LikertMakeR

synthesise and correlate related rating-scale data

Hume Winzar

February 2024

LikertMakeR

LikertMakeR synthesises and correlates Likert-scale and related rating-scale data. You decide the mean and standard deviation, and (optionally) the correlations among vectors, and the package will generate data with those same predefined properties.

The package generates a column of values that simulate the same properties as a rating scale. If multiple columns are generated, then you can use ***LikertMakeR*** to rearrange the values so that the new variables are correlated exactly in accord with a user-predefined correlation matrix.

Purpose

The package should be useful for teaching in the Social Sciences, and for scholars who wish to "replicate" rating-scale data for further analysis and visualisation when only summary statistics have been reported.

Motivation

I was prompted to write the functions in *LikertMakeR* after reviewing too many journal article submissions where authors presented questionnaire results with only means and standard deviations (often only the means), with no apparent understanding of scale distributions, and their impact on scale properties. Hopefully, this tool will help researchers, teachers, and other reviewers, to better think about rating-scale distributions, and the effects of variance, scale boundaries, and number of items in a scale.

Rating scale properties

A Likert scale is the mean, or sum, of several ordinal rating scales. Typically, they are bipolar (usually "agree-disagree") responses to propositions that are determined to be moderately-to-highly correlated and that capture various facets of a theoretical construct.

Rating scales, such as Likert scales, are not continuous or unbounded.

For example, a 5-point Likert scale that is constructed with, say, five items (questions) will have a summed range of between 5 (all rated '1') and 25 (all rated '5') with all integers in between, and the mean range will be ‘1’ to ‘5’ with intervals of 1/5=0.20. A 7-point Likert scale constructed from eight items will have a summed range between 8 (all rated ‘1’) and 56 (all rated ‘7’) with all integers in between, and the mean range will be ‘1’ to ‘7’ with intervals of 1/8=0.125.

Rating-scale boundaries define minima and maxima for any scale values. If the mean is close to one boundary then data points will gather more closely to that boundary and the data will always be skewed.

LikertMakeR **functions**

- *lfast()* and *lfast_R()* quickly generate a vector of values with approximate predefined moments
- *lexact()* also generates a vector of values, but accurate to two decimal places. (*lfast_R()* does this too, but more quickly, so lexact()_ remains as a legacy function)
- *lcor()* and *lcor_C()* take a dataframe of rating-scale values and rearranges the values in each column so that the columns are correlated to match a predefined correlation matrix.
- *makeCorrAlpha* constructs a random correlation matrix of given dimensions and predefined Cronbach’s Alpha
- *makeItems()* is a wrapper function for *lfast_R()* and *lcor_C()* to generate synthetic rating-scale data with predefined first and second moments and a predefined correlation matrix
- *alpha()* calculates Cronbach’s Alpha from a given correlation matrix or a given dataframe
- *eigenvalues()* calculates eigenvalues of a correlation matrix, reports on positive-definite status of the matrix and, optionally, displays a scree plot to visualise the eigenvalues

---

Using ***LikertMakeR***

Download and Install ***LikertMakeR***

from ***CRAN***

```
install.packages("LikertMakeR")

library(LikertMakeR)
```

development version from ***GitHub.***

```
library(devtools)

install_github("WinzarH/LikertMakeR")

library(LikertMakeR)
```

Generate synthetic rating-scale data

To synthesise a rating scale with ***LikertMakeR***, the user must input the following parameters:

- ***n***: sample size
- ***mean***: desired mean
- ***sd***: desired standard deviation

- ***lowerbound***: desired lower bound
- ***upperbound***: desired upper bound
- ***items***: number of items making the scale - default = 1
- ***seed***: optional seed for reproducibility

LikertMakeR offers three functions for synthesising a rating scale: ***lfast()*** and ***lexact()***, and a recent addition: ***lfast_R()***. *lfast_R()* is likely to replace *lexact()* in the next submission to CRAN.

---

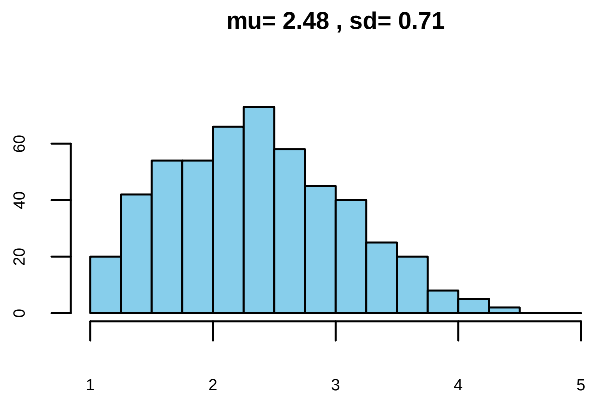
lfast() and *lfast_R()*

- ***lfast()*** draws a random sample from a scaled *Beta* distribution. It is very fast but does not guarantee exact mean and standard deviation. Recommended for relatively large sample sizes.
- ***lfast_R()*** draws repeated random samples from a scaled *Beta* distribution. It is slightly slower than *lfast()* but usually produces a vector with moments accurate to two decimal places.

lfast() & *lfast_R()* example

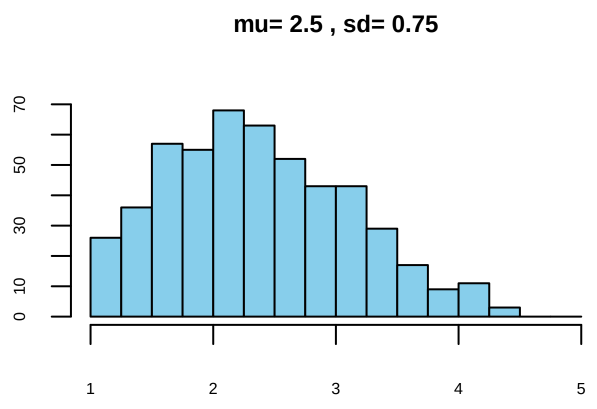
a four-item, five-point Likert scale *lfast()*

```r
x1 <- lfast(
  n = 512,
  mean = 2.5,
  sd = 0.75,
  lowerbound = 1,
  upperbound = 5,
  items = 4
)
```



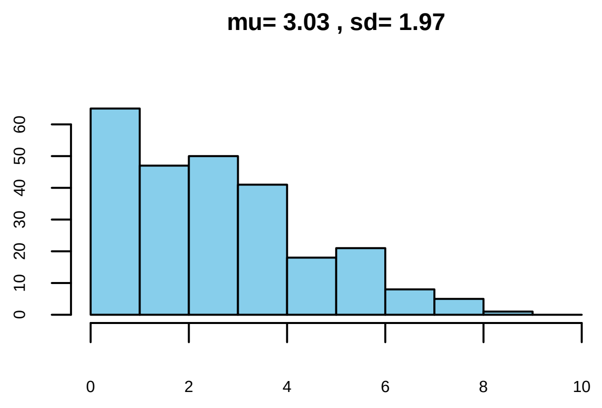
lfast_R()

```r
x2 <- lfast_R(
  n = 512,
  mean = 2.5,
  sd = 0.75,
  lowerbound = 1,
  upperbound = 5,
  items = 4
)
#> [1] "best solution in 332 iterations"
```



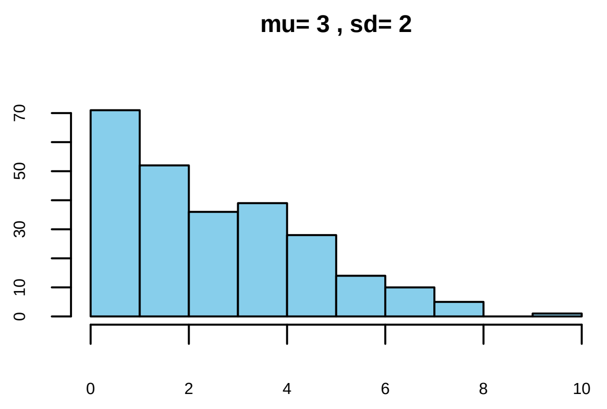
an 11-point likelihood-of-purchase scale *lfast()*

```r
x3 <- lfast(256, 3, 2, 0, 10)
```



lfast_R()

```r
x4 <- lfast_R(256, 3, 2, 0, 10)
#> [1] "best solution in 4172 iterations"
```



lexact()

lexact() attempts to produce a vector with exact first and second moments.

If feasible, ***lexact()*** should produce data with moments that are correct to two decimal places. Infeasible cases occur when the requested standard deviation is too large for the combination of mean, n-items, and scale boundaries.

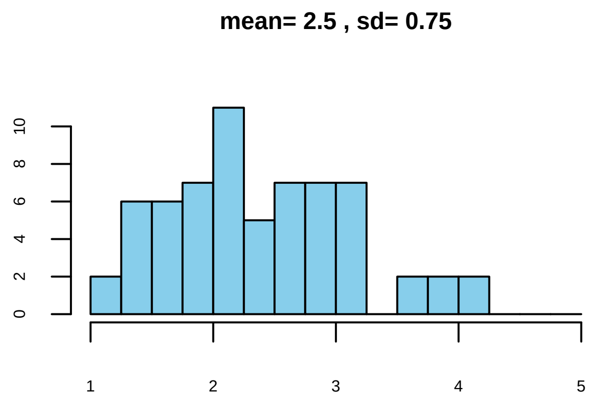
The optimisation process means that the function is considerably slower than ***lfast()*** or than ***lfast_R()***: taking exponentially more time with the number of items, and linearly more time with the sample size.

lexact() uses the *Differential Evolution* algorithm in the ***DEoptim*** package to find appropriate values within the desired constraints. The DEoptim package is described in Mullen, Ardia, Gil, Windover, & Cline (2011) doi:10.18637/jss.v040.i06.

lexact() example #1

```
x <- lexact(
  n = 64,
  mean = 2.5,
  sd = 0.75,
  lowerbound = 1,
  upperbound = 5,
  items = 4
)
#>
#> ***** summary of DEoptim object *****
#> best member   :  13 12 13 15 6 7 6 11 6 13 8 11 13 13 9 9 13 6 11 8 7 17 11 8 8 7 7 12 4 17 8 13 9 1
#> best value    :  0.06993
#> after         :  640 generations
#> fn evaluated  :  410240 times
#> *************************************
```

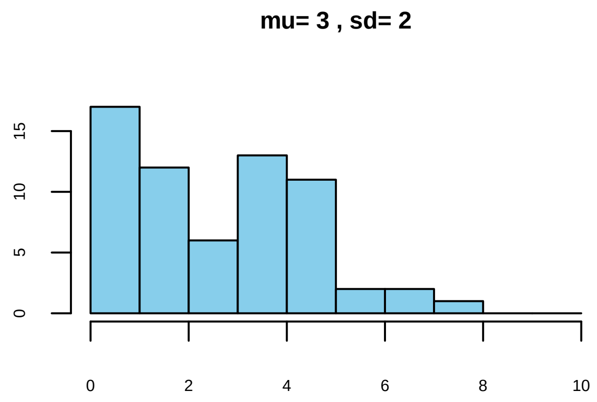
a four-item, five-point Likert scale



***lexact()* example #2**

```
x <- lexact(64, 3, 2, 0, 10)
#>
#> ***** summary of DEoptim object *****
#> best member   :  4 6 1 5 0 4 5 2 4 5 2 4 5 4 2 0 3 1 4 3 1 3 5 7 2 6 2 1 0 1 1 3 5 7 0 4 4 5 2 1 4 3
#> best value    :  0
#> after         :  80 generations
#> fn evaluated  :  51840 times
#> *************************************
```

11-point likelihood-of-purchase scale



Correlating vectors of synthetic rating scales

LikertMakeR offers another function, ***lcor()***, which rearranges the values in the columns of a data-set so that they are correlated at a specified level. It does not change the values - it swaps their positions within each column so that univariate statistics do not change, but their correlations with other vectors do.

lcor() and *lcor_C()*

lcor() systematically selects pairs of values in a column and swaps their places, and checks to see if this swap improves the correlation matrix. If the revised data-frame produces a correlation matrix closer to the target correlation matrix, then the swap is retained. Otherwise, the values are returned to their original places. This process is iterated across each column.

lcor_C() is an implementation of *lcor()* written in C++ it is here as a trial ahead of replacing *lcor()* in the near future. It is much faster!

To create the desired correlated data, the user must define the following data-frames:

- ***data***: a starter data set of rating-scales
- ***target***: the target correlation matrix

lcor() example

Let's generate some data: three 5-point Likert scales, each with five items.

```
## generate uncorrelated synthetic data

n <- 128
lowerbound <- 1
upperbound <- 5
items <- 5

mydat3 <- data.frame(
  x1 = lfast_R(n, 2.5, 0.75, lowerbound, upperbound, items),
  x2 = lfast_R(n, 3.0, 1.50, lowerbound, upperbound, items),
  x3 = lfast_R(n, 3.5, 1.00, lowerbound, upperbound, items)
)
#> [1] "best solution in 167 iterations"
#> [1] "best solution in 477 iterations"
#> [1] "best solution in 141 iterations"
```

The first five observations from this data-frame are:

```
#>    x1  x2  x3
#> 1 1.4 2.2 4.6
#> 2 3.8 1.2 3.0
#> 3 3.0 2.6 4.6
#> 4 1.4 5.0 3.2
#> 5 3.2 1.0 2.2
```

And the first and second moments (to 3 decimal places) are:

```
#>        x1    x2    x3
#> mean 2.50 3.002 3.500
#> sd   0.75 1.500 1.002
```

We can see that the data have first and second moments very close to what is expected.

The synthetic data have low correlations:

```
#>       x1    x2    x3
#> x1  1.00  0.24 -0.04
#> x2  0.24  1.00 -0.16
#> x3 -0.04 -0.16  1.00
```

```
## describe a target correlation matrix

tgt3 <- matrix(
  c(
    1.00, 0.85, 0.75,
    0.85, 1.00, 0.65,
    0.75, 0.65, 1.00
  ),
  nrow = 3
)
```

So now we have a data-frame with desired first and second moments, and a target correlation matrix.

```
## apply lcor() function

new3 <- lcor(mydat3, tgt3)
```

The first column of the new data-frame will not change, but values of the other columns are rearranged.

The first five observations from this data-frame are:

```
#>    x1  x2  x3
#> 1 1.4 1.2 1.4
#> 2 3.8 5.0 4.4
#> 3 3.0 3.2 4.0
#> 4 1.4 1.0 1.0
#> 5 3.2 4.6 4.8
```

And the new data frame is correlated close to our desired correlation matrix:

```
#>      x1   x2   x3
#> x1 1.00 0.85 0.75
#> x2 0.85 1.00 0.65
#> x3 0.75 0.65 1.00
```

```
## apply lcor_C function

new3C <- lcor_C(mydat3, tgt3)
```

lcor_C() should run more quickly than *lcor()*, with the same output:

```
#>      V1   V2   V3
#> V1 1.00 0.85 0.75
#> V2 0.85 1.00 0.65
#> V3 0.75 0.65 1.00
```

Generate a correlation matrix from Cronbach's Alpha

makeCorrAlpha()

makeCorrAlpha(), constructs a random correlation matrix of given dimensions and predefined Cronbach's Alpha

Random values generated by *makeCorrAlpha()* are volatile. *makeCorrAlpha()* may not generate a feasible (positive-definite) correlation matrix, especially when

- variance is high relative to
  - desired Alpha, and
  - desired correlation dimensions

makeCorrAlpha() will inform the user if the resulting correlation matrix is positive definite, or not.

If the returned correlation matrix is not positive-definite, because solutions are so volatile, a feasible solution still may be possible, and often is. The user is encouraged to try again, possibly several times, to find one.

makeCorrAlpha() examples

```


## define parameters

    items <- 4
    alpha <- 0.85
    variance <- 0.5


## apply makeCorrAlpha() function

    set.seed(42)

    cor_matrix_4 <- makeCorrAlpha(items, alpha, variance)
#> correlation values consistent with desired alpha in 2261 iterations
#> The correlation matrix is positive definite
```

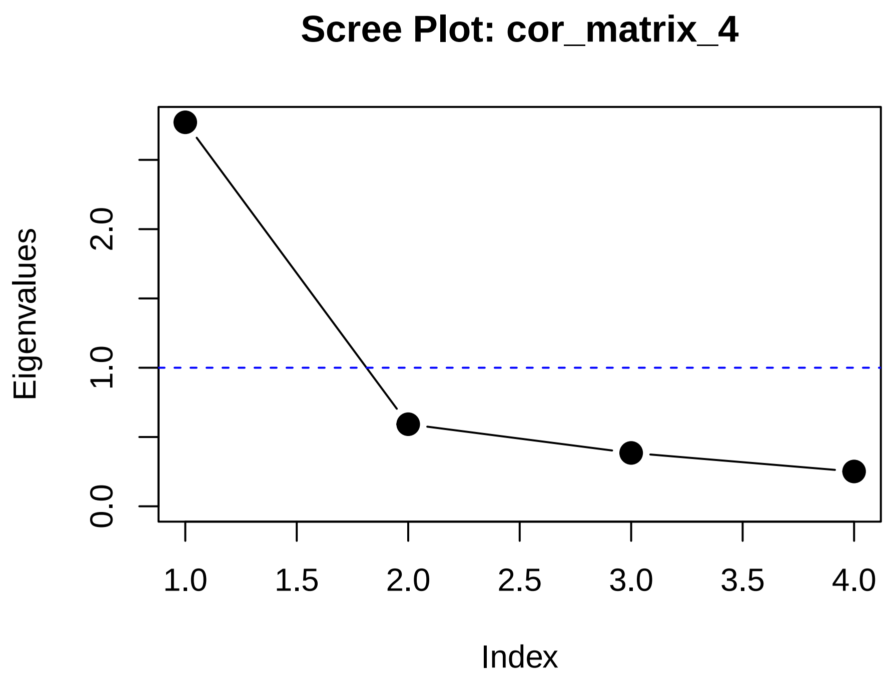
four variables, Alpha = 0.85

test output with Helper functions Cronbach's Alpha for the generated correlation matrix is:

```
## using helper function alpha()

    alpha(cor_matrix_4)
#> [1] 0.8500036
```

And the correlation matrix is positive-definite:

```
## using helper function eigenvalues()

eigenvalues(cor_matrix_4, 1)
```



```
#> cor_matrix_4  is positive-definite
#>
#> Eigenvalues:
#>  2.771181 0.5922674 0.3851331 0.2514183
#> [1] 2.7711812 0.5922674 0.3851331 0.2514183
```

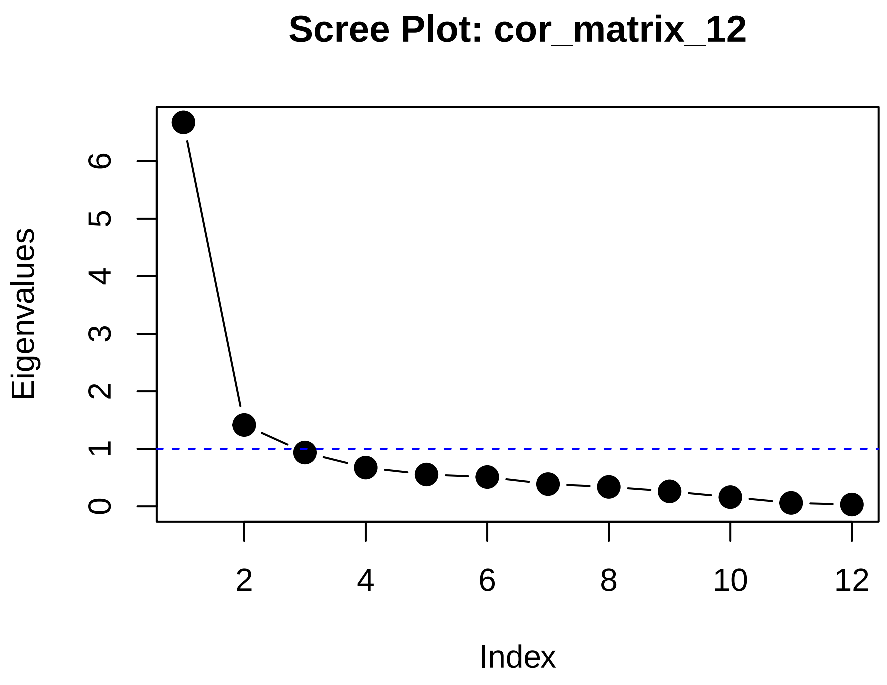
```
## define parameters

   items <- 12
   alpha <- 0.90
   variance <- 1.0


## apply makeCorrAlpha() function

    set.seed(42)

    cor_matrix_12 <- makeCorrAlpha(items, alpha, variance)
#> correlation values consistent with desired alpha in 26735 iterations
#> The correlation matrix is positive definite
```

four variables, Alpha = 0.85, larger variance

```
    alpha(cor_matrix_12)
#> [1] 0.9000022
```

```r
    eigenvalues(cor_matrix_12, 1)
```

test output



```
#> cor_matrix_12  is positive-definite
#> 
#> Eigenvalues:
#>  6.6755 1.414525 0.935596 0.6735671 0.5536723 0.5100483 0.3867301 0.3380953 0.2605829 0.1603447 0.05
#>  [1] 6.67549959 1.41452522 0.93559602 0.67356712 0.55367229 0.51004832
#>  [7] 0.38673007 0.33809529 0.26058292 0.16034466 0.05985932 0.03147918
```

Generate a dataframe of rating scales from a correlation matrix and predefined moments

makeItems()

makeItems() generates a dataframe of random discrete values from a *scaled Beta distribution* so the data replicate a rating scale, and are correlated close to a predefined correlation matrix.

Generally, means, standard deviations, and correlations are correct to two decimal places.

makeItems() is a wrapper function for

- *lfast_R()*, which takes repeated samples selecting a vector that best fits the desired moments, and
- *lcor_C()*, which rearranges values in each column of the dataframe so they closely match the desired correlation matrix.

makeItems() examples

```r
## define parameters
```

```
    n <- 128

    lowerbound <- rep(1, 4)
    upperbound <- rep(5, 4)

    dfMeans <- c(2.5, 3.0, 3.0, 3.5)
    dfSds <- c(1.0, 1.0, 1.5, 0.75)

    corMat <- matrix(
    c(
     1.00, 0.25, 0.35, 0.40,
     0.25, 1.00, 0.70, 0.75,
     0.35, 0.70, 1.00, 0.80,
     0.40, 0.75, 0.80, 1.00
     ),
     nrow = 4, ncol = 4
    )

## apply makeItems() function

    df <- makeItems(
       n = n,
       lowerbound = lowerbound,
       upperbound = upperbound,
       means = dfMeans,
       sds = dfSds,
       cormatrix = corMat
       )
#> [1] "best solution in 16384 iterations"
#> [1] "best solution in 16384 iterations"
#> [1] "best solution in 727 iterations"
#> [1] "best solution in 16384 iterations"

## test the function

    head(df); tail(df)
#>   V1 V2 V3 V4
#> 1  3  1  1  2
#> 2  2  1  1  2
#> 3  4  5  5  5
#> 4  4  5  5  5
#> 5  1  3  4  4
#> 6  3  1  1  2
#>     V1 V2 V3 V4
#> 123  2  3  1  3
#> 124  2  3  1  3
#> 125  3  4  5  4
#> 126  4  3  2  4
#> 127  2  3  2  4
#> 128  4  3  3  4

    apply(df, 2, mean) |> round(3)
#>  V1   V2   V3   V4
#> 2.5 3.0 3.0 3.5

    apply(df, 2, sd) |> round(3)
#>    V1    V2    V3    V4
#> 1.004 1.004 1.501 0.753

    cor(df) |> round(3)
#>       V1   V2    V3    V4
#> V1 1.000 0.25 0.350 0.396
#> V2 0.250 1.00 0.700 0.750
#> V3 0.350 0.70 1.000 0.801
#> V4 0.396 0.75 0.801 1.000
```

Helper functions

likertMakeR() includes two additional functions that may be of help when examining parameters and output.

- ***alpha()*** calculates Cronbach's Alpha from a given correlation matrix or a given dataframe
- ***eigenvalues()*** calculates eigenvalues of a correlation matrix, a report on whether the correlation matrix is positive definite, and produces an optional scree plot.

alpha()

alpha() accepts, as input, either a correlation matrix or a dataframe. If both are submitted, then the correlation matrix is used by default, with a message to that effect.

alpha() examples

```
## define parameters

### example data frame

    df <- data.frame(
     V1  =  c(4, 2, 4, 3, 2, 2, 2, 1),
     V2  =  c(4, 1, 3, 4, 4, 3, 2, 3),
     V3  =  c(4, 1, 3, 5, 4, 1, 4, 2),
     V4  =  c(4, 3, 4, 5, 3, 3, 3, 3)
    )


### example correlation matrix

    corMat <- matrix(
     c(
      1.00, 0.35, 0.45, 0.70,
      0.35, 1.00, 0.60, 0.55,
      0.45, 0.60, 1.00, 0.65,
      0.70, 0.55, 0.65, 1.00
     ),
     nrow = 4, ncol = 4
    )
```

```
## apply function examples

    alpha(cor_matrix = corMat)
#> [1] 0.8301887

    alpha( , df)
#> [1] 0.830008

    alpha(corMat, df)
#> Warning: Both cor_matrix and data present.
#>
#> Using cor_matrix by default.
#> [1] 0.8301887
```

eigenvalues()

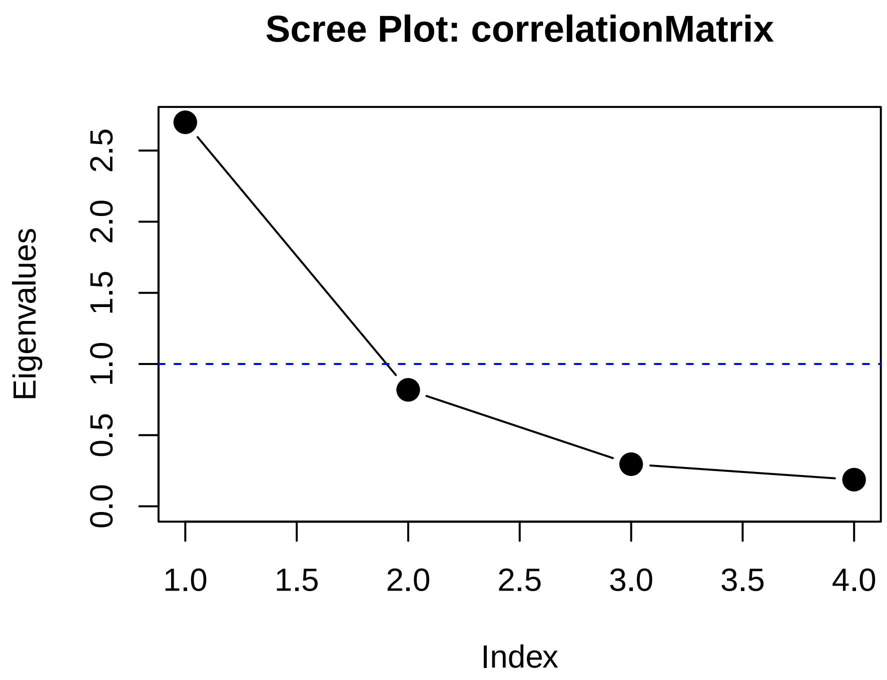
eigenvalues() calculates eigenvalues of a correlation matrix, reports on whether the matrix is positive-definite, and optionally produces a scree plot.

eigenvalues() examples

```

## define parameters

    correlationMatrix <- matrix(
     c(
      1.00, 0.25, 0.35, 0.40,
      0.25, 1.00, 0.70, 0.75,
      0.35, 0.70, 1.00, 0.80,
      0.40, 0.75, 0.80, 1.00
     ),
     nrow = 4, ncol = 4
    )


## apply function

    evals <- eigenvalues(cormatrix = correlationMatrix)
#> correlationMatrix  is positive-definite
#>
#> Eigenvalues:
#>  2.698682 0.8182523 0.2958544 0.1872111

    print(evals)
#> [1] 2.6986821 0.8182523 0.2958544 0.1872111
```

```


    evals <- eigenvalues(correlationMatrix, 1)
```

eigenvalues() function with optional scree plot



```
#> correlationMatrix  is positive-definite
#>
#> Eigenvalues:
#>  2.698682 0.8182523 0.2958544 0.1872111
```

Alternative methods & packages

LikertMakeR is intended for synthesising & correlating rating-scale data with means, standard deviations, and correlations as close as possible to predefined parameters. If you don't need your data to be close to exact, then other options may be faster or more flexible.

Different approaches include:

- sampling from a *truncated normal* distribution
- sampling with a predetermined probability distribution
- marginal model specification

sampling from a *truncated normal* distribution

Data are sampled from a normal distribution, and then truncated to suit the rating-scale boundaries, and rounded to set discrete values as we see in rating scales.

See Heiz (2021) for an excellent and short example using the following packages:

- truncnorm
- faux

- See also the *rLikert()* function from the responsesR package, Lalovic (2021), for an approach using optimal discretization and skew-normal distribution.

sampling with a predetermined probability distribution

- the following code will generate a vector of values with approximately the given probabilities. Good for simulating a single item.

```r
n <- 128
sample(1:5, n, replace = TRUE,
  prob = c(0.1, 0.2, 0.4, 0.2, 0.1)
)
```

marginal model specification

Marginal model specification extends the idea of a predefined probability distribution to multivariate and correlated data-frames.

- SimCorrMix: Simulation of Correlated Data with Multiple Variable Types Including Continuous and Count Mixture Distributions on CRAN.
- SimMultiCorrData: Simulation of Correlated Data with Multiple Variable Types on CRAN.
- lsasim: Functions to Facilitate the Simulation of Large Scale Assessment Data on CRAN. See Matta et al. (2018)
- GenOrd:Simulation of Discrete Random Variables with Given Correlation Matrix and Marginal Distributions on CRAN.
- SimCorMultRes: Simulates Correlated Multinomial Responses on CRAN. See Touloumis (2016)
- covsim: VITA, IG and PLSIM Simulation for Given Covariance and Marginals on CRAN. See Grønneberg et al. (2022)

References

Grønneberg, S., Foldnes, N., & Marcoulides, K. M. (2022). covsim: An R Package for Simulating Non-Normal Data for Structural Equation Models Using Copulas. *Journal of Statistical Software*, 102(1), 1–45. doi:10.18637/jss.v102.i03

Heinz, A. (2021), Simulating Correlated Likert-Scale Data In R: 3 Simple Steps (blog post) https://glaswasser.github.io/simulating-correlated-likert-scale-data/

Lalovic, M. (2021), *responsesR*: Simulate Likert scale item responses (on GitHub) https://github.com/markolalovic/responsesR

Matta, T.H., Rutkowski, L., Rutkowski, D. & Liaw, Y.L. (2018), lsasim: an R package for simulating large-scale assessment data. *Large-scale Assessments in Education* 6, 15. doi:10.1186/s40536-018-0068-8

Mullen, K. M., Ardia, D., Gil, D. L., Windover, D., & Cline, J. (2011). DEoptim: An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6), 1–26. doi:10.18637/jss.v040.i06

Touloumis, A. (2016), Simulating Correlated Binary and Multinomial Responses under Marginal Model Specification: The SimCorMultRes Package, *The R Journal* 8:2, 79-91. doi:10.32614/RJ-2016-034