Package 'popdemo'

May 18, 2016

Version 0.2-3

Date 2016-05-18

Title Demographic Modelling Using Projection Matrices

Author Iain Stott <iainmstott@gmail.com>, Dave Hodgson <D.J.Hodgson@exeter.ac.uk>, Stuart Townley <S.B.Townley@exeter.ac.uk>

Maintainer Iain Stott <iainmstott@gmail.com>

Depends R (>= 3.3.0)

Imports expm

Description Tools for modelling populations and demography using matrix projection models (MPMs). Designed to build on similar tools already available in 'popbio'. Specific foci are on indices of transient dynamics and use of control theory approaches, but 'popdemo' may also be useful for other implementations of MPMs, or matrix models in a more general sense.

License GPL (>= 2)

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2016-05-18 17:29:29

R topics documented:

`popdemo-package` *Demographic Modelling Using Projection Matrices*

Description

This package provides a set of tools designed for modelling population demography using population projection matrix (PPM) models. The focus is mainly on indices of transient dynamics, and the use of control theory (specifically transfer functions). However, `popdemo` includes a number of tools focussing on other areas of demographic modelling. Some tools may prove useful to anyone working with projection matrix models of any type, or indeed matrices in general.

The package is designed to build on those tools already available in `popbio`.

Details

Package:	popdemo
Type:	Package
Version:	1.0
Date:	2011-05-12
License:	GPL-2
LazyLoad:	yes

Selected functions (see bottom of page for a link to the full index):

Functions useful for working with matrices:
popdemo contains a number of tools to ease working with matrices (and specifically PPMs) in R. 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.
is.matrix_primitive, is.matrix_irreducible and is.matrix_ergodic facilitate diagnosis of matrix properties pertaining to ergodicity.

Population projection:
popdemo provides a simple means of projecting and plotting PPM models. project will project population dynamics and a plotting method is available via plot.projection.

Indices of transient density:
Transient dynamics are important to study in PPM models, and popdemo provides the means to work with transient indices of PPM models. reactivity, firststepatt measure immediate transient density of a population (within the first time step). maxamp and maxatt are near-term indices that measure the largest and smallest transient dynamics a population may exhibit overall. inertia measures asymptotic population density relative to stable state, and has many perturbation methods in the package (see below). All indices can be calculated using specific population structures, as well as bounds on population size.

Perturbation analysis:
Perturbation analysis is a key part of population studies. popdemo provides methods for nonlinear perturbation analysis of both asymptotic dynamics, using tfa and tfamatrix, and transient dynamics, using inertia.tfa and inertia.tfamatrix. These all have associated plotting methods linked to them: see plot.tfa and plot.tfamatrix). Sensitivity analyses are also available using transfer function methods: see tfsens, tfsensmatrix, inertia.tfsens and inertia.tfsensmatrix. (Traditional perturbation analyses are also available: sens, elas).

Indices of 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 Keyfitz.delta, projection.distance and Cohen.cumulative). There is also a means of calculating convergence time through simulation: convergence.time.

Author(s)

Iain Stott <iainmstott@gmail.com>, Dave Hodgson <D.J.Hodgson@exeter.ac.uk>, Stuart Townley <S.B.Townley@exeter.ac.uk> Maintainer: Iain Stott <iainmstott@gmail.com>

`blockmatrix` *Block-permute a reducible matrix*

Description

Block-permute a reducible matrix

Usage

```
blockmatrix(A)
```

Arguments

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

Details

Any reducible matrix may have its rows and columns simultaneously permuted so that it takes a specific structure, with irreducible square submatrices on the diagonal, zero submatrices in the lower triangle and nonzero submatrices in the upper triangle (see Caswell 2001; Stott et al. 2010). `blockmatrix` permutes 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.
order	the permutation of rows/columns of A in the block-permuted matrix.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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)
A

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

`Cohen.cumulative` *Calculate Cohen's cumulative distance*

Description

Calculate Cohen's cumulative distance metric for a given population projection matrix (PPM) model.

Usage

```
Cohen.cumulative(A, vector)
```

Arguments

`A`	a square, irreducible, non-negative numeric matrix of any dimension.
`vector`	a specified initial age/stage distribution of class vector or class matrix.

Details

Calculates the cumulative distance metric as outlined in Cohen (1979).

`Cohen.cumulative` 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

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

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

`convergence.time` *Calculate time to convergence*

Description

Calculate the time to convergence for a specified population projection matrix (PPM) model.

Usage

```
convergence.time(A, vector = "n", accuracy=1e-2, iterations=1e+5)
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate convergence time.
`accuracy`	The accuracy of convergence, 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

`convergence.time` works by simulating the model and manually determining when convergence to the given `accuracy` is reached. Convergence 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`. So for example, for an 8 by 8 matrix with dominant eigenvalue of 0.954 and convergence accuracy of 1e-2, growth of the model must remain between 0.954*(1-1e-2)=0.9445 and 0.954*(1+1e-2)=0.9635 for 10*8=80 iterations of the model for it to be considered 'converged'.

If `vector` is specified, then the convergence time of the projection of `vector` through `A` is returned. However, if `vector="n"` then convergence times of the set of stage-biased vectors is calculated. In practise, these projections are achieved using a set of standard basis vectors, each with every element equal to 0, except for a single element that is equal to 1 (i.e. for a 3 by 3 matrix, 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, `convergence.time` 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 the model.
If `vector` is not specified, a vector of convergence times for corresponding stage-biased projections (therefore, the length of the vector is equal to the dimension of `A`).

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

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

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

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

---

dr *Calculate damping ratio*

Description

Calculate the damping ratio of a given population projection matrix (PPM).

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 found 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

# Calculate damping ratio
dr(A)

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

`elas` *Calculate elasticity matrix*

Description

Calculate the elasticity matrix using eigenanalysis for a specified population projection matrix (PPM).

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 a reducible matrix 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).

Value

An elasticity matrix of equal dimension to A.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

sens

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)
A

# Calculate elasticities of dominant eigenvalue
elas(A)

# Calculate elasticities of first subdominant eigenvalue
elas(A, eval=2)
```

`firststepatt` *Calculate first-timestep attenuation*

Description

Calculate first-timestep attenuation for a specified population projection matrix (PPM) model.

Usage

```
firststepatt(A, vector="n", return.N=FALSE)
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate a case-specific first-timestep attenuation
`return.N`	(optional) if `TRUE`, returns population size in the first time interval (including effects of asymptotic growth and initial population size), alongside standardised first-timestep attenuation.

Details

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

If `vector` is not specified then the bound on first-timestep attenuation (the smallest first-timestep attenuation that may be achieved) is returned, otherwise a case-specific first-timestep attenuation for the specified matrix and demographic structure is calculated. Note that not all specified demographic structures will yield a first-timestep attenuation: if the model does not attenuate in the first timestep then an error is returned and `reactivity` should be used.

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

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

Value

If `vector="n"`, the bound on first-timestep attenuation of `A`.
If `vector` is specified, the case-specific first-timestep attenuation of the model.
If `return.N=TRUE`, a list with components:

`firststepatt`	the bound on or case-specific first-timestep attenuation
`N`	the population size at the first timestep, including the effects of initial population size and asymptotic growth.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

Other indices of transient density:
reactivity, maxamp, maxatt, inertia

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)
A

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

# Calculate the bound on first-timestep attenuation of A
firststepatt(A)

# Calculate case-specific first-timestep attenuation of A
# and initial
firststepatt(A, vector=initial)

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

---

inertia *Calculate population inertia*

Description

Calculate population inertia for a specified population projection matrix (PPM) model.

Usage

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

Arguments

A	a square, primitive, non-negative numeric matrix of any dimension
vector	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate a case-specific inertia
bound	(optional) specifies whether an upper or lower bound should be calculated (see details).
return.N	(optional) if TRUE, returns population size for 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.

Details

inertia returns a standardised measure of population inertia (Koons et al. 2007), so 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, so calculating the upper or lower bound on population inertia respectively (i.e. the largest and smallest values that inertia may take). 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

Other indices of transient density:
reactivity, firststepatt, maxamp, maxatt

Transfer function methods:
inertia.tfa, inertia.tfamatrix

Sensitivity methods: inertia.tfsens, inertia.tfsensmatrix

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)
A

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

# 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)
```

---

inertia.tfa *Analyse transfer function of population inertia*

Description

Analyse the transfer function of inertia of a given population projection matrix (PPM) model and perturbation structure.

Usage

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

Arguments

A	a square, primitive, nonnegative matrix of any dimension
d,e	numeric vectors that determine the perturbation structure (see details).
vector	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate a case-specific inertia
bound	(optional) specifies whether an upper or lower bound should be calculated (see details).
prange	a numeric vector giving the continuous range of perturbation magnitude.
digits	specifies which values of lambda should be excluded to avoid a computationally singular system (see details).

Details

inertia.tfa calculates the transfer function of inertia of a matrix given a perturbation structure and a range of desired perturbation magnitude. Currently inertia.tfa 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, so calculating the transfer function of the upper or lower bound on population inertia respectively. Specifying vector overrides calculation of a bound, and will yield a transfer function of 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 column vectors of class matrix, 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 in calculating the transfer function, prange is used to find a range of lambda from which the perturbation magnitudes are back-calculated and matched to their corresponding inertia, so that the output perturbation magnitude p will match prange in length and range but not in numerical value (see value and Hodgson & Townley 2004 for more information).

inertia.tfa uses the resolvent matrix in its calculation, which cannot be computed if any lambdarange 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. If not, then changing the range or step size of prange may help.

inertia.tfa will not work for imprimitive or 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.
lambda	the dominant eigenvalue of the perturbed matrix.
inertia	the inertia of the perturbed matrix model.

(Note that p will not usually be exactly the same as prange when 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.)

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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

Related:
inertia, inertia.tfamatrix

Sensitivity methods:
inertia.tfsens, inertia.tfsensmatrix

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)
A

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

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

```
transfer

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

# Plot lambda and inertia
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<-inertia.tfa(A, d=c(0,0,1), e=c(0,0.5,1), vector=initial,
                       prange=seq(-0.6,0.4,0.01))
transfer2

# Plot p and inertia
plot(transfer2)

# Plot lambda and inertia by hand
plot(transfer$inertia~transfer$lambda,type="l",
     xlab=expression(lambda),ylab="inertia")
```

`inertia.tfamatrix` *Analyse transfer function of inertia*

Description

Analyse the transfer function of inertia of a given population projection matrix (PPM) and perturbation structure.

Usage

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

Arguments

`A`	a square, irreducible, nonnegative matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate a case-specific inertia
`bound`	(optional) specifies whether an upper or lower bound should be calculated (see details).
`elementtype`	(optional) a matrix describing the structure of `A`: `"P"` denotes elements bounded at 1, i.e. survival, growth, regression, `"F"` denotes elements not bounded at 1, i.e. fecundity, fission, and `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 to avoid a computationally singular system (see details).

Details

inertia.tfamatrix uses inertia.tfa to calculate an array of transfer functions for each nonzero element in A. Each element is perturbed individually. The function is most useful for use with the S3 method plot.tfamatrix to visualise how perturbations affect the life cycle and easily compare the effect on different transitions. The default values for the function are enough to evaluate a complete range of perturbation, although these can be changed if so desired (see below).

The structure of the perturbations are determined by elementtype, Flim, Plim and plength. elementtype gives the nature of each transition, specifying whether perturbations should be bounded at 1 or not. If elementtype is not directly specified, then the function assigns its own types, with those in the first row attributed "F", and elsewhere in the matrix being attributed "F" if the element >1 and "P" if the element is <=1. Flim and Plim then determine the desired perturbation magnitude, expressed as a proportion of the magnitude of the elements, whilst plength determines the length of the perturbation vector. So, if an "F" element is equal to 0.5 and Flim=c(-1,10) and plength=100 then the perturbation to that element is seq(-1*0.5,10*0.5-0.5,100-1) = seq(-0.5,4.5,100). The maximum and minimum perturbed values of the element are thus c(0.5+(-1*0.5),0.5+(10*0.5-0.5)) = c(0,5). The process is the same for "P" elements, except that these are bounded to give a maximum perturbed value of 1 (as growth/survival elements cannot be greater than unity). Both "F" and "P" elements are bounded to give a minimum perturbed value of 0.

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

inertia.tfamatrix will not work for reducible matrices.

Value

A list containing numerical arrays:

p	the perturbation magnitude.
lambda	the dominant eigenvalue of the perturbed matrix.
inertia	the inertia of the perturbed matrix.

The first and second dimensions of the arrays are equivalent to the first and second dimensions of the matrix. The third dimension of the arrays are therefore the vectors returned by tfa. Therefore selecting e.g. $inertia[2,2,] will select the inertia values for element [2,2] of the matrix (for more informaiton see examples).

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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.tfamatrix

Related:
inertia, inertia.tfa

Sensitivity methods:
inertia.tfsens, inertia.tfsensmatrix

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)
A

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

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

# Plot the result
plot(tfmat)

# Plot inertia as a function of lambda
plot(tfmat, xvar="lambda", yvar="inertia")

# Plot the transfer function of element [3,2]
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)
etype
tfmat2 <- inertia.tfamatrix(B, vector=initial, elementtype=etype,
                            Flim=c(-0.5,2), Plim=c(-0.5,2))
tfmat2

# Plot the new matrix of transfer functions using default
# arguments
plot(tfmat2)
```

`inertia.tfsens` *Calculate sensitivity of inertia using transfer functions*

Description

Calculate the sensitivity of inertia to perturbations to a population projection matrix (PPM) model using a transfer function method.

Usage

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

Arguments

`A`	a square, primitive, nonnegative matrix of any dimension
`d,e`	numeric vectors that determine the perturbation structure (see details).
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate case-specific sensitvity of inertia.
`bound`	(optional) specifies whether an upper or lower bound should be calculated (see details).
`startval`	`inertia.tfsens` calculates the limit of the derivative of the transfer function as lambda 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 in the convergence algorithm are returned alongside the calculated sensitivity of inertia.
`plot.fit`	if `TRUE` then convergence of the algorithm is plotted as sensitivity~lambda.

Details

`inertia.tfsens` and `inertia.tfsensmatrix` differentiate a transfer function to find sensitivity of inertia to perturbation.

If `vector="n"` then either `bound="upper"` or `bound="lower"` must be specified, so calculating the sensitivity of the upper or lower bound on population inertia respectively. Specifying `vector` overrides calculation of a bound, and will yield sensitivity of case-specific inertia.

`inertia.tfsens` may be used to find sensitivity of a particular perturbation structure. A desired perturbation structure can be 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 column vectors of class matrix, 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.

`inertia.tfsensmatrix` returns a matrix of sensitivity values for observed transitions, where the sensitivity of each matrix element is evaluated seperately.

The formula used by `inertia.tfsens` and `inertia.tfsensmatrix` 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 become uninvertible (singular): if matrices are found to be singular then `tolerance` should be relaxed and made larger.

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

`sens`	the sensitivity of population 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 `inertia.tfsensmatrix`, a matrix containing sensitivity of inertia to each seperate element of `A`.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

Related:
inertia

Transfer function methods:
inertia.tfa, inertia.tfamatrix

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)
A

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

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

# Calculate the sensitivity of simultaneous perturbation to
# A[1,3] and A[2,3] for the lower bound on inertia
inertia.tfsens(A, d=c(1,0,0), e=c(0,1,1), bound="lower")

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

---

is.matrix_ergodic *Determine ergodicity of a matrix*

Description

Determine whether a matrix is ergodic or nonergodic

Usage

```
is.matrix_ergodic(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

is.matrix_ergodic 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
eigvec	the dominant left eigenvector of A.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

is.matrix_primitive, is.matrix_irreducible.

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)
A

# Diagnose ergodicity
is.matrix_ergodic(A)

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

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

`is.matrix_irreducible` *Determine reducibility of a matrix*

Description

Determine whether a matrix is irreducible or reducible

Usage

```
is.matrix_irreducible(A)
```

Arguments

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

Details

`is.matrix_irreducible` works on the premise that a matrix is irreducible if and only if (I+A) raised to the power of s-1 is positive, where I is the identity matrix of the same dimension as `A` and s is the dimension of A (see Caswell 2001).

Value

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

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

`is.matrix_primitive`, `is.matrix_ergodic`.

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)
A

# Diagnose reducibility
is.matrix_irreducible(A)

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

# Diagnose reducibility
is.matrix_irreducible(B)
```

`is.matrix_primitive` *Determine primitivity of a matrix*

Description

Determine whether a matrix is primitive or imprimitive

Usage

```
is.matrix_primitive(A)
```

Arguments

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

Details

`is.matrix_primitive` works on the premise that a matrix A is primitive if A raised to the power of s^2-(2*s)+2 is positive, where s is the dimension of A (see Caswell 2001).

Value

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

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

is.matrix_irreducible, is.matrix_ergodic.

Examples

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

# Diagnose primitivity
is.matrix_primitive(A)

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

# Diagnose primitivity
is.matrix_primitive(B)
```

Keyfitz.delta *Calculate Keyfitz's delta*

Description

Calculate Keyfitz's delta for a given population projection matrix (PPM) model.

Usage

```
Keyfitz.delta(A, vector)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
vector	a specified initial age/stage distribution of class vector or class matrix.

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).

Keyfitz.delta 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

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

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

Kreiss *Calculate Kreiss bounds*

Description

Calculate the upper or lower Kreiss bound for a specified population projection matrix (PPM).

Usage

```
Kreiss(A, bound=NULL, return.r=FALSE, theta=1,
       rlimit=100, step1=1e-3, step2=1e-6)
```

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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

# 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)` may 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 uses string-splitting and string-building techniques to convert 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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]")
A
```

`maxamp` *Calculate maximal amplification*

Description

Calculate maximal amplification for a specified population projection matrix (PPM) model.

Usage

```
maxamp(A, vector = "n", return.N=FALSE, return.t=FALSE, return.stage=FALSE,
        conv.iterations=1e+5, conv.accuracy=1e-5)
```

Arguments

`A`	a square, primitive, non-negative numeric matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which 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 `convergence.time`.
`conv.accuracy`	the accuracy of convergence (see details). Please see `accuracy` in `convergence.time`.

Details

`maxamp` returns a standardised measure of maximal amplification, so 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 PPM and demographic structure is calculated. Note that not all specified 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 interval 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 convergence.time, 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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 indices of transient density:
reactivity, firststepatt maxatt, inertia

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)
A

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

# 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 specified population projection matrix (PPM) model.

Usage

```
maxatt(A, vector = "n", return.N = FALSE,return.t=FALSE, return.stage=FALSE,
        conv.iterations=1e+5, conv.accuracy=1e-5)
```

Arguments

A	a square, primitive, non-negative numeric matrix of any dimension
vector	(optional) a specified initial age/stage distribution of class vector or class matrix with which 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 `convergence.time`.

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

Details

`maxatt` returns a standardised measure of maximal attenuation, so 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 largest maximal attenuation that may be achieved) is returned, otherwise a case-specific maximal attenuation for the specified PPM and demographic structure is calculated. Note that not all specified demographic structures will yield a maximal attenuation: if the model does not attenuate 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 interval 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 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).

`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 `convergence.time`, 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.
In addition, 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 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

Other indices of transient density:
reactivity, firststepatt, maxamp, inertia

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)
A

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

# 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)
```

---

minCS *Calculate the mininum column sum of a matrix*

Description

Calculate the minimum column sum of a matrix.

Usage

```
minCS(A)
```

Arguments

A	a numeric matrix of any dimension.

Details

minCS finds the smallest column sum of any numeric matrix.

Value

The minimum colmn sum of A.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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)
A

# Find the minimum column sum of A
minCS(A)
```

---

project *Project population dynamics*

Description

Project dynamics of a specified population projection matrix (PPM) model.

Usage

```
project(A, vector="n", time=100, standard.A=FALSE,
        standard.vec=FALSE, return.vec=FALSE)
```

Arguments

A	a square, non-negative numeric matrix of any dimension.
vector	(optional) a specified initial age/stage distribution of class vector or class matrix with which to project dynamics.
time	the number of projection intervals.
standard.A	(optional) if TRUE, standardises the PPM by scaling it by its dominant eigenvalue so that asymptotic dynamics of the projection are removed.
standard.vec	(optional) if TRUE, standardises vector to sum to 1 by scaling it by sum(vector).
return.vec	(optional) if TRUE, returns the time series of vectors of demographic distribution as well as overall population size.

Details

If `vector` is specified, then `project` will project the dynamics of the specified model using that vector. However, if `vector="n"`, then `project` will automatically project the set of stage-biased vectors of `A`. In practise, these projections are achieved using a set of standard basis vectors, each with every element equal to 0, except for a single element that is equal to 1 (i.e. for a 3 by 3 matrix, the set of stage-biased vectors are: `c(1,0,0)`, `c(0,1,0)` and `c(0,0,1)`.)

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

Projections have their own S3 plotting method to enable easy graphing.

Value

If `vector` is specified, a vector of population sizes of length `time+1`.
If `vector="n"`, a matrix of population projections: each column represents a single stage-biased projection and each column is of length `time+1`.
If `return.vec=TRUE`, a list with components:

`N`	the vector or matrix of population sizes
`vec`	If `vector` is specified, a matrix of demographic vectors from projection of `vector` through `A`. Each column represents the densities of one life stage in the projection. If `vector="n"`, an 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` in length). The second dimension represents the densities of each stage (and is therefore equal to the dimension of `A` in length). The third dimension represents each individual stage-biased projection (and is therefore also equal to the dimension of `A` in length). So for example, if we projected a 3 by 3 matrix for >10 time intervals (see examples) then the density of stage 3 in bias 2 at time 10 is found at element [11,3,2] (remembering that because element 1 represents t=0, then t=10 is found at element 11); the time series of densities of stage 2 in bias 1 is found using [,2,1]; the matrix of population vectors for bias 2 would be found using [,,2].

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

See Also

S3 plotting method: `plot.projection`

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)
A

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

```
initial

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

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

# Project stage-biased dynamics of standardised A over 30
# intervals and return demographic vectors
pr2 <- project(A, time=30, standard.A=TRUE, return.vec=TRUE)
pr2
plot(pr2)

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

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

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

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

# Project A over 50 intervals using a specified population structure
pr3 <- project(A, vector=initial, time=50)
pr3
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)
pr4
plot(pr4)

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

`projection.distance` *Calculate projection distance*

Description

Calculate projection distance for a given population projection matrix (PPM) model.

Usage

```
projection.distance(A, vector)
```

Arguments

A	a square, irreducible, non-negative numeric matrix of any dimension.
vector	a specified initial age/stage distribution of class vector or class matrix.

Details

projection.distance (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.

projection.distance 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

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

# Calculate projection distance
projection.distance(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)` may is equivalent to [0 1 2;0.5 0.1 0;0 0.6 0.6].

R2Matlab takes an R object of class matrix and uses string-building techniques to convert 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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)
A

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

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

`reactivity` *Calculate reactivity*

Description

Calculate reactivity (first-timestep amplification) for a specified population projection matrix (PPM) model.

Usage

```
reactivity(A, vector = "n", return.N = FALSE)
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to calculate a case-specific reactivity
`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.

Details

`reactivity` is a standardised measure of first-timestep amplification, so discounts the effects of both initial population size and asymoptotic growth (Stott et al. 2011)

If `vector` is not specified then the bound on reactivity (the largest reactivity that may be achieved) is returned, otherwise a case-specific reactivity for the specified PPM and demographic structure is calculated. Note that not all specified demographic structures will yield a reactivity: if the model does not amplify in the first timestep then an error is returned and `firststepatt` should be used.

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

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

Value

If `vector="n"`, the bound on reactivity of `A`.
If `vector` is specified, the case-specific reactivity of the model.
If `return.N=TRUE`, a list with components:

`reactivity`	the bound on or case-specific reactivity
`N`	the population size at the first timestep, including the effects of initial population size and asymptotic growth.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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 indices of transient density:
`firststepatt`, `maxamp` `maxatt`, `inertia`

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)
A

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

# Calculate the bound on reactivity of A
reactivity(A)

# Calculate case-specific reactivity of A
# when projected using specific demographic structure
reactivity(A, vector=initial)

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

`S3 plot method for matrices of transfer functions`
Plot matrices of transfer functions

Description

Plot a matrix of transfer functions

Usage

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

Arguments

`x`	an object of class 'tfamatrix' created using `tfamatrix` or `inertia.tfamatrix`.
`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 `tfamatrix` and `xvar="p"` and `yvar="inertia"` for objects created using `inertia.tfamatrix`.
`mar`	the margin limits on the plots: see `par`
`...`	arguments to be passed to methods: see `par` and `plot`.

Details

Plots matrices of transfer functions (class tfamatrix) created using `tfamatrix` or `inertia.tfamatrix`. 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.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

See Also

`tfamatrix`, `inertia.tfamatrix`

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)
A

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

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

# Plot the result
plot(tfmat)

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

# Plot the result
plot(tfmat2)

# Plot inertia as a function of lambda
plot(tfmat2, xvar="lambda", yvar="inertia")
```

`S3 plot method for projections`

Plot population dynamics

Description

Plot dynamics of a specified population projection matrix (PPM) model.

Usage

```
    ## S3 method for class 'projection'
plot(x, labs=TRUE, ...)
```

Arguments

`x`	an object of class 'projection' created using `project`.
`labs`	logical: if `FALSE`, then lines are not labelled in stage-biased projections.
`...`	arguments to be passed to methods: see `par` and `plot`.

Details

Plots population dynamics (time series of density) for objects of class 'projection' created using `project`. The method is particularly useful for sets of stage-biased projections (which `project` creates by default when `vector="n"`).

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

See Also

project

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)
A

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

# plot stage-biased dynamics of A over 70 intervals using
# standardised dynamics and log y axis
plot(project(A,time=70,standard.A=TRUE,standard.vec=TRUE),
     log="y",ylab="Years",xlab="Density")

# plot a projection of a specified initial stage structure
# for 10 intervals
plot(project(A,vector=initial,time=50))
```

S3 plot method for transfer functions

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' created using tfa or inertia.tfa.
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 and xvar="p" and yvar="inertia" for objects created using inertia.tfa.
...	arguments to be passed to methods: see par and plot.

Details

Plots transfer functions (class tfa) created using tfa or inertia.tfa.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

See Also

tfa, inertia.tfa

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)
A

# 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(A, d=c(0,0,1), e=c(0,1,0), lambdarange=lambdarange)
transfer

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

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

# Calculate the transfer function of upper bound on inertia
# given a perturbation to A[3,2]
transfer2<-inertia.tfa(A, d=c(0,0,1), e=c(0,1,0), bound="upper",
                       prange=seq(-0.6,0.4,0.01))
transfer2

# Plot the transfer function (defaults to p and inertia in
# this case)
plot(transfer2)

# Plot lambda and inertia
plot(transfer2, xvar="lambda", yvar="inertia")
```

---

sens *Calculate sensitivity matrix*

Description

Calculate the sensitivity matrix using eigenvectors for a specified population projection matrix (PPM).

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 a reducible matrix 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 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).

Value

A sensitivity matrix of equal dimension to A.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

elas

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)
```

`tf` *Calculate transfer function*

Description

Calculate the transfer function of a given matrix and perturbation structure.

Usage

```
tf(A, z, b, c, exp=-1)
```

Arguments

`A`	a matrix of any dimension.
`z`	a numeric vector representing the set of real values over which to analyse the transfer function.
`b,c`	a numeric column vectors of class matrix that determines the perturbation structure (see details).
`exp`	the exponent to raise the resolvent to. Normally, `exp=-1` but for some usages an exponent of larger magnitude may be needed.

Details

`tf` is primarily designed to be called by `tfa`, `tfsens`, `inertia.tfa` and `inertia.tfsens` for evaluating transfer functions and sensitivities of population projection matrix models. However, it can also function as a standalone means of calculating a transfer function. Currently `tf` can only work with rank-one, single-parameter perturbations.

For `tf` to work properly, the dimensions of `A`, `b` and `c` must match. If `A` is an n by m matrix, then `b` must be a column vector of length m and `c` must be a column vector of length n.

The perturbation structure is determined by `b%*%t(c)`. Therefore, the rows to be perturbed are determined by `b` and the columns to be perturbed are determined by `c`. The specific values in `b` and `c` 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 `b = matrix(c(0,0,1), ncol=1)` and `c = matrix(c(0,1,0), ncol=1)`. 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 `b = matrix(c(0,0,1), ncol=1)` and `c = matrix(c(0,0.5,1), ncol=1)`. See Hodgson et al. (2006) for more information on perturbation structures.

Value

a vector of perturbation magnitudes whose INVERSE will achieve the dominant eigenvalues given by `z` for the triple `A`, `b`, `c`.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

tfa, tfsens

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)
A

# Calculate the transfer function given a range of z and a perturbation structure
evals<-eigen(A)$values
lmax<-which.max(Re(evals))
lambda<-Re(evals[lmax])
z <- seq(lambda-0.04, lambda+0.04, 0.01)
z <- z[ z != lambda]
b <- matrix(c(0,0,1), ncol=1)
c <- matrix(c(0,0,1), ncol=1)
p<-1/tf(A, z=z, b=b, c=c)
p

## Plot p and z
plot(z~p,type="l", xlab="p",ylab=expression(lambda[max]))
```

---

tfa *Analyse transfer function*

Description

Analyse the transfer function of a given population projection matrix (PPM) and perturbation structure.

Usage

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

Arguments

A	a square, irreducible, nonnegative matrix of any dimension
d,e	numeric vectors that determine the perturbation structure (see details).
lambdarange	a numeric vector giving the range of lambda values (asymptotic growth rates) to be achieved.
prange	a numeric vector giving the range of perturbation magnitude.
digits	specifies which values of lambda should be excluded to avoid a computationally singular system (see details).

Details

tfa calculates the transfer function of a matrix given a perturbation structure and a range of desired lambda values or a range of desired perturbation magnitude. Currently tfa 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 column vectors of class matrix, 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 and matched to their corresponding inertia, so that 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 desired lambda values from which the corresponding perturbation values will be calculated. Only one of either prange or lambdarange may be specified.

tfa uses the resolvent matrix in its calculation, which cannot be computed if any lambdarange 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 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.
lambda	the dominant eigenvalue of the perturbed matrix.

(Note that p will not be exactly the same as prange when 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.)

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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

Related:
tfamatrix

Sensitivity methods:
tfsens

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)
A

# 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(A, d=c(0,0,1), e=c(0,1,0), lambdarange=lambdarange)
transfer

# Plot the transfer function (defaults to lambda~p)
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(A, d=c(0,0,1), e=c(0,0.5,1), prange=p)
transfer2

# Plot p and lambda by hand
plot(transfer$lambda~transfer$p,type="l", xlab="p",ylab=expression(lambda))
```

`tfamatrix` *Analyse transfer function*

Description

Analyse the transfer function of a given population projection matrix (PPM) and perturbation structure.

Usage

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

Arguments

`A`	a square, irreducible, nonnegative matrix of any dimension
`elementtype`	(optional) a matrix describing the structure of `A`: `"P"` denotes elements bounded at 1, i.e. survival, growth, regression, `"F"` denotes elements not bounded at 1, i.e. fecundity, fission, and `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 to avoid a computationally singular system (see details).

Details

`tfamatrix` calculates an array of transfer functions for each nonzero element in `A`. Each element is perturbed individually. The function is most useful for use with the S3 method `plot.tfamatrix` to visualise how perturbations affect the life cycle and easily compare the effect on different transitions.

The structure of the perturbations are determined by `elementtype`, `Flim`, `Plim` and `plength`. `elementtype` gives the nature of each transition, specifying whether perturbations should be bounded at 1 or not. If `elementtype` is not directly specified, then the function assigns its own types, with those in the first row attributed `"F"`, and elsewhere in the matrix being attributed `"F"` if the element >1 and `"P"` if the element is <=1. `Flim` and `Plim` then determine the desired perturbation magnitude, expressed as a proportion of the magnitude of the elements, whilst plength determines the length of the perturbation vector. So, if an "F" element is equal to 0.5 and `Flim=c(-1,10)` and `plength=100`

then the perturbation to that element is `seq(-1*0.5,10*0.5,100-1) = seq(-0.5,5,100)`. The process is the same for `"P"` elements, except that these are bounded to give a maximum perturbed value of 1 (as growth/survival elements cannot be greater than unity). Both `"F"` and `"P"` elements are bounded to give a minimum perturbed value of 0.

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

`tfamatrix` will not work for reducible matrices.

Value

A list containing numerical arrays:

`p`	the perturbation magnitude.
`lambda`	the dominant eigenvalue of the perturbed matrix.

The first and second dimensions of the arrays are equivalent to the first and second dimensions of the matrix. The third dimension of the arrays are therefore the vectors returned by `tfa`. Therefore selecting e.g. $lambda[3,2,] will select the lambda values for element [3,2] of the matrix (for more information see examples).

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

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.tfamatrix`

Related:
`tfa`

Sensitivity methods:
`tfsens`

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)
A

# Calculate the matrix of transfer functions using default arguments
```

```
tfmat<-tfamatrix(A)
tfmat

# Plot the result
plot(tfmat)

# Plot the transfer function of element [3,2]
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)
etype
tfmat2 <- tfamatrix(B, elementtype=etype, Flim=c(-0.5,2),
                    Plim=c(-0.5,2))
tfmat2

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

`tfsens` *Calculate sensitivity using transfer functions*

Description

Calculate the sensitivity of perturbations to a population projection matrix using a transfer function method.

Usage

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

Arguments

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

startval	tfsens calculates the limit of the derivative of the transfer function as lambda 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 in the convergence algorithm are returned alongside the calculated sensitivity of inertia.
plot.fit	if TRUE then convergence of the algorithm is plotted as sensitivity~lambda.

Details

tfsens and tfsensmatrix differentiate a transfer function to find sensitivity to perturbation. This provides an alternative method to using matrix eigenvectors to calculate the sensitivity matrix and is useful as it may utilise more complex perturbation structures.

tfsens may be used to find sensitivity of a particular perturbation structure. A desired 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 column vectors of class matrix, 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.

tfsensmatrix returns a matrix of sensitivity values for observed transitions (similar to that obtained when using sens to evaluate sensitivity using eigenvectors), where the sensitivity of each matrix element is evaluated seperately.

The formula used by tfsens and tfsensmatrix 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 become uninvertible (singular): if matrices are found to be singular then tolerance should be relaxed and made larger.

For tfsens, 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 tfsens, 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 `tfsensmatrix`, a matrix containing sensitivity of lambda-max to each seperate element of `A`.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

See Also

Transfer function methods:
`tfa`, `tfamatrix`

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)
A

# Calculate the sensitivity matrix
tfsensmatrix(A)

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

`Tort` *Desert tortoise matrix*

Description

Population projection matrix for the desert tortoise Gopherus agassizii with medium fecundity.

Usage

```
data(Tort)
```

Details

The matrix is based on a population in the Western Mojave desert. Stages are based on age and size (carapace length in mm):
Yearling (age 0-1)
Juvenile 1 (<60 mm)
Juvenile 2 (90-99mm)
Immature 1 (100-139mm)
Immature 2 (140-179mm)
Subadult (180-207mm)
Adult 1 (208-239mm)
Adult 2 (>240mm).

References

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

Examples

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

`truelambda` *Calculate simulated asymptotic growth*

Description

Calculate the true asymptotic growth of a specified population projection matrix (PPM) model using simulation.

Usage

```
truelambda(A, vector = "n", accuracy=1e-7, iterations=1e+5)
```

Arguments

`A`	a square, non-negative numeric matrix of any dimension
`vector`	(optional) a specified initial age/stage distribution of class vector or class matrix with which to project the model.
`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. So 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 is calculated. In practise, these projections are achieved using a set of standard basis vectors, each with every element equal to 0, except for a single element that is equal to 1 (i.e. for a 3 by 3 matrix, the set of stage-biased vectors are: c(1,0,0), c(0,1,0) and c(0,0,1).)

Value

If vector is specified, a numeric vector of length 2 giving the range in which asymptotic growth 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. Therefore, the number of rows is equal to the dimension of A.

Author(s)

Stott, I., Hodgson, D. J., Townley, S.

References

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

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)
A

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

# 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