cardinalR: Generating Interesting High-Dimensional Data Structures

by Jayani P. Gamage, Dianne Cook, Paul Harrison, Michael Lydeamore, and Thiyanga S. Talagala

Abstract Simulated high dimensional data is useful for testing, validating, and improving algorithms used in dimensionality reduction, supervised and unsupervised learning. High-dimensional data is characterized by multiple variables that are dependent or associated in some way, such as linear, nonlinear, clustering or anomalies. Here we provide new methods for generating a variety of high-dimensional structures using mathematical functions and statistical distributions organized into the R package cardinalR. Several example data sets are also provided. These will be useful for researchers to better understand how different analytical methods work and can be improved, with a special focus on nonlinear dimension reduction methods. This package enriches the existing toolset of benchmark datasets for evaluating algorithms.

1 Introduction

Generating synthetic datasets with clearly defined geometric properties is useful for evaluating and benchmarking algorithms in various fields, such as machine learning, data mining, and computational biology. Researchers often need to generate data with specific dimensions, noise characteristics, and complex underlying structures to test the performance and robustness of their methods. There are numerous packages available in R for generating synthetic data, each designed with unique characteristics and focus areas. The `geozoo` package (Schloerke, 2016) provides functions to generate standard high-dimensional data like cubes, spheres and simplexes, along with some prepared datasets. The `snedata` package (Melville, 2025) provides functions for generating common examples used in dimension reduction publications and to download benchmark data sets. The `splatter` package (Zappia et al., 2017) is designed to simulate complex biological data, capturing field-specific nuances such as batch effects and differential expression. The `mlbench` package (Leisch and Dimitriadou, 2024) provides access to benchmark datasets commonly associated with established classification or regression challenges. The `surreal` package (Balamuta, 2024) implements the "Residual (Sur)Realism" algorithm (Stefanski, 2007) to generate datasets that embed hidden images or text into residual plots, providing engaging visual demonstrations for teaching model diagnostics.

The current work implemented in the `cardinalR` R package builds on these approaches. It provides functions to generate a more extensive set of high-dimensional data structures, allowing users to: (i) construct high-dimensional datasets based on geometric shapes, including the option to enhance dimensionality by adding controlled noise dimensions; (ii) introduce adjustable levels of background noise to these structures; and (iii) combine the shapes to produce multiple clusters. The user can control characteristics such as number of dimensions, shape and sample size. It is designed to resource researchers with synthetic datasets to evaluate the performance and interpret the fit of NLDR methods, clustering algorithms, and visualization techniques. These datasets can also serve as benchmark examples for exploring how different choices of algorithm parameters affect the identification or representation of cluster and manifold structures in high-dimensional spaces.

The motivation for developing this package originated from our own work in studying nonlinear dimension reduction (NLDR) algorithms. We wanted to conduct a visualization experiment to understand perception and misperception of a variety of NLDR methods. This required simulated datasets with carefully controlled geometric and clustering properties. While some existing packages provided useful starting points, none fully supported the creation of flexible, high-dimensional data with the specific structural variations needed for our experiment. Developing these generators for research purposes underlies `cardinalR`, which is now a general-purpose package that should be useful for research and teaching.

The example data structures are best viewed using a tour (Asimov, 1985). These show the data as a sequence of low dimensional projections (typically 2D), providing a good sense of the shape in high dimensions. The interactive tour plots included in this paper are produced using the software langevitour (Harrison, 2023).

The paper is organized as follows. In the next section, we introduce the implementation of the `cardinalR` package on GitHub, including a demonstration of the package's key functions. We illustrate how a clustering data structure affects the dimension reductions in the Application section. Finally, we give a brief conclusion of the paper and discuss potential opportunities for the use of our data collection.

2 Usage

The `cardinalR` package is built on a modular framework where individual geometric generators (e.g., Gaussian, cone, sphere) create well-defined shapes (A full list of available shape generators are available at `https://jayanilakshika.github.io/cardinalR/reference/index.html`.), which can then be combined into a single dataset including scaling, rotation, and translation. The package is available on CRAN, and the source is available on GitHub at JayaniLakshika/cardinalR.

The main function, `gen_multicluster()`, is an all-in-one function that includes generating individual shapes, handling scaling and rotating of these shapes, and combining the result into a single unified dataset. This function and associated workflow allow flexible construction of complex, high-dimensional structures for evaluating clustering and dimension reduction methods. Figure 1 illustrates the workflow of `gen_multicluster()`.

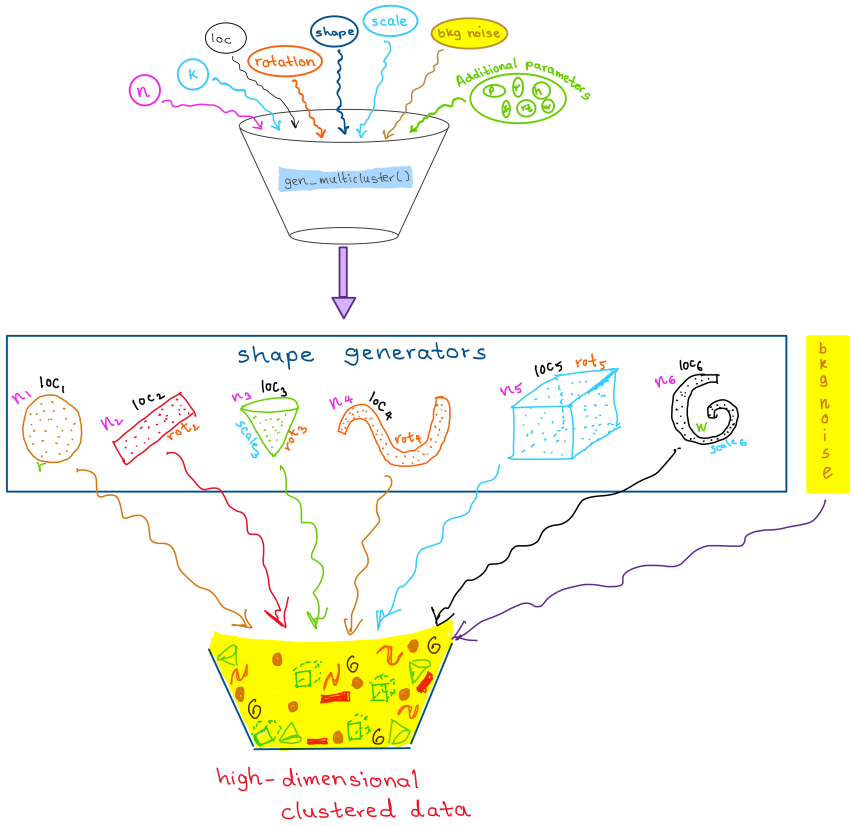


Figure 1: Workflow for generating high-dimensional clustered data. The user specifies input parameters such as the number of points (n), number of clusters (k), cluster locations, shapes, scaling, rotations, and optional background noise. Each cluster shape is generated by a shape generator, optionally rotated or scaled, and combined into a single dataset. Additional background noise can be added, and each observation is labeled by shape.

Users can control the number of clusters (`k`), and the number of points in each cluster (`n`). Each cluster can take on a different geometric shape (e.g., Gaussian, cone, uniform cube) by specifying the corresponding generator function (`shape`), can be scaled to adjust its spread, rotated using custom rotation matrices (`rotation`), and positioned at defined centroids (`loc`). The function ensures flexibility in cluster location and orientation, allowing users to simulate complex high-dimensional structures.

The following is an example of a three-shape multiclustered dataset. The first shape is Gaussian, the second conical, and the third a cube.

```
clust_data <- gen_multicluster(
  n = c(200, 300, 500),
  k = 3,
  loc = matrix(c(
    0, 0, 0, 0,
    5, 9, 0, 0,
    3, 4, 10, 7
```

Table 1: The main arguments for `gen_multicluster()`.

Argument	Type	Explanation
`n`	numeric (vector)	Number of points in each cluster.
`k`	numeric	Number of clusters.
`loc`	numeric (matrix)	Locations/centroids of clusters.
`scale`	numeric (vector)	Scaling factors of clusters.
`shape`	character (vector)	Shapes of clusters.
`rotation`	numeric (list)	Rotation matrices, one per cluster.
`is_bkg`	boolean	Background noise should exist or not.

```
  ), nrow = 3, byrow = TRUE),
  scale = c(3, 1, 2),
  shape = c("gaussian", "cone", "unifcube"),
  is_bkg = FALSE
)
```

Here, the shapes have 200, 300, and 500 points respectively (`n`), are positioned in 4-D space according to a location matrix, `loc`, and stretched according to the `scale`. The details of the individual shape generators and the noise elements are contained in the following sections.

3 Implementation

The main function of the package is `gen_multicluster()`, which generates datasets consisting of multiple clusters with user-specified characteristics. To maintain consistency across generators, the function identifies the arguments required by each chosen generator function and supplies only those arguments that are valid for that specific generator. This design enables the combination of cluster types with differing parameter requirements within the same dataset. When clusters are generated with fewer dimensions than others, the function augments the lower-dimensional clusters with additional Gaussian noise variables so that all clusters are represented in the same dimensional space. These noise dimensions are drawn independently from normal distributions

$$X \sim \mathcal{N}(m, s^2),$$

where the mean (m) is set to the average of the cluster coordinates and the standard deviation (s) defaults to 0.2.

An optional argument, `is_bkg`, adds background noise drawn from a multivariate normal distribution centered on the dataset's overall mean with standard deviations matching the observed spread. Extra arguments (`...`) can be passed to cluster generators, allowing further control over per-cluster characteristics like radius of the sphere. The main arguments of the `gen_multicluster()` function are shown in Table 1.

Shape generators

The shape generators form the foundation of the package, providing functions to create synthetic datasets from simple, well-defined geometric forms such as cones, pyramids, spheres, grids, and branching structures. Each generator includes the parameter `n`, which specifies the number of points to generate. Some functions, such as `gen_unifcube()`, also take the dimension `p`, while others include arguments specific to the geometry (e.g., radius for spheres (`r`), width for bands (`w`)). If higher-dimensional data are required, additional noise dimensions can be appended after data generation using any noise generator function. This flexibility allows users to construct both low- and high-dimensional datasets from the same underlying structures.

Table 2 outlines these functions. The main arguments of the functions described in Table 3.

Branching

A branching structure (Figure 2) captures trajectories that diverge or bifurcate from a common origin, similar to processes such as cell differentiation in biology (Trapnell et al., 2014). We introduce a set of

Table 2: cardinalR branching data generation functions

Function	Arguments	Explanation
`gen_expbranches`	`n, k`	Exponential shaped branches.
`gen_linearbranches`	`n, k`	Linear shaped branches.
`gen_curvybranches`	`n, k`	Curvy shaped branches.
`gen_orglinearbranches`	`n, p, k`	Linear shaped branches originated in one point.
`gen_orgcurvybranches`	`n, p, k`	Curvy shaped branches originated in one point.
`gen_cone`	`n, p, h, ratio`	Cone-shaped structure.
`gen_gridcube`	`n, p`	Cube with specified grid points along each axes.
`gen_unifcube`	`n, p`	Cube with uniform points.
`gen_gaussian`	`n, p, s`	Multivariate Gaussian cloud.
`gen_longlinear`	`n, p`	Long linear structure.
`gen_mobius`	`n`	Möbius strip in $3ext-D$.
`gen_quadratic`	`n`	Quadratic pattern in $2ext-D$.
`gen_cubic`	`n`	Cubic pattern in $2ext-D$.
`gen_pyrrect`	`n, p`	Rectangular-base, with a sharp or blunted apex.
`gen_pyrtri`	`n, p`	Triangular-base, with a sharp or blunted apex.
`gen_pyrstar`	`n, p`	Star-shaped base, with a sharp or blunted apex.
`gen_pyrfrac`	`n, p`	Pyramid with triangular pyramid-shaped holes.
`gen_scurve`	`n`	S-curve in $3ext-D$.
`gen_circle`	`n, p`	Circle.
`gen_curvycycle`	`n, p`	Curvy cell cycle.
`gen_unifsphere`	`n, r`	Uniform ball.
`gen_hollowsphere`	`n, p`	Hollow sphere.
`gen_gridedsphere`	`n`	Grided sphere.
`gen_clusteredspheres`	`n, k, r, loc`	Multiple small spheres within a big sphere.
`gen_hemisphere`	`n, p`	Hemisphere.
`gen_swissroll`	`n, w`	Swissroll structure.
`gen_trefoil4d`	`n, steps`	Trefoil in 4-D.
`gen_trefoil3d`	`n, steps`	Trefoil in 3-D.
`gen_crescent`	`n`	Crescent pattern.
`gen_curvycylinder`	`n, h`	Curvy cylinder.
`gen_sphericalspiral`	`n, spins`	Spherical spiral.
`gen_helicalspiral`	`n`	Helical spiral.
`gen_conicspiral`	`n, spins`	Conic spiral.
`gen_nonlinear`	`n, hc, non_fac`	Nonlinear hyperbola.

Table 3: The main arguments for branching shape generators.

Argument	Type	Explanation
`n`	integer	Number of points.
`k`	integer	Number of clusters.
`p`	integer	Number of dimensions.

data generation functions specifically designed to simulate high-dimensional branching structures with various geometries, numbers of points (n), and number of branches (k). Although these functions can generate multiple branches, they do not produce a formal *multicluster* dataset: the branches form a single connected structure, with multiple visually distinct arms rather than independent clusters.

The simplest structures are approximately linear branches in 2-D, generated by the `gen_linearbranches(n, k)` function. These consist of k short line segments in the first two dimensions, with added jitter to simulate variability. Mathematically, each branch i is defined as

$$X_1 \sim U(a_i, b_i), \quad X_2 = s_i(X_1 - x_{\text{start},i}) + y_{\text{start},i} + \epsilon, \quad \epsilon \sim U(0, \delta),$$

where $(x_{\text{start},i}, y_{\text{start},i})$ is the starting point of branch i, δ controls local jitter, and s_i is the slope, initialized as

$$s_i = \begin{cases} 0.5 & i = 1, \\ -0.5 & i = 2, \\ \text{randomly sampled from } [s_{\min}, s_{\max}] & i = 3, \ldots, k. \end{cases}$$

Branches 1 and 2 are initialized with fixed slopes and intercepts, while later branches are iteratively added at locations chosen to avoid overlap with existing branches, producing a set of connected linear paths.

To introduce curvature, the `gen_curvybranches(n, k)` function generates k curvilinear branches in 2-D. Branches 1 and 2 are simple parabolas defined as

$$\begin{aligned} &\text{Branch 1: } X_1 \sim U(0, 1), \quad X_2 = 0.1X_1 + X_1^2 + \epsilon, \\ &\text{Branch 2: } X_1 \sim U(-1, 0), \quad X_2 = 0.1X_1 - 2X_1^2 + \epsilon, \quad \epsilon \sim U(0, \delta), \end{aligned}$$

where δ controls local jitter. Additional branches are attached iteratively to existing structures. Each new branch i starts at a selected point $(x_{\text{start},i}, y_{\text{start},i})$ from the current structure and extends according to

$$X_1 \sim U(x_{\text{start},i}, x_{\text{start},i} + 1), \quad X_2 = 0.1X_1 - s_i(X_1^2 - x_{\text{start},i}) + y_{\text{start},i},$$

where s_i is a scale factor controlling the curvature of branch i. For the first few initial branches, s_i can be fixed (e.g., $s_1 = 1, s_2 = 2$), while for subsequent branches it is sampled from a predefined set, such as $s_i \in \{-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5\}$, to create variability in curvature.

The `gen_expbranches(n, k)` function creates k exponential branches in 2-D, radiating from a central region. Each branch i is defined as

$$X_1 \sim U(-2, 2), \quad X_2 = \exp(\sigma_i \, s_i \, X_1) + \epsilon, \quad \epsilon \sim U(0, \delta), \quad s_i \sim U(0.5, 2),$$

where $\sigma_i = (-1)^{i+1}$ alternates the sign of the exponent to produce mirror-symmetric branches. The parameter s_i controls the steepness of branch i, and δ introduces small local jitter.

High-dimensional generalizations are provided by `gen_orglinearbranches(n, p, k)` (Figure 2) and `gen_orgcurvybranches(n, p, k)`. Each branch is embedded in a unique or repeated 2-D subspace of the p-D space. When `allow_share = TRUE`, multiple branches may share the same subspace; otherwise, subspaces are sampled without replacement until all possible $\binom{p}{2}$ combinations are exhausted, after which additional branches may repeat subspaces. Linear branches follow

$$X_{i_1} \sim U(a_i, b_i), \quad X_{i_2} = s_i X_{i_1} + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

while curvilinear branches include a quadratic term

$$X_{i_1} \sim U(a_i, b_i), \quad X_{i_2} = -s_i X_{i_1}^2 + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

where a_i, b_i define the range of the first coordinate for branch i, and ϵ is Gaussian noise added to introduce variability. The scale factor s_i controls slope (linear branches) or curvature (curvilinear branches) and is assigned as follows: for the first $\binom{p}{2}$ branches, $s_i = 1$; for additional branches when $k > \binom{p}{2}$, s_i is randomly drawn from the set $\{1, 1.5, 2, \ldots, 8\}$.

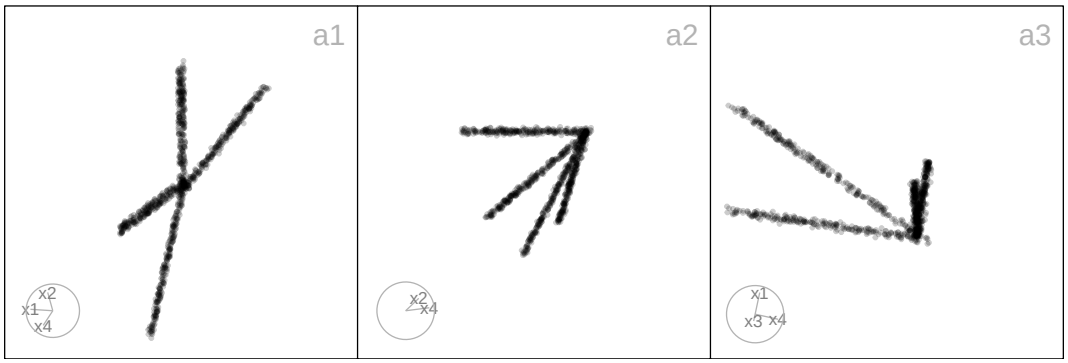


Figure 2: Three 2-D projections from the 4-D `orgcurvybranches` data. Each shows a different projection, illustrating how the linear branches appear from multiple viewing angles. These views highlight the dataset's underlying branching structure and demonstrate how projections reveal patterns that are otherwise hidden in higher dimensions.

Cone

To simulate a cone-shaped structure in arbitrary dimensions (Figure 3), we define a function `gen_cone(n, p, h, ratio)`, which creates a high-dimensional cone with options for a sharp or blunted apex, allowing for a dense concentration of points near the tip.

This function generates n points in p-D, where the last dimension, X_p, represents the height along the cone's axis, and the first $p-1$ dimensions define a shrinking hyperspherical cross-section toward the tip. Heights are sampled from a truncated exponential distribution, $X_p \sim \text{Exp}(\lambda = 2/h)$, capped at the cone height h, producing a higher density of points near the tip. At each height X_p, the radius of the cross-section decreases linearly from base to tip according to $r = r_{\text{min}} + (r_{\text{max}} - r_{\text{min}})X_p/h$, where $r_{\text{min}} = \text{ratio}$ and $r_{\text{max}} = 1$.

For each point, a direction in the first $p-1$ dimensions is sampled uniformly on a $(p-1)$-dimensional hypersphere using generalized spherical coordinates. The radial coordinates are scaled by the height-dependent radius r, producing the conical taper. In three dimensions ($p = 3$), this results in a classical 3-D cone, while for $p > 3$, additional dimensions provide a smooth embedding into higher-dimensional space, preserving the conical structure.

Cone-shaped structures appear in particle dispersions, light beams, and tapering processes, where spread decreases along one axis. They are also used to benchmark clustering and dimensionality reduction methods (Hadsell et al., 2006).

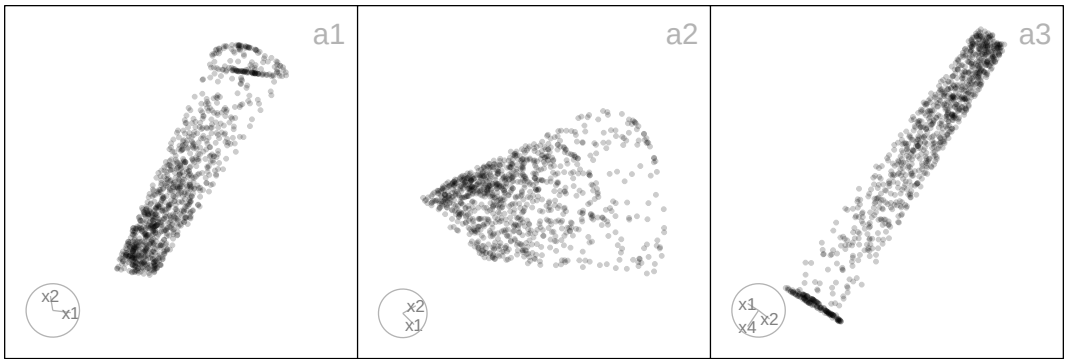


Figure 3: Three 2-D projections from the 3-D `cone` data. Points are concentrated near the tip along the height dimension, while the radius of the hyperspherical cross-section decreases linearly toward the apex. These projections show how the conical geometry is preserved.

Cube

A cube structure represents uniformly or systematically distributed points within a high-dimensional hypercube, providing a useful framework for assessing how well algorithms preserve uniformity, and boundary properties in high dimensions. We provide a set of functions to generate high-dimensional cube structures with flexible configurations, including regular grids, and uniform random points.

The function `gen_gridcube(n, p)` is a wrapper around `geozoo::cube.solid.grid()`. It generates a regular lattice of points in *p-D*, producing a uniform hypercube grid. Each axis contains equally spaced coordinates, resulting in a well-defined geometric structure.

By contrast, `gen_unifcube(n, p)` wraps `geozoo::cube.solid.random()`, producing uniformly distributed points within a *p-D* cube. To avoid including the cube's vertices, these points are removed after generation. This results in a hypercube filled with random, but evenly distributed, samples rather than structured lattice points.

Such cube-based structures are commonly used as benchmarks in Monte Carlo sampling, computational geometry, and density estimation, where assessing how algorithms behave under uniform or grid-like distributions is critical (Devroye, 1986; Niederreiter, 1992).

Gaussian

The `gen_gaussian(n, p, s)` function generates a multivariate Gaussian cloud in *p-D*, centered at the origin with user-defined covariance structure. Each point is independently drawn using the multivariate normal distribution with $X_i \sim N_p(\mathbf{0}, s)$, where s is a user-defined $p \times p$ positive-definite matrix.

Gaussian clouds are common benchmark structures in statistics and machine learning, used in clustering, classification, and anomaly detection, with applications in image segmentation, speech recognition, and forensic analysis (McLachlan and Peel, 2000).

Linear

The `gen_longlinear(n, p)` function generates a high-dimensional dataset representing a long linear structure with noise. Each variable is constructed by first creating a sequence of integers from 0 to $n-1$, adding Gaussian noise with standard deviation $0.03n$, and then applying a random scaling and shift. Specifically, for the j-th dimension, a scale factor $\text{scale}_j \sim U(-10, 10)$ determines the orientation and stretching of the line, while a shift factor $\text{shift}_j \sim U(-300, 300)$ is added inside the multiplication to offset the line. This results in each variable being of the form $X_j = \text{scale}_j \cdot \big((0, 1, \ldots, n-1) + \epsilon + \text{shift}_j\big)$, where $\epsilon \sim N(0, (0.03n)^2)$ is independent Gaussian noise.

This structure appears in *p-D* data when variation is driven by a single factor, such as time-course or sensor measurements, providing a useful test case for trajectory and regression methods (Trapnell et al., 2014).

Möbius

The `gen_mobius()` function is a wrapper around `geozoo::mobius()`, designed to simplify the generation of a Möbius strip in three dimensions for use in high-dimensional diagnostic studies. The function returns a tibble with n sampled points forming the surface of a Möbius strip.

The Möbius strip structure can model twisted or cyclic surfaces in physics and engineering, such as conveyor belts, molecular structures, or optical systems with non-orientable geometries (Optica – The Optical Society, 2023).

Polynomial

A polynomial structure generates data points that follow non-linear curvilinear relationships, such as quadratic or cubic trends, in space. To extend these patterns into high-dimensional settings, additional noise dimensions can be added. These patterns are useful for evaluating how well algorithms capture smooth, non-linear trajectories and curvature in the data. We provide functions for generating quadratic and cubic structures, enabling controlled experiments with different degrees of polynomial complexity.

The first is the quadratic curve of n points in two dimensions. This is generated using `gen_quadratic(n, range)`. The independent variable is defined as $X_1 \sim U(\text{range}[1], \text{range}[2])$, and a raw polynomial basis of degree 2 is applied to form $X_2 = X_1 - X_1^2 + \varepsilon_2$, where $\varepsilon_2 \sim U(0, 0.5)$. This produces a smooth parabolic arc opening downward, with vertical jitter introduced by the noise term.

The second is the cubic curve of n points in two dimensions. This is generated using `gen_cubic(n, range)`. The independent variable is defined as $X_1 \sim U(\text{range}[1], \text{range}[2])$, and a raw polynomial basis of degree 3 is applied to construct $X_2 = X_1 + X_1^2 - X_1^3 + \varepsilon_2$, where $\varepsilon_2 \sim U(0, 0.5)$. This produces

a more complex curvilinear structure than the quadratic case, with both upward and downward turning points.

Pyramid

A pyramid structure (Figure 4) represents data arranged around a central apex and base, useful for exploring how algorithms handle pointed or layered geometries in p-D space. The functions provided allow users to generate pyramids with rectangular, triangular, and star-shaped bases, and sharp or blunted apexes. Additionally, it is possible to create a pyramid with a fractal-like internal structure, enabling the study of non-convex and sparse regions.

Let $X_1, \ldots, X_p$ denote the coordinates of the generated points. For the rectangular, triangular, and star-shaped based pyramid generator functions, the final dimension, X_p, encodes the height of each point and is drawn from an exponential distribution capped at the maximum height h. That is, $X_p = z \sim \min\left(\text{Exp}(\lambda = 2/h),\ h\right)$. This distribution creates a natural skew toward smaller height values, resulting in a denser concentration of points near the pyramid's apex. For the star-shaped base pyramid, the final dimension is drawn from a uniform distribution. That is, $X_p = z \sim U(0, h)$.

The remaining dimensions are based on the specific pyramid shape. For the rectangular based pyramid, `gen_pyrrect(n, p, h, l_vec, rt)` (Figure 4 a), let $r_x(z)$ and $r_y(z)$ denote the half-widths of the rectangular cross-section at height z. That is, $r_x(z) = r_t + (l_x - r_t)z/h$, $r_y(z) = r_t + (l_y - r_t)z/h$. The first three coordinates are then defined as $X_1 \sim U(-r_x(z),\ r_x(z))$, $X_2 \sim U(-r_y(z),\ r_y(z))$, and $X_3 \sim U(-r_x(z),\ r_x(z))$.

For the triangular based pyramid, `gen_pyrtri(n, p, h, l, rt)` (Figure 4 b), let $r(z)$ denote the scaling factor (distance from the origin to triangle vertices) at height z. That is, $r(z) = r_t + (l - r_t)z/h$. A point in the triangle at height z is generated using barycentric coordinates (u, v) to ensure uniform sampling within the triangular cross-section: $u, v \sim U(0, 1)$, if $u + v > 1 : u \leftarrow 1 - u,\ v \leftarrow 1 - v$. The first three coordinates (triangle plane) are then: $X_1 = r(z)(1 - u - v)$, $X_2 = r(z)u$, and $X_3 = r(z)v$.

For the star based pyramid, `gen_pyrstar(n, p, h, rb)` (Figure 4 c), let the radius at height z, $r(z)$, be such that the radius scales linearly from zero (tip) to the base radius r_b. That is, $r(z) = r_b\,(1 - z/h)$. Each point is placed within a regular hexagon in the plane (X_1, X_2), using a randomly chosen hexagon sector angle $\theta \in \{0, \pi/3, 2\pi/3, \pi, 4\pi/3, 5\pi/3\}$ and a uniformly random radial scaling factor: $\theta \sim$ Uniform sample from 6 hexagon angles, $r_{\text{point}} \sim \sqrt{U(0, 1)}$. Then, the first two coordinates are: $X_1 = r(z)r_{\text{point}}\cos(\theta)$, and $X_2 = r(z)r_{\text{point}}\sin(\theta)$.

For rectangular and triangular pyramids, the remaining dimensions X_4 to X_{p-1}, and for star-based pyramids X_3 to X_{p-1}, are treated as noise.

Finally, for the Sierpinski-like pyramid, `gen_pyrfrac(n, p)` (Figure 4 d), let $X_1, X_2, \ldots, X_p$ denote the coordinates of the generated points. The generation process begins with an initial point $T_0 \in [0, 1]^p$ drawn from a uniform distribution: $T_0 \sim U(0, 1)^p$. Let $C_1, C_2, \ldots, C_{p+1}$ denote the corner vertices of a p-D simplex. At each iteration $i = 1, \ldots, n$, a new point is computed by taking the midpoint between the previous point T_{i-1} and a randomly selected vertex C_k: $T_i = 1/2(T_{i-1} + C_k)$, $C_k \in \{C_1, \ldots, C_{p+1}\}$. This recursive midpoint rule generates self-similar patterns with systematic voids (holes) between clusters of points. The points remain bounded inside the convex hull of the simplex. The final output is a $n \times p$ matrix where each row represents a point: $X = \{T_1, T_2, \ldots, T_n\}$, $X \in \mathbb{R}^{n \times p}$.

Pyramid structures mimic tapering or layered geometries seen in architecture, crystals, and fractal-like natural patterns (Mandelbrot, 1983).

S-curve

The S-curve is a smooth, non-linear manifold in 3-D space. Using `gen_scurve(n)`, it is defined by $X_1 = \sin(\theta)$, $X_2 \sim U(0, 2)$, $X_3 = \text{sign}(\theta)(\cos(\theta) - 1)$, $\theta \sim U(-3\pi/2, 3\pi/2)$.

This follows the `s_curve()` function from snedata (Melville, 2025), itself adapted from *scikit-learn*, but differs by returning a tibble with standardized names (`x1`, `x2`, `x3`), excluding the color variable, and omitting built-in noise (which can be added separately). S-curve is commonly used in manifold learning and dimension reduction as benchmarks for unfolding curved structure.

Sphere

Sphere-shaped structures are useful for evaluating how dimension reduction and clustering algorithms handle curved, symmetric manifolds in high-dimensional spaces. Throughout this section, we follow the standard mathematical terminology: a *sphere* refers to the hollow $(p - 1)$-dimensional surface in

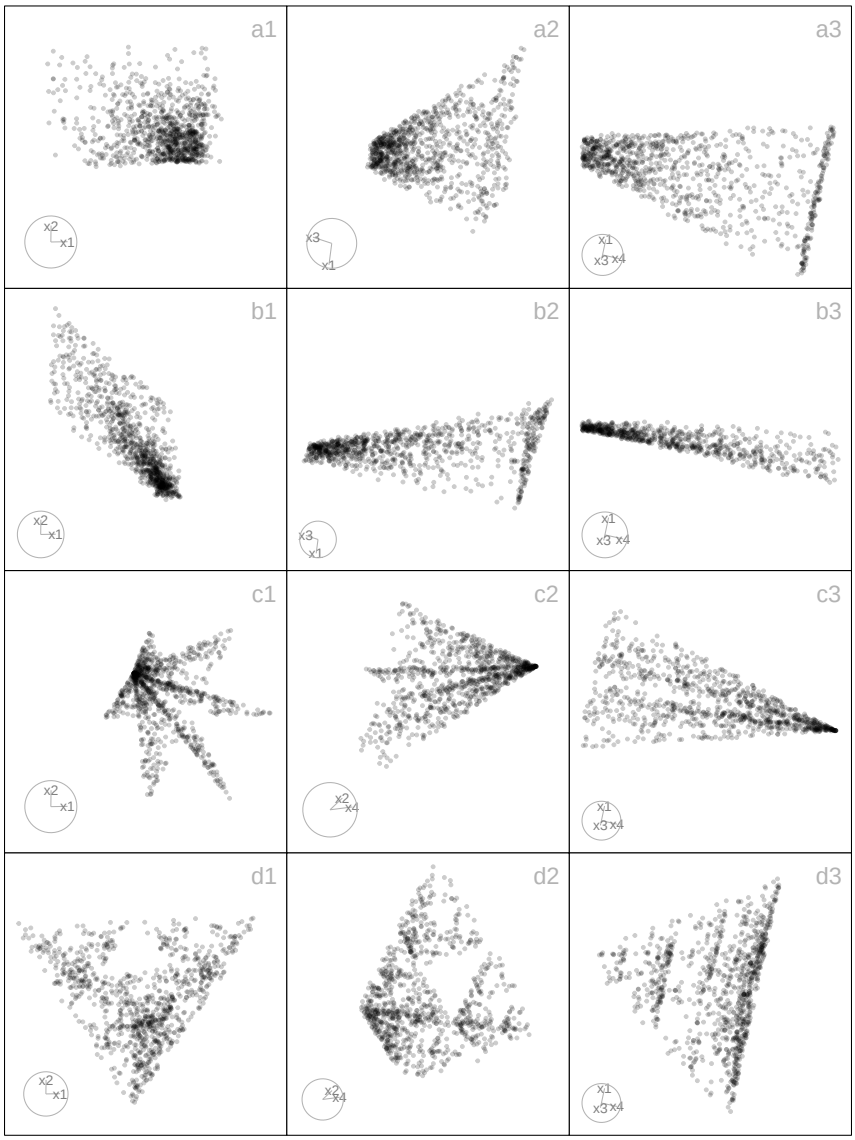


Figure 4: Three 2-*D* projections from 4-*D*, for the `pyrrect` (a1-a3), `pyrtri` (b1-b3), `pyrstar` (c1-c3), and `pyrholes` (d1-d3) data. The `pyrrrect` structure forms a dense rectangular base tapering to a narrow tip, while `pytri` shows a more triangular spread with sharper edges. `pyrstar` extends into multiple pointed branches radiating from a common core, and `pyrholes` reveals hollow or open regions within an otherwise compact shape. These projections illustrate a range of pyramid-like geometries that vary in density and structure.

$\mathbb{R}^p$, while a *ball* refers to the filled interior region. The functions generate a variety of spherical forms, including simple circles, uniform and hollow spheres, grid-based spheres, and complex arrangements like clustered spheres within a larger sphere. The first few coordinates define the main geometric form (circle, cycle, sphere, or hemisphere), while higher-dimensional embeddings are achieved by adding noise dimensions. Such spherical or hemispherical structures frequently appear in physical and biological systems, for example in models of celestial bodies, molecular shells, or cell membranes (Tinkham, 2003; Alberts et al., 2014).

The simplest case, `gen_circle(n, p)` creates a unit circle in two dimensions, with the remaining dimensions forming sinusoidal extensions of the angular parameter at progressively smaller scales (Figure 5 a). Let a latent angle variable θ is uniformly sampled from the interval $[0, 2\pi]$. Coordinates in the first two dimensions represent a perfect circle on the plane:

$$X_1 = \cos(\theta), \quad X_2 = \sin(\theta).$$

For dimensions X_3 through X_p, sinusoidal transformations of the angle θ are introduced. The first component is a scaling factor that decreases with the dimension index, defined as $\text{scale}_j = \sqrt{(0.5)^{j-2}}$

for $j = 3, \ldots, p$. The second component is a phase shift that is proportional to the dimension index, specifically designed to decorrelate the curves, given by the formula $\phi_j = (j-2)\pi/2p$. Each additional dimension is computed as: $X_j = \text{scale}_j \sin(\theta + \phi_j), \quad j = 3, \ldots, p$.

For the one-dimensional nonlinear cycle embedded in p-D space, `gen_curvycycle(n, p)` (Figure 5 b), let a latent angle variable θ is uniformly sampled from the interval $[0, 2\pi]$. The first three dimensions define a non-circular closed curve, referred to as a "curvy cycle". In this configuration, $X_1 = \cos(\theta)$ represents horizontal oscillation, while $X_2 = \sqrt{3}/3 + \sin(\theta)$ introduces a vertical offset to avoid centering the curve at the origin. Additionally, $X_3 = 1/3\cos(3\theta)$ introduces a third harmonic perturbation that intricately folds the curve three times along its path, creating a unique and complex shape that oscillates in both dimensions while incorporating the effects of the harmonic perturbation.

Together, these define a periodic, non-trivial, closed curve in 3-D with internal folds that produce a more complex geometry than a standard circle or ellipse. For dimensions X_4 through X_p, additional structured variability is introduced through decreasing amplitude scaling and phase-shifted sine waves. The scaling factor is defined as $\text{scale}_j = \sqrt{(0.5)^{j-3}}$ for j ranging from 4 to p, which means that the amplitude decreases as the dimension increases. Each dimension X_j is then calculated using the formula $X_j = \text{scale}_j \sin(\theta + \phi_j)$, where the phase shift ϕ_j is given by $\phi_j = (j-2)\pi/2p$.

Building on simple circular structures, the `gen_unifsphere(n, r)` function function extends the idea to three dimensions by generating n observations approximately uniformly distributed on the surface of a sphere of radius r.. Each observation is computed from spherical coordinates, with $u \sim U(-1, 1)$ representing $\cos(\phi)$ and $\theta \sim U(0, 2\pi)$ the azimuthal angle. Cartesian coordinates are then defined as

$$X_1 = r\sqrt{1-u^2}\cos(\theta), \quad X_2 = r\sqrt{1-u^2}\sin(\theta), \text{ and } X_3 = ru,$$

ensuring uniform distribution on the surface (not within) of the sphere.

In contrast, the `gen_hollowsphere(n, p)` function, a wrapper around `geozoo::sphere.hollow()`, generates n points uniformly distributed only on the surface of the $(p-1)$-dimensional sphere embedded in $\mathbb{R}^p$. This results in a hollow shell-like structure with no interior points. For example, when $p = 3$, `gen_unifsphere()` produces a solid ball in 3-D space, whereas `gen_hollowsphere()` produces only the spherical boundary. These paired structures allow controlled experiments to investigate how algorithms behave when data is concentrated throughout the full volume versus constrained to the boundary.

In addition, the `gen_gridedsphere(n)` function constructs a p-dimensional dataset consisting of approximately n points arranged on the surface of the unit $(p-1)$-sphere embedded in $\mathbb{R}^p$ (Figure 5 d). Rather than sampling points uniformly, this function creates a deterministic grid in spherical coordinates, using $(p-1)$ angular variables: the first $(p-2)$ angles are taken from $[0, \pi]$, and the final angle from $[0, 2\pi]$. The number of grid points along each angular dimension is determined by decomposing n into $(p-1)$ approximately equal integer factors via `gen_nproduct(n, p - 1)`.

Each grid point is subsequently mapped into Cartesian space via the standard hyperspherical-to-Cartesian transformation,

$$\begin{aligned}
X_1 &= \cos(\theta_1), \\
X_2 &= \sin(\theta_1)\cos(\theta_2), \\
X_3 &= \sin(\theta_1)\sin(\theta_2)\cos(\theta_3), \\
&\vdots \\
X_{p-1} &= \sin(\theta_1)\sin(\theta_2)\cdots\sin(\theta_{p-2})\cos(\theta_{p-1}), \\
X_p &= \sin(\theta_1)\sin(\theta_2)\cdots\sin(\theta_{p-2})\sin(\theta_{p-1}).
\end{aligned}$$

The result is a deterministic grid of points lying exactly on the surface of the unit $(p-1)$-sphere, without any additional noise dimensions.

For more heterogeneous structures, the `gen_clusteredspheres(n, k, r, loc)` function generates one large sphere of radius r_1 and k smaller spheres of radius r_2, each centered at a different random location (Figure 5 e). A large Uniform ball centered at the origin is created by sampling n_1 points uniformly on the surface of a p-D sphere with a radius of r_1. The sampling is executed using the function `gen_unifsphere(n_1, r_1)`, which generates the desired points in the specified dimensional space. In generation of k smaller Uniform balls, each sphere contains n_2 points that are sampled uniformly on a sphere with a radius of r_2. These spheres are positioned at distinct random locations in p-space, with the center of each sphere being drawn from a normal distribution $N(0, \texttt{loc}^2 I_p)$. Points on spheres are generated using the standard hyperspherical method, which involves sampling $u \sim U(-1, 1)$ to determine the cosine of the polar angle, and sampling $\theta \sim U(0, 2\pi)$ to determine the

azimuthal angle (for 3-D). Each observation is classified by cluster, with labels such as "big" for the large central sphere and "small_1" to "small_k" for the smaller spheres.

Finally, the `gen_hemisphere(n, p)` function restricts sampling to a hemisphere of a 4-D sphere (Figure 5 f). Using spherical coordinates, the azimuthal angle $\theta_1 \sim U(0, \pi)$ in the (x_1, x_2) plane, while the elevation angle $\theta_2 \sim U(0, \pi)$ in the (x_2, x_3) plane. Additionally, $\theta_3 \sim U(0, \pi/2)$ in the (x_3, x_4) plane, ensuring that the points remain restricted to a hemisphere. The coordinates are transformed into 4-D Cartesian space:

$$X_1 = \sin(\theta_1)\cos(\theta_2), \quad X_2 = \sin(\theta_1)\sin(\theta_2), quad X_3 = \cos(\theta_1)\cos(\theta_3), \quad X_4 = \cos(\theta_1)\sin(\theta_3).$$

This produces points on one side of a 4-D unit sphere, effectively generating a 4-D hemisphere.

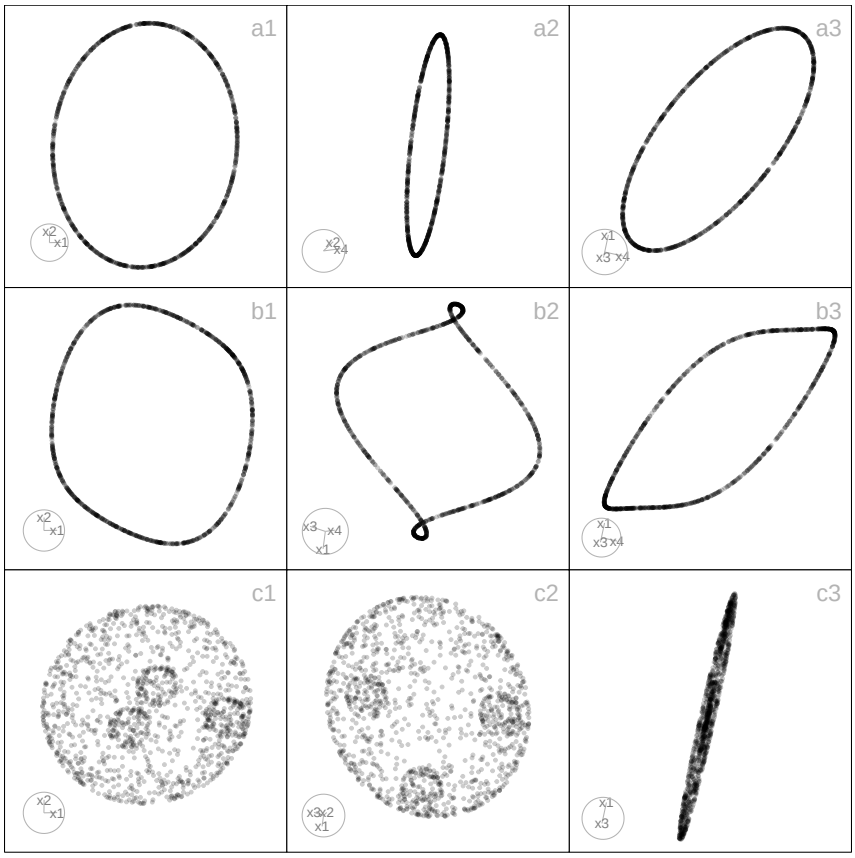


Figure 5: Three 2-D projections from 4-D, `circle` (a1-a3), `curvycycle` (b1-b3), and, 3-D `clusteredspheres` (c1-c3). The `circle` structure forms a smooth, closed loop, while `curvycycle` shows a wavy, continuous pattern forming a twisted ring. The `clusteredspheres` dataset displays multiple compact spherical groups that are clearly separated in higher dimensions but overlap slightly in some 2-D projections, highlighting how projection can distort spatial relationships. These projections show how simple cyclic, wavy curvilinear, and clustered structures appear in 2-D, emphasizing the effects of projection on density, continuity, and separation

Swiss Roll

The Swiss roll is a plane curled into 3-D, and is a commonly used example of a nonlinear manifold. The `gen_swissroll(n, w)` generates points as $X_1 = t\cos(t), \quad X_2 = t\sin(t), \quad X_3 \sim U(w_1, w_2), \quad t \sim U(0, 3\pi)$.

Compared with `snedata::swiss_roll()` (Melville, 2025), this implementation (i) samples t over $[0, 3\pi]$ instead of $[1.5\pi, 4.5\pi]$, (ii) allows a flexible vertical range $w = (w_1, w_2)$ rather than fixing $z \in [0, z_{\max}]$, and (iii) returns a tibble with `x1, x2, x3` instead of adding a color variable.

The Swiss roll is a classic benchmark for manifold learning, illustrating how a curved surface can be "unrolled" into lower dimensions. Similar spiral-like forms appear in galaxies, protein folding, and coiled materials (Agrafiotis and Xu, 2002).

Trefoil knots

The Trefoil is a closed, nontrivial one-dimensional manifold embedded in 3-D or 4-D space (Figure 6). The trefoil features topological complexity in the form of self-overlaps, making it a valuable test case for evaluating the ability of non-linear dimension reduction methods to preserve global structure, loops, and embeddings in high-dimensional data.

For the 4-D trefoil knot (Laurent, 2024), the function `gen_trefoil4d(n, steps)` generates the structure on the 3-sphere ($S^3 \subset \mathbb{R}^4$) using two angular parameters, θ and ϕ. A band of thickness around the knot path is controlled by the `steps` argument, while the number of θ and ϕ values is determined by the `steps` and `n` arguments, respectively (Figure 6 a). The coordinates are defined as

$$X_1 = \cos(\theta)\cos(\phi), \quad X_2 = \cos(\theta)\sin(\phi), \quad X_3 = \sin(\theta)\cos(1.5\phi), \text{ and } X_4 = \sin(\theta)\sin(1.5\phi),$$

where θ and ϕ trace the knot's path.

For the 3-D stereographic projection (Laurent, 2024), `gen_trefoil3d(n, steps)` maps each point $(X_1, X_2, X_3, X_4) \in \mathbb{R}^4$ to $(X_1', X_2', X_3') \in \mathbb{R}^3$ using $X_1' = X_1/(1 - X_4)$, $X_2' = X_2/(1 - X_4)$, and $X_3' = X_3/(1 - X_4)$, excluding points where $X_4 = 1$ to avoid division by zero (Figure 6 b).

The trefoil knot appears in molecular biology (DNA/protein knotting), fluid dynamics (knotted vortices), and physics (topological phases), making it a useful benchmark for testing whether dimension reduction preserves global loops and topology (Witten, 1985; Arsuaga et al., 2002).

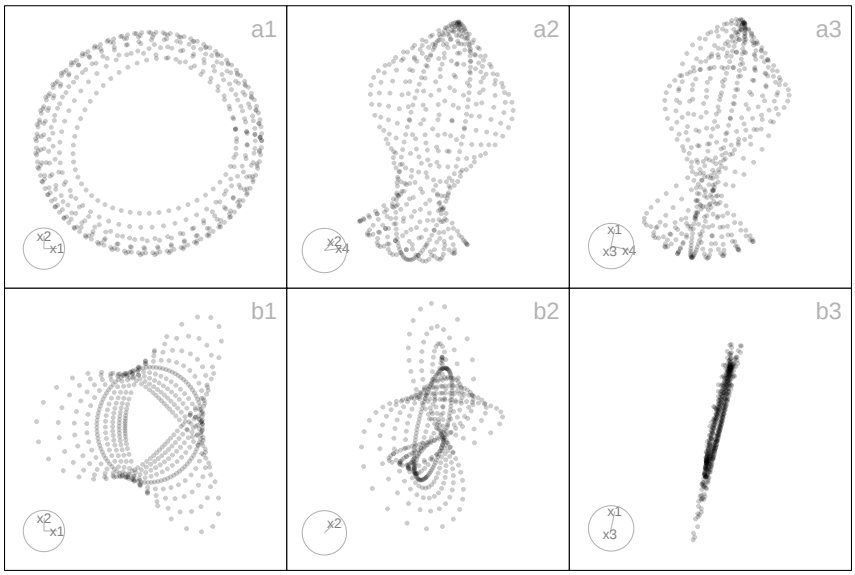


Figure 6: Three 2-D projections from 4-D, `trefoil4d` (a1-a3) and 3-D `trefoil3d` (b1-b3) data. The `trefoil4d` structure represents a higher-dimensional extension of the classic trefoil knot, revealing complex twisting and looping patterns that remain continuous across projections. In contrast, the `trefoil3d` dataset maintains a simpler, more compact knot-like form, showing how dimensional extension adds curvature and separation in the embedded space. These projections illustrate a range of looping structures in high-dimensions.

Trigonometric

Trigonometric-based structures provide flexible ways to simulate complex curved patterns and spirals that often arise in real-world high-dimensional data, such as in biological trajectories, or physical systems (Figure 7). The main geometry is defined by the first few coordinates: crescents ($p = 2$), cylinders, spirals, and helices ($p = 4$). These structures are particularly valuable for testing how well dimension reduction and clustering algorithms preserve intricate geometric and topological features (Calladine et al., 1997; Gershenfeld, 2000).

First, the `gen_crescent(n, p)` function generates a p-dimensional dataset of n observations based on a 2-D crescent-shaped manifold with optional structured high-dimensional noise (Figure 7 a). Let $\theta \in [\pi/6, 2\pi]$ be a sequence of n evenly spaced angles. The corresponding 2-D coordinates are defined by:

$$X_1 = \cos(\theta), \quad X_2 = \sin(\theta).$$

Second, the `gen_curvycylinder(n, p, h)` function generates a p-dimensional dataset of n observations structured as a 3-D cylindrical manifold with an added nonlinear curvy dimension, and optional noise dimensions when $p > 4$ (Figure 7 b). The core structure consists of a circular base and height values, extended by a nonlinear fourth dimension. Let $\theta \sim U(0, 3\pi)$ represent a random angle on a circular base and $z \sim U(0, h)$ represent the height along the cylinder. The coordinates are defined as: $X_1 = \cos(\theta)$ (Circular base, x-axis), $X_2 = \sin(\theta)$ (Circular base, y-axis), $X_3 = z$ (Linear height), and $X_4 = \sin(z)$ (Nonlinear curvy variation along height).

For a spiraling path on a spherical surface in the first four dimensions, `gen_sphericalspiral(n, p, spins)` (Figure 7 c), let $\theta \in [0, 2\pi \times \text{spins}]$ be the azimuthal angle (longitude), controls the number of spiral turns and the $\phi \in [0, \pi]$be the polar angle (latitude), controls the vertical sweep from the north to the south pole. Cartesian coordinates from spherical conversion: $X_1 = \sin(\phi)\cos(\theta)$, $X_2 = \sin(\phi)\sin(\theta)$, $X_3 = \cos(\phi) + \varepsilon$, where $\varepsilon \sim U(-0.5, 0.5)$ introduces vertical jitter, and $X_4 = \theta / \max(\theta)$: a normalized progression along the spiral path. This generates a spherical spiral curve embedded in 4-D space, combining both circular and vertical movement, with gentle curvature and non-linear progression.

For a helical spiral in four dimensions, `gen_helicalspiral(n, p)` (Figure 7 d), let $\theta \in [0, 5\pi/4]$ be a sequence of angles controlling rotation around a circle. Cartesian coordinates; $X_1 = \cos(\theta)$: circular trajectory along the x-axis, $X_2 = \sin(\theta)$: circular trajectory along the y-axis, $X_3 = 0.05\theta + \varepsilon_3$, with $\varepsilon_3 \sim U(-0.5, 0.5)$: linear progression (height) with vertical jitter, simulating a helix, and $X_4 = 0.1\sin(\theta)$: oscillates with θ, representing a periodic "wobble" along the fourth dimension.

Similarly, the `gen_conicspiral(n, p, spins)` function generates a dataset of n points forming a conical spiral in the first four dimensions of p-D (Figure 7 e). The geometry combines radial expansion, vertical elevation, and spiral deformation, simulating a structure that fans out like a 3-D conic helix. The shape is defined by parameter $\theta \in [0, 2\pi\text{spins}]$, controlling the angular progression of the spiral. The Archimedean spiral in the horizontal plane is represented by; $X_1 = \theta\cos(\theta)$ for radial expansion in x, and $X_2 = \theta\sin(\theta)$ for radial expansion in y. The growth pattern resembles a cone, with the height increasing according to $X_3 = 2\theta / \max(\theta) + \varepsilon_3$, with $\varepsilon_3 \sim U(-0.1, 0.6)$. Spiral modulation in the fourth dimension is represented by $X_4 = \theta\sin(2\theta) + \varepsilon_4$, with $\varepsilon_4 \sim U(-0.1, 0.6)$ which simulates a twisting helical component in a non-radial dimension.

Finally, the `gen_nonlinear(n, p, hc, non_fac)` function simulates a non-linear 2-D surface embedded in higher dimensions, constructed using inverse and trigonometric transformations applied to independent variables (Figure 7 f). The $X_1 \sim U(0.1, 2)$: base variable (avoids zero to prevent division errors), $X_3 \sim U(0.1, 0.8)$: independent auxiliary variable, $X_2 = hc/X_1 + \text{nonfac}\sin(X_1)$: non-linear combination of hyperbolic and sinusoidal transformations, creating sharp curvature and oscillation, and $X_4 = \cos(\pi X_1) + \varepsilon$, with $\varepsilon \sim U(-0.1, 0.1)$: additional nonlinear variation based on cosine, simulating more subtle periodic structure. These transformations together result in a non-linear surface warped in multiple ways: sharp vertical shifts due to inverse terms, smooth waves from sine and cosine, and additional jitter.

Generate a spherical or hyperspherical hole within a structure

The package provides functionality for generating datasets with spherical hole (in 2-D/3-D) or, more generally, hyperspherical hole (in higher dimensions). These structures are valuable for evaluating how dimension reduction methods and clustering algorithms handle incomplete manifolds or missing regions of the data space. A hyperspherical hole introduces topological complexity: the structure remains continuous but contains excluded regions (voids) that algorithms must correctly represent in lower-dimensional embeddings.

The core function `gen_hole(df, anchor, r)` removes points from a dataset that fall within a user-specified hypersphere. Formally, given data points ($x \in \mathbb{R}^p$), a center ($a \in \mathbb{R}^p$), and radius ($r > 0$), only points satisfying $\sqrt{\sum_{i=1}^{n}(x_i - a_i)^2} > r$ are retained. The anchor point (a) can either be user-specified or default to the dataset mean, and radius (r) is controlled by the user, with safeguards to avoid trivial or degenerate cases. Because it operates generically on any dataset, spherical or hyperspherical holes can be embedded in a wide range of geometric structures.

Two specialized wrappers illustrate this idea. The function `gen_scurvehole(n, r_hole)` generates an S-curve with a spherical hole by applying `gen_hole()` to the output of `gen_scurve()`. This structure has been used in prior diagnostic studies of NLDR methods (Tenenbaum et al., 2000, van der Maaten et al. (2007)), since it tests the ability of algorithms to capture non-linear manifolds that are not simply connected. The second wrapper, `gen_unifcubehole(n, p, r_hole)`, generates uniformly sampled cube data with a hyperspherical hole. By embedding a hyperspherical void inside a convex high-dimensional structure, this creates non-convex regions that challenge algorithms in terms of separability and neighborhood preservation.

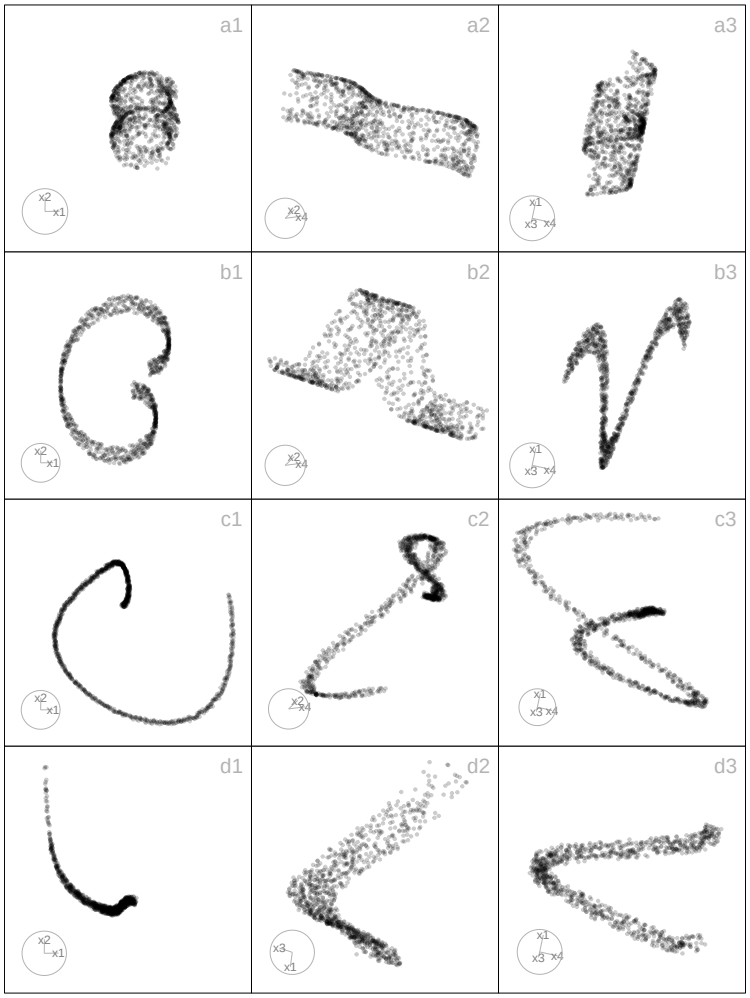


Figure 7: Three 2-*D* projections from 4-*D*, for the `curvycylinder` (a1-a3), `sphericalspiral` (b1-b3), `conicspiral` (c1-c3), and `nonlinear` (d1-d3) data. The `curvycylinder` shows a cylindrical manifold with a nonlinear twist along its height, producing smooth, continuous curvature. The `sphericalspiral` forms a spiral path on a spherical surface, combining circular and vertical motion in a helical form. The `conicspiral` spreads radially while ascending, forming a conical helix with twisting variations in a non-radial dimension. The `nonlinear` dataset exhibits a warped 2-*D* surface with sharp oscillations and smooth waves, reflecting complex nonlinear interactions. Each shows variations in curvature, density, and continuity.

Generate noise dimensions

High-dimensional data structures often benefit from the addition of auxiliary noise dimensions, which can be used to assess the robustness of dimensionality reduction and clustering algorithms. The functions in this section provide flexible ways to generate random noise dimensions, ranging from purely random Gaussian variables to more structured, wavy patterns that mimic non-linear distortions in high-dimensional space. These functions can be applied independently or combined with other geometric structures to create complex simulated datasets. Table 4 details these functions.

The `gen_noisedims(n, p, m, s)` function generates p independent Gaussian noise dimensions,

$$X_j \sim N(m_j, s_j^2), \quad j = 1, \ldots, p,$$

with odd-numbered dimensions multiplied by -1 to introduce sign alternation, enhancing variability and decorrelation.

For scenarios where noise should follow a smooth wavy pattern, `gen_wavydims1(n, p, theta)` generates dimensions as

$$X_j = \alpha_j \theta + \varepsilon_j, \quad \varepsilon_j \sim N(0, \sigma^2), \quad j = 1, \ldots, p,$$

Table 4: cardinalR noise dimensions generation functions

Function	Explanation
`gen_noisedims`	Gaussian noise dimensions with optional mean and standard deviation.
`gen_wavydims1`	Wavy noise dimensions based on a user-specified theta sequence with added jitter.
`gen_wavydims2`	Wavy noise dimensions using polynomial transformations of an existing dimension vector.
`gen_wavydims3`	Wavy noise dimensions using a combination of polynomial and sine transformations based on the first three dimensions of a dataset.

where each dimension is scaled by a different factor α_j, producing structured noise that oscillates along the latent parameter θ, mimicking trends or trajectories observed in real-world data.

The `gen_wavydims2(n, p, x_1)` function extends this approach by applying a non-linear transformation to an existing dimension vector x_1:

$$X_j = \beta_j \, (-1)^{\lfloor j/2 \rfloor} \, x_1^{k_j} + \varepsilon_j, \quad j = 1, \ldots, p,$$

where k_j is a randomly chosen polynomial power, β_j is a scaling factor, and ε_j is small uniform noise.

Finally, `gen_wavydims3(n, p, data)` generates noise for datasets with multiple correlated dimensions. The first three dimensions are small perturbations of the original coordinates (X_1, X_2, X_3), while higher dimensions are constructed via non-linear combinations, including polynomial and trigonometric transformations, e.g.,

$$X_j = f_j(X_1, X_2, X_3) + \varepsilon_j, \quad j > 3,$$

producing high-dimensional noise that preserves some geometric correlation with the base structure while introducing additional complexity.

Rotating shape generators

In *p-D* space, a rotation is an orthogonal transformation that changes the orientation of data while preserving its total variance and pairwise distances. The function `gen_rotation()` generates such rotation matrices for any dimension, given a list of rotation planes (axis pairs) and angles.

Multiple cluster examples

By using the shape generators mentioned above, we can create various examples of multiple clusters. The package includes some of these examples, which are described in Table 5.

Additional functions

The package includes various supplementary tools in addition to the shape generating functions mentioned earlier. These tools allow users to create background noise, randomize the rows of the data, relocate clusters, generate a vector whose product and sum are approximately equal to a target value, rotate structures, and normalize the data. Table 6 details these functions. More detailed explanations are available in `https://jayanilakshika.github.io/cardinalR/articles/03additionalfun.html`.

4 Application

This section demonstrates how the package can be used to generate complex high-dimensional datasets, evaluate dimension reduction (DR) and clustering methods. The example shows how diverse geometric structures can be simulated and analyzed to assess algorithmic behavior.

Table 5: cardinalR multiple clusters generation functions

Function	Explanation
`make_mobiusgau`	Möbius-like cluster combined with a Gaussian.
`make_multigau`	Multiple Gaussian clusters in high-dimensional space.
`make_curvygau`	Curvilinear cluster with a Gaussian cluster.
`make_klink_circles`	K-link circular clusters (non-linear circular patterns).
`make_chain_circles`	Chain-like circular clusters connected sequentially.
`make_klink_curvycycle`	K-link curvy cycle clusters (curvilinear loop structures).
`make_chain_curvycycle`	Chain-like curvy cycle clusters connected sequentially.
`make_gaucircles`	Circular clusters with a Gaussian cluster in the middle.
`make_gaucurvycycle`	Curvy circular clusters with a Gaussian in the middle.
`make_onegrid`	Single grid in two dimensions.
`make_twogrid_overlap`	Two overlapping grids.
`make_twogrid_shift`	Two grids shifted relative to each other.
`make_shape_para`	Parallel shaped clusters.

Table 6: cardinalR additional functions

Function	Explanation
`gen_bkgnoise`	Adds background noise.
`randomize_rows`	Randomizes the rows of input data.
`relocate_clusters`	Relocates the clusters.
`gen_nproduct`	Generates a vector of positive integers whose product is approximately equal to a target value.
`gen_nsum`	Generates a vector of positive integers whose summation is approximately equal to a target value.
`normalize_data`	Normalizes data.

Generating high-dimensional clustered data

To illustrate how high-dimensional clustered data can be generated using `cardinalR`, we generate a dataset with five clusters in 4-*D*, each representing distinct geometric characteristics: a *helical spiral* (elongated and twisted), a *hemisphere* (curved surface), a *uniform cube* (isotropic distribution), a *cone* (density gradient), and a *Gaussian* cluster (compact and spherical) (Figure 8). Each cluster has a unique number of points and scaling factor, representing variation in cluster size and spread across the 4-*D* space.

```
positions <- geozoo::simplex(p=4)$points
positions <- positions * 0.3

five_clusts <- gen_multicluster(n = c(2250, 1500, 750, 1250, 1750), k = 5,
                      loc = positions,
                      scale = c(0.25, 0.35, 0.3, 1, 0.3),
                      shape = c("helicalspiral", "hemisphere", "unifcube",
                                "cone", "gaussian"),
                      rotation = NULL,
                      is_bkg = FALSE)
```

Evaluating dimension reduction (DR) methods

We applied six popular DR techniques to the generated dataset: Principal Component Analysis (PCA) (Jolliffe, 2011), tSNE, uniform manifold approximation and projection (UMAP) (McInnes et al., 2018), potential of heat-diffusion for affinity-based trajectory embedding (PHATE) algorithm (Moon et al., 2019), large-scale dimensionality reduction Using triplets (TriMAP) (Amid and Warmuth, 2019), and pairwise controlled manifold approximation (PaCMAP) (Wang et al., 2021).

To assess their performance, we computed the hexbin error (HBE) between the observed high-dimensional data and the fitted values, defined as the high-dimensional mappings of the bin centroids

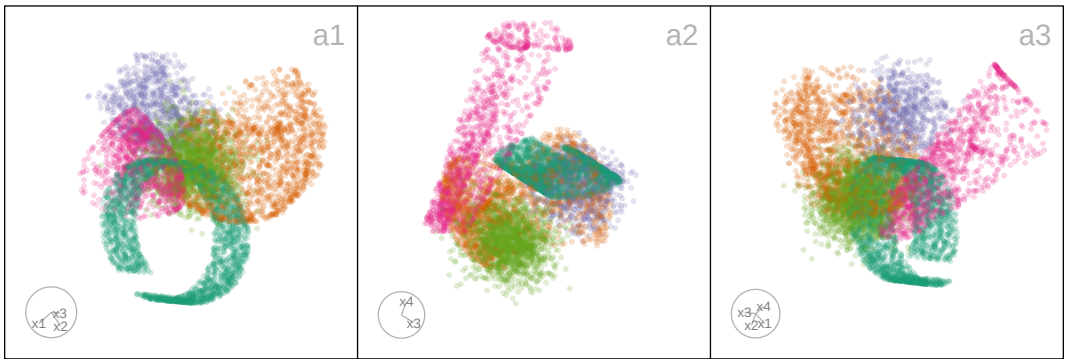


Figure 8: Three 2-*D* projections from 4-*D*, for the five clusters data. The helical spiral cluster is represented in dark green, the hemisphere cluster in orange, the uniform cube-shaped cluster in purple, the blunted cone cluster in pink, and the Gaussian-shaped cluster in light green.

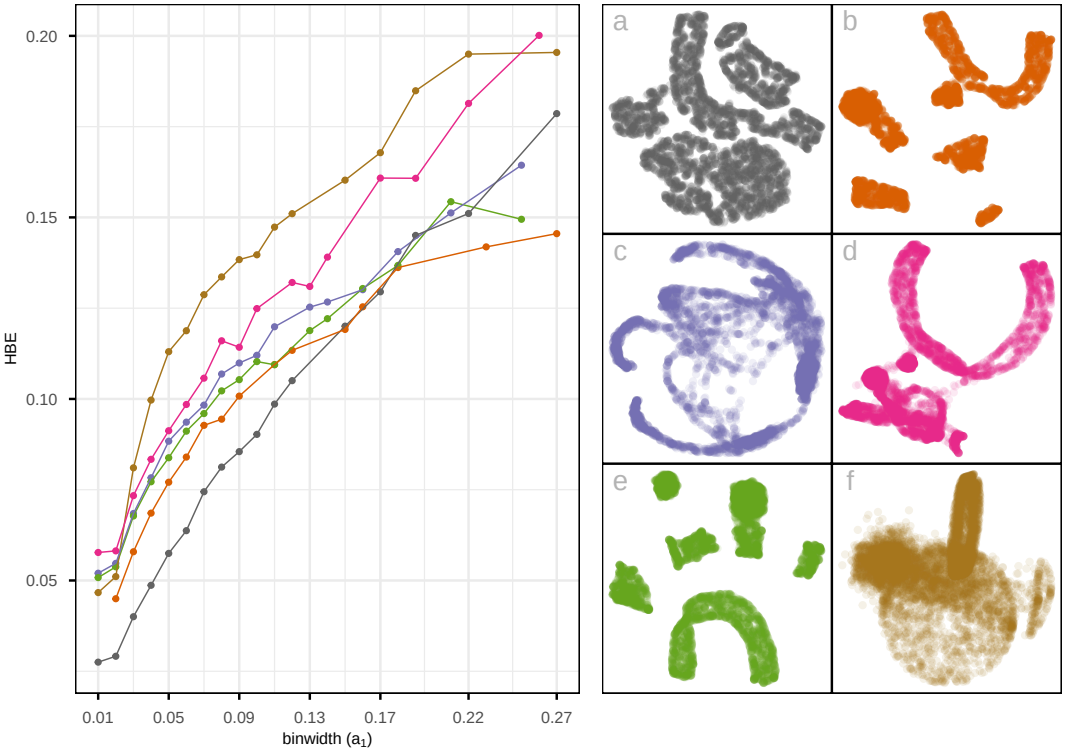


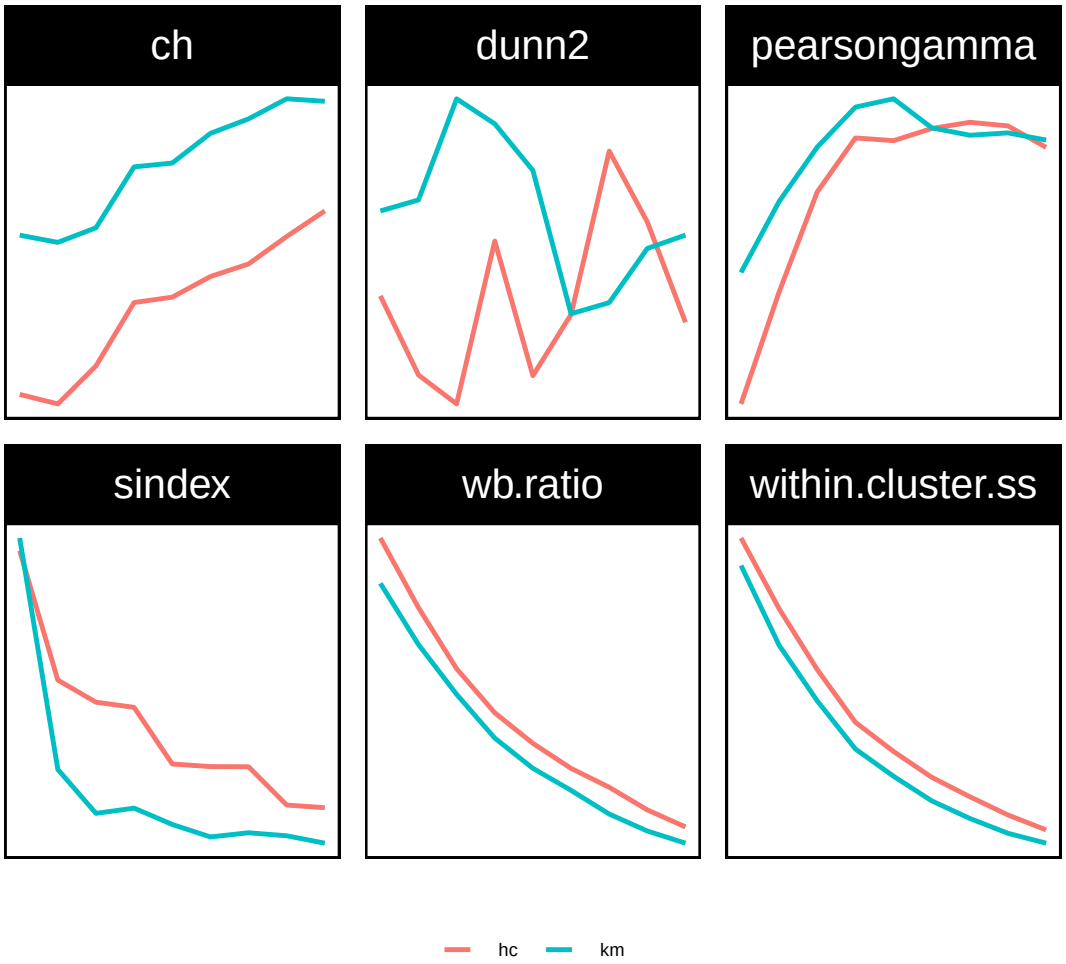
Figure 9: Assessing which of the 6 NLDR layouts ((a) tSNE, (b) UMAP, (c) PAHTE, (d) TriMAP, (e) PaCMAP, and (f) PCA) of the five clusters data is the better representation using HBE for varying binwidth (a_1). Colour is used for the lines and points in the left plot to match the scatterplots of the NLDR layouts (a-f). Layout f is universally poor. Layouts a and b are universally optimal. Layout b shows six well-separated clusters and layout a shows close clusters, thus layout a is the best choice.

(Gamage et al., 2019). A lower HBE indicates that the method better preserves the high-dimensional structure in its low-dimensional embedding.

As shown in Figure 9, tSNE (Figure 9 a) achieved the lowest HBE across bin widths (mostly tiny), indicating high preservation of both local and global structures. Its layout displays well-separated clusters with minimal inter-cluster distances, making it the most faithful representation of the underlying data structure. UMAP and PaCMAP (Figure 9 b and e) produced moderately accurate embeddings, although the six clusters appear more well-separated, while PHATE (Figure 9 c) show non-linear cluster structures irrespective of the original structure. Also, TriMAP (Figure 9 d) has high HBE, and show three clusters with small distances. PCA (Figure 9 f) failed to capture the non-linear geometry, leading to the highest HBE.

Benchmarking clustering algorithms

To further evaluate the structure of the generated data, we benchmarked three clustering algorithms: *k*-**means** (Chapter 20 of Boehmke and Greenwell, 2019), **hierarchical** (Murtagh and Contreras, 2012), and **model-based clustering** (Fraley and Raftery, 2002; Scrucca et al., 2023) using the simulated dataset. The model-based clustering was performed with the `"VVV"` covariance structure, allowing each cluster to vary in volume, shape, and orientation. Cluster validity statistics were computed using the `cluster.stats()` function from the `fpc` package (Hennig, 2024).



? shows a selection of cluster metrics for 2-10 clusters for each of the methods, *k*-means, hierarchical, ... As is typical, the suggestion of best solution varies between cluster statistics. There is some agreement that 4-5 clusters is better than other choices (say why), and that *k*-means performs better than others (this might not hold when you add the next methods).

Overall, all methods produced similar compactness and separation, as reflected by the *within–between cluster ratios (wb.ratio)* and *Dunn indices*. However, the **model-based clustering** achieved the highest *Corrected Rand Index* (0.75) and lowest *Variation of Information (VI)* (0.65), indicating the best recovery of the true underlying groups (Table **??**). In comparison, *k*-means and hierarchical clustering showed moderate agreement with the true labels. These findings demonstrate that mixture-based approaches can more effectively capture the heterogeneity of clusters in high-dimensional, non-spherical data.

5 Conclusion

The `cardinalR` package introduces a flexible framework for generating high-dimensional data structures with well-defined geometric properties. It addresses an important need in the evaluation of clustering, machine learning, and DR methods by enabling the construction of customized datasets with interpretable structures, noise characteristics, and clustering arrangements. In this way, `cardinalR` complements existing packages such as `geozoo`, `snedata`, and `mlbench`, while extending the scope to higher dimensions and more complex shapes.

The included structures cover a wide range of diagnostic settings. Branching shapes facilitate

the study of continuity and topological preservation, the S-curve with a hole allows investigation of incomplete manifolds, and clustered spheres assess separability on curved surfaces. The Möbius strip introduces challenges from non-orientable geometry, while gridded cubes and pyrholes test spatial regularity and clustering in sparse, non-convex regions.

These structures are designed to support not only algorithm diagnostics, but also teaching high-dimensional concepts, benchmarking reproducibility, and evaluating hyper-parameter sensitivity. By allowing users to adjust dimensionality, sample size, noise, and clustering properties, the package promotes transparent experimentation and comparative model evaluation. Together, these capabilities make `cardinalR` a versatile tool for generating interpretable, high-dimensional datasets that advance research, teaching, and evaluation of data-analytic methods.

Future extensions of `cardinalR` may include biologically inspired or application-driven data structures that would further broaden its utility in domains such as bioinformatics, forensic science, and spatial analysis.

6 Acknowledgements

The source material for this paper is available at github.com/JayaniLakshika/paper-cardinalR. This article is created using **knitr** (Xie, 2015) and **rmarkdown** (Xie et al., 2018) in R with the `rjtools::rjournal_article` template. These R packages were used for this work: `cli` (Csárdi, 2025), `tibble` (Müller and Wickham, 2023), `gtools` (Warnes et al., 2023), `dplyr` (Wickham et al., 2023), `stats` (R Core Team, 2025), `tidyr` (Wickham et al., 2024), `purrr` (Wickham and Henry, 2025), `mvtnorm` (Genz and Bretz, 2009), `geozoo` (Schloerke, 2016), and `MASS` (Venables and Ripley, 2002).

Bibliography

D. K. Agrafiotis and H. Xu. A self-organizing principle for learning nonlinear manifolds. *Proceedings of the National Academy of Sciences*, 99(25):15869–15872, 2002. doi: 10.1073/pnas.242424399. URL https://www.pnas.org/doi/abs/10.1073/pnas.242424399. [p11]

B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 2014. [p9]

E. Amid and M. K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets. *ArXiv*, abs/1910.00204, 2019. URL https://api.semanticscholar.org/CorpusID:203610264. [p16]

J. Arsuaga, M. Vazquez, S. Trigueros, D. W. L. Sumners, and J. Roca. Characterizing the entanglement of DNA molecules. *PNAS*, 99(8):5373–5377, 2002. doi: 10.1073/pnas.032095099. [p12]

D. Asimov. The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM Journal of Scientific and Statistical Computing*, 6(1):128–143, 1985. [p1]

J. J. Balamuta. *surreal: Create Datasets with Hidden Images in Residual Plots*, 2024. URL https://CRAN.R-project.org/package=surreal. R package version 0.0.1. [p1]

B. Boehmke and B. M. Greenwell. *Hands-On Machine Learning with R*. Chapman and Hall/CRC, 1st edition, 2019. doi: 10.1201/9780367816377. URL https://doi.org/10.1201/9780367816377. [p18]

C. R. Calladine, H. R. Drew, B. F. Luisi, and A. A. Travers. Understanding DNA: the molecule and how it works. 1997. [p12]

G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2025. URL https://CRAN.R-project.org/package=cli. R package version 3.6.4. [p19]

L. Devroye. *Non-Uniform Random Variate Generation(originally published with*. Springer-Verlag, 1986. URL http://cg.scs.carleton.ca/~luc/rnbookindex.html. [p7]

C. Fraley and A. E. Raftery. Model-Based Clustering, Discriminant Analysis, and Density Estimation. *Journal of the American Statistical Association*, 97(458):611–631, 2002. URL https://doi.org/10.1198/016214502760047131. [p18]

J. P. Gamage, D. Cook, P. Harrison, M. Lydeamore, and T. S. Talagala. Choosing Better NLDR Layouts by Evaluating the Model in the High-dimensional Data Space. *ArXiv*, abs/2506.22051, 2019. URL https://arxiv.org/abs/2506.22051. [p17]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Springer-Verlag, 2009. ISBN 978-3-642-01688-2. [p19]

N. Gershenfeld. The physics of information technology. 2000. [p12]

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. volume 2, pages 1735–1742, 2006. doi: 10.1109/CVPR.2006.100. [p6]

P. Harrison. langevitour: Smooth interactive touring of high dimensions, demonstrated with scrna-seq data. *The R Journal*, 15(2):206–219, 2023. doi: 10.32614/RJ-2023-046. [p1]

C. Hennig. *fpc: Flexible Procedures for Clustering*, 2024. URL https://CRAN.R-project.org/package=fpc. R package version 2.2-13. [p18]

I. Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. URL https://doi.org/10.1007/978-3-642-04898-2_455. [p16]

S. Laurent. Four-dimensional torus knots, 2024. URL https://laustep.github.io/stlahblog/posts/TorusKnot4D.html. Accessed: 2025-11-17. [p12]

F. Leisch and E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2024. URL https://CRAN.R-project.org/package=mlbench. R package version 2.1-6. [p1]

B. B. Mandelbrot. The fractal geometry of nature. *Earth Surface Processes and Landforms*, 8(4):406–406, 1983. doi: 10.1002/esp.3290080415. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/esp.3290080415. [p8]

L. McInnes, J. Healy, N. Saul, and L. Großberger. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29):861, 2018. URL https://doi.org/10.21105/joss.00861. [p16]

G. J. McLachlan and D. Peel. Finite Mixture Models. In *Wiley Series in Probability and Statistics*, 2000. URL https://api.semanticscholar.org/CorpusID:124985575. [p7]

J. Melville. *snedata: SNE Simulation Dataset Functions*, 2025. URL https://github.com/jlmelville/snedata. R package version 0.0.0.9001, commit beebcf91c365bf5006be08fb614585b4659c05c5. [p1, 8, 11]

K. R. Moon, D. van Dijk, Z. Wang, S. A. Gigante, D. B. Burkhardt, W. S. Chen, K. Yim, A. van den Elzen, M. J. Hirn, R. R. Coifman, N. B. Ivanova, G. Wolf, and S. Krishnaswamy. Visualizing Structure and Transitions in High-dimensional Biological Data. *Nature Biotechnology*, 37:1482–1492, 2019. doi: 10.1038/s41587-019-0337-2. [p16]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2023. URL https://CRAN.R-project.org/package=tibble. R package version 3.2.1. [p19]

F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, 2012. doi: 10.1002/widm.53. URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.53. [p18]

H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992. ISBN 0898712955. [p7]

Optica – The Optical Society. Optical Möbius Strips Yield New Secrets. https://www.optica-opn.org/home/newsroom/2023/january/optical_mobius_strips_yield_new_secrets/, 2023. Accessed: 2025-10-06. [p7]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2025. URL https://www.R-project.org/. [p19]

B. Schloerke. *geozoo: Zoo of Geometric Objects*, 2016. URL https://CRAN.R-project.org/package=geozoo. R package version 0.5.1. [p1, 19]

L. Scrucca, C. Fraley, T. B. Murphy, and A. E. Raftery. *Model-Based Clustering, Classification, and Density Estimation Using Mclust in R*. Chapman and Hall/CRC the R Series. CRC Press LLC, 1st ed. edition, 2023. ISBN 9781000868371. [p18]

L. A. Stefanski. Residual (Sur) Realism. *The American Statistician*, 61(2):163–177, 2007. doi: 10.1198/000313007X208407. URL http://www.jstor.org/stable/27643870. [p1]

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. doi: 10.1126/science.290.5500.2319. [p13]

M. Tinkham. *Group Theory and Quantum Mechanics*. Courier Corporation, 2003. [p9]

C. Trapnell, D. Cacchiarelli, J. Grimsby, P. Pokharel, S. Li, M. Morse, N. J. Lennon, K. J. Livak, T. S. Mikkelsen, and J. L. Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology*, 32(4):381–386, 2014. doi: 10.1038/nbt.2859. URL https://doi.org/10.1038/nbt.2859. [p3, 7]

L. van der Maaten, E. Postma, and H. Herik. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research - JMLR*, 10, 01 2007. [p13]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, fourth edition, 2002. URL https://www.stats.ox.ac.uk/pub/MASS4/. ISBN 0-387-95457-0. [p19]

Y. Wang, H. Huang, C. Rudin, and Y. Shaposhnik. Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMap, and PaCMAP for Data Visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021. URL http://jmlr.org/papers/v22/20-1061.html. [p16]

G. R. Warnes, B. Bolker, T. Lumley, A. Magnusson, B. Venables, G. Ryodan, and S. Moeller. *gtools: Various R Programming Tools*, 2023. URL https://CRAN.R-project.org/package=gtools. R package version 3.9.5. [p19]

H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2025. URL https://CRAN.R-project.org/package=purrr. R package version 1.0.4. [p19]

H. Wickham, R. François, L. Henry, K. Müller, and D. Vaughan. *dplyr: A Grammar of Data Manipulation*, 2023. URL https://CRAN.R-project.org/package=dplyr. R package version 1.1.4. [p19]

H. Wickham, D. Vaughan, and M. Girlich. *tidyr: Tidy Messy Data*, 2024. URL https://CRAN.R-project.org/package=tidyr. R package version 1.3.1. [p19]

E. Witten. Non-commutative geometry and knot theory. *Communications in Mathematical Physics*, 121 (3):351–399, 1985. doi: 10.1007/BF01210791. [p12]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2nd edition, 2015. URL https://yihui.name/knitr/. ISBN 978-1498716963. [p19]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, 2018. URL https://bookdown.org/yihui/rmarkdown. ISBN 978-1138359338. [p19]

L. Zappia, B. Phipson, and A. Oshlack. Splatter: simulation of single-cell RNA sequencing data. *Genome Biology*, 2017. doi: 10.1186/s13059-017-1305-0. URL http://dx.doi.org/10.1186/s13059-017-1305-0. [p1]

Jayani P. Gamage
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
https://jayanilakshika.netlify.app/
ORCiD: 0000-0002-6265-6481
jayani.piyadigamage@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
http://www.dicook.org/
ORCiD: 0000-0002-3813-7155
dicook@monash.edu

Paul Harrison
Monash University
MGBP, BDInstitute, VIC 3800 Australia
ORCiD: 0000-0002-3980-268X
paul.harrison@monash.edu

Michael Lydeamore
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
ORCiD: 0000-0001-6515-827X
michael.lydeamore@monash.edu

Thiyanga S. Talagala
University of Sri Jayewardenepura
Department of Statistics, Gangodawila, Nugegoda 10100 Sri Lanka
https://thiyanga.netlify.app/
ORCiD: 0000-0002-0656-9789
ttalagala@sjp.ac.lk