1 Introduction

As there is a variety of choices for the high-dimensional sparse data analysis, including LASSO and MCP, one may think of an alternative which behaves like if it is between them. From this idea, Moderately Clipped LASSO(MCL) was introduced and it obtains Oracle property under some conditions for least squares problems. This approach also can be applied to generalized linear models with specified likelihoods such as Poisson distributions. In this report, we will talk about the MCL and simulations of relative performances between some estimators with it.

2 About MCL

Consider the penalized estimation problem such that finding the minimizer of ;

$$Q(\boldsymbol{\beta}; \gamma, \lambda) = L(\boldsymbol{\beta}) + \sum_{j \in I} J(|\beta_j|; \gamma, \lambda)$$

where $\mathbf{y}$ and $\mathbf{X}$ are response column vector and design matrix with n observations each, L is a given loss function of $\boldsymbol{\beta}$, $\mathbf{y}$ and $\mathbf{X}$, $I = \{1, \cdots, p\}$ and $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_p)^\top$. Here, J, or the MCL panelty function, satisfies $J(0; \gamma, \lambda) = 0$ and

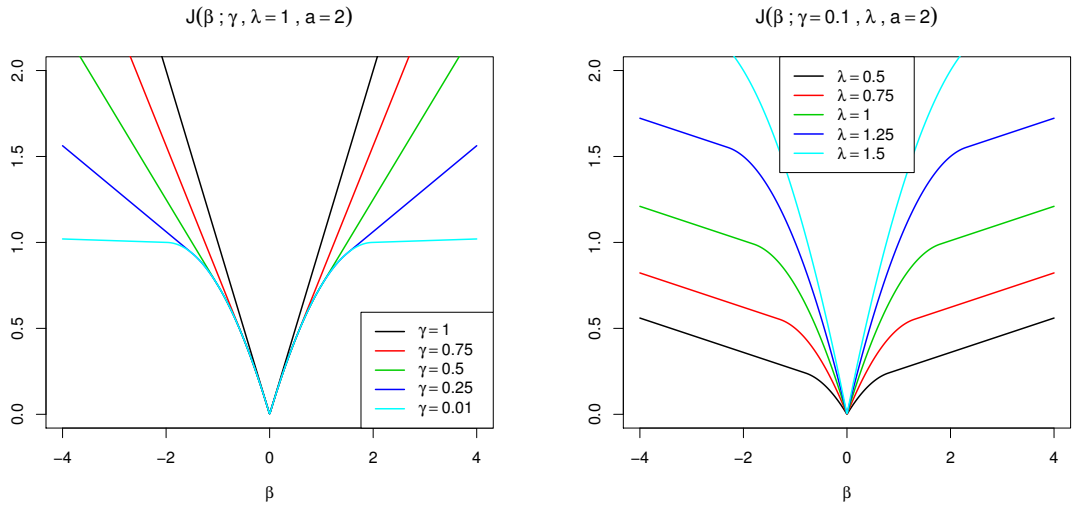
$$\nabla J(|\beta|; \gamma, \lambda) = \max\left(\gamma, \lambda - \frac{|\beta|}{a}\right)$$

$$J(|\beta|; \gamma, \lambda) = \begin{cases} \gamma|\beta| + \frac{a(\lambda-\gamma)^2}{2}, & |t| \geq a(\lambda - \gamma) \\ \left(\lambda - \frac{|\beta|}{2a}\right)|\beta|, & o.w. \end{cases}$$

where $\nabla J(a; \gamma, \lambda)$ denotes $\left.\frac{\partial J(t;\gamma,\lambda)}{\partial t}\right|_{t=a}$ and $0 \leq \gamma \leq \lambda$, $a > 1$.

2.1 Behaviour of MCL

Here, MCL with $\gamma = \lambda$ obtains the LASSO penalty and $\gamma = 0$ obtains MCP penalty so that MCL estimators with $0 \leq \gamma \leq \lambda$ locating between them. When a and γ are fixed, the slope is the same in all lambs. But as the λ increases, the absolute value of the penalty at the clipped position increases.



3 Numerical Study

3.1 Simulation Design : A Poisson Regression Example

Let $Y_i|\mathbf{x}_i \overset{ind.}{\sim} Poi(\mu_i)$ where $\log \mu_i = \mathbf{x}_i^\top \boldsymbol{\beta}$. Then, the negative log likelihoods becomes :

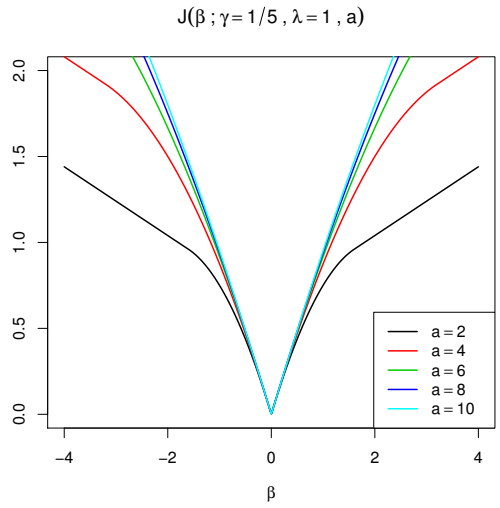
$$
\begin{aligned}
-l_n(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \{-\mathbf{x}_i^\top \boldsymbol{\beta} y_i + \exp(\mathbf{x}_i^\top \boldsymbol{\beta}) + \log(y_i!)\} \\
-\bigtriangledown l_n(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \mathbf{x}_i \{-y_i + \exp(\mathbf{x}_i^\top \boldsymbol{\beta})\} = \sum_{i=1}^{n} \mathbf{x}_i \{-y_i + \mu_i\} \\
-\bigtriangledown^2 l_n(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top \exp(\mathbf{x}_i^\top \boldsymbol{\beta}) = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top \mu_i.
\end{aligned}
$$

We reproduced a part of penalized Poisson regression simulations introduced by Hossain and Ahmed[1]. We used a Monte Carlo simulation experiment to examine the MSE performances of the MCL estimators. Our simulation is based on a Poisson regression model with $n = 30, 45, 180$. Each covariate vector $\mathbf{x}_i$ was generated from a multivariate standard normal distribution with zero mean vector and covariance matrix $\mathbf{I}_{p=q+3}$.

We set the true value of $\boldsymbol{\beta}$ at $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^\top, \mathbf{0}_q^\top)^\top$ with $\boldsymbol{\beta}_1 = (0.2, -1.2, 0.1)^\top$. We provide detailed results $q = 9, 15, 24, 33, 42$ and 51.

The number of replications in the simulation was 100 and based on the simulated data, we estimated MSE of the MCL estimators. We considered unrestricted estimator $\hat{\boldsymbol{\beta}}$ as the

[1]Shrinkage and Penalty Estimators of A Poisson Regression Model, Hossain, S. & Ahmed, E., Aust. N. Z. J. Stat. 54(3), 2012, 359373



'benchmark' estimator, and thus the performance of the estimators is evaluated in terms of the MSE relative to the MSE of $\hat{\boldsymbol{\beta}}$ (RelMSE). For any methods of estimation A, define

$$\text{RelMSE(A)} = \frac{\text{simulated MSE}(\hat{\boldsymbol{\beta}})}{\text{simulated MSE}(\hat{\boldsymbol{\beta}}^{\text{A}})}$$

of which value larger than 1 implies the superiority of the $\hat{\boldsymbol{\beta}}^{A}$.

3.2 Computation Algorithm

The MCL penalty can be decomposed as $J(|t|;\gamma,\lambda) = S(|t|;\gamma,\lambda) + \lambda|t|$ where S is a continuously differentiable concave function of t. Hence,

$$Q(\boldsymbol{\beta};\gamma,\lambda) = L(\boldsymbol{\beta}) + \sum_{j\in I} S(\beta_j;\gamma,\lambda) + \lambda||\boldsymbol{\beta}||_1$$

which is a sum of convex and concave functions. By CCCP, update current solution $\mathbf{b}^{(i)}$ as ;

$$\mathbf{b}^{(i+1)} = \arg\min_{\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}) + \nabla S(\mathbf{b}^{(i)};\gamma,\lambda)^\top \boldsymbol{\beta} + \lambda||\boldsymbol{\beta}||_1 \right\}.$$

where

$$\begin{aligned} L(\boldsymbol{\beta};\gamma,\lambda) &\approx L(\mathbf{b}^{(i)};\gamma,\lambda) + \nabla L(\mathbf{b}^{(i)};\gamma,\lambda)^\top(\boldsymbol{\beta}-\mathbf{b}^{(i)}) + (\boldsymbol{\beta}-\mathbf{b}^{(i)})^\top \nabla^2 L(\mathbf{b}^{(i)};\gamma,\lambda)(\boldsymbol{\beta}-\mathbf{b}^{(i)})/2 \\ &:= L^{(i)}(\boldsymbol{\beta};\gamma,\lambda) \propto \boldsymbol{\beta}^\top \nabla^2 L(\mathbf{b}^{(i)};\gamma,\lambda)\boldsymbol{\beta}/2 + \{\nabla L(\mathbf{b}^{(i)};\gamma,\lambda) - \nabla^2 L(\mathbf{b}^{(i)};\gamma,\lambda)\mathbf{b}^{(i)}\}^\top \boldsymbol{\beta} \end{aligned}$$

which is a form of $\boldsymbol{\beta}^\top \mathbf{U}^{(i)} \boldsymbol{\beta} + \mathbf{v}^{(i)\top} \boldsymbol{\beta}$ with $\mathbf{U}^{(i)} = \nabla^2 L(\mathbf{b}^{(i)};\gamma,\lambda)/2$, $\mathbf{v}^{(i)} = \nabla L(\mathbf{b}^{(i)};\gamma,\lambda) - \nabla^2 L(\mathbf{b}^{(i)};\gamma,\lambda)\mathbf{b}^{(i)}$. Now, we can approximate the original problem to solving the simple

LASSO :

$$\mathbf{b}^{(i+1)} \approx \arg\min_{\boldsymbol{\beta}} \{\boldsymbol{\beta}^\top \mathbf{U}^{(i)} \boldsymbol{\beta} + \mathbf{v}^{(i)\top} \boldsymbol{\beta} + \nabla S(\mathbf{b}^{(i)}; \gamma, \lambda)^\top \boldsymbol{\beta} + \lambda ||\boldsymbol{\beta}||_1\}.$$

Solving the equation $\nabla_j(\cdot) = 0$, we obtain

$$b_j^{(i+1)} = \frac{\text{sign}(a_j^{(i)})(|a_j^{(i)}| - \lambda)_+}{b_j^{(i)}}$$

with $a_j^{(i)} = -\mathbf{v}_j^{(i)} - \nabla_j S(\mathbf{b}^{(i)}; \gamma, \lambda) - 2\mathbf{U}_{[-j,j]}^{(i)\top} \boldsymbol{\beta}_{-j}$, $b_j^{(i)} = 2\mathbf{U}_{jj}^{(i)}$ given that $\mathbf{U}^{(i)}$ is a symmetric positive definite matrix.

To check the optimality, we used the KKT conditions which investigates whether

$$\nabla L(\mathbf{b}_\phi^{(i)}) + \nabla S(\mathbf{b}_\phi^{(i)}; \gamma, \lambda) = -\lambda \, \mathbf{sign}(\mathbf{b}_\phi^{(i)})$$

and

$$|\nabla L(\mathbf{b}_{\phi^c}^{(i)}) + \nabla S(\mathbf{b}_{\phi^c}^{(i)}; \gamma, \lambda)| \leq \lambda$$

for $\phi = \{j; \mathbf{b}_j^{(i)} \neq 0\}$.

3.3 Simulation Results

The results for simulations are summarized on Table 1. We used LASSO estimator with optimal tuning parameter $\hat{\lambda}$ which minimized the loss function and MCL with $a = 2.1$, $\gamma = \hat{\lambda}$. For SCAD, $a = 3.7$ and optimal λ was chosen through 5-fold CV.

The behaviors of different penalties are almost the same. As the sample size increases, RelMSE decreased for all methods which implies that the MSE of unrestricted model was similar to the restricted ones.

However, if q increases so that the true $\boldsymbol{\beta}$ becomes sparser, restricted models showed the superiority. The relative MSE of LASSO and MCL and the relative MSE of LASSO and MCL were featured in the last two column of the table. All other penalties including MCL were better than unrestricted estimators even under low dimensions.

On the other hand, the relative MSE between LASSO and the other were almost the same in all simulations.

In the sense of True selections and False selections, which corresponds to the number of non-zero estimators of non-zero true coefficients and number of zero estimators of non-zero true coefficients respectively, MCL and SCAD showed better numbers than LASSO. However MCL and SCAD failed to beat LASSO's False selections.

Table 1: Simulation Results

n	q	RelMSE(MCL)	RelMSE(lasso)	RelMSE(SCAD)	$\frac{MSE_{LASSO}}{MSE_{MCL}}$	$\frac{MSE_{LASSO}}{MSE_{SCAD}}$
30	51				0.996	0.988
30	42				1.004	0.995
30	33				1.012	1.003
30	24	247.632	245.914	245.522	1.007	0.998
30	15	1.394	1.395	1.434	0.999	1.028
30	9	1.406	1.409	1.396	0.998	0.991
45	51				0.998	0.988
45	42	1,135.769	1,137.776	1,131.858	0.998	0.995
45	33	335.363	336.542	335.114	0.996	0.996
45	24	1.360	1.362	1.358	0.998	0.997
45	15	1.312	1.313	1.296	1.000	0.987
45	9	1.297	1.295	1.289	1.002	0.996
180	51	1.234	1.234	1.231	1.000	0.997
180	42	1.245	1.245	1.243	1.000	0.998
180	33	1.252	1.251	1.249	1.001	0.998
180	24	1.234	1.233	1.232	1.000	0.999
180	15	1.259	1.259	1.256	1.000	0.998
180	9	1.244	1.243	1.243	1.000	0.999

4 Conclusion

According to the results of simulations, MCL and SCAD showed almost the same performances as LASSO and were better than in True variable selections. However, the results for False selections were way different than expected. In fact, the oracle property and other good properties of MCL were not clear in this project due to the small sample sizes. For further improvement of this project, we can consider growing the sample sizes, then we may observe the better performances of MCL in False selections.

5 Appendix : R Implementations

5.1 Local Functions Declaration

```
pos.fun <- function(x) x*I(x >= 0)

loss.fun = function(y.vec,x.mat,b.vec) {
  n <- length(y.vec)
  xb.vec = drop(x.mat %*% b.vec)
  ret = -sum(y.vec*xb.vec) + sum(exp(xb.vec))
  return(ret/n)
```

Table 2: Simulation Results : True Selection & False selection

< True Selection >					< False Selection >				
n	q	MCL	lasso	SCAD	n	q	MCL	lasso	SCAD
30	54	1	1	1	30	54	7	1	3
30	45	1	1	1	30	45	5	1	2
30	36	1	1	1	30	36	4	1	2
30	27	1	1	1	30	27	3	0	1.500
30	18	2	1	1.5	30	18	3	0	1
30	12	2	1	1	30	12	1.500	0	1
45	54	2	1	1	45	54	4.500	0	2
45	45	2	1	1	45	45	6.500	0.500	2.500
45	36	2	1	2	45	36	4	1	2
45	27	2	1	2	45	27	3	0	1
45	18	2	1	2	45	18	3	1	1
45	12	2	1	2	45	12	2.500	0	1
180	54	2	2	2	180	54	4	1	3.500
180	45	3	2	2	180	45	5	1	3.500
180	36	3	2	2.5	180	36	5.500	1.500	3
180	27	3	2	2	180	27	4	1	2
180	18	3	3	3	180	18	4	1	2.500
180	12	3	3	3	180	12	4	1	1

```
}
grad.fun = function(y.vec, x.mat, b.vec) {
  n <- length(y.vec)
  exb.vec = exp(drop(x.mat %*% b.vec))
  ret = crossprod(x.mat, -y.vec + exb.vec)
  return(ret/n)
}
hess.fun = function(y.vec, x.mat, b.vec) {
  n <- length(y.vec)
  exb.vec = exp(drop(x.mat %*% b.vec))
  ret = crossprod(x.mat, diag(exb.vec) %*% x.mat )
  return(ret/n)
}
kkt.fun <- function(g.vec, b.vec, lam, eps, prnt = F) {
  kkt1 = prod(abs(g.vec[b.vec != 0] + lam*sign(b.vec[b.vec != 0])) < eps)
  kkt2 = prod(abs(g.vec[b.vec == 0]) - lam < eps)
  return(kkt1 & kkt2)
}

# lasso ---------------------------------------------------------------------


quad.lasso.fun = function(q.mat, l.vec, lam, b.vec, iter.max, eps) {
  for (iter in 1:iter.max) {
    grad = 2*q.mat %*% b.vec + l.vec
    if (kkt.fun(grad, b.vec, lam, eps)) break
```

```
    for (j in 1:length(b.vec)) {
      a = 2*q.mat[j,j] ; b = -2*sum(q.mat[j,-j]*b.vec[-j]) - l.vec[j]
      if (abs(b) < lam) { b.vec[j] = 0
      } else {b.vec[j] = sign(b)*(abs(b) - lam)/a}
    }
  }
  return(list(b.vec = b.vec, lam = lam))
}


lasso.fun <- function(y.vec, x.mat, b.vec, lam, iter.max, eps) {
  for (iter in 1:iter.max) {
    h.mat <- hess.fun(y.vec, x.mat, b.vec) ; g.vec <- grad.fun(y.vec, x.mat, b.vec)
    if (kkt.fun(g.vec, b.vec, lam, eps)) break
    q.mat <- h.mat/2 ; l.vec <- g.vec - drop(h.mat %*% b.vec)
    b.vec <- quad.lasso.fun(q.mat, l.vec, lam, b.vec, iter.max, eps)$b.vec # update
  }
  return(list(b.vec = b.vec, g.vec = g.vec, lam = lam))
}


# SCAD ----------------------------------------------------------------------

grad.Jtd.fun <- function(b, a, lam) {
  t <- abs(b)
  if (t <= lam) {
    p <- lam
  } else {
    p <- pos.fun(a*lam - t)/(a - 1)
  }
  return(sign(b)*(p - lam))
}
grad.Jtd.fun <- Vectorize(grad.Jtd.fun, 'b')

scad.fun <- function(y.vec, x.mat, b.vec, a, lam, iter.max, eps) {
  for (iter in 1:iter.max) {
    h.mat <- hess.fun(y.vec, x.mat, b.vec) ; g.vec <- grad.fun(y.vec, x.mat, b.vec)
    if (kkt.fun(g.vec + grad.Jtd.fun(b.vec, a, lam), b.vec, lam, eps)) break
    q.mat <- h.mat/2 ; l.vec <- g.vec - drop(h.mat %*% b.vec) + grad.Jtd.fun(b.vec, a, lam)
    b.vec <- quad.lasso.fun(q.mat, l.vec, lam, b.vec, iter.max, eps)$b.vec # update
  }
  return(list(b.vec = b.vec, g.vec = g.vec + grad.Jtd.fun(b.vec, a, lam), lam = lam))
}


# MCL ----------------------------------------------------------------------

grad.J.fun <- Vectorize(function(t, a, gam, lam) (sign(t)*max(gam, lam - abs(t)/a)), 't')
# plot(function(t) grad.J.fun(t, 2, .5, 1), n = 1e3, -3, 3)

grad.S.fun <- function(t, a, gam, lam){grad.J.fun(t, a, gam, lam) - lam*sign(t)}
# plot(function(t) grad.S.fun(t, 2, .5, 1), n = 1e3, -3, 3)

mcl.fun <- function(y.vec, x.mat, b.vec, a, gam, lam, iter.max, eps) {
  for (iter in 1:iter.max) {
    h.mat <- hess.fun(y.vec, x.mat, b.vec);g.vec <- grad.fun(y.vec, x.mat, b.vec)
    if (kkt.fun(g.vec + grad.S.fun(b.vec, a, gam, lam), b.vec, lam, eps)) break
    q.mat <- h.mat/2 ; l.vec <- g.vec - drop(h.mat %*% b.vec) + grad.S.fun(b.vec, a, gam, lam)
    b.vec <- quad.lasso.fun(q.mat, l.vec, lam, b.vec, iter.max, eps)$b.vec
  }
  return(list(b.vec = b.vec, g.vec = g.vec + grad.S.fun(b.vec, a, gam, lam), lam = lam))
}
```

```
# lasso path ------------------------------------------------------------


lasso.path.fun <- function(y.vec, x.mat, lam.vec, iter.max = 1e3, eps = 1e-8) {

  b.mat <- NULL ; b.vec <- drop(solve(t(x.mat) %*% x.mat + diag(1, length(b.vec)))
                                %*% t(x.mat) %*% y.vec)
  g.vec <- grad.fun(y.vec, x.mat, b.vec)
  l.vec <- c()
  for (lam in lam.vec) {
    for (iter in 1:iter.max) {
      if (kkt.fun(g.vec, b.vec, lam, eps)) break
      v.vec <- (abs(g.vec) >= (lam - eps)) | (b.vec != 0)
      ax.mat <- x.mat[, v.vec, drop = F] # preserve as matrix
      fit <- lasso.fun(y.vec, ax.mat, b.vec[v.vec], lam, iter.max, eps)
      b.vec[v.vec] <- fit$b.vec
      g.vec <- grad.fun(y.vec, x.mat, b.vec)
    }
    l.vec <- c(l.vec, loss.fun(y.vec, x.mat, b.vec))
    b.mat <- cbind(b.mat, b.vec)
  }
  return(list(b.mat = b.mat, lam.vec = lam.vec, l.vec = l.vec))
}

# SCAD path -------------------------------------------------------------


scad.path.fun <- function(y.vec, x.mat, a, lam.vec, iter.max = 1e3, eps = 1e-8) {
  # SCAD with set control

  b.mat <- NULL ; b.vec <- drop(solve(t(x.mat) %*% x.mat + diag(1, length(b.vec)))
                                %*% t(x.mat) %*% y.vec)
  g.vec <- grad.fun(y.vec, x.mat, b.vec)
  l.vec <- c()
  for (lam in lam.vec) {
    for (iter in 1:iter.max) {
      Jtd <- grad.Jtd.fun(b.vec, a, lam)
      if (kkt.fun(g.vec + Jtd, b.vec, lam, eps)) break
      v.vec <- (abs(g.vec + Jtd) >= (lam - eps)) | (b.vec != 0)
      ax.mat <- x.mat[, v.vec, drop = F] # preserve as matrix
      fit <- scad.fun(y.vec, ax.mat, b.vec[v.vec], a, lam, iter.max, eps)
      b.vec[v.vec] <- fit$b.vec
      g.vec <- grad.fun(y.vec, x.mat, b.vec)
    }
    l.vec <- c(l.vec, loss.fun(y.vec, x.mat, b.vec))
    b.mat <- cbind(b.mat, b.vec)
  }
  return(list(b.mat = b.mat, lam.vec = lam.vec, l.vec = l.vec))
}

# MCL path --------------------------------------------------------------


mcl.path.fun <- function(y.vec, x.mat, a, gam, lam.vec, iter.max = 1e3, eps = 1e-8) {
  # MCL with set control

  b.mat <- NULL ; b.vec <- drop(solve(t(x.mat) %*% x.mat + diag(gam, length(b.vec)))
                                %*% t(x.mat) %*% y.vec)
  g.vec <- grad.fun(y.vec, x.mat, b.vec)
  l.vec <- c()
  for (lam in lam.vec) {
    for (iter in 1:iter.max) {
```

```
      gS <- grad.S.fun(b.vec, a, gam, lam)
      if (kkt.fun(g.vec + gS, b.vec, lam, eps)) break
      v.vec <- (abs(g.vec + gS) > (lam - eps)) | (b.vec != 0)
      ax.mat <- x.mat[, v.vec, drop = F] # preserve as matrix
      fit <- mcl.fun(y.vec, ax.mat, b.vec[v.vec], a, gam, lam, iter.max, eps)
      b.vec[v.vec] <- fit$b.vec
      g.vec <- grad.fun(y.vec, x.mat, b.vec)
    }
    l.vec <- c(l.vec, loss.fun(y.vec, x.mat, b.vec))
    b.mat <- cbind(b.mat, b.vec)
  }
  return(list(b.mat = b.mat, lam.vec = lam.vec, l.vec = l.vec))
}
```

5.2 Simulation Part

```
lib <- c('mvnfast', 'dplyr', 'tictoc', 'glmnet', 'profvis')
chck <- sapply(lib, require, character.only = TRUE)
if (sum(!chck) > 0) {install.packages(lib[!chck])}
rm(list = ls())
set.seed(1155)

user <- Sys.getenv("RSTUDIO_USER_IDENTITY")
paste('/Users/', user, '/Dropbox/Seo&Chang/PPois_with_SetControl.R', sep = '') %>%
  source

b1 <- rep(1, 3)
q <- c(5e2, 3e2) - 3
n <- c(1e2, 2e2, 4e2, 8e2)

models <- list()
for (ni in 1:length(n)) {
  for (qi in 1:length(q)) {
    eval(paste('n', n[ni], 'q', q[qi], sep = '') %>%
           paste('models$', ., ' <- list(n = n[', ni, '], b.vec = c(b1, rep(0, q[', qi,
                 ']))))', sep = '') %>% parse(text = .))
  }
}

K <- 1
cvf <- 5
imax <- 2e2
eps0 <- 1e-6
timer <- c()
modeliter <- 1
# profvis({for (modeliter in 1:length(models)) {
profvis({for (modeliter in 1:1) {

  k <- 1
  n <- models[[modeliter]]$n
  p <- length(models[[modeliter]]$b.vec)
  b.vec <- models[[modeliter]]$b.vec
  models[[modeliter]]$unres.MSE <- rep(NA, K)
  models[[modeliter]]$lasso.MSE <- rep(NA, K)
  # models[[modeliter]]$scad.MSE <- rep(NA, K)
  models[[modeliter]]$mcl.MSE <- rep(NA, K)
  r <- 10

  while (k <= K) {
    tic(paste(names(models)[modeliter], '~' , round(k/K*1e2, 1), '%',
```

```
            '(Est. rem. tm.', '=', round((K - k)*median(timer)/60, 1), 'ms)'))
    x.mat0 = rmvn(n, rep(0, p), sigma = diag(1/r, p))
    exb.vec = exp(drop(x.mat0 %*% b.vec)) ; m.vec = exb.vec
    y.vec0 = rpois(n, m.vec)
    lam.max <- max(abs(grad.fun(y.vec0, x.mat0, rep(0, p))))
    lam.min <- lam.max*1e-3
    lamv <- exp(seq(log(lam.max), log(lam.min), length.out = 50))
    # lasso.fit <- lasso.path.fun(y.vec0, x.mat0, lamv, imax, eps0)

    x.mat1 = rmvn(n/2, rep(0, p), sigma = diag(1/r, p))
    exb.vec = exp(drop(x.mat0 %*% b.vec)) ; m.vec = exb.vec
    y.vec1 = rpois(n/2, m.vec)

    fold.id <- split(sample(1:n), 1:cvf) ; fid <- 1:n

    lasso.cverr <- c()
    for (fold in 1:cvf) {
      x.mat <- x.mat0[fold.id[[fold]],]
      y.vec <- y.vec0[fold.id[[fold]]]
      y.mat <- matrix(rep(y.vec, length(lamv)), nrow = length(y.vec), byrow = F)
      lasso.fit <- lasso.path.fun(y.vec, x.mat, lamv, imax, eps0)
      haty.mat <- exp(x.mat %*% lasso.fit$b.mat)
      lasso.cverr <- rbind(lasso.cverr, colSums((y.mat - haty.mat)^2))
    }
    lasso.cverr <- colMeans(lasso.cverr)
    hatgam <- hatlam <- lamv[which.min(lasso.cverr)]
    lasso.fit <- lasso.path.fun(y.vec0, x.mat0, hatlam, imax, eps0)
    haty.vec <- exp(x.mat1 %*% lasso.fit$b.mat)
    models[[modeliter]]$lasso.MSE[k] <- mean((y.vec1 - haty.vec)^2)

    mcl.cverr <- c()
    mclamv <- lamv[lamv >= hatlam]
    for (fold in 1:cvf) {
      x.mat <- x.mat0[fold.id[[fold]],]
      y.vec <- y.vec0[fold.id[[fold]]]
      y.mat <- matrix(rep(y.vec, length(mclamv)), nrow = length(y.vec), byrow = F)
      mcl.fit <- mcl.path.fun(y.vec, x.mat, 2.1, hatgam, mclamv, imax, eps0)
      haty.mat <- exp(x.mat %*% mcl.fit$b.mat)
      mcl.cverr <- rbind(mcl.cverr, colSums((y.mat - haty.mat)^2))
    }
    mcl.cverr <- colMeans(mcl.cverr)
    hatlam <- mclamv[which.min(mcl.cverr)]
    mcl.fit <- mcl.path.fun(y.vec0, x.mat0, 2.1, hatgam, hatlam, imax, eps0)
    haty.vec <- exp(x.mat1 %*% mcl.fit$b.mat)
    models[[modeliter]]$mcl.MSE[k] <- mean((y.vec1 - haty.vec)^2)

    if (n >= p) {
      df0 <- data.frame(y.vec0, x.mat0)
      df1 <- data.frame(y.vec1, x.mat1)
      unres.fit <- glm(y.vec0 ~ . - 1, data = df0, family = 'poisson')
      haty.vec <- predict(unres.fit, df1[,-1])
      models[[modeliter]]$unres.MSE[k] <- mean((y.vec1 - haty.vec)^2)
    }

    tsq <- toc()
    timer <- c(timer, tsq$toc - tsq$tic)
    k <- k + 1
    print(hatgam) ; print(hatlam)
    boxplot(models[[modeliter]][-(1:2)], main = names(models)[modeliter])
  }
  save(models, file = paste('/Users/', user, '/Dropbox/Seo&Chang/Simulations.RData', sep = ''))
}})
```