Exercise - Gradient Descent

Qi

2025-04-17

Contents

1 Introduction

In this exercise, we explore the concept of **gradient descent**, a fundamental optimization algorithm used to minimize functions. We will apply it to a simple one-dimensional function and approximate the derivative using a secant slope (finite differences) instead of requiring the user to provide the analytical derivative.

2 What is Gradient Descent?

Gradient descent is an iterative optimization algorithm used to find the local minimum of a differentiable function. The idea is to start from an initial guess and move in the direction opposite to the gradient (i.e., the direction of steepest descent), scaled by a learning rate.

2.1 The General Update Rule:

Given a function $f(x)$, the update rule for gradient descent is:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

Where:

- x_t is the current estimate of the minimum
- η is the learning rate (step size)
- $\nabla f(x_t)$ is the derivative of f at x_t

In this exercise, we approximate the derivative numerically.

3 Function to Minimize

We will apply gradient descent to the following function:

$$f(x) = x + \frac{1}{x}, \quad x > 0$$

This function is smooth and has a unique minimum on the domain $(0, \infty)$.

4 Step-by-Step Implementation in R

4.1 Step 1: Define the Function

```r
f <- function(x) {
  y <- ifelse(x > 0, x + 1 / x, Inf)  # handle vectorized input
  return(y)
}
```

4.2 Step 2: Implement the Gradient Descent Algorithm with Numerical Derivative

```
gradient_descent <- function(f, x0, eta = 0.01, tol = 1e-6, max_iter = 1000, h = 1e-6) {
  x <- x0
  iter <- 0
  path <- numeric(max_iter)

  repeat {
    # numerical derivative via finite differences
    grad <- (f(x + h) - f(x - h)) / (2 * h)
    x_new <- x - eta * grad
    if (x_new <= 0) x_new <- x / 2  # keep x in (0, ∞)

    path[iter + 1] <- x_new
    if (abs(x_new - x) < tol || iter >= max_iter) {
      break
    }
    x <- x_new
    iter <- iter + 1
  }

  list(
    x_min = x_new,
    f_min = f(x_new),
    iterations = iter,
    path = path[1:iter]
  )
}
```

4.3 Step 3: Run the Algorithm

```
result <- gradient_descent(f, x0 = 2, eta = 0.05)

cat("Minimum point:", result$x_min, "\n")
```

```
## Minimum point: 1.000009
```

```
cat("Minimum value:", result$f_min, "\n")
```

```
## Minimum value: 2
```

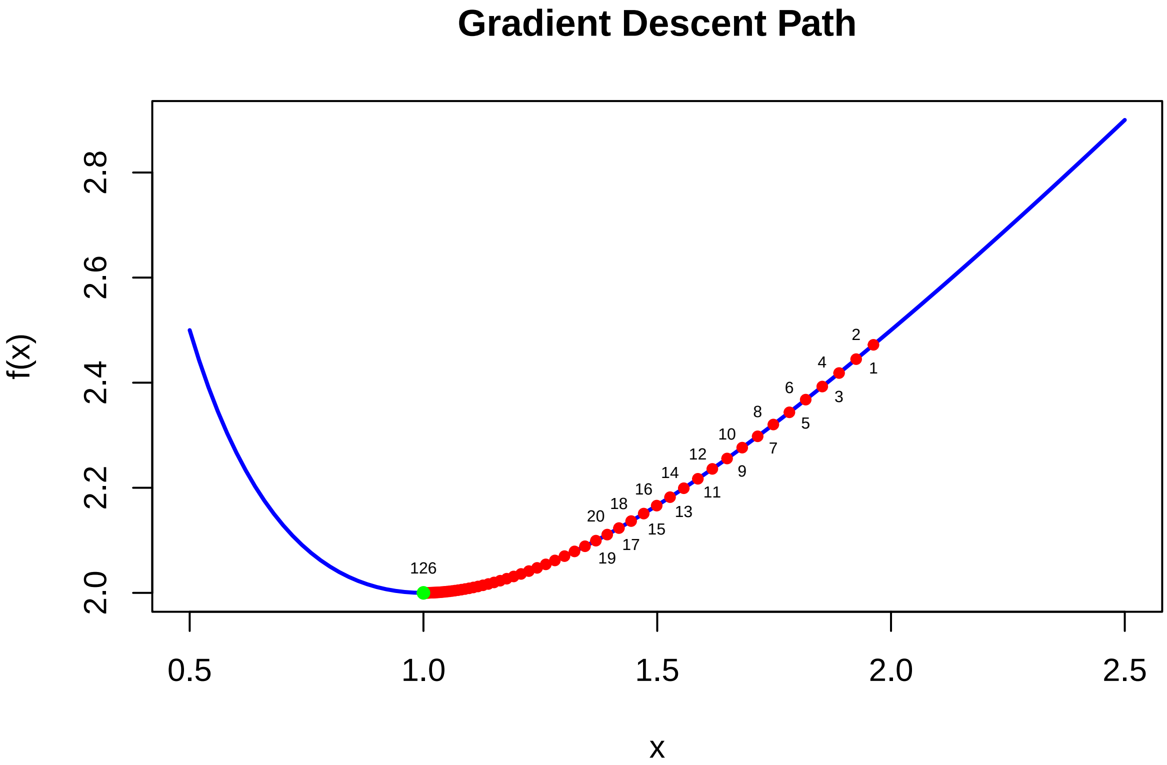
```
cat("Iterations:", result$iterations, "\n")
```

```
## Iterations: 126
```

4.4 Step 4: Visualize the Optimization Path

```r
curve(f, from = 0.5, to = 2.5, col = "blue", lwd = 2, ylab = "f(x)", main = "Gradient Descent Path")
points(result$path, f(result$path), col = "red", pch = 20)
points(result$path[length(result$path)], f(result$path)[length(result$path)], col = "green", pch = 20,
text(result$path[1:20], f(result$path[1:20]), labels = seq_along(result$path[1:20]), pos = c(1,3), cex

text(result$path[length(result$path)], f(result$path[length(result$path)]), labels = length(result$path
```



5 Discussion

- The gradient descent algorithm converged to a unique minimum near $x = 1$.
- Because the function is convex on $(0, \infty)$, the algorithm reliably converges regardless of the initial point (as long as $x > 0$).
- We used numerical differentiation instead of an analytical gradient to make the implementation more general.

6 Conclusion

This exercise illustrates how gradient descent works with numerical gradients on a convex function. Understanding the step-by-step behavior helps build intuition for tuning parameters and diagnosing optimization issues in more complex models.