Homework 2

Context

This assignment reinforces ideas in Module 2: Optimization. We focus specifically on implementing the Newton's method, EM, and MM algorithms.

Due date and submission

Please submit (via Canvas) a PDF containing a link to the web address of the GitHub repo containing your work for this assignment; git commits after the due date will cause the assignment to be considered late. Due date is Wednesday, 2/19 at 10:00AM.

Points

Problem	Points
Problem 0	15
Problem 1	30
Problem 2	5
Problem 3	30
Problem 4	20

Problem 0

This "problem" focuses on structure of your submission, especially the use git and GitHub for reproducibility, R Projects to organize your work, R Markdown to write reproducible reports, relative paths to load data from local files, and reasonable naming structures for your files.

To that end:

- create a public GitHub repo + local R Project; I suggest naming this repo / directory bios731_hw2_YourLastName (e.g. bios731_hw2_wrobel for Julia)
- Submit your whole project folder to GitHub
- Submit a PDF knitted from Rmd to Canvas. Your solutions to the problems here should be implemented in your .Rmd file, and your git commit history should reflect the process you used to solve these Problems.

Algorithms for logistic regression

For a given subject in a study, we are interested in modeling $\pi_i = P(Y_i = 1|X_i = x_i)$, where $Y_i \in \{0, 1\}$. The logistic regression model takes the form

$$\text{logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \log\left(\frac{P(Y_i = 1|X_i)}{1 - P(Y_i = 1|X_i)}\right) = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \ldots + \beta_p X_{pi}$$

- $Y_1, Y_2, \ldots, Y_n \sim Bernoulli(\pi)$
- PDF is $f(y_i; \pi) = \pi^{y_i}(1-\pi)^{1-y_i}$

Problem 1: Newton's method

- Derive likelihood, gradient, and Hessian for logistic regression for an arbitrary number of predictors p

Answer: Based on the PDF given, we have:

$$l(\boldsymbol{\beta}|\pi, Y) = \sum_i (Y_i \log(\pi_i) + (1 - Y_i) \log(1 - \pi_i))$$

Also, based on the logistic regression model we have:

$$\pi_i = \frac{e^{X_i^\top \beta}}{1 + e^{X_i^\top \beta}}$$

$$l(\beta|\boldsymbol{X}, Y) = \sum_i (Y_i (X_i^\top \beta) - \log(1 + e^{X_i^\top \beta}))$$

Therefore the gradient would be:

$$U(\boldsymbol{\beta}) = \sum_i (Y_i - \frac{e^{X_i^\top \beta}}{1 + e^{X_i^\top \beta}}) X_i$$

The Hessian would be:

$$H(\boldsymbol{\beta}) = -\sum_i \frac{e^{X_i^\top \beta}}{(1 + e^{X_i^\top \beta})^2} X_i X_i^\top$$

- What is the Newton's method update for β for logistic regression?

Answer: Based on the Score and Hessian calculated, the Newton's method update is:

$$\beta_{t+1} = \beta_t - \{H(\beta_t)\}^{-1} U(\beta_t) = \beta_t + \{\sum_i \frac{e^{X_i^\top \beta}}{(1 + e^{X_i^\top \beta})^2} X_i X_i^\top\}^{-1} \sum_i (Y_i - \frac{e^{X_i^\top \beta}}{1 + e^{X_i^\top \beta}}) X_i$$

- Is logistic regression a convex optimization problem? Why or why not?

Answer: No, because the Hessian is always smaller than 0, which makes it concave.

Problem 2: MM

(A) In constructing a minorizing function, first prove the inequality

$$-\log\{1 + \exp x_i^T \theta\} \geq -\log\{1 + \exp(X_i^T \theta^{(k)})\} - \frac{\exp(X_i^T \theta) - \exp(X_i^T \theta^{(k)})}{1 + \exp(X_i^T \theta^{(k)})}$$

with equality when $\theta = \theta^{(k)}$. This eliminates the log terms.

Answer: Organize the terms on both sides of the inequality, it's equivalent to:

$$\frac{1 + \exp(X_i^T \theta)}{1 + \exp(X_i^T \theta^{(k)})} - 1 \geq \log\{\frac{1 + \exp(X_i^T \theta)}{1 + \exp(X_i^T \theta^{(k)})}\}$$

Look $\frac{1+\exp(X_i^T \theta)}{1+\exp(X_i^T \theta^{(k)})}$ as a variable range $(0, +\infty)$, and this inequality is $x - 1 \geq \log(x)$, which is hold when $x \in (0, +\infty)$ and take equality when $x = 1$, because it is derivated as $1 - \frac{1}{x}$.

(B) Now apply the arithmetic-geometric mean inequality to the exponential function $\exp(X_i^T \theta)$ to separate the parameters. Assuming that θ has p components and that there are n observations, show that these maneuvers lead to a minorizing function

$$g(\theta|\theta^{(k)}) = -\frac{1}{p}\sum_{i=1}^{n}\frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}\sum_{j=1}^{p}\exp\{pX_{ij}(\theta_j-\theta_j^{(k)})\} + \sum_{i=1}^{n}Y_iX_i^T\theta = 0$$

up to a constant that does not depend on θ.

Answer: Considering the arithmetic-geometric mean inequality, we have:

$$\frac{\exp(X_i^T\theta)}{\exp(X_i^T\theta^{(k)})} = \prod_{j=1}^{p}\exp\{X_{ij}(\theta_j-\theta_j^{(k)})\} = \sqrt[p]{\prod_{j=1}^{p}\exp\{pX_{ij}(\theta_j-\theta_j^{(k)})\}} \le \frac{1}{p}\sum_{j=1}^{p}\exp\{pX_{ij}(\theta_j-\theta_j^{(k)})\}$$

Combing the 2 inequalities, we have a minorizing function for our likelihood function for logistric regression above:

$$\begin{aligned}
l(\theta|\boldsymbol{X},Y) &= \sum_i(Y_i(X_i^\top\theta) - \log(1+e^{X_i^\top\theta})) \ge \sum_i(Y_i(X_i^\top\theta) - \log\{1+\exp(X_i^T\theta^{(k)})\} - \frac{\exp(X_i^T\theta)-\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}) \\
&= \sum_i(Y_i(X_i^\top\theta) - \log\{1+\exp(X_i^T\theta^{(k)})\} - \frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}(\frac{\exp(X_i^T\theta)}{\exp(X_i^T\theta^{(k)})}-1)) \\
&\ge \sum_i(Y_i(X_i^\top\theta) - \frac{1}{p}\frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}\sum_{j=1}^{p}\exp\{pX_{ij}(\theta_j-\theta_j^{(k)})\} - \log\{1+\exp(X_i^T\theta^{(k)})\} + \frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}) \\
&= g(\theta|\theta^{(k)}) - \sum_i(\log\{1+\exp(X_i^T\theta^{(k)})\} + \frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})})
\end{aligned}$$

(C) Finally, prove that maximizing $g(\theta|\theta^{(k)})$ consists of solving the equation for each j.

$$-\sum_{i=1}^{n}\frac{\exp(X_i^T\theta^{(k)})X_{ij}\exp(-pX_{ij}\theta_j^{(k)})}{1+\exp(X_i^T\theta^{(k)})}\exp(pX_{ij}\theta_j) + \sum_{i=1}^{n}Y_iX_{ij} = 0$$

Answer: Because in $g(\theta|\theta^{(k)})$ we have eliminated the $f(\theta_i,\theta_j)$ terms, which indicate that we can maximize $g(\theta|\theta^{(k)})$ by maximizing each components of θ, without worrying their maximization depending on each other.

To maximize each components of $g(\theta|\theta^{(k)})$ we can take derivatives and get the value make it equal to 0:

$$\begin{aligned}
\frac{\partial g(\theta|\theta^{(k)})}{\partial\theta_j} &= -\frac{1}{p}\sum_{i=1}^{n}\frac{\exp(X_i^T\theta^{(k)})}{1+\exp(X_i^T\theta^{(k)})}\exp\{pX_{ij}(\theta_j-\theta_j^{(k)})\}\cdot pX_{ij} + \sum_{i=1}^{n}Y_iX_{ij} \\
&= -\sum_{i=1}^{n}\frac{\exp(X_i^T\theta^{(k)})X_{ij}\exp(-pX_{ij}\theta_j^{(k)})}{1+\exp(X_i^T\theta^{(k)})}\exp(pX_{ij}\theta_j) + \sum_{i=1}^{n}Y_iX_{ij} = 0
\end{aligned}$$

Note: there's no closed form solution for β_j with $j \neq 0$, in the following implementation part I would use Newton method to get the solution for the equation above.

Problem 3: simulation

Next we will implement logistic regression in R four different ways and compare the results using a short simulation study.

Simulation study specification:

- simulate from the model $logit(P(Y_i = 1|X_i)) = \beta_0 + \beta_1 X_i$
  - $\beta_0 = 1$
  - $\beta_1 = 0.3$
  - $X_i \sim N(0, 1)$
  - $n = 200$
  - $nsim = 1$
- For your implementation of MM and Newton's method, select your own starting value and stopping criterion, but make sure they are the same for the two algorithms

```r
# simulation
set.seed(2025)
n = 200
beta0 = 1
beta1 = 0.3
x = rnorm(n)
p = exp(beta0 + beta1 * x)/(1+exp(beta0 + beta1 * x))
X_mtx = cbind(1, x)
y = rbinom(n, 1, p)
```

- implement using Newton's method from 1.1 in R

```r
newton_logit = function(X, y, beta0 = c(0,0), tol = 1e-6, max_iter = 100) {

  beta_cur = beta0
  beta_history = gradient_vec = matrix(NA, nrow = max_iter,
                                       ncol = length(beta0))
  for (iter in 1:max_iter) {

    # store results
    beta_history[iter,] = beta_cur

    # Compute the gradient and hessian
    gradient = as.numeric(t(y-exp(X%*%beta_cur)/(1+exp(X%*%beta_cur)))%*%X)

    hessian_ls = as.list(rep(NA, length.out = length(y)))
    for(i in 1:length(y)){
      hessian_ls[[i]] = as.numeric(exp(X[i,]%*%beta_cur)/(1+exp(X[i,]%*%beta_cur))^2) * tcrossprod(X[i,]
    }

    hessian <- -1 * Reduce("+", hessian_ls)

    gradient_vec[iter,] = gradient

    # Check stopping criterion
    if(sqrt(sum(gradient^2)) < tol){
      message("Converged in", iter, "iterations.\n")
      break
    }

    # Update the solution
    beta_cur = beta_cur - solve(hessian) %*% gradient
  }
  hessian_ls = as.list(rep(NA, length.out = length(y)))
  for(i in 1:length(y)){
    hessian_ls[[i]] = as.numeric(exp(X[i,]%*%beta_cur)/(1+exp(X[i,]%*%beta_cur))^2) * tcrossprod(X[i,],
```

```
  }

  var <- solve(Reduce("+", hessian_ls))

  return(list(solution = beta_cur,
              beta_history = beta_history,
              gradient = gradient_vec,
              var = var,
              converged = (iter < max_iter),
              niter = iter))
}

fit.newton<-newton_logit(X_mtx, y)
```

```
## Converged in5iterations.
```

- implement using MM from 1.2 in R

```
mm_logit = function(X, y, beta0 = c(0,0), tol = 1e-6, max_iter = 200) {

  beta_cur = beta0
  beta_history = gradient_vec = matrix(NA, nrow = max_iter,
                                       ncol = length(beta0))
  inner_iter <- 0
  for (iter in 1:max_iter) {

    # store results
    beta_history[iter,] = beta_cur
    eta <- X %*% beta_cur
    mu <- exp(eta) / (1 + exp(eta))

    # set a value for beta_0 because X_i1 = 1 all the time
    beta_0 = log(sum(y)/sum((exp(eta)* exp(-2 *beta_cur[1]))/(1 + exp(eta))))/2

    # Newton method to calculate beta_1
    inner_diff <- Inf
    beta_1 = beta_cur[2]

    while (inner_diff > tol) {
      exp_term = exp(-2 * X[, 2] * beta_cur[2])
      denom = sum((mu * X[, 2] * exp_term) * exp(2 * X[, 2] * beta_1))
      num = sum(y * X[, 2])

      # Compute gradient and Hessian for Newton's update
      f_beta_1 = -denom + num
      f_prime_beta_1 = -2 * sum((mu * X[, 2]^2 * exp_term) * exp(2 * X[, 2] * beta_1))

      # Newton update
      beta_1_new = beta_1 - f_beta_1 / f_prime_beta_1

      inner_diff = abs(beta_1_new - beta_1)
      beta_1 = beta_1_new
      inner_iter = inner_iter + 1
    }
```

```
    # Check stopping criterion
    if(sqrt(sum((c(beta_0,beta_1)-beta_cur)^2)) < tol){
      message("Converged in", iter, "iterations.\n")
      break
    }

    # Update the solution
    beta_cur = c(beta_0,beta_1)
  }


  # Compute Fisher information matrix
  eta <- X %*% beta_cur
  mu <- exp(eta) / (1 + exp(eta))
  W <- diag(as.vector(mu * (1 - mu)), nrow = length(y))
  fisher_info <- t(X) %*% W %*% X
  var_cov_matrix <- solve(fisher_info)

  return(list(solution = beta_cur,
              beta_history = beta_history,
              converged = (iter < max_iter),
              niter = iter,
              inner_iter = inner_iter,
              var = var_cov_matrix))

}

fit.mm<-mm_logit(X_mtx,y)
```

```
## Converged in101iterations.
```

- implement using glm() in R

```
fit.glm = glm(y~x, family = binomial(link = "logit"))
summary(fit.glm)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial(link = "logit"))
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.0348     0.1650   6.271 3.58e-10 ***
## x             0.4328     0.1690   2.561   0.0104 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 233.30  on 199  degrees of freedom
## Residual deviance: 226.43  on 198  degrees of freedom
## AIC: 230.43
##
## Number of Fisher Scoring iterations: 4
```

- implement using `optim()` in R:
  - Use the option `method = "BFGS"`, which implements a Quasi-Newton approach

```r
# Negative Log-Likelihood Function
neg_log_likelihood <- function(X, y, beta) {
  eta <- X %*% beta
  log_lik <- sum(y * eta - log(1 + exp(eta)))  # Log-likelihood
  return(-log_lik)  # Negate for minimization
}

# Use optim() to minimize the negative log-likelihood
fit.optim <- optim(par = c(0,0),
                   fn = neg_log_likelihood,
                   X = X_mtx, y = y,
                   method = "BFGS",
                   hessian = T,
                   control = list(reltol = 1e-6))
```

You only need to run the simulation using **one simulated dataset**. For each of the four methods, report:

- $\hat{\beta}_0$, $\hat{\beta}_1$
- 95% confidence intervals for $\hat{\beta}_0$, $\hat{\beta}_1$
- computation time
- number of iterations to convergence

Make 2-3 plots or tables comparing your results, and summarize these findings in one paragraph.

```r
est_beta0<-c(fit.newton$solution[1],fit.mm$solution[1],fit.glm$coefficients[1],fit.optim$par[1])
est_beta1<-c(fit.newton$solution[2],fit.mm$solution[2],fit.glm$coefficients[2],fit.optim$par[2])

se_beta0<-c(sqrt(fit.newton$var[1,1]),sqrt(fit.mm$var[1,1]),summary(fit.glm)$coefficients[1,2],sqrt(sol
se_beta1<-c(sqrt(fit.newton$var[2,2]),sqrt(fit.mm$var[2,2]),summary(fit.glm)$coefficients[2,2],sqrt(sol

niter<-c(fit.newton$niter,fit.mm$niter,fit.glm$iter,fit.optim$counts[2])

beta0_lCI<-est_beta0-1.96*se_beta0
beta0_uCI<-est_beta0+1.96*se_beta0
beta1_lCI<-est_beta1-1.96*se_beta1
beta1_uCI<-est_beta1+1.96*se_beta1

# Create a table with method names
methods <- c("Newton", "MM", "GLM", "Optim")

# Combine everything into a dataframe
results_table <- data.frame(
  Method = methods,
  Beta0 = est_beta0,
  CI_Beta0_Lower = beta0_lCI,
  CI_Beta0_Upper = beta0_uCI,
  Beta1 = est_beta1,
  CI_Beta1_Lower = beta1_lCI,
  CI_Beta1_Upper = beta1_uCI,
  Iterations = niter
)

library(knitr)
```

```r
kable(results_table, digits = 4)
```

Method	Beta0	CI_Beta0_Lower	CI_Beta0_Upper	Beta1	CI_Beta1_Lower	CI_Beta1_Upper	Iterations
Newton	1.0348	0.7114	1.3582	0.4328	0.1016	0.7641	5
MM	1.0348	0.7114	1.3582	0.4328	0.1016	0.7641	101
GLM	1.0348	0.7114	1.3582	0.4328	0.1016	0.7641	4
Optim	1.0353	0.7176	1.3530	0.4318	0.1064	0.7572	6

```r
library(microbenchmark)

# Run the benchmarks
benchmark_results <- microbenchmark(
  Newton = newton_logit(X_mtx, y),
  MM = mm_logit(X_mtx, y),
  glm = glm(y ~ x, family = binomial(link = "logit")),
  optim = optim(par = c(0,0), fn = neg_log_likelihood, X = X_mtx, y = y, method = "BFGS", hessian = T, 
  times = 10  # Number of repetitions
)
```

```
## Warning in microbenchmark(Newton = newton_logit(X_mtx, y), MM = mm_logit(X_mtx,
## : less accurate nanosecond times to avoid potential integer overflows
## Converged in5iterations.
## Converged in101iterations.
## Converged in5iterations.
## Converged in101iterations.
##
## Converged in101iterations.
## Converged in5iterations.
## Converged in101iterations.
##
## Converged in101iterations.
##
## Converged in101iterations.
##
## Converged in101iterations.
## Converged in5iterations.
## Converged in101iterations.
##
## Converged in101iterations.
##
## Converged in101iterations.
## Converged in5iterations.
##
## Converged in5iterations.
##
## Converged in5iterations.
##
## Converged in5iterations.
```

```
##
## Converged in5iterations.
##
## Converged in5iterations.
```

```
print(benchmark_results)
```

```
## Unit: microseconds
##    expr      min       lq      mean    median       uq      max neval cld
##  Newton 2941.504 2993.697 3697.9581 3130.4320 3271.472 8924.019    10  a
##      MM 2713.954 2841.177 2856.6176 2859.2375 2886.728 2929.696    10  a
##     glm  428.327  460.143  488.8717  481.6270  497.740  621.560    10   b
##   optim  193.110  201.433  203.5732  204.3235  206.886  210.125    10   b
```

Among the 4 methods, Newton, MM and GLM give the same estimation and inference, while optim have a estimation with smaller bias and narroer confidence interval. MM iterate much more times compared to Newton (5), GLM (4) and optim (6), because I didn't have a closed form of $\hat{\beta}_1$, so I use Newton with in the iteration. As for computation time, Newton takes longest time, while optim takes shortest. Comparatively, optim achieve the best performance among the four.

Problem 4: EM algorithm for censored exponential data

This will be a continuation of the lab problem on EM for censored exponential data. Suppose we have survival times $t_1, \ldots, t_n \sim Exponential(\lambda)$.

- Do not observe all survival times because some are censored at times $c_1, \ldots, c_n$.
- Actually observe $y_1, \ldots, y_n$, where $y_i = min(t_i, c_i)$
  - Also have an indicator δ_i where $\delta_i = 1$ is $y_i \leq c_i$
    - i.e. $\delta_i = 1$ if not censored and $\delta_i = 0$ if censored

Do the following:

- Derive an EM algorithm to estimate the parameter λ. Show your derivation here and report updates for the **E-step** and **M-Step**.

Answer: Based on the setting of question we have the total likelihood:

$$L(t|\lambda) = \prod_i \frac{1}{\lambda} e^{-\frac{t_i}{\lambda}} = \prod_i \frac{1}{\lambda} e^{-\frac{\delta_i y_i + (1-\delta_i)t_i}{\lambda}} = L(y, t|\lambda)$$

which is because when $\delta_i = 1$, we have $y_i = t_i$ because of not censoring, but note here t_i are for those with $t_i > c_i$

Therefore, the log-likelihood is:

$$l(y, t|\lambda) = \sum_i (-\log\lambda - \frac{\delta_i y_i + (1-\delta_i)t_i}{\lambda}) = -n\log\lambda - \frac{1}{\lambda}\sum_i (\delta_i y_i + (1-\delta_i)t_i)$$

In **E-step**, our Q() function would be:

$$Q(\lambda|\lambda_0) = E_{t_i}(l(y, t|\lambda)|y, \delta, \lambda_0) = -n\log\lambda - \frac{1}{\lambda}\sum_i (\delta_i y_i + (1-\delta_i)E_{t_i}(t_i))$$

$$E_{t_i}(t_i|y_i, \delta_i = 0, \lambda_0) = \int_{y_i}^{\infty} t_i \frac{\frac{1}{\lambda_0} e^{-\frac{t_i}{\lambda_0}}}{\int_{y_i}^{\infty} \frac{1}{\lambda_0} e^{-\frac{t}{\lambda_0}} dt} dt_i = \int_{y_i}^{\infty} t_i \frac{\frac{1}{\lambda_0} e^{-\frac{t_i}{\lambda_0}}}{e^{-\frac{y_i}{\lambda_0}}} dt_i = t_i + \lambda_0$$

Plug in, Q() would be:

$$Q(\lambda|\lambda_0) = E_{t_i}(l(y,t|\lambda)|y,\delta,\lambda_0) = -n\log\lambda - \frac{1}{\lambda}\sum_i (y_i + (1-\delta_i)\lambda_0)$$

In **M-step**, we will maximize Q too get estimates of λ, which is achieved by iterate:

$$\hat{\lambda}^{(k+1)} = \frac{1}{n}\sum_i (y_i + (1-\delta_i)\hat{\lambda}^{(k)})$$

- Implement your EM in R and fit it to the `veteran` dataset from the `survival` package.
  - Report your fitted λ value. How did you monitor convergence?

  Answer: Convergence is set to be reached when $|\lambda^{(k+1)} - \lambda^{(k)}| < 10^{-6}$.

  - Report a 95% confidence interval for λ, and explain how it was obtained.

  Answer: I get the 95% confidence by bootstrap with B = 10000.

  - Compare 95% confidence interval and results from those obtained by fitting an accelerated failure time model (AFT) in R with exponential errors. You can fit an AFT model using the `survreg()` function from the `survival` package. If you choose `dist = "weibull` and `shape = 1` as parameter arguments, this will provide exponential errors.

```
EM_exp <- function(times, status, tol = 1e-6, max_iter = 1000) {
  lambda <- mean(times[status == 1])
  n <- length(times)
  iter <- 0
  diff <- Inf

  while (diff > tol && iter < max_iter) {

    # M-step: Update lambda
    lambda_new <- sum(times+(1-status)*lambda)/n

    # Check convergence
    diff <- abs(lambda_new - lambda)
    iter <- iter + 1

    lambda = lambda_new
  }

  list(lambda = lambda, iterations = iter, converged = (iter < max_iter))
}
```

```
library(survival)
library(dplyr)

# EM algorithm
fit.EM<-EM_exp(veteran$time, veteran$status)
print(paste("Lambda from EM algorithm:", fit.EM$lambda))
```

```
## [1] "Lambda from EM algorithm: 130.179687457469"
```

```
# Compute confidence interval
# non-parametric bootstrap
set.seed(123)
B = 10000
```

```r
boot_mean = rep(NA, B)
for(i in 1:B){
  index <- sample(1:dim(veteran)[1], dim(veteran)[1], replace = TRUE)
  boot_mean[i] = EM_exp(veteran$time[index], veteran$status[index])$lambda
}

em_CI<-quantile(boot_mean, probs = c(0.025,0.975))
print(paste0("95% CI for lambda (EM): (", em_CI[1],",",em_CI[2],")"))
```

```
## [1] "95% CI for lambda (EM): (103.864566922504,160.502115380874)"
```

```r
# the AFT model
fit.AFT<- survreg(Surv(time, status) ~ 1, data = veteran, dist = "weibull", scale = 1)
lambda_aft <- exp(fit.AFT$coefficients)
print(paste("Lambda from AFT model:", lambda_aft))
```

```
## [1] "Lambda from AFT model: 130.1796875"
```

```r
# Compute 95% CI for lambda using AFT model
se_lambda_aft <- sqrt(fit.AFT$var[1]) * lambda_aft  # Delta method
CI_lower_aft <- lambda_aft - 1.96 * se_lambda_aft
CI_upper_aft <- lambda_aft + 1.96 * se_lambda_aft

print(paste0("95% CI for lambda (AFT): (", CI_lower_aft,", ", CI_upper_aft, ")"))
```

```
## [1] "95% CI for lambda (AFT): (107.627207248021, 152.732167751979)"
```

Extra credit (up to 10 points)! Expected vs. observed information

Part A: Show that the expected and observed information are equivalent for logistic regression

Part B: Let's say you are instead performing probit regression, which is similar to logistic regression but with a different link function. Specifically, probit regression uses a probit link:

$$\Phi^{-1}(Pr[Y_i = 1|X_i]) = X_i^T\beta,$$

where Φ^{-1} is inverse of the CDF for the standard normal distribution. **Are the expected and observed information equivalent for probit regression?** Justify why or why not.