

How to Easily Plot a Function Curve in R

Authored by
stats writer

November 20, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Plot a Function Curve in R*. PSYCHOLOGICAL SCALES.
Retrieved from <https://scales.arabpsychology.com/?p=98458>

Plotting a function curve in R is a fundamental and relatively straightforward task essential for mathematical modeling and data visualization. To begin, users must first define the algebraic expression or relationship they wish to plot. Unlike plotting discrete data points using the standard `plot()` function, plotting a continuous curve requires specifying the function itself and the range of x-values (the domain) to be sampled. By utilizing specific functions designed for curve generation, such as `curve()` in Base R or `stat_function()` in ggplot2, R handles the underlying calculations and interpolation necessary to render a smooth visual representation.

After defining the function and range, optional arguments allow for extensive customization of the visual output. These parameters control essential aesthetic features such as line color, line thickness (width), and line style (solid, dashed, dotted). Understanding how to efficiently define, plot, and customize these curves is critical for producing clear, publication-ready graphics. This guide explores the two principal methodologies available within the R environment for plotting arbitrary functions.

Understanding Function Plotting in R

The ability to visualize mathematical functions is crucial for interpreting complex relationships in fields ranging from engineering to economics. In R, plotting a function curve means generating a high-resolution representation of the equation over a specified interval. When we execute a plotting command for a function, Base R or the chosen package automatically calculates hundreds or thousands of (x, y) coordinates within the provided domain and connects these points to form a smooth line.

This automated sampling process distinguishes function plotting from standard scatter plotting, where the user supplies every data point explicitly. For complex or highly nonlinear functions, using dedicated functions like `curve()` ensures adequate sampling density is achieved to avoid jagged or misleading visualizations. Furthermore, both primary methods in Base R and ggplot2 offer simplified syntax specifically tailored for functional equations, streamlining the coding effort required.

Overview of Primary Plotting Methods

When visualizing function curves in R, practitioners generally rely on two highly effective systems. The first is the native graphical system, often referred to as Base R graphics, which utilizes built-in functions like `curve()`. This approach is excellent for quick visualizations, simple scripts, and environments where minimal package dependency is desired. It is fast, efficient, and requires no external installation.

The second dominant method involves the use of the ggplot2 package, part of the Tidyverse

ecosystem. `ggplot2` is founded on the Grammar of Graphics, offering a layered approach to visualization that provides unparalleled flexibility and aesthetic quality. While it requires loading an external library, its powerful syntax and ability to integrate function plots seamlessly with complex datasets make it the preferred choice for sophisticated analyses and publication-quality graphics. Both methods are capable of producing the desired visualization, but they differ significantly in their syntax and customization potential.

You can use the following methods to plot a function curve in R:

Method 1: Use Base R

```
curve(x^3, from=1, to=50, xlab='x', ylab='y')
```

Method 2: Use ggplot2

```
library(ggplot2)
```

```
df <- data.frame(x=c(1, 100))
```

```
eq = function(x){x^3}
```

```
#plot curve in ggplot2
```

```
ggplot(data=df, aes(x=x)) +
```

```
stat_function(fun=eq)
```

Both methods will produce a plot that shows the curve of the function $y = x^3$.

Implementing Function Curves with Base R

The simplest and most direct way to plot a function in Base R is through the `curve()` function. This utility is specifically designed to calculate and plot mathematical expressions over a defined range. It takes the algebraic expression directly as its first argument, requiring only the variable `x` to be used within the expression. This simplicity makes `curve()` ideal for immediate visualization and quick checks of mathematical models.

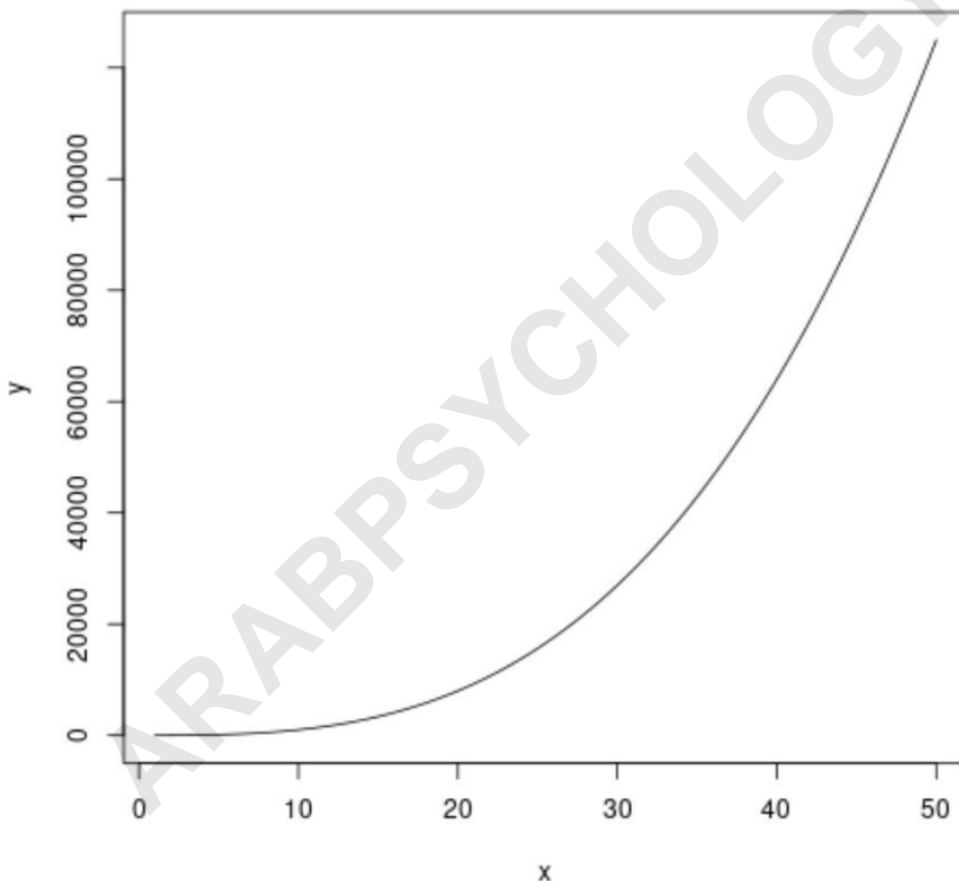
The core power of `curve()` lies in its required arguments: the expression itself, and the `from` and `to` parameters, which establish the minimum and maximum x-values, respectively. By setting `from` and `to`, the user controls the domain of the plot. Additionally, standard graphical parameters familiar to Base R users, such as `xlab` and `ylab` for axis labels, can be passed directly to the function call, ensuring immediate clarity in the resulting graphic. It is a highly efficient tool for single-function plotting.

Detailed Example 1: Base R `curve()` Syntax

This example demonstrates how to plot the function $y = x^3$ using the `curve()` function in Base R. We define the range from $x=1$ to $x=50$, providing a clear visualization of the cubic growth over this interval. This code generates the foundational plot before any aesthetic modifications are applied, showing the default line style and color chosen by the R environment.

The following code shows how to plot the curve of the function $y = x^3$ using the `curve()` function from Base R:

```
#plot curve using x-axis range of 1 to 50  
curve(x^3, from=1, to=50, xlab='x', ylab='y')
```



Enhancing Visuals in Base R

While the basic `curve()` plot provides the necessary mathematical information, customizing its appearance is essential for professional reporting and enhanced readability. The Base R graphics system allows for modification using standard graphical parameters, which are passed directly as

arguments within the `curve()` function call. These parameters control the physical appearance of the drawn line, significantly impacting the visual communication of the data.

The primary aesthetic parameters used for customizing lines include `lwd` (line width), which adjusts the thickness of the line, `col` (color), which sets the line color using names or hexadecimal codes, and `lty` (line type), which defines whether the line is solid, dashed, or dotted. Experimenting with these values allows users to create distinctive visualizations that comply with specific style guidelines or simply improve contrast and visibility on the final output medium.

Note that you can use the following arguments to modify the appearance of the curve:

lwd: Line width, typically an integer value.

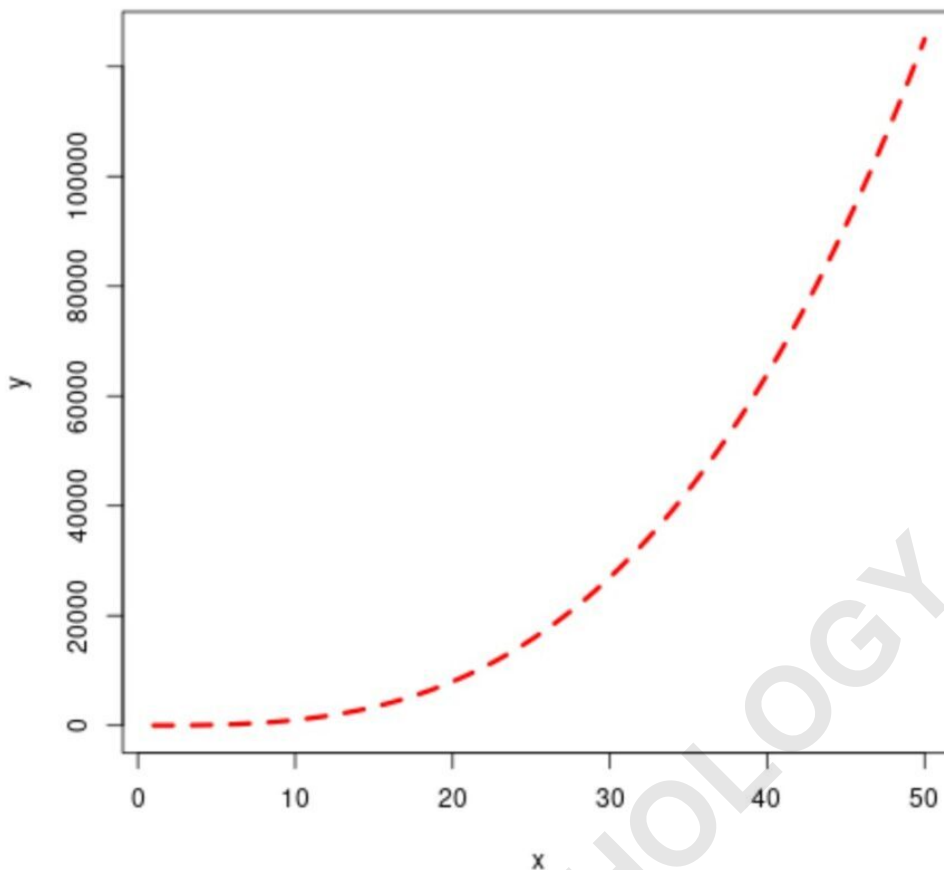
col: Line color, accepting R color names or HEX codes.

lty: Line style, using numeric codes (e.g., 1=solid, 2=dashed) or character strings.

The following code shows how to use these arguments in practice, setting the line to be thick (`lwd=3`), red, and dashed:

#plot curve using x-axis range of 1 to 50

`curve(x^3, from=1, to=50, xlab='x', ylab='y', lwd=3, col='red', lty='dashed')`



Feel free to play around with the values for these arguments to create the exact curve you'd like.

Leveraging the Power of the ggplot2 Package

For users who prefer the structured and highly aesthetic output of the Grammar of Graphics, the `ggplot2` package offers the `stat_function()` geometry. Unlike the `Base R` `curve()` function, which creates a plot from scratch, `stat_function()` is a layer that must be added to an existing `ggplot()` object. This integration allows function curves to be easily combined with other data representations, such as scatter plots or histograms, within the same coordinate system.

To use `stat_function()`, the function itself must first be defined as a standard R function object (e.g., `eq = function(x){x^3}`). This defined function is then passed to the `fun` argument within `stat_function()`. The surrounding `ggplot()` call requires a minimal data frame and an aesthetic mapping (`aes()`), typically just defining the x-axis range, although the data frame contents are often trivial since the curve is generated mathematically rather than from the data itself. This methodology ensures consistency and adherence to the structured framework of `ggplot2`.

Detailed Example 2: Utilizing `stat_function()` in `ggplot2`

This example illustrates the necessary setup and execution for plotting the curve $y = x^3$ using the `ggplot2` framework. We begin by loading the library, defining a minimal `data frame` (`df`) to establish the plotting domain, and creating the function object (`eq`). Finally, the `ggplot()` call initializes the plot, and `stat_function()` adds the calculated curve layer.

The following code shows how to plot the curve of the function $y = x^3$ using the `stat_function()` function from `ggplot2`:

`library(ggplot2)`

```
#define data frame
```

```
df <- data.frame(x=c(1, 100))
```

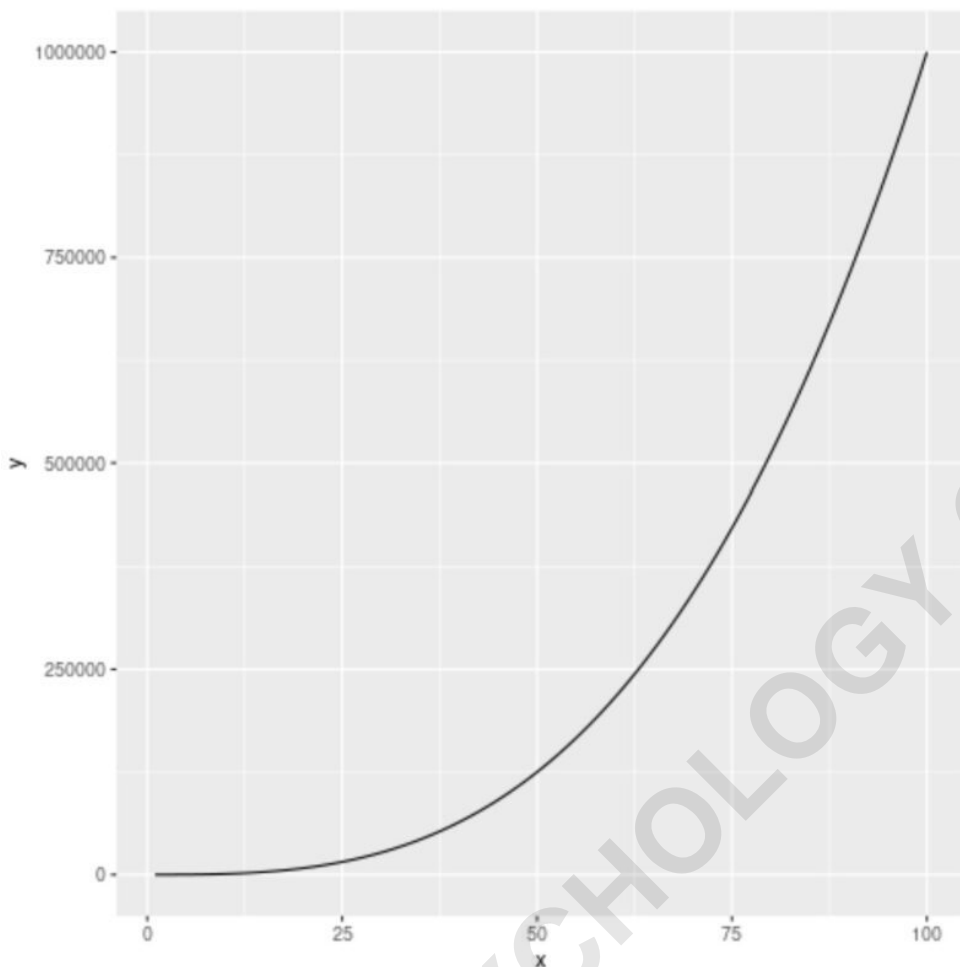
```
#define function
```

```
eq = function(x){x^3}
```

```
#plot curve in ggplot2
```

```
ggplot(data=df, aes(x=x)) +
```

```
  stat_function(fun=eq)
```



Customizing Aesthetics in ggplot2

A key advantage of using `ggplot2` is the ease with which plot aesthetics can be managed. Within the `stat_function()` layer, users can define line appearance parameters directly, similar to Base R, using attributes such as `lwd`, `col`, and `lty`. However, in `ggplot2`, these are typically set outside of the `aes()` mapping if the goal is to apply a static appearance to the entire line.

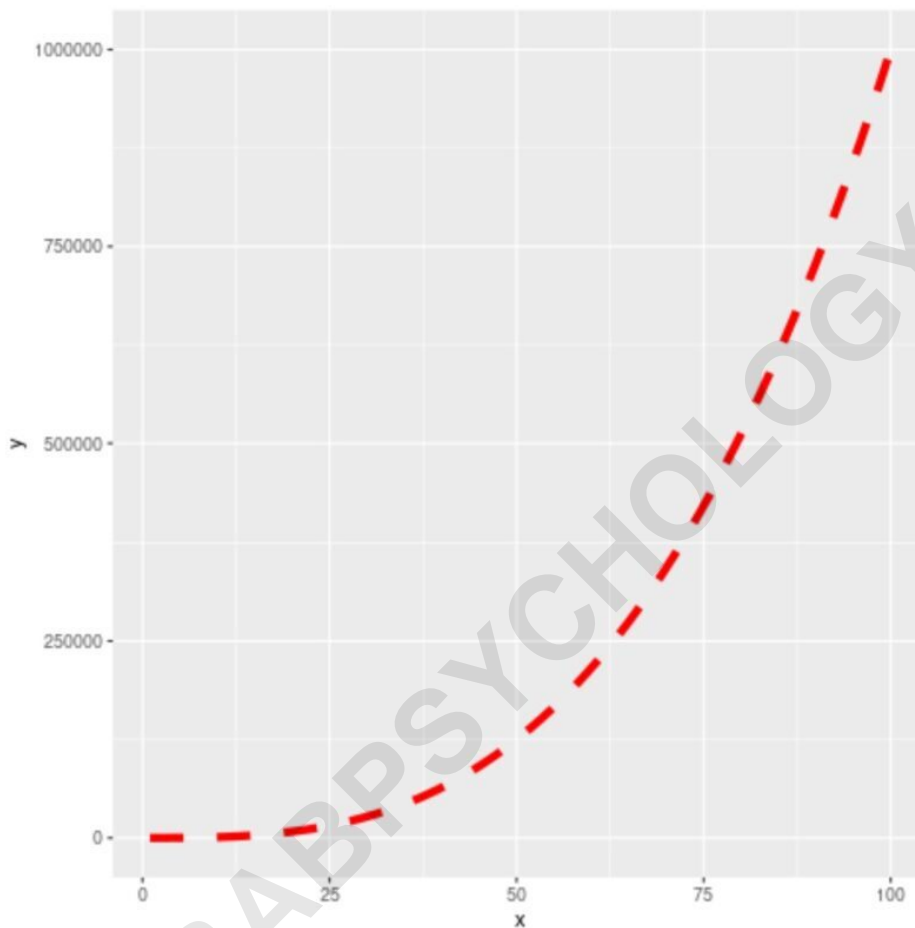
By defining these attributes--`lwd` for line thickness, `col` for color, and `lty` for line type--within the `stat_function()` call, we override the default `ggplot2` styling. This level of control ensures the resulting curve aligns perfectly with the desired graphical standards. For instance, setting the line to be red, dashed, and thicker enhances its visibility and distinction, particularly when multiple layers or functions are plotted simultaneously.

`library(ggplot2)`

```
#define data frame
df <- data.frame(x=c(1, 100))
```

```
#define function
eq = function(x){x^3}

#plot curve in ggplot2 with custom appearance
ggplot(data=df, aes(x=x)) +
  stat_function(fun=eq, lwd=2, col='red', lty='dashed')
```



Note: You can find the complete documentation for the ggplot2 `stat_function()` function at its official source, providing detailed information on all available arguments and customization options.