Package ‘MDPtoolbox’

October 12, 2022

Type Package

Title Markov Decision Processes Toolbox

Version 4.0.3

Date 2017-03-02

Author Iadine Chades, Guillaume Chapron, Marie-Josee Cros, Frederick Garcia, Regis Sabbadin

Maintainer Guillaume Chapron <gchapron@carnivoreconservation.org>

Description The Markov Decision Processes (MDP) toolbox proposes functions related to the resolution of discrete-time Markov Decision Processes: finite horizon, value iteration, policy iteration, linear programming algorithms with some variants and also proposes some functions related to Reinforcement Learning.

Depends Matrix, linprog

NeedsCompilation no

Repository CRAN

Date/Publication 2017-03-03 18:01:56

R topics documented:

`MDPtoolbox-package` *Markov Decision Processes Toolbox*

Description

The Markov Decision Processes (MDP) toolbox proposes functions related to the resolution of discrete-time Markov Decision Processes: finite horizon, value iteration, policy iteration, linear programming algorithms with some variants and also proposes some functions related to Reinforcement Learning.

Details

Package:	MDPtoolbox
Type:	Package
Version:	4.0.3
Date:	2017-03-02
License:	BSD (4.4)

Author(s)

Iadine Chadès <Iadine.Chades@csiro.au>
Guillaume Chapron <gchapron@carnivoreconservation.org>
Marie-Josée Cros <Marie-Josee.Cros@toulouse.inra.fr>
Fredérick Garcia <fgarcia@toulouse.inra.fr>
Régis Sabbadin <Regis.Sabbadin@toulouse.inra.fr>

References

Chadès, I., Chapron, G., Cros, M.-J., Garcia, F. & Sabbadin, R. 2014. MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. Ecography DOI:10.1111/ecog.00888
Puterman, M. L. 1994. Markov Decision Processes. John Wiley & Sons, New-York.

Examples

```
# Generates a random MDP problem
set.seed(0)
mdp_example_rand(2, 2)
mdp_example_rand(2, 2, FALSE)
mdp_example_rand(2, 2, TRUE)
mdp_example_rand(2, 2, FALSE, matrix(c(1,0,1,1),2,2))

# Generates a MDP for a simple forest management problem
MDP <- mdp_example_forest()

# Find an optimal policy
results <- mdp_policy_iteration(MDP$P, MDP$R, 0.9)

# Visualise the policy
results$policy
```

---

mdp_bellman_operator *Applies the Bellman operator*

Description

Applies the Bellman operator to a value function Vprev and returns a new value function and a Vprev-improving policy.

Usage

```
mdp_bellman_operator(P, PR, discount, Vprev)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
PR	reward array. PR can be a 2 dimension array [S,A] possibly sparse.
discount	discount factor. discount is a real number belonging to]0; 1].
Vprev	value fonction. Vprev is a vector of length S.

Details

mdp_bellman_operator applies the Bellman operator: PR + discount*P*Vprev to the value function Vprev. Returns a new value function and a Vprev-improving policy.

Value

V	new value fonction. V is a vector of length S.
policy	policy is a vector of length S. Each element is an integer corresponding to an action.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_bellman_operator(P, R, 0.9, c(0,0))

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_bellman_operator(P, R, 0.9, c(0,0))
```

---

mdp_check — *Checks the validity of a MDP*

Description

Checks the validity of a MDP

Usage

```
mdp_check(P, R)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.

Details

mdp_check checks whether the MDP defined by the transition probability array (P) and the reward matrix (R) is valid. If P and R are correct, the function returns an empty error message. In the opposite case, the function returns an error message describing the problem.

Value

Returns a character string which is empty if the MDP is valid. In the opposite case, the variable contains problem information

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_check(P, R)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_check(P, R)
```

`mdp_check_square_stochastic` *Checks if a matrix is square and stochastic*

Description

Checks whether a matrix is square and stochastic

Usage

```
mdp_check_square_stochastic(X)
```

Arguments

`X`	a matrix

Details

mdp_check_square_stochastic checks if the matrix (X) is square and stochastic (sums of rows equal to 1). If it is the case, the function returns an empty error message. In the opposite case, the function returns an error message describing the problem.

Value

Returns a character string which is empty if the matrix is square and stochastic. In the opposite case, the variable contains problem information.

Examples

```
M <- matrix(c(0.6116, 0.3884, 0, 1.0000), 2, 2, byrow=TRUE)

mdp_check_square_stochastic(M)
```

mdp_computePpolicyPRpolicy

Computes the transition matrix and the reward matrix for a fixed policy

Description

Computes the transition matrix and the reward matrix for a given policy.

Usage

```
mdp_computePpolicyPRpolicy(P, R, policy)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
policy	a policy. policy is a length S vector of integer representing actions.

Details

mdp_computePpolicyPRpolicy computes the state transition matrix and the reward matrix of a policy, given a probability matrix P and a reward matrix.

Value

Ppolicy	transition probability array of the policy. Ppolicy is a [S,S] matrix.
PRpolicy	reward matrix of the policy. PRpolicy is a vector of length S.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.6116, 0.3884, 0, 1.0000), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0.6674, 0.3326, 0, 1.0000), 2, 2, byrow=TRUE)
R <- array(0, c(2,2,2))
R[,,1] <- matrix(c(-0.2433, 0.7073, 0, 0.1871), 2, 2, byrow=TRUE)
R[,,2] <- matrix(c(-0.0069, 0.6433, 0, 0.2898), 2, 2, byrow=TRUE)
policy <- c(2,2)
mdp_computePpolicyPRpolicy(P, R, policy)

# With a sparse matrix (P)
P <- list()
P[[1]] <- Matrix(c(0.6116, 0.3884, 0, 1.0000), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0.6674, 0.3326, 0, 1.0000), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_computePpolicyPRpolicy(P, R, policy)
```

`mdp_computePR` *Computes a reward matrix for any form of transition and reward functions*

Description

Computes the reward associated to a state/action pair.

Usage

```
mdp_computePR(P, R)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.

Details

mdp_computePR computes the reward of a state/action pair, given a probability array P and a reward array possibly depending on arrival state.

Value

PR	reward matrix. PR is a [S,A] matrix.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.6116, 0.3884, 0, 1.0000), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0.6674, 0.3326, 0, 1.0000), 2, 2, byrow=TRUE)
R <- array(0, c(2,2,2))
R[,,1] <- matrix(c(-0.2433, 0.7073, 0, 0.1871), 2, 2, byrow=TRUE)
R[,,2] <- matrix(c(-0.0069, 0.6433, 0, 0.2898), 2, 2, byrow=TRUE)
mdp_computePR(P, R)

# With a sparse matrix (P)
P <- list()
P[[1]] <- Matrix(c(0.6116, 0.3884, 0, 1.0000), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0.6674, 0.3326, 0, 1.0000), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_computePR(P, R)
```

mdp_eval_policy_iterative
Evaluates a policy using an iterative method

Description

Evaluates a policy using iterations of the Bellman operator

Usage

```
mdp_eval_policy_iterative(P, R, discount, policy, V0, epsilon, max_iter)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
discount	discount factor. discount is a real number which belongs to [0; 1[.
policy	a policy. policy is a S length vector. Each element is an integer corresponding to an action.
V0	(optional) starting point. V0 is a S length vector representing an inital guess of the value function. By default, V0 is only composed of 0 elements.
epsilon	(optional) search for an epsilon-optimal policy. epsilon is a real greater than 0. By default, epsilon = 0.01.
max_iter	(optional) maximum number of iterations. max_iter is an integer greater than 0. If the value given in argument is greater than a computed bound, a warning informs that the computed bound will be used instead. By default, max_iter = 1000.

Details

mdp_eval_policy_iterative evaluates the value fonction associated to a policy applying iteratively the Bellman operator.

Value

Vpolicy	value fonction. Vpolicy is a S length vector.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
```

```
policy <- c(2,1)
mdp_eval_policy_iterative(P, R, 0.8, policy)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_eval_policy_iterative(P, R, 0.8, policy)
```

---

mdp_eval_policy_matrix

Evaluates a policy using matrix inversion and product

Description

Evaluates a policy using matrix operation

Usage

```
mdp_eval_policy_matrix(P, R, discount, policy)
```

Arguments

`P`	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
`R`	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
`discount`	discount factor. discount is a real number which belongs to [0; 1[.
`policy`	a policy. policy is a S length vector. Each element is an integer corresponding to an action.

Details

mdp_eval_policy_matrix evaluates the value fonction associated with a policy

Value

`Vpolicy`	value fonction. Vpolicy is a S length vector

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_eval_policy_matrix(P, R, 0.9, c(1,2))
```

```
# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_eval_policy_matrix(P, R, 0.9, c(1,2))
```

`mdp_eval_policy_optimality` *Computes sets of 'near optimal' actions for each state*

Description

Determines sets of 'near optimal' actions for all states

Usage

```
mdp_eval_policy_optimality(P, R, discount, Vpolicy)
```

Arguments

`P`	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
`R`	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
`discount`	discount factor. discount is a real number which belongs to [0; 1[.
`Vpolicy`	value function of the optimal policy. Vpolicy is a S length vector.

Details

For some states, the evaluation of the value function may give close results for different actions. It is interesting to identify those states for which several actions have a value function very close the optimal one (i.e. less than 0.01 different). We called this the search for near optimal actions in each state.

Value

`multiple`	existence of at least two 'nearly' optimal actions for a state. multiple is egal to true when at least one state has several epsilon-optimal actions, false if not.
`optimal_actions`	actions 'nearly' optimal for each state. optimal_actions is a [S,A] boolean matrix whose element optimal_actions(s, a) is true if the action a is 'nearly' optimal being in state s and false if not.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
Vpolicy <- c(42.4419, 36.0465)
mdp_eval_policy_optimality(P, R, 0.9, Vpolicy)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_eval_policy_optimality(P, R, 0.9, Vpolicy)
```

---

mdp_eval_policy_TD_0 *Evaluates a policy using the TD(0) algorithm*

Description

Evaluates a policy using the TD(0) algorithm

Usage

```
mdp_eval_policy_TD_0(P, R, discount, policy, N)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
discount	discount factor. discount is a real number which belongs to [0; 1[.
policy	a policy. policy is a S length vector. Each element is an integer corresponding to an action.
N	(optional) number of iterations to perform. N is an integer greater than the default value. By default, N is set to 10000

Details

mdp_eval_policy_TD_0 evaluates the value fonction associated to a policy using the TD(0) algorithm (Reinforcement Learning).

Value

Vpolicy	value fonction. Vpolicy is a length S vector.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_eval_policy_TD_0(P, R, 0.9, c(1,2))

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_eval_policy_TD_0(P, R, 0.9, c(1,2))
```

---

mdp_example_forest *Generates a MDP for a simple forest management problem*

Description

Generates a simple MDP example of forest management problem

Usage

```
mdp_example_forest(S, r1, r2, p)
```

Arguments

S	(optional) number of states. S is an integer greater than 0. By default, S is set to 3.
r1	(optional) reward when forest is in the oldest state and action Wait is performed. r1 is a real greater than 0. By default, r1 is set to 4.
r2	(optional) reward when forest is in the oldest state and action Cut is performed r2 is a real greater than 0. By default, r2 is set to 2.
p	(optional) probability of wildfire occurence. p is a real in]0, 1[. By default, p is set to 0.1.

Details

mdp_example_forest generates a transition probability (SxSxA) array P and a reward (SxA) matrix R that model the following problem. A forest is managed by two actions: Wait and Cut. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest.

Here is the modelisation of this problem. Let 1, ... S be the states of the forest. the Sth state being the oldest. Let Wait be action 1 and Cut action 2. After a fire, the forest is in the youngest state, that is state 1.

The transition matrix P of the problem can then be defined as follows:

$$P(,,1) = \begin{bmatrix} p & 1-p & 0 & \dots & \dots & 0 \\ p & 1-p & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & 1-p \\ p & 0 & 0 & \dots & 0 & 1-p \end{bmatrix}$$

$$P(,,2) = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & 0 & \dots & \dots & 0 \end{bmatrix}$$

The reward matrix R is defined as follows:

$$R(,1) = \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ r1 \end{bmatrix}$$

$$R(,2) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ r2 \end{bmatrix}$$

Value

P	transition probability array. P is a [S,S,A] array.
R	reward matrix. R is a [S,A] matrix

Examples

```
mdp_example_forest()
```

mdp_example_rand *Generates a random MDP problem*

Description

Generates a random MDP problem

Usage

```
mdp_example_rand(S, A, is_sparse, mask)
```

Arguments

S	number of states. S is an integer greater than 0
A	number of actions. A is an integer greater than 0
is_sparse	(optional) used to generate sparse matrices. is_sparse is a boolean. If it is set to true, sparse matrices are generated. By default, it is set to false.
mask	(optional) indicates the possible transitions between states. mask is a [S,S] matrix composed of 0 and 1 elements (0 indicates a transition probability always equal to zero). By default, mask is only composed of 1.

Details

mdp_example_rand generates a transition probability matrix (P) and a reward matrix (R). Optional arguments allow to define sparse matrices and pairs of states with impossible transitions.

Value

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S]. Elements of R are in]-1; 1[

Examples

```
mdp_example_rand(2, 2)
mdp_example_rand(2, 2, FALSE)
mdp_example_rand(2, 2, TRUE)
mdp_example_rand(2, 2, FALSE, matrix(c(1,0,1,1),2,2))
```

mdp_finite_horizon *Solves finite-horizon MDP using backwards induction algorithm*

Description

Solves finite-horizon MDP with backwards induction algorithm

Usage

```
mdp_finite_horizon(P, R, discount, N, h)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
discount	discount factor. discount is a real number which belongs to [0; 1[.
N	number of stages. N is an integer greater than 0.
h	(optional) terminal reward. h is a S length vector. By default, h = numeric(S).

Details

mdp_finite_horizon applies backwards induction algorithm for finite-horizon MDP. The optimality equations allow to recursively evaluate function values starting from the terminal stage. This function uses verbose and silent modes. In verbose mode, the function displays the current stage and the corresponding optimal policy.

Value

V	value fonction. V is a [S,(N+1)] matrix. Each column n is the optimal value fonction at stage n, with n = 1, ... N. V[,N+1] is the terminal reward.
policy	optimal policy. policy is a [S,N] matrix. Each element is an integer corresponding to an action and each column n is the optimal policy at stage n.
cpu_time	CPU time used to run the program

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_finite_horizon(P, R, 0.9, 3)
```

```
# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_finite_horizon(P, R, 0.9, 3)
```

---

mdp_LP *Solves discounted MDP using linear programming algorithm*

Description

Solves discounted MDP with linear programming

Usage

```
mdp_LP(P, R, discount)
```

Arguments

P	transition probability array. P is a 3 dimensions array [S,S,A]. Sparse matrix are not supported.
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
discount	discount factor. discount is a real which belongs to]0; 1[

Details

mdp_LP applies linear programming to solve discounted MDP for non-sparse matrix only.

Value

V	optimal value fonction. V is a S length vector
policy	optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function
cpu_time	CPU time used to run the program

Examples

```
# Only with a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_LP(P, R, 0.9)
```

mdp_policy_iteration *Solves discounted MDP using policy iteration algorithm*

Description

Solves discounted MDP with policy iteration algorithm

Usage

```
mdp_policy_iteration(P, R, discount, policy0, max_iter, eval_type)
```

Arguments

`P`	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
`R`	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
`discount`	discount factor. discount is a real which belongs to]0; 1[.
`policy0`	(optional) starting policy. policy0 is a S length vector. By default, policy0 is the policy which maximizes the expected immediate reward.
`max_iter`	(optional) maximum number of iterations to be done. max_iter is an integer greater than 0. By default, max_iter is set to 1000.
`eval_type`	(optional) define function used to evaluate a policy. eval_type is 0 for mdp_eval_policy_matrix use, mdp_eval_policy_iterative is used in all other cases. By default, eval_type is set to 0.

Details

mdp_policy_iteration applies the policy iteration algorithm to solve discounted MDP. The algorithm consists in improving the policy iteratively, using the evaluation of the current policy. Iterating is stopped when two successive policies are identical or when a specified number (max_iter) of iterations have been performed.

Value

`V`	optimal value fonction. V is a S length vector
`policy`	optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function
`iter`	number of iterations
`cpu_time`	CPU time used to run the program

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_policy_iteration(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_policy_iteration(P, R, 0.9)
```

---

mdp_policy_iteration_modified

Solves discounted MDP using modified policy iteration algorithm

---

Description

Solves discounted MDP using modified policy iteration algorithm

Usage

```
mdp_policy_iteration_modified(P, R, discount, epsilon, max_iter)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
discount	discount factor. discount is a real number which belongs to [0; 1[. For discount equals to 1, a warning recalls to check conditions of convergence.
epsilon	(optional) search for an epsilon-optimal policy. epsilon is a real in]0; 1]. By default, epsilon = 0.01.
max_iter	(optional) maximum number of iterations to be done. max_iter is an integer greater than 0. By default, max_iter = 1000.

Details

mdp_policy_iteration_modified applies the modified policy iteration algorithm to solve discounted MDP. The algorithm consists, like policy iteration one, in improving the policy iteratively but in policy evaluation few iterations (max_iter) of value function updates done. Iterating is stopped when an epsilon-optimal policy is found.

Value

V
: optimal value fonction. V is a S length vector

policy
: optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function.

iter
: number of iterations

cpu_time
: CPU time used to run the program

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R<- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_policy_iteration_modified(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_policy_iteration_modified(P, R, 0.9)
```

---

mdp_Q_learning — *Solves discounted MDP using the Q-learning algorithm (Reinforcement Learning)*

Description

Solves discounted MDP with the Q-learning algorithm (Reinforcement learning)

Usage

```
mdp_Q_learning(P, R, discount, N)
```

Arguments

P
: transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].

R
: reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.

discount
: discount factor. discount is a real which belongs to]0; 1[

N
: (optional) : number of iterations to perform. N is an integer that must be greater than the default value. By default, N is set to 10000.

Details

mdp_Q_learning computes the Q matrix, the mean discrepancy and gives the optimal value function and the optimal policy when allocated enough iterations. It uses an iterative method.

Value

`Q`	an action-value function that gives the expected utility of taking a given action in a given state and following an optimal policy thereafter. Q is a [S,A] matrix.
`mean_discrepancy`	discrepancy means over 100 iterations. mean_discrepancy is a vector of V discrepancy mean over 100 iterations. Then the length of the vector for the default value of N is 100.
`V`	value function. V is a S length vector.
`policy`	policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
# Not run
# mdp_Q_learning(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
# Not run
# mdp_Q_learning(P, R, 0.9)
```

---

mdp_relative_value_iteration

Solves MDP with average reward using relative value iteration algorithm

---

Description

Solves MDP with average reward using relative value iteration algorithm

Usage

```
mdp_relative_value_iteration(P, R, epsilon, max_iter)
```

Arguments

P	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
R	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
epsilon	(optional) : search for an epsilon-optimal policy. epsilon is a real in [0; 1]. By default, epsilon is set to 0.01
max_iter	(optional) : maximum number of iterations. max_iter is an integer greater than 0. By default, max_iter is set to 1000.

Details

mdp_relative_value_iteration applies the relative value iteration algorithm to solve MDP with average reward. The algorithm consists in solving optimality equations iteratively. Iterating is stopped when an epsilon-optimal policy is found or after a specified number (max_iter) of iterations is done.

Value

policy	optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function.
average_reward	average reward of the optimal policy. average_reward is a real.
cpu_time	CPU time used to run the program

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_relative_value_iteration(P, R)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_relative_value_iteration(P, R)
```

`mdp_span` *Evaluates the span of a vector*

Description

Computes the span of a vector.

Usage

```
mdp_span(W)
```

Arguments

`W`	a vector.

Details

mdp_span computes the span of the W vector: max W(s) - min W(s)

Value

the value of the span of the W vector.

`mdp_value_iteration` *Solves discounted MDP using value iteration algorithm*

Description

Solves discounted MDP with value iteration algorithm

Usage

```
mdp_value_iteration(P, R, discount, epsilon, max_iter, V0)
```

Arguments

`P`	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
`R`	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
`discount`	discount factor. discount is a real number which belongs to [0; 1[. For discount equals to 1, a warning recalls to check conditions of convergence.
`epsilon`	(optional) : search for an epsilon-optimal policy. epsilon is a real in]0; 1]. By default, epsilon = 0.01.

`max_iter` (optional) : maximum number of iterations. max_iter is an integer greater than 0. If the value given in argument is greater than a computed bound, a warning informs that the computed bound will be considered. By default, if discount is not egal to 1, a bound for max_iter is computed, if not max_iter = 1000.

`V0` (optional) : starting value function. V0 is a (Sx1) vector. By default, V0 is only composed of 0 elements.

Details

mdp_value_iteration applies the value iteration algorithm to solve discounted MDP. The algorithm consists in solving Bellman's equation iteratively. Iterating is stopped when an epsilon-optimal policy is found or after a specified number (max_iter) of iterations.

Value

`policy` optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function.

`iter` number of done iterations.

`cpu_time` CPU time used to run the program.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R<- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_value_iteration(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_value_iteration(P, R, 0.9)
```

`mdp_value_iterationGS` *Solves discounted MDP using Gauss-Seidel's value iteration algorithm*

Description

Solves discounted MDP with Gauss-Seidel's value iteration algorithm.

Usage

```
mdp_value_iterationGS(P, R, discount, epsilon, max_iter, V0)
```

Arguments

- `P` transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
- `R` reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
- `discount` discount factor. discount is a real which belongs to]0; 1]. For discount equals to 1, a warning recalls to check conditions of convergence.
- `epsilon` (optional) : search of an epsilon-optimal policy. epsilon is a real in]0; 1]. By default, epsilon is set to 0.01.
- `max_iter` (optional) : maximum number of iterations to be done. max_iter is an integer greater than 0. If the value given in argument is greater than a computed bound, a warning informs that the computed bound will be considered. By default, if discount is not equal to 1, a bound for max_iter is computed, if not max_iter is set to 1000.
- `V0` (optional) : starting value function. V0 is a S length vector. By default, V0 is only composed of 0 elements.

Details

mdp_value_iterationGS applies Gauss-Seidel's value iteration algorithm to solve discounted MDP. The algorithm consists, like value iteration, in solving Bellman's equation iteratively Vn+1(s) calculation is modified. The algorithm uses Vn+1(s) instead of Vn(s) whenever this value has been calculated. In this way, convergence speed is improved. Iterating is stopped when an epsilon-optimal policy is found or after a specified number (max_iter) of iterations.

Value

- `policy` epsilon-optimal policy. policy is a S length vector. Each element is an integer corresponding to an action which maximizes the value function.
- `iter` number of done iterations.
- `cpu_time` CPU time used to run the program.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_value_iterationGS(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_value_iterationGS(P, R, 0.9)
```

`mdp_value_iteration_bound_iter` *Computes a bound for the number of iterations for the value iteration algorithm*

Description

Computes a bound on the number of iterations for the value iteration algorithm

Usage

```
mdp_value_iteration_bound_iter(P, R, discount, epsilon, V0)
```

Arguments

`P`	transition probability array. P can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S].
`R`	reward array. R can be a 3 dimensions array [S,S,A] or a list [[A]], each element containing a sparse matrix [S,S] or a 2 dimensional matrix [S,A] possibly sparse.
`discount`	discount factor. discount is a real which belongs to]0; 1[.
`epsilon`	(optional) : search for an epsilon-optimal policy epsilon is a real in]0; 1]. By default, epsilon is set to 0.01.
`V0`	(optional) : starting value function. V0 is a S length vector. By default, V0 is only composed of 0 elements.

Details

mdp_value_iteration_bound_iter computes a bound on the number of iterations for the value iteration algorithm to find an epsilon-optimal policy with use of span for the stopping criterion.

Value

`max_iter`	maximum number of iterations to be done. max_iter is an integer greater than 0.
`cpu_time`	CPU time used to run the program.

Examples

```
# With a non-sparse matrix
P <- array(0, c(2,2,2))
P[,,1] <- matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE)
P[,,2] <- matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE)
R <- matrix(c(5, 10, -1, 2), 2, 2, byrow=TRUE)
mdp_value_iteration_bound_iter(P, R, 0.9)

# With a sparse matrix
P <- list()
P[[1]] <- Matrix(c(0.5, 0.5, 0.8, 0.2), 2, 2, byrow=TRUE, sparse=TRUE)
```

```
P[[2]] <- Matrix(c(0, 1, 0.1, 0.9), 2, 2, byrow=TRUE, sparse=TRUE)
mdp_value_iteration_bound_iter(P, R, 0.9)
```

Index