HW1

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------- tidyvers
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------ tidyverse_confl
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

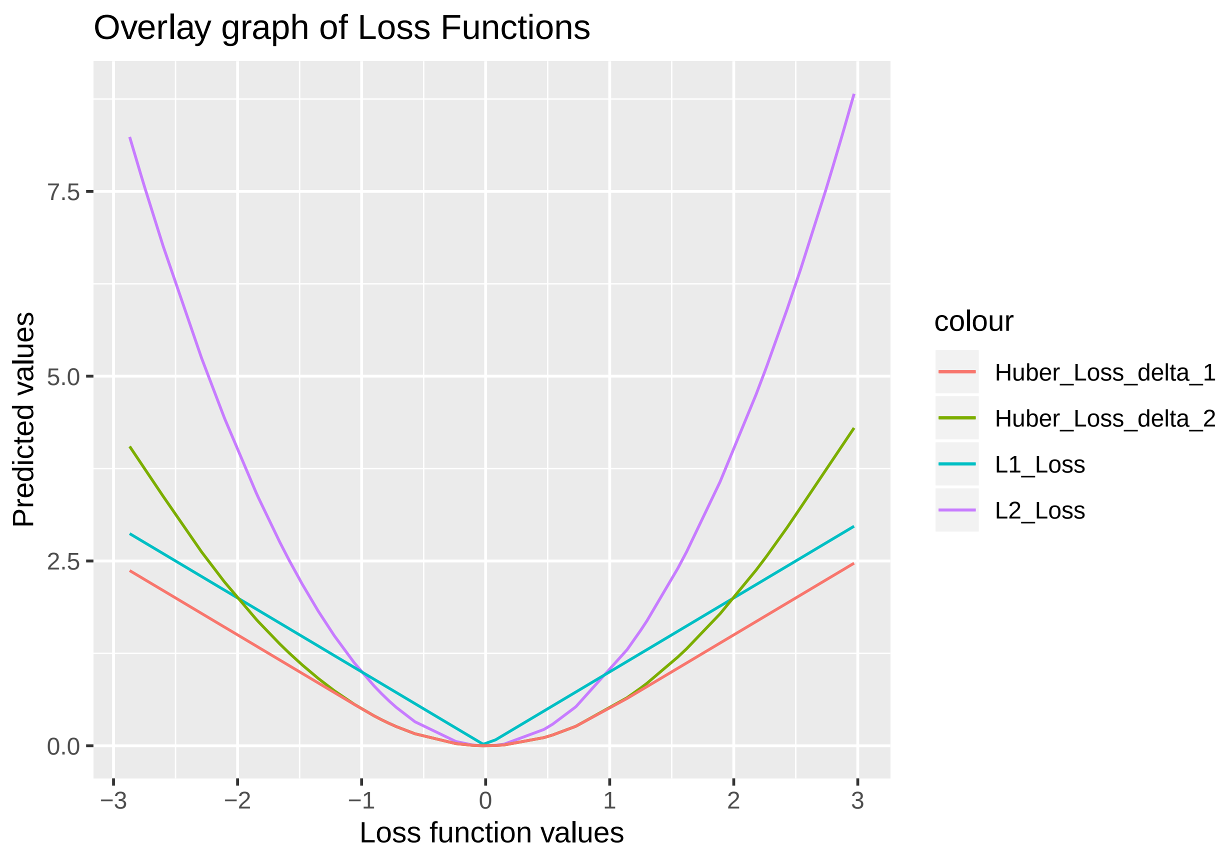
```
library(ggplot2)
library(matlib)
```

```
## Warning: package 'matlib' was built under R version 3.6.2
```

set.seed(12345)

Problem 4a

```
set.seed(123)
y_true <- 0
y_pred <-  sample(seq(from = -3.0, to = 3.0,by=0.01), size = 50, replace= TRUE)

diff <- y_pred-y_true

L2_loss <- diff * diff

MAE <- abs(diff)

calculateHuberLoss <- function(diff,delta){
  loss <- c()
  for(x in diff){
    if(abs(x)<=delta){
      loss <- c(loss,x^2/2)
    }else if(abs(x) > delta){
      loss <- c(loss,delta*abs(x) - delta^2/2)
    }
  }
  calculateHuberLoss <- loss
}

Huber_loss_delta1 <- calculateHuberLoss(diff,2.5)
Huber_loss_delta2 <- calculateHuberLoss(diff,1)


df <- data.frame(y_pred, L2_loss,MAE,Huber_loss_delta1,Huber_loss_delta2)


plot <- ggplot() + geom_line(data= df, aes(x=y_pred, y = L2_loss, color="L2_Loss")) +
  geom_line(data= df, aes(x=y_pred, y=MAE, color="L1_Loss")) +
  geom_line(data= df, aes(x=y_pred, y=Huber_loss_delta1, color="Huber_Loss_delta_2")) +
  geom_line(data= df, aes(x=y_pred, y=Huber_loss_delta2, color="Huber_Loss_delta_1")) +
  scale_fill_manual(name = "Legend",
                    values = c(L2_Loss="red", L1_Loss="blue",
                               Huber_Loss_delta_2="green", Huber_Loss_delta_1="black")) +
  xlab("Loss function values") + ylab("Predicted values") +
  ggtitle("Overlay graph of Loss Functions")

print(plot)
```



Inferences

1) L2 Loss minimizes the difference of squared error. It is sensitive to outliers and might affect the performance.
2) MAE is not continually differentiable and can tgus produce convergence issues in some cases. It is not much affected by outliers as it minimizes the absolute difference between the true and predicted value.
3) Huber loss behaves like MAE for delta greater than loss and behaves like L2 Loss for delta less than loss. It is thus less sensitive to outliers and proves to be a robust model with advantages of both L2 and MAE.

Problem 4b

```r
compute_theta_0 <- function(h_theta,y,alpha,m,lossfunc){
  switch(lossfunc,
         "Quad" = (alpha/m) * 2 * (h_theta-y),
         "MAE"= (alpha/m) * (h_theta-y)/abs(h_theta-y),
         "SquaredError" = (alpha/m) * (h_theta-y),
         "HuberLoss" = compute_huber_theta_0(h_theta,0,y,alpha,m,lossfunc),
  )
}

compute_theta_1 <- function(h_theta,x,y,alpha,m,lossfunc){
  switch(lossfunc,
         "Quad" = (alpha/m) * 2 * ((h_theta-y)*x),
         "MAE"= (alpha/m) * ((h_theta-y)*x)/abs(h_theta-y),
         "SquaredError" = (alpha/m) * (h_theta-y)*x,
         "HuberLoss" = compute_huber_theta_1(h_theta,x,y,alpha,m,lossfunc),
  )
}

compute_huber_theta_0 <- function(h_theta,x,y,alpha,m,lossfunc){
  delta <- 0.5
  diff <- h_theta - y
  theta_huber <- 0
  if (sum(abs(diff))/m <= delta){
      theta_huber <- (alpha/m) * diff
    }
    else{
      theta_huber <- (alpha/m) *  delta * diff/abs(diff)
    }
  compute_huber_theta_0 <- theta_huber
}

compute_huber_theta_1 <- function(h_theta,x,y,alpha,m,lossfunc){
  delta <- 0.6
  diff <- h_theta - y
  theta_huber <- 0
    if (sum(abs(diff))/m <= delta){
      theta_huber <- (alpha/m) * diff * x
    }
    else{
      theta_huber <- (alpha/m) *  delta * diff * x/abs(diff)
    }
  compute_huber_theta_1 <- theta_huber
}

# Batch Gradient Descent Algorithm

computeGradientDecent <- function(x,y,theta,alpha,iter,lossfunc){
  m <- length(y)
  h_theta <- theta[1] + theta[2] * x
  while (iter!=0) {
    temp1 <- theta[1] - sum(compute_theta_0(h_theta,y,alpha,m,lossfunc))
    temp2 <- theta[2] - sum(compute_theta_1(h_theta,x,y,alpha,m,lossfunc))
    theta[1] <- temp1
    theta[2] <- temp2
    h_theta <- theta[1] + theta[2] * x
    iter <- iter - 1
  }
  computeGradientDecent <- theta
}
```

Problem 4c

```
# Stochastic gradient descent

stochasticGradientDecent <- function(x,y,theta,alpha,iter,lossfunc){
  m <- length(y)
  df <- data.frame(x,y)
  h_theta <- theta[1] + theta[2] * x
  rows <- sample(nrow(df))
  df <- df[rows, ]
  i <- 1
  while (iter!=0) {
    for(i in 1:m){
      h_theta[i] <- theta[1] + theta[2] * df$x[i]
      temp1 <- theta[1] - compute_theta_0(h_theta[i],df$y[i],alpha,1,lossfunc)
      temp2 <- theta[2] - compute_theta_1(h_theta[i],df$x[i],df$y[i],alpha,1,lossfunc)
      theta[1] <- temp1
      theta[2] <- temp2
    }
    iter <- iter - 1
  }
  stochasticGradientDecent <- theta
}
```

Problem 5a

```
# Analytical Gradient Descent

computeAnalyticalGradientDescent <- function(x,y){
  A <- matrix(x)
  A <- cbind(c(1), A)
  theta <- inv(t(A) %*% A) %*% t(A) %*% y
}


x <- runif(50, min = -2, max = 2)
e <- rnorm(50,0,4)
iter = 1000
y <- 3 + 2*x + e
alpha <- 0.01
theta <- c(0,0)


theta_analytical <- computeAnalyticalGradientDescent(x,y)
```

```r
theta_batch <- computeGradientDecent(x,y,theta,alpha,1000,"SquaredError")
theta_stochastic <- stochasticGradientDecent(x,y,theta,alpha,1000,"SquaredError")

print(paste("Analytical Solution for Squared Error: ", theta_analytical))
```

```
## [1] "Analytical Solution for Squared Error:  2.9530713186053"
## [2] "Analytical Solution for Squared Error:  2.11375521103797"
```

```r
print(paste("Batch Gradient Descent for Squared Error: ", theta_batch))
```

```
## [1] "Batch Gradient Descent for Squared Error:  2.95294482139306"
## [2] "Batch Gradient Descent for Squared Error:  2.11379013622133"
```

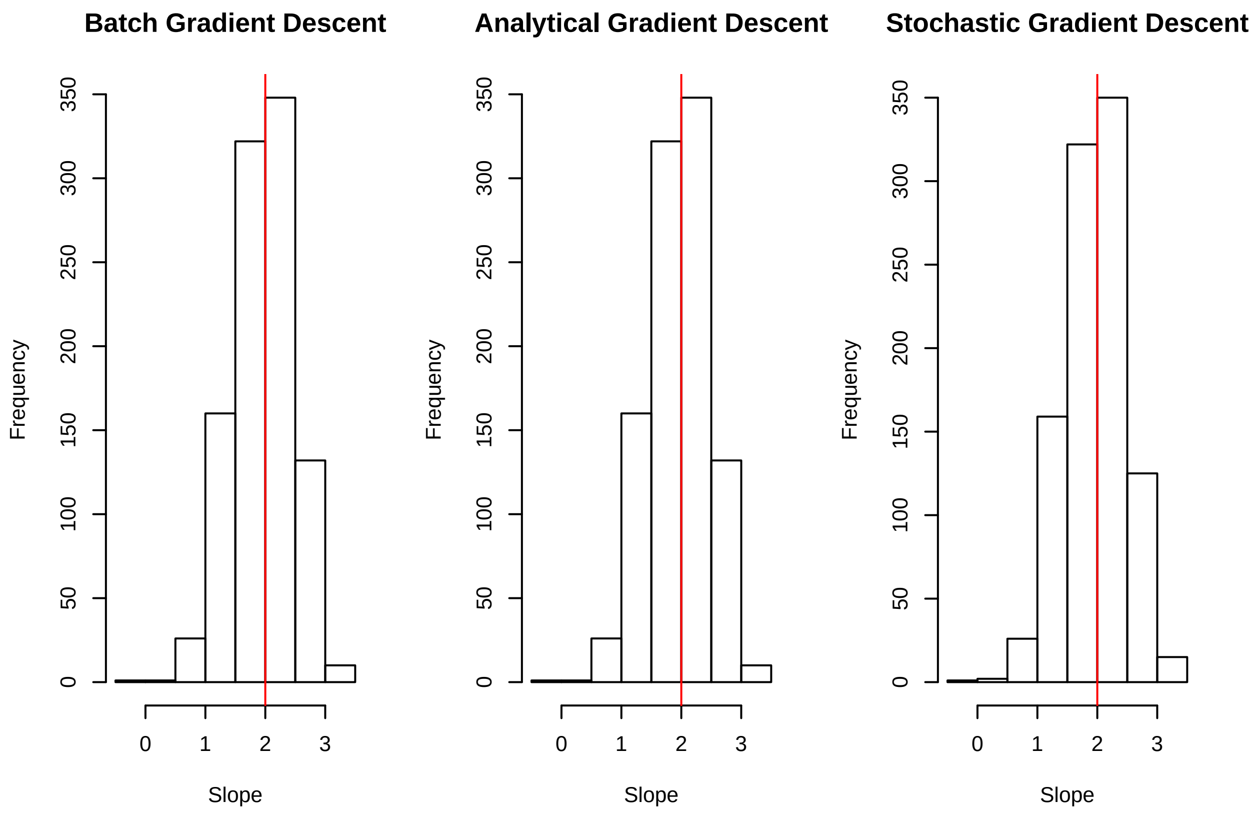
```r
print(paste("Stochastic Gradient Descent for Squared Error: ", theta_stochastic))
```

```
## [1] "Stochastic Gradient Descent for Squared Error:  2.89726308711559"
## [2] "Stochastic Gradient Descent for Squared Error:  2.13142839002772"
```

Problem 5b

```r
set.seed(124)
getSlopes <- function(lossfunc){
  slopeBatch <- c()
  slopeAnalytical <- c()
  slopeStochastic <- c()
  slopeTrue <- c()
  slopeActual <- c()
  lossfunc <- "SquaredError"
    for(epoch in 1:1000){
      x <- runif(50, min = -2, max = 2)
      e <- rnorm(50,0,4)
      iter = 1000
      y <- 3 + 2*x + e
      alpha <- 0.01
      theta <- c(0,0)
      theta_batch <- computeGradientDecent(x,y,theta,alpha,1000,lossfunc)
      slopeBatch <- c(slopeBatch,theta_batch[2])
      theta_Stochastic <- stochasticGradientDecent(x,y,theta,alpha,1000,lossfunc)
      slopeStochastic <- c(slopeStochastic,theta_Stochastic[2])
      theta_analytical <- computeAnalyticalGradientDescent(x,y)
      theta_analytical<- as.vector(theta_analytical)
      slopeAnalytical <- c(slopeAnalytical,theta_analytical[2])
      #slopeActual <- c(slopeActual, coef(lm(y~x))["x"])
    }

    slope <- data.frame(batch = slopeBatch, analytic = slopeAnalytical,
                        stochastic = slopeStochastic)
    getSlopes <- slope
}

slopes <- getSlopes("SquaredError")

par(mfrow=c(1,3))
hist(slopes$batch, xlab="Slope", main="Batch Gradient Descent",breaks=10)
```

```
abline(v=2,col="red")
hist(slopes$analytic, xlab="Slope", main="Analytical Gradient Descent",breaks=10)
abline(v=2,col="red")
hist(slopes$stochastic, xlab="Slope", main="Stochastic Gradient Descent",breaks=10)
abline(v=2,col="red")
```



Inferences

1) We see that the errors follow a normal distribution, where the true value of the slope is centered at 2. We can distinguish the true value from the red line.
2) The histograms of anaytical gradient descent and batch gradient descent are mostly similar to each other.
3) The Stochastic gradient descent learns faster and performs better than the other two as the slope is seen to be centered around 2 for more number of iterations

Problem 5c

```
x <- runif(50, min = -2, max = 2)
e <- rnorm(50,0,4)
iter = 1000
y <- 3 + 2*x + e
alpha <- 0.01
theta <- c(0,0)
```

```r
theta_analytical <- computeAnalyticalGradientDescent(x,y)
batch_MAE <- computeGradientDecent(x,y,theta,alpha,1000,"MAE")
batch_Huber <- computeGradientDecent(x,y,theta,alpha,1000,"HuberLoss")


print(paste("Analytical Solution for Squared Error: ", theta_analytical))
```

```
## [1] "Analytical Solution for Squared Error:  1.78753663761636"
## [2] "Analytical Solution for Squared Error:  1.97785892098406"
```

```r
print(paste("Batch Gradient Descent for MAE: ", batch_MAE))
```

```
## [1] "Batch Gradient Descent for MAE:  2.02160000000002"
## [2] "Batch Gradient Descent for MAE:  1.60382869962567"
```

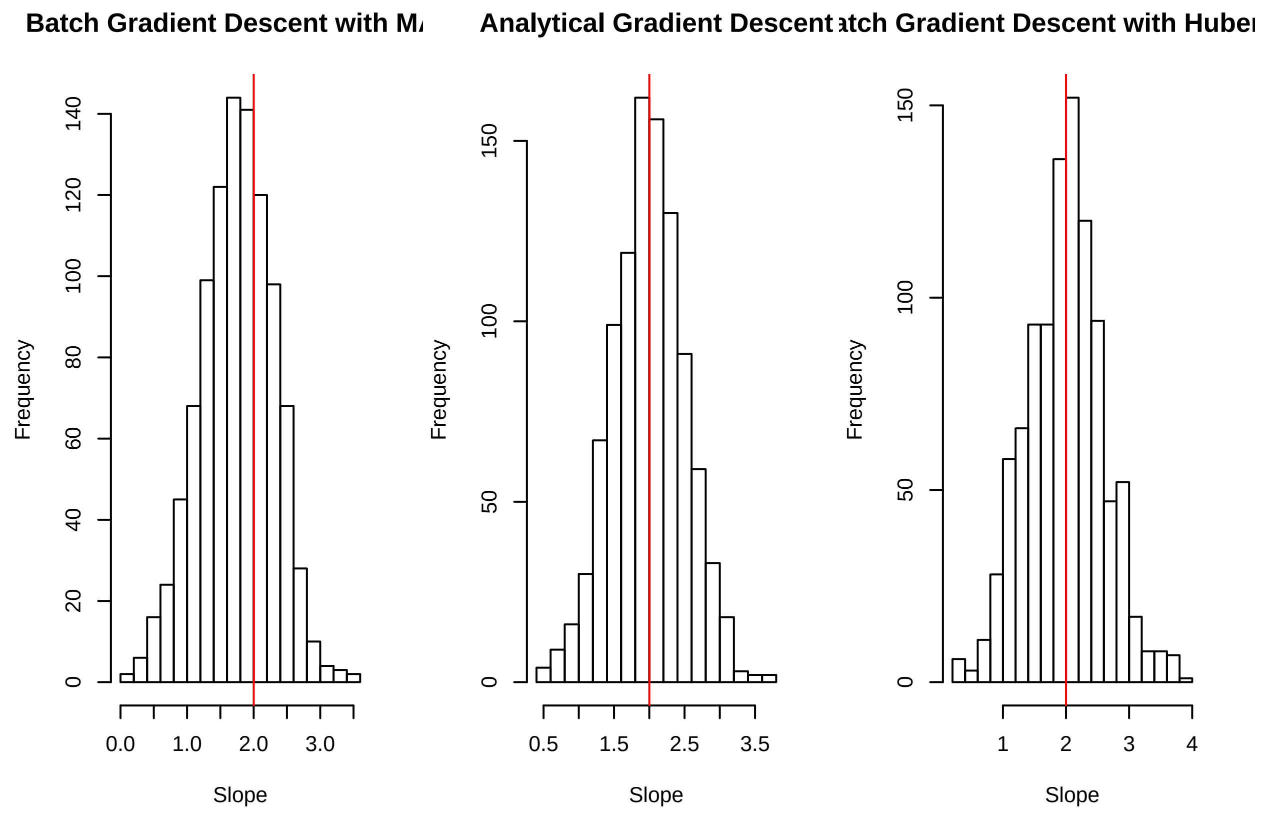
```r
print(paste("Batch Gradient Descent for Huber Lossr: ", batch_Huber))
```

```
## [1] "Batch Gradient Descent for Huber Lossr:  1.27159999999999"
## [2] "Batch Gradient Descent for Huber Lossr:  1.25244065815868"
```

Problem 5d

```r
set.seed(125)
computeSlope <- function(){
  slopeBatch <- c()
  slopeAnalytical <- c()
  slopeStochastic <- c()
  slopeActual <- c()
    for(epoch in 1:1000){
      x <- runif(50, min = -2, max = 2)
      e <- rnorm(50,0,4)
      iter = 1000
      y <- 3 + 2*x + e
      alpha <- 0.01
      theta <- c(0,0)
      theta_batch <- computeGradientDecent(x,y,theta,alpha,1000,"MAE")
      slopeBatch <- c(slopeBatch,theta_batch[2])
      theta_Stochastic <- stochasticGradientDecent(x,y,theta,alpha,1000,"HuberLoss")
      slopeStochastic <- c(slopeStochastic,theta_Stochastic[2])
      theta_analytical <- computeAnalyticalGradientDescent(x,y)
      theta_analytical<- as.vector(theta_analytical)
      slopeAnalytical <- c(slopeAnalytical,theta_analytical[2])
     # slopeActual <- c(slopeActual, coef(lm(y~x))["x"])
    }

    slope <- data.frame(batch = slopeBatch, analytic = slopeAnalytical,
                         stochastic = slopeStochastic)
    computeSlope <- slope
}

slopes <- computeSlope()

par(mfrow=c(1,3))
hist(slopes$batch, xlab="Slope", main="Batch Gradient Descent with MAE", breaks=15)
```

```r
abline(v=2,col="red")
hist(slopes$analytic, xlab="Slope", main="Analytical Gradient Descent", breaks=15)
abline(v=2,col="red")
hist(slopes$stochastic, xlab="Slope", main="Batch Gradient Descent with Huber Loss", breaks=15)
abline(v=2,col="red")
```



Inferences

1) We can see that the histogram of slopes calculated using various gradient algorithms follow a normal distribution
2) Analytic gradient descent is an overall good model as it is very robust and differntiable at all points
3) MAE is robust to outliers, but as the function is non-diffentiable at the minimum point, a convergence issue is seen due to which few predictions are poor
4) Huber Loss is a tradeoff between MAE and Analytical Gradient Descent. Based on the delta values, we see that huber loss performs like analytical gradient descent for loss values less than delta and like MAE for loss values greater than delta. This way it is able to take into account robustness by setting the optimum delta value

Problem 5e

```r
x <- runif(50, min = -2, max = 2)
e <- rnorm(50,0,4)
iter = 1000
y <- 3 + 2*x + e
```

```r
alpha <- 0.01
theta <- c(0,0)

y <- as.data.frame(y)

n_samp <- sample_frac(y, 0.1)
y<- vecsets::vsetdiff( y$y, n_samp$y )

inc_outlier <- sample_frac(n_samp, 0.5)
n_samp<- vecsets::vsetdiff( n_samp$y, inc_outlier$y )
inc_outlier <- inc_outlier * 3

n_samp <- as.data.frame(n_samp)

dec_outlier <- sample_frac(n_samp, 0.5)
n_samp <- vecsets::vsetdiff( n_samp$n_samp, dec_outlier$n_samp )
dec_outlier <- dec_outlier/3

n_samp <- c(n_samp,dec_outlier$n_samp)
n_samp <- c(n_samp,inc_outlier$y)

y <- c(y,n_samp)

theta_analytical_outlier <- computeAnalyticalGradientDescent(x,y)
theta_MAE_outlier <- computeGradientDecent(x,y,theta,alpha,1000,"MAE")
theta_Huber_outlier <- computeGradientDecent(x,y,theta,alpha,1000,"HuberLoss")

print(paste("Analytical Solution for Squared Error: ", theta_analytical_outlier))
```

```
## [1] "Analytical Solution for Squared Error:  2.91708282212538"
## [2] "Analytical Solution for Squared Error:  1.07234055432612"
```

```r
print(paste("Batch Gradient Descent for MAE: ", theta_MAE_outlier))
```

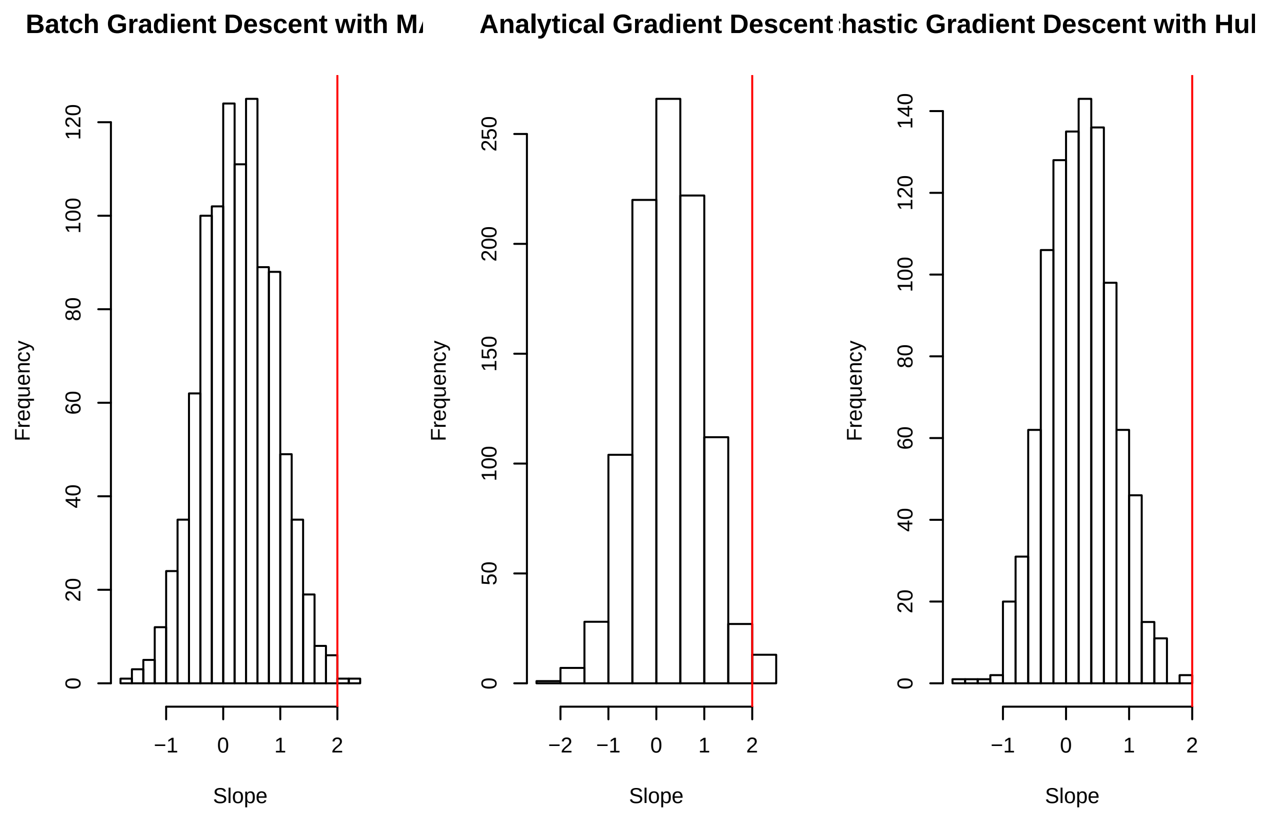
```
## [1] "Batch Gradient Descent for MAE:  1.97520000000003"
## [2] "Batch Gradient Descent for MAE:  1.27436579127458"
```

```r
print(paste("Batch Gradient Descent for Huber Lossr: ", theta_Huber_outlier))
```

```
## [1] "Batch Gradient Descent for Huber Lossr:  1.17479999999998"
## [2] "Batch Gradient Descent for Huber Lossr:  0.976947341531242"
```

Problem 5f

```r
set.seed(126)
computeSlopeOutlier <- function(){
  slopeBatch <- c()
  slopeAnalytical <- c()
  slopeStochastic <- c()
  slopeActual <- c()
    for(epoch in 1:1000){
      x <- runif(50, min = -2, max = 2)
      e <- rnorm(50,0,4)
      iter = 1000
      y <- 3 + 2*x + e
      alpha <- 0.01
      theta <- c(0,0)

      y <- as.data.frame(y)

      n_samp <- sample_frac(y, 0.1)
      y<- vecsets::vsetdiff( y$y, n_samp$y )

      inc_outlier <- sample_frac(n_samp, 0.5)
      n_samp<- vecsets::vsetdiff( n_samp$y, inc_outlier$y )
      inc_outlier <- inc_outlier * 3

      n_samp <- as.data.frame(n_samp)

      dec_outlier <- sample_frac(n_samp, 0.5)
      n_samp <- vecsets::vsetdiff( n_samp$n_samp, dec_outlier$n_samp )
      dec_outlier <- dec_outlier/3

      n_samp <- c(n_samp,dec_outlier$n_samp)
      n_samp <- c(n_samp,inc_outlier$y)

      y <- c(y,n_samp)

      theta_batch <- computeGradientDecent(x,y,theta,alpha,1000,"MAE")
      slopeBatch <- c(slopeBatch,theta_batch[2])
      theta_Stochastic <- computeGradientDecent(x,y,theta,alpha,1000,"HuberLoss")
      slopeStochastic <- c(slopeStochastic,theta_Stochastic[2])
      theta_analytical <- computeAnalyticalGradientDescent(x,y)
      theta_analytical<- as.vector(theta_analytical)
      slopeAnalytical <- c(slopeAnalytical,theta_analytical[2])
     # slopeActual <- c(slopeActual, coef(lm(y~x))["x"])
    }

    slope <- data.frame(batch = slopeBatch, analytic = slopeAnalytical,
                        stochastic = slopeStochastic)
    computeSlopeOutlier <- slope
}

slopes <- computeSlopeOutlier()

par(mfrow=c(1,3))
hist(slopes$batch, xlab="Slope", main="Batch Gradient Descent with MAE", breaks=15)
abline(v=2,col="red")
hist(slopes$analytic, xlab="Slope", main="Analytical Gradient Descent", breaks=15)
abline(v=2,col="red")
hist(slopes$stochastic, xlab="Slope", main="Stochastic Gradient Descent with Huber Loss", breaks=15)
abline(v=2,col="red")
```



Inferences

1) Here also, the histogram of slopes calculated using various gradient algorithms follow a normal distribution
2) Since it considers square of errors, snalytic gradient descent shows few distortions in the presence of large outliers. Howerver due to its robustness and continuitiy it is an overall good model
3) MAE being an absolute value of the error term, it is robust to outliers, but as the function is non-diffentiable at the minimum point, a convergence issue is seen due to which few predictions are poor
4) Huber Loss is a tradeoff between MAE and Analytical Gradient Descent. Based on the delta values, we see that huber loss performs like analytical gradient descent for loss values less than delta and like MAE for loss values greater than delta. As most values are centered around true value, the predictions are robust even in the presence of outliers