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 called `EntropyEst`, with the functions needed to created the Kozachenko-Leonenko entropy estimator (KLEE). The functions exported form this package are; `KLEE`, `GammaFun`, `Rho`, `VolD`, `NormalEnt`, `UniformEnt`, `ExpoEnt` and `EntBias`. The functions in this package I then used 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`. `Entropy-Estimators` also has only 1 dependent package (alongside the base R packages); `FNN`, which was used for the kth nearest neighbour function.

When running simulations and creating the graphical information in this paper, I also used 4 more packages; `ggplot2` for the graphical representation of the data, `Rcpp` to create a C++ for loop for faster computation, `dplyr` for the manipulation of data, and `readr` to read and write the csv files of data.

I will outline the important code used for the simulations in this appendix; 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

Using the definition of the Kozachenko-Leonenko estimator, found in Section **??**, I created the function `KLEE`. To do this I first had to make the following functions; `GammaFun`, `Rho`, `VolD`. For the purpose of this paper, I have only created the function to work out the estimator for a 1-dimensional sample.

Firstly, I created the function `Rho`, defined in equation **??**, which takes three arguments;

- `X`; a vector of a sample to work out the distance between a value and its kth nearest neighbour

- `k`; the order or nearest neighbour to be used
- `d`; the dimension of the sample, default value 1 - since currently can only work for 1-dimensional samples

and returns a vector of distances, where its first entry is the distance from the first value of `X` to its kth nearest neighbour. The function is given by the following code;

```
Rho <- function(X, k, d=1) {
  if (d == 1){
    # find the length of the sample
    n <- length(X)
    # check that k is not larger than the length of the
        vector
    stopifnot(n > k)

    # creating the matrix of kth nn distances for X
    NNdist <- FNN::knn.dist(data=X, k=k)

    # return the kth column of the matrix
    NNdist[,k]
  } else {
    return("Dimension is too high for this estimator")
  }
}
```

Next, I created the function `GammaFun`, defined in equation **??**, which takes one argument `m` and returns a numeric value, using the following code;

```
GammaFun <- function(m) {
  # check that m > 0
  stopifnot(m > 0)
  # writing the function for the integrand
  integrand <- function(x) {x^(m-1)*exp(-x)}
  # integrating the integrand from 0 to infinity
  res <- integrate(integrand, 0, Inf)
  # selecting the result of the integral from the
      integrate class, which is a list structure
  as.numeric(res$value)
}
```

Then, using the function above for Γ, we can define the function `VolD`, equation **??**, which is the volume of the d-dimensional unit euclidean ball. This function takes one argument `d` which is the dimension and returns a numeric value using the following code;

```
VolD <- function(d) {
  # the formula to find the d-dimensional euclidean unit
      ball
  (pi^(d/2))/GammaFun(1 + (d/2))
}
```

The only other function needed for the estimator is the digamma function which is defined in base R; thus, we can now define the estimator `KLEE`. This takes three arguments;

- `X`; a vector of a sample to estimate the entropy of
- `k`; the order or nearest neighbour to be used
- `d`; the dimension of the sample, default value 1 as before for `Rho`

and returns a numeric value which is the estimator of entropy for this distribution. The code written to define this function is as follows;

```
KLEE <- function(X, k, d=1) {
  if (d==1){
    # length of the sample
    n <- length(X)
    # check that k is smaller than the length of the
        sample
    stopifnot(k < n)
    # define the vector Roe of nearest neighbour
        distances
    NN <- Rho(X, k)
    # find the volume of the unit ball
    V1 <- VolD(1)
    # return the estimator
    (1/n)*sum(log((NN*V1*(n-1))/exp(digamma(k))))
  } else {
    # this would be changed to include higher dimensions
    return(”Dimension must be 1”)
  }
}
```

1.1.1 Example

To show that the `KLEE` function works, I am giving the example below where I can work out the estimation of entropy by hand, and check that my function works for this.

Consider a sample $X = \{3, 6, 1, 7, 2\}$, from this we can easily find the estimator of entropy for $k = 1$ and $k = 2$. For both estimators the sample szie $n = 5$, the volume of the d-dimensional euclidean ball is $V_d = V_1 = 2$. Then for $k = 1$, the vector of nearest neighbour distances is given by $NN_1 = \{1, 1, 1, 1, 1\}$ and the digamma function is given by $\Psi(1) = -\gamma \approx -0.57722$. Thus we have;

$$
\begin{aligned}
\hat{H}_{5,1}(X) &= \frac{1}{5}\sum_{i=1}^{5}\frac{\log(NN_{1,i}\cdot 2\cdot(5-1))}{\exp(-\gamma)} \\
&= \frac{1}{5\exp(-\gamma)}\sum_{i=1}^{5}\log(8) \\
&= \frac{1}{\exp(-\gamma)}\log(8) \\
&\approx something wrong
\end{aligned}
$$

For $k = 2$ we have the vector of nearest neighbour distances is given by $NN_2 = \{2, 3, 2, 4, 1\}$ and the digamma function is given by $\Psi(2) = -\gamma + 1 \approx 0.42278$. Thus we have;

$$
\begin{aligned}
\hat{H}_{5,2}(X) &= \frac{1}{5}\sum_{i=1}^{5}\frac{\log(NN_{2,i}\cdot 2\cdot(5-1))}{\exp(-\gamma+1)} \\
&= \frac{1}{5\exp(-\gamma+1)}\sum_{i=1}^{5}\log(8NN_{2,i}) \\
&= \frac{1}{5\exp(-\gamma+1)}\left(\log(16)+\log(24)+\log(16)+\log(32)+\log(8)\right) \\
&\approx something?wrong
\end{aligned}
$$

TODO- look at both above?

We can also find this using the function `KLEE`, by the following code;

```
> x <- c(3, 6, 1, 7, 2)
> KLEE(x, k=1)
[1] 2.656658
> KLEE(x, k=2)
[1] 2.430898
```

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
  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;

```
# 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.

```
# 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”)
```

1.5 Conclusion

The following code is what I created to make the graphs in Chapter **??**. I changed `N` depending on what range of numbers I wished to have in my graph. I used the packages `readr` to read in my simulated data, `ggplot2` to plot the graphs and `dplyr` and `tidyr` to change the shape of my data. I have to work with `Inf` values in the uniform data for $k = 1$ and 2 and in the exponential data for $k = 1$, so I set these to `NA` values to make use of the `na.rm` argument, which is used in many functions. The following code plots an example graph for the range of data from $N = 100, ..., 1000$;

```
# choose a sample size
N <- seq(100, 1000, 100) # chosen this for example

# read in the data as a data frame for each distribtuion
# select the rows with N specified above
# remove the column n
Ndata <- as.data.frame(read_csv(”./Data/data_normal.csv”)
    ) %>%
  filter(n %in% N) %>%
  select(-n)
Udata <- as.data.frame(read_csv(”./Data/data_uniform.csv”
    )) %>%
  filter(n %in% N) %>%
  select(-n)
Edata <- as.data.frame(read_csv(”./Data/data_expo.csv”))
    %>%
  filter(n %in% N) %>%
  select(-n)

# setting all Inf values to NA, so that we can use na.rm
    in mean function
Udata$k1[is.infinite(Udata$k1)] <- NA
Udata$k2[is.infinite(Udata$k2)] <- NA
Edata$k1[is.infinite(Edata$k1)] <- NA

# work out the mean of the estimated entropy
# removing any NA values
Ndata <- Ndata %>%
  summarise_each(funs(mean))
Udata <- Udata %>%
  summarise_each(funs(mean(., na.rm = TRUE)))
Edata <- Edata %>%
  summarise_each(funs(mean(., na.rm = TRUE)))
```

```
# turning data around rows become columns and finding the
    bias
Ndata <- as.data.frame(t(Ndata)) %>%
  mutate(normal = abs(V1 - NormalEnt(1)))
Udata <- as.data.frame(t(Udata)) %>%
  mutate(uniform = abs(V1 - UniformEnt(min=0, max=100)))
Edata <- as.data.frame(t(Edata)) %>%
  mutate(expo = abs(V1 - ExpoEnt(0.5)))

# join all the df's together and adding kth column
df <- bind_cols(list(as.data.frame(Ndata$normal),
                     as.data.frame(Udata$uniform),
                     as.data.frame(Edata$expo))) %>%
  mutate(k=1:11)

# Change the column names
colnames(df) <- c("Normal", "Uniform", "Exponential", "k"
    )

# change to different format
df <- gather(df, key= distribution, value=bias, Normal,
    Uniform, Exponential)

# Remove any rows which = NaN - this occurs when all
    values are NA
df <- df[!is.na(df$bias),]

# find the maximum value of bias
ymax <- max(df$bias)

# plot the graph
ggplot(aes(x=k, y=bias, col = distribution), data=df) +
  # add the points
  geom_point() +
  # add a line between points
  geom_line() +
  # labels on the x axis and location of ticks
  scale_x_continuous(labels = (1:11), breaks = (1:11)) +
  # setting axis limits and labels
  ylim(0, ymax) +
  ylab("Bias of estimator") +
  # adding a title to the graph
  ggtitle(paste0("Average bias for samples of size N=",
                 min(N), " to N=", max(N)))
```

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.