

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 where each observation is described by many features, or dimensions. Such a dataset might contain various types of structures that have complex geometric properties, such as nonlinear manifolds, clusters, or sparse distributions. We can generate data containing a variety of structures using mathematical functions and statistical distributions. Sampling from a multivariate normal distribution will generate data in an elliptical shape. Using a trigonometric function we can generate a spiral. A torus function can create a donut shape. 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.

1 Introduction

Generating synthetic datasets with clearly defined geometric properties is essential 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. Another useful package is `sndata` ([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, `mlbench` ([Leisch and Dimitriadou \(2024\)](#)) includes a collection of well-known benchmark datasets commonly associated with established classification or regression challenges. In the field of single-cell omics, `splatter` ([Zappia et al. \(2017\)](#)) is designed to simulate complex biological data, capturing field-specific nuances such as batch effects and differential expression. 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 for evaluating algorithm performance in high-dimensional settings.

By providing these aspects, `cardinalR` provides researchers with a method to generate more explainable and challenging synthetic datasets focused to the specific needs of evaluating algorithms in high-dimensions. This bridges the gap between geometric foundations and the flexible generation of complex synthetic data.

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_multicluseter().

Argument	Type	Explanation
n	numeric (vector)	Number of points in each cluster.
p	numeric	Number of dimensions.
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)	Plane and the corresponding angle along that plane for each cluster.
is_bkg	boolean	Background noise should exist or not.

Table 2: cardinalR branching data generation functions

Function	Explanation
gen_expbranches	Generate a structure with exponential shaped branches.
gen_linearbranches	Generate a structure with linear shaped branches.
gen_curvybranches	Generate a structure with curvy shaped branches.
gen_orglinearbranches	Generate a structure with linear shaped branches originated in one point.
gen_orgcurvybranches	Generate a structure with curvy shaped branches originated in one point.

2 Implementation

Installation

The development version can be installed from GitHub:

```
pak::pak("JayaniLakshika/cardinalR")
```

Usage

Main function

The main function of the package is `gen_multicluseter()`, which generates datasets consisting of multiple clusters with user-specified characteristics. Users can control the number of clusters (`k`), the number of points in each cluster (`n`), and the dimensionality of the space (`p`). Each cluster can take on a different geometric shape (e.g., Gaussian, cone, uniform cube), be scaled to adjust its spread, rotated in specified planes by given angles, and positioned at defined centroids (`loc`). The function ensures flexibility in cluster location and orientation, allowing users to simulate complex high-dimensional structures. 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.

The main arguments of the `gen_multicluseter()` function are shown in Table 1.

Branching

A branching structure (Figure 1) captures trajectories that diverge or bifurcate from a common origin, similar to processes such as cell differentiation in biology. We introduce a set of data generation functions specifically designed to simulate high-dimensional branching structures with various geometries, numbers of points, and number of branches. Although these functions can generate multiple branches, they do not produce a formal *multicluseter* dataset: the branches form a single connected structure, with multiple visually distinct arms rather than independent clusters. Table 2 outlines these functions. The main arguments of the functions described in Table 3.

Table 3: The main arguments for branching shape generators.

Argument	Explanation
n	A numeric value representing the number of points.
p	A numeric value representing the number of dimensions.
k	A numeric value representing the number of clusters.

gen_exptranches() The `gen_exptranches(n, p, k)` function generates a dataset of n points forming k exponential branches in 2-D, with optional noise dimensions to embed the structure in p -D (Figure 1 a). These branches grow in opposite directions, producing a radiating curvilinear structure from a central region.

Each branch i is constructed using $X_1 \sim U(-2, 2)$ and $X_2 = \exp(\pm s_i X_1) + \epsilon$, where: $\epsilon \sim U(0, 0.1)$ adds local jitter, $s_i \in [0.5, 2]$ is randomly sampled for each branch. The sign of the exponent alternates between branches: odd-numbered branches have a negative exponent (decaying pattern), and even-numbered branches have a positive exponent (growing pattern), producing mirror-symmetric curves with varying steepness. For $p > 2$, Gaussian noise $X_j \sim N(0, 0.1^2)$ is added to embed the 2-D branches into p -D, where $j = 3, \dots, p$.

```
expbranches <- gen_exptranches(n = 1000, p = 4, k = 4)
```

gen_linearbranches() The `gen_linearbranches(n, p, k)` function generates a dataset of n points forming k approximately linear branches in p -D (Figure 1 b). The core structure lies in the first two dimensions and additional dimensions carry Gaussian noise.

The `gen_linearbranches(n, p, k)` function generates a dataset of n points in p -D forming k linear branches. Each branch is a line segment with added jitter to simulate noise. Branches 1 and 2 are initialized with fixed slopes and intercepts: branch 1 has slope 0.5 and intercept 0, with $X_1 \sim U(-2, 8)$ and $X_2 = 0.5X_1 + \epsilon, \epsilon \sim U(0, 0.5)$; branch 2 has slope -0.5 and intercept 0, with $X_1 \sim U(-6, 2)$ and $X_2 = -0.5X_1 + \epsilon, \epsilon \sim U(0, 0.5)$.

Branches 3 to k are added iteratively. Each new branch starts at a location outside predefined exclusion zones to avoid overlap with the first two branches. Specifically, X_1 starting points are sampled outside the intervals $[-8, -7]$, $[-2, 2]$, and $[7, 8]$, and X_2 starting points are sampled outside $[7, 8]$. For each new branch i , X_1 values are drawn over a short interval $[x_{\text{start}}, x_{\text{start}} + 1]$, and X_2 values are calculated as $X_2 = s_i(X_1 - x_{\text{start}}) + y_{\text{start}} + \epsilon$, where s_i is a randomly chosen slope from a filtered sequence and $\epsilon \sim U(0, 0.2)$. For $p > 2$, Gaussian noise $X_j \sim N(0, 0.05^2)$ is added to embed the 2-D branches into p -D, for $j = 3, \dots, p$.

```
linearbranches <- gen_linearbranches(n = 1000, p = 4, k = 4)
```

gen_curvybranches() The `gen_curvybranches(n, p, k)` function generates a dataset of n points forming k curvilinear branches embedded in p -D (Figure 1 c). The underlying geometry lies in the first two dimensions, while the remaining $(p - 2)$ dimensions contain Gaussian noise.

Branch 1 is generated from $X_1 \sim U(0, 1)$, and $X_2 = 0.1X_1 + X_1^2 + \epsilon$, where $\epsilon \sim U(0, 0.05)$. This produces a gently upward-curving parabola in the right half-plane. Branch 2 is generated from $X_1 \sim U(-1, 0)$, and $X_2 = 0.1X_1 - 2X_1^2 + \epsilon$, where $\epsilon \sim U(0, 0.05)$. This creates a steeper, leftward-facing curve in the left half-plane. Branches 3 to k are then added iteratively. Each new branch i begins at a randomly chosen starting point $(x_{\text{start}}, y_{\text{start}})$ from the existing structure, where the horizontal coordinate x_{start} is restricted to $[-0.15, 0.15]$ to ensure connectivity with the initial branches. From this starting point, the branch extends over a unit-length interval $X_1 \in [x_{\text{start}}, x_{\text{start}} + 1]$, with structure $X_2 = 0.1X_1 - s_i(X_1^2 - x_{\text{start}}) + y_{\text{start}}, s_i \in -2, -1.5, -1, -0.5, 0, 0.5, 1.5$. Here, each branch i is assigned a single scale factor s_i , sampled independently from the set, which controls the curvature of that branch. The parameter $(x_{\text{start}}, y_{\text{start}})$ is the starting point chosen from the existing structure. This construction yields k spatially connected, nonlinear branches with varying curvature. For $p > 2$, Gaussian noise $X_j \sim N(0, 0.05^2)$ is added to embed the 2-D branches into p -D, where $j = 3, \dots, p$.

```
curvybranches <- gen_curvybranches(n = 1000, p = 4, k = 4)
```

gen_orglinearbranches() The `gen_orglinearbranches(n, p, k)` function generates a dataset of n points forming k approximately linear branches embedded in p -D (Figure 1 d). Each branch lies within a distinct 2-D subspace, while the remaining $p - 2$ dimensions contain Gaussian noise.

To construct each branch, a unique or repeated pair of dimensions is selected from the $\binom{p}{2}$ possible combinations of dimensions. If $k \leq \binom{p}{2}$, combinations are sampled without replacement. If $k > \binom{p}{2}$, additional pairs are sampled with replacement to reach k total branches. Each selected pair (i_1, i_2) defines the 2-D plane for branch i .

For branch i , n_i points are generated, where $\sum_{i=1}^k n_i = n$. The structure follows: $X_{i_1} \sim U(0, 2)$, and $X_{i_2} = -s_i X_{i_1} + \epsilon$, $\epsilon \sim U(0, 0.5)$, where s_i is a scale factor controlling the slope. When $k \leq \binom{p}{2}$, $s_i = 1$. When sampling with replacement, s_i is drawn from the set $1, 1.5, 2, \dots, 8$ in increments of 0.5. For $p > 2$, the remaining dimensions contain independent Gaussian noise: $X_j \sim N(0, 0.1^2)$, for $j \notin \{i_1, i_2\}$.

```
orglinearbranches <- gen_orglinearbranches(n = 1000, p = 4, k = 4)
```

gen_orgcurvybranches() The `gen_orgcurvybranches(n, p, k)` function generates a dataset of n points forming k curvilinear branches embedded in a p -dimensional space (Figure 1 e). Each branch is constructed in a unique or repeated 2-D subspace of the p -dimensional space, with curvature induced by a second-degree polynomial structure. The remaining $p - 2$ dimensions contain Gaussian noise.

Let $\binom{p}{2}$ denote the number of unique 2-D subspace combinations. When $k \leq \binom{p}{2}$, k distinct subspace pairs (i_1, i_2) are sampled without replacement. Otherwise, combinations are selected with replacement to reach k total branches. For each branch $i = 1, \dots, k$, a random scale factor s_i is used to vary the curvature; If $k \leq \binom{p}{2}$: $s_i = 1$, or If $k > \binom{p}{2}$: $s_i \in \{1, 1.5, \dots, 8\}$.

Each branch contains n_i points such that $\sum_{i=1}^k n_i = n$, where the vector (n_1, \dots, n_k) is randomly drawn using the helper function `gen_nsum()` to partition n . Within its assigned subspace, branch i is defined by: $X_{i_1} \sim U(0, 2)$, and $X_{i_2} = -s_i X_{i_1}^2 + \epsilon$, $\epsilon \sim U(0, 0.5)$. This forms a smooth downward-opening parabola in the plane defined by (X_{i_1}, X_{i_2}) , with the degree of curvature controlled by s_i . For $p > 2$, all remaining dimensions are independent Gaussian noise: $X_j \sim N(0, 0.1^2)$, for $j \notin \{i_1, i_2\}$.

```
orgcurvybranches <- gen_orgcurvybranches(n = 1000, p = 4, k = 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 the cross-section decreases linearly from base to tip according to $r = r_{\min} + (r_{\max} - r_{\min})X_p/h$, where $r_{\min} = \text{ratio}$ 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 angular components allow a smooth extension into higher dimensions, preserving the conical shape within multi-dimensional space.

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

Cube

A cube structure (Figure 3) 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 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 4 outlines these functions and their purposes.



Figure 1: Three 2-D projections from 4-D, for the ‘expbranches’ (a1-a3), ‘linearbranches’ (b1-b3), ‘curvybranches’ (c1-c3), ‘orglinearbranches’ (d1-d3), ‘orgcurvybranches’ (e1-e3) data.



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

Table 4: cardinalR cube data generation functions

Function	Explanation
gen_gridcube	Generate a cube with specified grid points along each axes.
gen_unifcube	Generate a cube with uniform points.
gen_cubehole	Generate a cube with a hole.

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.

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 3 b).

Finally, we consider a cube that has a central spherical hole. This is generated using the `gen_cubehole(n, p, r_hole)` function. The cube is generated as per the uniformly distributed cube, but points inside sphere of radius (r_{hole}) are removed, resulting in a hollow cube structure (Figure 3 c).

```
gridcube <- gen_gridcube(n = 1000, p = 4)
unifcube <- gen_unifcube(n = 1000, p = 4)
cubehole <- gen_cubehole(n = 3000, p = 4, r_hole = 0.5)
```

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))
```

Linear

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

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



Figure 3: Three 2-D projections from 4-D, for the ‘gridcube’ (a1-a3), ‘unifcube’ (b1-b3), and ‘cubehole’ (c1-c3) data.



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



Figure 5: Three 2-D projections from 4-D, for the ‘linear’ data.

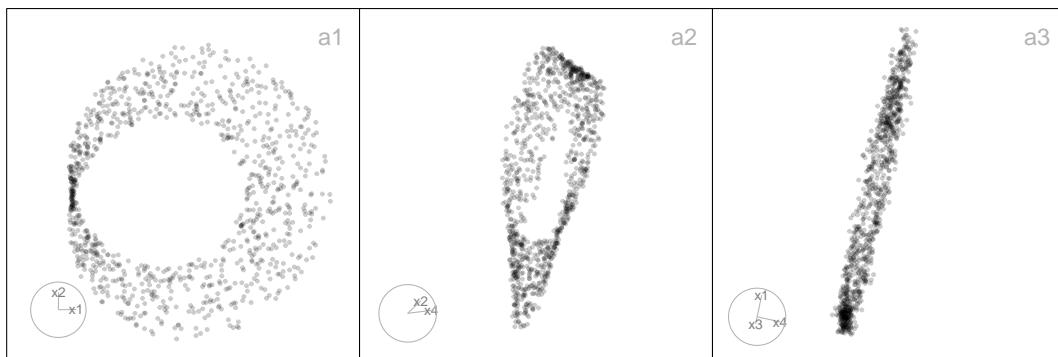


Figure 6: Three 2-D projections from 4-D, for the ‘mobius’ data.

Mobius

The `gen_mobius(n, p)` function generates a dataset of n points that form a Möbius strip embedded in the first three dimensions of a p -D structure (Figure 6). This classical non-orientable surface loops back on itself with a half-twist.

The Möbius strip is generated using the `geozoo::mobius` function with $p = 3$, and n defined as above. Each point is sampled from a parameterization defined by an angle $\theta \sim U(0, 2\pi)$, which specifies the position along the circular loop, and a width $w \sim U(-0.4, 0.4)$, which offsets the point across the strip’s band. The cartesian coordinates are $X_1 = (1 + (w/2) \cos(\theta/2)) \cos(\theta)$, $X_2 = (1 + (w/2) \cos(\theta/2)) \sin(\theta)$, and $X_3 = (w/2) \sin(\theta/2)$. This maps a 2-D band with a half-twist into 3-D space, forming a non-orientable one-sided surface. For $p > 3$, Gaussian noise $X_j \sim N(0, 0.05^2)$ is added to embed the 3-D Möbius into p -D, where $j = 4, \dots, p$.

```
mobius <- gen_mobius(n = 1000, p = 4)
```

Polynomial

A polynomial structure (Figure 7) generates data points that follow non-linear curvilinear relationships, such as quadratic or cubic trends, in high-dimensional space. 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 5 summarizes these functions and their purposes.

Table 5: cardinalR polynomial data generation functions

Function	Explanation
<code>gen_quadratic</code>	Generate a quadratic pattern.
<code>gen_cubic</code>	Generate a cubic pattern.

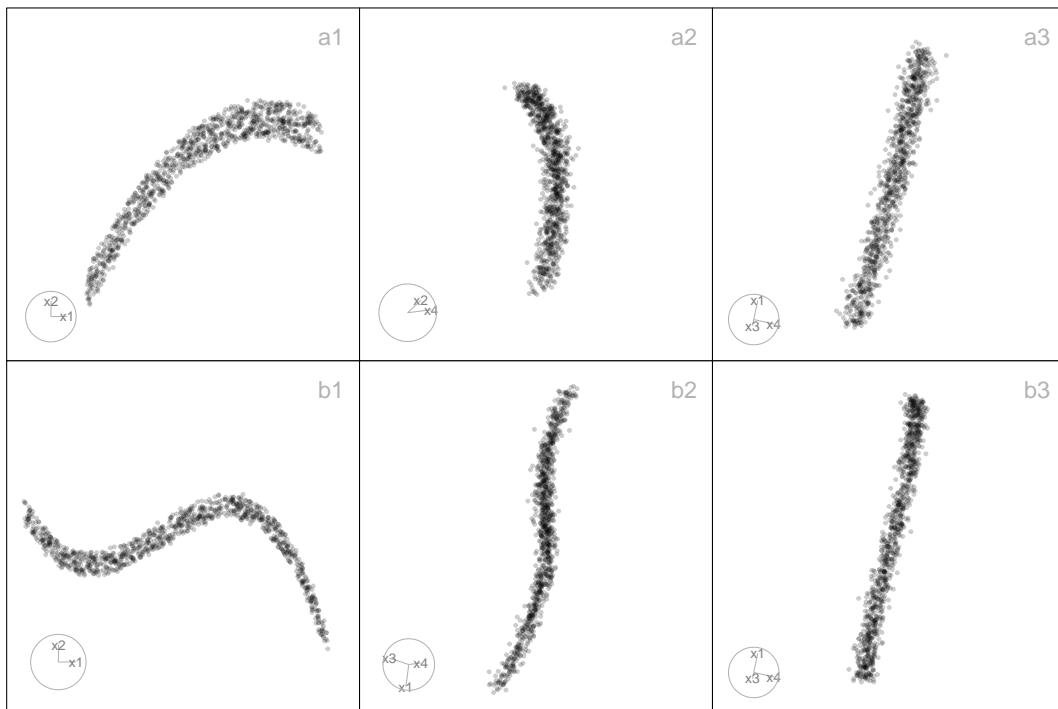


Figure 7: Three 2-D projections from 4-D, for the ‘quadratic’ (a1-a3) and ‘cubic’ (b1-b3) data.

gen_quadratic() The `gen_quadratic(n, p, range)` function generates a dataset of n points forming a quadratic curve in the first two dimensions (Figure 7 a). This 2-D parabolic structure is embedded within a higher-dimensional space with additive noise in the remaining dimensions.

The curve is constructed by randomly sampling inputs and applying a second-degree polynomial transformation. Let $X_1 \sim U(\text{range}[1], \text{range}[2])$ be the independent variable. A raw polynomial basis of degree 2 is used to form $X_2 = X_1 - X_1^2 + \varepsilon_2$, where $\varepsilon_2 \sim U(0, 0.5)$. This creates a smooth parabolic arc that opens downward, with jitter in the vertical direction to simulate noise. For $p > 2$, Gaussian noise $X_j \sim N(0, 0.1^2)$ is added to embed the 2-D structure into p -D, where $j = 3, \dots, p$.

```
quadratic <- gen_quadratic(n = 1000, p = 4)
```

gen_cubic() The `gen_cubic(n, p, range)` function generates a dataset of n points forming a cubic curve in the first two dimensions (Figure 7 b). This function creates a more complex curvilinear structure than a simple parabola.

The shape is generated using a third-degree raw polynomial basis expansion. Let $X_1 \sim U(\text{range}[1], \text{range}[2])$ be the base input. A cubic transformation with vertical jitter is used to define $X_2 = X_1 + X_1^2 - X_1^3 + \varepsilon_2$, where $\varepsilon_2 \sim U(0, 0.5)$. For $p > 2$, Gaussian noise $X_j \sim N(0, 0.1^2)$ is added to embed the 2-D structure into p -D, where $j = 3, \dots, p$.

```
cubic <- gen_cubic(n = 1000, p = 4)
```

Pyramid

A pyramid structure (Figure 8) 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 bases, and sharp or blunted apexes. Additionally, similar to the cube generator, it is possible to create a pyramids containing structured holes, enabling the study of non-convex and sparse regions. Table 6 summarizes these functions.

Let X_1, \dots, 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,

$$X_p = z \sim \min(\text{Exp}(\lambda = 2/h), h)$$

Table 6: 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_pyrholes	Pyramid containing triangular pyramid-shaped holes.

. 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 8 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 8 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 8 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: $\theta \sim \text{Uniform sample from 6 hexagon angles}$, $r_{\text{point}} \sim \sqrt{U(0, 1)}$. 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 all pyramid shapes, if $p > 3$, the remaining $p - 4$ dimensions (i.e., X_4 to X_{p-1}) are tapered toward zero with decreasing height and generated as: $X_j \sim U(-0.1, 0.1)(1 - z/h)$, for $j = 4, \dots, p - 1$. This tapering ensures that the structure narrows in all directions as it approaches the tip.

```
pyrrect <- gen_pyrrect(n = 1000, p = 4)
pyrtri <- gen_pyrtri(n = 1000, p = 4)
pyrstar <- gen_pyrstar(n = 1000, p = 4)
```

gen_pyrholes() The `gen_pyrholes(n, p)` function generates n points embedded in a p -D simplex using a chaotic attractor-like midpoint algorithm (Figure 8 d). The result is a fractal-like structure that reveals holes or gaps in the data cloud, forming a "Sierpinski-like pyramid" in high dimensions.

Let X_1, X_2, \dots, 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, \dots, C_{p+1} denote the corner vertices of a p -D simplex. At each iteration $i = 1, \dots, n$, a new point is computed by taking the midpoint between the previous point T_{i-1} and a randomly selected vertex C_k : $T_i = 1/2(T_{i-1} + C_k)$, $C_k \in \{C_1, \dots, C_{p+1}\}$. 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: $X = \{T_1, T_2, \dots, T_n\}$, $X \in \mathbb{R}^{n \times p}$.

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

S-curve

An S-curve structure (Figure 9) simulates data that lies along a smooth, non-linear manifold, making it a classic example for testing the performance of non-linear dimension reduction methods. This

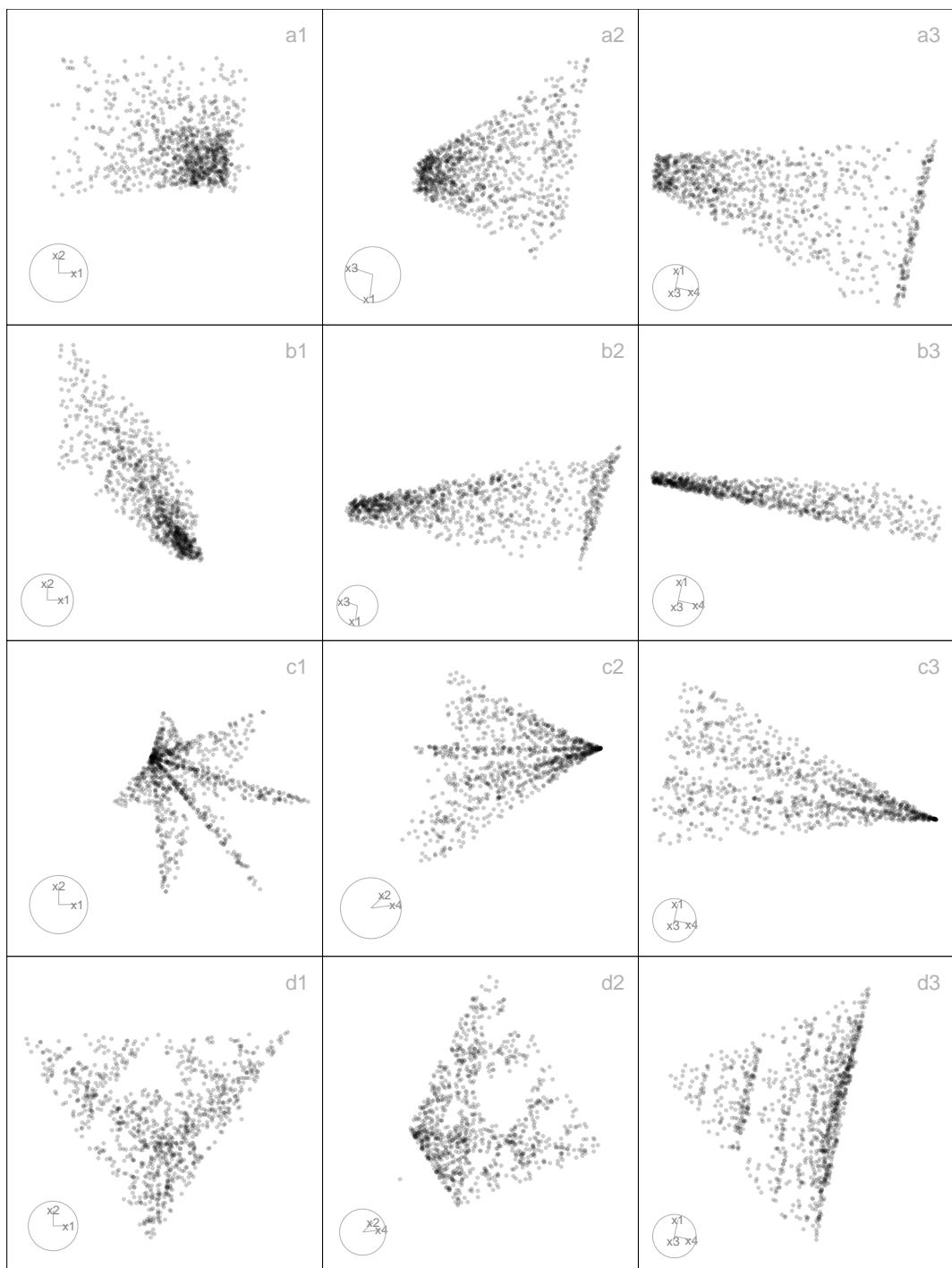


Figure 8: Three 2-D projections from 4-D, for the ‘pyrrect’ (a1-a3), ‘pyrtri’ (b1-b3), ‘pyrstar’ (c1-c3) and ‘pyrholes’ (d1-d3) data.

Table 7: cardinalR S-curve data generation functions

Function	Explanation
gen_scurve	Generate a S-curve.
gen_scurvehole	Generate a S-curve with a hole.

Table 8: cardinalR sphere data generation functions

Function	Explanation
gen_circle	Generate a circle.
gen_curvycycle	Generate a curvy cell cycle.
gen_unifsphere	Generate a uniform sphere.
gen_griddedsphere	Generate a grided sphere.
gen_clusteredspheres	Generate multiple small spheres within a big sphere.
gen_hemisphere	Generate a hemisphere.

package provides functions to generate standard S-curve shapes as well as variants containing holes to introduce missing or incomplete regions. These variations are useful for evaluating how well algorithms capture non-linear geometry and handle incomplete manifolds in high-dimensional data. Table 7 summarizes these functions.

gen_scurve() To simulate an S-curve structure in a higher-dimensional space, we define the function `gen_scurve(n, p)`, which generates n observations in p -D (Figure 9 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: $X_1 = \sin(\theta)$, $X_2 \sim U(0, 2)$, $X_3 = \text{sign}(\theta)(\cos(\theta) - 1)$. This configuration creates a horizontally curled shape in (X_1, X_3) , with additional band thickness in the X_2 direction. For $p > 3$, additional noise dimensions are appended introducing structured, wavy noise.

```
scurve <- gen_scurve(n = 1000, p = 4)
```

gen_scurvehole() To simulate a variant of the S-curve structure with a removed region (“hole”), we define the function `gen_scurvehole(n, p)` (Figure 9 b). This function produces n observations in p -D where the first three dimensions describe the S-curve manifold, and remaining dimensions add low-variance Gaussian noise. A subset of observations near a designated anchor point is excluded to introduce a hole in the manifold.

To simulate missing regions on the manifold, a fixed anchor point $(0, 1, 0, \dots)$ is defined in p -D. All observations within a Euclidean distance of $\sqrt{0.3} \approx 0.5477$ from the anchor are removed if $\sum_{j=1}^p (X_j - a_j)^2 \leq 0.3$. This exclusion creates a hole in the manifold centered near the middle vertical region of the S-curve.

```
scurvehole <- gen_scurvehole(n = 1000, p = 4)
```

Sphere

Sphere-shaped structures (Figure 10) are useful for evaluating how dimension reduction and clustering algorithms handle curved, symmetric manifolds in high-dimensional spaces. The package provides functions to generate a variety of spherical forms, including simple circles, uniform spheres, grid-based spheres, and complex arrangements like clustered spheres within a larger sphere. These shapes help test how well methods preserve symmetry, curvature, and cluster separation. Table 8 summarizes these functions.

gen_circle() The function `gen_circle(n, p)` generates a p -dimensional dataset of n observations, where the first two dimensions form a unit circle, and the remaining dimensions are structured sinusoidal extensions of the angular parameter with progressively smaller scale (Figure 10 a).

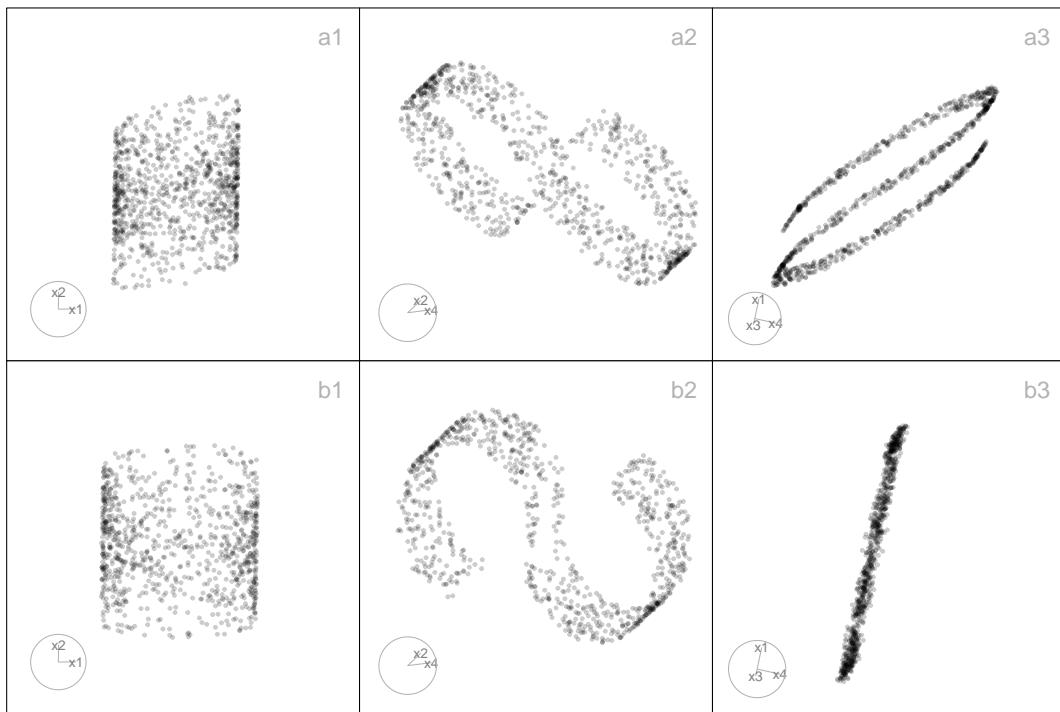


Figure 9: Three 2-D projections from 4-D, for the ‘scurve’ (a1-a3) and ‘scurvehole’ (b1-b3) data.

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: $X_1 = \cos(\theta)$, $X_2 = \sin(\theta)$. 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 $j = 3, \dots, p$. 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: $X_j = \text{scale}_j \sin(\theta + \phi_j)$, $j = 3, \dots, p$.

```
circle <- gen_circle(n = 1000, p = 4)
```

gen_curvycycle() The `gen_curvycycle(n, p)` function generates a p -dimensional dataset of n observations lying on a curved closed loop with controlled high-dimensional sinusoidal deviations (Figure 10 b). The structure forms a one-dimensional nonlinear cycle embedded in higher-dimensional space.

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 $\phi_j = (j - 2)\pi/2p$.

```
curvycycle <- gen_curvycycle(n = 1000, p = 4)
```

gen_unifsphere() The `gen_unifsphere(n, p, r)` function generates a p -dimensional dataset of n observations distributed approximately uniformly on the surface of a 3-D sphere of radius r , with additional Gaussian noise dimensions added when $p > 3$ (Figure 10 c).

Each observation lies on the surface of a sphere in 3-D, constructed by generating $u \sim U(-1, 1)$ which represents the cosine of the polar angle ϕ and $\theta \sim U(0, 2\pi)$ which represents the azimuthal angle. The corresponding Cartesian coordinates are calculated as; $X_1 = r\sqrt{1-u^2} \cos(\theta)$, $X_2 = r\sqrt{1-u^2} \sin(\theta)$, $X_3 = ru$, which gives points uniformly distributed on the surface of a 3-D sphere (not within). For $p > 3$, additional dimensions X_4 to X_p are generated as low-variance Gaussian noise: $X_j \sim N(0, 0.05^2)$, for $j = 4, \dots, p$.

```
unifsphere <- gen_unifsphere(n = 1000, p = 4)
```

gen_griddedsphere() The `gen_griddedsphere(n, p)` function generates a p -dimensional dataset of approximately n points evenly distributed on the surface of a $(p!-!1)$ -sphere (the unit sphere embedded in \mathbb{R}^p) (Figure 10 d).

The construction is based on a regular grid in spherical coordinates with $(p - 1)$ angular variables. Specifically, for dimensions $j = 1, \dots, p - 2$, the polar angles are defined on $[0, \pi]$. For the last angle ($j = p - 1$), the azimuthal angle is defined on $[0, 2\pi]$.

The number of grid steps along each angular dimension is determined by factoring n into $(p - 1)$ approximately equal integers via `gen_nproduct(n, p - 1)`.

Each grid point is then mapped to Cartesian coordinates using the standard hyperspherical-to-Cartesian transformation:

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

Thus, the output is a deterministic grid of points lying exactly on the unit $(p - 1)$ -sphere, with no added noise dimensions.

```
griddedsphere <- gen_griddedsphere(n = 1000, p = 4)
```

gen_clusteredspheres() The `gen_clusteredspheres(n, k, p, r, loc)` function generates a synthetic dataset of $n_1 + kn_2$ observations in p -dimensional space, consisting of one large sphere of radius r_1 and k smaller spheres of radius r_2 , each centered at a different random location (Figure 10 e).

A large uniform sphere 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, p, r_1)`, which generates the desired points in the specified dimensional space.

In generation of k smaller uniform spheres, 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 $N(0, 1\text{oc}^2 I_p)$.

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.

```
clusteredspheres <- gen_clusteredspheres(n = c(1000, 100), k = 3, p = 4, r = c(15, 3),
                                         loc = 10 / sqrt(3)) |>
  dplyr::select(-cluster)
```

gen_hemisphere() The `gen_hemisphere(n, p)` function generates a p -dimensional dataset of n observations distributed approximately uniformly on a 4-D hemisphere, optionally extended with Gaussian noise in additional dimensions when $p > 4$ (Figure 10 f).

Each observation is situated on a restricted 4-D spherical surface, defined by 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: $X_1 =$

Table 9: cardinalR trigonometric data generation functions

Function	Explanation
gen_crescent	Generate a crescent pattern.
gen_curvycylinder	Generate a curvy cylinder.
gen_sphericalspiral	Generate a spherical spiral.
gen_helicalspiral	Generate a helical spiral.
gen_conicspiral	Generate a conic spiral.
gen_nonlinear	Generate a nonlinear hyperbola.

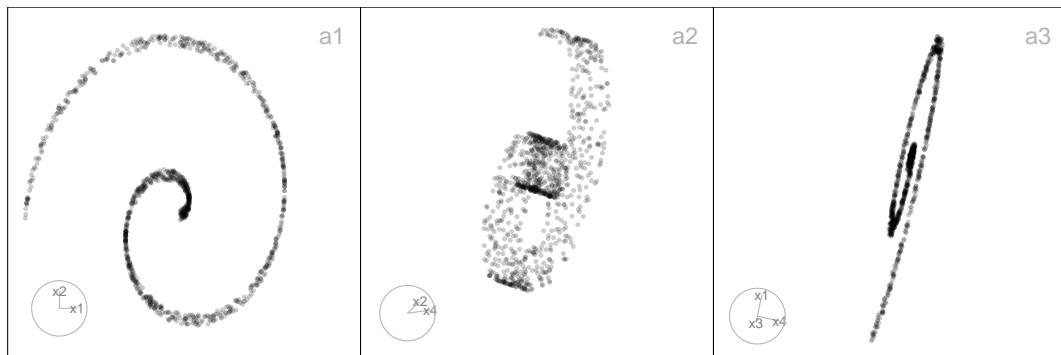
$\sin(\theta_1) \cos(\theta_2)$, $X_2 = \sin(\theta_1) \sin(\theta_2)$, $X_3 = \cos(\theta_1) \cos(\theta_3)$, $X_4 = \cos(\theta_1) \sin(\theta_3)$. This produces points on one side of a 4-D unit sphere, effectively generating a 4-D hemisphere. For $p > 4$, additional dimensions X_5 to X_p are added as low-variance Gaussian noise: $X_j \sim N(0, 0.05^2)$, for $j = 5, \dots, p$.

```
hemisphere <- gen_hemisphere(n = 1000, p = 4)
```

Swiss Roll

To further generalize the Swiss roll structure and introduce realistic noise, we define a function `gen_swissroll(n, p, w)`, where n is the number of points, p is the total number of dimensions, and w is the vertical range in the third dimension (Figure 11). The first three dimensions form the classic 3-D Swiss roll shape. The $X_1 = t \cos(t)$, $X_2 = t \sin(t)$, $X_3 \sim U(w_1, w_2)$, where $t \sim U(0, 3\pi)$. For $p > 3$, the remaining $p - 3$ dimensions are filled with small Gaussian noise to simulate high-dimensional complexity.

```
swissroll <- gen_swissroll(n = 1000, p = 4, w = c(-1, 1))
```

**Figure 11:** Three 2-D projections from 4-D, for the ‘swissroll’ data.

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 12). The package includes functions to generate a wide range of these forms from crescents and curvy cylinders to spherical, helical, and conic spirals, as well as nonlinear hyperbolas. These structures are particularly valuable for testing how well dimension reduction and clustering algorithms preserve intricate geometric and topological features. Table 9 summarizes these functions.

gen_crescent() 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 12 a).

The first two dimensions define a crescent-shaped curve using a semi-circular arc. Let $\theta \in [\pi/6, 2\pi]$ be a sequence of n evenly spaced angles. The corresponding 2-D coordinates are defined by: $X_1 = \cos(\theta)$, $X_2 = \sin(\theta)$. For $p > 2$, additional dimensions X_3 to X_p are generated using

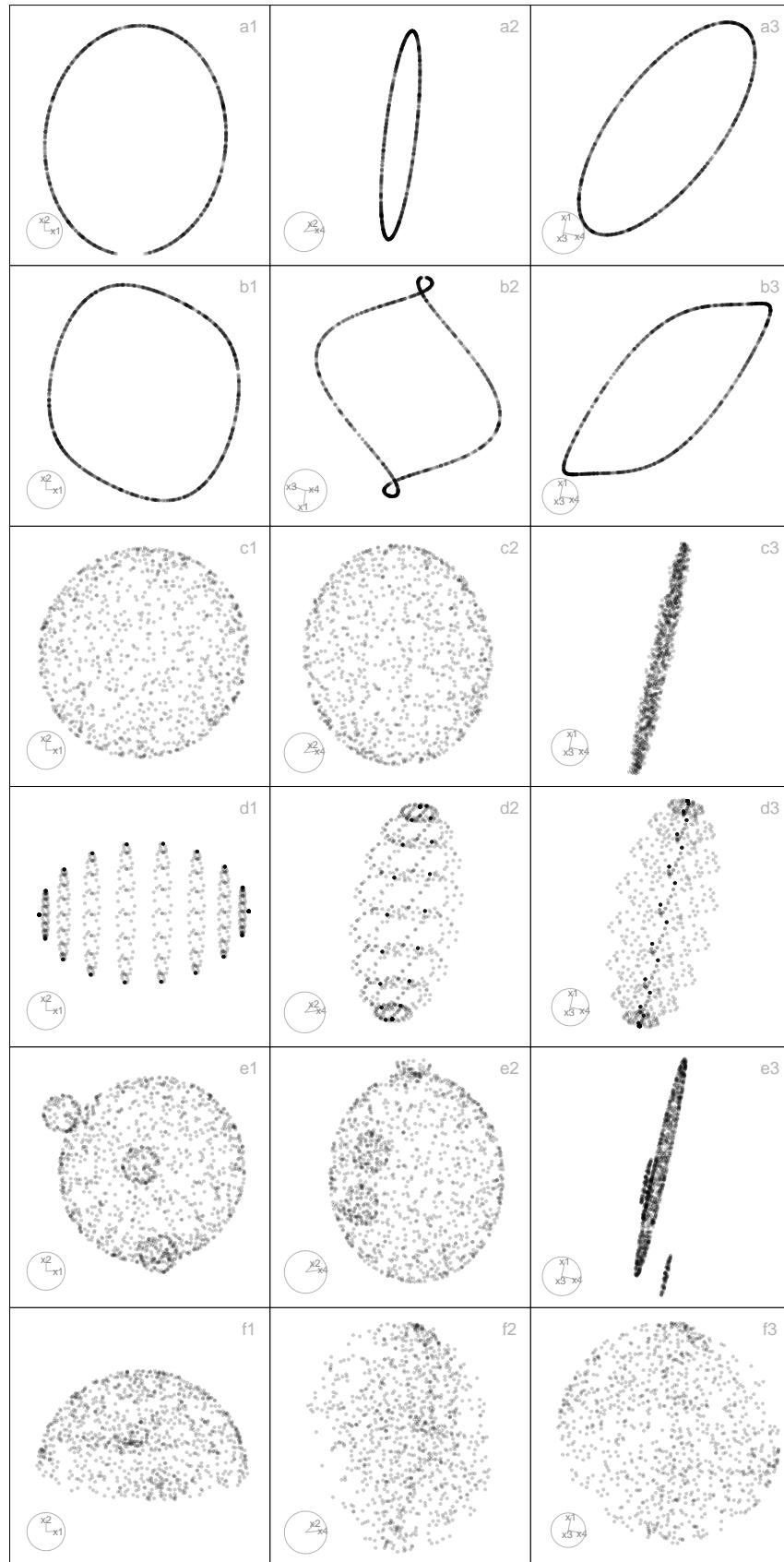


Figure 10: Three 2-D projections from 4-D, for the ‘circle’ (a1-a3), ‘curvycycle’ (b1-b3), ‘unifsphere’ (c1-c3), ‘griddedsphere’ (d1-d3), ‘clusteredspheres’ (e1-e3), and ‘hemisphere’ (f1-f3) data.

gen_wavydims1(): $X_j = s_j \theta + \varepsilon_j$, where $\varepsilon_j \sim N(0, 0.5^2)$, and s_j is a random scale. These noise dimensions are nonlinearly correlated with θ , producing structured, wave-like variation that aligns with the progression along the crescent curve.

```
crescent <- gen_crescent(n = 1000, p = 4)
```

gen_curvycylinder() 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 12 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). This forms a curvy cylindrical surface in 4-D, where the fourth dimension bends periodically along the height axis, resembling a helicoid or twisting wave along the cylinder. For $p > 4$, dimensions X_5 through X_p are generated as: $X_j \sim N(0, 0.05^2)$, for $j = 5, \dots, p$.

```
curvycylinder <- gen_curvycylinder(n = 1000, p = 4, h = 10)
```

gen_sphericalspiral() The `gen_sphericalspiral(n, p, spins)` function simulates a dataset of n observations that form a spiraling path on a spherical surface in the first four dimensions. When extended beyond 4-D, structured nonlinear noise dimensions are added to simulate more realistic high-dimensional manifolds (Figure 12 c).

The first three dimensions represent points on a unit sphere. 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.

If $p > 4$, the function appends structured, non-linear noise via `gen_wavydims2()`. These dimensions are functions of the first coordinate X_1 , introducing dependencies that preserve some geometric coherence. Each added dimension follows the form: $X_j = s_j (-1)^{\lfloor j/2 \rfloor} X_1^{a_j} + \eta_j$, where $a_j \in \{2, 3, 4, 5\}$ is a randomly chosen polynomial power, $s_j \sim U(0.5, 2)$ is a scale factor, $\eta_j \sim U(-\sigma_j, 2\sigma_j)$, with $\sigma_j \sim U(0, 0.05)$, adds mild randomness.

```
sphericalspiral <- gen_sphericalspiral(n = 1000, p = 4, spins = 1)
```

gen_helicalspiral() The `gen_helicalspiral(n, p)` function generates a dataset of n observations forming a helical spiral, embedded in the first four dimensions of p -D. Additional dimensions (if $p > 4$) are filled with structured noise to mimic high-dimensional complexity (Figure 12 d).

The first four coordinates follow a 3-D helix with an additional oscillating component. 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. This results in a helical spiral winding upward along the third axis, with gentle sinusoidal fluctuation in the fourth dimension. When $p > 4$, the remaining $p - 4$ dimensions are populated using `gen_noisedims()`. These are independent Gaussian noise dimensions: $X_j \sim N(0, 0.05^2)$, $j = 5, \dots, p$.

```
helicalspiral <- gen_helicalspiral(n = 1000, p = 4)
```

gen_conicspiral() 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 12 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,

Table 10: cardinalR trefoil data generation functions

Function	Explanation
gen_trefoil4d	Generate a trefoil in \$4\text{\textbackslash}text{-}D\$.
gen_trefoil3d	Generate a trefoil in \$3\text{\textbackslash}text{-}D\$.

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. This results in a 3-D spiral surface expanding upward and outward like a cone, with an oscillatory fourth dimension capturing spiral irregularities. For $p > 4$, the dataset includes isotropic Gaussian noise in the remaining dimensions via `gen_noisedims()`: $X_j \sim N(0, 0.05^2)$, $j = 5, \dots, p$.

```
conicspiral <- gen_conicspiral(n = 1000, p = 4, spins = 1)
```

gen_nonlinear() 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 12 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. If $p > 4$, the remaining dimensions are populated with Gaussian noise using `gen_noisedims()`: $X_j \sim N(0, 0.05^2)$, $j = 5, \dots, p$.

```
nonlinear <- gen_nonlinear(n = 1000, p = 4, hc = 1, non_fac = 0.5)
```

Trefoil knots

The Trefoil is a closed, nontrivial one-dimensional manifold embedded in 3-D or 4-D space (Figure 13). The trefoil features topological complexity in the form of self-overlaps, making it a valuable test case for evaluating the ability of NLDR methods to preserve global structure, loops, and embeddings in high-dimensional data. Table 10 summarizes these functions.

gen_trefoil4d() The function `gen_trefoil4d()` generates a 4-D trefoil knot using two angular parameters, θ and ϕ . The knot is constructed on the 3-sphere ($S^3 \subset \mathbb{R}^4$), with a band of thickness around the knot path, controlled by `steps` (Figure 13 a).

The coordinates of the 4-D trefoil knot are defined as $X_1 = \cos(\theta) \cos(\phi)$, $X_2 = \cos(\theta) \sin(\phi)$, $X_3 = \sin(\theta) \cos(1.5\phi)$, and $X_4 = \sin(\theta) \sin(1.5\phi)$, where θ and ϕ are angular parameters that trace out the knot's structure on the 3-sphere $S^3 \subset \mathbb{R}^4$.

The number of theta and phi values is determined by the `steps` and `n` arguments, respectively. Additional independent noise dimensions can be added via `gen_noisedims()` to extend the structure into $p > 4$ dimensions.

```
trefoil4d <- gen_trefoil4d(n = 500, p = 4, steps = 5)
```

gen_trefoil3d() The function `gen_trefoil3d()` generates a 3-D stereographic projection of the 4-D trefoil knot by mapping 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 the transformation $X'_1 = X_1/(1 - X_4)$, $X'_2 = X_2/(1 - X_4)$, and $X'_3 = X_3/(1 - X_4)$, while excluding points where $X_4 = 1$ to avoid division by zero (Figure 13 b).

```
trefoil3d <- gen_trefoil3d(n = 500, p = 4, steps = 5)
```

As with the 4-D version, this function supports optional noise dimensions to embed the projected knot into higher dimensions.



Figure 12: Three 2-D projections from 4-D, for the ‘crescent’ (a1-a3), ‘curvycylinder’ (b1-b3), ‘sphericspiral’ (c1-c3), ‘helicalspiral’ (d1-d3), ‘conicspiral’ (e1-e3), and ‘nonlinear’ (f1-f3) data.

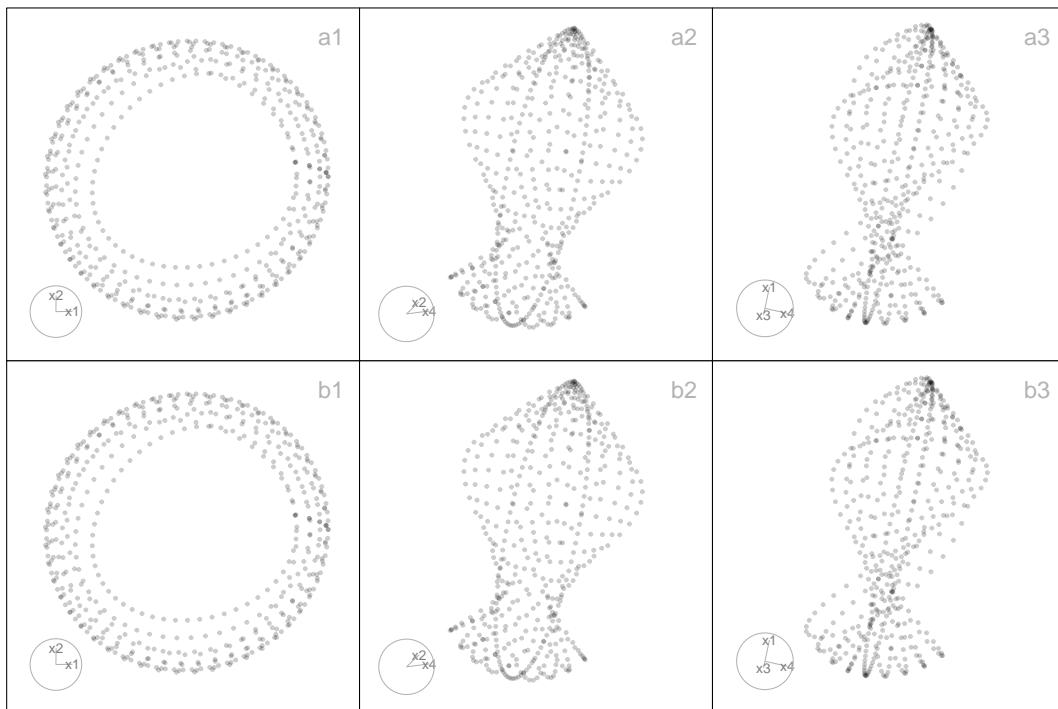


Figure 13: Three 2-D projections from 4-D, for the ‘trefoil4d’ (a1-a3) and ‘trefoil3d’ (b1-b3) data.

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 11.

Additional functions

The package includes various supplementary tools in addition to the shape generating functions mentioned earlier. These tools allow users to generate noise dimensions with a normal distribution and various wavy patterns, create background noise, randomize the rows of the data, reposition clusters, generate a vector whose product and sum are approximately equal to a target value, rotate structures, and normalize the data. Table 12 details these functions.

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 14).

```

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

## To generate data
five_clusts <- gen_multiclus(n = c(2250, 1500, 750, 1250, 1750), p = 4, k = 5,
                               loc = positions,
                               scale = c(0.4, 0.35, 0.3, 1, 0.3),
                               shape = c("helicalspiral", "hemisphere", "unifcube",
                                         "cone", "gaussian"),

```

Table 11: cardinalR multiple clusters generation functions

Function	Explanation
make_mobiusgau	Generates a Möbius-like cluster combined with a Gaussian cluster.
make_multigau	Generates multiple Gaussian clusters in high-dimensional space.
make_curvygau	Generates a curvilinear cluster with a Gaussian cluster.
make_klink_circles	Generates K-link circular clusters (non-linear circular patterns).
make_chain_circles	Generates chain-like circular clusters connected sequentially.
make_klink_curvycycle	Generates K-link curvy cycle clusters (curvilinear loop structures).
make_chain_curvycycle	Generates chain-like curvy cycle clusters connected sequentially.
make_gaucircles	Generates circular clusters with a Gaussian cluster in the middle.
make_gaucurvycycle	Generates curvy circular clusters with a Gaussian cluster in the middle.
make_onegrid	Generates a single grid in two dimensions.
make_twogrid_overlap	Generates two overlapping grids.
make_twogrid_shift	Generates two grids shifted relative to each other.
make_shape_para	Generates parallel shaped clusters.
make_three_clust_	Generates three clusters with different shapes. (eg:- 01, 02, ..., 20)

Table 12: cardinalR additional functions

Function	Explanation
gen_noisedims	Generates additional noise dimensions.
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_wavydims1	Generates random noise dimensions with wavy pattern generated with theta.
gen_wavydims2	Generates random noise dimensions with wavy pattern generated with power functions.
gen_wavydims3	Generates random noise dimensions with wavy pattern generated with power and sine functions.
gen_rotation	Generates rotations.
normalize_data	Normalizes data.

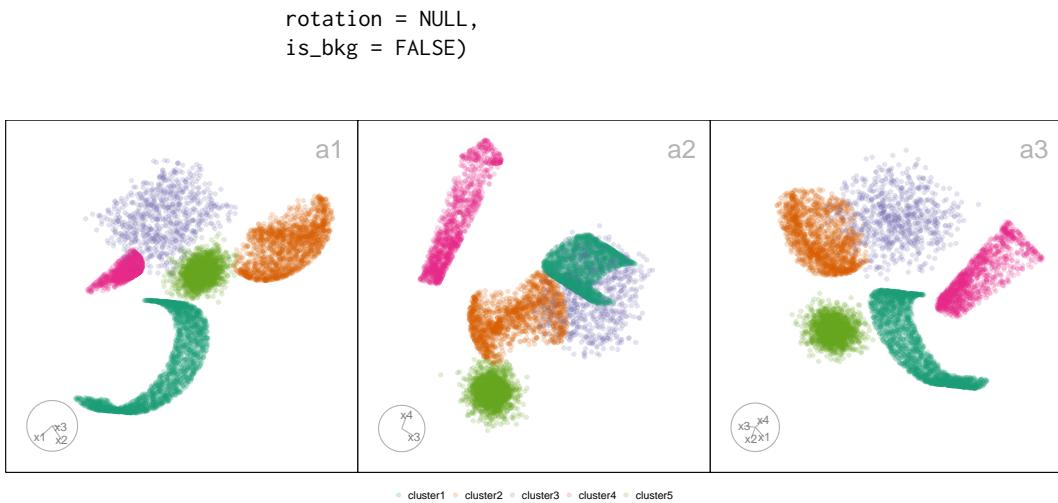


Figure 14: Three 2-D projections from 4-D, for the five clusters data. The helical spiral cluster is represented in dark green, the hemisphere cluster in orange, the uniform cube-shaped cluster in purple, the blunted cone cluster in pink, and the Gaussian-shaped cluster in light green.

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.

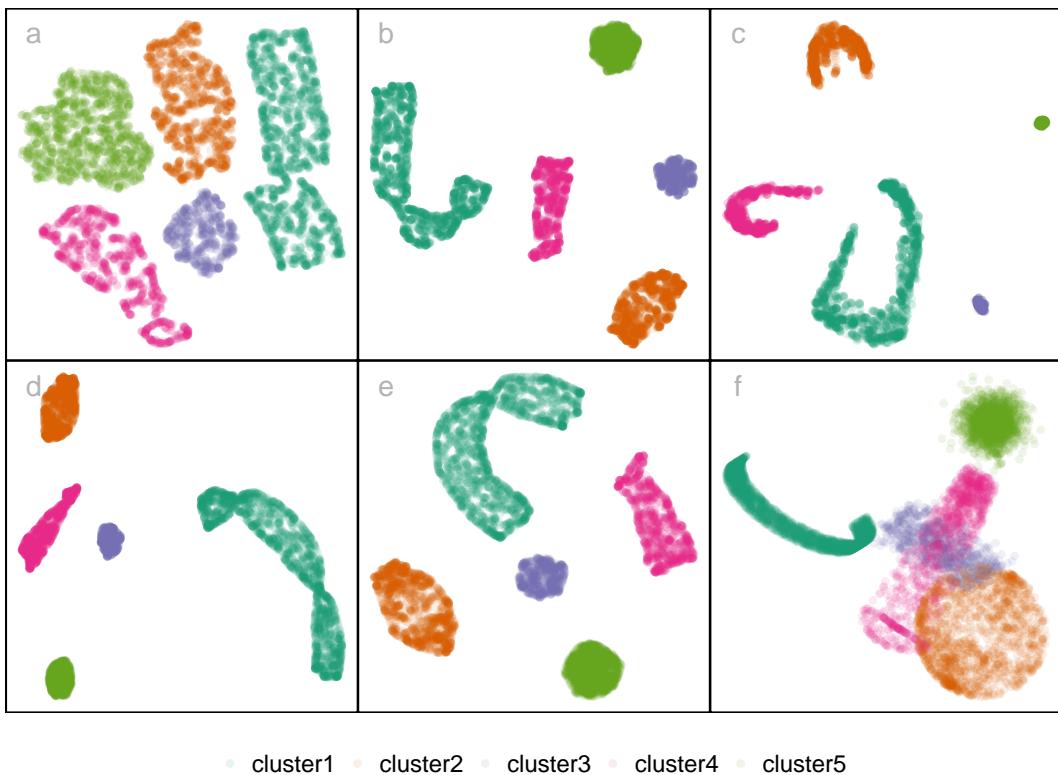


Figure 15: Six different dimension reduction representations of the five clusters data using default hyperparameter settings: (a) tSNE, (b) UMAP, (c) PAHTE, (d) TriMAP, (e) PaCMAP, and (f) PCA.

UMAP, PHATE, TriMAP, and PaCMAP effectively separate the five clusters and show the preservation of the global structure (Figure 15). 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.

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 dimensionality reduction (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 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 <https://github.com/JayaniLakshika/paper-cardinalR>.

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). This article is created using `knitr` (Xie, 2015) and `rmarkdown` (Xie et al., 2018) in R with the `rjtools::rjournal_article` template.

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>. [p20]
- G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2025. URL <https://CRAN.R-project.org/package=cli>. R package version 3.6.4. [p23]
- 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. [p23]
- 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. [p20]
- 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. [p20]
- 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>. [p20]
- J. Melville. *snedata: SNE Simulation Dataset Functions*, 2025. URL <https://github.com/jlmelville/snedata>. R package version 0.0.0.9001, commit beebebcf91c365bf5006be08fb614585b4659c05c5. [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. [p20]
- K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2023. URL <https://CRAN.R-project.org/package=tibble>. R package version 3.2.1. [p23]
- 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/>. [p23]
- B. Schloerke. *geozoo: Zoo of Geometric Objects*, 2016. URL <https://CRAN.R-project.org/package=geozoo>. R package version 0.5.1. [p1, 23]
- 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. [p23]
- 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/v22/201061.html>. [p20]
- G. R. Warnes, B. Bolker, T. Lumley, A. Magnusson, B. Venables, G. Rydon, and S. Moeller. *gtools: Various R Programming Tools*, 2023. URL <https://CRAN.R-project.org/package=gtools>. R package version 3.9.5. [p23]
- H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2025. URL <https://CRAN.R-project.org/package=purrr>. R package version 1.0.4. [p23]
- 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. [p23]
- H. Wickham, D. Vaughan, and M. Girlich. *tidy: Tidy Messy Data*, 2024. URL <https://CRAN.R-project.org/package=tidyr>. R package version 1.3.1. [p23]
- 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. [p23]
- 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. [p23]
- 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