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

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

Abstract We introduce ggfunction, an R package that extends ggplot2 to provide a principled, unified interface for visualizing mathematical functions and probability distributions. The package is organized around two complementary families. The first is a dimensional taxonomy that classifies 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 that arises naturally in hypothesis testing and interval estimation. By embedding function evaluation, numerical integration, and validation directly into the ggplot2 layer system, ggfunction allows users to move from mathematical definition to publication-quality visualization in a single, composable call. We describe the design of the package, demonstrate its use across both families, and discuss its role in statistical pedagogy.

1 Introduction

The visualization of mathematical functions is a fundamental task in both applied mathematics and statistics education. Whether an instructor is sketching a probability density on the board, a researcher is inspecting a scalar field, or a student is exploring the behavior of a parametric curve, the need to move quickly from a function's definition to its graphical representation arises constantly. In R, the **ggplot2** system (Wickham, 2010, 2016) has become the dominant framework for data visualization, yet its native support for plotting functions—as opposed to data—remains limited. The built-in `stat_function()` evaluates a univariate function over a specified range and renders the result as a path. This single mechanism handles $f\colon \mathbb{R} \to \mathbb{R}$ adequately, but offers no support for parametric curves, scalar fields, vector fields, or the rich family of probability distribution functions that appear throughout statistics.

Several packages address parts of this gap. The **ggdist** package (Kay, 2024) provides sophisticated distribution visualizations oriented toward Bayesian uncertainty communication. The **mosaic** package (Pruim et al., 2017) offers `plotDist()` for pedagogical work, but outside the grammar-of-graphics paradigm. The **metR** package provides geoms for meteorological contour and vector-field displays, though it is designed around gridded data rather than function objects. None of these packages offers a unified treatment of functions classified by their dimensional signature.

The **ggfunction** package fills this gap by extending **ggplot2** with a principled taxonomy of function types, organized by input and output dimension. It provides two complementary families of geoms and stats:

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 of these layers follows the standard **ggplot2** extension pattern: a `Stat` object evaluates the user-supplied function and computes the data required for rendering, and a corresponding `Geom` object handles drawing. This architecture ensures that **ggfunction** layers compose naturally with the full ecosystem of **ggplot2** scales, coordinates, themes, and facets.

The remainder of this article is organized as follows. We first describe the design principles underlying the package and its relationship to the grammar of graphics. We then present the dimensional taxonomy, illustrating each function type with examples. Next, we develop the probability distribution family in detail, with particular attention to the region-shading capabilities that support statistical pedagogy. We conclude with remarks on implementation and future directions.

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 its domain and codomain dimensions. Let m and n denote the dimensions of the domain and codomain, respectively. The four classes are:

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

Each class admits a natural visual encoding:

- **Scalar functions** ($\mathbb{R} \to \mathbb{R}$) are rendered as curves in the Cartesian plane.
- **Parametric curves** ($\mathbb{R} \to \mathbb{R}^2$) are rendered as directed paths with optional arrowheads and color-mapped time.
- **Scalar fields** ($\mathbb{R}^2 \to \mathbb{R}$) are rendered as raster heatmaps, contour lines, or filled contour regions.
- **Vector fields** ($\mathbb{R}^2 \to \mathbb{R}^2$) are rendered as streamlines obtained by numerical integration.

This taxonomy is exhaustive for continuous, real-valued functions in two display dimensions. The naming convention adopted by the package—`geom_function_1d_1d()`, `geom_function_1d_2d()`, `geom_function_2d_1d()`, and `geom_function_2d_2d()`—encodes the dimensional signature directly, making the classification explicit at the point of use.

2.2 Integration with the grammar of graphics

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

Each **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 or sequence and produces a data frame of computed values. The `Geom` then renders those values according to the visual encoding appropriate for the function's type. Because the result is a standard **ggplot2** layer, it composes with `+` alongside other layers, scales, and themes.

A key design choice is the use of `rlang::inject()` (Henry and Wickham, 2024) to splice additional function arguments supplied through the `args` parameter. This mechanism allows users to parameterize functions without resorting to closures or anonymous wrappers:

```
# Instead of wrapping:
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 mapped to a Cartesian curve. The `geom_function_1d_1d()` layer generalizes `ggplot2::stat_function()` by adding support for shading a subinterval of the domain. The function is evaluated at n equally-spaced points (default $n = 101$) over the specified `xlim` range, and the resulting (x, y) pairs are rendered as a path.

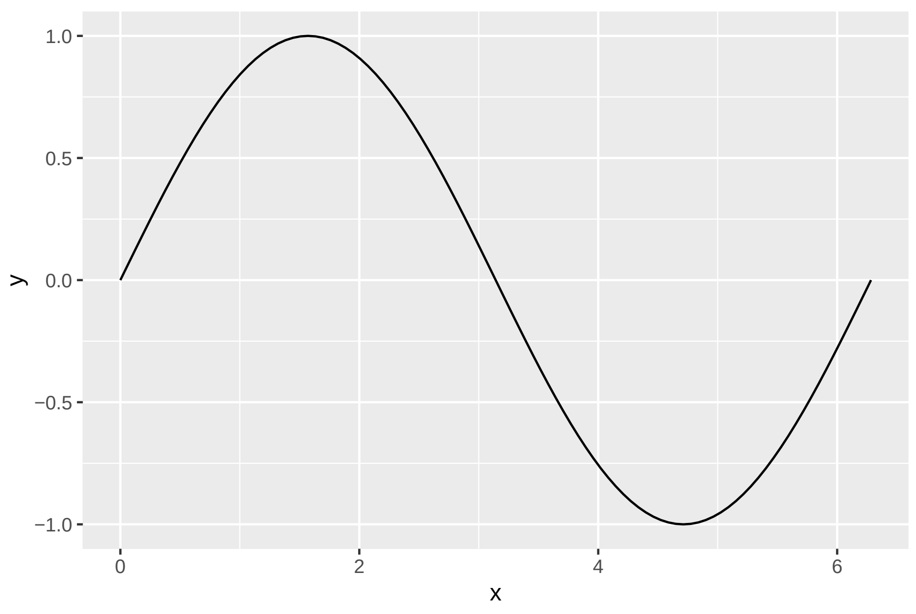

Figure 1: The sine function over one full period.

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

The shade_from and shade_to parameters specify an interval $[a, b]$ over which the region between the curve and the x-axis is filled. Boundary values are computed by linear interpolation via stats::approxfun(), ensuring that the shaded region aligns precisely with the requested interval even when a or b fall between grid points.

```
ggplot() +
  geom_function_1d_1d(
    fun = dnorm, xlim = c(-3, 3),
    shade_from = -1, shade_to = 1, fill = "steelblue"
  )
```

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. The geom_function_1d_2d() layer evaluates the user-supplied function at a sequence of time values from t0 to T with step size dt, producing a data frame with columns t, x, and y. The default aesthetic maps color to after_stat(t), providing a visual encoding of progression along the curve.

```
f_spiral <- function(t) c(sin(t), t * cos(t))
ggplot() +
  geom_function_1d_2d(fun = f_spiral, T = 20, tail_point = TRUE)
```

The tail_point parameter adds a marker at the starting position, useful for indicating the initial condition. An arrow specification (defaulting to a closed arrowhead) indicates the direction of traversal.

The args parameter supports parameterized curve families. For instance, the Lissajous figures $\gamma(t) = (A\sin(at + \delta),\ B\sin(bt))$ can be explored by varying the frequency and phase parameters:

```
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, T = 2 * pi, color = "black", arrow = NULL,
    args = list(A = 1, B = 1, a = 3, b = 2, delta = pi/2)
  )
```

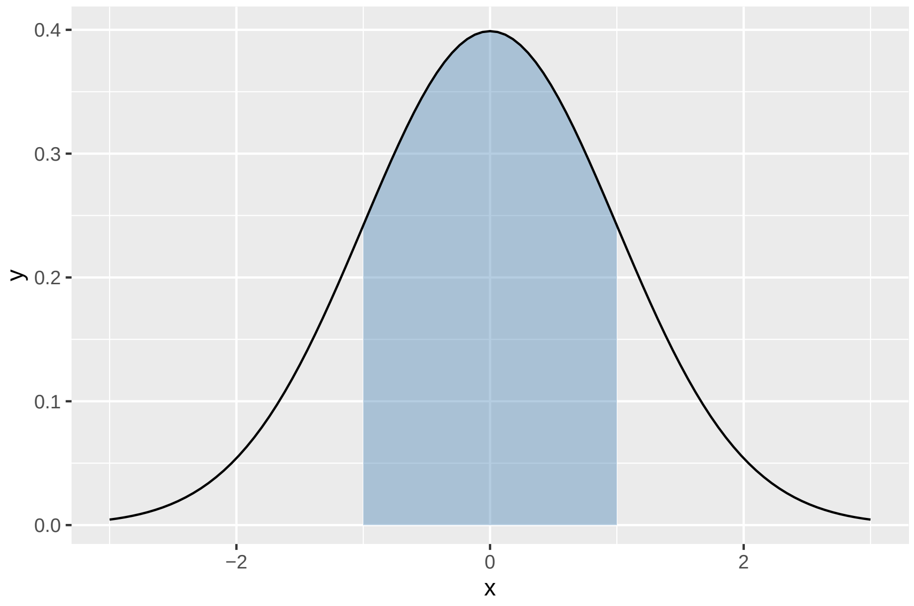

Figure 2: The standard normal density with the interval $[-1, 1]$ shaded, corresponding to approximately 68% of the total area.

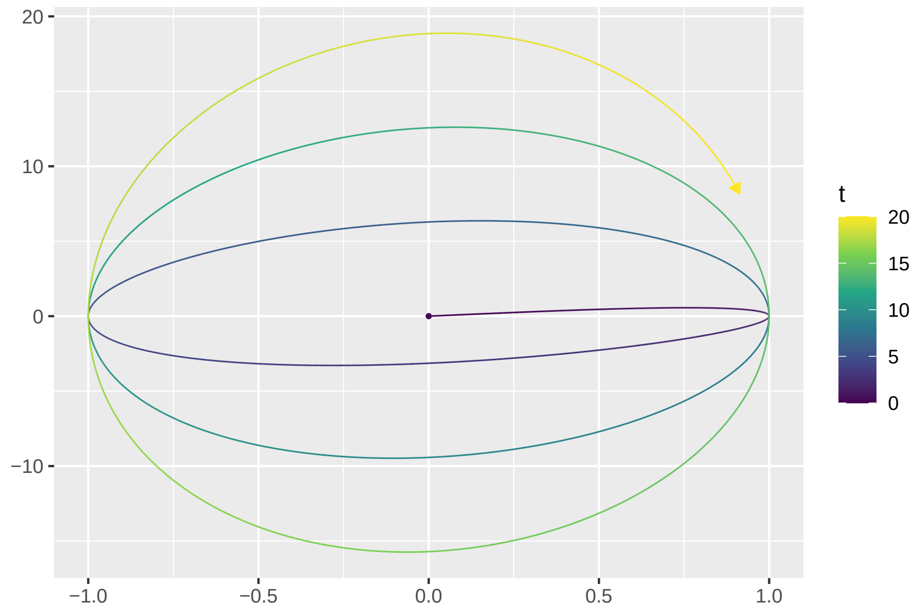

Figure 3: A parametric spiral defined by $\gamma(t) = (\sin t,\ t\cos t)$ for $t \in [0, 20]$, with color encoding the time parameter.

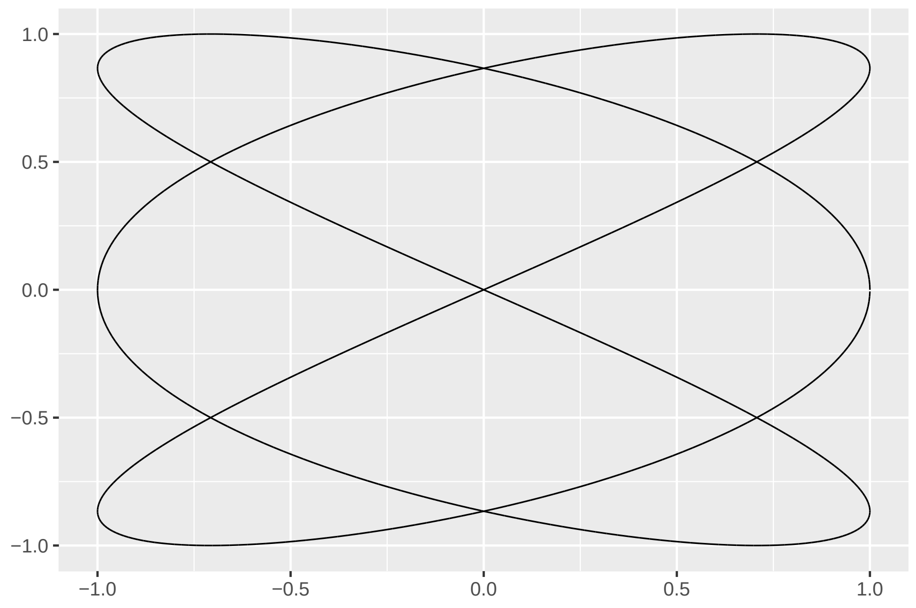

Figure 4: A Lissajous figure with frequency ratio $a/b = 3/2$ and phase offset $\delta = \pi/2$.

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. The `geom_function_2d_1d()` layer evaluates the user-supplied function on a regular $n \times n$ grid (default $n = 50$) over the specified `xlim` and `ylim` ranges. The function must accept a numeric vector of length two and return a scalar. Internally, the helper `vectorize()` applies the function row-wise over the grid matrix.

Three visualization modes are supported, controlled by the `type` parameter:

- `"raster"` (default): a heatmap with fill mapped to the computed values via `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(v) exp(-(v[1]^2 + v[2]^2) / 2)

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

```
library(gridExtra)
p_contour <- ggplot() +
  geom_function_2d_1d(
    fun = f_gaussian, xlim = c(-3, 3), ylim = c(-3, 3), type = "contour"
  ) + ggtitle("Contour lines")

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

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, then passes the grid forward via `setup_data()`.

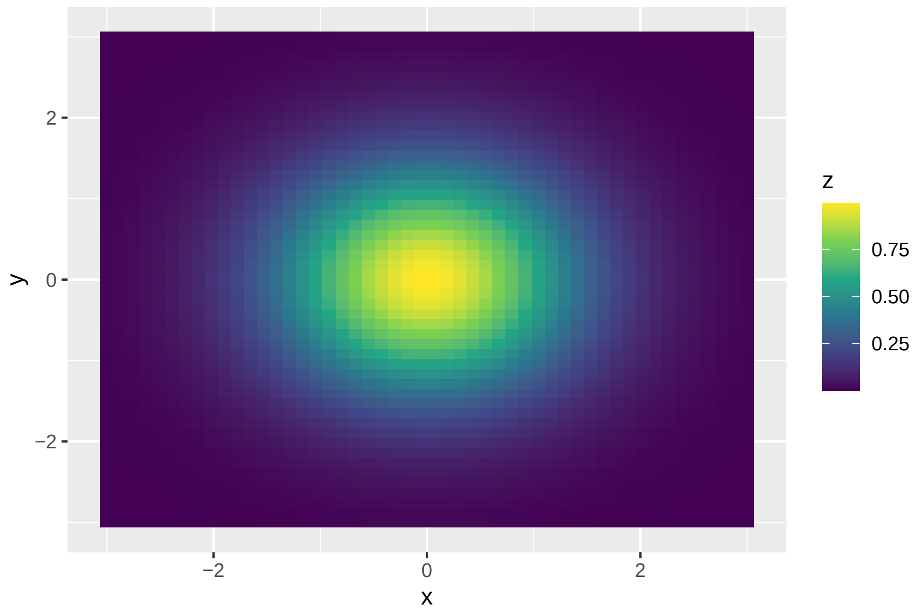

Figure 5: A Gaussian bump $f(x, y) = \exp(-(x^2 + y^2)/2)$ rendered as a raster heatmap.

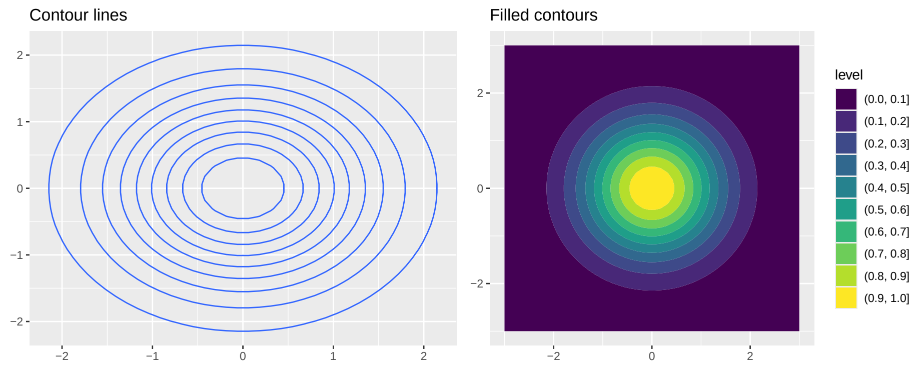

Figure 6: The same Gaussian bump rendered as contour lines (left) and filled contours (right).

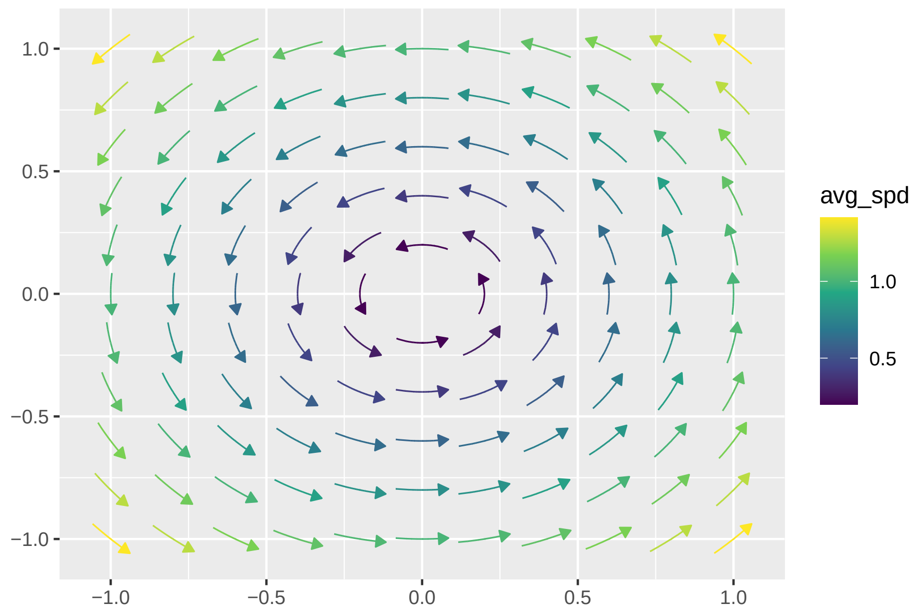

Figure 7: The rotation field $\mathbf{F}(x, y) = (-y, x)$ visualized as streamlines. Each curve follows the counterclockwise flow induced by the field.

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. The `geom_function_2d_2d()` layer 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 with arrowheads.

```
f_rotation <- function(v) c(-v[2], v[1])
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`.

4 The probability distribution family

Statistics education relies heavily on the visual language of probability distributions. Instructors routinely sketch densities, shade tail areas to illustrate p-values, and draw cumulative distribution functions to connect probabilities with quantiles. The **ggfunction** package provides a family of seven geoms that map directly onto the standard functions associated with a probability distribution.

4.1 Density functions: `geom_pdf()`

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

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

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

Three shading modes are supported, corresponding to common pedagogical operations:

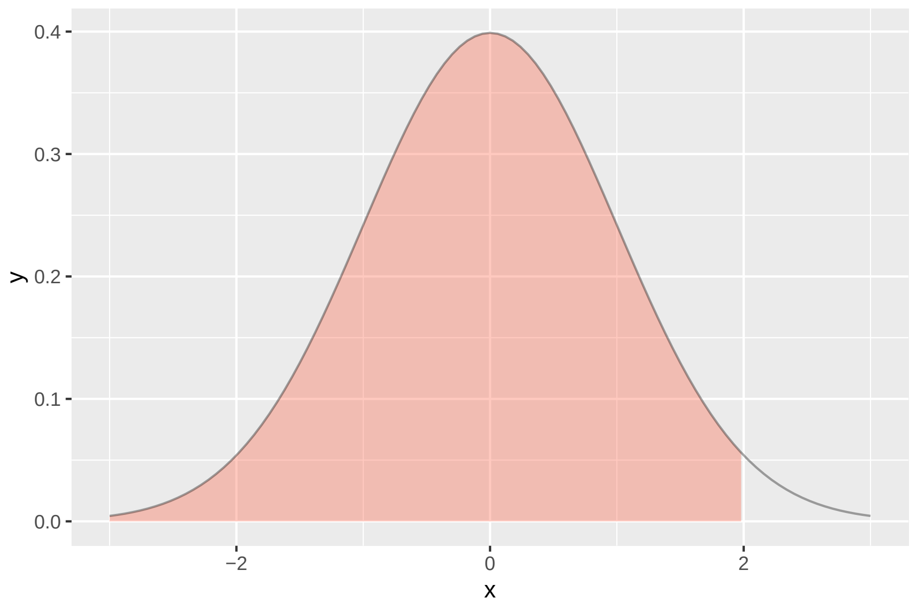

Figure 8: Standard normal density with the region corresponding to $P(X \leq x_{0.975})$ shaded.

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, fill = "tomato")
```

Two-sided interval. The `p_lower` and `p_upper` parameters define a central region. This is the natural representation for a $(1 - \alpha)$ confidence interval, where the shaded area corresponds to the middle $1 - \alpha$ of the distribution.

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

Tail shading. Setting `shade_outside = TRUE` inverts the two-sided region, shading both tails. This directly represents the rejection region in 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, fill = "tomato"
  )
```

The shading boundaries are determined by accumulating area under the density using the trapezoidal rule. For n evaluation points $x_1, \ldots, x_n$ with corresponding 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 cumulative values A_k / A_n are then compared against the specified probability thresholds to determine the shading boundaries.

4.2 Cumulative distribution functions: `geom_cdf()`

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

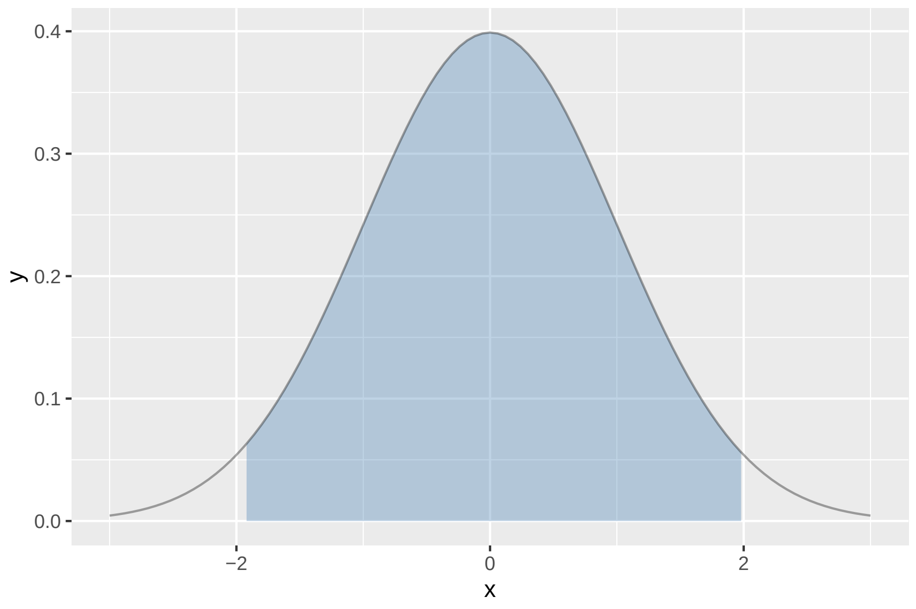

Figure 9: The central 95% of the standard normal density, corresponding to the interval between the 2.5th and 97.5th percentiles.

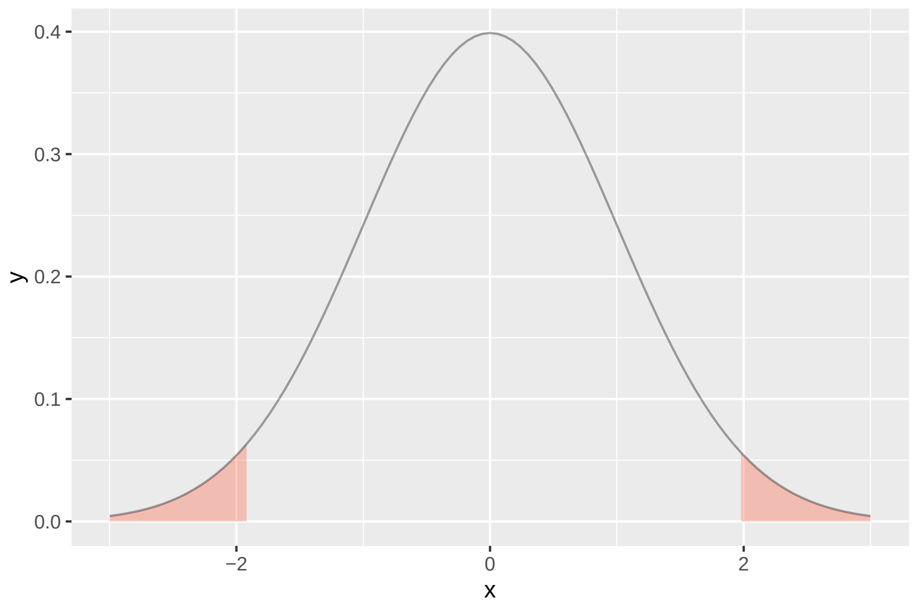

Figure 10: The rejection region of a two-sided test at $\alpha = 0.05$, shading the area outside the central 95%.

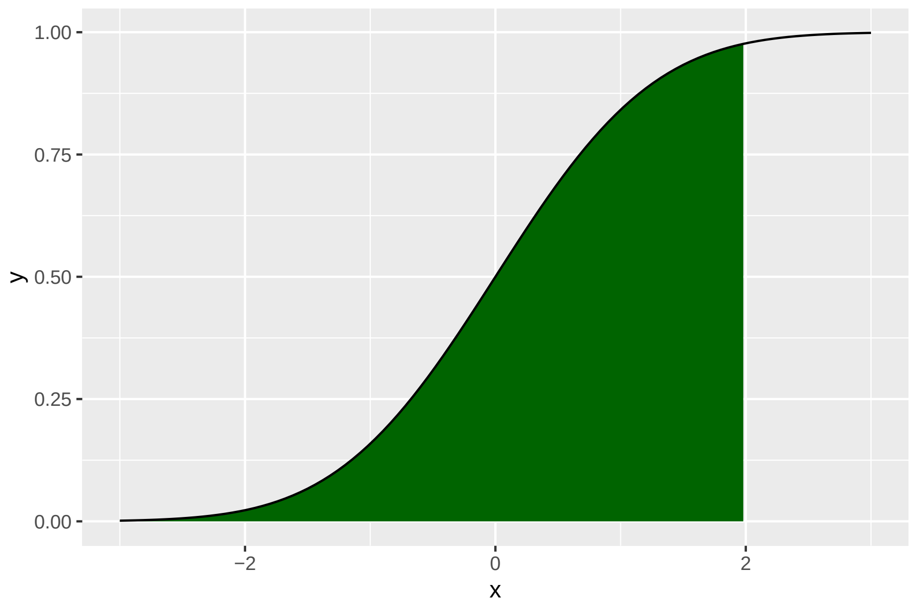

Figure 11: The standard normal CDF with the region below $p = 0.975$ shaded.

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

The `geom_cdf()` layer 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 probability levels.

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

4.3 Probability mass functions: `geom_pmf()`

For a discrete random variable with support on the integers, the probability mass function $p(k) = P(X = k)$ satisfies $\sum_k p(k) = 1$. The `geom_pmf()` layer evaluates the PMF at each integer in the specified `xlim` range and renders the result as a lollipop chart—vertical segments from the x-axis to the probability value, capped with points. The `StatPMF` object validates the normalization property by summing the computed probabilities.

```
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. The `geom_qf()` layer evaluates a user-supplied quantile function (e.g., `qnorm`) over the interval $(0, 1)$.

```
ggplot() +
  geom_qf(fun = qnorm, args = list(mean = 0, sd = 1))
```

4.5 Discrete cumulative distribution functions: `geom_discrete_cdf()`

For discrete distributions, the CDF is a step function. The `geom_discrete_cdf()` layer takes a PMF (not a CDF) as input, computes the cumulative sums, and renders the result using step-function vertices constructed by the internal `build_step_polygon()` helper. Like `geom_cdf()`, it supports `p`-based shading.

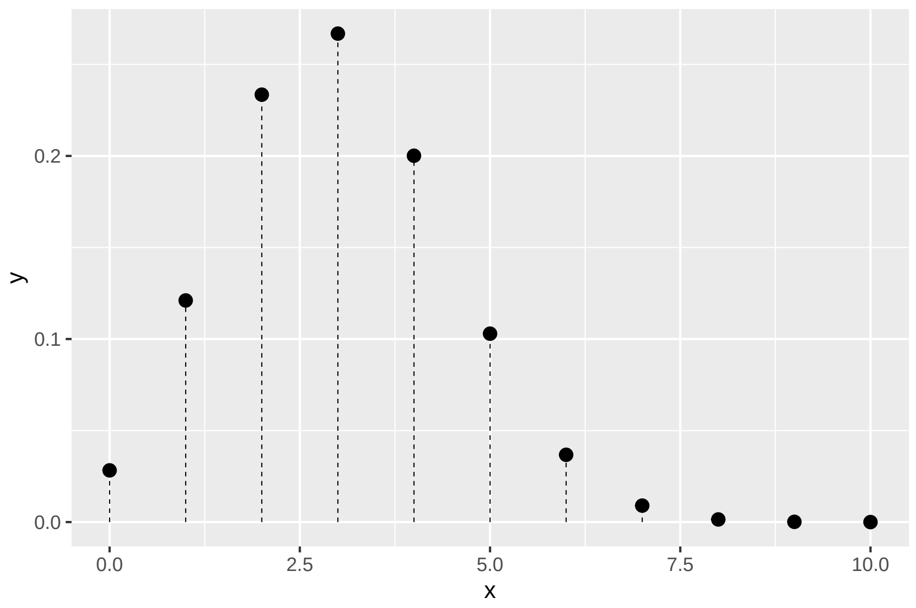

Figure 12: The PMF of a Binomial(10, 0.3) distribution, rendered as a lollipop chart.

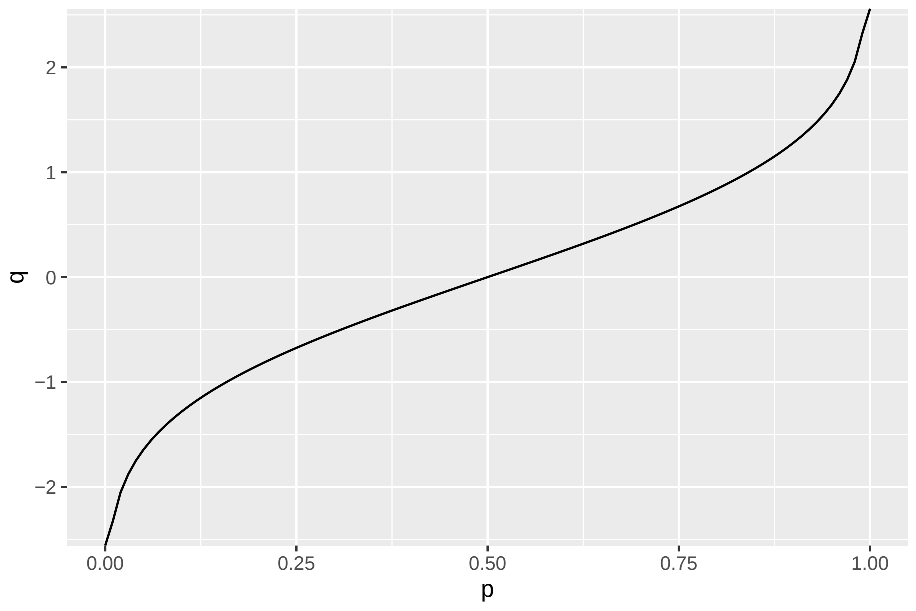

Figure 13: The quantile function of the standard normal distribution.

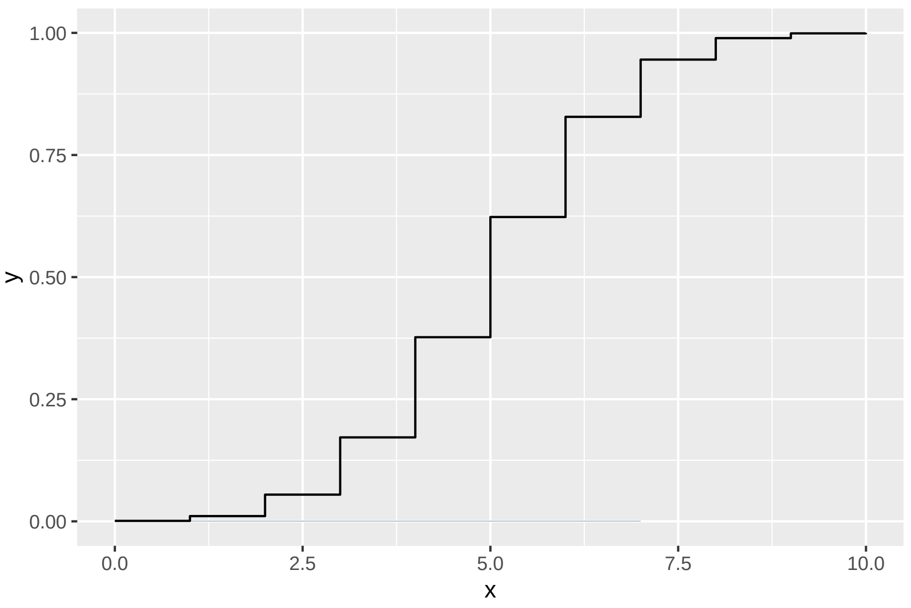

Figure 14: The discrete CDF of a Binomial(10, 0.5) distribution with the region below $p = 0.9$ shaded.

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

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. The `geom_survival()` layer accepts a CDF (e.g., `pexp`) 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). The `geom_hf()` layer requires both a PDF and a CDF, supplied via `pdf_fun` and `cdf_fun` respectively. The `args` parameter applies to both; where the two functions require different parameterizations, `pdf_args` and `cdf_args` provide overrides.

The hazard function involves division by $S(x)$, which approaches zero in the tail. The `StatHF` object guards against this by replacing values where $S(x) \leq 0$ with `NaN`, and the `GeomHF` renderer filters these before drawing.

```
ggplot() +
  geom_hf(
```

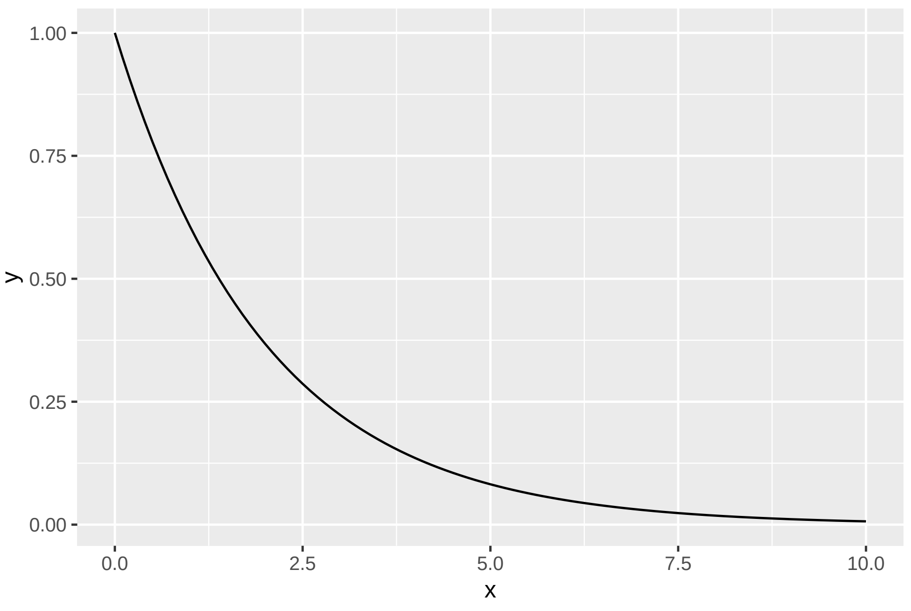

Figure 15: The survival function of an Exponential($\lambda = 0.5$) distribution, representing $S(x) = e^{-0.5x}$.

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

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

```
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, passing the user-supplied function and its parameters through to the stat's `compute_group()` method. This 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. For instance, the `StatFunction2d` object 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 standard mechanism for passing additional arguments to a function—the `...` (dots) argument—does not compose well when the function is called inside a stat's `compute_group()` method, which must also receive ggplot2-internal parameters. The **ggfunction** package adopts the approach used by `ggplot2::stat_function()`: additional arguments are collected in a named list (`args`) and injected at the call site using `rlang::inject()`:

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

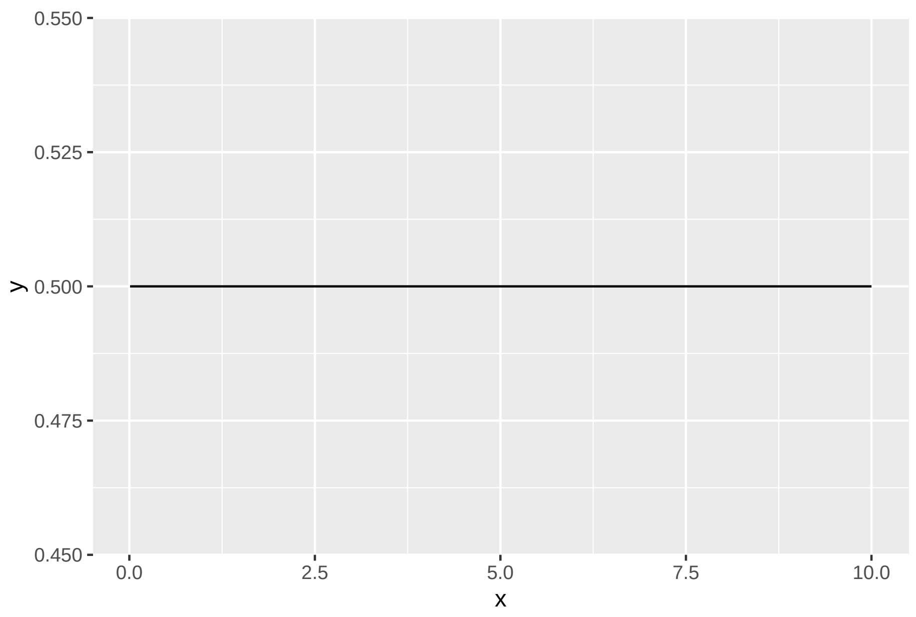

Figure 16: The hazard function of an Exponential($\lambda = 0.5$) distribution, which is constant at $h(x) = 0.5$—the memoryless property.

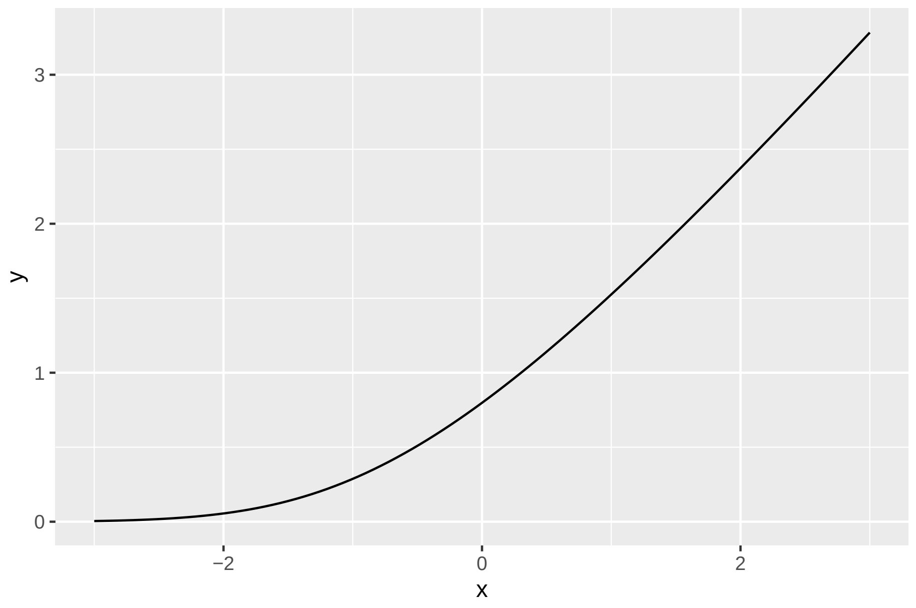

Figure 17: The hazard function of the standard normal distribution, illustrating the increasing failure rate.

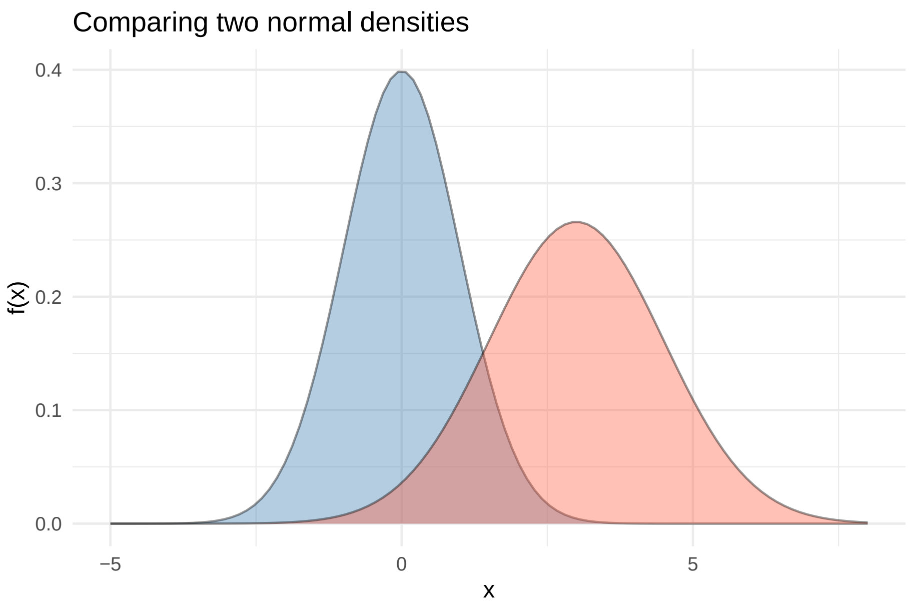

Figure 18: Two normal densities overlaid with custom colors, scales, and theme, demonstrating composability with the grammar of graphics.

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

5.3 Numerical considerations

Several layers involve numerical integration or accumulation:

- **PDF validation** uses `stats::integrate()` with adaptive quadrature to verify normalization.
- **Shading boundaries** are computed by trapezoidal accumulation (Equation (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 grid points.
- **Vector field integration** (via **ggvfields**) uses fourth-order Runge–Kutta 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) provides a good balance between visual fidelity and computational cost. Users may increase n for functions with fine-scale structure.

6 Composability and the grammar

Because each **ggfunction** layer is a standard **ggplot2** layer, it composes freely with the rest of the grammar. Users can overlay multiple functions, apply custom themes, adjust scales, and add annotations exactly as they would with any other geom:

```
ggplot() +
  geom_pdf(fun = dnorm, args = list(mean = 0, sd = 1),
           xlim = c(-5, 8), fill = "steelblue", alpha = 0.4) +
  geom_pdf(fun = dnorm, args = list(mean = 3, sd = 1.5),
           xlim = c(-5, 8), fill = "tomato", 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 framework, rather than providing standalone plotting functions.

7 Summary

The **ggfunction** package provides a principled extension of **ggplot2** 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.

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

8 Acknowledgments

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

References

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. [p7]

L. Henry and H. Wickham. *rlang: Functions for Base Types and Core R and Tidyverse Features*, 2024. URL `https://CRAN.R-project.org/package=rlang`. R package. [p2]

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]

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]

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]

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]

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`