

ggfunction: A Grammar of Graphics for Mathematical Functions and Probability Distributions

by Dusty Turner, James Otto, Rodney X. Sturdivant, and David Kahle

Abstract The **ggfunction** R package extends **ggplot2** by providing a principled, unified interface for visualizing mathematical functions and probability distributions. It is organized around two complementary families. The first is a dimensional taxonomy classifying functions by their input and output dimensions: scalar functions $f: \mathbb{R} \rightarrow \mathbb{R}$, parametric curves $\gamma: \mathbb{R} \rightarrow \mathbb{R}^2$, scalar fields $f: \mathbb{R}^2 \rightarrow \mathbb{R}$, and vector fields $\mathbf{F}: \mathbb{R}^2 \rightarrow \mathbb{R}^2$. The second family provides specialized geoms for probability distributions—density, cumulative distribution, mass, quantile, discrete cumulative distribution, discrete quantile, discrete survival, and hazard functions—each with built-in support for region shading central to hypothesis testing and interval estimation. By embedding function evaluation, numerical integration, and validation directly into the **ggplot2** layer system, **ggfunction** lets users move from a mathematical definition to a publication-quality visualization in a single, composable call.

1 Introduction

Visualizing mathematical functions is one of the most common tasks in applied mathematics and statistics—and one of the most under-supported in the grammar of graphics. Whether an instructor is shading a rejection region under a normal curve, a researcher is inspecting a scalar field over a spatial domain, or a student is tracing a parametric spiral, the need to move quickly from a function’s definition to its graphical representation arises constantly. In R, **ggplot2** (Wickham, 2010, 2016) has become the dominant visualization framework, yet its native support for plotting functions remains narrow. The built-in `stat_function()` evaluates a univariate function over a range and draws it as a path—perfectly adequate for $f: \mathbb{R} \rightarrow \mathbb{R}$, but offering nothing for parametric curves, scalar fields, vector fields, or the rich family of functions associated with probability distributions.

Several packages address pieces of this gap. **ggdist** (Kay, 2024) provides sophisticated distribution visualizations oriented toward Bayesian uncertainty communication. **mosaic** (Pruim et al., 2017) offers `plotDist()` for pedagogical work, but it is built on the **lattice** (Sarkar, 2008) framework rather than the grammar of graphics. **metR** provides contour and vector-field displays for meteorological data, though it requires precomputed grids rather than function objects. None of these packages offers a unified treatment of functions organized by their dimensional structure.

ggfunction fills this gap by extending **ggplot2** with a principled taxonomy of function types, classified by the dimensions of their domain and codomain. It provides two complementary families of geoms:

1. A **dimensional taxonomy** covering four function types: $\mathbb{R} \rightarrow \mathbb{R}$, $\mathbb{R} \rightarrow \mathbb{R}^2$, $\mathbb{R}^2 \rightarrow \mathbb{R}$, and $\mathbb{R}^2 \rightarrow \mathbb{R}^2$.
2. A **probability distribution family** providing specialized layers for density, cumulative distribution, mass, quantile, discrete cumulative distribution, discrete quantile, discrete survival, survival, and hazard functions.

Each layer follows the **ggplot2** extension pattern: a `Stat` object evaluates the user-supplied function on a grid or sequence, and a corresponding `Geom` renders the result. Because every **ggfunction** layer is a standard **ggplot2** layer, it composes freely with the full ecosystem of scales, coordinates, themes, and facets.

The remainder of this article is organized as follows. We describe the design principles underlying the package in Section 2. Sections 3 and 4 present the dimensional taxonomy and probability distribution family, respectively, illustrating each with examples. Section 5 addresses implementation details, Section 6 demonstrates composability with the broader grammar, Section 7 surveys related packages, and Section 8 concludes.

2 Design principles

2.1 A taxonomy grounded in dimension

The central organizing principle of **ggfunction** is that any function visualizable in two-dimensional space belongs to one of four classes, determined by the dimensions of its domain m and codomain n :

$$\begin{aligned} f: \mathbb{R} &\rightarrow \mathbb{R} & (m = 1, n = 1) \\ \gamma: \mathbb{R} &\rightarrow \mathbb{R}^2 & (m = 1, n = 2) \\ f: \mathbb{R}^2 &\rightarrow \mathbb{R} & (m = 2, n = 1) \\ \mathbf{F}: \mathbb{R}^2 &\rightarrow \mathbb{R}^2 & (m = 2, n = 2) \end{aligned} \tag{1}$$

Each class admits a natural visual encoding:

- **Scalar functions** ($\mathbb{R} \rightarrow \mathbb{R}$): curves in the Cartesian plane, optionally with region shading between the curve and the x -axis.
- **Parametric curves** ($\mathbb{R} \rightarrow \mathbb{R}^2$): directed paths with color encoding the time parameter.
- **Scalar fields** ($\mathbb{R}^2 \rightarrow \mathbb{R}$): raster heatmaps, contour lines colored by level value, or filled contour regions.
- **Vector fields** ($\mathbb{R}^2 \rightarrow \mathbb{R}^2$): short arrows at grid points (default) or streamlines computed by numerical integration of the field.

The naming convention—`geom_function_1d_1d()`, `geom_function_1d_2d()`, `geom_function_2d_1d()`, and `geom_function_2d_2d()`—encodes the dimensional signature directly, making the function's type explicit at the point of use.

2.2 Integration with the grammar of graphics

The grammar of graphics (Wilkinson, 2005) decomposes a graphic into orthogonal components: data, aesthetic mappings, geometric objects, statistical transformations, scales, coordinate systems, and facets. **ggplot2** implements this decomposition through the `ggproto` object system, which allows new `Stat` and `Geom` classes to be defined and composed with existing infrastructure.

Every **ggfunction** layer follows this pattern. The user supplies a function object via the `fun` parameter and domain bounds via `xlim` and, where appropriate, `ylim`. The `Stat` evaluates the function on a suitable grid and returns a data frame whose columns correspond to the aesthetic variables expected by the `Geom`. Because the result is a standard **ggplot2** layer, it composes with `+` alongside any other layer, scale, or theme.

A key design choice is the use of `rlang::inject()` to splice additional function arguments supplied through the `args` parameter. This allows users to parameterize functions without closures or anonymous wrappers:

```
# instead of wrapping in an anonymous function:
ggplot() + geom_pdf(fun = function(x) dnorm(x, mean = 5, sd = 2), xlim = c(0, 10))

# users can write:
ggplot() + geom_pdf(fun = dnorm, xlim = c(0, 10), args = list(mean = 5, sd = 2))
```

This pattern is consistent across all **ggfunction** layers and mirrors the `args` parameter already familiar from `ggplot2::stat_function()`.

3 The dimensional taxonomy

3.1 Scalar functions: $\mathbb{R} \rightarrow \mathbb{R}$

The simplest case is a univariate function rendered as a curve in the plane. `geom_function_1d_1d()` generalizes `ggplot2::stat_function()` by adding support for region shading. The function is evaluated at n equally-spaced points (default $n = 101$) over the specified `xlim`, and the resulting (x, y) pairs are drawn as a path.

```
library("ggfunction")
```

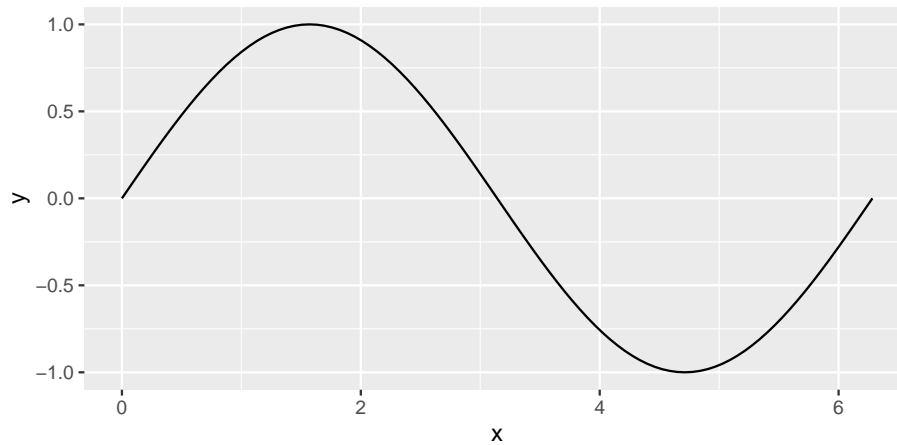


Figure 1: The sine function over one full period.

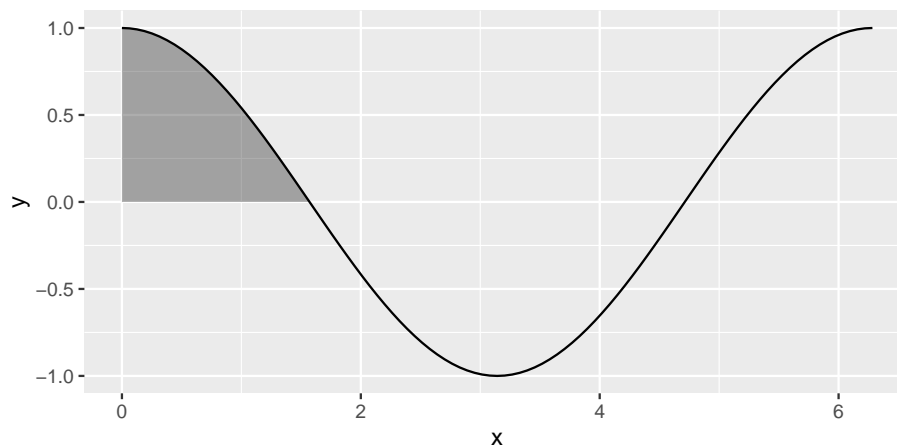


Figure 2: The cosine function over $[0, 2\pi]$ with the interval $[0, \pi/2]$ shaded; the shaded area equals $\int_0^{\pi/2} \cos(x) dx = 1$.

```
ggplot() +
  geom_function_1d_1d(fun = sin, xlim = c(0, 2 * pi))
```

The `shade_from` and `shade_to` parameters fill the region between the curve and the x -axis over a specified interval $[a, b]$. Boundary values are computed by linear interpolation via `stats::approxfun()`, so the shaded region aligns precisely with the requested interval even when a or b fall between evaluation points.

```
ggplot() +
  geom_function_1d_1d(
    fun = cos, xlim = c(0, 2 * pi),
    shade_from = 0, shade_to = pi / 2
  )
```

3.2 Parametric curves: $\mathbb{R} \rightarrow \mathbb{R}^2$

A parametric curve $\gamma(t) = (x(t), y(t))$ maps a scalar parameter—typically time—to a trajectory in the plane. `geom_function_1d_2d()` evaluates the user-supplied function over the range specified by `tlim` with step size `dt`, producing columns `t`, `x`, and `y`. By default, color is mapped to `after_stat(t)` to encode progression along the curve.

```
lemniscate <- function(t) c(cos(t) / (1 + sin(t)^2), sin(t) * cos(t) / (1 + sin(t)^2))
```

```
ggplot() +
  geom_function_1d_2d(fun = lemniscate, tlim = c(0, 1.9 * pi), tail_point = TRUE)
```

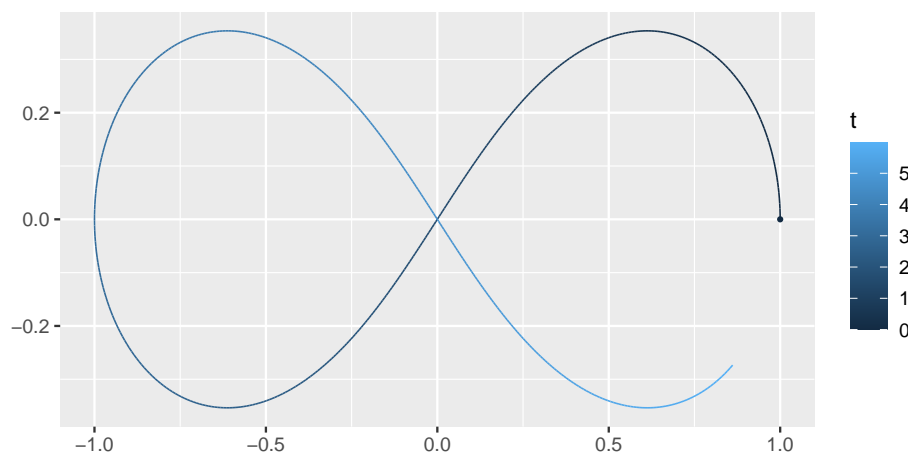


Figure 3: The lemniscate of Bernoulli $\gamma(t) = (\cos t / (1 + \sin^2 t), \sin t \cos t / (1 + \sin^2 t))$ —a figure-eight curve along which the product of distances to the two foci is constant—with color encoding the parameter t .

The `tail_point` parameter adds a marker at the starting position, useful for indicating an initial condition. The `args` parameter supports parameterized curve families. Lissajous figures $\gamma(t) = (\sin(at + \delta), \sin(bt))$ are a natural example: the frequency ratio a/b governs the curve's topology, while a single function definition handles all cases via `args`:

```
lissajous <- function(t, a = 3, b = 2, delta = pi/2) {
  c(sin(a * t + delta), sin(b * t))
}

p1 <- ggplot() +
  geom_function_1d_2d(
    fun = lissajous, tlim = c(0, 1.9 * pi),
    args = list(a = 1, b = 1)
  ) + ggtitle("a = 1, b = 1")

p2 <- ggplot() +
  geom_function_1d_2d(
    fun = lissajous, tlim = c(0, 1.9 * pi),
    args = list(a = 2, b = 1)
  ) + ggtitle("a = 2, b = 1")

p3 <- ggplot() +
  geom_function_1d_2d(
    fun = lissajous, tlim = c(0, 1.9 * pi),
    args = list(a = 3, b = 2)
  ) + ggtitle("a = 3, b = 2")

(p1 | p2 | p3) + plot_layout(guides = "collect")
```

3.3 Scalar fields: $\mathbb{R}^2 \rightarrow \mathbb{R}$

A scalar field assigns a real value to each point in a two-dimensional domain. `geom_function_2d_1d()` evaluates the user-supplied function on a regular $n \times n$ grid (default $n = 50$) over $xlim \times ylim$. The function must accept a numeric vector of length two and return a scalar; internally, a helper applies it row-wise over the grid. Three visualization modes are available through the `type` argument:

- "raster" (default): a heatmap with fill mapped to `after_stat(z)`.
- "contour": iso-level curves with colour mapped to `after_stat(level)`, producing a colored legend automatically.
- "contour_filled": filled regions between contour levels rendered by `ggplot2::GeomContourFilled`.

```
f_sc <- function(u) {
```

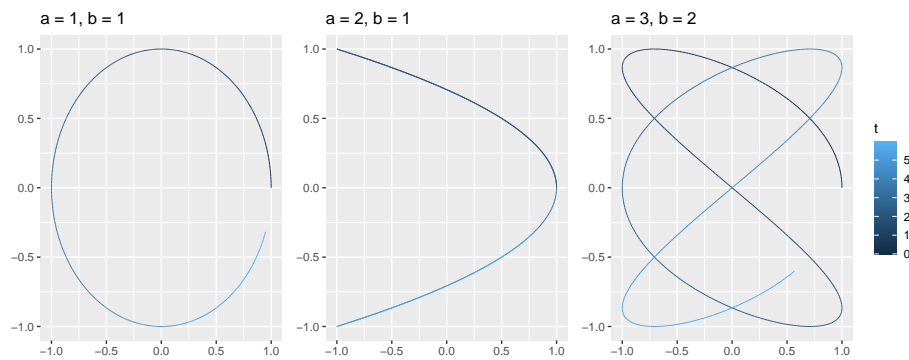


Figure 4: Lissajous figures $\gamma(t) = (\sin(at + \pi/2), \sin(bt))$ for three frequency ratios a/b . Each ratio produces a qualitatively distinct closed curve; the same function definition is reused across all three panels via the `args` parameter.

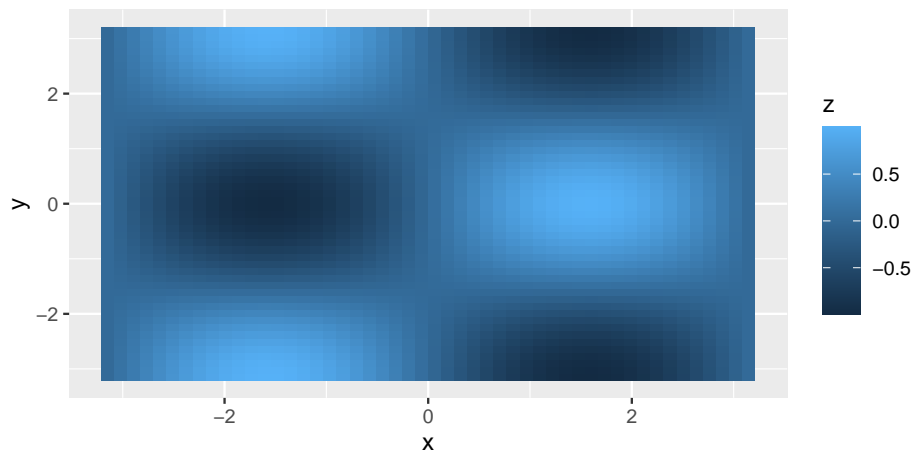


Figure 5: The function $f(x, y) = \sin(x) \cos(y)$ over $[-\pi, \pi]^2$ rendered as a raster heatmap.

```
x <- u[1]; y <- u[2]
sin(x) * cos(y)
}

ggplot() +
  geom_function_2d_1d(fun = f_sc, xlim = c(-pi, pi), ylim = c(-pi, pi))
```

The contour modes are selected simply by changing type:

```
p_contour <- ggplot() +
  geom_function_2d_1d(
    fun = f_sc, xlim = c(-pi, pi), ylim = c(-pi, pi), type = "contour"
  ) + ggtitle("type = \"contour\"")

p_filled <- ggplot() +
  geom_function_2d_1d(
    fun = f_sc, xlim = c(-pi, pi), ylim = c(-pi, pi), type = "contour_filled"
  ) + ggtitle("type = \"contour_filled\"")

p_contour | p_filled
```

For the contour variants, the `StatFunction2dContour` and `StatFunction2dContourFilled` classes extend the corresponding `ggplot2` stats. The `setup_params()` method pre-computes the function grid so that `z.range` is available for automatic break computation before any data is rendered.

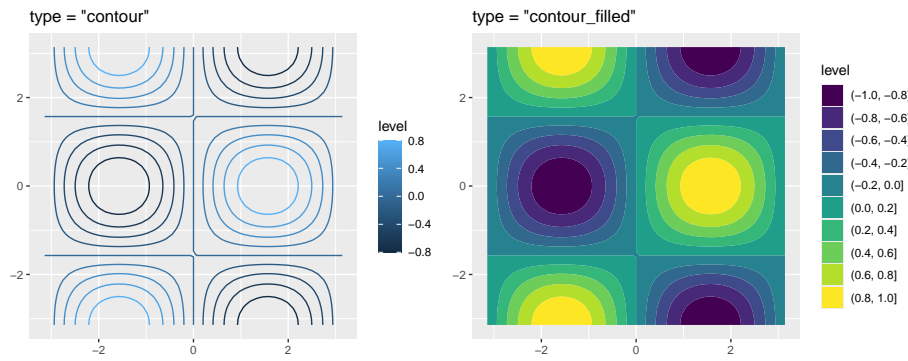


Figure 6: The same function rendered as contour lines (left) and filled contours (right).

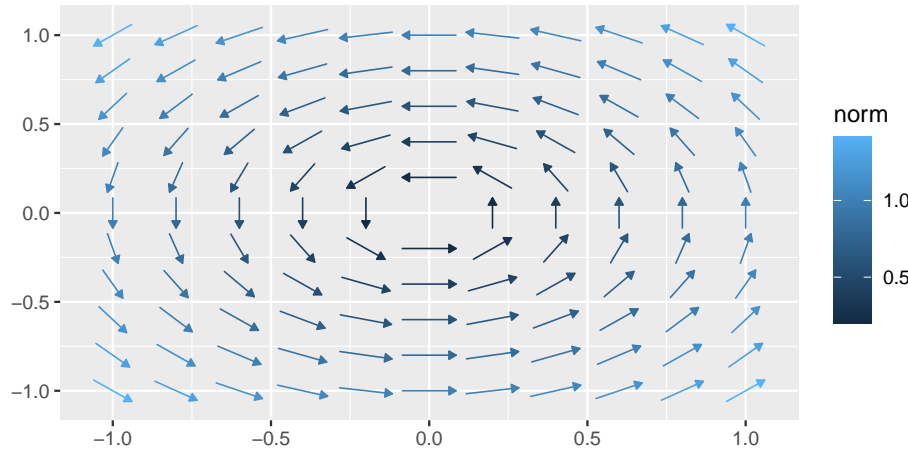


Figure 7: The rotation field $F(x, y) = (-y, x)$ displayed as short arrows at each grid point.

3.4 Vector fields: $\mathbb{R}^2 \rightarrow \mathbb{R}^2$

A vector field $F(x, y) = (F_1(x, y), F_2(x, y))$ assigns a direction and magnitude to each point in the plane. `geom_function_2d_2d()` delegates to **ggvfields** (Turner and Kahle, 2026) and supports two display modes through the `type` argument.

Vector arrows (default). With `type = "vector"` (the default), short arrows are drawn at each grid point, oriented according to the field direction and scaled relative to the grid spacing.

```
f_rotation <- function(u) {
  x <- u[1]; y <- u[2]
  c(-y, x)
}

ggplot() +
  geom_function_2d_2d(fun = f_rotation, xlim = c(-1, 1), ylim = c(-1, 1))
```

Streamlines. With `type = "stream"`, integral curves are computed by numerical integration of the field using a fourth-order Runge–Kutta method (Turner and Kahle, 2026) and rendered as directed paths seeded on a regular grid.

```
ggplot() +
  geom_function_2d_2d(fun = f_rotation, xlim = c(-1, 1), ylim = c(-1, 1),
    type = "stream")
```

Users whose primary interest is vector fields will find that **ggvfields** provides a richer collection of tools, including gradient fields, stream plots, and potential functions.

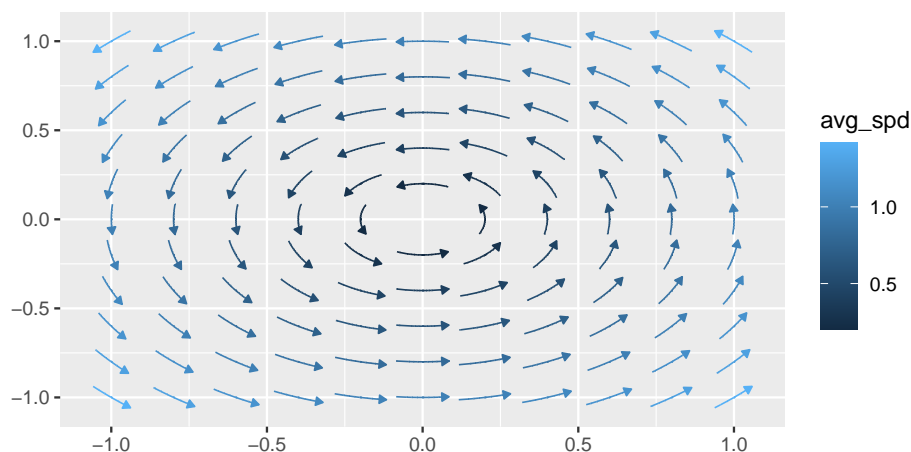


Figure 8: The same rotation field rendered as streamlines, each following the counterclockwise flow induced by the field.

4 The probability distribution family

Statistics education depends heavily on the visual language of probability distributions. Instructors shade tail areas to illustrate p -values, draw CDFs to connect probabilities with quantiles, and mark central regions to delineate confidence intervals. **ggfunction** provides a family of nine geoms that map directly onto the standard functions associated with a probability distribution, each with built-in shading support. Throughout, the user supplies an R function (e.g., `dnorm`, `pnorm`, `qnorm`) and a range; **ggfunction** handles evaluation, shading, and validation automatically.

4.1 Density functions: `geom_pdf()`

The probability density function (PDF) of a continuous random variable X satisfies

$$f(x) \geq 0 \quad \text{and} \quad \int_{-\infty}^{\infty} f(x) dx = 1. \quad (2)$$

`geom_pdf()` evaluates a user-supplied density and renders it as a filled area with an overlaid outline. The underlying `StatPDF` validates the normalization property (2) using `stats::integrate()`, issuing a diagnostic via `cli` (Csárdi, 2024) if the integral departs from unity by more than a specified tolerance—a guard against accidentally passing an unnormalized function.

Four shading modes are supported, corresponding to common pedagogical operations.

Single threshold. The `p` parameter specifies a cumulative probability. When `lower.tail = TRUE` (the default), the region from the left boundary to the p -quantile is shaded, representing $P(X \leq x_p) = p$. Setting `lower.tail = FALSE` shades the complementary upper tail.

Two-sided interval. The `p_lower` and `p_upper` parameters define a central region—the natural representation for a $(1 - \alpha)$ confidence interval.

Tail shading. Setting `shade_outside = TRUE` inverts the two-sided region, shading both tails. This directly represents the rejection region of a two-sided hypothesis test at level α .

```
p1 <- ggplot() +
  geom_pdf(fun = dnorm, xlim = c(-3, 3), p = 0.975) +
  ggtitle("p = 0.975")

p2 <- ggplot() +
  geom_pdf(
    fun = dnorm, xlim = c(-3, 3),
    p_lower = 0.025, p_upper = 0.975
  ) + ggtitle("Central 95%")

p3 <- ggplot() +
  geom_pdf(
    fun = dnorm, xlim = c(-3, 3),
    p_lower = 0.025, p_upper = 0.975, shade_outside = TRUE
```

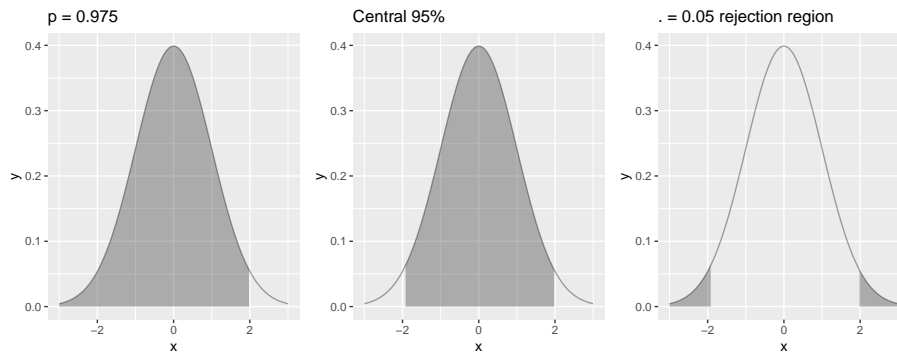


Figure 9: Three shading modes for `geom_pdf()`: lower tail ($p = 0.975$, left), central 95% interval (center), and two-tailed rejection region at $\alpha = 0.05$ (right).

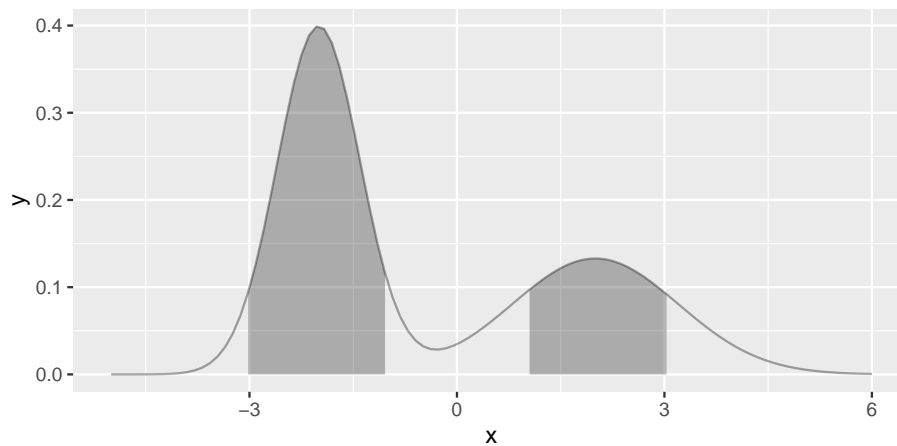


Figure 10: The 80% highest density region of an asymmetric bimodal density ($0.6\mathcal{N}(-2, 0.6^2) + 0.4\mathcal{N}(2, 1.2^2)$), shading both modes as two disjoint intervals with more area allocated to the taller, narrower component.

```
) + ggtitle("\u03b1 = 0.05 rejection region")
```

```
p1 | p2 | p3
```

Highest density region. The `shade_hdr` parameter specifies a coverage probability and shades the corresponding highest density region (HDR)—the smallest subset of the domain containing the specified probability mass. For multimodal densities the HDR may be disconnected, producing multiple disjoint shaded intervals. The algorithm follows [Otto and Kahle \(2023\)](#): evaluate the density on the grid, normalize the values to sum to one, sort in descending order, accumulate until the target coverage is reached, and shade all grid intervals at or above the resulting density threshold.

```
f_mix <- function(x) 0.6 * dnorm(x, mean = -2, sd = 0.6) + 0.4 * dnorm(x, mean = 2, sd = 1.2)
ggplot() +
  geom_pdf(fun = f_mix, xlim = c(-5, 6), shade_hdr = 0.8)
```

The shading boundaries for the interval-based modes are determined by trapezoidal accumulation. For n evaluation points x_1, \dots, x_n with density values y_1, \dots, y_n , the cumulative area at the k -th point is

$$A_k = \sum_{i=1}^{k-1} \frac{y_i + y_{i+1}}{2} (x_{i+1} - x_i). \quad (3)$$

The normalized values A_k/A_n are then compared against the specified probability thresholds to determine the shading boundaries.

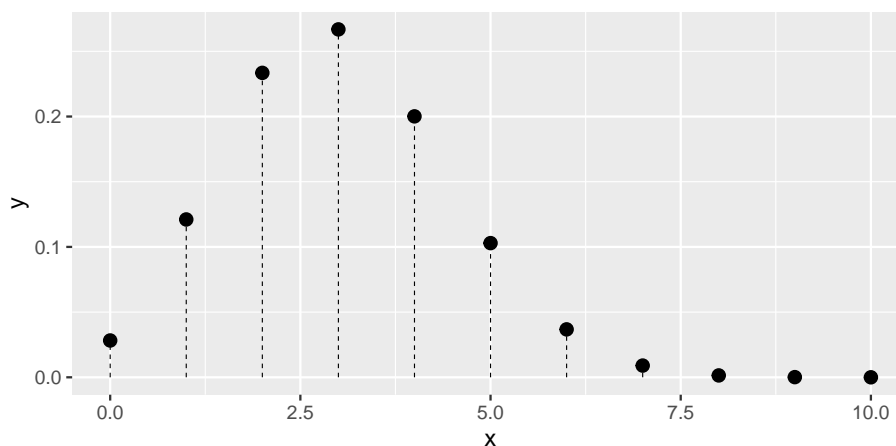


Figure 11: The PMF of a Binomial(10,0.3) distribution, rendered as a lollipop chart.

4.2 Probability mass functions: `geom_pmf()`

For a discrete random variable with integer support, the PMF $p(k) = P(X = k)$ satisfies $\sum_k p(k) = 1$. `geom_pmf()` evaluates the PMF at each integer in `xlim` and renders the result as a lollipop chart—vertical segments from the x -axis to the probability value, capped with points. The underlying `StatPMF` validates normalization by summing the computed probabilities and issues a warning if they depart significantly from unity. When the support is not a sequence of consecutive integers, the `support` argument accepts an explicit numeric vector that overrides `xlim`.

```
ggplot() +
  geom_pmf(fun = dbinom, xlim = c(0, 10), args = list(size = 10, prob = 0.3))
```

The `support` argument enables distributions whose mass points are not consecutive integers. Figure~12 illustrates this with the exact distribution of the sample mean $\bar{X}_n = n^{-1} \sum_{i=1}^n X_i$ of $n = 10$ i.i.d. Bernoulli(0.3) draws. The support is $\{0, 0.1, 0.2, \dots, 1\}$ and $P(\bar{X}_n = k/n) = \binom{n}{k} 0.3^k 0.7^{n-k}$. The Central Limit Theorem approximation $\bar{X}_n \approx \mathcal{N}(0.3, 0.3 \times 0.7/10)$ is overlaid as a scaled density curve whose peak matches the tallest lollipop, facilitating direct visual comparison of shape without a secondary axis.

```
p <- 0.3; n <- 10
sd_mean <- sqrt(p * (1 - p) / n)
f_mean <- function(x) dbinom(round(x * n), size = n, prob = p)
max_pmf <- max(f_mean(seq(0, 1, by = 1/n)))
scale <- max_pmf / dnorm(p, mean = p, sd = sd_mean)
f_clt <- function(x) scale * dnorm(x, mean = p, sd = sd_mean)

ggplot() +
  geom_pmf(fun = f_mean, support = seq(0, 1, by = 1/n)) +
  geom_pdf(fun = f_clt, xlim = c(0, 1))
```

`geom_pmf()` supports the same shading modes as `geom_pdf()`. The `p` parameter shades lollipops up to the p -quantile; `p_lower` and `p_upper` define a two-sided interval; `shade_outside = TRUE` inverts it; and `shade_hdr` shades the smallest set of mass points whose total probability meets or exceeds the target coverage. Unshaded lollipops are rendered in grey to preserve the distributional context. Because a discrete distribution may not achieve the exact target coverage, `shade_hdr` finds the smallest HDR with coverage \geq the requested level and issues a diagnostic via `cli` when the actual coverage differs from the target by more than 0.5 percentage points.

```
p1 <- ggplot() +
  geom_pmf(fun = dbinom, xlim = c(0, 10),
    args = list(size = 10, prob = 0.5), p = 0.8) +
  ggtitle("p = 0.8")

p2 <- ggplot() +
  geom_pmf(fun = dbinom, xlim = c(0, 10),
```

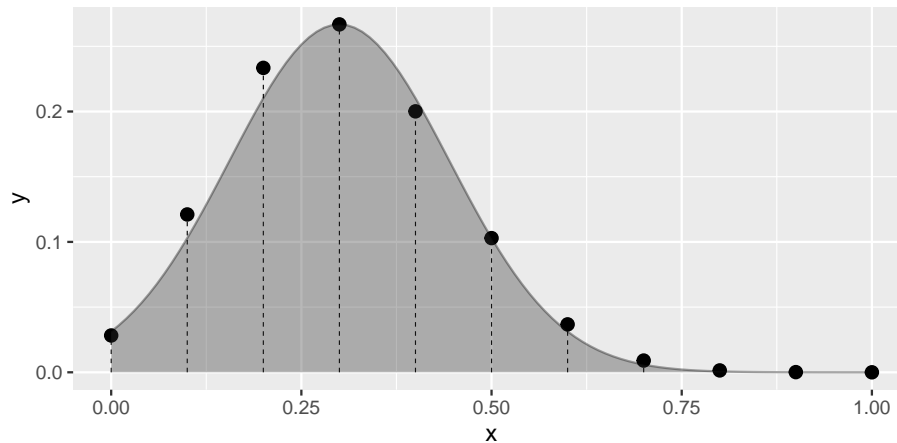


Figure 12: Exact distribution of the sample mean of 10 i.i.d. Bernoulli(0.3) draws (lollipops) with the CLT normal approximation overlaid as a scaled density curve.

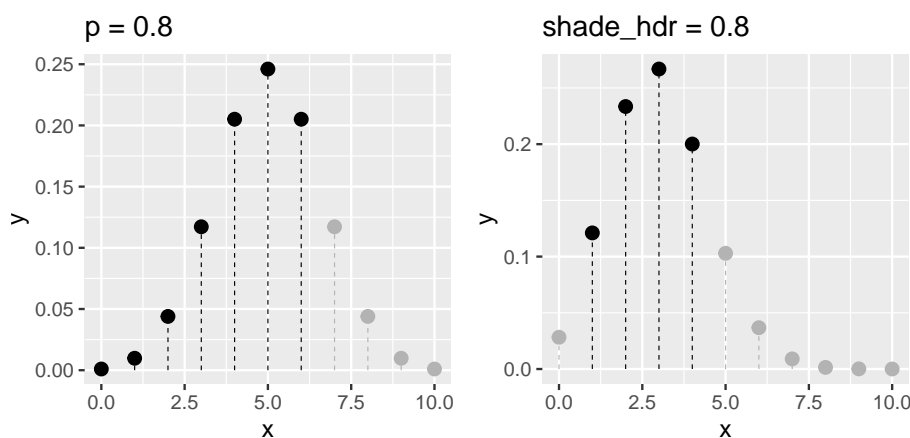


Figure 13: Shading modes for `geom_pmf()`: the lower 80% by cumulative probability (left) and the 80% HDR of a Binomial(10, 0.3) distribution (right). Unshaded lollipops are shown in grey.

```
args = list(size = 10, prob = 0.3), shade_hdr = 0.8) +
ggtitle("shade_hdr = 0.8")
```

p1 | p2

4.3 Cumulative distribution and survival functions

The CDF $F(x) = P(X \leq x)$ and survival function $S(x) = 1 - F(x) = P(X > x)$ are complementary summaries of a distribution. **ggfunction** provides continuous and discrete variants of both.

Continuous CDF (`geom_cdf()`). `geom_cdf()` evaluates a user-supplied CDF (e.g., `pnorm`) and renders it as a line. It supports the same `p`, `p_lower/p_upper`, and `shade_outside` shading parameters as `geom_pdf()`.

```
ggplot() +
  geom_cdf(fun = pnorm, xlim = c(-3, 3))
```

Discrete CDF (`geom_cdf_discrete()`). For discrete distributions, the CDF is a right-continuous step function. `geom_cdf_discrete()` takes a PMF, computes cumulative sums, and renders the result with horizontal segments, dashed vertical jumps, open circles at the pre-jump value (the left limit), and closed circles at the achieved value. The `show_points` and `show_vert` parameters independently suppress the endpoint circles or vertical jump segments. The `support` argument accepts an explicit numeric vector of mass points, overriding `xlim` for distributions with non-integer or non-consecutive support.

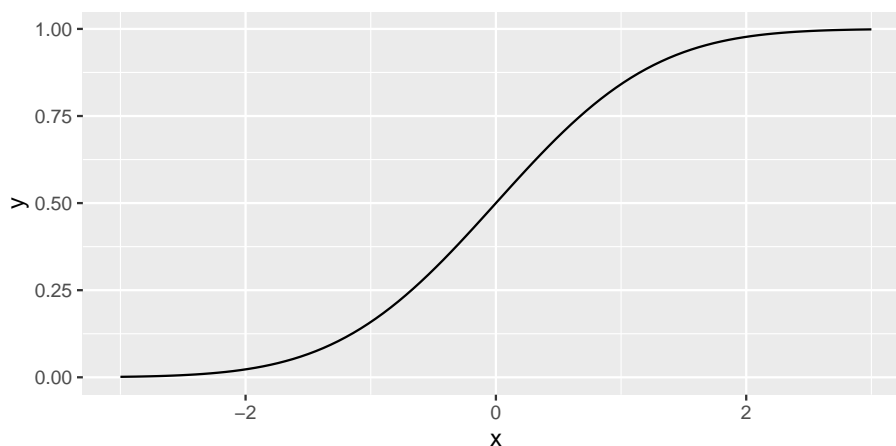


Figure 14: The standard normal CDF.

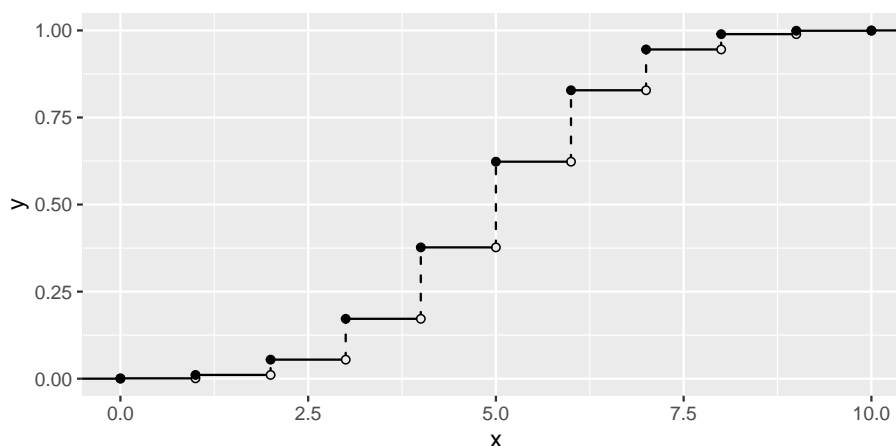


Figure 15: The discrete CDF of a Binomial(10, 0.5) distribution.

```
ggplot() +
  geom_cdf_discrete(
    fun = dbinom, xlim = c(0, 10), args = list(size = 10, prob = 0.5)
  )
```

Continuous survival (`geom_survival()`). In reliability theory and biostatistics, $S(x) = 1 - F(x) = P(X > x)$ gives the probability that the event of interest has not yet occurred by time x . `geom_survival()` accepts a CDF and plots its complement.

```
ggplot() +
  geom_survival(fun = pexp, xlim = c(0, 10), args = list(rate = 0.5))
```

Discrete survival (`geom_survival_discrete()`). For discrete distributions, $S(x) = 1 - F(x)$ is itself a right-continuous step function. `geom_survival_discrete()` computes S from a PMF and renders it with the same visual conventions as `geom_cdf_discrete()`. The support argument behaves as in `geom_cdf_discrete()`.

```
ggplot() +
  geom_survival_discrete(
    fun = dbinom, xlim = c(0, 10), args = list(size = 10, prob = 0.5)
  )
```

4.4 Quantile functions

The quantile function $Q(p) = \inf\{x : F(x) \geq p\}$ inverts the CDF. **ggfunction** provides continuous and discrete variants.

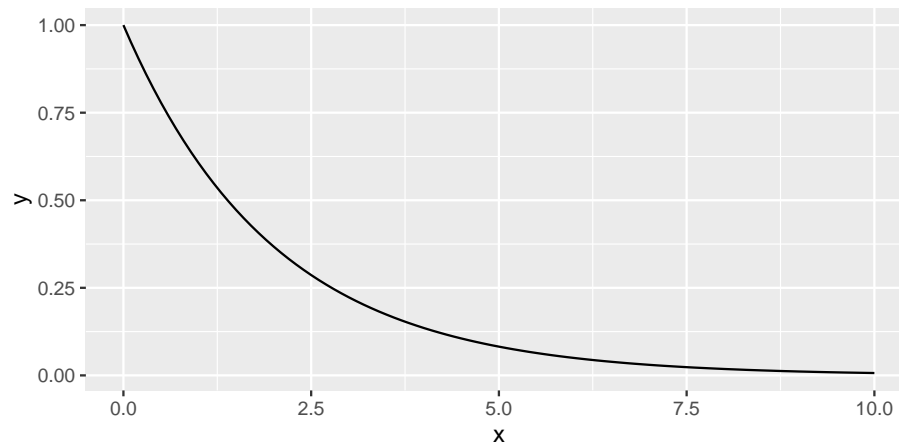


Figure 16: The survival function of an Exponential(0.5) distribution, $S(x) = e^{-0.5x}$.

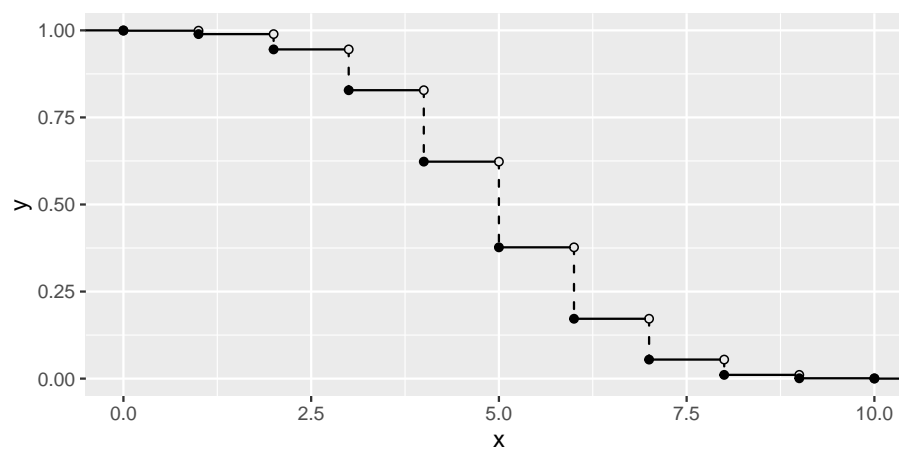


Figure 17: The discrete survival function of a Binomial(10, 0.5) distribution, descending from 1 to 0.

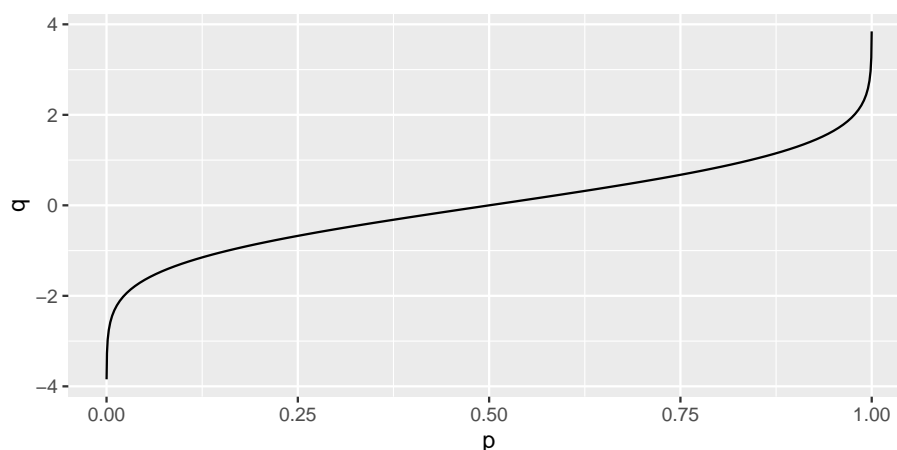


Figure 18: The quantile function of the standard normal distribution.

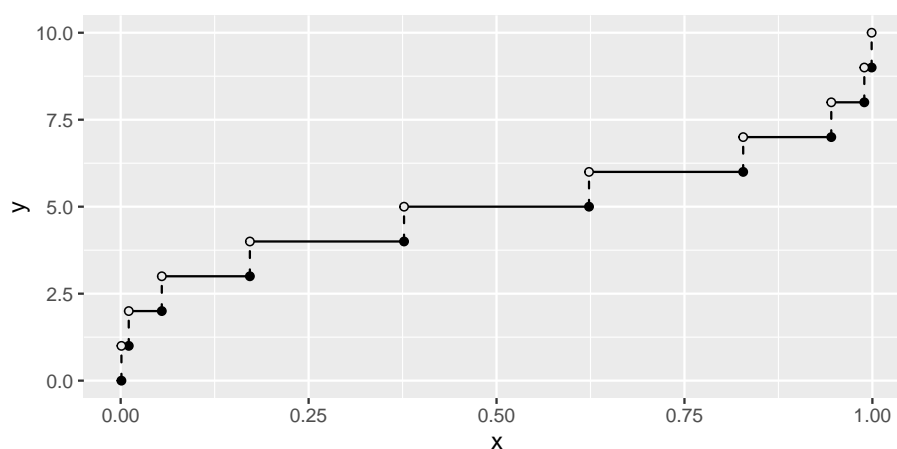


Figure 19: The discrete quantile function of a Binomial(10, 0.5) distribution as a left-continuous step function on $[0, 1]$.

Continuous (`geom_qf()`). `geom_qf()` evaluates a user-supplied quantile function (e.g., `qnorm`) over the unit interval $(0, 1)$. Evaluation points are placed at Chebyshev nodes of the first kind, $p_k = (1 - \cos((2k - 1)\pi/2n))/2$ for $k = 1, \dots, n$, which cluster near 0 and 1 where quantile functions are typically most curved and avoid evaluating at the exact endpoints, preventing $\pm\infty$ for unbounded distributions.

```
ggplot() +
  geom_qf(fun = qnorm)
```

Discrete (`geom_qf_discrete()`). The quantile function of a discrete distribution is a left-continuous step function on $[0, 1]$. `geom_qf_discrete()` takes a PMF, computes the cumulative sum, and renders the inverse function with horizontal segments on $[0, 1]$, dashed vertical jumps, closed circles at the bottom of each jump (the value is achieved), and open circles at the top (the next value is not yet reached). The support argument behaves as in `geom_cdf_discrete()`.

```
ggplot() +
  geom_qf_discrete(
    fun = dbinom, xlim = c(0, 10), args = list(size = 10, prob = 0.5)
  )
```

4.5 Hazard functions: `geom_hf()`

The hazard function

$$h(x) = \frac{f(x)}{S(x)} = \frac{f(x)}{1 - F(x)} \quad (4)$$

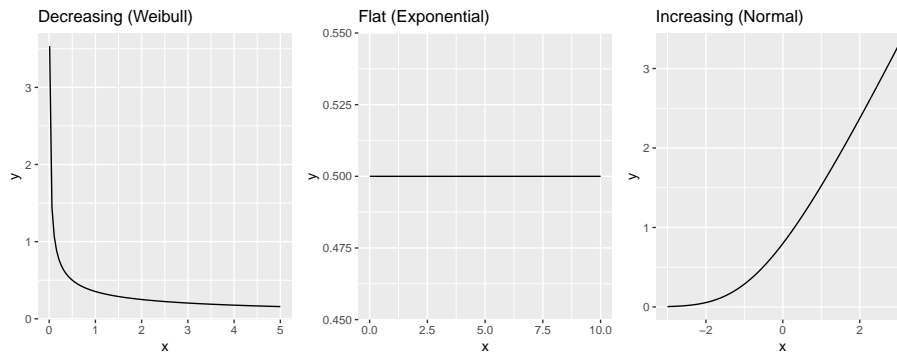


Figure 20: Three canonical hazard shapes: decreasing (Weibull(shape=0.5, scale=2), left), constant (Exponential(0.5), center), and increasing ($\mathcal{N}(0,1)$, right).

represents the instantaneous rate of failure at time x , conditional on survival to that point (Casella and Berger, 2002). `geom_hf()` requires both a PDF and a CDF, supplied via `pdf_fun` and `cdf_fun` respectively. The `args` parameter applies to both; `pdf_args` and `cdf_args` provide overrides where the two functions require different parameterizations.

Division by $S(x)$ can cause numerical instability in the tail as $S(x) \rightarrow 0$. StatHF guards against this by replacing values where $S(x) \leq 0$ with NaN, and GeomHF filters these before rendering.

Three canonical hazard shapes arise in reliability modeling: decreasing (infant mortality, where early failures dominate), constant (the memoryless Exponential distribution), and increasing (wear-out failure, where older units fail at higher rates).

```
p_decr <- ggplot() +
  geom_hf(
    pdf_fun = dweibull, cdf_fun = pweibull,
    xlim = c(0.01, 5), args = list(shape = 0.5, scale = 2)
  ) + ggtitle("Decreasing (Weibull)")

p_flat <- ggplot() +
  geom_hf(
    pdf_fun = dexp, cdf_fun = pexp,
    xlim = c(0.01, 10), args = list(rate = 0.5)
  ) + ggtitle("Flat (Exponential)")

p_incr <- ggplot() +
  geom_hf(
    pdf_fun = dnorm, cdf_fun = pnorm,
    xlim = c(-3, 3), args = list(mean = 0, sd = 1)
  ) + ggtitle("Increasing (Normal)")

p_decr | p_flat | p_incr
```

5 Implementation

5.1 Architecture

Each **ggfunction** layer is implemented as a pair of ggproto objects—a Stat and a Geom—wired together by a constructor function. The constructor creates a standard `ggplot2` `layer()` call, threading the user-supplied function and its parameters through to the stat’s `compute_group()` method. That method performs the core computation: evaluating the function on a grid or sequence and returning a data frame whose columns correspond to the aesthetic variables expected by the geom.

The separation of computation (stat) from rendering (geom) is not merely organizational. It allows users to substitute alternative geoms or compose multiple layers from the same stat. The `StatFunction2d` object, for instance, computes a grid of scalar-field values that can be rendered by `GeomRaster`, `GeomContour`, or `GeomContourFilled`, selected at construction time through the `type` parameter.

5.2 Function injection

R's ... mechanism does not compose well when a function is called inside a stat's `compute_group()` method, which must simultaneously receive **ggplot2**-internal parameters. **ggfunction** adopts the approach used by `ggplot2::stat_function()`: additional arguments are collected in a named list (`args`) and injected at the call site via `rlang::inject()`:

```
fun_injected <- function(x) {
  rlang::inject(fun(x, !!!args))
}
```

This pattern is simple, composable, and avoids the ambiguity of routing function arguments through ...

5.3 Numerical considerations

Several layers involve numerical integration or accumulation:

- **PDF validation** uses `stats::integrate()` with adaptive quadrature to verify that the supplied function integrates to one over the specified domain.
- **Shading boundaries** are computed by trapezoidal accumulation (3), which is efficient and sufficient for the smooth densities encountered in practice.
- **Boundary interpolation** in `GeomFunction1d` uses `stats::approxfun()` to compute exact y -values at shade endpoints that fall between evaluation points.
- **Quantile function evaluation** in `geom_qf()` uses Chebyshev nodes of the first kind rather than a uniform grid, concentrating evaluation points near $p = 0$ and $p = 1$ where the function is most curved and avoiding evaluation at the exact endpoints.
- **Vector field integration** (via **ggvfields**) uses a fourth-order Runge–Kutta method with configurable step size and maximum iterations.

The default resolution of $n = 101$ for one-dimensional functions and $n = 50$ for two-dimensional grids (yielding 2500 evaluation points) balances visual fidelity against computational cost. Users may increase n for functions with fine-scale structure.

6 Composability and the grammar

Because every **ggfunction** layer is a standard **ggplot2** layer, it composes freely with the rest of the grammar. Layers can be overlaid, scaled, themed, annotated, and faceted exactly as one would with any other geom. The following example superimposes two normal densities with different means and spreads, demonstrating that function visualization and standard **ggplot2** idioms are fully interoperable:

```
ggplot() +
  geom_pdf(
    fun = dnorm, xlim = c(-5, 8),
    args = list(mean = 0, sd = 1), alpha = 0.4
  ) +
  geom_pdf(
    fun = dnorm, xlim = c(-5, 8),
    args = list(mean = 3, sd = 1.5), alpha = 0.4
  ) +
  labs(x = "x", y = "f(x)", title = "Comparing two normal densities") +
  theme_minimal()
```

This composability is the primary advantage of embedding function visualization within the grammar of graphics, rather than providing standalone plotting functions.

7 Related packages

Several existing packages overlap with **ggfunction** in purpose, and understanding their scope helps clarify where **ggfunction** fits.

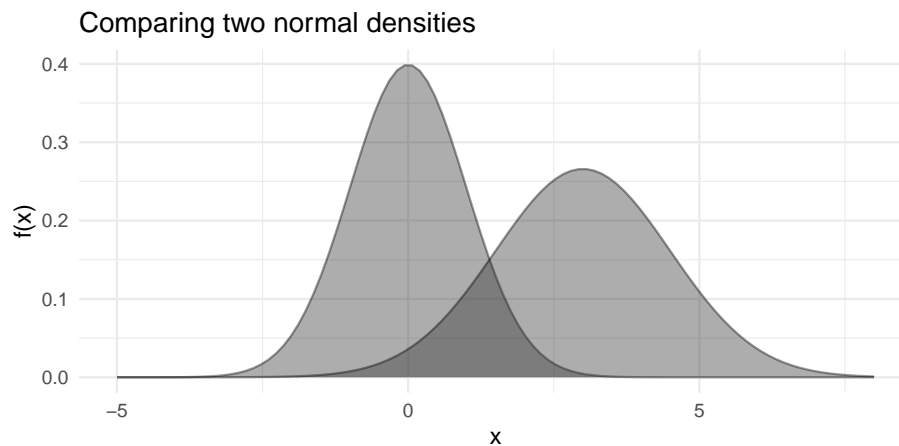


Figure 21: Two normal densities with different means and spreads overlaid in a single plot, demonstrating composability with the `ggplot2` grammar of graphics.

1. **ggplot2** (Wickham, 2016). The `stat_function()` layer handles scalar functions $\mathbb{R} \rightarrow \mathbb{R}$ competently, and `geom_function_1d_1d()` builds directly on this foundation. **ggplot2** provides no analogues for parametric curves, scalar fields, vector fields, or probability-specific shading operations.
2. **ggdist** (Kay, 2024). This package excels at visualizing distributional uncertainty in Bayesian workflows: posterior distributions, predictive intervals, and uncertainty estimates from fitted models. Its geoms work primarily with samples or distributional objects rather than the $d/p/q/h$ function families that **ggfunction** targets.
3. **ggdensity** (Otto and Kahle, 2023). This package improves bivariate density visualization in **ggplot2**, with a particular focus on highest density regions (HDRs). **ggfunction**'s `shade_hdr` parameter in `geom_pdf()` follows the HDR computation approach introduced there: normalizing density values, sorting in descending order, and accumulating until the target coverage is reached.
4. **mosaic** (Pruim et al., 2017). The `plotDist()` function provides quick plots of named distributions and supports shading, but it is built on the **lattice** (Sarkar, 2008) graphics framework rather than **ggplot2**, and therefore cannot be composed with **ggplot2** layers.
5. **metR** (Campitelli, 2024). This package provides contour and vector-field geoms designed for meteorological data, requiring precomputed grids as inputs rather than function objects. Users who already have gridded data may find **metR** more convenient; users who want to visualize a function directly will find **ggfunction** more natural.
6. **ggvfields** (Turner and Kahle, 2026). **ggfunction**'s `geom_function_2d_2d()` delegates to **ggvfields** for streamline computation. For users whose primary interest is vector fields—including stream plots, gradient fields, and potential functions—**ggvfields** offers a substantially richer collection of tools built on the same **ggplot2** extension architecture.

Taken together, these packages reflect the breadth of interest in function visualization within the R ecosystem. **ggfunction** is distinguished by its unified dimensional taxonomy and its integration of the full probability distribution function family—capabilities that, to our knowledge, no other single package provides within the grammar of graphics.

8 Summary

ggfunction extends **ggplot2** with a principled framework for visualizing mathematical functions and probability distributions. Its organizing taxonomy—classifying functions by the dimensions of their domain and codomain—yields a small, memorable API that covers the function types most commonly encountered in mathematics and statistics. The probability distribution family covers nine geoms spanning continuous and discrete distributions: density, CDF, PMF, quantile, discrete CDF, discrete quantile, discrete survival, survival, and hazard functions. Each supports the shading operations central to statistical pedagogy—marking quantiles, delineating confidence regions, highlighting rejection areas, and shading highest density regions.

By implementing each layer as a `ggproto` `stat`–`geom` pair, the package inherits the full composability of the grammar of graphics. Functions can be overlaid, themed, faceted, and annotated using the

same tools that **ggplot2** users already know. The args injection pattern provides a clean interface for parameterized function families, and built-in validation catches common errors—such as unnormalized densities—before they produce misleading graphics.

ggfunction is available at <https://github.com/dusty-turner/ggfunction> and can be installed via `pak::pak("dusty-turner/ggfunction")`.

References

- E. Campitelli. *metR: Tools for Easier Analysis of Meteorological Fields*, 2024. URL <https://CRAN.R-project.org/package=metR>. R package. [p16]
- G. Casella and R. L. Berger. *Statistical Inference*. Duxbury, 2nd edition, 2002. [p14]
- G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2024. URL <https://CRAN.R-project.org/package=cli>. R package. [p7]
- M. Kay. ggdist: Visualizations of distributions and uncertainty in the grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):983–993, 2024. doi: 10.1109/TVCG.2023.3327195. [p1, 16]
- J. Otto and D. Kahle. ggdensity: Improved bivariate density visualization in R. *The R Journal*, 15(2): 220–236, 2023. doi: 10.32614/RJ-2023-048. [p8, 16]
- R. Pruim, D. T. Kaplan, and N. J. Horton. The mosaic package: Helping students to “think with data” using R. *The R Journal*, 9(1):77–102, 2017. doi: 10.32614/RJ-2017-024. [p1, 16]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. ISBN 978-0-387-75968-5. URL <https://lmdvr.r-forge.r-project.org>. [p1, 16]
- D. Turner and D. Kahle. *ggvfields: Vector Field Visualizations with ggplot2*, 2026. URL <https://github.com/dusty-turner/ggvfields>. R package version 0.1.0. [p6, 16]
- H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1): 3–28, 2010. doi: 10.1198/jcgs.2009.07098. [p1]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2nd edition, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2-book.org/>. [p1, 16]
- L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag New York, 2nd edition, 2005. doi: 10.1007/0-387-28695-0. [p2]

Dusty Turner
United States Military Academy
West Point, NY
ORCID: 0000-0000-0000-0000
dusty.s.turner@gmail.com

James Otto
Alcon Inc.
Fort Worth, TX
james.otto@alcon.com

Rodney X. Sturdivant
Baylor University
Waco, TX
rodney_sturdivant@baylor.edu

David Kahle
Baylor University
Waco, TX
david_kahle@baylor.edu