Regression_Implementation

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
```

For this demonstration we will use the iris dataset

```r
X = iris[,1:2]
y = iris[,3]

# Get maximum and minimum values
maxs <- apply(as.matrix(X), 2, function(x) max(x))
mins <- apply(as.matrix(X), 2, function(x) min(x))

# Normalize Xs for faster convergence
X <- apply(as.matrix(X), 2, function(x) (x - min(x))/(max(x) - min(x)))

# Combine normalized X and y into a dataset
test_frame <- cbind(X, as.matrix(y))

# Add a column of 1s (to be used in obtaining the intercept coefficient)
X <- cbind(rep(1, length(y)), X)

# Cost function
cost <- function(theta){
  preds <- theta %*% t(X)
  resids <- preds - y
  SSE <- sum(resids^2)
  return(SSE/(2*length(y)))
}

# Gradient of cost function
grad <- function(theta){
  preds <- theta %*% t(X)
  resids <- preds - y
  return((1/length(y))*apply(X, 2, function(x) sum(x*resids)))
}

# Initialize thetas
theta <- rep(0, ncol(X))

# Create empty data frame to log thetas
theta_df <- data.frame(matrix(rep(0, ncol(X)*1000), nrow = 1000))
```

```r
names(theta_df) <- c("Intercept", "Sepal.Length", "Sepal.Width")

# Initialize a vector containing the cost
cost_vec <- rep(0, 1000)

# Iterate through the data 1000 times using batch GD
for(i in 1:1000){
  cost_vec[i] <- cost(theta)
  theta <- theta - .1*grad(theta)
  theta_df[i,] <- theta
}

print(theta)
```

```
##               Sepal.Length  Sepal.Width
##     2.361849     6.406516    -3.068555
```

```r
# To test your result
test_frame <- as.data.frame(test_frame)
names(test_frame) <- c("Sepal.Length", "Sepal.Width", "Petal.Length")
glm(Petal.Length ~ ., data=test_frame)
```

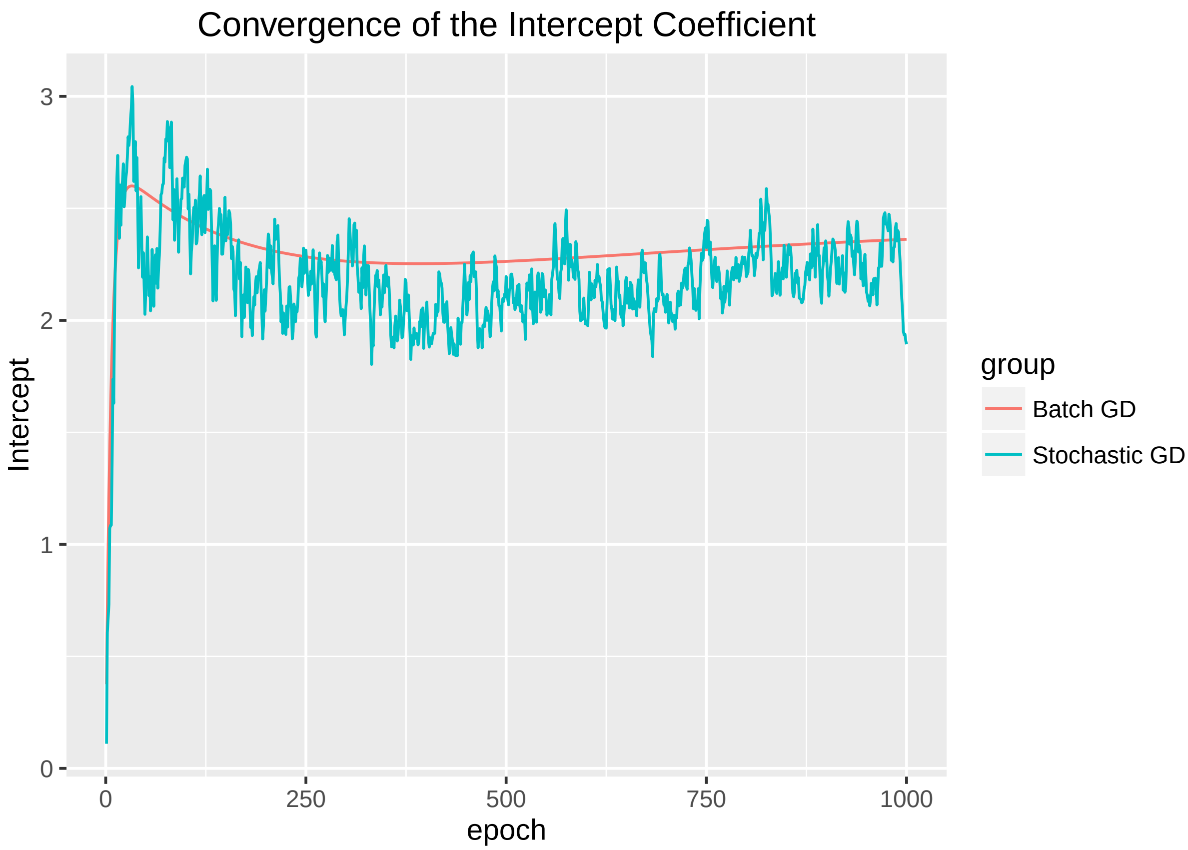
```
##
## Call:  glm(formula = Petal.Length ~ ., data = test_frame)
##
## Coefficients:
##  (Intercept)  Sepal.Length   Sepal.Width
##        2.433         6.392        -3.213
##
## Degrees of Freedom: 149 Total (i.e. Null);  147 Residual
## Null Deviance:       464.3
## Residual Deviance: 61.44     AIC: 299.8
```

```r
# We can see our algorithm converged correctly

theta_s <- rep(0, ncol(X))
theta_df_s <- data.frame(matrix(rep(0, ncol(X)*1000), nrow = 1000))
names(theta_df_s) <- c("Intercept", "Sepal.Length", "Sepal.Width")

cost_vec_stoch <- rep(0, 1000)

df_s <- cbind(rep(1, length(y)), test_frame)
names(df_s)[1] <- "intercept"

# Now try with stochastic
for(i in 1:1000){
  # Take a sample from our data
  samp <- df_s[sample(nrow(df_s), 1),]
  x <- samp[,1:ncol(samp)-1]
  y_samp <- samp[,ncol(samp)]
  # Get prediction with current theta
  pred <- theta_s %*% t(x)
  resid <- pred - y_samp
  gradient <- apply(x, 2, function(x) sum(x*resid))
  theta_s <- theta_s - .1*gradient
```

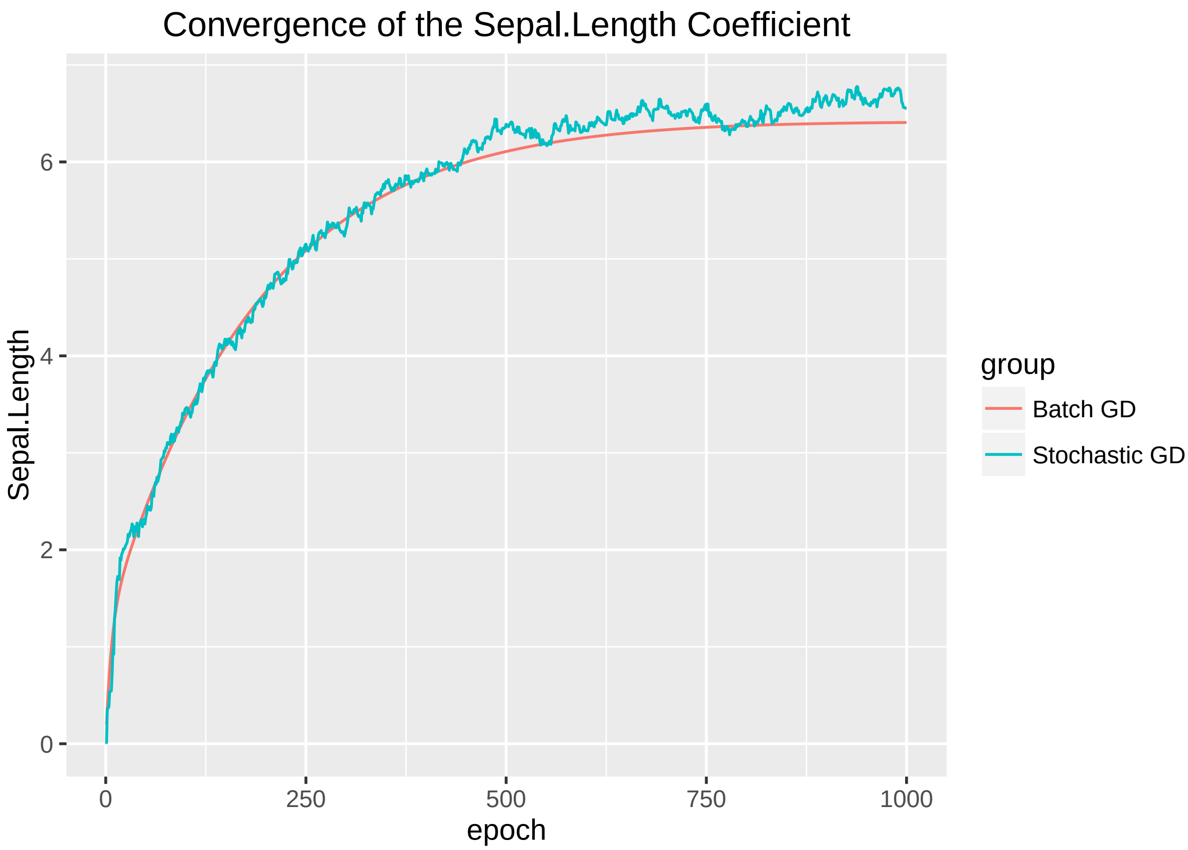
```r
  cost_vec_stoch[i] <- cost(theta_s)
  theta_df_s[i,] <- theta_s
}

print(theta_s)
```

```
##    intercept Sepal.Length  Sepal.Width
##     1.893521     6.549885    -3.050596
```

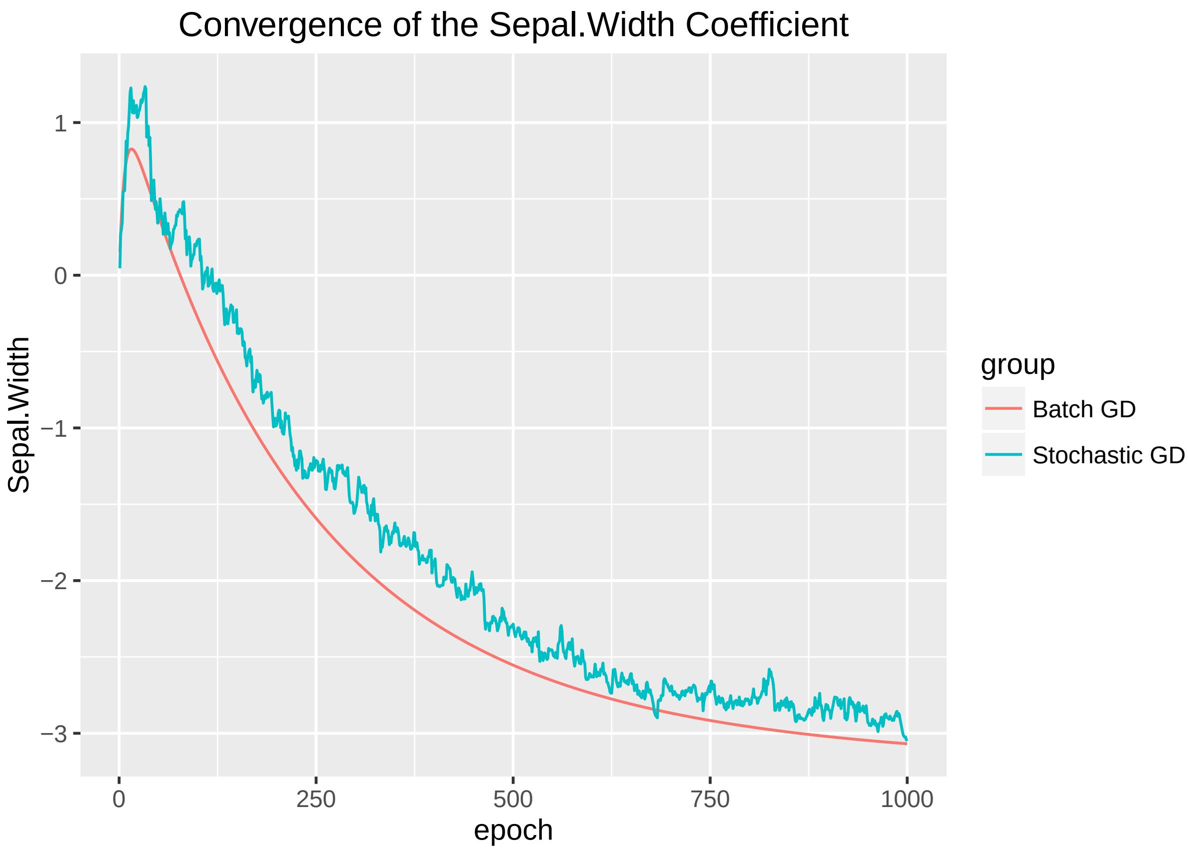
```r
# We can see that we are close to the correct values, but not entirely there by 1000
# iterations

epoch = 1:1000
cost_df <- data.frame(cost_vec, epoch)
cost_df$group <- "Batch GD"
cost_df_s <- data.frame(cost_vec_stoch, epoch)
cost_df_s$group <- "Stochastic GD"
names(cost_df_s)[1] <- "cost_vec"
combined_cost <- rbind(cost_df, cost_df_s)

theta_df$group <- rep("Batch GD", nrow(theta_df))
theta_df$epoch <- 1:1000
theta_df_s$group <- rep("Stochastic GD", nrow(theta_df_s))
theta_df_s$epoch <- 1:1000
coefficient_df <- rbind(theta_df, theta_df_s)

# Let's compare how our coefficients converged for the two methods
ggplot(coefficient_df, aes(x = epoch, y = Intercept, colour = group)) +
  geom_line() + ggtitle("Convergence of the Intercept Coefficient")
```



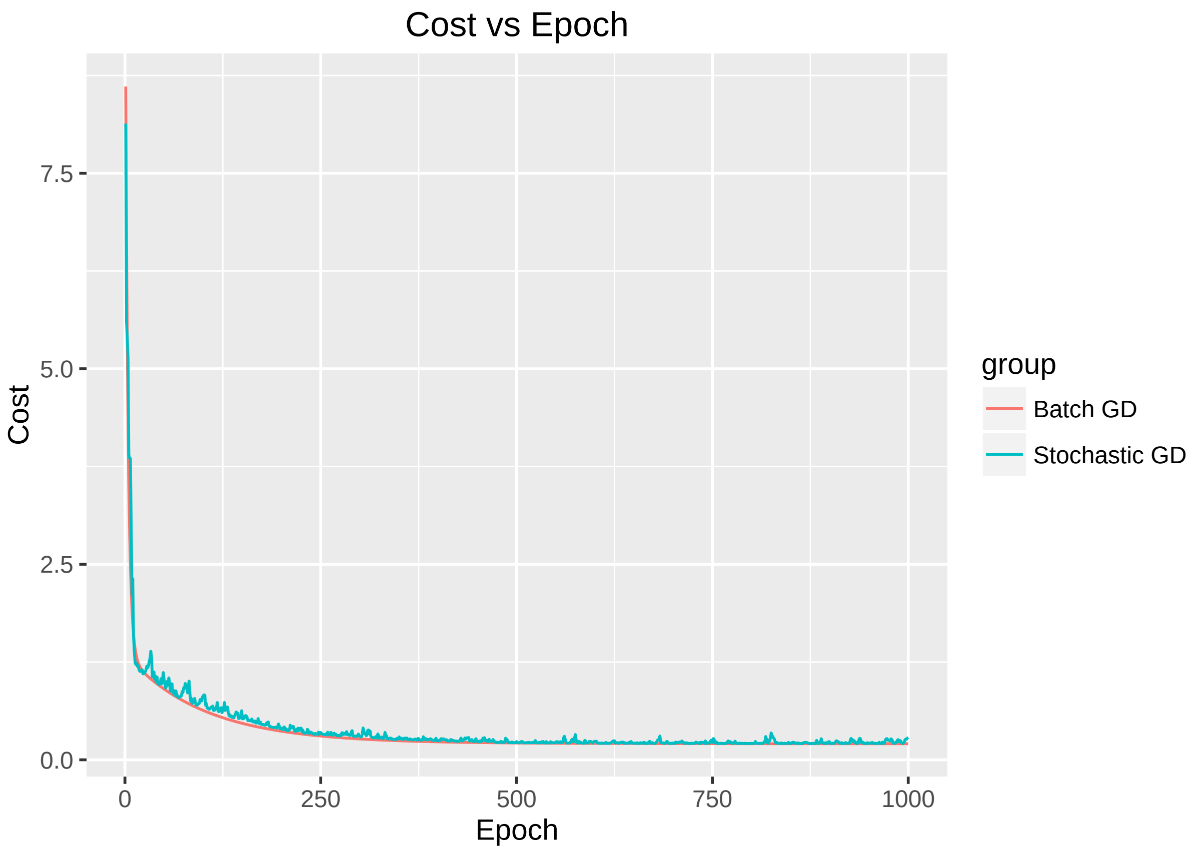

```r
ggplot(coefficient_df, aes(x = epoch, y = Sepal.Length, colour = group)) +
  geom_line() + ggtitle("Convergence of the Sepal.Length Coefficient")
```



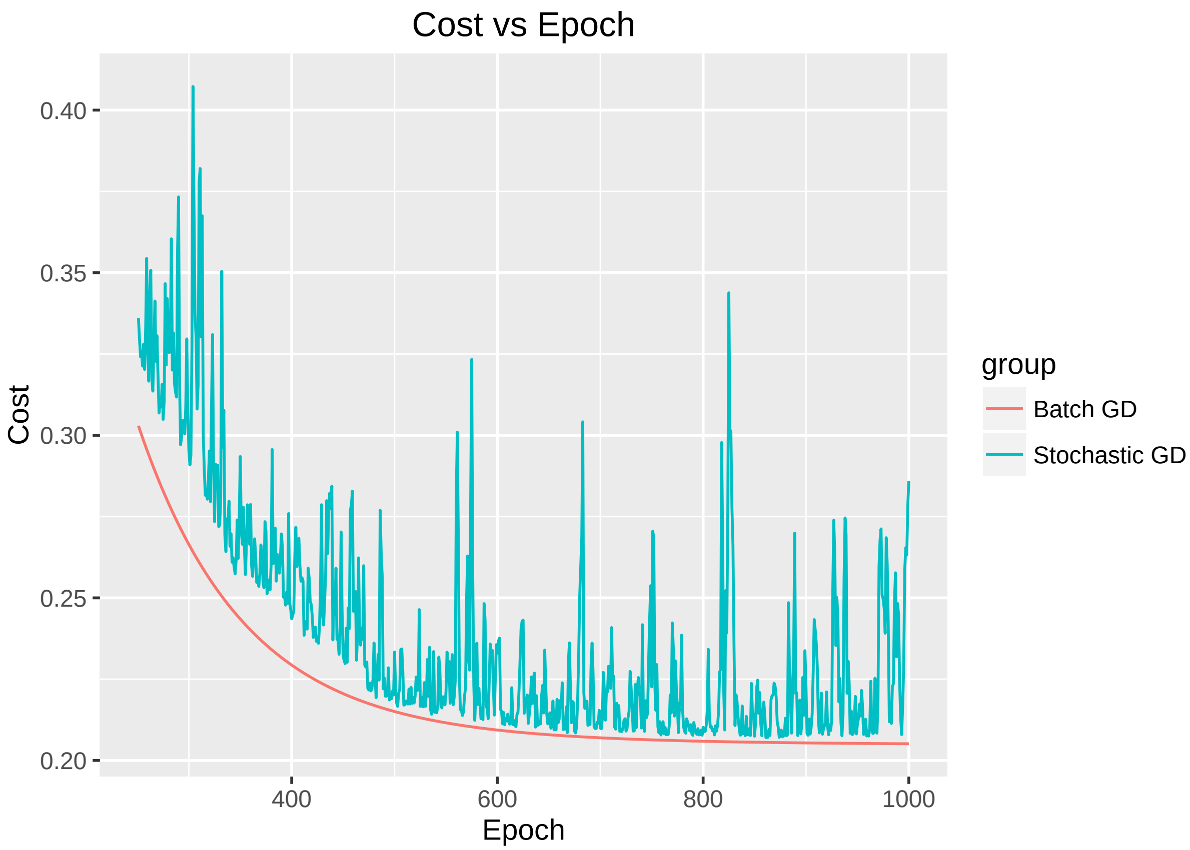

```
ggplot(coefficient_df, aes(x = epoch, y = Sepal.Width, colour = group)) +
  geom_line() + ggtitle("Convergence of the Sepal.Width Coefficient")
```




```
# We can see that both methods converged relatively well by 1000 epochs. However,
# due to variation from one observation to the next, convergence using stochastic
# gradient descent is less smooth

ggplot(combined_cost, aes(x = epoch, y = cost_vec, colour = group)) +
  geom_line() + xlab("Epoch") + ylab("Cost") + ggtitle("Cost vs Epoch")
```




```r
# At first glance, the cost values between the two methods seem comparable
# However, if we zoom in a bit, we can see that the cost for stochastic GD
# tends to be higher than that for batch gradient descent (and more unstable)

combined_cost2 <- combined_cost %>% filter(epoch > 250)
ggplot(combined_cost2, aes(x = epoch, y = cost_vec, colour = group)) +
  geom_line() + xlab("Epoch") + ylab("Cost") + ggtitle("Cost vs Epoch")
```


Cost vs Epoch
Cost
0.40
0.35
0.30
0.25
0.20
400
600
800
1000
Epoch
group
Batch GD
Stochastic GD