Stats 102B HW 4

Bryan Mui - UID 506021334

Due Wed, June 4, 11:00 pm

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
train <- read_csv("train_data.csv")
```

```
Rows: 600 Columns: 601
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (601): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15,...

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.
```

```
val <- read_csv("validation_data.csv")
```

```
Rows: 200 Columns: 601
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (601): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15,...

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.
```

Problem 1

Consider the function

$$f(x) = \frac{1}{4}x^4 - x^2 + 2x$$

Part (α)

Using the pure version of Newton's algorithm report x_k for $k = 20$ (after running the algorithm for 20 iterations) based on the following 5 initial points:

1. $x_0 = -1$
2. $x_0 = 0$
3. $x_0 = 0.1$
4. $x_0 = 1$
5. $x_0 = 2$

Newton's pure algorithm is as follows:

1. Select $x_0 \in \mathbb{R}^n$
2. While stopping criterion $>$ tolerance do:
    1. $x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$
    2. Calculate value of stopping criterion($|f(x_{k+1}) - f(x_k)| \leq \epsilon$)

Gradient: $\nabla f(x) = \frac{\partial}{\partial x} = f'(x) = x^3 - 2x + 2$

Hessian: $\nabla^2 f(x) = \frac{\partial^2}{\partial x^2} = f''(x) = 3x^2 - 2$

```
# params
max_iter <- 20
starting_points <- c(-1, 0, 0.1, 1, 2)
stopping_tol <- 1e-6


# algorithm
newton_pure_alg <- function(max_iter, starting_point, stopping_tol) {
  beta <- starting_point
  iterations_ran <- 0
  betas_vec <- c(beta)

  obj <- function(x) {
  return(1/4 * x^4 - x^2 + 2*x)
  }
  grad <- function(x) {
    x^3 - 2*x + 2
  }
  hessian <- function(x) {
    3*x^2 - 2
  }

  for(i in 1:max_iter) {
    beta_new <- beta - (grad(beta) / hessian(beta))
    betas_vec[i+1] <- beta_new
    if(abs(beta_new - beta) <= stopping_tol) { break }
    beta <- beta_new
```

```
  }
  iterations_ran <- i
  return(list(iterations=iterations_ran, betas=betas_vec))
}

# running the alg
cat("Newton's Algorithm(Pure) For Different Starting Points: \n")
```

```
Newton's Algorithm(Pure) For Different Starting Points:
```

```
for (starting_point in starting_points) {
  result <- newton_pure_alg(max_iter, starting_point, stopping_tol)
  cat("\nStarting Point:", starting_point, "\nIterations:", result$iterations, "\nBetas:\n")
  print(result$betas)
  cat("\n", "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~", "\n")
}
```

```
Starting Point: -1
Iterations: 8
Betas:
[1] -1.000000 -4.000000 -2.826087 -2.146719 -1.842326 -1.772848 -1.769301
[8] -1.769292 -1.769292

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Starting Point: 0
Iterations: 20
Betas:
 [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Starting Point: 0.1
Iterations: 20
Betas:
 [1] 1.000000e-01 1.014213e+00 7.965577e-02 1.009099e+00 5.222653e-02
 [6] 1.003965e+00 2.332944e-02 1.000804e+00 4.806795e-03 1.000035e+00
[11] 2.072525e-04 1.000000e+00 3.865288e-07 1.000000e+00 1.345590e-12
[16] 1.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00
[21] 0.000000e+00

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Starting Point: 1
Iterations: 20
Betas:
 [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Starting Point: 2
Iterations: 9
```

```
Betas:
 [1]  2.0000000  1.4000000  0.8989691 -1.2887793 -2.1057673 -1.8292000
 [7] -1.7717158 -1.7692966 -1.7692924 -1.7692924

 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Part (i) What do you observe?

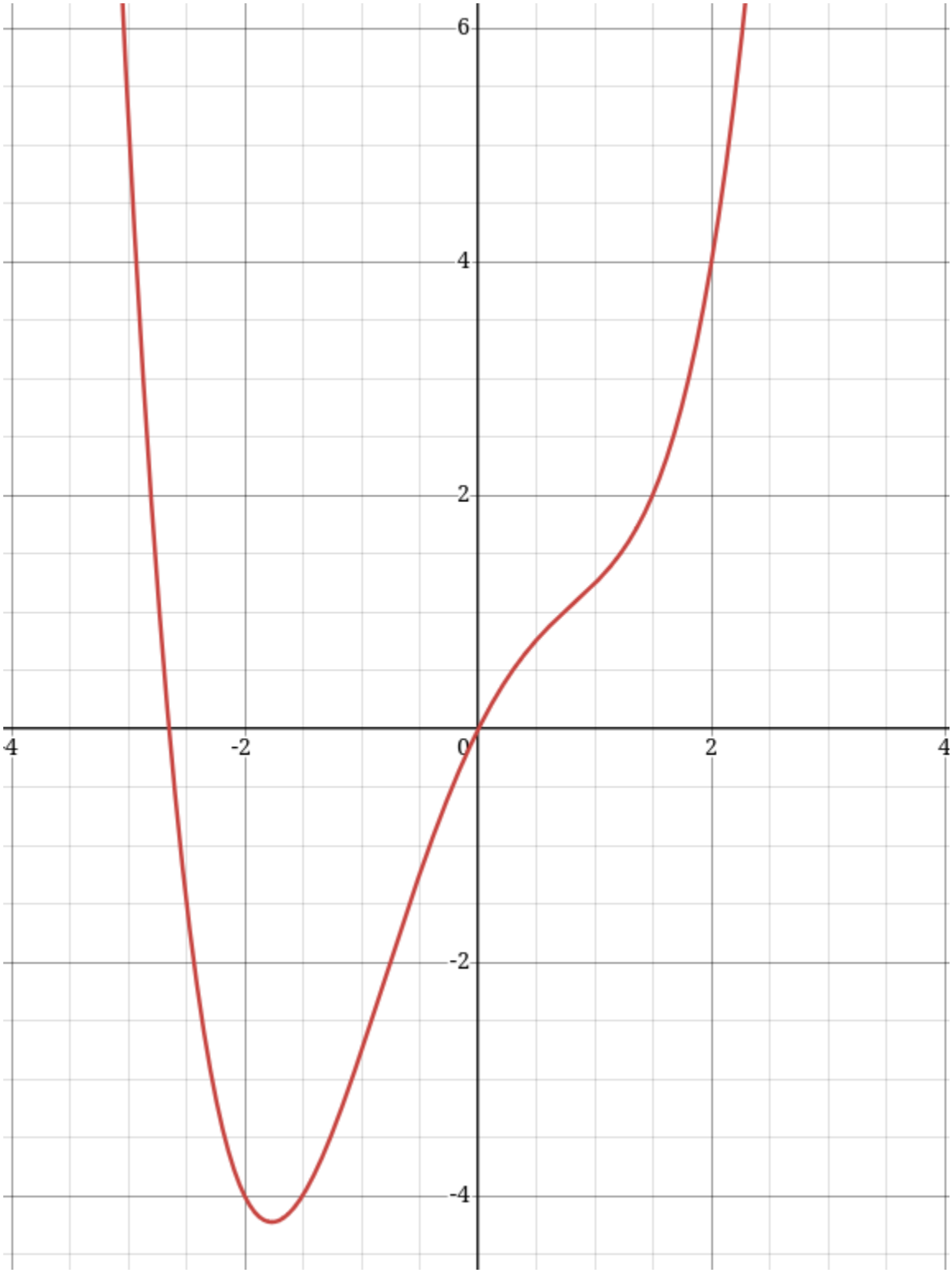

Figure 1: Plot of the Objective Function f(x)

I observe that given the code output, the algorithms that did not converge oscillate between 0 1 0 1, which means that it gets stuck when the curvature of the graph changes. I think that the graph goes from concave up to concave down at this point, which will mess up the gradient descent calculation. The Hessian is the second derivative, or second order gradient, meaning the concavity of the function will affect the sign. Given the update step, $x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$, we can see that if the Hessian's sign goes from negative to positive(concave down to concave up) or vice versa then it will change the sign of the update step to be gradient ascent(away from the minimum) rather than towards it

Part (ii) How can you fix the issue reported in (i)?

I believe that the issue is normally fixed by drawing the graph and getting and understanding of the shape, and therefore choosing one of the starting points that does end up converging. However, in cases where that isn't possible(if the function cannot be plotted or is very complex), I would try to build a vector of possible starting points and perform a grid search where we run the algorithm on multiple starting points and various parameters. This would allow us to find a starting point where the algorithm would converge. From the lecture slides, there are alternative Newton's algorithms like Newton's algorithm with damping, which I would implement if there is too much trouble with the pure Newton's algorithm.

Problem 2

Consider the data in the train data.csv file. The first 600 columns correspond to the predictors and the last column to the response y.

Part (i) Implement that proximal gradient algorithm for Lasso regression, by modifying appropriately your code from Homework 1.

To select a good value for the regularization parameter λ use the data in the validation data.csv to calculate the sum-of-squares error validation loss.

The Proximal Gradient Descent Algorithm(LASSO):

Part(ii) Show a plot of the training and validation loss as a function of iterations. Report the number of regression coefficients estimated as zero based on the best value of λ you selected.

Plotting Training Loss:

Plotting Validation Loss: