Lab 6 report

Nicolas Taba & Yuki Washio

14/12/2020

Question 1

1 Define function to maximize

```
f_x <- function(x) {
  return((x^2 / exp(x)) - 2 * exp((-(9 * sin(x)) / (x^2 + x + 1))))
}
```

2 Define crossover()

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

3 Define mutate()

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

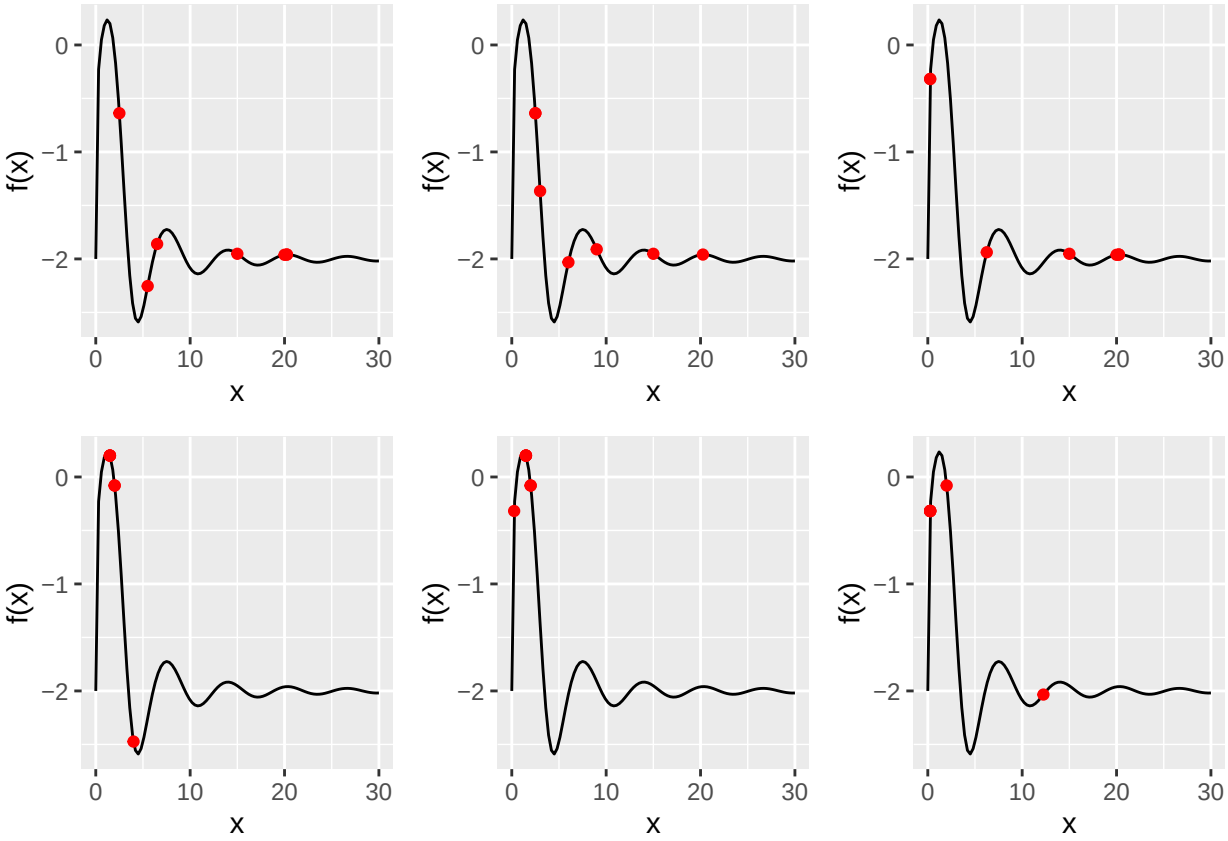
4 Define genetic algorithm

```
library(ggplot2)

genetic <- function(maxiter, mutprob) {
  # a) plot function f in range [0,30]
  my_plot = ggplot() + stat_function(fun=f_x) + xlim(0,30) + xlab("x") + ylab("f(x)")
  my_plot
  # b) initial population for the genetic algorithm as X = (0, 5, 10, 15, . . . , 30)
  initial_pop = seq(0,30,5)
  population = initial_pop
  # c) compute vector Values that contain the function values for each in population
  function_values_pop = f_x(population)
  # d) iterate
  for (i in 1:maxiter) {
    # sample two indexes from population
    parents = sample(1:length(population), 2)
    # select index with smallest objective function as victim
    victim = order(function_values_pop)[1]
    # Parents are used to produce a new kid by crossover. Mutate this kid with probability mutprob (use
    new_kid = crossover(parents[1], parents[2])
```

```
    if (as.logical(rbinom(1, size=1, prob=mutprob))) {
      new_kid = mutate(new_kid)
    }
    # The victim is replaced by the kid in the population and the vector Values is updated.
    population[victim] = new_kid
    function_values_pop = f_x(population)
    # The current maximal value of the objective function is saved.
    max_val = max(function_values_pop)
  }
  # e) Add the final observations to the current plot in another colour.
  final_plot = my_plot + geom_point(aes(x=population, y=function_values_pop), color="red")
  final_plot
  return(list(max_val=max_val, initial_pop=initial_pop, final_pop=population, final_fun_values=function_
}
```

We can observe a maximum value at around $f(1.5) = 0.2$.

5 Try out different parameter combinations




```
## 1
## maxiter:  10
## mutprob:  0.1
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  5.5 20.25 6.5 15 20 20.25 2.5
## max_val:  -0.6380611
##
## 2
```

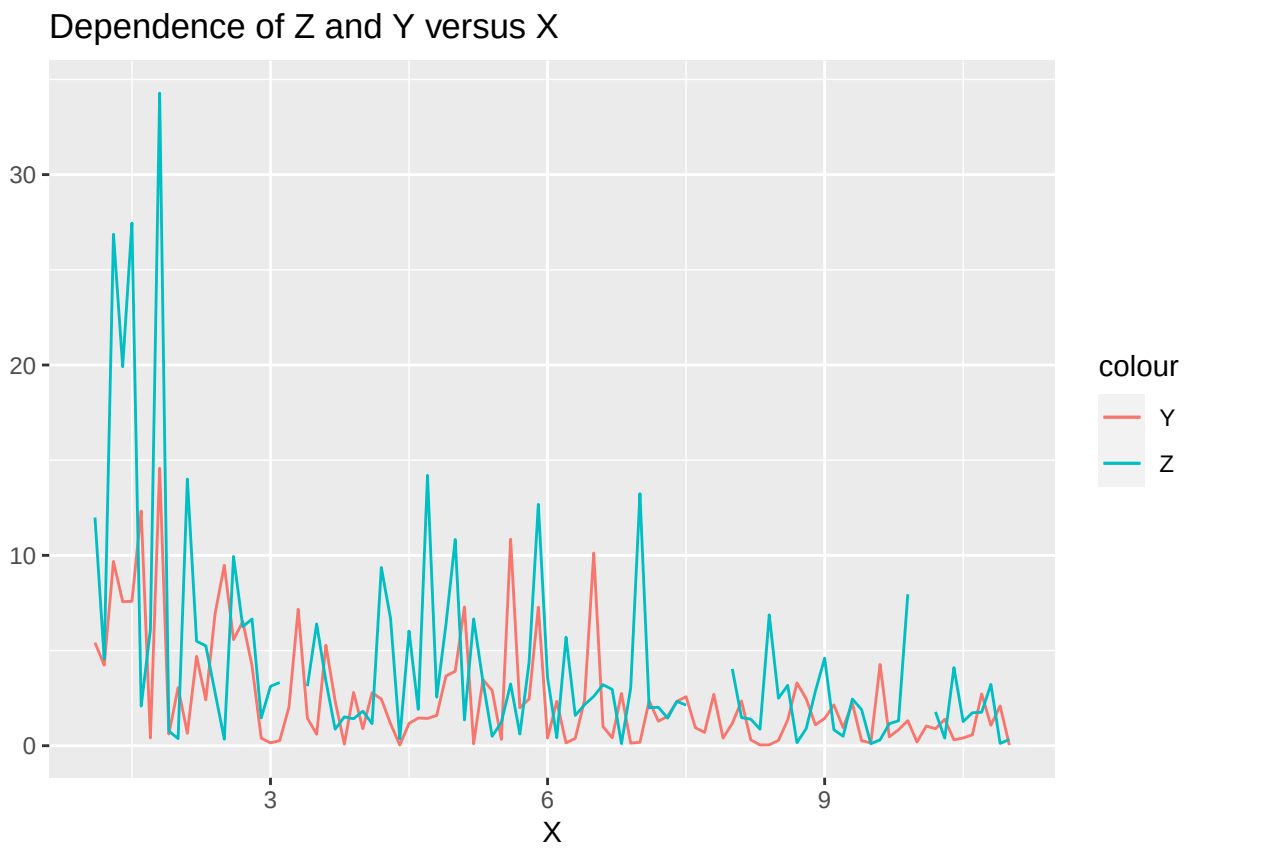
```
## maxiter:  10
## mutprob:  0.5
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  6 9 20.25 15 3 2.5 2.5
## max_val:  -0.6380611
##
## 3
## maxiter:  10
## mutprob:  0.9
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  0.25 20.25 20.25 15 20 0.25 6.25
## max_val:  -0.3179788
##
## 4
## maxiter:  100
## mutprob:  0.1
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  1.5 1.5 1.5 4 2 2 1.5
## max_val:  0.1998964
##
## 5
## maxiter:  100
## mutprob:  0.5
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  1.5 1.5 1.5 1.5 0.25 2 2
## max_val:  0.1998964
##
## 6
## maxiter:  100
## mutprob:  0.9
## initial_pop:  0 5 10 15 20 25 30
## final_popp:  12.25 0.25 0.25 0.25 0.25 2 0.25
## max_val:  -0.07995372
```

We are plotting all possible combinations. The first plot row shows all plots where the maximum number of iterations is 10. The second plot row shows all plots where the maximum number of iterations is 100. We can observe that our genetic algorithm is not able to find the maximum with just 10 iterations no matter how big the mutation probability is. Though, we can also observe that given 100 iterations our genetic algorithm returns good results in most cases. Setting mutation probability to 0.1 or 0.5 returns the correct maximum value. For mutation probability 0.9 we were not able to hit the true maximum. This could be because of the high probability of not mutating the crossover result.

Question 2

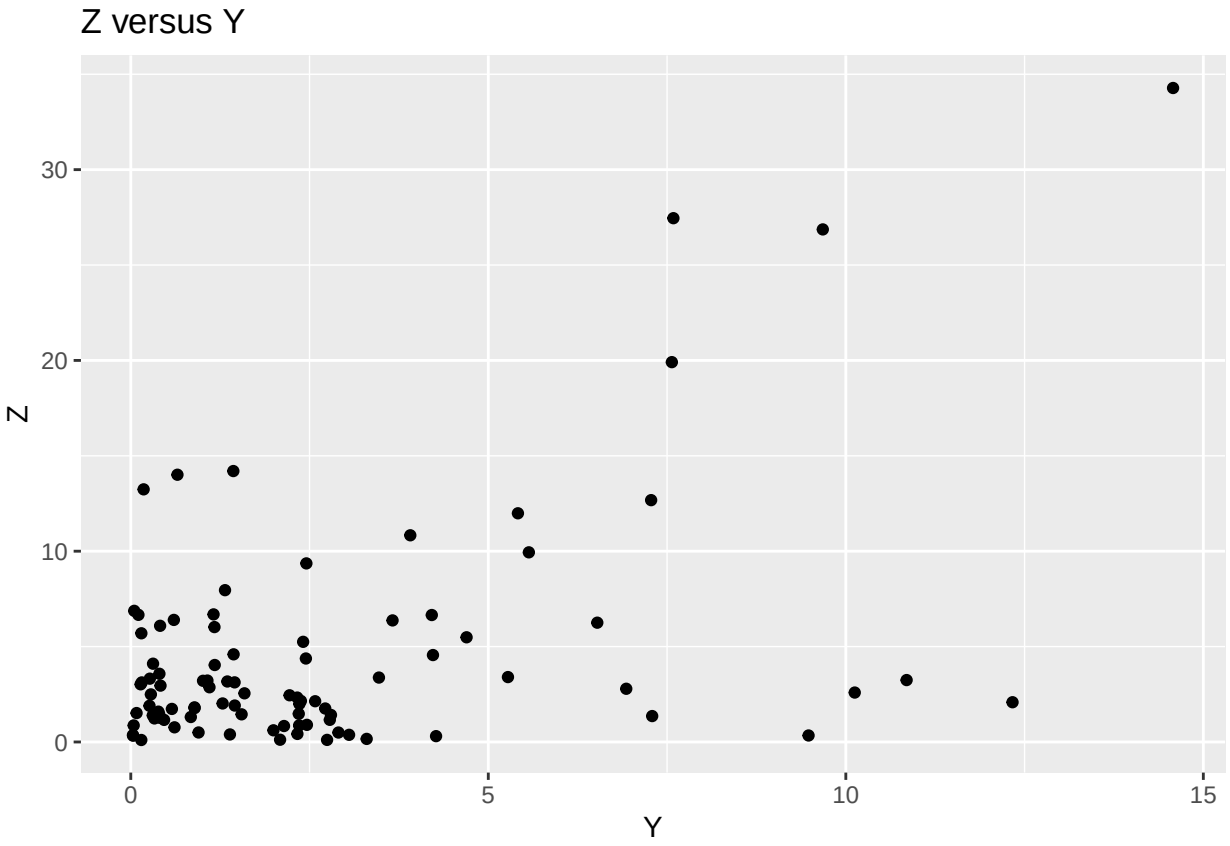
1: Make a time series plot of the data with respect to X

```
data <- read.csv("physical1.csv")
library(ggplot2)

plot1 <- ggplot(data)+
  geom_line(aes(x=X, y=Y, color = "Y"))+
  geom_line(aes(x=X, y=Z, color = "Z"))+
  labs(x="X", y="", title = "Dependence of Z and Y versus X")

plot2 <- ggplot(data, aes(x=Y, y=Z))+
  geom_point()+
  labs(title="Z versus Y")

plot1
```




```
plot2
```

```
## Warning: Removed 8 rows containing missing values (geom_point).
```



The two processes seem related when plotting the Y and Z variable against X. Both processes have a decaying trend with sharp spikes that seem to have a delay between them. The second plot represents a plot of the Z variable bersus Y. There seems to be no clear correlation between these two variables. We can assume that the two processes are independent even if they stem from the X variable.

2: derive an EM algorithm that estimates lambda

Variables Y and Z obey the following model:

$$Y_i \sim Exp\left(\frac{X_i}{\lambda}\right) \text{ which has a density function } f(Y_i) = \frac{X_i}{\lambda} \cdot \exp\left(-\frac{X_i}{\lambda} Y_i\right)$$

and

$$Z_i \sim Exp\left(\frac{X_i}{2\lambda}\right) \text{ which has a density function } f(Z_i) = \frac{X_i}{2\lambda} \cdot \exp\left(-\frac{X_i}{2\lambda} Z_i\right)$$

We then compute the likelihood of this model. The two variables are assumed to be independent. This yields a likelihood that is the product of the joint probabilities. From it, we then compute the following log-likelihood:

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

where the logarithm of a product becomes a sum over all points, the exponential term is reduced to itself. We can further simplify this expression by computing the first logarithm terms that do not depend on Y_i and Z_i repsectively in each sum:

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

We have missing data in our Z_i set which makes the previous equation impossible to compute. We must impute some data to replace the missing data. A reasonable approach would be to replace missing values for our random variable with the average value of Z_i at the missing point X_i. The mean value of Z_i at any X_i point, under the exponential model, is : $2\lambda/X_i$. The λ in this expression is an estimate of it. In practice, we will choose the value of lambda of the previous point. We thus separate the set into "missing set" and "non-missing set". We can then compute the expected log-likelihood from this information which adds another term to the previous equation of the log-likelihood by separating the second term into 2. We will be performing the second term sum of the log-likelihood from 1 to m and then calculating the missing values from $m+1$ to n and replacing the sum over Z_i by our estimate:

$$E\Big[\ln(L(\lambda|Y,Z))\Big] = \left[\sum_{i=1}^{n} -n\ln(\lambda) + \ln(X_i) + \sum_{i=1}^{n}\frac{X_i}{\lambda}Y_i\right] + \left[\sum_{j=1}^{m} -m\ln(2\lambda) + \ln(X_j) - \sum_{j=1}^{m}\frac{X_j}{2\lambda}Z_j\right] + \left[\sum_{k=m+1}^{n} -(n-m)\ln(2\lambda) + \ln(X_k) - \sum_{k=m+1}^{n}\frac{X_k}{2\lambda}\cdot\frac{2\lambda_k}{X_k}\right]$$

The X_k terms cancel out. We can now perform the maximization step by taking the derivative of the expected log-likelihood with respect to λ and setting it to zero. The middle term of each term of this expression will be reduced to zero as they do not depend on λ and we can factorize out $1/\lambda$ from the subsequent equation:

$$\frac{dE\Big[\ln(L(\lambda|Y,Z))\Big]}{d\lambda} = 0$$

$$\implies \left(-n + \sum_{i=1}^{n}\frac{X_i}{\lambda}Y_i - m + \sum_{j=1}^{m}\frac{X_j}{2\lambda}Z_j - (n-m) + (n-m)\frac{\lambda_k}{\lambda}\right) = 0$$

We factor out again $1/\lambda$ and can compute the constants to get $-2n$. By rearranging the terms we obtain the maximum likelihood estimate for λ as:

$$\hat{\lambda}_{ML} = \frac{\sum_{i=1}^{n} X_iY_i + \frac{1}{2}\sum_{j=1}^{m} X_jZ_j + (n-m)\lambda_k}{2n}$$

Where Z_j is our observed data, m is the number of observed points for our random variable and λ_k will be the previous value.

Implement the previously calculated algorithm in R

We implement the EM algorithm with the followinf restrictions: $\lambda_0 = 100$ and the stopping condition is when the change in λ is smaller than 10^{-3}

```
floglik <- function(data, lambda, lambda_prev){
  X <- data[,1]
  Y <- data[,2]
  Z <- data[,3]
  # missing and not missing data
  index <- is.na(Z)
```

```
  X_missing <- X[index]
  Z_not <- Z[!index]
  X_not <- X[!index]
  # n and m
  n <- length(X)
  m <- length(Z_not)
  # log likelihood
  Y_term <- (- n * log(lambda)) + sum(log(X)) - sum(X*Y/lambda)
  Z_not_term <- (- m * log(2*lambda)) + sum(log(X_not)) - sum(X_not*Z_not/(2*lambda))
  Z_missing_term <- (-(n-m)*log(2*lambda)) + sum(log(X_missing)) - ((n-m)*lambda_prev/lambda)

  log_like <- Y_term + Z_not_term + Z_missing_term
  return(log_like)
}

EM_algo <- function(data, eps, k_max, lambda_0){
  X <- data[,1]
  Y <- data[,2]
  Z <- data[,3]
  # missing and not missing data
  index <- is.na(Z)
  X_missing <- X[index]
  Z_not <- Z[!index]
  X_not <- X[!index]
  # n and m
  n <- length(X)
  m <- length(Z_not)

  #initialize everything
  k <- 1
  lambda_current <- lambda_0
  lambda_prev <- lambda_0+1+100*eps
  llvalcurr <- floglik(data, lambda_current, lambda_prev)

  # EM algorithm
  while( (abs(lambda_prev-lambda_current) > eps) &&  (k < (k_max+1)) ){
    lambda_prev <- lambda_current

    # EM step
    lambda_current <- (sum(X*Y) + sum(X_not*Z_not/2) + (n-m)*lambda_current)/(2*n)
    # new log-likelihood
    llvalcurr <- floglik(data, lambda_current, lambda_prev)
    k <- k+1
  }
  results <- list(steps = k, lambda = lambda_current)
  return(results)
}

epsilon <- 0.001
my_lambda_0 <- 100
my_k_max <- 100000

res <- EM_algo(data, eps = epsilon, k_max = my_k_max, lambda_0 = my_lambda_0)
```

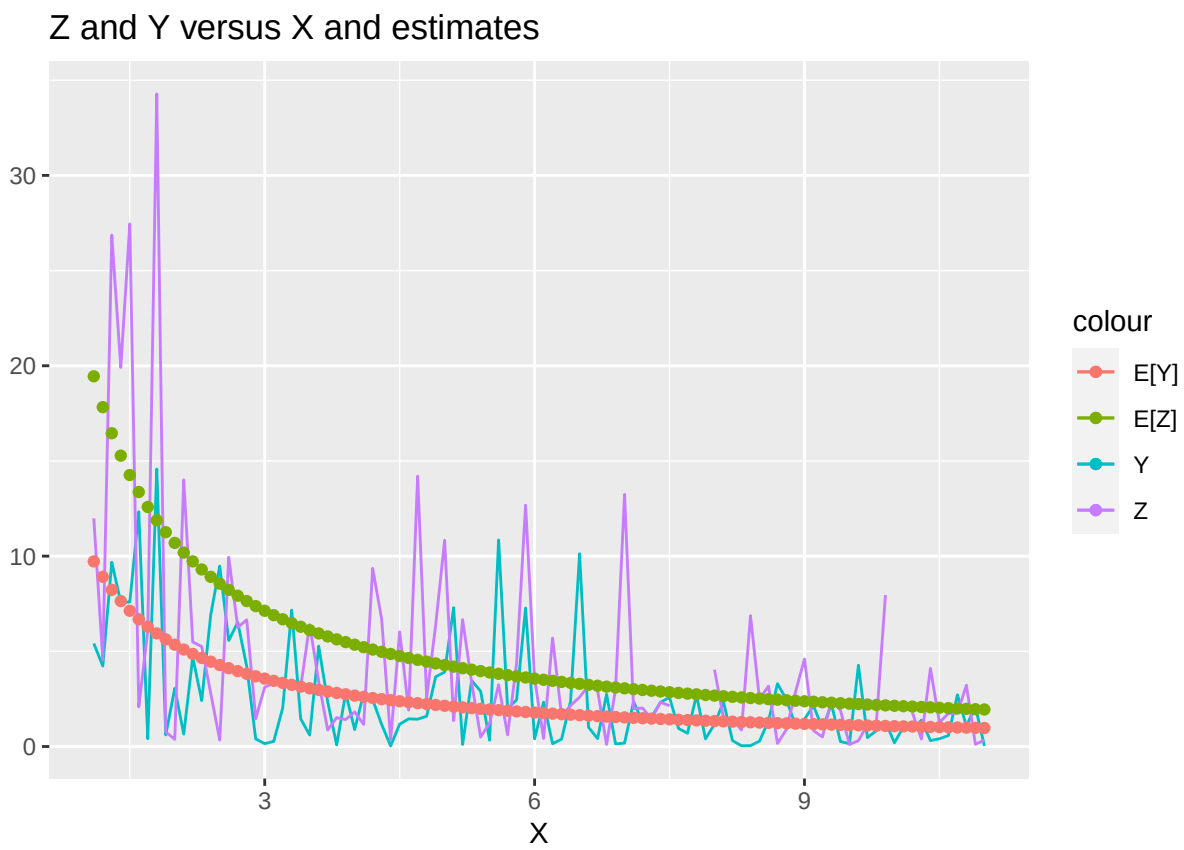
```
res
```

```
## $steps
## [1] 6
##
## $lambda
## [1] 10.69566
```

the optimal λ is 10.696 and it took 6 iterations for the algorithm to satisfy our convergence criterion.

4 Plot E[Y] and E[Z]

```
# Expectation value of random exponential variable is the inverse of its parameter
E_Y <- res[[2]]/data[,1]
E_Z <- (2*res[[2]])/data[,1]
df_plot <- data.frame(X = data[,1], Y = data[,2], Z = data[,3], E_Y = E_Y, E_Z = E_Z)


plot3 <- ggplot(df_plot)+
  geom_line(aes(x=X, y=Y, color = "Y"))+
  geom_line(aes(x=X, y=Z, color = "Z"))+
  geom_point(aes(x= X, y=E_Y, color = "E[Y]"))+
  geom_point(aes(x=X, y=E_Z, color = "E[Z]"))+
  labs(x="X", y="", title = "Z and Y versus X and estimates")

plot3
```



The expected value of our variables seem to follow the general trend of the noisy data.. The Y data has peaks

that are smaller than our Z data. The difference in expectation values between the two processes probably takes those differences into account.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
f_x <- function(x) {
  return((x^2 / exp(x)) - 2 * exp((-(9 * sin(x)) / (x^2 + x + 1))))
}
crossover <- function(x, y) {
  return((x + y) / 2)
}
mutate <- function(x) {
  return((x^2 %% 30))
}
library(ggplot2)

genetic <- function(maxiter, mutprob) {
  # a) plot function f in range [0,30]
  my_plot = ggplot() + stat_function(fun=f_x) + xlim(0,30) + xlab("x") + ylab("f(x)")
  my_plot
  # b) initial population for the genetic algorithm as X = (0, 5, 10, 15, . . . , 30)
  initial_pop = seq(0,30,5)
  population = initial_pop
  # c) compute vector Values that contain the function values for each in population
  function_values_pop = f_x(population)
  # d) iterate
  for (i in 1:maxiter) {
    # sample two indexes from population
    parents = sample(1:length(population), 2)
    # select index with smallest objective function as victim
    victim = order(function_values_pop)[1]
    # Parents are used to produce a new kid by crossover. Mutate this kid with probability mutprob (use
    new_kid = crossover(parents[1], parents[2])
    if (as.logical(rbinom(1, size=1, prob=mutprob))) {
      new_kid = mutate(new_kid)
    }
    # The victim is replaced by the kid in the population and the vector Values is updated.
    population[victim] = new_kid
    function_values_pop = f_x(population)
    # The current maximal value of the objective function is saved.
    max_val = max(function_values_pop)
  }
  # e) Add the final observations to the current plot in another colour.
  final_plot = my_plot + geom_point(aes(x=population, y=function_values_pop), color="red")
  final_plot
  return(list(max_val=max_val, initial_pop=initial_pop, final_pop=population, final_fun_values=function
}
max_iterations = c(10, 100)
mutation_probabilities = c(0.1, 0.5, 0.9)

set.seed(1234567890)
res1 = genetic(10, 0.1)
res2 = genetic(10, 0.5)
res3 = genetic(10, 0.9)
res4 = genetic(100, 0.1)
res5 = genetic(100, 0.5)
```

```
res6 = genetic(100, 0.9)
results = list(res1, res2, res3, res4, res5, res6)

library(gridExtra)
grid.arrange(res1$plot, res2$plot, res3$plot, res4$plot, res5$plot, res6$plot, ncol=3)

for (i in 1:length(results)) {
  cat(i, "\nmaxiter: ", results[[i]]$maxiter, "\nmutprob: ", results[[i]]$mutprob, "\ninitial_pop: ", r
}
data <- read.csv("physical1.csv")
library(ggplot2)

plot1 <- ggplot(data)+
  geom_line(aes(x=X, y=Y, color = "Y"))+
  geom_line(aes(x=X, y=Z, color = "Z"))+
  labs(x="X", y="", title = "Dependence of Z and Y versus X")

plot2 <- ggplot(data, aes(x=Y, y=Z))+
  geom_point()+
  labs(title="Z versus Y")

plot1
plot2

floglik <- function(data, lambda, lambda_prev){
  X <- data[,1]
  Y <- data[,2]
  Z <- data[,3]
  # missing and not missing data
  index <- is.na(Z)
  X_missing <- X[index]
  Z_not <- Z[!index]
  X_not <- X[!index]
  # n and m
  n <- length(X)
  m <- length(Z_not)
  # log likelihood
  Y_term <- (- n * log(lambda)) + sum(log(X)) - sum(X*Y/lambda)
  Z_not_term <- (- m * log(2*lambda)) + sum(log(X_not)) - sum(X_not*Z_not/(2*lambda))
  Z_missing_term <- (-(n-m)*log(2*lambda)) + sum(log(X_missing)) - ((n-m)*lambda_prev/lambda)

  log_like <- Y_term + Z_not_term + Z_missing_term
  return(log_like)
}

EM_algo <- function(data, eps, k_max, lambda_0){
  X <- data[,1]
  Y <- data[,2]
  Z <- data[,3]
  # missing and not missing data
  index <- is.na(Z)
  X_missing <- X[index]
  Z_not <- Z[!index]
```

```
  X_not <- X[!index]
  # n and m
  n <- length(X)
  m <- length(Z_not)

  #initialize everything
  k <- 1
  lambda_current <- lambda_0
  lambda_prev <- lambda_0+1+100*eps
  llvalcurr <- floglik(data, lambda_current, lambda_prev)

  # EM algorithm
  while( (abs(lambda_prev-lambda_current) > eps) &&  (k < (k_max+1)) ){
    lambda_prev <- lambda_current

    # EM step
    lambda_current <- (sum(X*Y) + sum(X_not*Z_not/2) + (n-m)*lambda_current)/(2*n)
    # new log-likelihood
    llvalcurr <- floglik(data, lambda_current, lambda_prev)
    k <- k+1
  }
  results <- list(steps = k, lambda = lambda_current)
  return(results)
}

epsilon <- 0.001
my_lambda_0 <- 100
my_k_max <- 100000

res <- EM_algo(data, eps = epsilon, k_max = my_k_max, lambda_0 = my_lambda_0)
res
# Expectation value of random exponential variable is the inverse of its parameter
E_Y <- res[[2]]/data[,1]
E_Z <- (2*res[[2]])/data[,1]
df_plot <- data.frame(X = data[,1], Y = data[,2], Z = data[,3], E_Y = E_Y, E_Z = E_Z)


plot3 <- ggplot(df_plot)+
  geom_line(aes(x=X, y=Y, color = "Y"))+
  geom_line(aes(x=X, y=Z, color = "Z"))+
  geom_point(aes(x= X, y=E_Y, color = "E[Y]"))+
  geom_point(aes(x=X, y=E_Z, color = "E[Z]"))+
  labs(x="X", y="", title = "Z and Y versus X and estimates")

plot3
```