homework-3

Name: Brian Deng

```
library(bis557)
```

Name: Brian Deng (BIS557 HW3)

Question 1

We have the equation:

$$H(l) = X^T D X,$$

where $D_{i,i} = p_i(1 - p_i)$.

We know that $D = I$ (identity matrix) for the linear Hessian.

To make the logistic Hessian matrix ill-conditioned, we want to set many of the **diagonal elements to be infinitesimally close to 0**, so that:

$$\exists i : D_{i,i} \to 0.$$

This means that **most of the probabilities should be very close to 0 or 1** ($p_i \to 0$ or $p_i \to 1$).

Example

Let's create a 120×8 matrix using the rng `set.seed(2020)`. To make the probabilities really close to 0 or 1, let $\beta = (200, 200, ..., 200)$. We will check the **condition number** κ for the linear Hessian and the logistic Hessian. A high condition number indicates ill-condition.

```
# Dimensions
set.seed(2020)
n <- 120; p <- 8

# Matrix X, Coefficient betas, and Probabilities
beta <- rep(200, p)
X <- cbind(1, matrix(data = rnorm(n * (p - 1)), nrow = n, ncol = p - 1))
probs <- 1 / (1 + exp(-X %*% beta)) # logistic
print(quantile(probs, c(0.1, 0.25, 0.5, 0.75, 0.9))) # mostly near 0 or 1
#>           10%           25%           50%           75%           90%
#> 5.643474e-241  4.690671e-95  1.000000e+00  1.000000e+00  1.000000e+00

# Linear Hessian and Logistic Hessian
print(Hessian_linear <- kappa(t(X) %*% X)) # Answer: 2.33
#> [1] 2.329629
D <- diag(as.vector(probs) * (1 - as.vector(probs)), nrow = n, ncol = n)
```

```r
print(Hessian_logis <- kappa(t(X) %*% D %*% X)) # Answer: 3.01e+13
#> [1] 3.006434e+13
```

Here, we see that $\kappa(X^tX) = 2.33$ (low condition number) and $\kappa(X^tDX) = 3.01 \times 10^{13}$ (high condition number).

Thus, the **linear Hessian is well-conditioned** but the **logistic Hessian is ill-conditioned**.

Question 2

The function will be called `bis557::glm_hw3b()`. Here, we use only the first-order condition to solve coefficients from a **generalized** linear model (GLM) using the gradient descent algorithm.

For each observation i, we let:

$$\mathbb{E}[y_i] = g^{-1}(\langle x_i, \beta \rangle),$$

where g is the link function.

The **gradient/score** of the likelihood function l is:

$$\bigtriangledown_\beta l(\beta) = X^T(y - \mathbb{E}[y]).$$

The gradient descent algorithm tells us to update β so that:

$$\beta \leftarrow \beta - \alpha \bigtriangledown_\beta l(\beta),$$

where α is the step size.

The example below uses GLM to solve coefficients with a Poisson-distributed response variable, first using a **constant** step size.

```r
# Givens (with constant step size)
set.seed(2020)
n <- 3000; p <- 4; maxiter <- 500; steps <- rep(1e-3, maxiter)

# Create covariates X, response variable y, and the true beta coefficients
X <- cbind(1, matrix(rnorm(n * (p-1)), ncol = p-1))
beta <- c(-1, 0.2, 0.1, 0.3)
y <- rpois(n, lambda = exp(X %*% beta))

# Use GLM to solve for coefficients
b_hat <- glm_hw3b(X, y, family = poisson(link = "log"), steps, maxiter)

# Compare true vs estimated coefficients
print(cbind(beta, b_hat$coefficients))
#>      beta
#> [1,] -1.0 -1.04550072
#> [2,]  0.2  0.20388280
#> [3,]  0.1  0.07509437
#> [4,]  0.3  0.29112418
```

The estimated coefficients $\hat{\beta}$ are very close to the actual coefficients β for constant step size.

Second, we will use an **adaptive** step size $\alpha_t = O\left(\frac{1}{t}\right)$, for the same data.

```r
# Adaptive Step Size
set.seed(2020)
steps2 <- 5e-3/(1:maxiter)

# Use GLM to solve for coefficients
b_hat2 <- glm_hw3b(X, y, family = poisson(link = "log"),
                   steps = steps2, maxiter)

# Compare true vs estimated coefficients
print(cbind(beta, b_hat2$coefficients))
#>      beta
#> [1,] -1.0 -1.04552307
#> [2,]  0.2  0.20389155
#> [3,]  0.1  0.07509649
#> [4,]  0.3  0.29113946
```

The estimated coefficients $\hat{\beta}$ are also very close to the actual coefficients β for adaptive step size.

Therefore, the performance between a constant step size and an adaptive step size is **very similar**, and performs really well depending on the user's choice of step size.

Question 3

The function will be called `bis557::multiclass_hw3c()`. This is the generalized logistic regression to predict **multiple** classes. The approach used is the *one-vs-all* approach, where for each of the K classes, a **separate** logistic regression model is deployed (coefficients and probabilities), where the model predicts whether the observation is in class k or not (for $k = \{1, 2, ..., K\}$).
We use the Newton-Raphson method (second-order). Then, for each observation, the class with the **highest probability** is chosen.

The example below predicts the species of the iris using this function with **96%** accuracy. Notice that we can use *fewer* iterations due to the second-order condition (compared to Question 2).

```r
data(iris)
species_pred <- multiclass_hw3c(X = iris[,-5], y = iris$Species, maxiter = 60)
print(paste0("Prediction Accuracy = ", mean(iris$Species == species_pred)))
#> [1] "Prediction Accuracy = 0.96"
```