Laboratoria 7 i 8. Algorytm Gaussa-Newtona

Paweł Matławski

5 05 2021

Zadanie nr 1

Wygenerujmy n obserwacji $Y_1, ..., Y_n$ postaci: $Y_i = \beta_1 + \beta_2 e^{-\beta_3 x_i} + \epsilon_i$, gdzie ϵ_i $i.i.d$ $N(0, \sigma^2)$

```
library(matlib)
library(MASS)
library(pracma)
```

```
##
## Attaching package: 'pracma'

## The following objects are masked from 'package:matlib':
##
##     angle, inv
```

```
library(xtable)
library(knitr)
set.seed(420)
sigma_2 = 0.5
x_i <- seq(0.1,1000,by = 0.1)
n = length(x_i)
beta <- c(80, 100, 0.005)
eps <- rnorm(n, mean = 0, sd = sqrt(sigma_2))
Y <- beta[1] + beta[2]*(exp(-beta[3]*x_i)) + eps
```

Zadanie nr 2

Funkcja g ma postać $g(\boldsymbol{x}, \boldsymbol{\beta}) = \beta_1 + \beta_2 e^{-\beta_3 x_i}$, a jej gradient $\nabla(\boldsymbol{g}, \boldsymbol{\beta}) = [1, e^{-\beta_3 x_i}, -\beta_2 e^{-\beta_3 x_i}]$.

```
g_func <- function(x, beta){
  result = beta[1]+beta[2]*exp(-beta[3]*x)
  return(result)
}

gradient <- function(x, i, beta)
{
  return(c(1,exp(-beta[3]*x[i]),-beta[2]*x[i]*exp(-beta[3]*x[i])))
}
```

Zadanie nr 3 i nr 4

Za pomocą algorytmu Gaussa-Newtona wyznaczmy estymatory $(\hat{\beta_1}, \hat{\beta_2}, \hat{\beta_3}, \hat{\sigma^2})$ parametrów $(\beta_1, \beta_2, \beta_3, \sigma^2)$.

```r
macierz_zer <- matrix(rep(0, 3*n), ncol = 3, nrow = n)

g_grad <- function(x,beta)
{
  for(i in 1:n){
    macierz_zer[i,]<-gradient(x,i,beta)
  }
  return(macierz_zer)
}

loglikelihood<-function(beta, g, sigma){
  return(-n/2*log(2*pi)-n/2*log(sigma)-1/(2*sigma)*sum((Y-g)^2))
}

gauss_newton <- function(n, x,  beta_0){
  e<-Y-g_func(x, beta_0)
  e<-matrix(e, nrow = n, ncol = 1)
  zera<-matrix(rep(0, 3*100), ncol = 3, nrow = 1000)
  sigma_k<-matrix(rep(0, 100), ncol = 1, nrow = 1000)
  loglik_k<-matrix(rep(0, 100), ncol = 1, nrow = 1000)
  k<-1
  beta_0<-matrix(beta_0,nrow=3,ncol=1)
  sigma_k[k]<- sum((Y-g_func(x_i, beta_0))^2)/(n-3)
  loglik_k[k]<- loglikelihood(beta_1, g_func(x_i, beta_0), sigma_k[k])
  zera[k,]<-beta_0
  repeat{
    beta_1<-beta_0+inv(t(g_grad(x,beta_0))%*%g_grad(x,beta_0))%*%t(g_grad(x, beta_0))%*%e
    if(norm(beta_1-beta_0,type = "2")<=0.0001 || k>=1000) break;
    beta_0<-beta_1
    e<-Y-g_func(x, beta_0)
    k<-k+1
    zera[k,]<-beta_0
    sigma_k[k,] <- sum((Y-g_func(x_i, beta_0))^2)/(n-3)
    loglik_k[k,] <- loglikelihood(beta_1, g_func(x_i, beta_0), sigma_k[k])
  }
  wynik <- cbind.data.frame(zera[0:k,],sigma_k[0:k,],loglik_k[0:k,])
  colnames(wynik) <- c("beta_1","beta_2","beta_3", "sigma^2", "loglik")
  return(xtable(wynik, digits = 5))
}

gauss_newton(n, x_i, c(79, 101, 0.004))
```

% latex table generated in R 4.0.5 by xtable 1.8-4 package % Thu May 06 02:44:06 2021

	beta_1	beta_2	beta_3	sigma^2	loglik
1	79.00000	101.00000	0.00400	22.03910	-29651.97578
2	80.80800	98.61563	0.00490	1.45221	-16053.32609
3	80.01228	99.98250	0.00500	0.49031	-10624.25992
4	80.00635	99.99374	0.00500	0.49029	-10624.10446

W przedstawionej powyższej tabelce możemy zauważyć, że algorytm wykonał trzy iteracje.

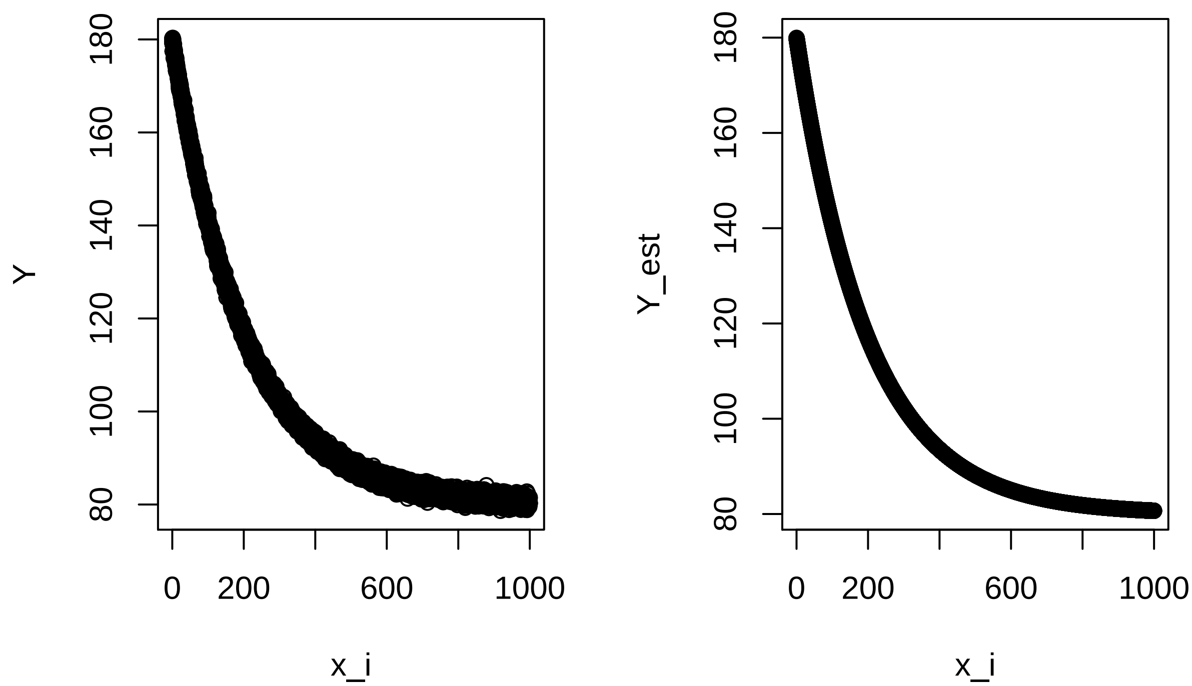
Zadanie nr 5

Patrząc na końcowe wyniki parametrów β z tabelki i porównując je z rzeczywistymi $(\beta_1, \beta_2, \beta_3) = (80, 100, 0.005)$ widzimy, że algorytm sprawdził się dobrze w krótkim "czasie" (3 kroki).

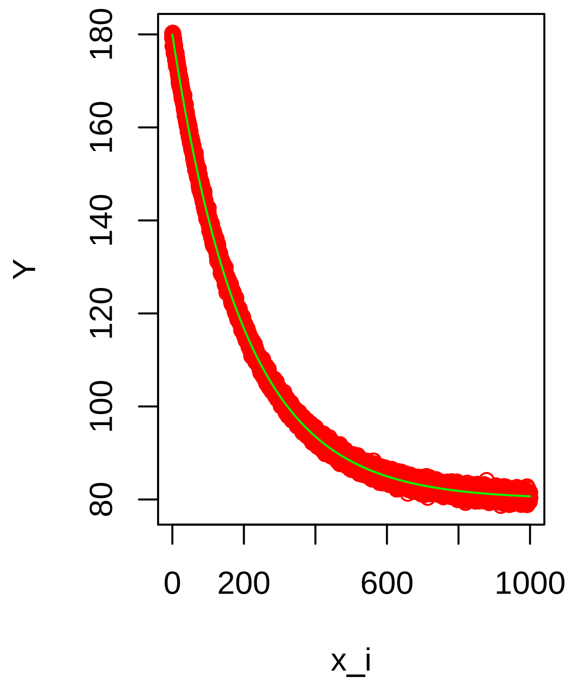
Zadanie nr 6

Pokażmy teraz jak wygląda wykres wartości rzeczywistych i dla porównania wykres wartości estymowanych, przy oszacowanych parametrach.

```
par(mfrow=c(1,2))
Y_est = 80.00635 + 99.99374*exp(-0.00500*x_i)
plot(x_i, Y)
plot(x_i, Y_est)
```




```
plot(x_i, Y, col = 'red')
lines(x_i, Y_est, col = 'green')
```



Widzimy, że wykresy niemal są identyczne. W estymowanych wartościach widać brak błędów losowych ϵ.

Zadanie nr 7

Oszacujmy teraz macierz kowariancji dla $\hat{\beta}$. Skorzystamy ze wzoru z wykładu $Cov(\hat{\beta}) = \sigma^2(\boldsymbol{G^T G})^{-1}$

```
covmat_grad <- g_grad(x_i, c(80.00635,99.99374, 0.00500))
covmat <- sigma_2 * inv(t(covmat_grad)%*%covmat_grad)
xtable(covmat, digits=8)
```

% latex table generated in R 4.0.5 by xtable 1.8-4 package % Thu May 06 02:44:07 2021

	1	2	3
1	0.00019191	-0.00002406	0.00000004
2	-0.00002406	0.00100589	0.00000005
3	0.00000004	0.00000005	0.00000000

Widzimy, że estymowana macierz kowariancji przyjmuje bardzo małe wartości

Zadanie nr 8

W zadaniu 5 stwierdziliśmy, że algorytm sprawdził się dobrze. Mieliśmy jednak punkt bardzo bliski wartości rzeczywistym. Sprawdźmy jeszcze jeden podobny punkt, a następnie punkt bardziej oddalony.

```
gauss_newton <- function(n, x,  beta_0){
  e<-Y-g_func(x, beta_0)
  e<-matrix(e, nrow = n, ncol = 1)
```

```
  zera<-matrix(rep(0, 3*100), ncol = 3, nrow = 1000)
  sigma_k<-matrix(rep(0, 100), ncol = 1, nrow = 1000)
  loglik_k<-matrix(rep(0, 100), ncol = 1, nrow = 1000)
  k<-1
  beta_0<-matrix(beta_0,nrow=3,ncol=1)
  sigma_k[k]<- sum((Y-g_func(x_i, beta_0))^2)/(n-3)
  loglik_k[k]<- loglikelihood(beta_1, g_func(x_i, beta_0), sigma_k[k])
  zera[k,]<-beta_0
  repeat{
    beta_1<-beta_0+ginv(t(g_grad(x,beta_0))%*%g_grad(x,beta_0))%*%t(g_grad(x, beta_0))%*%e
    if(norm(beta_1-beta_0,type = "2")<=0.0001 || k>=1000) break;
    beta_0<-beta_1
    e<-Y-g_func(x, beta_0)
    k<-k+1
    zera[k,]<-beta_0
    sigma_k[k,] <- sum((Y-g_func(x_i, beta_0))^2)/(n-3)
    loglik_k[k,] <- loglikelihood(beta_1, g_func(x_i, beta_0), sigma_k[k])
  }
  wynik <- cbind.data.frame(zera[0:k,],sigma_k[0:k,],loglik_k[0:k,])
  colnames(wynik) <- c("beta_1","beta_2","beta_3", "sigma^2", "loglik")
  return(xtable(wynik, digits = 5))
}


gauss_newton(n, x_i, c(82, 98, 0.001))
```

% latex table generated in R 4.0.5 by xtable 1.8-4 package % Thu May 06 02:44:07 2021

	beta_1	beta_2	beta_3	sigma^2	loglik
1	82.00000	98.00000	0.00100	2066.03658	-52354.82210
2	82.00000	98.00000	0.00256	339.76111	-43329.09905
3	82.00000	98.00000	0.00417	31.64228	-31460.35680
4	82.00000	98.00000	0.00514	2.15067	-18016.78634
5	80.06441	97.95306	0.00490	0.70144	-12414.77688

```
gauss_newton(n, x_i, c(200, 200, 100))
```

% latex table generated in R 4.0.5 by xtable 1.8-4 package % Thu May 06 02:44:07 2021

	beta_1	beta_2	beta_3	sigma^2	loglik
1	200.00000	200.00000	100.00000	10634.47002	-60547.16480
2	99.87124	200.00000	100.00001	605.69251	-46219.74756

Powtórzyłem implementację algorytmu zmieniając odwracanie macierzy z funkcji inv and ginv (generalized inverse), ponieważ algorytm miał brakujące dane w warunkach, co skutkowało brakiem rezultatu. Z jakiegoś powodu jednak, algorytm dalej nie radzi sobie dla oddalonych wartości, przestając dokonywać iteracji.