ML: Assignment 1

January 14, 2024

Time used for reading: 5
Time used for the basic assignment: 7
Time used for extra assignment (VG): 0

Good with lab:
One really good thing with the lab was that we got to first derive the gradient and the implement the function which made me understand gradient more profound.

Things to improve in the lab:
However I was pretty difficult to understand some of the instructions for section 1 since we first derived the gradient of the likelihood function and then was not supposed to use it when implementing the functions in R.

Anything that was difficult with the material?
At the start, understanding the gradient was difficult but after reading some it became clear.

Task 1: Basic, Stochastic, and Mini-Batch Gradient Descent

```
library(uuml)
data("binary")
binary$gre_sd <- (binary$gre - mean(binary$gre))/sd(binary$gre)
binary$gpa_sd <- (binary$gpa - mean(binary$gpa))/sd(binary$gpa)
X <- model.matrix(admit ~ gre_sd + gpa_sd, binary)
y <- binary$admit
```

1.1: Implement the gradient for logistic regression

For this subsection we will implement the gradient for logistic regression and we have been given the likelihood function for the logistic regression which looks as follows:

$$L(\theta, \mathbf{y}, \mathbf{X}) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i}$$

Further, the formula for the logit link is as follows:

$$logit(p_i) = log(\frac{p_i}{1-p_i}) = \boldsymbol{x}_i\theta$$

where $\boldsymbol{x}_i$ denotes the i:th row of the design matrix $\boldsymbol{X}$ and $\theta : 1 \times P$ is a parameter vector.

1.1.1

For this subsection we will derive the gradient for the negative log likelihood with respect to $\theta : p \times 1$ where we have that $\text{NLL}(\theta, \mathbf{y}, \mathbf{X}) = -l(\theta, \mathbf{y}, \mathbf{X})$.

$$\text{NLL}(\theta, \mathbf{y}, \mathbf{X}) = -l(\theta, \mathbf{y}, \mathbf{X}) = -\sum_{i=1}^{n} y_i \mathbf{x}_i \theta + \log\left(1 + \exp\left(\mathbf{x}_i \theta\right)\right)$$

Now when the negative log likelihood NLL has been defined, we will compute the gradient with respect to parameter vector, i.e $\boldsymbol{\theta} : px1$ according to below:

$$\frac{\partial NLL(\theta, \boldsymbol{y}, \boldsymbol{X})}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{n} y_i \boldsymbol{x}_i + \frac{1}{1 + exp(\boldsymbol{x}_i \theta)} exp(\boldsymbol{x}_i \theta) \boldsymbol{x}_i = -\sum_{i=1}^{n} \left(y_i + \frac{exp(\boldsymbol{x}_i \theta)}{1 + exp(\boldsymbol{x}_i \theta)} \right) \boldsymbol{x}_i$$

This is the gradient for the negative log likelihood with respect to the θ.

1.1.2

Now we will implement the gradient as a function in R where the function return the gradient of $\frac{1}{n} l\left(\theta, \boldsymbol{y}, \boldsymbol{X},\right)$. Further, we will present the output of the function when using $\theta = (0, 0, 0)'$ and $\theta = (-1, 0.5, 0.5)'$.

```
ll_grad <- function(y, X, theta){
  n <- length(y)
  gradient <- t(y - exp(X%*% theta )/(1+exp(X%*% theta)))%*%X/n
  return(gradient)
}
ll_grad(y, X, theta = c(0,0,0))

##      (Intercept)     gre_sd     gpa_sd
## [1,]     -0.1825 0.08574746 0.08285471

ll_grad(y, X, theta = c(-1,0.5,0.5))

##      (Intercept)     gre_sd      gpa_sd
## [1,]  0.02174332 -0.0395161 -0.04264481
```

1.2: Implement Gradient Descent

1.2.1

Now we will run logistic regression in R to get an maximum likelihood estimate of θ using the glmfunction. Since the design matrix X already consists of an intercept, ie that the first column in X is a column of ones, we will state the formula argument in glm as $y\ -1 + X$.

```
# link function default set to logit link
# we are including -1 since the design matrix already include an intercept
glm_log_reg <- glm(formula = y  ~ -1 + X, family= binomial())
glm_log_reg$coefficients

## X(Intercept)      Xgre_sd      Xgpa_sd
##   -0.8097503    0.3108184    0.2872087
```

1.2.2a: ordinary gradient descent

For this subsection we will implement three gradient descent algorithms as three separate R functions. The first gradient descent algorithm that will be implemented is the ordinary gradient descent, the second is stochastic gradient descent and the third is minibatch gradient descent.

```r
# Example 1.2.2a
mbsgd_ord <- function(y, X, sample_size, eta, epochs) {

  results <- matrix(0, ncol = ncol(X) + 2L, nrow = epochs)
  colnames(results) <- c("epochs", "nll", colnames(X))

  theta <- rep(0.0, ncol(X)) # this is the parameter vector

  for ( j in 1:epochs) {

    gradient <- -ll_grad(y, X, theta)
    theta <- theta - eta * t(gradient)

    results[j, "epochs"] <- j
    results[j, "nll"] <- ll(y, X, theta)
    results[j, -(1:2)] <- theta
  }
  return(results)

}
```

1.2.2b: stochastic gradient descent

```r
# Example 1.2.2b
mbsgd_sto <- function(y, X, sample_size, eta, epochs){

  results <- matrix(0.0, ncol = ncol(X) + 2L, nrow = epochs)
  colnames(results) <- c("epoch", "nll", colnames(X))

  theta <- rep(0.0, ncol(X))

  for(j in 1:epochs){
    random <- sample(length(y))

    for (i in random) {

      theta_grad <- -ll_grad(y[i], X[i,], theta)
      theta <- theta - eta*t(theta_grad)

      results[j, "epoch"] <- j
      results[j, "nll"] <- ll(y, X, theta)
      results[j, -(1:2)] <- theta
    }
  }
  return(results)
}
```

1.2.2c: minibatch gradient descent

```
# Example 1.2.2c
mbsgd_mini <- function(y, X, sample_size, eta, epochs){
  results <- matrix(0.0, ncol = ncol(X) + 2L, nrow = epochs)
  colnames(results) <- c("epoch", "nll", colnames(X))

  theta <- rep(0, ncol(X))

  for(j in 1:epochs){

    num_batch <- length(y)/sample_size

    for (i in 1:num_batch) {

      start <-(i-1)* sample_size + 1
      end <- min(i* sample_size, length(y))
      x_batch <- X[start:end,]
      y_batch <- y[start:end]
      theta_grad <- -ll_grad(y_batch, x_batch, theta)
      theta <- theta - eta*t(theta_grad)

      results[j, "epoch"] <- j
      results[j, "nll"] <- ll(y, X, theta)
      results[j, -(1:2)] <- theta
    }
  }
return(results)
}
```

1.2.3

Now we will try the algorithm for different parameter values of η and run the algorithm for roughly 500 epochs. We will use three different η, one where the optimizer diverge, one η where the optimizer converge very slowly and one η where the optimizer converge quicker. The value of η will fixed for all 500 epochs. There will be one plot for the negative loglikelihood value for all observations for a given θ and the value of one θ parameter element where we include the true values obtained from the glmfunction as a horizontal line in the figure.

```
par(mfrow=c(3,2))
result1 <- mbsgd_ord(y, X, sample_size = 10, eta = 0.2, epochs = 500)
plot(result1[,2], type = "l", main = "eta = 0.2", ylab = "NLL", xlab = "epochs")
plot(result1[,4], type = "l", main = "eta = 0.2", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")

result2 <- mbsgd_ord(y, X, sample_size = 10, eta = 0.01, epochs = 500)
plot(result2[,2], type = "l", main = "eta = 0.01", ylab = "NLL", xlab = "epochs")
plot(result2[,4], type = "l", main = "eta = 0.01", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")

result3 <- mbsgd_ord(y, X, sample_size = 10, eta = 40, epochs = 500)
plot(result3[,2], type = "l", main = "eta = 40", ylab = "NLL", xlab = "epochs")
plot(result3[,4], type = "l", main = "eta = 40", ylab = "gre_sd", xlab = "epochs")
```

```r
abline(h = glm_log_reg$coefficients[2], col = "blue")
```

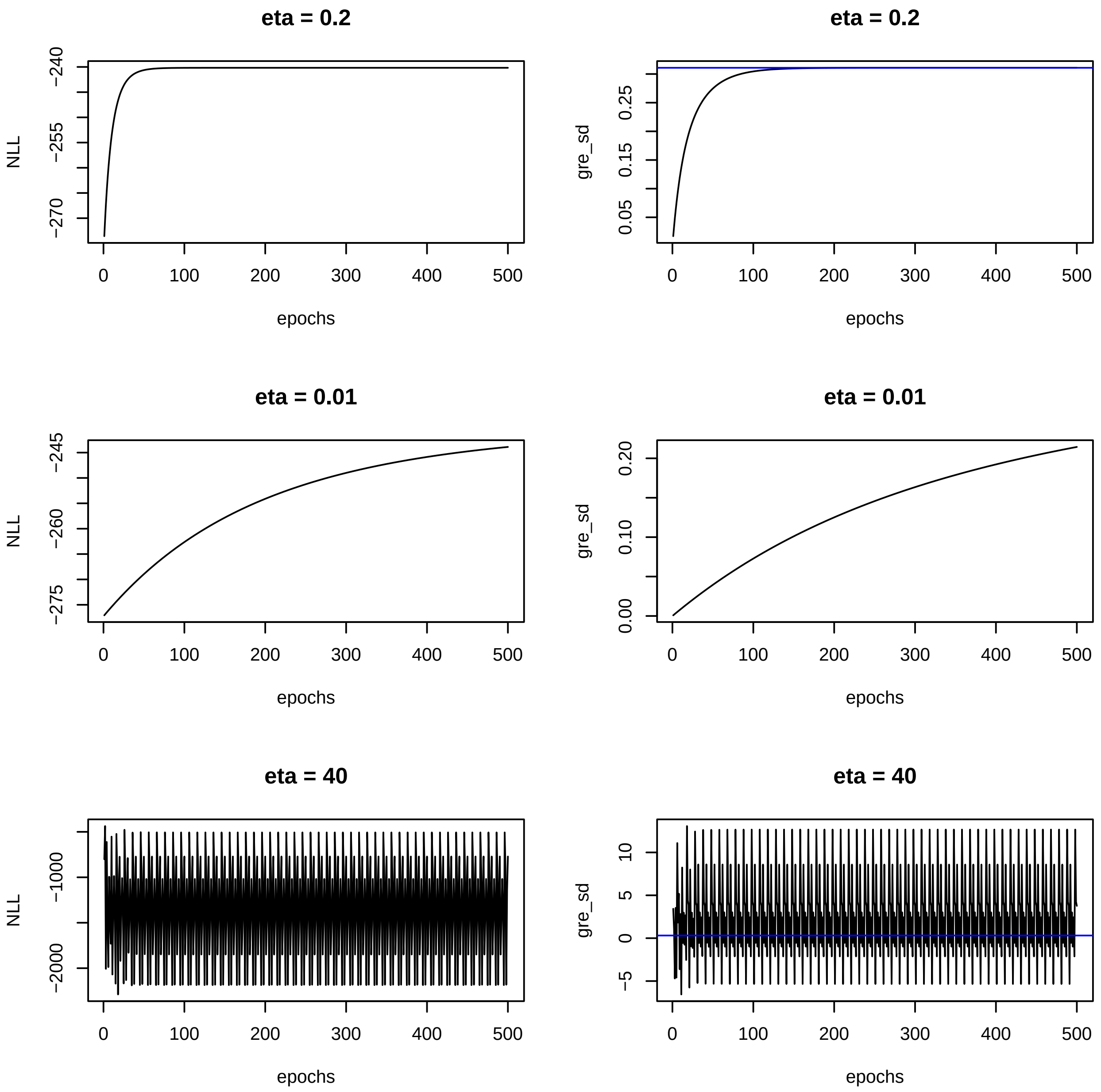

Figure 1: Plots for ordinary gradient decent

```r
par(mfrow=c(3,2))
result4 <- mbsgd_sto(y, X, sample_size = 10, eta = 0.002, epochs = 500)
plot(result4[,2], type = "l", main = "eta = 0.002", ylab = "NLL", xlab = "epochs")
plot(result4[,4], type = "l", main = "eta = 0.002", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")
```

```r
result5 <- mbsgd_sto(y, X, sample_size = 10, eta = 0.00001, epochs = 500)
plot(result5[,2], type = "l", main = "eta = 0.00001", ylab = "NLL", xlab = "epochs")
plot(result5[,4], type = "l", main = "eta = 0.00001", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")

result6 <- mbsgd_sto(y, X, sample_size = 10, eta = 0.2, epochs = 500)
plot(result6[,2], type = "l", main = "eta = 0.2", ylab = "NLL", xlab = "epochs")
plot(result6[,4], type = "l", main = "eta = 0.2", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")
```

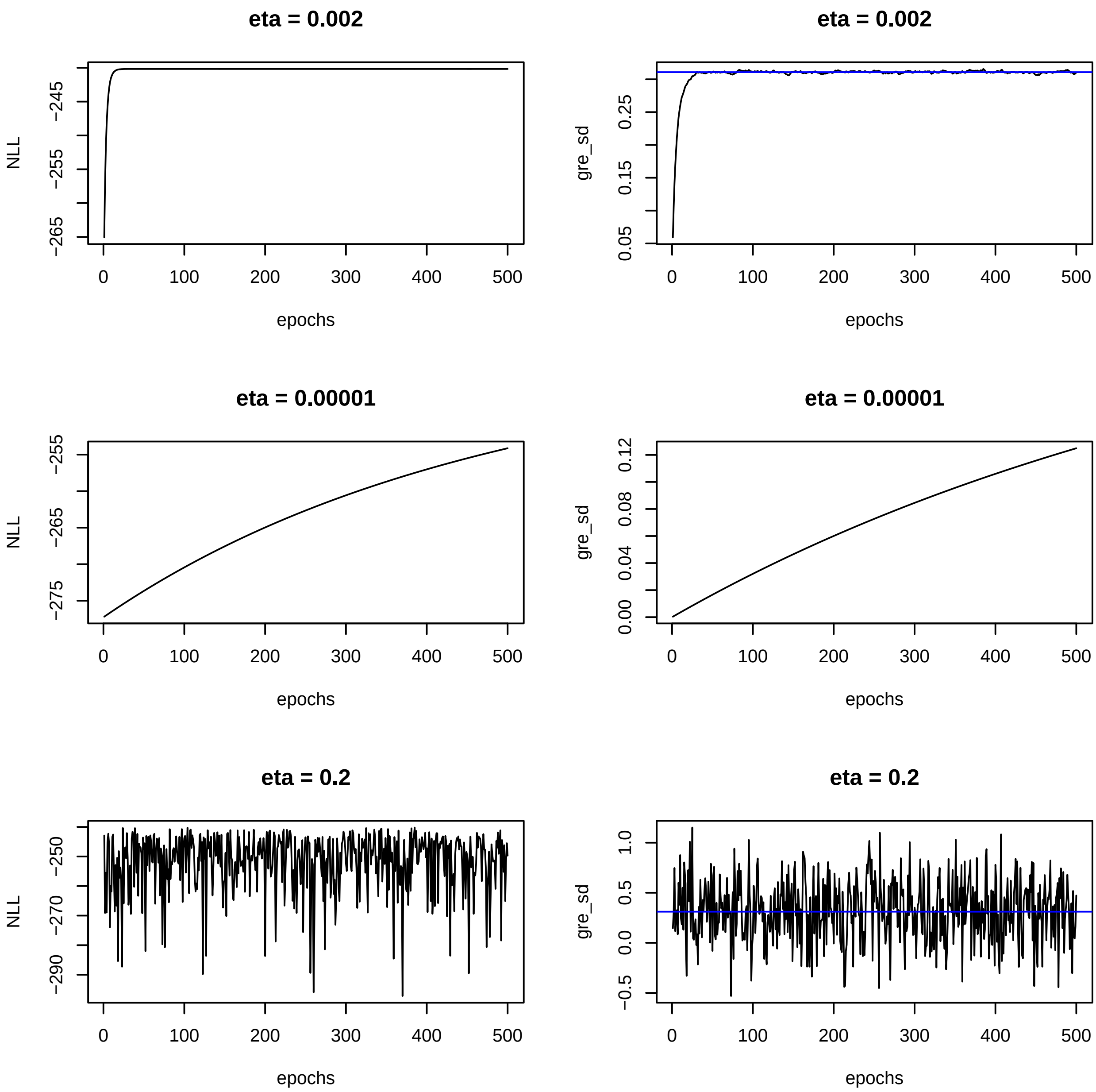

Figure 2: Plots for stochastic gradient decent

```
par(mfrow=c(3,2))
result7 <- mbsgd_mini(y, X, sample_size = 10, eta = 0.02, epochs = 500)
plot(result7[,2], type = "l", main = "eta = 0.02", ylab = "NLL", xlab = "epochs")
plot(result7[,4], type = "l", main = "eta = 0.02", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")

result8 <- mbsgd_mini(y, X, sample_size = 10, eta = 0.001, epochs = 500)
```

```r
plot(result8[,2], type = "l", main = "eta = 0.001", ylab = "NLL", xlab = "epochs")
plot(result8[,4], type = "l", main = "eta = 0.001", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")

result9 <- mbsgd_mini(y, X, sample_size = 10, eta = 50, epochs = 500)
plot(result9[,2], type = "l", main = "eta = 50", ylab = "NLL", xlab = "epochs")
plot(result9[,4], type = "l", main = "eta = 50", ylab = "gre_sd", xlab = "epochs")
abline(h = glm_log_reg$coefficients[2], col = "blue")
```

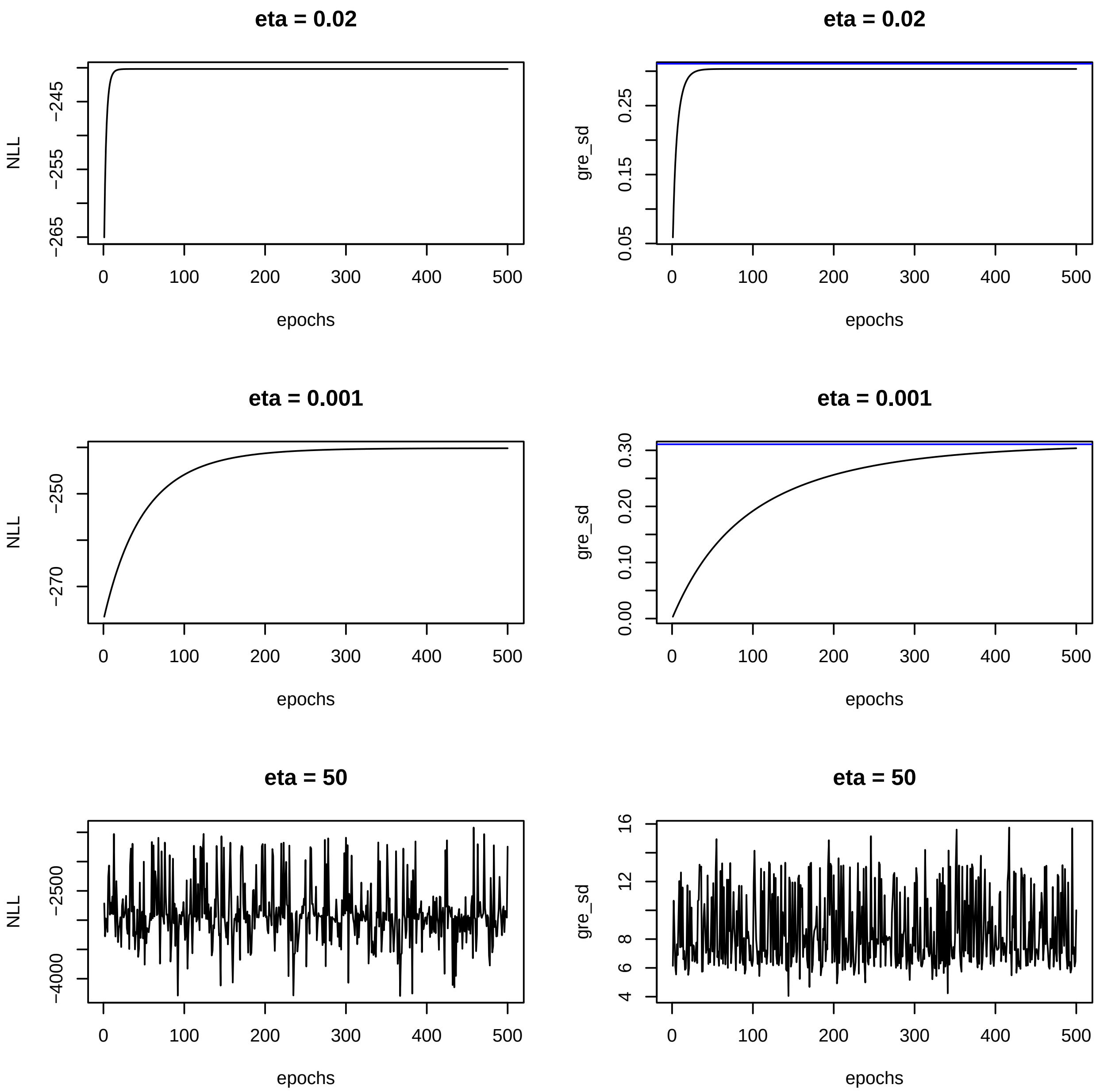

Figure 3: Plots for mini batch gradient decent

Task 2: Regularized Regression

The dataset prob2_train and prob2_test contains of simulated data with 240 explanatory variables and 1 numerical response variable y. We access the data by running the following code:

```
library(uuml)
data("prob2_train")
data("prob2_test")
dim(prob2_train)

## [1] 200 241

# both prob2_train and prob2_test is a high dimensional data sets
# since the number of variables exceeds the number of
# observations ie p > n.

X <- as.matrix(prob2_train[,-241])
y <- as.matrix(prob2_train[,"y"])
X_test <- as.matrix(prob2_test[,-241])
y_test <- as.matrix(prob2_test[,"y"])
```

2.1

Now we will fit the linear model to the training data and analyze the result

```
fit_lm <- lm(y  ~  X)
fit_lm

##
## Call:
## lm(formula = y ~ X)
##
## Coefficients:
## (Intercept)          XV1          XV2          XV3          XV4          XV5
##   4.3087984   -0.4316758    2.4079401    1.6937271    2.0317683    0.6108938
##         XV6          XV7          XV8          XV9         XV10         XV11
##  -5.7093759    1.1600590    2.4372020    2.9902034    2.6017002   -0.1473137
##        XV12         XV13         XV14         XV15         XV16         XV17
##  -0.0748678    0.0742704   -0.2058030    0.5626711    0.2048994    0.1699691
##        XV18         XV19         XV20         XV21         XV22         XV23
##  -0.0007517    0.0795042   -0.4930804   -0.4362306    0.2336115   -0.1530809
##        XV24         XV25         XV26         XV27         XV28         XV29
##  -0.0552062   -0.3068697    0.1770069   -0.0431627   -0.1638641   -0.5018599
##        XV30         XV31         XV32         XV33         XV34         XV35
##  -0.2242053    0.0465612    0.0559708    0.1957272    0.3067858    0.0400114
##        XV36         XV37         XV38         XV39         XV40         XV41
##  -0.3777543    0.1307259   -0.0708513    0.0636356    0.0114887    0.5508829
##        XV42         XV43         XV44         XV45         XV46         XV47
##   0.0094897    0.1023520    0.3155741    0.0066689   -0.1082339    0.1964005
##        XV48         XV49         XV50         XV51         XV52         XV53
##  -0.3088216    0.1180358    0.0677216    0.4599936    0.0939214   -0.2771823
##        XV54         XV55         XV56         XV57         XV58         XV59
##  -0.0866554    0.0749474    0.5939570    0.2075603    0.1437229   -0.6161527
##        XV60         XV61         XV62         XV63         XV64         XV65
```

```
##   0.4929975   0.0981843  -0.0009557   0.3872461  -0.0449157   0.1430436
##        XV66        XV67        XV68        XV69        XV70        XV71
##   0.0326142   0.5234240   0.0020706   0.1369822   0.2024444   0.2218340
##        XV72        XV73        XV74        XV75        XV76        XV77
##  -0.0320236   0.0630302  -0.3436118  -0.5275949   0.0585524   0.0985884
##        XV78        XV79        XV80        XV81        XV82        XV83
##   0.0547782   0.1562050   0.1116718   0.3538088   0.0178448   0.2156009
##        XV84        XV85        XV86        XV87        XV88        XV89
##   0.0130489   0.3569716  -0.1987783  -0.2396075  -0.0561885   0.0965378
##        XV90        XV91        XV92        XV93        XV94        XV95
##  -0.1475249  -0.1366777  -0.2381790  -0.0597348  -0.0814485  -0.4252071
##        XV96        XV97        XV98        XV99       XV100       XV101
##  -0.0553790   0.0405767  -0.1424641  -0.2093864   0.0908628  -0.3454779
##       XV102       XV103       XV104       XV105       XV106       XV107
##  -0.4188662  -0.2023816   0.2899301  -0.1297615   0.1853976  -0.2321716
##       XV108       XV109       XV110       XV111       XV112       XV113
##  -0.2227464  -0.4836678  -0.2692728   0.2190310   0.1031363  -0.0085835
##       XV114       XV115       XV116       XV117       XV118       XV119
##   0.1472213  -0.0155265  -0.0696621  -0.3096262   0.0332686   0.4557763
##       XV120       XV121       XV122       XV123       XV124       XV125
##   0.6678850  -0.0099470   0.2386709   0.0581779   0.6507261  -0.1543057
##       XV126       XV127       XV128       XV129       XV130       XV131
##   0.1315269  -0.1505891  -0.1956084  -0.2149680   0.0374145   0.5312256
##       XV132       XV133       XV134       XV135       XV136       XV137
##   0.1036440   0.2550223  -0.0863616  -0.1737729  -0.1909159   0.0676095
##       XV138       XV139       XV140       XV141       XV142       XV143
##  -0.1550384   0.4711040   0.0466799  -0.2250087  -0.7106237  -0.2128081
##       XV144       XV145       XV146       XV147       XV148       XV149
##   0.0947188   0.0264550  -0.1378737   0.0492007   0.2928196  -0.3995356
##       XV150       XV151       XV152       XV153       XV154       XV155
##  -0.1251145  -0.3301422  -0.1777531   0.0433764  -0.0780777  -0.4844445
##       XV156       XV157       XV158       XV159       XV160       XV161
##  -0.2232337  -0.2852452   0.1532338  -0.1288014   0.0872907   0.2949159
##       XV162       XV163       XV164       XV165       XV166       XV167
##  -1.2774742   0.2637773   0.8292256   0.6332147  -0.2537465   0.5900564
##       XV168       XV169       XV170       XV171       XV172       XV173
##  -0.8314709  -1.1447605  -2.0481609  -0.4067819  -1.8887082   0.3119814
##       XV174       XV175       XV176       XV177       XV178       XV179
##   1.0171499  -0.6616739   0.0568105  -0.5237333   0.3078611  -0.9908706
##       XV180       XV181       XV182       XV183       XV184       XV185
##  -1.3119816   0.3289180   0.8022772  -0.5824423  -0.6563732  -0.4329948
##       XV186       XV187       XV188       XV189       XV190       XV191
##  -0.9425236  -0.4325537   0.2102461   0.0345217   0.9879106  -2.3389023
##       XV192       XV193       XV194       XV195       XV196       XV197
##  -0.3761237  -0.6830105   1.3675606   1.7789825   0.3813573   0.7872641
##       XV198       XV199       XV200       XV201       XV202       XV203
##  -0.8698616  -1.3944968          NA          NA          NA          NA
##       XV204       XV205       XV206       XV207       XV208       XV209
##          NA          NA          NA          NA          NA          NA
##       XV210       XV211       XV212       XV213       XV214       XV215
##          NA          NA          NA          NA          NA          NA
```

```
##        XV216        XV217        XV218        XV219        XV220        XV221
##           NA           NA           NA           NA           NA           NA
##        XV222        XV223        XV224        XV225        XV226        XV227
##           NA           NA           NA           NA           NA           NA
##        XV228        XV229        XV230        XV231        XV232        XV233
##           NA           NA           NA           NA           NA           NA
##        XV234        XV235        XV236        XV237        XV238        XV239
##           NA           NA           NA           NA           NA           NA
##        XV240
##           NA
```

We have that the number of observations for the design matrix X is less than the number of columns / variables in the "training" design matrix X. Hence we get NA for the last 41 covariates.

2.2

Now we will use glmnetfunction from the glmnet package to fit the linear lasso regression to the training data with $\lambda = 1$.

```
library(glmnet)

## Loading required package:  Matrix

## Loaded glmnet 4.1-7

fit_glmnet <- glmnet(X, y, lambda = 1)
estimated_coef <- coefficients(fit_glmnet)
summary(estimated_coef)

## 241 x 1 sparse Matrix of class "dgCMatrix", with 9 entries
##    i j          x
## 1  1 1 31.4929611
## 2  3 1  2.1305973
## 3  4 1  0.8213360
## 4  5 1  1.8581608
## 5  7 1 -5.2947105
## 6  8 1  0.8503707
## 7  9 1  1.7790410
## 8 10 1  2.2202348
## 9 11 1  1.9040888
```

When looking at the output we can see that the model use 9 coefficients when fitting the model, the first being intercept.

2.3 and 2.4

Now we will implement 10 fold cross validation on the training data as a function that has fold variable, X, y and λ value as input and then outputs the RMSE.

```
library(uuml)

glmnet_cv <- function(num_folds, X, y, lambda) {
  X_trainfold <- list()
  X_valfold <- list()
  y_trainfold <- list()
```

```
  y_valfold <- list()
  rmse_values <- c()
  fold <- sample(1:10, nrow(X), replace = TRUE)

  for (i in 1:num_folds) {

    X_trainfold <- X[fold!=i,]
    y_trainfold <- y[fold!=i]
    X_valfold <- X[fold==i,]
    y_valfold <- y[fold==i]

    fit_mod <- glmnet(X_trainfold, y_trainfold,  alpha = 1, lambda = lambda)
    pred_y<- predict(fit_mod, newx = X_valfold)
    rmse_values[i] <- rmse(pred_y, y_valfold)
  }
  output <- mean(rmse_values)
  return(output)
}
```

2.5

Now we will calculate the RMSE for $\lambda = 1$ using 5 fold cross validation on the training data.

```
set.seed(123)
glmnet_cv(num_folds = 5, X, y, lambda= 1)

## [1] 3.260543
```

we have that the RMSE for $\lambda = 1$ using 5 fold cross validation corresponds to 3.260543 when having seed 123.

2.6

Now we will look for the value of the hyper parameter λ that results in the lowest / best root mean squared error, RMSE, with 10fold crossvalidation. This will be done on the training data

```
set.seed(123)
lambda_vector <- seq(0,1, by = 0.01)
optimal_lambda<- 0
optimal_rmse <-Inf

for (i in lambda_vector) {
  rmse <- glmnet_cv(num_folds = 10, X, y, lambda = i)
  if (rmse < optimal_rmse) {
    optimal_rmse <- rmse
    optimal_lambda <- i
  }
}
optimal_lambda

## [1] 0.07
```

Based on the output above we have that $\lambda = 0.07$ results in the lowest RMSE when using 10 fold cross validation and seed 123.

2.7

Now we will the best model to do predictions on the test set.

```
set.seed(123)
best_model <- glmnet(X,y, lambda = optimal_lambda)
pred_best_model <- predict(best_model, newx = X_test)
rmse(pred_best_model, y_test)
```

```
## [1] 0.8038035
```

As can be seen in the output above, the RMSE on the test set corresponds to approximately 0.804.