Package 'popdemo'

May 9, 2026

Title Demographic Modelling Using Projection Matrices

Version 1.3-3

Maintainer Iain Stott <iainmstott@gmail.com>

Description Tools for modelling populations and demography using matrix projection models, with deterministic and stochastic model implementations. Includes population projection, indices of short- and long-term population size and growth, perturbation analysis, convergence to stability or stationarity, and diagnostic and manipulation tools.

Depends R (>= 4.1.0)

Imports expm, graphics, methods, MCMCpack, stats

Suggests knitr, magick, rmarkdown

VignetteBuilder knitr

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Iain Stott [aut, cre],
Dave Hodgson [aut],
Stuart Townley [aut],
Stephen Ellner [ctb]

Repository CRAN

Date/Publication 2026-03-09 15:30:10 UTC

Contents

---

popdemo-package *Provides tools for demographic modelling using projection matrices*

Description

popdemo provides tools for modelling populations and demography using matrix projection models (MPMs), with deterministic and stochastic model implementations. These tools include population projection, indices of short- and long-term population size and growth, perturbation analysis, convergence to stability or stationarity, and diagnostic and manipulation tools. This includes:

POPULATION PROJECTION popdemo provides a simple means of projecting and plotting PPM models. project provides a means to project and plot population dynamics of both deterministic and stochastic models. Many methods are available for working with population projections: see Projection-class and Projection-plots

ASYMPTOTIC DYNAMICS The `eigs` function provides a simple means to calculate asymptotic population dynamics using matrix eigenvalues.

TRANSIENT DYNAMICS There are functions for calculating transient dynamics at various points of the population projection. `reac` measures immediate transient density of a population (within the first time step). `maxamp`, `maxatt` are near-term indices that measure the largest and smallest transient dynamics a population may exhibit overall, respectively. `inertia` measures asymptotic population density relative to stable state, and has many perturbation methods in the package (see below). All transient indices can be calculated using specific population structures, as well as bounds on population size.

PERTURBATION ANALYSIS Methods for linear perturbation (sensitivity and elasticity) analysis of asymptotic dynamics are available through the `sens`, `tfs_lambda` and `tfsm_lambda` functions. Elasticity analysis is also available using the `elas` function. Sensitivity analysis of transient dynamics is available using the `tfs_inertia` and `tfsm_inertia` functions. Methods for nonlinear perturbation (transfer function) analysis of asymptotic dynamics is achieved using `tfa_lambda` and `tfam_lambda`, whilst transfer function analysis of transient dynamics is available with `tfa_inertia` and `tfam_inertia`. These all have associated plotting methods linked to them: see `plot.tfa` and `plot.tfam`).

MODEL CONVERGENCE Information on the convergence of populations to stable state can be useful, and popdemo provides several means of analysing convergence. `dr` measures the damping ratio, and there are several distance measures available (see `KeyfitzD`, `projectionD` and `CohenD`). There is also a means of calculating convergence time through simulation: `convt`.

DIAGNOSTIC TOOLS `isPrimitive`, `isIrreducible` and `isErgodic` facilitate diagnosis of matrix properties pertaining to ergodicity.

UTILITIES `Matlab2R` allows coding of matrices in a Matlab style, which also facilitates import of multiple matrices simultaneously if comma-seperated files are used to import dataframes. Its analogue, `R2Matlab`, converts R matrices to Matlab-style strings, for easier export.

Author(s)

Maintainer: Iain Stott <iainmstott@gmail.com>

Authors:

- Dave Hodgson <D.J.Hodgson@exeter.ac.uk>
- Stuart Townley <S.B.Townley@exeter.ac.uk>

Other contributors:

- Stephen Ellner <spe2@cornell.edu> [contributor]

`blockmatrix` *Block-permute a reducible matrix*

Description

Conjugate a reducible matrix into block upper triangular form

Usage

```
blockmatrix(A)
```

Arguments

A	a square, reducible, non-negative numeric matrix of any dimension

Details

Any square, reducible, non-negative matrix may have its rows and columns conjugated so that it takes a block upper triangular structure, with irreducible square submatrices on the diagonal, zero submatrices in the lower triangle and non-negative submatrices in the upper triangle (Caswell 2001; Stott et al. 2010). `blockmatrix` permutes the rows and columns of a reducible matrix into this form, which enables further evaluation (e.g. computation of eigenvalues of submatrices).

Value

a list containing components:

`blockmatrix` the block-permuted matrix.

`stage.order` the permutation of rows/columns of `A` in the block-permuted matrix.

References

- Caswell (2001) Matrix population models 2nd ed. Sinauer.
- Stott et al. (2010) Methods. Ecol. Evol., 1, 242-252.

Examples

```
# Create a 3x3 reducible PPM
A <- matrix(c(0,1,0,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3)
dimnames(A) <- list(c("Juv", "Pre-R", "R"), c("Juv", "Pre-R", "R"))
A

# Block-permute the matrix
blockmatrix(A)
```

CohenD *Calculate Cohen's cumulative distance*

Description

Calculate Cohen's cumulative distance metric for a population matrix projection model.

Usage

```
CohenD(A, vector)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
vector	a numeric vector or one-column matrix describing the age/stage distribution used to calculate the distance.

Details

Calculates the cumulative distance metric as outlined in Cohen (1979). Will not work for reducible matrices and returns a warning for imprimitive matrices (although will not function for imprimitive matrices with nonzero imaginary components in the dominant eigenpair).

Value

Cohen's D1.

References

- Cohen (1979) SIAM J. Appl. Math., 36, 169-175.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.

See Also

Other DistanceMeasures: `KeyfitzD()`, `projectionD()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate Cohen cumulative distance
CohenD(A, vector=initial)
```

---

convt *Calculate time to convergence*

Description

Calculate the time to convergence of a population matrix projection model from the model projection

Usage

```
convt(A, vector = "n", accuracy = 0.01, iterations = 1e+05)
```

Arguments

A	a square, non-negative numeric matrix of any dimension
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution used to calculate the projection.
accuracy	the accuracy with which to determine convergence on asymptotic growth, expressed as a proportion (see details).
iterations	the maximum number of iterations of the model before the code breaks. For slowly-converging models and/or high specified convergence accuracy, this may need to be increased.

Details

convt works by simulating the given model and manually determining growth when convergence to the given accuracy is reached. Convergence on an asymptotic growth is deemed to have been reached when the growth of the model stays within the window determined by accuracy for 10*s iterations of the model, with s equal to the dimension of A. For example, projection of an 8 by 8 matrix with convergence accuracy of 1e-2 is deemed to have converged on asymptotic growth when 10*8=80 consecutive iterations of the model have a growth within 1-1e-2=0.99 (i.e. 99%) and 1+1e-2=1.01 (i.e. 101%) of each other.

If vector is specified, the convergence time of the projection of vector through A is returned. If vector="n" then asymptotic growths of the set of 'stage-biased' vectors are calculated. These projections are achieved using a set of standard basis vectors equal in number to the dimension of A. These have every element equal to 0, except for a single element equal to 1, i.e. for a matrix of dimension 3, the set of stage-biased vectors are: c(1,0,0), c(0,1,0) and c(0,0,1).

Due to the way in which convergence is defined, convt can only properly work for strongly ergodic models. Therefore, it will not function for imprimitive (therefore potentially weakly ergodic) or reducible (therefore potentially nonergodic) models.

Value

If vector is specified, the convergence time of vector projected through A.

If vector is not specified, a numeric vector of convergence times for corresponding stage-biased projections: the length of the vector is equal to the dimension of A; the first entry is the convergence time of [1,0,0,...], the second entry is the convergence time of [0,1,0,...], etc.).

References

- Stott et al. (2011) Ecol. Lett., 14, 959-970.

See Also

Other ConvergenceMeasures: dr(), truelambda()

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the convergence time of the 3 stage-biased
# populations within 0.1% of lambda-max
( convt(A, accuracy=1e-3) )

# Calculate the convergence time of the projection of initial and A
# to within 0.001% of lambda-max
( convt(A, vector=initial, accuracy=1e-5) )
```

---

dr *Calculate damping ratio*

Description

Calculate the damping ratio of a given population matrix projection model.

Usage

```
dr(A, return.time = FALSE, x = 10)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
return.time	(optional) a logical argument determining whether an estimated convergence time should be returned.
x	(optional) the logarithm used in determining estimated time to convergence (see details).

Details

The damping ratio is calculated as the ratio of the dominant eigenvalue to the modulus of the largest subdominant eigenvalue. Time to convergence can be estmimated by calculating `log(dr)/log(x)`, which is the time taken for the dominant eigenvalue to become `x` times larger than the largest subdominant eigenvalue.

Value

If `return.time=FALSE`, the damping ratio of `A`.
If `return.time=TRUE`, a list containing components:

dr the damping ratio of `A`

t the estimated time to convergence.

References

- Caswell (2001) Matrix Population Models 2nd. ed. Sinauer.
- Stott et al. (2010) Ecol. Lett., 14, 959-970.

See Also

Other ConvergenceMeasures: `convt()`, `truelambda()`

Examples

```
# Create a 3x3 PPM
A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3)

# Calculate damping ratio
dr(A)

# Calculate damping ratio and time to convergence using a
# multiple of 10
dr(A, return.time=TRUE, x=10)
```

`eigs` *Calculate asymptotic growth*

Description

Dominant eigenstuff of a population matrix projection model.

Usage

```
eigs(A, what = "all", check = TRUE)
```

Arguments

`A`	a square, nonnegative numeric matrix of any dimension.
`what`	what components of the dominant eigenstuff should be returned. A character vector, which may include: `"lambda"` the dominant eigenvalue (lambda) `"ss"` the dominant right eigenvector (stable stage) `"rv"` the dominant left eigenvector (reproductive value) the default, `"all"`, returns all of the above.
`check`	(logical) determines whether the dominant eigenvalue is checked for nonzero imaginary component, and largest absolute value. If either of these occur, then the dominant eigenvalue may not be described as truly dominant.

Details

eigs gives the dominant eigenstuff of a population projection model. This includes the dominant eigenvalue (asymptotic population growth), the dominant right eigenvector (stable age/stage distribution), and the dominant left eigenvector (reproductive value). The dominant eigenvalue is the eigenvalue with the largest real component, and the dominant eigenvectors are the eigenvectors that correspond to this eigenvalue. If the matrix is reducible, then there may be other real or complex eigenvalues whose absolute value are equal in magnitude to that of the dominant eigenvalue. In this case, eigs returns the first one, and gives a warning "More than one eigenvalues have equal absolute magnitude", for information.

Value

A list with possible components that depends on the contents of what:

lambda the dominant eigenvalue, which describes asymptotic population growth (if A is primitive; see isPrimitive). A real, nonnegative numeric vector of length 1.

ss the dominant right eigenvector, which describes the stable age/stage structure (if A is primitive; see isPrimitive). A real, nonnegative numeric vector equal to the dimension of A in length, scaled to sum to 1.

rv the dominant left eigenvector, which describes the reproductive value (if A is primitive; see isPrimitive). A real, nonnegative numeric vector equal to the dimension of A in length, scaled so that rv

If only one of these components is returned, then the value is not a list, but a single numeric vector.

Examples

```
# load the desert tortoise data
data(Tort)

# find the dominant eigenvalue
eigs(Tort, "lambda")

#find the stable stage structure
eigs(Tort, "ss")

#find the reproductive value
eigs(Tort, "rv")

#find both dominant eigenvectors
eigs(Tort, c("ss","rv"))

#find all eigenstuff
eigs(Tort)
```

`elas` *Calculate elasticity matrix*

Description

Calculate the elasticity matrix for a specified population matrix projection model using eigenvectors.

Usage

```
elas(A, eval = "max")
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension
`eval`	the eigenvalue to evaluate. Default is `eval="max"`, which evaluates the dominant eigenvalue (the eigenvalue with largest REAL value: for imprimitive or reducible matrices this may not be the first eigenvalue). Otherwise, specifying e.g. `eval=2` will evaluate elasticity of the eigenvalue with second-largest modulus.

Details

`elas` uses the eigenvectors of `A` to calculate the elasticity matrix of the specified eigenvalue, see section 9.1 in Caswell (2001). Same method as `elasticity` in `popbio` but can also evaluate subdominant eigenvalues.

Value

A numeric (real or complex) matrix of equal dimension to `A`.

References

- Caswell (2001) Matrix Population Models 2nd ed. Sinauer.

See Also

Other PerturbationAnalyses: `sens()`, `tfa_inertia()`, `tfa_lambda()`, `tfam_inertia()`, `tfam_lambda()`, `tfs_inertia()`, `tfs_lambda()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate sensitivities of dominant eigenvalue
elas(A)
# Calculate sensitivities of first subdominant eigenvalue,
```

```
# only for observed transitions
elas(A, eval=2)
```

`inertia` *Calculate population inertia*

Description

Calculate population inertia for a population matrix projection model.

Usage

```
inertia(A, vector = "n", bound = NULL, return.N = FALSE, t = NULL)
```

Arguments

`A`	a square, primitive, irreducible, non-negative numeric matrix of any dimension
`vector`	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate a 'case-specific' maximal amplification
`bound`	(optional) specifies whether an upper or lower bound should be calculated (see details).
`return.N`	(optional) if `TRUE`, returns population size for a specified `t` (including effects of asymptotic growth and initial population size), alongside standardised inertia.
`t`	(optional) the projection interval at which `N` is to be calculated. Calculation of `N` is only accurate for `t` where the model has converged (see details)

Details

A nonstable population, when it achieves asymptotic growth following transient dynamics, is a fixed ratio of the size of a population projected with the same initial size but stable structure. `inertia` calculates the value of this ratio (Koons et al. 2007)

If `vector="n"` then either `bound="upper"` or `bound="lower"` must be specified, which calculate the upper or lower bound on population inertia (i.e. the largest and smallest values that inertia may take) respectively. Specifying `vector` overrides calculation of a bound, and will yield a 'case-specific' value for inertia.

`inertia` will not work with imprimitive or reducible matrices.

Value

If `vector="n"`, the upper bound on inertia of `A` if `bound="upper"` and the lower bound on inertia of `A` if `bound="lower"`.
If `vector` is specified, the case-specific inertia of the model.
If `return.N=TRUE` and `t` is specified, a list with components:

inertia the bound on or case-specific inertia

N the population size at specified `t`.

References

- Koons et al. (2007) Ecology, 88, 2867-2867.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.

See Also

Transfer function methods for inertia: `inertia.tfa`, `inertia.tfamatrix`, `inertia.tfsens`, `inertia.tfsensmatrix`

Other TransientIndices: `Kreiss()`, `maxamp()`, `maxatt()`, `reac()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the upper bound on inertia of A
inertia(A,bound="upper")

# Calculate the lower bound on inertia of A
inertia(A,bound="lower")

# Calculate case-specific inertia of A and initial
inertia(A, vector=initial)

# Calculate case-specific inertia of A and initial and
# return realised population size at t=25
inertia(A, vector=initial, return.N=TRUE, t=25)
```

---

isErgodic *Determine ergodicity of a matrix*

Description

Determine whether a matrix is ergodic or nonergodic

Usage

```
isErgodic(A, digits = 12, return.eigvec = FALSE)
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension.
`digits`	the number of digits that the dominant left eigenvector should be rounded to.
`return.eigvec`	(optional) logical argument determining whether or not the dominant left eigenvector should be returned.

Details

isErgodic works on the premise that a matrix is ergodic if and only if the dominant left eigenvector (the reproductive value vector) of the matrix is positive (Stott et al. 2010).

In rare cases, R may calculate that the dominant left eigenvector of a nonergodic matrix contains very small entries that are approximate to (but not equal to) zero. Rounding the dominant eigenvector using digits prevents mistakes.

Value

If return.eigvec=FALSE, either TRUE (for an ergodic matrix) or FALSE (for a nonergodic matrix).

If return.eigvec=TRUE, a list containing elements:

ergodic TRUE or FALSE, as above

eigvec the dominant left eigenvector of A

References

- Stott et al. (2010) Methods Ecol. Evol., 1, 242-252.

See Also

Other PerronFrobeniusDiagnostics: isIrreducible(), isPrimitive()

Examples

```
# Create a 3x3 ergodic PPM
( A <- matrix(c(0,0,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Diagnose ergodicity
isErgodic(A)

# Create a 3x3 nonergodic PPM
B<-A; B[3,2] <- 0; B

# Diagnose ergodicity and return left eigenvector
isErgodic(B, return.eigvec=TRUE)
```

---

isIrreducible *Determine reducibility of a matrix*

Description

Determine whether a matrix is irreducible or reducible

Usage

```
isIrreducible(A)
```

Arguments

A	a square, non-negative numeric matrix of any dimension.

Details

isIrreducible works on the premise that a matrix **A** is irreducible if and only if (**I**+**A**)^(s-1) is positive, where **I** is the identity matrix of the same dimension as **A** and s is the dimension of **A** (Caswell 2001).

Value

TRUE (for an irreducible matrix) or FALSE (for a reducible matrix).

References

- Caswell (2001) matrix Population Models, 2nd. ed. Sinauer.

See Also

Other PerronFrobeniusDiagnostics: isErgodic(), isPrimitive()

Examples

```
# Create a 3x3 irreducible PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Diagnose reducibility
isIrreducible(A)

# Create a 3x3 reducible PPM
B<-A; B[3,2] <- 0; B

# Diagnose reducibility
isIrreducible(B)
```

---

isPrimitive *Determine primitivity of a matrix*

Description

Determine whether a matrix is primitive or imprimitive

Usage

```
isPrimitive(A)
```

Arguments

A	a square, non-negative numeric matrix of any dimension.

Details

isPrimitive works on the premise that a matrix **A** is primitive if **A**^(s^2-(2*s)+2) is positive, where s is the dimension of **A** (Caswell 2001).

Value

TRUE (for an primitive matrix) or FALSE (for an imprimitive matrix).

References

- Caswell (2001) matrix Population Models, 2nd. ed. Sinauer.

See Also

Other PerronFrobeniusDiagnostics: isErgodic(), isIrreducible()

Examples

```
# Create a 3x3 primitive PPM
( A <- matrix(c(0,1,2,0.5,0,0,0,0.6,0), byrow=TRUE, ncol=3) )

# Diagnose primitivity
isPrimitive(A)

# Create a 3x3 imprimitive PPM
B<-A; B[1,2] <- 0; B

# Diagnose primitivity
isPrimitive(B)
```

KeyfitzD *Calculate Keyfitz's delta*

Description

Calculate Keyfitz's delta for a population matrix projection model.

Usage

```
KeyfitzD(A, vector)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
vector	a numeric vector or one-column matrix describing the age/stage distribution used to calculate the distance.

Details

Keyfitz's delta is the sum of the differences between the stable demographic vector (the dominant right eigenvector of A) and the demographic distribution vector of the population (given by vector). KeyfitzD will not work for reducible matrices and returns a warning for imprimitive matrices (although will not function for imprimitive matrices with nonzero imaginary components in the dominant eigenpair).

Value

Keyfitz's delta.

References

- Keyfitz (1968) Introduction to the Mathematics of Populations. Addison-Wesley.
- Stott et al. (2010) Ecol. Lett., 14, 959-970.

See Also

Other DistanceMeasures: CohenD(), projectionD()

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate Keyfitz's delta
KeyfitzD(A, vector=initial)
```

Kreiss *Calculate Kreiss bounds*

Description

Calculate the upper or lower Kreiss bound for a population matrix projection model.

Usage

```
Kreiss(
  A,
  bound = NULL,
  return.r = FALSE,
  theta = 1,
  rlimit = 100,
  step1 = 0.001,
  step2 = 1e-06
)
```

Arguments

`A`	a square, irreducible, non-negative numeric matrix of any dimension
`bound`	(optional) specifies whether an upper or lower bound should be calculated.
`return.r`	(optional) specifies whether the value of r at which the Kreiss bound is achieved should be returned (see details).
`theta`	the value to which the Kriess bound is to be assessed relative to (see details).
`rlimit`	the maximum value of r that may be reached before the code breaks (see details).
`step1`, `step2`	determine the iterative process in calculating the Kreiss bound (see details).

Details

`Kreiss` by default returns a standardised Kreiss bound relative to both asymptotic growth/decline and initial population density (Townley & Hodgson 2008; Stott et al. 2011). It uses an iterative process that evaluates a function of the resolvent of `A` over a range of values r where r>`theta`. This iterative process finds the maximum/minimum of the function for the upper/lower bounds respectively. The process is determined using `step1` and `step2`: in order to increase accuracy but keep computation time low, the function is evaluated forward in steps equal to `step1` until the maximum/minimum is passed and then backward in steps of `step2` to more accurately find the maximum/minimum itself. Therefore, `step1` should be larger than `step2`. The balance between both will determine computation time, whilst accuracy is determined almost solely by `step2`. The defaults should be sufficient for most matrices.

`theta` defaults to 1, which means the Kriess bound is assessed relative to both asymptotic growth and initial population size. Sometimes, the maximum/minimum of the function occurs at r–>`theta`, in which case r is equal to `theta+step2`. Setting `return.r=TRUE` tells the function to return the value of r where the maximum/minimum occurs alongside the value of the Kreiss bound. r may not exceed `rlimit`.

`Kreiss` will not work with reducible matrices, and returns a warning for imprimitive matrices.

Value

The upper or lower Kreiss bound of `A`.
If `return.r=TRUE`, a list with components:

bound the upper or lower Kriess bound

r the value of r at which the function is minimised/maximised.

References

- Stott et al. (2011) Ecol. Lett., 14, 959-970.
- Townley & Hodgson (2008) J. Appl. Ecol., 45, 1836-1839.

See Also

Other TransientIndices: `inertia()`, `maxamp()`, `maxatt()`, `reac()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the upper Kreiss bound of A
Kreiss(A, bound="upper")

# Calculate the lower Kreiss bound of A
Kreiss(A, bound="lower")

# Calculate the upper Kreiss bound of A and return
# the value of r at which the function is maximised
Kreiss(A, bound="upper", return.r=TRUE)
```

`Matlab2R` *Read Matlab style matrices into R*

Description

Read a matrix coded in a Matlab style into R to create an object of class matrix

Usage

```
Matlab2R(M)
```

Arguments

`M`	an object of class character that represents a numeric matrix coded in a Matlab style.

Details

Matlab reads matrices using a unique one-line notation that can prove useful for storage in databases and importing multiple matrices into a program at once, amongst other applications. This notation is by row, with "[" and "]" to specify the beginning and end of the matrix respectively, ";" to specify a new row and a space between each matrix element. Thus, the R matrix created using `matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3)` is equivalent to [0 1 2;0.5 0.1 0;0 0.6 0.6].

Matlab2R takes a Matlab-coded matrix expressed as a character string and converts it into an R object of class matrix. As well as providing a simpler means of matrix notation in R, it also enables simultaneous import of multiple matrices of varying dimensions, using comma-seperated dataframes and tables.

Value

An object of class matrix.

See Also

R2Matlab

Examples

```
# Create a 3x3 PPM using Matlab2R
( A<-Matlab2R("[0 1 2;0.5 0.1 0;0 0.6 0.6]") )
```

---

maxamp *Calculate maximal amplification*

Description

Calculate maximal amplification for a population matrix projection model.

Usage

```
maxamp(
  A,
  vector = "n",
  return.N = FALSE,
  return.t = FALSE,
  return.stage = FALSE,
  conv.iterations = 1e+05,
  conv.accuracy = 1e-05
)
```

Arguments

A	a square, primitive, non-negative numeric matrix of any dimension
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate a 'case-specific' maximal amplification.
return.N	(optional) if TRUE, returns population size at the point of maximal amplification (including effects of asymptotic growth and initial population size), alongside standardised maximal amplification.

`return.t` (optional) if `TRUE`, returns the time at which maximal amplification occurs in the population projection.

`return.stage` (optional) if `TRUE` and `vector="n"`, returns the stage that achieves the bound on maximal amplification.

`conv.iterations` the maximum number of iterations allowed when calulating convergence time (see details). Please see `iterations` in `convt`.

`conv.accuracy` the accuracy of convergence (see details). Please see `accuracy` in `convt`.

Details

`maxamp` returns a standardised measure of maximal amplification, discounting the effects of both initial population size and asymoptotic growth (Stott et al. 2011).

If `vector` is not specified then the bound on maximal amplification (the largest maximal amplification that may be achieved) is returned, otherwise a 'case-specific' maximal amplification for the specified matrix and demographic structure is calculated. Note that not all demographic structures will yield a maximal amplification: if the model does not amplify then an error is returned.

Setting `return.N=T`, `return.t=T` and `return.stage=T` results in the function returning realised population size at maximal amplification (including the effects of asymptotic growth and initial population size), the time at which maximal amplification occurs and (if `vector="n"`), the stage-bias that results in the bound on maximal amplification, respectively. NOTE that `N` is not indicative of maximum possible population size for a non-standardised model: merely the population size at the point of maximal amplification (i.e. largest positive deviation from lambda-max).

`max.amp` uses a simulation technique, using `project` to project the dynamics of the model before evaluating maximum projected density over all t. `conv.accuracy` and `conv.iterations` are passed to `convt`, which is used to find the point of model convergence in order to ensure maximal amplification is correctly captured in model projection.

`maxamp` will not work for imprimitive or reducible matrices.

Value

If `vector="n"`, the bound on maximal amplification of `A`.
If `vector` is specified, the case-specific maximal amplification of the model.
If `return.N=TRUE`, `return.t=TRUE` and/or `return.stage=TRUE`, a list with possible components:

maxamp the bound on or case-specific maximal amplification

N the population size at the point of maximal amplification, including the effects of initial population size and asymptotic growth. NOTE that `N` is not indicative of maximum possible population size for a non-standardised model: merely the population size at the point of maximal amplification (i.e. largest positive deviation from lambda-max).

t the projection interval at which maximal amplification is achieved.

stage (only if `vector="n"`), the stage that achieves the bound on maximal amplification.

References

- Neubert & Caswell (1997) Ecology, 78, 653-665.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.
- Townley & Hodgson (2008) J. Appl. Ecol., 45, 1836-1839.

See Also

Other TransientIndices: `Kreiss()`, `inertia()`, `maxatt()`, `reac()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the bound on maximal amplification of A
maxamp(A)

# Calculate the bound on maximal amplification of A and
# return the stage that achieves it
maxamp(A, return.stage=TRUE)

# Calculate case-specific maximal amplification of A
# and initial
maxamp(A, vector=initial)

# Calculate case-specific maximal amplification of A
# and initial and return realised population size and the
# time at which it is achieved
maxamp(A, vector=initial, return.N=TRUE, return.t=TRUE)
```

---

maxatt *Calculate maximal attenuation*

Description

Calculate maximal attenuation for a population matrix projection model.

Usage

```
maxatt(
  A,
  vector = "n",
  return.N = FALSE,
  return.t = FALSE,
```

```
  return.stage = FALSE,
  conv.iterations = 1e+05,
  conv.accuracy = 1e-05
)
```

Arguments

A	a square, primitive, non-negative numeric matrix of any dimension
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate a 'case-specific' maximal attenuation
return.N	(optional) if TRUE, returns population size at the point of maximal attenuation (including effects of asymptotic growth and initial population size), alongside standardised maximal attenuation.
return.t	(optional) if TRUE, returns the time at which maximal attenuation occurs in the population projection.
return.stage	(optional) if TRUE and vector="n", returns the stage that achieves the bound on maximal attenuation.
conv.iterations	the maximum number of iterations allowed when calulating convergence time (see details). Please see iterations in convt.
conv.accuracy	the accuracy of convergence (see details). Please see accuracy in convt.

Details

maxatt returns a standardised measure of maximal attenuation, discounting the effects of both initial population size and asymoptotic growth (Stott et al. 2011).

If vector is not specified then the bound on maximal attenuation (the greatest maximal attenuation that may be achieved) is returned, otherwise a 'case-specific' maximal attenuation for the specified matrix and demographic structure is calculated. Note that not all demographic structures will yield a maximal attenuation: if the model does not amplify then an error is returned.

Setting return.N=T, return.t=T and return.stage=T results in the function returning realised population size at maximal attenuation (including the effects of asymptotic growth and initial population size), the time at which maximal attenuation occurs and (if vector="n"), the stage-bias that results in the bound on maximal attenuation, respectively. NOTE that N is not indicative of minuium possible population size for a non-standardised model: merely the population size at the point of maximal attenuation (i.e. largest negative deviation from lambda-max).

max.att uses a simulation technique, using project to project the dynamics of the model before evaluating minimum projected density over all t. conv.accuracy and conv.iterations are passed to convt, which is used to find the point of model convergence in order to ensure maximal attenuation is correctly captured in model projection.

maxatt will not work for imprimitive or reducible matrices.

Value

If vector="n", the bound on maximal attenuation of A.
If vector is specified, the case-specific maximal attenuation of the model.

If `return.N=TRUE`, `return.t=TRUE` and/or `return.stage=TRUE`, a list with possible components:

maxatt the bound on or case-specific maximal attenuation

N the population size at the point of maximal attenuation, including the effects of initial population size and asymptotic growth. NOTE that `N` is not indicative of minimum possible population size for a non-standardised model: merely the population size at the point of maximal attenuation (i.e. largest negative deviation from lambda-max).

t the projection interval at which maximal attenuation is achieved.

stage (only if `vector="n"`), the stage that achieves the bound on maximal attenuation.

References

- Neubert & Caswell (1997) Ecology, 78, 653-665.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.
- Townley & Hodgson (2008) J. Appl. Ecol., 45, 1836-1839.

See Also

Other TransientIndices: `Kreiss()`, `inertia()`, `maxamp()`, `reac()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(3,1,1) )

# Calculate the bound on maximal attenuation of A
maxatt(A)

# Calculate the bound on maximal attenuation of A and
# return the stage that achieves it
maxatt(A, return.stage=TRUE)

# Calculate case-specific maximal attenuation of A
# and initial
maxatt(A, vector=initial)

# Calculate case-specific maximal attenuation of A
# and initial and return realised population size and the
# time at which it is achieved
maxatt(A, vector=initial, return.N=TRUE, return.t=TRUE)
```

Pbear *Polar bear matrices*

Description

Matrix projection model for the polar bear *Ursus maritimus*, with 5 matrices corresponding to years 2001-2005. The matrices are based on a population in the southern Beaufort Sea. During 2001-2003, ice conditions were classified as "good", but in 2004-2005, ice conditions were classified as "poor". Poor ice conditions lead to worse population performance. Stages are based on age and reproductive status:
Stage-1: 2-year-old
Stage 2: 3-year-old
Stage 3: 4-year-old
Stage 4: adult (5+ years old), available to breed
Stage 5: adult, with cub (0-1 years old)
Stage 6: adult, with yearling (1-2 years old).

Usage

```
data(Pbear)
```

Format

List object containing matrices.

Details

The population structure is `c(0.106, 0.068, 0.106, 0.461, 0.151, 0.108))`

References

- Hunter et al. (2010) Ecology, 91, 2883-2897.

Examples

```
#read in data
data(Pbear)
Pbear
```

plot.tfa *Plot transfer function*

Description

Plot a transfer function

Usage

```
## S3 method for class 'tfa'
plot(x, xvar = NULL, yvar = NULL, ...)
```

Arguments

x	an object of class 'tfa' (transfer function analysis) created using tfa_lambda or tfa_inertia.
xvar, yvar	(optional) the variables to plot on the x and y axes. May be "p", "lambda" or "inertia". Defaults to xvar="p" and yvar="lambda" for objects created using tfa_lambda and xvar="p" and yvar="inertia" for objects created using tfa_inertia.
...	arguments to be passed to methods: see par and plot.

Details

plot.tfa plots transfer functions (class tfa) created using tfa_lambda or tfa_inertia.

See Also

Constructor functions: tfa_lambda, tfa_inertia

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the transfer function of A[3,2] given a range of lambda
evals <- eigen(A)$values
lmax <- which.max(Re(evals))
lambda <- Re(evals[lmax])
lambdarange <- seq(lambda-0.1, lambda+0.1, 0.01)
( transfer <- tfa_lambda(A, d=c(0,0,1), e=c(0,1,0), lambdarange=lambdarange) )

# Plot the transfer function
plot(transfer)

# Create an initial stage structure
( initial <- c(1,3,2) )
```

```
# Calculate the transfer function of upper bound on inertia
# given a perturbation to A[3,2]
( transfer<-tfa_inertia(A, d=c(0,0,1), e=c(0,1,0), bound="upper",
                        prange=seq(-0.6,0.4,0.01)) )

# Plot the transfer function (defaults to inertia ~ p)
plot(transfer)

# Plot inertia against lambda
plot(transfer, xvar="lambda", yvar="inertia")
```

`plot.tfam` *Plot transfer function*

Description

Plot a matrix of transfer functions

Usage

```
## S3 method for class 'tfam'
plot(x, xvar = NULL, yvar = NULL, mar = c(1.1, 1.1, 0.1, 0.1), ...)
```

Arguments

`x`	an object of class 'tfam' (transfer function analysis matrix) created using `tfam_lambda` or `tfam_inertia`.
`xvar, yvar`	(optional) the variables to plot on the x and y axes. May be `"p"`, `"lambda"` or `"inertia"`. Defaults to `xvar="p"` and `yvar="lambda"` for objects created using `tfam_lambda`, and `xvar="p"` and `yvar="inertia"` for objects created using `tfam_inertia`.
`mar`	the margin limits on the plots: see `par`
`...`	arguments to be passed to methods: see `par` and `plot`.

Details

`plot.tfam` plots matrices of transfer functions (class `tfam`) created using `tfam_lambda` or `tfam_inertia`. The plot is laid out to correspond with the nonzero entries of the matrix used to generate the transfer functions, for easy visual comparison of how perturbation affects different matrix elements.

See Also

Constructor functions: `tfam_lambda`, `tfam_inertia`

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the matrix of transfer functions using default arguments
( tfmat<-tfam_lambda(A) )

# Plot the matrix of transfer functions
plot(tfmat)

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the matrix of transfer functions for inertia and
# specified initial stage structure using default arguments
( tfmat2<-tfam_inertia(A,vector=initial) )

# Plot the result (defaults to inertia ~ p)
plot(tfmat2)

# Plot inertia ~ lambda
plot(tfmat2, xvar="lambda", yvar="inertia")
```

---

popdemo-deprecated *Deprecated functions in the popdemo package*

Description

Deprecated functions in the popdemo package

Usage

```
Cohen.cumulative(...)
convergence.time(...)
inertia.tfa(...)
inertia.tfamatrix(...)
inertia.tfsens(...)
inertia.tfsensmatrix(...)
is.matrix_ergodic(...)
is.matrix_irreducible(...)
is.matrix_primitive(...)
Keyfitz.delta(...)
projection.distance(...)
tfa(...)
tfamatrix(...)
tfsens(...)
tfsensmatrix(...)
```

```
minCS(...)
reactivity(...)
firststepatt(...)
```

Arguments

...	Parameters to be passed to the new function versions

Details

Many functions have become deprecated as of popdemo_1.0-0 (meaning they will stop working at some point in the future). In most cases, this is because functions needed to be re-named. For now the older function names will work but issue a warning, but you should use the new function names wherever possible. Please update your code, and I'm sorry for the inconvenience!

Avoiding S3 troubles Most deprecated functions needed to be renamed because they included a period in the function name: the new function names don't use periods, which is a better approach for playing nicely with the S3 object-oriented system (see Hadley Wickham's OO field guide for more info). These are:

`Cohen.cumulative`	now called `CohenD`
`convergence.time`	now called `convt`
`inertia.tfa`	now called `tfa_inertia`
`inertia.tfamatrix`	now called `tfam_inertia`
`inertia.tfsens`	now called `tfs_inertia`
`inertia.tfsensmatrix`	now called `tfsm_inertia`
`is.matrix_ergodic`	now called `isErgodic`
`is.matrix_irreducible`	now called `isIrreducible`
`is.matrix_primitive`	now called `isPrimitive`
`Keyfitz.delta`	now called `KeyfitzD`
`projection.distance`	now called `projectionD`

Consistency Some other functions have been renamed to keep consistency with new functions, and also to further avoid problems with S3 methods by making sure classes and functions don't have the same names:

`tfa`	now called `tfa_lambda`
`tfamatrix`	now called `tfam_lambda`
`tfsens`	now called `tfs_lambda`
`tfsensmatrix`	now called `tfsm_lambda`

Hidden functions Some functions have been made internal (they're "hidden" but you can still use them):

`minCS`	now called `.minCS`
`tf`	now called `.tf`

Merged functions Two functions are deprecated because they have been merged into one:

`reactivity,firststepatt`	now handled by `reac`.

Before, `reactivity` handled first-timestep amplification and `firststepatt` handled first-timestep attenuation. This is silly, because a projection EITHER amplifies OR attenuates in the first timestep. Desptite the semantics, `reac` now deals with both amplification and attenuation in the first timestep, everything that was calculable in the previous two functions is also calculable in the one new function.

`project` *Project population dynamics*

Description

Project dynamics of a specified population matrix projection model.

Usage

```
project(
  A,
  vector = "n",
  time = 100,
  standard.A = FALSE,
  standard.vec = FALSE,
  return.vec = TRUE,
  Aseq = "unif",
  Astart = NULL,
  draws = 1000,
  alpha.draws = "unif",
  PREcheck = TRUE
)
```

Arguments

`A`	a matrix, or list of matrices. If `A` is a matrix, then `project` performs a 'deterministic' projection, where the matrix does not change with each timestep. If `A` is a list of matrices, then `project` performs a 'stochastic' projection where the matrix varies with each timestep. The sequence of matrices is determined using `Aseq`. Matrices must be square, non-negative and numeric. If `A` is a list, all matrices must have the same dimension. 'Projection' objects inherit names from `A`: if `A` is a matrix, stage names (in mat and vec slots) are inherited from its column names..

`vector` (optional) a numeric vector or matrix describing the age/stage distribution(s) used to calculate the projection. Single population vectors can be given either as a numeric vector or one-column matrix. Multiple vectors are specified as a matrix, where each column describes a single population vector. Therefore the number of rows of the matrix should be equal to the matrix dimension, whilst the number of columns gives the number of vectors to project. `vector` may also take either "n" (default) to calculate the set of stage-biased projections (see details), or "diri" to project random population vectors drawn from a dirichlet distribution (see details).

`time` the number of projection intervals.

`standard.A` (optional) if `TRUE`, scales each matrix in `A` by dividing all elements by the dominant eigenvalue. This standardises asymptotic dynamics: the dominant eigenvalue of the scaled matrix is 1. Useful for assessing transient dynamics.

`standard.vec` (optional) if `TRUE`, standardises each `vector` to sum to 1, by dividing each vector by its sum. Useful for assessing projection relative to initial population size.

`return.vec` (optional) if `TRUE`, returns the time series of demographic (st)age vectors as well as overall population size.

`Aseq` (optional, for stochastic projections only) the sequence of matrices in a stochastic projection. `Aseq` may be either:

- "unif" (default), which results in every matrix in `A` having an equal, random chance of being chosen at each timestep.
- a square, nonnegative left-stochastic matrix describing a first-order Markov chain used to choose the matrices. The transitions are defined COLUMNWISE: each column j describes the probability of choosing stage (row) i at time t+1, given that stage (column) j was chosen at time t. `Aseq` should have the same dimension as the number of matrices in `A`.
- a numeric vector giving a specific sequence which corresponds to the matrices in `A`.
- a character vector giving a specific sequence which corresponds to the names of the matrices in `A`.

`Astart` (optional) in a stochastic projection, the matrix with which to initialise the projection (either numeric, corresponding to the matrices in `A`, or character, corresponding to the names of matrices in `A`). When `Astart = NULL` (the default), a random initial matrix is chosen.

`draws` if `vector="diri"`, the number of population vectors drawn from dirichlet.

`alpha.draws` if `vector="diri"`, the alpha values passed to `rdirichlet`: used to bias draws towards or away from a certain population structure.

`PREcheck` many functions in `popdemo` first check Primitivity, Reducibility and/or Ergodicity of matrices, with associated warnings and/or errors if a matrix breaks any assumptions. Set `PREcheck=FALSE` if you want to bypass these checks.

Details

If `vector` is specified, `project` will calculate population dynamics through time by projecting this vector / these vectors through `A`. If multiple vectors are specified, a separate population projection is calculated for each.

If vector="n", project will automatically project the set of 'stage-biased' vectors of A. Effectively, each vector is a population consisting of all individuals in one stage. These projections are achieved using a set of standard basis vectors equal in number to the dimension of A. The vectors have every element equal to 0, except for a single element equal to 1, i.e. for a matrix of dimension 3, the set of stage-biased vectors are: c(1,0,0), c(0,1,0) and c(0,0,1). Stage-biased projections are useful for seeing how extreme transient dynamics can be.

If vector="diri", project draws random population vectors from the dirichlet distribution. draws gives the number of population vectors to draw. alpha.draws gives the parameters for the dirichlet and can be used to bias the draws towards or away from certain population structures. The default is alpha.draws="unif", which passes rep(1,dim) (where dim is the dimension of the matrix), resulting in an equal probability of any random population vector. Relative values in the vector give the population structure to focus the distribution on, and the absolute value of the vector entries (and their sum) gives the strength of the distribution: values greater than 1 make it more likely to draw from nearby that population structure, whilst values less than 1 make it less likely to draw from nearby that population structure.

Projections returned are of length time+1, as the first element represents the population at t=0.

Projections have their own plotting method (see Projection-plots) to enable easy graphing.

In addition to the examples below, see the "Deterministic population dynamics" and "Stochastic population dynamics" vignettes for worked examples that use the project function.

Value

A Projection-class item. 'Projection' objects inherit from a standard array, and can be treated as such. Therefore, if if vector is specified, the 'Projection' object will behave as:

- if a single vector is given, a numeric vector of population sizes of length time+1
- if multiple vectors are given, a numeric matrix of population projections where each column represents a single population projection and is of length time+1
- if vector="n", a numeric matrix of population projections where each column represents a single stage-biased projection and is of length time+1.
- if vector="diri", a numeric matrix of population projections where each column represents projection of a single vector draw and each column is of length time+1

See documentation on Projection-class objects to understand how to access other slots (e.g. (st)age vectors through the population projection) and for S4 methods (e.g. plotting projections). Some examples for understanding the structure of 3D arrays returned when return.vec=TRUE: when projecting a 3 by 3 matrix for >10 time intervals (see examples), element [11,3,2] represents the density of stage 3 at time 10 for either vector 2 (multiple vectors), stage-bias 2 (vector="n") or draw 2 (vector="diri"); note that because element 1 represents t=0, then t=10 is found at element 11. The vector [,3,2] represents the time series of densities of stage 3 in the projection of vector 2 / stage-bias 2 / draw 2. The matrix [,,2] represents the time series of all stages in the projection of vector 2 / stage-bias 2 / draw 2.

Note that the projections inherit the labelling from A and vector, if it exists. Both stage and vector names are taken from the COLUMN names of A and vector respectively. These may be useful for selecting from the projection object, and for labelling graphs when plotting Projection objects.

See Also

Projection-class Projection-plots

Examples

```
### USING PROJECTION OBJECTS

# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Project stage-biased dynamics of A over 70 intervals
( pr <- project(A, vector="n", time=70) )
plot(pr)

# Access other slots
vec(pr)  #time sequence of population vectors
bounds(pr)  #bounds on population dynamics
mat(pr)  #matrix used to create projection
Aseq(pr)  #sequence of matrices (more useful for stochastic projections)
projtype(pr)  #type of projection
vectype(pr)  #type of vector(s) initiating projection

# Extra information on the projection
nproj(pr)  #number of projections
nmat(pr)  #number of matrices (more usefulk for stochastic projections)
ntime(pr)  #number of time intervals

# Select the projection of stage 2 bias
pr[,2]

# Project stage-biased dynamics of standardised A over 30 intervals
( pr2 <- project(A, vector="n", time=30, standard.A=TRUE) )
plot(pr2)

#Select the projection of stage 2 bias
pr2[,2]

# Select the density of stage 3 in bias 2 at time 10
vec(pr2)[11,3,2]

# Select the time series of densities of stage 2 in bias 1
vec(pr2)[,2,1]

#Select the matrix of population vectors for bias 2
vec(pr2)[,,2]

# Create an initial stage structure
( initial <- c(1,3,2) )

# Project A over 50 intervals using a specified population structure
( pr3 <- project(A, vector=initial, time=50) )
plot(pr3)
```

```
# Project standardised dynamics of A over 10 intervals using
# standardised initial structure and return demographic vectors
( pr4 <- project(A, vector=initial, time=10, standard.vec=TRUE,
                 standard.A=TRUE, return.vec=TRUE) )
plot(pr4)

# Select the time series for stage 1
vec(pr4)[,1]

### DETERMINISTIC PROJECTIONS

# Load the desert Tortoise matrix
data(Tort)

# Create an initial stage structure
Tortvec1 <- c(8, 7, 6, 5, 4, 3, 2, 1)

# Create a projection over 30 time intervals
( Tortp1 <- project(Tort, vector = Tortvec1, time = 10) )

# plot p1
plot(Tortp1)
plot(Tortp1, bounds = TRUE) #with bounds

# new display parameters
plot(Tortp1, bounds = TRUE, col = "red", bty = "n", log = "y",
     ylab = "Number of individuals (log scale)",
     bounds.args = list(lty = 2, lwd = 2) )

# multiple vectors
Tortvec2 <- cbind(Tortvec1, c(1, 2, 3, 4, 5, 6, 7, 8))
plot(project(Tort, vector = Tortvec2), log = "y")
plot(project(Tort, vector = Tortvec2), log = "y", labs = FALSE) #no labels

# dirichlet distribution
# darker shading indicates more likely population size
Tortshade <- project(Tort, time = 30, vector = "diri", standard.A = TRUE,
              draws = 500, alpha.draws = "unif")
plot(Tortshade, plottype = "shady", bounds = TRUE)

### STOCHASTIC PROJECTIONS
# load polar bear data
data(Pbear)

# project over 50 years with uniform matrix selection
Pbearvec <- c(0.106, 0.068, 0.106, 0.461, 0.151, 0.108)
p2 <- project(Pbear, Pbearvec, time = 50, Aseq = "unif")

# stochastic projection information
Aseq(p2)
projtype(p2)
nmat(p2)
```

```
# plot
plot(p2, log = "y")
```

`Projection-class` *'Projection' object S4 class*

Description

Projection objects are created using the `project` function. Primarily, they contain overall population size over time: they can be treated as a vector (single population projection) or matrix (multiple population projections; see information on slot ".Data" below). They also contain further information on the population projection. These extra pieces of information are described below in the "Slots" section, and the methods for accessing them appear below. These are:

`vec` access population vectors

`bounds` access bounds on population dynamics

`mat` access projection matrix/matrices used to create projection(s)

`Aseq` access projection matrix sequence used to create projection(s)

`projtype` find out projection type

`vectype` access type of vector used to initiate population projection(s)

Other methods for accessing basic information from the projection are:

`nproj` access projection matrix/matrices used to create projection

`nmat` number of projection matrices used to create projection(s)

`ntime` number of time intervals

Plotting and display methods for 'Projection' objects can be found on the `Projection-plots` page.

Usage

```
vec(object)

## S4 method for signature 'Projection'
vec(object)

bounds(object)

## S4 method for signature 'Projection'
bounds(object)

mat(object, ...)

## S4 method for signature 'Projection'
```

```
mat(object, return = "simple")

Aseq(object)

## S4 method for signature 'Projection'
Aseq(object)

projtype(object)

## S4 method for signature 'Projection'
projtype(object)

vectype(object)

## S4 method for signature 'Projection'
vectype(object)

nproj(object)

## S4 method for signature 'Projection'
nproj(object)

nmat(object)

## S4 method for signature 'Projection'
nmat(object)

ntime(object)

## S4 method for signature 'Projection'
ntime(object)

show(object)

## S4 method for signature 'Projection'
show(object)
```

Arguments

object	an object of class "Projection" generated using project
...	further arguments (see method, below)
return	either "simple", "list", or "array": used for accessing the 'mat' slot from a Projection object. Note that only list or array can be used for stochastic projections, which have more than one matrix.

Details

In addition to the examples below, see the "Deterministic population dynamics" and "Stochastic population dynamics" vignettes for worked examples that use the 'Projection' objects.

Slots

`.Data` One or more time series of population sizes. 'Projection' objects inherit from a standard array, and can be treated as such. Therefore, if `vector` is specified, the 'Projection' object will behave as:

- if a single `vector` is given, a numeric vector of population sizes of length `time+1`
- if multiple `vectors` are given, a numeric matrix of population projections where each column represents a single population projection and is of length `time+1`
- if `vector="n"`, a numeric matrix of population projections where each column represents a single stage-biased projection and is of length `time+1`.
- if `vector="diri"`, a numeric matrix of population projections where each column represents projection of a single vector draw and each column is of length `time+1`.

`vec` Age- or stage-based population vectors. `vec` will be:

- If a single `vector` is specified, a numeric matrix of demographic vectors from projection of `vector` through `A`. Each column represents the densities of one life (st)age in the projection.
- If multiple `vectors` are specified, a three-dimensional array of demographic vectors from projection of the set of initial vectors through `A`. The first dimension represents time (and is therefore equal to `time+1`). The second dimension represents the densities of each stage (and is therefore equal to the dimension of `A`). The third dimension represents each individual projection (and is therefore equal to the number of initial vectors given).
- If `vector="n"`, a three-dimensional array of demographic vectors from projection of the set of stage-biased vectors through `A`. The first dimension represents time (and is therefore equal to `time+1`). The second dimension represents the densities of each stage (and is therefore equal to the dimension of `A`). The third dimension represents each individual stage-biased projection (and is therefore also equal to the dimension of `A`).
- If`vector="diri"`, a three-dimensional array of demographic vectors from projection of the dirichlet vector draws projected through `A`. The first dimension represents time (and is therefore equal to `time+1`). The second dimension represents the densities of each stage (and is therefore equal to the dimension of `A`). The third dimension represents projection of each population draw (and is therefore equal to `draws`).

Some examples for understanding the structure of 3D arrays returned when `return.vec=TRUE`: when projecting a 3 by 3 matrix for >10 time intervals, element [11,3,2] represents the density of stage 3 at time 10 for either vector 2 (multiple vectors), stage-bias 2 (`vector="n"`) or draw 2 (`vector="diri"`); note that because element 1 represents t=0, then t=10 is found at element 11. The vector [,3,2] represents the time series of densities of stage 3 in the projection of vector 2 / stage-bias 2 / draw 2. The matrix [,,2] represents the time series of all stages in the projection of vector 2 / stage-bias 2 / draw 2.

Note that the projections inherit the labelling from `A` and `vector`, if it exists. Both stage and vector names are taken from the COLUMN names of `A` and `vector` respectively. These may be useful for selecting from the `projection` object, and are used when labelling plots of Projection objects containing multiple population projections.

Set `return.vec = FALSE` when calling `project` to prevent population vectors from being saved: in this case, `vec` is equal to `numeric(0)`. This may be necessary when projecting large numbers of vectors, as is the case when `vector = "diri"`.

`bounds` The bounds on population dynamics (only for deterministic projections). These represent the maximum and minimum population sizes achieveable at each time interval of the projection. `bounds` is a matrix with 2 columns (lower and upper bounds, in that order), and the number of rows is equal to `time + 1`.

`mat` The matrix/matrices used in the population projection. In their raw form `mat` is always a three-dimensional array, where the third dimension is used to index the different matrices. However, by using the `mat()` accessor function below, it is possible to choose different ways of representing the matrices (matrix, list, array).

`Aseq` The sequence of matrices used in the projection. For deterministic projections (where there is only 1 matrix) this will always be `rep(1, time)`. For stochastic projections (with more than 1 matrix), if `Aseq` is given to `project` as a numeric or character vector then this slot will take that value. If a matrix describing a random markov process is passed, the `Aseq` slot will be a single random chain.

`projtype` The type of projection. Either "deterministic" (single matrix; time-invariant), or "stochastic" (multiple matrices; time-varying).

`vectype` The type of vector passed to `project`. May be "single" (one vector; one population projection), "multiple" (more than one vector; several population projections), "bias" (stage-biased vectors; `vector = "n"`), or "diri" (vectors drawn from the dirichlet distribution; `vector = "diri"`).

See Also

`project` `Projection-plots`

Examples

```
### USING PROJECTION OBJECTS

# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Project stage-biased dynamics of A over 70 intervals
( pr <- project(A, vector="n", time=70) )
plot(pr)

# Access other slots
vec(pr)  #time sequence of population vectors
bounds(pr)  #bounds on population dynamics
mat(pr)  #matrix used to create projection
Aseq(pr)  #sequence of matrices (more useful for stochastic projections)
projtype(pr)  #type of projection
vectype(pr)  #type of vector(s) initiating projection

# Extra information on the projection
nproj(pr)  #number of projections
nmat(pr)  #number of matrices (more usefulk for stochastic projections)
ntime(pr)  #number of time intervals

# Select the projection of stage 2 bias
pr[,2]
```

```
# Project stage-biased dynamics of standardised A over 30 intervals
( pr2 <- project(A, vector="n", time=30, standard.A=TRUE) )
plot(pr2)

#Select the projection of stage 2 bias
pr2[,2]

# Select the density of stage 3 in bias 2 at time 10
vec(pr2)[11,3,2]

# Select the time series of densities of stage 2 in bias 1
vec(pr2)[,2,1]

#Select the matrix of population vectors for bias 2
vec(pr2)[,,2]

# Create an initial stage structure
( initial <- c(1,3,2) )

# Project A over 50 intervals using a specified population structure
( pr3 <- project(A, vector=initial, time=50) )
plot(pr3)

# Project standardised dynamics of A over 10 intervals using
# standardised initial structure and return demographic vectors
( pr4 <- project(A, vector=initial, time=10, standard.vec=TRUE,
                 standard.A=TRUE, return.vec=TRUE) )
plot(pr4)

# Select the time series for stage 1
vec(pr4)[,1]
```

`Projection-plots` *Plot methods for 'Projection' objects*

Description

This page describes print and plot methods for `Projection-class`. Example code is below, or worked examples using these methods are available in the "Deterministic population dynamics" and "Stochastic population dynamics" vignettes.

Usage

```
plot(x, y, ...)

## S4 method for signature 'Projection,missing'
plot(
  x,
```

```
  y,
  bounds = FALSE,
  bounds.args = NULL,
  labs = TRUE,
  plottype = "lines",
  ybreaks = 20,
  shadelevels = 100,
  ...
)
```

Arguments

x	an object of class "Projection" generated using project
y	not used
...	arguments to be passed to methods: see par and plot.
bounds	logical: indicates whether to plot the bounds on population density.
bounds.args	A list of graphical parameters for plotting the bounds if bounds=T. The name of each list element indicates the name of the argument. Could include, e.g. list(lwd=2,lty=3,col="darkred").
labs	logical: if TRUE, the plot includes more than one projection and plottype="lines", then lines are automatically labelled according to the names contained in the 'projection' object.
plottype	for projections generated from dirichlet draws (see project), plottype has two options. "lines" will plot each projection as a separate line. "shady" will plot shaded contours showing the probabilities of population densities over over time, calculated across the set of projections from dirichlet draws. By default this shaded plot is a gradient of black to white (with black representing higher probabilities), but this can be overridden by using the 'col' argument (see examples).
ybreaks	if plottype="shady", gives the number of breaks on the y axis for generating the grid for the shade plot. A larger number of breaks means a finer resolution grid for the shading.
shadelevels	if plottype="shady" and a palette of colours is not specified using 'col', then shadelevels gives the number of colour/shading levels to use when generating the black and white shade plot. A larger number of levels means a finer resolution on the shade plot of population density (see examples).

Details

plot plot a Projection object

See Also

project Projection-class

Examples

```
### Desert tortoise matrix
data(Tort)

# Create an initial stage structure
Tortvec1 <- c(8, 7, 6, 5, 4, 3, 2, 1)

# Create a projection over 30 time intervals
( Tortp1 <- project(Tort, vector = Tortvec1, time = 10) )

# plot p1
plot(Tortp1)
plot(Tortp1, bounds = TRUE) #with bounds

# new display parameters
plot(Tortp1, bounds = TRUE, col = "red", bty = "n", log = "y",
     ylab = "Number of individuals (log scale)",
     bounds.args = list(lty = 2, lwd = 2) )

# multiple vectors
Tortvec2 <- cbind(Tortvec1, c(1, 2, 3, 4, 5, 6, 7, 8))
plot(project(Tort, vector = Tortvec2), log = "y")
plot(project(Tort, vector = Tortvec2), log = "y", labs = FALSE) #no labels

# dirichlet distribution
# darker shading indicates more likely population size
Tortshade <- project(Tort, time = 30, vector = "diri", standard.A = TRUE,
             draws = 500, alpha.draws = "unif")
plot(Tortshade, plottype = "shady", bounds = TRUE)

### STOCHASTIC PROJECTIONS
# load polar bear data
data(Pbear)

# project over 50 years with uniform matrix selection
Pbearvec <- c(0.106, 0.068, 0.106, 0.461, 0.151, 0.108)
p2 <- project(Pbear, Pbearvec, time = 50, Aseq = "unif")

# stochastic projection information
Aseq(p2)
projtype(p2)
nmat(p2)

# plot
plot(p2, log = "y")

### USING PROJECTION OBJECTS

# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Project stage-biased dynamics of A over 70 intervals
```

```
( pr <- project(A, vector="n", time=70) )
plot(pr)

# Access other slots
vec(pr)  #time sequence of population vectors
bounds(pr)  #bounds on population dynamics
mat(pr)  #matrix used to create projection
Aseq(pr)  #sequence of matrices (more useful for stochastic projections)
projtype(pr)  #type of projection
vectype(pr)  #type of vector(s) initiating projection

# Extra information on the projection
nproj(pr)  #number of projections
nmat(pr)  #number of matrices (more usefulk for stochastic projections)
ntime(pr)  #number of time intervals

# Select the projection of stage 2 bias
pr[,2]

# Project stage-biased dynamics of standardised A over 30 intervals
( pr2 <- project(A, vector="n", time=30, standard.A=TRUE) )
plot(pr2)

#Select the projection of stage 2 bias
pr2[,2]

# Select the density of stage 3 in bias 2 at time 10
vec(pr2)[11,3,2]

# Select the time series of densities of stage 2 in bias 1
vec(pr2)[,2,1]

#Select the matrix of population vectors for bias 2
vec(pr2)[,,2]

# Create an initial stage structure
( initial <- c(1,3,2) )

# Project A over 50 intervals using a specified population structure
( pr3 <- project(A, vector=initial, time=50) )
plot(pr3)

# Project standardised dynamics of A over 10 intervals using
# standardised initial structure and return demographic vectors
( pr4 <- project(A, vector=initial, time=10, standard.vec=TRUE,
                 standard.A=TRUE, return.vec=TRUE) )
plot(pr4)

# Select the time series for stage 1
vec(pr4)[,1]

### DETERMINISTIC PROJECTIONS
```

```
# Load the desert Tortoise matrix
data(Tort)

# Create an initial stage structure
Tortvec1 <- c(8, 7, 6, 5, 4, 3, 2, 1)

# Create a projection over 30 time intervals
( Tortp1 <- project(Tort, vector = Tortvec1, time = 10) )

# plot p1
plot(Tortp1)
plot(Tortp1, bounds = TRUE) #with bounds

# new display parameters
plot(Tortp1, bounds = TRUE, col = "red", bty = "n", log = "y",
     ylab = "Number of individuals (log scale)",
     bounds.args = list(lty = 2, lwd = 2) )

# multiple vectors
Tortvec2 <- cbind(Tortvec1, c(1, 2, 3, 4, 5, 6, 7, 8))
plot(project(Tort, vector = Tortvec2), log = "y")
plot(project(Tort, vector = Tortvec2), log = "y", labs = FALSE) #no labels

# dirichlet distribution
# darker shading indicates more likely population size
Tortshade <- project(Tort, time = 30, vector = "diri", standard.A = TRUE,
             draws = 500, alpha.draws = "unif")
plot(Tortshade, plottype = "shady", bounds = TRUE)

### STOCHASTIC PROJECTIONS
# load polar bear data
data(Pbear)

# project over 50 years with uniform matrix selection
Pbearvec <- c(0.106, 0.068, 0.106, 0.461, 0.151, 0.108)
p2 <- project(Pbear, Pbearvec, time = 50, Aseq = "unif")

# stochastic projection information
Aseq(p2)
projtype(p2)
nmat(p2)

# plot
plot(p2, log = "y")
```

`projectionD` *Calculate projection distance*

Description

Calculate projection distance for a population matrix projection model.

Usage

```
projectionD(A, vector)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
vector	a numeric vector or one-column matrix describing the age/stage distribution used to calculate the distance.

Details

projectionD (Haridas & Tuljapurkar 2007) is the difference between the reproductive value of a population with demographic distribution given by vector and the reproductive value of a population in stable state.

projectionD will not work for reducible matrices and returns a warning for imprimitive matrices (although will not function for imprimitive matrices with nonzero imaginary components in the dominant eigenpair).

Value

Projection distance.

References

- Haridas & Tuljapurkar (2007) Ecol. Lett., 10, 1143-1153.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.

See Also

Other DistanceMeasures: CohenD(), KeyfitzD()

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate projection distance
projectionD(A, vector=initial)
```

R2Matlab *Convert matrices into Matlab style strings*

Description

Convert R objects of class matrix into character strings that represent the matrix in a Matlab style

Usage

```
R2Matlab(A, noquote = FALSE)
```

Arguments

A	a numeric matrix of any dimension
noquote	(optional) if noquote=TRUE then the returned character vector is printed without quotes.

Details

Matlab reads matrices using a unique one-line notation that can prove useful for storage in databases and importing multiple matrices into a program at once, amongst other applications. This notation is by row, with "[" and "]" to specify the beginning and end of the matrix respectively, ";" to specify a new row and a space between each matrix element. Thus, the R matrix created using `matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3)` is equivalent to [0 1 2;0.5 0.1 0;0 0.6 0.6].

R2Matlab takes an R object of class matrix converts it into a Matlab-style character string that may be useful for exporting into databases.

Value

Object of class character representing A in a Matlab style.

See Also

Matlab2R

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Code the matrix in a Matlab style
R2Matlab(A)

# Print without quotes
R2Matlab(A, noquote=TRUE)
```

reac *Calculate reactivity and first-timestep attenuation*

Description

Calculate reactivity (first-timestep amplification) and first-timestep attenuation for a population matrix projection model.

Usage

```
reac(A, vector = "n", bound = NULL, return.N = FALSE)
```

Arguments

A	a square, non-negative numeric matrix of any dimension
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution used to calculate a 'case-specific' reactivity/ first-timestep attenuation
bound	(optional) specifies whether an upper or lower bound should be calculated (see details).
return.N	(optional) if TRUE, returns population size in the first time interval (including effects of asymptotic growth and initial population size), alongside standardised reactivity/first-timestep attenuation.

Details

reac returns a standardised measure of first-timestep amplification or attenuation, discounting the effects of both initial population size and asymoptotic growth (Stott et al. 2011).

If vector="n" then either bound="upper" or bound="lower" must be specified, which calculate the upper or lower bound on first-timestep amplification and attenuation (i.e. the largest and smallest values that reactivity and first-timestep attenuation may take) respectively. Specifying vector overrides calculation of a bound, and will yield a 'case-specific' reactivity/first-timestep attenuation.

If return.N=T then the function also returns realised population size (including the effects of asymptotic growth and initial population size).

reac works with imprimitive and irreducible matrices, but returns a warning in these cases.

NOTE: reac replaces reactivity and firststepatt as of version 1.0-0. Although semantically 'reactivity' and 'first-timestep attenuation' are different (the former is an amplification in the first timestep and the latter an attenuation in the first timestep), as a population matrix projection model EITHER amplifies OR attenuates in the first timestep, it made no sense to have two separate functions to calculate one thing (transient dynamics in the first timestep).

Value

If vector="n", the upper bound on reactivity of A if bound="upper" and the lower bound on first-timestep attenuation of A if bound="lower".
If vector is specified, the 'case-specific' reactivity or first-timestep attenuation of the model.

If return.N=TRUE, a list with components:

reac the bound on or case-specific reactivity or first-timestep attenuation

N the population size at the first timestep, including the effects of initial population size and asymptotic growth.

References

- Neubert & Caswell (1997) Ecology, 78, 653-665.
- Stott et al. (2011) Ecol. Lett., 14, 959-970.
- Townley & Hodgson (2008) J. Appl. Ecol., 45, 1836-1839.

See Also

Other TransientIndices: `Kreiss()`, `inertia()`, `maxamp()`, `maxatt()`

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create initial stage structures
( initial1 <- c(1,3,2) )
( initial2 <- c(3,1,1) )

# Calculate the upper bound on reactivity of A
reac(A, bound="upper")

# Calculate the lower bound on first-timestep attenuation of A
reac(A, bound="lower")

# Calculate case-specific reactivity of A
# when projected using specific demographic structure
# that amplifies
reac(A, vector=initial1)

# Calculate case-specific reactivity of A
# and initial1 and return realised population size
reac(A, vector=initial1, return.N=TRUE)

# Calculate case-specific first-timestep attenuation of A
# when projected using a specific demographic structure that
#attenuates
reac(A, vector=initial2)

# Calculate case-specific first-timestep attenuation of A
# and initial2 and return realised population size
reac(A, vector=initial2, return.N=TRUE)#'
```

sens *Calculate sensitivity matrix*

Description

Calculate the sensitivity matrix for a population matrix projection model using eigenvectors.

Usage

```
sens(A, eval = "max", all = FALSE)
```

Arguments

A	a square, non-negative numeric matrix of any dimension
eval	the eigenvalue to evaluate. Default is eval="max", which evaluates the dominant eigenvalue (the eigenvalue with largest REAL value: for imprimitive or reducible matrices this may not be the first eigenvalue). Otherwise, specifying e.g. eval=2 will evaluate sensitivity of the eigenvalue with second-largest modulus.
all	(optional) if FALSE, then only sensitivity values for observed transitions (nonzero entries in A) are returned.

Details

sens uses the eigenvectors of A to calculate the sensitivity matrix of the specified eigenvalue, see section 9.1 in Caswell (2001). Same method as sensitivity in popbio but can also evaluate subdominant eigenvalues.

Value

A numeric (real or complex) matrix of equal dimension to A.

References

- Caswell (2001) Matrix Population Models 2nd ed. Sinauer.

See Also

Other PerturbationAnalyses: elas(), tfa_inertia(), tfa_lambda(), tfam_inertia(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Examples

```
# Create a 3x3 PPM
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate sensitivities of dominant eigenvalue
sens(A)
# Calculate sensitivities of first subdominant eigenvalue,
# only for observed transitions
sens(A, eval=2, all=FALSE)
```

`stoch` *Project population dynamics*

Description

Analyse long-term dynamics of a stochastic population matrix projection model.

Usage

```
stoch(
  A,
  what = "all",
  Aseq = "unif",
  vector = NULL,
  Astart = NULL,
  iterations = 10000,
  discard = 1000,
  PREcheck = FALSE
)
```

Arguments

`A`	a list of matrices. `stoch` uses `project` to perform a stochastic' projection where the matrix varies with each timestep. The sequence of matrices is determined using `Aseq`. Matrices must be square, non-negative and numeric, and all matrices must have the same dimension.
`what`	Deprecated: later versions will always return all values of growth, log growth and variance in growth. This argument determines what should be returned. A character vector with possible entries "lambda" (to calcualate stochastic growth), "log_lambda" (to calcualate logarithm of stochastic growth), "var" (to calculate variance in log stochastic growth) and/or "all" (to calculate all values).
`Aseq`	the sequence of matrices in a stochastic projection. `Aseq` may be either: • "unif" (default), which results in every matrix in `A` having an equal, random chance of being chosen at each timestep.

- a square, nonnegative left-stochastic matrix describing a first-order markov chain used to choose the matrices. This should have the same dimension as the number of matrices in A.
- a numeric vector giving a specific sequence which corresponds to the matrices in A.
- a character vector giving a specific sequence which corresponds to the names of the matrices in A.

vector — (optional) a numeric vector describing the age/stage distribution used to calculate the projection. If vector is not specified, a random vector is generated. Long-term stochastic dynamics should usually be the same for any vector, although if all the matrices in A are reducible (see isIrreducible), that may not be the case.

Astart — (optional) in a stochastic projection, the matrix with which to initialise the projection (either numeric, corresponding to the matrices in A, or character, corresponding to the names of matrices in A). When Astart = NULL, a random initial matrix is chosen.

iterations — the number of projection intervals. The default is 1e+5.

discard — the number of initial projection intervals to discard, to discount near-term effects arising from the choice of vector. The default is 1e+3

PREcheck — many functions in popdemo first check Primitivity, Reducibility and/or Ergodicity of matrices, with associated warnings and/or errors if a matrix breaks any assumptions. Set PREcheck=FALSE if you want to bypass these checks.

Details

Calculates stochastic growth and its variance for a given stochastic population matrix projection model.

Value

A numeric vector with three possible elements: "lambda" (the geometric mean stochastic population growth rate) "log_lambda" (the arithmetic mean of the logarithm of the stochastic population growth rate) and "var" (the variance of the logarithm of the stochastic population growth rate). Values returned depend on what's passed to what.

Examples

```
# load the Polar bear data
( data(Pbear) )

# Find the stochastic info for a time series with uniform probability of each
# matrix
( all_unif <- stoch(Pbear, what = "all", Aseq = "unif") )

# Find the stochastic growth for a time series with uniform probability of each
# matrix
( lambda_unif <- stoch(Pbear, what = "lambda", Aseq = "unif") )
```

```
# Find the variance in stochastic growth for a time series with uniform
# probability of each matrix
( var_unif <- stoch(Pbear, what = "var", Aseq = "unif") )

# Find stochastic growth and its variance for a time series with a sequence of
# matrices where "bad" years happen with probability q
q <- 0.5
prob_seq <- c(rep(1-q,3)/3, rep(q,2)/2)
Pbear_seq <- matrix(rep(prob_seq,5), 5, 5)
( all_q <- stoch(Pbear, what = "all", Aseq = Pbear_seq) )
```

`tfam_inertia` *Transfer function Analysis*

Description

Transfer function analysis of inertia of a population matrix projection model for all matrix elements.

Usage

```
tfam_inertia(
  A,
  bound = NULL,
  vector = "n",
  elementtype = NULL,
  Flim = c(-1, 10),
  Plim = c(-1, 10),
  plength = 100,
  digits = 1e-10
)
```

Arguments

`A`	a square, primitive, nonnegative numeric matrix of any dimension
`bound`	(optional) specifies whether the transfer funciton of an upper or lower bound on inertia should be calculated (see details).
`vector`	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate the transfer function of a 'case-specific' inertia
`elementtype`	(optional) a character matrix of the same dimension as `A` describing the structure of `A`: `"P"` denotes elements bounded between 0 and 1, i.e. survival, growth, regression; `"F"` denotes elements not bounded at 1, i.e. fecundity, fission; `NA` denotes absent elements (see details).
`Flim`, `Plim`	the perturbation ranges for `"F"` and `"P"` elements, expressed as a proportion of their magnitude (see details).
`plength`	the desired length of the perturbation ranges.

digits specifies which values of lambda should be excluded from analysis to avoid a computationally singular system (see details).

Details

tfam_inertia calculates an array of transfer functions of population inertia. A separate transfer function for each nonzero element of A is calculated (each element perturbed independently of the others). The function is most useful for use with the S3 method plot.tfam to visualise how perturbations affect the life cycle transitions, and easily compare the (nonlinear) effect of perturbation to different transitions on the dominant eigenvalue.

The sizes of the perturbations are determined by elementtype, Flim, Plim and plength. elementtype gives the type of each element, specifying whether perturbations should be bounded at 1 (elementtype = "P") or not (elementtype = "F"). If elementtype is not directly specified, the function assigns its own types, with those in the first row attributed "F", and elsewhere in the matrix attributed "F" if the value of the element >1 and "P" if the value of the element is <=1. Flim and Plim determine the desired perturbation magnitude, expressed as a proportion of the magnitude of the elements of A, whilst plength determines the length of the perturbation vector. For example, if an "F" element is equal to 0.5, Flim=c(-1,10) and plength=100 then the perturbation to that element is seq(-1*0.5,10*0.5,100-1). The process is the same for "P" elements, except that these are truncated to a maximum value of 1 (growth/survival elements cannot be greater than 1). Both "F" and "P" elements are truncated to a minimum value of 0.

tfam_inertia uses tfa_inertia to calculate transfer functions. digits is passed to tfa_inertia to prevent the problem of singular matrices (see details in tfa_inertia).

tfam_inertia will not work for reducible matrices.

Value

A list containing numerical arrays:

p perturbation magnitudes

lambda dominant eigenvalues of perturbed matrices

inertia inertias of perturbed matrices

The first and second dimensions of the arrays are equivalent to the first and second dimensions of A. The third dimension of the arrays are the vectors returned by tfa_inertia. e.g. $inertia[3,2,] selects the inertia values for the transfer function of element [3,2] of the matrix.

References

- Stott et al. (2012) Methods Ecol. Evol., 3, 673-684.
- Hodgson et al. (2006) J. Theor. Biol., 70, 214-224.

See Also

S3 plotting method: plot.tfam

Other TransferFunctionAnalyses: tfa_inertia(), tfa_lambda(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Other PerturbationAnalyses: elas(), sens(), tfa_inertia(), tfa_lambda(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2))

# Calculate the matrix of transfer functions for the upper bound on
# inertia, using default arguments
( tfmat<-tfam_inertia(A,bound="upper") )

# Plot the transfer function using the S3 method (defaults to p
# and inertia in this case)
plot(tfmat)

# Plot inertia against lambda using the S3 method
plot(tfmat, xvar="lambda", yvar="inertia")

# Plot the transfer function of element [3,2] without the S3 method
par(mfrow=c(1,1))
par(mar=c(5,4,4,2)+0.1)
plot(tfmat$inertia[3,2,]~tfmat$p[3,2,],xlab="p",ylab="lambda",type="l")

# Create a new matrix with fission of adults
B <- A; B[2,3] <- 0.9; B

# Calculate the matrix of transfer functions for specified
# initial stage structure, using chosen arguments
# that give the exact structure of the new matrix
# and perturb a minimum of half the value of an element and
# a maximum of double the value of an element
( etype <- matrix(c(NA, "F", "F", "P", "P", "F", NA, "P", "P"),
                  ncol=3, byrow=TRUE) )
( tfmat2 <- tfam_inertia(B, vector=initial, elementtype=etype,
                             Flim=c(-0.5,2), Plim=c(-0.5,2)) )

# Plot the new matrix of transfer functions using the S3 method
plot(tfmat2)
```

`tfam_lambda` *Transfer function analysis*

Description

Transfer function analysis of the dominant eigenvalue of a population matrix projection model for all matrix elements.

Usage

```
tfam_lambda(
  A,
  elementtype = NULL,
  Flim = c(-1, 10),
  Plim = c(-1, 10),
  plength = 100,
  digits = 1e-10
)
```

Arguments

A	a square, irreducible, nonnegative numeric matrix of any dimension
elementtype	(optional) a character matrix of the same dimension as A describing the structure of A: "P" denotes elements bounded between 0 and 1, i.e. survival, growth, regression; "F" denotes elements not bounded at 1, i.e. fecundity, fission; NA denotes absent elements (see details).
Flim, Plim	the perturbation ranges for "F" and "P" elements, expressed as a proportion of their magnitude (see details).
plength	the desired length of the perturbation ranges.
digits	specifies which values of lambda should be excluded from analysis to avoid a computationally singular system (see details).

Details

tfam_lambda calculates an array of transfer functions of the dominant eigenvalue of A. A separate transfer function for each nonzero element of A is calculated (each element perturbed independently of the others). The function is most useful for use with the S3 method plot.tfam to visualise how perturbations affect the life cycle transitions, and easily compare the (nonlinear) effect of perturbation to different transitions on the dominant eigenvalue.

The sizes of the perturbations are determined by elementtype, Flim, Plim and plength. elementtype gives the type of each element, specifying whether perturbations should be bounded at 1 (elementtype = "P") or not (elementtype = "F"). If elementtype is not directly specified, the function assigns its own types, with those in the first row attributed "F", and elsewhere in the matrix attributed "F" if the value of the element >1 and "P" if the value of the element is <=1. Flim and Plim determine the desired perturbation magnitude, expressed as a proportion of the magnitude of the elements of A, whilst plength determines the length of the perturbation vector. For example, if an "F" element is equal to 0.5, Flim=c(-1,10) and plength=100 then the perturbation to that element is seq(-1*0.5,10*0.5,100-1). The process is the same for "P" elements, except that these are truncated to a maximum value of 1 (growth/survival elements cannot be greater than 1). Both "F" and "P" elements are truncated to a minimum value of 0.

tfam_lambda uses tfa_lambda to calculate transfer functions. digits is passed to tfa_lambda to prevent the problem of singular matrices (see details in tfa_lambda).

tfam_lambda will not work for reducible matrices.

Value

A list containing numerical arrays:

p perturbation magnitudes

lambda dominant eigenvalues of perturbed matrices

The first and second dimensions of the arrays are equivalent to the first and second dimensions of A. The third dimension of the arrays are the vectors returned by tfa_lambda. e.g. $lambda[3,2,] selects the lambda values for the transfer function of element [3,2] of the matrix.

References

- Townley & Hodgson (2004) J. Appl. Ecol., 41, 1155-1161.
- Hodgson et al. (2006) J. Theor. Biol., 70, 214-224.

See Also

S3 plotting method: plot.tfa

Other TransferFunctionAnalyses: tfa_inertia(), tfa_lambda(), tfam_inertia(), tfs_inertia(), tfs_lambda()

Other PerturbationAnalyses: elas(), sens(), tfa_inertia(), tfa_lambda(), tfam_inertia(), tfs_inertia(), tfs_lambda()

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the matrix of transfer functions using default arguments
( tfmat<-tfam_lambda(A) )

# Plot the result using the S3 method
plot(tfmat)

# Plot the transfer function of element [3,2] without using the S3 method
par(mfrow=c(1,1))
par(mar=c(5,4,4,2)+0.1)
plot(tfmat$lambda[3,2,]~tfmat$p[3,2,],xlab="p",ylab="lambda",type="l")

# Create a new matrix with fission of adults
B <- A; B[2,3] <- 0.9; B

# Calculate the matrix of transfer functions using chosen arguments
# that give the exact structure of the new matrix
# and perturb a minimum of half the value of an element and
# a maximum of double the value of an element
( etype <- matrix(c(NA, "F", "F", "P", "P", "F", NA, "P", "P"),
                  ncol=3, byrow=TRUE) )
( tfmat2 <- tfam_lambda(B, elementtype=etype, Flim=c(-0.5,2),
                        Plim=c(-0.5,2)) )
```

```
# Plot the new matrix of transfer functions using the S3 method
plot(tfmat2)
```

`tfa_inertia` *Transfer function Analysis*

Description

Transfer function analysis of inertia of a population matrix projection model using a specified perturbation structure.

Usage

```
tfa_inertia(A, d, e, vector = "n", bound = NULL, prange, digits = 1e-10)
```

Arguments

`A`	a square, primitive, nonnegative numeric matrix of any dimension
`d, e`	numeric vectors that determine the perturbation structure (see details).
`vector`	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate the transfer function of a 'case-specific' inertia
`bound`	(optional) specifies whether the transfer funciton of an upper or lower bound on inertia should be calculated (see details).
`prange`	a numeric vector giving the range of perturbation magnitude (see details)
`digits`	specifies which values of lambda should be excluded from analysis to avoid a computationally singular system (see details).

Details

`tfa_inertia` calculates the transfer function of inertia of a population matrix projection model given a perturbation structure (specified using `d` and `e`), and a range of desired perturbation magnitude (`prange`). Currently `tfa_inertia` can only work with rank-one, single-parameter perturbations (see Hodgson & Townley 2006).

If `vector="n"` then either `bound="upper"` or `bound="lower"` must be specified, which calculate the transfer function of the upper or lower bound on population inertia (i.e. the largest and smallest values that inertia may take) respectively. Specifying `vector` overrides calculation of a bound, and will yield a transfer function of a 'case-specific' inertia.

The perturbation structure is determined by `d%*%t(e)`. Therefore, the rows to be perturbed are determined by `d` and the columns to be perturbed are determined by `e`. The specific values in `d` and `e` will determine the relative perturbation magnitude. So for example, if only entry [3,2] of a 3 by 3 matrix is to be perturbed, then `d = c(0,0,1)` and `e = c(0,1,0)`. If entries [3,2] and [3,3] are to be perturbed with the magnitude of perturbation to [3,2] half that of [3,3] then `d = c(0,0,1)`

and e = c(0,0.5,1). d and e may also be expressed as numeric one-column matrices, e.g. d = matrix(c(0,0,1), ncol=1), e = matrix(c(0,0.5,1), ncol=1). See Hodgson et al. (2006) for more information on perturbation structures.

The perturbation magnitude is determined by prange, a numeric vector that gives the continuous range of perterbation magnitude to evaluate over. This is usually a sequence (e.g. prange=seq(-0.1,0.1,0.001)), but single transfer functions can be calculated using a single perturbation magnitude (e.g. prange=-0.1). Because of the nature of the equation used to calculate the transfer function, prange is used to find a range of lambda from which the perturbation magnitudes are back-calculated, and matched to their orresponding inertia, so the output perturbation magnitude p will match prange in length and range but not in numerical value (see Stott et al. 2012 for more information).

tfa_inertia uses the resolvent matrix in its calculation, which cannot be computed if any lambda in the equation are equal to the dominant eigenvalue of A. digits specifies the values of lambda that should be excluded in order to avoid a computationally singular system. Any values equal to the dominant eigenvalue of A rounded to an accuracy of digits are excluded. digits should only need to be changed when the system is found to be computationally singular, in which case increasing digits should help to solve the problem.

tfa_inertia will not work for reducible matrices.

There is an S3 plotting method available (see plot.tfa and examples below)

Value

A list containing numerical vectors:

p perturbation magnitudes

lambda dominant eigenvalues of perturbed matrices

inertia inertias of perturbed matrices

(Note that p will not be exactly the same as prange when prange is specified, as the code calculates p for a given lambda rather than the other way around, with prange only used to determine max, min and number of lambda values to evaluate.)

References

- Stott et al. (2012) Methods Ecol. Evol., 3, 673-684.
- Hodgson et al. (2006) J. Theor. Biol., 70, 214-224.

See Also

S3 plotting method: plot.tfa

Other TransferFunctionAnalyses: tfa_lambda(), tfam_inertia(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Other PerturbationAnalyses: elas(), sens(), tfa_lambda(), tfam_inertia(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the transfer function of upper bound on inertia
# given a perturbation to A[3,2]
( transfer<-tfa_inertia(A, d=c(0,0,1), e=c(0,1,0), bound="upper",
                        prange=seq(-0.6,0.4,0.01)) )

# Plot the transfer function using the S3 method (defaults to p
# and inertia in this case)
plot(transfer)

# Plot inertia against lambda using the S3 method
plot(transfer, xvar="lambda", yvar="inertia")

# Calculate the transfer function of case-specific inertia
# given perturbation to A[3,2] and A[3,3] with perturbation
# to A[3,2] half that of A[3,3]
( transfer2<-tfa_inertia(A, d=c(0,0,1), e=c(0,0.5,1), vector=initial,
                         prange=seq(-0.6,0.4,0.01)) )

# Plot inertia against p using the S3 method
plot(transfer2)

# Plot inertia against lambda without using the S3 method
plot(transfer$inertia~transfer$lambda,type="l",
     xlab=expression(lambda),ylab="inertia")
```

`tfa_lambda` *Transfer function analysis*

Description

Transfer function analysis of the dominant eigenvalue of a population matrix projection model using a specified perturbation structure.

Usage

```
tfa_lambda(A, d, e, prange = NULL, lambdarange = NULL, digits = 1e-10)
```

Arguments

`A`	a square, irreducible, nonnegative numeric matrix of any dimension
`d, e`	numeric vectors that determine the perturbation structure (see details).

prange	a numeric vector giving the range of perturbation magnitude (see details)
lambdarange	a numeric vector giving the range of lambda values (asymptotic growth rates) to be achieved (see details).
digits	specifies which values of lambda should be excluded from analysis to avoid a computationally singular system (see details).

Details

tfa_lambda calculates the transfer function of the dominant eigenvalue of a matrix (A), given a perturbation structure (specified using d and e), and either a range of target values for asymptotic population growth (lambdavalues) or a range of desired perturbation magnitude (prange). Currently tfa_lambda can only work with rank- one, single-parameter perturbations (see Hodgson & Townley 2004).

The perturbation structure is determined by d%*%t(e). Therefore, the rows to be perturbed are determined by d and the columns to be perturbed are determined by e. The specific values in d and e will determine the relative perturbation magnitude. So for example, if only entry [3,2] of a 3 by 3 matrix is to be perturbed, then d = c(0,0,1) and e = c(0,1,0). If entries [3,2] and [3,3] are to be perturbed with the magnitude of perturbation to [3,2] half that of [3,3] then d = c(0,0,1) and e = c(0,0.5,1). d and e may also be expressed as numeric one-column matrices, e.g. d = matrix(c(0,0,1), ncol=1), e = matrix(c(0,0.5,1), ncol=1). See Hodgson et al. (2006) for more information on perturbation structures.

The perturbation magnitude is determined by prange, a numeric vector that gives the continuous range of perterbation magnitude to evaluate over. This is usually a sequence (e.g. prange=seq(-0.1,0.1,0.001)), but single transfer functions can be calculated using a single perturbation magnitude (e.g. prange=-0.1). Because of the nature of the equation, prange is used to find a range of lambda from which the perturbation magnitudes are back-calculated, so the output perturbation magnitude p will match prange in length and range but not in numerical value (see value). Alternatively, a vector lambdarange can be specified, representing a range of target lambda values from which the corresponding perturbation values will be calculated. Only one of either prange or lambdarange may be specified.

tfa_lambda uses the resolvent matrix in its calculation, which cannot be computed if any lambda are equal to the dominant eigenvalue of A. digits specifies the values of lambda that should be excluded in order to avoid a computationally singular system. Any values equal to the dominant eigenvalue of A rounded to an accuracy of digits are excluded. digits should only need to be changed when the system is found to be computationally singular, in which case increasing digits should help to solve the problem.

tfa_lambda will not work for reducible matrices.

There is an S3 plotting method available (see plot.tfa and examples below)

Value

A list containing numerical vectors:

p the perturbation magnitude(s).

lambda the dominant eigenvalue(s) of the perturbed matrix(matrices).

(Note that p will not be exactly the same as prange when prange is specified, as the code calculates p for a given lambda rather than the other way around, with prange only used to determine max, min and number of lambda values to evaluate.)

References

- Townley & Hodgson (2004) J. Appl. Ecol., 41, 1155-1161.
- Hodgson et al. (2006) J. Theor. Biol., 70, 214-224.

See Also

S3 plotting method: plot.tfa

Other TransferFunctionAnalyses: tfa_inertia(), tfam_inertia(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Other PerturbationAnalyses: elas(), sens(), tfa_inertia(), tfam_inertia(), tfam_lambda(), tfs_inertia(), tfs_lambda()

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the transfer function of A[3,2] given a range of lambda
evals <- eigen(A)$values
lmax <- which.max(Re(evals))
lambda <- Re(evals[lmax])
lambdarange <- seq(lambda-0.1, lambda+0.1, 0.01)
( transfer <- tfa_lambda(A, d=c(0,0,1), e=c(0,1,0), lambdarange=lambdarange) )

# Plot the transfer function using the S3 method
plot(transfer)

# Calculate the transfer function of perturbation to A[3,2] and A[3,3]
# with perturbation to A[3,2] half that of A[3,3], given a range of
# perturbation values
p<-seq(-0.6,0.4,0.01)
( transfer2 <- tfa_lambda(A, d=c(0,0,1), e=c(0,0.5,1), prange=p) )

# Plot p and lambda without using the S3 method
plot(transfer$lambda~transfer$p, type="l", xlab="p", ylab=expression(lambda))
```

---

tfs_inertia *Calculate sensitivity of inertia using transfer functions*

Description

Calculate the sensitivity of population inertia of a population matrix projection model using differentiation of the transfer function.

Usage

```
tfs_inertia(A, d=NULL, e=NULL, vector="n", bound=NULL, startval=0.001,
            tolerance=1e-10,return.fit=FALSE,plot.fit=FALSE)
tfsm_inertia(A,vector="n",bound=NULL,startval=0.001,tolerance=1e-10)
```

Arguments

A	a square, primitive, nonnegative numeric matrix of any dimension
d, e	numeric vectors that determine the perturbation structure (see details).
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution ('demographic structure') used to calculate the transfer function of a 'case-specific' inertia
bound	(optional) specifies whether the transfer funciton of an upper or lower bound on inertia should be calculated (see details).
startval	tfs_inertia calculates the limit of the derivative of the transfer function as lambda of the perturbed matrix approaches the dominant eigenvalue of A (see details). startval provides a starting value for the algorithm: the smaller startval is, the quicker the algorithm should converge.
tolerance	the tolerance level for determining convergence (see details).
return.fit	if TRUE (and only if d and e are specified), the lambda and sensitivity values obtained from the convergence algorithm are returned alongside the sensitivity at the limit.
plot.fit	if TRUE then convergence of the algorithm is plotted as sensitivity~lambda.

Details

tfs_inertia and tfsm_inertia differentiate a transfer function to find sensitivity of population inertia to perturbations.

tfs_inertia evaluates the transfer function of a specific perturbation structure. The perturbation structure is determined by d%*%t(e). Therefore, the rows to be perturbed are determined by d and the columns to be perturbed are determined by e. The values in d and e determine the relative perturbation magnitude. For example, if only entry [3,2] of a 3 by 3 matrix is to be perturbed, then d = c(0,0,1) and e = c(0,1,0). If entries [3,2] and [3,3] are to be perturbed with the magnitude of perturbation to [3,2] half that of [3,3] then d = c(0,0,1) and e = c(0,0.5,1). d and e may also be expressed as numeric one-column matrices, e.g. d = matrix(c(0,0,1), ncol=1), e = matrix(c(0,0.5,1), ncol=1). See Hodgson et al. (2006) for more information on perturbation structures.

tfsm_inertia returns a matrix of sensitivity values for observed transitions (similar to that obtained when using sens to evaluate sensitivity of asymptotic growth), where a separate transfer function for each nonzero element of A is calculated (each element perturbed independently of the others).

The formula used by tfs_inertia and tfsm_inertia cannot be evaluated at lambda-max, therefore it is necessary to find the limit of the formula as lambda approaches lambda-max. This is done using a bisection method, starting at a value of lambda-max + startval. startval should be small, to avoid the potential of false convergence. The algorithm continues until successive sensitivity calculations are within an accuracy of one another, determined by tolerance: a tolerance

of 1e-10 means that the sensitivity calculation should be accurate to 10 decimal places. However, as the limit approaches lambda-max, matrices are no longer invertible (singular): if matrices are found to be singular then `tolerance` should be relaxed and made larger.

For `tfs_inertia`, there is an extra option to return and/or plot the above fitting process using `return.fit=TRUE` and `plot.fit=TRUE` respectively.

Value

For `tfs_inertia`, the sensitivity of inertia (or its bound) to the specified perturbation structure. If `return.fit=TRUE` a list containing components:

sens the sensitivity of inertia (or its bound) to the specified perturbation structure

lambda.fit the lambda values obtained in the fitting process

sens.fit the sensitivity values obtained in the fitting process.
For `tfsm_inertia`, a matrix containing sensitivity of inertia (or its bound) to each separate element of `A`.

References

- Stott et al. (2012) Methods Ecol. Evol., 3, 673-684.

See Also

Other TransferFunctionAnalyses: `tfa_inertia()`, `tfa_lambda()`, `tfam_inertia()`, `tfam_lambda()`, `tfs_lambda()`

Other PerturbationAnalyses: `elas()`, `sens()`, `tfa_inertia()`, `tfa_lambda()`, `tfam_inertia()`, `tfam_lambda()`, `tfs_lambda()`

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the sensitivity matrix for the upper bound on inertia
tfsm_inertia(A, bound="upper",tolerance=1e-7)

# Calculate the sensitivity of simultaneous perturbation to
# A[1,2] and A[1,3] for specified initial stage structure
# and return and plot the fitting process
tfs_inertia(A, d=c(1,0,0), e=c(0,1,1), vector=initial,tolerance=1e-7,
            return.fit=TRUE,plot.fit=TRUE)
```

`tfs_lambda` *Calculate sensitivity using transfer functions*

Description

Calculate the sensitivity of the dominant eigenvalue of a population matrix projection model using differentiation of the transfer function.

Usage

```
tfs_lambda(A, d=NULL, e=NULL, startval=0.001, tolerance=1e-10,
           return.fit=FALSE, plot.fit=FALSE)
tfsm_lambda(A, startval=0.001, tolerance=1e-10)
```

Arguments

`A`	a square, nonnegative numeric matrix of any dimension.
`d`, `e`	numeric vectors that determine the perturbation structure (see details).
`startval`	`tfs_lambda` calculates the limit of the derivative of the transfer function as lambda of the perturbed matrix approaches the dominant eigenvalue of `A` (see details). `startval` provides a starting value for the algorithm: the smaller `startval` is, the quicker the algorithm should converge.
`tolerance`	the tolerance level for determining convergence (see details).
`return.fit`	if `TRUE` the lambda and sensitivity values obtained from the convergence algorithm are returned alongside the sensitivity at the limit.
`plot.fit`	if `TRUE` then convergence of the algorithm is plotted as sensitivity~lambda.

Details

`tfs_lambda` and `tfsm_lambda` differentiate a transfer function to find sensitivity of the dominant eigenvalue of `A` to perturbations. This provides an alternative method to using matrix eigenvectors to calculate the sensitivity matrix and is useful as it may incorporate a greater diversity of perturbation structures.

`tfs_lambda` evaluates the transfer function of a specific perturbation structure. The perturbation structure is determined by `d%*%t(e)`. Therefore, the rows to be perturbed are determined by `d` and the columns to be perturbed are determined by `e`. The values in `d` and `e` determine the relative perturbation magnitude. For example, if only entry [3,2] of a 3 by 3 matrix is to be perturbed, then `d = c(0,0,1)` and `e = c(0,1,0)`. If entries [3,2] and [3,3] are to be perturbed with the magnitude of perturbation to [3,2] half that of [3,3] then `d = c(0,0,1)` and `e = c(0,0.5,1)`. `d` and `e` may also be expressed as numeric one-column matrices, e.g. `d = matrix(c(0,0,1), ncol=1)`, `e = matrix(c(0,0.5,1), ncol=1)`. See Hodgson et al. (2006) for more information on perturbation structures.

`tfsm_lambda` returns a matrix of sensitivity values for observed transitions (similar to that obtained when using `sens` to evaluate sensitivity using eigenvectors), where a separate transfer function for each nonzero element of `A` is calculated (each element perturbed independently of the others).

The formula used by `tfs_lambda` and `tfsm_lambda` cannot be evaluated at lambda-max, therefore it is necessary to find the limit of the formula as lambda approaches lambda-max. This is done using a bisection method, starting at a value of lambda-max + `startval`. `startval` should be small, to avoid the potential of false convergence. The algorithm continues until successive sensitivity calculations are within an accuracy of one another, determined by `tolerance`: a `tolerance` of 1e-10 means that the sensitivity calculation should be accurate to 10 decimal places. However, as the limit approaches lambda-max, matrices are no longer invertible (singular): if matrices are found to be singular then `tolerance` should be relaxed and made larger.

For `tfs_lambda`, there is an extra option to return and/or plot the above fitting process using `return.fit=TRUE` and `plot.fit=TRUE` respectively.

Value

For `tfs_lambda`, the sensitivity of lambda-max to the specified perturbation structure. If `return.fit=TRUE` a list containing components:

sens the sensitivity of lambda-max to the specified perturbation structure

lambda.fit the lambda values obtained in the fitting process

sens.fit the sensitivity values obtained in the fitting process.
For `tfsm_lambda`, a matrix containing sensitivity of lambda-max to each element of `A`.

References

- Hodgson et al. (2006) J. Theor. Biol., 70, 214-224.

See Also

Other TransferFunctionAnalyses: `tfa_inertia()`, `tfa_lambda()`, `tfam_inertia()`, `tfam_lambda()`, `tfs_inertia()`

Other PerturbationAnalyses: `elas()`, `sens()`, `tfa_inertia()`, `tfa_lambda()`, `tfam_inertia()`, `tfam_lambda()`, `tfs_inertia()`

Examples

```
# Create a 3x3 matrix
( A <- matrix(c(0,1,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Calculate the sensitivity matrix
tfsm_lambda(A)

# Calculate the sensitivity of simultaneous perturbation to
# A[1,2] and A[1,3]
tfs_lambda(A, d=c(1,0,0), e=c(0,1,1))

# Calculate the sensitivity of simultaneous perturbation to
# A[1,2] and A[1,3] and return and plot the fitting process
tfs_lambda(A, d=c(1,0,0), e=c(0,1,1),
           return.fit=TRUE, plot.fit=TRUE)
```

Tort *Desert tortoise matrix*

Description

Matrix Projection Model for the desert tortoise *Gopherus agassizii* with medium fecundity. The matrix is based on a population in the Western Mojave desert. Stages are based on age and size (carapace length in mm):
Stage 1: Yearling (age 0-1)
Stage 2: Juvenile 1 (<60 mm)
Stage 3: Juvenile 2 (90-99mm)
Stage 4: Immature 1 (100-139mm)
Stage 5: Immature 2 (140-179mm)
Stage 6: Subadult (180-207mm)
Stage 7: Adult 1 (208-239mm)
Stage 8: Adult 2 (>240mm).

Usage

```
data(Tort)
```

Format

Object of class `matrix`

References

- Doak et al. (1994) Ecol. Appl., 4, 446-460.

Examples

```
# read in data and view
data(Tort)
Tort
```

truelambda *Calculate asymptotic growth*

Description

Calculate the true asymptotic growth of a population matrix projection model from the model projection

Usage

```
truelambda(A, vector = "n", accuracy = 1e-07, iterations = 1e+05)
```

Arguments

A	a square, non-negative numeric matrix of any dimension
vector	(optional) a numeric vector or one-column matrix describing the age/stage distribution used to calculate the projection.
accuracy	the accuracy with which to determine convergence on asymptotic growth, expressed as a proportion (see details).
iterations	the maximum number of iterations of the model before the code breaks. For slowly-converging models and/or high specified convergence accuracy, this may need to be increased.

Details

truelambda works by simulating the given model and manually determining growth when convergence to the given accuracy is reached. Convergence on an asymptotic growth is deemed to have been reached when the growth of the model stays within the window determined by accuracy for 10*s iterations of the model, with s equal to the dimension of A. For example, projection of an 8 by 8 matrix with convergence accuracy of 1e-2 is deemed to have converged on asymptotic growth when 10*8=80 consecutive iterations of the model have a growth within 1-1e-2=0.99 (i.e. 99%) and 1+1e-2=1.01 (i.e. 101%) of each other.

If vector is specified, then the asymptotic growth of the projection of vector through A is returned. If vector="n" then asymptotic growths of the set of 'stage-biased' vectors are calculated. These projections are achieved using a set of standard basis vectors equal in number to the dimension of A. These have every element equal to 0, except for a single element equal to 1, i.e. for a matrix of dimension 3, the set of stage-biased vectors are: c(1,0,0), c(0,1,0) and c(0,0,1).

Asymptotic growth should be equal to the dominant eigenvalue of the matrix. For non-ergodic models this may not be the case: asymptotic growth will depend on the population structure that's projected. truelambda provides a means to check what the true asymptotic growth of a non-ergodic model is.

Value

If vector is specified, a numeric vector of length 2 giving the range in which asymptoticgrowth of the model lies.

If vector is not specified, a 2-column matrix with each row giving the range in which asymptotic growth lies for its corresponding stage-biased projection: the number of rows is equal to the dimension of A; the first row is the range when projecting [1,0,0,...], the second entry is the range when projecting [0,1,0,...], etc.

References

- Stott et al. (2010) Methods Ecol. Evol., 1, 242-252.

See Also

Other ConvergenceMeasures: convt(), dr()

Examples

```
# Create a 3x3 irreducible PPM
( A <- matrix(c(0,0,2,0.5,0.1,0,0,0.6,0.6), byrow=TRUE, ncol=3) )

# Create an initial stage structure
( initial <- c(1,3,2) )

# Calculate the true asymptotic growth of the stage-biased
# projections of A
truelambda(A)

# Calculate the true asymptotic growth of the projection of
# A and initial
truelambda(A, vector=initial)

# Create a 3x3 reducible, nonergodic PPM
B<-A; B[3,2] <- 0; B

# Calculate the true asymptotic growth of the 3 stage-biased
# projections of B
truelambda(B)
```

Index