BIOSTAT C161 HW3

:)

Hanxi Chen

2025-10-31

```r
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.4.3
```

```r
library(dplyr)
library(caret)
```

Q1

(iii)

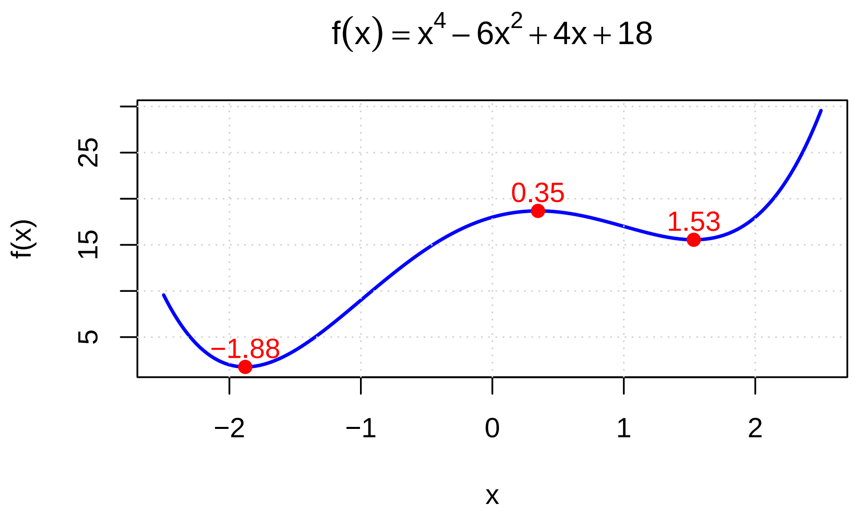
Plot the function f over x $[-2.5, 2.5]$. How many local/global minima do you see? What are their approximate values? Can there be other local minima?

Sol:

```r
# Clear environment
rm(list = ls())



f <- function(x) {
  x^4 - 6*x^2 + 4*x + 18
}

x_vals <- seq(-2.5, 2.5, by = 0.01)
y_vals <- f(x_vals)

plot(x_vals, y_vals, type = "l", col = "blue", lwd = 2,
     main = expression(f(x) == x^4 - 6*x^2 + 4*x + 18),
     xlab = "x", ylab = "f(x)")

grid()
f_prime <- function(x) 4*x^3 - 12*x + 4
roots <- uniroot.all <- function(f, interval, n = 1000, ...) {
  xseq <- seq(interval[1], interval[2], len = n)
  yseq <- f(xseq)
  sign_change <- which(diff(sign(yseq)) != 0)
  sapply(sign_change, function(i)
    uniroot(f, c(xseq[i], xseq[i + 1]), ...)$root)
}

critical_points <- uniroot.all(f_prime, c(-2.5, 2.5))
critical_points
```

```
[1] -1.8793794  0.3472973  1.5320821
```

```r
points(critical_points, f(critical_points), col = "red", pch = 19)
text(critical_points, f(critical_points) + 2,
     labels = round(critical_points, 2), col = "red")
```



```r
data.frame(
  x = critical_points,
  f_x = f(critical_points)
)
```

```
           x       f_x
1 -1.8793794  1.765578
2  0.3472973 18.680045
3  1.5320821 15.554378
```

The plot of the function over the interval $[-2.5, 2.5]$ shows three stationary points at approximately x = −1.88, 0.35, and 1.53. Based on their function values f(−1.88) 1.77, f(0.35) 18.68, and f(1.53) 15.55, the points at x = −1.88 and x = 1.53 are local minima, while x = 0.35 is a local maximum. The global minimum occurs at x −1.88 because it gives the smallest f(x). There cannot be any additional local minima since the derivative 4x^3 − 12x + 4 is cubic and thus has at most three real roots.

(iv)

Write your own code for S steps of GD on this function (do not use built-in or third-party GD codes).

Sol:

```r
# Gradient Descent
gradient_descent <- function(x0, alpha, S) {
  x_vals <- numeric(S + 1)  # store all iterates
  x_vals[1] <- x0           # initial value
  grad_vals <- numeric(S + 1)  # store gradients

  for (s in 1:S) {
    grad <- f_prime(x_vals[s])
    grad_vals[s] <- grad
    x_vals[s + 1] <- x_vals[s] - alpha * grad
  }

  grad_vals[S + 1] <- f_prime(x_vals[S + 1])  # gradient at last step


  return(data.frame(Step = 0:S,
                    x = x_vals,
                    f_x = f(x_vals),
                    gradient = grad_vals))
}
```

(v)

Repeat (i) and (ii) using your code with S = 20 steps. Do you observe convergence in both cases?

Sol:

```r
# Case 1: Start from x(0) = 1
result_1 <- gradient_descent(x0 = 1, alpha = 0.1, S = 20)
print(result_1)
```

```
   Step        x      f_x      gradient
1     0 1.000000 17.00000 -4.0000000000
```

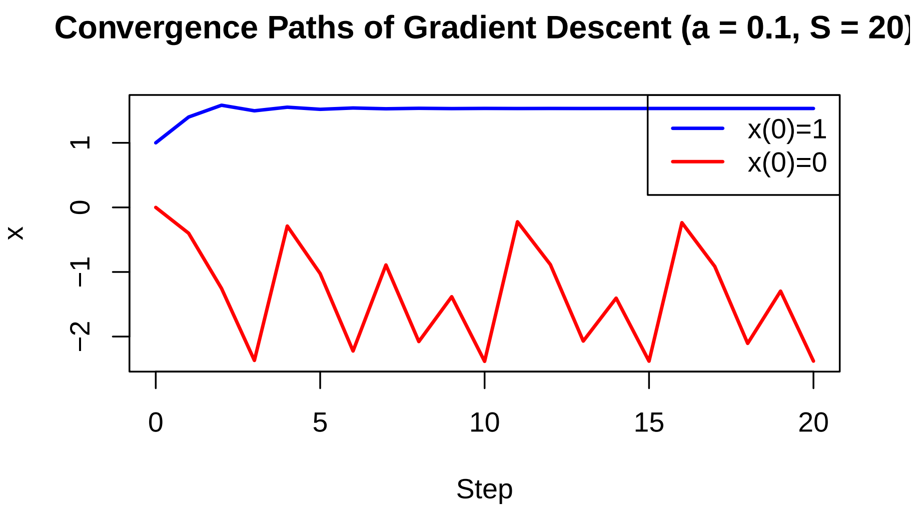
```
2      1 1.400000 15.68160 -1.8240000000
3      2 1.582400 15.57563  0.8604535849
4      3 1.496355 15.56442 -0.5544413756
5      4 1.551799 15.55757  0.3258336573
6      5 1.519215 15.55570 -0.2050942422
7      6 1.539725 15.55485  0.1245284454
8      7 1.527272 15.55456 -0.0774512605
9      8 1.535017 15.55445  0.0475001130
10     9 1.530267 15.55440 -0.0293927124
11    10 1.533206 15.55439  0.0180900918
12    11 1.531397 15.55438 -0.0111713403
13    12 1.532515 15.55438  0.0068845274
14    13 1.531826 15.55438 -0.0042481318
15    14 1.532251 15.55438  0.0026192715
16    15 1.531989 15.55438 -0.0016157492
17    16 1.532151 15.55438  0.0009964086
18    17 1.532051 15.55438 -0.0006145838
19    18 1.532112 15.55438  0.0003790316
20    19 1.532074 15.55438 -0.0002337761
21    20 1.532098 15.55438  0.0001441804
```

```
# Case 2: Start from x(0) = 0
result_2 <- gradient_descent(x0 = 0, alpha = 0.1, S = 20)
print(result_2)
```

```
   Step          x       f_x   gradient
1     0  0.0000000 18.000000   4.000000
2     1 -0.4000000 15.465600   8.544000
3     2 -1.2544000  6.017247  11.157509
4     3 -2.3701509  6.371307 -20.816574
5     4 -0.2884935 16.353582   7.365878
6     5 -1.0250813  8.699088  11.992388
7     6 -2.2243201  3.895865 -13.328344
8     7 -0.8914858 10.297198  11.863807
9     8 -2.0778665  2.424417  -6.950597
10    9 -1.3828067  4.652181  10.017121
11   10 -2.3845188  6.676144 -21.618601
12   11 -0.2226587 16.814362   6.627750
13   12 -0.8854337 10.368953  11.848510
14   13 -2.0702846  2.372859  -6.650193
15   14 -1.4052653  4.430034   9.762877
16   15 -2.3815530  6.612276 -21.452085
17   16 -0.2363445 16.722590   6.783327
18   17 -0.9146772 10.021442  11.915125
19   18 -2.1061897  2.637438  -8.098250
20   19 -1.2963648  5.555459  10.841894
```

```
21   20 -2.3805541  6.590875 -21.396114
```

```r
# Plot convergence paths
plot(result_1$Step, result_1$x, type = "l", col = "blue", lwd = 2,
     ylim = range(c(result_1$x, result_2$x)),
     main = "Convergence Paths of Gradient Descent (  = 0.1, S = 20)",
     xlab = "Step", ylab = "x")
lines(result_2$Step, result_2$x, col = "red", lwd = 2)
legend("topright", legend = c("x(0)=1", "x(0)=0"), col = c("blue", "red"), lwd = 2)
```



The plot shows the convergence behavior of gradient descent with = 0.1 and S = 20 for two different initial points. When starting from x(0) = 1 (blue line), the algorithm quickly stabilizes near x 1.53, and the final gradient value of 0.000144 indicates that it has effectively converged to a local minimum. In contrast, when starting from x(0) = 0 (red line), the sequence oscillates strongly between negative and positive values, showing that the updates overshoot due to a large learning rate. The final gradient of −21.396 further confirms divergence rather than convergence. Therefore, gradient descent converges successfully only for the initial value x(0) = 1, while it fails to converge when starting from x(0) = 0 under the same learning rate.

(vi)

Now set the learning rate to = 0.01 and repeat (v). Explain why GD performs differently from (v)

Sol:

```r
# Case 1: Starting from x(0) = 1
result_x1_small_alpha <- gradient_descent(x0 = 1, alpha = 0.01, S = 20)
print(result_x1_small_alpha)
```

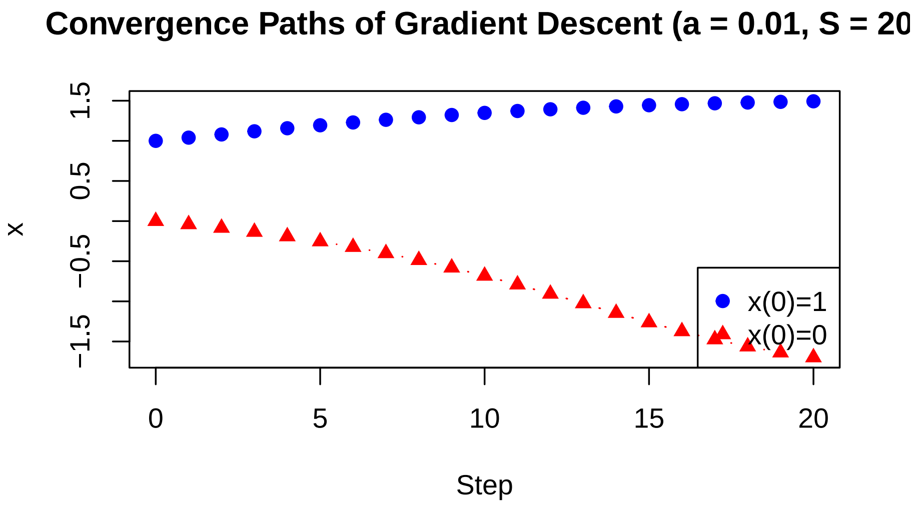
```
   Step        x      f_x   gradient
1     0 1.000000 17.00000 -4.0000000
2     1 1.040000 16.84026 -3.9805440
3     2 1.079805 16.68285 -3.9215400
4     3 1.119021 16.53086 -3.8232643
5     4 1.157253 16.38715 -3.6877014
6     5 1.194130 16.25416 -3.5184957
7     6 1.229315 16.13374 -3.3207383
8     7 1.262523 16.02703 -3.1006107
9     8 1.293529 15.93447 -2.8649282
10    9 1.322178 15.85583 -2.6206474
11   10 1.348385 15.79033 -2.3744008
12   11 1.372129 15.73679 -2.1321135
13   12 1.393450 15.69379 -1.8987381
14   13 1.412437 15.65982 -1.6781168
15   14 1.429218 15.63336 -1.4729619
16   15 1.443948 15.61304 -1.2849314
17   16 1.456797 15.59762 -1.1147662
18   17 1.467945 15.58604 -0.9624615
19   18 1.477570 15.57742 -0.8274449
20   19 1.485844 15.57106 -0.7087435
21   20 1.492931 15.56641 -0.6051298
```

```r
# Case 2: Starting from x(0) = 0
result_x0_small_alpha <- gradient_descent(x0 = 0, alpha = 0.01, S = 20)
print(result_x0_small_alpha)
```

```
   Step           x       f_x  gradient
1     0  0.00000000 18.000000  4.000000
2     1 -0.04000000 17.830403  4.479744
3     2 -0.08479744 17.617718  5.015130
4     3 -0.13494874 17.351270  5.609555
5     4 -0.19104429 17.018167  6.264641
6     5 -0.25369070 16.603225  6.978979
7     6 -0.32348049 16.089190  7.746370
8     7 -0.40094419 15.457528  8.553513
9     8 -0.48647932 14.690119  9.377227
10    9 -0.58025159 13.772204 10.181555
11   10 -0.68206714 12.696863 10.915573
12   11 -0.79122287 11.470825 11.513346
13   12 -0.90635633 10.120516 11.898055
14   13 -1.02533688  8.696023 11.992231
```

```
15   14 -1.14525919  7.269595 11.734537
16   15 -1.26260457  5.925939 11.100028
17   16 -1.37360485  4.744817 10.116441
18   17 -1.47476926  3.781640  8.867067
19   18 -1.56343992  3.054993  7.474937
20   19 -1.63818929  2.547311  6.072872
21   20 -1.69891801  2.217251  4.772516
```

```
plot(result_x1_small_alpha$Step, result_x1_small_alpha$x,
     type = "b", col = "blue", pch = 19,
     ylim = range(c(result_x1_small_alpha$x, result_x0_small_alpha$x)),
     main = "Convergence Paths of Gradient Descent (  = 0.01, S = 20)",
     xlab = "Step", ylab = "x")
lines(result_x0_small_alpha$Step, result_x0_small_alpha$x,
      type = "b", col = "red", pch = 17)
legend("bottomright", legend = c("x(0)=1", "x(0)=0"),
       col = c("blue", "red"), pch = c(19, 17))
```



With a smaller learning rate (= 0.01), the updates become smoother and more stable. The path starting from x(0) = 1 moves slowly toward the local minimum near x   1.53, while the path from x(0) = 0 moves gradually toward the global minimum near x   −1.88. However, after 20 steps, neither starting point has reached a point where the gradient is close to 0, indicating that convergence is not yet achieved due to the slower update rate.

Q2

The College.csv dataset contains admissions data for a sample of 777 universities. We want to predict the number of applications received ("Apps") using the other variables in the dataset

(i)

Let the first 600 observations be the training set and the remaining 177 observations be the test set

Sol:

```r
college <- read.csv("./College.csv")

head(college)
```

```
                             X Private Apps Accept Enroll Top10perc Top25perc
1 Abilene Christian University     Yes 1660   1232    721        23        52
2           Adelphi University     Yes 2186   1924    512        16        29
3               Adrian College     Yes 1428   1097    336        22        50
4          Agnes Scott College     Yes  417    349    137        60        89
5    Alaska Pacific University     Yes  193    146     55        16        44
6            Albertson College     Yes  587    479    158        38        62
  F.Undergrad P.Undergrad Outstate Room.Board Books Personal PhD Terminal
1        2885         537     7440       3300   450     2200  70       78
2        2683        1227    12280       6450   750     1500  29       30
3        1036          99    11250       3750   400     1165  53       66
4         510          63    12960       5450   450      875  92       97
5         249         869     7560       4120   800     1500  76       72
6         678          41    13500       3335   500      675  67       73
  S.F.Ratio perc.alumni Expend Grad.Rate
1      18.1          12   7041        60
2      12.2          16  10527        56
3      12.9          30   8735        54
4       7.7          37  19016        59
5      11.9           2  10922        15
6       9.4          11   9727        55
```

```r
college <- college[, -1]
college$Private <- as.factor(college$Private)
train <- college[1:600, ]
test <- college[601:777, ]

dim(train)
```

```
[1] 600  18
```

```r
dim(test)
```

```
[1] 177  18
```

```r
train_x <- model.matrix(Apps ~ ., data = train)[, -1]  # remove intercept
test_x <- model.matrix(Apps ~ ., data = test)[, -1]
train_y <- train$Apps
test_y <- test$Apps
```

(ii)

fit the OLS regression on the training set, and report the test error obtained. For the rest of the problem, let the penalization parameter vary on the 1000-point grid from 0.01 to 60 .

Sol:

```r
library(glmnet)

# Fit OLS model (lambda = 0 no regularization)
ols_fit <- lm(Apps ~ ., data = train)

ols_pred <- predict(ols_fit, newdata = test)

ols_mse <- mean((ols_pred - test_y)^2)
ols_mse
```

```
[1] 1502077
```

```r
sqrt(ols_mse)
```

```
[1] 1225.593
```

```r
lambda_grid <- seq(0.01, 60, length = 1000)
```

The test MSE = 1502077, RMSE is 1225.593.

(iii)

Fit the LASSO regression on the training set, with the penalization parameter chosen by 20-fold cross-validation. Report the test error obtained

Sol:

```r
# alpha = 1 for LASSO
set.seed(123)
lasso_cv <- cv.glmnet(train_x, train_y, alpha = 1, lambda = lambda_grid, nfolds = 20)

# Best lambda
```

```r
best_lambda_lasso <- lasso_cv$lambda.min
best_lambda_lasso
```

```
[1] 0.01
```

```r
lasso_pred <- predict(lasso_cv, s = best_lambda_lasso, newx = test_x)
lasso_mse <- mean((lasso_pred - test_y)^2)
lasso_rmse <- sqrt(lasso_mse)

lasso_mse
```

```
[1] 1499849
```

```r
lasso_rmse
```

```
[1] 1224.683
```

The optimal penalty parameter selected by 20-fold cross-validation is = 0.01, yielding a test MSE of approximately 1,499,849 and a corresponding RMSE of about 1224.68, indicating the average prediction error in the number of applications is around 1225.

(iv)

Fit the ridge regression on the training set, with the penalization parameter chosen by leave-one-out cross-validation. Report the test error obtained. **Sol:**

```r
set.seed(123)

ridge_cv <- cv.glmnet(train_x, train_y, alpha = 0, lambda = lambda_grid,
                      nfolds = nrow(train))
```

```
Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
fold
```

```r
# Best lambda
best_lambda_ridge <- ridge_cv$lambda.min
best_lambda_ridge
```

```
[1] 0.01
```

```r
ridge_pred <- predict(ridge_cv, s = best_lambda_ridge, newx = test_x)
ridge_mse <- mean((ridge_pred - test_y)^2)
ridge_rmse <- sqrt(ridge_mse)

ridge_mse
```

```
[1] 1501329
```

```r
ridge_rmse
```

```
[1] 1225.287
```

The optimal   selected by leave-one-out cross-validation is 0.01, giving a test MSE of approximately 1,501,329 and an RMSE of about 1225.29. This test error is nearly identical to that of the LASSO model, suggesting that both methods achieve similar predictive performance on this dataset.

(v)

which of the three models do you prefer? Is there much difference among the test errors?

Sol: All three models: OLS, LASSO, and ridge regression have very similar test errors, with RMSE values all around 1225. This indicates that neither penalization method provides a meaningful improvement over ordinary least squares for predicting the number of applications. Since the predictive performance is nearly identical, the OLS model would be preferred for its simplicity and ease of interpretation, although LASSO could still be useful for variable selection if model sparsity is desired.