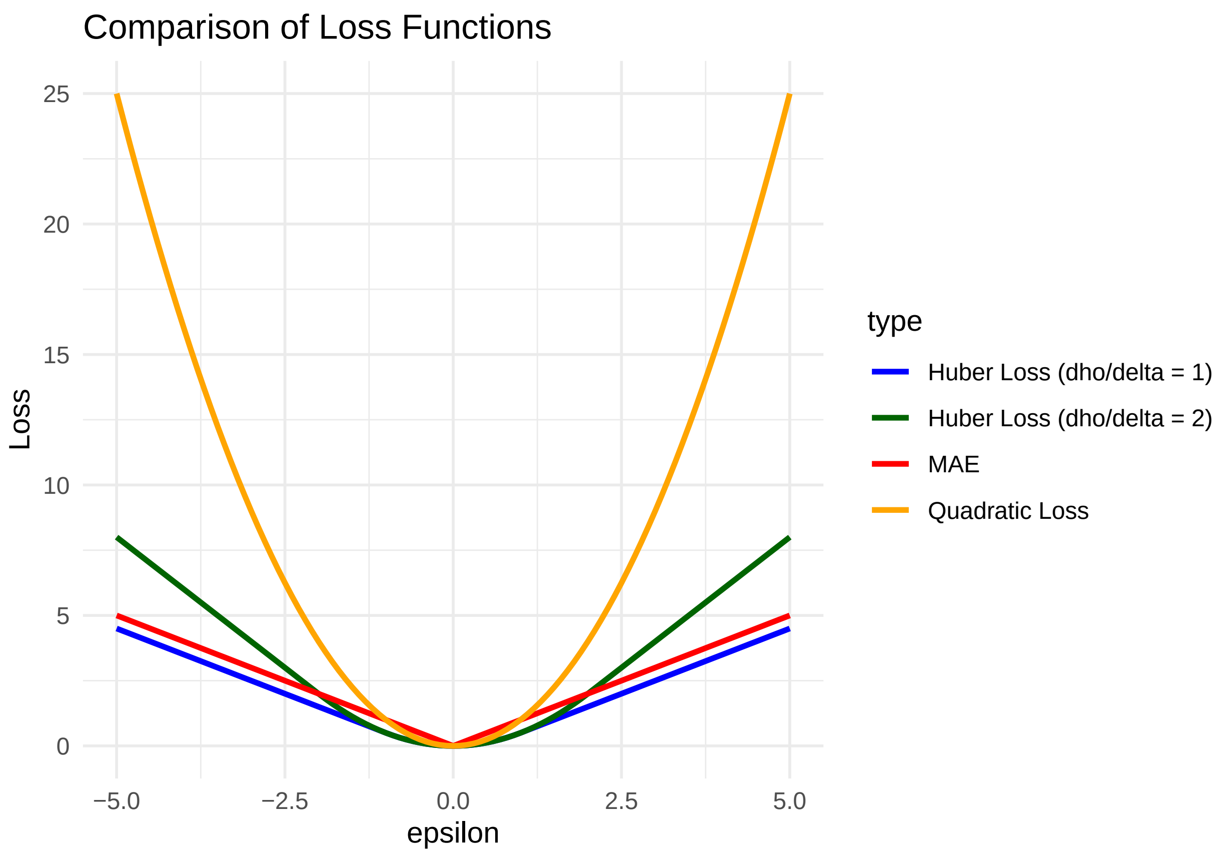
Comparative Study of Linear Regression Loss Functions

M.G.Manjusha

2024-02-22

I) Over Layed Graphs - Comparisions of Loss functions

```r
e <- seq(-5, 5, by = 0.1)
#Quadratic loss
quad_loss <- e^2
df_quad<- data.frame(e=e,loss=quad_loss,type = "Quadratic Loss")

#Mean Absolute Error
loss_mae <- abs(e)
df_mae <- data.frame(e = e, loss = loss_mae,type = "MAE")

#Huber loss - threshold parameters
delta_1 <- 1
delta_2 <- 2
e_huber<- e

# Computing Huber loss function for each epsilon value and each delta value
huber_loss_1 <- ifelse(abs(e_huber) <= delta_1, 0.5 * e_huber^2,
                       delta_1 * (abs(e_huber) - 0.5 * delta_1))
huber_loss_2 <- ifelse(abs(e_huber) <= delta_2, 0.5 * e_huber^2,
                       delta_2 * (abs(e_huber) - 0.5 * delta_2))

df_huber_1 <- data.frame(e = e_huber, loss = huber_loss_1,
                         type = "Huber Loss (dho/delta = 1)")
df_huber_2 <- data.frame(e = e_huber, loss = huber_loss_2,
                         type = "Huber Loss (dho/delta = 2)")

#Plotting all 3 plots
combined_df <- rbind(df_quad, df_mae, df_huber_1, df_huber_2)

library(ggplot2)
ggplot(combined_df, aes(x = e, y = loss, color = type)) +
  geom_line(size = 1) +
  labs(x = "epsilon", y = "Loss", title = "Comparison of Loss Functions") +
  scale_color_manual(values = c("blue", "darkgreen", "red", "orange")) +
  theme_minimal()
```



Advantages and Disadvantages of the functions

Quadratic Loss (MSE):

Advantages:

- It is a convex function that makes it differentiable, which makes it easy to calculate gradients for optimization algorithms like gradient descent.
- It heavily penalizes larger errors, which can be a good property when large errors are not wantedd.
- It has a nice statistical interpretation, as minimizing MSE leads to the maximum likelihood estimation under the assumption that errors are normally distributed.

Disadvantages:

- Since the errors are squared,it becomes sensitive to outliers, causing outliers to have a disproportionately large effect on the total loss.
- It may lead to over-emphasis on large errors at the expense of numerous small errors, which might not be ideal in all situations.

Mean Absolute Error (MAE):

Advantages:

- It is robust to outliers, as each error contributes linearly to the total loss, hence preventing the disproportionate effect of outliers.
- It is easy to understand and interpret because it directly reflects the average magnitude of errors.

Disadvantages:

- It is not differentiable at zero, which can complicate the use of gradient-based optimization methods.
- It may not sufficiently penalize large errors, depending on the context, which could be a drawback when accuracy is crucial.

Huber Loss:

Advantages:

- It combines the best of both MAE and MSE. For errors smaller than dho/delta(parameter), it is quadratic, and for larger errors, it is linear, making it less sensitive to outliers than MSE.
- It is differentiable everywhere, which makes it suitable for gradient-based optimization algorithms.
- The dho/delta parameter can be adjusted to control the sensitivity to outliers.

Disadvantages:

- It introduces an additional hyperparameter dho/delta, which needs to be tuned, adding to the complexity of the model.
- The piecewise definition of the function can make it a bit harder to understand as well as implement compared to MSE or MAE.

II) Gradient Descent Implementation

```
gradient_step_quad <- function(x, y,theta_0, theta_1) {
    pred <- theta_0 + theta_1 * x
    # Calculate gradients
    error <- pred-y
    gradient_0 <- (2 * sum(error))
    gradient_1 <- (2 * sum(error * x))

  return(c(gradient_0, gradient_1))
}

gradient_step_mae<- function(x,y,theta_0,theta_1){
    n<-length(y)
    pred <- theta_0 + theta_1*x
    #gradients calculation
    error <- pred-y
    gradient_0 <- mean(ifelse(error>0,1,-1))
    gradient_1 <- mean(ifelse(error>0,x,-x))
    return(c(gradient_0, gradient_1))
}

gradient_step_huber <- function(x, y,theta_0, theta_1,dho=1){

    pred <- theta_0 + theta_1*x
    error <- pred - y
    abs_error <- abs(error)
    # Applying condition based on absolute residuals
    cond <- abs_error <= dho
    gradient_0 <- mean(ifelse(cond,error, dho*sign(error)))
    gradient_1 <- mean(ifelse(cond,error*x, dho*sign(error)*x))
    return(c(gradient_0, gradient_1))
}
```

Batch gradient descent implementation

```r
batch_gradient_descent<-function(x,y,theta_0,theta_1,gradient_step_fun,alpha,iteration){
  for (i in 1:iteration) {
    gradient <- gradient_step_fun(x,y,theta_0,theta_1)
    theta_0 <- theta_0 - alpha*gradient[1]
    theta_1 <- theta_1 - alpha*gradient[2]
    }
  return(c(theta_0, theta_1))
}
```

III) Stochastic Gradient Descent implementation

```r
stochastic_gradient_descent <- function(x, y, theta_0, theta_1, gradient_step_fun, alpha, iteration) {
  for (i in 1:iteration) {
    shuffle_index <- sample(1:length(x))
    x <- x[shuffle_index]
    y <- y[shuffle_index]

    #Updating parameters for each data point
    for (j in 1:length(x)) {
      gradient <- gradient_step_fun(x[j], y[j], theta_0, theta_1)
      theta_0 <- theta_0 - alpha * gradient[1]
      theta_1 <- theta_1 - alpha * gradient[2]
    }
  }
  return(c(theta_0, theta_1))
}
```

Master function to call Gradient or Stochastic gradient descent

```r
grad_descent <- function(x,y,theta_0,theta_1,gradient_step_fun,optim,alpha,iteration){
  new_thetas <- optim(x,y,theta_0,theta_1,gradient_step_fun,alpha,iteration)
  return(new_thetas)
}
```

Batch and Stochastic Gradient Descent calculation

```r
set.seed(123)
x <- runif(50, -2, 2)
e<- rnorm(50, mean = 0, sd = 4)
y <- 3 + 2*x + e
theta_0 <- rnorm(1)
theta_1 <- rnorm(1)
alpha <- 0.01
iteration <- 1000

cat("Batch gradient Descent with Quadratic Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_quad,batch_gradient_descent,alpha,iteration),"\n")
```

```
## Batch gradient Descent with Quadratic Loss: 3.194078 2.381745
```

```r
cat("Batch gradient Descent with Mean Absolute Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_mae,batch_gradient_descent,alpha,iteration),"\n")
```

```
## Batch gradient Descent with Mean Absolute Loss: 2.868771 2.226768
```

```
cat("Batch gradient Descent with Huber Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_huber,batch_gradient_descent,alpha,iteration),"\n")
```

```
## Batch gradient Descent with Huber Loss: 2.890835 2.257367
```

```
cat("Stochastic gradient Descent with Quadratic Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_quad,stochastic_gradient_descent,alpha,iteration),"\n")
```

```
## Stochastic gradient Descent with Quadratic Loss: 3.276427 2.284515
```

```
cat("Stochastic gradient Descent with Mean absolute Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_mae,stochastic_gradient_descent,alpha,iteration),"\n")
```

```
## Stochastic gradient Descent with Mean absolute Loss: 3.165571 2.955896
```

```
cat("Stochastic gradient Descent with Huber Loss:",grad_descent(x,y,theta_0,theta_1,
                gradient_step_huber,stochastic_gradient_descent,alpha,iteration),"\n")
```

```
## Stochastic gradient Descent with Huber Loss: 3.203063 2.779048
```

Fitting Linear Regression using Simulated Data

```
set.seed(123)
alpha <- 0.01
iteration <- 1000
x <- runif(50, -2, 2)
e<- rnorm(50, mean = 0, sd = 4)
y <- 3 + 2*x + e
theta_0 <- rnorm(1)
theta_1 <-rnorm(1)

#i) Analytical Solution
X <- cbind(1, x)
analytical <- solve(t(X) %*% X) %*% t(X) %*% y

#ii) Batch gradient descent slopes - Quadratic loss
batch_quad <- grad_descent(x, y, theta_0, theta_1, gradient_step_quad,
                           batch_gradient_descent, alpha, iteration)

#iii) Stochastic gradient descent slopes - Quadratic loss
sgd_quad <- grad_descent(x, y, theta_0, theta_1, gradient_step_quad,
                         stochastic_gradient_descent, alpha, iteration)

cat("Linear Regression with Quadratic loss \n")
```

```
## Linear Regression with Quadratic loss
```

```
cat("Analytical solution: Theta_0=",analytical[1],"Theta_1=",analytical[2],"\n")
```

```
## Analytical solution: Theta_0= 3.194078 Theta_1= 2.381745
```

```
cat("Batch Gradient Descent: Theta_0=",batch_quad[1],"Theta_1=",batch_quad[2],"\n")
```

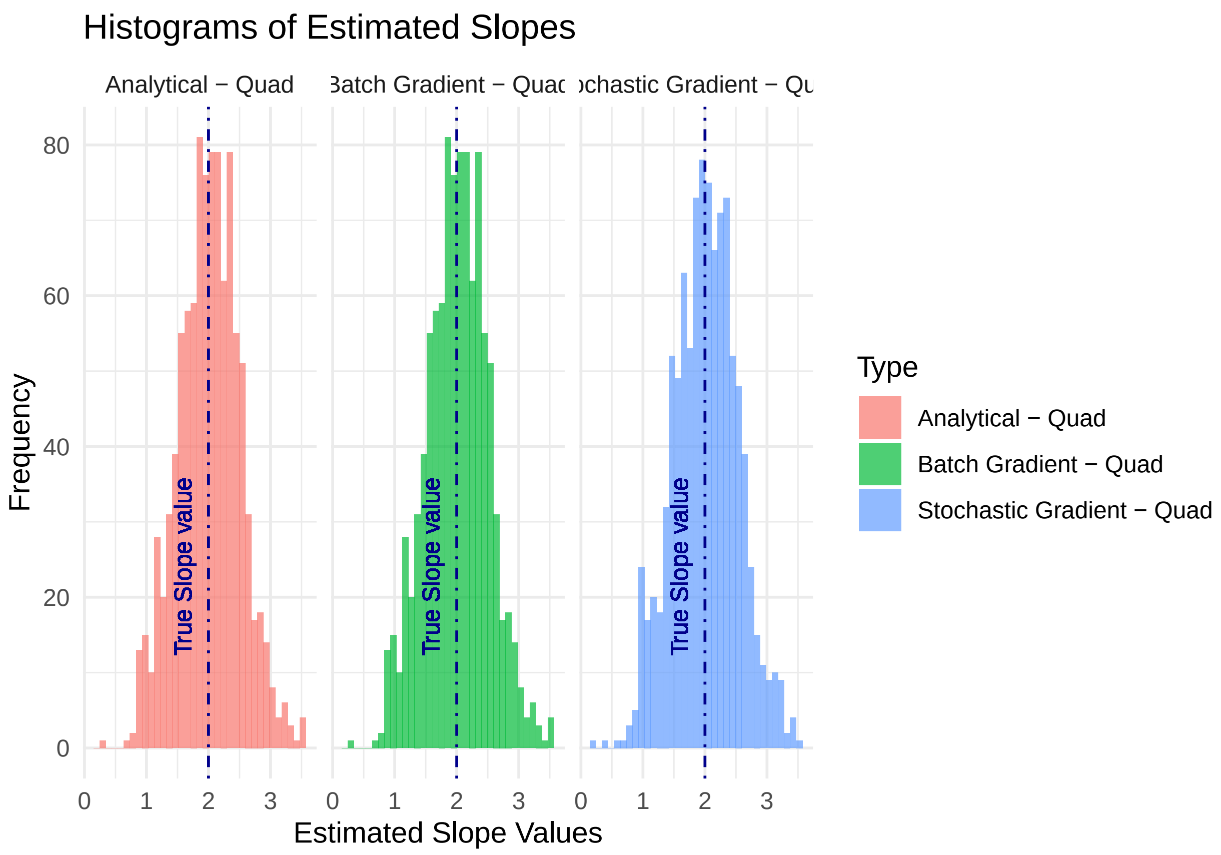
```
## Batch Gradient Descent: Theta_0= 3.194078 Theta_1= 2.381745
```

```
cat("Stochastic Gradient Descent: Theta_0=",sgd_quad[1],"Theta_1=",sgd_quad[2],"\n")
```

```
## Stochastic Gradient Descent: Theta_0= 3.276427 Theta_1= 2.284515
```

Repeating 1000 times

```
set.seed(123)
alpha <- 0.01
iteration <- 1000

library(ggplot2)

slopes <-  function(){
  x <- runif(50, min = -2, max = 2)
  e <- rnorm(50, mean = 0, sd = 4)
  y <- 3 + 2 * x + e
  iteration <- 1000
  theta_0 <- 0
  theta_1 <- 0

  #Analytical solution
  X <- cbind(1, x)
  analytical <- solve(t(X) %*% X) %*% t(X) %*% y

  slopes_batch_quad <- grad_descent(x, y, theta_0, theta_1, gradient_step_quad,
                                   batch_gradient_descent, alpha, iteration)

  slopes_sgd_quad <- grad_descent(x, y, theta_0, theta_1, gradient_step_quad,
                                 stochastic_gradient_descent, alpha, iteration)

  return(c(analytical[2],slopes_batch_quad[2], slopes_sgd_quad[2]))
}

esti_slopes<-replicate(iteration,slopes())
df <- data.frame(slope = c(esti_slopes[1,],esti_slopes[2,],esti_slopes[3,]),
                 Type = c(rep("Analytical - Quad",iteration),
                          rep("Batch Gradient - Quad",iteration),
                          rep("Stochastic Gradient - Quad",iteration)))

ggplot(df, aes(x = slope, fill = Type)) +
  geom_histogram(position = position_dodge(width = 0.75), alpha = 0.7, bins = 35) +
  geom_vline(xintercept = 2, color = "darkblue", linetype = "dotdash") +
  facet_wrap(~Type)+
  labs(x = "Estimated Slope Values", y = "Frequency",
       title = "Histograms of Estimated Slopes") +
  geom_text(aes(x = 2, y = 30, label = "True Slope value"),position = "dodge",
            vjust = -1, hjust = 0.75, color = "darkblue", angle = 90, size = 3)+
  theme_minimal()
```



From the histogram we can see that all three algorithms— Analytical, Batch Gradient, and Stochastic Gradient using Quadratic loss—perform similarly in estimating the slope, clustering around the true value with comparable variability.

Analytical method and Batch Gradient Descent : Both of these methods have similar estimated slope values with similar precision.They both are quite narrow which makes then more sensitive to the extreme values and gives lower accuracy compared to Stochastic Gradient Descent

Stochastic Batch Gradient Descent: This is a bit wider compared to the other 2, which can be said it is less sensitive to extreme values than the other two methods and hence gives better precision.

Fitting Linear Regression with Quadratic Loss

```
#analytical <- analytical_sol(x, y, theta_0, theta_1,iteration, FALSE)
#ii) Batch gradient descent slopes - MAE
batch_mae <- grad_descent(x, y, theta_0, theta_1, gradient_step_mae,
                          batch_gradient_descent, alpha, iteration)

#iii) Batch gradient descent slopes - Huber
batch_huber <- grad_descent(x, y, theta_0, theta_1, gradient_step_huber,
                            batch_gradient_descent, alpha, iteration)

cat("Linear Regression with Quadratic loss \n")
```

```
## Linear Regression with Quadratic loss
```

```
cat("Analytical solution: Theta_0=",analytical[1],"Theta_1=",analytical[2],"\n")
```

```
## Analytical solution: Theta_0= 3.194078 Theta_1= 2.381745
```

```r
cat("Batch Gradient Descent MAE: Theta_0=",batch_mae[1],"Theta_1=",batch_mae[2],"\n")
```

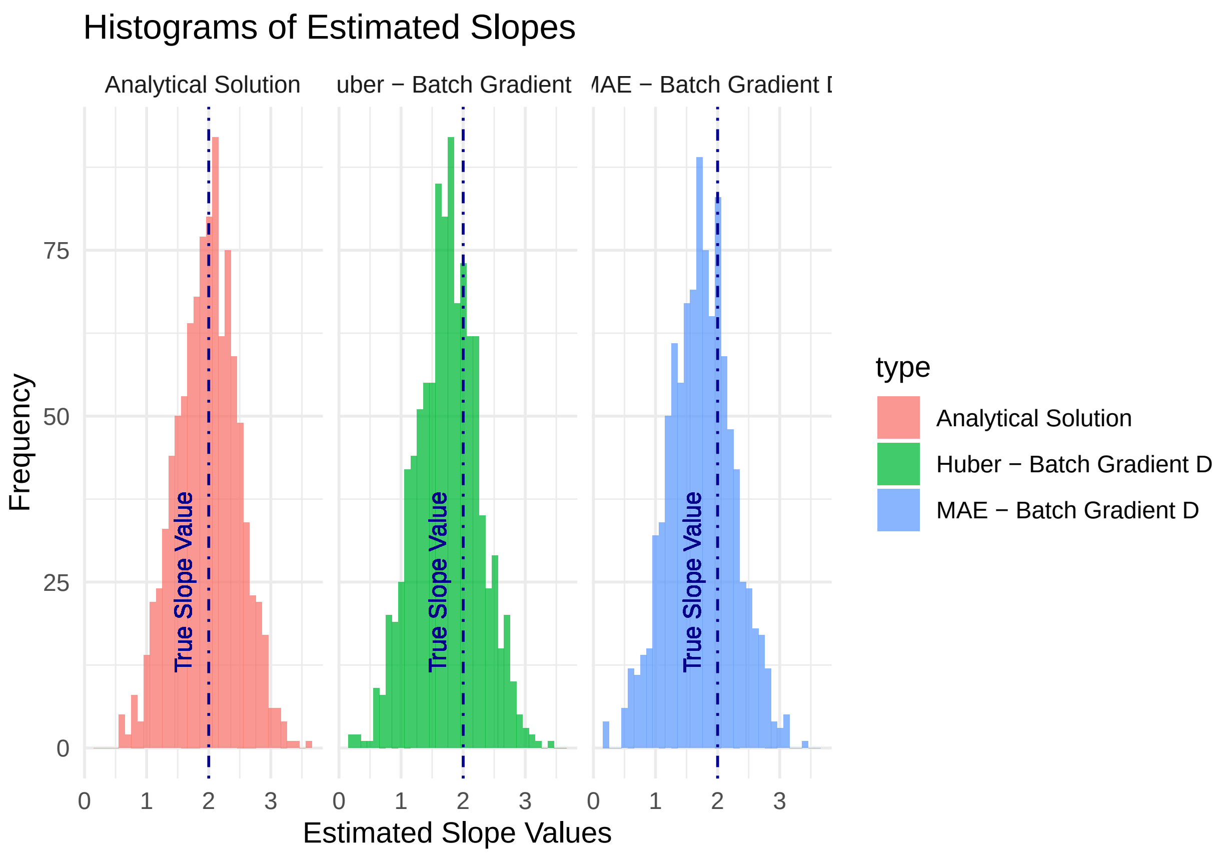
```
## Batch Gradient Descent MAE: Theta_0= 2.868771 Theta_1= 2.226768
```

```r
cat("Batch Gradient Descent Huber: Theta_0=",batch_huber[1],"Theta_1=",batch_huber[2],"\n")
```

```
## Batch Gradient Descent Huber: Theta_0= 2.890835 Theta_1= 2.257367
```

Repeating 1000 times

```r
# Initialize variables
alpha <- 0.01
iteration <- 1000

slopes <- function() {

  x <- runif(50, min = -2, max = 2)
  e<- rnorm(50, mean = 0, sd = 4)
  y <- 3 + 2 * x + e
  iteration <- 1000
  theta_0<-0
  theta_1<-0

  X <- cbind(1, x)
  analytical <- solve(t(X) %*% X) %*% t(X) %*% y

  batch_mae <- grad_descent(x,y,theta_0,theta_1, gradient_step_mae,
                            batch_gradient_descent, alpha, iteration)

  batch_huber <- grad_descent(x,y,theta_0,theta_1, gradient_step_huber,
                              batch_gradient_descent, alpha, iteration)

  return(c(analytical[2], batch_mae[2], batch_huber[2]))
}

esti_slopes <- replicate(iteration, slopes())

df <- data.frame(
  slope = c(esti_slopes[1,], esti_slopes[2,], esti_slopes[3,]),
  type = c(rep("Analytical Solution", iteration),
           rep("MAE - Batch Gradient D", iteration),
           rep("Huber - Batch Gradient D", iteration))
)

# Plotting histograms
ggplot(df, aes(x = slope, fill = type)) +
  geom_histogram(position = position_dodge(width = 0.5), alpha = 0.75, bins = 35) +
  geom_vline(xintercept=2, color = "darkblue", linetype = "dotdash") +
  labs(x = "Estimated Slope Values", y = "Frequency", title = "Histograms of Estimated Slopes") +
  facet_wrap(~type)+
  geom_text(aes(x = 2, y = 20, label = "True Slope Value"), position = position_nudge(y = 5),
          vjust = -1, hjust = 0.5, color = "darkblue", angle = 90, size = 3)+
  theme_minimal()
```



The histogram plot shows the distribution of estimated slope values using three different algorithms:

Analytical Solution, Batch Gradient Descent using Huber loss, and Batch Gradient Descent using MAE.

Analytical Solution: Here the estimates are centered around the true value, which says it has high accuracy and low variance.

Batch Gradient Descent using Huber loss: It is usually designed to be robust to outliers,it shows a slightly wider distribution compared to analytical, hence it can be said it is less sensitive to extreme values than the analytical method.

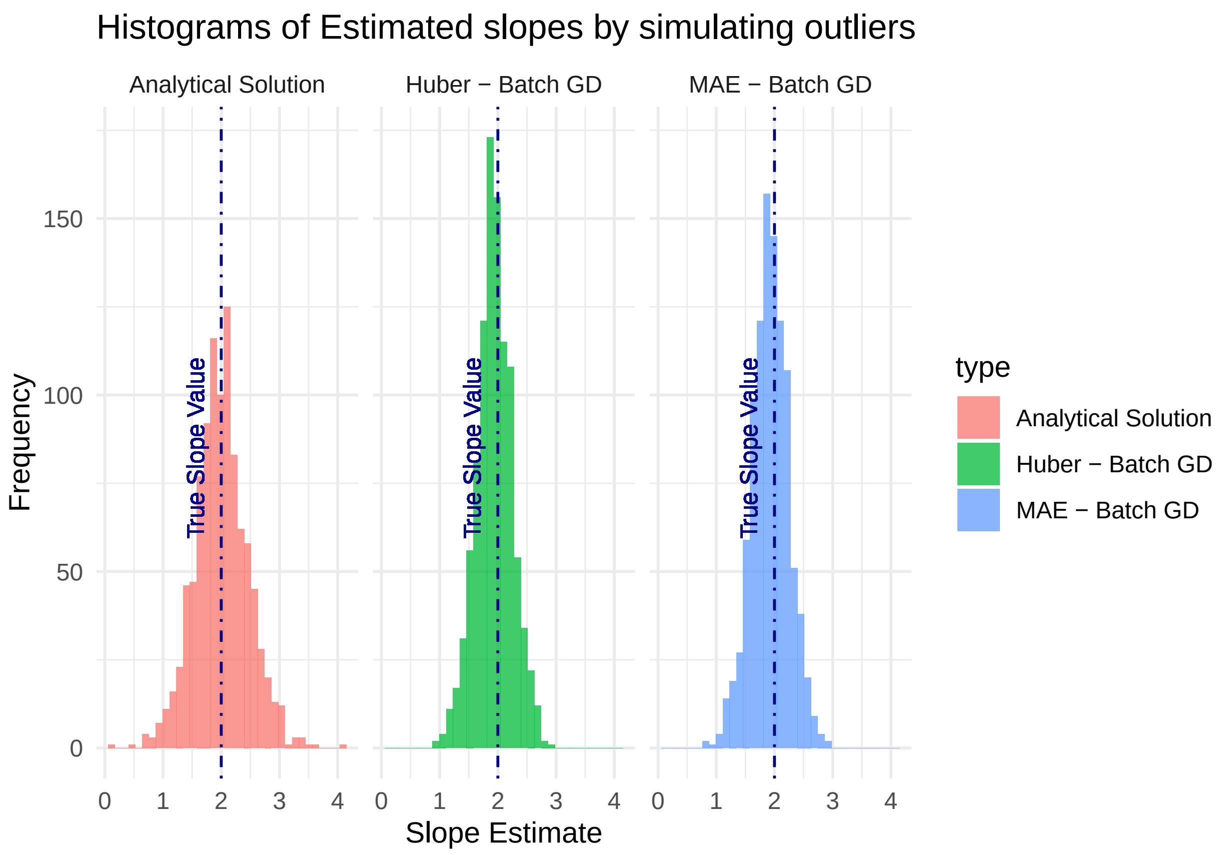
Batch Gradient Descent using Mean Absolute Error: The slope estimates have the broadest distribution, which might indicate it has higher robustness to outliers but has less precision compared to both Analytical solution and Batch gradient using Huber loss.

Simulate outliers

```r
set.seed(123)
n <- 50
iteration <- 1000
alpha <- 0.01

simulate_Y <- function(x) {
  e <- rnorm(n, mean = 0, sd = 2)
  y <- 3 + 2*x + e
  for(i in 1:n){
    if(runif(1) < 0.1){
      if(runif(1) < 0.5){
        y[i] <- y[i] + 2*abs(y[i])
      } else {
        y[i] <- y[i] - 2*abs(y[i])
      }
      }
  }
  return(y)
}

slopes <- function() {
  x <- runif(n, min = -2, max = 2)
  y <- simulate_Y(x)
  X <- cbind(1, x)

  # Analytical solution
  analytical <- solve(t(X) %*% X) %*% t(X) %*% y
  theta_1_analytical <- analytical[2,] # Extract slope

  batch_mae <- grad_descent(x, y, 0, 0, gradient_step_mae,
                            batch_gradient_descent, alpha, iteration)[2]
  batch_huber <- grad_descent(x, y, 0, 0, gradient_step_huber,
                              batch_gradient_descent, alpha, iteration)[2]

  return(c(theta_1_analytical, batch_mae, batch_huber))
}
```

Repeating 1000 times

```r
esti_slopes <- replicate(1000, slopes())

df <- data.frame(
  slope_Estimate = c(esti_slopes[1,], esti_slopes[2,], esti_slopes[3,]),
  type = c(rep("Analytical Solution", iteration),
           rep("MAE - Batch GD", iteration),
           rep("Huber - Batch GD", iteration))
)

# Plotting histograms
ggplot(df, aes(x = slope_Estimate, fill = type)) +
  geom_histogram(position = position_dodge(width = 0.5), alpha = 0.75, bins = 35) +
  geom_vline(xintercept = 2, color = "darkblue", linetype = "dotdash") +
  facet_wrap(~type)+
  labs(x = "Slope Estimate", y = "Frequency",
       title = "Histograms of Estimated slopes by simulating outliers ") +
  geom_text(aes(x = 2, y = 80, label = "True Slope Value"),position = position_nudge(y = 5),
          vjust = -1, hjust = 0.5, color = "darkblue", angle = 90, size = 3)+
  theme_minimal()
```



The histogram illustrates different variability in slope estimates among the algorithms by using simulated outliers.

Analytical solution: This shows a wider distribution compared to the Batch gradient descent using Huber loss and Batch gradient descent using Mean absolute error. It has less accuracy as well.

Batch gradient descent using Huber Loss and Mean absolute error: These are known for their durability to outliers. These have a similar graph but, MAE has a bit wider compared to Huber loss. These show tighter distributions, indicating their robustness against outliers and potentially providing more precise slope values.