Computer Lab 1 Computational Statistics - Report

Contents

Question 1: Be careful when comparing

Consider the following two R code snippets

```
# snippets 1
x1 = 1/3 ; x2 = 1/4
if(x1 - x2 == 1/12){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
```

```
## [1] "Subtraction is wrong"
```

```
# snippets 2
x1 = 1 ; x2 = 1/2
if(x1 - x2 == 1/2){
print("Subtraction is correct")
```

```
} else{
print("Subtraction is wrong")
}
```

```
## [1] "Subtraction is correct"
```

1. Check the results of the snippets. Comment what is going on.

We can see that the output of the two snippets is different.

- 1- Snippet: x_1 is stored with the value $1/3$, which is a consecutive number in decimal form $(0.33333\ldots)$. Especially floating point number are not stored with the "correct" value, but one that is very similar to the "real" value. This is due to the memory capacity of numbers in 32 or 64 bits. Thus information is lost due to regression, which leads to a rounding error. This is also called underflow.
- 2 - snippet: In this case the output shows that the calculation is correct. In this case, no rounding error occurred due to memory optimization.

2. If there are any problems, suggest improvements.

```
# snippets 1 - improved
x1 = 1/3 ; x2 = 1/4
if(isTRUE(all.equal(x1 - x2, 1/12))){
  print("Subtraction is correct")
} else{
  print("Subtraction is wrong")
}
```

```
## [1] "Subtraction is correct"
```

One way to fix the output error of snippet 1 is to use the function *all.equal*. This function tests if two objects are "near equality". We can now see from the output that the calculation is correct.

Question 2: Derivative

From the defintion of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula:

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

1. Write your own R function

to calculate the derivative of $f(x) = x$ in this way with $\epsilon = 10^{-15}$

The function:

```
epsilon = 10**-15
derivate = function(x){
  deriv = ((x + epsilon) - x) / epsilon
  return(deriv)
}
```

2. Evaluate your derivative function at x = 1 and x = 100000.

```
x1= 1
x2= 100000
derivate(x1)
```

```
## [1] 1.110223
```

```
derivate(x2)
```

```
## [1] 0
```

3. What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

Since $f(x) = x$, the true value should be 1. The reason why the results are wrong might be:

When $x_1 = 1$, $x_2 = 10^{-15}$ exceeds the expression ability of R. It becomes $1 + 1.110^{-15} \neq 1 + 10^{-15}$, which is also wrong and underflow. Then the numerator thereby becomes

$$1 + 1.1 * 10^{-15} - 1 \approx 1.110223^{-15} \neq 1.110^{-15}$$

(underflow again) with the result $1.110223 \neq 1$.

When $x = 10^5$, 10^{-15} is too small compared with x. So it is underflow and happens that $10^5 + 10^{-15} = 10^5$. The numerator becomes 0 and so does the derivative.

Question 3: Variance

A known formula for estimating the variance based on a vector of n observations is

$$\text{Var}(\vec{x}) = \frac{1}{n-1}(\sum_{i=i}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2)$$

1. Write your own R function, myvar, to estimate the variance in this way.

```
# write the preseted var function - input vector x
myvar = function(x){
  n = length(x)
  var_calculation = (1/(n-1)) * (sum(x^2)-(1/n)*(sum(x)^2))
  return(var_calculation)
}
```

2. Generate a vector x = (x1, . . . , x10000) with 10000 random numbers with mean 10^8 and variance 1.

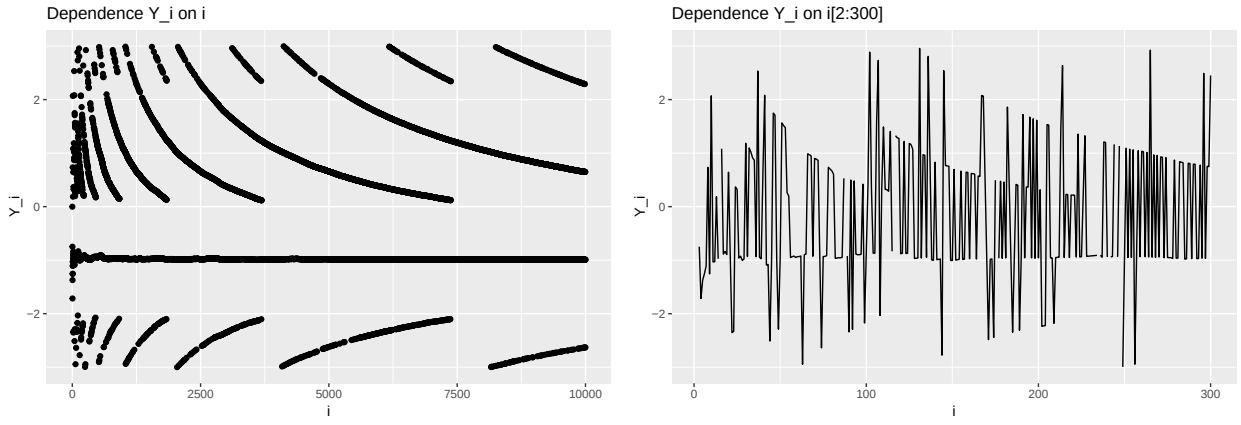
```
# initialize vector
n = 10000
x_vector = rnorm(n, mean = 10^8, sd = 1)
```

3. Compute the difference $Y_i = myvar(X_i) - var(X_i)$

For each subset Xi ={x1,. . . ,xi}, i=1,. . . ,10000 compute the difference $Y_i = myvar(Xi) - var(Xi)$, where $var(X_i)$ is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?

```
# try - 2
y = c()
for (i in 1:length(x_vector)) {
  y[i] = myvar(x_vector[1:i]) - var(x_vector[1:i])
}
# remove the first value - variance of one pont can??t be calculated - does not make sense
y = y[-1]

# create data framem for the plot
data_dependence_Y_i = data.frame(y_value = y,
                                 iteration = 1:length(y))
# create plot dependence Yi on i
plot_dependence_Y_i =
  ggplot(data = data_dependence_Y_i) +
  geom_point(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)

plot_dependence_Y_i_2to300 =
  ggplot(data = data_dependence_Y_i[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i[2:300]") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)
```

The plots:



At the y-axis we can recognize the Y_i, which can be calculated as follows: Yi = myvar(Xi)- var(Xi). On the x-axis there is the subset with the respective size i. If the functions would output the same output, then all points would be distributed on the 0 horizontal axis. However, we can't see this course. At the y_i value of about -1 there is a continuous line. In the upper part of the visualization it can be seen that the value runs recurrently towards 0. Similar approximate progression can also be seen in the lower part, but against the previously mentioned value of about -1. Since the initialize vector is a very small number, the calculation with the *var()* function may handle these values better than the *myvar* function.

It is obvious that the difference between estimated variance and real variance increases regularly, so the result is not good enough. The reason might be that $\sum_{i=1}^{n} x^2 \approx 10^{20}$ and $(\sum_{i=1}^{n} x)^2 \approx 10^{24}$ are two large numbers.

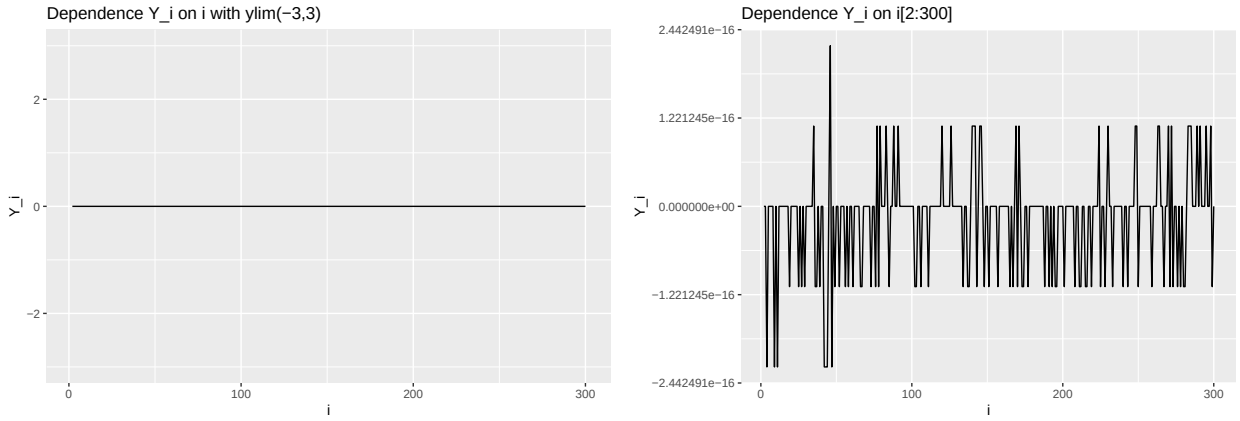
So it is overflow intermittently.

4. Better implementation of variance estimator

How can you better implement a variance estimator? Find and implement a formula that will give the same results as var()?

```
##3.4#################################################################
myvar1 <- function(x){
  m <- mean(x)
  sum((x-m)**2)/(length(x)-1)
}
y1 <- sapply(1:n, function(i){
  myvar1(x_vector[1:i])-var(x_vector[1:i]) # changed xbar to x_vector
})
y1 = y1[-1]
data_dependence_Y_i_improved = data.frame(y_value = y1,
                                  iteration = 1:length(y1))

plot_dependence_Y_i_2to300_improved_scale3 =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i with ylim(-3,3)") + ylab("Y_i") + xlab("i")+
  ylim(-3,3)

plot_dependence_Y_i_2to300_improved =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i[2:300]") + ylab("Y_i") + xlab("i")
```

Improved plot:



$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Here we use the sample variance formula, which is an unbias estimate of real variance for a finite sequence. Evidently, the result is better and the differnces between estiamted and real variances are much smaller than the previous formula.

Question 4: Linear Algebra

The Excel file "tecator.xls" contains the results of a study aimed to investigate whether a near infrared absorbance spectrum and the levels of moisture and fat can be used to predict the protein content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. The worksheet you need to use is "data"" (or file "tecator.csv"). It contains data from 215 samples of finely chopped meat. The aim is to fit a linear regression model that could predict protein content as function of all other variables.

1. Import the data set to R

```
##4.1#################################################################
data <- as.matrix(read.csv("tecator.csv"))
data <- data[,-1]
X <- data[,-102]
Y <- data[,102]
```

2. Compute optimal regression coefficients

Optimal regression coefficients can be found by solving a system of the type $A\ \vec{\beta} = \vec{b}$ where $A = X^T X$ and $\vec{b} = X^T \vec{y}$ for the given data set. The matrix X are the observations of the absorbance records, levels of moisture and fat, while $\vec{y}$ are the protein levels.

```
##4.2#################################################################
Xe0 <- cbind(1,X)
A0 <- t(Xe0)%*%Xe0
b0 <- t(Xe0)%*%Y
```

3. Solve with default solver *solve()*

Try to solve $A\ \vec{\beta} = \vec{b}$ with default solver solve(). What kind of result did you get? How can this result be explained?

```
##4.3#################################################################
tryCatch(solve(A0,b0),
         error=function(e){cat("ERROR :\n",conditionMessage(e),"\n")})
```

"Error in solve.default(A, b_vector) : system is computationally singular: reciprocal condition number = 7.13971e-17"

An error occurs when the function *solve* is used. The reciprocal of condition number of matrix A is too small, and such value represents the tolerance *tol* for detecting linear dependencies in the columns of A. So some columns of A is linear dependent within default tolerance, which means the matrix A is not invertible here.

4. Check the condition number of the matrix A (function kappa()) and consider how it is related to your conclusion in step 3.

```
##4.4#################################################################
beta0 <- solve(A0,b0,tol = 7.78804e-17)
kappa(A0)
```

```
## [1] 8.523517e+14
```

We still make the model by such A and b with smaller tolerance. The function *kappa* can provide us the condition number of matrix A, which follows the following formula:

$$\kappa(A) = ||A|| * ||A^{-1}||.$$

It is to measure for a matrix A of solving a linear-equation system and show the significance of scaling data. The larger the condition number is, the worse matrix A does. Because the tolerance would be much smaller, when the condition number increases.

5. Scale the data set and repeat steps 2-4. How has the result changed and why?

```
##4.5#################################################################
X <- scale(data[,-102])
Xe1 <- cbind(1,X)
A1 <- t(Xe1)%*%Xe1
b1 <- t(Xe1)%*%Y
beta1 <- solve(A1,b1)
# beta1 <- solve(t(Xe1)%*%Xe1)%*%t(Xe1)%*%Y
kappa(A1)
```

```
## [1] 490471520661
```

After scaling the data, the function *solve* works well for new matrix A. Its corresponding condition number ($\approx 4.9 * 10^{11}$) is much smaller than the previous one ($\approx 8.5 * 10^{14}$). Theoretically, the model trained by scaled data might be better than the one by un-scaled data.

The mse unscaled:

```
## [1] 0.0762971
```

The mse scaled:

```
## [1] 0.07629706
```

According to the MSE, the model with scaled data has smaller MSE and better performance confirmly. One of reason could be that when the range of matrix A is very large, it is easy to occur underflow when calculating the determinant and inverse of such matrix. This would have a negative effect to prediction. Additionally, the condition number might be large based on un-scaled data since its value is in accordance with the determinant of original matrix A and its inverse.

Appendix

```
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
# the lirbraries used in this lab
library(ggplot2)
# snippets 1
x1 = 1/3 ; x2 = 1/4
if(x1 - x2 == 1/12){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
# snippets 2
x1 = 1 ; x2 = 1/2
if(x1 - x2 == 1/2){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
# snippets 1 - improved
x1 = 1/3 ; x2 = 1/4
if(isTRUE(all.equal(x1 - x2, 1/12))){
  print("Subtraction is correct")
} else{
  print("Subtraction is wrong")
}
epsilon = 10**-15
derivate = function(x){
  deriv = ((x + epsilon) - x) / epsilon
  return(deriv)
}
x1= 1
x2= 100000
derivate(x1)
derivate(x2)
# write the preseted var function - input vector x
myvar = function(x){
  n = length(x)
  var_calculation = (1/(n-1)) * (sum(x^2)-(1/n)*(sum(x)^2))
  return(var_calculation)
}
# initialize vector
n = 10000
x_vector = rnorm(n, mean = 10^8, sd = 1)
# try - 2
y = c()
for (i in 1:length(x_vector)) {
  y[i] = myvar(x_vector[1:i]) - var(x_vector[1:i])
}
# remove the first value - variance of one pont can??t be calculated - does not make sense
y = y[-1]

# create data framem for the plot
data_dependence_Y_i = data.frame(y_value = y,
```

```
                                  iteration = 1:length(y))
# create plot dependence Yi on i
plot_dependence_Y_i =
  ggplot(data = data_dependence_Y_i) +
  geom_point(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)

plot_dependence_Y_i_2to300 =
  ggplot(data = data_dependence_Y_i[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i[2:300]") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)
plot_dependence_Y_i

plot_dependence_Y_i_2to300
##3.4#################################################################
myvar1 <- function(x){
  m <- mean(x)
  sum((x-m)**2)/(length(x)-1)
}
y1 <- sapply(1:n, function(i){
  myvar1(x_vector[1:i])-var(x_vector[1:i]) # changed xbar to x_vector
})
y1 = y1[-1]
data_dependence_Y_i_improved = data.frame(y_value = y1,
                                  iteration = 1:length(y1))

plot_dependence_Y_i_2to300_improved_scale3 =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i with ylim(-3,3)") + ylab("Y_i") + xlab("i")+
  ylim(-3,3)

plot_dependence_Y_i_2to300_improved =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i[2:300]") + ylab("Y_i") + xlab("i")
plot_dependence_Y_i_2to300_improved_scale3

plot_dependence_Y_i_2to300_improved
##4.1#################################################################
data <- as.matrix(read.csv("tecator.csv"))
data <- data[,-1]
X <- data[,-102]
Y <- data[,102]
##4.2#################################################################
Xe0 <- cbind(1,X)
A0 <- t(Xe0)%*%Xe0
b0 <- t(Xe0)%*%Y
##4.3#################################################################
tryCatch(solve(A0,b0),
         error=function(e){cat("ERROR :\n",conditionMessage(e),"\n")})
```

```
##4.4#####################################################################
beta0 <- solve(A0,b0,tol = 7.78804e-17)
kappa(A0)
##4.5#####################################################################
X <- scale(data[,-102])
Xe1 <- cbind(1,X)
A1 <- t(Xe1)%*%Xe1
b1 <- t(Xe1)%*%Y
beta1 <- solve(A1,b1)
# beta1 <- solve(t(Xe1)%*%Xe1)%*%t(Xe1)%*%Y
kappa(A1)
mse_unscaled <- mean((Xe0%*%beta0-Y)**2)
mse_scaled <- mean((Xe1%*%beta1-Y)**2)
mse_unscaled
mse_scaled
```

Computer Lab 2 Computational Statistics

Contents

Question 1: Optimizing a model parameter

The file *mortality rate.csv* contains information about mortality rates of the fruit flies during a certain period.

1. Import this file

to *R* and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets by using the following code:

```
# import the data
data = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data$LMR = log(data$Rate)
```

2. Write your own function `myMSE`

that for given parameters λ and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty λ (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement):

$$predictiveMSE = \frac{1}{length(test)} \sum_{i \leq length(test)} (Ytest[i] - fYpred(X[i]))^2,$$

where fYpred(X[i]) is the predicted value of Y if X is X[i]. Read on R's functions for prediction so that you do not have to implement it yourself.

```
# the myMSE() function
# input of the function- parameter lambda & list(vector(X, Y, Xtest, Ytest)) pars
# list(vector(X, Y, Xtest, Ytest)) pars
input_vector = list(X = train$Day, Y = train$LMR , Xtest = test$Day, Ytest = test$LMR)

# create an empty counter
counter = 0

myMSE = function(lambda,pars){
# create the model
  model = loess(formula = pars$Y ~ pars$X,
                enp.target = lambda)# just use enp.target or
                #,span = 0.75) # default - smoothing parameter
  #make the prediction
  pred <- predict(object = model,
                  newdata = pars$Xtest)
  # calculate the mse
  predictiveMSE = (1/length(pars$Ytest)) * sum((pars$Ytest - pred)^2)

  #print and return the result
  counter <<- counter + 1
  #print(predictiveMSE)
  return(predictiveMSE)
}
```

3. Estimate MSEs by `myMSE()`

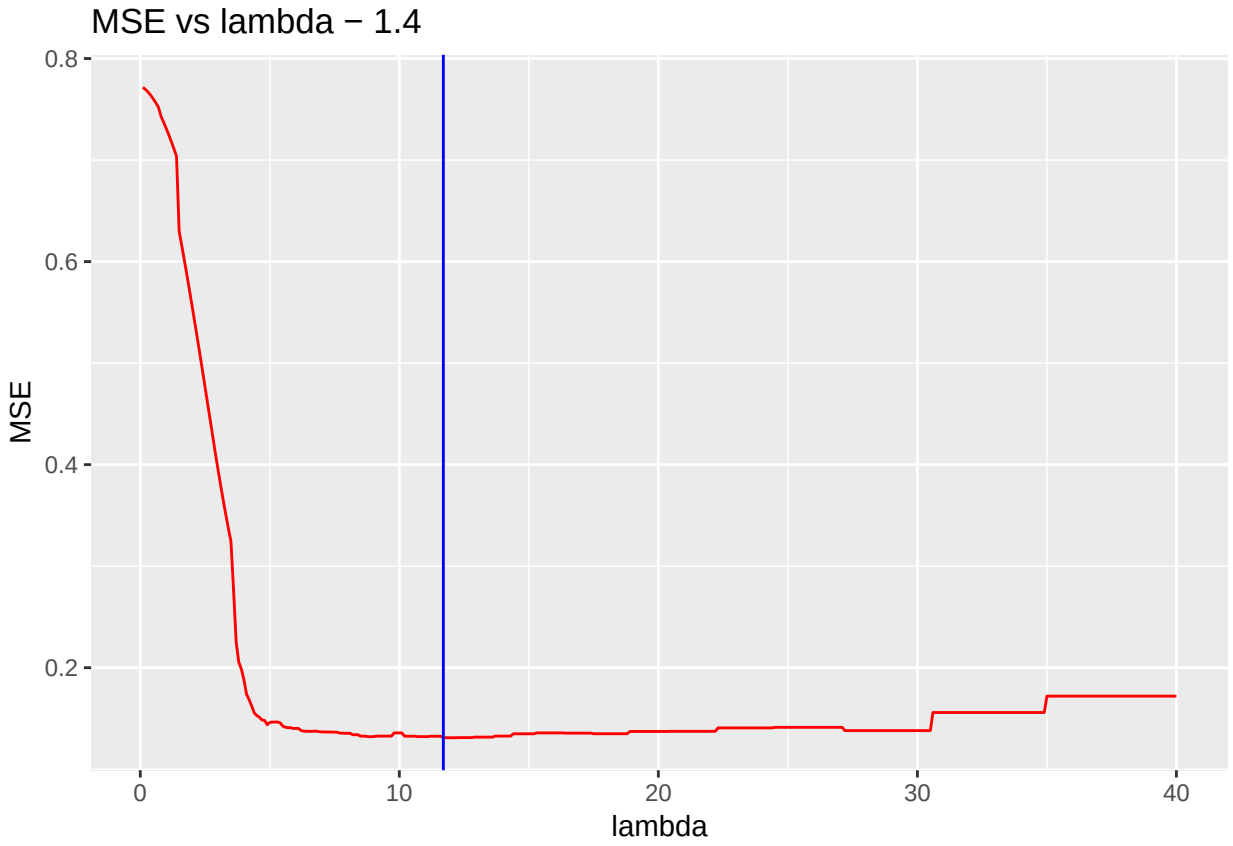
Use a simple approach: use function `myMSE()`, training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: $\lambda = 0.1,0.2,\ldots,40$

```
# initialize lambdas
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# use the function myMSE - for all lambda values
mse_result = c()
for (i in 1:length(lambdas)) {
  mse_result[i] = myMSE(lambdas[i], input_vector)
}
```

4. Create a plot of the MSE vs. λ

and comment on which λ value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?

The plot:



This subtask is answered in section 1.6.

5. Use `optimize()` function

for the same purpose, specify range for search [0.1,40] and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many `myMSE()` function evaluations were required? Compare to step 4.

```
# set the counter to 0
counter = 0
opti = optimize(f = myMSE,
                interval = c(0.1,40),
                pars = input_vector, # specify the input for the function  pars
                tol = 0.01) # dedired accuracy
```

This subtask is answered in section 1.6.

6. Use `optim()` function

and BFGS method with starting point $\lambda = 35$ to find the optimal λ value. How many `myMSE()` function evaluations were required (read ?optim)? Compare the results you obtained with the results from step 5 and make conclusions.

```
#?optim()
opti2 = optim(par = 35,
              fn = myMSE,
              method = "BFGS",
              pars= input_vector)
```

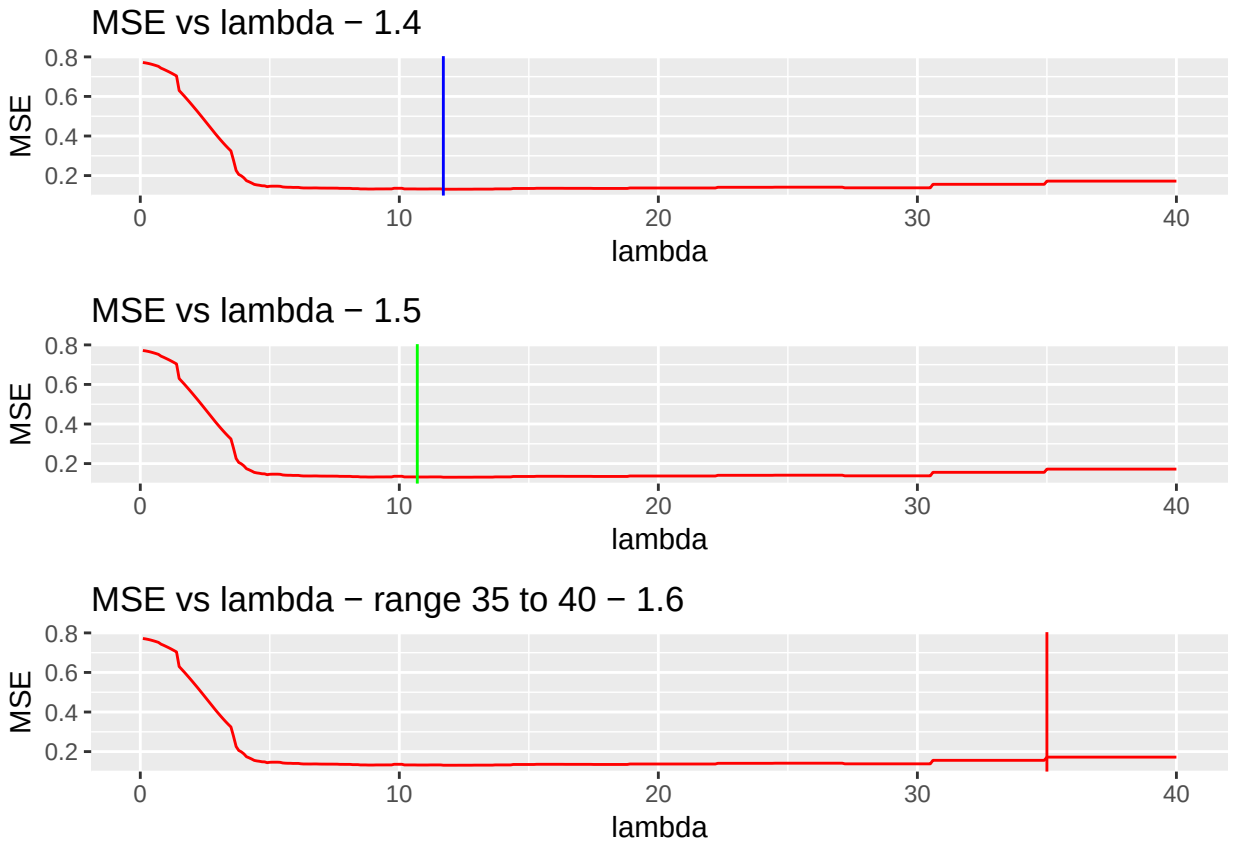
	Minumium.MSE	Optimal.lambda	Iter..of.myMSE
Result 1.4	0.1310470	11.70000	400
Result 1.5	0.1321441	10.69361	18
Result 1.6	0.1719996	35.00000	1

Above can we see the optimal MSE, optimal λ and the number of evaluations were used. Since we implemented a for loop with 400 iterations to evaluate all the lambda values for the best λ-value. At the 117 iteration did we find the minimal MSE value.

According to `help()`, the `optimize()` function was able to find the optimal value of λ by a combination of golden section search and successive parabolic interpolation. The number of iterations the function `myMSE()` needed is 18. Its minimum MSE is larger than the one in step 4, but `optimize()` is much quicker and effective.

In the last run did we evaluated the λ-interval from 35 to 40. Method "BFGS" is a quasi-Newton method, which could not only operate as fast as Newton method but also search the local minimum effciently. The output *counts* shows `optim()` evaluates `myMSE()` function once and its gradient once to build up a picture of the surface to be optimized. The table below shows that it found the best value of λ in the point 35. Which is the optimal values for this specific (local) interval, but not for the whole (global) interval. Thus, `optim()` is the fastest function to find optimal λ, but users should be careful with the initial values.

Plot *MSE vs* λ of task 1.4, 1.5 and 1.6 combined



Question 2: Maximizing likelihood

The file *data.RData* contains a sample from normal distribution with some parameters μ, σ. For this question read ?optim in detail.

1. Load the data to R environment.

```
rm(list = ls())
# load the data into the environment
load("data.RData")
```

2. Write down the log-likelihood function for 100 observations

and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

The probability density function (PDF) of normal distribution is:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in X,$$

where $X = (x_1, x_2, ..., x_{100})$. The parameters μ and σ are based on the sample X.

Its corresponding likelihood function:

$$\begin{aligned} f(x_1, x_2, ..., x_n \mid \mu, \sigma) =& \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \\ =& (\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}, \end{aligned}$$

and Log-likelihood:

$$\begin{aligned} log(f(x_1, x_2, ..., x_n \mid \mu, \sigma)) =& log((\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}) \\ =& nlog(\frac{1}{\sigma\sqrt{2\pi}}) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \\ =& -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \\ =& -\frac{n}{2}log(2\pi) - \frac{n}{2}log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2. \end{aligned}$$

Call $log(f(x_1, x_2, ..., x_n \mid \mu, \sigma))$ as L, the derivation of the log-likelihood - μ should be

$$\frac{\partial L}{\partial \mu} = -\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) = 0,$$

and corresponding solution is

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Then we derivate of the log-likelihood - σ

$$\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{\sigma^3} = 0,$$

and its solution thereby is

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}{n}}.$$

```
#data
# Use the derived formule to obtain parameter estimates for the loaded data.
# fromula for mu
mymu <- function(x) sum(x)/length(x)
mu_hat =  mymu(data)

# formula for sigma
mysd <- function(x) sqrt(sum((x-mymu(x))**2)/length(x))
sigma_hat = mysd(data)
```

Value of $\hat{\mu}$:

```
## [1] 1.275528
```

Value of $\hat{\sigma}$:

```
## [1] 2.005976
```

3. Optimize the minus log-likelihood function

with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

Negative log-likelihood:

$$L(y) = -log(y)$$

```
# minus loglikelihood - function
# n - number of observations
minus_loglikelihood <- function(par,x){
  mu=par[1]
  sd=par[2]
  n=length(x)
  logllk <- -log(2*pi)*n/2-log(sd**2)*n/2-sum((x-mu)**2)/(2*sd**2)
  -logllk
}

#  the gradient function
gradient_function <- function(par,x){
  dmu <- sum(par[1]-x)/par[2]**2
  dsd <- length(x)/par[2]-sum((x-par[1])**2)/(par[2]**3)
  return(c(dmu,dsd))
}
```

```r
pars <- c(mymu(data),mysd(data))
minus_loglikelihood(pars, x=data)
```

```
## [1] 211.5069
```

Optimize the minus log-likelihood function:

```r
opti_BFGS = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      x=data) #quasi-Newton method

opti_BFGS_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      gr = gradient_function,
      x=data)

opti_CG = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      x=data) #conjugate gradients method

opti_CG_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      gr = gradient_function,
      x=data)
```

	Method	Gradient.specified	mu	sigma	Num.fun	Num.gr	convergence
opti_BFGS	BFGS	FALSE	1.275527	2.005977	37	15	0
opti_BFGS_gr	BFGS	TRUE	1.275527	2.005977	38	15	0
opti_CG	CG	FALSE	1.275528	2.005977	210	35	0
opti_CG_gr	CG	TRUE	1.275528	2.005977	53	17	0

Maximizing likelihood function is a question with exponent, calculating exponent and the value with power n are very expensive compared with calculating its logarithm, since its logarithm is additive directly. Thus, maximizing log-likelihood can be more economical and effecient than maximize likelihood.

4. Did the algorithms converge in all cases?

What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

Yes, all the algorithms converge to 0, which means the data is really normal distributed and such 4 methods work well based on our data. According to the table, all the estimated μ and σ are simiar. However, the evaluations are very different among these 4 methods. CG methods have more evaluations both in function and gradient, especially the one without specified gradient. It seems that CG method is not a good choice when the gradient is not given. On the contrary, the results of two BFGS methods are very close, which might mean that BFGS method is more robust on normal distributed sample.

References

```
#sources used
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
#https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/#nll
#https://en.wikipedia.org/wiki/Likelihood_function
#https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood
```

Appendix

```
rm(list=ls())
Sys.setlocale(locale = "english")
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
# list packages we use in this lab
library(ggplot2)
#install.packages("gridExtra") # to put the plots togeter
library(gridExtra)
# import the data
data = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data$LMR = log(data$Rate)
# split the data into train and test set
n=dim(data)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
# the myMSE() function
# input of the function- parameter lambda & list(vector(X, Y, Xtest, Ytest)) pars
# list(vector(X, Y, Xtest, Ytest)) pars
input_vector = list(X = train$Day, Y = train$LMR , Xtest = test$Day, Ytest = test$LMR)

# create an empty counter
counter = 0

myMSE = function(lambda,pars){
# create the model
  model = loess(formula = pars$Y ~ pars$X,
                enp.target = lambda)# just use enp.target or
                #,span = 0.75) # default - smoothing parameter
  #make the prediction
  pred <- predict(object = model,
                  newdata = pars$Xtest)
  # calculate the mse
  predictiveMSE = (1/length(pars$Ytest)) * sum((pars$Ytest - pred)^2)

  #print and return the result
  counter <<- counter + 1
  #print(predictiveMSE)
  return(predictiveMSE)
```

```
}
# initialize lambdas
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# use the function myMSE - for all lambda values
mse_result = c()
for (i in 1:length(lambdas)) {
  mse_result[i] = myMSE(lambdas[i], input_vector)
}

# the min mse value
mse_min = min(mse_result)
# the lamnda index of the min vlaue
mse_min_index = which.min(mse_result)
# the min lambda value
lambda_optimal = lambdas[mse_min_index]
# create the plot of the MSE values
# data frame for the plot
mse_plot_data = data.frame(mse_values = mse_result,
                           lamda_values = lambdas)

mse_plot = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - 1.4") +
  geom_vline(xintercept = lambda_optimal, color = "blue", size = 0.5) +
  xlab("lambda") + ylab("MSE")
mse_plot
#create data frame with result values
result14 = data.frame(
  "Minumium MSE"=mse_min,
  "Optimal lambda"=lambda_optimal,
  "Iter. of myMSE"=counter)
#knitr::kable(result14)
# set the counter to 0
counter = 0
opti = optimize(f = myMSE,
                interval = c(0.1,40),
                pars = input_vector, # specify the input for the function  pars
                tol = 0.01) # dedired accuracy
#create data frame with result values
result15 = data.frame(
  "Minumium MSE"=opti$objective,
  "Optimal lambda"=opti$minimum,
  "Iter. of myMSE"=counter)
#knitr::kable(result15)

mse_plot2 = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - 1.5 ") +
  geom_vline(xintercept = opti$minimum, color = "green", size = 0.5) +
  xlab("lambda") + ylab("MSE")
#?optim()
opti2 = optim(par = 35,
```

```
              fn = myMSE,
              method = "BFGS",
              pars= input_vector)

#create data frame with result values
result16 = data.frame(
  "Minumium MSE"=opti2$value,
  "Optimal lambda"=opti2$par,
  "Iter. of myMSE"=opti2$counts[1])
#knitr::kable(result16)
# create a table for the result for 1.4, 1.5, 1.6
result1 = rbind(result14,result15,result16)
rowname = c("Result 1.4", "Result 1.5", "Result 1.6")
rownames(result1) = rowname
knitr::kable(result1)
mse_plot3 = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - range 35 to 40 - 1.6") +
  geom_vline(xintercept = opti2$par, color = "red", size = 0.5) +
  xlab("lambda") + ylab("MSE")
grid.arrange(mse_plot, mse_plot2, mse_plot3, nrow = 3)
rm(list = ls())
# load the data into the environment
load("data.RData")
#data
# Use the derived formule to obtain parameter estimates for the loaded data.
# fromula for mu
mymu <- function(x) sum(x)/length(x)
mu_hat =  mymu(data)

# formula for sigma
mysd <- function(x) sqrt(sum((x-mymu(x))**2)/length(x))
sigma_hat = mysd(data)
mu_hat
sigma_hat
# minus loglikelihood - function
# n - number of observations
minus_loglikelihood <- function(par,x){
  mu=par[1]
  sd=par[2]
  n=length(x)
  logllk <- -log(2*pi)*n/2-log(sd**2)*n/2-sum((x-mu)**2)/(2*sd**2)
  -logllk
}

#  the gradient function
gradient_function <- function(par,x){
  dmu <- sum(par[1]-x)/par[2]**2
  dsd <- length(x)/par[2]-sum((x-par[1])**2)/(par[2]**3)
  return(c(dmu,dsd))
}

pars <- c(mymu(data),mysd(data))
```

```r
minus_loglikelihood(pars, x=data)

opti_BFGS = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      x=data) #quasi-Newton method

opti_BFGS_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      gr = gradient_function,
      x=data)

opti_CG = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      x=data) #conjugate gradients method

opti_CG_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      gr = gradient_function,
      x=data)

# create a data frame of results
# vector of the column names
colnames_result_dataframe <- c("Method", "Gradient.specified", "convergence", "mu", "sigma", "Num.fun",

#opti_BFGS_gr
opti_BFGS_gr_df =  data.frame("BFGS", T , opti_BFGS_gr$convergence, opti_BFGS_gr$par[1],opti_BFGS_gr$pa
colnames(opti_BFGS_gr_df) = colnames_result_dataframe

#opti_CG
opti_CG_df =  data.frame("CG", F, opti_CG$convergence, opti_CG$par[1],opti_CG$par[2],opti_CG$counts[1],
colnames(opti_CG_df) = colnames_result_dataframe

#opti_CG_gr
opti_CG_gr_df =  data.frame("CG", T, opti_CG_gr$convergence, opti_CG_gr$par[1],opti_CG_gr$par[2],opti_C
colnames(opti_CG_gr_df) = colnames_result_dataframe

# create data frame with values of first evaluation
result_df = data.frame("Method" = "BFGS",
                       "Gradient specified" = F,
                       "convergence" = opti_BFGS$convergence,
                       "mu" = opti_BFGS$par[1],
                       "sigma"= opti_BFGS$par[2],
                       "Num.fun" = opti_BFGS$counts[1],
                       "Num.gr" = opti_BFGS$counts[2])
# combind the data frame by row
result_df = rbind(result_df, opti_BFGS_gr_df)
result_df = rbind(result_df, opti_CG_df)
result_df = rbind(result_df, opti_CG_gr_df)
result_df <- result_df[,c(1,2,4,5,6,7,3)]
```

```r
# vector of row names for the table
rownames_table = c("opti_BFGS", "opti_BFGS_gr","opti_CG", "opti_CG_gr")
rownames(result_df) = rownames_table

knitr::kable(result_df)
#sources used
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
#https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/#nll
#https://en.wikipedia.org/wiki/Likelihood_function
#https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood
```

Computer Lab 3 Computational Statistics

Contents

Question 1: Cluster sampling

An opinion pool is assumed to be performed in several locations of Sweden by sending interviewers to this location. Of course, it is unreasonable from the financial point of view to visit each city. Instead, a decision was done to use random sampling without replacement with the probabilities proportional to the number of inhabitants of the city to select 20 cities. Explore the file *population.xls*. Note that names in bold are counties, not cities.

1. Import necessary information to R.

```
# load the data
data = read.csv2("population.csv", encoding = "latin1")
data[,1] <- as.character(data[,1])
```

2. Use a uniform random number generator

to create a function that selects 1 city from the whole list by the probability scheme offered above (do not use standard sampling functions present in R).

```
# create the select city function - based on max
selectCity = function(data){
  # proportional to the number of inhabitants of the city
  data$proportinalPopulation = data$Population/sum(data$Population)
  # generate random numbers - include to dataset
  data$rn_uniform = runif(n=nrow(data),0,1)
  # probabilities proportional to the number of inhabitants
  data$prob_proportinalPopulation = data$proportinalPopulation * data$rn_uniform
  # take one city - max prob_proportinalPopulation
```

```
  city_index = which.max(data$prob_proportinalPopulation)
  # print(data[city_index,][1]) # make the print nicer

  return(data[city_index,1])
}
```

3. Use the function you have created in step 2

as follows:

- (a) Apply it to the list of all cities and select one city
- (b) Remove this city from the list
- (c) Apply this function again to the updated list of the cities
- (d) Remove this city from the list
- (e) . . . and so on until you get exactly 20 cities.

```
# create a function to select 20 cities
select20cities = function(data){
  cities_remain <- data
  selected_cities <- c()
  for (i in 1:20) {
  selected_city <- selectCity(cities_remain)
  selected_cities <- c(selected_cities,selected_city)
  cities_remain <- cities_remain[-which(cities_remain[,1]==selected_city),]
  }
  return(data.frame(Municipality=selected_cities))
}
```

4. Run the program.

Which cities were selected? What can you say about the size of the selected cities?

```
# list of all selected cities
selected_cities <- select20cities(data)
idx <- match(selected_cities$Municipality, data[,1])
selected_cities$Population <- data[idx,2]
selected_cities <- selected_cities[order(selected_cities[,2],decreasing = TRUE),]
rownames(selected_cities) <- c()
```

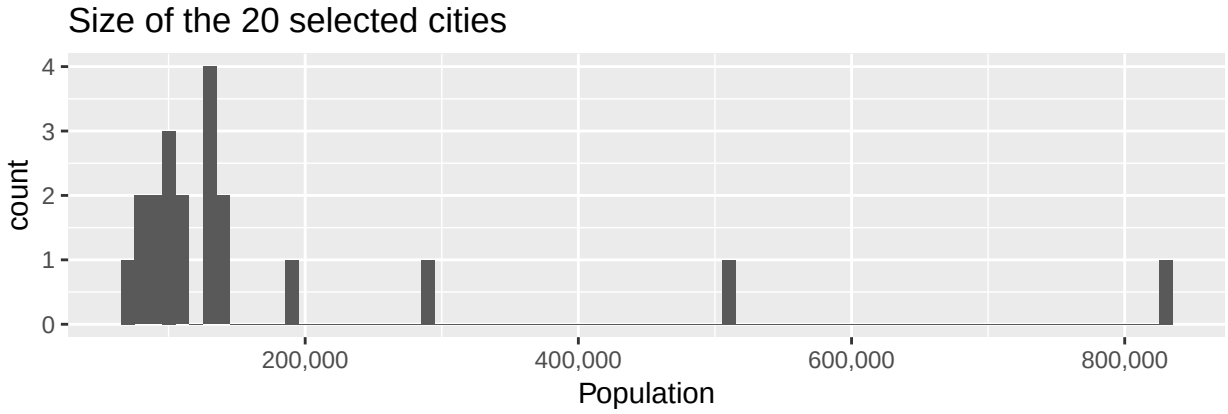
Municipality	Population
Stockholm	829417
Göteborg	507330
Malmö	293909
Uppsala	194751
Linköping	144690
Västerås	135936
Örebro	134006
Norrköping	129254
Helsingborg	128359
Jönköping	126331
Umeå	114075

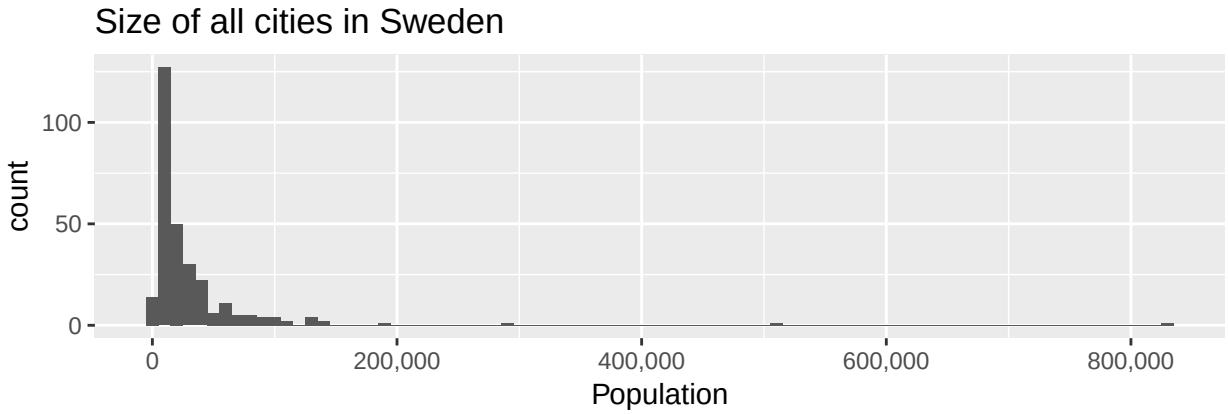
Municipality	Population
Lund	109147
Borås	102458
Eskilstuna	95577
Sundsvall	95533
Gävle	94352
Södertälje	85270
Karlstad	84736
Haninge	76237
Luleå	73950

The output above shows all the cities we selected, it is obvious that the cities with large population are quite easy to be selected, especially Stockholm, Goteborg, Malmo and Uppsala. This can be seen in a later visualization.

5. Plot one histogram showing the size of all cities of the country.

Plot another histogram showing the size of the 20 selected cities. Conclusions?





The population of most of Swedish cities is around (0,100000), with relatively small size compared with other big cities. Most of the cities we randomly select belong to big-size cities with huge population ($\geq$ 100000), and only 7 cities' populations are smaller than 100000.

Question 2: Different distributions

The double exponential (Laplace) distribution is given by formula:

$$DE(\mu, \alpha) = \frac{\alpha}{2} \exp(-\alpha \mid x - \mu \mid)$$

1. Write a code generating double exponential distribution DE(0, 1)

from Unif(0, 1) by using the inverse CDF method. Explain how you obtained that code step by step. Generate 10000 random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

Firstly, we create a sequence of random numbers which are uniform distributed,

$$u \sim \text{Unif}(0, 1).$$

Since the cumulative distribution function (CDF) of Laplace distribution Y is

$$F_{DE}(x) = \frac{1}{2} + \frac{1}{2}\text{sgn}(\text{x} - \mu_{\text{DE}})[1 - \exp(-\alpha|\text{x} - \mu_{\text{DE}}|)] = \text{y}$$

and the deduction

$$P[F_Y^{-1}(u) \leq y] = P[u \leq F_Y(y)] = F_U[F_Y(y)] = F_Y(y),$$

We combine the duduction and u into the inverse CDF of DE(0,1), and we thereby get such formula as

$$F_{DE}^{-1}(u) = \mu_{DE} - \frac{\text{sgn}(\text{u} - \mu_{\text{Unif}})}{\alpha} \ln[1 + sgn(u - \mu_{\text{Unif}}) - sgn(u - \mu_{\text{Unif}})2u].$$

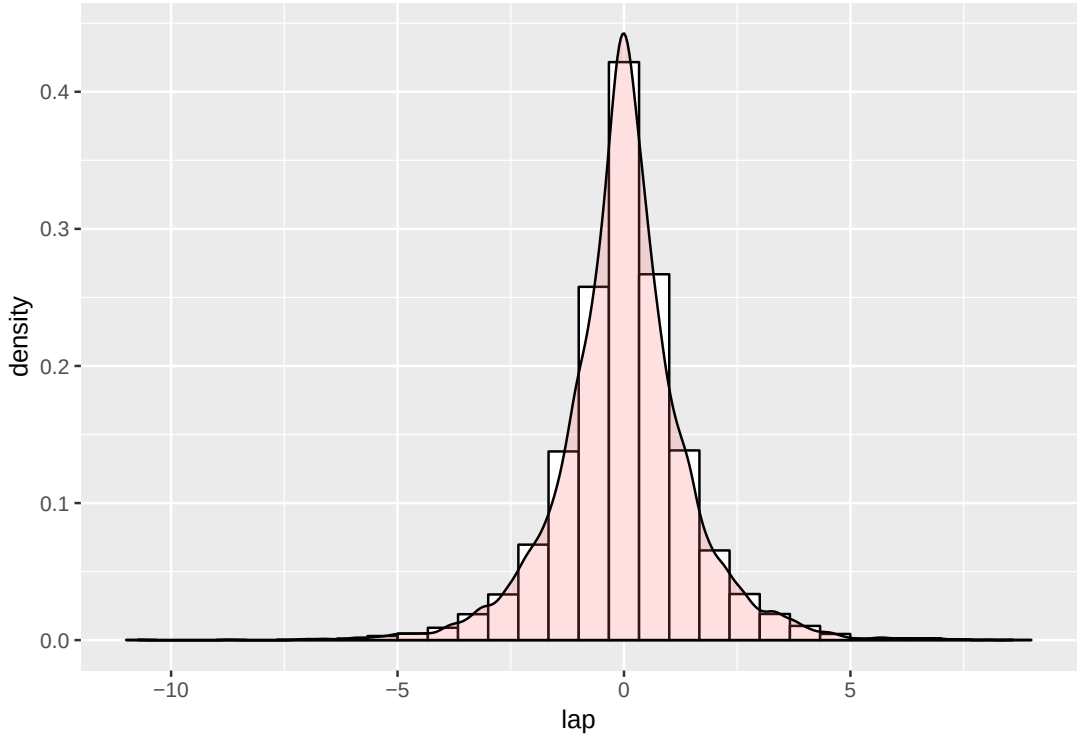
Substituting $\mu_{\text{Unif}} = 0.5$ into the formula,

$$Y = F_{DE}^{-1}(u) = \mu_{DE} - \frac{\text{sgn}(\text{u} - 0.5)}{\alpha} \ln(1 - 2|u - 0.5|).$$

Thus, we conclude that $Y \sim DE(0, 1)$ and thereby get a sequence y following Laplace distribution by the inverse CDF function of Laplace distribution.

```r
# generate random numbers
n=10000
set.seed(123456)
x_rand <- runif(n=n, min=0, max=1)
data1 <- data.frame(unif=x_rand)
```

```r
# the inverse laplace with mu = 0.5
laplace_distribution = function(mu, alpha, p){
  # result <- mu-(1/alpha)*sign(p-0.5)*log(1-2*abs(p-0.5))
  result <- mu-sign(p-0.5)*1/alpha*log(1+sign(p-0.5)-sign(p-0.5)*2*p)
  return(result)
}
```



2. Use the Acceptance/rejection method

with DE(0,1) as a majorizing density to generate N(0,1) variables. Explain step by step how this was done. How did you choose constant c in this method? Generate 2000 random numbers N(0,1) using your code and plot the histogram. Compute the average rejection rate R in the acceptance/rejection procedure. What is the expected rejection rate ER and how close is it to R? Generate 2000 numbers from N(0,1) using standard rnorm() procedure, plot the histogram and compare the obtained two histograms.

Firstly, we treat the sequene `data1$lap` as $Y \sim f_Y$. Then we find the probability density function (PDF) of Laplace distribution DE(0,1), which is

$$f_Y(x) = \frac{1}{2}\exp(- \mid x \mid).$$

Meanwhile, we find the PDF of normal distribution N(0,1),

$$f_X(x) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{x^2}{2}).$$

We want a c which is larger than f_X/f_Y. Since we have

$$\frac{f_X}{f_Y} = \sqrt{\frac{2}{\pi}}\exp(-\frac{x^2}{2} + |x|),$$

it will get its maximum when $x = \pm 1$. So

$$c = \max \frac{f_X}{f_Y} = \sqrt{\frac{2}{\pi}}\exp(\frac{1}{2}) \approx 1.315.$$

By creating a sequence $u \sim \text{Unif}(0, 1)$, if it statisfies

$$u_i \leq \frac{f_X(Y_i)}{cf_Y(Y_i)},$$

```
pdf_norm = function(x){
  exp(-(x**2)/2)/(sqrt(2*pi))
}
pdf_lap = function(x){
  (1/2)*exp(-abs(x))
}

c <- sqrt(2/pi)*exp(0.5)
n2 <- 2000
accept <- c()

for (i in 1:n) {
  if(length(accept)>=n2){
    break()
  }
  Y <- data1[i,2]
  u <- runif(1)
  g <- pdf_lap(Y)
  f <- pdf_norm(Y)

  if(u<=f/(c*g)){
    accept <- c(accept,Y)
  }
}
data2 <- data.frame(val=accept, id="esti")
reject_rate <- 1-n2/i
```

ejection rate R

```
## [1] 0.2595335
```

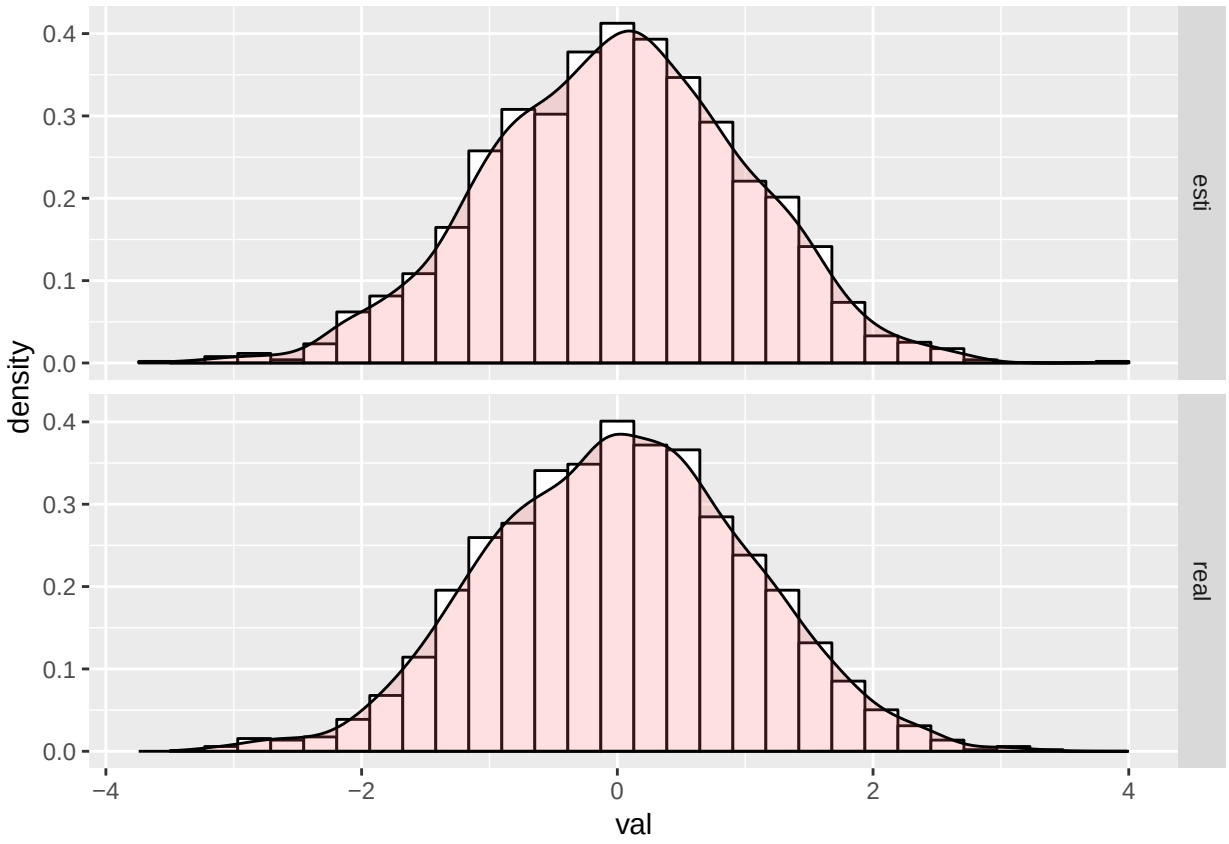
Rejection rate E(R):

```
1-1/c
```

```
## [1] 0.2398265
```

The expected rejection rate should be $\mathrm{E(R)} = 1/\mathrm{c} \approx 0.24$. The real rejection rate R is around 0.25, which is also close to E(R). The following figures are based on our estimated normal distributed variables and samples from `rnorm()` respectively. It is right that such two figures like similar mutually.

```
data3 <- data.frame(val=rnorm(n=n2,0,1),id ="real")
data3 <- rbind(data2,data3)

ggplot(data = data3, aes(x=val)) +
 geom_histogram(aes(y=..density..),
                colour="black",
                fill="white",
                bins=30)+
  geom_density(alpha=.2, fill="#FF6666")+
  facet_grid(id~.)
```


density
0.4
0.3
0.2
0.1
0.0
esti
real
−4
−2
0
2
4
val

Appendix

```
knitr::opts_chunk$set(echo = TRUE, out.height = "200px")
#Sys.setlocale(locale="english")
# libraries used
rm(list=ls())
library(ggplot2)
#install.packages("gridExtra") # to put the plots togeter
library(gridExtra)

# set working directory
# use code for encoding
Sys.setlocale("LC_ALL", 'en_US.UTF-8')
# load the data
data = read.csv2("population.csv", encoding = "latin1")
data[,1] <- as.character(data[,1])
# create the select city function - based on max
selectCity = function(data){
  # proportional to the number of inhabitants of the city
  data$proportinalPopulation = data$Population/sum(data$Population)
  # generate random numbers - include to dataset
  data$rn_uniform = runif(n=nrow(data),0,1)
  # probabilities proportional to the number of inhabitants
  data$prob_proportinalPopulation = data$proportinalPopulation * data$rn_uniform
  # take one city - max prob_proportinalPopulation
  city_index = which.max(data$prob_proportinalPopulation)
  # print(data[city_index,][1]) # make the print nicer

  return(data[city_index,1])
}
# create a function to select 20 cities
select20cities = function(data){
  cities_remain <- data
  selected_cities <- c()
  for (i in 1:20) {
  selected_city <- selectCity(cities_remain)
  selected_cities <- c(selected_cities,selected_city)
  cities_remain <- cities_remain[-which(cities_remain[,1]==selected_city),]
  }
  return(data.frame(Municipality=selected_cities))
}
# list of all selected cities
selected_cities <- select20cities(data)
idx <- match(selected_cities$Municipality, data[,1])
selected_cities$Population <- data[idx,2]
selected_cities <- selected_cities[order(selected_cities[,2],decreasing = TRUE),]
rownames(selected_cities) <- c()
knitr::kable(selected_cities)

# histogram - 2 selected cities
histplot_20cities = ggplot(selected_cities)+
  geom_histogram(binwidth=10000,aes(x=Population)) +
  ggtitle("Size of the 20 selected cities") +
```

```
  scale_x_continuous(labels = scales::comma)

histplot_cities = ggplot(data)+
  geom_histogram(binwidth=10000,aes(x=Population)) +
  ggtitle("Size of all cities in Sweden")+
  scale_x_continuous(labels = scales::comma)
# put the plots together
grid.arrange(histplot_20cities, histplot_cities, ncol = 1)
# clean environment
rm(list=ls())
# generate random numbers
n=10000
set.seed(123456)
x_rand <- runif(n=n, min=0, max=1)
data1 <- data.frame(unif=x_rand)
ggplot(data1,aes(x=unif))+
   geom_histogram(aes(y=..density..),
                  colour="black",
                  fill="white",
                  bins=30)+
  geom_density(alpha=.2, fill="#FF6666")
# the inverse laplace with mu = 0.5
laplace_distribution = function(mu, alpha, p){
  # result <- mu-(1/alpha)*sign(p-0.5)*log(1-2*abs(p-0.5))
  result <- mu-sign(p-0.5)*1/alpha*log(1+sign(p-0.5)-sign(p-0.5)*2*p)
  return(result)
}

data1$lap <- laplace_distribution(0,1, x_rand)

ggplot(data = data1, aes(x=lap)) +
 geom_histogram(aes(y=..density..),
                  colour="black",
                  fill="white",
                  bins=30)+
  geom_density(alpha=.2, fill="#FF6666")
pdf_norm = function(x){
  exp(-(x**2)/2)/(sqrt(2*pi))
}
pdf_lap = function(x){
  (1/2)*exp(-abs(x))
}

c <- sqrt(2/pi)*exp(0.5)
n2 <- 2000
accept <- c()

for (i in 1:n) {
  if(length(accept)>=n2){
    break()
  }
  Y <- data1[i,2]
  u <- runif(1)
```

```
  g <- pdf_lap(Y)
  f <- pdf_norm(Y)

  if(u<=f/(c*g)){
    accept <- c(accept,Y)
  }
}
data2 <- data.frame(val=accept, id="esti")
reject_rate <- 1-n2/i
reject_rate
1-1/c
data3 <- data.frame(val=rnorm(n=n2,0,1),id ="real")
data3 <- rbind(data2,data3)

ggplot(data = data3, aes(x=val)) +
 geom_histogram(aes(y=..density..),
                colour="black",
                fill="white",
                bins=30)+
  geom_density(alpha=.2, fill="#FF6666")+
  facet_grid(id~.)
```

Computer Lab 4 Computational Statistics

Contents

Question 1: Computations with Metropolis - Hastings

Consider the following probability density function:

$$f(x) \propto x^5 e^{-x}, \ x > 0$$

You can see that the distribution is known up to some constant of proportionality. If you are interested (NOT part of the Lab) this constant can be found by applying integration by parts multiple times and equals 120.

1. Use Metropolis-Hastings algorithm

to generate samples from this distribution by using proposal distribution as log-normal $\text{LN}(\text{X}_\text{t}, 1)$, take some starting point. Plot the chain you obtained as a time series plot. What can you guess about the convergence of the chain? If there is a burn-in period, what can be the size of this period?

```
rm(list=ls())
set.seed(123456)
#pdf -  probability density function
pdf_target = function(x){
  if(x<= 0){
    stop("x needs to be bigger than 0")
  }
  # stopifnot(x>0) # check - x >0
  return(x^5 * exp(-x))
}
```

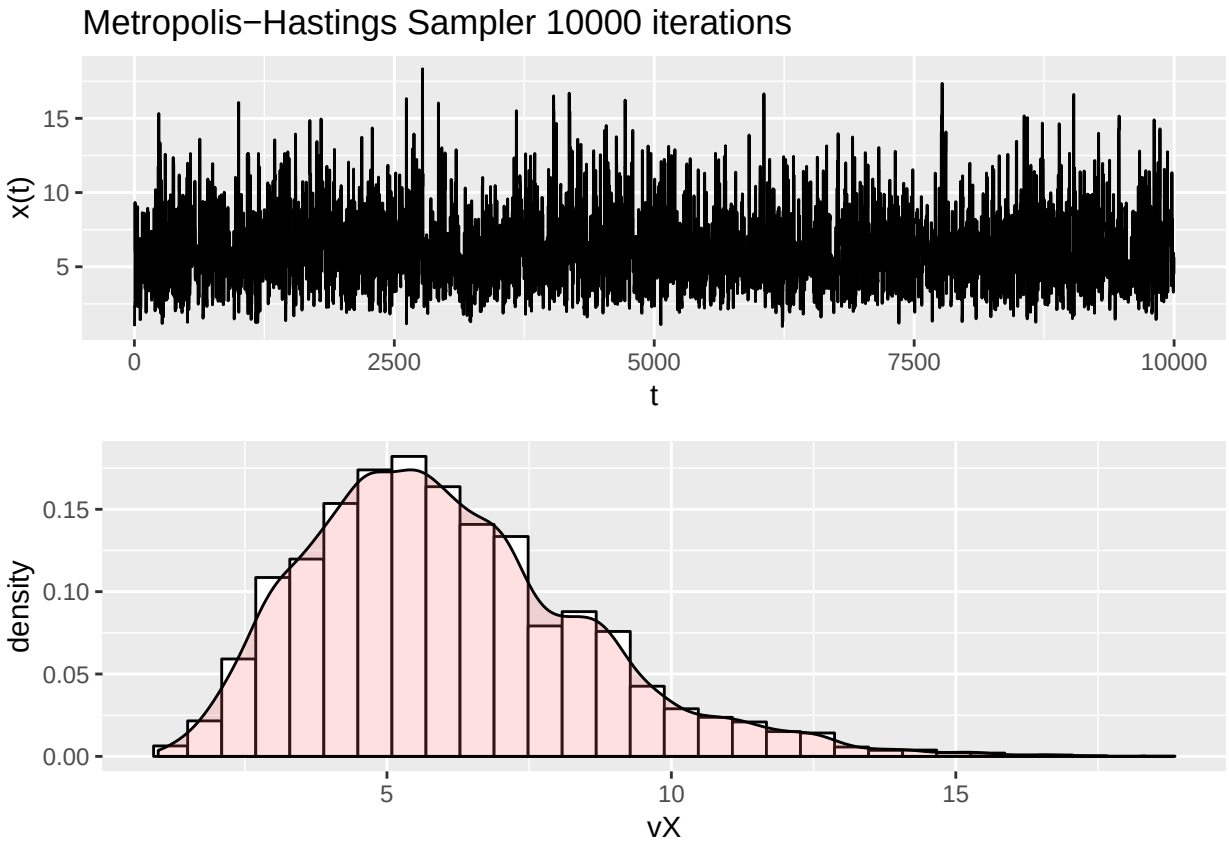
Below you can see the code for the Metropolis-Hastings algorithm. We decided to initialize the starting point (X_0) with 1 and use 10000 for t, as max iterations.

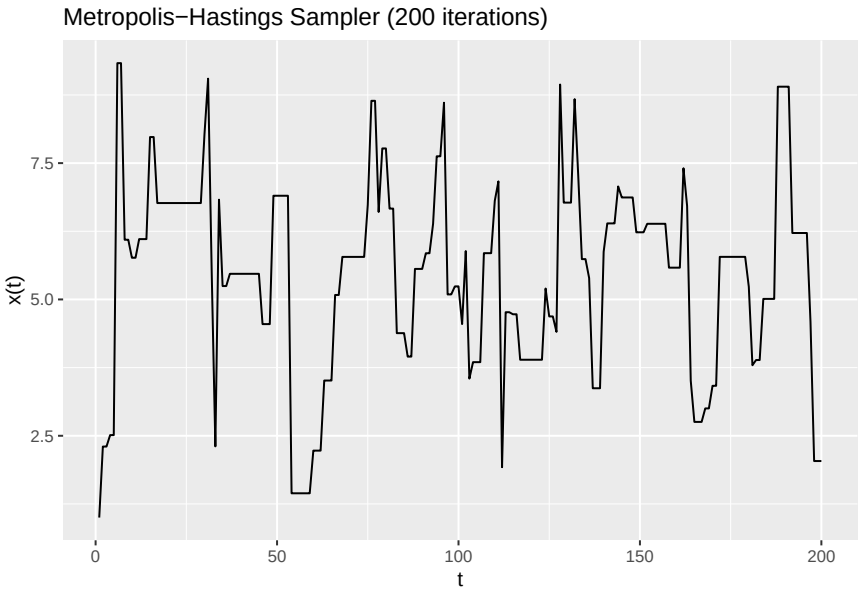
```
# Metropolis-Hastings Sampler

# Initilize chain to X_0, t=0
t_max = 10000
x_0 = 1 # starting point

# algorithm from the slide
MCMC <- function(t_max, x_0){
  rej = 0
  # browser()
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rlnorm(n = 1, meanlog = log(X), sdlog = 1)
    u = runif(1,0,1)
    num = pdf_target(Y) * dlnorm(X, meanlog = log(Y), sdlog = 1)
    den  = pdf_target(X) * dlnorm(Y, meanlog = log(X), sdlog = 1)
    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
```

```
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_1 = MCMC(t_max, x_0)
```

```
## Reject rate:  0.5611
```

Plot of the chain obtained as a time series plot:





The sample points seem to be convergent around 2.5 and 15. We also draw another plot with 200 iterations to find the burn-in period, but it seems that burn-in period is so short (from 0 to 10).

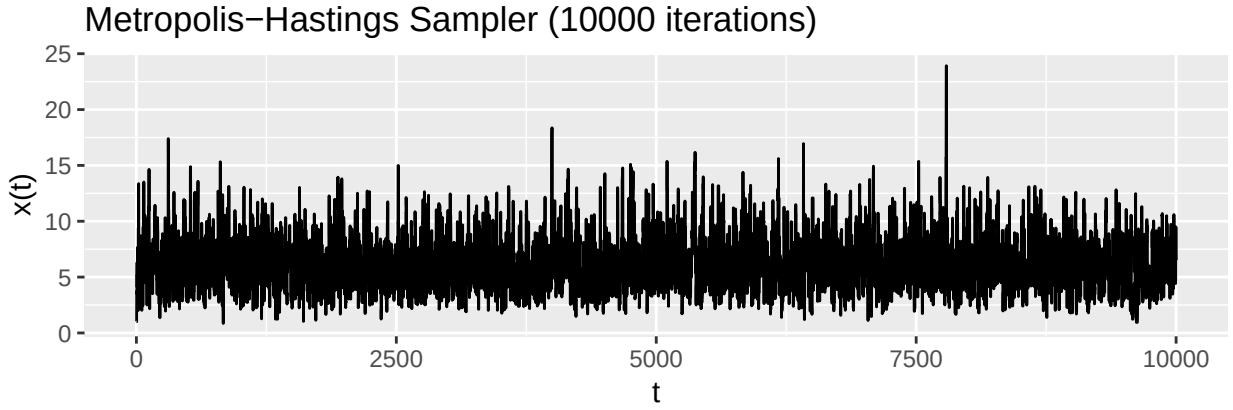
2. Perform Step 1 by using the chi-square distribution

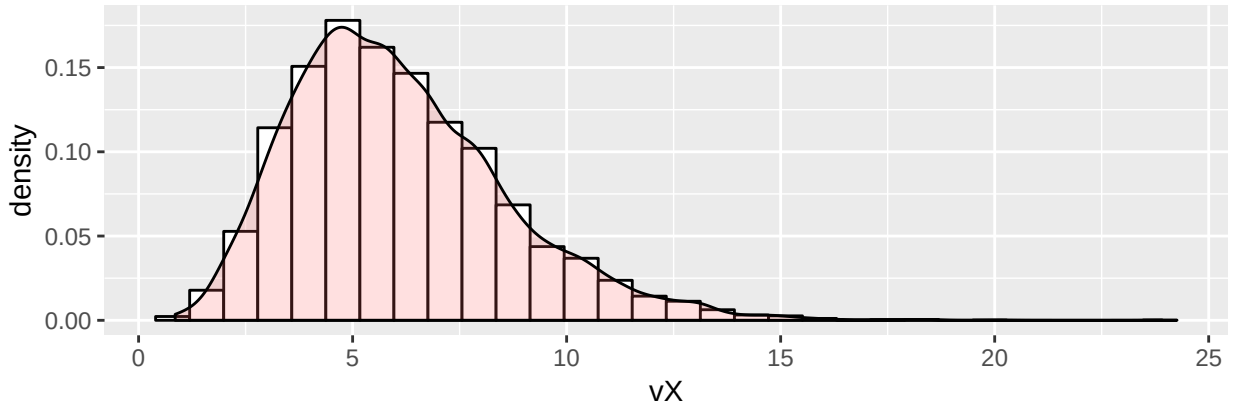
χ^2 ($\lfloor X_t + 1 \rfloor$) as a proposal distribution, where$\lfloor x \rfloor$ is the floor function, meaning the integer part of x for positive x, i.e.$\lfloor 2.95 \rfloor = 2$

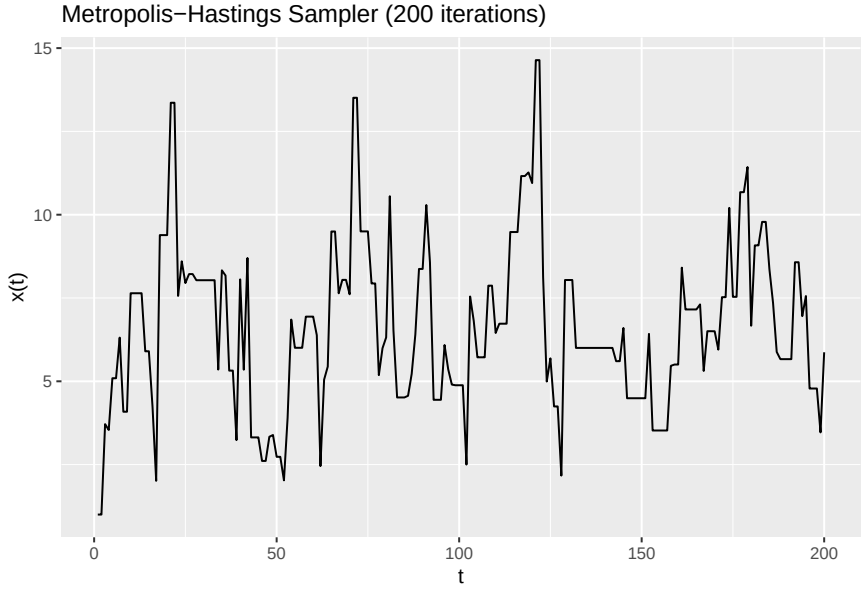
Metropolis-Hastings algorithm with chi-square as proposal distribution

```
MCMC2 <- function(t_max, x_0){
  rej = 0
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rchisq(1,df=floor(X+1))
    u = runif(1,0,1)
    num = pdf_target(Y)*dchisq(X,floor(Y+1))
    den = pdf_target(X)*dchisq(Y,floor(X+1))

    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_2 = MCMC2(t_max, x_0)
```

```
## Reject rate:  0.4005
```







3. Compare the results of Steps 1 and 2 and make conclusions.

In step 2, the MH method based on chi-square distribution has similar burn-in period (0~10) and convergent interval (2.5~15). However, the rejection rate of the method in step 2 is around 0.4, but in step 1 is more than 0.5, which means that chi-square distribution might be a better proposal distribution than log-normal

distribution to generate the samples of our target distribution.

4. Generate 10 MCMC sequences

using the generator from Step 2 and starting points 1, 2, . . . , or 10. Use the Gelman-Rubin method to analyze convergence of these sequences.

```
# Generation 10 MCMC sequences with start points 1~10
set.seed(123456)
df <- data.frame(vN=1:t_max)
for (i in 1:10) {
  df <- cbind(df,MCMC2(t_max,i)$vX)
}
```

```
## Reject rate:  0.3988
## Reject rate:  0.4042
## Reject rate:  0.3955
## Reject rate:  0.3852
## Reject rate:  0.3997
## Reject rate:  0.4016
## Reject rate:  0.407
## Reject rate:  0.4011
## Reject rate:  0.4001
## Reject rate:  0.3959
```

```
mcmc_list = list()

for (i in 1:10){
  mcmc_list[[i]] = as.mcmc(df[,i+1], start = i)
}

# Gelman-Rubin method:
gelman.diag(mcmc_list)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

Since both potential scale reduction factor and their upper confidence limits are 1's, we can ensure that our samples are convergent.

5. Estimate

$$\int_0^\infty x f(x) dx$$

using the samples from Steps 1 and 2.

Down here follows the rule on how to calculate a definite integral based on lecture slide.

To estimate a θ which is

$$\theta = \int_D f(x) dx,$$

we can decompse such $f(x)$ into:

$$f(x) = g(x)p(x),$$

where

$$\int_D p(x)dx = 1.$$

- Thus, if $X \sim p(.)$ and

$$\theta = E[g(X)] = \int_D g(x)p(x)dx,$$

we can conclude that

$$\hat{\theta} = \frac{1}{n}\sum_i^n g(x_i),\ \forall_i \sim p(.)$$

Our target function is a probability density function, whose integral must be 1. Since we already have the samples from the target distribution by Metropolis-Hastings algorithm, the final integral of $xf(x)$ can be estimated as the mean of such samples.

Mean of sample 1:

```
## [1] 6.061493
```

Mean of sample 2:

```
## [1] 6.085729
```

6. In fact

The distribution generated is a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

PDF of gamma distribution $X \approx \Gamma(\alpha, \beta)$:

$$f(x;\alpha,\beta) = \frac{\beta^\alpha x^{\alpha-1}e^{-\beta x}}{\Gamma(\alpha)}$$

for $x > 0$, and $\alpha, \beta > 0$.

Based on the standard formula, we know that $\alpha = 6$ and $\beta = 1$ in our target PDF. Thus, we can calculate the mean of our target PDF as

$$E[X] = \frac{\alpha}{\beta} = \frac{6}{1} = 6,$$

which closes to our estimated results.

Question 2: Gibbs sampling

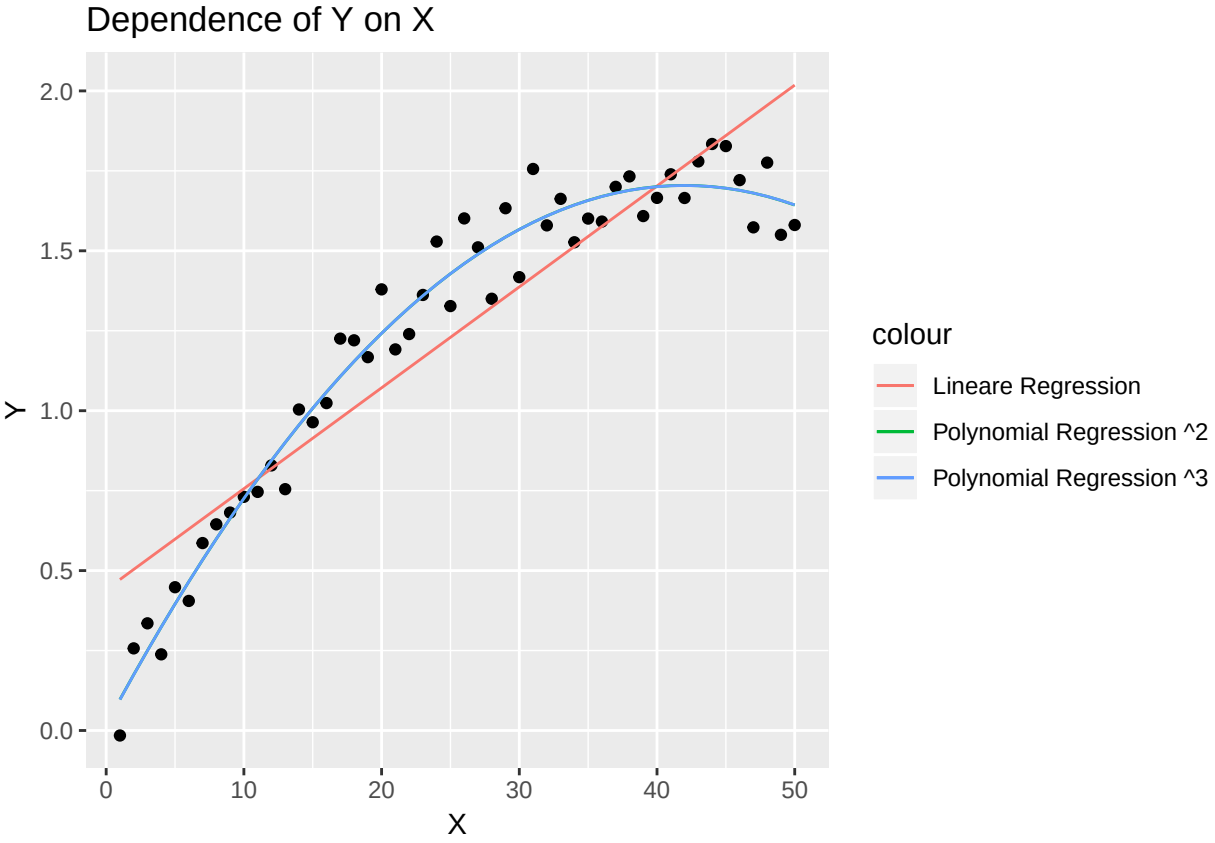
A concentration of a certain chemical was measured in a water sample, and the result was stored in the data *chemical.RData* having the following variables:

- X: day of the measurement
- Y: measured concentration of the chemical.

The instrument used to measure the concentration had certain accuracy; this is why the measurements can be treated as noisy. Your purpose is to restore the expected concentration values.

1. Import the data to R and plot the dependence of Y on X.

What kind of model is reasonable to use here?



It could already be seen that a linear regression does not describe the data too much. However, a second degree polynomial regression is well consistent with the course of the data. As expected can be seen that the progression of polynomials 2 and 3 overlap, so a second grade is sufficient.

2. Present the formula showing the likelihood $p(\vec{Y} \mid \vec{\mu})$ and the prior $p(\vec{\mu})$

A researcher has decided to use the following (random-walk) Bayesian model (n=number of observations, $\vec{\mu} = (\mu_1, ..., \mu_n)$ are unknown parameters):

$$Y_i = \mathcal{N}(\mu, \sigma = 0.2), \quad i = 1, ..., n$$

where the prior is

$$p(\mu_1) = 1$$

$$p(\mu_{i+1} \mid \mu_i) = \mathcal{N}(\mu_i, 0.2) \quad i = 1, ..., n-1$$

Present the formulae showing the likelihood and the prior: $p(\vec{Y} \mid \vec{\mu})$ and $p(\vec{\mu})$.

Likelihood:

$$\begin{aligned}\mathcal{L}[p(\vec{Y}|\vec{\mu}, 0.2)] =& \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi \cdot 0.2}} \exp(-\frac{(y_i - \mu_i)^2}{2 \cdot 0.2}) \\ =& \Big(\frac{1}{\sqrt{0.4\pi}}\Big)^n \exp\Big(-\frac{\sum_{i=1}^{n}(y_i - \mu_i)^2}{0.4}\Big)\end{aligned}$$

Prior:

$$\begin{aligned}p(\vec{\mu}) &= p(\mu_1) \cdot p(\mu_2|\mu_1) \cdot p(\mu_3|\mu_2) \cdot \cdots \cdot p(\mu_n|\mu_{n-1}) \\ &= 1 \cdot \prod_{i=2}^{n} p(\mu_n|\mu_{n-1}) \\ &= \frac{1}{\sqrt{0.4\pi}} \exp(-\frac{(\mu_2 - \mu_1)^2}{0.4}) \cdots \exp(-\frac{(\mu_n - \mu_{n-1})^2}{0.4}) \\ &= \Big(\frac{1}{\sqrt{0.4\pi}}\Big)^{n-1} \exp(-\frac{\sum_{i=2}^{n}(\mu_i - \mu_{i-1})^2}{0.4})\end{aligned}$$

3. Use Bayes' Theorem

to get the posterior up to a constant proportionality, and then fnd out the distributions of $(\mu_i|\vec{\mu}_{-i}, \vec{Y})$, where μ_{-i} is a vector containing all μ values except of μ_i.

Firstly, we calculate

$$\begin{aligned}p(\vec{\mu}, \vec{Y}) =& \ p(\vec{Y}|\vec{\mu}) \cdot p(\vec{\mu}) \\ \propto& \ \exp\Big(-\frac{\sum_{i=1}^{n}(y_i - \mu_i)^2}{0.4}\Big) \cdot \exp(-\frac{\sum_{i=2}^{n}(\mu_i - \mu_{i-1})^2}{0.4}) \\ =& \ \exp(-\frac{\sum_{i=1}^{n}(y_i - \mu_i)^2 + \sum_{i=2}^{n}(\mu_i - \mu_{i-1})^2}{0.4}) \quad .\end{aligned}$$

Since

$$p(\mu_i|\vec{\mu}_{-i}, \vec{Y}) \cdot p(\vec{\mu}_{-i}, \vec{Y}) = p(\mu_i, \vec{\mu}_{-i}, \vec{Y}) = p(\vec{\mu}, \vec{Y}) \quad ,$$

we can conclude that

$$\begin{aligned}p(\mu_1|\vec{\mu}_{-1}, \vec{Y}) =& \ \frac{p(\vec{\mu}, \vec{Y})}{p(\vec{\mu}_{-1}, \vec{Y})} \\ \propto& \ \exp\Big(-\frac{(y_1 - \mu_1)^2 + (\mu_2 - \mu_1)^2}{2\sigma^2}) \\ \propto& \ \exp(-\frac{(\mu_1 - (y_1 + \mu_2)/2)^2}{2\sigma^2/2}) \quad ,\end{aligned}$$

and

$$\begin{aligned}p(\mu_n|\vec{\mu}_{-n}, \vec{Y}) =& \ \frac{p(\vec{\mu}, \vec{Y})}{p(\vec{\mu}_{-n}, \vec{Y})} \\ \propto& \ \exp\Big(-\frac{(y_n - \mu_n)^2 + (\mu_n - \mu_{n-1})^2}{2\sigma^2}) \\ \propto& \ \exp(-\frac{(\mu_n - (y_n + \mu_{n-1})/2)^2}{2\sigma^2/2}) \quad ,\end{aligned}$$

and

$$\begin{aligned} p(\mu_i|\vec{\mu}_{-i},\vec{Y}) = & \frac{p(\vec{\mu},\vec{Y})}{p(\vec{\mu}_{-i},\vec{Y})} \\ \propto & \exp\Big(-\frac{(y_i-\mu_i)^2+(\mu_{i+1}-\mu_i)^2+(\mu_i-\mu_{i-1})^2}{2\sigma^2}\Big) \\ \propto & \exp(-\frac{(\mu_i-(y_i+\mu_{i-1}+\mu_{i+1})/3)^2}{2\sigma^2/3}), \quad i\in(1,n). \end{aligned}$$

Thus, the results of our deduction can be shown as

$$(\mu_i|\vec{\mu}_{-i},\vec{Y}) \sim \begin{cases} N(\frac{y_1+\mu_2}{2},0.1) & i=1 \\ N(\frac{y_i+\mu_{i-1}+\mu_{i+1}}{3},\frac{0.2}{3}) & Otherwise \\ N(\frac{y_n+\mu_{n-1}}{2},0.1) & i=n \end{cases}$$

4.Use the distributions derived in Step 3 to implement a Gibbs sampler

that uses $\vec{\mu}^0=(0,...,0)$ as a starting point. Run the Gibbs sampler to obtain 1000 values of $\vec{\mu}$ and then compute the expected value of $\vec{\mu}$ by using a Monte Carlo approach. Plot the expected value of $\vec{\mu}$ versus X and Y versus X in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of $\vec{\mu}$ can catch the true underlying dependence between Y and X?

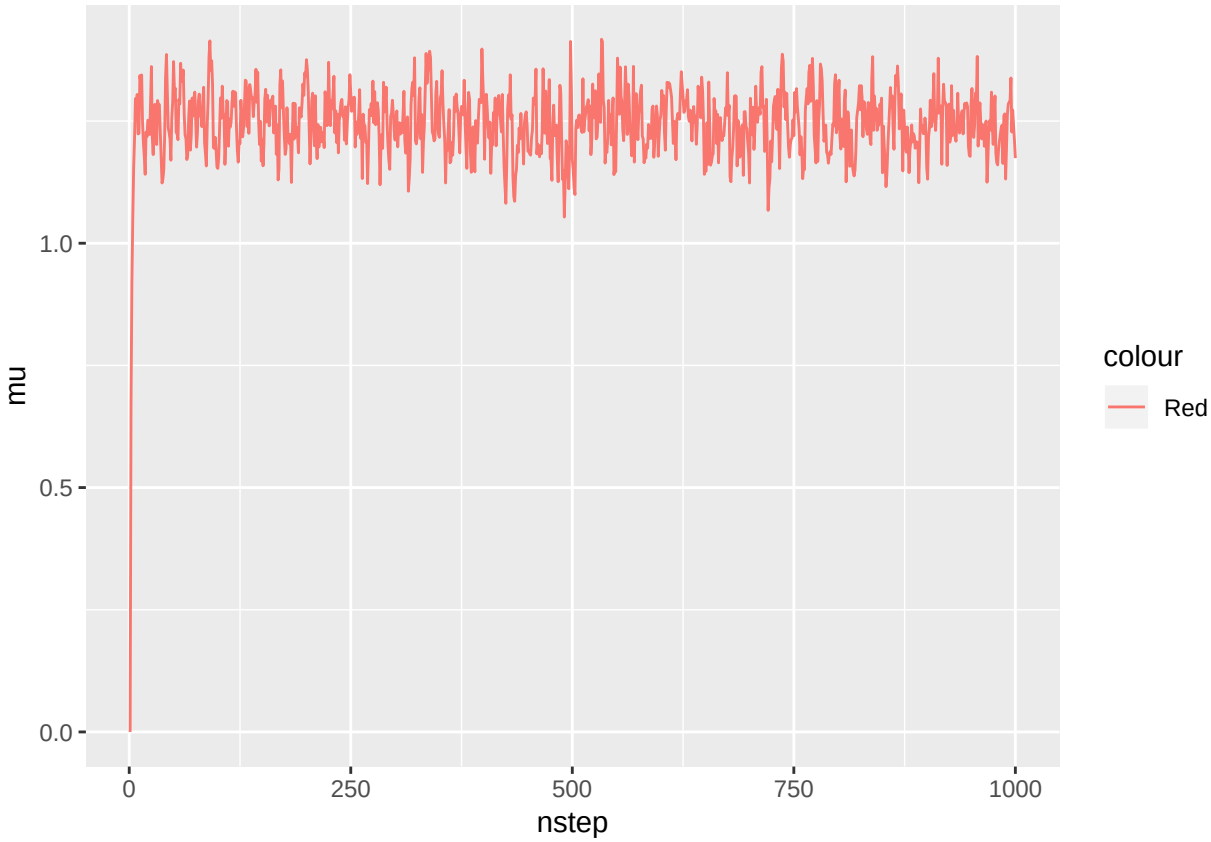
```
nstep = 1000
d = length(Y)
mu0 = rep(0,d)

gibbs = function(nstep, mu0, y){
    d <- length(mu0)
    mat_mu <- matrix(0, nrow=nstep, ncol=d)
    mat_mu[1,] <- mu0

    for (t in 2:nstep) {
      mat_mu[t,1] = rnorm(1,(y[1]+mat_mu[t-1,2])/2,sqrt(0.1))
      for (i in 2:(d-1)) {
        mat_mu[t,i] = rnorm(1,(y[i]+mat_mu[t,i-1]+mat_mu[t-1,i+1])/3,sqrt(0.2/3))
      }
      mat_mu[t,d] = rnorm(1,(y[d]+mat_mu[t,d-1])/2,sqrt(0.1))
    }
    mat_mu
}

re = gibbs(nstep, mu0, Y)
```

Expected value of $\vec{\mu}$ versus X and Y versus X:

In the generated visualization a very similar course of Y and mu can be seen. This means that the noise could be removed well. In the area the variance is greatest. It can also be said that the expected value of $\vec{\mu}$ can catch the true underlying dependence between Y and X well.

5.Trace plot

Make a trace plot for μ_n and comment on the burn-in period and convergence.



In the visualization can we see an early burn-in, already after some interactions does it converges.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE, out.width = "480px")
# libraries used in this lab
library(ggplot2)
# install.packages("coda")
library(coda)
library(gridExtra)
# infromation & marterial
# vey nice blog - MCMC and the Metropolis-Hastings algorithm
#https://blog.stata.com/2016/11/15/introduction-to-bayesian-statistics-part-2-mcmc-and-the-metropolis-h
rm(list=ls())
set.seed(123456)
#pdf -  probability density function
pdf_target = function(x){
  if(x<= 0){
    stop("x needs to be bigger than 0")
  }
  # stopifnot(x>0) # check - x >0
  return(x^5 * exp(-x))
}
# Metropolis-Hastings Sampler

# Initilize chain to X_0, t=0
t_max = 10000
x_0 = 1 # starting point

# algorithm from the slide
MCMC <- function(t_max, x_0){
  rej = 0
  # browser()
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rlnorm(n = 1, meanlog = log(X), sdlog = 1)
    u = runif(1,0,1)
    num = pdf_target(Y) * dlnorm(X, meanlog = log(Y), sdlog = 1)
    den  = pdf_target(X) * dlnorm(Y, meanlog = log(X), sdlog = 1)
    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_1 = MCMC(t_max, x_0)
```

```r
# create time series plot

# create the plot object
points11= ggplot(data = data_1_1, aes(x= vN, y= vX)) +
  geom_line()+
  ggtitle("Metropolis-Hastings Sampler 10000 iterations")+
  ylab("x(t)")+
  xlab("t")

density11= ggplot(data_1_1,aes(x=vX))+
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)+
  geom_density(alpha=.2, fill="#FF6666")

grid.arrange(points11,density11,nrow=2)
ggplot(data = data_1_1[1:200,], aes(x= vN, y= vX)) +
  geom_line()+
  ggtitle("Metropolis-Hastings Sampler (200 iterations)")+
  ylab("x(t)")+
  xlab("t")

MCMC2 <- function(t_max, x_0){
  rej = 0
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rchisq(1,df=floor(X+1))
    u = runif(1,0,1)
    num = pdf_target(Y)*dchisq(X,floor(Y+1))
    den = pdf_target(X)*dchisq(Y,floor(X+1))

    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_2 = MCMC2(t_max, x_0)
# create the plot - t_max = 10.000

# create the plot object
points12= ggplot(data = data_1_2, aes(x= vN, y= vX)) +
  geom_line()+
```

```
  ggtitle("Metropolis-Hastings Sampler (10000 iterations)")+
  ylab("x(t)")+
  xlab("t")

density12= ggplot(data_1_2,aes(x=vX))+
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)+
  geom_density(alpha=.2, fill="#FF6666")

grid.arrange(points12,density12,nrow=2)

# create the plot object
ggplot(data = data_1_2[1:200,], aes(x= vN, y= vX)) +
  geom_line()+
  ggtitle("Metropolis-Hastings Sampler (200 iterations)")+
  ylab("x(t)")+
  xlab("t")
# Generation 10 MCMC sequences with start points 1~10
set.seed(123456)
df <- data.frame(vN=1:t_max)
for (i in 1:10) {
  df <- cbind(df,MCMC2(t_max,i)$vX)
}
mcmc_list = list()

for (i in 1:10){
  mcmc_list[[i]] = as.mcmc(df[,i+1], start = i)
}

# Gelman-Rubin method:
gelman.diag(mcmc_list)
mean_set1 = mean(data_1_1$vX)
mean_set1
mean_set2 = mean(data_1_2$vX)
mean_set2
rm(list=ls())
set.seed(123456)
# set working directory
# import the data
load("chemical.RData")
# create table for plot
data = data.frame("X" = X,
                  "Y" = Y)
# plot dependence of Y on X
X_Y_dependence = ggplot(data, aes(x = X, y = Y)) +
  geom_point()+
  ggtitle("Dependence of Y on X") +
  geom_smooth()
# which model would fit the data
# lookes like a polinomial regression -> test until poly 3
model1 = lm(Y ~  X,
```

```
            data = data)
model2 = lm(Y ~  X + I(X^2),
            data = data)
model3 = lm(Y ~  X + I(X^2)+ I(X^3),
            data = data)

cols <- c("Lineare Regression"="#62c76b",
          "Polynomial Regression ^2"="#3591d1",
          "Polynomial Regression ^3"="#f04546")

X_Y_dependence_fit = ggplot(data, aes(x = X, y = Y)) +
  geom_point()+
  ggtitle("Dependence of Y on X") +
  geom_line(aes(x = X, y = predict(model1), colour = "Lineare Regression")) +
  geom_line(aes(x = X, y = predict(model2), colour = "Polynomial Regression ^2")) +
  geom_line(aes(x = X, y = predict(model3), colour = "Polynomial Regression ^3"))
X_Y_dependence_fit
nstep = 1000
d = length(Y)
mu0 = rep(0,d)

gibbs = function(nstep, mu0, y){
    d <- length(mu0)
    mat_mu <- matrix(0, nrow=nstep, ncol=d)
    mat_mu[1,] <- mu0

    for (t in 2:nstep) {
      mat_mu[t,1] = rnorm(1,(y[1]+mat_mu[t-1,2])/2,sqrt(0.1))
      for (i in 2:(d-1)) {
        mat_mu[t,i] = rnorm(1,(y[i]+mat_mu[t,i-1]+mat_mu[t-1,i+1])/3,sqrt(0.2/3))
      }
      mat_mu[t,d] = rnorm(1,(y[d]+mat_mu[t,d-1])/2,sqrt(0.1))
    }
    mat_mu
}

re = gibbs(nstep, mu0, Y)
data_2_4 = data.frame(cbind(X=X,Y=Y, mu=colMeans(re)))
data_melt  <- reshape2::melt(data_2_4, id="X")
ggplot(data_melt, aes(x=X,y=value,color=variable))+
  geom_line()+
  geom_point()

data_gribbs = data.frame(nstep=1:1000,mu=rowMeans(re))
ggplot(data_gribbs, aes(x=nstep, y=mu, color = "Red"))+
  geom_line()
```

Computer Lab 5 Computational Statistics

Contents

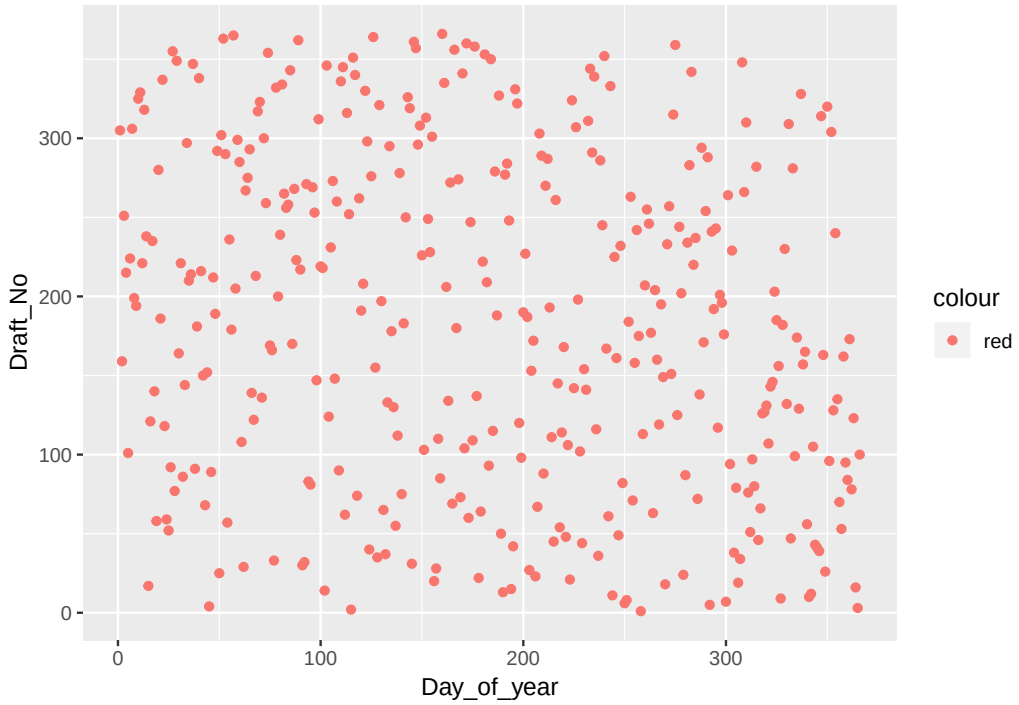
Question 1: Hypothesis testing

In 1970, the US Congress instituted a random selection process for the military draft. All 366 possible birth dates were placed in plastic capsules in a rotating drum and were selected one by one. The first date drawn from the drum received draft number one, the second date drawn received draft number two, etc. Then, eligible men were drafted in the order given by the draft number of their birth date. In a truly random lottery there should be no relationship between the date and the draft number. Your task is to investigate whether or not the draft numbers were randomly selected. The draft numbers (`Y=Draft_No`) sorted by day of year (`X=Day_of_year`) are given in the file *lottery.xls*.

1. Make a scatterplot

of Y versus X and conclude whether the lottery looks random.

Scatterplot of Y versus X

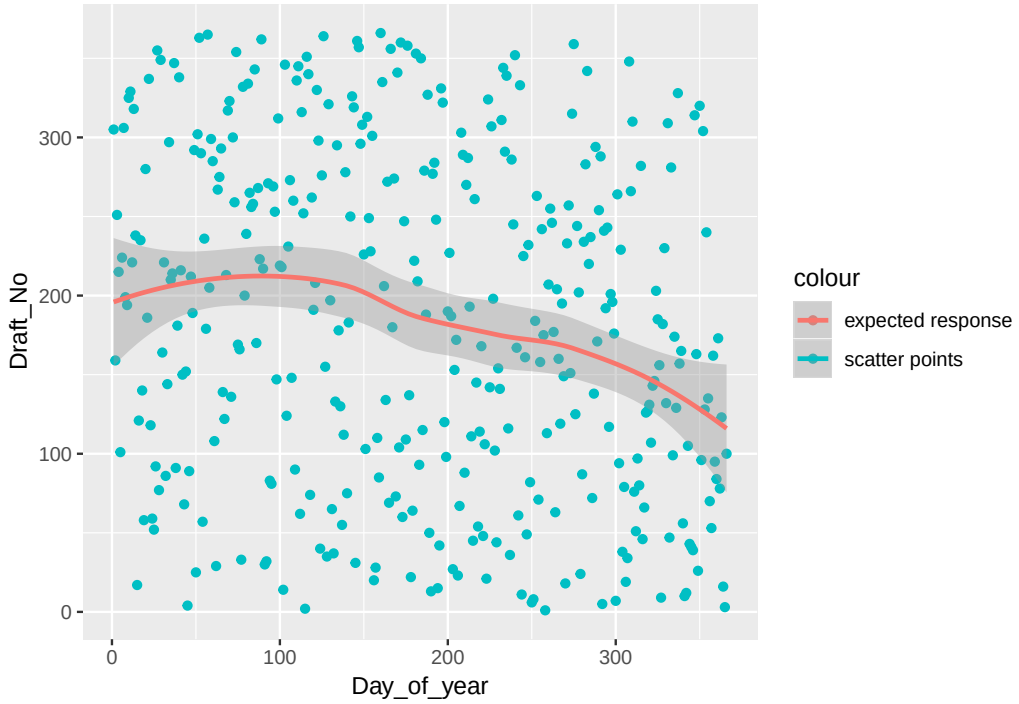


Since the dots are spaced over the whole area, it seems like the military draft is truly random.

2. Compute an estimate

$\hat{Y}$ of the expected response as a function of X by using a loess smoother (use `loess()`), put the curve $\hat{Y}$ versus X in the previous graph and state again whether the lottery looks random.

Scatterplot of $\hat{Y}$ versus X



It does look like the loess smoother decreases slowly after X (= Day of year) passes number 100. This could mean the draft is not truly random.

3. Check whether the lottery is random

To check whether the lottery is random, it is reasonable to use test statistics

$$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a}, \quad \text{where} \quad \mathrm{X_b} = \mathrm{argmax_X Y(X)}, \quad \mathrm{X_a} = \mathrm{argmin_X Y(X)}$$

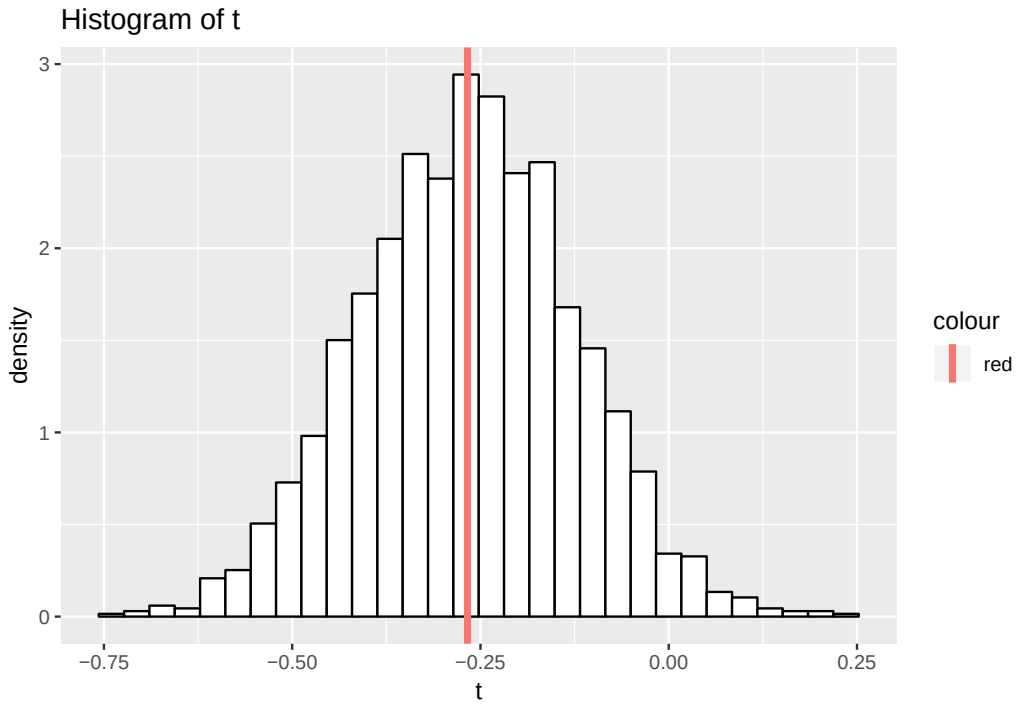
If this value is significantly greater than zero, then there should be a trend in the data and the lottery is not random. Estimate the distribution of T by using a non–parametric bootstrap with $B = 2000$ and comment whether the lottery is random or not. What is the p–value of the test?

```
# use non-parametric bootstrap
# need data = data1, statistic = loess & test stat, R = 2000
stats = function(data,vn){
  datatemp<-data[vn,]
  # Y=Draft_No & X=Day_of_year
  # create the loess regression based on the sample data
  reg = loess(Draft_No ~ Day_of_year, data = datatemp)
  # the X which owns the max/min Y in original data
  X_b = data$Day_of_year[which.max(data$Draft_No)]
  X_a = data$Day_of_year[which.min(data$Draft_No)]
  # Y_hat(X)
  fit_X_b = predict(reg,newdata=X_b)
  fit_X_a = predict(reg,newdata=X_a)
  # the given test statistic
  T_stat = (fit_X_b - fit_X_a) / (X_b-X_a)
  return(T_stat)
```

```
}

#non-parametric bootstrap
set.seed(123456)
# dt=data1[order(data1$Draft_No),]
myboot = boot(data = data1,
              statistic = stats,
              R = 2000)

# summary
myboot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data1, statistic = stats, R = 2000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* -0.2671794 -4.28431e-05    0.142308
```

```
# plot distribution
# plot(myboot, index = 1)
```



The test statistic T is around -0.267, which is smaller than 0. So the lottery is random. We used following calculation to estimate the p-value:

$$\hat{p} = \frac{\sum T(X) \geqslant 0}{B}$$

```r
# calculate the p-value
# null hypothesis: T>=0, hv tendancy
p_val = mean(myboot$t>=0, na.rm = TRUE)
p_val
```

```
## [1] 0.028
```

Here p-value is smaller than significant level $\alpha = 0.05$, it also shows the data is random. So it seems the linear relation of lottery is not exist.

4. Tests the hypothesis

Implement a function depending on *data* and B that tests the hypothesis:

- *H0*: Lottery is random
- *H1*: Lottery is non–random

by using a permutation test with statistics `T`. The function is to return the p–value of this test. Test this function on our data with 'B = 2000.

```r
set.seed(123456)
permutation_test <- function(B, data1){
  n = dim(data1)[1]
  stat = numeric(B)
  for(b in 1:B){
    # randoming X
    Gb = sample(data1$Day_of_year, n)
    data11 <- data1
    data11$Day_of_year <- Gb
    stat[b] <- stats(data11,1:n)
  }
  return(stat)
}

stat <- permutation_test(B=2000, data1)
stat0 <- stats(data1,1:nrow(data1))

re = data.frame( stat0 ,mean(abs(stat)>=abs(stat0)) )
colnames(re) <- c("average_statistic","p_value")
re
```

```
##   average_statistic p_value
## 1        -0.2671794  0.0955
```

Since the the p-value is larger than 0.05, we cannot reject the null hypothesis.

5.Make a crude estimate

Make a crude estimate of the power of the test constructed in Step 4:

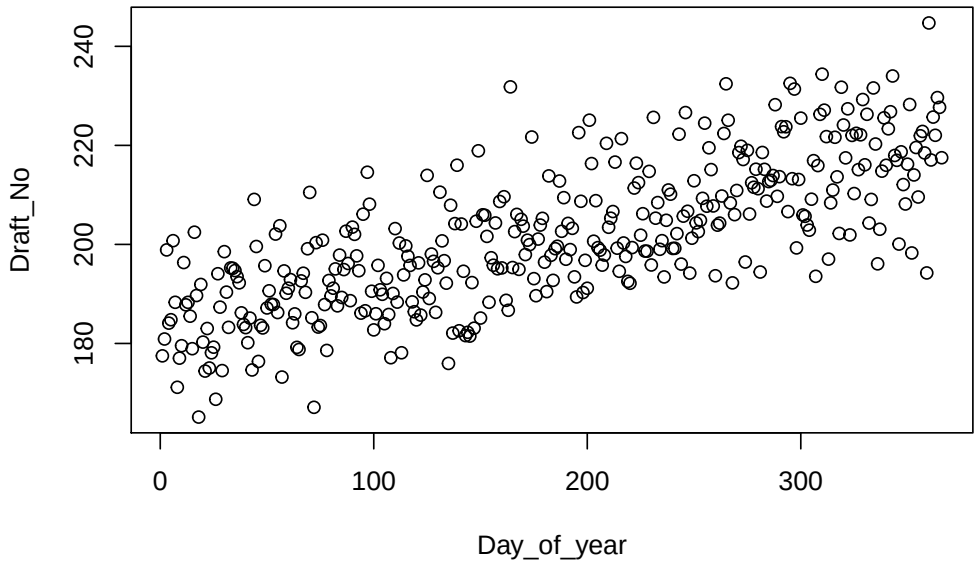
(a) Generate (an obviously non-random) dataset with n = 366 observations by using same X as in the original data set and $Y(x) = \max(0, \min(\alpha x + \beta, 366))$, where $\alpha = 0.1$ and $\beta \sim N(183, sd = 10)$.

(b) Plug these data into the permutation test with B = 200 and note whether it was rejected.

(c) Repeat Steps 5a-5b for $\alpha = 0.2, 0.3, ..., 10$.

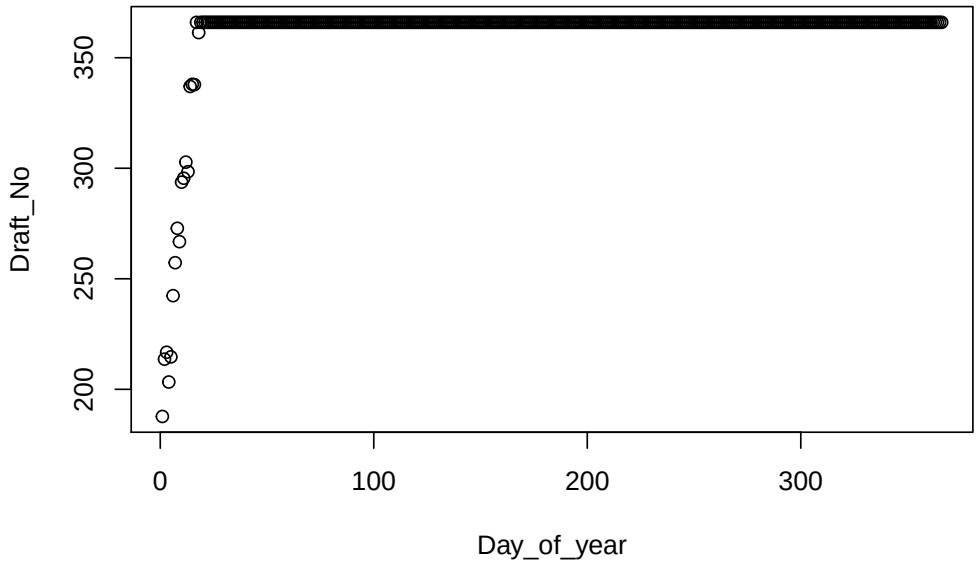
What can you say about the quality of your test statistics considering the value of the power?

```
set.seed(123)
ce <- function(a, plot=FALSE){
  ## step (a)
  n <- 366
  b <- rnorm(n, mean=183, sd = 10)
  X <- data1$Day_of_year
  Y <- sapply(1:n,function(i){
    t <- min(a*X[i]+b[i],366)
    t <- max(0,t)
  })
  data15 <- data.frame(Day_of_year=X,Draft_No=Y)
  if(plot){
    plot(data15)
  }
  ## step (b)
  stat <- permutation_test(B=200, data15)

  stat0 <- stats(data15,1:n)
  re = data.frame( stat0 ,mean(abs(stat)>=abs(stat0)) )
  options(digits = 5)
  colnames(re) <- c("test_statistic","p_value")
  return(re)
}

ce(a=0.1, plot=TRUE)
```




```
##   test_statistic p_value
## 1       0.099556       0
```

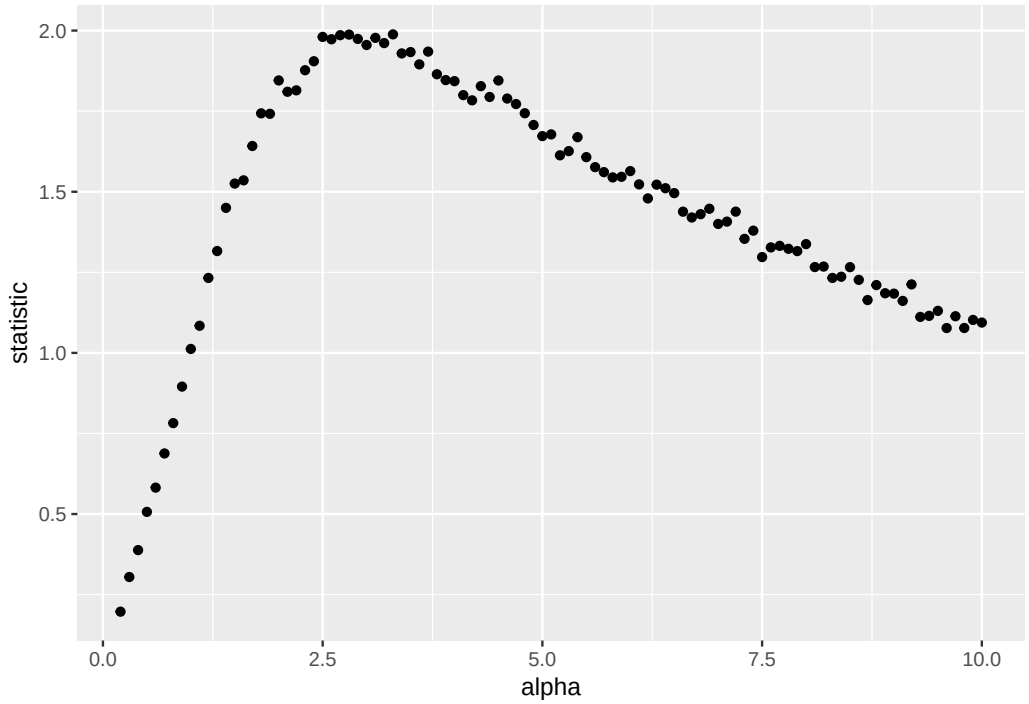
```
ce(a=10, TRUE)
```




```
##   test_statistic p_value
## 1         1.0911       0
```

Since the p-values here are smaller than 0.05 and test-statistics are larger than 0, so we can reject the null

hypothesis directly. Obviously, the data is non-random.



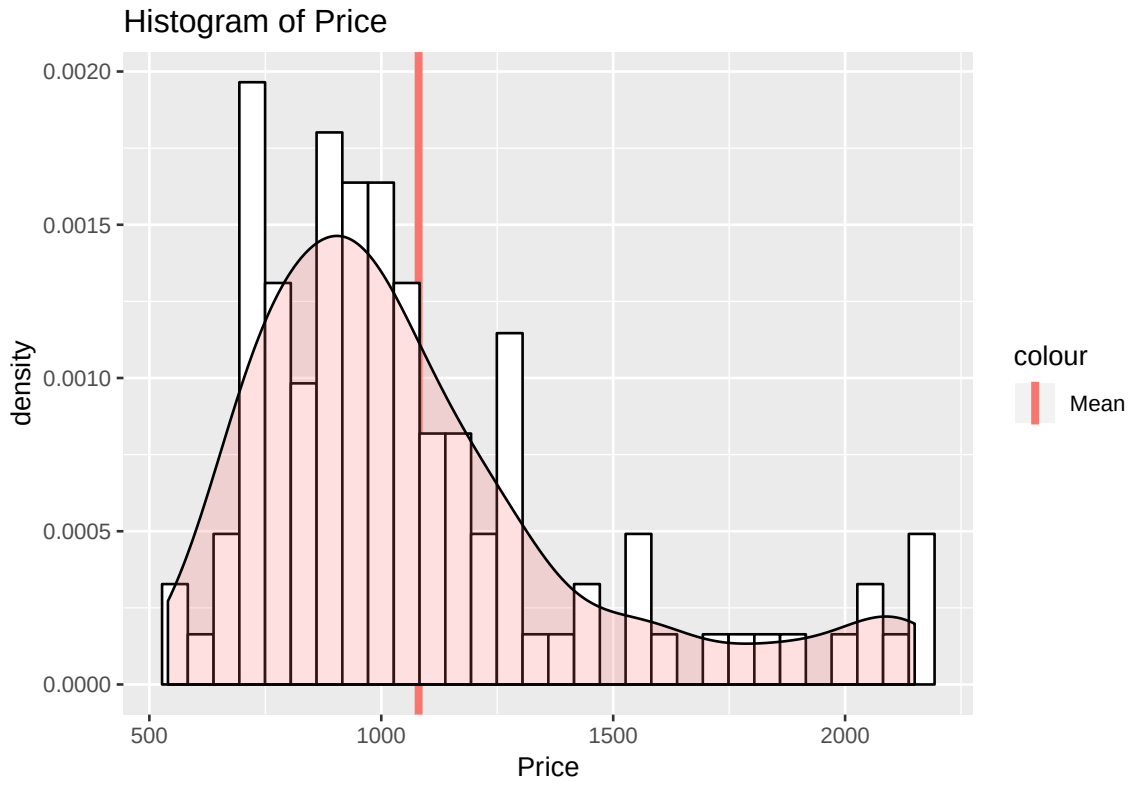
Based on the plot of alpha and test-statistic, we know that the test statistic we use is very strong. It expresses the non-randomness of our data and tends to 1.

Question 2: Bootstrap, jackknife and confidence intervals

The data you are going to continue analyzing is the database of home prices in Albuquerque, 1993. The variables present are `Price`; `SqFt`: the area of a house; `FEATS`: number of features such as dishwasher, refrigerator and so on; 'Taxes: annual taxes paid for the house. Explore the file *prices1.xls*.

1. Plot the histogram of Price.

Does it remind any conventional distribution? Compute the mean price.



The present distribution looks like a possion or gamma distribution.

The mean of the price is:

```
price_mean
```

```
## [1] 1080.5
```

2. Bootstrap

Estimate the distribution of the mean price of the house using bootstrap. Determine the bootstrap bias-correction and the variance of the mean price. Compute a 95% confidence interval for the mean price using bootstrap percentile, bootstrap BCa, and first-order normal approximation.

Estimate the distribution of the mean price of the house using bootstrap:

```
# function - estiamte mean
stat1 <- function(vec,vn){
    return(mean(vec[vn]))
```

```
}
B=1000

set.seed(123456)
res1 = boot(data2$Price, stat1, R=B)
res1
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data2$Price, statistic = stat1, R = B)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*   1080.5  1.6962      36.157
```

According to the slide, we know that the bias-corrected estimator is

$$T_1 := 2T(D) - \frac{1}{B}\sum_{i=1}^{B} T_i^* \ ,$$

and variance of estimator by bootstrap is

$$\widehat{Var[T(.)]} = \frac{1}{B-1}\sum_{i=1}^{B}(T(D_i^*) - \overline{T(D^*)})^2.$$

Determine the bootstrap bias-correction and the variance of the mean price:

```
# bias correcred estimator
2*res1$t0-mean(res1$t)
```

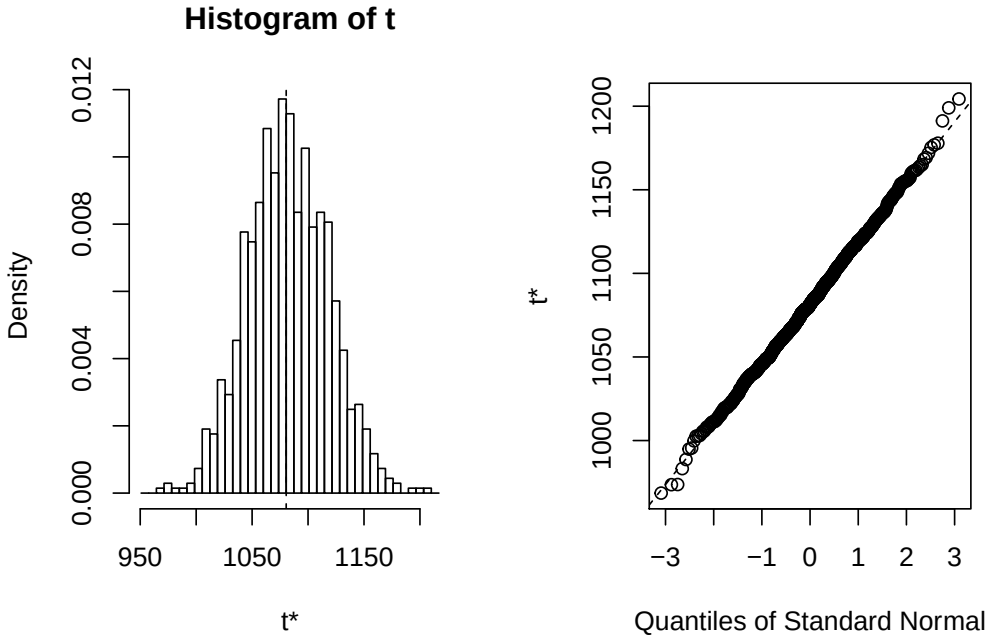
```
## [1] 1078.8
```

```
# variance of mean price (output of statistic)
var_boot <- 1/(B-1)*sum((res1$t-mean(res1$t))^2 )
var_boot
```

```
## [1] 1307.3
```

Compute a 95% confidence interval for the mean price using bootstrap percentile, bootstrap BCa, and first-order normal approximation

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = res1, type = c("perc", "bca", "norm"))
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (1008, 1150 )   (1012, 1155 )   (1014, 1155 )
## Calculations and Intervals on Original Scale
```



3. Jackknife

Estimate the variance of the mean price using the jackknife and compare it with the bootstrap estimate.

- Jackknife (n = B)

$$\widehat{Var[T(.)]} = \frac{1}{n(n-1)} \sum_{i=1}^{n} ((\mathrm{T_i^*}) - \mathrm{J(T)})^2,$$

where

$$T_i^* = nT(D) - (n-1)T(D_i^*)$$

and

$$J(T) = \frac{1}{n} \sum_{i=1}^{n} T_i^*$$

```
# other jackknife function
# library(bootstrap)
# res2 <- jackknife(x=data2$Price,stat1)

n = nrow(data2)
constant = 1/(n*(n-1))

T_i_star = sapply(1:n, function(i){
  n * mean(data2$Price) - (n-1) * mean(data2[-i,1])
})

J_T = (1/n) * sum(T_i_star)

Var_T_jackknife = constant * sum((T_i_star - J_T)^2)
Var_T_jackknife
```

```
## [1] 1320.9
```

boostrap	jackknife
1307.3	1320.9

It seems that bootstrap here is better than jackknife because of its smaller variance.

4. Compare the confidence intervals

obtained with respect to their length and the location of the estimated mean in these intervals.

	Len.CI	Esti.mean
percent	143.01	1083.5
bca	141.95	1084.5
norm	141.73	1078.8

On the one hand, percentile has the largest confidence intervals, but BCa and normal approximation have similar confidence intervals. On the other hand, the interval-means of percentile and BCa are similar (≈ 1084), but of normal approximation is around 1079.

Appendix

```
Sys.setlocale(locale = "English")
knitr::opts_chunk$set(echo = TRUE, out.height = "270px")
# libraries used in this lab
library(ggplot2)
#install.packages("boot") # to create booststarp
library(boot)
# scatterplot
rm(list=ls())
data1 = read.csv2("lottery.csv")

plot_11 = ggplot(data = data1,aes(x = Day_of_year, y = Draft_No)) +
  geom_point(aes(color = "red"))
plot_11
# loess(formula = Draft_No~Day_of_year, data = data1)

plot_11 +
  # geom_smooth()` using method = 'loess' and formula 'y ~ x'
  geom_smooth(method ="loess", formula = y~x, aes(color="blue")) +
  scale_color_discrete(labels=c("expected response",
                                "scatter points")
                       )
# use non-parametric bootstrap
# need data = data1, statistic = loess & test stat, R = 2000
stats = function(data,vn){
  datatemp<-data[vn,]
  # Y=Draft_No & X=Day_of_year
  # create the loess regression based on the sample data
  reg = loess(Draft_No ~ Day_of_year, data = datatemp)
  # the X which owns the max/min Y in original data
  X_b = data$Day_of_year[which.max(data$Draft_No)]
  X_a = data$Day_of_year[which.min(data$Draft_No)]
  # Y_hat(X)
  fit_X_b = predict(reg,newdata=X_b)
  fit_X_a = predict(reg,newdata=X_a)
  # the given test statistic
  T_stat = (fit_X_b - fit_X_a) / (X_b-X_a)
  return(T_stat)
}

#non-parametric bootstrap
set.seed(123456)
# dt=data1[order(data1$Draft_No),]
myboot = boot(data = data1,
              statistic = stats,
              R = 2000)

# summary
myboot
# plot distribution
# plot(myboot, index = 1)
d <- data.frame(t=myboot$t)
```

```r
mean_t = mean(myboot$t,na.rm = TRUE)
ggplot(data = d, aes(x = t)) +
  ggtitle("Histogram of t") +
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30) +
  geom_vline(aes(xintercept = mean_t, color = "red"),size=1.5)
# calculate the p-value
# null hypothesis: T>=0, hv tendancy
p_val = mean(myboot$t>=0, na.rm = TRUE)
p_val
set.seed(123456)
permutation_test <- function(B, data1){
  n = dim(data1)[1]
  stat = numeric(B)
  for(b in 1:B){
    # randoming X
    Gb = sample(data1$Day_of_year, n)
    data11 <- data1
    data11$Day_of_year <- Gb
    stat[b] <- stats(data11,1:n)
  }
  return(stat)
}

stat <- permutation_test(B=2000, data1)
stat0 <- stats(data1,1:nrow(data1))

re = data.frame( stat0 ,mean(abs(stat)>=abs(stat0)) )
colnames(re) <- c("average_statistic","p_value")
re

set.seed(123)
ce <- function(a, plot=FALSE){
  ## step (a)
  n <- 366
  b <- rnorm(n, mean=183, sd = 10)
  X <- data1$Day_of_year
  Y <- sapply(1:n,function(i){
    t <- min(a*X[i]+b[i],366)
    t <- max(0,t)
  })
  data15 <- data.frame(Day_of_year=X,Draft_No=Y)
  if(plot){
    plot(data15)
  }
  ## step (b)
  stat <- permutation_test(B=200, data15)

  stat0 <- stats(data15,1:n)
  re = data.frame( stat0 ,mean(abs(stat)>=abs(stat0)) )
  options(digits = 5)
```

```r
  colnames(re) <- c("test_statistic","p_value")
  return(re)
}

ce(a=0.1, plot=TRUE)
ce(a=10, TRUE)
## step (c)
b <- c()
a <- seq(0.2,10,0.1)
for (i in 1:length(a)) {
  re = ce(a[i])
  b[i] <- re$test_statistic
}
result <- data.frame(alpha=a, statistic=b)
ggplot(data = result,aes(x=alpha,y=statistic))+
  geom_point()
rm(list = ls())
data2 = read.csv2("prices1.csv")
# calculate the mean of the price
price_mean = mean(data2$Price) # 1080.473

# create the plot - hist + distribution + mean
ggplot(data = data2, aes(x = Price)) +
  #geom_histogram(color = "#33CCFF", fill = "#33CCFF") +
  ggtitle("Histogram of Price") +
  geom_vline(aes(xintercept = price_mean, color = "Mean"),size=1.5) +
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)+
  geom_density(alpha=.2, fill="#FF6666")
  # geom_density(kernel = "poisson") create a possion distribution
price_mean
# function - estiamte mean
stat1 <- function(vec,vn){
    return(mean(vec[vn]))
}
B=1000

set.seed(123456)
res1 = boot(data2$Price, stat1, R=B)
res1
# bias correcred estimator
2*res1$t0-mean(res1$t)

# variance of mean price (output of statistic)
var_boot <- 1/(B-1)*sum((res1$t-mean(res1$t))^2 )
var_boot

# default is a 95% confidence interval
ci <- boot.ci(res1, type = c("perc", "bca", "norm"))
print(ci)
```

```r
plot(res1)
# other jackknife function
# library(bootstrap)
# res2 <- jackknife(x=data2$Price,stat1)

n = nrow(data2)
constant = 1/(n*(n-1))

T_i_star = sapply(1:n, function(i){
  n * mean(data2$Price) - (n-1) * mean(data2[-i,1])
})

J_T = (1/n) * sum(T_i_star)

Var_T_jackknife = constant * sum((T_i_star - J_T)^2)
Var_T_jackknife
table = data.frame(boostrap = var_boot, jackknife= Var_T_jackknife)
knitr::kable(table)
percent<-ci$percent
bca<-ci$bca
norm<-ci$normal
intervals = rbind(percent[4:5],bca[4:5],norm[2:3])

Len.CI=intervals[,2]-intervals[,1]
Esti.mean=intervals[,1]+Len.CI/2
table2 <- data.frame(Len.CI,Esti.mean)
rownames(table2) <- c("percent","bca","norm")
knitr::kable(table2)
```

Computer Lab 6 Computational Statistics

Contents

Question 1: Genetic algorithm

In this assignment, you will try to perform one-dimensional maximization with the help of a genetic algorithm.

1. Define the function

$$f(x) := \frac{x^2}{e^x} - 2\exp(\frac{-9\sin x}{x^2+x+1})$$

```
# define the function
func = function(x){
  return((x^2/exp(x)) - 2 * exp(-(9 * sin(x))/ (x^2 +x +1)))
}
```

2. Define the function `crossover()`

for two scalars x and y it returns their "kid as $(x+y)/2$.

```
# crossover function
crossover = function(x,y){
  kid = (x+y)/2
  return(kid)
}
```

3. Define the function `mutate()`

that for a scalar x returns the result of the integer division x2 mod 30. (Operation mod is denoted in R as %%).

```
# mutate function
mutate = function(x){
  return(x^2 %%30)
}
```

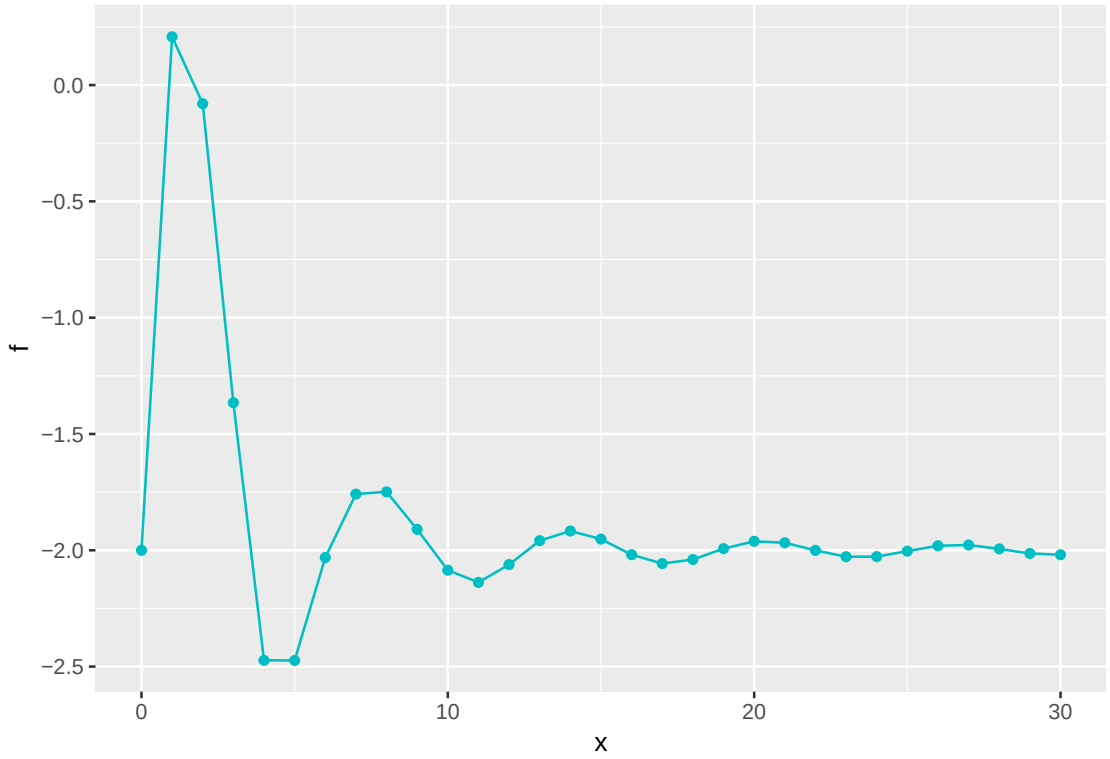
4. Write a function

that depends on the parameters `maxiter` and `mutprob` and:

- (a) Plots function f in the range from 0 to 30. Do you see any maximum value?
- (b) Defines an initial population for the genetic algorithm as $X = (0, 5, 10, 15, ..., 30)$.
- (c) Computes vector `Values` that contains the function values for each population point.
- (d) Performs `maxiter` iterations where at each iteration
  - i. Two indexes are randomly sampled from the current population, they are further used as parents (use *sample()*).
  - ii. One index with the smallest objective function is selected from the current population, the point is referred to as victim (use *order()*).
  - iii. Parents are used to produce a new kid by crossover. Mutate this kid with probability *mutprob* (use *crossover()*, *mutate()*).

- iv. The victim is replaced by the kid in the population and the vector *Values* is updated.
- v. The current maximal value of the objective function is saved.

- (e) Add the final observations to the current plot in another colour.



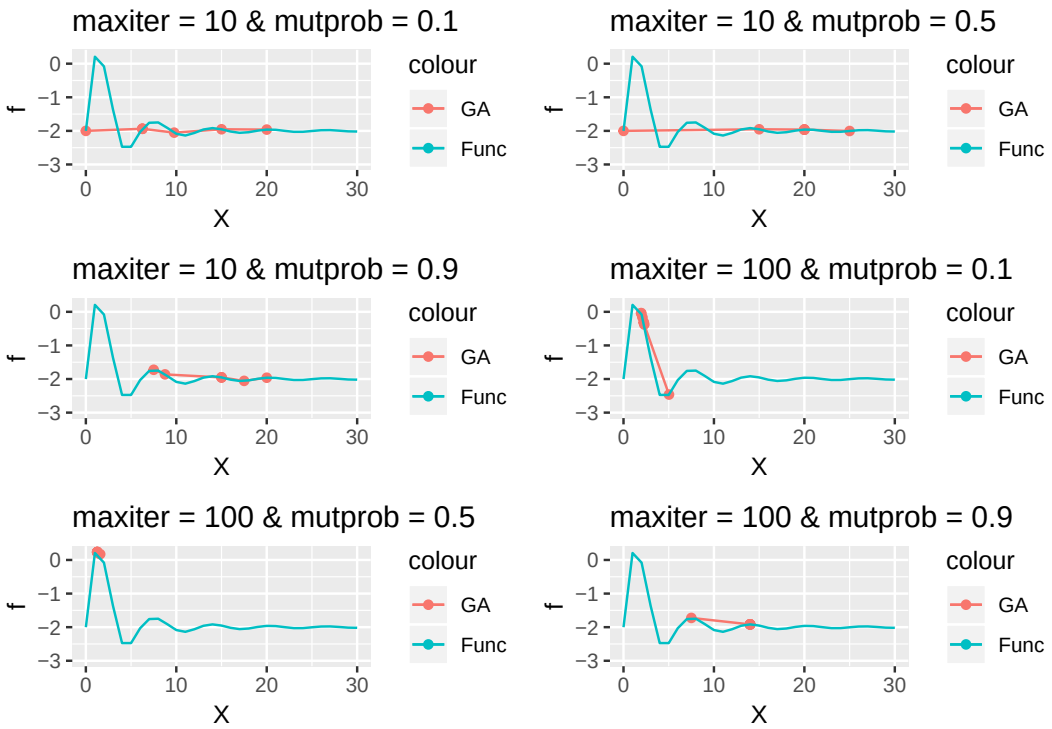
The maximum value might be between 0 and 2.

```
#1.4b#################################################################
# initial population
X <- seq(0,30,5)

#1.4c#################################################################
# the function values for each population points
Values <- func(X)

#1.4d#################################################################
set.seed(1234567)
func4 <- function(pars, animation = F){
  maxiter = pars$maxiter
  mutprob = pars$mutprob
  name = pars$name
  tX <- X
  for(i in 1:maxiter) {
    samples <- sample(tX, 2, replace = F)
    id <- which.min(func(tX))
    kid <- crossover(samples[1],samples[2])
    if(runif(1)>mutprob){
      kid <- mutate(kid)
    }
    tX[id] <- kid
    tX <- sort(tX)
```

```
    if(animation){    # Here we can see the change step by step
      plot(tX,func(tX),type = "b",xlim=c(0,30), ylim = c(-3,0.25),col="Blue")
      lines(x=seq(0,30),y=f(seq(0,30)))
      Sys.sleep(0.2)
    }
  }

#1.4e#############################################################
  dt <- data.frame(X=tX,f=func(tX))
  pl = ggplot(dt,aes(x=X,y=f,color="Blue"))+
    geom_point()+
    geom_line()+
    geom_line(data=dataa,aes(x=x,y=f,color="Red"))+
    ylim(-3,0.25)+
    xlim(0,30)+
    ggtitle(name)+
    scale_color_discrete(labels=c("GA","Func"))
  return(pl)
}
```

5. Run your code with different combinations

of **maxiter**= 10, 100 and **mutprob**= 0.1, 0.5, 0.9. Observe the initial population and final population. Conclusions?



To begin with, the results with maxiter=100 have better average performance than the ones with maxiter=10. Most of them get the maximum around X=1, except for the one with mutprob=0.9.

Based on our code, mutprob represents threshold of mutation. If it is large, the childern would be hard to

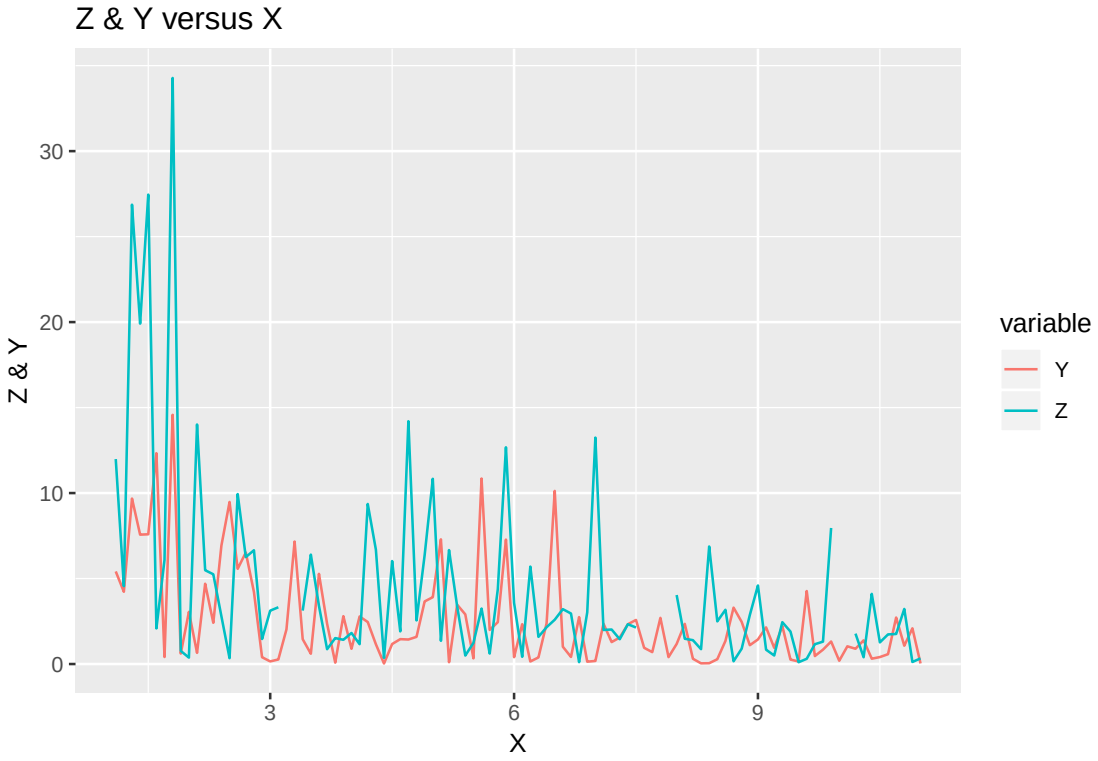
mutate. According to the plots, large mutprob might restrict the search range, which help us find a local optimal, not a global optimal.

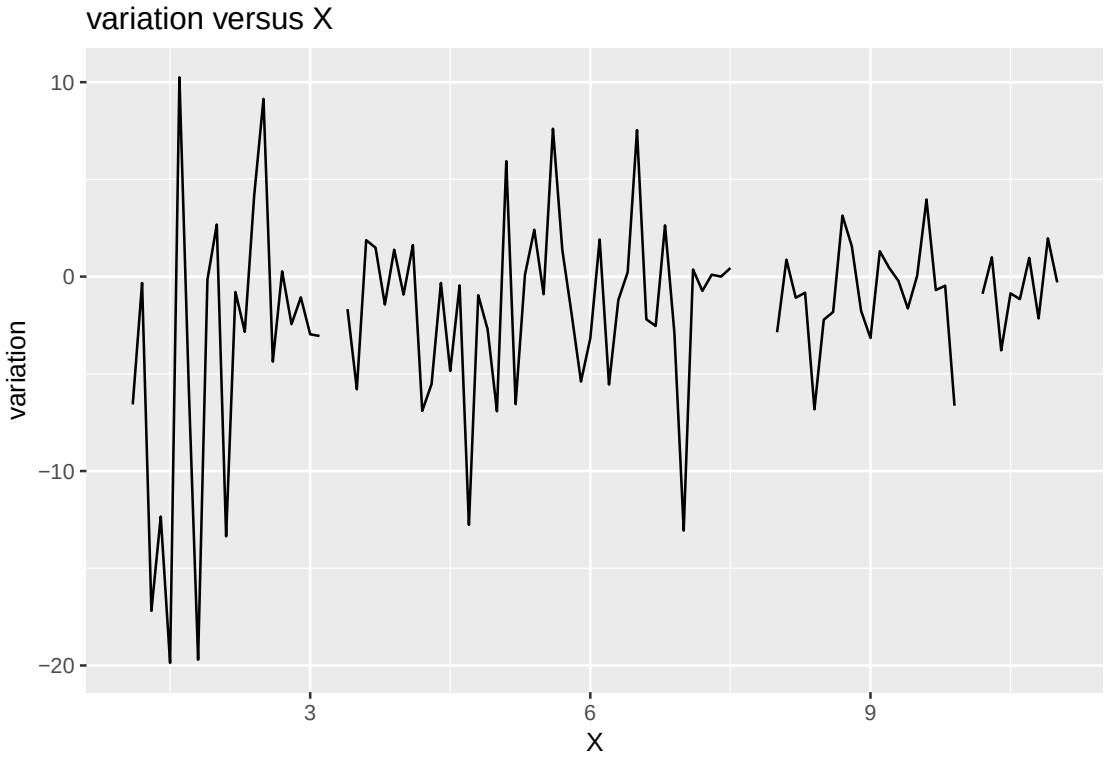
Question 2: EM algorithm

The data file *physical.csv* describes a behavior of two related physical processes $Y = Y(X)$ and $Z = Z(X)$.

1. Make a time series plot

describing dependence of Z and Y versus X. Does it seem that two processes are related to each other? What can you say about the variation of the response values with respect to X?





It does not seem that the two processes are related to each other. In the beginning, the Z value does sleep off more than double of Y. Also, the movements of the processes rarely lie on top of each other. Based on the variation of such two responses, we can recognize the Z values are incomplete in some parts, the curve has gaps. However, the variation seems to decrease when X increases.

2. Derive an EM algorithm that estimates λ

Note that there are some missing values of Z in the data which implies problems in estimating models by maximum likelihood. Use the following model

$$Y_i \sim \exp(X_i/\lambda), \quad Z_i \sim \exp(X_i/2\lambda)$$

where λ is some unknown parameter. *The goal is to derive an EM algorithm that estimates* λ.

EM algorithm is a algorithm based on maximum likelihood estimation. We have to find the a probability density function (PDF) $p(x, z, \theta)$, where x is a sequence of observed data , z is latent data and θ are unknown parameters. Afterwards, there are two steps. Firstly, we expose the latent data z by maximum log-liklihood and current data x. Secondly, we get the estimate of parameters θ based on the maximum log-liklihood.

Expectation

In our case, we know the PDFs of Y and Z are

$$f(Y_i) = \frac{X_i}{\lambda} e^{-\frac{X_i}{\lambda} Y_i}, f(Z_i) = \frac{X_i}{2\lambda} e^{-\frac{X_i}{2\lambda} Z_i},$$

where the missing values in Z can be considered as latent data and the others are observed data. Since we consider that all the data are independent and we can thereby get the liklihood, which is

$$\begin{aligned}\mathcal{L}(\lambda|Y,Z) &= \prod f(Y_i)\cdot\prod f(Z_i)\\ &= \prod(\frac{X_i}{\lambda}e^{-\frac{X_i}{\lambda}Y_i})\ \cdot\ \prod(\frac{X_i}{2\lambda}e^{-\frac{X_i}{2\lambda}Z_i})\\ &= \frac{\prod X_i}{\lambda^n}e^{-\frac{\sum X_iY_i}{\lambda}}\ \cdot\ \frac{\prod X_i}{2^n\lambda^n}e^{-\frac{\sum X_iZ_i}{2\lambda}}\ .\end{aligned}$$

Thus, the log-likelihood should be

$$\log\mathcal{L}(\lambda|Y,Z) = \prod(\ln X_i) - n\ln\lambda - \frac{\sum X_iY_i}{\lambda} + \prod(\ln X_i) - n\ln(2\lambda) - \frac{1}{2\lambda}\sum X_iZ_i.$$

In the expectation step, we use maximum log-liklihood estimate to expose/express the missing data Z_{miss} by the other data (Y, Z_{obs}) in the last iteration, and thereby estimate our target paramter λ. Additionally, we replace the miss data with the expected value of Z.

$$\begin{aligned}Q(\lambda,\lambda_k) =& E[\log\mathcal{L}(\lambda|Y,Z_{miss})|(\lambda_k,Y,Z_{obs})]\\ =& \prod(\ln X_i) - n\ln\lambda_k - \frac{\sum X_iY_i}{\lambda_k} + \prod(\ln X_i) - n\ln(2\lambda_k) - \frac{1}{2\lambda_k}[\sum_{obs}X_iZ_i + \sum_{miss}X_i\frac{2\lambda_{k-1}}{X_i}]\\ =& 2\prod(\ln X_i) - n\ln\lambda_k - \frac{\sum X_iY_i}{\lambda_k} - n\ln(2\lambda_k) - \frac{1}{2\lambda_k}\sum_{obs}X_iZ_i - \frac{1}{\lambda_k}\sum_{miss}\lambda_{k-1}.\end{aligned}$$

maximization

To obtain estimate λ, we need to solve that the partial erivative of $Q(\lambda,\lambda_k)$ equals 0.

$$\nabla\lambda = -\frac{2n}{\lambda_k} + \frac{\sum X_iY_i}{\lambda_k^2} + \frac{1}{2\lambda_k^2}\sum_{obs}X_iZ_i + \frac{1}{\lambda_k^2}\sum_{miss}\lambda_{k-1} = 0$$

$$\lambda_k = \frac{1}{4n}(2\sum X_iY_i + \sum_{obs}X_iZ_i + 2\sum_{miss}\lambda_{k-1})$$

3.Implement this algorithm in R

use $\lambda_0 = 100$ and convergence criterion "stop if the change in λ is less than 0.001". What is the optimal λ and how many iterations were required to compute it?

```
EM <- function(data2, eps, kmax=500, lda){
  X <- data2$X
  Y <- data2$Y
  Z <- data2$Z
  obs <- !is.na(Z)
  n <- length(X)
  m <- sum(is.na(Z))
```

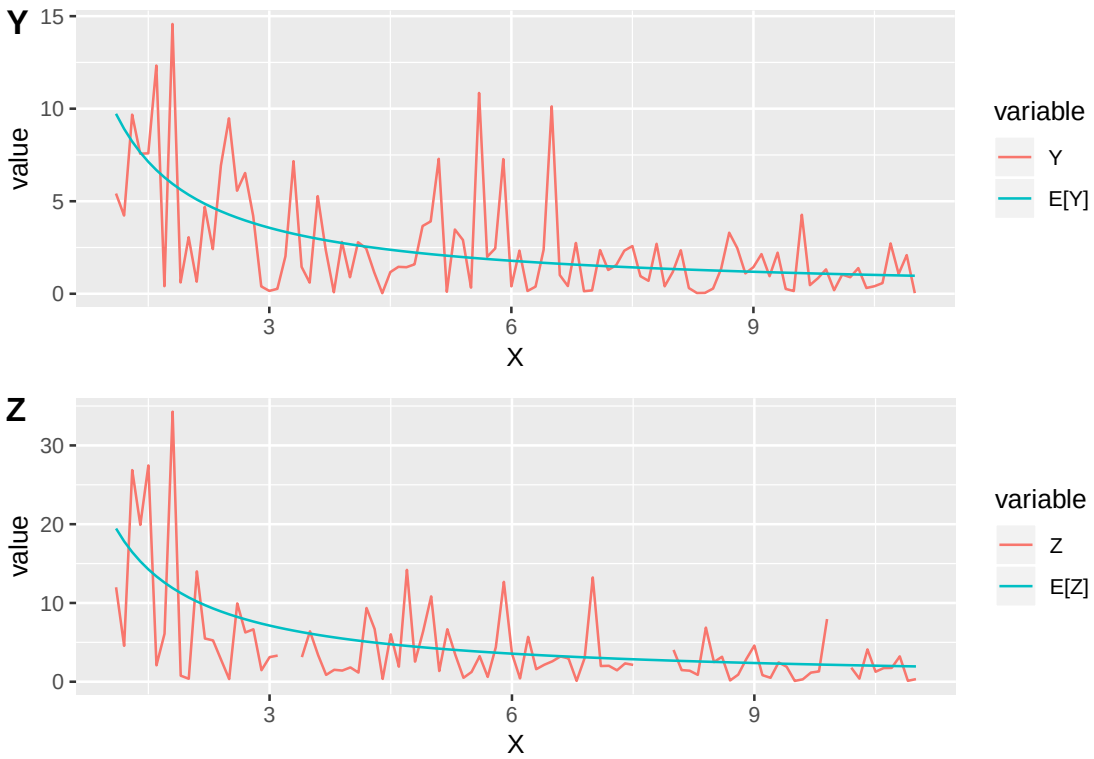
```
  # browser()
  loglik <- function(lda,ldap){
    return(2*sum(log(X))-n*log(lda)-(X%*%Y)/lda-n*log(2*lda)
           -(X[obs]%*%Z[obs])/(2*lda)-m*ldap/lda)
  }
  # initial state
  k <- 0
  llvalprev <- lda*10+10  # make some difference
  llvalcurr <- lda
  llk <- loglik(llvalcurr,llvalprev)

  print(c("iter","lda","llk"))
  print(c(k, llvalcurr,llk))

  while ((abs(llvalprev-llvalcurr)>eps) && (k<(kmax+1))){
    llvalprev <- llvalcurr
    # since we already know how to get argmax lda,
    # we can use the solution directly
    llvalcurr <- 1/(4*n)*(2*X%*%Y+X[obs]%*%Z[obs]+2*m*llvalprev)

    k<-k+1
    llk <- loglik(llvalcurr,llvalprev)
    print(c(k, llvalcurr,llk))
  }
  return(llvalcurr)
}

lda_opt <- EM(data2, eps=0.001, lda=100)
```

```
## [1] "iter" "lda"  "llk"
## [1]    0.0000  100.0000 -761.7647
## [1]    1.00000   14.26782 -470.99635
## [1]    2.00000   10.83853 -416.01655
## [1]    3.00000   10.70136 -413.46921
## [1]    4.00000   10.69587 -413.36664
## [1]    5.00000   10.69566 -413.36253
```

According to the code, it is obvious that the log-liklihood increases during iterations. We get the optimal $\lambda = 10.69566$ after 5 iterations.

4. Plot $E[Y]$ and $E[Z]$ versus X

in the same plot as Y and Z versus X. Comment whether the computed λ seems to be reasonable.



In accordance with the plot, the paths (blue) of bath E[Y] and E[Z] pass through the original data (red), which show similar tendencies. We can therefore conclude that the λ we get is reasonable.

Appendix

```
knitr::opts_chunk$set(echo = TRUE, out.height = "280px")
# library used in this lab
library(ggplot2) # ex 2.1 - time series plot
library(gridExtra)
# clean the environment
rm(list=ls())
# define the function
func = function(x){
  return((x^2/exp(x)) - 2 * exp(-(9 * sin(x))/ (x^2 +x +1)))
}
# crossover function
crossover = function(x,y){
  kid = (x+y)/2
  return(kid)
}
# mutate function
mutate = function(x){
  return(x^2 %%30)
}
#1.4a#################################################################
dataa <- data.frame(x=0:30,f=func(0:30))
plot1.4=ggplot(dataa,aes(x=x,y=f))+
  geom_line(color="#00BFC4")+
  geom_point(color="#00BFC4")    # max x=1
plot1.4
#1.4b#################################################################
# initial population
X <- seq(0,30,5)

#1.4c#################################################################
# the function values for each population points
Values <- func(X)

#1.4d#################################################################
set.seed(1234567)
func4 <- function(pars, animation = F){
  maxiter = pars$maxiter
  mutprob = pars$mutprob
  name = pars$name
  tX <- X
  for(i in 1:maxiter) {
    samples <- sample(tX, 2, replace = F)
    id <- which.min(func(tX))
    kid <- crossover(samples[1],samples[2])
    if(runif(1)>mutprob){
      kid <- mutate(kid)
    }
    tX[id] <- kid
    tX <- sort(tX)
    if(animation){    # Here we can see the change step by step
      plot(tX,func(tX),type = "b",xlim=c(0,30), ylim = c(-3,0.25),col="Blue")
```

```
      lines(x=seq(0,30),y=f(seq(0,30)))
      Sys.sleep(0.2)
    }
  }

#1.4e#############################################################
  dt <- data.frame(X=tX,f=func(tX))
  pl = ggplot(dt,aes(x=X,y=f,color="Blue"))+
    geom_point()+
    geom_line()+
    geom_line(data=dataa,aes(x=x,y=f,color="Red"))+
    ylim(-3,0.25)+
    xlim(0,30)+
    ggtitle(name)+
    scale_color_discrete(labels=c("GA","Func"))
  return(pl)
}
maxiter = c(10,100)
mutprob = c(0.1,0.5,0.9)
names = c("maxiter = 10 & mutprob = 0.1",
          "maxiter = 100 & mutprob = 0.5",
          "maxiter = 10 & mutprob = 0.9",
          "maxiter = 100 & mutprob = 0.1",
          "maxiter = 10 & mutprob = 0.5",
          "maxiter = 100 & mutprob = 0.9")
pairs = data.frame(maxiter=rep(maxiter,3),mutprob=rep(mutprob,2),name=names)
pairs = split(pairs,pairs[,3])

plot(arrangeGrob(grobs=lapply(t(pairs), func4)))
# clean the environment
rm(list=ls())
# load the data
data2 = read.csv("physical1.csv")
# Z & Y versus X

data21 <- reshape2::melt(data2, id.va = "X")
ggplot(data = data21,aes(x=X,y=value,color=variable)) +
  geom_line()+
  # geom_line(aes(x = X, y=Y),color = "#F8766D") +
  # geom_line(aes(x = X, y=Z),color = "#00BFC4") +
  ggtitle("Z & Y versus X") +
  ylab("Z & Y")

ggplot(data = data.frame(X=data2$X,variation=data2$Y-data2$Z), aes(x = X)) +
  geom_line(aes(y = variation)) +
  ggtitle("variation versus X")
EM <- function(data2, eps, kmax=500, lda){
  X <- data2$X
  Y <- data2$Y
  Z <- data2$Z
  obs <- !is.na(Z)
  n <- length(X)
  m <- sum(is.na(Z))
```

```
  # browser()
  loglik <- function(lda,ldap){
    return(2*sum(log(X))-n*log(lda)-(X%*%Y)/lda-n*log(2*lda)
           -(X[obs]%*%Z[obs])/(2*lda)-m*ldap/lda)
  }
  # initial state
  k <- 0
  llvalprev <- lda*10+10  # make some difference
  llvalcurr <- lda
  llk <- loglik(llvalcurr,llvalprev)

  print(c("iter","lda","llk"))
  print(c(k, llvalcurr,llk))

  while ((abs(llvalprev-llvalcurr)>eps) && (k<(kmax+1))){
    llvalprev <- llvalcurr
    # since we already know how to get argmax lda,
    # we can use the solution directly
    llvalcurr <- 1/(4*n)*(2*X%*%Y+X[obs]%*%Z[obs]+2*m*llvalprev)

    k<-k+1
    llk <- loglik(llvalcurr,llvalprev)
    print(c(k, llvalcurr,llk))
  }
  return(llvalcurr)
}

lda_opt <- EM(data2, eps=0.001, lda=100)
X <- data2$X
EY <- c(lda_opt)/X
EZ <- 2*EY

data241 <- cbind(data2[,-3],data.frame(EY=EY))
data241 <- reshape2::melt(data241, id.va="X")
p1 <- ggplot(data241, aes(x=X, y=value, color=variable))+
  geom_line()+
  scale_color_discrete(labels=c("Y","E[Y]"))

data242 <- cbind(data2[,-2],data.frame(EY=EZ))
data242 <- reshape2::melt(data242, id.va="X")
p2 <- ggplot(data242, aes(x=X, y=value, color=variable))+
  geom_line()+
  scale_color_discrete(labels=c("Z","E[Z]"))

cowplot::plot_grid(p1, p2, labels = c('Y', 'Z'),ncol=1)
```