Computational Statistics - Lab 01

Maximilian Pfundstein

21 januari 2019

Contents

1 Question 1: Be Careful When Comparing

```
################################################################################
# Question 1 - Be Careful When Comparing
################################################################################

x1 = 1/3
x2 = 1/4

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

```
## [1] "Substraction is wrong."
```

```
x1 = 1
x2 = 1/2

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

```
## [1] "Substraction is correct."
```

Questions:

1. Check the results of the snippets. Comment what is going on.
2. If there are any problems, suggest improvements.

Answers:

1. The first substraction is not wrong - it is perfectly working as defined in IEEE_754 which is a commonly used definition for floating point numbers. To make a long strory short: You have an infinite amount of

real numbers for instance between 0.0 and 1.0 but just 32 or 64 bits for the represetation (so 2^{32} states or 2^{64} states) so it's impossible to represent every number (t.ex. try writing down 1/3 in the decimal system, at one point you simply must stop). The second substraction does not have a floating point error as you can represent multiples of the power of two in the binary system $2^{-1} = 0.5$.

2. You cannot get rid of the "problem", just use more bits for representing the numbers until you have the desired precision or use/write a class which can handle a specific amount of numbers behind the decimal point (which will be slower for sure).

2 Question 2: Derivative

Question: Write your own R function to calculate the derivative of `f(x) = x` in this way with $\epsilon = 10^{-15}$.

```
##################################################################################
# Question 2: Derivative
##################################################################################

epsilon = 10^(-15)

f_prime = function(x) {
  return( (f(x + epsilon) - f(x)) / epsilon)
}

f = function(x) {
  return(x)
}
```

Question: Evaluate your derivative function at `x = 1` and `x = 100000`.

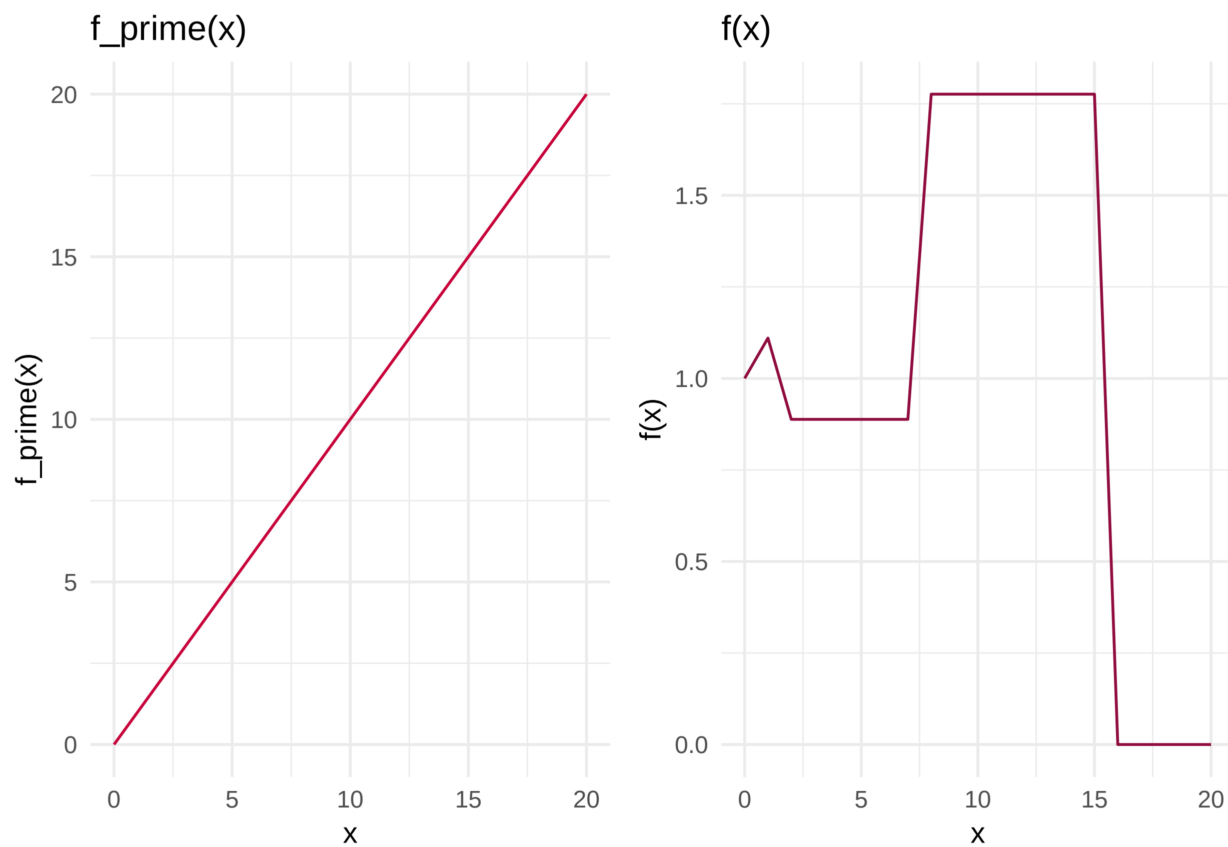
```
first = f_prime(1)
second = f_prime(100000)

print(first)
```

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

```
print(second)
```

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

The following plots show the function `f` and `f_prime` in the interval `0...20`. We observe that the derivative seems to take discrete values and is `0` around `x = 16`.



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

Answer: The values and plots can be seen above.

The derivate of `f(x) = x` is `f'(x) = 1` so the true slope is `1` at all spots.

As `epsilon` is a really small number and we do calculations with a rather big number (`x`) we run into precision problems which are more heavy if the magnitudes of the numbers is greatly different. Therefore we see that if we take `x = 1` the error is smaller, but still big enough to give an undesired result. As we take `x = 100000` the difference in magnitude increased further so we obtain the weird result `0` which s obviously totally wrong.

If you evaluate only the nominator you will see that it's `0` after `x = 16` so the result will always be `0` (assuming the denominator is unequal to `0`).

3 Question 3: Variance

Question: Write your own R function, `myvar`, to estimate the variance in this way.

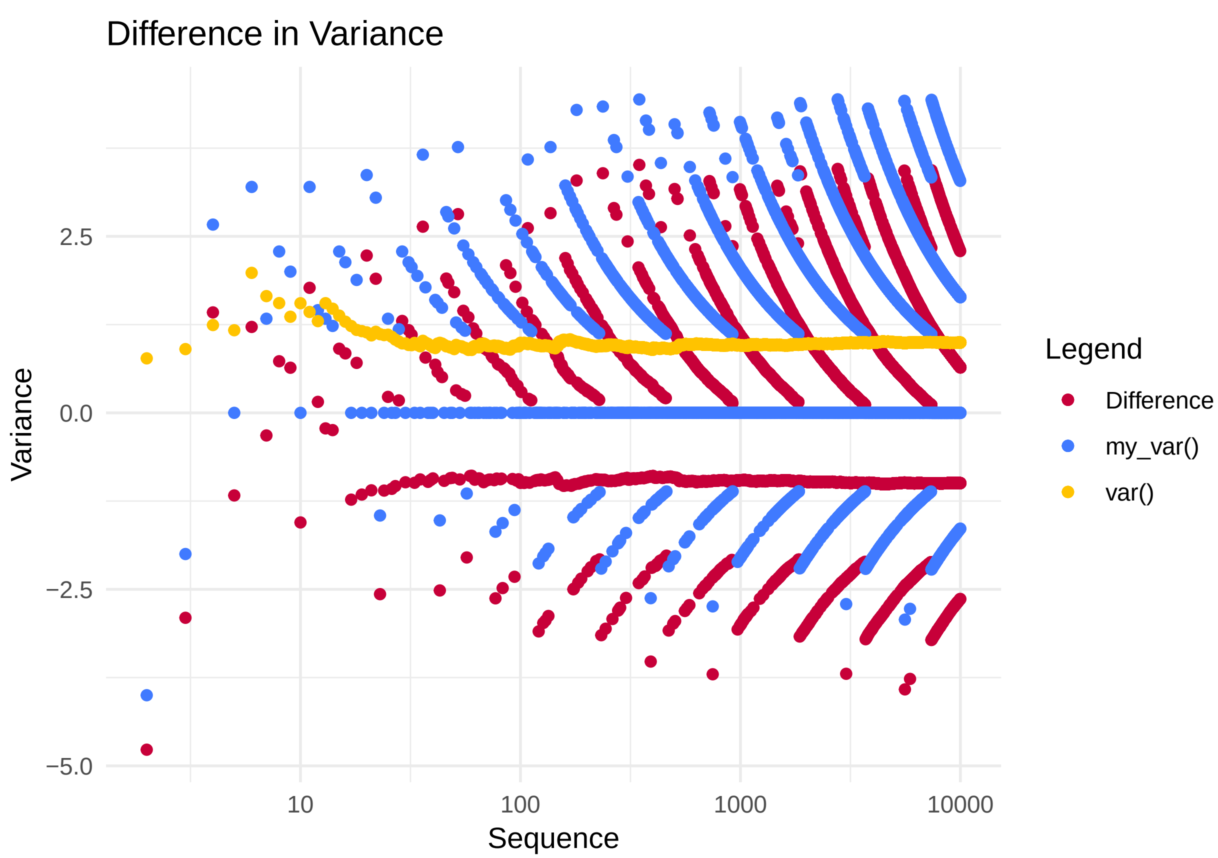
```
################################################################################
# Question 3: Variance
################################################################################

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

Question: Generate a vector $x = (x_1, ..., x_{10000})$ with 10000 random numbers with mean 10^8 and variance 1.

```r
v = rnorm(10000, mean = 10^8, sd = 1)
```

Question: For each subset $X_i = \{x1, ..., x_i\}, i = 1, ..., 10000$ compute the difference $Y_i = myvar(X_i) - var(X_i)$, 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?

```r
X = data.frame()

for (i in 1:length(v)) {
  Xi = v[1:i]
  vec_myvar = myvar(as.vector(Xi))
  vec_var = var(Xi)
  Yi = vec_myvar - vec_var
  Yi_index = list(index = i, value = Yi, vec_myvar = vec_myvar, vec_var = vec_var)
  X = rbind(X, Yi_index)
}

ggplot(X[2:nrow(X),]) +
  geom_point(aes(x = index, y = value, colour = "Difference")) +
  geom_point(aes(x = index, y = vec_myvar,  colour = "my_var()")) +
  geom_point(aes(x = index, y = vec_var, colour = "var()")) +
  labs(title = "Difference in Variance", y = "Variance",
  x = "Sequence", color = "Legend") +
  scale_color_manual(values = c("#C70039", "#407AFF", "#FFC300")) +
  scale_x_log10() +
  theme_minimal()
```



Answer: When interpretating the plot one has to keep in mind that each subset contains one value more than the previous one, so we should observe that we're getting better estimations for the variance which an increased index. Let's focus on the first values (`index < 10`). They're highly scattered which is to be expected for such a small number of data points. With increasing `index` we see that some pattern is repeated. We have datapoints around `-1` as well as trails towards the center. If we look at the trail that is on the upper half, ending at around an index of `100` we observe the following values:

```
kable(X[X$index > 70 & X$index < 100,])
```

	index	value	vec_myvar	vec_var
71	71	0.8825033	1.828571	0.9460681
72	72	-0.9456352	0.000000	0.9456352
73	73	-0.9447091	0.000000	0.9447091
74	74	0.8064492	1.753425	0.9469754
75	75	0.7819890	1.729730	0.9477407
76	76	-0.9541195	0.000000	0.9541195
77	77	-2.6274791	-1.684211	0.9432685
78	78	-0.9310509	0.000000	0.9310509
79	79	0.6914509	1.641026	0.9495747
80	80	0.6801090	1.620253	0.9401441
81	81	-0.9406388	0.000000	0.9406388
82	82	-0.9314892	0.000000	0.9314892
83	83	-2.4822989	-1.560976	0.9213233
84	84	0.6318797	1.542169	0.9102890
85	85	0.6159062	1.523810	0.9079034
86	86	2.0908453	3.011765	0.9209194
87	87	0.5781176	1.488372	0.9102545
88	88	0.5680727	1.471264	0.9031917
89	89	0.5572984	1.454546	0.8972470
90	90	1.9801758	2.876405	0.8962287
91	91	0.4936162	1.422222	0.9286060
92	92	-0.9369944	0.000000	0.9369944
93	93	0.4397444	1.391304	0.9515599
94	94	-2.3219996	-1.376344	0.9456555
95	95	1.7863717	2.723404	0.9370325
96	96	-0.9593970	0.000000	0.9593970
97	97	0.3835445	1.333333	0.9497888
98	98	-0.9410972	0.000000	0.9410972
99	99	-0.9605839	0.000000	0.9605839

As we see the trails are **not** continuous as they heavily fluctuade. So, as it seems like they do not converge and are not continuous, this function (`my_var()`) seems not to be a good estimator for the variance. As we have rather big values with just a small variance from the created data and we have the sum of quite a few data, we might run into an underflow and precision problems.

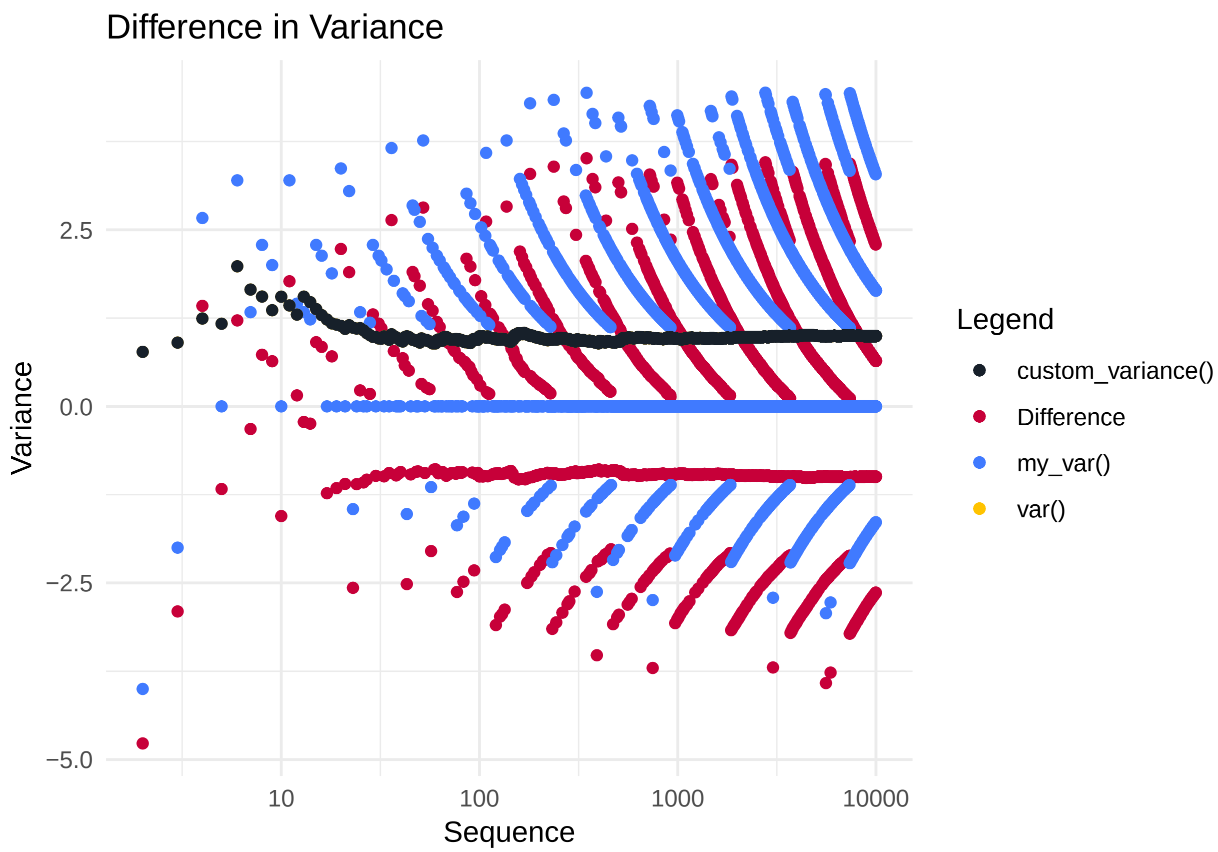
Question: How can you better implement a variance estimator? Find and implement a formula that will give the same results as `var()`?

Answer: We use this formula:

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

This is the sample standard deviance and thus is biased (if we assume our $\vec{v}$ as a sample from a population).

```
custom_variance = function(x) {
  diff_mean = x - mean(x)
  return(sum(diff_mean^2 / (length(x) - 1)))
}
```



It can be seen that this function does a way better job at converging and returns almost (or exactly) the same values as the built in function and this covering the `var()` plot almost perfectly.

4 Question 4: Linear Algebra

Question: Import the data set to R.

```
################################################################################
# Question 4: Linear Algebra
################################################################################

data = read.csv2("tecator.csv", sep=",", dec=".")
kable(head(data[, c(1, 101, 102, 103, 104)]))
```

Sample	Channel100	Fat	Protein	Moisture
1	2.81920	22.5	16.7	60.5
2	3.17942	40.1	13.5	46.0
3	2.54816	8.4	20.5	71.0
4	2.79622	5.9	20.7	72.8
5	3.13753	25.5	15.5	58.3

Sample	Channel100	Fat	Protein	Moisture
6	3.45307	42.7	13.7	44.0

Question: Optimal regression coeffcients can be found by solving a system of the type $A\vec{\beta} = \vec{b}$ where $A = X^T X$ and $\vec{b} = X^T \vec{y}$. Compute A and $\vec{b}$ 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.

```
X = as.matrix(data[, c(1:102, 104)])
Y = as.matrix(data[, c(103)])
A = t(X) %*% X
b = t(X) %*% Y
```

Question: 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?

```
tryCatch(
    expr = {
        beta = solve(A) %*% b
    },
    error = function(e){
        paste("That escalated rather quickly: ", e)
    }
)
```

```
## [1] "That escalated rather quickly:  Error in solve.default(A): system is computationally singular: r
```

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

```
kappa(A)
```

```
## [1] 4.286972e+15
```

As we mostly work with *rational* numbers we are used to the fact that almost every number has a inverse. An inverse a^{-1} is defined as that element that, multiplied with a results in the *neutral element*, at least for multiplications in arithmetics. In arithmetics we only have one number that does not have an multiplicative inverse which is `0` as `1/0` is undefined.

With matirces there are way more such matrices that do not have an inverse, exactly then when:

- The matrix is not a square.
- The determinant and thus the span is `0`.

If we imagine a matrix as a linear transformation the determinant is the factor by which the space is streched or compressed. Thus a determinant of `0` tells us if the given linear transformation is squishing the amount of dimensions.

So it's reasonable that we have matrices that do not have an inverse. If we wanted to use the inverse for calculating or solving an equation, we have to find a different way (t.ex. QR-decomposition).

The `kappa()` function computes an estimate of the condition number of a matrix. Given a linear equation $Ax = b$ the number gives us an estimation of how inaccurate the approximation of x is going to be. One can also say that it says how much x is going to change in respect to b. So if we have a large condition number it means that a small error in b is likely to cause a large error in x. As our number here is rather large, we can conclude that our features are linearly dependant.

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

```r
data_scaled = scale(data)

X = as.matrix(data_scaled[, c(1:102, 104)])
Y = as.matrix(data_scaled[, c(103)])
A = t(X) %*% X
b = t(X) %*% Y

beta = solve(A) %*% b
```

The result has changed as we scaled the data. Before we ran into computational issues as the scale for each feature was on a different scope/scale which can lead to those errors. The `sclae()` made them of equal size and thus solved the problem.

Last but not least, let's look at the new conditional number, which should be smaller:

```r
kappa(A)
```

```
## [1] 664322330621
```

It is smaller as we'd have expected. So everyone is happy and we can conclude this lab!

```
                                      _.--"""--,
                                    .'          `\
  ._"""""""_.                     .'              |
 /          '.                   /            .-._/
|             `.                |             |
 \              \          .-._ |          _   \
  `""'-.         \_.-.     \   `          ( \__/
        |             )     '=.       .,   \
       /             (         \     /  \  /
     /`               `\        |   /    `'
     '..-`\        _.-. `\ _.__/   .=.
          |  _    / \  '.-`    `-.'  /
          \_/ |  |   './ _     _  \.'
               '-'    | /       \ |
                      |  .-. .-.  |   H A V E   N I C E   D A Y !
                      \ / o| |o \ /
                       |   / \   |           M O O !
                      / `"`   `"` \
                     /             \
                    | '._.'         \
                    |  /             |
                     \ |             |
                      ||    _    _   /
                      /|\  (_\  /_) /
                      \ \'._  ` '_.'
                       `""` `"""`
```

5 Source Code

```r
knitr::opts_chunk$set(echo = TRUE, cache = TRUE, include = TRUE, eval = TRUE)
library(ggplot2)
library(knitr)
library(gridExtra)
```

```r
################################################################################
# Question 1 - Be Careful When Comparing
################################################################################

x1 = 1/3
x2 = 1/4

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


x1 = 1
x2 = 1/2

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


################################################################################
# Question 2: Derivative
################################################################################

epsilon = 10^(-15)

f_prime = function(x) {
  return( (f(x + epsilon) - f(x)) / epsilon)
}

f = function(x) {
  return(x)
}


first = f_prime(1)
second = f_prime(100000)

print(first)
print(second)


sequence = seq(from = 0, to = 20, by = 1)
func = f(sequence)
deri = f_prime(sequence)
df = data.frame(sequence, deri)

p1 = ggplot(df) +
  geom_line(aes(x = sequence, y = func), color = "#C70039") +
```

```r
  labs(title = "f_prime(x)", y = "f_prime(x)", x = "x") +
  theme_minimal()

p2 = ggplot(df) +
  geom_line(aes(x = sequence, y = deri), color = "#900C3F") +
  labs(title = "f(x)", y = "f(x)", x = "x") +
  theme_minimal()

grid.arrange(p1, p2, nrow = 1)


################################################################################
# Question 3: Variance
################################################################################

myvar = function(x) return(1/(length(x)-1)  * (sum(x^2) - (sum(x)^2)/length(x)))


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


X = data.frame()

for (i in 1:length(v)) {
  Xi = v[1:i]
  vec_myvar = myvar(as.vector(Xi))
  vec_var = var(Xi)
  Yi = vec_myvar - vec_var
  Yi_index = list(index = i, value = Yi, vec_myvar = vec_myvar, vec_var = vec_var)
  X = rbind(X, Yi_index)
}

ggplot(X[2:nrow(X),]) +
  geom_point(aes(x = index, y = value, colour = "Difference")) +
  geom_point(aes(x = index, y = vec_myvar,  colour = "my_var()")) +
  geom_point(aes(x = index, y = vec_var, colour = "var()")) +
  labs(title = "Difference in Variance", y = "Variance",
  x = "Sequence", color = "Legend") +
  scale_color_manual(values = c("#C70039", "#407AFF", "#FFC300")) +
  scale_x_log10() +
  theme_minimal()


kable(X[X$index > 70 & X$index < 100,])


custom_variance = function(x) {
  diff_mean = x - mean(x)
  return(sum(diff_mean^2 / (length(x) - 1)))
}


for (i in 1:length(v)) {
```

```
  Xi = v[1:i]
  X$vec_customvar[[i]] = custom_variance(as.vector(Xi))
}

ggplot(X[2:nrow(X),]) +
  geom_point(aes(x = index, y = value, colour = "Difference")) +
  geom_point(aes(x = index, y = vec_myvar,  colour = "my_var()")) +
  geom_point(aes(x = index, y = vec_var, colour = "var()")) +
  geom_point(aes(x = index, y = vec_customvar, colour = "custom_variance()")) +
  labs(title = "Difference in Variance", y = "Variance",
  x = "Sequence", color = "Legend") +
  scale_color_manual(values = c("#17202A", "#C70039", "#407AFF", "#FFC300")) +
  scale_x_log10() +
  theme_minimal()



################################################################################
# Question 4: Linear Algebra
################################################################################

data = read.csv2("tecator.csv", sep=",", dec=".")
kable(head(data[, c(1, 101, 102, 103, 104)]))


X = as.matrix(data[, c(1:102, 104)])
Y = as.matrix(data[, c(103)])
A = t(X) %*% X
b = t(X) %*% Y


tryCatch(
    expr = {
        beta = solve(A) %*% b
    },
    error = function(e){
        paste("That escalated rather quickly: ", e)
    }
)


kappa(A)


data_scaled = scale(data)

X = as.matrix(data_scaled[, c(1:102, 104)])
Y = as.matrix(data_scaled[, c(103)])
A = t(X) %*% X
b = t(X) %*% Y

beta = solve(A) %*% b
```

```r
kappa(A)
```

```
                                      _.--"""__,
                                    .'          `\
  ._""""""_.                      .'              |
 /          '.                   /            .-._/
|             `.                |             |
 \              \          .-._ |          _   \
  `""'_.         \_.-.     \   `          ( \__/
        |             )     '=.       .,   \
       /             (         \     /  \  /
     /`               `\        |   /    `'
     '..-`\        _.-. `\ _.__/   .=.
          |  _    / \  '.-`    `-.'  /
          \_/ |  |   './ _     _  \.'
               '_'    | /       \ |
                      |  .-. .-.  |   H A V E   N I C E   D A Y !
                      \ / o| |o \ /
                       |   / \   |           M O O !
                      / `"`   `"` \
                     /             \
                    | '._.'         \
                    |  /             |
                     \ |             |
                      ||    _    _   /
                      /|\  (_\  /_) /
                      \ \'._  ` '_.'
                       `""` `"""`
```