

# Package: spinebil (via r-universe)

May 13, 2026

**Type** Package

**Title** Investigating New Projection Pursuit Index Functions

**Version** 1.0.5

**Description** Projection pursuit is used to find interesting low-dimensional projections of high-dimensional data by optimizing an index over all possible projections. The 'spinebil' package contains methods to evaluate the performance of projection pursuit index functions using four methods. A paper describing the methods can be found at [doi:10.1007/s00180-020-00954-8](https://doi.org/10.1007/s00180-020-00954-8).

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**URL** <https://uschilaa.github.io/spinebil/index.html>

**BugReports** <https://github.com/uschiLaa/spinebil/issues>

**Depends** R (>= 4.0.0)

**Imports** tourr, ggplot2, tibble, stats, dplyr, tidyr, tictoc, cassowaryr, rlang

**Suggests** minerva, testthat, purrr, furrr, future, quarto, knitr

**VignetteBuilder** quarto

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**Config/pak/sysreqs** libglpk-dev libgs10-dev libicu-dev libxml2-dev

**Repository** <https://uschilaa.r-universe.dev>

**Date/Publication** 2026-04-13 01:22:43 UTC

**RemoteUrl** <https://github.com/uschilaa/spinebil>

**RemoteRef** HEAD

**RemoteSha** 9ba67c3e484f24137321192e9b960667461f7a45

## Contents

basis_matrix . . . . .	2
basis_vector . . . . .	3
compare_smoothing . . . . .	3
data_gen . . . . .	4
distance_dist . . . . .	5
distance_to_sp . . . . .	5
get_index_mean . . . . .	6
get_trace . . . . .	7
jitter_angle . . . . .	7
jitter_points . . . . .	8
noise_gen . . . . .	8
pipe_data . . . . .	9
plot_rotation . . . . .	10
plot_smoothing_comparison . . . . .	10
plot_trace . . . . .	11
ppi_mean . . . . .	11
ppi_noise_threshold . . . . .	12
ppi_samplesize_effect . . . . .	13
ppi_scale . . . . .	14
profile_rotation . . . . .	16
scag_index . . . . .	16
sin_data . . . . .	17
spiral_data . . . . .	18
squint_angle_estimate . . . . .	19
time_sequence . . . . .	20
<b>Index</b>	<b>21</b>

---

basis_matrix	<i>Generate 2-d basis in directions i, j in n dimensions (i,j &lt;= n)</i>
--------------	--

---

### Description

Generate 2-d basis in directions i, j in n dimensions (i,j <= n)

### Usage

```
basis_matrix(i, j, n)
```

### Arguments

i	first basis direction
j	second basis direction
n	number of dimensions

**Value**

basis matrix

---

basis_vector	<i>Generate basis vector in direction i in n dimensions (i &lt;= n)</i>
--------------	---

---

**Description**

Generate basis vector in direction i in n dimensions (i &lt;= n)

**Usage**

basis\_vector(i, n)

**Arguments**

i	selected direction
n	number of dimensions

**Value**

basis vector

---

compare_smoothing	<i>Compare traces with different smoothing options.</i>
-------------------	---

---

**Description**

Compare traces with different smoothing options.

**Usage**

compare\_smoothing(d, tPath, idx, alphaV = c(0.01, 0.05, 0.1), n = 10)

**Arguments**

d	Data matrix
tPath	Interpolated tour path (as list of projections)
idx	Index function
alphaV	Jitter amounts to compare (for jittering angle or points)
n	Number of evaluations entering mean value calculation

**Value**

Table of mean index values

**Examples**

```
d <- as.matrix(spiral_data(30, 3))
tPath <- tourr::save_history(d, max_bases=2)
tPath <- as.list(tourr::interpolate(tPath, 0.3))
idx <- scag_index("stringy")
compS <- compare_smoothing(d, tPath, idx, alphaV = c(0.01, 0.05), n=2)
plot_smoothing_comparison(compS)
```

data\_gen

*Generate Synthetic Data with Various Structures***Description**

Generates either:

- Structured (x, y) scatter data (linear, sine, circle, etc.), or
- A matrix of scaled orthogonal polynomial features.

**Usage**

```
data_gen(type = "all", n = 500, degree = NULL, seed = NULL)
```

**Arguments**

type	Character string. Options: <ul style="list-style-type: none"> <li>• "polynomial" for orthogonal polynomial features</li> <li>• "linear", "sine", "circle", "cluster", "snake", "outliers", "sparse", "clumpy", "skewed", "striated", "concave", "monotonic", "doughnut", or "all" to generate all scatter structures.</li> </ul>
n	Integer. Number of samples to generate. Default is 500.
degree	Integer. Degree of polynomial features (only for type = "polynomial").
seed	Optional integer. Sets random seed for reproducibility.

**Value**

- If type = "polynomial", returns a matrix (n x degree).
- Otherwise a tibble with columns:
  - x: Numeric vector of x-values
  - y: Numeric vector of y-values
  - structure: Character name of the structure type

**Examples**

```
data_gen("linear", n = 200)
data_gen("polynomial", degree = 4, n = 200)
data_gen("all", n = 200)
```

---

distance\_dist                    *Collecting all pairwise distances between input planes.*

---

### Description

The distribution of all pairwise distances is useful to understand the optimisation in a guided tour, to compare e.g. different optimisation methods or different number of noise dimensions.

### Usage

```
distance_dist(planes, nn = FALSE)
```

### Arguments

planes	Input planes (e.g. result of guided tour)
nn	Set true to only consider nearest neighbour distances (dummy, not yet implemented)

### Value

numeric vector containing all distances

### Examples

```
planes1 <- purrr::rerun(10, tourr::basis_random(5))
planes2 <- purrr::rerun(10, tourr::basis_random(10))
d1 <- distance_dist(planes1)
d2 <- distance_dist(planes2)
d <- tibble::tibble(dist=c(d1, d2), dim=c(rep(5,length(d1)),rep(10,length(d2))))
ggplot2::ggplot(d) + ggplot2::geom_boxplot(ggplot2::aes(factor(dim), dist))
```

---

distance\_to\_sp                    *Collecting distances between input planes and input special plane.*

---

### Description

If the optimal view is known, we can use the distance between a given plane and the optimal one as a proxy to diagnose the performance of the guided tour.

### Usage

```
distance_to_sp(planes, special_plane)
```

### Arguments

planes	Input planes (e.g. result of guided tour)
special_plane	Plane defining the optimal view

**Value**

numeric vector containing all distances

**Examples**

```
planes <- purrr::rerun(10, tourr::basis_random(5))
special_plane <- basis_matrix(1,2,5)
d <- distance_to_sp(planes, special_plane)
plot(d)
```

---

get\_index\_mean

*Evaluate mean index value over n jittered views.*

---

**Description**

Evaluate mean index value over n jittered views.

**Usage**

```
get_index_mean(proj, d, alpha, idx, method = "jitter_angle", n = 10)
```

**Arguments**

proj	Original projection plane
d	Data matrix
alpha	Jitter amount (for jittering angle or points)
idx	Index function
method	Select between "jitterAngle" (default) and "jitterPoints" (otherwise we return original index value)
n	Number of evaluations entering mean value calculation

**Value**

Mean index value

---

get_trace	<i>Tracing the index over an interpolated planned tour path.</i>
-----------	--

---

**Description**

Tracing is used to test if the index value varies smoothly over an interpolated tour path. The index value is calculated for the data `d` in each projection in the interpolated sequence. Note that all index functions must take the data in 2-d matrix format and return the index value.

**Usage**

```
get_trace(d, m, index_list, index_labels)
```

**Arguments**

<code>d</code>	data
<code>m</code>	list of projection matrices for the planned tour
<code>index_list</code>	list of index functions to calculate for each entry
<code>index_labels</code>	labels used in the output

**Value**

index values for each interpolation step

**Examples**

```
d <- spiral_data(100, 4)
m <- list(basis_matrix(1,2,4), basis_matrix(3,4,4))
index_list <- list(tourr::holes(), tourr::cmass())
index_labels <- c("holes", "cmass")
trace <- get_trace(d, m, index_list, index_labels)
plot_trace(trace)
```

---

jitter_angle	<i>Re-evaluate index after jittering the projection by an angle alpha.</i>
--------------	--

---

**Description**

Re-evaluate index after jittering the projection by an angle `alpha`.

**Usage**

```
jitter_angle(proj, d, alpha, idx)
```

**Arguments**

proj	Original projection plane
d	Data matrix
alpha	Jitter angle
idx	Index function

**Value**

New index value

---

jitter_points	<i>Re-evaluate index after jittering all points by an amount alpha.</i>
---------------	---

---

**Description**

Re-evaluate index after jittering all points by an amount alpha.

**Usage**

```
jitter_points(proj_data, alpha, idx)
```

**Arguments**

proj_data	Original projected data points
alpha	Jitter amount (passed into the jitter() function)
idx	Index function

**Value**

New index value

---

noise_gen	<i>Generate Synthetic Noise</i>
-----------	---------------------------------

---

**Description**

Generate Synthetic Noise

**Usage**

```
noise_gen(n = 500, type = "gaussian", level = 0.01, seed = NULL)
```

**Arguments**

n	Integer. Number samples to generate. Default is 500.
type	Character string specifying the type of noise to generate. Supported types: <ul style="list-style-type: none"> <li>• "gaussian": Standard normal distribution.</li> <li>• "uniform": Uniform distribution between -level and +level.</li> <li>• "lognormal": Log-normal distribution.</li> <li>• "t_distributed": Heavy-tailed t-distribution with 3 degrees of freedom.</li> <li>• "cauchy": Extremely heavy-tailed Cauchy distribution.</li> <li>• "beta_noise": Beta distribution shifted and scaled to [-level, level].</li> <li>• "exponential": Positive-only exponential distribution.</li> <li>• "microstructure": Oscillatory sinusoidal pattern with additive Gaussian noise.</li> </ul>
level	Numeric. Controls the scale (standard deviation, range, or spread) of the noise. Default is 0.01.
seed	Optional integer. Sets a random seed for reproducibility.

**Value**

A tibble with two columns:

- value: Numeric vector of generated noise samples.
- type: Character string indicating the type of noise.

**Examples**

```
# Gaussian noise with small scale
noise_gen(500, type = "gaussian", level = 0.05)

# Heavy-tailed noise
noise_gen(500, type = "t_distributed", level = 0.1)
```

---

pipe\_data

*Generating a sample of points on a pipe*

---

**Description**

Points are drawn from a uniform distribution between -1 and 1, the pipe structure is generated by rejecting points if they are not on a circle with radius 1 and thickness *t* in the last two parameters.

**Usage**

```
pipe_data(n, p, t = 0.1)
```

**Arguments**

n	number of sample points to generate
p	sample dimensionality
t	thickness of circle, default=0.1

**Value**

sample points in tibble format

**Examples**

```
pipe_data(100, 4)
pipe_data(100, 2, 0.5)
```

---

plot_rotation	<i>Plot rotation traces of indexes obtained with profileRotation.</i>
---------------	---

---

**Description**

Plot rotation traces of indexes obtained with profileRotation.

**Usage**

```
plot_rotation(res_mat)
```

**Arguments**

res_mat	data (result of profileRotation)
---------	----------------------------------

**Value**

ggplot visualisation of the tracing data

---

plot_smoothing_comparison	<i>Plot the comparison of smoothing methods.</i>
---------------------------	--

---

**Description**

Plotting method for the results of compareSmoothing. The results are mapped by facetting over values of alpha and mapping the method (jitter\_angle, jitter\_points, no\_smoothing) to linestyle and color (black dashed, black dotted, red solid). By default legend drawing is turned off, but can be turned on via the lPos argument, e.g. setting to "bottom" for legend below the plot.

**Usage**

```
plot_smoothing_comparison(sm_mat, lPos = "none")
```

**Arguments**

sm_mat	Result from compare_smoothing
lPos	Legend position passed to ggplot2 (default is none for no legend shown)

**Value**

ggplot visualisation of the comparison

---

plot_trace	<i>Plot traces of indexes obtained with <a href="#">get_trace</a>.</i>
------------	--

---

**Description**

Plot traces of indexes obtained with [get\\_trace](#).

**Usage**

```
plot_trace(res_mat, rescY = TRUE)
```

**Arguments**

res_mat	data (result of <a href="#">get_trace</a> )
rescY	bool to fix y axis range to [0,1]

**Value**

ggplot visualisation of the tracing data

---

ppi_mean	<i>Simulate and Summarize Projection Pursuit Index (PPI) Values</i>
----------	---

---

**Description**

Simulate and Summarize Projection Pursuit Index (PPI) Values

**Usage**

```
ppi_mean(data, index_fun, n_sim = 100, n_obs = 300)
```

**Arguments**

data	A data frame or matrix. Must have at least two columns.
index_fun	A function taking two numeric vectors (x, y) and returning a scalar index.
n_sim	Integer. Number of simulations. Default is 100.
n_obs	Integer. Number of observations to sample in each simulation. Default is 300.

**Value**

A tibble with:

- var\_i, var\_j: Names of variable pairs
- mean\_index: Mean index value over simulations

**Examples**

```
data <- as.data.frame(data_gen(type = "polynomial", degree = 2))
ppi_mean(data, scag_index("stringy"), n_sim = 10)
```

---

ppi\_noise\_threshold *Estimate the 95th Percentile of a Projection Pursuit Index Under Noise*

---

**Description**

This function estimates the 95th percentile of a projection pursuit index under synthetic noise data.

**Usage**

```
ppi_noise_threshold(
  index_fun,
  n_sim = 100,
  n_obs = 500,
  noise_type = "gaussian",
  noise_level = 0.01,
  seed = NULL
)
```

**Arguments**

index_fun	A function that takes either a 2-column matrix or two numeric vectors and returns a scalar index.
n_sim	Integer. Number of index evaluations to simulate. Default is 100.
n_obs	Integer. Number of observations per noise sample. Default is 500.
noise_type	Character. Type of noise to use (e.g., "gaussian", "t_distributed", etc.). Default is "gaussian".
noise_level	Numeric. Controls the scale/spread of the generated noise. Default is 0.01.
seed	Optional integer. Random seed for reproducibility.

**Value**

A single numeric value: the estimated 95th percentile of the index under noise.

**Examples**

```
ppi_noise_threshold(
  index_fun = scag_index("stringy"),
  noise_type = "cauchy",
  noise_level = 0.1,
  n_sim = 10,
  n_obs = 100
)
```

---

ppi\_sample\_size\_effect *Simulate Effect of Sample Size on a Projection Pursuit Index*

---

**Description**

For a given index function, simulates how the index behaves across a range of sample sizes. By default, standard bivariate Gaussian noise is used, but the user may instead supply a 2-column dataset to sample from.

**Usage**

```
ppi_sample_size_effect(
  index_fun,
  sample_sizes = 100,
  n_sim = 100,
  data = NULL,
  csv_file = NULL,
  index_args = list(),
  workers = 1
)
```

**Arguments**

index_fun	A function returning a numeric scalar index. It may accept either a 2-column matrix/data frame, or two numeric vectors (x, y).
sample_sizes	Numeric vector (e.g., 100, c(50,100,200), seq(30,200,5)). Default is 100.
n_sim	Integer. Number of simulations per sample size.
data	Optional 2-column data frame, tibble, or matrix to sample from. If NULL, standard bivariate Gaussian noise is used.
csv_file	Optional file path to save results as CSV.
index_args	A named list of additional arguments passed to index_fun.
workers	Integer. Number of parallel workers to use. Default is 1.

**Value**

A tibble with:

- `sample_size`: sample size used for each simulation block
- `percentile95`: 95th percentile of the index values over simulations

**Examples**

```
## Not run:
ppi_samplesize_effect(
  cassowaryr::sc_stringy,
  sample_sizes = c(100, 300),
  n_sim = 3,
  index_args = list(binner = "hex"),
  workers = 2
)

ppi_samplesize_effect(
  cassowaryr::sc_skinny,
  sample_sizes = c(100, 300),
  n_sim = 3,
  data = data_gen("polynomial", degree = 2),
  workers = 2
)

## End(Not run)
```

---

ppi\_scale

*Simulate and Compare Index Scale on Structured vs Noisy Data*

---

**Description**

Performs simulations to compute a projection pursuit index on structured (sampled) data and on random noise, allowing a comparison of index scale across contexts.

**Usage**

```
ppi_scale(
  data,
  index_fun,
  noise_fun = NULL,
  n_sim = 100,
  n_obs = 500,
  index_args = list(),
  seed = NULL,
  workers = 1
)
```

**Arguments**

<code>data</code>	A data frame or tibble with at least two numeric columns.
<code>index_fun</code>	A function returning a numeric scalar index. It may accept either a 2-column matrix/data frame, or two numeric vectors (x, y).
<code>noise_fun</code>	Optional function for generating noise data. It should take <code>n_obs</code> as input and return a 2-column matrix or data frame. By default, standard bivariate Gaussian noise is used.
<code>n_sim</code>	Integer. Number of simulations. Default is 100.
<code>n_obs</code>	Integer. Number of observations per simulation. Default is 500.
<code>index_args</code>	A named list of additional arguments passed to <code>index_fun</code> .
<code>seed</code>	Optional integer seed for reproducibility.
<code>workers</code>	Integer. Number of parallel workers to use. Default is 1.

**Value**

A tibble with columns:

- `simulation`: simulation number
- `var_i, var_j`: variable names
- `var_pair`: pair name as a string
- `sigma`: "structure" for structured data, "noise" for noisy data
- `index`: index value returned by `index_fun`

**Examples**

```
ppi_scale(  
  data_gen("polynomial", degree = 3),  
  cassowaryr::sc_stringy,  
  n_sim = 2,  
  index_args = list(binner = "hex"),  
  workers = 2  
)
```

```
ppi_scale(  
  data_gen("polynomial", degree = 3),  
  scag_index("stringy"),  
  n_sim = 2  
)
```

---

profile_rotation	<i>Test rotation invariance of index functions for selected 2-d data set.</i>
------------------	---

---

### Description

Ideally a projection pursuit index should be rotation invariant, we test this explicitly by profiling the index while rotating a distribution.

### Usage

```
profile_rotation(d, index_list, index_labels, n = 200)
```

### Arguments

d	data (2 column matrix containing distribution to be rotated)
index_list	list of index functions to calculate for each entry
index_labels	labels used in the output
n	number of steps in the rotation (default = 200)

### Value

index values for each rotation step

### Examples

```
d <- as.matrix(sin_data(30, 2))
index_list <- list(tourr::holes(), scag_index("stringy"), mine_indexE("MIC"))
index_labels <- c("holes", "stringy", "mic")
pRot <- profile_rotation(d, index_list, index_labels, n = 50)
plot_rotation(pRot)
```

---

scag_index	<i>Matching index functions to the required format.</i>
------------	---

---

### Description

These are convenience functions that format scagnostics and mine index functions for direct use with the guided tour or other functionalities in this package.

**Usage**

```

scag_index(index_name)

mine_index(index_name)

mine_indexE(index_name)

holesR()

cmassR()

```

**Arguments**

index\_name      Index name to select from group of indexes.

**Value**

function taking 2-d data matrix and returning the index value

**Functions**

- scag\_index(): Scagnostics index from cassowaryr package
- mine\_index(): MINE index from minerva package
- mine\_indexE(): MINE index from minerva package (updated estimator)
- holesR(): rescaling the tourr holes index
- cmassR(): rescaling the tourr cmass index

---

sin\_data

*Generating sine wave sample*


---

**Description**

n-1 points are drawn from a normal distribution with mean=0, sd=1, the points in the final direction are calculated as the sine of the values of direction n-1 with additional jittering controlled by the jitter factor f.

**Usage**

```
sin_data(n, p, f = 1)
```

**Arguments**

n                    number of sample points to generate  
p                    sample dimensionality  
f                    jitter factor, default=1

**Value**

sample points in tibble format

**Examples**

```
sin_data(100, 4)
sin_data(100, 2, 200)
```

---

spiral\_data

*Generating spiral sample*

---

**Description**

n-2 points are drawn from a normal distribution with mean=0, sd=1, the points in the final two direction are sampled along a spiral by sampling the angle from a normal distribution with mean=0, sd=2\*pi (absolute values are used to fix the orientation of the spiral).

**Usage**

```
spiral_data(n, p)
```

**Arguments**

n	number of sample points to generate
p	sample dimensionality

**Value**

sample points in matrix format

**Examples**

```
spiral_data(100, 4)
```

---

`squint_angle_estimate` *Estimating squint angle of 2-d structure in high-d dataset under selected index.*

---

### Description

We estimate the squint angle by interpolating from a random starting plane towards the optimal view until the index value of the selected index function is above the selected cutoff. Since this depends on the direction, this is repeated with  $n$  randomly selected planes giving a distribution representative of the squint angle.

### Usage

```
squint_angle_estimate(  
  data,  
  indexF,  
  cutoff,  
  structure_plane,  
  n = 100,  
  step_size = 0.01  
)
```

### Arguments

<code>data</code>	Input data
<code>indexF</code>	Index function
<code>cutoff</code>	Threshold index value above which we assume the structure to be visible
<code>structure_plane</code>	Plane defining the optimal view
<code>n</code>	Number of random starting planes (default = 100)
<code>step_size</code>	Interpolation step size fixing the accuracy (default = 0.01)

### Value

numeric vector containing all squint angle estimates

### Examples

```
data <- spiral_data(50, 4)  
indexF <- scag_index("stringy")  
cutoff <- 0.7  
structure_plane <- basis_matrix(3,4,4)  
squint_angle_estimate(data, indexF, cutoff, structure_plane, n=1)
```

---

time_sequence	<i>Time each index evaluation for projections in the tour path.</i>
---------------	---

---

**Description**

Index evaluation timing may depend on the data distribution, we evaluate the computing time for a set of different projections to get an overview of the distribution of computing times.

**Usage**

```
time_sequence(d, t, idx, pmax)
```

**Arguments**

d	Input data in matrix format
t	List of projection matrices (e.g. interpolated tour path)
idx	Index function
pmax	Maximum number of projections to evaluate (cut t if longer than pmax)

**Value**

numeric vector containing all distances

**Examples**

```
d <- as.matrix(spiral_data(500, 4))
t <- purrr::map(1:10, ~ tourr::basis_random(4))
idx <- scag_index("stringy")
time_sequence(d, t, idx, 10)
```

# Index

basis\_matrix, 2  
basis\_vector, 3

cmassR(scag\_index), 16  
compare\_smoothing, 3

data\_gen, 4  
distance\_dist, 5  
distance\_to\_sp, 5

get\_index\_mean, 6  
get\_trace, 7, 11

holesR(scag\_index), 16

jitter\_angle, 7  
jitter\_points, 8

mine\_index(scag\_index), 16  
mine\_indexE(scag\_index), 16

noise\_gen, 8

pipe\_data, 9  
plot\_rotation, 10  
plot\_smoothing\_comparison, 10  
plot\_trace, 11  
ppi\_mean, 11  
ppi\_noise\_threshold, 12  
ppi\_sample\_size\_effect, 13  
ppi\_scale, 14  
profile\_rotation, 16

scag\_index, 16  
sin\_data, 17  
spiral\_data, 18  
squint\_angle\_estimate, 19

time\_sequence, 20