Package ‘SimJoint’

April 14, 2020

Type Package

Title Simulate Joint Distribution

Version 0.3.7

Author Charlie Wusuo Liu

Maintainer Charlie Wusuo Liu <liuwusuo@gmail.com>

Description Simulate multivariate correlated data given nonparametric marginals and their joint structure characterized by a Pearson or Spearman correlation matrix. The simulator engages the problem from a purely computational perspective. It assumes no statistical models such as copulas or parametric distributions, and can approximate the target correlations regardless of theoretical feasibility. The algorithm integrates and advances the Iman-Conover (1982) approach <doi:10.1080/03610918208812265> and the Ruscio-Kaczetow iteration (2008) <doi:10.1080/00273170802285693>. Package functions are carefully implemented in C++ for squeezing computing speed, suitable for large input in a manycore environment. Precision of the approximation and computing speed both substantially outperform various CRAN packages to date. Benchmarks are detailed in function examples. A simple heuristic algorithm is additionally designed to optimize the joint distribution in the post-simulation stage. The heuristic demonstrated good potential of achieving the same level of precision of approximation without the enhanced Iman-Conover-Ruscio-Kaczetow. The package contains a copy ofPermuted Congruential Generator from <http://www.pcg-random.org>.

License GPL-3

Encoding UTF-8

LazyData true

Imports Rcpp (>= 1.0.0), RcppParallel

LinkingTo Rcpp, RcppParallel, RcppArmadillo

SystemRequirements GNU make

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-04-14 17:20:09 UTC
**R topics documented:**

- `decor` ......................................................... 2
- `exportRandomState` ................................. 3
- `LHSpmf` .................................................. 4
- `postSimOpt` ............................................ 5
- `SJpearson` .............................................. 7
- `SJpearsonPMF` ........................................ 15
- `SJpearsonPMF` ........................................ 18
- `SJpearsonPMF` ........................................ 20
- `xSJpearson` ............................................. 23
- `xSJpearsonPMF` ........................................ 25

**Index**  28

---

**decor**

Create uncorrelated data

**Description**

Create a matrix where columns are (Pearson) uncorrelated.

**Usage**

```r
decor(seedMat)
```

**Arguments**

- `seedMat` A matrix where the number of rows is no less than the number of columns. The function will change `seedMat`.

**Details**

Algorithm: for `i = 2` to `ncol(seedMat)`, the function replaces the first `i - 1` elements of the `i`th column with values such that the new `i`th column becomes uncorrelated with the first `i - 1` columns.

**Value**

None.

**Examples**

```r
set.seed(123)
X = matrix(rnorm(1000), ncol = 10)
corMat = cor(X)
summary(corMat[corMat < 1]) # Off-diagonal.
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -0.19271 -0.05648 -0.02272 -0.01303 0.01821 0.24521
```
SimJoint::decor(X)
corMat2 = cor(X)
summary(corMat2[corMat2 < 1])
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -2.341e-17 -3.627e-18 3.766e-18 4.018e-18 1.234e-17 3.444e-17

---

**exportRandomState**

Export Permuted Congruential Generator

**Description**

Export all the bits needed for seeding Permuted Congruential Generator.

**Usage**

`exportRandomState(seed)`

**Arguments**


**Details**

The returned integer vector `Value` supplies all the bits necessary for determining the state of a pcg64 generator. `Value` can seed for all functions that need a RNG in this package. It will change after the function call, ready for seeding the pcg64 generator in the next function call.

**Value**

An integer vector of size 4.

**Examples**

```r
# Make a random PMF.
set.seed(456)
val = seq(0, 15, len = 100)
pmf = data.frame(
  val = val, P = dgamma(val, shape = 2, scale = 2) + runif(100) * 0.1)
pmf$P = pmf$P / sum(pmf$P)

# `completeRandomState` comprises all the bits of a pcg64 engine seeded by 456. It is similar to R's `Random.seed`.
completeRandomState = SimJoint::exportRandomState(456)
# `completeRandomState` is changed in each run of `LHSpmf()`.
```

---

```r
SimJoint::decor(X)
corMat2 = cor(X)
summary(corMat2[corMat2 < 1])
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -2.341e-17 -3.627e-18 3.766e-18 4.018e-18 1.234e-17 3.444e-17
```
LHSpmf

Sample from probability mass function

Description

Sample from a probability mass function (PMF) via Latin hypercube sampling.

Usage

LHSpmf(pmf, sampleSize, seed)

Arguments

pmf
A 2-column data frame as a PMF. The 1st column is sorted and contains value points. The 2nd column contains probabilities. Probabilities should sum up to 1.

sampleSize
Sample size.

seed
An integer vector of size 1 or 4. Both seed a pcg64 RNG while the latter gives the complete state of the RNG.

Value

Random samples from pmf as a numeric vector of size sampleSize.

Examples

# Make a random PMF.
val = seq(0, 15, len = 100)
pmf = data.frame(val = val, P = dgamma(val, shape = 2, scale = 2))

pmf$P = pmf$P / sum(pmf$P)
pmfSample = SimJoint::LHSpmf(pmf, 1000, 123)
hist(pmfSample, breaks = 200)
postSimOpt

Post simulation optimization

Description
Impose the target correlation matrix via a heuristic algorithm.

Usage
postSimOpt(
  X,
  cor,
  Xcor = matrix(),
  acceptProb = 1,
  seed = 123L,
  convergenceTail = 10000L
)

Arguments
X An N x K numeric matrix of K marginal distributions (samples). Columns need not be sorted.

cor A K x K target correlation matrix. The matrix should be positive semi-definite.

Xcor The K x K correlation matrix of X. If empty, calculate the correlations inside. Default empty.

acceptProb A numeric vector of probabilities that sum up to 1. In each iteration, the entry having the largest error in the current correlation matrix will be selected with probability acceptProb[1] for correction; the entry having the second largest error will be selected with probability acceptProb[2] for correction, etc. Default 1, meaning the entry with the worst error is always chosen.

seed An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object. Default 123.

convergenceTail An integer. If the last convergenceTail iterations did not reduce the cost function, return. Default 100000.

Details
Algorithms are detailed in the package vignette. Examples of usage also appeared in functions like SJpearson().

Value
A list of size 2.

X A numeric matrix of size N x K, the simulated joint distribution.

Cor Pearson correlation matrix of X.
Examples

```r
# =============================================================================
# Use one of the examples for SJpearson()
# =============================================================================
set.seed(123)
N = 10000L
K = 10L

# Several 2-parameter PDFs in R:
marginals = list(rbeta, rcauchy, rf, rgamma, rnorm, runif, rweibull)
Npdf = length(marginals)

if(Npdf >= K) chosenMarginals =
    marginals[sample(Npdf, K, replace = TRUE)] else chosenMarginals =
    marginals[c(1L : Npdf, sample(Npdf, K - Npdf, replace = TRUE))]

# Sample from the marginal PDFs.
marginals = as.matrix(as.data.frame(lapply(chosenMarginals, function(f)
{
    para = sort(runif(2, 0.1, 10))
    rst = f(N, para[1], para[2])
    sort(rst)
})))
dimnames(marginals) = NULL

frechetUpperCor = cor(marginals) # The correlation matrix should be
# upper-bounded by that of the perfectly rank-correlated
# joint (Frechet upper bound). The lower bound is characterized by
# d-countercomonotonicity and depends not only on marginals.
cat("Range of maximal correlations between marginals:",
    range(frechetUpperCor[frechetUpperCor < 1]))
# Two perfectly rank-correlated marginals can have a Pearson
# correlation below 0.07. This is due to highly nonlinear functional
# relationships between marginal PDFs.

# Create a valid correlation matrix upper-bounded by `frechetUpperCor`.
while(TRUE)
{
    targetCor = sapply(frechetUpperCor, function(x)
        runif(1, -0.1, min(0.3, x * 0.8)))
    targetCor = matrix(targetCor, ncol = K)
    targetCor[lower.tri(targetCor)] = t(targetCor)[lower.tri(t(targetCor))]
    diag(targetCor) = 1
    if(min(eigen(targetCor)$values) >= 0) break # Stop once the correlation
    # matrix is semi-positive definite. This loop could run for
    # a long time if we do not bound the uniform by 0.3.
}
```
result = SimJoint::SJpearson(
    X = marginals, cor = targetCor, stochasticStepDomain = c(0, 1),
    errorType = "meanSquare", seed = 456, maxCore = 1, convergenceTail = 8)

# # Code blocks are commented due to execution time constraint by CRAN check.
# system.time({postOptResult = SimJoint::postSimOpt(
#     X = result$X, cor = targetCor, convergenceTail = 10000)})
# # user  system elapsed
# # 6.66 0.00  6.66
#
# system.time({directOptResult = SimJoint::postSimOpt(
#     X = marginals, cor = targetCor, convergenceTail = 10000)})
# # user  system elapsed
# # 8.48 0.00  8.48
# # sum((result$cor - targetCor) ^ 2)
# # [1] 0.02209447
# # sum((resultOpt$cor - targetCor) ^ 2)
# # [1] 0.0008321346
# # sum((directOptResult$cor - targetCor) ^ 2)
# # [1] 0.02400257

SJpearson

Simulate joint given marginals and Pearson correlations.

Description

Reorder elements in each column of a matrix such that the column-wise Pearson correlations approximate a given correlation matrix.

Usage

SJpearson(
    X,
    cor,
    stochasticStepDomain = as.numeric(c(0, 1)),
    errorType = "meanSquare",
    seed = 123L,
    maxCore = 7L,
    convergenceTail = 8L,
    iterLimit = 100000L,
    verbose = TRUE
)
Arguments

\(X\) An \(N \times K\) numeric matrix of \(K\) marginal distributions (samples). Columns are sorted.

\(\text{cor}\) A \(K \times K\) correlation matrix. The matrix should be positive semi-definite.

\(\text{stochasticStepDomain}\) A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default \([0, 1]\). See the package vignette for more details.

\(\text{errorType}\) Cost function for convergence test.

"meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.

"maxRela": maximal absolute relative error.

"meanSquare": mean squared error. Default.

\(\text{seed}\) An integer or an integer vector of size 4. A single integer seeds a \(\text{pcg64}\) generator the usual way. An integer vector of size 4 supplies all the bits for a \(\text{pcg64}\) object. Default 123.

\(\text{maxCore}\) An integer. Maximal threads to invoke. Default 7. Better be no greater than the total number of virtual cores on machine.

\(\text{convergenceTail}\) An integer. If the last \(\text{convergenceTail}\) iterations resulted in equal cost function values, return. Default 8.

\(\text{iterLimit}\) An integer. The maximal number of iterations. Default 100000.

\(\text{verbose}\) A boolean value. \(\text{TRUE}\) prints progress.

Details

Algorithms are detailed in the package vignette.

Value

A list of size 2.

\(X\) A numeric matrix of size \(N \times K\), the simulated joint distribution.

\(\text{cor}\) Pearson correlation matrix of \(X\).

Examples

# Commented code blocks either require external source, or would exceed # execution time constraint for CRAN check.

# Benchmark against R package \`SimMultiCorrData\`. Use the same example
# from <https://cran.r-project.org/web/packages/SimMultiCorrData/
# vignettes/workflow.html>.
# set.seed(123)
N = 10000L # Sample size.
K = 10L # 10 marginals.
# Sample from 3 PDFs, 2 nonparametric PMFs, 5 parametric PMFs:
marginals = cbind(
  rnorm(N), rchisq(N, 4), rbeta(N, 4, 2),
  SimJoint::LHSpmf(data.frame(val = 1:3, P = c(0.3, 0.45, 0.25)), N,
    seed = sample(1e6L, 1)),
  SimJoint::LHSpmf(data.frame(val = 1:4, P = c(0.2, 0.3, 0.4, 0.1)), N,
    seed = sample(1e6L, 1)),
  rpois(N, 1), rpois(N, 5), rpois(N, 10),
  rnbinom(N, 3, 0.2), rnbinom(N, 6, 0.8))
# The seeding for 'LHSpmf()' is unhealthy, but OK for small examples.
marginals = apply(marginals, 2, function(x) sort(x))

# Create the example target correlation matrix 'Rey':
set.seed(11)
Rey <- diag(1, nrow = K)
for (i in 1:nrow(Rey)) {
  for (j in 1:ncol(Rey)) {
    if (i > j) Rey[i, j] <- runif(1, 0.2, 0.7)
    Rey[j, i] <- Rey[i, j]
  }
}

system.time({result = SimJoint::SJpearson(
  X = marginals, cor = Rey, errorType = "meanSquare", seed = 456,
  maxCore = 1, convergenceTail = 8, verbose = FALSE))
# user  system elapsed
# 0.30 0.00  0.29
# One the same platform, single-threaded speed (Intel i7-4770 CPU
# @ 3.40GHz, 32GB RAM, Windows 10, g++ 4.9.3 -Ofast, R 3.5.2) is more
# than 50 times faster than `SimMultiCorrData::rcorrvar()':
# user  system elapsed
# 16.05 0.34 16.42

# Check error statistics.
summary(as.numeric(round(cor(result$X) - Rey, 6)))
# Min. 1st Qu. Median  Mean 3rd Qu.  Max.
# -0.000365 -0.000133 -0.000028 -0.000047 0.000067 0.000301

# Post simulation optimization further reduce the errors:
resultOpt = SimJoint::postSimOpt(
  X = result$X, cor = Rey, convergenceTail = 10000)
summary(as.numeric(round(cor(resultOpt$X) - Rey, 6)))
# Min. 1st Qu. Median  Mean 3rd Qu.  Max.
# -7.10e-05 -3.10e-05 -1.15e-05 -6.48e-06 9.00e-06 7.10e-05

# Max error magnitude is less than 1% of that from
# `SimMultiCorrData:::rcorrvar()`
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -0.008336 -0.001321 0 -0.000329 0.001212 0.00339

# This table is reported in Step 4, correlation methods 1 or 2.

# Use the above example and benchmark against John Ruscio & Walter
# Kaczetow (2008) iteration. The R code released with their paper was
# erroneous. A corrected version is given by Github user "nicebread":
# <https://gist.github.com/nicebread/4045717>, but his correction was
# incomprehensive and can only handle 2-dimensional instances. Please change
# Line 32 to `Target.Corr <- rho` and source the file.

# Test Ruscio-Kaczetow's code.
set.seed(123)
RuscioKaczetow = GenData(Pop = as.data.frame(marginals),
Rey, N = 1000) # By default, the function takes 1000
# samples from each marginal population of size 10000.
summary(round(as.numeric(cor(RuscioKaczetow) - Rey), 6))
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -0.183274 -0.047461 -0.015737 -0.008008 0.027475 0.236662

result = SimJoint::SJpearson(
   X = apply(marginals, 2, function(x) sort(sample(x, 1000, replace = TRUE))),
cor = Rey, errorType = "maxRela", maxCore = 2) # CRAN does not allow more
# than 2 threads for running examples.
summary(round(as.numeric(cor(result$X) - Rey), 6))
# Min. 1st Qu. Median Mean 3rd Qu. Max.
# -0.0055640 -0.0014850 -0.0004810 -0.0007872 0.0000000 0.0025920
resultOpt = SimJoint::postSimOpt(
   X = result$X, cor = Rey, convergenceTail = 10000)
summary(as.numeric(round(cor(resultOpt$X) - Rey, 6)))
# Min. 1st Qu. Median Mean 3rd Qu. Max.

# Benchmark against R package `GenOrd`
# <https://cran.r-project.org/web/packages/GenOrd/index.html> using the
# example above Statistics cannot be collected because it has been running
# for more than 10 hours.
# Library `GenOrd` should have been installed and attached.
# system.time({resultGenOrd = ordsample(N, marginal = lapply(1L : K, function(x) (1 : (N - 1)) / N), Rey, support = as.data.frame(marginals)))}}

# Benchmark against R package `EnvStats` using its manual example on Page 1156 of <https://cran.r-project.org/web/packages/EnvStats/EnvStats.pdf>. The function `simulateVector()` imposes rank correlations.

# Library `EnvStats` should have been installed and attached.

cor.mat = matrix(c(1, 0.8, 0, 0.5, 0.8, 1, 0, 0.7, 0, 0, 1, 0.2, 0.5, 0.7, 0.2, 1), 4, 4)
pareto.rns <- simulateVector(100, "pareto", list(location = 10, shape = 2), sample.method = "LHS", seed = 56)
mat <- simulateMvMatrix(1000, distributions = c(Normal = "norm", Lognormal = "lnormAlt", Beta = "beta", Empirical = "emp"),
    param.list = list(Normal = list(mean=10, sd=2),
        Lognormal = list(mean=10, cv=1),
        Beta = list(shape1 = 2, shape2 = 3),
        Empirical = list(obs = pareto.rns)),
cor.mat = cor.mat, seed = 47, sample.method = "LHS")

round(cor(mat, method = "spearman"), 2)
# Normal Lognormal Beta Empirical
# Normal 1.00 0.78 -0.01 0.47
# Lognormal 0.78 1.00 -0.01 0.67
# Beta -0.01 -0.01 1.00 0.19
# Empirical 0.47 0.67 0.19 1.00

# Imposing rank correlations is equivalent to imposing Pearson correlations on ranks.

set.seed(123)
marginals = cbind(sort(rnorm(1000, 10, 2)),
    sort(rlnormAlt(1000, 10, 1)),
    sort(rbeta(1000, 2, 3)),
    sort(sample(pareto.rns, 1000, replace = TRUE)))
marginalsRanks = cbind(1:1000, 1:1000, 1:1000, 1:1000)
# Simulate the joint for ranks:
tmpResult = SimJoint::SJpearson(X = marginalsRanks, cor = cor.mat, errorType = "meanSquare", seed = 456, maxCore = 2, convergenceTail = 8, verbose = TRUE)$X
# Reorder "marginals" by ranks.
result = matrix(mapply(function(x, y) y[as.integer(x)],
    as.data.frame(tmpResult),
    as.data.frame(marginals), SIMPLIFY = TRUE), ncol = 4)
round(cor(result, method = "spearman"), 2)
# 1.0 0.8 0.0 0.5
# Play random numbers.
set.seed(123)
N = 2000L
K = 20L
# The following essentially creates a mixture distribution.
marginals = c(runif(10000L, -2, 2), rgamma(10000L, 2, 2), rnorm(20000L))
marginals = matrix(sample(marginals, length(marginals)), ncol = K)
# This operation made the columns comprise samples from the same
# mixture distribution.
marginals = apply(marginals, 2, function(x) sort(x))

# May take a while to generate valid correlation matrix.
while(TRUE)
{
    targetCor = matrix(runif(K * K, -0.1, 0.4), ncol = K)
    targetCor[lower.tri(targetCor)] = t(targetCor)[lower.tri(t(targetCor))]
    diag(targetCor) = 1
    if(all(eigen(targetCor)$values >= 0)) break
}

result = SimJoint::SJpearson(
    X = marginals, cor = targetCor, errorType = "meanSquare", seed = 456,
    maxCore = 2, convergenceTail = 8, verbose = TRUE)
resultOpt = SimJoint::postSimOpt(
    X = result$X, cor = targetCor, convergenceTail = 10000)

# Visualize errors and correlation matrices.
par(mfrow = c(2, 2))
hist(resultOpt$cor - targetCor, breaks = K * K, main = NULL,
     xlab = "Error")
hist(resultOpt$cor / targetCor - 1, breaks = K * K, main = NULL,
     xlab = "Relative error")
zlim = range(range(targetCor[targetCor < 1]),
             range(resultOpt$cor[resultOpt$cor < 1]))
col = colorRampPalette(c("blue", "red", "yellow"))(K * K)
tmp = targetCor[, K : 1L]
image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
      main = "Target cor", col = col)
tmp = resultOpt$cor[, K : 1L]
image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
      main = "Cor reached", col = col)
par(mfrow = c(1, 1))

# An example where the functional relationships between marginals are highly
# nonlinear and the target correlations are hard to impose. Other packages
# would fail or report theoretical infeasibility.
# set.seed(123)
N = 10000L
K = 10L

# Several 2-parameter PDFs in R:
marginals = list(rbeta, rcauchy, rf, rgamma, rnorm, runif, rweibull)
Npdf = length(marginals)

if(Npdf >= K) chosenMarginals =
marginals[sample(Npdf, K, replace = TRUE)] else chosenMarginals =
marginals[c(1L : Npdf, sample(Npdf, K - Npdf, replace = TRUE)])

# Sample from the marginal PDFs.
marginals = as.matrix(as.data.frame(lapply(chosenMarginals, function(f)
{
    para = sort(runif(2, 0.1, 10))
    rst = f(N, para[1], para[2])
    sort(rst)
}))
dimnames(marginals) = NULL

frechetUpperCor = cor(marginals) # The correlation matrix should be
# upper-bounded by that of the perfectly rank-correlated
# joint (Frechet upper bound). The lower bound is characterized by
# d-countercomonotonicity and depends not only on marginals.
cat("Range of maximal correlations between marginals: ",
    range(frechetUpperCor[frechetUpperCor < 1]))
# Two perfectly rank-correlated marginals can have a Pearson
# correlation below 0.07. This is due to high nonlinearities
# in marginal PDFs.

# Create a valid correlation matrix upper-bounded by 'frechetUpperCor'.
while(TRUE)
{
    targetCor = sapply(frechetUpperCor, function(x)
    runif(1, -0.1, min(0.3, x * 0.8)))
    targetCor = matrix(targetCor, ncol = K)
    targetCor[lower.tri(targetCor)] = t(targetCor)[lower.tri(t(targetCor))]
```r
diag(targetCor) = 1
if(min(eigen(targetCor)$values) >= 0) break # Stop once the correlation
# matrix is semi-positive definite. This loop could run for
# a long time if we do not bound the uniform by 0.3.
}

result = SimJoint::SJpearson(
  X = marginals, cor = targetCor, stochasticStepDomain = c(0, 1),
  errorType = "meanSquare", seed = 456, maxCore = 2, convergenceTail = 8)
  # resultOpt = SimJoint::postSimOpt( # Could take many seconds.
  #  X = result$X, cor = targetCor, convergenceTail = 10000)

  # # Visualize errors and correlation matrices.
  # par(mfrow = c(2, 2))
  # hist(resultOpt$cor - targetCor, breaks = K * K, main = NULL,
  #  xlab = "Error")
  # hist(resultOpt$cor / targetCor - 1, breaks = K * K, main = NULL,
  #  xlab = "Relative error")
  # zlim = range(range(targetCor[targetCor < 1]),
  #  range(resultOpt$cor[resultOpt$cor < 1]))
  # col = colorRampPalette(c("blue", "red", "yellow"))(K * K)
  # tmp = targetCor[, K : 1L]
  # image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
  #  main = "Target cor", col = col)
  # tmp = resultOpt$cor[, K : 1L]
  # image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
  #  main = "Cor reached", col = col)
  # par(mfrow = c(1, 1))

# Different `errorType` could make a difference.
result = SimJoint::SJpearson(
  X = marginals, cor = targetCor, stochasticStepDomain = c(0, 1),
  errorType = "maxRela", seed = 456, maxCore = 2, convergenceTail = 8)
  # resultOpt = SimJoint::postSimOpt(
  #  X = result$X, cor = targetCor, convergenceTail = 10000)

  # # Visualize errors and correlation matrices.
  # par(mfrow = c(2, 2))
  # hist(resultOpt$cor - targetCor, breaks = K * K, main = NULL,
  #  xlab = "Error")
  # hist(resultOpt$cor / targetCor - 1, breaks = K * K, main = NULL,
  #  xlab = "Relative error")
  # zlim = range(range(targetCor[targetCor < 1]),
  #  range(resultOpt$cor[resultOpt$cor < 1]))
  # col = colorRampPalette(c("blue", "red", "yellow"))(K * K)
  # tmp = targetCor[, K : 1L]
  # image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
  #  main = "Target cor", col = col)
  # tmp = resultOpt$cor[, K : 1L]
```

---

14

SJpearson
SJpearsonPMF

Simulate joint with marginal PMFs and Pearson correlations.

**Description**

Sample from marginal probability mass functions via Latin hypercube sampling and then simulate the joint distribution with Pearson correlations.

**Usage**

```r
SJpearsonPMF(
  PMFs,
  sampleSize,
  cor,
  stochasticStepDomain = as.numeric(c(0, 1)),
  errorType = "meanSquare",
  seed = 123L,
  maxCore = 7L,
  convergenceTail = 8L,
  iterLimit = 100000L,
  verbose = TRUE
)
```

**Arguments**

- **PMFs**: A list of data frames. Each data frame has 2 columns, a value vector and a probability vector. Probabilities should sum up to 1. Let the size of PMFs be $K$.
- **sampleSize**: An integer. The sample size $N$.
- **cor**: A $K \times K$ correlation matrix. The matrix should be positive semi-definite.
- **stochasticStepDomain**: A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default [0, 1]. See the package vignette for more details.
- **errorType**: Cost function for convergence test.
  - "meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.
  - "maxRela": maximal absolute relative error.
  - "meanSquare": mean squared error. Default.
- **seed**: An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object.
maxCore  An integer. Maximal threads to invoke. Default 7. Better be no greater than the
total number of virtual cores on machine.

convergenceTail  An integer. If the last convergenceTail iterations resulted in equal cost func-
tion values, return. Default 8.

iterLimit  An integer. The maximal number of iterations. Default 100000.

verbose  A boolean value. TRUE prints progress.

Details

Algorithms are detailed in the package vignette.

Value

A list of size 2.

X  A numeric matrix of size \(N \times K\), the simulated joint distribution.

cor  Pearson correlation matrix of X.

Examples

```r
# Use the same example from <https://cran.r-project.org/web/packages/
# SimMultiCorrData/vignettes/workflow.html>.

set.seed(123)
N = 10000L # Sample size.
K = 10L # 10 marginals.
# 3 PDFs, 2 nonparametric PMFs, 5 parametric PMFs:
PMFs = c(
  apply(cbind(rnorm(N), rchisq(N, 4), rbeta(N, 4, 2)), 2, function(x)
    data.frame(val = sort(x), P = 1.0 / N)),
  list(data.frame(val = 1:3 + 0.0, P = c(0.3, 0.45, 0.25))),
  list(data.frame(val = 1:4 + 0.0, P = c(0.2, 0.3, 0.4, 0.1))),
  apply(cbind(rpois(N, 1), rpois(N, 5), rpois(N, 10),
            rnbinom(N, 3, 0.2), rnbinom(N, 6, 0.8)), 2, function(x)
    data.frame(val = as.numeric(sort(x)), P = 1.0 / N))
)

# Create the target correlation matrix 'Rey':
set.seed(11)
Rey <- diag(1, nrow = 10)
for (i in 1:nrow(Rey)) {
  for (j in 1:ncol(Rey)) {
    if (i > j) Rey[i, j] <- runif(1, 0.2, 0.7)
    Rey[j, i] <- Rey[i, j]
  }
}
```

system.time({result = SimJoint::SJpearsonPMF(
  PMFs = PMFs, sampleSize = N, cor = Rey, errorType = "meanSquare",
  seed = 456, maxCore = 2, convergenceTail = 8, verbose = TRUE))})

# Check relative errors.
summary(as.numeric(abs(result$cor / Rey - 1)))

# Play with random nonparametric PMFs.

set.seed(123)
N = 2000L
K = 20L

# Create totally random nonparametric PMFs:
PMFs = lapply(1L : K, function(x)
{
  p = runif(2, 1, 10)
  result = data.frame(
    val = sort(rnorm(200)), P = runif(200))
  result$P = result$P / sum(result$P)
  result
})

# Create a valid correlation matrix upper-bounded by `frechetUpperCor`.
while(TRUE)
{
  targetCor = matrix(runif(K * K, -0.1, 0.3), ncol = K)
  targetCor[lower.tri(targetCor)] = t(targetCor)[lower.tri(t(targetCor))]
  diag(targetCor) = 1
  if(min(eigen(targetCor)$values) >= 0) break # Break once the correlation
  # matrix is semi-positive definite. This loop could be running for quite
  # a long time if we do not bound `runif()`.
}

result = SimJoint::SJpearsonPMF(
  PMFs = PMFs, sampleSize = N, cor = targetCor, stochasticStepDomain = c(0, 1),
  errorType = "meanSquare", seed = 456, maxCore = 2, convergenceTail = 8)

# Visualize errors and correlation matrices.
par(mfrow = c(2, 2))
hist(result$cor - targetCor, breaks = K * K, main = NULL,
     xlab = "Error", cex.lab = 1.5, cex.axis = 1.25)
hist(result$cor / targetCor - 1, breaks = K * K, main = NULL,
     xlab = "Relative error", ylab = "", cex.lab = 1.5, cex.axis = 1.25)
```r
zlim = range(range(targetCor[targetCor < 1]), range(result$cor[result$cor < 1]))
col = colorRampPalette(c("blue", "red", "yellow"))(K * K)
tmp = targetCor[, K : 1L]
image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
      main = "Target cor", col = col)
tmp = result$cor[, K : 1L]
image(tmp, xaxt = "n", yaxt = "n", zlim = zlim, bty = "n",
      main = "Cor reached", col = col)
par(mfrow = c(1, 1))
```

---

**SJspearman**

*Simulate joint given marginals and Spearman correlations.*

**Description**

Reorder elements in each column of a matrix such that the column-wise Spearman correlations approximate a given correlation matrix.

**Usage**

```r
SJspearman(
X,
cor,
stochasticStepDomain = as.numeric(c(0, 1)),
errorType = "meanSquare",
seed = 123L,
maxCore = 7L,
convergenceTail = 8L,
iterLimit = 100000L,
verbose = TRUE
)
```

**Arguments**

- **X**: An \( N \times K \) numeric matrix of \( K \) marginal distributions (samples). Columns are sorted.
- **cor**: A \( K \times K \) correlation matrix. The matrix should be positive semi-definite.
- **stochasticStepDomain**: A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default \([0, 1]\). See the package vignette for more details.
- **errorType**: Cost function for convergence test.
  - "meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.
  - "maxRela": maximal absolute relative error.
  - "meanSquare": mean squared error. Default.
seed  An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object.

maxCore  An integer. Maximal threads to invoke. Default 7. Better be no greater than the total number of virtual cores on machine.

convergenceTail  An integer. If the last convergenceTail iterations resulted in equal cost function values, return. Default 8.

iterLimit  An integer. The maximal number of iterations. Default 100000.

verbose  A boolean value. TRUE prints progress.

Details
Algorithms are detailed in the package vignette.

Value
A list of size 2.

X  A numeric matrix of size N x K, the simulated joint distribution.

cor  Spearman correlation matrix of X.

Examples

# Use the same example from <https://cran.r-project.org/web/packages/SimMultiCorrData/vignettes/workflow.html>.
# ===============================================================================
# = Use the same example from <https://cran.r-project.org/web/packages/SimMultiCorrData/vignettes/workflow.html>.
# ===============================================================================
set.seed(123)
N = 10000L # Sample size.
K = 10L # 10 marginals.
# Sample from 3 PDFs, 2 nonparametric PMFs, 5 parametric PMFs:
marginals = cbind(
  rnorm(N), rchisq(N, 4), rbeta(N, 4, 2),
  LHSpmf(data.frame(val = 1:3, P = c(0.3, 0.45, 0.25)), N, seed = sample(1e6L, 1)),
  LHSpmf(data.frame(val = 1:4, P = c(0.2, 0.3, 0.4, 0.1)), N, seed = sample(1e6L, 1)),
  rpois(N, 1), rpois(N, 5), rpois(N, 10),
  rnbinom(N, 3, 0.2), rnbinom(N, 6, 0.8))
# The seeding for `LHSpmf()` is unhealthy, but OK for small examples.
marginals = apply(marginals, 2, function(x) sort(x))

# Create the target correlation matrix `Rey` treated as Spearman
data sample correlation.
set.seed(11)
Rey <- diag(1, nrow = 10)
for (i in 1:nrow(Rey)) {

for (j in 1:ncol(Rey)) {
  if (i > j) Rey[i, j] <- runif(1, 0.2, 0.7)
  Rey[j, i] <- Rey[i, j]
}

result = SimJoint::SJspearman(
  X = marginals, cor = Rey, errorType = "meanSquare", seed = 456,
  maxCore = 1, convergenceTail = 8, verbose = TRUE)

# Check relative errors.
summary(as.numeric(abs(cor(result$X, method = "spearman") / Rey - 1)))

# Another way to impose rank correlation is to supply rank matrix
# to SJpearson():
system.time({
  reorderedRanks = SimJoint::SJpearson(
    X = apply(marginals, 2, function(x) rank(x)), cor = Rey,
    errorType = "meanSquare", seed = 456, maxCore = 1,
    convergenceTail = 8, verbose = TRUE))

# Reordering according to ranks:
result = apply(rbind(reorderedRanks$X, marginals), 2, function(x)
  {
    x[(N + 1L) : (2L * N)][as.integer(x[1L : N])]
  })

# Check the relative errors.
summary(as.numeric(abs(cor(result, method = "spearman") / Rey - 1)))

SJspearmanPMF

Simulate joint with marginal PMFs and Spearman correlations.

Description
Sample from marginal probability mass functions via Latin hypercube sampling and then simulate
the joint distribution with Spearman correlations.

Usage
SJspearmanPMF(
  PMFs,
  sampleSize,
  cor,
stochasticStepDomain = as.numeric(c(0, 1)),
errorType = "meanSquare",
seed = 123L,
maxCore = 7L,
convergenceTail = 8L,
iterLimit = 100000L,
verbose = TRUE
)

Arguments

PMFs A list of data frames. Each data frame has 2 columns, a value vector and a probability vector. Probabilities should sum up to 1. Let the size of PMFs be K.
sampleSize An integer. The sample size N.
cor A K x K correlation matrix. The matrix should be positive semi-definite.
stochasticStepDomain A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default [0, 1]. See the package vignette for more details.
errorType Cost function for convergence test.
"meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.
"maxRela": maximal absolute relative error.
"meanSquare": mean squared error. Default.
seed An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object.
maxCore An integer. Maximal threads to invoke. Default 7. Better be no greater than the total number of virtual cores on machine.
convergenceTail An integer. If the last convergenceTail iterations resulted in equal cost function values, return. Default 8.
iterLimit An integer. The maximal number of iterations. Default 100000.
verbose A boolean value. TRUE prints progress.

Details

Algorithms are detailed in the package vignette.

Value

A list of size 2.

X A numeric matrix of size N x K, the simulated joint distribution.
cor Spearman correlation matrix of X.
# Play with completely random nonparametric PMFs.

```r
set.seed(123)
N = 2000L
K = 20L

# Create totally random nonparametric PMFs:
PMFs = lapply(1L : K, function(x)
{
  p = runif(2, 1, 10)
  result = data.frame(val = sort(rnorm(200)), P = runif(200))
  result$P = result$P / sum(result$P)
  result
})

# Create a valid correlation matrix upper-bounded by `frechetUpperCor`
while(TRUE)
{
  targetCor = matrix(runif(K * K, -0.1, 0.3), ncol = K)
  targetCor[lower.tri(targetCor)] = t(targetCor)[lower.tri(t(targetCor))]
  diag(targetCor) = 1
  if(min(eigen(targetCor)$values) >= 0) break # Break once the correlation
  # matrix is semi-positive definite. This loop could be running for quite
  # a long time if we do not bound `runif()`.
}
```

result = SimJoint::SJspearmanPMF(
  PMFs = PMFs, sampleSize = N, cor = targetCor, stochasticStepDomain = c(0, 1),
  errorType = "meanSquare", seed = 456, maxCore = 1, convergenceTail = 8)

# Visualize errors and correlation matrices.
par(mfrow = c(2, 2))
hist(result$cor - targetCor, breaks = K * K, main = NULL,
  xlab = "Error", cex.lab = 1.5, cex.axis = 1.25)
par(mfrow = c(1, 1))
**xSJpearson**
Simulate joint given marginals, Pearson correlations and uncorrelated support matrix.

**Description**

Users specify the uncorrelated random source instead of using a permutated \( X \) to left-multiply the correlation matrix decomposition. See the package vignette for more details.

**Usage**

```r
xSJpearson(
  X,
  cor,
  noise,
  stochasticStepDomain = as.numeric(c(0, 1)),
  errorType = "meanSquare",
  seed = 123L,
  maxCore = 7L,
  convergenceTail = 8L,
  iterLimit = 100000L,
  verbose = TRUE
)
```

**Arguments**

- **X**
  An \( N \times K \) numeric matrix of \( K \) marginal distributions (samples). Columns are sorted.

- **cor**
  A \( K \times K \) correlation matrix. The matrix should be positive semi-definite.

- **noise**
  An \( N \times K \) arbitrary numeric matrix where columns are (more or less) uncorrelated. Exact zero correlations are unnecessary.

- **stochasticStepDomain**
  A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default \([0, 1]\). See the package vignette for more details.

- **errorType**
  Cost function for convergence test.
  "meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.
  "maxRela": maximal absolute relative error.
  "meanSquare": mean squared error. Default.

- **seed**
  An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object.

- **maxCore**
  An integer. Maximal threads to invoke. Default 7. Better be no greater than the total number of virtual cores on machine.
convergenceTail
An integer. If the last convergenceTail iterations resulted in equal cost function values, return. Default 8.

iterLimit
An integer. The maximal number of iterations. Default 100000.

verbose
A boolean value. TRUE prints progress.

Details
Algorithms are detailed in the package vignette.

Value
A list of size 2.

X A numeric matrix of size N x K, the simulated joint distribution.

cor Pearson correlation matrix of X.

Examples

```r
# Use the same example from <https://cran.r-project.org/web/packages/
# SimMultiCorrData/vignettes/workflow.html>.

set.seed(123)
N = 10000L # Sample size.
K = 10L # 10 marginals.
# Sample from 3 PDFs, 2 nonparametric PMFs, 5 parametric PMFs:
marginals = cbind(
  rnorm(N), rchisq(N, 4), rbeta(N, 4, 2),
  SimJoint::LHSpmf(data.frame(val = 1:3, P = c(0.3, 0.45, 0.25)), N,
    seed = sample(1e6L, 1)),
  SimJoint::LHSpmf(data.frame(val = 1:4, P = c(0.2, 0.3, 0.4, 0.1)), N,
    seed = sample(1e6L, 1)),
  rpois(N, 1), rpois(N, 5), rpois(N, 10),
  rnbinom(N, 3, 0.2), rnbinom(N, 6, 0.8))
# The seeding for SimJoint::LHSpmf() is unhealthy, but OK for small examples.
marginals = apply(marginals, 2, function(x) sort(x))

# Create the target correlation matrix 'Rey':
set.seed(11)
Rey <- diag(1, nrow = K)
for (i in 1:nrow(Rey)) {
  for (j in 1:ncol(Rey)) {
    if (i > j) Rey[i, j] <- runif(1, 0.2, 0.7)
    Rey[j, i] <- Rey[i, j]
  }
}
```

system.time({
  result = SimJoint::xSJpearson(
    X = marginals, cor = Rey, noise = matrix(runif(N * K), ncol = K),
    errorType = "meanSquare", seed = 456, maxCore = 1,
    convergenceTail = 8, verbose = TRUE))
})

summary(as.numeric(round(cor(result$X) - Rey, 6)))

---

**xSJpearsonPMF**  
Simulate joint with marginal PMFs, Pearson correlations and uncorrelated support matrix.

**Description**

Sample from marginal probability mass functions via Latin hypercube sampling and then simulate the joint distribution with Pearson correlations. Users specify the uncorrelated random source instead of using permuted marginal samples to left-multiply the correlation matrix decomposition.

**Usage**

```r
xSJpearsonPMF(
  PMFs,
  sampleSize,
  cor,
  noise,
  stochasticStepDomain = as.numeric(c(0, 1)),
  errorType = "meanSquare",
  seed = 123L,
  maxCore = 7L,
  convergenceTail = 8L,
  iterLimit = 100000L,
  verbose = TRUE
)
```

**Arguments**

- **PMFs**  
  A list of data frames. Each data frame has 2 columns, a value vector and a probability vector. Probabilities should sum up to 1. Let the size of PMFs be K.

- **sampleSize**  
  An integer. The sample size N.

- **cor**  
  A K x K positive semi-definite correlation matrix.

- **noise**  
  An N x K arbitrary numeric matrix where columns are (more or less) uncorrelated. Exact zero correlations are unnecessary.

- **stochasticStepDomain**  
  A numeric vector of size 2. Range of the stochastic step ratio for correcting the correlation matrix in each iteration. Default [0, 1]. See the package vignette for more details.
errorType  Cost function for convergence test.
"meanRela": average absolute relative error between elements of the target correlation matrix and the correlation matrix approximated in each iteration.
"maxRela": maximal absolute relative error. "meanSquare": mean squared error. Default.

seed  An integer or an integer vector of size 4. A single integer seeds a pcg64 generator the usual way. An integer vector of size 4 supplies all the bits for a pcg64 object.

maxCore  An integer. Maximal threads to invoke. Default 7. Better be no greater than the total number of virtual cores on machine.

convergenceTail  An integer. If the last convergenceTail iterations resulted in equal cost function values, return. Default 8.

iterLimit  An integer. The maximal number of iterations. Default 100000.

verbose  A boolean value. TRUE prints progress.

Details

Algorithms are detailed in the package vignette.

Value

A list of size 2.

X  A numeric matrix of size \( N \times K \), the simulated joint distribution.

cor  Pearson correlation matrix of X.

Examples

```r
# Use the same example from <https://cran.r-project.org/web/packages/SimMultiCorrData/vignettes/workflow.html>.
set.seed(123)
N = 10000L # Sample size.
K = 10L # 10 marginals.
# 3 PDFs, 2 nonparametric PMFs, 5 parametric PMFs:
PMFs = c(
  apply(cbind(rnorm(N), rchisq(N, 4), rbeta(N, 4, 2)), 2, function(x)
    data.frame(val = sort(x), P = 1.0 / N)),
  list(data.frame(val = 1:3 + 0.0, P = c(0.3, 0.45, 0.25))),
  list(data.frame(val = 1:4 + 0.0, P = c(0.2, 0.3, 0.4, 0.1))),
  apply(cbind(rpois(N, 1), rpois(N, 5), rpois(N, 10),
          rnbinom(N, 3, 0.2), rnbinom(N, 6, 0.8)), 2, function(x)
    data.frame(val = as.numeric(sort(x)), P = 1.0 / N))
)

# Create the target correlation matrix 'Rey':
```
set.seed(11)
Rey <- diag(1, nrow = 10)
for (i in 1:nrow(Rey)) {
  for (j in 1:ncol(Rey)) {
    if (i > j) Rey[i, j] <- runif(1, 0.2, 0.7)
    Rey[j, i] <- Rey[i, j]
  }
}

system.time({result = SimJoint::xSJpearsonPMF(
  PMFs = PMFs, sampleSize = N, noise = matrix(runif(N * K), ncol = K),
  cor = Rey, errorType = "meanSquare", seed = 456, maxCore = 1,
  convergenceTail = 8, verbose = TRUE))

# Check relative errors.
summary(as.numeric(abs(result$cor / Rey - 1)))
Index

decor, 2
exportRandomState, 3
LHSpmf, 4
postSimOpt, 5
SJpearson, 7
SJpearsonPMF, 15
SJspearman, 18
SJspearmanPMF, 20
xSJpearson, 23
xSJpearsonPMF, 25