k
  Info[k,] <- c(k, ak, zeta, powera, ck, rsquared, sigma)
}

# save the Info data to a csv file
write_csv(Info, "../Data/normal_info.csv")
```

These tables are shown in Appendix **??**, and from these I found the information in Tables **??**, **??**, **??**, **??**, **??** and **??**. Then to create Tables **??**, **??** and **??**, I just had to modify the summary tables found above to include two extra columns with the k^{a_k} and $\frac{k^{a_k}}{c_k}$, which was done by the following;

```r
# read in the summary data as a data frame
Info <- as.data.frame(read_csv("./Data/normal_info.csv"))

# make sure k is an integer not a factor for the
    following computation
Info$k <- as.integer(Info$k)

# create a new data frame, Info2 with c, k^a and (k^a)/c
Info2 <- Info %>%
  mutate("k^a" = k^-ak, "(k^a)/c" = ((k^-ak)/ck)) %>%
  select(`k^a`, ck, `(k^a)/ck`)
```

From this table I than created the graphs shown in Figures **??**, **??** and **??**, using the below code.

```r
# Graph (a) k against c
ggplot(data=Info2, aes(x=k, y=ck)) +
  # plotting the points
  geom_point() +
  # x axis labels
  scale_x_continuous(breaks = c(2:11), labels = c(2:11))
      +
  theme_minimal()

# Graph (b) k^a against c
ggplot(data=Info2, aes(x=`k^a`, y=ck)) +
  # plotting the points
```

```
  geom_point() +
  theme_minimal()
```

The last part of analysis conducted, was plotting all the regression lines of the logarithm of N against the logarithm of the bias, for each k on the same graph. To do this I used the summary data, read in as `Info`, and the `xmin`, `xmax`, `ymin` and `ymax` found when plotting the graphs for each k separately. The following code was then used to create the graphs in Figures ??, ?? and ??;

```
# make k a factor
Info$k <- as.factor(Info$k)

# plot graph of comparison for each k
ggplot()+
  # add the lines for each k
  geom_abline(aes(intercept=zeta, slope=a, colour=k),
      data=Info, size=1) +
  # set the axis limits
  ylim(c(ymin, ymax)) +
  xlim(c(xmin, xmax))+
  # set the axis labels
  xlab("log(N)") +
  ylab("log(Bias(H))") +
  # set the graph title
  ggtitle("Comparison of the regression lines for Normal
      distribution")


# plot graph of comparison for each k - enlarged
ggplot()+
  # add the ines for each k
  geom_abline(aes(intercept=zeta, slope=a, colour=k),
      data=Info, size=1) +
  # set tha axis limits - smaller this time for the
      enlarged plot
  ylim(c(-9.5, -7.5)) +
  xlim(c(9, 11))+
  # set the axis labels
  xlab("log(N)") +
  ylab("log(Bias(H))") +
  # set the graph title
  ggtitle("Comparison of the regression lines for Normal
      distribution")
```

Bibliography

[1] R project. https://www.r-project.org/about.html. Accessed: 18th March 2017.

[2] Hadley Wickham. http://hadley.nz/. Accessed: 18th March 2017.