MATH 3190 Homework 7 Solutions

Focus: Notes 9

Due April 6, 2024

Your homework should be completed in R Markdown or Quarto and Knitted to an html or pdf document. You will "turn in" this homework by uploading to your GitHub Math_3190_Assignment repository in the Homework directory.

Some of the parts in this homework, like 1a, 2a, 2b, and 2c require writing down some math-heavy expressions. You may either type it up using LaTeX style formatting in R Markdown, or you can write it by hand (neatly) and include pictures or scans of your work in your R Markdown document.

Problem 1 (33 points)

Suppose we want to find the value of x and the objection function value for finding the global maximum and the minimum of the function $f(x) = \ln(1 + x^2)(1 + x)e^{-x^2}$. We will implement gradient descent and Newton's method for finding these extrema.

Part a (4 points)

Find $f'(x)$ for this function. Do this by hand as a nice derivative refresher. Feel free to check your answer using something like Wolfram Alpha.

$$f'(x) = \left(\frac{2x}{1+x^2}\right) * (1+x)e^{-x^2} + \ln(1+x^2) * \left(e^{-x^2} + (1+x) * e^{-x^2} * -2x\right)$$

Part b (12 points)

Write a function that implements gradient descent. Use a backtracking line search each iteration to find γ_k: the step size. This function should take eight inputs:

- The starting value (or vector)
- The function to optimize
- The function's derivative (or gradient)
- A logical indicating if we want to find a max with a default of `FALSE`.
- The maximum number of iterations with a default of 200
- The initial step size with a default of 1
- The line search beta parameter with a default of 0.5.
- The stopping tolerance with a default of `1e-10`.

The output of this function should be a list with the x value (or vector) that optimizes the function, the function value at that point, and the number of iterations it took to converge. This function should work in the case of one dimension or multiple dimensions since you'll use it again in problem 2.

```
graDesc <- function(startVal, f, fPrime, maximum = FALSE, maxIt = 200,
                    step = 1, betaLine = 0.5, tolerance = 1e-10){
  if(maximum == T){
    g <- function(x){
      -f(x)
    }
    gPrime <- function(x){
      -fPrime(x)
    }


  }else{
    g <- f
    gPrime <- fPrime
  }
  for(k in 1:maxIt){
    while(g(startVal - step * gPrime(startVal)) >
          g(startVal) - step/4 * sum(gPrime(startVal)^2)){
      step <- betaLine * step
    }
    xnew <- startVal - step * gPrime(startVal)
    if(norm(startVal - xnew, type = "2") < tolerance){
      break
    }
    startVal <- xnew
  }
  c(startVal, k, f(startVal))
  }
```

Part c (3 points)

Use your gradient descent function to find the global **min** of $f(x) = \ln(1+x^2)(1+x)e^{-x^2}$. Try using several starting points. Keep track of and report the number of iterations needed to converge at the different starting points.

```
f <- function(x){
  (log(1 + x^2))*(1 + x)*exp(-x^2)
}

fPrime <- function(x){
  (2*x / (1+x^2))*(1 + x)*exp(-x^2) + log(1 + x^2) *
    (exp(-x^2) + (1 + x)*exp(-x^2)* - 2*x)
}
```

```
minimums <- tibble(x = numeric(),
                   iterations = numeric(),
                   fx = numeric(),
                   startVal = numeric())

for(i in seq(-2, 2, 1)){
  minimums <- add_row(minimums,
                      x = graDesc(startVal = i, f = f, fPrime = fPrime)[1],
```

```
                      iterations = graDesc(startVal = i, f = f,
                                           fPrime = fPrime)[2],
                      fx = graDesc(startVal = i, f = f, fPrime = fPrime)[3],
                      startVal = i)
}
minimums
```

```
## # A tibble: 5 x 4
##       x iterations        fx startVal
##   <dbl>      <dbl>     <dbl>    <dbl>
## 1 -1.47         37 -0.0623         -2
## 2 -1.47         29 -0.0623         -1
## 3  0             1  0                0
## 4  3.35        200  0.000148         1
## 5  3.35        200  0.000144         2
```

Part d (3 points)

Use your gradient descent function to find the global **max** of $f(x) = \ln(1+x^2)(1+x)e^{-x^2}$. Again, try several starting points. Keep track of and report the number of iterations needed to converge at the different starting points.

```
maximums <- tibble(x = numeric(),
                   iterations = numeric(),
                   fx = numeric(),
                   startVal = numeric())

for(i in seq(-2, 2, 1)){
  maximums <- add_row(maximums,
                      x = graDesc(startVal = i, f = f, fPrime = fPrime,
                                  maximum = T)[1],
                      iterations = graDesc(startVal = i, f = f,
                                           fPrime = fPrime, maximum = T)[2],
                      fx = graDesc(startVal = i, f = f, fPrime = fPrime,
                                   maximum = T)[3],
                      startVal = i)
}
maximums
```

```
## # A tibble: 5 x 4
##       x iterations        fx startVal
##   <dbl>      <dbl>     <dbl>    <dbl>
## 1 -3.21        200 -0.000182       -2
## 2 -0.541        15  0.0879         -1
## 3  0             1  0                0
## 4  0.987        10  0.510            1
## 5  0.987        14  0.510            2
```

Part e (6 points)

Use the fact that

$$f''(x) = \frac{2e^{-x^2}}{(1+x^2)^2}\left((2x^3+2x^2-3x-1)(1+x^2)^2\ln(1+x^2) - 4x^5 - 4x^4 - 3x^3 - 5x^2 + 3x + 1\right)$$

to implement Newton's method to find the global max **and** the global min. Use the same starting points you did in parts c and d. Keep track of and report the number of iterations needed to converge at the different starting points. And yes, that second derivative is very messy! This is one reason why Newton's method is less popular. You don't have to write a function for this part to implement Newton's method, but you can if you'd like.

```
newton <- function(f, fPrime, fDP, startVal, maxIt = 200,
                   maximum = FALSE, tolerance = 1e-10){
    if(maximum){
    g <- function(x){
      -f(x)
    }
    gPrime <- function(x){
      -fPrime(x)
    }
    gDP <- function(x){
      -fDP(x)
    }


  }else{
    g <- f
    gPrime <- fPrime
    gDP <- fDP
  }
  for(k in 1:maxIt){
    xnew <- startVal - gPrime(startVal)/gDP(startVal)
    if (abs(startVal - xnew) < 1e-10){
      break
    }
    startVal <- xnew
  }
  c(startVal, k, g(startVal))
}
```

```
fDP <- function(x){
  (2*exp(-x^2))/(1 + x^2)^2*
    ((2*x^3 + 2*x^2 - 3*x - 1)*(1 + x^2)^2*log(1 + x^2) -
                              4*x^5 - 4*x^4 - 3*x^3 - 5*x^2 + 3*x + 1)
}
```

```
newtons <- tibble(x = numeric(),
                  iterations = numeric(),
                  fx = numeric(),
                  startVal = numeric())

for(i in seq(-2, 1, 0.5)){
```

```
  newtons <- add_row(newtons,
                     x = newton(startVal = i, f = f, fPrime = fPrime,
                                fDP = fDP)[1],
                     iterations = newton(startVal = i, f = f,
                                         fPrime = fPrime, fDP = fDP)[2],
                     fx = newton(startVal = i, f = f, fPrime = fPrime,
                                 fDP = fDP)[3],
                     startVal = i)
}
```

Part f (3 points)

Compare the number of iterations needed for convergence for gradient descent and for Newton's method.

The gradient descent took 29 and 10 iterations to converge to minimum and maximum, respectively. Newton's method took 4 for both.

Part g (2 points)

Use the `optimize()` function in **R** to find the global max and min.

```
optimize(f, interval = c(-2, 2), maximum = T)
```

```
## $maximum
## [1] 0.9868812
##
## $objective
## [1] 0.5101816
```

```
optimize(f, interval = c(-2, 2), maximum = F)
```

```
## $minimum
## [1] -1.469442
##
## $objective
## [1] -0.06232362
```

Problem 2 (27 points)

We implemented optimization to maximize the likelihood for a Poisson regression problem in the notes to estimate the β_i values for $i = 0, \ldots, p-1$. Let's do something similar for find the $\boldsymbol{\beta}$ vector by maximizing the likelihood in logistic regression.

In logistic regression, we attempt to predict the probability of a success for a given case. We can use Bernoulli random variable for this. For a Bernoulli random variable, $P(Y = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$ for $y_i = 0, 1$ where p_i is the probability of a success. In our case, for logistic regression, we have $\text{logit}(p_i) = \mathbf{X}_i\boldsymbol{\beta}$. That means $p_i = \dfrac{\exp(\mathbf{X}_i\boldsymbol{\beta})}{1+\exp(\mathbf{X}_i\boldsymbol{\beta})}$.

Part a (4 points)

Find the likelihood function, $L(\boldsymbol{\beta}|\mathbf{X}, \boldsymbol{y})$, for the logistic regression for a sample of size n.

$L(\boldsymbol{\beta}\|\mathbf{X}, \boldsymbol{y}) = \prod_{i=1}^{n} \left[\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right]^{y_i} \left[1 - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right]^{1-y_i}$

Part b (6 points)

Show that the log likelihood function is

$$\ell(\boldsymbol{\beta}|\mathbf{X}, \boldsymbol{y}) = \sum_{i=1}^{n} y_i \ln\left(\frac{\exp(\mathbf{X}_i\boldsymbol{\beta})}{1+\exp(\mathbf{X}_i\boldsymbol{\beta})}\right) + \sum_{i=1}^{n}(1-y_i)\ln\left(\frac{1}{1+\exp(\mathbf{X}_i\boldsymbol{\beta})}\right)$$

and then show it can be equivalently written

$$\ell(\boldsymbol{\beta}|\mathbf{X}, \boldsymbol{y}) = \sum_{i=1}^{n} y_i(\mathbf{X}_i\boldsymbol{\beta}) - \sum_{i=1}^{n} \ln\Big(1+\exp(\mathbf{X}_i\boldsymbol{\beta})\Big).$$

Log likelihood: $\ell(\boldsymbol{\beta}|\mathbf{X}, \boldsymbol{y}) = \ln\left(\prod_{i=1}^{n}\left[\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right]^{y_i} * \left[1 - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right]^{1-y_i}\right)$

$\sum_{i=1}^{n} y_i(\mathbf{X}\boldsymbol{\beta}) - \sum_{i=1}^{n}\ln(1+\exp(\mathbf{X}_i\boldsymbol{\beta}))$

$\ln(xy) = \ln(x) + \ln(y)$ and $\ln(x^n) = n * \ln(x)$

Log Likelihood Rewritten: $\ell(\beta|\mathbf{x}, \boldsymbol{y}) = \sum_{i=1}^{n} y_i\left[\ln\Big(\exp(x_i\beta) - \ln(1+\exp(x_i\beta))\Big)\right] + \sum_{i=1}^{n}(i-y_i)\left[\ln(1) - \ln\Big(1+\exp(x_i\beta)\Big)\right]$

$\sum_{i=1}^{n} y_i(x_i * \beta) - y_i\big(\ln(1+\exp(x_i\beta))\big) + \sum_{i=1}^{n} y_i\Big(\ln(1+\exp(x_i\beta)\Big) - \ln\Big(1+\exp(x_i\beta)\Big)$

$\sum_{i=1}^{n} y_i(x_i * \beta) - \sum_{i=1}^{n}\ln\Big(1+\exp(x_i\beta)\Big)$

Part c (5 points)

Find the gradient of the log likelihood if we have three β's. That is, if $\mathbf{X}_i\boldsymbol{\beta} = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$.

$\ell(\beta\_0, \beta\_1, \beta * 2\|X, y) = \sum_{i=1}^{n} y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) - \sum_{i=1}^{n}\ln\Big(1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})\Big)$

$$\frac{d\ell}{d\beta_0} = \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} \frac{\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}$$

$$\frac{d\ell}{d\beta_1} = \sum_{i=1}^{n} y_i x_{i1} - \sum_{i=1}^{n} \frac{x_{i1}\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}$$

$$\frac{d\ell}{d\beta_2} = \sum_{i=1}^{n} y_i x_{i2} - \sum_{i=1}^{n} \frac{x_{i2}\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}$$

$$\nabla\ell = \begin{bmatrix} \frac{d\ell}{\beta_0} \\ \frac{d\ell}{d\beta_1} \\ \frac{d\ell}{\beta_2} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} \frac{\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})} \\ \sum_{i=1}^{n} y_i x_{i1} - \sum_{i=1}^{n} \frac{x_{i1}\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})} \\ \sum_{i=1}^{n} y_i x_{i2} - \sum_{i=1}^{n} \frac{x_{i2}\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1+\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})} \end{bmatrix}$$

Part d (3 points)

In Homework 3, Problem 2, part a, you read in the `adult` dataset (from the UC Irvine database). This is the one that we used to predict whether a person makes over $50K a year based on some other variables. The data came from the Census Bureau in 1994 and can be found in the Data folder in my Math3190_S24 GitHub repo. More info on the dataset can be found in the "adult.names" file.

Read in the dataset and put column names like you did in HW 3. Then fit a logistic regression model using the `glm()` function that predicts salary from age/10 and sex. Note, we should divide the age by 10 in our model because this makes the gradient descent more stable (for whatever reason). We can divide by 10 in the `glm()` function by wrapping it in the `I()` function. Like this: `glm(salary ~ I(age/10) + sex, data = adult, family = binomial)`.

```
adult <-  read_csv("adult.data", col_names = FALSE)|>
  rename("age" = "X1", "wClass" = "X2", "fnlwgt" = "X3",
              "education" = "X4", "education-num" = "X5", "mStatus" = "X6",
              "occup" = "X7", "relationship" = "X8", "race" = "X9",
              "sex" = "X10", "capGain" = "X11", "capLoss" = "X12",
              "hours_per_week" = "X13", "nCountry" = "X14",
              "salary" = "X15") |>
  tibble()
```

```
## Rows: 48842 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (9): X2, X4, X6, X7, X8, X9, X10, X14, X15
## dbl (6): X1, X3, X5, X11, X12, X13
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
adult$salary = as_factor(adult$salary)
```

```
adultMod <- glm(salary ~ I(age/10) + sex, data = adult, family = binomial)
summary(adultMod)
```

```
##
## Call:
## glm(formula = salary ~ I(age/10) + sex, family = binomial, data = adult)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6466  -0.7626  -0.5853  -0.3209   2.4314
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.629827   0.043440  -83.56   <2e-16 ***
## I(age/10)    0.382809   0.008144   47.01   <2e-16 ***
## sexMale      1.242016   0.028505   43.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 53751  on 48841  degrees of freedom
## Residual deviance: 48966  on 48839  degrees of freedom
## AIC: 48972
##
## Number of Fisher Scoring iterations: 4
```

Part e (2 points)

Define `y`, `x1`, and `x2`. `y` should be a vector of 0's and 1's with a 1 indicating that the person had a salary above \$50,000. `x1` should be a vector that contains all of the age values (divided by 10) and then `x2` should be an indicator variable that indicates if the person is male or not. Taking columns from the matrix obtained using the `model.matrix()` may be useful here.

```
newData <- model.matrix(adultMod)
newData <- newData |>
  as_tibble() |>
  mutate(y = case_when(
    (adult$salary == '<=50K') ~ 0,
    (adult$salary == '>50K') ~ 1
  ))
```

```
y <- newData[,4]
x1 <- newData[,2]
x2 <- newData[3]
```

Part f (5 points)

Use your gradient descent function from problem 1 to find estimates for β_0, β_1, and β_2 for predicting salary from age and sex. Enter the following in your function:

- Use the vector `c(0, 0, 0)` as your starting point.
- Instead of starting the step size, γ_k, at 1, start it at 0.1 to save some time with the line search.
- Make sure the maximum number of iterations is at least 200.

You'll know you did this right if the optimized values you end up with match (or are very close to) the estimates from your logistic regression model you fit in part d. This may take a minute or two to run.

```
likelihood <- function(beta){
  sum(y*(beta[1] + beta[2]*x1 + beta[3]*x2)) -
    sum(log(1 + exp(beta[1] + beta[2]*x1 + beta[3]*x2)))

}

gradient <- function(beta){
  dlb0 <- sum(y) - (sum(exp(beta[1] + beta[2]*x1 + beta[3]*x2)/
                        (1 + exp(beta[1] + beta[2]*x1 + beta[3]*x2))))

  dlb1 <- sum(y*x1) - (sum((x1*exp(beta[1] + beta[2]*x1 + beta[3]*x2))/
                        (1 + exp(beta[1] + beta[2]*x1 + beta[3]*x2))))
```

```
  dlb2 <- sum(y*x2) - (sum((x2*exp(beta[1] + beta[2]*x1 + beta[3]*x2))/
                           (1 + exp(beta[1] + beta[2]*x1 + beta[3]*x2))))
  c(dlb0, dlb1, dlb2)
}
startingVals <- c(0, 0, 0)
```

```
graDesc(startVal = startingVals, f = likelihood, fPrime = gradient,
        maximum = T, step = 0.1, maxIt = 1000)
```

```
## [1] -3.481258e+00  3.629999e-01  1.171210e+00  1.000000e+03 -2.448925e+04
```

I don't know why, but I had to increase the iterations to 1000 to get anything even remotely close.

Part g (2 points)

Use the `optim()` function in **R** to find the estimate for the β's here. This should be much faster since this function is much better optimized (no pun intended). The results here should also be very close to the numbers we obtained the slope in the model we fit in part d, but may not match exactly.

```
optim(startingVals, likelihood, control = list(fnscale = -1))
```

```
## $par
## [1] -3.6304477  0.3829196  1.2420241
##
## $value
## [1] -24483.08
##
## $counts
## function gradient
##      118       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```