Computational statistics - Lab 6

Alexander Karlsson (aleka769)
Lennart Schilling (lensc874)
Group 11
2019-03-08

Contents

Collaborations

The second assignment was done in collaboration with group07 (Gustav Lundberg and Faton Rekathati). We discussed the solutions in general and shared equations, but the interpretations and coding are done individually!

Used packages

```
library(ggplot2)      # Nice plotting
library(gridExtra)    # Side-by-side plots
library(dplyr)        # Pipe-programming
library(tidyr)        # Long format data.frames
theme_set(theme_bw())
```

Question 1

1.1 Objective function

In this assignment we are asked to perform a geretic algorithm on an objective function (below). It is of interest to maximize this function by selecting x-values that suits it better (avoids the minimum).

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

1.2 Crossover function

The crossover function can be seen as a genetic code passing from two parents. This might or might not be realistic, but it makes at least some form of sense that halv of some arbirtary genetic code are passed equally from both parents.

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

1.3 Mutate function

Sometimes, something weird happens in the genetic code. This might be due to mutations. Below is a function that transforms the values from the parents into something different (however much this transformation makes sense...).

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

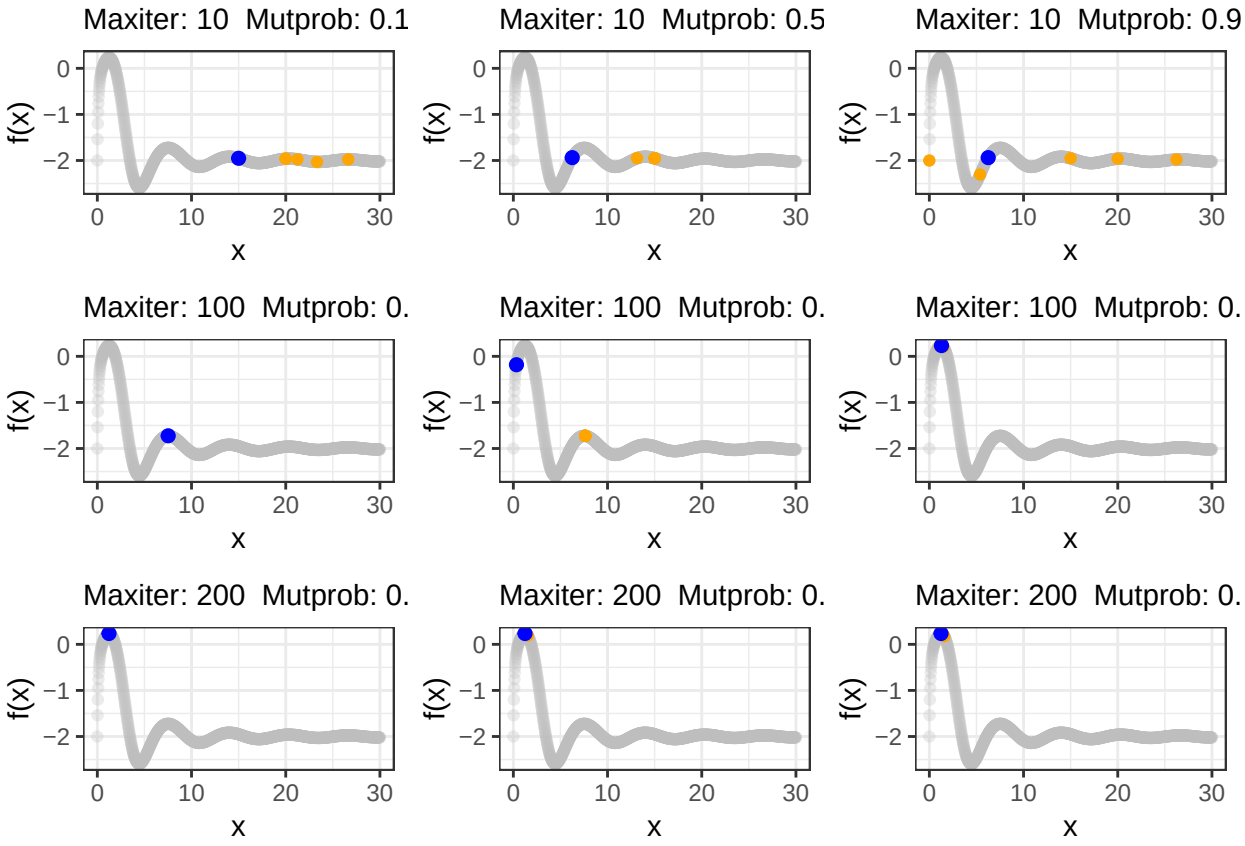
1.4 Genetic algorithm

```
genAlgorithm = function(maxiter, mutprob){
  # (a):
  x = seq(0, 30, length.out = 1000)
  p = ggplot() + geom_point(aes(x, f(x)), color = "gray", alpha = .3)

  # (b):
  X = seq(0, 30, by = 5) -> InitX

  # (c):
  Values = f(X)
  ObjFun = c()

  # (d):
  for(i in 1:maxiter){
    Parents     = sample(1:7, 2)                          # (i)

    #Victims     = order(Values)
    VictimIndex = which.min(Values)                       # (ii) kind of...
```

```
    Victim      = Values[VictimIndex]                       # (ii)

    ParOffspring = crossover(X[Parents[1]], X[Parents[2]]) # (iii)

    if (sample(x = c(T, F), size = 1, prob = c(mutprob, 1-mutprob))){
      ParOffspring = mutate(ParOffspring)                   # (iii)
    }

    X[VictimIndex]      = ParOffspring                      # (iv)
    Values[VictimIndex] = f(ParOffspring)                   # (iv)

    ObjFun[i] = max(Values)                                 # (v)
  }

  p2 = p + geom_point(aes(X, Values), color="orange", size = 1.5) +
    geom_point(aes(X[which.max(Values)], max(Values)), color = "blue", size = 2)

  RetList = list("Plot"   = p2,
                 "Pop"    = data.frame("InitPop"  = InitX,
                                       "FinalPop" = X),
                 "ObjFun" = ObjFun)
  return(RetList)
}
```

The complete (pointwise) objective function and the simulated population X are plotted below for different values of iterations and mutation probabilities.

```
iters = c(10,10,10,100,100,100,200,200,200)
probs = c(.1, .5, .9, .1, .5, .9, .1, .5, .9)

set.seed(123456)
PlotList = lapply(1:length(probs), function(i){
  genAlgorithm(maxiter = iters[i],
               mutprob = probs[i])$Plot +
    labs(subtitle = paste0("Maxiter: ", iters[i], "  ",
                           "Mutprob: ", probs[i]))
})

do.call(grid.arrange, PlotList)
```

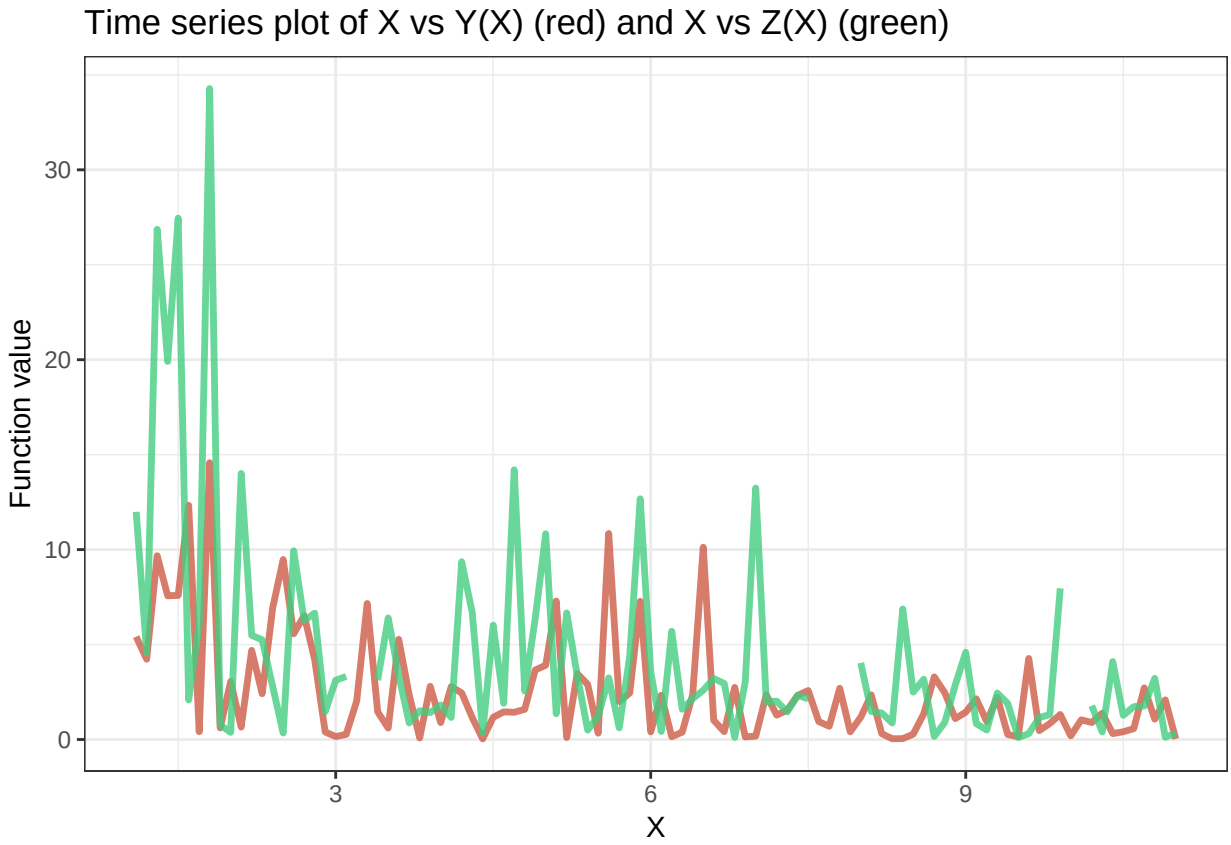


The appearance of the above graphs indicates that an increase in both `maxiter` and `mutprob` tends to shift the population towards the local (probably global) maximum at x ≈ 1. The objective function has a clear global maximum that is much higher than any other values, this helps speed up the initial populations transformation.

Question 2

2.1 Time series plot

```
phys = read.csv("physical1.csv")

ggplot(phys, aes(x = X)) +
  geom_line(aes(y = Y), size = 1.2, alpha = 0.8, color = "coral3") +
  geom_line(aes(y = Z), size = 1.2, alpha = 0.8, color = "seagreen3") +
  labs(title = "Time series plot of X vs Y(X) (red) and X vs Z(X) (green)",
       y = "Function value")
```



We see that the two processes have a similar decreasing pattern with increased valuies of X and also include a lot of noise. The noise makes it harder to get a general idea of the distributions. We also observe that there are missing values for Z(X).

2.2 Derive EM algorithm

We assume that these two chemical processes are dependent on X and independent of each other. The distributional assumptions are stated below:

$$Y_i \sim exp(X_i/\lambda) \quad , \quad Z_i \sim exp(X_i/2\lambda),$$

where the pdf for an exponential distribution is defined as:

$$f(u|\theta) = \begin{cases} \theta e^{-\theta u} & , \quad u \geq 0 \\ 0 & , \quad u < 0 \end{cases}$$

The expected value for an exponential distribution is $\frac{1}{\theta}$. For our variables Y_i and Z_i, we obtain the following:

$$\begin{aligned} Y_i \sim exp\left(\frac{X_i}{\lambda}\right) \quad &\Rightarrow \quad f_1(Y_i|X_i,\lambda) = \frac{X_i}{\lambda} e^{-\frac{X_i}{\lambda}Y_i} \quad \Rightarrow \quad E[Y_i] = \frac{\lambda}{X_i} \\ Z_i \sim exp\left(\frac{X_i}{2\lambda}\right) \quad &\Rightarrow \quad f_2(Z_i|X_i,\lambda) = \frac{X_i}{2\lambda} e^{-\frac{X_i}{\lambda}Z_i} \quad \Rightarrow \quad E[Z_i] = \frac{2\lambda}{X_i} \end{aligned}$$

Since these expected values will be used later, we define the functions below...

```
expY = function(lambda, x) lambda / x
expZ = function(lambda, x) (2*lambda) / x
```

Asuming that the two variables are in fact independent, their joint probability function can be obtained by multiplying the individual ones, giving:

$$\begin{aligned} g(Y_i, Z_i|X_i,\lambda) =& f_1(Y_i|X_i,\lambda) \cdot f_2(Z_i|X_i,\lambda) = \\ =& \frac{X_i}{\lambda} \cdot \frac{X_i}{2\lambda} \cdot e^{-\frac{X_i}{\lambda}y} \cdot e^{-\frac{X_i}{2\lambda}z} = \\ =& \left(\frac{X_i}{\sqrt{2}\lambda}\right)^2 \cdot e^{-\frac{X_i}{\lambda}(y+\frac{z}{2})} \end{aligned}$$

We continue with the product of all the observations, i.e. the likelihood:

$$\mathcal{L}(\lambda; X, Y, Z) = \prod_{i=1}^{n} f(Y_i, Z_i|X_i,\lambda) = \left(\sqrt{2}\lambda\right)^{-2n} \prod_{i=1}^{n} X_i^2 \cdot e^{-\frac{1}{\lambda}\sum_{i=1}^{n} X_i(Y_i + \frac{Z_i}{2})}$$

As some of the Z's are missing, these will be imputed with their expected values. However, since the true value of λ is unkown, the current esitmate of it is used giving:

$$\hat{E[Z_i]} = \frac{2\hat{\lambda}}{X_i},$$

The observations can be partitioned into two sets, denoted O for those with observed values of Z and U for those with unobserved values of Z. If n is used to denote the total number of observations, $n = |O| + |U|$ and r is used to denote the number of missing Z, $r = |U|$.

Notably, when we later take the differentiate w.r.t. λ in the log-likelihood, the $\hat{\lambda}$ is treated as a constant and thus enabling us to calculate λ for the next iteration.

$$\mathcal{L}(\lambda; X, Y, Z) = \left(\sqrt{2}\lambda\right)^{-2n} \prod_{i=1}^{n} X_i^2 \cdot e^{-\frac{1}{\lambda}\left[\sum_{i=1}^{n} X_i \cdot Y_i + \frac{1}{2}\sum_{i \in O} X_i \cdot Z_i + \sum_{i \in U} \frac{2\hat{\lambda}}{2X_i} X_i\right]}$$

The logarithm of the likelihood, $\ln\mathcal{L}(\lambda; X, Y, Z)$ can be shortened by observing that the last part of the exponent above cancels out wtih the exception of $\hat{\lambda}$.

$$\ln\mathcal{L}(\lambda) = \ln\left[\mathcal{L}(\lambda; X, Y, Z)\right] = 2\left(\sum_{i=1}^{n}\ln(X_i) - n\cdot\ln(\sqrt{2}\lambda)\right) - \frac{1}{\lambda}\left(\sum_{i=1}^{n}X_i\cdot Y_i + \frac{1}{2}\sum_{i\in O}X_i\cdot Z_i + (n-r)\hat{\lambda}\right)$$

The R implementation of the above expression can be seen below:

```
Expectation = function(d, lambda, Obs, n, r){
  first_expr  = 2*(sum(log(d$X)) - n*log(sqrt(2)*lambda))
  second_expr = (-1/lambda) * (sum(d$X*d$Y) + (1/2) * sum(d$X[Obs] * d$Z[Obs]) + (n - r) * lambda)

  # Complete expression:
  return(first_expr + second_expr)
}
```

The maximization is then done by setting the derivative of the function to 0 and solving for λ. Since the log-likelihood is a concave function, the result will be a maximum.

$$\begin{aligned}\frac{\partial\ln\mathcal{L}(\lambda)}{\partial\lambda} = 0 \Rightarrow \\ \frac{2n}{\lambda} = \frac{1}{\lambda^2}\left[\sum_{i=1}^{n}X_i\cdot Y_i + \frac{1}{2}\sum_{i\in O}X_i\cdot Z_i + (n-r)\hat{\lambda}\right] \Rightarrow \\ \lambda = \frac{\sum_{i=1}^{n}X_i\cdot Y_i + \frac{1}{2}\sum_{i\in O}X_i\cdot Z_i + (n-r)\hat{\lambda}}{2n}\end{aligned}$$

We define the above calculations as a function below:

```
Maximization = function(d, lambda, Obs, n, r){
  return( ((sum(d$X * d$Y) + sum(d$X[Obs] + d$Z[Obs]) + (n-r) * lambda)) / (2*n) )
}
```

2.3 Implementation in R

With the E-step and M-step defined, we can start implementing the complete algorithm.

```
emAlgorithm = function(data, Lambda, tol, maxIter){
  # Setup:
  n       = nrow(data)
  Obs     = which(!is.na(data$Z))
  UnObs   = setdiff(1:n, Obs)
  r       = n - length(UnObs)
  i       = 2

  LogLikelihood = c(NA, Expectation(data, Lambda, Obs, n, r))
  Lambda[2]     = Maximization(data, Lambda, Obs, n, r)

  while(abs(Lambda[i-1] - Lambda[i]) > tol & i <= maxIter){
    LogLikelihood[i+1] = Expectation(data, Lambda[i], Obs, n, r)
    Lambda[i+1]        = Maximization(data, Lambda[i], Obs, n, r)
    i             = i + 1
  }

  data.frame("LogLikelihood"  = LogLikelihood,
```

```
              "LambdaEstimate" = Lambda)
}
```

The results of the algorithm, with λ_0 initialized as 100, a tolerance of 0.001 and iterations upper bound of 1000 iterations can be seen below.

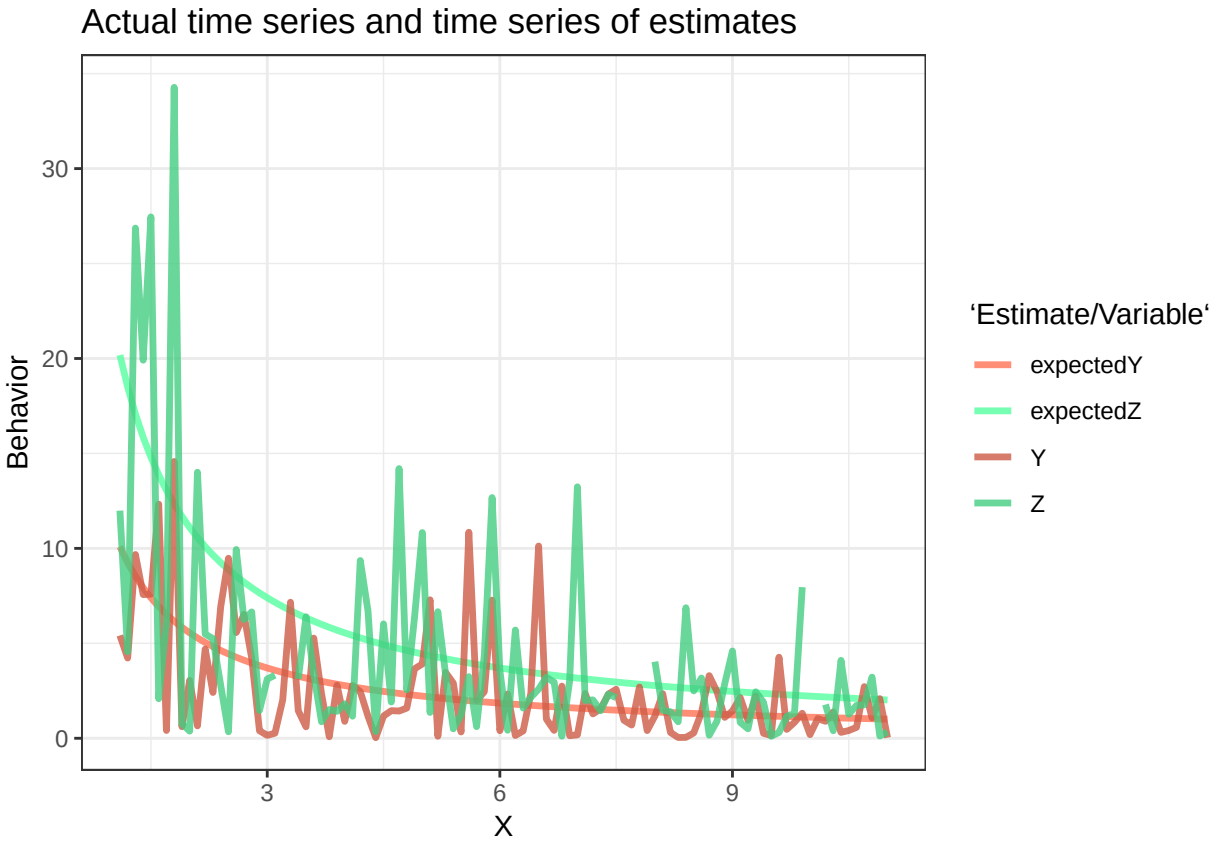
```
emOutput = emAlgorithm(data = phys, Lambda = 100, tol = 0.001, maxIter = 1000)
knitr::kable(emOutput[-1,], row.names = FALSE)
```

LogLikelihood	LambdaEstimate
-688.9647	14.65549
-424.4805	11.24171
-413.9948	11.10516
-413.7968	11.09970
-413.7894	11.09948

The algorithm converges quickly with only 5 iterations (omitting the initial value λ_0). Whether or not this is a reasonable result we will see in the next sub-assignment.

2.4 Comparison of dependent variables and estimates

```
optLambda = emOutput$LambdaEstimate[nrow(emOutput)]
X_        = phys$X
phys %>% dplyr::mutate("expectedY" = expY(optLambda, X_),
                       "expectedZ" = expZ(optLambda, X_)) %>%
  gather(., key = 'Estimate/Variable', 'Behavior', Y:expectedZ)  %>%
  ggplot(., aes(X, Behavior, color = `Estimate/Variable`)) +
  geom_line(size = 1.2, alpha = .8) +
  ggtitle("Actual time series and time series of estimates") +
  scale_color_manual(values = c("coral1", "seagreen1", "coral3", "seagreen3"))
```



The estimates for Y_i and Z_i seem to follow the general trend in the data fairly well, i.e. the results from the obtained λ are reasonable. The actual time series of the response variables have high values early and in the middle of the series that no exponential distribution can follow.

Appendix

All code is shown in the document