

Homework on optimization algorithms.

P8160 Advanced Statistical Computing

Problem 1:

Design an optimization algorithm to find the minimum of the continuously differentiable function

$$f(x) = -e^{-x} \sin(x)$$

on the closed interval $[0, 1.5]$. Write out your algorithm and implement it into **R**.

Answer: your answer starts here...

To find the minimum of a continuously function, we first make some changes to the function let

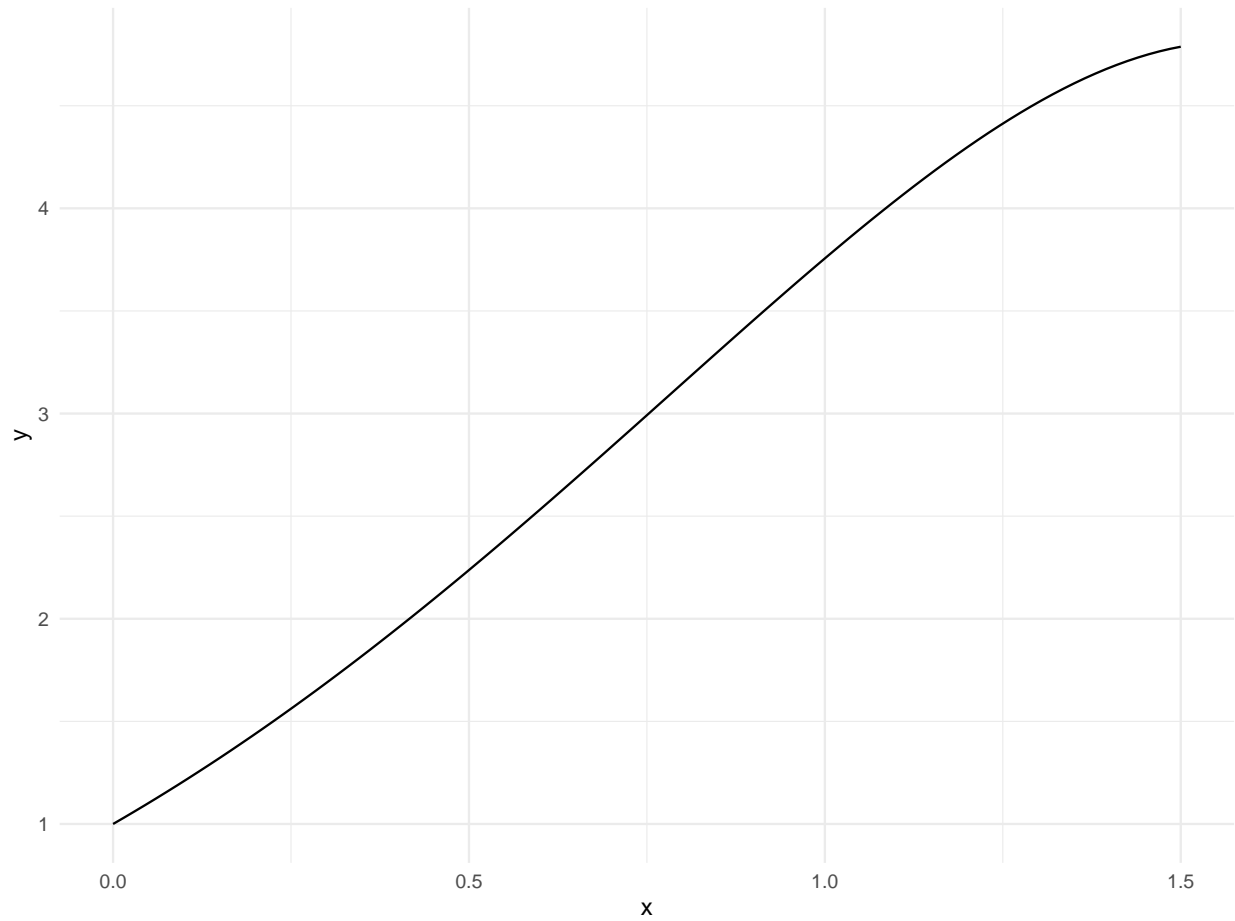
$$g(x) = e^x \sin(x)$$

and instead find the maximum of $g(x)$.

The gradient of $g(x)$ is :

$$\nabla g(x) = e^x (\sin(x) + \cos(x))$$

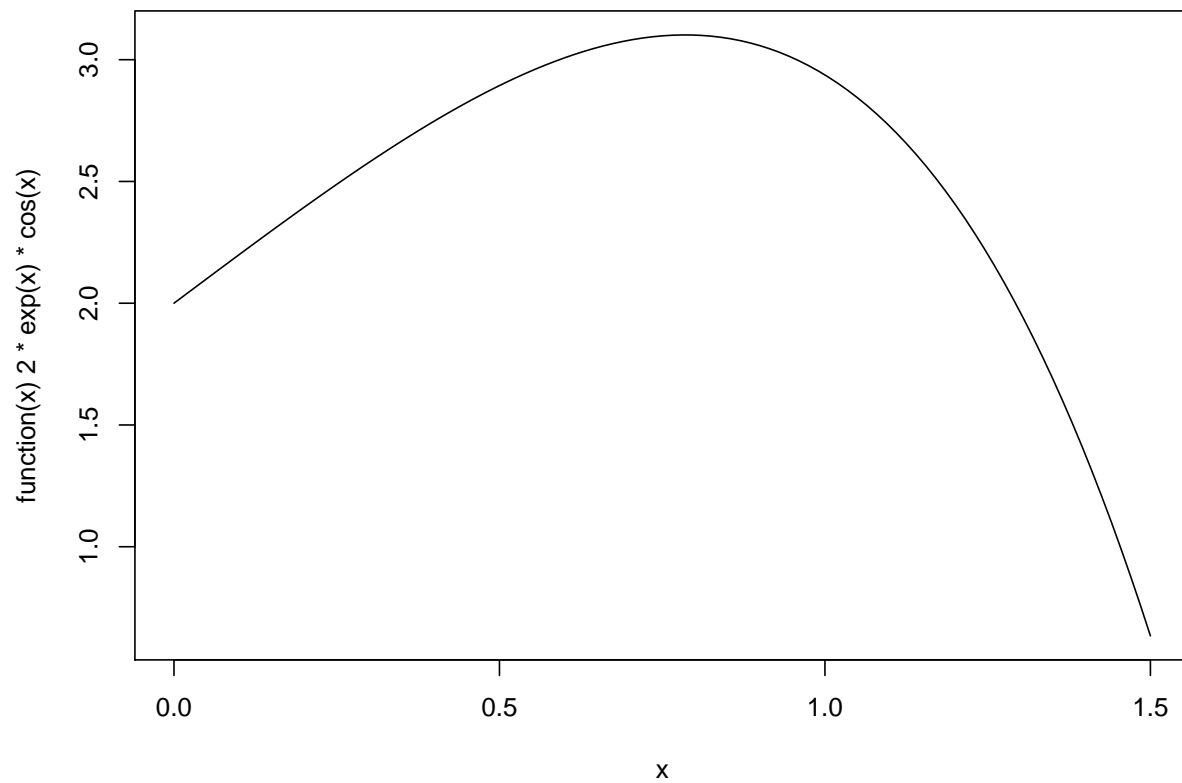
```
ggplot(tibble(x = seq(0,1.5,length = 10)),aes(x))+  
  geom_function(fun = function(x) exp(x)*(sin(x)+cos(x)))
```



and the Hessian is:

$$\nabla^2 g(x) = 2e^x \cos(x)$$

```
plot(function(x) 2*exp(x)*cos(x),xlim = c(0,1.5))
```



the hessian is greater than 0 everywhere in $[0,1.5]$, so we can't use Newton method.

```
goose_egg =
  function(
    fun,
    left = NULL,
    right = NULL,
    range = NULL,
    ratio = 0.618,
    tol = 10e-4,
    ...
  ){
    if (!any(left,right)){
      left = range[1]
      right = range[2]
    }

    mid_1 = left + ratio*(right - left)

    f_mid_1 = fun(mid_1)

    mid_2 = mid_1 + ratio*(right-mid_1)

    f_mid_2 = fun(mid_2)
```

```

f_left = fun(left)

f_right = fun(right)

i = 1

while (abs(f_left - f_right)>tol && i<1000){
  i = i + 1
  if (f_mid_1 < f_mid_2) {
    f_left = f_mid_1
    left = mid_1
  } else {
    f_right = f_mid_2
    right = mid_2
  }
  mid_1 = left + ratio * (right - left)
  f_mid_1 = fun(mid_1)
  mid_2 = mid_1 + ratio * (right - mid_1)
  f_mid_2 = fun(mid_2)
}
return(mean(mid_1,mid_2))
}

```

```

x_max = goose_egg(function(x) exp(x)*sin(x),range = c(0,1.5))

print(x_max)

```

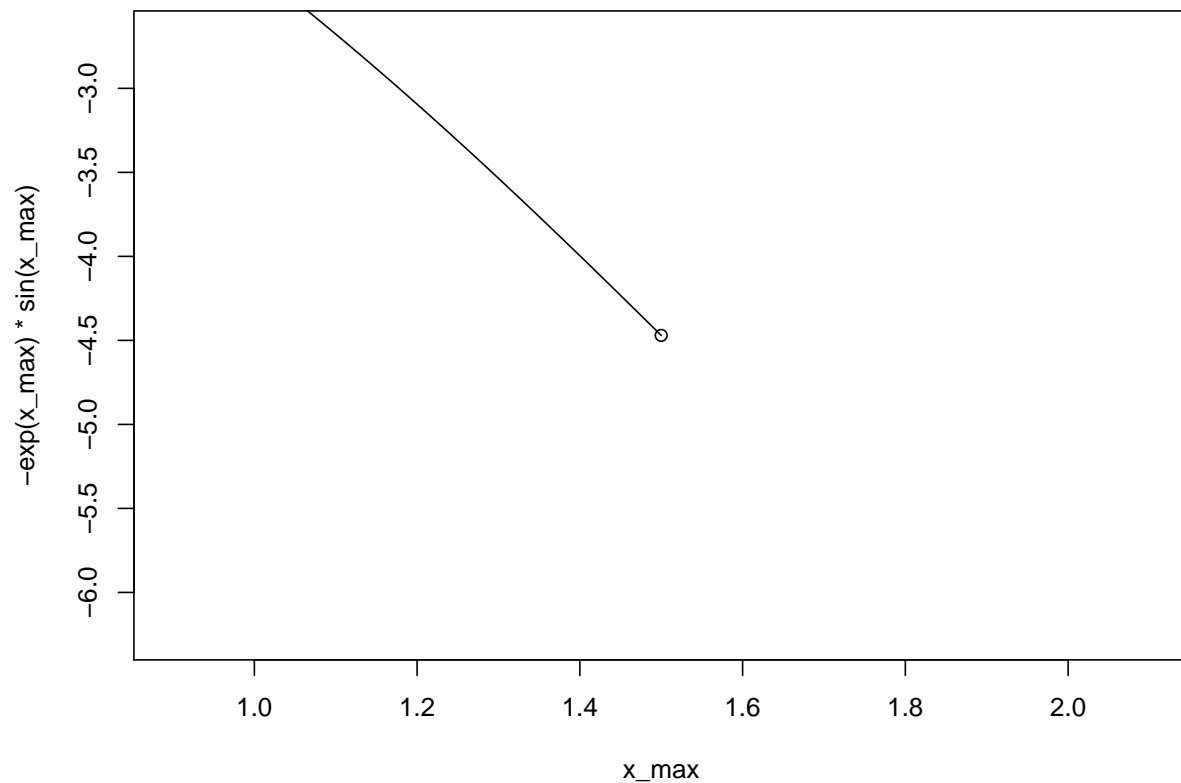
```
## [1] 1.5
```

```

plot(x_max,-exp(x_max)*sin(x_max))

plot(function(x) {-exp(x)*sin(x)}, xlim = c(0,1.5), add = T)

```



Problem 2:

The Poisson distribution, written as

$$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$$

for $\lambda > 0$, is often used to model “count” data — e.g., the number of events in a given time period.

A Poisson regression model states that

$$Y_i \sim \text{Poisson}(\lambda_i),$$

where

$$\log \lambda_i = \alpha + \beta x_i$$

for some explanatory variable x_i . The question is how to estimate α and β given a set of independent data $(x_1, Y_1), (x_2, Y_2), \dots, (x_n, Y_n)$.

1. Generate a random sample (x_i, Y_i) with $n = 500$ from the Poisson regression model above. You can choose the true parameters (α, β) and the distribution of X .
2. Write out the likelihood of your simulated data, and its Gradient and Hessian functions.
3. Develop a modified Newton-Raphson algorithm that allows the step-halving and re-direction steps to ensure ascent directions and monotone-increasing properties.
4. Write down your algorithm and implement it in R to estimate α and β from your simulated data.

Answer: your answer starts here...

2.1

```
print("hello world")
```

```
X = rbind(rep(1,500),rnorm(500))
Beta = runif(2)
lambda = exp(t(X)%*%Beta)
Y = map(lambda,~rpois(1,.x)) %>% unlist()
dat = list(y = Y, x=X)
ans = glm(Y~0+t(X),family = poisson())
```

2.2

- The log-likelihood of Poisson distribution is

$$l(Y; \lambda) = \sum \{y * \log(\lambda) - \lambda - \log(y!)\}$$

OR

$$l(Y; \alpha, \beta) = \sum \{y * (\alpha + x\beta) - \exp(\alpha + x\beta) - \log(y!)\}$$

- The Score funtion is

$$\nabla(Y; \alpha, \beta) = \frac{\partial}{\partial \lambda} l(Y; \lambda) = (\sum \{y - \exp(\alpha + x\beta)\}, \sum \{y * x - x * \exp(\alpha + x\beta)\})$$

- The hessian is

$$\begin{aligned} \nabla^2(Y; \lambda) &= \frac{\partial^2}{\partial \lambda^2} l(Y; \lambda) \\ &= \begin{pmatrix} \sum -\exp(\alpha + x\beta) & \sum -x * \exp(\alpha + x\beta) \\ \sum -x * \exp(\alpha + x\beta) & \sum -x^2 * \exp(\alpha + x\beta) \end{pmatrix} \end{aligned}$$

which is negative defined everywhere.

```
Poisson =
function(Y,X,
        theta_vec) {
  lambda = exp(t(X) %*% theta_vec)
  loglink = sum(Y * log(lambda) - lambda - log(factorial(Y)))

  fisher = var(Y * log(lambda) - lambda - log(factorial(Y)))

  gradient = c(sum(Y - lambda), sum(Y * X[2,] - X[2,] * lambda))

  hessian = matrix(c(
    sum(-lambda),
    sum(-X[2,] * lambda),
    sum(-X[2,] * lambda),
    sum((-X[2,] ^ 2) * lambda)
  ), ncol = 2)
```

```

    return(list(
      loglink = loglink,
      fisher = fisher,
      gradient = gradient,
      hessian = hessian
    ))
  }

Poisson(Y,X,c(7,2))

```

```

## $loglink
## [1] -3974589
##
## $fisher
##           [,1]
## [1,] 8.21e+08
##
## $gradient
## [1] -3980250 -7302954
##
## $hessian
##           [,1]      [,2]
## [1,] -3981198 -7.3e+06
## [2,] -7303206 -1.6e+07

```

2.3

the Newton method updating is:

$$\nabla g(x_{k+1}) = \nabla g(x_k) + \eta * \nabla^2 g(x_k)(x_{k+1} - x_k)$$

where η is the step size that ensure $\nabla g(x_{k+1}) > \nabla g(x_k)$

```

#Develop a modify Newton-Raphson algorithm that allows the
#step-halving and
#re-direction steps
#to ensure ascent directions and monotone-increasing properties.

newton_update =
  function(fun,
    previous_theta,
    y,x,
    step_size = 1,
    optimizer = F,
    backtracking = T,
    tol = 1e-8) {
    #take previous gradient and a updated hessian, return update gradient with
    #backtracking
    #if (abs(fun(y,x,previous_theta)$loglink) == Inf) stop("Check your log-likelihood")
    trial = 0

    gradient = fun(y,x,previous_theta)$gradient

```

```

if (is.function(optimizer)) {
  hessian = optimizer(y,x, fun,) # get H
} else{
  if (is.numeric(optimizer)) {
    H = optimizer # use H
  } else{
    hessian = fun(y,x, previous_theta)$hessian
    H = solve(hessian)
    while (all(eigen(H)$values > 0)) {# eigen decomposition
      P = eigen(hessian)
      lambda = max(P$values)
      hessian =
        P$vectors %*% (diag(P$values) - (lambda + tol) * diag(length(P$values))) %*%
        solve(P$vectors)

      H = solve(hessian)
    }
  }
}

#updating
cur_theta = previous_theta - step_size * H %*% gradient

#backtracking
while (backtracking & fun(y,x,cur_theta)$loglink < fun(y,x,previous_theta)$loglink & trial < 2000) {
  step_size = step_size / 2

  trial = trial + 1 # avoid dead loops

  cur_theta = previous_theta - step_size * H %*% gradient
}

return(cur_theta)
}

newton_update(fun = Poisson,previous_theta = c(7,2), y=Y,x =X)

```

```

##      [,1]
## [1,]    6
## [2,]    2

```

```

naive_newton =
function(fun,
  init_theta = 1,
  y,x,
  tol = 1e-8,
  maxtiter = 2000,
  optimizer = F,
  ...) {

  f = fun(y,x,init_theta)

  if (any(is.null(f$loglink),

```



```

      is.null(f$gradient),
      is.null(f$hessian))) {
    stop("fun input must return both gradient and hessian")
  }
  result = tibble()

  i = 0

  cur_theta = init_theta

  prevlog = -Inf # \nabla g(x_{k})

  while (any(abs(f$loglink)==Inf,abs(f$loglink - prevlog) > tol) && i < maxtiter) {
    i = i + 1
    prev_theta = cur_theta
    prevlog = f$loglink
    cur_theta = newton_update(fun, prev_theta,y,x)
    f = fun(y,x,cur_theta)
    result =
      rbind(result, tibble(
        iter = i,
        x_i = list(prev_theta),
        'g(x_i)' = prevlog
      ))
  }
  return(list(theta = cur_theta,result = result))
}

```

```

Beta_hat = naive_newton(Poisson,init_theta = c(7,2),Y,X)$theta

tibble(
  term = c("alpha", "beta"),
  theta = Beta,
  theta_hat = Beta_hat
) %>%
  knitr::kable()

```

term	theta	theta_hat
alpha	0.585	0.606
beta	0.234	0.247

Problem 3:

The data *breast-cancer.csv* have 569 row and 33 columns. The first column **ID** labels individual breast tissue images; The second column **Diagnosis** identifies if the image is coming from cancer tissue or benign cases (M=malignant, B = benign). There are 357 benign and 212 malignant cases. The other 30 columns correspond to mean, standard deviation and the largest values (points on the tails) of the distributions of the following 10 features computed for the cellnuclei;

- radius (mean of distances from center to points on the perimeter)

- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness ($\text{perimeter}^2 / \text{area} - 1.0$)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The goal is to build a predictive model based on logistic regression to facilitate cancer diagnosis;

1. Build a logistic model to classify the images into malignant/benign, and write down your likelihood function, its gradient and Hessian matrix.
2. Build a logistic-LASSO model to select features, and implement a path-wise coordinate-wise optimization algorithm to obtain a path of solutions with a sequence of descending λ 's.
3. Write a report to summarize your findings.

3

3.1

the data is a binomial outcome response, which follows a Bernoulli distribution, Using logit link, which

$$\log\left(\frac{p}{1-p}\right) = X^T \beta$$

s.t

$$p = \frac{\exp(X\beta)}{1 + \exp(X\beta)}$$

the log-likelihood of Bernoulli is

$$l(Y; \beta) = \sum \{y * \log\left(\frac{p}{1-p}\right) + \log(1-p)\} = \sum \{y * X^T \beta - \log(1 + \exp(X^T \beta))\}$$

The gradient is

$$\nabla l(Y; \beta) = \left(\frac{\partial}{\partial \beta_i} l(Y; \beta) \right) = \left(\sum (y * x_i - \frac{x_i \exp(X^T \beta)}{1 + \exp(X^T \beta)}) \right)$$

and the hessian is

$$\nabla^2 l(Y; \beta) = \left(\sum -\frac{x_i * x_j * \exp(X^T \beta)}{(1 + \exp(X^T \beta))^2} \right)$$

```

Bernoulli =
function(y,x,
        theta_vec){
  if (length(theta_vec)!=ncol(x)) stop("length of theta_vec must match dim of x")
  if (is.factor(y)) y = y %>% as.numeric()-1
  Y = y
  X = x
  loglink = sum(Y * X%%theta_vec - log(1+exp(X%%theta_vec)))

  if (abs(loglink) == Inf){
    stop("Choose a better starting value")
  }

  fisher = var(Y * X%%theta_vec - log(1+exp(X%%theta_vec)))

  X = x*1e-0

  gradient = map(1:length(theta_vec),
                 ~ sum(Y * X[, .x] - X[, .x] * exp(X %% theta_vec) /
                     (1 + exp(X %% theta_vec)))) %>% unlist()

  hessian =
    expand.grid(i = seq(1,length(theta_vec)),
               j = seq(1,length(theta_vec))) %>%
    summarise(beta = map2(i,j,~sum(-X[, .x]*X[, .y]*exp(X%%theta_vec)/(1+exp(X%%theta_vec))^2))) %>%
    unnest(beta) %>%
    pull(beta)

  hessian = matrix(hessian,ncol = length(theta_vec))

  return(
    list(loglink = loglink,
         fisher = fisher,
         gradient = gradient*1e+0,
         hessian = hessian*1e+0)
  )
}

Bernoulli(Y,X,rep(0,10))

```

```

## $loglink
## [1] -394
##
## $fisher
##      [,1]
## [1,]    0
##
## $gradient
## [1] 317.09  907.67 1707.73 -21099.85  5.60 -1.09 -8.82
## [8] -4.74  10.64  4.58
##
## $hessian
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]

```

```
## [1,] -30154 -39461 -196955 -1489947 -194.847 -2.23e+02 -205.45
## [2,] -39461 -55557 -257249 -1865996 -264.207 -2.94e+02 -258.39
## [3,] -196955 -257249 -1287036 -9765529 -1270.701 -1.47e+03 -1358.71
## [4,] -1489947 -1865996 -9765529 -78593927 -9101.146 -1.10e+04 -11005.14
## [5,] -195 -264 -1271 -9101 -1.349 -1.50e+00 -1.30
## [6,] -223 -294 -1467 -11036 -1.500 -1.94e+00 -1.85
## [7,] -205 -258 -1359 -11005 -1.300 -1.85e+00 -2.02
## [8,] -114 -141 -754 -6154 -0.713 -9.68e-01 -1.02
## [9,] -366 -498 -2387 -17084 -2.514 -2.81e+00 -2.44
## [10,] -125 -172 -815 -5750 -0.869 -9.62e-01 -0.82
##      [,8]      [,9]      [,10]
## [1,] -114.280 -366.09 -125.098
## [2,] -141.187 -498.30 -171.984
## [3,] -753.925 -2387.38 -815.186
## [4,] -6153.567 -17083.93 -5750.212
## [5,] -0.713 -2.51 -0.869
## [6,] -0.968 -2.81 -0.962
## [7,] -1.023 -2.44 -0.820
## [8,] -0.554 -1.33 -0.443
## [9,] -1.331 -4.78 -1.632
## [10,] -0.443 -1.63 -0.568
```

```
# Bernoulli(dat,ans$coefficients[-1] %>% as.vector())
```

3.2

```
soft_threshold =
  function(beta, lambda) {
    if (abs(beta)>lambda){
      if (beta >0) return(beta - lambda)
      return(beta +lambda)
    }
    return(0)
  }
```

```
bfun =
  function(y, x, theta_vec) {
    if (is.factor(y))
      y = y %>% as.numeric() - 1
    p_prev = exp(x %*% theta_vec) / (1 + exp(x %*% theta_vec)) # row matrix

    w_prev = p_prev * (1 - p_prev) # row matrix

    z_prev = x %*% theta_vec + (y - p_prev) / w_prev # row * 1 matrix

    loglink =
      #sum(w_prev * (z_prev - x %*% theta_vec) ^ 2) / (2 * length(y))
      sum(w_prev * (z_prev - x %*% theta_vec) ^ 2) / (2)

    return(list(
      loglink = loglink,
```

```

    p = p_prev,
    w = w_prev,
    z = z_prev
  ))
}

gfun = function(y,x,theta_vec){
  z = y
  w = rep(1/length(y),length(y))
  loglink = sum(w *(y - x %*% theta_vec) ^ 2) / 2
  return(
    list(z=z,w=w,loglink=loglink)
  )
}

```

```

Qlasso =
function(y,
  x,
  family_fun,
  lambda = 0,
  theta_vec = NA,
  maxiter = 500,
  tol = 1e-6,
  intercept = T,
  ...) {
  # data preparation
  if (is.factor(y))
    y = y %>% as.numeric() - 1
  y = y
  x = scale(x)
  xsd = attr(x, "scaled:scale") %>% as.vector()
  if (intercept) {
    x = cbind(rep(1, length(y)), x) # add alpha
    xsd = append(1, xsd)
  }

  # Manually choosing starting value
  if (!is.na(theta_vec)) {
    if (length(theta_vec) != ncol(x))
      theta_vec = append(rep(0, ncol(x) - length(theta_vec)), theta_vec)
  } else
    theta_vec = rep(0, ncol(x))

  #update part
  iter = 0

  cur_result = family_fun(y, x, theta_vec)$loglink

  if (abs(cur_result) == Inf|is.na(cur_result))
    stop("Diverge at starting value")

  prev_result = -Inf

  while (abs(cur_result - prev_result) > tol

```

```

    && iter < maxiter) {
  iter = iter + 1

  prev_result = cur_result

  for (i in 1:length(theta_vec)) {
    #weight
    fun_prev = family_fun(y, x, theta_vec)
    z_prev = fun_prev$z # working response
    w_prev = fun_prev$w # weight

    #update
    cur_theta =
      sum((z_prev - x[, -i] %% theta_vec[-i]) * x[, i] * w_prev)

    #soft-threshold
    if (i > 1) {
      cur_theta =
        soft_threshold(cur_theta,
                      lambda) / sum(w_prev * (x[, i] ^ 2))
    } else {
      cur_theta = cur_theta / sum(w_prev * (x[, i] ^ 2))
    }

    #update theta
    theta_vec[[i]] = cur_theta
  }
  cur_result = family_fun(y, x, theta_vec)$loglink + lambda * sum(abs(theta_vec))
}
return(list(coefficient = theta_vec / xsd,
           loglikelihood = cur_result))
}

```

```

cv_Qlasso =
function(y,x,family_fun,lambda=NA,number = 5,intercept = T){
  result = tibble()
  lambda = as.vector(lambda)
  if (all(is.na(lambda))) {
    Y = y
    if (is.factor(y)) Y = y %% as.numeric()-1
    Y = as.vector(Y)
    max_lambda = length(Y)*max(colSums(diag(Y) %% scale(x)))
    lambda = exp(seq(log(max_lambda),-10,length=20))
  }
  block_length = length(y)/number
  for (lam in lambda) {
    for (i in 1:number) {
      #create training partition
      test_index = (1 + (number-1)*block_length):(number*block_length)
      trainX = x[-test_index,]
      testX = x[test_index,]
      trainY = y[-test_index]
      testY = y[test_index]

```

```

#train
trainF = Qlasso(trainY,trainX,family_fun,lambda = lam,intercept = intercept)
if (intercept) {prev_coef = trainF$coefficient[-1]}else {prev_coef = trainF$coefficient}
#test
testF = family_fun(testY,testX,prev_coef)
result =
  result %>%
  rbind(expand.grid(coefficient = trainF$coefficient,
                    lambda = lam,
                    loglikelihood = testF$logli+ lam*sum(abs(trainF$coefficient))) %>%
        cbind(tibble(term = (1:length(trainF$coefficient))-1) %>%
              mutate_all(as.character)))
    )
}
}
# take the mean of everything
result = result %>%
  group_by(lambda,term) %>%
  summarise(across(c(coefficient,loglikelihood),mean))%>%
  arrange((loglikelihood)) %>%
  nest(coef = c(coefficient,term))

coef = result[1,] %>%
  unnest() %>%
  pull(coefficient)

lambda = result[1,"lambda"] %>% as.numeric()

loglikelihood = result[1,"loglikelihood"] %>% as.numeric()

return(list(
  coefficient = coef,
  loglikelihood = loglikelihood,
  lambda = lambda,
  cvtable = result
))
}

```

```

tibble(glm = glm(Y_g~X_g)$coefficient,
  glmnet_0lambda = coef(glmnet(X_g,Y_g,lambda = 0)) %>% as.vector(),
  Qlasso_0lambda = Qlasso(Y_g,X_g,gfun)$coefficient,
  Qlasso_01lambda = Qlasso(Y_g,X_g,gfun,5)$coefficient,
  glmnet_01lambda = coef(glmnet(X_g,Y_g,lambda = 5)) %>% as.vector()
) %>%
knitr::kable(caption = "Function comparison at Gaussian Family")

```

Table 2: Function comparison at Gaussian Family

glm	glmnet_0lambda	Qlasso_0lambda	Qlasso_01lambda	glmnet_01lambda
0.076	0.076	3.306	3.306	2.480
6.042	6.042	6.042	2.061	2.082

glm	glmnet_0lambda	Qlasso_0lambda	Qlasso_01lambda	glmnet_01lambda
6.986	6.986	6.986	2.533	2.556
3.413	3.413	3.413	0.000	0.000
6.519	6.519	6.519	3.248	3.265
0.504	0.504	0.504	0.000	0.000
8.601	8.601	8.601	4.482	4.503
6.227	6.227	6.227	0.524	0.554
0.305	0.305	0.305	0.000	0.000
7.785	7.785	7.785	3.410	3.431
0.469	0.469	0.469	0.000	0.000

```
tibble(glm = glm(Y~X,family = binomial())$coefficient,
  glmnet_0lambda = coef(glmnet(X,Y,"binomial",lambda = 0)) %>% as.vector(),
  Qlasso_0lambda = Qlasso(Y,X,bfun)$coefficient,
  Qlasso_56.9lambda = Qlasso(Y,X,bfun,56.9)$coefficient,
  glmnet_01lambda = coef(glmnet(X,Y,"binomial",lambda =0.1)) %>% as.vector()
) %>%
knitr::kable(caption = "Function comparision at binomial family")
```

Table 3: Function comparision at binomial family

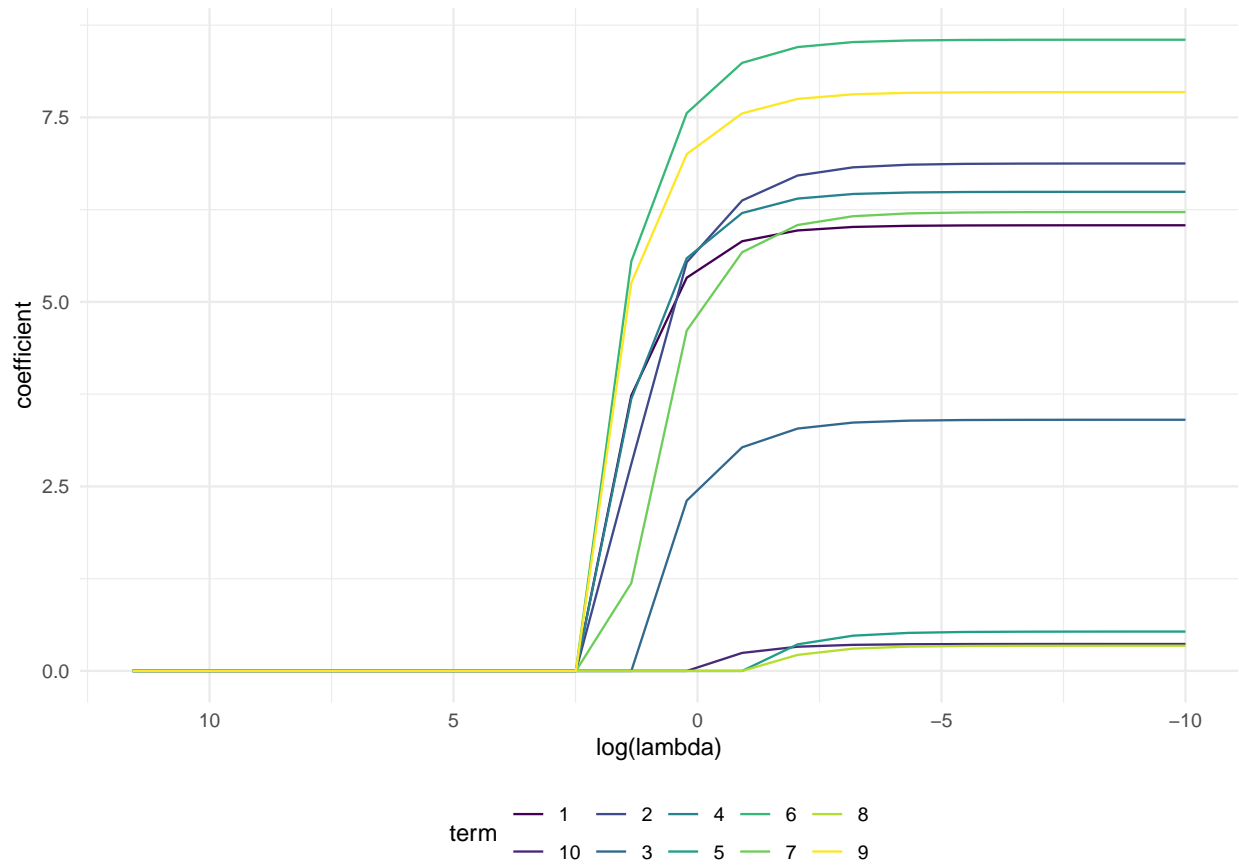
glm	glmnet_0lambda	Qlasso_0lambda	Qlasso_56.9lambda	glmnet_01lambda
7.360	8.317	-0.244	0.587	4.282
2.049	1.608	1.814	0.000	0.000
-0.385	-0.384	-0.381	-0.009	-0.009
0.072	0.120	-0.010	-0.022	-0.022
-0.040	-0.038	-0.031	0.000	0.000
-76.432	-75.580	-78.093	0.000	0.000
1.462	0.452	6.939	0.000	0.000
-8.469	-8.730	-9.772	0.000	0.000
-66.822	-66.866	-62.523	-29.644	-29.656
-16.278	-16.226	-16.658	0.000	0.000
68.337	67.619	52.700	0.000	0.000

From above tables we can see that the **Qlasso** function of mine works reasonably close to **glmnet** except for the intercept terms in the case of Gaussian family. However, the result of binomial family behave rather different with L1 penalties. The function produce similar result with $\lambda = 0$ except for **intercept** and once adding λ , the effect of λ is rather different to **glmnet**.

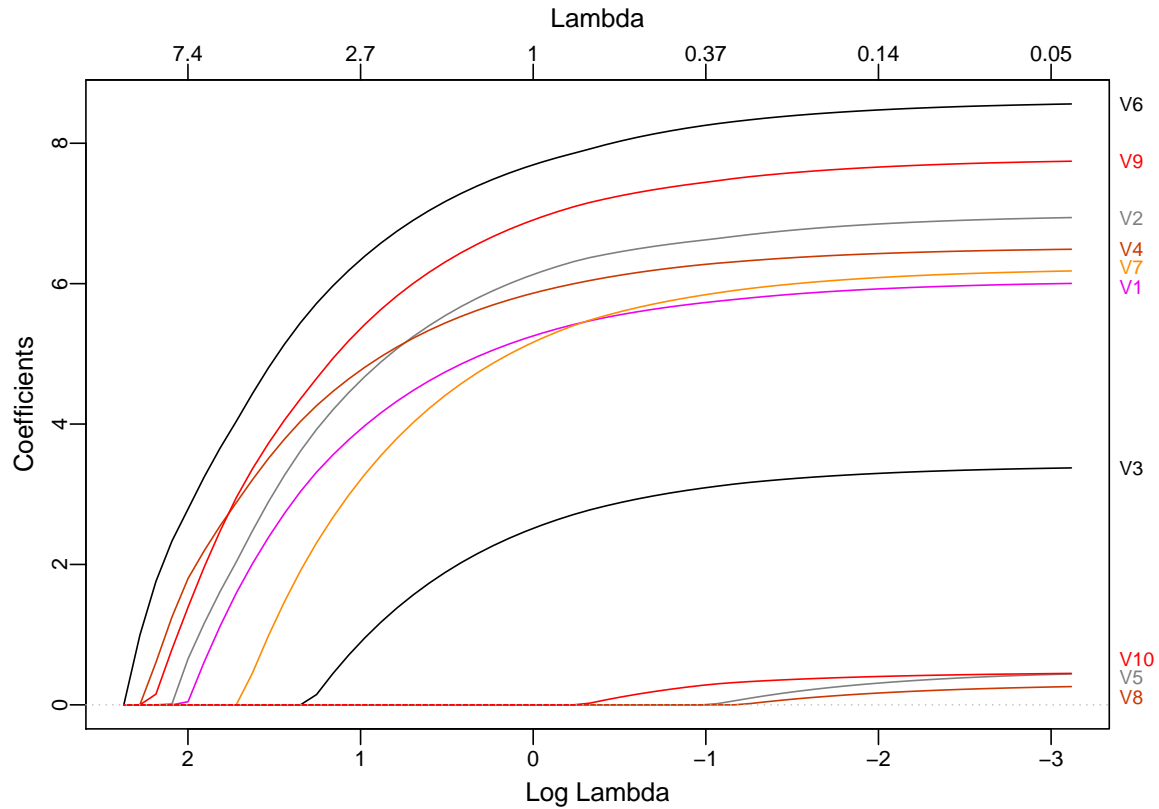
```
gaussian_Qlasso = cv_Qlasso(Y_g,X_g,gfun)

gaussian_Qlasso$cvtable %>%
  unnest(coef) %>%
  filter(term != 0) %>%
  ggplot(aes(x = log(lambda), y = coefficient, color = term)) +
  geom_line() +
  scale_x_reverse() +
  labs(title = "Cross validate Qlasso selection at Gaussian Family")
```


Cross validate Qlasso selection at Gaussian Family



```
gaussian_glmnet = cv.glmnet(X_g,Y_g)
plotmo::plot_glmnet(gaussian_glmnet$glmnet.fit)
```



```
tibble(term = c("intercept", colnames(X_g %>% as_tibble()))),
  Qlasso = gaussian_Qlasso$coef,
  glmnet = coef(gaussian_glmnet) %>% as.vector() %>%
  knitr::kable(caption = "Gaussian family result")
```

Table 4: Gaussian family result

term	Qlasso	glmnet
intercept	4.056	0.123
V1	6.038	5.961
V2	0.365	6.892
V3	6.876	3.333
V4	3.403	6.457
V5	6.492	0.370
V6	0.533	8.514
V7	8.553	6.130
V8	6.218	0.211
V9	0.342	7.699
V10	7.843	0.424

Using cross validation we can see that for gaussian, the penalty is in the same scale as the `glmnet`, but because

of the intercept is different, the optimum model chosen is different. same result(bug) can be observed in the binomial family, but rather deteriorate. The penalty although work similar to `glmnet` in terms of trends, but on a rather different scale(sample size). Also, the intercept produce by the function is not even the $\log(\frac{\bar{Y}}{1-\bar{Y}})|X = 0$, so the objective function failed to aid the function to choose the optimum model.

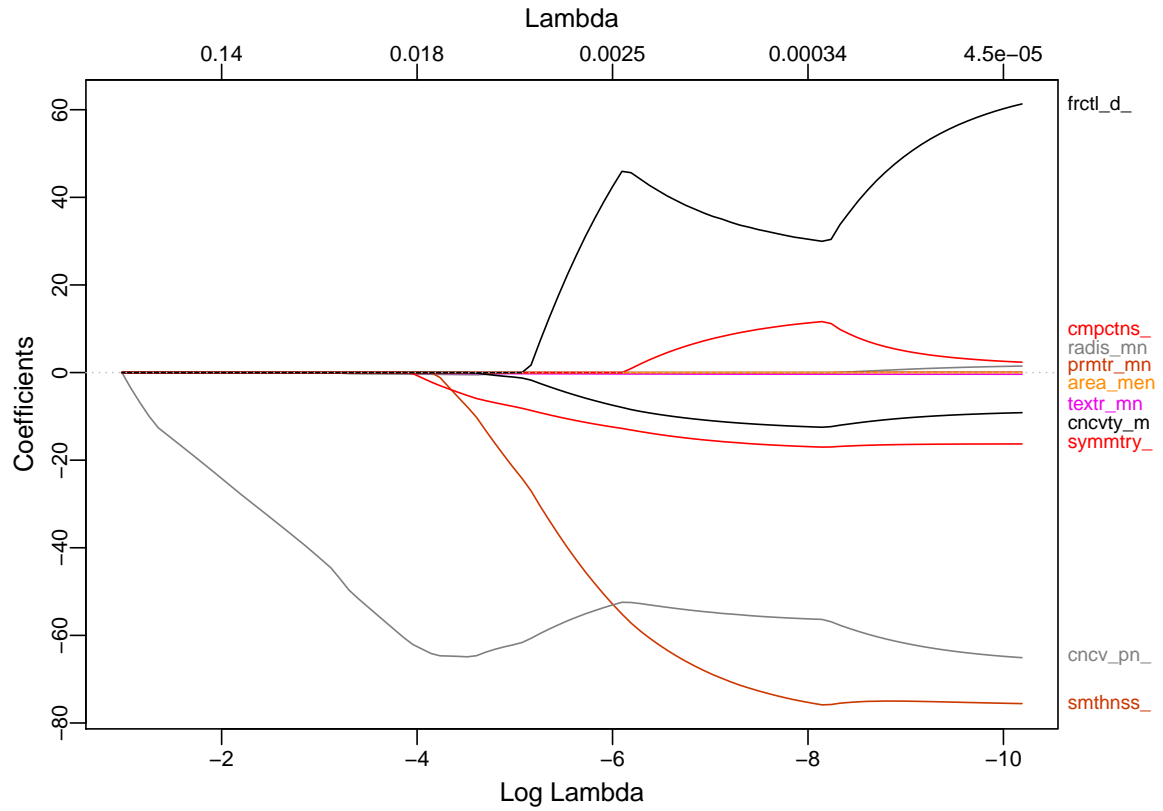
```
binomial_Qlasso = cv_Qlasso(Y,X,bfun)
```

```
binomial_Qlasso$cvtable %>%
  unnest(coef) %>%
  filter(term != 0) %>%
  ggplot(aes(x = log(lambda/length(Y)), y = coefficient, color = term)) +
  geom_line() +
  scale_x_reverse() +
  labs(title = "Cross validate Qlasso selection at Binomial Family")
```



```
binomial_glmnet = cv.glmnet(X,Y,family = "binomial")
```

```
plotmo::plot_glmnet(binomial_glmnet$glmnet.fit)
```



```
tibble(term = c("intercept", colnames(X)),
  Qlasso = binomial_Qlasso$coef,
  glmnet = coef(binomial_glmnet) %>% as.vector()) %>%
  knitr::kable(caption = "Binomial family result")
```

Table 5: Binomial family result

term	Qlasso	glmnet
intercept	0.373	14.867
radius_mean	0.000	-0.438
texture_mean	0.000	-0.201
perimeter_mean	0.000	0.000
area_mean	0.000	0.000
smoothness_mean	0.000	-3.431
compactness_mean	0.000	0.000
concavity_mean	0.000	0.000
concave_points_mean	0.000	-64.722
symmetry_mean	0.000	-3.772
fractal_dimension_mean	0.000	0.000

3.3

The result from