Package 'humanleague'

April 17, 2023

Type Package

Title Synthetic Population Generator

Version 2.2.0

Description Generates high-entropy integer synthetic populations from marginal and (optionally) seed data using quasirandom sampling,
in arbitrary dimensionality (Smith, Lovelace and Birkin (2017) <doi:10.18564/jasss.3550>).
The package also provides an implementation of the Iterative Proportional Fitting (IPF) algorithm (Zaloznik (2011) <doi:10.13140/2.1.2480.9923>).

License MIT + file LICENCE

Encoding UTF-8

Imports Rcpp (>= 0.12.8)

LinkingTo Rcpp

RoxygenNote 7.0.2

Suggests testthat

NeedsCompilation yes

Author Andrew Smith [aut, cre],
Steven Johnson [ctb] (Sobol sequence generator implementation),
Massachusetts Institute of Technology [cph] (Sobol sequence generator implementation),
John Burkhardt [ctb, cph] (C++ implementation of incomplete gamma function),
G Bhattacharjee [ctb] (Original FORTRAN implementation of incomplete gamma function)

Maintainer Andrew Smith <andrew@friarswood.net>

Repository CRAN

Date/Publication 2023-04-17 19:10:02 UTC

R topics documented:

`flatten` *Convert multidimensional array of counts per state into table form. Each row in the table corresponds to one individual*

Description

This function

Usage

```
flatten(stateOccupancies, categoryNames)
```

Arguments

`stateOccupancies`	an arbitrary-dimension array of (integer) state occupation counts.
`categoryNames`	a string vector of unique column names.

Value

a DataFrame with columns corresponding to category values and rows corresponding to individuals.

Examples

```
gender=c(51,49)
age=c(17,27,35,21)
states=qis(list(1,2),list(gender,age))$result
table=flatten(states,c("Gender","Age"))
print(nrow(table[table$Gender==1,])) # 51
print(nrow(table[table$Age==2,])) # 27
```

`humanleague` *humanleague*

Description

R package for synthesising populations from aggregate and (optionally) seed data

Details

See README.md for detailed information and examples.

Overview

The package contains algorithms that use a number of different microsynthesis techniques:

- Iterative Proportional Fitting (IPF), *a la* **mipfp** package
- Quasirandom Integer Sampling (QIS) (no seed population) -
- Quasirandom Integer Sampling of IPF (QISI): A combination of the two techniques whereby IPF solutions are used to sample an integer population.

The latter provides a bridge between deterministic reweighting and combinatorial optimisation, offering advantages of both techniques:

- generates high-entropy integral populations
- can be used to generate multiple populations for sensitivity analysis
- is less sensitive than IPF to convergence issues when there are a high number of empty cells present in the seed
- relatively fast computation time, though running time is linear in population

The algorithms:

- support arbitrary dimensionality* for both the marginals and the seed.
- produce statistical data to ascertain the likelihood/degeneracy of the population (where appropriate).

[* excluding the legacy functions retained for backward compatibility with version 1.0.1]

The package also contains the following utility functions:

- a Sobol sequence generator -
- functionality to convert fractional to nearest-integer marginals (in 1D). This can also be achieved in multiple dimensions by using the QISI algorithm.
- functionality to 'flatten' a population into a table: this converts a multidimensional array containing the population count for each state into a table listing individuals and their characteristics.

Functions

`integerise` *Generate integer population from a fractional one where the 1-d partial sums along each axis have an integral total*

Description

This function will generate the closest integer array to the fractional population provided, preserving the sums in every dimension.

Usage

```
integerise(population)
```

Arguments

`population`	a numeric vector of state occupation probabilities. Must sum to unity (to within double precision epsilon)

Value

an integer vector of frequencies that sums to pop.

Examples

```
prob2IntFreq(c(0.1,0.2,0.3,0.4), 11)
```

ipf *Multidimensional IPF*

Description

C++ multidimensional IPF implementation

Usage

```
ipf(seed, indices, marginals)
```

Arguments

seed	an n-dimensional array of seed values
indices	a List of 1-d arrays specifying the dimension indices of each marginal as they apply to the seed values
marginals	a List of arrays containing marginal data. The sum of elements in each array must be identical

Value

an object containing:

- a flag indicating if the solution converged
- the population matrix
- the total population
- the number of iterations required
- the maximum error between the generated population and the marginals

Examples

```
ageByGender = array(c(1,2,5,3,4,3,4,5,1,2), dim=c(5,2))
ethnicityByGender = array(c(4,6,5,6,4,5), dim=c(3,2))
seed = array(rep(1,30), dim=c(5,2,3))
result = ipf(seed, list(c(1,2), c(3,2)), list(ageByGender, ethnicityByGender))
```

prob2IntFreq — *Generate integer frequencies from discrete probabilities and an overall population.*

Description

This function will generate the closest integer vector to the probabilities scaled to the population.

Usage

```
prob2IntFreq(pIn, pop)
```

Arguments

pIn	a numeric vector of state occupation probabilities. Must sum to unity (to within double precision epsilon)
pop	the total population

Value

an integer vector of frequencies that sum to pop, and the RMS difference from the original values.

Examples

```
prob2IntFreq(c(0.1,0.2,0.3,0.4), 11)
```

qis — *Multidimensional QIS*

Description

C++ multidimensional Quasirandom Integer Sampling implementation

Usage

```
qis(indices, marginals, skips = 0L)
```

Arguments

indices	a List of 1-d arrays specifying the dimension indices of each marginal
marginals	a List of arrays containing marginal data. The sum of elements in each array must be identical
skips	(optional, default 0) number of Sobol points to skip before sampling

Value

an object containing:

- a flag indicating if the solution converged
- the population matrix
- the exepected state occupancy matrix
- the total population
- chi-square and p-value

Examples

```
ageByGender = array(c(1,2,5,3,4,3,4,5,1,2), dim=c(5,2))
ethnicityByGender = array(c(4,6,5,6,4,5), dim=c(3,2))
result = qis(list(c(1,2), c(3,2)), list(ageByGender, ethnicityByGender))
```

---

qisi *QIS-IPF*

Description

C++ QIS-IPF implementation

Usage

```
qisi(seed, indices, marginals, skips = 0L)
```

Arguments

seed	an n-dimensional array of seed values
indices	a List of 1-d arrays specifying the dimension indices of each marginal
marginals	a List of arrays containing marginal data. The sum of elements in each array must be identical
skips	(optional, default 0) number of Sobol points to skip before sampling

Value

an object containing:

- a flag indicating if the solution converged
- the population matrix
- the exepected state occupancy matrix
- the total population
- chi-square and p-value

Examples

```
ageByGender = array(c(1,2,5,3,4,3,4,5,1,2), dim=c(5,2))
ethnicityByGender = array(c(4,6,5,6,4,5), dim=c(3,2))
seed = array(rep(1,30), dim=c(5,2,3))
result = qisi(seed, list(c(1,2), c(3,2)), list(ageByGender, ethnicityByGender))
```

---

sobolSequence *Generate Sobol' quasirandom sequence*

Description

Generate Sobol' quasirandom sequence

Usage

```
sobolSequence(dim, n, skip = 0L)
```

Arguments

dim	dimensions
n	number of variates to sample
skip	number of variates to skip (actual number skipped will be largest power of 2 less than k)

Value

a n-by-d matrix of uniform probabilities in (0,1).

Examples

```
sobolSequence(2, 1000, 1000) # will skip 512 numbers!
```

---

unitTest *Entry point to enable running unit tests within R (e.g. in testthat)*

Description

Entry point to enable running unit tests within R (e.g. in testthat)

Usage

```
unitTest()
```

Value

a List containing, number of tests run, number of failures, and any error messages.

Examples

```
unitTest()
```

Index