Assignment 2 - Question 2

Toni Ciobanu (20910287)

2024-02-17

QUESTION 2: Price's Function

(a) [4 points] Write `rho` and `gradient` functions for Price's Function which take a single vector-valued input `theta`.

SOLUTION:

```r
# rho function
rho <- function(theta) {
  100 * (theta[2] - theta[1]^2)^2 + (6.4 * (theta[2] - 0.5)^2 - theta[1] - 0.6)^2
}

# gradient function
gradient <- function(theta) {
  grad1 <- -400 * theta[1] * (theta[2] - theta[1]^2) - 2 *
    (6.4 * (theta[2] - 0.5)^2 - theta[1] - 0.6)
  grad2 <- 200 * (theta[2] - theta[1]^2) + 25.6 * (theta[2] - 0.5) *
    (6.4 * (theta[2] - 0.5)^2 - theta[1] - 0.6)
  return(c(grad1, grad2))
}
```

(b) [5 points] In this question, you will explore the surface of Price's Function using gradient descent. In particular, you will consider 5 different starting values and explore the impact of changing one's starting location. Using the `gradientDescent()` function (from class) together with the `gridLineSearch()` and `testConvergence()` functions (from class) as well as the `rho` and `gradient` functions from part (a), find the solution to

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^2}{\operatorname{argmin}}\, \rho(\boldsymbol{\theta})$$

for each of the following five starting values. In each case, state which minima you've converged to (A, B, or C) and be sure to include the output from the `gradientDescent()` function.

SOLUTION:

Lets Start off by defining all functions needed from class:

```r
gradientDescent <- function(theta = 0, rhoFn, gradientFn, lineSearchFn, testConvergenceFn,
    maxIterations = 100, tolerance = 1e-06, relative = FALSE, lambdaStepsize = 0.01,
    lambdaMax = 0.5) {
```

```r
    converged <- FALSE
    i <- 0

    while (!converged & i <= maxIterations) {
        g <- gradientFn(theta)  ## gradient
        glength <- sqrt(sum(g^2))  ## gradient direction
        if (glength > 0)
            d <- g/glength

        lambda <- lineSearchFn(theta, rhoFn, d, lambdaStepsize = lambdaStepsize,
            lambdaMax = lambdaMax)

        thetaNew <- theta - lambda * d
        converged <- testConvergenceFn(thetaNew, theta, tolerance = tolerance,
            relative = relative)
        theta <- thetaNew
        i <- i + 1
    }

    ## Return last value and whether converged or not
    list(theta = theta, converged = converged, iteration = i, fnValue = rhoFn(theta))
}

### Where testCovergence might be (relative or absolute)
testConvergence <- function(thetaNew, thetaOld, tolerance = 1e-10, relative = FALSE) {
    sum(abs(thetaNew - thetaOld)) < if (relative)
        tolerance * sum(abs(thetaOld)) else tolerance
}

### line searching could be done as a simple grid search
gridLineSearch <- function(theta, rhoFn, d, lambdaStepsize = 0.01, lambdaMax = 1) {
    ## grid of lambda values to search
    lambdas <- seq(from = 0, by = lambdaStepsize, to = lambdaMax)
    ## line search
    rhoVals <- sapply(lambdas, function(lambda) {
        rhoFn(theta - lambda * d)
    })
    ## Return the lambda that gave the minimum
    lambdas[which.min(rhoVals)]
}
```

i. $\widehat{\boldsymbol{\theta}}_0 = (1.1, 1.75)$

```r
result_i <- gradientDescent(
  theta = c(1.1, 1.75),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence
)

result_i
```

```
## $theta
## [1] 0.9999632 1.0079989
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 15
##
## $fnValue
## [1] 0.00918294
```

The minima the starting point converged to is C (1, 1)

ii. $\widehat{\boldsymbol{\theta}}_0 = (-0.5, 0.6)$

```r
result_ii <- gradientDescent(
  theta = c(-0.5, 0.6),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence
)

result_ii
```

```
## $theta
## [1] -0.6628252  0.4375563
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 2
##
## $fnValue
## [1] 0.008022531
```

The minima the starting point converged to is A (-0.664, 0441)

iii. $\widehat{\boldsymbol{\theta}}_0 = (-0.2, -0.8)$

```r
result_iii <- gradientDescent(
  theta = c(-0.2, -0.8),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence
)

result_iii
```

```
## $theta
## [1] 0.3455740 0.1131815
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 29
##
## $fnValue
## [1] 0.004038851
```

The minima the starting point converged to is B (0.342, 0.116)

iv. $\widehat{\boldsymbol{\theta}}_0 = (0.1, 1.1)$

```
result_iv <- gradientDescent(
  theta = c(0.1, 1.1),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence
)

result_iv
```

```
## $theta
## [1] 0.3469271 0.1139806
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 18
##
## $fnValue
## [1] 0.004113065
```

The minima the starting point converged to is B (0.342, 0.116)

v. $\widehat{\boldsymbol{\theta}}_0 = (0.75, 0.6)$

```
result_v <- gradientDescent(
  theta = c(0.75, 0.6),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence
)

result_v
```

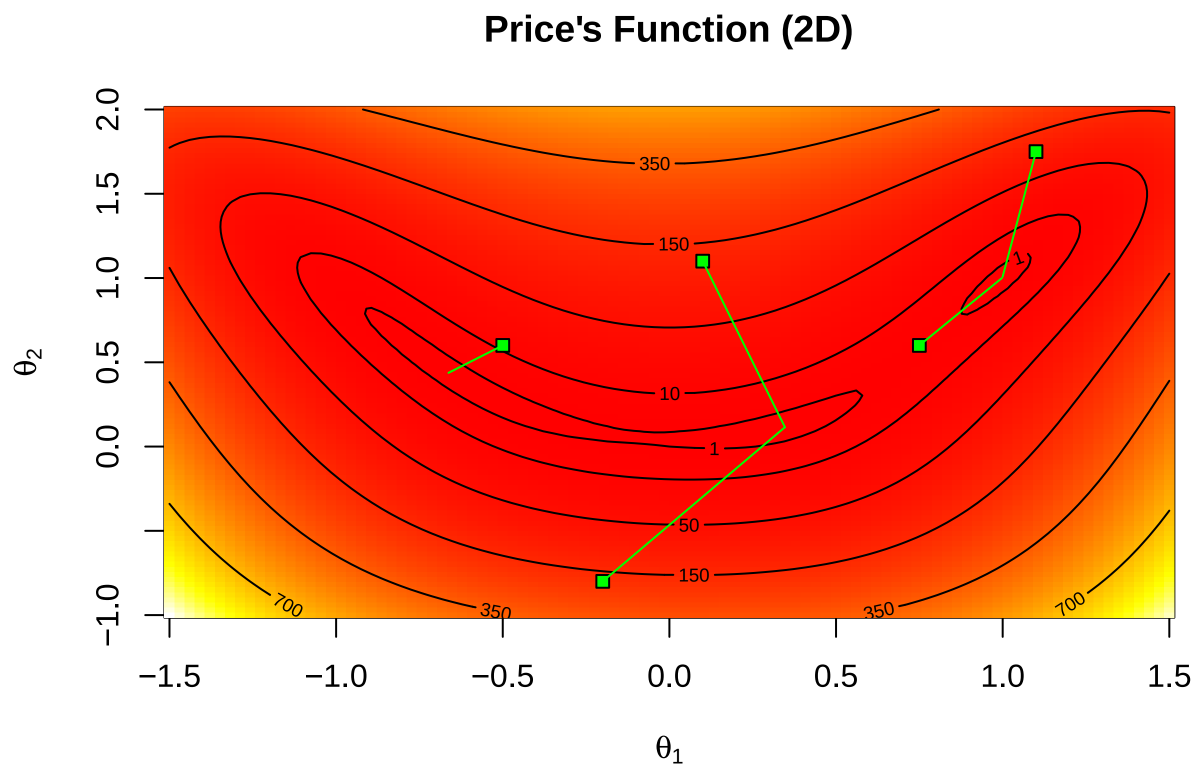
```
## $theta
## [1] 0.9975885 0.9996567
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 19
##
## $fnValue
## [1] 0.002001588
```

The minima the starting point converged to is C (1, 1)

(c) [5 points] Recreate the contour plot shown above. You may find the functions `outer()`, `image()`, and `contour()` useful for this task. Include on this plot `green` squares at each of the starting points specified in (b) as well as `green` line `segments()` connecting these starting points with their respective points of convergence.

SOLUTION:

```r
# Define Price's Function
rho <- function(theta1, theta2) {
  100 * (theta2 - theta1^2)^2 + (6.4 * (theta2 - 0.5)^2 - theta1 - 0.6)^2
}

# Create a grid of values for theta1 and theta2
theta1 <- seq(-1.5, 1.5, length = 100)
theta2 <- seq(-1, 2, length = 100)

# Calculate Rho values on the grid
Rho <- outer(theta1, theta2, "rho")

image(theta1, theta2, Rho, col = heat.colors(1000),
      xlab = bquote(theta[1]),
      ylab = bquote(theta[2]), main = "Price's Function (2D)")

# Plot the contour map
contour(theta1, theta2, Rho, add=T, levels = c(1,10,50,150,350,700))

# Add starting points as green squares
points(x = c(1.1, -0.5, -0.2, 0.1, 0.75),
       y = c(1.75, 0.6, -0.8, 1.1, 0.6), pch = 22, bg = "green")

# Draw green line segments connecting starting points with convergence points
segments(x0 = c(1.1, -0.5, -0.2, 0.1, 0.75),
         y0 = c(1.75, 0.6, -0.8, 1.1, 0.6),
         x1 = c(0.9999632, -0.6628252, 0.3455740, 0.3469271, 0.9975885),
         y1 = c(1.0079989, 0.4375563, 0.1131815, 0.1139806, 0.9996567), col = "green")
```



(d) [2 points] Based on what you found in part (b), and visualized in part (c), explain the importance of the starting value when performing non-convex optimization (when locating a global optimum is desired).

SOLUTION:

From what I've found in part b and my visualizations in part c, I can understand the importance of relying on multiple data points to understand the function's behavior better. Point's A, B, or C are all local minimums to each individual starting point, and the points converge to their respective local minimum. However, that doesn't mean that it is the global minimum. There could be other local minimums as well, clearly demonstrated by Price's Function.This also shows that although gradient descent is a great tool for optimization, the effectiveness for irregular functions like this one is heavily dependent on the chosen starting point.

(e) [4 points] Repeat the process from part (b) for an additional starting value: $\widehat{\boldsymbol{\theta}}_0 = (-1, -0.5)$. Describe what occurs when using the `gradientDescent()` function with this starting value, where the inputs `maxIterations`, `tolerance`, `relative`, `lambdaStepsize`, and `lambdaMax` are set to their default values from the course notes. Try to obtain better performance by adjusting *one* of the inputs in the previous sentence from its default value. Which input can be tweaked to yield better performance, and why is this input important for gradient descent methods?

SOLUTION:

```r
# rho function
rho <- function(theta) {
  100 * (theta[2] - theta[1]^2)^2 + (6.4 * (theta[2] - 0.5)^2 - theta[1] - 0.6)^2
}

# gradient function
gradient <- function(theta) {
  grad1 <- -400 * theta[1] * (theta[2] - theta[1]^2) - 2 * (6.4 * (theta[2] - 0.5)^2 - theta[1] - 0.6)
  grad2 <- 200 * (theta[2] - theta[1]^2) + 25.6 * (theta[2] - 0.5) * (6.4 * (theta[2] - 0.5)^2 - theta[
  return(c(grad1, grad2))
}

# Gradient Descent with default settings
gradientDescent(
  theta = c(-1, -0.5),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence,
  maxIterations = 100,
  tolerance = 1e-06,
  relative = FALSE,
  lambdaStepsize = 0.01,
  lambdaMax = 0.5
)
```

```
## $theta
## [1] -0.5791405  0.3312629
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 7
##
## $fnValue
## [1] 0.02775267
```

Initially, we can observe a new convergence point other than A, B, or C. This convergence point looks to be around (-0.58, 0.33), which is close to 0 on the contour map given above.

```r
# Gradient Descent with updated lambdaStepSize
gradientDescent(
  theta = c(-1, -0.5),
  rhoFn = rho,
  gradientFn = gradient,
  lineSearchFn = gridLineSearch,
  testConvergenceFn = testConvergence,
  maxIterations = 100,
  tolerance = 1e-06,
  relative = FALSE,
  lambdaStepsize = 0.1,
  lambdaMax = 0.5
)
```

```
## $theta
## [1] -0.3456598  0.1107975
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 3
##
## $fnValue
## [1] 0.5189409
```

Increasing the step size from 0.01 to 0.1 seems to have helped the point reach a convergence rate closer to B (0.342, 0.116). It is still further off than the rest of the points, as the step size likely skipped over the convergence point, but it is accurate to the nearest 2 decimal places. This input is important for gradient descent methods because it takes less iterations to get to where you'd need to go. If you want to sacrifice precise numbers for efficiency, a larger step size is a good way to go. A larger step size can also be somewhat risky, as it can skip right over local minimas completely.