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

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

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 $f\colon \mathbb{R} \to \mathbb{R}$, parametric curves $\gamma\colon \mathbb{R} \to \mathbb{R}^2$, scalar fields $f\colon \mathbb{R}^2 \to \mathbb{R}$, and vector fields $\mathbf{F}\colon \mathbb{R}^2 \to \mathbb{R}^2$. The second family provides specialized geoms for probability distributions—density, cumulative distribution, mass, quantile, 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 $f\colon \mathbb{R} \to \mathbb{R}$, 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 outside 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: $\mathbb{R} \to \mathbb{R}$, $\mathbb{R} \to \mathbb{R}^2$, $\mathbb{R}^2 \to \mathbb{R}$, and $\mathbb{R}^2 \to \mathbb{R}^2$.
2. A **probability distribution family** providing specialized layers for density, cumulative distribution, mass, quantile, 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:

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

Each class admits a natural visual encoding:

- **Scalar functions** ($\mathbb{R} \to \mathbb{R}$): curves in the Cartesian plane, optionally with region shading between the curve and the x-axis.
- **Parametric curves** ($\mathbb{R} \to \mathbb{R}^2$): directed paths with color encoding the time parameter.
- **Scalar fields** ($\mathbb{R}^2 \to \mathbb{R}$): raster heatmaps, contour lines, or filled contour regions.
- **Vector fields** ($\mathbb{R}^2 \to \mathbb{R}^2$): 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, args = list(mean = 5, sd = 2), xlim = c(0, 10))
```

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: $\mathbb{R} \to \mathbb{R}$

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.

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

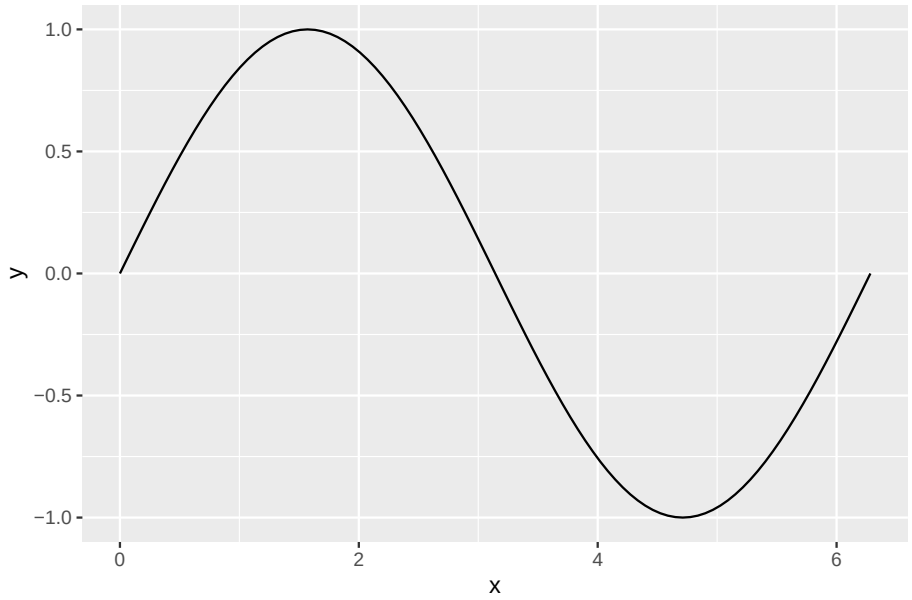

Figure 1: The sine function over one full period.

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 = dnorm, xlim = c(-3, 3),
    shade_from = -1, shade_to = 1
  )
```

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

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.

```
f_spiral <- function(t) c(sin(t), t * cos(t))

ggplot() +
  geom_function_1d_2d(fun = f_spiral, tlim = c(0, 20), tail_point = TRUE)
```

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 $\gamma(t) = (A\sin(at + \delta),\ B\sin(bt))$, for example, can be explored by varying frequency and phase:

```
lissajous <- function(t, A = 1, B = 1, a = 3, b = 2, delta = pi/2) {
  c(A * sin(a * t + delta), B * sin(b * t))
}

ggplot() +
  geom_function_1d_2d(
    fun = lissajous, tlim = c(0, 2 * pi), arrow = NULL,
    args = list(A = 1, B = 1, a = 3, b = 2, delta = pi/2)
  )
```

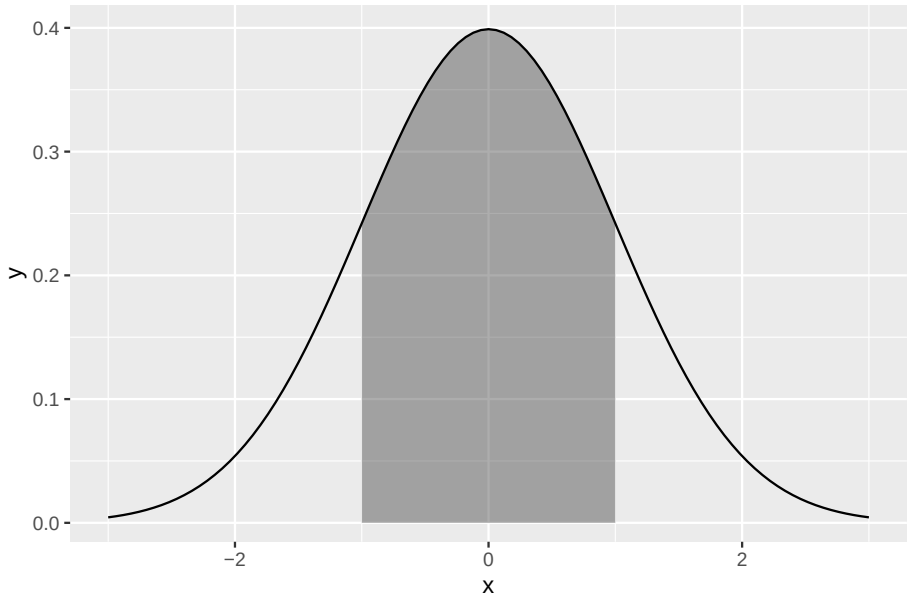

Figure 2: The standard normal density with the interval $[-1, 1]$ shaded, corresponding to approximately 68% of the total area.

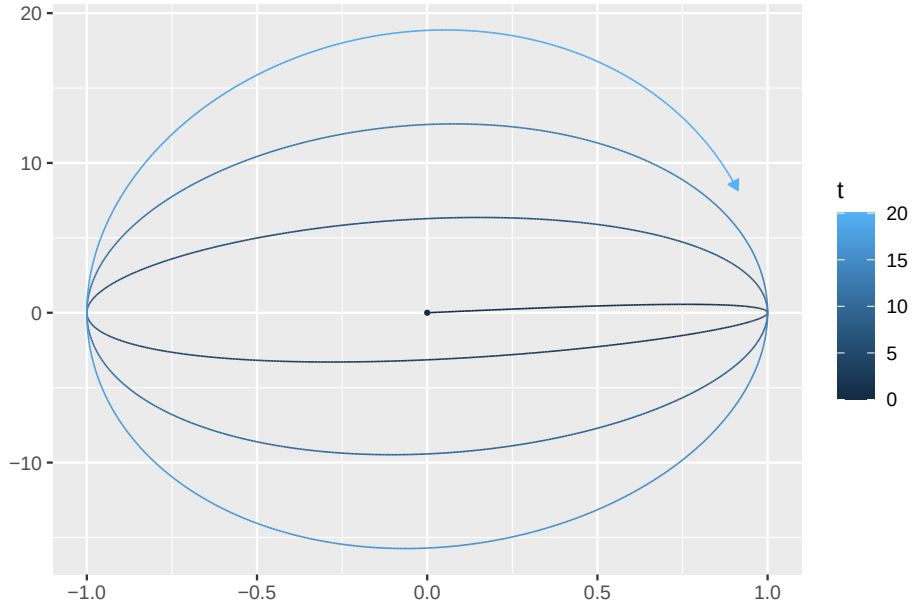

Figure 3: A parametric spiral $\gamma(t) = (\sin t,\ t\cos t)$ for $t \in [0, 20]$, with color encoding the time parameter.

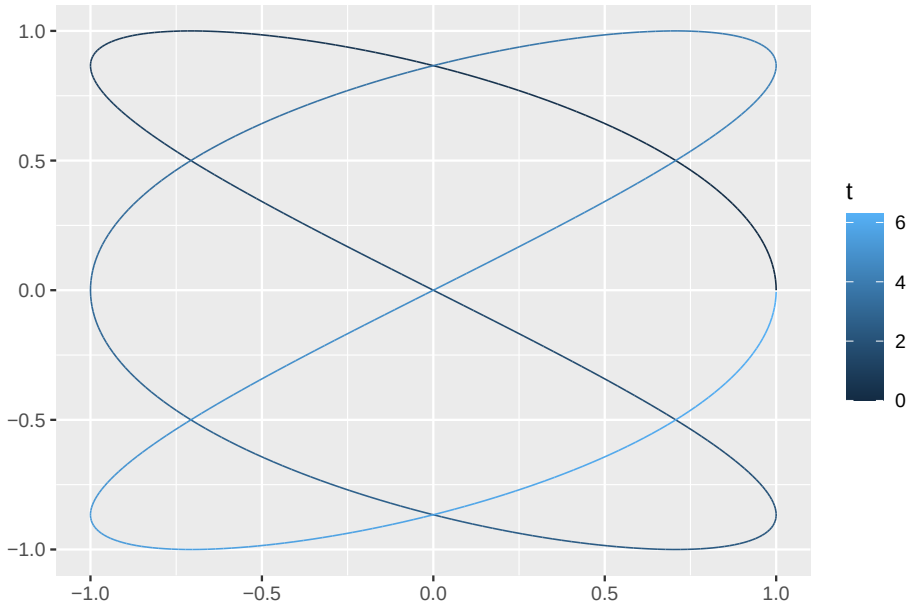

Figure 4: A Lissajous figure with frequency ratio $a/b = 3/2$ and phase offset $\delta = \pi/2$.

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

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 rendered by `ggplot2::GeomContour`.
- `"contour_filled"`: filled regions between contour levels rendered by `ggplot2::GeomContourFilled`.

```
f_gaussian <- function(u) {
  x <- u[1]; y <- u[2]
  exp(-(x^2 + y^2) / 2)
}

ggplot() +
  geom_function_2d_1d(fun = f_gaussian, xlim = c(-3, 3), ylim = c(-3, 3))
```

The contour modes are selected simply by changing `type`:

```
library("gridExtra")

p_contour <- ggplot() +
  geom_function_2d_1d(
    fun = f_gaussian, xlim = c(-3, 3), ylim = c(-3, 3), type = "contour"
  ) + ggtitle("type = \"contour\"")

p_filled <- ggplot() +
  geom_function_2d_1d(
    fun = f_gaussian, xlim = c(-3, 3), ylim = c(-3, 3), type = "contour_filled"
  ) + ggtitle("type = \"contour_filled\"")

grid.arrange(p_contour, p_filled, ncol = 2)
```

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.

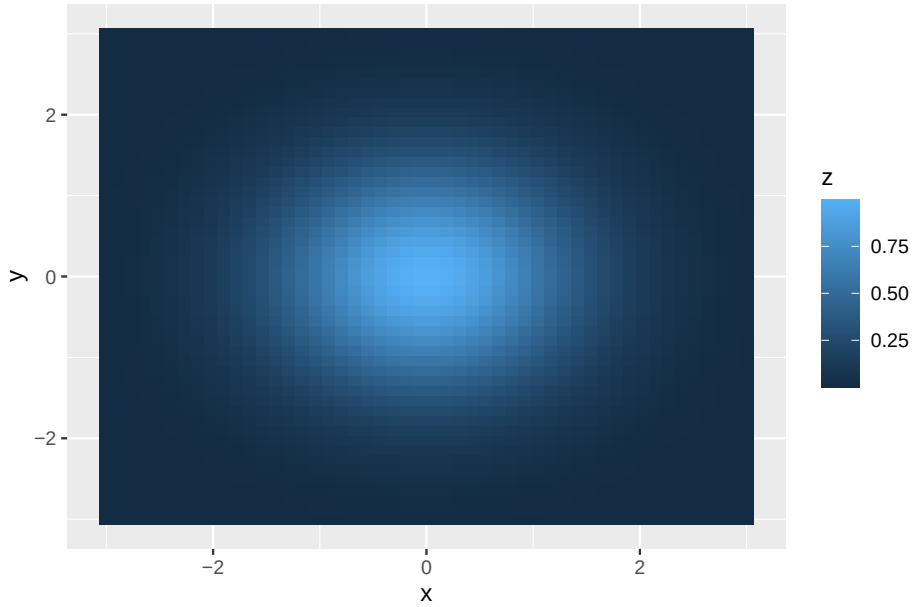

Figure 5: A Gaussian bump $f(x, y) = \exp(-(x^2 + y^2)/2)$ rendered as a raster heatmap.

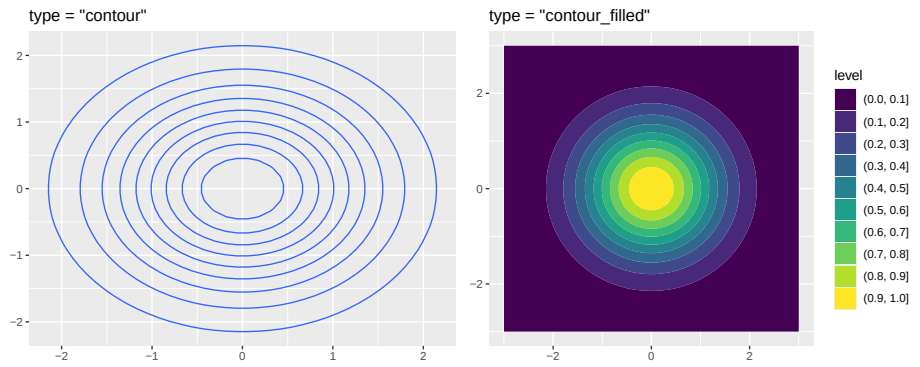

Figure 6: The same Gaussian bump rendered as contour lines (left) and filled contours (right).

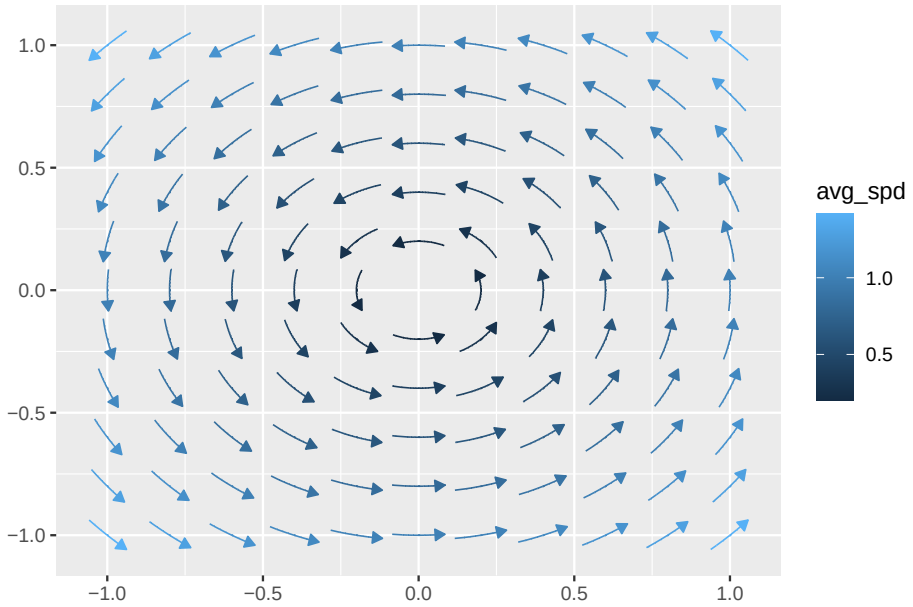

Figure 7: The rotation field $\mathbf{F}(x, y) = (-y, x)$ visualized as streamlines. Each curve follows the counterclockwise flow induced by the field.

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

A vector field $\mathbf{F}(x, y) = (F_1(x, y),\ F_2(x, y))$ assigns a direction and magnitude to each point in the plane. Visualizing such fields requires computing integral curves—trajectories that are everywhere tangent to the field. `geom_function_2d_2d()` provides this capability by delegating to **ggvfields** (Turner and Kahle, 2026), which numerically integrates the field using a fourth-order Runge–Kutta method. Streamlines are seeded on a regular grid and rendered as directed paths.

```
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))
```

The `normalize` parameter (default `TRUE`) scales streamline lengths relative to the grid spacing, producing a uniform visual density. Setting `normalize = FALSE` allows streamlines to reflect the true magnitude of the field through their arc length, controlled by the integration time `T`. 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.

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 seven 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

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

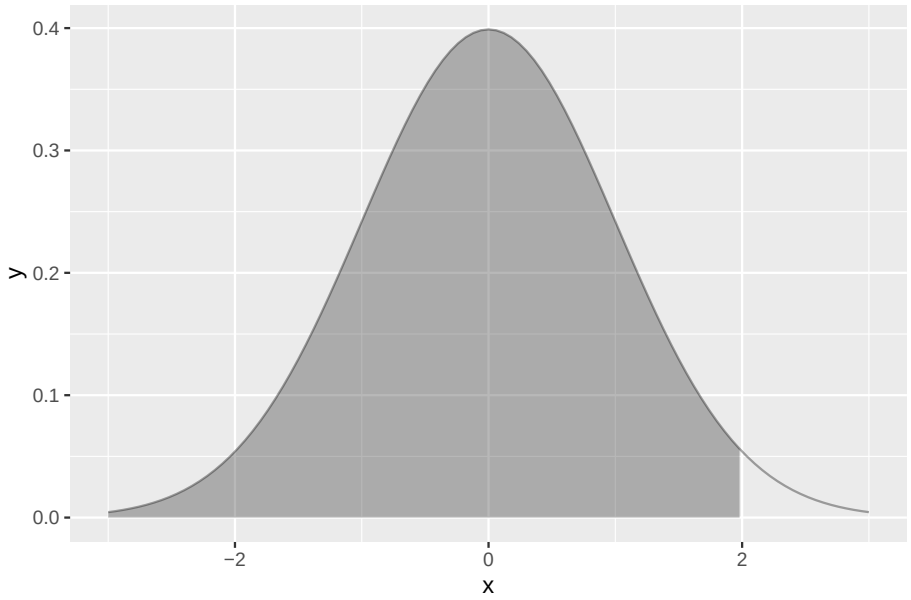

Figure 8: Standard normal density with the lower 97.5% shaded.

`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.

Three 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.

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

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

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

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.

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

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

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

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

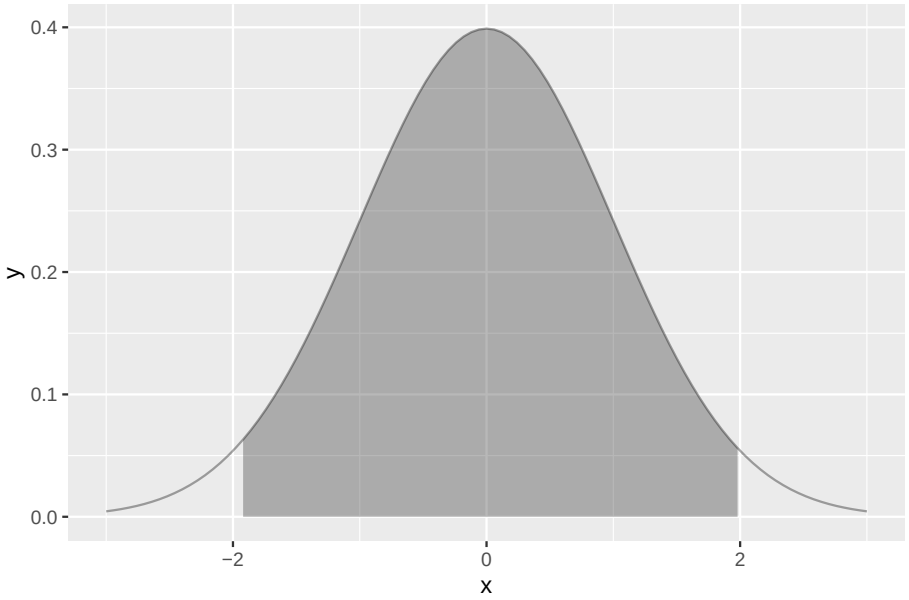

Figure 9: The central 95% of the standard normal density, corresponding to the interval between the 2.5th and 97.5th percentiles.

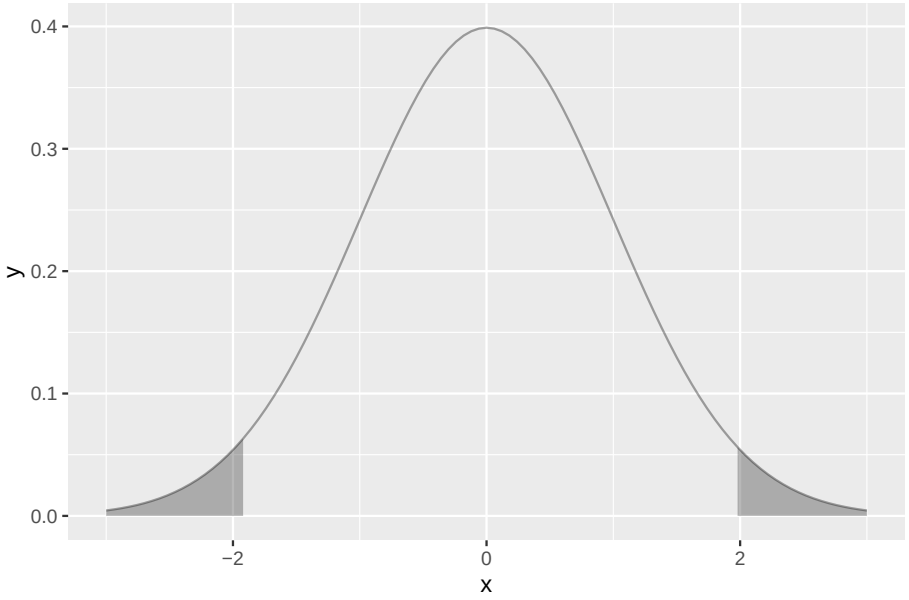

Figure 10: The rejection region of a two-sided test at $\alpha = 0.05$, shading the area outside the central 95%.

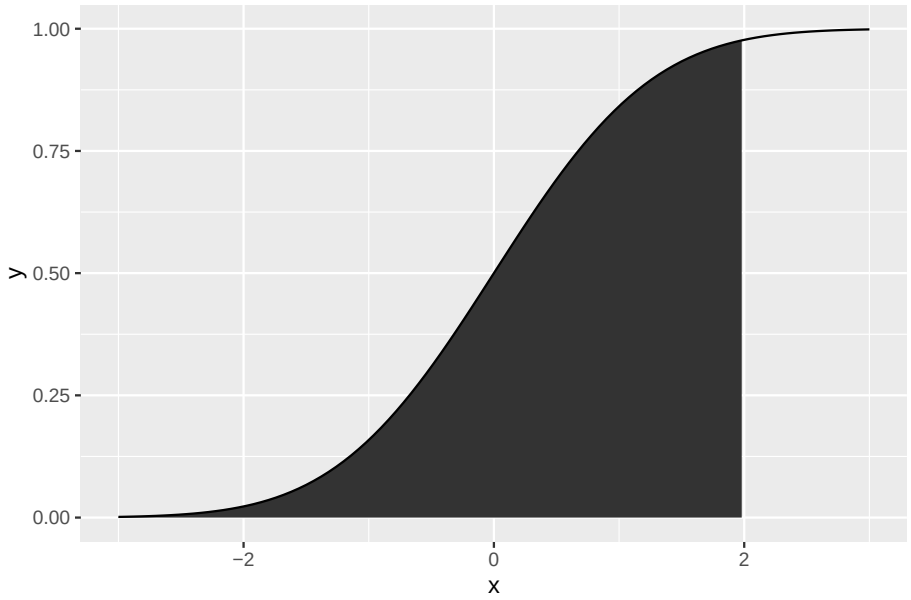

Figure 11: The standard normal CDF with the region below $p = 0.975$ shaded.

4.2 Cumulative distribution functions: `geom_cdf()`

The CDF $F(x) = P(X \leq x)$ is the integral of the density:

$$F(x) = \int_{-\infty}^{x} f(t)\, dt. \tag{4}$$

`geom_cdf()` evaluates a user-supplied CDF directly (e.g., `pnorm`) and supports the same `p`, `p_lower`/`p_upper`, and `shade_outside` parameters as `geom_pdf()`. Since the CDF maps directly to probabilities, the shading thresholds correspond to horizontal lines at the specified levels.

```
ggplot() +
  geom_cdf(fun = pnorm, xlim = c(-3, 3), p = 0.975)
```

4.3 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.

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

4.4 Quantile functions: `geom_qf()`

The quantile function $Q(p) = F^{-1}(p) = \inf\{x : F(x) \geq p\}$ inverts the CDF. `geom_qf()` evaluates a user-supplied quantile function (e.g., `qnorm`) over the unit interval $(0, 1)$.

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

4.5 Discrete CDFs: `geom_discrete_cdf()`

For discrete distributions, the CDF is a step function. `geom_discrete_cdf()` takes a PMF as input, computes cumulative sums, and renders the result as step-function vertices constructed by the internal `build_step_polygon()` helper. Like `geom_cdf()`, it supports `p`-based shading.

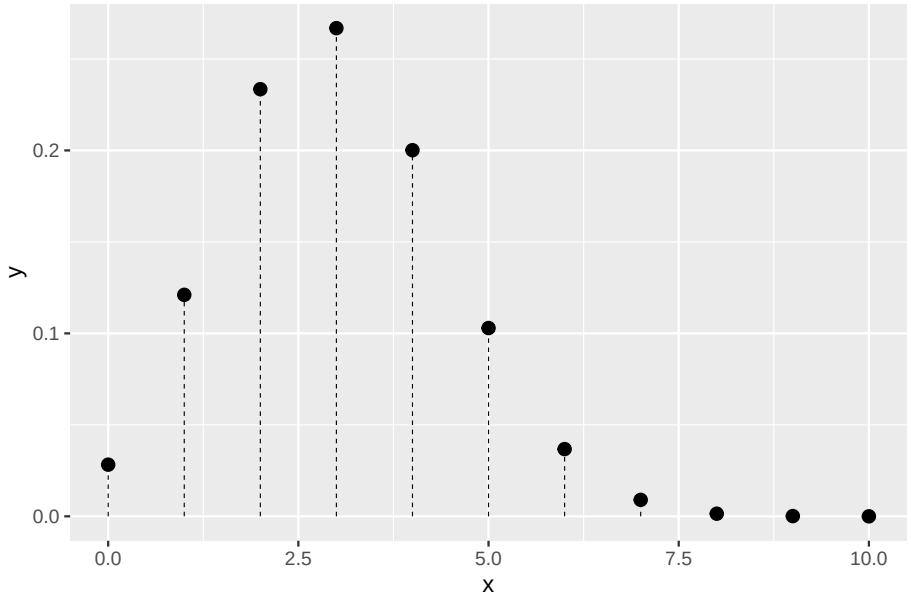

Figure 12: The PMF of a Binomial$(10, 0.3)$ distribution, rendered as a lollipop chart.

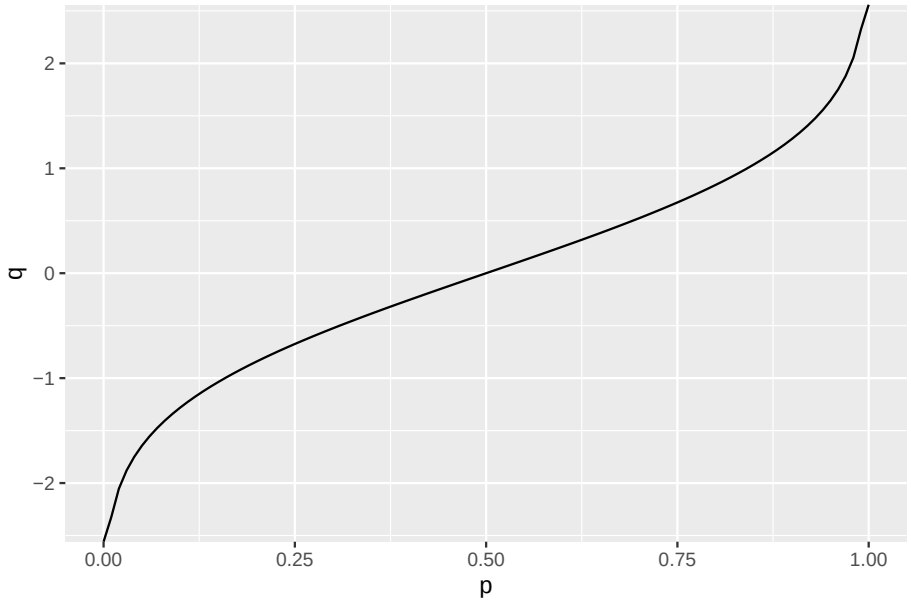

Figure 13: The quantile function of the standard normal distribution.

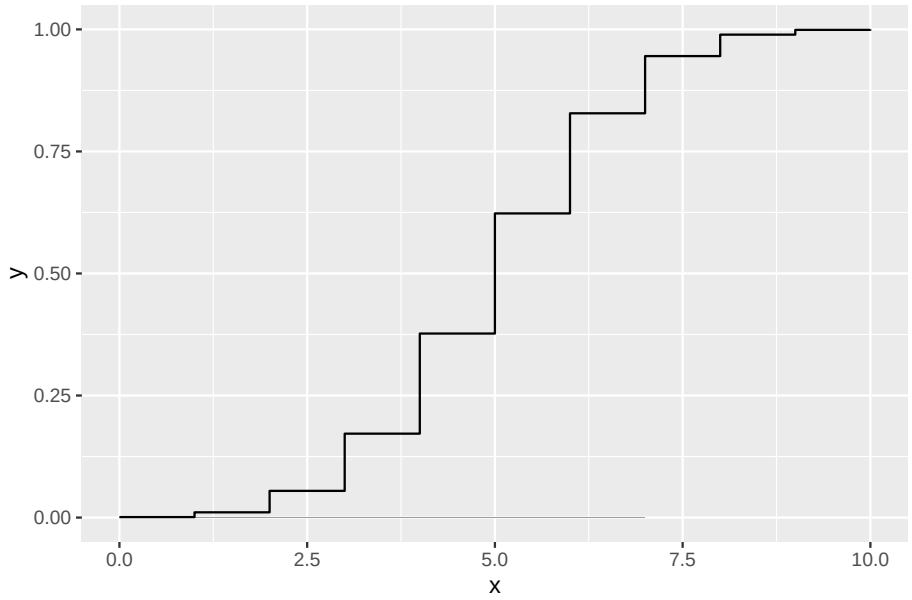

Figure 14: The discrete CDF of a Binomial$(10, 0.5)$ distribution with the region below $p = 0.9$ shaded.

```
ggplot() +
  geom_discrete_cdf(
    fun = dbinom, args = list(size = 10, prob = 0.5),
    xlim = c(0, 10), p = 0.9
  )
```

4.6 Survival functions: `geom_survival()`

In reliability theory and biostatistics, the survival function

$$S(x) = 1 - F(x) = P(X > x) \tag{5}$$

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, args = list(rate = 0.5), xlim = c(0, 10))
```

4.7 Hazard functions: `geom_hf()`

The hazard function

$$h(x) = \frac{f(x)}{S(x)} = \frac{f(x)}{1 - F(x)} \tag{6}$$

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 $S(x) \to 0$. `StatHF` guards against this by replacing values where $S(x) \leq 0$ with `NaN`, and `GeomHF` filters these before rendering.

```
ggplot() +
  geom_hf(
    pdf_fun = dexp, cdf_fun = pexp,
    args = list(rate = 0.5), xlim = c(0.01, 10)
  )
```

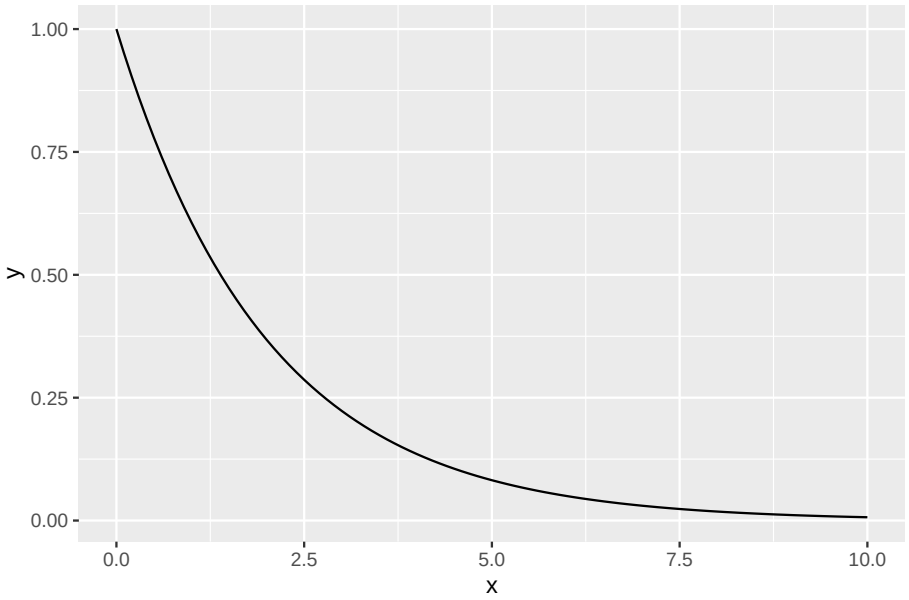

Figure 15: The survival function of an Exponential(0.5) distribution, $S(x) = e^{-0.5x}$.

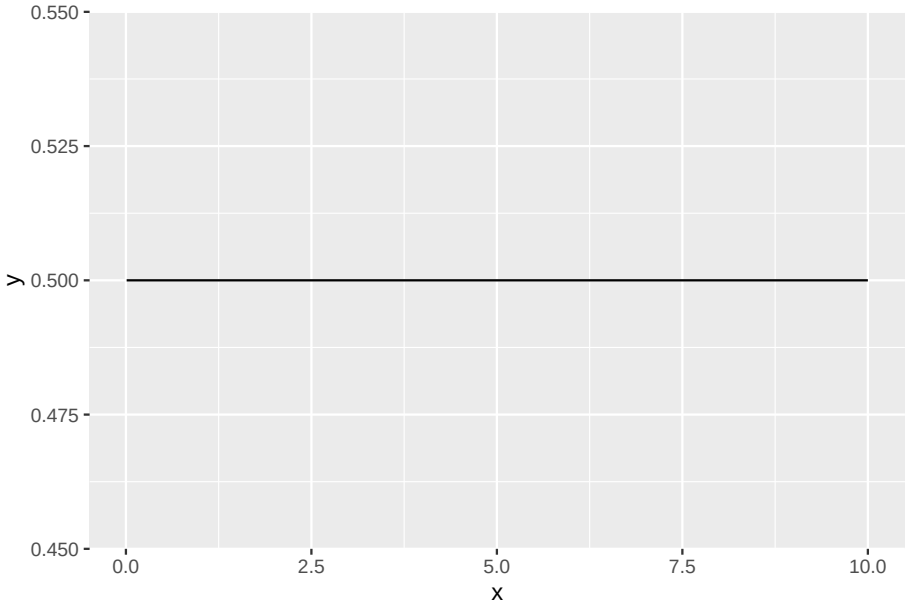

Figure 16: The hazard function of an Exponential(0.5) distribution, constant at $h(x) = 0.5$—a consequence of the memoryless property.

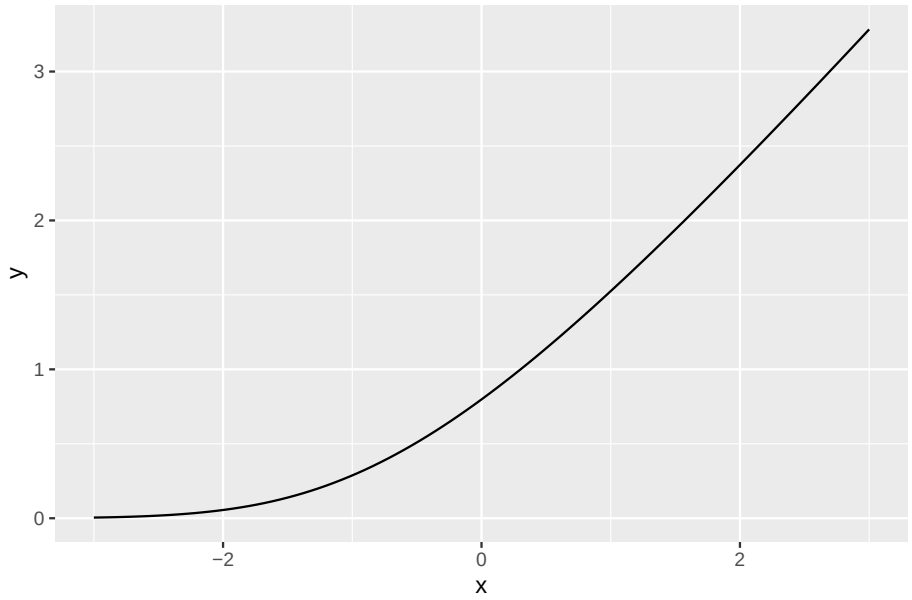

Figure 17: The increasing hazard function of the standard normal distribution.

For distributions without the memoryless property, the hazard function reveals structure not apparent from the density alone. The standard normal distribution has an increasing hazard, for instance:

```
ggplot() +
  geom_hf(
    pdf_fun = dnorm, cdf_fun = pnorm,
    args = list(mean = 0, sd = 1), xlim = c(-3, 3)
  )
```

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 `...`.

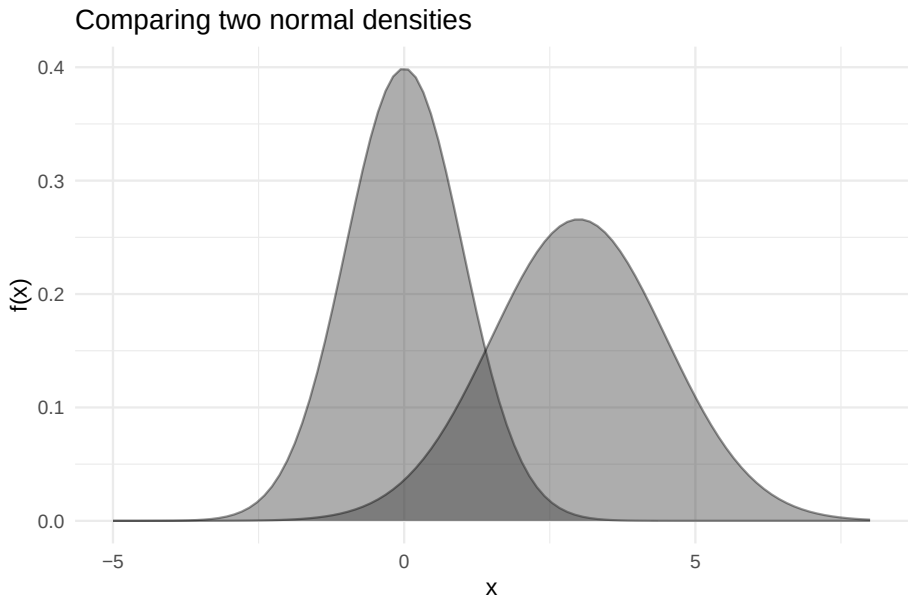

Figure 18: Two normal densities with different means and spreads overlaid in a single plot, demonstrating composability with the ggplot2 grammar of graphics.

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.
- **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, args = list(mean = 0, sd = 1),
    xlim = c(-5, 8), alpha = 0.4
  ) +
  geom_pdf(
    fun = dnorm, args = list(mean = 3, sd = 1.5),
    xlim = c(-5, 8), 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.

1. **ggplot2** (Wickham, 2016). The `stat_function()` layer handles scalar functions $\mathbb{R} \to \mathbb{R}$ 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. **mosaic** (Pruim et al., 2017). The `plotDist()` function provides quick plots of named distributions and supports shading, but it operates outside the grammar of graphics and cannot be composed with **ggplot2** layers.
4. **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.
5. **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 adds specialized support for the shading operations central to statistical pedagogy: marking quantiles, delineating confidence regions, and highlighting rejection areas.

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")`.

9 Acknowledgments

We thank the developers of **ggplot2** for the extension system that makes packages like **ggfunction** possible.

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. [p12]

G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2024. URL `https://CRAN.R-project.org/package=cli`. R package. [p8]

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]

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. Turner and D. Kahle. *ggvfields: Vector Field Visualizations with ggplot2*, 2026. URL https://github.com/dusty-turner/ggvfields. R package version 0.1.0. [p7, 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

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

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