cardinalR: Generating interesting high-dimensional data structures

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

Abstract A high-dimensional dataset is one where each observation is described by many features, or dimensions, with associations between them. These datasets contain nonlinear manifolds in image and speech recognition, clusters in genomics and forensic analysis, and sparse distributions in text mining. Data with a variety of structures can be generated using mathematical functions and statistical distributions to create test datasets. High-dimensional data structures are useful for testing, validating, and improving algorithms used in dimensionality reduction, clustering, machine learning, and visualization. Their controlled complexity allows researchers to understand challenges posed in data analysis and helps to develop robust analytical methods across diverse scientific fields like bioinformatics, machine learning, and forensic science. Functions to generate a large variety of structures in high dimensions are organized into the R package `cardinalR`, along with some already generated examples, adding to 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)) offers a large collection of geometric objects, allowing users to create and analyze specific shapes, primarily in lower-dimensional spaces. The package is `snedata` (Melville (2025)), which provides tools for generating simplified datasets useful for evaluating dimensionality reduction techniques like tSNE, often focusing on understanding and evaluating low-dimensional embeddings of complex data structures. Additionally, `splatter` (Zappia et al. (2017)) is designed to simulate complex biological data, capturing field-specific nuances such as batch effects and differential expression. In contrast, `mlbench` (Leisch and Dimitriadou (2024)) includes a collection of well-known 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. Meanwhile, the `DHARMa` package (Hartig (2024)) adopts a simulation-based approach to create scaled quantile residuals for generalized linear (mixed) models and related frameworks, supporting model diagnostics through intuitive residuals, plots, and tests for common misspecification issues.

While these packages are valuable, their scope is often limited to specific applications or low-dimensional structures. To address this gap, this paper introduces the `cardinalR` R package. This package provides a collection of functions designed to generate customizable data structures in any number of dimensions, starting from basic geometric shapes. `cardinalR` offers important functionalities that extend beyond the capabilities of existing tools, 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 high-dimensional datasets into a single multi-faceted, clustered dataset in a space of arbitrary dimension. By using clearly defined geometric shapes and controllable characteristics such as number of dimensions, sample size; `cardinalR` allows researchers to generate transparent and interpretable synthetic datasets useful for evaluating the performance of nonlinear dimensionality reduction (NLDR) methods, clustering algorithms, and visualization techniques. Moreover, these datasets can serve as benchmark examples for exploring how different algorithmic choices affect the identification or representation of cluster and manifold structures in high-dimensional spaces.

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.

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.

2 Implementation

The `cardinalR` R package is available on GitHub at JayaniLakshika/cardinalR.

Usage

Main function

The main function of the package is `gen_multicluster()`, which generates datasets consisting of multiple clusters with user-specified characteristics. 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.

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 a collection of functions to create synthetic data structures based on simple, well-defined geometric structures. These include fundamental shapes such as cones, pyramids, spheres, grids, and branching structures. If a shape is not inherently defined in more than three dimensions, additional noise dimensions can be added to embed the structure into higher-dimensional space. Users can specify how these noise dimensions are generated (e.g., Gaussian, wavy) (`noise_fun`), offering control over the embedding process. All shape generators allow the user to define the number of points (`n`) and dimensions (`p`), and most include additional arguments to customize specific characteristics of the structure.

Cube A cube structure (Figure 1) represents uniformly or systematically distributed points within a high-dimensional hypercube, providing a useful framework for assessing how well algorithms preserve uniformity, spacing, and boundary properties in high dimensions. We provide a set of

Table 2: cardinalR cube data generation functions

Function	Explanation
gen_gridcube	Cube with specified grid points along each axes.
gen_unifcube	Cube with uniform points.

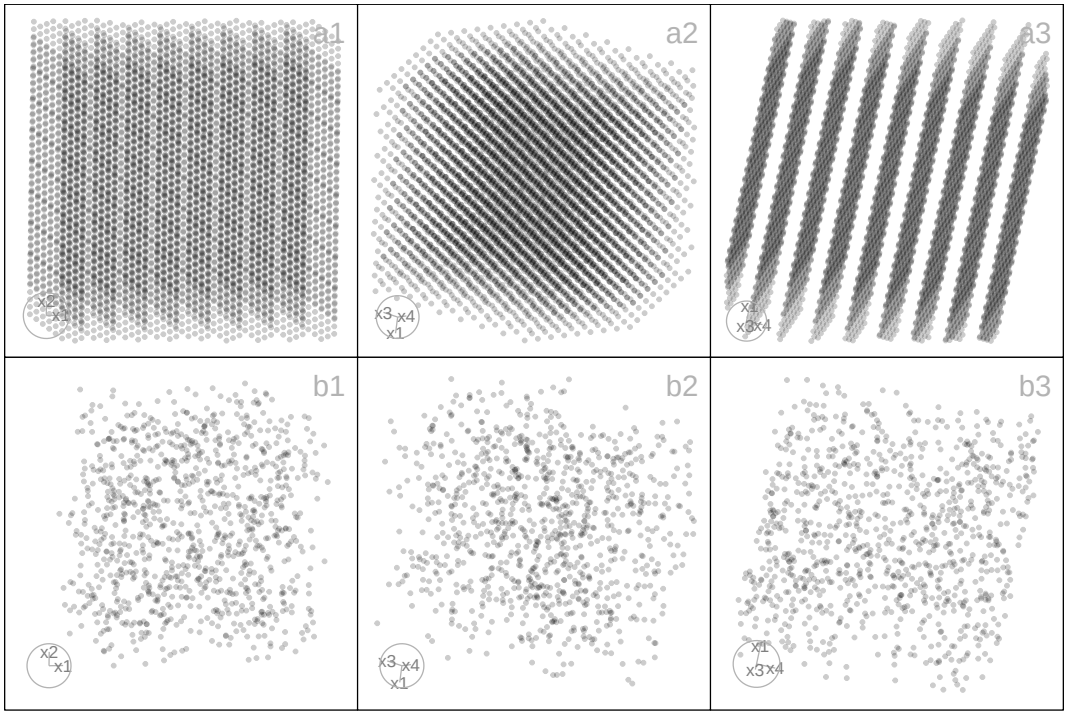

Figure 1: Three 2-*D* projections from 4-*D*, for the 'gridcube' (a1-a3), 'unifcube' (b1-b3), and 'cubehole' (c1-c3) data.

functions to generate high-dimensional cube structures with flexible configurations, including regular grids, uniform random points, and cubes with missing regions or holes. These structures are valuable for testing the ability of algorithms to maintain uniform spacing or to detect gaps in the data. Table 2 outlines these functions and their purposes.

The first is the regular grid of points of n points in p dimensions. This is generated using `gen_gridcube(n, p)`. The number of grid points along each axis is determined by finding the nearest integer factors whose product is close to n. Each dimension is then normalized to lie in the interval $[0, 1]$, so that the resulting structure forms a true p-D hypercube. This produces a lattice of evenly spaced points along all axes, providing a uniform and interpretable high-dimensional grid.

```
gridcube <- gen_gridcube(n = 1000, p = 4)
```

An extension to the regular grid of points is to consider the points being uniformly distributed along each axis, as opposed to evenly spaced. The function `gen_unifcube(n, p)` is identical to the regular grid of points, except instead of points being placed in integer grid coordinates, they are placed at a uniformly distributed point inside the p-D cube (Figure 1 b).

```
unifcube <- gen_unifcube(n = 1000, p = 4)
```

Cone To simulate a cone-shaped structure in arbitrary dimensions (Figure 2), 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

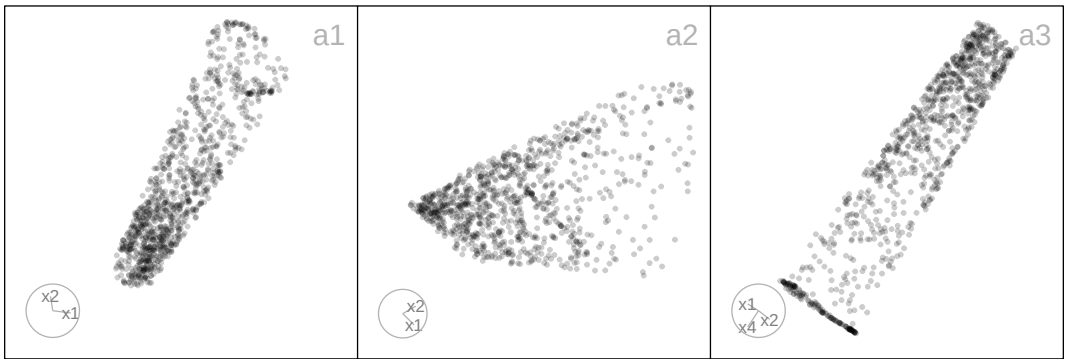

Figure 2: Three 2-D projections from 4-D, for the 'cone' data.

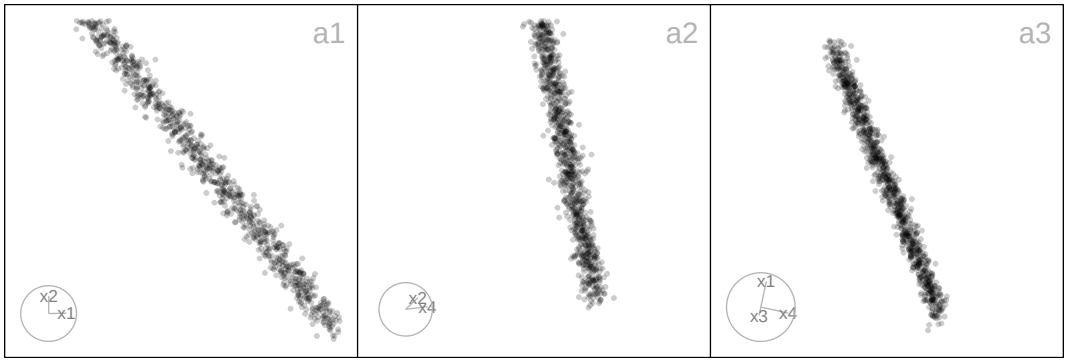

Figure 3: Three 2-D projections from 4-D, for the 'linear' data.

the cross-section decreases linearly from base to tip according to $r = r_{\min} + (r_{\max} - r_{\min})X_p/h$, where $r_{\min}$ = ratio and $r_{\max} = 1$.

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 <- gen_cone(n = 1000, p = 4, h = 5, ratio = 0.5)
```

Linear The `gen_longlinear(n, p)` function generates a high-dimensional dataset representing a long linear structure with noise. Each variable is formed as $X_i = \text{scale}_i \cdot (0, 1, \ldots, n-1+\epsilon) + \text{shift}_i$, where scale_$i$ $\sim U(-10, 10)$ determines the orientation of the line in each dimension, shift_i $\sim U(-300, 300)$ offsets the line to separate dimensions, and $\epsilon \sim N(0, (0.03n)^2)$ introduces Gaussian noise.

```
linear <- gen_longlinear(n = 1000, p = 4)
```

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 (Figure 4). 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.

```
gau <- gen_gaussian(n = 1000, p = 4, s = diag(4))
```

Pyramid A pyramid structure (Figure 5) represents data arranged around a central apex and base, useful for exploring how algorithms handle pointed or layered geometries in high-dimensional space. The functions provided allow users to generate pyramids with rectangular, triangular, and star-shaped

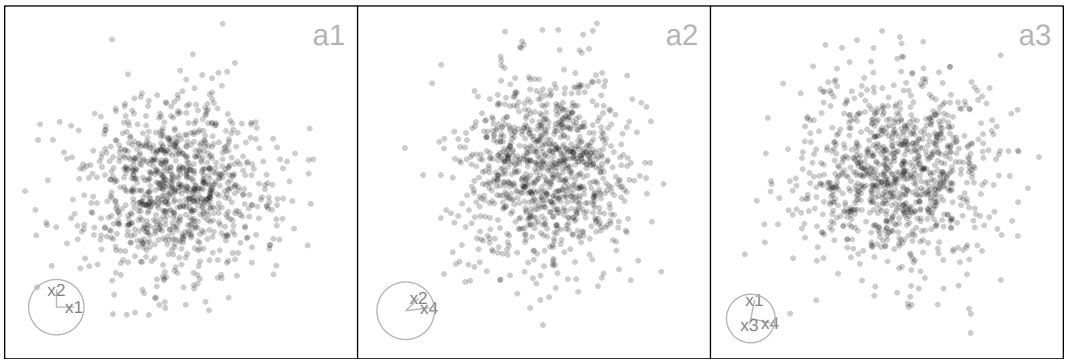

Figure 4: Three 2-D projections from 4-D, for the 'gau' data.

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. Table 3 summarizes these functions.

Let $X_1, \ldots, X_p$ denote the coordinates of the generated points. For the rectangular and triangular 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 5 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:

$$X_1 \sim U(-r_x(z),\ r_x(z)), \quad X_2 \sim U(-r_y(z),\ r_y(z)), \text{ and } X_3 \sim U(-r_x(z),\ r_x(z)).$$

```
pyrrect <- gen_pyrrect(n = 1000, p = 4)
```

For the triangular based pyramid, `gen_pyrtri(n, p, h, l, rt)` (Figure 5 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: $u, v \sim U(0,1), \quad \text{if } u + v > 1 : u \leftarrow 1 - u,\ v \leftarrow 1 - v$. The first three coordinates (triangle plane) are then: $X_1 = r(z)(1 - u - v)$, $X_2 = r(z)u$, and $X_3 = r(z)v$.

```
pyrtri <- gen_pyrtri(n = 1000, p = 4)
```

For the star based pyramid, `gen_pyrstar(n, p, h, rb)` (Figure 5 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 $X_2 = r(z) r_{\text{point}} \sin(\theta)$.

```
pyrstar <- gen_pyrstar(n = 1000, p = 4)
```

For all the above pyramid shapes, if $p > 3$, the remaining $p - 3$ dimensions (i.e., X_4 to X_{p-1}) are additional noise.

Finally, for the Sierpinski-like pyramid, `gen_pyrfrac(n, p)` (Figure 5 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: $T_i = 1/2(T_{i-1} + C_k), \quad C_k \in \{C_1, \ldots, C_{p+1}\}$. 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: $X = \{T_1, T_2, \ldots, T_n\}, \quad X \in \mathbb{R}^{n \times p}$.

Table 3: cardinalR pyramid data generation functions

Function	Explanation
gen_pyrrect	Rectangular-base pyramid, with a sharp or blunted apex.
gen_pyrtri	Triangular-base pyramid, with a sharp or blunted apex.
gen_pyrstar	Star-shaped base pyramid, with a sharp or blunted apex.
gen_pyrfrac	Pyramid containing triangular pyramid-shaped holes.

Table 4: cardinalR polynomial data generation functions

Function	Explanation
gen_quadratic	Quadratic pattern.
gen_cubic	Cubic pattern.

```
pyrholes <- gen_pyrfrac(n = 1000, p = 4)
```

Polynomial A polynomial structure generates data points that follow non-linear curvilinear relationships, such as quadratic or cubic trends, in 2-D 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. Table 4 summarizes these functions and their purposes.

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.

```
quadratic <- gen_quadratic(n = 1000)
```

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.

```
cubic <- gen_cubic(n = 1000)
```

S-curve An S-curve structure (Figure 6) simulates data that lies along a smooth, non-linear manifold.

For the S-curve structure, `gen_scurve(n, p)` (Figure 6 a), the 3-D geometry is constructed by introducing a latent parameter, $\theta \sim U(-3\pi/2, 3\pi/2)$. This parameter controls the curvature of the manifold. The first three dimensions form the S-curve structure:

$$X_1 = \sin(\theta), \quad X_2 \sim U(0, 2), \quad X_3 = \text{sign}(\theta)(\cos(\theta) - 1)$$

. This configuration creates a horizontally curled shape in (X_1, X_3), with additional band thickness in the X_2 direction.

```
scurve <- gen_scurve(n = 1000)
```

Swiss Roll To further generalize the Swiss roll structure and introduce realistic noise, we define a function `gen_swissroll(n, w)`, where n is the number of points, p is the total number of dimensions,

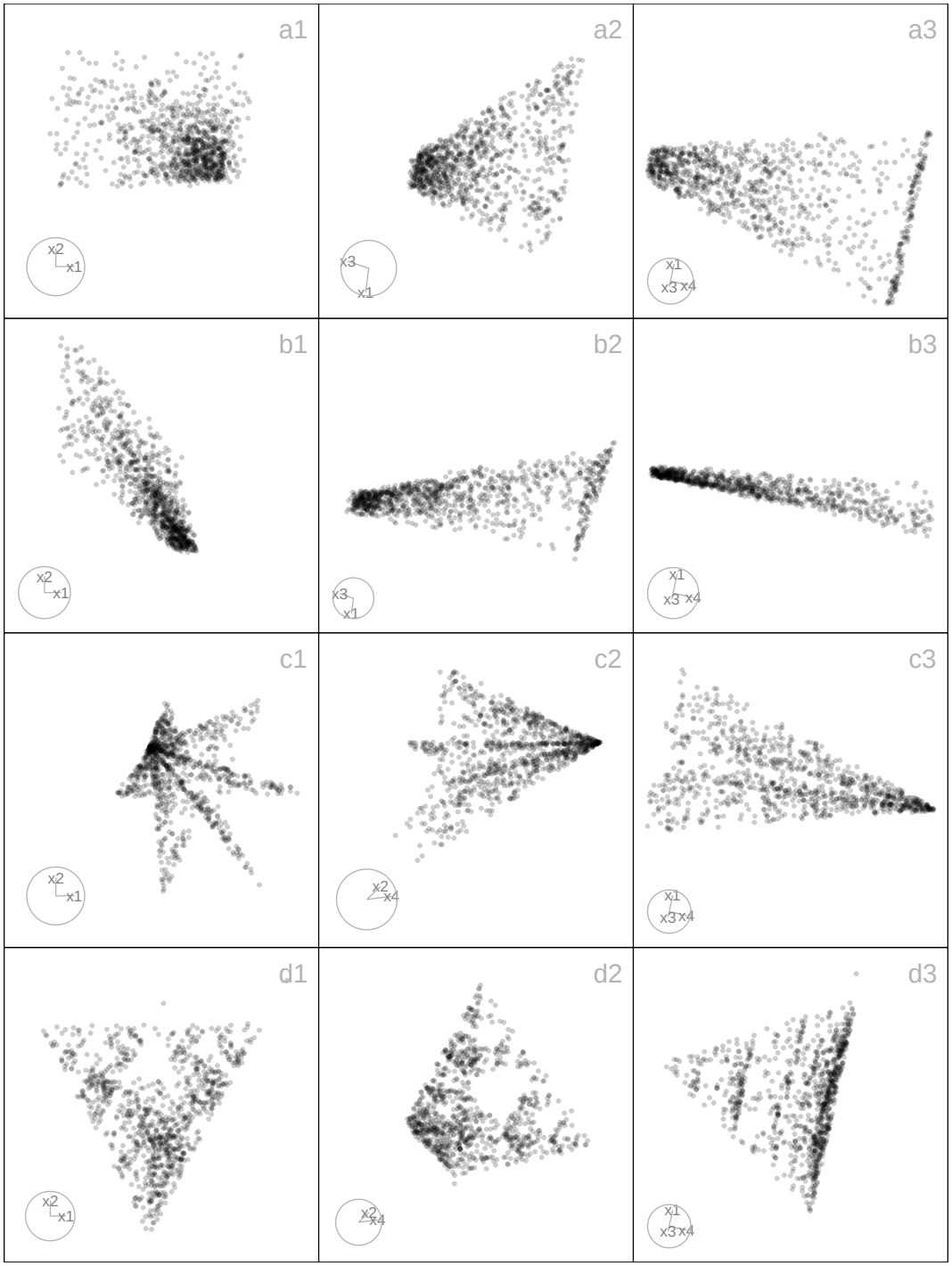

Figure 5: Three 2-*D* projections from 4-*D*, for the 'pyrrect' (a1-a3), 'pyrtri' (b1-b3), 'pyrstar' (c1-c3), and 'pyrholes' (d1-d3) data.

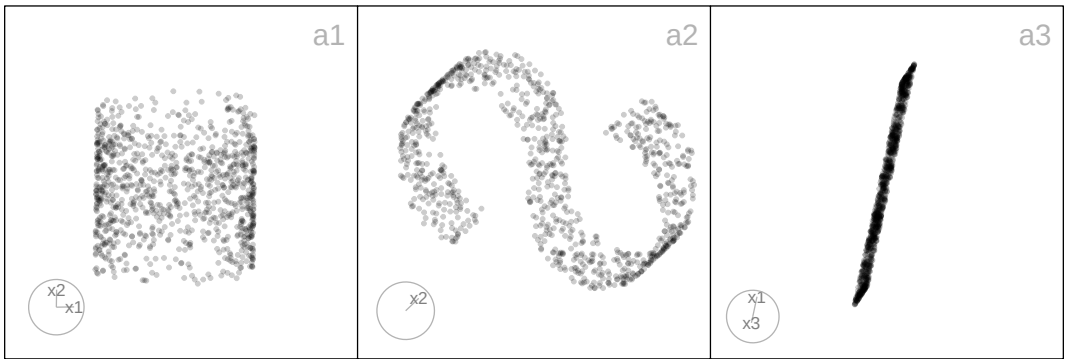

Figure 6: Three 2-D projections from 4-D, for the 'scurve' (a1-a3) and 'scurvehole' (b1-b3) data.

Table 5: cardinalR trefoil data generation functions

Function	Explanation
gen_trefoil4d	Trefoil in $4\text{-}D$.
gen_trefoil3d	Trefoil in $3\text{-}D$.

and w is the vertical range in the third dimension (Figure 7). The first three dimensions form the classic 3-D Swiss roll shape. The

$$X_1 = t\cos(t), \quad X_2 = t\sin(t), \quad X_3 \sim U(w_1, w_2), \text{ where } t \sim U(0, 3\pi).$$

```
swissroll <- gen_swissroll(n = 1000, w = c(-1, 1))
```

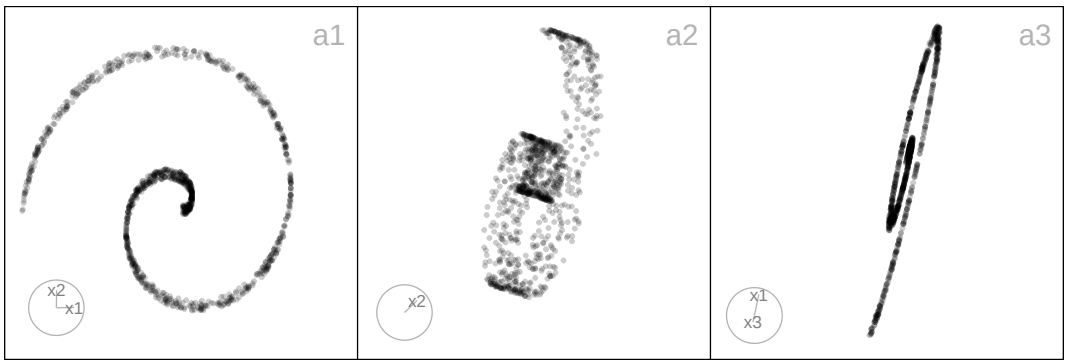

Figure 7: Three 2-D projections from 3-D, for the 'swissroll' data.

Trefoil knots The Trefoil is a closed, nontrivial one-dimensional manifold embedded in 3-D or 4-D space (Figure 8). 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. Table 5 summarizes these functions.

For the 4-D trefoil knot, 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 8 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.

```
trefoil4d <- gen_trefoil4d(n = 500, steps = 5)
```

For the 3-D stereographic projection, `gen_trefoil3d(n, steps)` maps each point $(X_1, X_2, X_3, X_4) \in \mathbb{R}^4$ to

$$(X_1', X_2', X_3') \in \mathbb{R}^3 \text{ using } X_1' = X_1/(1 - X_4), \quad X_2' = X_2/(1 - X_4), \text{ and } X_3' = X_3/(1 - X_4),$$

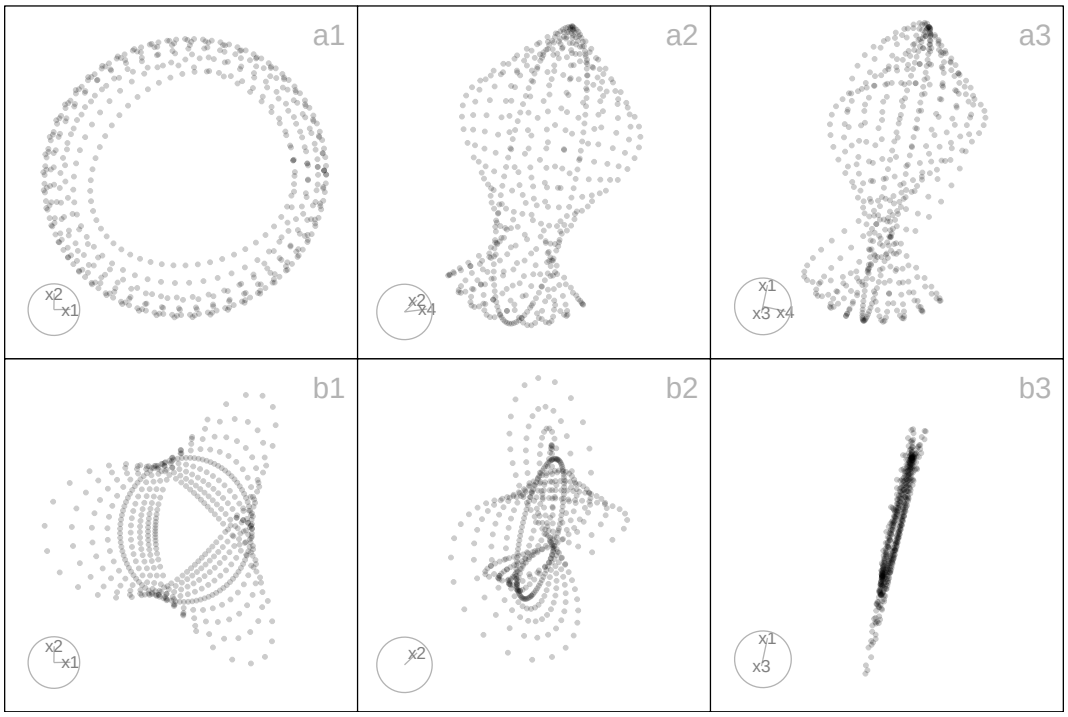

Figure 8: Three 2-*D* projections from 4-*D*, for the 'trefoil4d' (a1-a3) and 'trefoil3d' (b1-b3) data.

Table 6: 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.

excluding points where $X_4 = 1$ to avoid division by zero (Figure 8 b).

```
trefoil3d <- gen_trefoil3d(n = 500, steps = 5)
```

Wrappers

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 6 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.

Table 7: 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 cluster 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.
make_three_clust_	Three clusters with different shapes. (eg:- 01, 02, ..., 20)

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

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

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.

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 7.

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 8 details these functions.

Table 8: cardinalR additional functions

Function	Explanation
gen_bkgnoise	Adds background noise.
randomize_rows	Randomizes the rows.
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.
gen_rotation	Generates rotations.
normalize_data	Normalizes data.

3 Application

This section illustrates the use of package by generating a synthetic dataset to evaluate the performance of six popular dimension reduction techniques: Principal Component Analysis (PCA) (Jolliffe, 2011), t-distributed stochastic neighbor embedding (tSNE) (Maaten and Hinton, 2008), 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).

The following code generates a dataset of five clusters, positioned with equal inter-cluster distances in 4-*D* space (Figure 9). Each cluster was chosen to reflect distinct geometric and topological properties, allowing us to test how well DR methods preserve both local and global structures. The *helical spiral* cluster is designed to evaluate methods on elongated, twisting structures that challenge linear embeddings such as PCA and require preservation of curvilinear continuity. The *hemisphere* provides a curved surface with partial coverage of a 3-*D* manifold, useful for testing neighborhood preservation and unfolding in algorithms like UMAP and tSNE. The *uniform cube* represents isotropic, uniformly distributed data and serves as a control cluster with simple geometric structure to assess baseline embedding fidelity. The *cone* introduces variable density along one axis, mimicking structures where point density changes with geometry, helping evaluate how well algorithms maintain relative distances in non-uniform distributions. Finally, the *Gaussian* cluster is a standard multivariate normal distribution, included to assess algorithm performance on simple, spherical, high-density clusters. Together, these clusters create a challenging synthetic dataset suitable for benchmarking and exploring the strengths and weaknesses of different dimensionality reduction techniques.

```
positions <- geozoo::simplex(p=4)$points
positions <- positions * 0.8

## To generate data
five_clusts <- gen_multicluster(n = c(2250, 1500, 750, 1250, 1750), k = 5,
                      loc = positions,
                      scale = c(0.4, 0.35, 0.3, 1, 0.3),
                      shape = c("helicalspiral", "hemisphere", "unifcube",
                                "cone", "gaussian"),
                      rotation = NULL,
                      is_bkg = FALSE)
```

The five clusters have different geometric structures and each contain different number of points. Specifically, the helical spiral cluster includes 2250 points and was generated with a scale parameter of 0.4. The hemisphere cluster consists of 1500 points with a scale parameter of 0.35. The uniform cube-shaped cluster contains 750 points and uses a scale parameter of 0.3. The blunted cone cluster includes 1250 points, generated with a scale parameter of 1. Finally, the Gaussian-shaped cluster contains 1750 points and was generated with a scale parameter of 0.3.

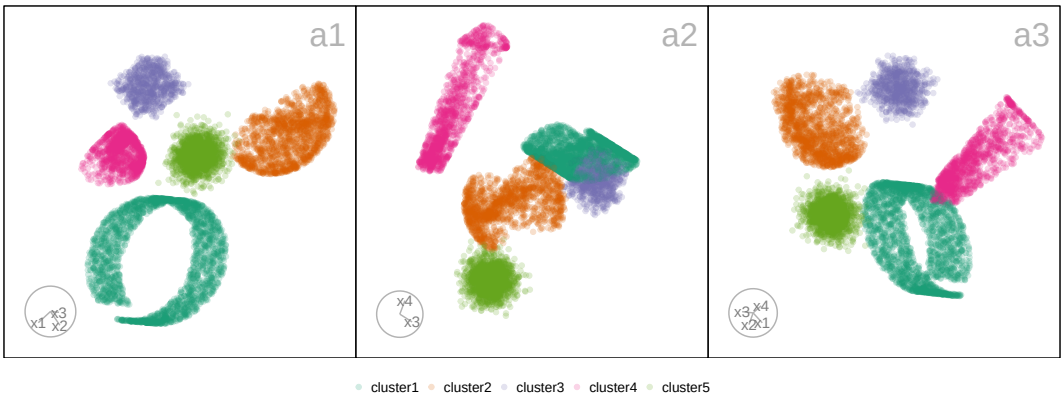

Figure 9: Three 2-*D* projections from 4-*D*, for the five clusters data. The helical spiral cluster is represented in dark green, the hemisphere cluster in orange, the uniform cube-shaped cluster in purple, the blunted cone cluster in pink, and the Gaussian-shaped cluster in light green.

UMAP, PHATE, TriMAP, and PaCMAP effectively separate the five clusters and show the preservation of the global structure (Figure 10). However, PHATE reveals three non-linear clusters, even though two of them do not show non-linearity. UMAP, TriMAP, and PaCMAP successfully maintain the local structures of the data. In contrast, tSNE divides the non-linear cluster into sub-clusters. Also, tSNE fails to preserve the distances between the clusters. PCA, on the other hand, preserves the local structures of the clusters, but some clusters are incorrectly merged that should remain distinct.

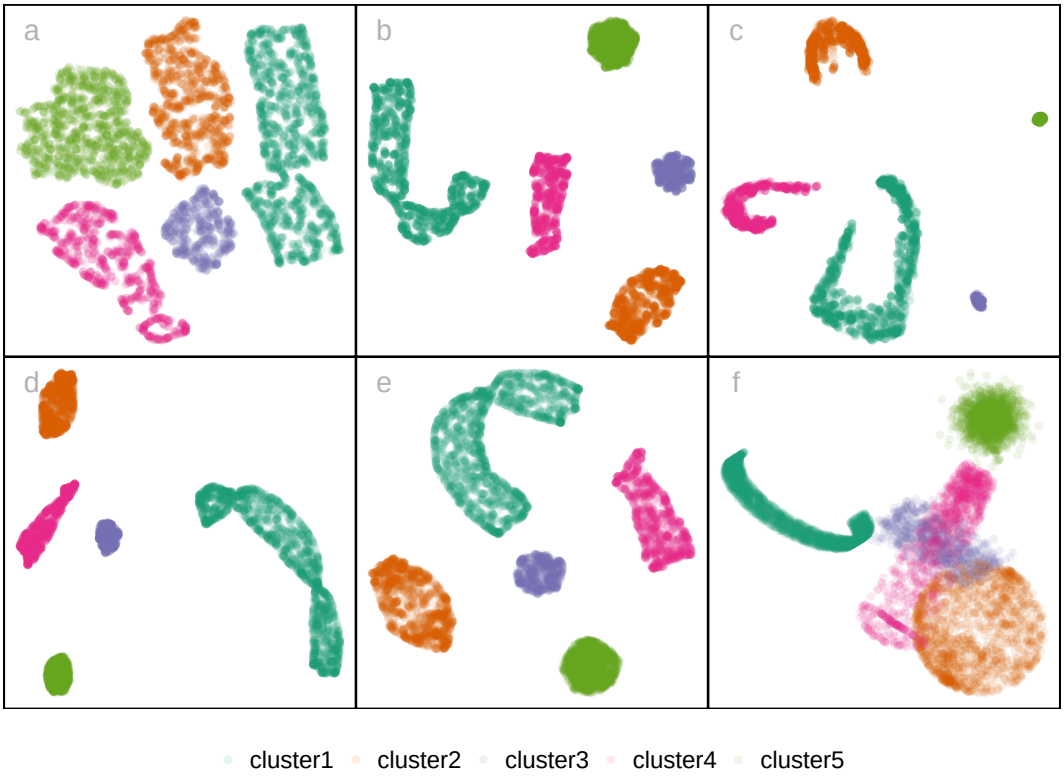

Figure 10: Six different dimension reduction representations of the five clusters data using default hyperparameter settings: (a) tSNE, (b) UMAP, (c) PAHTE, (d) TriMAP, (e) PaCMAP, and (f) PCA.

4 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 motivation for developing this package originated from the need to design a perception–misperception experiment, aimed at investigating how well NLDR methods preserve inter-cluster structure. To conduct this study, we 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 gradually led to the design of `cardinalR` as a general-purpose package, so that other researchers can benefit from the same tools for simulation, benchmarking, and teaching.

The included structures cover a wide range of diagnostic settings. Branching shapes facilitate the study of continuity and topological preservation, the Scurve 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 hyperparameter sensitivity. By allowing users to adjust dimensionality, sample size, noise, and clustering properties, the package promotes transparent experimentation and comparative model evaluation.

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

5 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

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. [p11]

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]

G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2025. URL https://CRAN.R-project.org/package=cli. R package version 3.6.4. [p13]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heidelberg, 2009. ISBN 978-3-642-01688-2. [p13]

F. Hartig. *DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models*, 2024. URL https://CRAN.R-project.org/package=DHARMa. R package version 0.4.7. [p1]

I. Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_455. URL https://doi.org/10.1007/978-3-642-04898-2_455. [p11]

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]

L. V. D. Maaten and G. E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605, 2008. [p11]

L. McInnes, J. Healy, N. Saul, and L. Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi: 10.21105/joss.00861. URL https://doi.org/10.21105/joss.00861. [p11]

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

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. [p11]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2023. URL https://CRAN.R-project.org/package=tibble. R package version 3.2.1. [p13]

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/. [p13]

B. Schloerke. *geozoo: Zoo of Geometric Objects*, 2016. URL https://CRAN.R-project.org/package=geozoo. R package version 0.5.1. [p1, 13]

L. A. Stefanski. Residual (sur) realism. *The American Statistician*, 61(2):163–177, 2007. ISSN 00031305. URL http://www.jstor.org/stable/27643870. [p1]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL https://www.stats.ox.ac.uk/pub/MASS4/. ISBN 0-387-95457-0. [p13]

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. [p11]

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. [p13]

H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2025. URL https://CRAN.R-project.org/package=purrr. R package version 1.0.4. [p13]

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. [p13]

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. [p13]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL https://yihui.name/knitr/. ISBN 978-1498716963. [p13]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. URL https://bookdown.org/yihui/rmarkdown. ISBN 978-1138359338. [p13]

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