# Use optim Function in R (2 Examples)

Authored by

**Mohammed looti**

October 30, 2025

The **optim** function in **R** provides a robust tool for **general-purpose optimizations**. It is specifically designed to find the minimum or maximum of a given objective function, making it incredibly versatile for solving a wide array of statistical, mathematical, and machine learning problems. This powerful function allows users to define custom objective functions and search systematically for the optimal parameter values that minimize or maximize these functions, offering a flexible alternative to relying solely on specialized model fitting routines.

## Understanding the optim() Function Syntax and Parameters

The **optim** function in **R** utilizes a precise and powerful syntax to facilitate its optimization tasks. Grasping the purpose of each fundamental component is essential for defining custom models and achieving convergence to a reliable solution.

The fundamental syntax is:

**optim(par, fn, data, ...)**

Here's a detailed breakdown of each critical parameter and its role in the optimization process:

**par**: This argument requires a vector representing the initial values for the parameters that the function will attempt to optimize. Providing appropriate starting points is vital, as the initial guess can significantly influence the optimization process, affecting both convergence speed and the likelihood of locating a global optimum rather than a local one. For example, when optimizing a **linear regression model**, this vector would contain the initial guesses for the intercept and slope **coefficients**.
**fn**: This is the objective function that **optim** will iteratively evaluate and attempt to minimize. This function must accept the parameter vector (`par`) as its first argument and must return a single numeric value representing the "cost" or "error" we aim to reduce. For maximization problems, a simple workaround is to structure the function to return the negative of the value to be maximized, effectively transforming it into a minimization problem.
**data**: This optional, but highly useful, parameter allows you to pass additional data structures or variables required by your objective function `fn`. It is particularly helpful when `fn` needs external information, such as observed values contained within a **data frame**. The remaining `...` argument allows for passing further auxiliary arguments either to `fn` or directly to the specific optimization method being employed by **optim**.

The flexibility inherent in defining a custom objective function `fn` is the reason why **optim** remains a cornerstone tool for advanced statistical modeling and machine learning applications in **R**. It empowers users to implement complex loss functions or bespoke likelihood functions that may not be readily available in standard statistical packages.

## Case Study 1: Estimating Linear Regression Coefficients

One of the most straightforward and illustrative applications of the **optim** function is using it to estimate the **coefficients** for a **linear regression model**. This process involves defining a custom objective function that mathematically quantifies the discrepancy between the observed data and the model's predictions, and subsequently instructing **optim** to find the parameter values that minimize this calculated discrepancy.

## Setting Up the Objective Function for Linear Optimization

In standard linear regression, the universally accepted objective is to minimize the **residual sum of squares** (RSS). The RSS metric calculates the sum of the squared differences between the actual observed values (y) and the values predicted by the model (?). By minimizing this sum, we identify the line that represents the best possible fit for the given data set.

Consider a simple linear regression model structured as: y = β0 + β1 * x, where β0 is the intercept and β1 is the slope. Our primary goal is to determine the values of β0 and β1 that yield the lowest possible RSS. The following **R** code snippet demonstrates how to set up the necessary data and define the objective function for this minimization task:

```
#create data frame
df <- data.frame(x=c(1, 3, 3, 5, 6, 7, 9, 12),
y=c(4, 5, 8, 6, 9, 10, 13, 17))

#define function to minimize residual sum of squares
min_residuals <- function(data, par) {
with(data, sum((par + par * x - y)^2))
}

#find coefficients of linear regression model
optim(par=c(0, 1), fn=min_residuals, data=df)

$par
2.318592 1.162012

$value
11.15084

$counts
function gradient
79 NA
```

$convergence
0

$message
NULL

Within this implementation, `par` is assigned to represent the intercept (β0) and `par` represents the slope (β1). The `min_residuals` function efficiently calculates the sum of squared differences, which **optim** then iteratively minimizes by adjusting the values within the `par` vector until convergence is achieved.

## Interpreting the Results and Validation

The output generated by the **optim** function provides several critical metrics for assessing the optimization run. The most significant element for model estimation is the `$par` component, which holds the final, optimized **coefficients**. In the example above, `$par` returns the vector ` 2.318592 1.162012`.

These values correspond directly to the intercept and slope, respectively. Based on these optimized parameters, our derived **linear regression model** equation is established as:

y = 2.318 + 1.162x

The `$value` component indicates the minimum achievable value of the objective function--in this case, the minimized **residual sum of squares**. A lower value signifies a better fit of the model to the data. Crucially, the `$convergence` code provides assurance regarding the process: a value of `0` generally confirms that the optimization algorithm successfully converged to a solution.

### Validating Coefficients with lm()

To ensure the accuracy and reliability of the results obtained from **optim**, it is standard practice to validate them against the **lm()** function, which is R's highly efficient, built-in function for fitting linear models. The **lm()** function uses direct, analytical methods (like ordinary least squares) to calculate the optimal **coefficients** that mathematically minimize the **residual sum of squares**.

Executing the **lm()** function on our sample **data frame** yields the following output:

**#find coefficients of linear regression model using lm() function**
**lm(y ~ x, data=df)**

Call:
lm(formula = y ~ x, data = df)

Coefficients:
(Intercept) x
2.318 1.162

As confirmed by the output, the **coefficient** values retrieved from **lm()** (Intercept: 2.318, x: 1.162) are nearly identical to those derived using the **optim** function. This consistency validates our approach and proves that **optim** is perfectly capable of solving standard linear regression problems, even if specialized functions like **lm()** are usually preferred for their computational speed and specialized diagnostic features.

## Case Study 2: Applying optim() to Quadratic Regression Models

The true versatility of the **optim** function becomes apparent when dealing with models that extend beyond simple **linear regression**, such as **quadratic regression models**. A quadratic model introduces a squared term of the independent variable, enabling the model to effectively capture non-linear, curvilinear relationships between the predictors and the response variable. The generalized mathematical form of a quadratic regression model is $y = \beta0 + \beta1 * x + \beta2 * x2$.

## Defining the Quadratic Objective Function

Analogous to the linear case, the objective in **quadratic regression** remains finding the set of **coefficients** (β0, β1, β2) that minimize the **residual sum of squares**. Consequently, the objective function must be adapted to incorporate the squared term (x2) and must account for three distinct parameters. This modification demonstrates how easily the objective function can be tailored for a **quadratic model**.

The following **R** code illustrates this entire process, starting from the creation of a new **data frame** and culminating in the definition and execution of the three-parameter optimization using **optim**:

```
#create data frame
df <- data.frame(x=c(6, 9, 12, 14, 30, 35, 40, 47, 51, 55, 60),
y=c(14, 28, 50, 70, 89, 94, 90, 75, 59, 44, 27))

#define function to minimize residual sum of squares
min_residuals <- function(data, par) {
with(data, sum((par + par*x + par*x^2 - y)^2))
}

#find coefficients of quadratic regression model
optim(par=c(0, 0, 0), fn=min_residuals, data=df)
```

$par
-18.261320 6.744531 -0.101201

$value
309.3412

$counts
function gradient
218 NA

$convergence
0

$message
NULL

In this adapted function, `par` functions as the intercept, `par` is the **coefficient** for the linear term `x`, and `par` is the **coefficient** for the squared term `x^2`. The initial guesses for the `par` vector are conservatively set to `c(0, 0, 0)`, which is a standard procedure when there is no specific prior knowledge about the expected values of the **coefficients**.

## Analyzing Output and Cross-Verification

Upon successful execution, the **optim** function provides the optimized parameters in the `$par` element. For this specific **quadratic regression model**, the output reveals the vector ` -18.261320 6.744531 -0.101201`.

These values correspond, in order, to β0, β1, and β2. Consequently, the fitted **quadratic regression model** can be formally expressed as:

y = -18.261 + 6.744x - 0.101x2

The resulting `$value` of `309.3412` represents the minimized **residual sum of squares** achieved by this **quadratic model**, and the `convergence` code of `0` confirms that the numerical optimization algorithm successfully located a minimum.

### Cross-Verification using R's lm() Function

To cross-validate the **optim** results for quadratic regression, we again turn to the powerful **lm()** function. To utilize **lm()** for a quadratic fit, we must explicitly create a squared term (e.g., `x2`) within our **data frame**, which **lm()** will then treat as a separate, independent predictor.

The **R** code required for this verification step is:

```
#create data frame
df <- data.frame(x=c(6, 9, 12, 14, 30, 35, 40, 47, 51, 55, 60),
y=c(14, 28, 50, 70, 89, 94, 90, 75, 59, 44, 27))

#create a new variable for x^2
df$x2 <- df$x^2

#fit quadratic regression model
quadraticModel <- lm(y ~ x + x2, data=df)

#display coefficients of quadratic regression model
summary(quadraticModel)$coef
```

Estimate Std. Error t value Pr(>|t|)
(Intercept) -18.2536400 6.185069026 -2.951243 1.839072e-02
x 6.7443581 0.485515334 13.891133 6.978849e-07
x2 -0.1011996 0.007460089 -13.565470 8.378822e-07

The **coefficients** obtained from the **lm()** output (Intercept: -18.25364, x: 6.7443581, x2: -0.1011996) show remarkable alignment with those calculated by the **optim** function. Minor numerical differences are expected due to the distinct nature of the optimization algorithms used (iterative search in **optim** versus direct matrix solution in **lm()**). Nonetheless, this close match confirms that **optim** is a highly reliable tool for estimating **coefficients** in both linear and **quadratic regression models**, especially when the modeling scenario necessitates the implementation of a custom objective function.

## Conclusion: The Power and Flexibility of optim()

The **optim** function in **R** stands as an exceptionally powerful and flexible utility for general-purpose numerical optimization. While specialized functions like **lm()** are often the more computationally efficient choice for standard **linear** and **quadratic regression models**, **optim**'s core strength lies in its ability to handle completely custom objective functions and tackle complex optimization problems that lack a direct analytical solution or a readily available dedicated function.

This article showcased its practical application by successfully estimating **coefficients** for both linear and **quadratic regression models** through the minimization of the **residual sum of squares**. The consistent results demonstrated when cross-validated with **lm()** firmly underscore **optim**'s reliability. For researchers, data scientists, and engineers working with unique model specifications, non-standard loss functions, or scenarios where direct model fitting functions are

insufficient, **optim** provides an indispensable and highly adaptable approach to finding optimal solutions.

## Further Learning and Resources

To deepen your expertise in optimization techniques and their sophisticated application in **R**, we recommend exploring additional resources. Mastering these concepts will enable you to confidently tackle more intricate problems and optimize various statistical and machine learning models.

The following tutorials explain how to perform other common operations in **R**: