Lab 3 - Computational Statistics (732A90)

Julius Kittler (julki092), Vinay Bengaluru Ashwath Narayan Murthy (vinbe289)

February 13, 2018

Contents

1 Assignment 1: Cluster sampling **1**
1.1 Task 1: Data import 1
1.2 Task 2: Generating discrite RVs with Unif(0, 1) 2
1.3 Task 3 and 4: Generating discrite RVs with Unif(0, 1) - multiple times 3
1.4 Task 5: Histogram of sample space vs. sampled instances 4

2 Assignment 2: Different distributions **5**
2.1 Task 1: Inverse CDF method (for DE(0, 1)) 5
2.2 Task 2: Acceptance/rejection method (for N(0, 1) using DE(0, 1)) 9

3 Appendix **13**

1 Assignment 1: Cluster sampling

1.1 Task 1: Data import

First, the data from `population.csv` is imported into R. This file contains the number of inhabitants by city, which is all the information we need.

Note: A variable `Rank` is added, representing the rank order of the city w.r.t. `Population`. Furthermore, the top 10 cities with the largest amount of inhabitants are shown below.

	Municipality	Population	Rank
16	Stockholm	829417	1
146	Göteborg	507330	2
112	Malmö	293909	3
32	Uppsala	194751	4
47	Linköping	144690	5
221	Västerås	135936	6
211	Örebro	134006	7
50	Norrköping	129254	8
101	Helsingborg	128359	9
62	Jönköping	126331	10

1.2 Task 2: Generating discrite RVs with Unif(0, 1)

Now, we use the uniform random number generator `runif()` to create a function that selects `n` cities from the whole list by the following probability scheme: *random sampling without replacement is done with the probabilities proportional to the number of inhabitants of the city to select 20 cities.*

More specifically, the function returns the indices of the `n` cities (to make the function more general and re-usable for the next task). Note that sorting does not really matter here since it does not change the interval lengths for the cities in the range from 0 to 1. Consequently, sorting would not change the probability any city getting selected since it is represented with an interval of the same length in the range from 0 to 1, just at a different position. Therefore, the code for sorting was commented out.

```
random_sampling = function(x, n = 20){

  # Input:
  # - vector x, with counts that will be converted to probabilities
  # - integer n, representing the number of indices to be sampled

  # Output: Indices of n cities that were randomly sampled (with their
  # probabilites proportional to the number of inhabitants from x)

  # Prepare and modify objects ------------------------------------------------

  U = runif(n) # Generate n random numbers from Uniform dist with [0, 1]
  out = integer(n) # Prepare vector with randomly sampled integers

  idx = 1:length(x)   # Create indices

  # Sorting does not really matter here because we don't have very small values
  # idx = idx[order(x)] # Sort indices according values of x ascendingly
  # x = sort(x)         # Sort x ascendingly

  # Obtain randomly sampled indices -------------------------------------------

  for (i in 1:n){

    # Obtain first index i at which U <= p_0 + ... p_i
    P_cum = cumsum(x / sum(x)) # Compute cumulative probabilities
    idx_sampled = min(which(U[i] <= P_cum))
    out[i] = idx[idx_sampled]

    # Remove this value from x and from idx
    x = x[-idx_sampled]
    idx = idx[-idx_sampled]

  }
```

```
  # Return results
  return(out)
}
```

This function is run for `n = 1` to sample just one city. We set the seed `12345` to make the result reproducible.

```
set.seed(12345)
idx = random_sampling(df$Population, n = 1)
cat("Sampled city: ", as.character(df$Municipality[idx]),"\n")
cat("Inhabitants (number): ", df$Population[idx],"\n")
cat("Inhabitants (rank): ", df$Rank[idx],"\n")
```

```
## Sampled city:  Jönköping
```

```
## Inhabitants (number):  126331
```

```
## Inhabitants (rank):  10
```

1.3 Task 3 and 4: Generating discrite RVs with Unif(0, 1) - multiple times

Here, we can re-use the function from the previous task. The only difference is that the parameter `n` is set to 20. Again, we set the seed `12345` to make the result reproducible.

Selected Cities (Sorted by Population)

	Municipality	Population	Rank
16	Stockholm	829417	1
146	Göteborg	507330	2
112	Malmö	293909	3
47	Linköping	144690	5
62	Jönköping	126331	10
272	Umeå	114075	11
238	Gävle	94352	17
107	Kristianstad	78788	24
177	Uddevalla	51518	43
287	Piteå	40860	53
17	Sundbyberg	37722	64
41	Strängnäs	32024	76
2	Danderyd	31150	79
91	Karlshamn	30918	80
59	Gislaved	29212	84
105	Höör	15261	146
96	Bjuv	14813	153
196	Sunne	13345	165
52	Vadstena	7420	245

	Municipality	Population	Rank
263	Dorotea	2900	288

Size of cities

In general, we would expect the mean `Population` of the selected cities to be larger than the overall mean `Population` (since cities with larger number of residents are should be selected with a larger probability). The printout below shows that this is indeed the case.

```
## Overall mean :          32209.25
```

```
## Mean of sampled 20:  124801.75
```

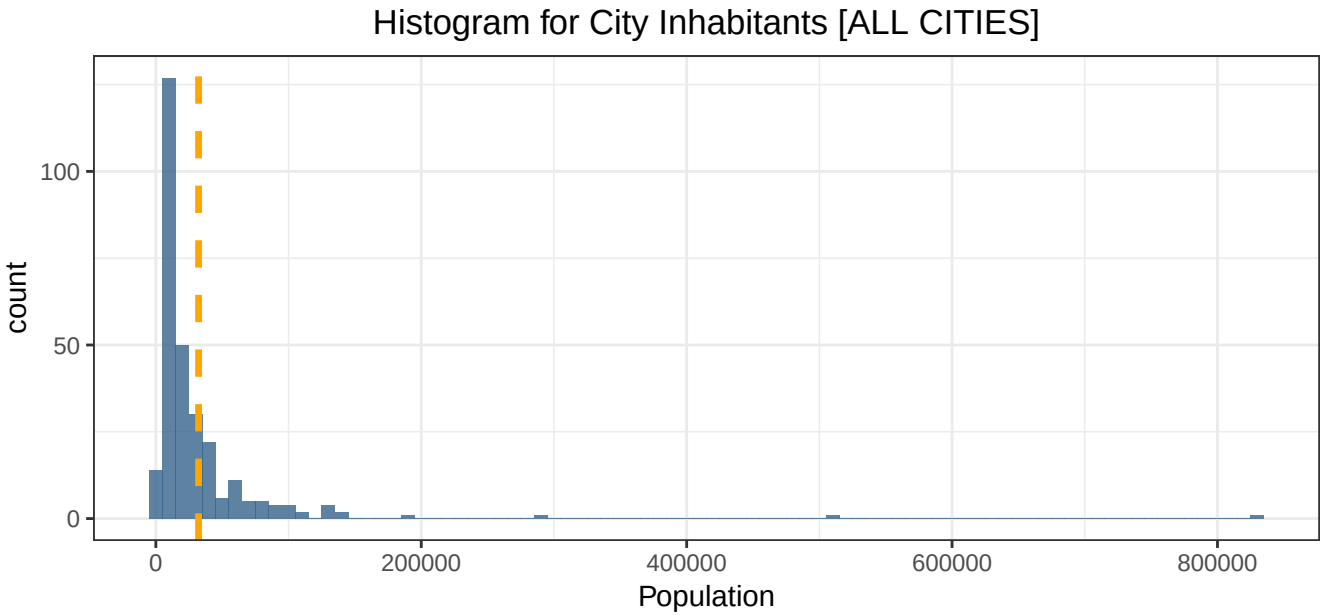
1.4 Task 5: Histogram of sample space vs. sampled instances

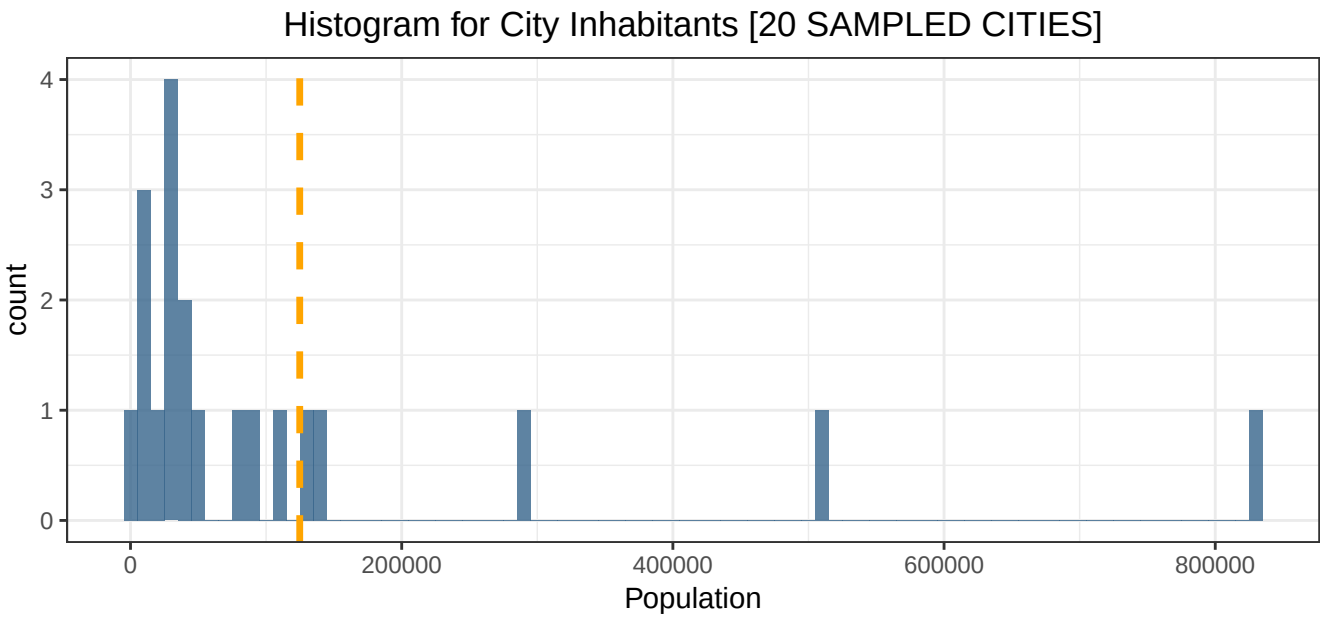
Here, we create two histograms: - one histogram showing the size of all cities of - one histogram showing the size of the 20 sampled cities

Conclusions

It is very visible that in the histogram for the 20 sampled cities, we have more cities with very large number of inhabitants than for the histogram with all cities. There are three bars in the histogram for the 20 sampled cities (at ca. 300k, 500k and 800k) that illustrate this. Furthermore, the mean is displayed as a vertical orange line and we can see that it is noticeably larger in the histogram for the 20 sampled cities.

We can conclude that the sampling procedure works as expected because it samples cities with more residents with a larger probability.





2 Assignment 2: Different distributions

We are given the double exponential (Laplace) distribution:

$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{-\alpha|x-\mu|}$$

2.1 Task 1: Inverse CDF method (for DE(0, 1))

Here, a number generator for the double exponential distribution $DE(0, 1)$ is created from $Unif(0, 1)$ by using the inverse CDF method. To use the CDF method, the following steps are required:

- Compute CDF (returning P(X <= x) given $x \sim DE(0, 1)$)
- Compute inverse of CDF (returning $x \sim DE(0, 1)$ given $P(X <= x)$) from $Unif(0, 1)$)

2.1.1 Compute CDF

We want to compute the following integral:

$$P(X <= x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha|x-\mu|} dx$$

Two cases need to be considered because of the absolute value term in the exponent. Depending on the case, a different integral needs to be computed.

- **a)** $x >= \mu$
- **b)** $x < \mu$

CDF when $x < \mu$

$$P(X <= x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{\alpha(x-\mu)} dx$$

Note that $x < \mu$ implies that $-\alpha|x-\mu| = -\alpha(\mu - x)$ which can be rewritten as $-\alpha\mu - (-\alpha)x = -\alpha\mu + \alpha x = \alpha(x-\mu))$.

We apply u substitution, letting $u = \alpha(x-\mu))$ and $du = \alpha dx$. Afterwards, the integral is simple to evaluate.

$$\int_{-\infty}^{x} \frac{1}{2} e^{u} dx = \left[\frac{1}{2} e^{u}\right]_{-\infty}^{x} = \left[\frac{1}{2} e^{\alpha(x-\mu)}\right]_{-\infty}^{x} = \frac{1}{2} e^{\alpha(x-\mu)} - (-\frac{1}{2} e^{-\infty}) = \frac{1}{2} e^{\alpha(x-\mu)} - 0 = \frac{1}{2} e^{\alpha(x-\mu)}$$

CDF when $x >= \mu$

$$P(X <= x) = \int_{\mu}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)} dx + \int_{-\infty}^{\mu} \frac{\alpha}{2} e^{\alpha(x-\mu)} dx$$

For the first integral, we apply u substitution, letting $u = -\alpha(x-\mu)$ and $du = -\alpha dx$, i.e. replacing α with -1. Afterwards, the integral is simple to evaluate.

$$\int_{\mu}^{x} -\frac{1}{2} e^{u} dx = \left[-\frac{1}{2} e^{u}\right]_{\mu}^{x} = \left[-\frac{1}{2} e^{-\alpha(x-\mu)}\right]_{\mu}^{x} = -\frac{1}{2} e^{-\alpha(x-\mu)} - (-\frac{1}{2} e^{0}) = \frac{1}{2} - \frac{1}{2} e^{-\alpha(x-\mu)}$$

Note that since we require $x >= \mu$ for this case, the lower limit was changed from $-\infty$ to μ. Furthermore, note that $x >= \mu$ implies that $|x-\mu| = (x-\mu)$.

However, we must also consider all cases when $x < \mu$ to obtain a proper CDF. For this part, we can however use the previously computed CDF for $x < \mu$:

$$P(X <= \mu) = \frac{1}{2} e^{\alpha(x-\mu)} = \frac{1}{2} e^{\alpha(\mu-\mu)} = \frac{1}{2} e^{0} = \frac{1}{2}$$

Therefore, the CDF for $x >= \mu$ is:

$$P(X <= x) = \frac{1}{2} - \frac{1}{2} e^{-\alpha(x-\mu)} + \frac{1}{2} = 1 - \frac{1}{2} e^{-\alpha(x-\mu)}$$

2.1.2 Compute Inverse of CDF

Now, we compute the inverse for both cases because we want to be able to return x from $DE(0,1)$ given $P(X <= x)$, where $P(X <= x)$ is obtained from $Unif(0,1)$. We use the CDFs computed in the previous step and solve for x.

Inverse of CDF when $x < \mu$

$$y = \frac{1}{2} e^{\alpha(x-\mu)}$$

$$2y = e^{\alpha(x-\mu)}$$
$$log(2y) = \alpha(x-\mu)$$
$$\frac{1}{\alpha}log(2y) = x-\mu$$
$$x = \frac{1}{\alpha}log(2y)+\mu$$

Inverse of CDF when $x >= \mu$

$$y = 1-\frac{1}{2}e^{-\alpha(x-\mu)}$$
$$y-1 = -\frac{1}{2}e^{-\alpha(x-\mu)}$$
$$-2y+2 = e^{-\alpha(x-\mu)}$$
$$log(-2y+2) = -\alpha(x-\mu)$$
$$-\frac{1}{\alpha}log(-2y+2) = x-\mu$$
$$x = -\frac{1}{\alpha}log(-2y+2)+\mu$$

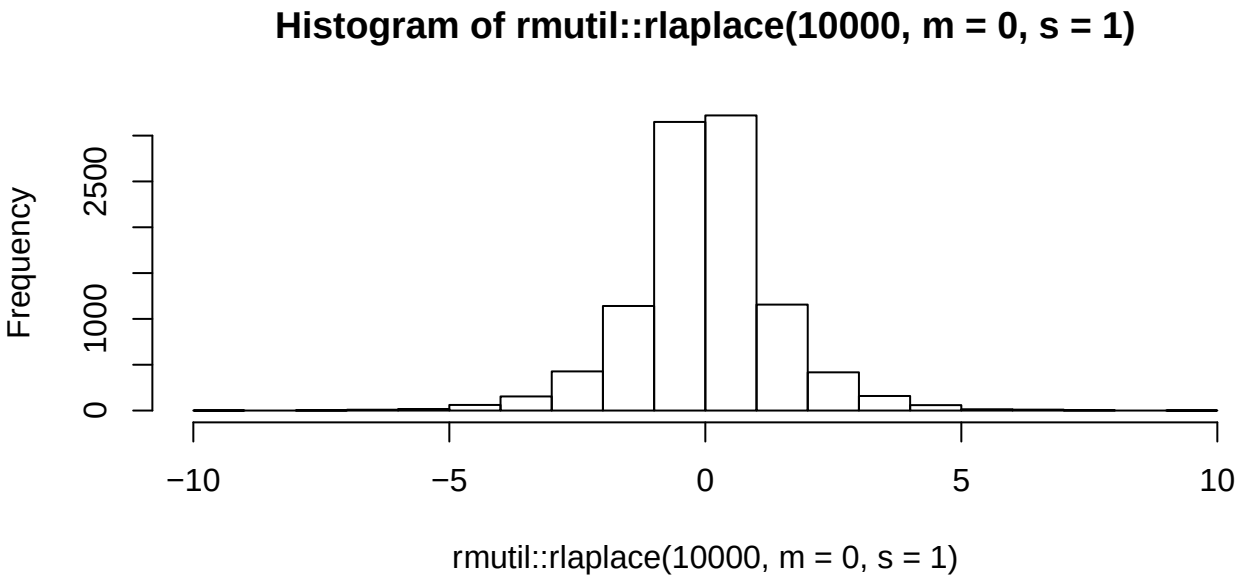
2.1.3 Evaluate Results

First, we generate 10000 values from $Unif(0,1)$. Then, we input these values as y into the inverse of the CDF to obtain the values from $DE(0,1)$.

Expected Result

The following is what we would expect our final result to look like.

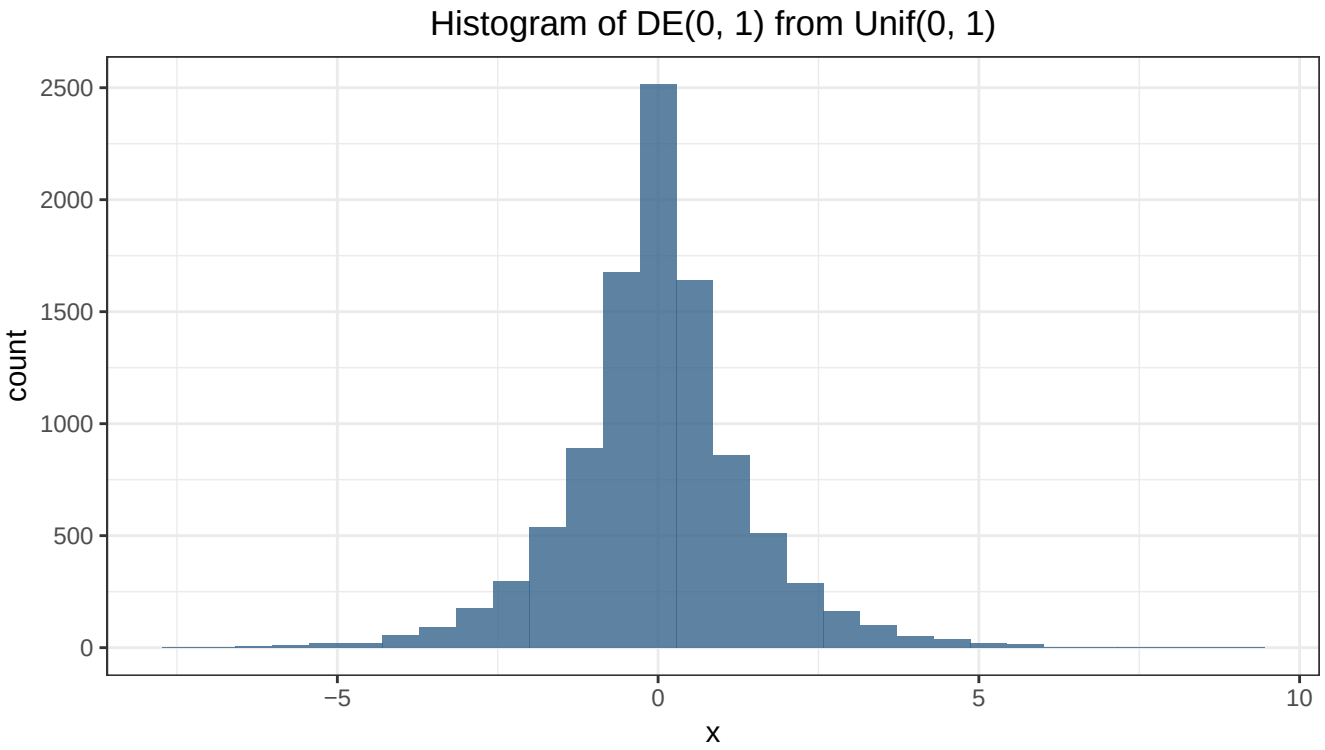
```
hist(rmutil::rlaplace(10000, m = 0, s = 1))
```



Actual Result

As we can see, our result returned from the inverse CDF method resembles the expected result very closly. Therefore, the solution looks reasonable.

```r
DE = function(n = 10000){

  # Set parameters
  alpha = 1
  mu = 0

  # Sample from Unif(0, 1)
  y = runif(n)

  # Convert to DE(0, 1)
  x = ifelse(- 1/alpha * log(-2 * y + 2) + mu >= mu,
             - 1/alpha * log(-2 * y + 2) + mu, # Option 1: x >= mu
             1/alpha * log(2 * y) + mu) # Option 2: x < mu

  # ALTERNATIVE WITH SIGN FUNCTION
  # x = mu - 1/alpha * sign(y - 0.5) * log(1 - 2 * abs(y - 0.5))

  return(x)
}

x = DE()
df_plot = data.frame(x = x)

ggplot(df_plot, aes(x = x)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8) +
  labs(title = "Histogram of DE(0, 1) from Unif(0, 1)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```



2.2 Task 2: Acceptance/rejection method (for N(0, 1) using DE(0, 1))

Here, we use the Acceptance/rejection method with $DE(0,1)$ as a majorizing density to generate $N(0,1)$ variables. We generate 2000 random numbers $N(0,1)$ using our code and plot the histogram. Afterwards, we generate 2000 numbers from $N(0,1)$ using standard `rnorm()` procedure, plot the histogram and compare the obtained two histograms.

2.2.1 Histogram with Acceptance/rejection Method

For the acceptanc/rejection method, we need the probability density functions for $N(0,1)$ and for $DE(0,1)$. Therefore, we implement them as follows.

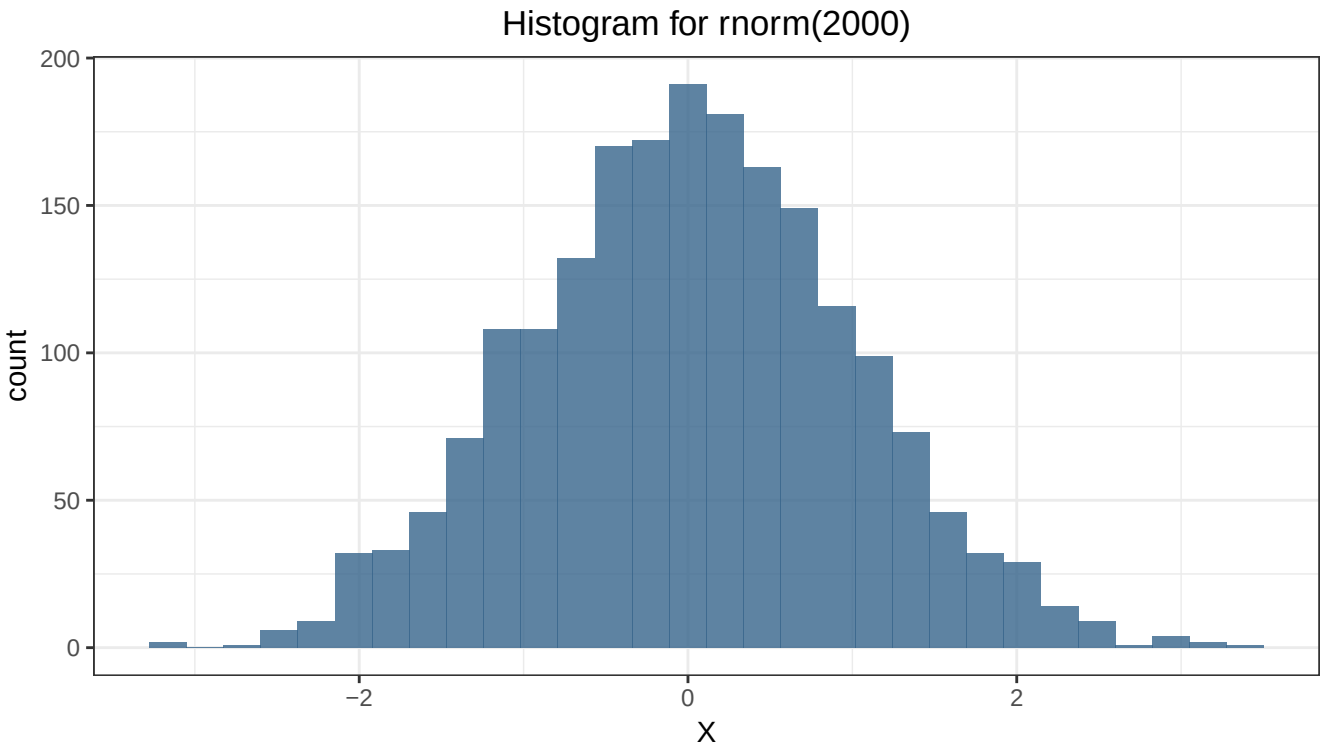
```
DE_pdf = function(x,  mu = 0, alpha = 1){

  # Compute and return density
  density = alpha/2 * exp(-alpha * abs(x - mu))
  return(density)

}

N_pdf = function(x, mu = 0, sigma = 1){

  # Compute and return density
  density = 1/(sqrt(2*pi*sigma^2)) * exp(- (x - mu)^2 / (2 * sigma^2))
  return(density)
```

```
}
```

The implementation of the acceptance/rejection method can be found below. The choice of the parameter c is explained in the last section of this lab.

```
acceptance_rejection = function(n = 2000){

  # Set parameters
  out = numeric(n)
  c = sqrt(2*exp(1)/pi)
  cnt_accepted = integer(n)
  cnt_rejected = integer(n)

  # Generate values 1 by 1
  for (i in 1:n){

    X_not_generated = TRUE

    while (X_not_generated){

      Y = DE(1)
      U = runif(1)

      if (U <= (N_pdf(Y) / (c * DE_pdf(Y)))){

        out[i] = Y
        X_not_generated = FALSE
        cnt_accepted[i] = cnt_accepted[i] + 1

      } else {

        cnt_rejected[i] = cnt_rejected[i] + 1

      }

    }

  }

  # Return output
  avg_rejection_rate = sum(cnt_rejected) / sum(cnt_rejected + cnt_accepted)
  return(list(X = out, avg_rejection_rate = avg_rejection_rate))

}
```

The histogram of 2000 points sampled with the acceptance/rejection method looks as follows.



Average rejection rate R in the acceptance/rejection method

The average rejection rate is printed out below. We can see that it is rather low which is good since it allows for a relatively fast speed of the acceptance/rejection method.

```
## Average rejection rate:  0.2236
```

Expected rejection rate E[R] and comparison to R

The expected rejection rate depends on the parameter `c`. If `c` is larger, the rejection rate will also be larger. Therefore, we want `c` to be as small as possible.

The number of required draws until the first acceptance follows a Geometric distribution with mean `c`: number of draws until first acceptance $\sim Geometric(1/c)$.

In our case, `c = sqrt(2*exp(1)/pi) = 1.3155`. Note that 1 of this was for the last iteration, which got accepted. Therefore, the theoretical rejection rate is: `(1.3155 - 1) / 1.3155 = 0.2398`. Interestingly, the observed rejection rate is very similar to the theoretical rejection rate.

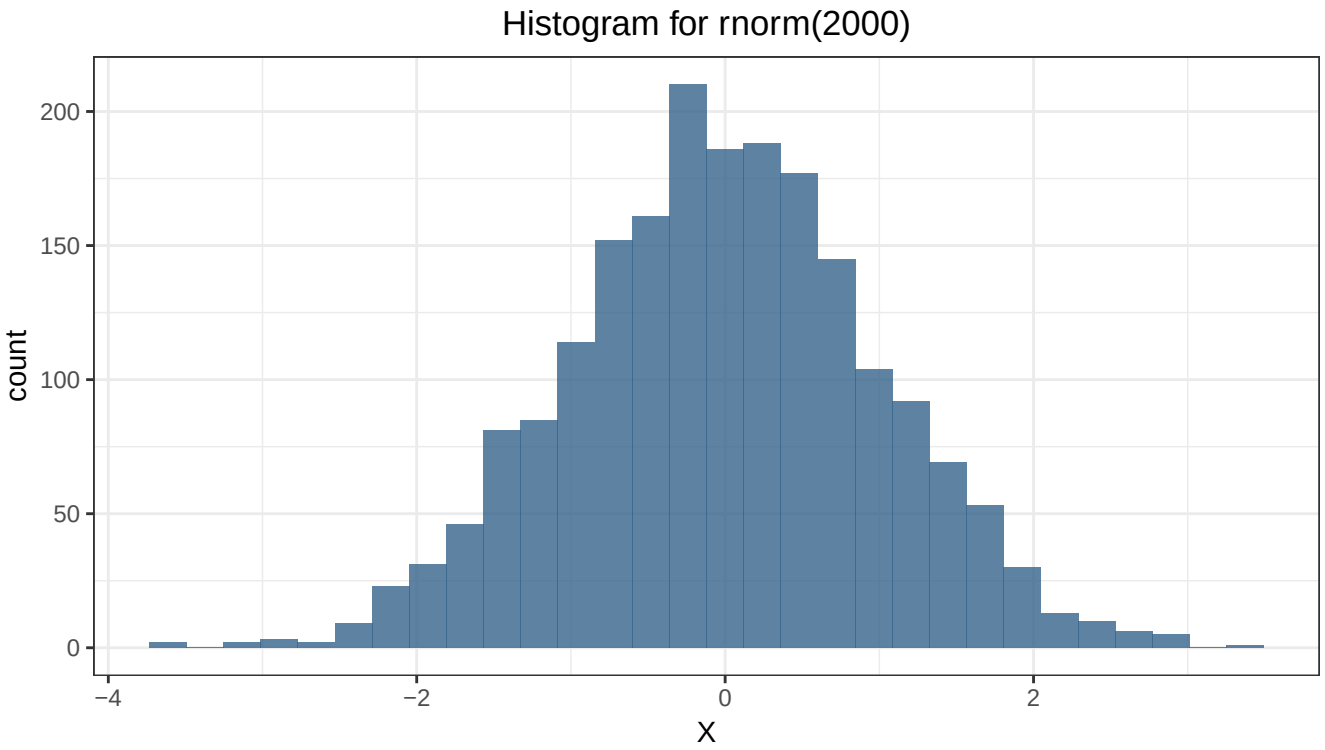
2.2.2 Histogram with rnorm() Function

The histogram of 2000 points sampled with `rnorm()` looks as follows. As we can see, the two histograms look very similar. Note that some differences between the histograms may be explained by the randomness (due to the use of `runif()` for the acceptance/rejection method and `rnorm()` for the other method). By eye, it would be very difficult to make a guess which histogram belongs to which method.

Conclusion

The acceptance/rejection method seems like an appropriate method to generate values from $N(0, 1)$

in our case. Note that it would be difficult to make a comparison with the inverse CDF method because we cannot obtain a closed form solution for the CDF of the normal distribution.



2.2.3 Choice of the paramter c

Choice of c

We are given in the lecture slides that we need to find a constant c that fulfills the following condition for all values of x:

$$c \cdot f_Y(x) \geq f_X(x)$$
$$c \geq \frac{f_X(x)}{f_Y(x)}$$

In order to find a c that fulfills this condition, we need to find the maximum of the following expression and make c equal to the value of this expression at the maximum. Note that we do not want to make c even larger because a larger value of c can cause a larger rejection rate and hence a larger number of required draws.

$$h(x) = \frac{f_X(x)}{f_Y(x)} = \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}{\frac{\alpha}{2}\exp(-\alpha|x-\mu|)} = \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)}{\frac{1}{2}\exp(-\sqrt{x^2})} = \frac{2}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \exp\left(\sqrt{(x^2)}\right)$$
$$= \frac{2}{\sqrt{2\pi}} \exp\left(\sqrt{(x^2)} - \frac{x^2}{2}\right) = \frac{2}{\sqrt{2\pi}} e^{\left(\sqrt{(x^2)} - \frac{x^2}{2}\right)}$$

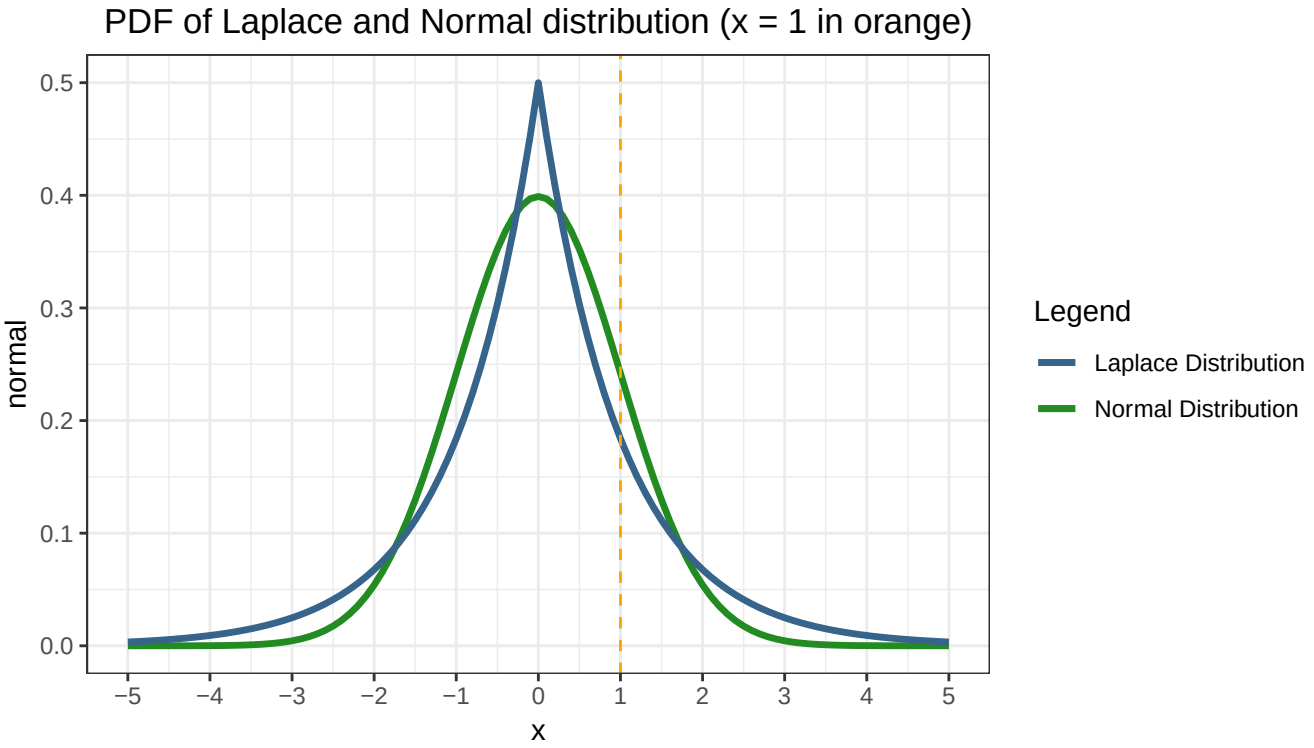
Our goal is to find:

$$\max\left(\frac{f_X(x)}{f_Y(x)}\right)$$

To find the maximum, we take the first derivative, set it equal to zero and solve for x:

$$h(x)' = \frac{\partial h(x)}{\partial x} = \frac{\partial}{\partial x}\left(\frac{2}{\sqrt{2\pi}}e^{\left(\sqrt{(x^2)}-\frac{x^2}{2}\right)}\right) = \frac{2}{\sqrt{2\pi}}e^{\left(\sqrt{(x^2)}-\frac{x^2}{2}\right)}\cdot(1-x) = 0$$

We can see that the above expression is true when x = 1. We know that this must be a maximum because of the shape of the normal distribution and the double exponential distribution:



Lastly, we obtain the value of c by pluggin x into h(x):

$$h(1) = \frac{2}{\sqrt{2\pi}}e^{\left(\sqrt{(1^2)}-\frac{1^2}{2}\right)} = \frac{2}{\sqrt{2\pi}}\sqrt{e} = 2\sqrt{\frac{e}{2\pi}} = \sqrt{\frac{2^2e}{2\pi}} = \sqrt{\frac{2e}{\pi}} = c$$

3 Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE,
```

```
                      cache = TRUE, cache.path = "cache/", fig.path = "cache/")

# options(digits=22)
options(scipen=999)

library(dplyr)
library(ggplot2)
library(magrittr)

set.seed(12345)


# ------------------------------------------------------------------------------
# Assignment 1, Task 1
# ------------------------------------------------------------------------------

df = read.csv2("population.csv", fileEncoding="ISO-8859-1")
df$Rank = rank(-df$Population)

knitr::kable(head(df[order(df$Population, decreasing = TRUE), ], 10))


# ------------------------------------------------------------------------------
# Assignment 1, Task 2
# ------------------------------------------------------------------------------


random_sampling = function(x, n = 20){

  # Input:
  # - vector x, with counts that will be converted to probabilities
  # - integer n, representing the number of indices to be sampled

  # Output: Indices of n cities that were randomly sampled (with their
  # probabilites proportional to the number of inhabitants from x)

  # Prepare and modify objects ------------------------------------------------

  U = runif(n) # Generate n random numbers from Uniform dist with [0, 1]
  out = integer(n) # Prepare vector with randomly sampled integers

  idx = 1:length(x)   # Create indices

  # Sorting does not really matter here because we don't have very small values
  # idx = idx[order(x)] # Sort indices according values of x ascendingly
  # x = sort(x)         # Sort x ascendingly
```

```r
  # Obtain randomly sampled indices ------------------------------------------

  for (i in 1:n){

    # Obtain first index i at which U <= p_0 + ... p_i
    P_cum = cumsum(x / sum(x)) # Compute cumulative probabilities
    idx_sampled = min(which(U[i] <= P_cum))
    out[i] = idx[idx_sampled]

    # Remove this value from x and from idx
    x = x[-idx_sampled]
    idx = idx[-idx_sampled]

  }

  # Return results
  return(out)
}


set.seed(12345)
idx = random_sampling(df$Population, n = 1)
cat("Sampled city: ", as.character(df$Municipality[idx]),"\n")
cat("Inhabitants (number): ", df$Population[idx],"\n")
cat("Inhabitants (rank): ", df$Rank[idx],"\n")


set.seed(12345)
idx = random_sampling(df$Population, n = 1)
cat("Sampled city: ", as.character(df$Municipality[idx]),"\n")
cat("Inhabitants (number): ", df$Population[idx],"\n")
cat("Inhabitants (rank): ", df$Rank[idx],"\n")


# ------------------------------------------------------------------------------
# Assignment 1, Task 3 and 4
# ------------------------------------------------------------------------------

set.seed(12345)
idx = random_sampling(df$Population, n = 20)
df_sub = df[idx, ]

knitr::kable(df_sub[order(df_sub$Population, decreasing = TRUE), ])


cat("Overall mean :        ", round(mean(df$Population), 2), "\n")
```

```r
cat("Mean of sampled 20: ", round(mean(df_sub$Population), 2), "\n")


# ------------------------------------------------------------------------
# Assignment 1, Task 5
# ------------------------------------------------------------------------

ggplot(df, aes(x = Population)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8, binwidth = 10000) +
  geom_vline(xintercept = mean(df$Population), color = "orange",
             linetype = "dashed", size = 1.2) +
  labs(title = "Histogram for City Inhabitants [ALL CITIES]") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))

ggplot(df_sub, aes(x = Population)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8, binwidth = 10000) +
  geom_vline(xintercept = mean(df_sub$Population), color = "orange",
             linetype = "dashed", size = 1.2) +
  labs(title = "Histogram for City Inhabitants [20 SAMPLED CITIES]") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------
# Assignment 2, Task 1
# ------------------------------------------------------------------------


hist(rmutil::rlaplace(10000, m = 0, s = 1))


DE = function(n = 10000){

  # Set parameters
  alpha = 1
  mu = 0

  # Sample from Unif(0, 1)
  y = runif(n)

  # Convert to DE(0, 1)
  x = ifelse(- 1/alpha * log(-2 * y + 2) + mu >= mu,
             - 1/alpha * log(-2 * y + 2) + mu, # Option 1: x >= mu
             1/alpha * log(2 * y) + mu) # Option 2: x < mu

  # ALTERNATIVE WITH SIGN FUNCTION
  # x = mu - 1/alpha * sign(y - 0.5) * log(1 - 2 * abs(y - 0.5))
```

```r
  return(x)
}

x = DE()
df_plot = data.frame(x = x)

ggplot(df_plot, aes(x = x)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8) +
  labs(title = "Histogram of DE(0, 1) from Unif(0, 1)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------------
# Assignment 2, Task 2
# ------------------------------------------------------------------------------


DE_pdf = function(x,  mu = 0, alpha = 1){

  # Compute and return density
  density = alpha/2 * exp(-alpha * abs(x - mu))
  return(density)

}

N_pdf = function(x, mu = 0, sigma = 1){

  # Compute and return density
  density = 1/(sqrt(2*pi*sigma^2)) * exp(- (x - mu)^2 / (2 * sigma^2))
  return(density)

}


acceptance_rejection = function(n = 2000){

  # Set parameters
  out = numeric(n)
  c = sqrt(2*exp(1)/pi)
  cnt_accepted = integer(n)
  cnt_rejected = integer(n)

  # Generate values 1 by 1
  for (i in 1:n){

    X_not_generated = TRUE
```

```
    while (X_not_generated){

      Y = DE(1)
      U = runif(1)

      if (U <= (N_pdf(Y) / (c * DE_pdf(Y)))){

        out[i] = Y
        X_not_generated = FALSE
        cnt_accepted[i] = cnt_accepted[i] + 1

      } else {

        cnt_rejected[i] = cnt_rejected[i] + 1

      }

    }

  }

  # Return output
  avg_rejection_rate = sum(cnt_rejected) / sum(cnt_rejected + cnt_accepted)
  return(list(X = out, avg_rejection_rate = avg_rejection_rate))

}


res = acceptance_rejection()
df_plot = data.frame(X = res$X)

ggplot(df_plot, aes(x = X)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8) +
  labs(title = "Histogram for rnorm(2000)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


cat("Average rejection rate: ", round(res$avg_rejection_rate, 4), "\n")


X = rnorm(2000)
df_plot = data.frame(X = X)

ggplot(df_plot, aes(x = X)) +
  geom_histogram(fill = "steelblue4", alpha = 0.8) +
  labs(title = "Histogram for rnorm(2000)") +
```

```r
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


df_plot = data.frame(laplace = DE_pdf(seq(-5, 5, 0.1)),
                     normal = N_pdf(seq(-5, 5, 0.1)),
                     x = seq(-5, 5, 0.1))

ggplot(df_plot, aes(x = x)) +
  geom_line(aes(y = normal, color = "Normal Distribution"), size = 1.2) +
  geom_line(aes(y = laplace, color = "Laplace Distribution"), size = 1.2) +
  geom_vline(aes(), xintercept = 1, color = "orange", linetype = "dashed") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
  labs(title = "PDF of Laplace and Normal distribution (x = 1 in orange)") +
  scale_color_manual(name ="Legend", values = c("steelblue4", "forestgreen")) +
  scale_x_continuous(breaks = seq(-5, 5, 1))
```