Lab 6

Stefano Toffol (steto820), Mim Kemal Tekin (mimte666)

08 March, 2019

Question 1: Genetic Algorithm

Task 1.1

Define the function

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

```
f_func = function(x){
  a = (x^2)/(exp(1)^x)
  b = 2*exp((-9*sin(x)) / (x^2 + x + 1))
  return (a-b)
}
```

Task 1.2

Define the function `crossover()`*: for two scalars x and y it returns their "kid" as $(x+y)/2$.*

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

Task 1.3

Define the function `mutate()` *that for a scalar x returns the result of the integer division x^2 mod 30.*

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

Task 1.4

Write a function that depends on the parameters maxiter and mutprob

We can see the plot of the function that we create above (Figure 1). This functions follows a sinusiodal behaviour and there are 5 maximum value in the function between [0,30]. We have 1 global and 4 local maximum. Our target is capture the global maximum value by using Genetic Algorithm which uses `mutate()` and `crossover()` functions implemented above.

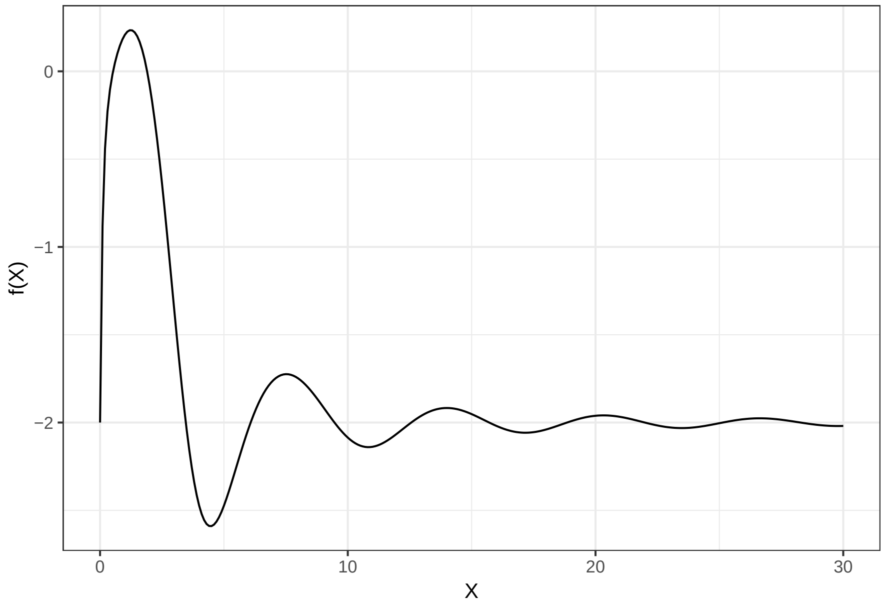

Figure 1: Plot of the function

```
maximize_genetic = function(maxiter, mutprob){
  # create a data frame between 0-30 and the values that are ans of f function
  data_f = data.frame(x = 0:300/10)
  data_f$y = sapply(data_f$x, f_func)

  # define the initial population as data.frame
  population0 = data.frame(x = 0:6*5)
  population0$y = sapply(population0$x, f_func)
  pop_size = dim(population0)[1]
  population = population0

  # initial plot
  initial_plot = ggplot() +
    geom_line(aes(x=data_f$x, y=data_f$y)) +
    geom_point(aes(x=population0$x, y=population0$y),
               color = "red", alpha = 0.35, size=3) +
    labs(title = paste("maxiter=",maxiter,"; mutprob=",mutprob),
         x = "X", y = "f(X)")

  for(i in 1:maxiter){
    # order the current population
    population = population[order(population$y, decreasing = T), ]
    # select parents || we select the parents without 1st element.
    # Because we want to save the largest element and we already ordered the pop
    index_parents = sample(1:pop_size, 2)
    parents = population[index_parents,]
    # select victim index which is the last index an ordered population
    index_victim = pop_size

    # create the kid with crossover
```

```r
    kid_x = crossover(parents$x[1], parents$x[2])
    # check mutation prob
    if(runif(1) <= mutprob){
      kid_x = mutate(kid_x)
      # print(paste("mutated_x:",kid_x,"; i:", i))
    }
    # replace kid with victim
    population$x[index_victim] = kid_x
    population$y[index_victim] = f_func(kid_x)
  }

  # order the current population
  population = population[order(population$y, decreasing = T), ]
  # add the final population to the initial plot
  initial_plot = initial_plot +
    geom_point(aes(x = population$x[2:pop_size], y = population$y[2:pop_size]),
               color="blue", alpha=0.6, size=3) +
    geom_point(aes(x = population$x[1], y = population$y[1]),
               color="brown", alpha=0.8, size=3) +
    theme_bw()
  return(b=initial_plot)
}
```

Task 1.5

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?

We run Genetic Algorithm with different arguments and we plot all results like above (Figure 2). Plots in the same column has same iteration counts. The red points are the initial populations. Brown and blue points are the final population that we obtain. Brown point is the highest value of that final population.
In the algorithm that we implemented, we define an initial population which is fixed before. We select a parent in every iteration randomly and we create a kid by taking mean of these 2 parents. After this, if uniform number which is drawn randomly is less than the mutation probability we mutate this kid by the function that created called `mutate()`. After that we replace this kid with the weakest individual which is smallest number in the population, since the problem is a maximization problem. But the approach of genetic algoritm is based on a random approach and more suitable for discrete domain problems. The target of algorithm is finding the best population for the problem by optimizing the population with cross-over, mutation and different selection strategies. In this problem we have a continues function which is created the function in the Task 1.1 and we are trying to solve this in a discrete way with genetic algorithm. This gives us an advantage in this domain, in contrast of hill climbing algorithms for maximizing problem, it reduces capturing the local maximum value. This way we have a bunch of individuals in population and it give a chance us to jumping over local minimums. But this does not mean that we will find a guaranteed global maximum, there is no guaranteed convergence even to local minimum.
If we examine plots of 10 iteration when we increase the mutation probability we can see the final population has wider spread. Only `maxiter=10` and `mutprob=0.9` algorithm execution has an individual which is really close to global maximum. But actually we cannot say this is because out mutation probability is high or another reason. Because 10 iteration is not enough to find an optimal population as we can see. We have to continue iterating until there is no changing individual in population or having a change less than a treshold. In the plot at right top side which has `maxiter=100` and `mutprob=0.1` we can see all indivudials stucked at somewhere irrelevant, this shows us sometimes we can get really wrong convergence. This is a side effect of having a discrete approach. In the following plot whic has `maxiter=100` and `mutprob=0.5` we have better solution with an increased mutation probability. Keeping mutation probability in a acceptable level can help

the convergence as we can see. Convergence is in right direction but it is only close to global. The algorithm without another mutation cannot find the global maximum because our cross-over function just takes the mean of two parents, so we need at least one mutation to change the weakest individual with a point which is located left of global maximum. But again everything depends on the next parents and an existence of this kind of mutation. In next steps it can happen such a situation like previous plot. In the last plot we finally find the global maximum with a really close convergence. In this execution we can see we have `mutprob=0.9` which means we will have many mutations. In this execution it did not affect the result in a bad way, but in general mutation should be rarer than this otherwise stability of population may be affected badly. Because this mutation function can make the individuals jump to long distances in the range.

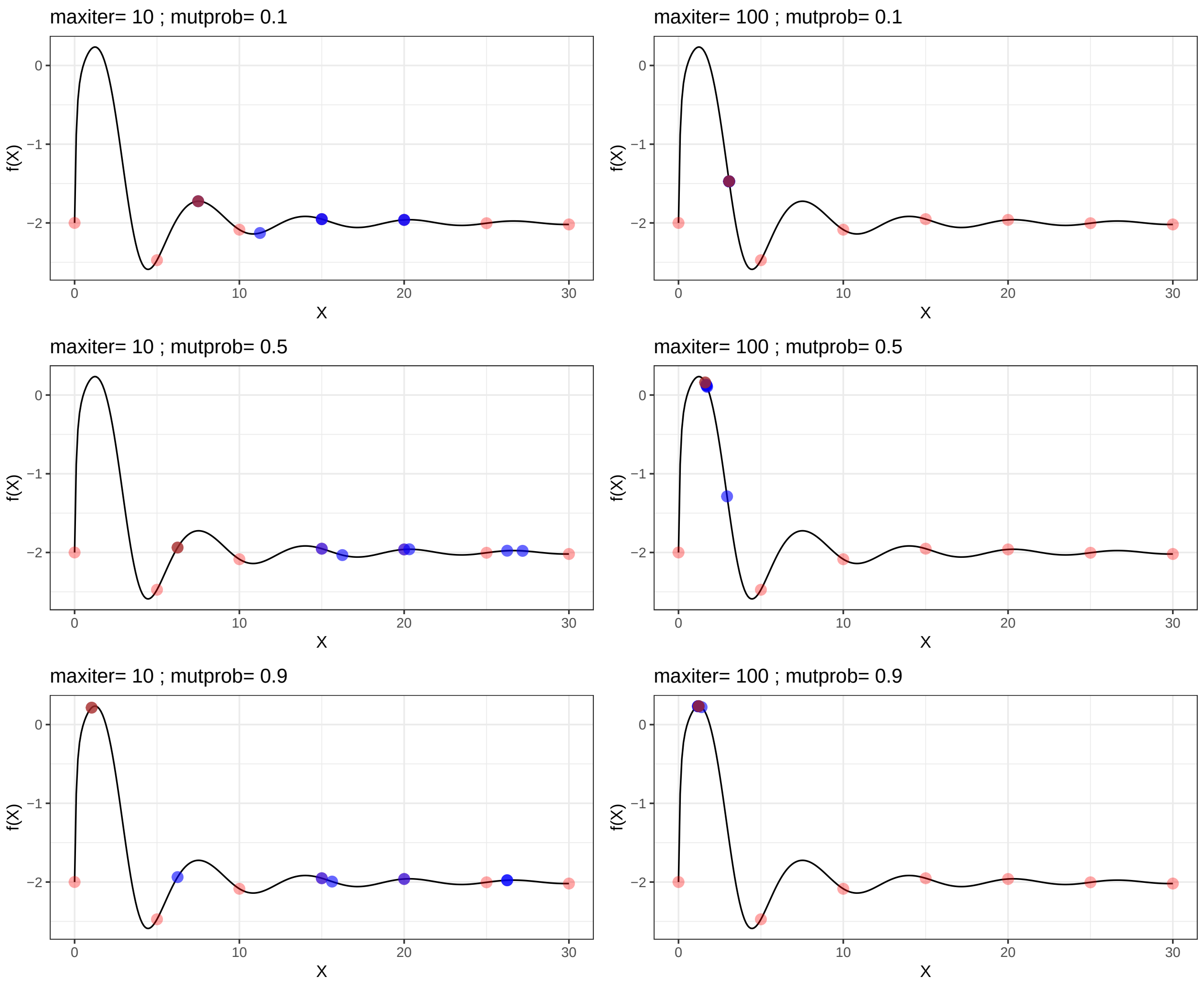

Figure 2: Results of the Genetic Algorithm using all the different combinations of iteration counts (columns) and mutation probability (rows).

Question 2

The data we have to analyse regard two different, yet related, physical processes Y and Z, both stochastic functions of the explanatory variable X.
In Figure 3 we can see the time series plot of our data. Both variables, despite the pretty wide oscillations registered, are starting at generally high value and tend to decrease over time following an exponential trend. The variable Z has three short strokes of missing values (4 observations maximum per stroke), corresponding to values of X around 3, 8 and 10. Nonetheless the two processes show a pretty strong relationship, both in the general trend and in the individual picks, and differ substantially for only the starting point ($Z_0 > Y_0$) and the variability ($Var(Z) = 35.5513 \ > \ Var(Y) = 8.682$). We can therefore state that the two variables are linked.

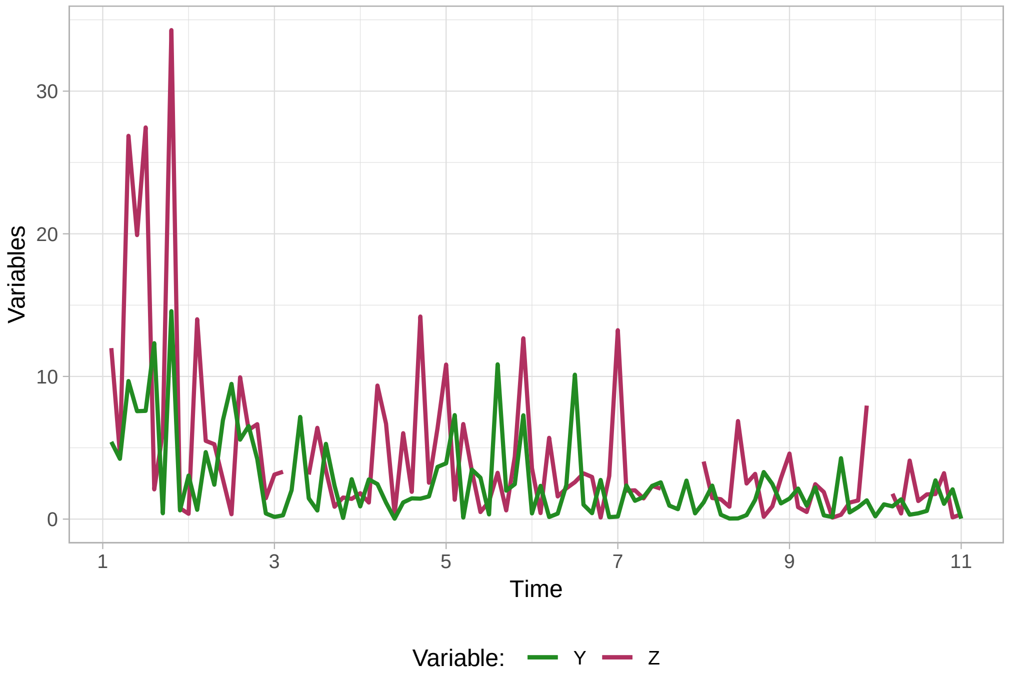

Figure 3: Time series of the two response variables.

We then assume that the two variables are realization of the following random independent variables:

$$Y_i \sim Exp\left(\frac{X_i}{\lambda}\right), \qquad Z_i \sim Exp\left(\frac{X_i}{2\lambda}\right)$$

In order to estimate the actual value of the parameter λ we are asked to implement the *EM algorithm.* Before jumping into the code, it's necessary to make some hand-written calculations.

The density functions of the two variables are:

$$f(Y_i) = \frac{X_i}{\lambda} \cdot \exp\left(-\frac{X_i}{\lambda}Y_i\right), \qquad f(Z_i) = \frac{X_i}{2\lambda} \cdot \exp\left(-\frac{X_i}{2\lambda}Z_i\right)$$

We then want to compute the log-likelihood of the model. The fact that the two variables are assumed to be independent (even though originated from the same observations X), allows us to treat the joint probability as the product of their marginals:

$$\mathcal{L}(\lambda|Y,Z) = \prod_{i=1}^{n} P(Y_i, Z_i|\lambda) = \prod_{i=1}^{n} P(Y_i|\lambda)P(Z_i|\lambda) = \prod_{i=1}^{n}\left[\frac{X_i}{\lambda}\cdot\exp\left(-\frac{X_i}{\lambda}Y_i\right)\right]\cdot\prod_{i=1}^{n}\left[\frac{X_i}{2\lambda}\cdot\exp\left(-\frac{X_i}{2\lambda}Z_i\right)\right]$$

The resulting log-likelihood is therefore the following:

$$\begin{aligned}\ell(\lambda|Y,Z) = \ln(\mathcal{L}(\lambda|Y,Z)) &= \sum_{i=1}^{n}\left[\ln\left(\frac{X_i}{\lambda}\right) - \frac{X_i}{\lambda}Y_i\right] + \sum_{i=1}^{n}\left[\ln\left(\frac{X_i}{2\lambda}\right) - \frac{X_i}{2\lambda}Z_i\right] \\ &= \left[\sum_{i=1}^{n}\ln(X_i) - n\ln(\lambda) + \sum_{i=1}^{n}\frac{X_i}{\lambda}Y_i\right] + \left[\sum_{i=1}^{n}\ln(X_i) - n\ln(2\lambda) - \sum_{i=1}^{n}\frac{X_i}{2\lambda}Z_i\right]\end{aligned}$$

We haven't however considered yet the presence of missing values among the Z variable, which will make impossible to compute the likelihood of our model using all the available data. In fact being Z a random variable we will never know the actual value of those missing data points, but we can try to predict the average value Z will assume at those corresponding X.
The easiest way to impute a missing value is to predict it with the mean of the variable. Having assumed an exponential model with parameter $\theta_i = X_i/2\lambda$, the mean will be equal to $1/\theta_i = 2\lambda/X_i$, which consists in our prediction for the missing values.
Consequentially we can divide Z in the observed and unobserved values, so $Z = (Z^o, Z^u)$, and compute the *expected* log-likelihood $E\Big[\ell(\lambda|Y,Z)\Big]$, for which we will be able to get an actual result since the missing values will be substituted with their imputations.

The expected log-likelihood will therefore be the log-likelihood computed above but considering the distinction between the vector of observed (Z^o, of length r) and unobserved (Z^u, of length $n-r$) values. The inputation of the missing values will require an intermediate estimate λ_k of the unknown parameter λ:

$$\begin{aligned}E\Big[\ell(\lambda|Y,Z)\Big] = &\overbrace{\left[\sum_{i=1}^{n}\ln(X_i) - n\ln(\lambda) + \sum_{i=1}^{n}\frac{X_i}{\lambda}Y_i\right]}^{E\left[\ell(\lambda|Y)\right]} + \overbrace{\left[\sum_{j=1}^{r}\ln(X_j) - r\ln(2\lambda) - \sum_{j=1}^{r}\frac{X_j}{2\lambda}Z_j^o\right]}^{E\left[\ell(\lambda|Z^o)\right]} + \cdots \\ &\cdots + \underbrace{\left[\sum_{m=r+1}^{n}\ln(X_m) - (n-r)\ln(2\lambda) - \sum_{m=r+1}^{n}\frac{\cancel{X_m}}{\cancel{2}\lambda}\cdot\frac{\cancel{2}\lambda_k}{\cancel{X_m}}\right]}_{E\left[\ell(\lambda|Z^u)\right]}\end{aligned}$$

We can now compute the derivative of the expected log-likelihood according to λ, in order to obtain its maximum likelihood estimation $\hat{\lambda}_{ML}$:

$$
\begin{aligned}
\frac{\delta \ell(\lambda|Y,Z)}{\delta\lambda} &= \left[0 - \frac{n}{\lambda} + \sum_{i=1}^{n} \frac{X_i}{\lambda^2} Y_i\right] + \left[0 - \frac{r}{\lambda} + \sum_{j=1}^{r} \frac{X_j}{2\lambda^2} Z_j^o\right] + \left[0 - \frac{n-r}{\lambda} + (n-r)\frac{\lambda_k}{\lambda^2}\right] \\
\implies \quad & \cancel{\frac{1}{\lambda}} \left(-n + \sum_{i=1}^{n} \frac{X_i}{\lambda} Y_i - r + \sum_{j=1}^{r} \frac{X_j}{2\lambda} Z_j^o - (n-r) + (n-r)\frac{\lambda_k}{\lambda} \right) = 0 \\
\implies \quad & -2n + \frac{1}{\lambda} \left(\sum_{i=1}^{n} X_i Y_i + \frac{1}{2} \sum_{j=1}^{r} X_j Z_j^o + (n-r)\lambda_k \right) = 0 \\
\implies \quad & \hat{\lambda}_{ML} = \frac{\sum_{i=1}^{n} X_i Y_i + \frac{1}{2}\sum_{j=1}^{r} X_j Z_j^o + (n-r)\lambda_k}{2n}
\end{aligned}
$$

It is now time to actually implement the algorithm. Following the requests of the task, the starting point should be $\lambda_0 = 100$ and the convergence should be reached when the change in successive estimates of the parameter, δ_λ, is $\delta_\lambda < 10^{-3}$. The code used to implement the algorithm is the following:

```r
# Expected log-likelihood function
twoexp_Ellik <- function(process, time, lambda, lambda_previous) {

  # Split the data into the individual processes
  x <- time
  y <- process[,1]
  z <- process[,2]
  # Check what are the observed Z
  index_obs <- !is.na(z)
  z_obs <- z[index_obs]
  x_obs <- x[index_obs]
  z_unobs <- z[!index_obs]
  x_unobs <- x[!index_obs]
  # Store the length of the vectors
  n <- length(time)   # Total number of observations
  r <- length(z_obs)  # Total number of complete cases

  # Likelihood contributions of each process
  loglik_Y <- -n*log(lambda) + sum(log(x)) - sum(x*y/lambda)
  loglik_Zobs <- -r*log(2*lambda) + sum(log(x_obs)) - sum(x_obs*z_obs/(2*lambda))
  loglik_Zunobs <- -(n-r)*log(2*lambda) + sum(log(x_unobs)) - (n-r)*lambda_previous/lambda

  return( loglik_Y + loglik_Zobs + loglik_Zunobs )

}
```

```r
EM_twoexp <- function(process, time, epsilon, max_iter, lambda_start){

  # Split the data into the individual processes
  x <- time
  y <- process[,1]
  z <- process[,2]
  # Variable of the observed Z (and their associated X values)
  index_obs <- !is.na(z)
  z_obs <- z[index_obs]
  x_obs <- x[index_obs]
  # Store the length of the vectors
  n <- length(time)   # Total number of observations
  r <- length(z_obs)  # Total number of complete cases

  # Initialize iterations count and lambda
  iter <- 1
  lambda_prev <- lambda_start-1-epsilon*100
  lambda_curr <- lambda_start
  # Initialize the log-likelihood
  loglik_curr <- twoexp_Ellik(process, time, lambda_curr, lambda_prev)

  # Variable to later check if covergence was reached
  converged <- F

  while ( (abs(lambda_prev-lambda_curr)>epsilon) && (iter<(max_iter+1)) ) {

    if(iter%%2==0)
      message(paste("Itaration n.o ", iter, "; Lambda = ", lambda_curr,
                    ", Likelihood = ", loglik_curr, sep = ""))

    lambda_prev <- lambda_curr

    # E-step and M-step together: the actual E-step was solved analytically
    lambda_curr <- (sum(x*y) + sum(x_obs*z_obs/2) + (n-r)*lambda_curr)/(2*n)

    # Re-compute the log-likelihood
    loglik_curr <- twoexp_Ellik(process, time, lambda_curr, lambda_prev)
    iter <- iter+1

  }

  # Check if convergence was reached
  if(iter<(max_iter+1))
    converged <- T

  return(list(estimate = lambda_curr, loglik = loglik_curr,
              convergence = converged, iterations = iter))

}


# Complete the task
epsilon <- 0.001
```

```r
maximum_iterations <- 1e+04
lambda_start <- 100
EM_estimate <- EM_twoexp(process, time, epsilon, maximum_iterations, lambda_start)
```

Using the requested criteria, the algorithm reach convergence within 6 iterations, the maximum likelihood estimation of the parameter is $\lambda_{ML} = 10.695655$, which corresponds to an expected log-likelihood $E\Big[\ell(\lambda|Y, Z)\Big] = -413.362535$.

We also tried to run the algorithm using a different starting point and a much higher precision, in order to evaluate its performances in case of far away starting points. It turned out that even decreasing the convergence criteria to $\epsilon = 10^{-5}$ and starting from $\lambda_0 = 10^5$ the results are practically the same, and the convergence is reached within 10 iterations.

In order to visually evaluate the performances of the model we plot the expected values of both Y and Z versus time. The predictions are corresponding to the mean of the exponential distribution using the estimated parameter λ_{ML}. Therefore the predictions correspond to $E[Y] = \frac{\lambda_{ML}}{X}$ and $E[Z] = \frac{2\lambda_{ML}}{X}$.
In Figure 4 they are represented as red and black dots respectively. As we can observe the mean manages to capture the overall trend of the data, even though it does not model the large portion of random noise present: consider for instance the peaks for $1 \leq X \leq 2$ and the ones between $4 \leq X \leq 7$. Probably for this reason the predictions for the process Z (black dots) tend to overestimate the real data: the sum of the differences between observed and predicted values (sum of the residuals) is in fact equal to -61.3123. On the other hand Y is slightly overestimated, since the same difference is equal to 6.2842. Nonetheless the computed λ does indeed seem reasonable, since it captures the general trend of the data. We should also consider that the variance of the model ($Var(Y) = (\lambda/X)^2$ and $Var(Z) = (2\lambda/X)^2$) respects the one observed in the real data.

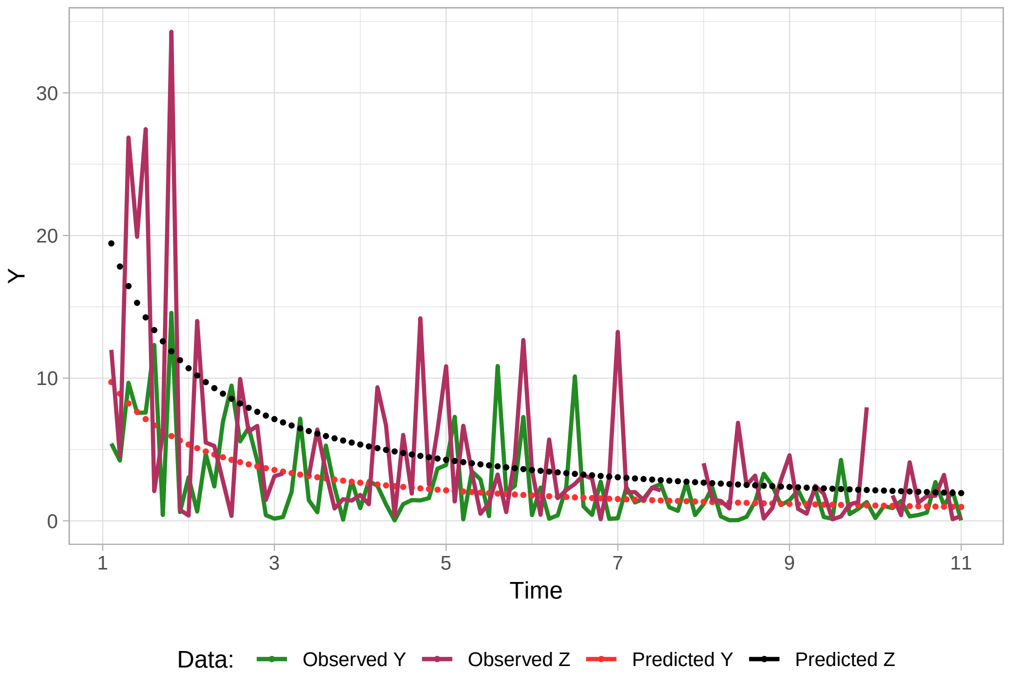

Figure 4: Time series of both the observed and of the predicted values of the two responses variables.

Appendix

```r
knitr::opts_chunk$set(echo = F, message = F, error = F, warning = F,
                      fig.align='center', out.width="70%")


f_func = function(x){
  a = (x^2)/(exp(1)^x)
  b = 2*exp((-9*sin(x)) / (x^2 + x + 1))
  return (a-b)
}


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


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


library(ggplot2)
library(gridExtra)

data_f = data.frame(x = 0:300/10)
data_f$y = sapply(data_f$x, f_func)
ggplot() +
    geom_line(aes(x=data_f$x, y=data_f$y)) +
    labs(x = "X", y="f(X)") +
    theme_bw()


maximize_genetic = function(maxiter, mutprob){
  # create a data frame between 0-30 and the values that are ans of f function
  data_f = data.frame(x = 0:300/10)
  data_f$y = sapply(data_f$x, f_func)

  # define the initial population as data.frame
  population0 = data.frame(x = 0:6*5)
  population0$y = sapply(population0$x, f_func)
  pop_size = dim(population0)[1]
  population = population0

  # initial plot
  initial_plot = ggplot() +
    geom_line(aes(x=data_f$x, y=data_f$y)) +
    geom_point(aes(x=population0$x, y=population0$y),
               color = "red", alpha = 0.35, size=3) +
    labs(title = paste("maxiter=",maxiter,"; mutprob=",mutprob),
         x = "X", y = "f(X)")
  for(i in 1:maxiter){
    # order the current population
    population = population[order(population$y, decreasing = T), ]
    # select parents || we select the parents without 1st element.
    # Because we want to save the largest element and we already ordered the pop
    index_parents = sample(1:pop_size, 2)
    parents = population[index_parents,]
    # select victim index which is the last index an ordered population
    index_victim = pop_size

    # create the kid with crossover
    kid_x = crossover(parents$x[1], parents$x[2])
    # check mutation prob
    if(runif(1) <= mutprob){
      kid_x = mutate(kid_x)
      # print(paste("mutated_x:",kid_x,"; i:", i))
    }
    # replace kid with victim
    population$x[index_victim] = kid_x
    population$y[index_victim] = f_func(kid_x)
  }

  # order the current population
  population = population[order(population$y, decreasing = T), ]
  # add the final population to the initial plot
  initial_plot = initial_plot +
    geom_point(aes(x = population$x[2:pop_size], y = population$y[2:pop_size]),
               color="blue", alpha=0.6, size=3) +
    geom_point(aes(x = population$x[1], y = population$y[1]),
               color="brown", alpha=0.8, size=3) +
    theme_bw()
  return(b=initial_plot)
}


result_plots = list()
maxiter = c(10, 100)
mutprob = c(0.1, 0.5, 0.9)
set.seed(12345)
for(prob in mutprob){
  for(iter in maxiter){
    result_plots[[paste("pl", iter, prob, sep = "_")]] =
      maximize_genetic(iter, prob)
  }
}
grid.arrange(grobs = result_plots, ncol=2)


# ------------------------------------------------------------------------------
# A2
# ------------------------------------------------------------------------------
# Read the data
data <- read.csv("../datasets/physical1.csv")
# Split in process and time
process <- data[,2:3]
time <- data[,1]


library(ggplot2)

ggplot(data, aes(x = X)) +
  geom_line(aes(y = Z, col = "Z"), size = .9) +
  geom_line(aes(y = Y, col = "Y"), size = .9) +
  scale_color_manual(values = c("forestgreen", "maroon"), name = "Variable: ") +
  labs(y = "Variables", x = "Time") +
  theme_light() +
  scale_x_continuous(breaks = seq(1, 11, 2)) +
  theme(legend.position = "bottom")


# Expected log-likelihood function
twoexp_Ellik <- function(process, time, lambda, lambda_previous) {

  # Split the data into the individual processes
  x <- time
  y <- process[,1]
  z <- process[,2]
  # Check what are the observed Z
  index_obs <- !is.na(z)
  z_obs <- z[index_obs]
  x_obs <- x[index_obs]
  z_unobs <- z[!index_obs]
  x_unobs <- x[!index_obs]
  # Store the length of the vectors
  n <- length(time)   # Total number of observations
  r <- length(z_obs)  # Total number of complete cases

  # Likelihood contributions of each process
  loglik_Y <- -n*log(lambda) + sum(log(x)) - sum(x*y/lambda)
  loglik_Zobs <- -r*log(2*lambda) + sum(log(x_obs)) - sum(x_obs*z_obs/(2*lambda))
  loglik_Zunobs <- -(n-r)*log(2*lambda) + sum(log(x_unobs)) - (n-r)*lambda_previous/lambda

  return( loglik_Y + loglik_Zobs + loglik_Zunobs )

}





EM_twoexp <- function(process, time, epsilon, max_iter, lambda_start){

  # Split the data into the individual processes
  x <- time
```

```r
  y <- process[,1]
  z <- process[,2]
  # Variable of the observed Z (and their associated X values)
  index_obs <- !is.na(z)
  z_obs <- z[index_obs]
  x_obs <- x[index_obs]
  # Store the length of the vectors
  n <- length(time)    # Total number of observations
  r <- length(z_obs)  # Total number of complete cases

  # Initialize iterations count and lambda
  iter <- 1
  lambda_prev <- lambda_start-1-epsilon*100
  lambda_curr <- lambda_start
  # Initialize the log-likelihood
  loglik_curr <- twoexp_Ellik(process, time, lambda_curr, lambda_prev)

  # Variable to later check if covergence was reached
  converged <- F

  while ( (abs(lambda_prev-lambda_curr)>epsilon) && (iter<(max_iter+1)) ) {

    if(iter%%2==0)
      message(paste("Itaration n.o ", iter, "; Lambda = ", lambda_curr,
                    ", Likelihood = ", loglik_curr, sep = ""))

    lambda_prev <- lambda_curr

    # E-step and M-step together: the actual E-step was solved analytically
    lambda_curr <- (sum(x*y) + sum(x_obs*z_obs/2) + (n-r)*lambda_curr)/(2*n)

    # Re-compute the log-likelihood
    loglik_curr <- twoexp_Ellik(process, time, lambda_curr, lambda_prev)
    iter <- iter+1

  }

  # Check if convergence was reached
  if(iter<(max_iter+1))
    converged <- T

  return(list(estimate = lambda_curr, loglik = loglik_curr,
              convergence = converged, iterations = iter))

}


# Complete the task
epsilon <- 0.001
maximum_iterations <- 1e+04
lambda_start <- 100
EM_estimate <- EM_twoexp(process, time, epsilon, maximum_iterations, lambda_start)
```

```r
# Let's try again with higher accuracy
epsilon <- 0.00001
# Pick a starting lambda far away from the optimum
lambda_start <- 1e+05
EM_estimate2 <- EM_twoexp(process, time, epsilon, maximum_iterations, lambda_start)

# Predictions
Y_hat <- EM_estimate$estimate/data$X
Z_hat <- 2*EM_estimate$estimate/data$X

data_hat <- data.frame(X = data$X, Y = data$Y, Y_hat = Y_hat,
                       Z = data$Z, Z_hat = Z_hat)


p1 <- ggplot(data_hat, aes(x = X)) +
  geom_line(aes(y = Y, col = "Real value"), size = .9) +
  geom_point(aes(y = Y_hat, col = "Prediction"), size = 1, shape = 16) +
  scale_color_manual(values = c("firebrick1", "forestgreen"), name = "Data: ") +
  labs(y = "Y", x = "Time") +
  theme_light() +
  scale_x_continuous(breaks = seq(1, 11, 2)) +
  theme(legend.position = "bottom")

p2 <- ggplot(data_hat, aes(x = X)) +
  geom_line(aes(y = Z, col = "Real value"), size = .9) +
  geom_point(aes(y = Z_hat, col = "Prediction"), size = .9, shape = 16) +
  scale_color_manual(values = c("firebrick1", "maroon"), name = "Data: ") +
  labs(y = "Z", x = "Time") +
  theme_light() +
  scale_x_continuous(breaks = seq(1, 11, 2)) +
  theme(legend.position = "bottom")

ggplot(data_hat, aes(x = X)) +
  geom_line(aes(y = Y, col = "Observed Y"), size = .9) +
  geom_point(aes(y = Y_hat, col = "Predicted Y"), size = 1, shape = 16) +
  geom_line(aes(y = Z, col = "Observed Z"), size = .9) +
  geom_point(aes(y = Z_hat, col = "Predicted Z"), size = 1, shape = 16) +
  scale_color_manual(values = c("forestgreen", "maroon", "firebrick1", "black"),
                     name = "Data: ") +
  labs(y = "Y", x = "Time") +
  theme_light() +
  scale_x_continuous(breaks = seq(1, 11, 2)) +
  theme(legend.position = "bottom")
```