Lab 1 - Computational Statistics (732A90)

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

January 31, 2018

Contents

1 Assignment 1

1.1 Task 1

1.1.1 Snippet 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"
```

1.1.2 Snippet 2

```
x1 = 1; x2 = 1 / 2

if (x1 - x2 == 1/2) {
  print("Subtraction is correct")
  } else {
  print("Subtraction is wrong")
}
```

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

1.1.3 Answer

The first comparison `1/3 - 1/4 == 1/12` issues FALSE whereas the second comparison `1 - 1/2 == 1/2` issues TRUE.

If we compare the rational numbers to the 22nd decimal, `1/3 - 1/4` and `1/12` differ:

```
1/3 - 1/4
```

```
## [1] 0.0833333333333333314829616
```

```
1/12
```

```
## [1] 0.0833333333333333287074
```

... whereas `1 - 1/2` and `1/2` are the same (trailing 0's are not printed).

```
1 - 1/2
```

```
## [1] 0.5
```

```
1/2
```

```
## [1] 0.5
```

In conclusion, `1/3 - 1/4` and `1/12` differ because of the representation of rational numbers on 64 bits in the form $\overset{+}{-}0.d_1d_2...d_p \cdot b^e$. In other words, the representation of `1/3` minus the representation of `1/4` does not equal the representation of `1/12`.

The **main issue** here is that for the case with `1/3 - 1/4`, we have a number with infinite decimals whereas for the case with `1 - 1/2`, we just have one decimal.

1.2 Task 2

Since we would actually want the `1/3 - 1/4 == 1/12` to be `TRUE` in an R script, we need to use a different method to check this condition:

```
x1 = 1 / 3; x2 = 1 / 4
isTRUE(all.equal(x1 - x2, 1/12))
```

```
## [1] TRUE
```

2 Assignment 2

2.1 Task 1

Below, you can find the implemented function of the derivative of $f(x) = x$

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

where $\epsilon = 10^{-15}$.

```r
derivative = function(x){
  epsilon = 10^(-15)
  num = (x + epsilon) - x
  den = epsilon
  return(num/den)
}
```

2.2 Task 2

Tthe function returns the following values for `x = 1` and `x = 100000` respectively:

```
## derivative(1):  1.110223024625156540424
```

```
## derivative(100000):  0
```

2.3 Task 3

- The true derivative of $f(x) = x$, $f(x)' = 1$. Since the derivative is a constant, it should be equal to 1 regardless of the value of x at which it is evaluated.
- Problem: Unfortunately, the function returns different values (see task 2).

Explanation:

Let us investigate the **numerators in the derivative function** for both x values. For the correct output, we would expect the value $10^{-15} = 1.00000000000000077705e-15$ as numerator for both cases (x = 1 and x = 100000). The numerator for `x = 100000`, however, evaluates to `0` and the numerator for `x = 1` evaluates to `1.110223024625156540424e-15`:

```
## Numerator for x = 100000:  0
```

```
## Numerator for x = 1:  0.000000000000001110223024625156540424
```

Let us drill down and only preset the **first part of the numerator, i.e. x + e**:

```
## x + e for x = 100000:  100000
```

```
## x + e for x = 1:  1.000000000000001110223
```

We can see that when adding the very small number $e = 10^{-15}$ to $x = 100000$, the result stays 100000. We have an **underflow problem** since decimals are lost. When adding the same small number $e = 10^{-15}$ to $x = 1$, the underflow does not occur because x is a relatively small number and decimals can be saved as well up to some point.

The **main reason for the underflow problem with** $x = 100000$ is that the there is not enough space available for R to save the number 100000 together with the decimals. The number 1 requires less space, therefore the decimals can be kept.

The **main reason for the wrong solution with** $x = 1$ is that we are are not using a difference ϵ that is small enough. Actually, the derivative is defined for the limit when $\epsilon \to 0$. However, the problem is that we cannot make ϵ much smaller. If we did, then we would encounter an underflow problem with $x = 1$ as well and get a result of 0 (like for `x = 100000` in the example above).

3 Assignment 3

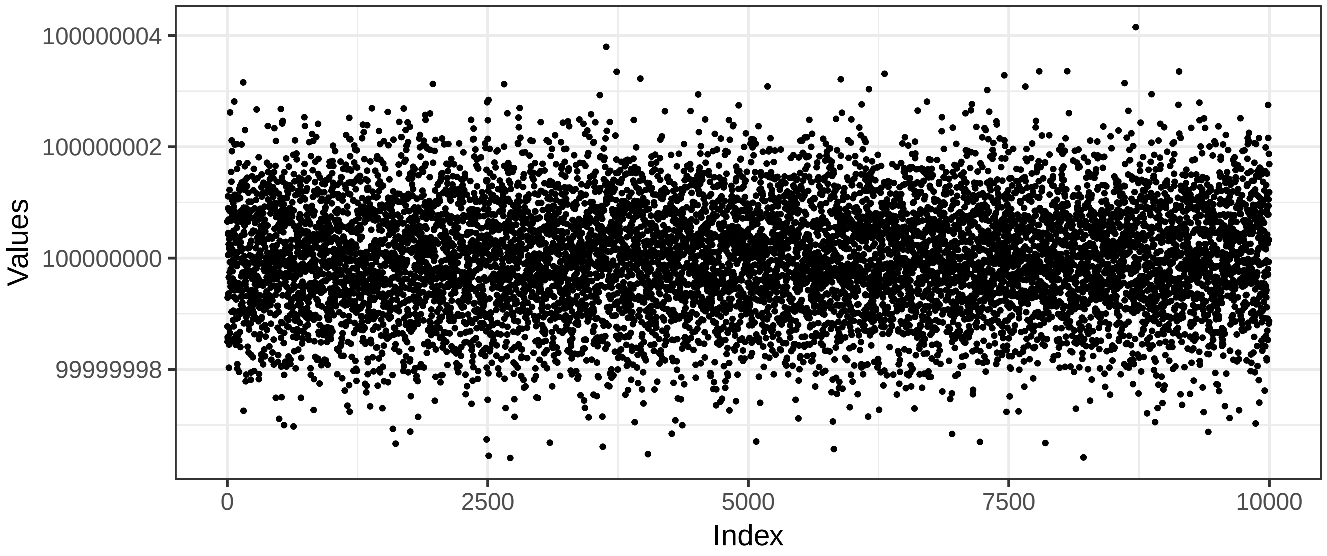
3.1 Task 1

Find below the function for computing the variance for a vector x.

```
myvar = function(x){
  n = length(x)
  res = 1/(n - 1) * (sum(x^2) - 1/n * sum(x)^2)
  return(res)
}
```

3.2 Task 2

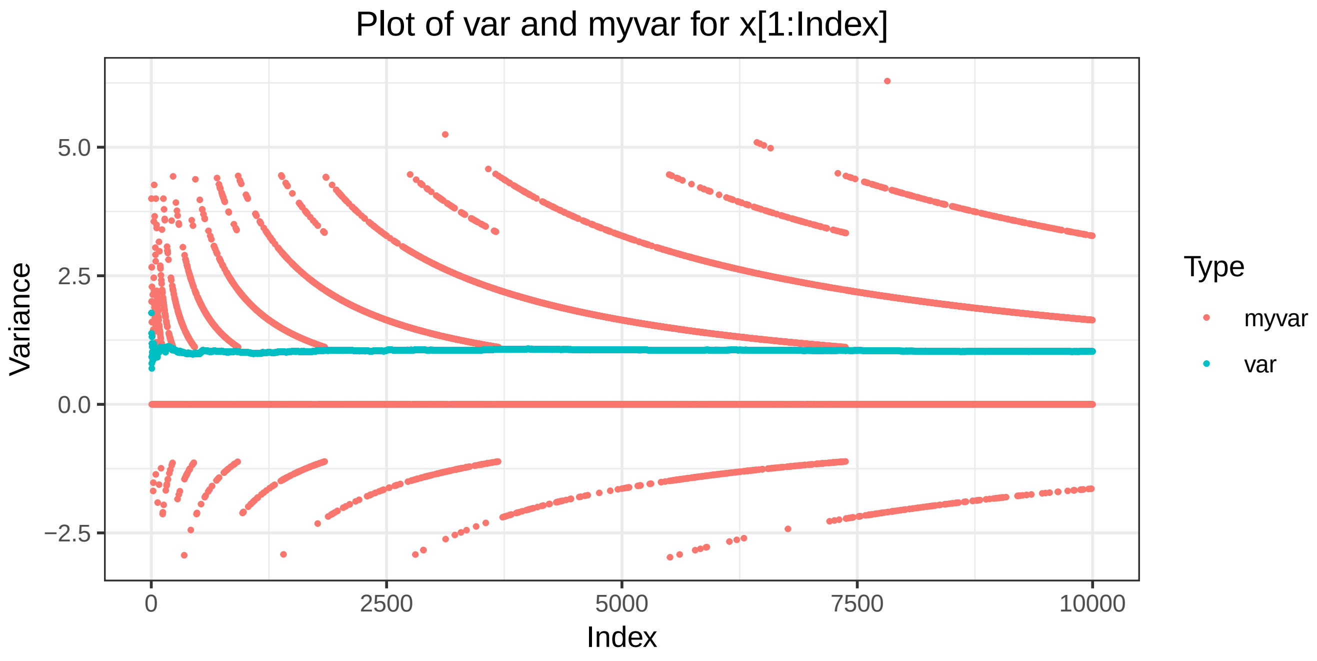
A vector $x = (x_1, ..., x_10000)$ with 10000 random numbers with mean $10^8) and variance 1 is generated. The randomly generated numbers are plotted by index below.



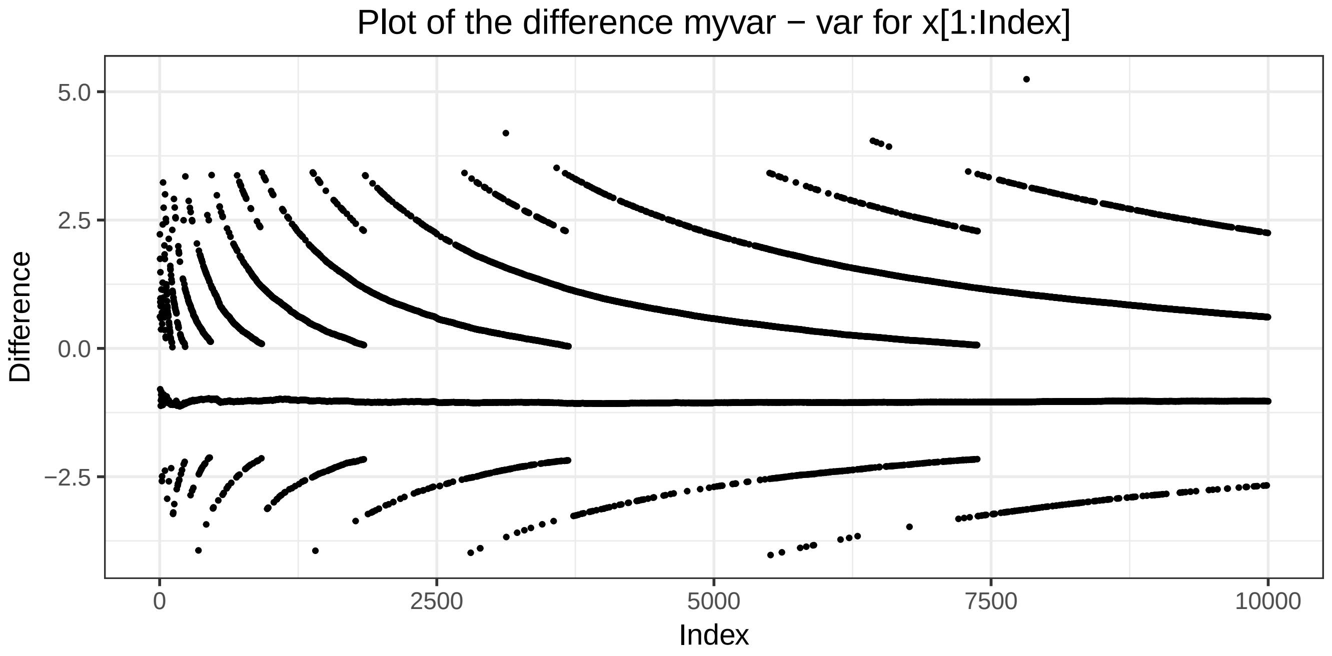
3.3 Task 3

Note that the true variance is 1. Therefore, we expect a variance of 1 to be calculated with the `var` and the `myvar` functions, especially when a larger amount of sampled values is used for calculating the variance.

First, the results for `myvar` and `var` are plotted separately, using x[1:Index] as input for Index = 1, ..., 10000. One can see that the `var` does indeed return values of 1, especially for larger `Index` values. The results of `myvar` seem to fluctuate strongly however. Weirdly, at some positions the variance returns negative values, which should not be the case since the variance is by definition computed using the squares of the difference between individual values and their mean.



Second, the difference `myvar - var` for x[1:Index] as input is plotted below. The shape of the results for myvar and var (see previous plot) is also reflected in this difference.



Conclusions from the plot:

The variance returned by the `myvar` function strongly differs from the variance returned by the `var` function. It seems like for the `myvar` function, additional values in x can have a huge impact since the `myvar` values are jumping up and down from negative to positive values reversely from index to index.

Performance of the function:

The `myvar` function does not perform well since it does not return the correct, expected result (i.e. values of 1) like the `var` function does.

Explanation:

Basically, we need to explain a) why we get negative values with the `myvar` function b) why the variance jumps up and down.

First of all, we have to note that all values in the vector x are very large (due to the mean of 10ˆ8). However, these values also have numerous decimals (due to the variance of 1). This situation raises the suspicion that we have an underflow problem. Now, when does this problem occur?

We need to consider two terms in the equation, the first sum and the second sum. Looking at the elements in the first sum: when square a very large number e.g. 99999999.36501321196556, then we have an underflow problem `99999999.36501321196556^2 = 9999999873002642`, i.e. the decimals are discarded. Now, looking at the second sum, when summing up two very large values, we do not have an underflow problem e.g. `99999999.36501321196556 + 100000000.76248191297054 = 200000000.1274951100349`. But after summing up all the values in x and squaring the result, we definitely have an overflow problem as well (similar to the one when squaring the individual numers in the first sum). Now, the result depends on which decimals get cut off due to the underflow problem: If larger decimals are cut off in the first sum than in the second sum, the result will be negative. If it is the other way around, the result will be positive. A single additional element in the vector x can change this balance because it will affect both terms w.r.t. its decimals.

Therefore, regarding b), the value of the variance can jump up and down when new values are added. Regarding a), it can be negative when relatively larger decimals are cut off due to overflow in the first sum than in the second sum.

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

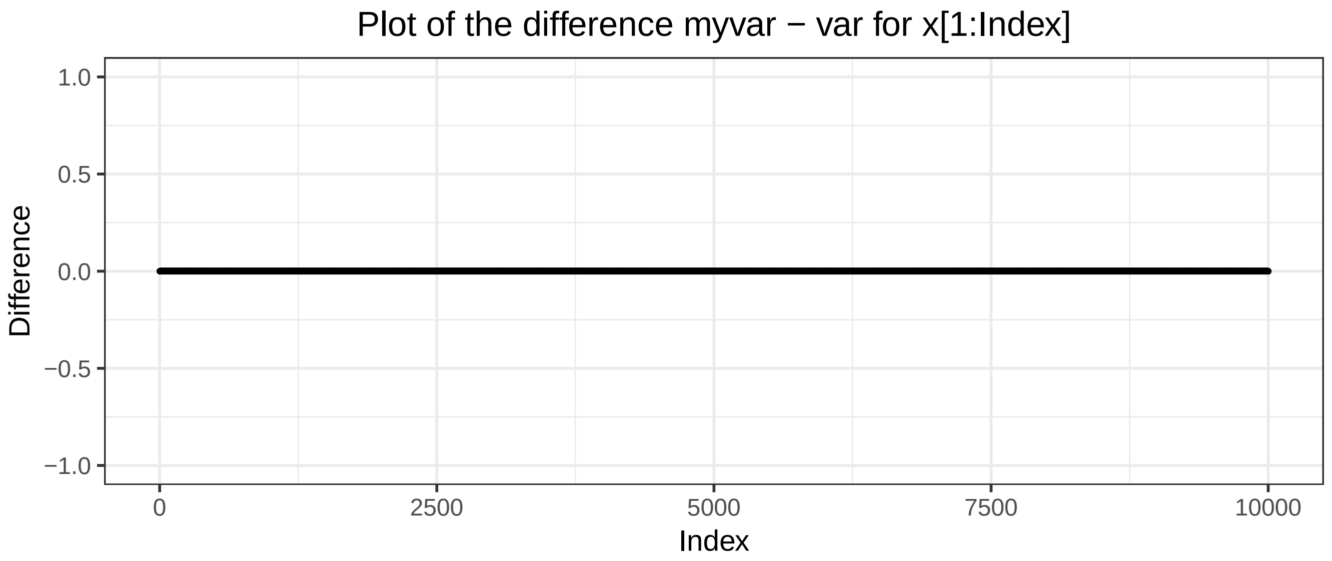
3.4 Task 4

One could compute the variance using the following *sample variance* formula.

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2$$

If we implement the sample variance formula as follows, we obtain values of approximately 0 for the difference `myvar - var` for all indices (plotted below).

```r
myvar = function(x){
  n = length(x)
  mean(x)
  res = 1/(n - 1) * sum((x - mean(x))^2)
  return(res)
}
```



4 Assignment 4

4.1 Task 1

First, the data from `tecator.csv` is imported into R.

4.2 Task 2

Second, we compute $A = X^T X$ and $b = X^T y$.

Note: The instructions did not specify that an intercept should be used. Therefore, we did not include any intercept here.

```
X = as.matrix(df[, -c(1, ncol(df)-1)])
y = as.matrix(df$Protein)

A = t(X) %*% X
b = as.numeric(t(X) %*% y)
```

4.3 Task 3

Third, we try to solve $A\beta = b$ using the `solve` function. However, the following error is returned: $Errorinsolve.default(A, b) : systemiscomputationallysingular : reciprocalconditionnumber = 7.13971e - 17$. In other words, we have a singular linear system (it does not row-reduce to the identity matrix) but we require a non-singular linear system in order to compute the inverse of A which wee need to solve for beta:

$$\begin{aligned} A\beta &= b \\ A^{-1}A\beta &= A^{-1}b \\ I\beta &= A^{-1}b \\ \beta &= A^{-1}b \end{aligned}$$

```
# Note: this code is not executed here because it's error prevents knitting.
res = solve(A, b)
```

Explanation:

A matrix is singular when at least one of its rows can be represented as a linear combination of any of the other rows. Since the rows and columns of A, a 102x102 matrix, represent the feature variables, we can conclude that R seems to find linear dependencies between the variables.

4.4 Task 4

Here, we check the condition number $k(A) = ||A||\,||A^{-1}||$ where $||\cdot||$ is the norm. As we know from the lecture, a large k(A) value is a bad sign.

```
## kappa(A): 1157834236871692.25
```

Explanation:

A large condition number k(A) is a bad sign because it implies strong deviation from a non-singular matrix. Therefore, the result is related to the conclusion from task 3, namely that the error occured because the matrix A is singular.

The following example illustrates that a large k(A) corresponds to strong deviation from a non-singular matrix.

```
## nonsingular:
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
## kappa(nonsingular): 1
```

```
## singular:
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    0
```

```
## kappa(singular): Inf
```

4.5 Task 5

Now, we scale the features in the data set (i.e. all columns in X) and repeat steps 2-4.

How did the results change:

The results change noticeably:

- We can solve the linear system with `solve(A, b)` (instead of getting an error).
- We get condition number that is less than 0.1% of the size of the previous condition number.

```
## kappa(A): 490471520662.049987793
```

```
## kappa(A_scaled)/kappa(A_unscaled): 0.000400000000000000191687
```

Explanation:

Basically, we need to explain why scaling enables the resolve function to work fine. Since the `resolve` function does not give a "singularity error" anymore, it seems to recognize the matrix A as a nonsingular matrix after scaling. Weirdly, if you calculate the determinant before and after scaling, the result is still 0, which would, theoretically, imply that the matrix is still singular after scaling. However, it is possible that the `resolve` function somehow has a different process which allows to recognize the matrix as nonsingular.

In general, one other aspect that might be affected by scaling is the density of the values. Consider the following diagram from the lecture slides. If we have very dense values that only differ in their very last decimals, R might consider different values (and vectors) to be the same, when e.g. somehow not considering the very last decimals. Scaling would help to spread out the values.

The following example illustrates this point: the vector `c(0.000000012, 0.000000011, 0.000000010)` is scaled to `c(1.000000000000000008881784, 0.000000000000000000000,`

-0.9999999999999992228439) After scaling, the values are clearly spread out and noticeably different.

```
x = c(0.000000012, 0.000000011, 0.000000010)
x_scaled = as.numeric(scale(x)); x_scaled
```

```
## [1]  1.0000000000000008881784  0.0000000000000000000000
## [3] -0.9999999999999992228439
```

5 Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE,
                      cache = TRUE, cache.path = "cache/", fig.path = "cache/")

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

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


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


x1 = 1 / 3; x2 = 1 / 4

if (x1 - x2 == 1/12){
  print("Subtraction is correct")
  }else {
  print("Subtraction is wrong")
}


x1 = 1; x2 = 1 / 2

if (x1 - x2 == 1/2) {
  print("Subtraction is correct")
  } else {
  print("Subtraction is wrong")
}
```

```r
1/3 - 1/4
1/12
1 - 1/2
1/2

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


x1 = 1 / 3; x2 = 1 / 4
isTRUE(all.equal(x1 - x2, 1/12))


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


derivative = function(x){
  epsilon = 10^(-15)
  num = (x + epsilon) - x
  den = epsilon
  return(num/den)
}


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

cat("derivative(1): ", derivative(1), "\n")
cat("derivative(100000): ", derivative(100000), "\n")


# --------------------------------------------------------------------------------
# Assignment 2, Task 3
# --------------------------------------------------------------------------------

e = 10^(-15)
x = 100000
cat("Numerator for x = 100000: ", x + e - x, "\n")
x = 1
cat("Numerator for x = 1: ", x + e - x, "\n")
```

```r
e = 10^(-15)
x = 100000
cat("x + e for x = 100000: ", x + e, "\n")
x = 1
cat("x + e for x = 1: ", x + e, "\n")

# -----------------------------------------------------------------------------
# Assignment 3, Task 1
# -----------------------------------------------------------------------------

myvar = function(x){
  n = length(x)
  res = 1/(n - 1) * (sum(x^2) - 1/n * sum(x)^2)
  return(res)
}


# -----------------------------------------------------------------------------
# Assignment 3, Task 2
# -----------------------------------------------------------------------------

x = rnorm(10000, mean = 10^8, sd  = 1)

df_plot = data.frame(Index = 1:10000, Values = x)
ggplot(df_plot, aes(x = Index, y = Values)) +
  geom_point(size = 0.5) + theme_bw()


# -----------------------------------------------------------------------------
# Assignment 3, Task 3
# -----------------------------------------------------------------------------

res_myvar = unlist(lapply(as.list(1:10000), function(idx) myvar(x[1:idx])))
res_var = unlist(lapply(as.list(1:10000), function(idx) var(x[1:idx])))

df_plot = data.frame(Index = 1:10000, myvar = res_myvar, var = res_var)
df_plot %<>% tidyr::gather(key = Type, value = Variance, -Index)

ggplot(df_plot, aes(x = Index, y = Variance, color = Type)) +
  geom_point(size = 0.5) + theme_bw() +
  labs(title = "Plot of var and myvar for x[1:Index]") +
  theme(plot.title = element_text(hjust = 0.5))


df_plot = data.frame(Index = 1:10000, Difference = res_myvar - res_var)
```

```r
ggplot(df_plot, aes(x = Index, y = Difference)) +
  geom_point(size = 0.5) + theme_bw() +
  labs(title = "Plot of the difference myvar - var for x[1:Index]") +
  theme(plot.title = element_text(hjust = 0.5))


myvar = function(x){
  n = length(x)
  res = 1/(n - 1) * (sum(x^2) - 1/n * sum(x)^2)
  return(res)
}

# -----------------------------------------------------------------------------
# Assignment 3, Task 4
# -----------------------------------------------------------------------------

myvar = function(x){
  n = length(x)
  mean(x)
  res = 1/(n - 1) * sum((x - mean(x))^2)
  return(res)
}


res_myvar = unlist(lapply(as.list(1:10000), function(idx) myvar(x[1:idx])))
res_var = unlist(lapply(as.list(1:10000), function(idx) var(x[1:idx])))

df_plot = data.frame(Index = 1:10000, Difference = res_myvar - res_var)

ggplot(df_plot, aes(x = Index, y = Difference)) +
  geom_point(size = 0.5) + theme_bw() +
  labs(title = "Plot of the difference myvar - var for x[1:Index]") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(limits = c(-1, 1))



# -----------------------------------------------------------------------------
# Assignment 4, Task 1
# -----------------------------------------------------------------------------

df = read.csv("tecator.csv")



# -----------------------------------------------------------------------------
# Assignment 4, Task 2
# -----------------------------------------------------------------------------
```

```
X = as.matrix(df[, -c(1, ncol(df)-1)])
y = as.matrix(df$Protein)

A = t(X) %*% X
b = as.numeric(t(X) %*% y)


# -----------------------------------------------------------------------------
# Assignment 4, Task 3
# -----------------------------------------------------------------------------


# Note: this code is not executed here because it's error prevents knitting.
res = solve(A, b)


# -----------------------------------------------------------------------------
# Assignment 4, Task 4
# -----------------------------------------------------------------------------

K = kappa(A)
cat("kappa(A):", K, "\n")


cat("nonsingular: \n")
nonsingular = diag(rep(1, 3)); nonsingular
cat("kappa(nonsingular):", kappa(nonsingular), "\n\n")

cat("singular: \n")
singular = nonsingular; singular[3, 3] = 0; singular
cat("kappa(singular):", kappa(singular), "\n")


# -----------------------------------------------------------------------------
# Assignment 4, Task 5
# -----------------------------------------------------------------------------

# Task 2: Computing A and b
X = scale(df[, -c(1, ncol(df)-1)])
y = as.matrix(df$Protein)

A = t(X) %*% X
b = as.numeric(t(X) %*% y)

# Task 3: Solving linear system
```

```r
res = solve(A, b)

# Task 4: Computing condition number
K_prev = K
K = round(kappa(A), 2)
cat("kappa(A):", K, "\n")
cat("kappa(A_scaled)/kappa(A_unscaled):", round(K/K_prev, 4), "\n")


knitr::include_graphics("images/scaling.png")


x = c(0.000000012, 0.000000011, 0.000000010)
x_scaled = as.numeric(scale(x)); x_scaled
```