Question 1.

$$\vec{\nabla} f(x,y) = (\partial_x, \partial_y) f(x,y)$$
$$= (-2x + 2x \cdot 2(x^2+2y^2-1),\ 8y^3 + 4y \cdot 2(x^2+2y^2-1))$$
$$= (-2x + 4x(x^2+2y^2-1),\ 8y^3 + 8y(x^2+2y^2-1))$$
$$= (-2x + 4x^3 + 8xy^2 - 4x,\ 8y^3 + 8yx^2 + 16y^3 - 8y)$$
$$= (4x^3 + 8xy^2 - 6x,\ 24y^3 + 8yx^2 - 8y)$$

$$\vec{\nabla} f(x,y) = 0 \Rightarrow \begin{cases} 4x^3 + 8xy^2 - 6x = 0 & ① \\ 24y^3 + 8yx^2 - 8y = 0. & ② \end{cases}$$

$$\Rightarrow \begin{cases} 2x^3 + 4xy^2 - 3x = 0 \\ 3y^3 + yx^2 - y = 0 \end{cases}$$

$$\Rightarrow \begin{cases} x(2x^2 + 4y^2 - 3) = 0. \\ y(3y^2 + x^2 - 1) = 0. \end{cases}$$

$$\Rightarrow ① \begin{cases} x = 0. \\ y = 0 \end{cases} \quad ② \begin{cases} x = 0. \\ 3y^2 + x^2 - 1 = 0 \Rightarrow y = \pm\frac{\sqrt{3}}{3} \end{cases} \quad ③ \begin{cases} 2x^2 + 4y^2 - 3 = 0 \\ \Rightarrow x = \pm\sqrt{\frac{3}{2}} = \pm\frac{\sqrt{6}}{2} \\ y = 0 \end{cases}$$

$$④ \begin{cases} 2x^2 + 4y^2 - 3 = 0 \\ 3y^2 + x^2 - 1 = 0 \Rightarrow x^2 = 1 - 3y^2. \end{cases} \Rightarrow 2 - 6y^2 + 4y^2 - 3 = 0.$$
$$2y^2 = -1 \Rightarrow y^2 = -\frac{1}{2} \text{ impossible.}$$

Hence by the definition of the stationary point

$(0,0)$, $(0, \pm\frac{\sqrt{3}}{3})$, $(\pm\frac{\sqrt{6}}{2}, 0)$ are the stationary point.

Compute the Hessian.

$$\nabla^2 f(x,y) = \begin{pmatrix} 12x^2 + 8y^2 - 6 & 16xy \\ 16xy & 72y^2 + 8x^2 - 8 \end{pmatrix}$$

$$\nabla^2 f(0,0) = \begin{pmatrix} -6 & 0 \\ 0 & -8 \end{pmatrix}$$ here eigen values are $-6<0$ and $-8<0$.

- hence the $\nabla^2 f(0,0) < 0$, so $(0,0)$ is a strict local maximum

$$\nabla^2 f\left(0, \pm\frac{\sqrt{3}}{3}\right) = \begin{pmatrix} 8\times\frac{1}{3} - 6 & 0 \\ 0 & 72\times\frac{1}{3} - 8 \end{pmatrix}$$

$$= \begin{pmatrix} -\frac{10}{3} & 0 \\ 0 & 16 \end{pmatrix}$$ here eigen values are $-\frac{10}{3} < 0$ and $16 > 0$.

- hence $\nabla^2 f\left(0, \pm\frac{\sqrt{3}}{3}\right)$ are undefined, so $\left(0, \pm\frac{\sqrt{3}}{3}\right)$ are saddle points

$$\nabla^2 f\left(\pm\frac{\sqrt{6}}{2}, 0\right) = \begin{pmatrix} 12\times\frac{6}{4} - 6 & 0 \\ 0 & 8\times\frac{6}{4} - 8 \end{pmatrix}$$

$$= \begin{pmatrix} 12 & 0 \\ 0 & 4 \end{pmatrix}$$ here eigen values $12 > 0$ $4 > 0$

- hence $\nabla^2 f\left(\pm\frac{\sqrt{6}}{2}, 0\right) > 0$, so $\left(\pm\frac{\sqrt{6}}{2}, 0\right)$ are strict local minimum

Question 2.

i) show $\nabla l(\underline{\theta}) = \underline{X}^T(\underline{y} - \hat{\underline{y}}(\underline{\theta}))$

since $\underline{X} = \begin{pmatrix} -\underline{x}_1^T- \\ \vdots \\ -\underline{x}_n^T- \end{pmatrix}$ $\underline{x}_i^T \Rightarrow 1\times 3$ vector

so $\underline{x}_i \Rightarrow 3\times 1$ vector.

$$l(\underline{\theta}) = \log L(\theta) = \sum_{i=1}^{n} y_i \log h(\underline{x}_i^T\underline{\theta}) + (1-y_i)\log(1-h(\underline{x}_i^T\underline{\theta}))$$

$$\nabla l(\underline{\theta}) = \sum_{i=1}^{n} y_i \underline{x}_i \cdot h'(\underline{x}_i^T\underline{\theta}) \frac{1}{h(\underline{x}_i^T\underline{\theta})} + (1-y_i)\underline{x}_i\left(-h'(\underline{x}_i^T\underline{\theta})\right)\frac{1}{1-h(\underline{x}_i^T\underline{\theta})}$$

$$= \sum_{i=1}^{n} y_i \underline{x}_i h'(\underline{x}_i^T\underline{\theta})\frac{1}{h(\underline{x}_i^T\underline{\theta})} - (1-y_i)\underline{x}_i \frac{h'(\underline{x}_i^T\underline{\theta})}{1-h(\underline{x}_i^T\underline{\theta})}$$

$$= \sum_{i=1}^{n} \underline{x}_i \left(\frac{y_i h'(\underline{x}_i^T\underline{\theta})}{h(\underline{x}_i^T\underline{\theta})} - \frac{(1-y_i)h'(\underline{x}_i^T\underline{\theta})}{1-h(\underline{x}_i^T\underline{\theta})}\right)$$

$$= \sum_{i=1}^{n} \underline{x}_i \left(\frac{y_i h'(\underline{x}_i^T\underline{\theta}) - y_i h'(\underline{x}_i^T\underline{\theta})h(\underline{x}_i^T\underline{\theta}) - (1-y_i)h'(\underline{x}_i^T\underline{\theta})h(\underline{x}_i^T\underline{\theta})}{h(\underline{x}_i^T\underline{\theta})(1-h(\underline{x}_i^T\underline{\theta}))}\right)$$

$$= \sum_{i=1}^{n} \underline{x}_i \left(\frac{y_i h'(\underline{x}_i^T\underline{\theta}) - (y_i+1-y_i)h'(\underline{x}_i^T\underline{\theta})h(\underline{x}_i^T\underline{\theta})}{h(\underline{x}_i^T\underline{\theta})(1-h(\underline{x}_i^T\underline{\theta}))}\right)$$

$$= \sum_{i=1}^{n} \underline{x}_i \left(\frac{(y_i - h(\underline{x}_i^T\underline{\theta}))h'(\underline{x}_i^T\underline{\theta})}{h(\underline{x}_i^T\underline{\theta})(1-h(\underline{x}_i^T\underline{\theta}))}\right)$$

$$h(t) = \frac{1}{1+e^{-t}} \qquad h'(t) = -1\cdot e^{-t}\cdot(-1)\frac{1}{(1+e^{-t})^2}.$$

$$= \frac{e^{-t}}{(1+e^{-t})^2}$$

hence $h'(t) = \frac{e^{-t}}{(1+e^{-t})^2} = \frac{e^{-t}}{1+e^{-t}}\cdot\frac{1}{1+e^{-t}} = h(t)(1-h(t))$

so $h'(\underline{x}_i^T\underline{\theta}) = h(\underline{x}_i^T\underline{\theta})(1-h(\underline{x}_i^T\underline{\theta}))$

▶ so $\nabla l(\theta) = \sum_{i=1}^{n} \underline{x}_i (y_i - h(\underline{x}_j^T\underline{\theta}))$

$$= \sum_{i=1}^{n} \underline{x}_i (y_i - \hat{y}_i) = \underline{X}^T(\underline{y} - \hat{\underline{y}}(\underline{\theta}))$$

OPT_CW_Q2

Weicheng YU

11/7/2021

i

The prove of the gradient of loglike(theta) is above,Write the function to compute the gradient and compute the gradient in theta=(0,0,0)

```
load("/Users/apple/Downloads/CW1_PimaData.rda")
h <- function(tt){ 1/(1+exp(-tt))
}
X_pos = X[y==1,]
X_neg = X[y==0,]
loglike <- function(theta){ sum(log(h(X_pos%*%theta)))+sum(log(1-h(X_neg%*%theta)))
}

gradient_log<-function(theta){
  y_hat<-h(X%*%theta)
  gradient<-t(X)%*%(y-y_hat)
  rownames(gradient)<-NULL
  colnames(gradient)<-NULL
  return(gradient)
}
theta_initial=c(0,0,0)
grad_initial<-gradient_log(theta_initial)
cat("The gradient of l(theta) at theta=c(0,0,0) is",grad_initial)
```

```
## The gradient of l(theta) at theta=c(0,0,0) is -24.5 -816 -753.5
```

Then check the gradient numerically by finite difference approximation

```
library(numDeriv)
central_diff<-function(theta,eps){
  d1<-(loglike(theta+c(eps,0,0))-loglike(theta-c(eps,0,0)))/(2*eps)
  d2<-(loglike(theta+c(0,eps,0))-loglike(theta-c(0,eps,0)))/(2*eps)
  d3<-(loglike(theta+c(0,0,eps))-loglike(theta-c(0,0,eps)))/(2*eps)
  return(c(d1,d2,d3))
}
central_diff(theta_initial,eps=10^-5)
```

```
## [1]  -24.5 -816.0 -753.5
```

These two results are almost the same so that the gradient function I build is reasonable.

ii

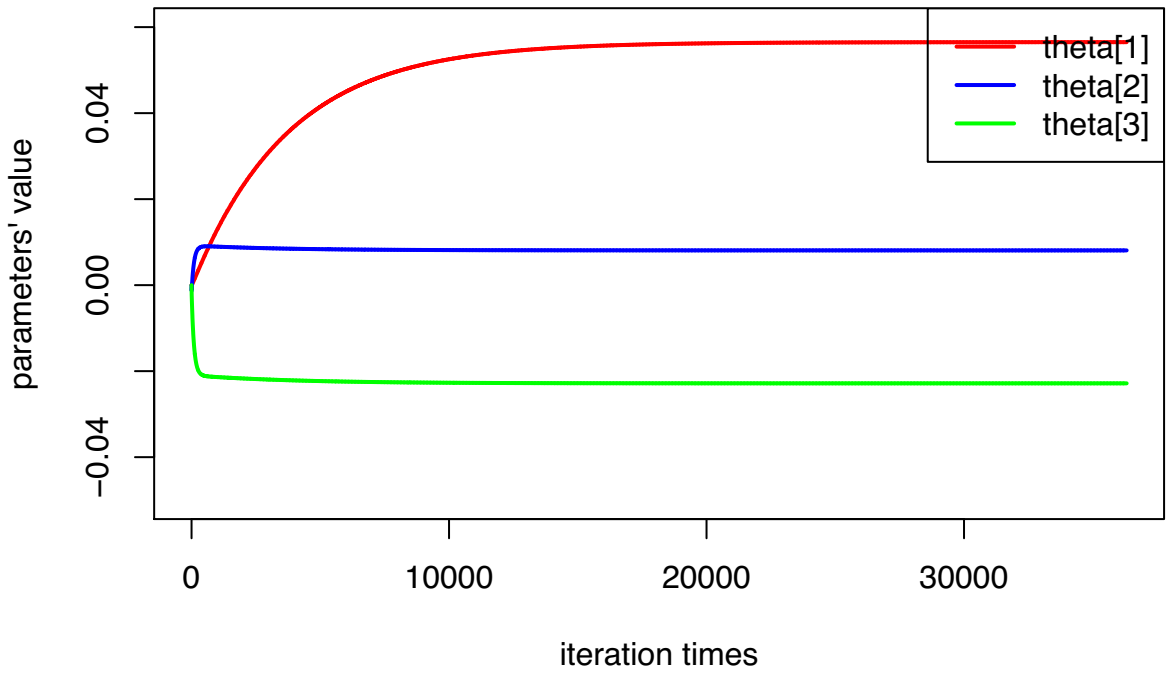
Using the gradient descent method with fixed stepsize to find the maximum likelihood estimates of theta and calculate the number of iterations required for converges

```
graddesc_const_step <- function(g,x0,tbar,epsilon, maxiter=10^6){
  # Gradient method with constant stepsize. The method
  # terminates when ||g(x)|| <epsilon
  # INPUT
  # g - gradient of the objective function
  # x0 - initial point
  # t  -  constant stepsize
  # epsilon ... tolerance parameter
  # maxiter - the maximum number of iterations

  ####
  # OUTPUT
  # x.opt = optimal solution (up to a tolerance) of max f(x)
  # iter = the number of the iterations
  # trajectory = the record of the change of the x

  x=x0
  grad=g(x)
  iter=1
  trajectory <- matrix(0, nr=maxiter, nc=length(x0))
  trajectory[1,]<-x
  while (sum(grad^2)>epsilon^2&&iter<maxiter){
    iter=iter+1
    x=x+tbar*grad
    grad=g(x);
    trajectory[iter,]<-x
  }
  return(list(x.opt=x, trajectory=trajectory[1:iter,], iter=iter-1))
}
out1<-graddesc_const_step(gradient_log,theta_initial,tbar = 10^-6,epsilon=10^-3)
cat("The maximum likelihood estimates of theta by this method is","\n",
    out1$x.opt,"\n",
    "And the corresponding value of loglike(theta) is",loglike(out1$x.opt),"\n",
    "The number of iterations required is",out1$iter)
```

```
## The maximum likelihood estimates of theta by this method is
##  0.05650613 0.008079365 -0.02284138
##  And the corresponding value of loglike(theta) is -64.85019
##  The number of iterations required is 36315
```

Then we plot the trajectory of the three parameters

```
plot(c(1:(out1$iter+1)),out1$trajectory[,1],type="l",ylab="parameters' value",xlab="iteration times",col
lines(c(1:(out1$iter+1)),out1$trajectory[,2],type="l",col="blue",lwd=2)
lines(c(1:(out1$iter+1)),out1$trajectory[,3],type="l",col="green",lwd=2)
legend("topright",c("theta[1]","theta[2]","theta[3]"),col=c("red","blue","green"),lty=1,lwd=2)
```



iii

Try step-size 10^-5

```
out7<-graddesc_const_step(gradient_log,theta_initial,tbar = 10^-5,epsilon=10^-3)
print(paste("The number of iterations required is",out7$iter))
```

```
## [1] "The number of iterations required is 999999"
```

I tried the larger step-size like tbar=10^-5 but the algorithm failed to converges(reaches the maxiter). I think the reason of it is that as the step-size is too large, it may overshoot the maximum then cause the faile of converge.

iv

Then we try the gradient method with backtracking.The parameters s=1,alpha=0.25,beta=0.5

```
graddesc_backtracking <- function(f,g,x0,s, alpha,  beta, epsilon, maxiter=10^6){
  # Gradient method with backtracking. The method
  # terminates when ||g(x)|| <epsilon
  # INPUT
  # x0 - initial point
  # s ......... initial choice of stepsize
  # alpha ..... tolerance parameter for the stepsize selection
  # beta ...... the constant in which the stepsize is multiplied
  #              at each backtracking step (0<beta<1)
  # epsilon ... tolerance parameter
  # maxiter - the maximum number of iterations
```

```
  ####
  # OUTPUT
  # x.opt = optimal solution (up to a tolerance) of max f(x)
  # iter = the number of the iterations
  # trajectory = the record of the change of the x
  stopifnot(beta>0, beta<1, alpha>0, alpha<1)

  x=x0
  grad=g(x)
  fun_val=f(x)
  iter=1
  trajectory <- matrix(0, nr=maxiter, nc=length(x0))
  trajectory[1,]<-x
  while (sum(grad^2)>epsilon^2&&iter<maxiter){
    iter=iter+1
    tbar=s
    while (f(x+tbar*grad)-fun_val<alpha*tbar*sum(grad^2)){
      tbar=beta*tbar
    }
    x=x+tbar*grad
    fun_val=f(x)
    grad=g(x);
    trajectory[iter,]<-x
  }
  return(list(x.opt=x, trajectory=trajectory[1:iter,], iter=iter-1))
}
out2<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.25,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta by this method is","\n",
    out2$x.opt,"\n",
    "And the corresponding value of loglike(theta) is",loglike(out2$x.opt),"\n",
    "The number of iterations required is",out2$iter)
```

```
## The maximum likelihood estimates of theta by this method is
##  0.05650686 0.008079352 -0.02284141
##  And the corresponding value of loglike(theta) is -64.85019
##  The number of iterations required is 7835
```

We could find the gradient method with backtracking is obviously more effienct than the method with fixed step.

Then we try other values of s,alpha and beta. Firstly we can try to change alpha and control other values same

```
##Alpha=0.1
try1<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.1,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try1$x.opt,"\n",
    "The number of iterations required is",try1$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650696 0.008079348 -0.02284141
##  The number of iterations required is 7881
```

```r
##Alpha=0.3
try2<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.3,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try2$x.opt,"\n",
    "The number of iterations required is",try2$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650674 0.008079352 -0.0228414
##  The number of iterations required is 6164
```

```r
##Alpha=0.5
try3<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.5,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try3$x.opt,"\n",
    "The number of iterations required is",try3$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650621 0.008079363 -0.02284138
##  The number of iterations required is 2815
```

```r
##Alpha=0.8
try4<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.8,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try4$x.opt,"\n",
    "The number of iterations required is",try4$iter)
```

```
## The maximum likelihood estimates of theta is 0.0565062 0.008079363 -0.02284138
##  The number of iterations required is 1930
```

```r
##Alpha=0.9
try5<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.9,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try5$x.opt,"\n",
    "The number of iterations required is",try5$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650624 0.008079363 -0.02284139
##  The number of iterations required is 2833
```

Then I change the value of beta

```r
##Beta=0.1
try6<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.25,beta=0.1,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try6$x.opt,"\n",
    "The number of iterations required is",try6$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650651 0.008079358 -0.0228414
##  The number of iterations required is 9373
```

```r
##Beta=0.2
try7<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.25,beta=0.2,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try7$x.opt,"\n",
    "The number of iterations required is",try7$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650622 0.008079363 -0.02284138
##  The number of iterations required is 2082
```

```
##Beta=0.8
try8<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1,alpha=0.25,beta=0.8,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try8$x.opt,"\n",
    "The number of iterations required is",try8$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650737 0.00807934 -0.02284143
##  The number of iterations required is 8547
```

Finally,we try to change the value of s

```
##s=0.8
try9<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=0.8,alpha=0.25,beta=0.5,epsilon=10^-3)
cat("The maximum likelihood estimates of theta is",try9$x.opt,"\n",
    "The number of iterations required is",try9$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650654 0.008079356 -0.0228414
##  The number of iterations required is 6415
```

```
##s=1.2
try10<-graddesc_backtracking(loglike,gradient_log,theta_initial,s=1.2,alpha=0.25,beta=0.5,epsilon=10^-3
cat("The maximum likelihood estimates of theta is",try10$x.opt,"\n",
    "The number of iterations required is",try10$iter)
```

```
## The maximum likelihood estimates of theta is 0.05650624 0.008079363 -0.02284139
##  The number of iterations required is 856
```

Discussion:

1. Changing the value of the s,alpha and beta will influence the iteration numbers(the efficiency of the backtracking algorithm) and will slightly change the final optimal value of theta.
2. As the value of alpha, beta and s is around 0.8, 0.2 and 1.2 repectively, the algorithm needs the less number of iterations. However, if we want to find the accurate optimal value of alpha, beta and s to make the iteration times reach the least, we need further research.

v The way to compute the Hessian matrix is on the last page of Q2

Then we build the function of the Hessian matrix of loglike

```
diff_h<-function(tt){
  h(tt)*(1-h(tt))#I have proved it in question i
}
Hessian<-function(theta){
  hess<-matrix(0,nr=length(theta),nc=length(theta))
  for(i in 1:length(theta)){
    for(j in 1:length(theta)){
      hess[i,j]=-t(X[,i])%*%diag(c(diff_h(X%*%theta)))%*%X[,j]
    }
  }
  return(hess)
}
Hessian(theta_initial)
```

```
##           [,1]       [,2]       [,3]
## [1,]   -850.25  -14343.25   -8262.25
## [2,] -14343.25 -375875.50 -205490.75
## [3,]  -8262.25 -205490.75 -127827.25
```

Then let me check the Hessian function.

```
numDeriv::hessian(loglike,theta_initial)
```

```
##             [,1]       [,2]       [,3]
## [1,]   -850.2502  -14343.25   -8262.25
## [2,] -14343.2500 -375875.50 -205490.75
## [3,]  -8262.2499 -205490.75 -127827.25
```

Thses two results are quite approximate, hence the Hessian function I produce is reasonable.

vi

With this Hessian function we could implement a Pure Newton's method to find the maximum likelihood estimate of theta.

```
pure_Newton <- function(g,H, x0, epsilon, maxiter=10^6){
  # Pure Newton's method. The method
  # terminates when ||g(x)|| <epsilon
  # INPUT
  # g - gradient of the objective function
  # H - a function to compute the Hessian matrix
  # x0 - initial point
  # epsilon ... tolerance parameter
  # maxiter - the maximum number of iterations

  ####
  # OUTPUT
  # x.opt = optimal solution (up to a tolerance) of max f(x)
  # iter = the number of the iterations
  # trajectory = the record of the change of the x

  x=x0
  trajectory <- matrix(0, nr=maxiter, nc=length(x0))
  trajectory[1,]<-x
  iter<-1
  while (sum(g(x)^2)>epsilon^2&& iter<maxiter){
    iter=iter+1
    x=x-solve(H(x), g(x))
    trajectory[iter,]<-x
  }
  return(list(x.opt=x, trajectory=trajectory[1:iter,], iter=iter-1))
}
out3<-pure_Newton(gradient_log,Hessian,theta_initial,epsilon=10^-3)
cat("The maximum likelihood estimates of theta by this method is","\n",
    out3$x.opt,"\n",
    "And the corresponding value of loglike(theta) is",loglike(out3$x.opt),"\n",
    "The number of iterations required is",out3$iter)
```

```
## The maximum likelihood estimates of theta by this method is
##  0.05650989 0.008079293 -0.02284151
##  And the corresponding value of loglike(theta) is -64.85019
##  The number of iterations required is 3
```

The Pure Newton's method is even much more effient than the gradient method with backtracking!

vii

In order to improve the prediction accuracy, we could try to find the parameter to minimize -loglike(theta)+lambda*sum((theta)^2)

```
lreg<-function(theta,lambda){
  f=-loglike(theta)+lambda*sum((theta)^2)
  return(f)
}
grad_lreg<-function(theta,lambda){
  f=-gradient_log(theta)+2*lambda*theta
  return(f)
}
grad_lreg(theta_initial,lambda=10^3)
```

```
##       [,1]
## [1,]  24.5
## [2,] 816.0
## [3,] 753.5
```

Check the gradient of the function

```
numDeriv::grad(lreg,theta_initial,lambda=10^3)
```

```
## [1]  24.5 816.0 753.5
```

Build the Hessian of the lreg

```
Hessian_lreg<-function(theta,lambda){
  h_hess=-Hessian(theta)+2*lambda*diag(c(1,1,1))
  return(h_hess)
}
Hessian_lreg(theta_initial,lambda=10^3)
```

```
##          [,1]      [,2]      [,3]
## [1,]  2850.25  14343.25   8262.25
## [2,] 14343.25 377875.50 205490.75
## [3,]  8262.25 205490.75 129827.25
```

Then we check the Hessian.

```
numDeriv::hessian(lreg,theta_initial,lambda=10^3)
```

```
##           [,1]      [,2]      [,3]
## [1,]  2850.25  14343.25   8262.25
## [2,] 14343.25 377875.50 205490.75
## [3,]  8262.25 205490.75 129827.25
```

Now I could use the Pure Newton method to find the parameters to minimize lreg(The optimal theta)

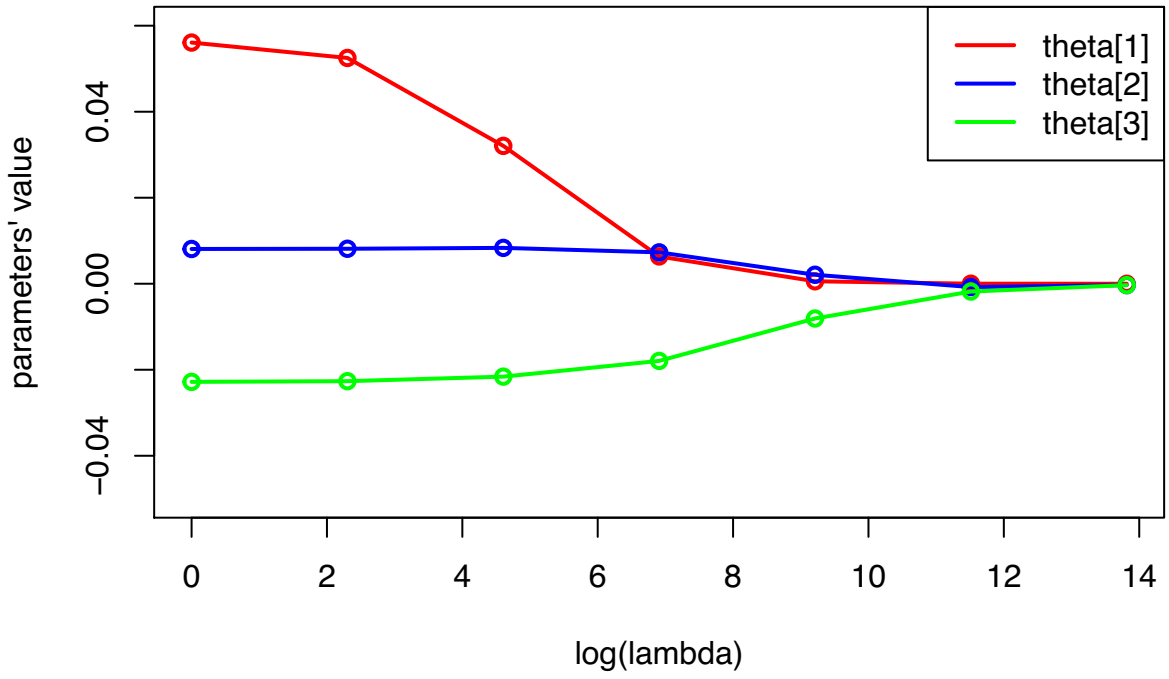
```
pure_Newton_reg <- function(g,H, x0, epsilon,lambda, maxiter=10^6){
  # Pure Newton's method. The method
  # terminates when ||g(x)|| <epsilon
  # INPUT
  # g - gradient of the objective function
  # H - a function to compute the Hessian matrix
  # x0 - initial point
  # epsilon ... tolerance parameter
  # maxiter - the maximum number of iterations

  ####
  # OUTPUT
  # x.opt = optimal solution (up to a tolerance) of max f(x)
  # iter = the number of the iterations
  # trajectory = the record of the change of the x

  x=x0
  trajectory <- matrix(0, nr=maxiter, nc=length(x0))
  trajectory[1,]<-x
  iter<-1
  while (sum(g(x,lambda)^2)>epsilon^2&& iter<maxiter){
    iter=iter+1
    x=x-solve(H(x,lambda), g(x,lambda))
    trajectory[iter,]<-x
  }
  return(list(x.opt=x, trajectory=trajectory[1:iter,], iter=iter-1))
}
out5<-pure_Newton_reg(grad_lreg,Hessian_lreg,theta_initial,epsilon=10^-3,lambda=10^3,maxiter=10^6)
theta_opt<-out5$x.opt
cat("The optimal theta here is",out5$x.opt,"\n",
    "The number of iterations required is",out5$iter)
```

```
## The optimal theta here is 0.006343552 0.007297859 -0.01794749
##  The number of iterations required is 3
```

Plot the setimate value of the parameters as lambda varies from 1 to 10^6

```
plt<-matrix(0,nr=3,nc=7)
for(i in 0:6){
  out6<-pure_Newton_reg(grad_lreg,Hessian_lreg,theta_initial,epsilon=10^-3,lambda=10^i,maxiter=10^6)
  plt[,i+1]<-out6$x.opt
}
plot(c(log(10^(0:6))),plt[1,],type="o",ylab="parameters' value",xlab="log(lambda)",col="red",lwd=2,ylim=
```

```
lines(c(log(10^(0:6))),plt[2,],type="o",col="blue",lwd=2)
lines(c(log(10^(0:6))),plt[3,],type="o",,col="green",lwd=2)
legend("topright",c("theta[1]","theta[2]","theta[3]"),col=c("red","blue","green"),lty=1,lwd=2)
```



Hessian matrix:

$$\nabla l(\underline{\theta}) = \underline{X}^T(\underline{y} - \underline{\hat{y}}(\underline{\theta})) = \sum_{i=1}^{n} \underline{x}_i (y_i - h(\underline{x}_i^T \underline{\theta}))$$

here $\underline{X} = \begin{pmatrix} -\underline{x}_1^T- \\ \vdots \\ -\underline{x}_n^T- \end{pmatrix}$ so $\underline{x}_i^T \Rightarrow 1\times 3$ vector, $\underline{x}_i \Rightarrow 3\times 1$ vector

$h'(t) = h(t)(1-h(t))$ (show it in question i)

$$H: \nabla^2 l(\underline{\theta}) = \frac{\partial}{\partial \underline{\theta}^T}\left(\sum_{i=1}^{n} \underline{x}_i (y_i - h(\underline{x}_i^T \underline{\theta}))\right)$$

$$= \sum_{i=1}^{n} -\underline{x}_i h'(\underline{x}_i^T \underline{\theta}) \underline{x}_i^T = \sum_{i=1}^{n} -\underline{x}_i h(\underline{x}_i^T \underline{\theta})(1-h(\underline{x}_i^T \underline{\theta})) \underline{x}_i^T$$

$$= -X^T M X$$

here $M = \begin{pmatrix} h(\underline{x}_1^T\underline{\theta})(1-h(\underline{x}_1^T\underline{\theta})) & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & -h(\underline{x}_n^T\underline{\theta})(1-h(\underline{x}_n^T\underline{\theta})) \end{pmatrix}_{n\times n}$

Then we use the code I built above to compute the Hessian matrix

OPT_CW1_Q3

Weicheng YU

11/6/2021

i

Firstly I need to code the function of f(t;g,k) and its gradient

```
f<-function(t,theta){
   f=20-(theta[1]/theta[2])*(t+(1/theta[2])*exp(-theta[2]*t)-1/theta[2])
   return(f)
}
grad_f<-function(t,theta){
  fn<-c()
  g=theta[1]
  k=theta[2]
  fn[1]=-(1/k)*(t+(1/k)*exp(-k*t)-1/k)
  fn[2]=(g/k^2)*(t+(1/k)*exp(-k*t)-1/k)-(g/k)*(-(1/k^2+t/k)*exp(-k*t)+1/k^2)
  return(fn)
}
grad_f(0.25,c(10,2))
```

```
## [1] -0.02663266  0.02040831
```

$$S(g,k)=\sum_{i=1}^{b}(f(t_i;g,k)-y_i)^2$$

$$\nabla S(g,k)=\left(\frac{\partial S(g,k)}{\partial g},\frac{\partial S(g,k)}{\partial k}\right)$$

$$=\sum_{i=1}^{b}2(f(t_i;g,k)-y_i)\left(\frac{\partial f_i}{\partial g},\frac{\partial f_i}{\partial k}\right)$$

Check the answer numerically

```
library(numDeriv)
numDeriv::grad(f,c(10,2),t=0.25)
```

```
## [1] -0.02663266  0.02040831
```

Then we could code the function of objective fuction s(g,k) and try g=10 and k=2

```
tvalue<-seq(0.25,2,0.25)
ytrue<-c(19.956,17.528,15.987,14.445,9.631,6.663,2.134,0.121)
s<-function(theta,t=tvalue,y=ytrue){
  s=sum((f(t,theta)-y)^2)
  return(s)
}
grad_s<-function(theta,t=tvalue,y=ytrue){
  grad_obj<-c(0,0)
  for(i in 1:length(t)){
    grad_obj=grad_obj+2*(f(t[i],theta)-y[i])*grad_f(t[i],theta)
```

```
  }
  return(grad_obj)
}
grad_s(c(10,2))
```

```
## [1] -49.31635 158.32410
```

Check the gradient function numerically

```
numDeriv::grad(s,c(10,2))
```

```
## [1] -49.31635 158.32410
```

These two results are quite approximate, hence the gradient function of the objective function I build above is reasonable.

ii

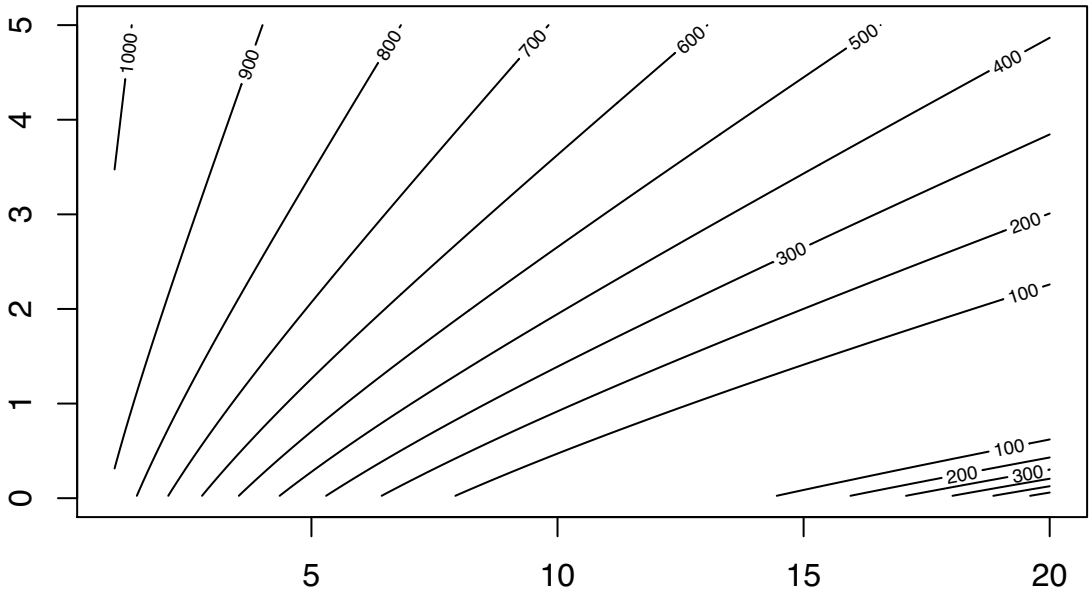
Then we create a coutour plot of s(g,k) with g in [1,20] and k in [0,5]

```
g<-seq(1,20,length.out=200)
k<-seq(0,5,length.out=200)
out<-matrix(0,nr=length(g),nc=length(k))
for(i in 1:length(g)){
  for (j in 1:length(k)){
    out[i,j]<-s(c(g[i],k[j]))
  }
}
library(pracma)
```

```
##
## Attaching package: 'pracma'
```

```
## The following objects are masked from 'package:numDeriv':
##
##     grad, hessian, jacobian
```

```
contour(g,k,out)
```



iii

Implement a pure Gauss-Newton algorithm to find the optimal value of g and k with the initial g=10 and k=2.Firstly I should build a jacobian matrix of f

```
J<-function(tvalue,theta){
  out<-matrix(nr=length(tvalue),nc=length(theta))
  for (i in 1:length(tvalue)){
    out[i,]<-grad_f(tvalue[i],theta)
  }
  return(out)
}
J(tvalue,c(10,2))
```

```
##              [,1]       [,2]
## [1,] -0.02663266 0.02040831
## [2,] -0.09196986 0.12954790
## [3,] -0.18078254 0.35119445
## [4,] -0.28383382 0.67667642
## [5,] -0.39552125 1.08672812
## [6,] -0.51244677 1.56116918
## [7,] -0.63254935 2.08260701
## [8,] -0.75457891 2.63736729
```

Check the jocabian

```
numDeriv::jacobian(f,c(10,2),t=tvalue)
```

```
##              [,1]       [,2]
## [1,] -0.02663266 0.02040831
## [2,] -0.09196986 0.12954790
## [3,] -0.18078254 0.35119445
```

```
## [4,] -0.28383382 0.67667642
## [5,] -0.39552125 1.08672812
## [6,] -0.51244677 1.56116918
## [7,] -0.63254935 2.08260701
## [8,] -0.75457891 2.63736729
```

Creat the F(t;g,k) which is [f(t1;g,k)-y1,f(t2;g,k)-y2,...,f(td;g,k)-yb]

```
FF<-function(tvalue,theta,y=yture){
  out<-c()
  for(i in 1:length(tvalue)){
    out[i]=f(tvalue[i],theta)-y[i]
  }
  return(out)
}
```

Then code the Gauss-Newton algorithm

```
gauss_newton <- function(s, J, FF, theta0, t, y, epsilon){
# s = objective of the form sum(f(ti;g,k)-yi)^2
# J = Jacobian matrix function
# FF = the set of {f(t1;g,k)-y1,f(t2;g,k)-y2,...,f(td;g,k)-yb}
# tvalue = vector of theta values to be summed over # fn vector of observations
# y=set of yi
# epsilon = tolerance
x=theta0
trajectory=x
JJ <- J(t, x)
grad <- 2*t(JJ)%*% FF(t, x, y)
iter <- 0
while(sum(grad^2)>epsilon^2){
    iter <- iter+1
    d = solve(t(JJ)%*%JJ, grad)
    x = x-0.5*d
    trajectory=rbind(trajectory, c(x))
    JJ <- J(t, x)
    grad = 2*t(JJ)%*% FF(t, x, y)
}
  return(list(x.opt=x, trajectory=trajectory, iter=iter,s.opt=s(x,t,y)))
}
```

Try to estimate the parameters

```
out4<-gauss_newton(s, J, FF, theta0=c(10,2), t=tvalue, y=ytrue, epsilon=10^-3)
iter=out4$iter
g.opt=out4$x.opt[1]
k.opt=out4$x.opt[2]
s.opt=out4$s.opt
print(paste("iter=",iter))
```

```
## [1] "iter= 7"
```

```r
print(paste("g.opt=",g.opt))
```

```
## [1] "g.opt= 18.8574138480728"
```

```r
print(paste("k.opt=",k.opt))
```

```
## [1] "k.opt= 1.07835398753026"
```

```r
print(paste("s.opt=",s.opt))
```

```
## [1] "s.opt= 3.90419634223413"
```

After 7 iterations the function converges. The optimal values of (g,k) is (18.8574,1.07835) and the value of the bojective at the optima is s(g,k)=3.9.