Lab 6 - Computational Statistics (732A90)

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

March 8, 2019

Contents

1 Assignment 1: Genetic Algorithm

In this assignment, a one-dimensional maximization is performed with a genetic algorithm.

1.1 Task 1: Definition of f(x)

We are given the following function `f(x)`, which is implemented below.

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

```
f = function(x){

  res = (x^2 / exp(x)) - 2 * exp(-(9 * sin(x)) / (x^2 + x + 1))
  return(res)

}
```

1.2 Task 2: Definition of crossover()

Below, the function `crossover()` is implemented. I takes two scalars `x` and `y` and returns their `kid` as `(x + y)/2`.

```
crossover = function(x, y){

  res = (x + y) / 2
  return(res)

}
```

1.3 Task 3: Definition of mutate()

Below, the function `mutate()` is implemented. I takes a scalar `x` and returns the result of the integer division $x^2 \bmod 30$.

```
mutate = function(x){

  res = x^2 %% 30
  return(res)

}
```

1.4 Task 4: Genetic algorithm

Here a function is implemented that depends on the parameters maxiter and mutprob and:

(a) Plots function f in the range from 0 to 30.
(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 pop- ulation, the point is referred to as victim (use order()).
    iii. Parents are used to produce a new kid by crossover. Mutate this kid with prob- ability 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.

```
genetic = function(maxiter, mutprob){

  # (a) ------------------------------------------------------------------------
```

```r
df_plot = data.frame(x = seq(0, 30, 0.1), fx = f(seq(0, 30, 0.1)))
p = ggplot(df_plot) + theme_bw() +
  geom_line(aes(x = x, y = fx, color = "f(x) in general"),
            size = 0.5, alpha = 0.8, show.legend = FALSE) +
  geom_vline(xintercept = 1.2, color = "red", linetype = "dashed") +
  labs(subtitle = paste0("maxiter: ", maxiter, " | mutprob: ", mutprob)) +
  theme(plot.subtitle = element_text(hjust = 0.5))

# (b) -------------------------------------------------------------------

X = seq(0, 30, 5)

# (c) -------------------------------------------------------------------

Values = f(X)

# (d) -------------------------------------------------------------------

max_Values = numeric(maxiter)
max_X = numeric(maxiter)

for (i in 1:maxiter){

  parent_idx = sample(1:length(X), 2) # sample parent indices
  victim_idx = order(Values)[1] # sample victim index
  kid = crossover(X[parent_idx[1]], X[parent_idx[2]]) # create kid

  if (mutprob > runif(n = 1, min = 0, max = 1)){
    kid = mutate(kid) # mutate kid with probability mutprob
  }

  X[victim_idx] = kid # replace the victim by the kid
  Values[victim_idx] = f(kid) # update the Values
  max_Values[i] = max(Values) # save the current max value
  max_X[i] = X[which.max(Values)]

}

# (e) -------------------------------------------------------------------

df_final = data.frame(x = X, fx = Values)
df_initial = data.frame(x = seq(0, 30, 5), fx = f(seq(0, 30, 5)))
df_maxval = data.frame(x = max_X[maxiter], fx = max_Values[maxiter])

p = p + geom_point(data = df_final,
              aes(x = x, y = fx,color = "Final Observations"), size = 3,
```

```
                shape = 21, show.legend = FALSE) +
          geom_point(data = df_initial,
                aes(x = x, y = fx,color = "Initial Observations"), size = 3,
                shape = 21, show.legend = FALSE) +
          geom_point(data = df_maxval,
                aes(x = x, y = fx,color = "Max. Value"), size = 3, alpha = 0.7,
                shape = 19, show.legend = FALSE) +
          scale_color_manual(values = c("steelblue4", "red",
                                        "forestgreen", "red"))

  # Prepare output
  return(list(X = X, Values = Values, max_Values = max_Values, p = p))

}
```

Do we see any maximum value for (a)?

It does seem like there is a maximum at ca. `x = 1.2` which was marked with a vertical, dashed red line in the plots below.

1.5 Task 5: Algorithm execution and conclusions

Here, the code is run with different combinations of `maxiter = 10, 100` and `mutprob = 0.1, 0.5, 0.9`.

Conclusions

The initial population is plotted in green and the final population is plotted in red (and its max. value is highlighted with red background). Recall that we have a maximization problem. Hence, the best result would be to have all red circles at the `x` value indicated by the vertical, dashed, red line, which represents the maximum of f(x) for the given range of x values.

maxiter = 10

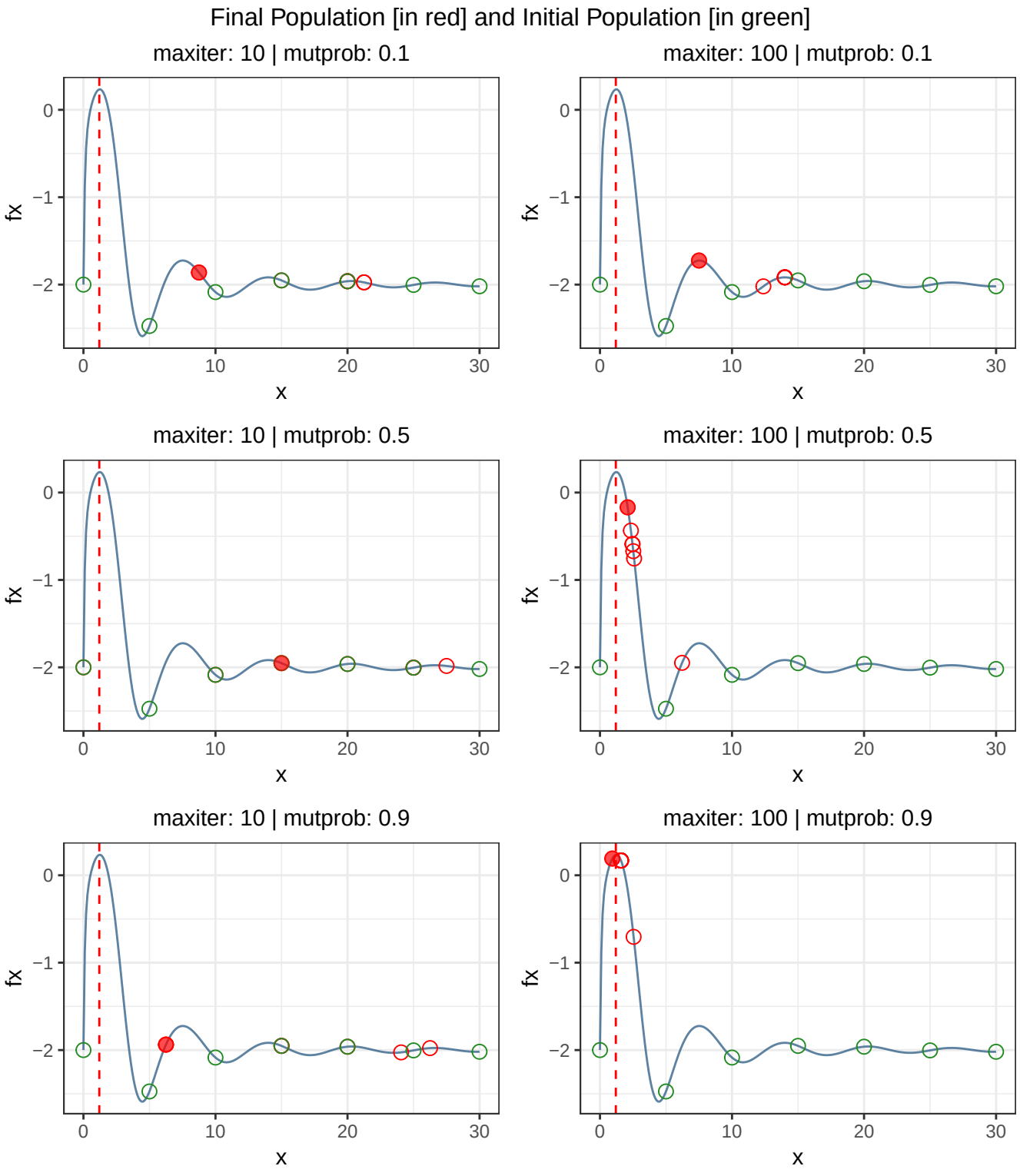
With `maxiter = 10`, the final population is not much better off than the initial population since the final population does not have any observations located at noticeably higher values of `f(x)`, with exception of maybe the plot for `mutprob = 0.1` where one observation has a slightly larger `f(x)` value (likely by chance).

We conclude that `maxiter = 10` does not seem to be large enough to optimize the result.

maxiter = 100

With `maxiter = 100`, we get noticeably better results than with `maxiter = 10`, for all values of `mutprob`. Still, we observe that we get better results for larger values of `mutprob`. With `mutprob = 0.1`, none of the final observations is very close to the maximum, with `mutprob = 0.5`, all except one of the final observations are very close to the maximum and with `mutprob = 0.9`, all final observations are very close to the maximum and some seem to be almost at the maximum.

We conclude that larger values of `maxiter` *and* `mutprob` *improve the optimization result. In other words, more iterations and more mutations lead to* `x` *values corresponding to larger* `f(x)` *values.*



2 Assignment 2: EM Algorithm

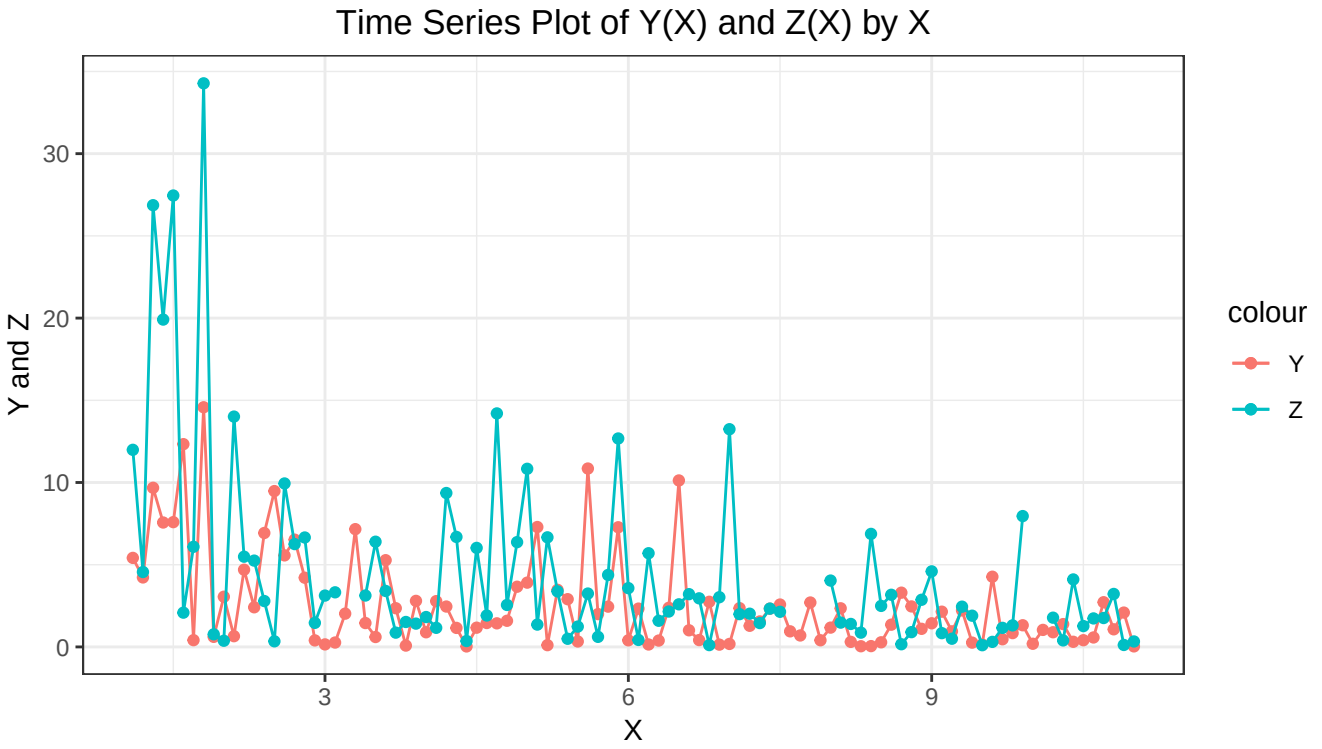
2.1 Task 1: Time series plot

Relationship between the two processes

It does seem like there is a relationship between the two processes since both of them seem to get fewer large values with increasing value of X. However, `Z(X)` can take larger values than `Y(X)` across most displayed values of X. In other words, the largest spikes of `Z(X)` are larger than the largest spikes of `Y(X)`, comparing values in e.g. certain intervals like $[0, 1.5]$, $[1.5, 3]$,..., $[9, 10.5]$.

Variation of the response values with respect to X

The response values do vary noticeably across all values of X. While all response values have a lower bound of 0, upwards spikes are very visible in the plot. The variation seems larger for smaller values of X (as described above).



2.2 Task 2 and 3: EM (Method 1)

Note: Two versions of the EM were derived and implemented. Both lead to the *exact same result*. The difference between the two versions is as follows:

- In method 1, the missing Z_i values are updated in every E-step. In method 2, no computations are done in the E-step (i.e. the E-step is just 'on paper').
- In method 1, λ_{MLE} is computed by taking the derivative of loglik and all Z_i are considered as complete (we replaced the missing Z_i values before). In method 2, λ_{MLE} is computed by taking the derivative of E[loglik] and we treat the missing Z_i and the observed Z_i separately.

2.2.1 Task 2: Derivation of EM algorithm

We are given the following models where Y, Z and X are the variables from our dataset.

$$Y_i \sim \text{exponential}\left(\text{rate} = \frac{X_i}{\lambda}\right)$$
$$Z_i \sim \text{exponential}\left(\text{rate} = \frac{X_i}{2\lambda}\right)$$

Further, we are given the hint that there are missing values in Z. This is indeed the case (see the printout below). However, there are no missing values for X and Y. The EM algorithm is generally used to find "the maximum-likelihood estimate of the parameters of an underlying distribution from a given data set when the data is incomplete or has missing values" (http://www-prima.imag.fr/jlc/Courses/2002/ENSI2.RNRF/EM-tutorial.pdf).

Table 1: 8 missing values of Z

	X	Y	Z
22	3.2	2.0246952	NA
23	3.3	7.1632522	NA
66	7.6	0.9480226	NA
67	7.7	0.6965180	NA
68	7.8	2.7002114	NA
69	7.9	0.4036646	NA
90	10.0	0.1922749	NA
91	10.1	1.0357660	NA

The EM-algorithm consists of two steps:

1. **expectation step (also 'estimation' step)**: In the expectation step, we update the estimates of the missing values (in other cases of latent variable values) given the current parameter θ and the data.
2. **maximization step**: In the maximization step, we update the paramter θ given the previously updated estimates of the missing values and the data. For this, we usually use maximum likelihood estimation.

In the following, we replace θ with λ since this is the unknown parameter that we want to estimate.

E-step

We are given that $Z_i \sim \exp(X_i/(2\lambda))$ which means that Z_i is esponentially distributed with rate $X_i/(2\lambda)$. We want to conduct the following update in every E-step for all Z_i that are `NA` values (corresponding to the 8 observations printed out above). The remaining Z_i do not need to be updated because we already have the true values and hence, estimating them would only add noise.

$$Z_i = E[Z_i|X_i, \lambda] = \frac{1}{\text{rate}} = \frac{1}{X_i/(2\lambda)} = \frac{2\lambda}{X_i}$$

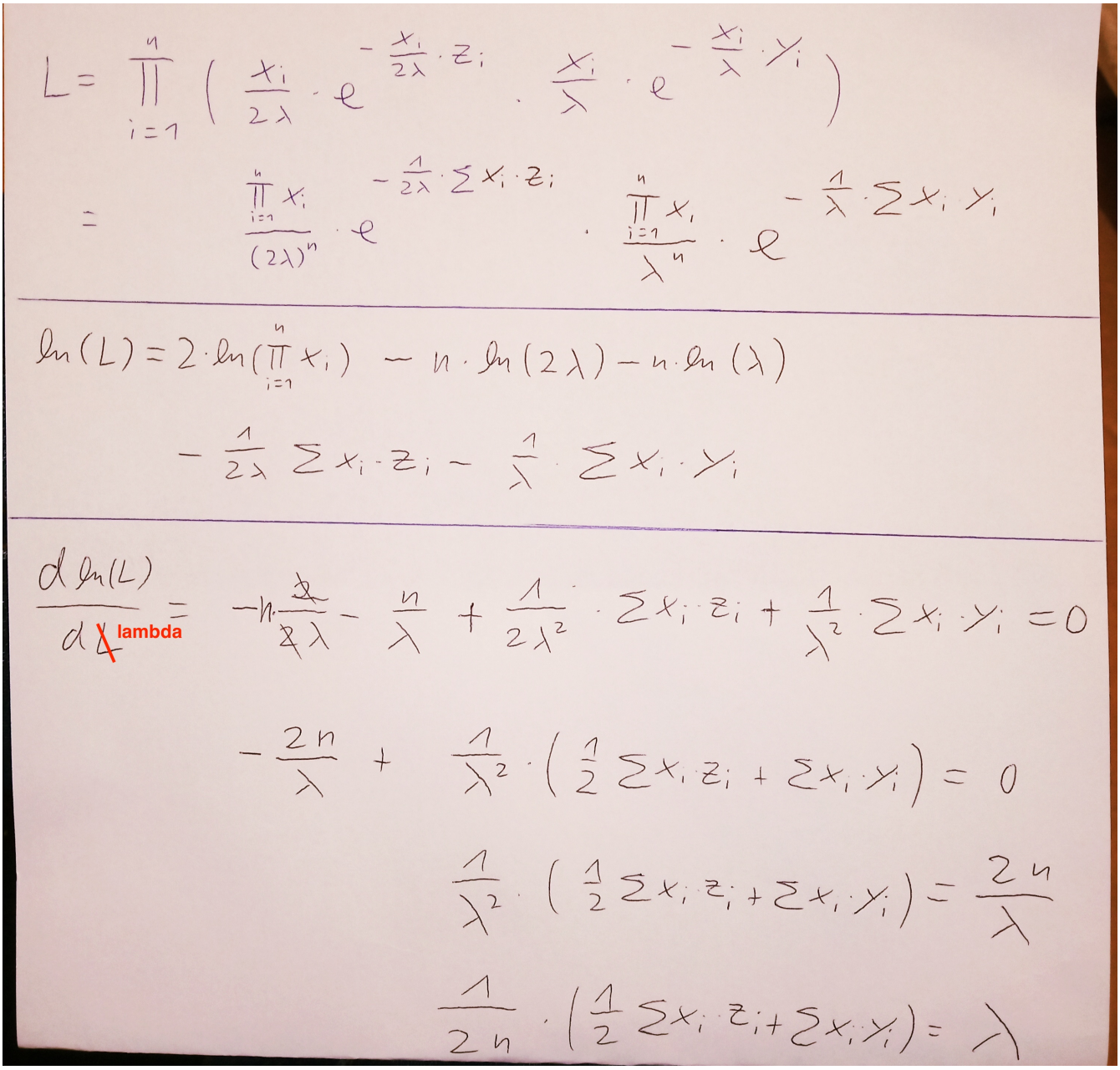
We know from theory that $\mu = \frac{1}{\text{rate}}$ for an exp. distribution $f(x) = \mathrm{r} \cdot e^{-\mathrm{r} \cdot x}$, where r is the rate.

M-step

To perform the M-step, we need to derive the maximum likelihood estimator for θ, λ_{MLE}. As usual, we find the likelihood, take the natural logarithm and take the first derivative w.r.t λ. Then, we set the derivative equal to 0 and solve for λ. The result is the following λ_{MLE}.

$$\lambda_{\text{MLE}} = \frac{1}{2N}\left(\frac{1}{2}X \bullet Z + X \bullet Y\right)$$

Note that in the vector Z, the `NA` values have been replaced by the values from the E-step. Please see the picture below for the complete derivation.



Reference: http://staff.ustc.edu.cn/~zwp/teach/MVA/em.pdf

2.2.2 Task 3: Implementation of EM-algorithm

Below, the previously derived EM-algorithm is implemented. The output shows that the **optimal lambda** is 10.7 and that 4 iterations were required to find this value.

```
# EM-algorithm (Method 1)  ---------------------------------------------------

N = nrow(df)
lambda = 100
iter = 0

X = as.matrix(df$X)
Y = as.matrix(df$Y)
Z = as.matrix(df$Z)
Z_NA_idx = which((is.na(Z)))
X_dot_Y = as.numeric(t(X) %*% Y)

while(TRUE){

  # Estimation Step: Update missing Z values
  Z[Z_NA_idx, 1] = 2 * lambda / X[Z_NA_idx]

  # Maximization Step: Update lambda
  lambda_prev = lambda
  lambda = 1/(2 * N) * (0.5 * t(X) %*% Z + X_dot_Y)

  # Stop if no relevant change in lambda
  if (abs(lambda - lambda_prev) < 0.001) break

  # Print status
  iter = iter + 1
  cat("Iteration: ", iter, " | Lambda: ", lambda, "\n")

}
```

```
## Iteration:  1  | Lambda:  14.26782
## Iteration:  2  | Lambda:  10.83853
## Iteration:  3  | Lambda:  10.70136
## Iteration:  4  | Lambda:  10.69587
```

2.3 Task 2 and 3: EM (Method 2)

2.3.1 Task 2: Derivation of EM algorithm

The derivation of λ_{MLE}, used in every M-step of method 2, is shown in the pictures below. The final formula for λ_{MLE} follows subsequently in LaTeX format.

$$L = \prod_{i=1}^{n} \left(\frac{x_i}{2\lambda} \cdot e^{-\frac{x_i}{2\lambda} \cdot z_i} \cdot \frac{x_i}{\lambda} \cdot e^{-\frac{x_i}{\lambda} y_i} \right)$$

$$= \prod_{i=1}^{n} \left(\frac{x_i}{\lambda} \cdot e^{-\frac{x_i}{\lambda} \cdot y_i} \right) \cdot \prod_{i=1}^{n_{obs}} \left(\frac{x_i}{2\lambda} \cdot e^{-\frac{x_i}{2\lambda} \cdot z_i} \right) \cdot \prod_{i=1}^{n_{unobs}} \left(\frac{x_i}{2\lambda} \cdot e^{-\frac{x_i}{2\lambda} \cdot z_i} \right)$$

$$= \frac{\prod_{i=1}^{n} x_i}{\lambda^n} \cdot e^{-\frac{1}{\lambda} \cdot \sum_{i=1}^{n} x_i y_i} \cdot \frac{\prod_{i=1}^{n_{obs}} x_i}{2\lambda^{n_{obs}}} \cdot e^{-\frac{1}{2\lambda} \cdot \sum_{i=1}^{n_{obs}} x_i \cdot z_i} \cdot \frac{\prod_{i=1}^{n_{unobs}} x_i}{2\lambda^{n_{unobs}}} \cdot e^{-\frac{1}{2\lambda} \sum_{i=1}^{n_{unobs}} x_i \cdot z_i}$$

$$\ln(L) = \underbrace{2 \cdot \sum_{i=1}^{n} \ln(x_i) - n \cdot \ln(\lambda) - n \cdot \ln(2\lambda) - \frac{1}{\lambda} \cdot \sum_{i=1}^{n} x_i y_i - \frac{1}{2\lambda} \cdot \sum_{i=1}^{n_{obs}} x_i \cdot z_i}_{a} \underbrace{- \frac{1}{2\lambda} \cdot \sum_{i=1}^{n_{unobs}} x_i \cdot z_i}_{b}$$

a is constant (given)
b is not constant (containing missing z_i)

$$E[\ln(L)] = E[a + b] = a + E[b]$$

$$= a - \frac{1}{2\lambda} \cdot E\left[\sum_{i=1}^{n_{unobs}} x_i \cdot z_i\right]$$

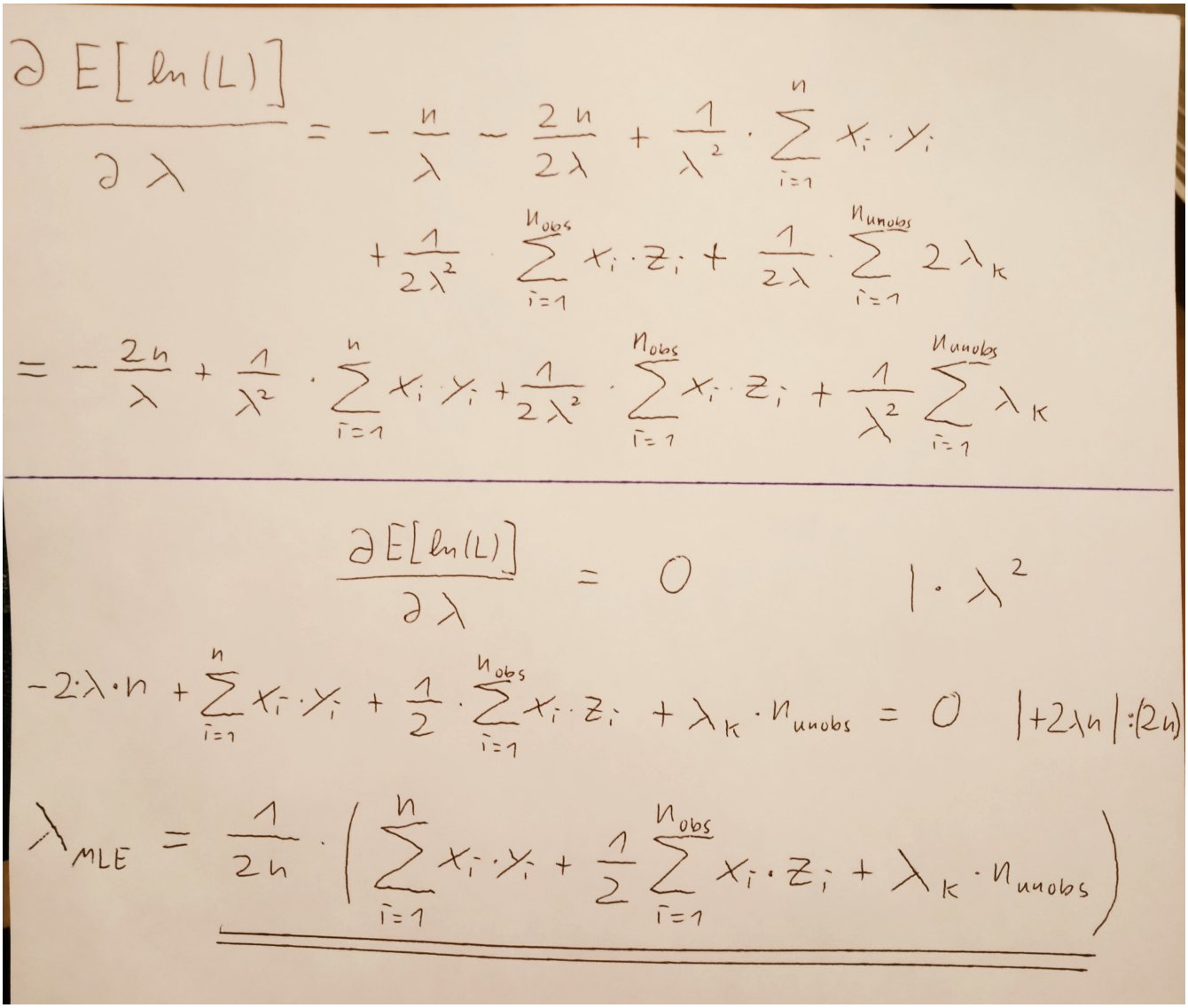
$$= a - \frac{1}{2\lambda} \cdot \sum_{i=1}^{n_{unobs}} x_i \cdot E[z_i]$$

$$= a - \frac{1}{2\lambda} \cdot \sum_{i=1}^{n_{unobs}} x_i \cdot \frac{2\lambda_k}{x_i}$$

λ_k is the λ from the previous iter.

$$= a - \frac{1}{\lambda} \cdot (n_{unobs} \cdot \lambda_k)$$

$$= Q(\lambda, \lambda^k)$$



Final formula

$$\lambda_{\text{MLE}} = \frac{1}{2N}\left(\sum_{i=1}^{N} X_i Y_i + \sum_{i=1}^{N_{obs}} X_i Z_i + \lambda_k N_{unobs}\right)$$

2.3.2 Task 3: Implementation of EM-algorithm

Comparing the printout of method 2 (below) with the printout of method 1 (above), we see that they are identical. Consequently, it seems like we can derive the EM-algorithm in different ways.

```
# EM-algorithm (Method 2)  ----------------------------------------------------

N = nrow(df)
lambda = 100
iter = 0
```

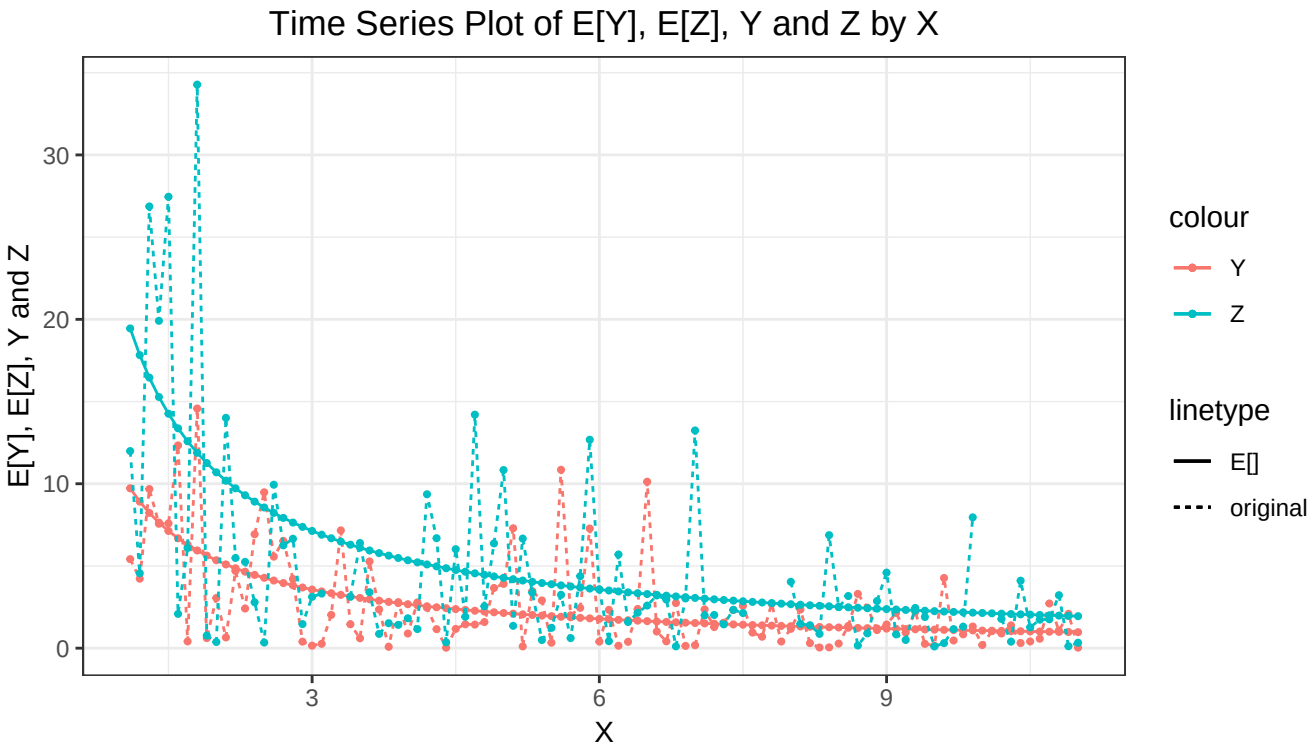
```r
X = as.matrix(df$X)
Y = as.matrix(df$Y)
Z = as.matrix(df$Z)

Z_NA_idx = which((is.na(Z)))
N_NA = length(Z_NA_idx)
Z_obs = as.matrix(df$Z[-Z_NA_idx])
X_obs = as.matrix(df$X[-Z_NA_idx])

X_dot_Y = as.numeric(t(X) %*% Y)
X_dot_Z_obs = as.numeric(t(X_obs) %*% Z_obs)

while(TRUE){

  # Estimation Step: on paper

  # Maximization Step: Update lambda
  lambda_prev = lambda
  lambda = 0.5/N * (X_dot_Y + 0.5 * X_dot_Z_obs + N_NA * lambda_prev)

  # Stop if no relevant change in lambda
  if (abs(lambda - lambda_prev) < 0.001) break

  # Print status
  iter = iter + 1
  cat("Iteration: ", iter, " | Lambda: ", lambda, "\n")

}
```

```
## Iteration:  1  | Lambda:  14.26782
## Iteration:  2  | Lambda:  10.83853
## Iteration:  3  | Lambda:  10.70136
## Iteration:  4  | Lambda:  10.69587
```

2.4 Task 4: Plot of $E[Y]$ and $E[Z]$ by X and conclusions

Below, we plot $E[Y]$, Y, $E[Z]$ and Z by X in the same plot. The downwards sloping green line represents $E[Z]$ and seems to approximate the center of the corresponding points Z well across X. Likewise, the downwards sloping red line represents $E[Y]$ and seems to approximate the center of the corresponding points Y well across X. Note this is exactly what we expect from a good λ_{MLE}. The reason for this is that we used λ_{MLE} to compute $E[Y]$ and $E[Z]$, the means of Y and Z for all corresponding values of X. Now, if $E[Y]$ and $E[Z]$ are roughly at the center of Y and Z, then λ_{MLE} did indeed help to compute means that represent the underlying distribution.

Consequently, we conclude that the estimated $\lambda_{\text{MLE}} = 10.7$ *seems to be reasonable.*



3 Appendix

```
# Set up general options

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

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

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

set.seed(12345)


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


f = function(x){
```

```
  res = (x^2 / exp(x)) - 2 * exp(-(9 * sin(x)) / (x^2 + x + 1))
  return(res)

}


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


crossover = function(x, y){

  res = (x + y) / 2
  return(res)

}


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


mutate = function(x){

  res = x^2 %% 30
  return(res)

}


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



genetic = function(maxiter, mutprob){

  # (a) ----------------------------------------------------------------------

  df_plot = data.frame(x = seq(0, 30, 0.1), fx = f(seq(0, 30, 0.1)))
  p = ggplot(df_plot) + theme_bw() +
    geom_line(aes(x = x, y = fx, color = "f(x) in general"),
              size = 0.5, alpha = 0.8, show.legend = FALSE) +
```

```r
  geom_vline(xintercept = 1.2, color = "red", linetype = "dashed") +
  labs(subtitle = paste0("maxiter: ", maxiter, " | mutprob: ", mutprob)) +
  theme(plot.subtitle = element_text(hjust = 0.5))

# (b) -------------------------------------------------------------------------

X = seq(0, 30, 5)

# (c) -------------------------------------------------------------------------

Values = f(X)

# (d) -------------------------------------------------------------------------

max_Values = numeric(maxiter)
max_X = numeric(maxiter)

for (i in 1:maxiter){

  parent_idx = sample(1:length(X), 2) # sample parent indices
  victim_idx = order(Values)[1] # sample victim index
  kid = crossover(X[parent_idx[1]], X[parent_idx[2]]) # create kid

  if (mutprob > runif(n = 1, min = 0, max = 1)){
    kid = mutate(kid) # mutate kid with probability mutprob
  }

  X[victim_idx] = kid # replace the victim by the kid
  Values[victim_idx] = f(kid) # update the Values
  max_Values[i] = max(Values) # save the current max value
  max_X[i] = X[which.max(Values)]

}

# (e) -------------------------------------------------------------------------

df_final = data.frame(x = X, fx = Values)
df_initial = data.frame(x = seq(0, 30, 5), fx = f(seq(0, 30, 5)))
df_maxval = data.frame(x = max_X[maxiter], fx = max_Values[maxiter])

p = p + geom_point(data = df_final,
                   aes(x = x, y = fx,color = "Final Observations"), size = 3,
                   shape = 21, show.legend = FALSE) +
        geom_point(data = df_initial,
                   aes(x = x, y = fx,color = "Initial Observations"), size = 3,
                   shape = 21, show.legend = FALSE) +
```

```
          geom_point(data = df_maxval,
                aes(x = x, y = fx,color = "Max. Value"), size = 3, alpha = 0.7,
                shape = 19, show.legend = FALSE) +
          scale_color_manual(values = c("steelblue4", "red",
                                        "forestgreen", "red"))

  # Prepare output
  return(list(X = X, Values = Values, max_Values = max_Values, p = p))

}


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

set.seed(12345)

res = list(genetic(maxiter = 10, mutprob = 0.1)$p,
           genetic(maxiter = 100, mutprob = 0.1)$p,
           genetic(maxiter = 10, mutprob = 0.5)$p,
           genetic(maxiter = 100, mutprob = 0.5)$p,
           genetic(maxiter = 10, mutprob = 0.9)$p,
           genetic(maxiter = 100, mutprob = 0.9)$p)

plot_list_arranged = gridExtra::arrangeGrob(grobs = res, ncol = 2)
gridExtra::grid.arrange(plot_list_arranged,
            top = "Final Population [in red] and Initial Population [in green]",
            padding = unit(0.5, "line"))


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

df = read.csv("physical1.csv")

ggplot(df, aes(x = X)) +
  geom_point(aes(y = Y, color = "Y")) + geom_line(aes(y = Y, color = "Y")) +
  geom_point(aes(y = Z, color = "Z")) + geom_line(aes(y = Z, color = "Z")) +
  labs(title = "Time Series Plot of Y(X) and Z(X) by X", y = "Y and Z") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


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

```r
# ------------------------------------------------------------------------------


knitr::kable(df[is.na(df$Z), ], booktabs = T,
             caption = paste0(nrow(df[is.na(df$Z), ]), " missing values of Z"))


knitr::include_graphics("images/Method1_M-step.jpg")


# Reference: Example from Olegs slides 2016 ------------------------------------

em.norm <- function(Y){

  Yobs <- Y[!is.na(Y)]
  Ymiss <- Y[is.na(Y)]
  n <- length(c(Yobs, Ymiss)) # N is the overall length!
  r <- length(Yobs)

  # Initial values
  mut <- 1 # mu at time t?
  sit <- 0.1 # sigma at time t?

  # Define log-likelihood function
  ll <- function(y, mu, sigma2, n){
    -0.5*n*log(2*pi*sigma2)-0.5*sum((y-mu)^2)/sigma2
  }

  # Compute the log-likelihood for the initial values
  lltm1 <- ll(Yobs, mut, sit, n)

  repeat{

    # E-step
    EY <- sum(Yobs) + (n-r)*mut # WHY???
    EY2 <- sum(Yobs^2) + (n-r)*(mut^2 + sit)# WHY???

    # M-step
    mut1 <- EY / n
    sit1 <- EY2 / n - mut1^2

    # Update parameter values
    mut <- mut1
    sit <- sit1

    # Compute log-likelihood using current estimates
```

```
    llt <- ll(Yobs, mut, sit, n)

    # Print current parameter values and likelihood
    cat(mut, sit, llt, "\n")

    # Stop if converged
    if (abs(lltm1 - llt) < 0.001) break
    lltm1 <- llt

  }
}

# Generate complete data N(5,1) and set 5 last as missing values
x <- rnorm(20,5)
x[16:20] <- NA
em.norm(x)


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


# EM-algorithm (Method 1)  -----------------------------------------------------

N = nrow(df)
lambda = 100
iter = 0

X = as.matrix(df$X)
Y = as.matrix(df$Y)
Z = as.matrix(df$Z)
Z_NA_idx = which((is.na(Z)))
X_dot_Y = as.numeric(t(X) %*% Y)

while(TRUE){

  # Estimation Step: Update missing Z values
  Z[Z_NA_idx, 1] = 2 * lambda / X[Z_NA_idx]

  # Maximization Step: Update lambda
  lambda_prev = lambda
  lambda = 1/(2 * N) * (0.5 * t(X) %*% Z + X_dot_Y)

  # Stop if no relevant change in lambda
  if (abs(lambda - lambda_prev) < 0.001) break
```

```r
  # Print status
  iter = iter + 1
  cat("Iteration: ", iter, " | Lambda: ", lambda, "\n")

}



knitr::include_graphics("images/Method2_M-step_Part1.jpg")



knitr::include_graphics("images/Method2_M-step_Part2.jpg")




# EM-algorithm (Method 2)  ------------------------------------------------

N = nrow(df)
lambda = 100
iter = 0

X = as.matrix(df$X)
Y = as.matrix(df$Y)
Z = as.matrix(df$Z)

Z_NA_idx = which((is.na(Z)))
N_NA = length(Z_NA_idx)
Z_obs = as.matrix(df$Z[-Z_NA_idx])
X_obs = as.matrix(df$X[-Z_NA_idx])

X_dot_Y = as.numeric(t(X) %*% Y)
X_dot_Z_obs = as.numeric(t(X_obs) %*% Z_obs)

while(TRUE){

  # Estimation Step: on paper

  # Maximization Step: Update lambda
  lambda_prev = lambda
  lambda = 0.5/N * (X_dot_Y + 0.5 * X_dot_Z_obs + N_NA * lambda_prev)

  # Stop if no relevant change in lambda
  if (abs(lambda - lambda_prev) < 0.001) break
```

```r
  # Print status
  iter = iter + 1
  cat("Iteration: ", iter, " | Lambda: ", lambda, "\n")

}



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


df_plot = data.frame(X = df$X, E_Y = lambda / df$X, E_Z = 2 * lambda / df$X)

ggplot(df_plot, aes(x = X)) +
  geom_point(aes(y = Y, color = "Y"), size = 0.75) +
  geom_line(aes(y = Y, color = "Y", linetype = "original")) +
  geom_point(aes(y = E_Y, color = "Y"), size = 0.75) +
  geom_line(aes(y = E_Y, color = "Y", linetype = "E[]")) +

  geom_point(aes(y = Z, color = "Z"), size = 0.75) +
  geom_line(aes(y = Z, color = "Z", linetype = "original")) +
  geom_point(aes(y = E_Z, color = "Z"), size = 0.75) +
  geom_line(aes(y = E_Z, color = "Z", linetype = "E[]")) +

  labs(title = "Time Series Plot of E[Y], E[Z], Y and Z by X",
       y = "E[Y], E[Z], Y and Z") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```