[STAT 4610] HW-11 / Michael Ghattas

Michael Ghattas

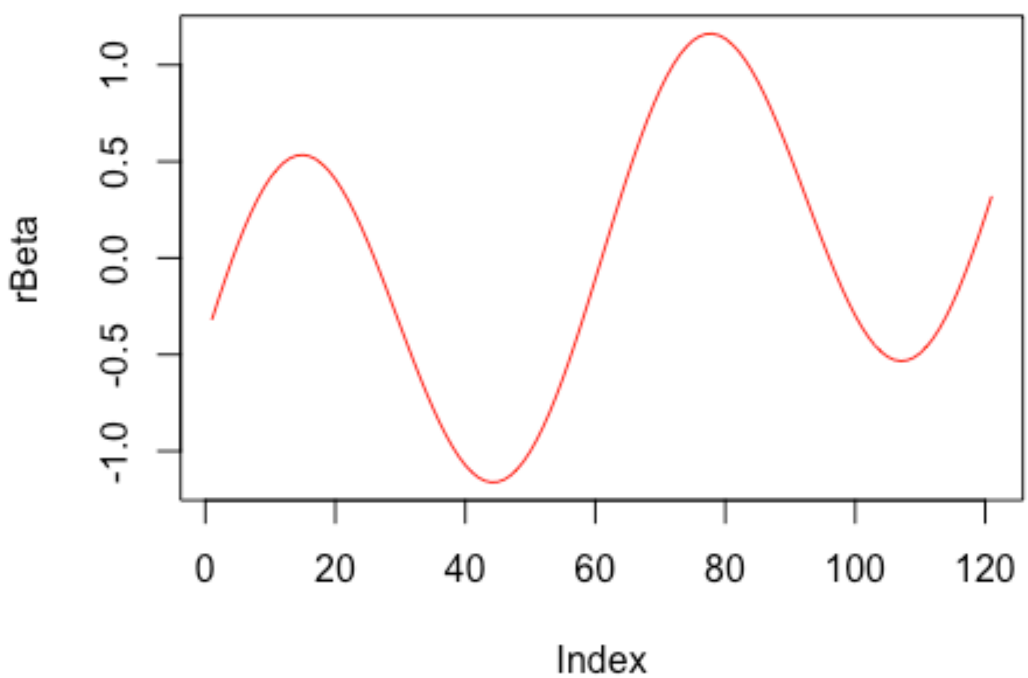
11/29/2022

Chapter 10

Problem-6

Part (a)

```
b = seq(from = -6, by = 0.1, length.out = 121)
rBeta = sin(b) + (b / 10)
plot(rBeta, type = "l", col = "red", lwd = 1)
```



Part (b)

$$\cos(\beta) + \frac{1}{10}$$

Part (c)

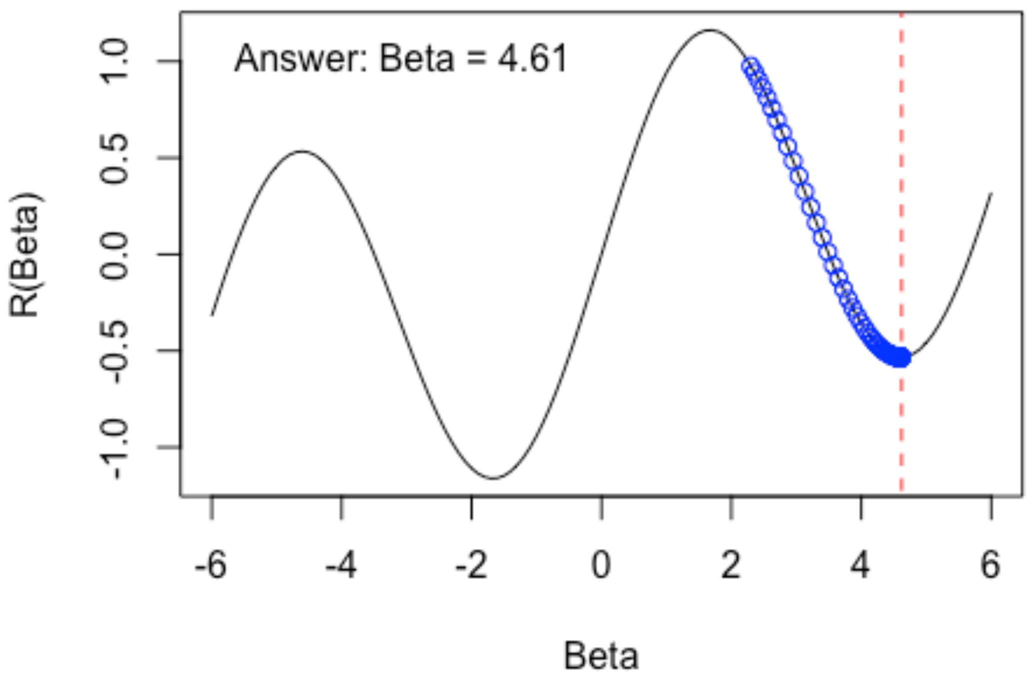
```r
b = seq(from = -6, by = 0.1, length.out = 121)
rBeta = function(b) {
  sin(b) + (b / 10)
}
graDesc = function(b) {
  cos(b) + (1 / 10)
}

plot(b , rBeta(b), type = "l", xlab = "Beta", ylab = "R(Beta)")
lines (c(4.62, 4.62), c(-2, 2), col = "red", lty = 2)

b = 2.3
bTrace = b
rTrace = rBeta(b)
rho = 0.1

for (step in 1:74) {
  b = b - (rho * graDesc(b))
  bTrace = c(bTrace, b)
  rTrace <- c(rTrace, rBeta(b))
}

bGrid = data.frame(Beta = bTrace, R = rTrace); bGrid
```

```
##          Beta           R
## 1  2.300000  0.97570521
## 2  2.356628  0.94246322
## 3  2.417369  0.90429111
## 4  2.482270  0.86080841
## 5  2.551311  0.81172610
## 6  2.624389  0.75689003
```

```
## 7  2.701310  0.69632619
## 8  2.781773  0.63028265
## 9  2.865369  0.55926118
## 10 2.951578  0.48403075
## 11 3.039779  0.40561617
## 12 3.129261  0.32525774
## 13 3.219253  0.24434293
## 14 3.308952  0.16431633
## 15 3.397554  0.08657944
## 16 3.484296  0.01239472
## 17 3.568481 -0.05719267
## 18 3.649507 -0.12140546
## 19 3.726883 -0.17975338
## 20 3.800238 -0.23202267
## 21 3.869321 -0.27824283
## 22 3.933989 -0.31863925
## 23 3.994204 -0.35358061
## 24 4.050005 -0.38352806
## 25 4.101505 -0.40899092
## 26 4.148864 -0.43049121
## 27 4.192281 -0.44853755
## 28 4.231979 -0.46360751
## 29 4.268193 -0.47613717
## 30 4.301166 -0.48651610
## 31 4.331139 -0.49508643
## 32 4.358347 -0.50214443
## 33 4.383016 -0.50794385
## 34 4.405361 -0.51269999
## 35 4.425584 -0.51659422
## 36 4.443873 -0.51977837
## 37 4.460403 -0.52237888
## 38 4.475336 -0.52450065
## 39 4.488820 -0.52623035
## 40 4.500991 -0.52763942
## 41 4.511974 -0.52878661
```

```
## 42 4.521881 -0.52972009
## 43 4.530817 -0.53047935
## 44 4.538875 -0.53109665
## 45 4.546139 -0.53159838
## 46 4.552688 -0.53200605
## 47 4.558590 -0.53233723
## 48 4.563909 -0.53260620
## 49 4.568703 -0.53282461
## 50 4.573022 -0.53300193
## 51 4.576914 -0.53314588
## 52 4.580420 -0.53326272
## 53 4.583578 -0.53335754
## 54 4.586424 -0.53343449
## 55 4.588987 -0.53349694
## 56 4.591296 -0.53354760
## 57 4.593376 -0.53358871
## 58 4.595249 -0.53362205
## 59 4.596936 -0.53364911
## 60 4.598456 -0.53367105
## 61 4.599825 -0.53368886
## 62 4.601057 -0.53370330
## 63 4.602167 -0.53371501
## 64 4.603167 -0.53372451
## 65 4.604068 -0.53373221
## 66 4.604879 -0.53373846
## 67 4.605609 -0.53374353
## 68 4.606267 -0.53374764
## 69 4.606859 -0.53375098
## 70 4.607392 -0.53375368
## 71 4.607873 -0.53375587
## 72 4.608305 -0.53375765
## 73 4.608695 -0.53375909
## 74 4.609046 -0.53376026
## 75 4.609362 -0.53376121
```

```
minR = min(bGrid$R)
minB = subset(bGrid, bGrid$R == minR)

lines (bTrace, rTrace, type = "b", col = "blue")
text (-6, 1, "Answer: Beta = 4.61", col = "black", pos = 4)
```



Part (d)

```
b = seq(from = -6, by = 0.1, length.out = 121)
rBeta = function(b) {
  sin(b) + (b / 10)
}
graDesc = function(b) {
  cos(b) + (1 / 10)
}
```

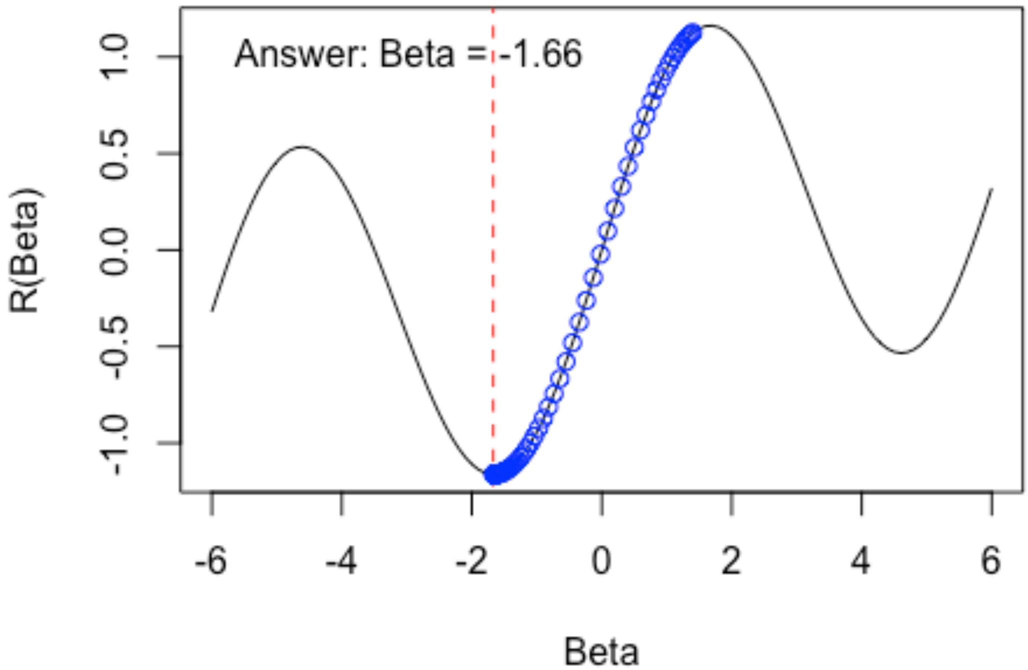
```
plot(b , rBeta(b), type = "l", xlab = "Beta", ylab = "R(Beta)")
lines (c(-1.67, -1.67), c(-2, 2), col = "red", lty = 2)

b = 1.4
bTrace = b
rTrace = rBeta(b)
rho = 0.1

for (step in 1:74) {
  b = b - (rho * graDesc(b))
  bTrace = c(bTrace, b)
  rTrace <- c(rTrace, rBeta(b))
}

bGrid = data.frame(Index = seq.int(nrow(bGrid)), Beta = bTrace, R = rTrace);
bGrid
```

```
##    Index       Beta          R
## 1      1 1.40000000 1.12544973
## 2      2 1.37300329 1.11780297
## 3      3 1.34335270 1.10858128
## 4      4 1.31080392 1.09747232
## 5      5 1.27509660 1.08410813
## 6      6 1.23595568 1.06805825
## 7      7 1.19309381 1.04882375
## 8      8 1.14621522 1.02583289
## 9      9 1.09502131 0.99844014
## 10    10 1.03921855 0.96593024
## 11    11 0.97852915 0.92753009
## 12    12 0.91270480 0.88243139
## 13    13 0.84154400 0.82982719
## 14    14 0.76491277 0.76896534
## 15    15 0.68276849 0.69922016
## 16    16 0.59518560 0.62018101
## 17    17 0.50238115 0.53175195
```

```
## 18    18  0.40473730  0.43425103
## 19    19  0.30281672  0.32849161
## 20    20  0.19736668  0.21582449
## 21    21  0.08930805  0.09812018
## 22    22 -0.02029342 -0.02232137
## 23    23 -0.13027283 -0.14293195
## 24    24 -0.23942548 -0.26108708
## 25    25 -0.34657291 -0.37433378
## 26    26 -0.45062715 -0.48059288
## 27    27 -0.55064456 -0.57830108
## 28    28 -0.64586331 -0.66647441
## 29    29 -0.73572135 -0.74469424
## 30    30 -0.81985603 -0.81303321
## 31    31 -0.89808868 -0.87194625
## 32    32 -0.97039928 -0.92215129
## 33    33 -1.03689629 -0.96451855
## 34    34 -1.09778574 -0.99997937
## 35    35 -1.15334258 -1.02945850
## 36    36 -1.20388600 -1.05382877
## 37    37 -1.24975931 -1.07388463
## 38    38 -1.29131439 -1.09032992
## 39    39 -1.32890016 -1.10377552
## 40    40 -1.36285456 -1.11474336
## 41    41 -1.39349920 -1.12367391
## 42    42 -1.42113618 -1.13093542
## 43    43 -1.44604638 -1.13683345
## 44    44 -1.46848905 -1.14162008
## 45    45 -1.48870194 -1.14550234
## 46    46 -1.50690216 -1.14864968
## 47    47 -1.52328723 -1.15120038
## 48    48 -1.53803635 -1.15326708
## 49    49 -1.55131176 -1.15494136
## 50    50 -1.56326010 -1.15629761
## 51    51 -1.57401371 -1.15739620
## 52    52 -1.58369197 -1.15828605
```

```
## 53    53 -1.59240244 -1.15900684
## 54    54 -1.60024200 -1.15959071
## 55    55 -1.60729786 -1.16006368
## 56    56 -1.61364852 -1.16044684
## 57    57 -1.61936461 -1.16075725
## 58    58 -1.62450969 -1.16100875
## 59    59 -1.62914094 -1.16121253
## 60    60 -1.63330978 -1.16137765
## 61    61 -1.63706251 -1.16151145
## 62    62 -1.64044074 -1.16161988
## 63    63 -1.64348193 -1.16170776
## 64    64 -1.64621977 -1.16177898
## 65    65 -1.64868457 -1.16183670
## 66    66 -1.65090362 -1.16188349
## 67    67 -1.65290146 -1.16192141
## 68    68 -1.65470016 -1.16195215
## 69    69 -1.65631962 -1.16197707
## 70    70 -1.65777771 -1.16199727
## 71    71 -1.65909054 -1.16201365
## 72    72 -1.66027259 -1.16202693
## 73    73 -1.66133689 -1.16203769
## 74    74 -1.66229520 -1.16204642
## 75    75 -1.66315808 -1.16205349
```

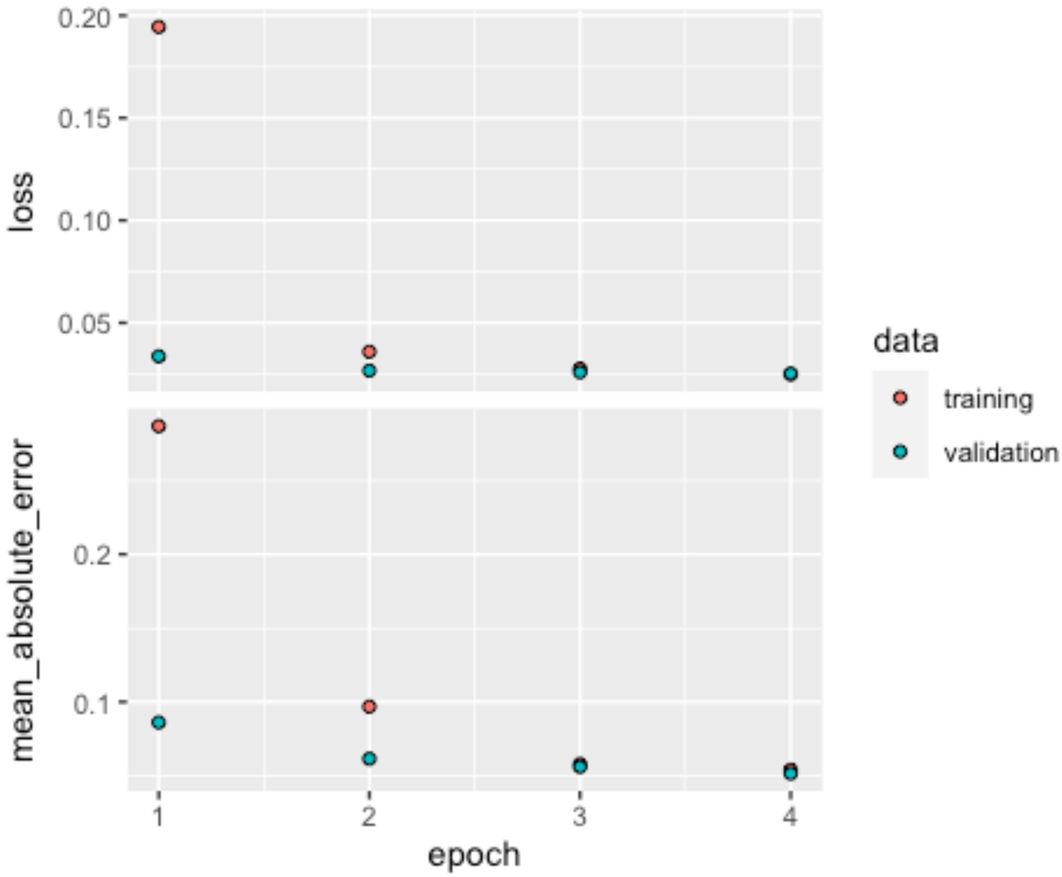
```
minR = min(bGrid$R)
minB = subset(bGrid, bGrid$R == minR)

lines (bTrace, rTrace, type = "b", col = "blue")
text (-6, 1, "Answer: Beta = -1.66", col = "black", pos = 4)
```



Problem-7

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.1.2

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.1.2

## Loaded glmnet 4.1-4

library(keras)

## Warning: package 'keras' was built under R version 4.1.2

rawData = read.csv("/Users/Home/Documents/Michael_Ghattas/School/CU_Boulder/
BA-BS/2022/Fall/STAT_4610/Labs/Data/Default.csv")
```

```r
data = na.omit(rawData)
data$default = ifelse(data$default == "Yes", 1 , 0)
data$student = ifelse(data$student == "Yes", 1 , 0)

n <- nrow(data)
set.seed(123)
ntest <- trunc(n / 3)
testid <- sample(1:n, ntest)

logfit <- glm(default ~ ., data = data[-testid, ], family = "binomial")
lpred <- predict(logfit, data[testid, ])

x <- scale(model.matrix(default ~ . - 1, data = data))
y <- data$default
cvfit <- cv.glmnet(x[-testid, ], y[-testid], type.measure = "mae")
cpred <- predict(cvfit, x[testid, ], s = "lambda.min")

modnn <- keras_model_sequential() %>% layer_dense(units = 10, activation =
"relu", input_shape = ncol(x)) %>% layer_dropout(rate = 0.4) %>%
layer_dense(units = 1)

## Loaded Tensorflow version 2.4.1

x <- model.matrix(default ~ . - 1, data = data) %>% scale()
modnn %>% compile(loss = "mse", optimizer = optimizer_rmsprop(), metrics =
list("mean_absolute_error"))
history <- modnn %>% fit(x[-testid, ], y[-testid], epochs = 4, batch_size =
24, validation_data = list(x[testid, ], y[testid]))

plot(history)
```




```
npred <- predict(modnn, x[testid, ])
mean(abs(y[testid] - cpred))

## [1] 0.06309743

mean(abs(y[testid] - npred))

## [1] 0.05143043
```

-> By setting the epochs = 4 and batch_size = 24, we are able to reduce the error to ~0.051
-> Error from the linear logistic regression ~0.063
-> Better performance was achieved by the neural network