ST509 - Takehome Midterm Solution

1. (a)

$$\begin{aligned}\nabla f(\boldsymbol{\beta}) &= \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \\ &= \sum_{i=1}^{n} \left(y_i - e^{\mathbf{x}_i^T \boldsymbol{\beta}}\right) \mathbf{x}_i \\ &= \mathbf{X}^T \left(\mathbf{y} - e^{\mathbf{X}\boldsymbol{\beta}}\right)\end{aligned}$$

and

$$\begin{aligned}\mathbf{H}(\boldsymbol{\beta}) &= \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} \\ &= -\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T e^{\mathbf{x}_i^T \boldsymbol{\beta}} \\ &= -\mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}) \mathbf{X}\end{aligned}$$

where $\mathbf{W}(\boldsymbol{\beta}) = Diag\left\{\mu(\mathbf{x}_1; \boldsymbol{\beta}), \cdots, \mu(\mathbf{x}_n; \boldsymbol{\beta})\right\}$.

(b)

$$\begin{aligned}\boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} + \left\{\mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}^{(t)}) \mathbf{X}\right\}^{-1} \mathbf{X}^T \left(\mathbf{y} - e^{\mathbf{X}^T \boldsymbol{\beta}^{(t)}}\right) \\ &= \left\{\mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}^{(t)}) \mathbf{X}\right\}^{-1} \mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}^{(t)}) \mathbf{z}, \qquad \mathbf{z} = \mathbf{X}\boldsymbol{\beta}^{(t)} + \{\mathbf{W}(\boldsymbol{\beta}^{(t)})\}^{-1} \left(\mathbf{y} - e^{\mathbf{X}^T \boldsymbol{\beta}^{(t)}}\right) \\ &= (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{z}}\end{aligned}$$

where $\tilde{\mathbf{X}} = \{\mathbf{W}(\boldsymbol{\beta}^{(t)})\}^{1/2} \mathbf{X}$ and $\tilde{\mathbf{z}} = \{\mathbf{W}(\boldsymbol{\beta}^{(t)})\}^{1/2} \mathbf{z}$.

(c)

```
my.posreg <- function(x, y, beta0, eps = 1.0e-5, max.iter = 100)
  # x: (n*p) predictor matrix
  # y: response vector
  # beta0: initial beta for NR algorithm
  # eps: convergence criterion
  # max.iter: maximum number of iterations of the NR algorithm
{
  # write your own code here
  beta <- beta0
  for (iter in 1:max.iter)
  {
    eta <- x %*% beta
    mu <- w <- c(exp(eta))
    z <- eta + (y - mu)/w

    tilde.x <- x * sqrt(w)
    tilde.z <- z * sqrt(w)
```

```
      qr.obj <- qr(tilde.x)
      new.beta <- backsolve(qr.obj$qr, qr.qty(qr.obj, tilde.z))

      if (max(abs(new.beta - beta)) < eps) break
      beta <- new.beta
    }
    if (iter == max.iter) warning("Algorithm may not be converged!")

    # output
    beta.mle <- c(new.beta)
    return(beta.mle) # mle of beta
  }
```

2. (Lasso-penalized Poisson Regression)

```
my.posreg.lasso <- function(x, y, beta0, lambda, eps = 1.0e-5, max.iter = 100)
  # lambda: regularization parameter
  # others: identical to those in posreg.lasso()
  {
  # write your own code here
  n <- length(y)
  p <- ncol(x)

  beta <- beta0
  for (iter in 1:max.iter)  {
    eta <- x %*% beta
    mu <- w <- c(exp(eta))
    z <- eta + (y - mu)/w

    tilde.x <- x * sqrt(w)
    tilde.z <- z * sqrt(w)


    # CD for poslasso ##################

    # standardize data
    str.tilde.x <- t(t(tilde.x) - apply(tilde.x, 2, mean))
    norm.tilde.x <- apply(str.tilde.x^2, 2, mean)
    str.tilde.x <- t(t(str.tilde.x)/sqrt(norm.tilde.x))

    # transformation beta
    str.beta <- beta * sqrt(norm.tilde.x)

    # residual
    r <- (tilde.z - str.tilde.x %*% str.beta)

    # CD update
    new.str.beta <- str.beta
    for (j in 1:p)
    {
      xj <- 1/n * crossprod(str.tilde.x[,j],  r) + str.beta[j]
      new.str.beta[j] <- S(xj, lambda)
      r <- r - (new.str.beta[j] - str.beta[j]) * str.tilde.x[,j]
    }
```

```
    # transform back
    new.beta <- new.str.beta / sqrt(norm.tilde.x)

    if (max(abs(new.beta - beta)) < eps) break
    beta <- new.beta
  }
  if (iter == max.iter) warning("Algorithm may not be converged!")

  # output
  beta.lasso <- new.beta
  return(beta.lasso) # lasso-penalized solution
}
```

where the soft thresholding operator function is given by

```
# set soft-thresholding function
S <- function(z, lambda) {
  (z - lambda) * (z > lambda) +
    (z + lambda) * (z < -lambda) +
    0 * (abs(z) <= lambda)
}
```

3. (a)

$$L_c(\theta) = \log\left\{\prod_{i=1}^{n} \frac{1}{\theta} e^{-x_i/\theta}\right\} = -n\log\theta - \frac{1}{\theta}\sum_{i=1}^{n} x_i$$

$$\begin{aligned}
Q(\theta;\theta^{(t)},\mathbf{y},\boldsymbol{\delta}) &= E_{\theta^{(t)}}\left[L_c(\theta)|\mathbf{y},\boldsymbol{\delta}\right] \\
&= -n\log\theta - \frac{1}{\theta}\sum_{i=1}^{n} E_{\theta^{(t)}}\left(x_i|y_i,\delta_i\right) \\
&= -n\log\theta - \frac{1}{\theta}\sum_{i=1}^{n_u} y_i - \sum_{i=n_u+1}^{n} E_{\theta^{(t)}}\left(x_i|x_i>c_i\right) \\
&= -n\log\theta - \frac{1}{\theta}\sum_{i=1}^{n_u} y_i - \sum_{i=n_u+1}^{n}\left(\theta^{(t)}+y_i\right) \\
&= -n\log\theta - \frac{1}{\theta}\sum_{i=1}^{n} y_i - \frac{1}{\theta}(n-n_u)\theta^{(t)}.
\end{aligned}$$

(b) The updating equation for M-step is

$$\theta^{(t+1)} = \frac{1}{n}\left\{\sum_{i=1}^{n} y_i + (n-n_u)\theta^{(t)}\right\}$$

which solves

$$\frac{\partial Q(\theta)}{\partial\theta} = -\frac{n}{\theta} - \frac{1}{\theta^2}\left\{\sum_{i=1}^{n} y_i + (n-n_u)\theta^{(t)}\right\} = 0$$

Therefore

```
"my.em4cen" <- function(y, d, theta0, eps = 1.0e-5, max.iter = 100)
  # y: observed, possibly censored time
  # d: censoring indicator
  # theta0: initial value for EM
  # eps: convergence criterion
  # max.iter: maximum number of iterations of the NR algorithm
  {
    # write your own code here
    theta <- theta0
    n.c <- sum(1 - d)
    for (iter in 1:max.iter)
    {
      new.theta <- 1/n * (sum(y) + n.c * theta)
      if (abs(new.theta - theta) < eps) break
      theta <- new.theta
    }
    if (iter == max.iter) warning("Algorithm may not be converged!")

    # output
    theta.mle <- new.theta
    return(theta.mle) # mle
  }
```