Homework on optimization algorithms.

P8160 Advanced Statistical Computing

Problem 1:

Design an optmization 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

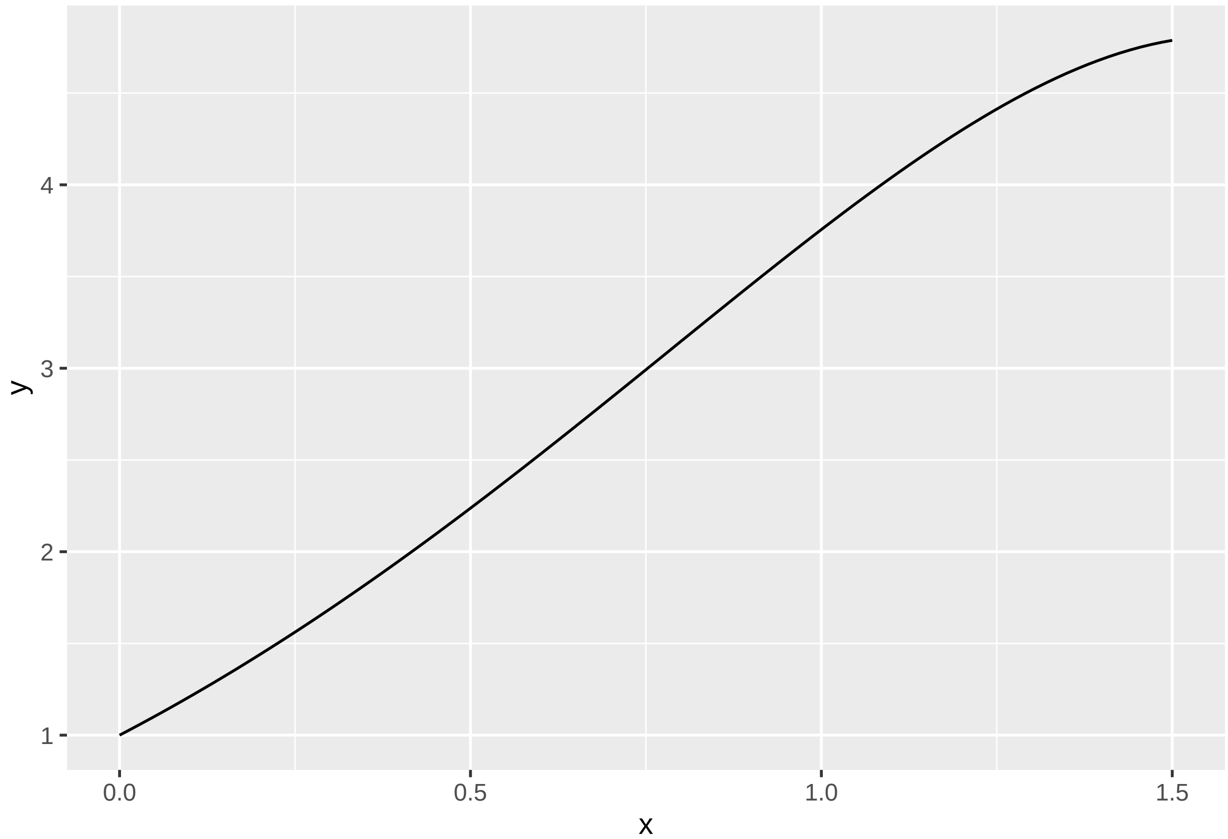
$$g(x) = e^{x}\sin(x)$$

and instead find the maximum of g(x).

The gradient of $g(x)$ is :

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

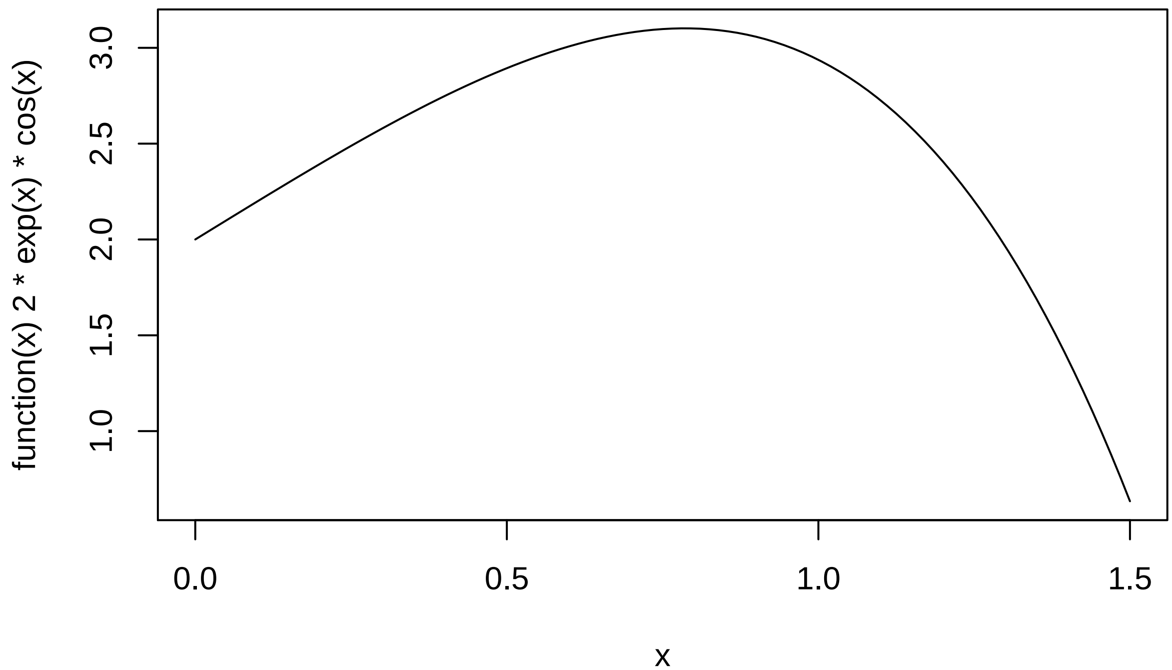
```
ggplot(tibble(x = seq(0,1.5,length = 10)),aes(x))+
  geom_function(fun = function(x) exp(x)*(sin(x)+cos(x)))
```



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))
  }
```

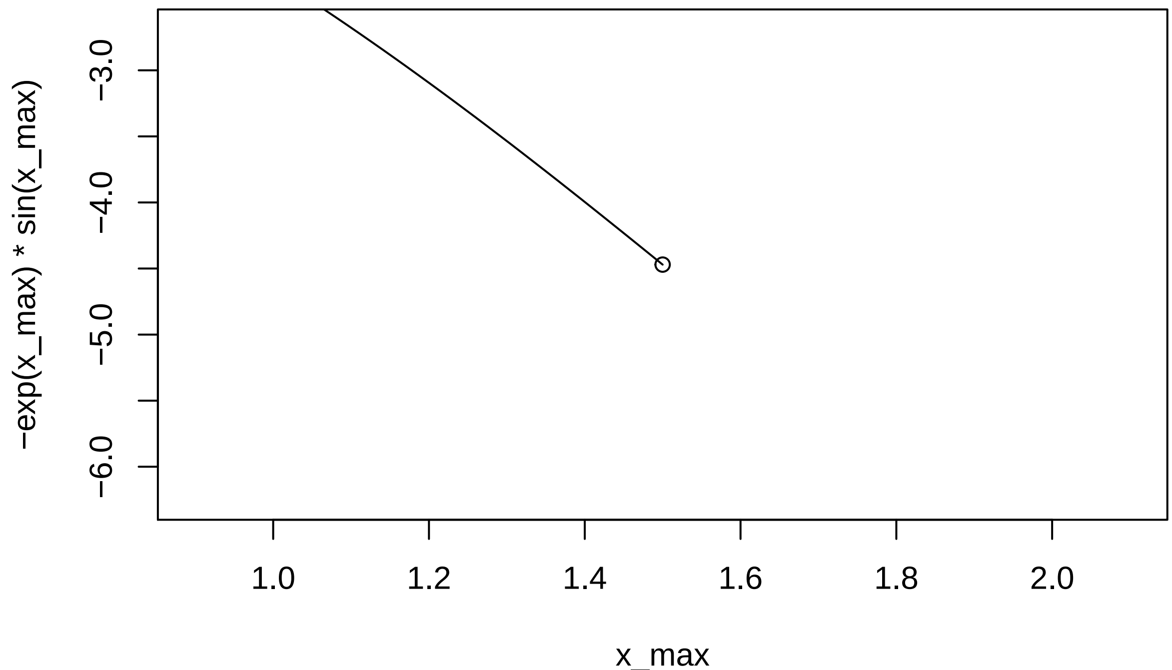
```
x_max = goose_egg(function(x) exp(x)*sin(x),range = c(0,1.5))

print(x_max)
```

```
## [1] 1.499962
```

```
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 $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$.

1. Generate a random sample (x_i, Y_i) with $n = 500$ from the Possion 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 modify 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")
```

```
## 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

$$\nabla^2(Y;\lambda) = \frac{\partial^2}{\partial\lambda^2} l(Y;\lambda)$$

$$= \begin{pmatrix} \sum -exp(\alpha + x\beta) & \sum -x * exp(\alpha + x\beta) \\ \sum -x * exp(\alpha + x\beta) & \sum -x^2 * exp(\alpha + x\beta) \end{pmatrix}$$

which is negative defined everywhere.

```
Poisson =
  function(data,
           theta_vec) {
    Y = data$y
    X = data$x
    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(dat,c(7,2))
```

```
## $loglink
## [1] -3974589
##
## $fisher
##            [,1]
## [1,] 821143964
##
## $gradient
## [1] -3980250 -7302954
##
## $hessian
##           [,1]      [,2]
## [1,] -3981198  -7303206
## [2,] -7303206 -15987659
```

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,
           data,
           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(data,previous_theta)$loglink) == Inf) stop("Check your log-likelihood")
    trial = 0

    gradient = fun(data,previous_theta)$gradient

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

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

    #backtracking
    while (backtracking & fun(data,cur_theta)$loglink - fun(data,previous_theta)$loglink < 0 & trial < 
      step_size = step_size / 2

      trial = trial + 1 # avoild dead loops

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

    return(cur_theta)
  }

newton_update(fun = Poisson,previous_theta = c(7,2), data = dat)
```

```
##          [,1]
## [1,] 6.001291
## [2,] 1.999426
```

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

```
  f = fun(data,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 (abs(f$loglink - prevlog) > tol && i < maxtiter) {
    i = i + 1
    prev_theta = cur_theta
    prevlog = f$loglink
    cur_theta = newton_update(fun, prev_theta,data)
    f = fun(data,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),data = dat)$theta

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

term	theta	theta_hat
alpha	0.5854990	0.6064289
beta	0.2339566	0.2472194

Problem 3:

The data *breast-cancer.csv* have 569 row and 33 columns. The first column **ID** lables individual breast tissue images; The second column **Diagnonsis** indentifies 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 (perimeter$\hat{2}$ / 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.