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. For example, `geozoo` (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 `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, `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 particularly simulate complex biological data, effectively capturing nuances such as batch effects and differential expression.

There is a valuable opportunity to improve the generation of high-dimensional data structures by integrating geometric principles with advanced noise control and customizable clustering. The `geozoo` package provides a strong foundation but could be enhanced to support high-dimensional extensions with controlled noise and user-defined parameters for clustering. Similarly, while `snedata` focuses on abstract datasets for dimensionality reduction, adding features for generating high-dimensional data from geometric layouts would enhance its usability. The `mlbench` package could also benefit from allowing users to create datasets with specific geometric structures and noise profiles. Additionally, although `splatter` specializes in biological data simulation, it could be expanded to offer a broader framework for generating diverse geometric structures across dimensions, enabling detailed control over noise and clustering. Addressing these areas could lead to more robust high-dimensional data generation tools.

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) create complex clustered data arrangements by using fundamental geometric forms, while maintaining their positions, scales, orientations, and sample sizes in arbitrary dimensional spaces. By providing these integrated features, `cardinalR` aims to provide researchers 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 next section, introduces the implementation of `cardinalR` package on CRAN and GitHub, including demonstration of the package's key functions. We illustrate how a clustering data structure affect the dimension reductions in **Application** section. Finally, we give a brief conclusion of the paper and discuss potential opportunities for use of our data collection.

Table 1: The main arguments for gen_multicluster().

Argument	Explanation
n	A numeric vector representing the number of points in each cluster.
p	A numeric value representing the number of dimensions.
k	A numeric value representing the number of clusters.
loc	A numeric matrix representing the locations/centroids of clusters.
scale	A numeric vector representing the scaling factors of clusters.
shape	A character vector representing the shapes of clusters.
rotation	A numeric list which contains plane and the corresponding angle along that plane for each cluster.
is_bkg	A Boolean value representing the background noise should exist or not.

2 Implementation

Installation

The development version can be installed from GitHub:

```
devtools::install_github("JayaniLakshika/cardinalR")
```

Usage

Main function

The main function of the package is `gen_multicluster()`. This function generates clusters of various shapes, allowing users to specify the number of points in each cluster, as well as their locations, scaling, and rotations across specific dimensions. Additionally, users can add background noise into the generated data by using the `is_bkg` option.

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

Branching

A branching structure captures trajectories that diverge or bifurcate from a common origin, similar 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 geometry, number of points, and the number of branches. Table 2 outlines these functions. The main arguments of the functions described in Table 3.

`gen_expbranches()` The `gen_expbranches(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. 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 alternates between branches (odd: negative, even: positive exponent). This creates mirror-symmetric branches with different steepness and curvature where odd-numbered branches: decay pattern (reflected) and even-numbered branches: growth pattern. 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 $j = 3, \ldots, p$.

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

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.

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

Each branch is a segment of a line with added jitter to simulate measurement noise. The branches differ in direction and location. Branches 1 and 2 are initialized with fixed slopes and intercepts. The Branch 1 is generated from $X_1 \sim U(-2, 8)$, $X_2 = 0.5X_1 + \epsilon$, where $\epsilon \sim U(0, 0.5)$. The Branch 2 is generated from $X_1 \sim U(-6, 2)$, $X_2 = -0.5X_1 + \epsilon$, where $\epsilon \sim U(0, 0.5)$. Branches 3 to k are added iteratively. Each additional branch i starts at a location outside predefined exclusion zones to avoid overlap with the initial two branches. The X_1 values are defined over a short range, from x_{start} to $x_{start} + 1$. The X_2 value is calculated using the formula $X_2 = s_i(X_1 - x_{start}) + y_{start} + \epsilon$, where s_i is a chosen slope from a selected branch, 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, where $j = 3, \ldots, p$.

```
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. 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 $X_2 = 0.1 \cdot X_1 + 1 \cdot X_1^2 + \epsilon$, 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 $X_2 = 0.1 \cdot X_1 - 2 \cdot X_1^2 + \epsilon$, where $\epsilon \sim U(0, 0.05)$. This creates a steeper, leftward-facing curve in the left half-plane. Branches 3 to k are added iteratively. Each new branch i begins at a

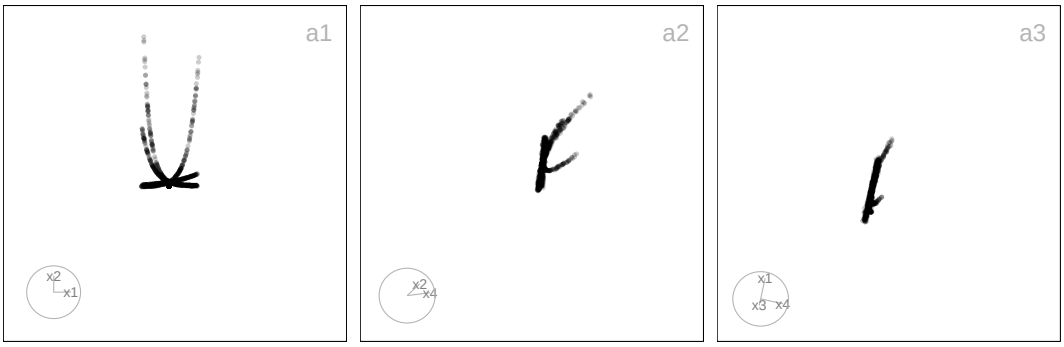

Figure 1: Three 2-D projections from 4-D, for the 'expbranches' data.

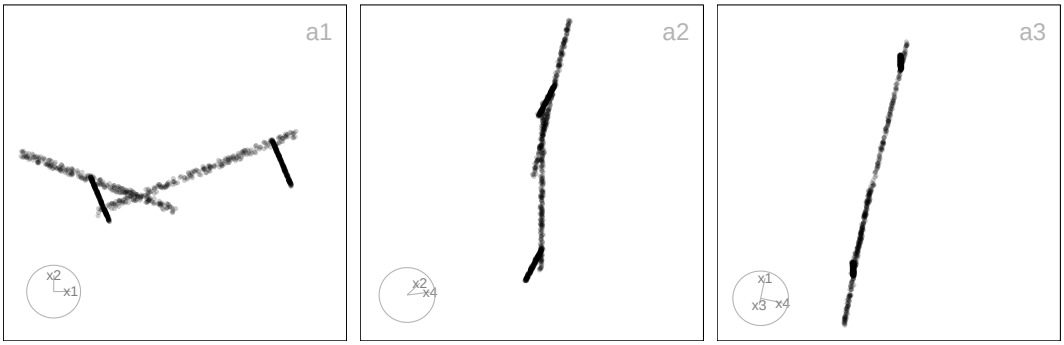

Figure 2: Three 2-D projections from 4-D, for the 'linearbranches' data.

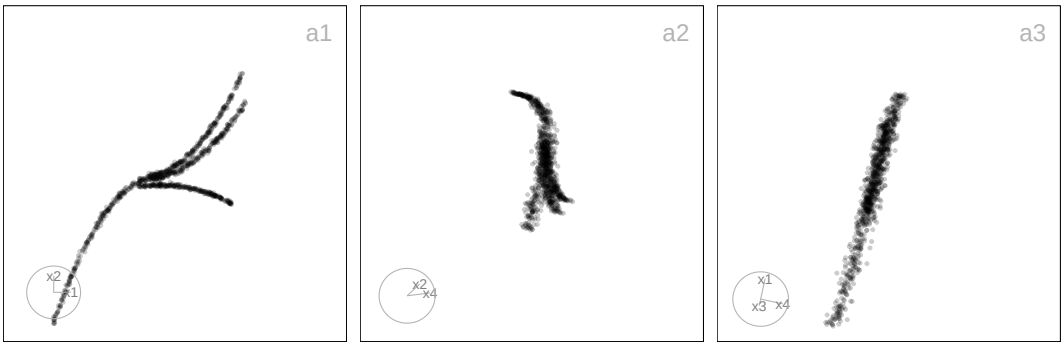

Figure 3: Three 2-D projections from 4-D, for the 'curvybranches' data.

randomly chosen point within a restricted horizontal band: $X_1 \in [-0.15, 0.15]$ (to ensure connectivity with earlier branches), spans a unit-length interval on X_1:$X_1 \in [x_{\text{start}}, x_{\text{start}} + 1]$, and has the structure:

$$X_2 = 0.1 \cdot X_1 - s_i \cdot (X_1^2 - x_{\text{start}}) + y_{\text{start}}, \quad s_i \in \{-2, -1.5, -1, -0.5, 0, 0.5, 1.5\}.$$

Here, s_i is a sampled scale factor (slope-like term) controlling curvature, and $(x_{\text{start}}, y_{\text{start}})$ is the starting point 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 $j = 3, \ldots, p$.

```
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 a p-dimensional space. Each branch lies primarily 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 2-D combinations. 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 (x_{i1}, x_{i2}) 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:

$$x_{i1} \sim U(0, 2), \quad x_{i2} = -s_i \cdot x_{i1} + \epsilon, \quad \epsilon \sim U(0, 0.5),$$

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 $1, 1.5, 2, \ldots, 8$ in increments of 0.5.

For $p > 2$, the remaining dimensions contain independent Gaussian noise:

$$x_j \sim N(0, 0.1^2), \quad \text{for } j \notin \{i1, i2\}.$$

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

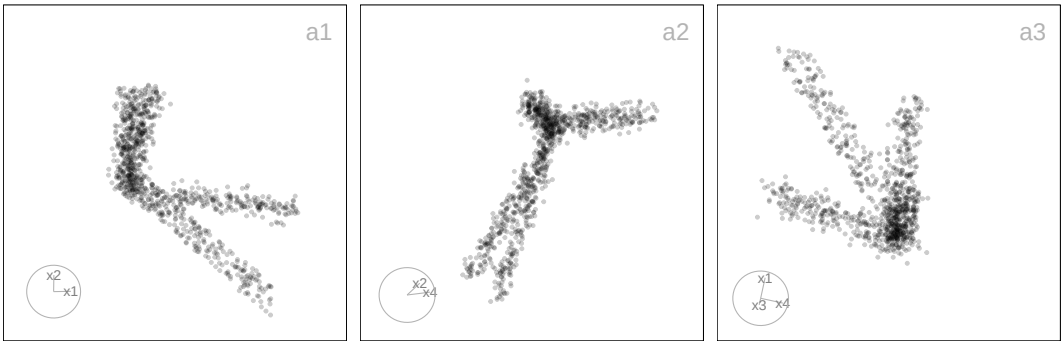

Figure 4: Three 2-D projections from 4-D, for the 'orglinearbranches' data.

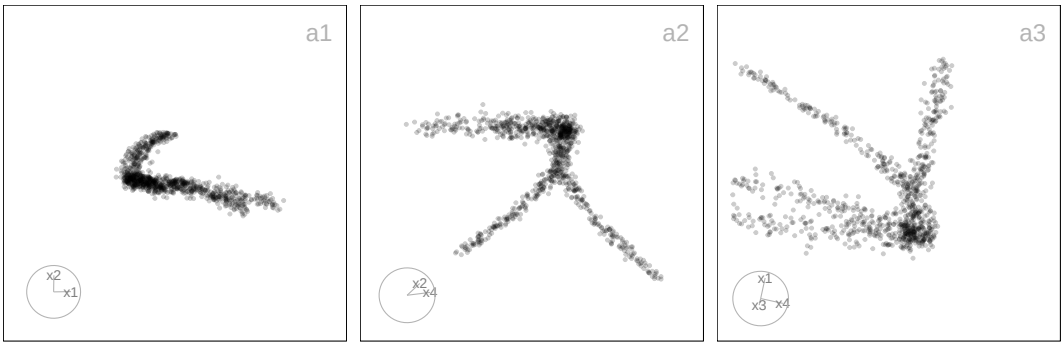

Figure 5: Three 2-D projections from 4-D, for the 'orgcurvybranches' data.

`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. 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 (x_{i1}, x_{i2}) are sampled without replacement. Otherwise, combinations are selected with replacement to reach k total branches. For each branch $i = 1, \ldots, k$, a random scale factor s_i is used to vary the curvature:

- If $k \leq \binom{p}{2}$: $s_i = 1$
- If $k > \binom{p}{2}$: $s_i \sim$ Uniform sample from $1, 1.5, \ldots, 8$

Each branch contains n_i points such that $\sum_{i=1}^{k} n_i = n$, where the vector $(n_1, \ldots, n_k)$ is randomly drawn using the helper function `gen_nsum()` to partition n.

Within its assigned subspace (x_{i1}, x_{i2}), branch i is defined by:

$$x_{i1} \sim U(0, 2), \quad x_{i2} = -s_i \cdot x_{i1}^2 + \epsilon, \quad \epsilon \sim U(0, 0.5)$$

This forms a smooth downward-opening parabola in the plane defined by (x_{i1}, x_{i2}), with the degree of curvature controlled by s_i.

For $p > 2$, all remaining dimensions are independent Gaussian noise:

$$x_j \sim N(0, 0.1^2), \quad \text{for } j \notin \{i1, i2\}.$$

```
orgcurvybranches <- gen_orgcurvybranches(n = 1000, p = 4, k = 4)
```

Cone

To simulate a cone-shaped structure in arbitrary dimensions, 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.

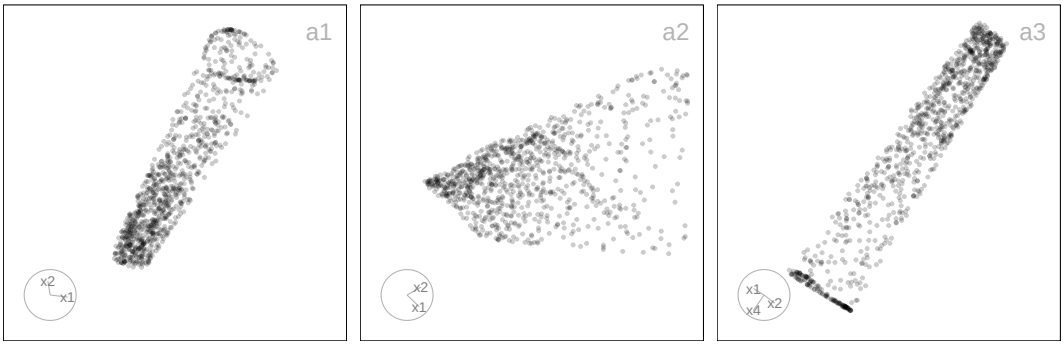

Figure 6: Three 2-D projections from 4-D, for the 'cone' data.

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.

This function generates `n` points in a `p`-dimensional space, where the last dimension (`X_p`) represents height along the cone's axis, and the remaining dimensions define a shrinking hyperspherical cross-section as one moves toward the tip.

Points along the height axis are drawn from an exponential distribution to increase density near the tip:

$$X_p = h_i \sim \text{Exp}(\lambda = 2/h), \quad \text{truncated at } h$$

The effective radius at height h_i decreases linearly from base to tip, controlled by a shape ratio parameter:

$$r_i = r_{\min} + (r_{\max} - r_{\min}) \cdot \frac{h_i}{h}, \quad \text{where } r_{\min} = \text{ratio},\ r_{\max} = 1$$

For each point, a direction is sampled uniformly from a $(p-1)$-dimensional hypersphere using generalized spherical coordinates (angles). The radial coordinates are scaled by r_i, which ensures a conical taper. Specifically, when $p = 3$, this scaling results in a classic 3-D cone. However, for $p > 3$, the introduction of additional angular components allows for a smooth extension into higher dimensions, preserving the conical shape while accommodating the complexities of multi-dimensional geometry.

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

Cube

`gen_gridcube()` The `gen_gridcube(n, p)` constructs a regular grid using approximately n points by evenly partitioning each of the p dimensions. Specifically, the number of grid points along each axis is determined by `gen_nproduct(n, p)`, which calculates the nearest balanced integer factors whose product is close to n. The full grid is generated via `expand_grid()`, giving Cartesian combinations of these evenly spaced positions.

Each variable X_j represents integer grid coordinates (e.g., 1, 2, 3, ...) in dimension j:

$$X_j \in 1, \ldots, k_j, \text{ where } \prod_{j=1}^{p} k_j \approx n.$$

This results in a hypercube-shaped lattice in p-dimensional space.

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

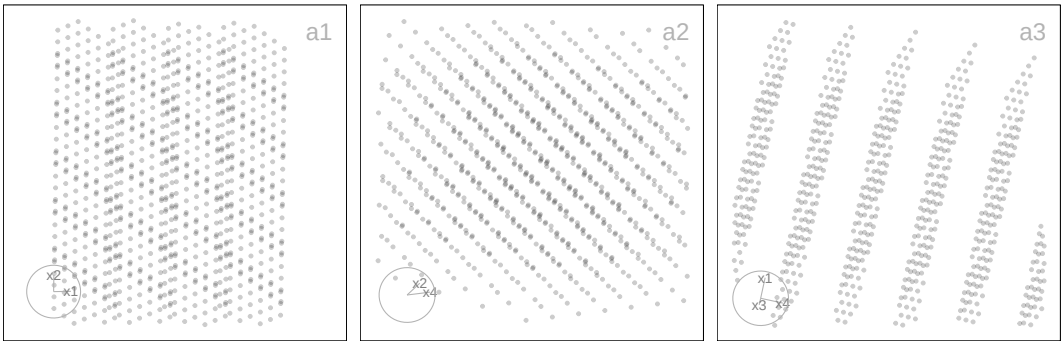

Figure 7: Three 2-*D* projections from 4-*D*, for the 'gridcube' data.

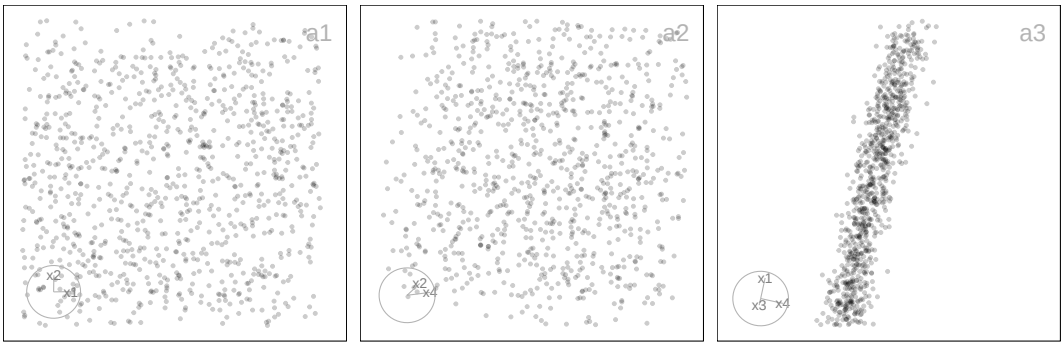

Figure 8: Three 2-*D* projections from 4-*D*, for the 'unifcube' data.

`gen_unifcube()` The `gen_unifcube(n, p)` function generates n points uniformly distributed within a 3-*D* cube centered at the origin, optionally embedded in a higher-dimensional space with additional noise dimensions.

Points are sampled independently from $X_1, X_2, X_3 \sim U(-0.5, 0.5)$ of a cube of side length 1. This results in a uniform cloud of points filling the unit cube in 3-*D* space.

For $p > 3$, the remaining dimensions are filled with Gaussian noise:

$$X_j \sim N(0, 0.05^2), \quad j = 4, \ldots, p.$$

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

`gen_cubehole()` The `gen_cubehole(n, p)` function generates a dataset of n points sampled uniformly inside a cube, then removes points that fall within a central spherical hole, resulting in a hollow cube structure in p-dimensional space.

The initial points are drawn from a uniform cube centered at the origin: $X_1, X_2, X_3 \sim U(-0.5, 0.5)$.

The center of the cube is $(0, 0, 0)$. A spherical hole of radius 0.5 is carved out by filtering out points whose Euclidean distance from the origin is less than or equal to 0.5:

Remove if $\sqrt{X_1^2 + X_2^2 + X_3^2} \leq 0.5$.

This results in a shell-like point cloud that fills the cube but avoids the spherical center.

For $p > 3$, Gaussian noise is added to the remaining dimensions (via `gen_unifcube()`):

$$X_j \sim N(0, 0.05^2), \quad j = 4, \ldots, p.$$

.

```
cubehole <- gen_cubehole(n = 1000, p = 4)
```

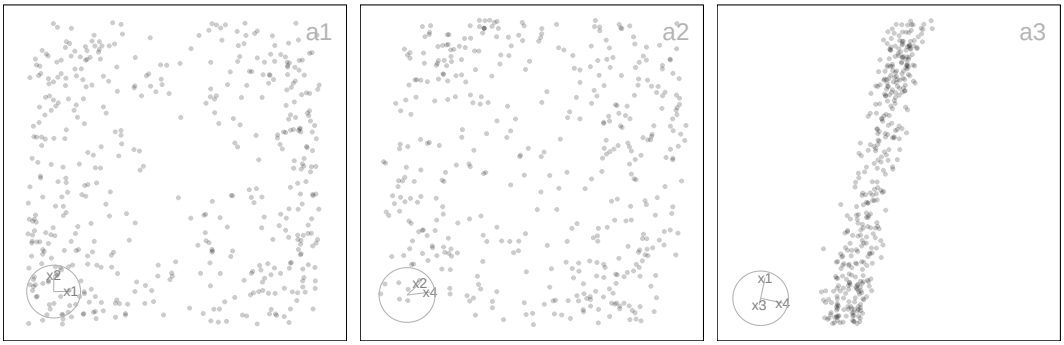

Figure 9: Three 2-*D* projections from 4-*D*, for the 'cubehole' data.

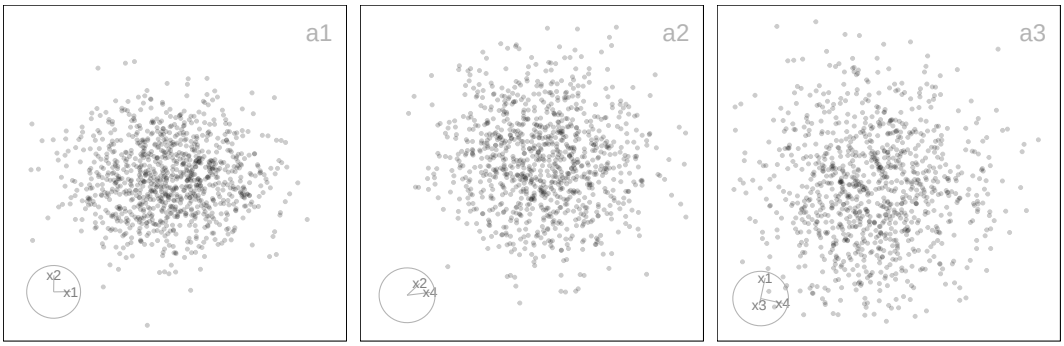

Figure 10: Three 2-*D* projections from 4-*D*, for the 'gau' data.

Gaussian

The `gen_gaussian(n, p, s)` function generates a multivariate Gaussian cloud in *p-D*, centered at the origin with user-defined covariance structure. Each point is independently drawn using the multivariate normal distribution with $X_i \sim N_p(\mathbf{0}, \Sigma)$, where Σ 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 simulates a high-dimensional linear structure with a dominant linear trend and small additive noise. Each variable X_i is created as $X_i = \text{scale}_i \cdot (0, 1, \ldots, n-1+\epsilon) + \text{shift}_i$, where $\text{scale}_i \sim U(-10, 10)$ is randomly chosen direction and stretch, $\text{shift}_i \sim U(-300, 300)$ is large offset to spread features apart, and $\epsilon \sim N(0, 0.03 \cdot n)$ is Gaussian noise.

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

Mobius

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

The Mobius strip is generated using `geozoo::mobius(n, p = 3)$points`, which samples n points from a parametrization of the surface defined by angle θ and strip width w. The cartesian coordinates are;

$$X_1 = \left(1 + \frac{w}{2}\cos(\theta/2)\right)\cos(\theta),$$
$$X_2 = \left(1 + \frac{w}{2}\cos(\theta/2)\right)\sin(\theta),$$

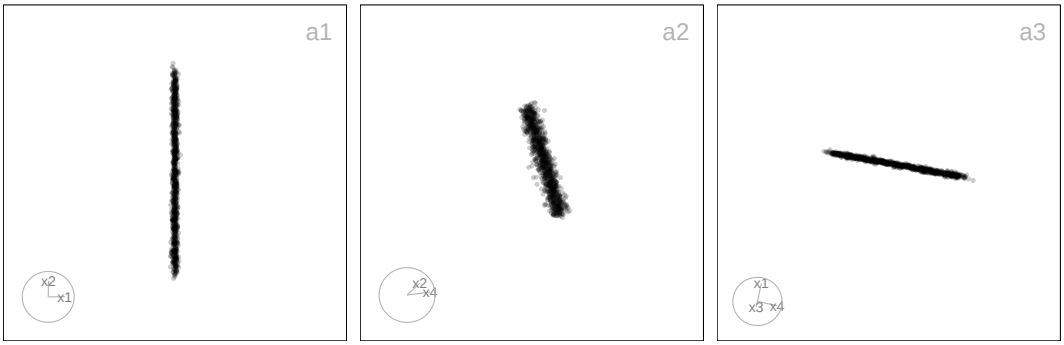

Figure 11: Three 2-D projections from 4-D, for the 'linear' data.

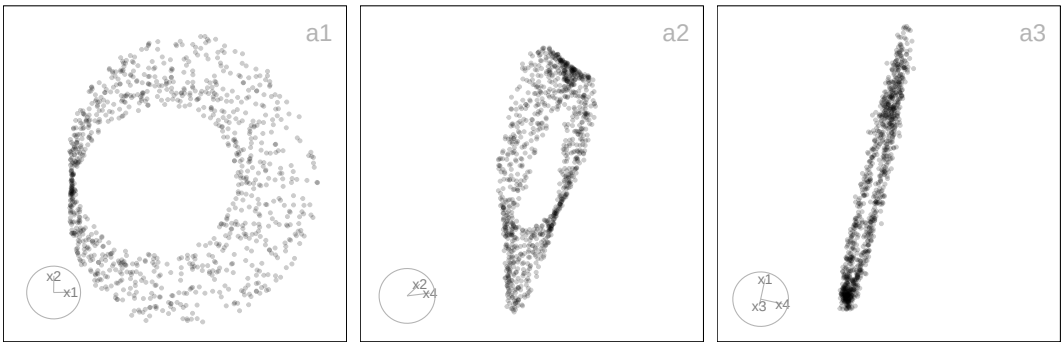

Figure 12: Three 2-D projections from 4-D, for the 'mobius' data.

$$X_3 = \frac{w}{2}\sin(\theta/2).$$

This maps a 2-D band with a half-twist into 3-D space, forming a non-orientable 1-sided surface.

For $p > 3$, the function appends $p - 3$ dimensions with small Gaussian noise:

$$x_j \sim N(0, 0.05^2), \quad j = 4, \ldots, p.$$

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

Polynomial

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

The curve is constructed by drawing uniformly spaced 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 measurement noise or intrinsic variability.

For $p > 2$, the dataset includes Gaussian noise in the remaining dimensions via `gen_noisedims()`:

$$X_j \sim N(0, 0.1^2), \quad j = 3, \ldots, p.$$

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

Table 5: cardinalR polynomial data generation functions

Function	Explanation
gen_quadratic	Generate a quadratic pattern.
gen_cubic	Generate a cubic pattern.

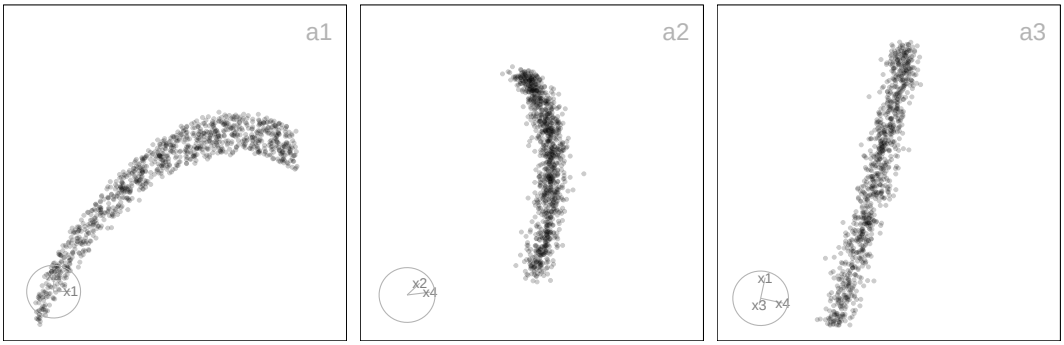

Figure 13: Three 2-*D* projections from 4-*D*, for the 'quadratic' data.

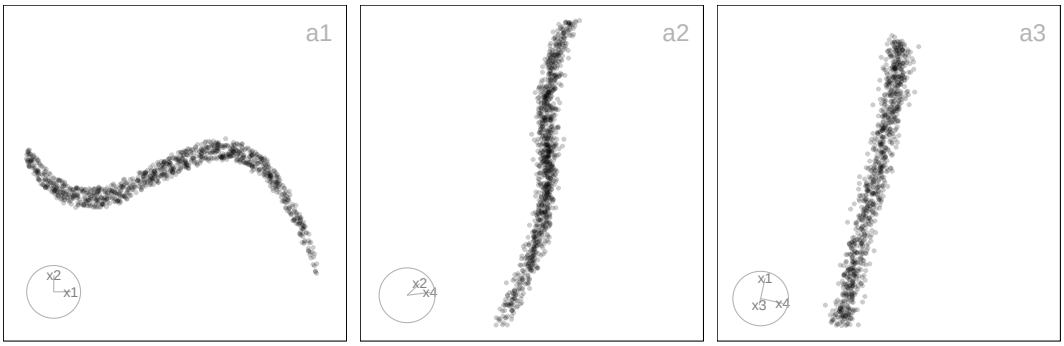

Figure 14: Three 2-*D* projections from 4-*D*, for the 'cubic' data.

`gen_cubic()` The `gen_cubic(n, p, range)` function generates a dataset of n points forming a cubic curve in the first two dimensions of a p-dimensional space. 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 $\varepsilon_2 \sim U(0, 0.5)$.

When $p > 2$, the remaining dimensions consist of independent Gaussian noise features generated via `gen_noisedims()`:

$$X_j \sim N(0, 0.1^2)\$for\$j = 3, \ldots, p.$$

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

Pyramid

`gen_pyrrect()` The `gen_pyrrect(n, p, h, l_vec, rt)` function generates a dataset of n points forming a high-dimensional pyramid-like structure with a rectangular base. The pyramid is embedded

Table 6: cardinalR pyramid data generation functions

Function	Explanation
gen_pyrrect	Generate a pyramid with a rectangular base, with the option of a sharp or blunted apex.
gen_pyrtri	Generate a pyramid with a triangular base, with the option of a sharp or blunted apex.
gen_pyrstar	Generate a pyramid with a star-shape base, with the option of a sharp or blunted apex.
gen_pyrholes	Generate a pyramid with triangular pyramid shaped holes.

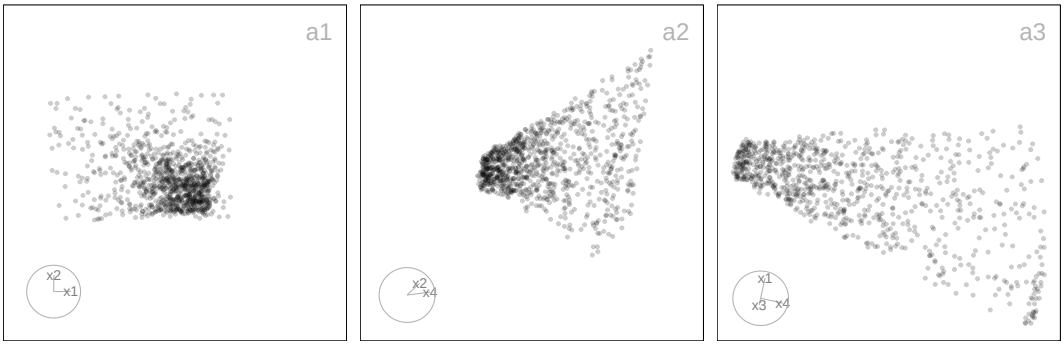

Figure 15: Three 2-D projections from 4-D, for the 'pyrrect' data.

in a p-dimensional space, with a tip at height zero and base at height h. The shape tapers linearly from the base dimensions (l_x, l_y) to a smaller rectangular section with side lengths $(2r_t, 2r_t)$ at the tip. Points are distributed more densely near the tip, simulating a natural skew toward smaller height values.

Let $x_1, x_2, \ldots, x_p$ denote the coordinates of the generated points. The final dimension x_p encodes the height of each point and is drawn from an exponential distribution capped at h:

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

.

Let $r_x(z)$ and $r_y(z)$ denote the half-widths of the rectangular cross-section at height z:

$$r_x(z) = r_t + (l_x - r_t) \cdot \frac{z}{h}, \quad r_y(z) = r_t + (l_y - r_t) \cdot \frac{z}{h}$$

.

The first three coordinates are then defined as:

$$x_1 \sim U(-r_x(z),\ r_x(z)), \quad x_2 \sim U(-r_y(z),\ r_y(z)), \quad x_3 \sim U(-r_x(z),\ r_x(z))$$

.

For $p > 3$, the remaining $p - 4$ dimensions (i.e., x_4 to x_{p-1}) are tapered toward zero with decreasing height and generated as:

$$x_j \sim U(-0.1, 0.1) \cdot \left(1 - \frac{z}{h}\right), \quad \text{for } j = 4, \ldots, p-1.$$

This tapering ensures that the structure narrows in all directions as it approaches the tip.

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

`gen_pyrtri()` The `gen_pyrtri(n, p, h, l, rt)` function generates a dataset of n points forming a high-dimensional pyramid-like structure with a triangular cross-section. The structure is embedded in a p-dimensional space, with the tip located at height 0 and the base at height h. The triangle expands linearly in size from tip to base, with more points concentrated near the tip.

Let $x_1, x_2, \ldots, x_p$ denote the coordinates of the generated points. The final coordinate \$x_p\$ encodes the height z of each point and is drawn from an exponential distribution capped at h:

$$x_p = z \sim \min\left(\text{Exp}(\lambda = 2/h),\ h\right).$$

Let $r(z)$ denote the scaling factor (distance from the origin to triangle vertices) at height z:

$$r(z) = r_t + (l - r_t) \cdot \frac{z}{h}.$$

A point in the triangle at height z is generated using barycentric coordinates (u, v) to ensure uniform sampling within the triangular cross-section:

$$u, v \sim U(0,1), \quad \text{if } u + v > 1 : u \leftarrow 1 - u,\ v \leftarrow 1 - v.$$

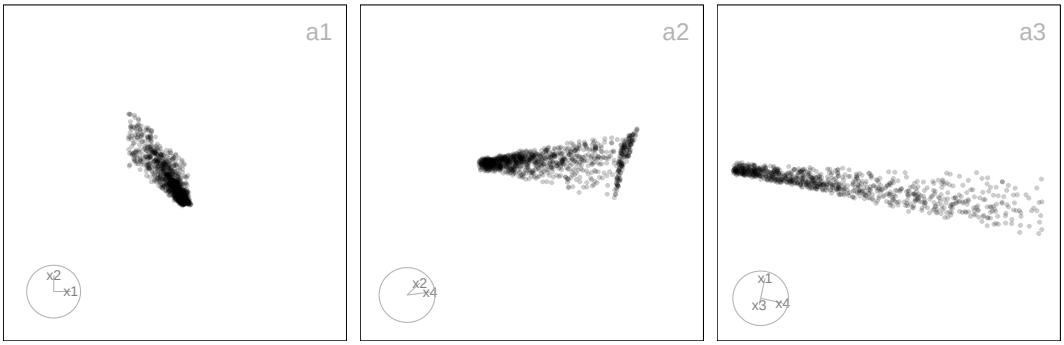

Figure 16: Three 2-D projections from 4-D, for the 'pyrtri' data.

The first three coordinates (triangle plane) are then:

$$x_1 = r(z)(1-u-v), \quad x_2 = r(z)\cdot u, \quad x_3 = r(z)\cdot v$$

For dimensions $j = 4, \ldots, p-1$, values are tapered linearly toward zero near the tip:

$$x_j \sim U(-0.1, 0.1)\cdot\left(1-\frac{z}{h}\right).$$

The `gen_pyrtri(n, p, h, l, rt)` function generates a dataset of n points forming a high-dimensional pyramid-like structure with a triangular cross-section. The structure is embedded in a p-dimensional space, with the tip located at height 0 and the base at height h. The triangle expands linearly in size from tip to base, with more points concentrated near the tip.

Let $x_1, x_2, \ldots, x_p$ denote the coordinates of the generated points. The final coordinate x_p encodes the height z of each point and is drawn from an exponential distribution capped at h:

$$x_p = z \sim \min\left(\text{Exp}(\lambda = 2/h),\ h\right).$$

Let $r(z)$ denote the scaling factor (distance from the origin to triangle vertices) at height z:

$$r(z) = r_t + (l - r_t)\cdot\frac{z}{h}.$$

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:

$$x_1 = r(z)(1-u-v), \quad x_2 = r(z)\cdot u, \quad x_3 = r(z)\cdot v$$

For dimensions $j = 4, \ldots, p-1$, values are tapered linearly toward zero near the tip:

$$x_j \sim U(-0.1, 0.1)\cdot\left(1-\frac{z}{h}\right).$$

The result is a tapered, high-dimensional triangle-based pyramid with more points near the apex and greater width at the base.

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

`gen_pyrstar()` The `gen_pyrstar(n, p, h, rb)` function generates a dataset of n points forming a high-dimensional pyramid-shaped structure with a hexagonal base. The pyramid extends vertically from height $z = 0$ (tip) to height $z = h$ (base), embedded in a p-dimensional space. The distribution concentrates more points near the base, and the hexagonal spread tapers toward the tip.

Let $x_1, x_2, \ldots, x_p$ denote the coordinates of the generated points. The final coordinate x_p encodes the height z:

$$x_p = z \sim U(0, h).$$

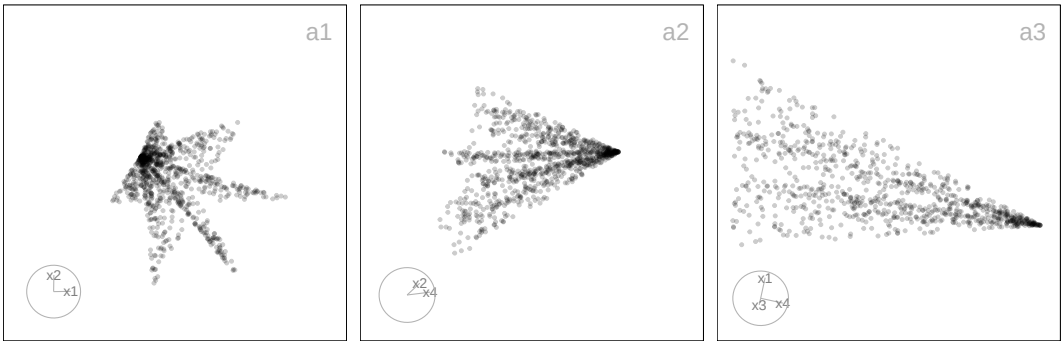

Figure 17: Three 2-D projections from 4-D, for the 'pyrstar' data.

The radius at height z scales linearly from zero (tip) to the base radius r_b:

$$r(z) = r_b \cdot \left(1 - \frac{z}{h}\right).$$

Each point is placed within a regular hexagon in the plane (x_1, x_2), using a randomly chosen hexagon sector angle $\theta \in \{0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \frac{4\pi}{3}, \frac{5\pi}{3}\}$ and a uniformly random radial scaling factor:

$$\theta \sim \text{Uniform sample from 6 hexagon angles}, \quad r_{\text{point}} \sim \sqrt{U(0,1)}.$$

Then, the first two coordinates are:

$$x_1 = r(z) \cdot r_{\text{point}} \cdot \cos(\theta), \quad x_2 = r(z) \cdot r_{\text{point}} \cdot \sin(\theta)$$

For dimensions $j = 3, \ldots, p-1$, values are tapered toward zero near the tip:

$$x_j \sim U(-0.1, 0.1) \cdot \left(1 - \frac{z}{h}\right).$$

This results in a smooth, star-like pyramid in p-dimensional space, with a broad hexagonal base and narrow tip.

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

`gen_pyrholes()` The `gen_pyrholes(n, p)` function generates n points embedded in a p-dimensional simplex using a chaotic attractor-like midpoint algorithm. 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 $x_1, x_2, \ldots, x_p$ 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-dimensional 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:

$$T_i = \frac{1}{2}(T_{i-1} + C_k), \quad C_k \in \{C_1, \ldots, C_{p+1}\}.$$

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:

$$X = \{T_1, T_2, \ldots, T_n\}, \quad X \in \mathbb{R}^{n \times p}.$$

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

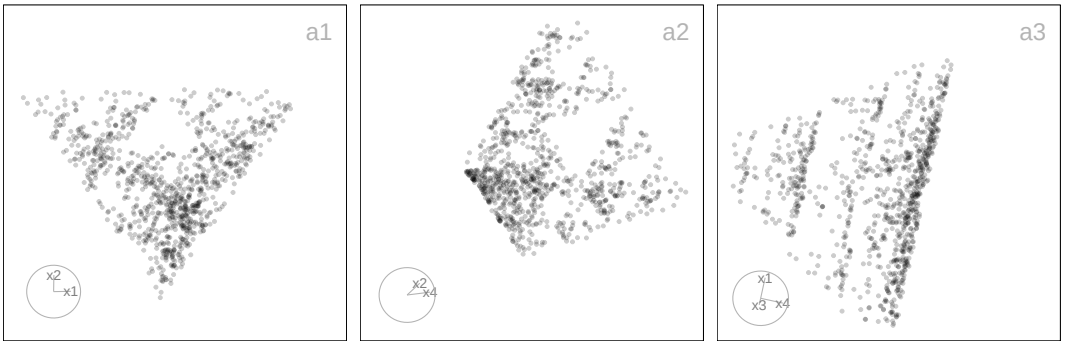

Figure 18: Three 2-D projections from 4-D, for the 'pyrholes' data.

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.

S-curve

`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` dimensions.

The 3-D geometry is constructed by introducing a latent parameter, $\theta \sim U\left(-\frac{3\pi}{2}, \frac{3\pi}{2}\right)$. This parameter controls the curvature of the manifold. The first three dimensions form the S-curve structure:

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

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 perturbations.

```
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)`. This function produces `n` observations in a p-dimensional space 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 $(0, 1, 0, \ldots)$ is defined in p-dimensional space. All observations within a Euclidean distance of $\sqrt{0.3} \approx 0.5477$ from the anchor are removed:

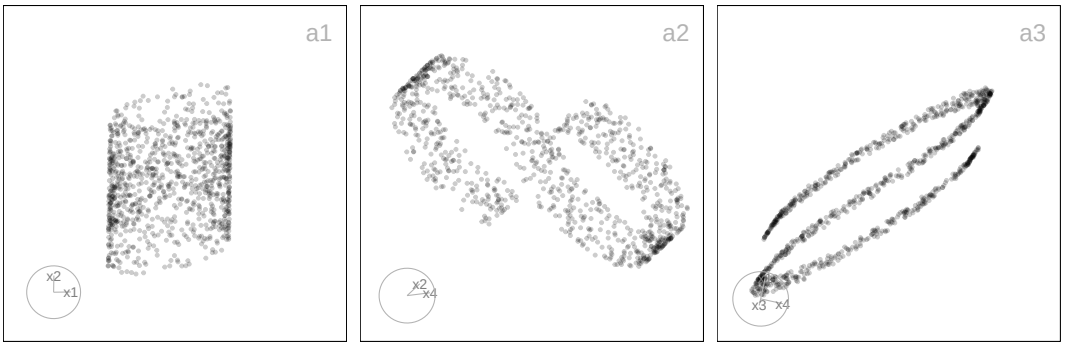

Figure 19: Three 2-D projections from 4-D, for the 'scurve' data.

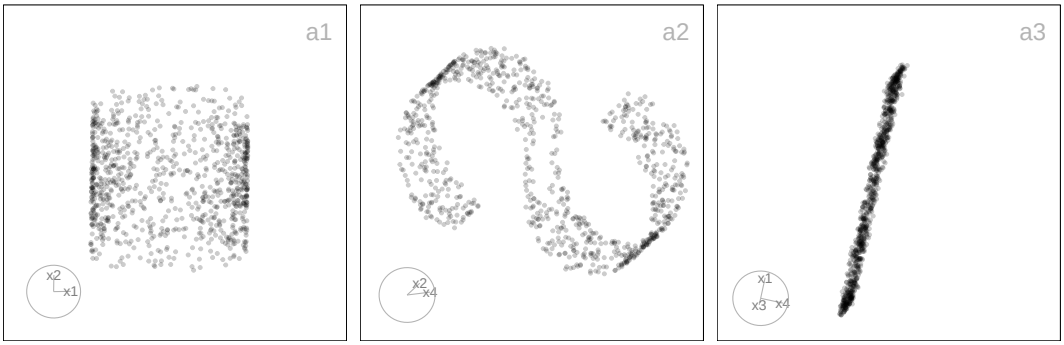

Figure 20: Three 2-D projections from 4-D, for the 'scurvehole' data.

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_gridedsphere	Generate a grided sphere.
gen_clusteredspheres	Generate multiple small spheres within a big sphere.
gen_hemisphere	Generate a hemisphere.

$$\text{remove if} \quad \sum_{j=1}^{p} (X_j - a_j)^2 \leq 0.3$$

.

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

`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.

A latent angle variable θ is uniformly sampled from the interval $[0, 2\pi]$. Coordinates in the first two dimensions represent a perfect circle on the plane:

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

For dimensions X_3 through X_p, sinusoidal transformations of the angle θ are introduced. The first component is a scaling factor that decreases with the dimension index, defined as $\text{scale}_j = \sqrt{(0.5)^{j-2}}$ for $j = 3, \ldots, p$. The second component is a phase shift that is proportional to the dimension index, specifically designed to decorrelate the curves, given by the formula $\phi_j = (j-2) \cdot \frac{\pi}{2p}$.

Each additional dimension is computed as:

$$X_j = \text{scale}_j \cdot \sin(\theta + \phi_j), \quad j = 3, \ldots, p$$

This structure retains a one-dimensional manifold (the circle) but allows it to be embedded non-linearly and decorrelatively in higher dimensions. The decreasing scale ensures that higher dimensions add less variation, preserving the dominance of the circular shape while adding complexity.

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

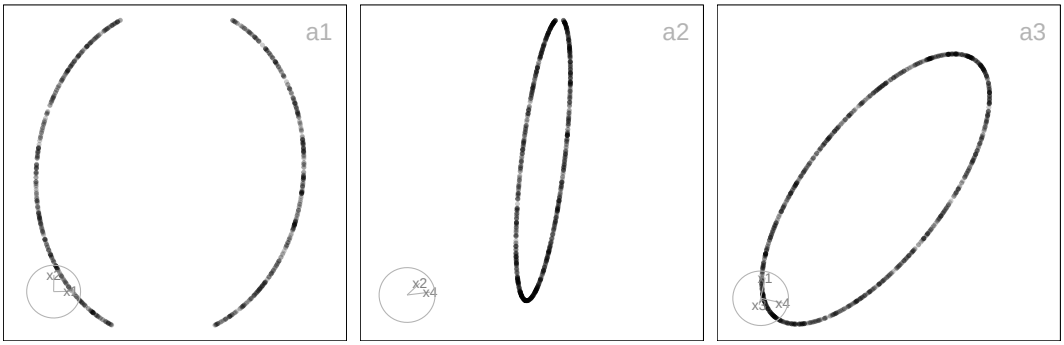

Figure 21: Three 2-D projections from 4-D, for the 'circle' data.

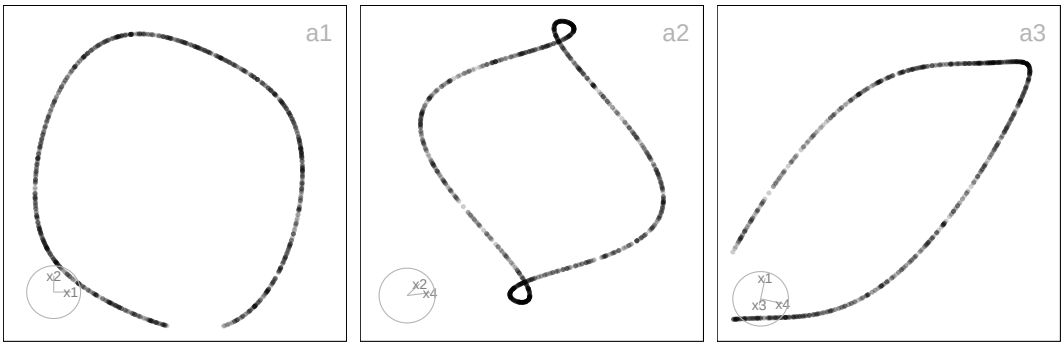

Figure 22: Three 2-D projections from 4-D, for the 'curvycycle' data.

`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. 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, $X_3 = \frac{1}{3} \cdot \cos(3\theta)$ 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 $X_j = \text{scale}_j \cdot \sin(\theta + \phi_j)$, where the phase shift ϕ_j is given by $\phi_j = (j-2) \cdot \frac{\pi}{2p}$.

This introduces decorrelated, low-variance sinusoidal variations across dimensions, retaining the cyclical nature while embedding the data in higher-dimensional space with a coherent but subtle structure.

```
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 $p > 3$.

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;

$$X_1 = r \cdot \sqrt{1 - u^2} \cdot \cos(\theta),$$

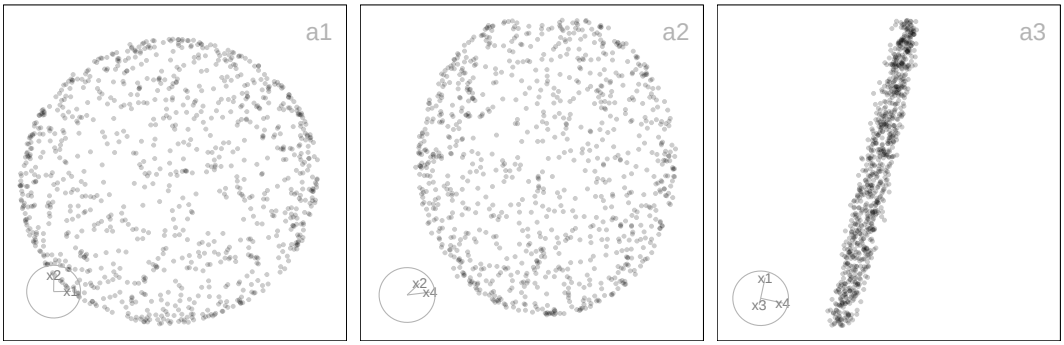

Figure 23: Three 2-D projections from 4-D, for the 'unifsphere' data.

$$X_2 = r \cdot \sqrt{1-u^2} \cdot \sin(\theta),$$
$$X_3 = r \cdot u,$$

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:

$$X_j \sim N(0, 0.05^2), \text{ for } j = 4, \ldots, p.$$

This extends the 3-D spherical manifold into p-dimensional space with orthogonal Gaussian noise, preserving the spherical structure while allowing for testing in higher-dimensional settings.

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

`gen_gridedsphere()` The `gen_gridedsphere(n, p)` function generates a p-dimensional dataset of approximately n observations evenly distributed on the surface of a 3-D unit sphere, with optional Gaussian noise dimensions when $p > 3$.

The base structure consists of a 3-D spherical surface created using a regular grid in spherical coordinates, where $\theta \in [0, 2\pi]$ represents the azimuthal angle (longitude) and $\phi \in [0, \pi]$ denotes the polar angle (co-latitude).

The number of grid steps along each dimension is determined by factoring n into two approximately equal integers via `gen_nproduct(n, p = 2)`.

Each point on the sphere is computed using the spherical-to-Cartesian transformation:

$$X_1 = \sin(\phi) \cdot \cos(\theta),$$
$$X_2 = \sin(\phi) \cdot \sin(\theta),$$
$$X_3 = \cos(\phi).$$

This forms a structured grid of points on the unit sphere, ideal for studying how regular geometric manifolds are transformed under nonlinear dimensionality reduction.

For $p > 3$, the manifold is embedded in a higher-dimensional space by adding Gaussian noise dimensions:

$$X_j \sim N(0, 0.05^2), \text{ for } j = 4, \ldots, p.$$

These dimensions represent low-variance orthogonal noise, preserving the spherical shape while testing robustness in high-dimensional settings.

```
gridedsphere <- gen_gridedsphere(n = 1000, p = 4)
```

`gen_clusteredspheres()` The `gen_clusteredspheres(n, k, p, r, loc)` function generates a synthetic dataset of $n_1 + k \cdot n_2$ 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.

A large uniform sphere centered at the origin is created by sampling n_1 points uniformly on the surface of a p-dimensional 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.

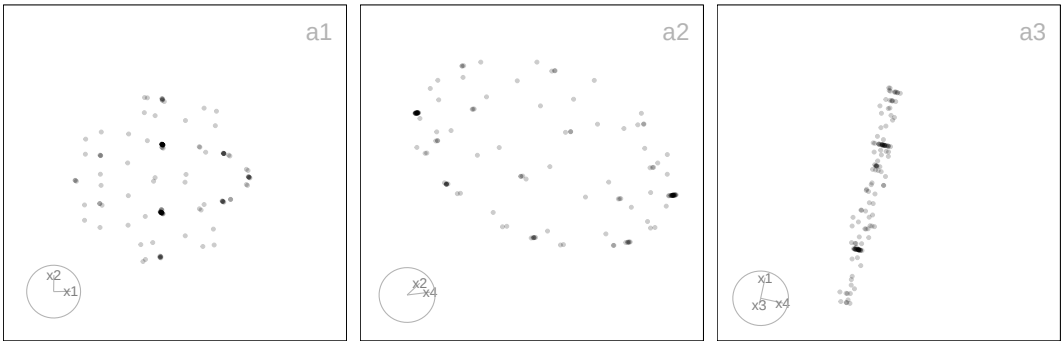

Figure 24: Three 2-D projections from 4-D, for the 'gridedsphere' data.

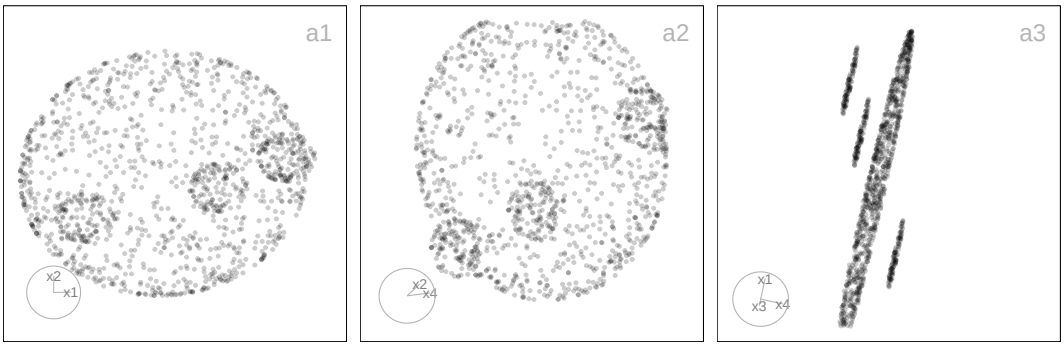

Figure 25: Three 2-D projections from 4-D, for the 'clusteredspheres' data.

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 $N(0, \texttt{loc}^2 I_p)$.

Points on spheres are generated using the standard hyperspherical method, which involves sampling $u \sim U(-1, 1)$ to determine the cosine of the polar angle, and sampling $\theta \sim U(0, 2\pi)$ to determine the azimuthal angle (for 3D).

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 $p > 4$.

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:

$$
\begin{aligned}
X_1 &= \sin(\theta_1) \cdot \cos(\theta_2), \\
X_2 &= \sin(\theta_1) \cdot \sin(\theta_2), \\
X_3 &= \cos(\theta_1) \cdot \cos(\theta_3), \\
X_4 &= \cos(\theta_1) \cdot \sin(\theta_3).
\end{aligned}
$$

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:

$$
X_j \sim N(0, 0.05^2), \text{ for } j = 5, \ldots, p.
$$

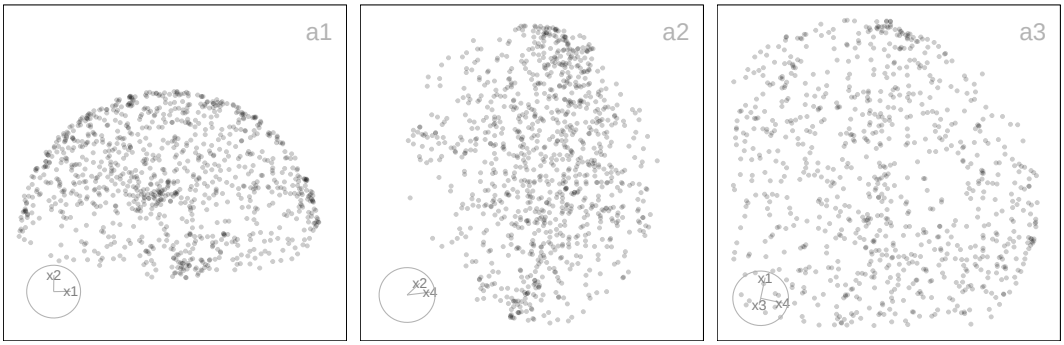

Figure 26: Three 2-*D* projections from 4-*D*, for the 'hemisphere' data.

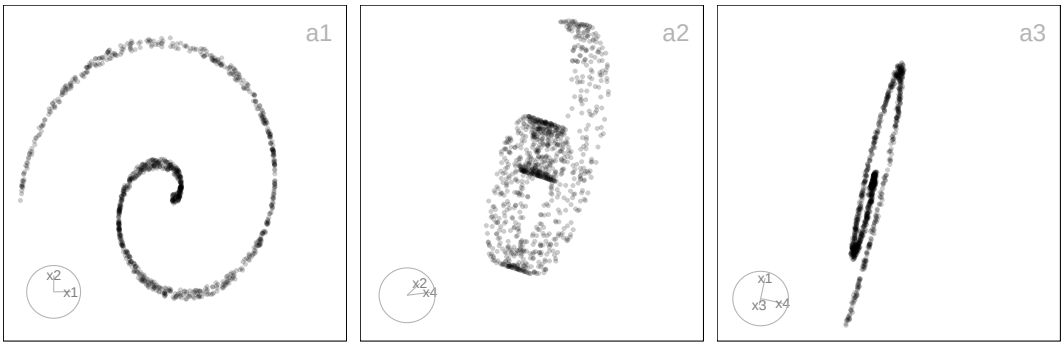

Figure 27: Three 2-*D* projections from 4-*D*, for the 'swissroll' data.

These dimensions maintain the spherical structure while simulating embedding into higher-dimensional space with small orthogonal perturbations.

```
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.

The first three dimensions form the classic 3-*D* Swiss roll shape:

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

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))
```

Trigonometric

`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.

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:

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

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.

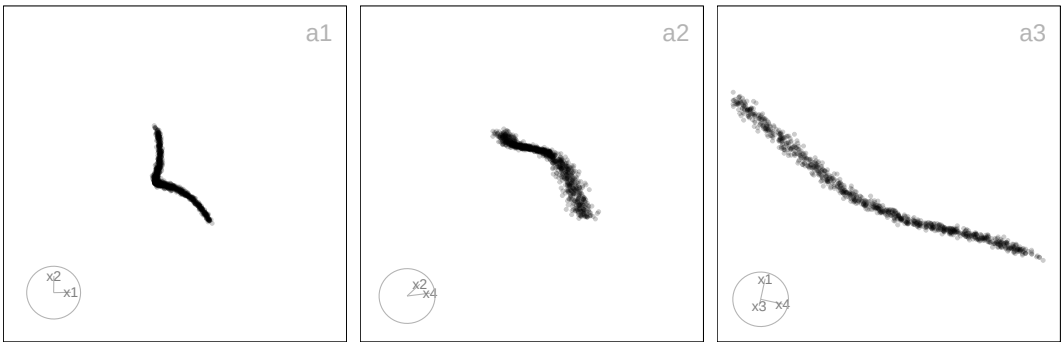

Figure 28: Three 2-D projections from 4-D, for the 'crescent' data.

This forms a nonlinear curvilinear structure in the shape of a tilted crescent in the X_1-X_2 plane.

For $p > 2$, additional dimensions X_3 to X_p are generated using `gen_wavydims1()`:

$$X_j = s_j \cdot \theta + \varepsilon_j, \text{ where } \varepsilon_j \sim N(0, 0.5^2), \text{ and } s_j \text{ is a random scale.}$$

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 $p > 4$.

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),
- $X_4 = \sin(z)$ (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:

$$X_j \sim N(0, 0.05^2), \text{ for } j = 5, \ldots, p.$$

These dimensions represent unstructured, low-variance noise.

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

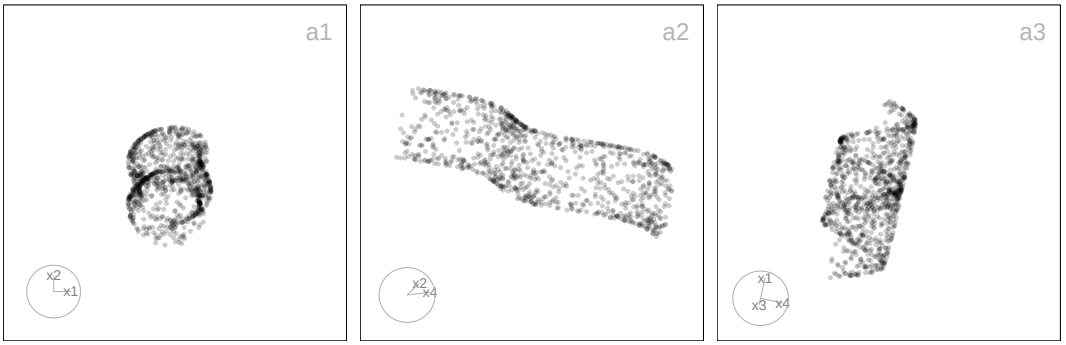

Figure 29: Three 2-D projections from 4-D, for the 'curvycylinder' data.

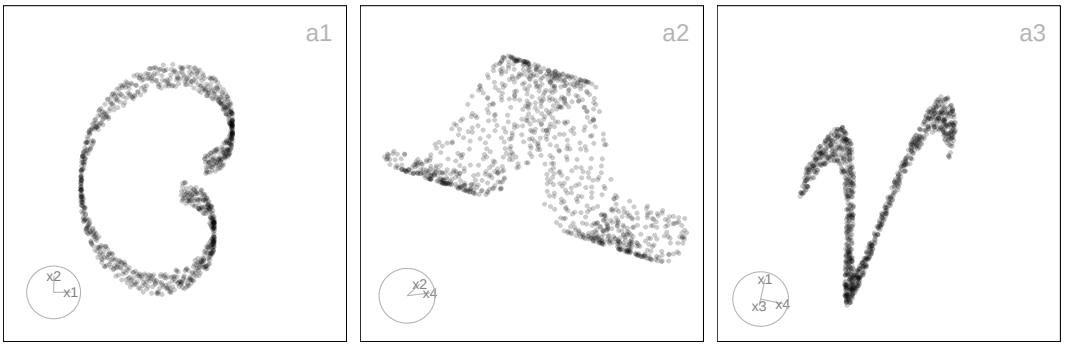

Figure 30: Three 2-D projections from 4-D, for the 'sphericalspiral' data.

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.

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:

- $X_1 = \sin(\phi) \cdot \cos(\theta)$
- $X_2 = \sin(\phi) \cdot \sin(\theta)$
- $X_3 = \cos(\phi) + \varepsilon$, where $\varepsilon \sim U(-0.5, 0.5)$ introduces vertical jitter.
- $X_4 = \theta / \max(\theta)$: 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:

$$X_j = s_j \cdot (-1)^{\lfloor j/2 \rfloor} \cdot X_1^{a_j} + \eta_j,$$

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 a p-dimensional space. Additional dimensions (if $p > 4$) are filled with structured noise to mimic high-dimensional complexity.

The first four coordinates follow a 3-D helix with an additional oscillating component. Let $\theta \in [0, \frac{5\pi}{4}]$ be a sequence of angles controlling rotation around a circle.

Cartesian coordinates: * $X_1 = \cos(\theta)$: circular trajectory along the x-axis.

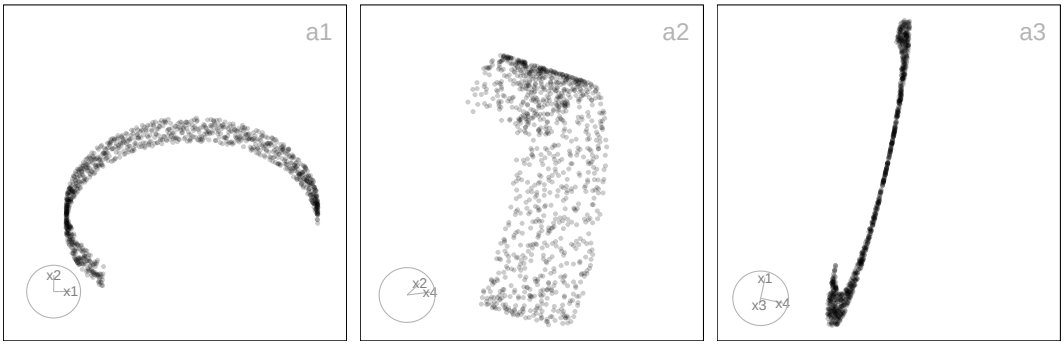

Figure 31: Three 2-D projections from 4-D, for the 'helicalspiral' data.

- $X_2 = \sin(\theta)$: circular trajectory along the y-axis.
- $X_3 = 0.05 \cdot \theta + \varepsilon_3$, with $\varepsilon_3 \sim U(-0.5, 0.5)$: linear progression (height) with vertical jitter, simulating a helix.
- $X_4 = 0.1 \cdot \sin(\theta)$: 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:

$$X_j \sim N(0, 0.05^2), \quad j = 5, \ldots, p.$$

```
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 a p-dimensional space. 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 $\theta \in [0, 2\pi \cdot \text{spins}]$, controlling the angular progression of the spiral. The mapping to Cartesian space is as follows:

The Archimedean spiral in the horizontal plane is represented by;

$X_1 = \theta \cdot \cos(\theta)$ for radial expansion in x. $X_2 = \theta \cdot \sin(\theta)$ for radial expansion in y.

The growth pattern resembles a cone, with the height increasing according to $X_3 = 2 \cdot \theta / \max(\theta) + \varepsilon_3$, with $\varepsilon_3 \sim U(-0.1, 0.6)$.

Spiral modulation in the fourth dimension is represented by $X_4 = \theta \cdot \sin(2\theta) + \varepsilon_4$, 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()`:

$$X_j \sim N(0, 0.05^2), \quad j = 5, \ldots, p.$$

```
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. The key geometry lies in dimensions 1–4, where:

$X_1 \sim U(0.1, 2)$: base variable (avoids zero to prevent division errors). $X_3 \sim U(0.1, 0.8)$: independent auxiliary variable. $X_2 = \frac{hc}{X_1} + \text{nonfac} \cdot \sin(X_1)$: non-linear combination of **hyperbolic and sinusoidal transformations**, creating sharp curvature and oscillation. $X_4 = \cos(\pi \cdot X_1) + \varepsilon$, with

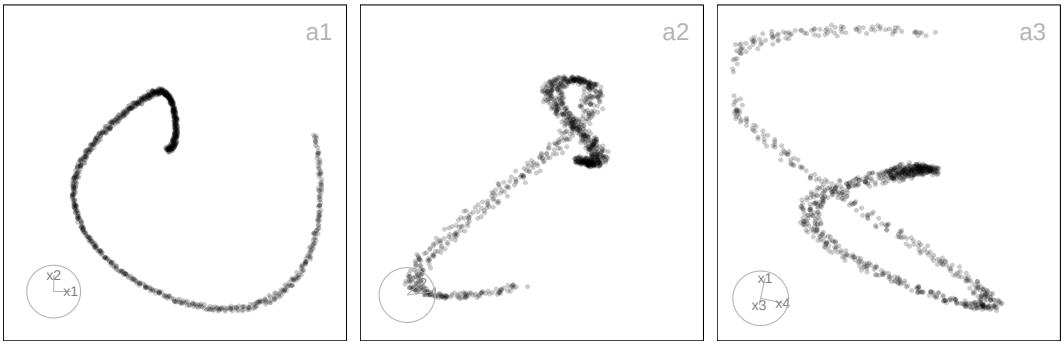

Figure 32: Three 2-D projections from 4-D, for the 'conicspiral' data.

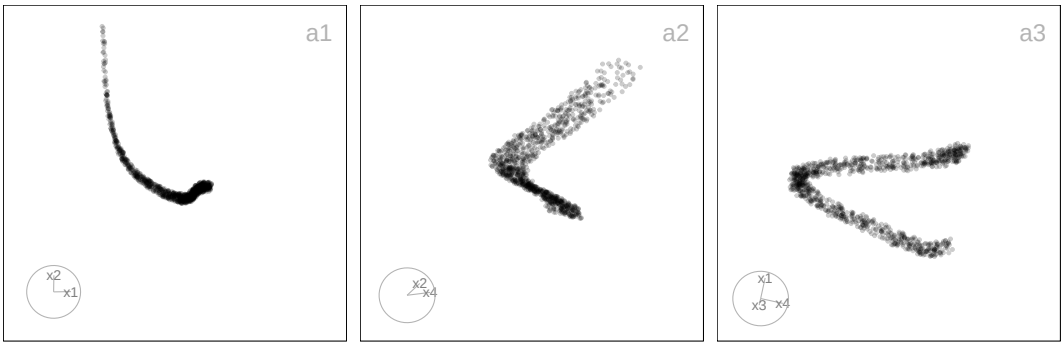

Figure 33: Three 2-D projections from 4-D, for the 'nonlinear' data.

$\varepsilon \sim U(-0.1, 0.1)$: 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()`:

$$X_j \sim N(0, 0.05^2), \quad j = 5, \ldots, p.$$

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

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 10.

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 11 details these functions.

Table 10: cardinalR multiple clusters generation functions

Function	Explanation
make_mobiusgau	
make_multigau	

Table 11: 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.

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 34).

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

## To generate data
five_clusts <- gen_multicluster(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"),
                      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.

UMAP, PHATE, TriMAP, and PaCMAP effectively separate the five clusters and show the preservation of the global structure (Figure 35). 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.

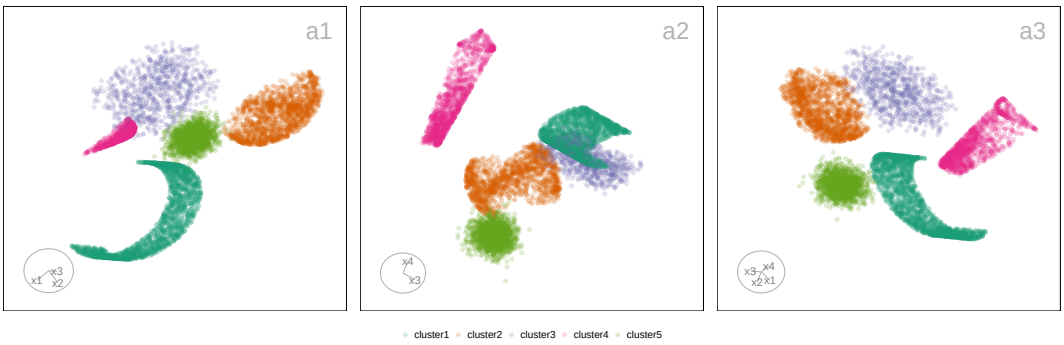

Figure 34: Three 2-*D* projections from 4-*D*, for the 'mobiusgau' data. The helical spiral cluster is represented in dark green, the hemisphere cluster in orange, the uniform cube-shaped cluster in purple, the blunted cone cluster in pink, and the Gaussian-shaped cluster in light green.

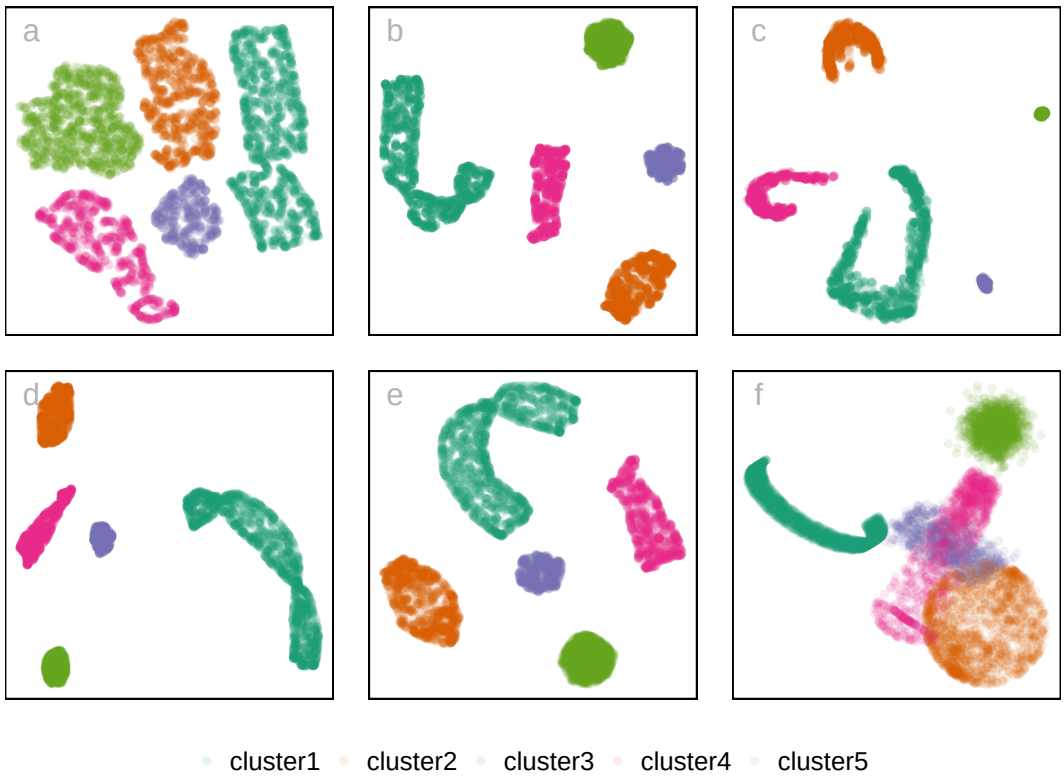

Figure 35: Six different dimension reduction representations of the 'mobiusgau' data using default hyperparameter settings: (a) tSNE, (b) UMAP, (c) PAHTE, (d) TriMAP, (e) PaCMAP, and (f) PCA.

4 Conclusion

The `cardinalR` package contributes a flexible and extensible framework for generating high-dimensional data structures with well-defined geometric properties. It addresses a growing need in the evaluation of machine learning, clustering, and nonlinear dimensionality reduction (NLDR) methods—offering researchers the ability to design customized datasets with interpretable structures, noise characteristics, and clustering arrangements. This makes `cardinalR` a useful complement to existing tools like `geozoo`, `snedata`, and `mlbench`, expanding the scope to higher dimensions and more complex shapes.

The suite of included structures is diverse and targeted. For example, **branching shapes** allow investigation of continuity and topological preservation in low-dimensional embeddings. The **S-curve with a hole** enables exploration of how methods handle incomplete manifolds. **Clustered spheres** assess cluster separation on curved surfaces, while the **Möbius strip** challenges NLDR methods with non-orientable geometry. **Gridded cubes** test spatial regularity, and **pyrholes** represent non-convex, sparse shapes useful for testing clustering performance in irregular high-dimensional regions.

These structures are not only valuable for algorithm diagnostics but may also be used for **teaching high-dimensional concepts**, **benchmarking reproducibility**, and **evaluating hyperparameter sensitivity**. The package's design encourages users to adjust dimensionality, noise, sample size, and clustering properties, promoting **transparent experimentation** and **comparative model evaluation**.

Looking ahead, future extensions of `cardinalR` could incorporate dynamic shape generation, integration with supervised learning simulations, or visual interfaces to assist users in interactively exploring generated structures. Expanding the library to include more biologically inspired or real-world data analogs may also broaden its utility across domains like bioinformatics, forensic science, and spatial data analysis.

5 Code

The code is available at `https://github.com/JayaniLakshika/cardinalR`, and source material for this paper is available at `https://github.com/JayaniLakshika/paper-cardinalR`.

6 Acknowledgements

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`. [p24]

G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2025. URL `https://CRAN.R-project.org/package=cli`. R package version 3.6.4. [p26]

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. [p26]

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`. [p24]

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. [p24]

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`. [p24]

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. [p24]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2023. URL https://CRAN.R-project.org/package=tibble. R package version 3.2.1. [p26]

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/. [p26]

B. Schloerke. *geozoo: Zoo of Geometric Objects*, 2016. URL https://CRAN.R-project.org/package=geozoo. R package version 0.5.1. [p1, 26]

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. [p26]

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. [p24]

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. [p26]

H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2025. URL https://CRAN.R-project.org/package=purrr. R package version 1.0.4. [p26]

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. [p26]

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. [p26]

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. [p26]

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. [p26]

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