Numerical Optimization: Basic Concepts

Code

Ruoqing Zhu

Last Updated: August 20, 2024

A Motivating Example: Linear Regression

Although we have the analytic solution of a linear regression, it can still be treated as an optimization problem. In addition, this view of a linear regression will benefit us when we introduce the Lasso and Ridge regressions.

Let's consider a simple linear regression, in which we observe a set of observations $\{x_i, y_i\}_{i=1}^n$, our optimization problem is to find parameters β_0 and β_1 to minimize the objection function, i.e., residual sum of squares (RSS):

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \ell(\boldsymbol{\beta}) = \frac{1}{n}\sum_i (y_i - \beta_0 - x_i\beta_1)^2$$

Let's first generate a set of data. We have two parameters, an intercept $\beta_= 0.5$ and a slope $\beta_1 = 1$.

Hide

```
# generate data from a simple linear model
set.seed(20)
n = 150
x <- rnorm(n)
y <- 0.5 + x + rnorm(n)
```

Suppose we have a candidate estimation of $\boldsymbol{\beta}$, say $\widehat{\boldsymbol{\beta}} = c(0.3, 1.5)$, then we can calculate the RSS using

$$\frac{1}{n}\sum_i (y_i - 0.3 - 1.5 \times x_i)^2$$

We can then make a function that calculates the RSS for any value of $\boldsymbol{\beta}$:

Hide

```
  # calculate the residual sum of squares for a grid of beta values
  rss <- function(b, trainx, trainy) mean((trainy - b[1] - trainx * b[2])^2)

  # Try it on a beta value
  rss(b = c(0.3, 1.5), trainx = x, trainy = y)
## [1] 1.371949
```

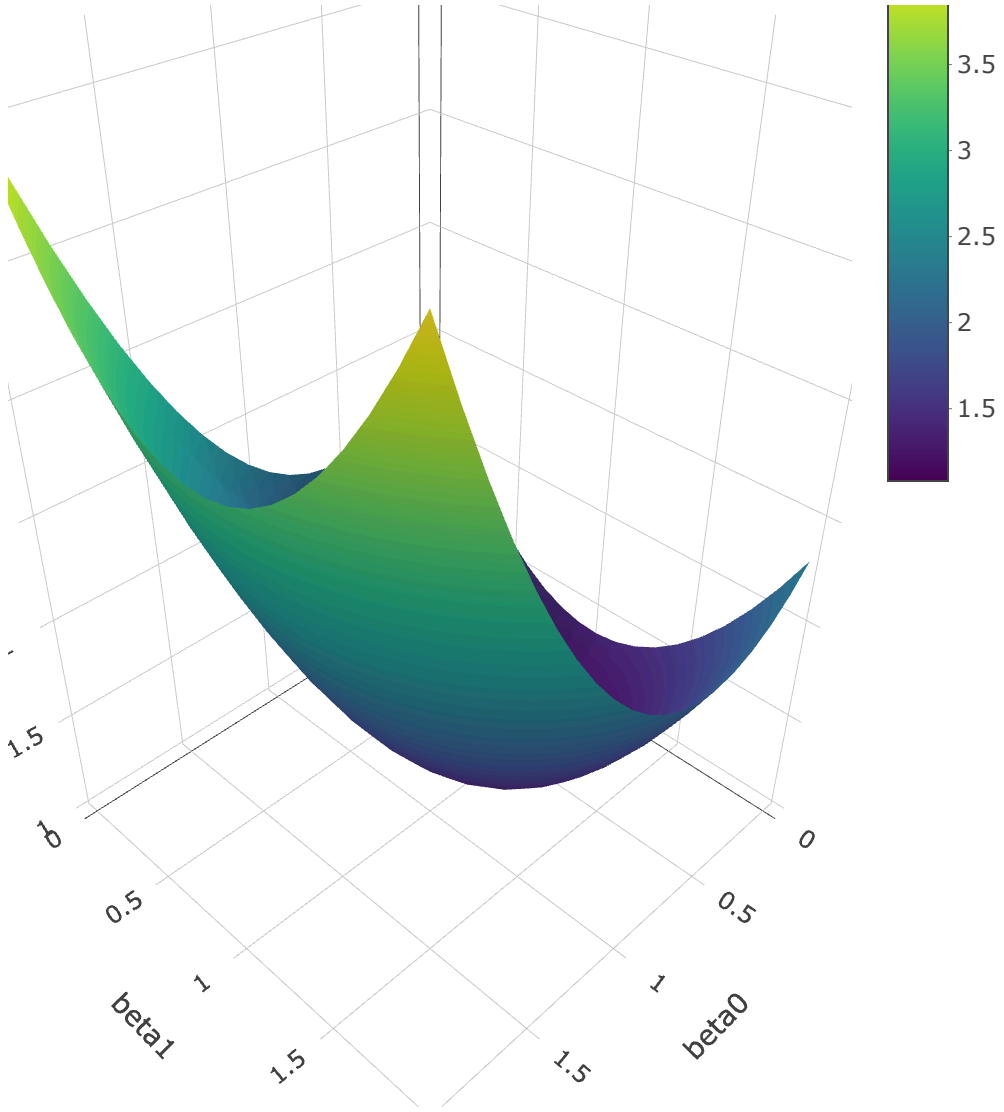
Doing this on all such $\boldsymbol{\beta}$ values would allow us to create a surface of the RSS, as a function of the parameters. The following R code uses the `plotly` package to create a 3D plot to more intuitively understand the problem.

Hide

```
  # create a grid of beta values and the corresponding RSS
  b0 <- b1 <- seq(0, 2, length = 20)
  z = matrix(apply(expand.grid(b0, b1), 1, rss, x, y), 20, 20)

  onepoint = data.frame("x" = 0.3, "y" = 1.5, "z" = rss(c(0.3, 1.5), x, y))

  # 3d plot of RSS using `plotly`
  library(plotly)
## Loading required package: ggplot2
##
## Attaching package: 'plotly'
## The following object is masked from 'package:ggplot2':
##
##     last_plot
## The following object is masked from 'package:stats':
##
##     filter
## The following object is masked from 'package:graphics':
##
##     layout
  plot_ly(x = b0, y = b1) %>%
      layout(plot_bgcolor='rgb(254, 247, 234)') %>%
      layout(paper_bgcolor='transparent') %>%
      add_surface(z = t(z),
                  colorscale = 'Viridis') %>%
      layout(scene = list(xaxis = list(title = "beta0"),
             yaxis = list(title = "beta1"),
             zaxis = list(title = "RSS"))) %>%
      add_markers(data = onepoint,
                x = ~x, y = ~y, z = ~z,
                marker = list(size = 6, color = "red", symbol = 104))
```



As we can see, the pre-specified point $(0.3, 1.5)$ is not at the bottom of this curve. For this optimization problem, our goal is to minimize the RSS. Hence, we would like to know what are the corresponding $\boldsymbol{\beta}$ values. Numerical optimization is a research field that investigates such problems and their properties.

The `optim()` function

As a starting point, let's introduce the `optim()` function, which can be used to solve such problems. By default, its solves a minimization problem.

Hide

```
# The solution can be solved by any optimization algorithm
lm.optim <- optim(par = c(2, 2), fn = rss, trainx = x, trainy = y)
```

- The `par` argument specifies an initial value. In this case, it is $\beta_0 = \beta_1 = 2$.
- The `fn` argument specifies the name of a function (`rss` in this case) that can calculate the objective function. This function may have multiple arguments. However, the first argument has to be the parameter(s) that is being optimized. In addition, the parameters need to be supplied to the function as a vector, but not matrix, list or other formats.

- The arguments `trainx = x`, `trainy = y` specifies any additional arguments that the objective function `fn` (`rss`) needs. It behaves the same as if you are supplying this to the function `rss` it self.

Hide

```
  lm.optim
## $par
## [1] 0.5669844 1.0169299
##
## $value
## [1] 1.076857
##
## $counts
## function gradient
##       63       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The result shows that the estimated parameters (`$par`) are 0.567 and 1.017, with a functional value 1.077. The convergence code is 0, meaning that the algorithm converged. The parameter estimates are almost the same as `lm()`.

Hide

```
  # The solution form lm()
  summary(lm(y ~ x))$coefficients
##             Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 0.566921 0.08531319  6.645175 5.429515e-10
## x           1.016770 0.08626936 11.785996 4.974715e-23
```

Basic Principles

We will first introduce several properties that would help us to determine if a point $\mathbf{x}$ attains the optimum. Then some examples will follow. These properties are usually applied to unconstrained optimization problems. They are essentially just describing the landscape around a point $\mathbf{x}^*$ such that it becomes the local optimizer. Before we proceed, here are some things to be aware of:

- All the following statements are **multidimensional** since we may have more than one parameter to optimize. Hence, partial derivatives ∇ are always vectors and second derivatives ∇^2 are matrices, although some examples could be one-dimensional.
- Instead of using $\boldsymbol{\beta}$ as the argument of the function, we will use $\mathbf{x}$, which is a p-dimensional vector. This may seem conflicting with our other lecture notes. However, this notation is consistent with the literature on this topic.
- To grasp the idea, we could assume that the functions are smooth, e.g., continuously differentiable. This would give us nice properties to help the analysis. Later on, we may see other examples that are not differentiable, such as the Lasso. But we will have other tools to deal with it.

- We will also assume that we are dealing with a minimization problem of $f(\mathbf{x})$, while for maximization of $f(\mathbf{x})$, we can always consider minimizing $-f(\mathbf{x})$. Hence they are essentially the same.

First-order Property

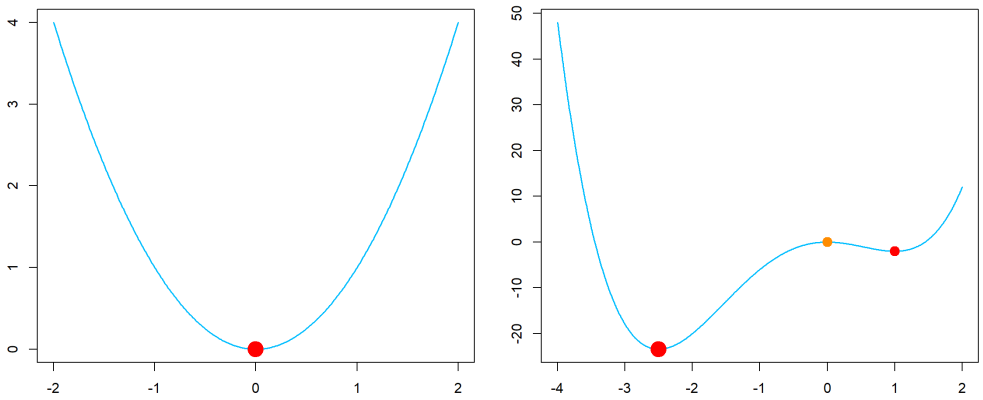
First-Order Necessary Condition:

> If f is continuously differentiable in an open neighborhood of local minimum $\mathbf{x}^*$, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

To see why a local optimum must satisfy this property, it might be intuitive to see the following plot, which is generated using functions

$$\begin{aligned} \text{Left:} \quad & f_1(x) = x^2 \\ \text{Right:} \quad & f_2(x) = x^4 + 2 \times x^3 - 5 \times x^2 \end{aligned}$$

It is easy to see that the minimizer of the left function is 0, while the minimizer of the right one is around -2.5, but there are also other points that are interesting.



On the left hand side, we have a **convex function**, which looks like a bowl. The intuition is that, if $f(\mathbf{x})$ is a function that is smooth enough, and $\mathbf{x}$ is a point with $\nabla f(\mathbf{x}^*) \neq 0$, then by the Taylor expansion, we have, for any new point $\mathbf{x}^{new}$ in the neighborhood of $\mathbf{x}$, we can approximate its function value

$$f(\mathbf{x}^{\text{new}}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})(\mathbf{x}^{\text{new}} - \mathbf{x})$$

Regardless of whether $\nabla f(\mathbf{x})$ is positive or negative, we can always find a new point $\mathbf{x}^{\text{new}}$ that makes $\nabla f(\mathbf{x})(\mathbf{x}^{\text{new}} - \mathbf{x})$ smaller than 0. Hence, this new point would have a smaller functional value than $\mathbf{x}$.

Let's apply this result to the two problems above. Note that this would not be possible with more complex functions. By taking derivatives we have

$$\begin{aligned} \text{Left:} \quad & \nabla f_1(x) = 2x \\ \text{Right:} \quad & \nabla f_2(x) = 4x^3 + 6x^2 - 10x = 2x(x-1)(2x+5) \end{aligned}$$

Hence, for the left one, the only point that makes the derivative 0 is $x = 0$, which is indeed what we see on the figure. For the right one, there are three x values that would make this 0: $x = 0, 1$ and -2.5. Are they all minimizers of this function? Of course not, however, for different reasons.

- $x = 0$ is a maximizer rather than a minimizer. Since we only checked if the slope if "flat" but didn't care if its facing upward or downward, our condition cannot tell the difference. That's why $\nabla f(\mathbf{x}^*) = \mathbf{0}$ is **only a necessary condition, but not a sufficient condition**.
- $x = 1$ is in fact a minimizer, although it is **not a global minimizer, but only a local minimizer**. In this course, we will see some examples of this such as the k-means clustering algorithm, which could lead to a local minimizer. Although we will not focus on this issue, there are two things to consider:
    - When the problem we are trying to solve is a **convex function** (also needs to be in a convex domain), a local minimum is guaranteed to be a global minimum. This is the case of $f_1(x)$.
    - For a non-convex problem, numerical approaches (instead of analytic solution) can start from different initial points (see our example of `optim()`) and this may lead to different solutions, some are local, some can be global. Hence, try different initial points, and compare their end results can give us a better chance of getting a global solution.

Practice question

Suppose $f(x) = \exp(-x) + x$, what is the minimizer of this function? Produce a figure to visualize the result.

Show

Second-order Property

Second-order Sufficient Condition:

> f is twice continuously differentiable in an open neighborhood of $\mathbf{x}^*$. If $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite, i.e.,
>
> $$\nabla^2 f(\mathbf{x}) = \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right) = \mathbf{H}(\mathbf{x}) \succeq 0$$
>
> then $\mathbf{x}^*$ is a strict local minimizer of f.

Here $\mathbf{H}(\mathbf{x})$ is called the **Hessian matrix**, which will be frequently used in second-order methods. We can easily check this property for our examples:

$$\begin{aligned} \text{Left:} \quad & \nabla^2 f_1(x) = 2 \\ \text{Right:} \quad & \nabla^2 f_2(x) = 12x^2 + 12x - 10 \end{aligned}$$

Hence for f_1, the Hessian is positive definite, and the solution is a minimizer. While for f_2, $\nabla^2 f_2(-2.5) = 35$, $\nabla^2 f_2(0) = -10$ and $\nabla^2 f_2(1) = 14$. This implies that -2.5 and 1 are local minimizers and 0 is a local maximizer. These conditions are sufficient, but again, they only discuss local properties, not global properties.

Algorithm

Most optimization algorithms follow the same idea: starting from a point $\mathbf{x}^{(0)}$ (which is usually specified by the user) and move to a new point $\mathbf{x}^{(1)}$ that improves the objective function value. Repeatedly performing this to get a sequence of points $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \dots$ until the certain stopping criterion is reached.

We have introduced two properties that provide a candidate of a new target point: second-order methods (Newton's method) and first-order methods (gradient descent). Repeatedly applying these strategies in an iterative algorithm would allow us to find a good **optimizer** after a good number of iterations. Of course, this iterative algorithm has to stop at certain point. Before showing examples, we mention that a **stopping criterion** could be

- Using the gradient of the objective function: $\|\nabla f(\mathbf{x}^{(k)})\| < \epsilon$
- Using the (relative) change of distance: $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|/\|\mathbf{x}^{(k-1)}\| < \epsilon$ or $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \epsilon$
- Using the (relative) change of functional value: $|f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k-1)})| < \epsilon$ or $|f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k-1)})|/|f(\mathbf{x}^{(k)})| < \epsilon$
- Stop at a pre-specified number of iterations.

Most algorithms differ in terms of how to move from the current point $\mathbf{x}^{(k)}$ to the next, better target point $\mathbf{x}^{(k+1)}$. This may depend on the smoothness or structure of f, constrains on the domain, computational complexity, memory limitation, and many others.

The `optim()` function, again

We can use the `optim()` function to help us. Let's first define the functions.

Hide

```
f1 <- function(x) x^2
f2 <- function(x) x^4 + 2*x^3 - 5*x^2
```

We will use a method called `BFGS`, which a very popular approach and be introduced later. For $f_1(x)$, the algorithm finds the minimizer in just one step, while for the second problem, its depends on the initial point.

Hide

```
  # First problem
  optim(par = 3, fn = f1, method = "BFGS")
## $par
## [1] -8.384004e-16
##
## $value
## [1] 1.75742e-29
##
## $counts
## function gradient
##        8        3
##
## $convergence
## [1] 0
##
## $message
## NULL

  # Second problem, ends with local minimizer 1
  optim(par = 10, fn = f2, method = "BFGS")
## $par
## [1] 0.9999996
##
## $value
## [1] -2
##
## $counts
## function gradient
##       37       12
##
## $convergence
## [1] 0
##
## $message
## NULL

  # Try a new starting value, ends with global minimizer -2.5
  optim(par = 3, fn = f2, method = "BFGS")
## $par
## [1] -2.5
##
## $value
## [1] -23.4375
##
## $counts
## function gradient
##       17        6
##
## $convergence
## [1] 0
##
```

```
## $message
## NULL
```

Practice question

Going back to the example $f(x) = \exp(-x) + x$. Write the corresponding objective function and use the `optim()` function to solve it.

Show