Question 4. BGD vs. SGD

1. Copy the BGD code and convert it into a Stochastic Gradient Descent algorithm by applying necessary changes inside. Name the code SGD.

```
BGD <- function(x, y, theta0, alpha = 0.01, epsilon = 1e-8, max_iter=25000){

  # Inputs
  # x      : The input variables (M columns)
  # y      : Output variables    (1 column)
  # theta0 : Initial weight vector (M+1 columns)

  x     <- as.matrix(x)
  y     <- as.matrix(y)
  N     <- nrow(x)
  i     <- 0
  theta <- theta0
  x     <- cbind(1, x) # Adding 1 as first column for intercept
  imprv <- 1e10
  cost  <- (1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y)
  delta <- 1
  while(imprv > epsilon & i < max_iter){cost
    i <- i + 1
    grad <- 0
    for(j in 1:length(y)){
      grad_chng <- x[j, ] * c(y[j]-x[j, ] %*% theta)
      grad <- grad + grad_chng
    }
    theta <- theta + (alpha / N) * grad
    cost  <- append(cost, (1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y))
    imprv <- abs(cost[i+1] - cost[i])
    if((cost[i+1] - cost[i]) > 0) stop("Cost is increasing. Try reducing alpha.")
  }
  print(paste0("Stopped in ", i, " iterations"))

  cost <- cost[-1]
  return(list(theta,cost))
}
```

```
SGD <- function(x, y, theta0, alpha = 0.01, epsilon = 1e-8, max_iter=25000){

  # Inputs
  # x      : The input variables (M columns)
  # y      : Output variables    (1 column)
  # theta0 : Initial weight vector (M+1 columns)

  x     <- as.matrix(x)
  y     <- as.matrix(y)
  N     <- nrow(x)
  i     <- 0
```

```
  theta <- theta0
  x     <- cbind(1, x) # Adding 1 as first column for intercept
  imprv <- 1e10
  cost  <- (1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y)
  delta <- 1
  while(i < max_iter){cost
    i <- i + 1
    grad <- 0
    new_cost <- 0
    for(j in 1:length(y)){
      random <- runif(1,min=1,max=length(y))
      grad_chng <- x[random, ] * c(y[random]-x[random, ] %*% theta)
      grad <- grad + grad_chng
      theta <- theta + (alpha / N) * grad
      new_cost <- sum((1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y))
    }
    cost  <- append(cost, new_cost)
    imprv <- abs(cost[i+1] - cost[i])
  }
  print(paste0("Stopped in ", i, " iterations"))

  cost <- cost[-1]
  return(list(theta,cost))
}
```

2. Compare BGD and SGD.

- Which algorithm yielded minimum loss?

The Batch Gradient Algorithm yielded minimum loss when comparing the results of 10 iterations (epochs). The BGD converged to the minima much quicker than SGD after 10 iterations. Given the fact that the training data set is small, and that SGD uses sum of all the data set to determine the next most optimal step, the step sizes tend to be bigger and are more optimized per epoch than that of SGD. Furthermore, BGD approach always strives for convergence towards the minima; while SGD fluctuates around the minima due to the existence of oscillations.

From a pure SGD algorithm standpoint, there are more iterations required to converge to the optimal loss of 0 because the step sizes for SGD is smaller by nature as it is calculated for every training data point, as opposed to a training data set of BGD.

- Plot the resulting losses. Does the resulting losses differ?

The following graph represents the comparison of BGD algorithm (black) vs. SGD algorithm (orange) at 10 iterations:

```
res <- BGD(x, y, c(1, -1), alpha = 0.005, epsilon = 1e-5, max_iter = 10)
```

```
## [1] "Stopped in 10 iterations"
```

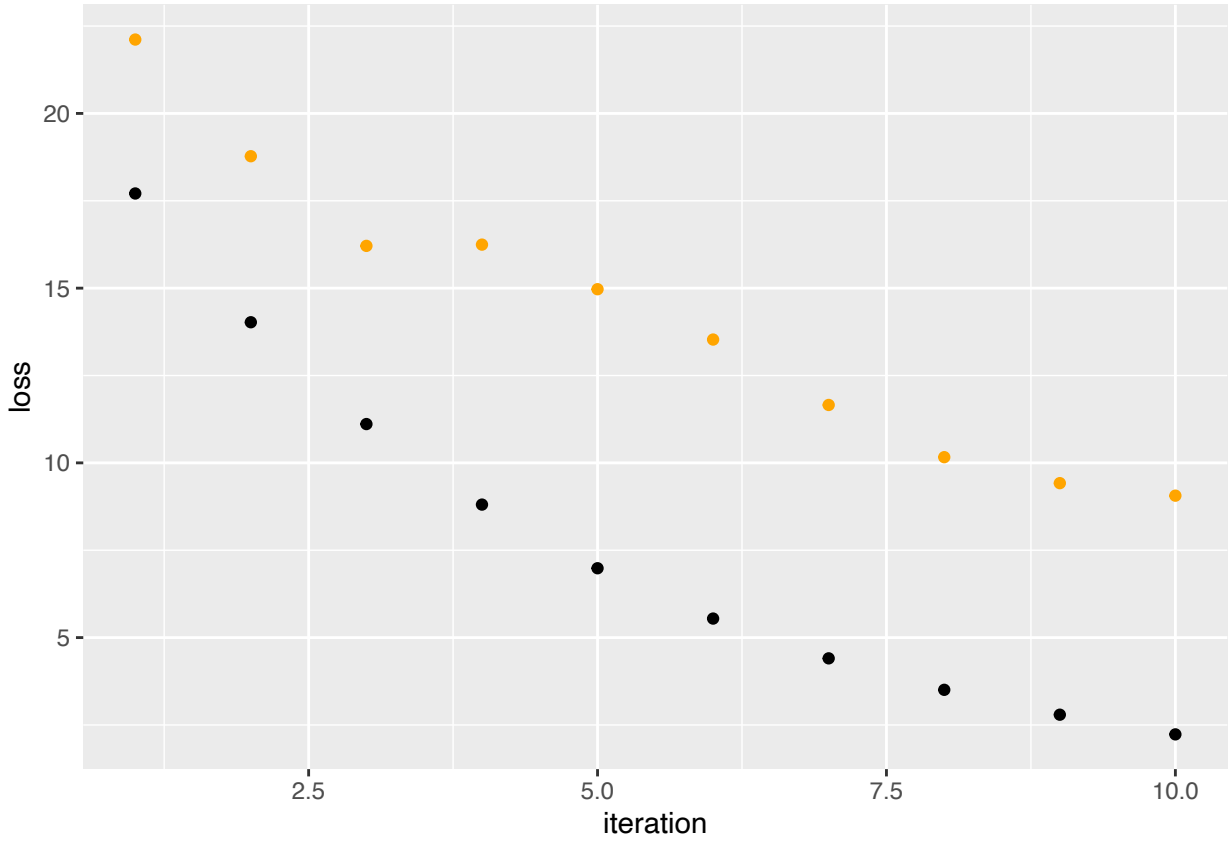
```
res2 <- SGD(x, y, c(1, -1), alpha = 0.005, epsilon = 1e-5, max_iter = 10)
```

```
## [1] "Stopped in 10 iterations"
```

```
theta_bgd <- res[[1]]
loss_bgd  <- res[[2]]

theta_sgd <- res2[[1]]
loss_sgd <- res2[[2]]

ggplot() +
    geom_point(aes(x=1:length(loss_bgd), y=loss_bgd)) +
    geom_point(aes(x=1:length(loss_sgd), y=loss_sgd), color="orange") +
    labs(x='iteration') + labs(y='loss')
```



Here, we can see that the BGD approach actually converges to the minima much quicker than SGD, due to the fact that the step sizes are bigger per iteration.

The following graph provides a more "zoomed-out" view of the comparison, where we compare BGD algorithm (black) vs. SGD algorithm (orange) at 60 iterations:

```
res <- BGD(x, y, c(1, -1), alpha = 0.005, epsilon = 1e-5, max_iter = 60)
```

```
## [1] "Stopped in 60 iterations"
```

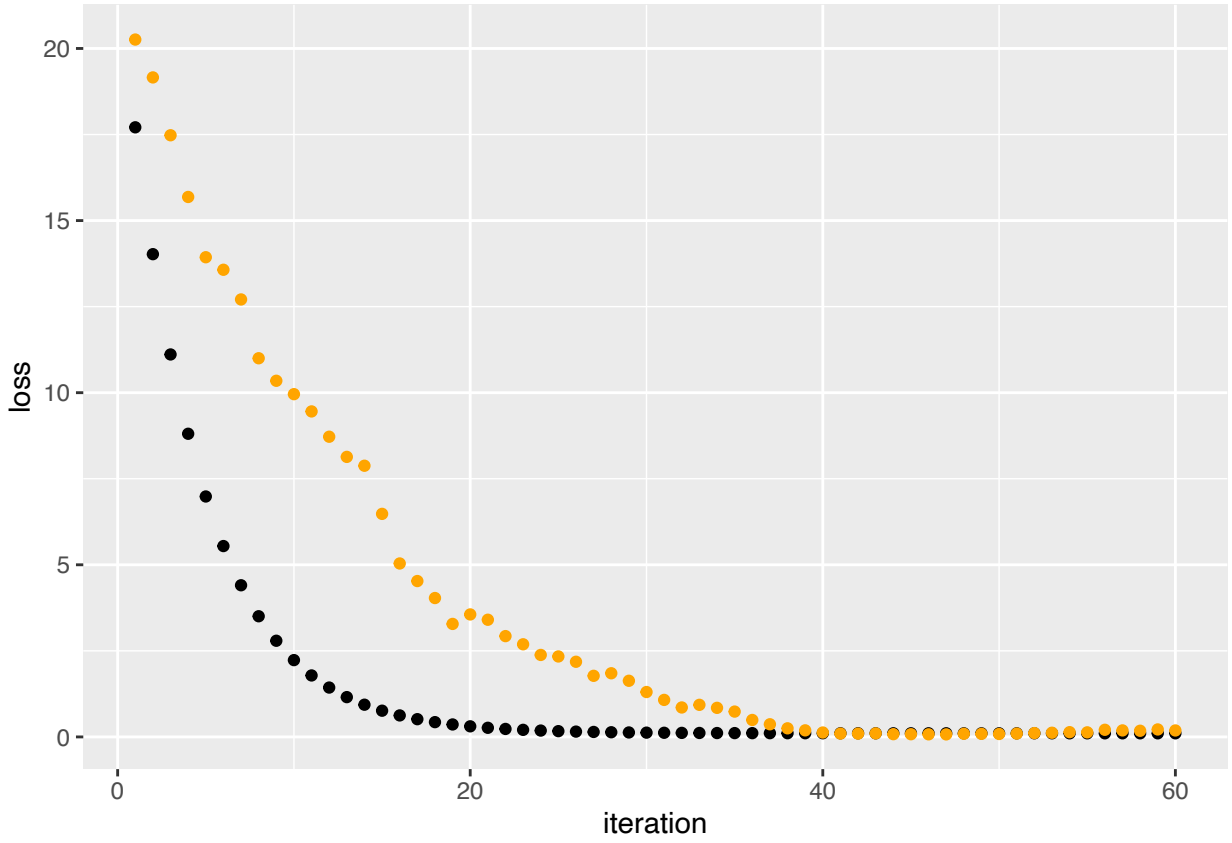
```
res2 <- SGD(x, y, c(1, -1), alpha = 0.005, epsilon = 1e-5, max_iter = 60)
```

```
## [1] "Stopped in 60 iterations"
```

```
theta_bgd <- res[[1]]
loss_bgd  <- res[[2]]

theta_sgd <- res2[[1]]
loss_sgd <- res2[[2]]

ggplot() +
    geom_point(aes(x=1:length(loss_bgd), y=loss_bgd)) +
    geom_point(aes(x=1:length(loss_sgd), y=loss_sgd), color="orange") +
    labs(x='iteration') + labs(y='loss')
```



Here, we can we see the nature of "noisiness" of SGD, as a random data point is used to determine the subsequent step sizes. Although SGD algorithm converges to the minima, it stills shows signs of oscillations due to the stochistic nature of the algorithm.