ProbabilisticNetwork

Exported

a_var(p::Array{Float64, N}) Variance of a series of additive Bernoulli events

f(p): (p(1-p))

source: ProbabilisticNetwork/src/./proba_utils.jl:63

---

centrality_katz(A::Array{Float64, 2}) Measures Katz's centrality for each node in a unipartite network.

Keyword arguments

- `a` (def. 0.1), the weight of each subsequent connection
- `k` (def. 5), the maximal path length considered

source: ProbabilisticNetwork/src/./centrality.jl:10

---

connectance(A::Array{Float64, 2}) Expected connectance for a probabilistic matrix, measured as the number of expected links, divided by the size of the matrix.

source: ProbabilisticNetwork/src/./connectance.jl:15

---

connectance_var(A::Array{Float64, 2}) Expected variance of the connectance for a probabilistic matrix, measured as the variance of the number of links divided by the squared size of the matrix.

source: ProbabilisticNetwork/src/./connectance.jl:23

---

degree(A::Array{Float64, 2}) Expected degree

source: ProbabilisticNetwork/src/./degree.jl:20

---

degree_in(A::Array{Float64, 2}) Expected number of ingoing degrees

source: ProbabilisticNetwork/src/./degree.jl:14

---

degree_out(A::Array{Float64, 2}) Expected number of outgoing degrees

source: ProbabilisticNetwork/src/./degree.jl:8

---

i_esp(p::Float64) Expected value of a single Bernoulli event

Simply f(p): p

source: ProbabilisticNetwork/src/./proba_utils.jl:37

---

i_var(p::Float64) Variance of a single Bernoulli event

f(p): p(1-p)

source: ProbabilisticNetwork/src/./proba_utils.jl:50

---

links(A::Array{Float64, 2}) Expected number of links for a probabilistic matrix

source: ProbabilisticNetwork/src/./connectance.jl:2

---

links_var(A::Array{Float64, 2}) Expected variance of the number of links for a probabilistic matrix

source: ProbabilisticNetwork/src/./connectance.jl:8

---

m_var(p::Array{Float64, N}) Variance of a series of multiplicative Bernoulli events

source: ProbabilisticNetwork/src/./proba_utils.jl:72

---

make_bernoulli(A::Array{Float64, 2}) Generate a random 0/1 matrix from probabilities

Returns a matrix B of the same size as A, in which each element B(i,j) is 1 with probability A(i,j).

source: ProbabilisticNetwork/src/./matrix_utils.jl:22

---

make_binary(A::Array{Float64, 2}) Returns the adjacency/incidence matrix from a probability matrix

Returns a matrix B of the same size as A, in which each element B(i,j) is 1 if A(i,j) is greater than 0.

source: ProbabilisticNetwork/src/./matrix_utils.jl:73

---

make_threshold(A::Array{Float64, 2}, k::Float64) Generate a random 0/1 matrix from probabilities

Returns a matrix B of the same size as A, in which each element B(i,j) is 1 if A(i,j) is $>$ `k`. This is probably unwise to use this function since this practice is of questionnable relevance, but it is included for the sake of exhaustivity.

`k` must be in [0;1[.

source: ProbabilisticNetwork/src/./matrix_utils.jl:56

---

make_unipartite(A::Array{Float64, 2}) Transforms a bipartite network into a unipartite network

Note that this function returns an asymetric unipartite network.

source: ProbabilisticNetwork/src/./matrix_utils.jl:7

---

nestedness(A::Array{Float64, 2}) Nestedness of a matrix, using the Bastolla et al. (XXXX) measure

Returns three values:

- nestedness of the entire matrix
- nestedness of the columns
- nestedness of the rows

source: ProbabilisticNetwork/src/./nestedness.jl:36

---

nestedness_axis(A::Array{Float64, 2}) Nestedness of a single axis (called internally by `nestedness`)

source: ProbabilisticNetwork/src/./nestedness.jl:5

---

nodiag(A::Array{Float64, 2}) Sets the diagonal to 0

Returns a copy of the matrix A, with the diagonal set to 0. Will fail if the matrix is not square.

source: ProbabilisticNetwork/src/./matrix_utils.jl:38

---

null1(A::Array{Float64, 2}) Given a matrix `A`, `null1(A)` returns a matrix with the same dimensions, where every interaction happens with a probability equal to the connectance of `A`.

source: ProbabilisticNetwork/src/./nullmodels.jl:7

---

null2(A::Array{Float64, 2}) Given a matrix `A`, `null2(A)` returns a matrix with the same dimensions, where every interaction happens with a probability equal to the degree of each species.

source: ProbabilisticNetwork/src/./nullmodels.jl:46

---

null3in(A::Array{Float64, 2}) Given a matrix `A`, `null3in(A)` returns a matrix with the same dimensions, where every interaction happens with a probability equal to the in-degree (number of predecessors) of each species, divided by the total number of possible predecessors.

source: ProbabilisticNetwork/src/./nullmodels.jl:34

---

null3out(A::Array{Float64, 2}) Given a matrix `A`, `null3out(A)` returns a matrix with the same dimensions, where every interaction happens with a probability equal to the out-degree (number of successors) of each species, divided by the total number of possible successors.

source: ProbabilisticNetwork/src/./nullmodels.jl:20

---

nullmodel(A::Array{Float64, 2}) This function is a wrapper to generate replicated binary matrices from a template probability matrix `A`.

If you use julia on more than one CPU, *i.e.* if you started it with `julia -p k` where `k` is more than 1, this function will distribute each trial to one worker. Which means that it's fast.

Note that you will get a warning if there are less networks created than have been requested. Not also that this function generates networks, but do not check that their distribution is matching what you expect. Simulated annealing routines will be part of a later release.

Keyword arguments

- `n` (def. 1000), number of replicates to generate
- `max` (def. 10000), number of trials to make

source: ProbabilisticNetwork/src/./nullmodels.jl:78

Internal

@checkprob(p) Quite crude way of checking that a number is a probability

The two steps are

1. The number should be of the `Float64` type – if not, will yield a `TypeError`
2. The number should belong to [0,1] – if not, will throw a `DomainError`

signature: checkprob(p)

source: ProbabilisticNetwork/src/./proba_utils.jl:15