

MarmoteMDP

Generated by Doxygen 1.8.13

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	7
4.1	averageMDP Class Reference	7
4.1.1	Detailed Description	8
4.1.2	Constructor & Destructor Documentation	9
4.1.2.1	averageMDP() [1/3]	9
4.1.2.2	averageMDP() [2/3]	9
4.1.2.3	averageMDP() [3/3]	10
4.1.2.4	~averageMDP()	11
4.1.3	Member Function Documentation	11
4.1.3.1	changeIndex()	12
4.1.3.2	getChain()	12
4.1.3.3	policyCost()	13
4.1.3.4	policyIteration()	13
4.1.3.5	policyIterationModified()	14
4.1.3.6	policyIterationModifiedAA()	15
4.1.3.7	relativeValueIteration()	16

4.1.3.8	valueIteration()	17
4.1.3.9	valueIterationGS()	17
4.1.3.10	valueIterationMC()	18
4.1.4	Member Data Documentation	19
4.1.4.1	indexEtatSpecifique	19
4.1.4.2	rho	19
4.1.4.3	type_c	19
4.2	continuousDiscountedMDP Class Reference	19
4.2.1	Detailed Description	20
4.2.2	Member Function Documentation	21
4.2.2.1	discountedMDP()	21
4.2.3	Member Data Documentation	22
4.2.3.1	accumulateRewards	22
4.2.3.2	beta	22
4.2.3.3	type_c	22
4.3	discountedMDP Class Reference	22
4.3.1	Detailed Description	24
4.3.2	Constructor & Destructor Documentation	24
4.3.2.1	discountedMDP() [1/3]	24
4.3.2.2	discountedMDP() [2/3]	25
4.3.2.3	discountedMDP() [3/3]	26
4.3.2.4	~discountedMDP()	27
4.3.3	Member Function Documentation	27
4.3.3.1	getChain()	27
4.3.3.2	policyCost()	28
4.3.3.3	policyCostbyIndex()	28
4.3.3.4	policyIteration()	29
4.3.3.5	policyIterationModified()	30
4.3.3.6	policyIterationModifiedGS()	30
4.3.3.7	valueIteration()	31

4.3.3.8	valueIterationGS()	32
4.3.4	Member Data Documentation	32
4.3.4.1	beta	33
4.3.4.2	type_c	33
4.4	feedbackSolutionMDP Class Reference	33
4.4.1	Detailed Description	34
4.4.2	Constructor & Destructor Documentation	35
4.4.2.1	feedbackSolutionMDP()	35
4.4.2.2	~feedbackSolutionMDP()	35
4.4.3	Member Function Documentation	35
4.4.3.1	getAction()	35
4.4.3.2	getActionIndex()	36
4.4.3.3	getValue()	36
4.4.3.4	getValueIndex()	36
4.4.3.5	setAction()	37
4.4.3.6	setActionIndex()	37
4.4.3.7	setValue()	38
4.4.3.8	writeSolution()	38
4.4.3.9	writeSolutionByDim()	39
4.4.4	Member Data Documentation	39
4.4.4.1	action	39
4.4.4.2	value	39
4.5	finiteHorizonDiscountedMDP Class Reference	40
4.5.1	Detailed Description	41
4.5.2	Constructor & Destructor Documentation	41
4.5.2.1	finiteHorizonDiscountedMDP() [1/2]	42
4.5.2.2	finiteHorizonDiscountedMDP() [2/2]	42
4.5.2.3	~finiteHorizonDiscountedMDP()	43
4.5.3	Member Function Documentation	44
4.5.3.1	getChain()	44

4.5.3.2	policyCost()	45
4.5.3.3	policyIteration()	45
4.5.3.4	policyIterationModified()	46
4.5.3.5	valueIteration() [1/2]	47
4.5.3.6	valueIteration() [2/2]	48
4.5.3.7	valueIterationGS()	48
4.5.3.8	writeMDP()	49
4.5.4	Member Data Documentation	49
4.5.4.1	beta_	49
4.5.4.2	horizon_	49
4.5.4.3	type_c	50
4.6	finiteHorizonMDP Class Reference	50
4.6.1	Detailed Description	51
4.6.2	Constructor & Destructor Documentation	52
4.6.2.1	finiteHorizonMDP() [1/2]	52
4.6.2.2	finiteHorizonMDP() [2/2]	53
4.6.2.3	~finiteHorizonMDP()	53
4.6.3	Member Function Documentation	54
4.6.3.1	getChain()	54
4.6.3.2	policyCost()	55
4.6.3.3	policyIteration()	55
4.6.3.4	policyIterationModified()	56
4.6.3.5	valueIteration() [1/2]	57
4.6.3.6	valueIteration() [2/2]	58
4.6.3.7	valueIterationGS()	58
4.6.3.8	writeMDP()	59
4.6.4	Member Data Documentation	59
4.6.4.1	horizon_	59
4.6.4.2	type_c	60
4.7	genericMDP Class Reference	60

4.7.1	Detailed Description	62
4.7.2	Constructor & Destructor Documentation	62
4.7.2.1	genericMDP() [1/3]	62
4.7.2.2	genericMDP() [2/3]	63
4.7.2.3	genericMDP() [3/3]	64
4.7.3	Member Function Documentation	65
4.7.3.1	addMatrix()	65
4.7.3.2	clearRew()	65
4.7.3.3	cost_perStage()	66
4.7.3.4	deleMatrix()	66
4.7.3.5	getChain()	67
4.7.3.6	policyCost()	67
4.7.3.7	policyIteration()	68
4.7.3.8	policyIterationModified()	69
4.7.3.9	valueIteration()	69
4.7.3.10	valueIterationGS()	70
4.7.4	Member Data Documentation	71
4.7.4.1	actionSpace	71
4.7.4.2	opMin	71
4.7.4.3	rewards	71
4.7.4.4	stateSpace	71
4.7.4.5	transitions	71
4.7.4.6	type_r	72
4.7.4.7	type_t	72
4.8	nonStationarySolutionMDP Class Reference	72
4.8.1	Detailed Description	74
4.8.2	Constructor & Destructor Documentation	74
4.8.2.1	nonStationarySolutionMDP()	74
4.8.2.2	~nonStationarySolutionMDP()	75
4.8.3	Member Function Documentation	75

4.8.3.1	getAction()	75
4.8.3.2	getActionAtStep()	76
4.8.3.3	getActionAtStepIndex()	76
4.8.3.4	getHorizon()	77
4.8.3.5	getValue()	77
4.8.3.6	getValueAtStep()	78
4.8.3.7	getValueAtStepIndex()	78
4.8.3.8	setAction()	79
4.8.3.9	setActionAtStep()	79
4.8.3.10	setActionAtStepIndex()	80
4.8.3.11	setValue()	80
4.8.3.12	setValueAtStep()	81
4.8.3.13	setValueAtStepIndex()	81
4.8.3.14	writeSolution()	82
4.8.3.15	writeSolutionByDim()	83
4.8.4	Member Data Documentation	83
4.8.4.1	action	83
4.8.4.2	horizon_	83
4.8.4.3	value	84
4.9	solutionMDP Class Reference	84
4.9.1	Detailed Description	85
4.9.2	Constructor & Destructor Documentation	85
4.9.2.1	solutionMDP()	85
4.9.2.2	~solutionMDP()	86
4.9.3	Member Function Documentation	86
4.9.3.1	setSize()	86
4.9.3.2	writeSolution()	87
4.9.3.3	writeSolutionByDim()	87
4.9.4	Member Data Documentation	88
4.9.4.1	size_	88

4.10 structuralPropertiesPol Class Reference	88
4.10.1 Detailed Description	89
4.10.2 Constructor & Destructor Documentation	89
4.10.2.1 structuralPropertiesPol()	89
4.10.2.2 ~structuralPropertiesPol()	90
4.10.3 Member Function Documentation	90
4.10.3.1 monotonicityPol() [1/2]	90
4.10.3.2 monotonicityPol() [2/2]	91
4.10.3.3 monotonicityPolByDim() [1/2]	91
4.10.3.4 monotonicityPolByDim() [2/2]	92
4.10.3.5 sSPol() [1/2]	92
4.10.3.6 sSPol() [2/2]	93
4.10.3.7 sSPolbyDim() [1/2]	93
4.10.3.8 sSPolbyDim() [2/2]	94
4.10.3.9 thresholdPol() [1/2]	94
4.10.3.10 thresholdPol() [2/2]	95
4.10.4 Member Data Documentation	95
4.10.4.1 space_	95
4.11 structuralPropertiesVF Class Reference	95
4.11.1 Detailed Description	96
4.11.2 Constructor & Destructor Documentation	96
4.11.2.1 structuralPropertiesVF()	97
4.11.2.2 ~structuralPropertiesVF()	97
4.11.3 Member Function Documentation	97
4.11.3.1 monotonicityCX() [1/2]	98
4.11.3.2 monotonicityCX() [2/2]	98
4.11.3.3 monotonicityVF() [1/2]	99
4.11.3.4 monotonicityVF() [2/2]	99
4.11.3.5 monotonicityVFByDim() [1/2]	100
4.11.3.6 monotonicityVFByDim() [2/2]	100

4.11.4	Member Data Documentation	101
4.11.4.1	space_	101
4.12	totalRewardMDP Class Reference	101
4.12.1	Detailed Description	102
4.12.2	Constructor & Destructor Documentation	103
4.12.2.1	totalRewardMDP() [1/3]	103
4.12.2.2	totalRewardMDP() [2/3]	103
4.12.2.3	totalRewardMDP() [3/3]	104
4.12.2.4	~totalRewardMDP()	105
4.12.3	Member Function Documentation	105
4.12.3.1	getChain()	106
4.12.3.2	policyCost()	106
4.12.3.3	policyIteration()	107
4.12.3.4	policyIterationModified()	108
4.12.3.5	valueIteration()	108
4.12.3.6	valueIterationGS()	109
4.12.4	Member Data Documentation	110
4.12.4.1	type_c	110
5	File Documentation	111
5.1	algin.h File Reference	111
5.1.1	Detailed Description	112
5.1.2	Function Documentation	112
5.1.2.1	Inversion()	112
5.1.2.2	Norm()	112
5.1.2.3	prodScaIV()	113
5.1.2.4	produitMatMat()	113
5.1.2.5	produitMatVect()	114
5.1.2.6	ResolutionSysLin()	114
5.1.2.7	Span()	115
5.1.2.8	SpanRecup()	115
5.1.2.9	Transpose()	116
5.1.2.10	verifMat()	116
5.1.2.11	verifMat2D()	117
5.1.2.12	verifMatP()	117
5.1.2.13	verifVd()	117
5.1.2.14	verifVi()	118
	Index	119

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

genericMDP	60
averageMDP	7
continuousDiscountedMDP	19
discountedMDP	22
finiteHorizonDiscountedMDP	40
finiteHorizonMDP	50
totalRewardMDP	101
solutionMDP	84
feedbackSolutionMDP	33
nonStationarySolutionMDP	72
structuralPropertiesPol	88
structuralPropertiesVF	95

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

averageMDP	
Class averageMDP header: definition of an infinite horizon average MDP class	7
continuousDiscountedMDP	
Class continuousDiscountedMDP header: definition of a continuous time infinite horizon discounted MDP	19
discountedMDP	
Class discountedMDP header: definition of an infinite horizon discounted MDP class	22
feedbackSolutionMDP	
Class feedbackSolutionMDP : implementation of a feedbackSolutionMDP class	33
finiteHorizonDiscountedMDP	
Class finiteHorizonDiscountedMDP : definition of a finite horizon MDP class with discounted cost	40
finiteHorizonMDP	
Class finiteHorizonMDP : definition of a finite horizon MDP class	50
genericMDP	
Class genericMDP header : implementation of an abstract MDP class that represents an MDP object and can be inherited by other classes	60
nonStationarySolutionMDP	
Class nonStationarySolutionMDP : implementation of a nonStationarySolutionMDP class	72
solutionMDP	
Class solutionMDP : implementation of an abstract solutionMDP class	84
structuralPropertiesPol	
Class structuralPropertiesVF : implementation of a class to test structural Properties of Value Function	88
structuralPropertiesVF	
Class structuralPropertiesVF : implementation of a class to test structural Properties of Value Function	95
totalRewardMDP	
Class totalRewardMDP header: definition of an infinite horizon total reward MDP class	101

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

algin.h	Library of linear algebra functions (root programming)	111
averageMDP.h		??
continuousDiscountedMDP.h		??
discountedMDP.h		??
feedbackSolutionMDP.h		??
finiteHorizonDiscountedMDP.h		??
finiteHorizonMDP.h		??
genericMDP.h		??
nonStationarySolutionMDP.h		??
solutionMDP.h		??
structuralPropertiesPol.h		??
structuralPropertiesVF.h		??
totalRewardMDP.h		??

Chapter 4

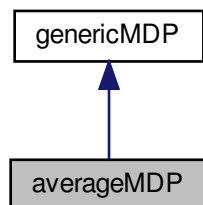
Class Documentation

4.1 averageMDP Class Reference

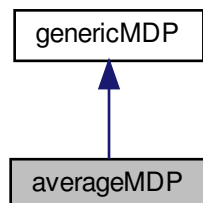
Class [averageMDP](#) header: definition of an infinite horizon average MDP class.

```
#include <averageMDP.h>
```

Inheritance diagram for averageMDP:



Collaboration diagram for averageMDP:



Public Member Functions

- void [writeMDP](#) ()
function providing printing function to stdout.
- [averageMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, sparseMatrix *rews)
Constructor to create [averageMDP](#) object.
- [averageMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews)
Constructor to create [averageMDP](#) object.
- [averageMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, sparseMatrix *rews)
Constructor to create [averageMDP](#) object with no vector of transition in parameters.
- [~averageMDP](#) ()
destructor to delete [averageMDP](#) object.
- void [changeIndex](#) (int index)
A function to change the index of the state used in value iteration.
- [solutionMDP](#) * [valueIteration](#) (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm.
- [solutionMDP](#) * [relativeValueIteration](#) (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value relative iteration algorithm.
- [solutionMDP](#) * [policyIteration](#) (int maxIter)
A function to solve (discrete time) MDP using policy iteration algorithm.
- [solutionMDP](#) * [valueIterationGS](#) (double epsilon, int maxIter)
A function that should implements relative value iteration with Gauss Seidel decomposition.
- double * [policyCost](#) ([solutionMDP](#) *policy, double epsilon, int maxIter)
A function to evaluate the average cost of a policy with iteration of the power.
- [solutionMDP](#) * [policyIterationModified](#) (double epsilon, int maxIter, double delta, int maxIter)
A function to solve (discrete time) MDP using modified policy iteration algorithm.
- [solutionMDP](#) * [policyIterationModifiedAA](#) (double epsilon, int maxIter, double delta, int maxIter)
A function to solve (discrete time) MDP using modified policy iteration algorithm and an auto adaptation of error.
- [solutionMDP](#) * [valueIterationMC](#) (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm for multichain problem.
- sparseMatrix * [getChain](#) ([solutionMDP](#) *policy)
A function to build the transition matrix associated to a policy.

Protected Attributes

- double [rho](#)
- std::string [type_c](#)
- int [indexEtatSpecifique](#)

4.1.1 Detailed Description

Class [averageMDP](#) header: definition of an infinite horizon average MDP class.

Author

Emmanuel Hyon.

Version

1.1

Date

january 2019

This class is inherited from the abstract class [genericMDP](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 averageMDP() [1/3]

```
averageMDP::averageMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews )
```

Constructor to create [averageMDP](#) object.

Author

EH

Version

0.91

Date

jan 2019

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an * entry of the matrix is the reward for state indexed by line and action indexed by * column.

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector * is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for * state in line and action in column.

4.1.2.2 averageMDP() [2/3]

```
averageMDP::averageMDP (
    std::string r,
```

```

    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews )

```

Constructor to create [averageMDP](#) object.

Author

EH

Version

0.9.1

Date

jan 2019

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	vector<sparseMatrix*> vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

`rews` vector of sparse matrix objects: an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.1.2.3 [averageMDP\(\)](#) [3/3]

```

averageMDP::averageMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    sparseMatrix * rews )

```

Constructor to create [averageMDP](#) object with no vector of transition in parameters.

Author

Hyon

Version

1

Date

july 2020

Parameters

<i>r</i>	: string, type_r rule can be "min" , "max".
<i>states</i>	: marmoteSet. State space
<i>actions</i>	: marmoteSet. Action space
<i>rewards</i>	: reward : sparseMatrix object (state, action) an entry of the matrix is the reward for state line and action column.

Returns

none.

this third constructor does not take any vector of transition.

The transition should be added after

rewards : reward structure : (state, action) an entry of the sparseMatrix is the reward for state line and action column.

4.1.2.4 ~averageMDP()

```
averageMDP::~~averageMDP ( ) [inline]
```

destructor to delete [averageMDP](#) object.

Author

EH

4.1.3 Member Function Documentation

4.1.3.1 `changeIndex()`

```
void averageMDP::changeIndex (
    int index ) [inline]
```

A function to change the index of the state used in value iteration.

Author

Hyon

Version

3

Date

juil 2018

Parameters

<i>index</i>	int the index of the new
--------------	--------------------------

4.1.3.2 `getChain()`

```
sparseMatrix * averageMDP::getChain (
    solutionMDP * policy ) [virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the policy.
---------------	--

Returns

a `sparseMatrix`

Implements [genericMDP](#).

4.1.3.3 policyCost()

```
double * averageMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [virtual]
```

A function to evaluate the average cost of a policy with iteration of the power.

Author

Hyon

Version

0.1

Date

Feb 2019

Parameters

<i>policy</i>	solutionMDP : object to get of the action
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

a pointer on a double which is the average cost in each state (identical in unichain model).

Implements [genericMDP](#).

4.1.3.4 policyIteration()

```
solutionMDP * averageMDP::policyIteration (
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using policy iteration algorithm.

Author

Hyon

Version

3

Date

nov 2018

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns[solutionMDP](#) object.**Warning**

not implemented

Implements [genericMDP](#).**4.1.3.5 policyIterationModified()**

```
solutionMDP * averageMDP::policyIterationModified (  
    double epsilon,  
    int maxIter,  
    double delta,  
    int maxInIter ) [virtual]
```

A function to solve (discrete time) MDP using modified policy iteration algorithm.

Author

Hyon

Version

0.1

Date

Feb 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

[solutionMDP](#) object.

also called modified value iteration in Puterman's book

In the value part of the solution. The average gain is placed.

For an unichain problem the same value is given in the solution for any state.

Implements [genericMDP](#).

4.1.3.6 policyIterationModifiedAA()

```
solutionMDP * averageMDP::policyIterationModifiedAA (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter )
```

A function to solve (discrete time) MDP using modified policy iteration algorithm and an auto adaptation of error.

Author

Hyon

Version

0.1

Date

Feb 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

`solutionMDP` object.

Also called modified value iteration in Puterman's book.

Here the error for the computation of the iterative solution of the value function is decreased as soon as it is larger than the span.

In the value part of the solution. The average gain is placed.

For an unichain problem : the same value is given in the solution for any state.

AA means AutoAdaptatable

4.1.3.7 `relativeValueIteration()`

```
solutionMDP * averageMDP::relativeValueIteration (
    double epsilon,
    int maxIter )
```

A function to solve (discrete time) MDP using value relative iteration algorithm.

Author

Hyon

Version

0.1

Date

jan 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

`solutionMDP` object.

This is the relative value iteration algorithm which compute a Bellman Equation including bias an average gain.

For unichain problem The average gain is placed in the solution for any state.

4.1.3.8 valueIteration()

```
solutionMDP * averageMDP::valueIteration (
    double epsilon,
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm.

Author

Hyon

Version

0.1

Date

jan 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

This is the value iteration algorithm which compute a Bellman Equation defined in part 8.5.1 of Puterman.

In the value part of the solution.

For an unichain problem The average gain is placed for any state in the solution returned.

Implements [genericMDP](#).

4.1.3.9 valueIterationGS()

```
solutionMDP * averageMDP::valueIterationGS (
    double epsilon,
    int maxIter ) [virtual]
```

A function that should implements relative value iteration with Gauss Seidel decomposition.

Author

Hyon

Version

3

Date

apr 2020

Parameters

<i>epsilon</i>	double the precision of the computation
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Warning

not implemented

Implements [genericMDP](#).

4.1.3.10 valueIterationMC()

```
solutionMDP * averageMDP::valueIterationMC (  
    double epsilon,  
    int maxIter )
```

A function to solve (discrete time) MDP using value iteration algorithm for multichain problem.

Author

Hyon

Version

0.1

Date

jul 2020

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

This is the value iteration algorithm for multichain problem

It implements the algorithm described in Puterman chapter 8

The average gain is placed for any state of state of the recureent class in the solution returned.

4.1.4 Member Data Documentation

4.1.4.1 indexEtatSpecifique

```
int averageMDP::indexEtatSpecifique [protected]
```

index de l'etat specifique pour calculer le cout moyen

4.1.4.2 rho

```
double averageMDP::rho [protected]
```

avergae value

4.1.4.3 type_c

```
std::string averageMDP::type_c [protected]
```

MDP criteria: here "average"

The documentation for this class was generated from the following files:

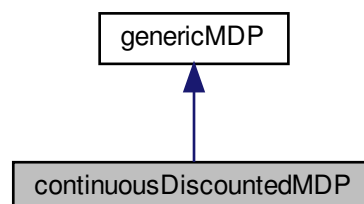
- averageMDP.h
- averageMDP.cpp

4.2 continuousDiscountedMDP Class Reference

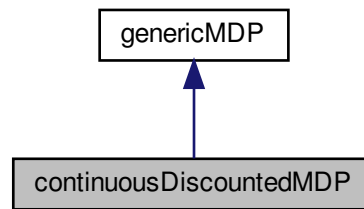
Class [continuousDiscountedMDP](#) header: definition of a continuous time infinite horizon discounted MDP.

```
#include <continuousDiscountedMDP.h>
```

Inheritance diagram for continuousDiscountedMDP:



Collaboration diagram for continuousDiscountedMDP:



Public Member Functions

- void `writeMDP` ()
function providing printing function to stdout.
- `discountedMDP` (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews, double b)
Constructor to create `discountedMDP` object.

Protected Attributes

- double `beta`
- std::string `type_c`
- sparseMatrix * `accumulateRewards`

4.2.1 Detailed Description

Class `continuousDiscountedMDP` header: definition of a continuous time infinite horizon discounted MDP.

Author

Emmanuel Hyon

Version

3

Date

dec 2018

This class is inherited from the abstract class `genericMDP`.

This class is dedicated to the modeling and the solving of discounted infinite horizon MDP

4.2.2 Member Function Documentation

4.2.2.1 discountedMDP()

```
continuousDiscountedMDP::discountedMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews,
    double b )
```

Constructor to create [discountedMDP](#) object.

Author

EH

Version

0.1

Date

Oct 2020

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an entry of the matrix is the reward for state indexed by line and action indexed by * column. These costs are instanteneousCosts
<i>accumRews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an entry of the matrix is the reward for state indexed by line and action indexed by * column. these costs are given in per unit of time
<i>b</i>	double : discount factor beta

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state : $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

trans : transition structures vector of `sparseMatrix` objects an entry of the vector `*` is a transition structure matrix from state to state for the action at vector index.

rews vector of sparse matrix objects an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.2.3 Member Data Documentation

4.2.3.1 accumulateRewards

```
sparseMatrix* continuousDiscountedMDP::accumulateRewards [protected]
```

a rewards (costs) structure (row : state, column : action)

4.2.3.2 beta

```
double continuousDiscountedMDP::beta [protected]
```

discount factor

4.2.3.3 type_c

```
std::string continuousDiscountedMDP::type_c [protected]
```

MDP criteria: here "infinite discounted"

The documentation for this class was generated from the following file:

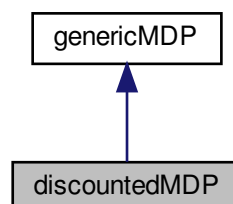
- `continuousDiscountedMDP.h`

4.3 discountedMDP Class Reference

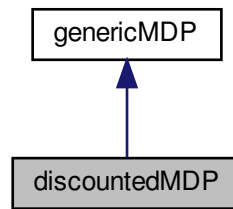
Class [discountedMDP](#) header: definition of an infinite horizon discounted MDP class.

```
#include <discountedMDP.h>
```

Inheritance diagram for `discountedMDP`:



Collaboration diagram for discountedMDP:



Public Member Functions

- void `writeMDP` ()
function providing printing function to stdout.
- `discountedMDP` (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, sparseMatrix *rews, double b)
Constructor to create `discountedMDP` object.
- `discountedMDP` (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews, double b)
Constructor to create `discountedMDP` object.
- `discountedMDP` (std::string r, marmoteSet *states, marmoteSet *actions, sparseMatrix *rews, double b)
Constructor to create `discountedMDP` object with no vector of transition in parameters.
- virtual `~discountedMDP` ()
destructor to delete `discountedMDP` object.
- `solutionMDP` * `valueIteration` (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm.
- `solutionMDP` * `valueIterationGS` (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.
- `solutionMDP` * `policyIterationModified` (double epsilon, int maxIter, double delta, int maxInIter)
A function to solve (discrete time) MDP using modified policy iteration algorithm.
- `solutionMDP` * `policyIterationModifiedGS` (double epsilon, int maxIter, double delta, int maxInIter)
A function to solve (discrete time) MDP using modified policy iteration algorithm and Gauss Seidel improvement.
- `solutionMDP` * `policyIteration` (int maxIter)
A function to solve (discrete time) MDP using policy iteration algorithm.
- double * `policyCost` (`solutionMDP` *policy, double epsilon, int maxIter)
A function to evaluate the cost of a policy with iteration of the power.
- double * `policyCostbyIndex` (`solutionMDP` *policy, double epsilon, int maxIter)
A function to evaluate the cost of a policy with iteration of the power and the scan of the set is done by index.
- sparseMatrix * `getChain` (`solutionMDP` *policy)
A function to build the transition matrix associated to a policy.

Protected Attributes

- double `beta`
- std::string `type_c`

4.3.1 Detailed Description

Class [discountedMDP](#) header: definition of an infinite horizon discounted MDP class.

Author

Emmanuel Hyon

Version

3

Date

dec 2018

This class is inherited from the abstract class [genericMDP](#).

This class is dedicated to the modeling and the solving of discounted infinite horizon MDP

4.3.2 Constructor & Destructor Documentation

4.3.2.1 [discountedMDP\(\)](#) [1/3]

```
discountedMDP::discountedMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews,
    double b )
```

Constructor to create [discountedMDP](#) object.

Author

EH

Version

1

Date

dec 2018

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an entry of the matrix is the reward for state indexed by line and action indexed by * column.
<i>b</i>	double : discount factor beta

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for state in line and action in column.

4.3.2.2 discountedMDP() [2/3]

```
discountedMDP::discountedMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews,
    double b )
```

Constructor to create [discountedMDP](#) object.

Author

EH

Version

1

Date

dec 2018

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	vector<sparseMatrix*> vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.
<i>b</i>	double : discount factor beta

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state : $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector `*` is a transition structure matrix from state to state for the action at vector index.

`rews` vector of sparse matrix objects an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.3.2.3 discountedMDP() [3/3]

```
discountedMDP::discountedMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    sparseMatrix * rews,
    double b )
```

Constructor to create `discountedMDP` object with no vector of transition in parameters.

Author

Hyon

Version

1

Date

july 2020

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	: marmoteSet. State space
<i>actions</i>	: marmoteSet. Action space
<i>rews</i>	: reward : sparseMatrix object (state, action) an entry of the matrix is the reward for state line and action column.
<i>b</i>	double : discount factor beta

Returns

none.

this third constructor does not take any vector of transition.

The transition should be added after

rews : reward structure : (state, action) an entry of the sparseMatrix is the reward for state line and action column.

4.3.2.4 ~discountedMDP()

```
discountedMDP::~~discountedMDP ( ) [virtual]
```

destructor to delete [discountedMDP](#) object.

Author

EH

4.3.3 Member Function Documentation

4.3.3.1 getChain()

```
sparseMatrix * discountedMDP::getChain (
    solutionMDP * policy ) [virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	<code>solutionMDP*</code> : pointer to a solution that contains the policy.
---------------	---

Returns

a sparseMatrix

Implements [genericMDP](#).

4.3.3.2 policyCost()

```
double * discountedMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [virtual]
```

A function to evaluate the cost of a policy with iteration of the power.

Author

EH

Version

1.1

Date

april 2018

Parameters

<i>policy</i>	solutionMDP : object to get of the action
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

double.

Implements [genericMDP](#).

4.3.3.3 policyCostbyIndex()

```
double * discountedMDP::policyCostbyIndex (
    solutionMDP * policy,
    double epsilon,
    int maxIter )
```

A function to evaluate the cost of a policy with iteration of the power and the scan of the set is done by index.

Author

EH

Version

3

Date

april 2018

Parameters

<i>policy</i>	solutionMDP : object to get of the action
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

double.

For purpose of debug

4.3.3.4 policyIteration()

```
solutionMDP * discountedMDP::policyIteration (
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using policy iteration algorithm.

Author

EH

Version

3

Date

jan 2018

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns

[solutionMDP](#) object.

Warning

not fully implemented

Implements [genericMDP](#).

4.3.3.5 policyIterationModified()

```
solutionMDP * discountedMDP::policyIterationModified (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter ) [virtual]
```

A function to solve (discrete time) MDP using modified policy iteration algorithm.

Author

EH

Version

1.4

Date

Feb 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

`solutionMDP` object.

also called hybrid value iteration in Powell

Implements `genericMDP`.

4.3.3.6 policyIterationModifiedGS()

```
solutionMDP * discountedMDP::policyIterationModifiedGS (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter )
```

A function to solve (discrete time) MDP using modified policy iteration algorithm and Gauss Seidel improvement.

Author

EH

Version

1.4

Date

Feb 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

[solutionMDP](#) object.

also called hybrid value iteration in Powell

4.3.3.7 valueIteration()

```
solutionMDP * discountedMDP::valueIteration (
    double epsilon,
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm.

Author

EH

Version

3

Date

jan 2018

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Implements [genericMDP](#).

4.3.3.8 valueIterationGS()

```
solutionMDP * discountedMDP::valueIterationGS (  
    double epsilon,  
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.

Author

EH

Version

1

Date

feb 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Warning

not fully implemented

Implements [genericMDP](#).

4.3.4 Member Data Documentation

4.3.4.1 beta

```
double discountedMDP::beta [protected]
```

discount factor

4.3.4.2 type_c

```
std::string discountedMDP::type_c [protected]
```

MDP criteria: here "infinite discounted"

The documentation for this class was generated from the following files:

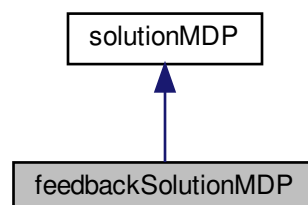
- discountedMDP.h
- discountedMDP.cpp

4.4 feedbackSolutionMDP Class Reference

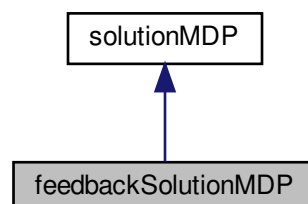
Class [feedbackSolutionMDP](#): implementation of a feedbackSolutionMDP class.

```
#include <feedbackSolutionMDP.h>
```

Inheritance diagram for feedbackSolutionMDP:



Collaboration diagram for feedbackSolutionMDP:



Public Member Functions

- [feedbackSolutionMDP](#) ()
Constructor to create feedbackSolution object.
- [~feedbackSolutionMDP](#) ()
destructor to delete feedbackSolution object.
- void [setAction](#) (int *a)
setter of the action vector
- int * [getAction](#) ()
getter of the action vector
- void [setValue](#) (double *t)
setter of the value function vector
- double * [getValue](#) ()
getter of the value function vector
- void [setActionIndex](#) (int indice, int value)
setter of only one action at index
- int [getActionIndex](#) (int indice)
getter of only one action at index
- double [getValueIndex](#) (int indice)
getter of only one value at index
- void [writeSolution](#) ()
A function to print the solution object.
- void [writeSolutionByDim](#) (int d, marmoteSet *set)
A function to print the solution object w.r.t. a dimension.

Protected Attributes

- int * [action](#)
- double * [value](#)

4.4.1 Detailed Description

Class [feedbackSolutionMDP](#): implementation of a feedbackSolutionMDP class.

This class is inherited from the abstract class [solutionMDP](#).

Author

Hyon, lip6.

Version

3

Date

jan 2018

4.4.2 Constructor & Destructor Documentation

4.4.2.1 feedbackSolutionMDP()

```
feedbackSolutionMDP::feedbackSolutionMDP ( )
```

Constructor to create feedbackSolution object.

Author

EH

Constructeur qui initialise deux vecteurs null pour les deux attributs

4.4.2.2 ~feedbackSolutionMDP()

```
feedbackSolutionMDP::~~feedbackSolutionMDP ( )
```

destructor to delete feedbackSolution object.

Author

AM EH

4.4.3 Member Function Documentation

4.4.3.1 getAction()

```
int * feedbackSolutionMDP::getAction ( )
```

getter of the action vector

Author

EH

Date

july 2019

Returns

a pointer to the array of int that code the indexes of the optimal action.

4.4.3.2 `getActionIndex()`

```
int feedbackSolutionMDP::getActionIndex (
    int indice )
```

getter of only one action at index

Author

AM EH

Date

feb 2018

Parameters

<i>indice</i>	: int index de la valeur de l'action à retourner
---------------	--

Returns

int the index of the action

4.4.3.3 `getValue()`

```
double * feedbackSolutionMDP::getValue ( )
```

getter of the value function vector

Author

EH

Date

july 2019

Returns

double * Pointeur sur un vecteur (tableau) de double de dimension le cardinal de l'espace d etat.

4.4.3.4 `getValueIndex()`

```
double feedbackSolutionMDP::getValueIndex (
    int indice )
```

getter of only one value at index

Author

EH

Date

feb 2019

Parameters

<i>indice</i>	: int index de la valeur à retourner
---------------	--------------------------------------

Returns

double the value at indice

4.4.3.5 setAction()

```
void feedbackSolutionMDP::setAction (
    int * a )
```

setter of the action vector

Author

AM EH

Date

feb 2018

Parameters

<i>a</i>	: int * . Pointeur sur un vecteur d'entiers de dimension le cardinal de l'espace d etat. Chaque entree représente l'index de l'action à effectuer quand on est dans l'etat dont l'index est la coordonnée.
----------	--

Returns

none.

4.4.3.6 setActionIndex()

```
void feedbackSolutionMDP::setActionIndex (
    int indice,
    int value )
```

setter of only one action at index

Author

AM EH

Date

feb 2018

Parameters

<i>indice</i>	: int index de la valeur de l'action à modifier
<i>value</i>	: int nouvelle valeur

Returns

none.

4.4.3.7 setValue()

```
void feedbackSolutionMDP::setValue (
    double * t )
```

setter of the value function vector

Author

AM EH

Date

feb 2018

Parameters

<i>t</i>	: double * Pointeur sur un vecteur de double de dimension le cardinal de l'espace d etat. Chaque entree représente la valeur est la valeur de la fonction de valeur dans l etat dont l'index est la coordonnée du vecteur
----------	---

Returns

none.

4.4.3.8 writeSolution()

```
void feedbackSolutionMDP::writeSolution ( ) [virtual]
```

A function to print the solution object.

Author

Abood Mourad.

Returns

none.

Reimplemented from [solutionMDP](#).

4.4.3.9 writeSolutionByDim()

```
void feedbackSolutionMDP::writeSolutionByDim (
    int d,
    marmoteSet * set ) [virtual]
```

A function to print the solution object w.r.t. a dimension.

Author

E.H.

Date

Jul 2020

Parameters

<i>d</i>	an integer giving the dimension
<i>set</i>	the stateSpace of the problem

Returns

none.

Reimplemented from [solutionMDP](#).

4.4.4 Member Data Documentation

4.4.4.1 action

```
int* feedbackSolutionMDP::action [protected]
```

A vector of actions representing the optimal policy.

4.4.4.2 value

```
double* feedbackSolutionMDP::value [protected]
```

A vector of associated values.

The documentation for this class was generated from the following files:

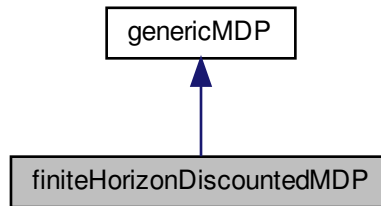
- feedbackSolutionMDP.h
- feedbackSolutionMDP.cpp

4.5 finiteHorizonDiscountedMDP Class Reference

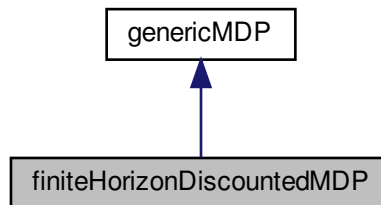
Class [finiteHorizonDiscountedMDP](#): definition of a finite horizon MDP class with discounted cost.

```
#include <finiteHorizonDiscountedMDP.h>
```

Inheritance diagram for finiteHorizonDiscountedMDP:



Collaboration diagram for finiteHorizonDiscountedMDP:



Public Member Functions

- void [writeMDP](#) ()
function providing printing function to stdout.
- [finiteHorizonDiscountedMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparse↵ Matrix *> trans, sparseMatrix *rews, int horiz, double b)
Constructor to create [finiteHorizonDiscountedMDP](#) object.
- [finiteHorizonDiscountedMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparse↵ Matrix *> trans, std::vector< sparseMatrix *> rews, int horiz, double b)
Constructor to create [finiteHorizonDiscountedMDP](#) object.
- [~finiteHorizonDiscountedMDP](#) ()
destructor to delete finiteMDP object.
- [solutionMDP](#) * [valueIteration](#) (double epsilon, int maxIter)
A function to solve (discrete time finite horizon MDP with discounted cost using DP algorithm.

- `solutionMDP * valueIteration ()`
A function to solve discrete time finite horizon MDP with discounted cost without parameters.
- `solutionMDP * valueIterationGS (double epsilon, int maxIter)`
A function that has no significance for finite horizon problem.
- `solutionMDP * policyIterationModified (double epsilon, int maxIter, double delta, int maxInIter)`
A function that has no significance for finite horizon.
- `solutionMDP * policyIteration (int maxIter)`
A function that has no significance for finite horizon.
- `double * policyCost (solutionMDP *policy, double epsilon, int maxIter)`
A function to evaluate the cost of a policy.
- `sparseMatrix * getChain (solutionMDP *policy)`
A function to build the transition matrix associated to a policy.

Protected Attributes

- int `horizon_`
- std::string `type_c`
- double `beta_`

4.5.1 Detailed Description

Class `finiteHorizonDiscountedMDP`: definition of a finite horizon MDP class with discounted cost.

Author

Hyon, lip6.

Version

0.1

Date

12th May 2020

This class is inherited from the abstract class `genericMDP`.

It describes a finite horizon MDP model with horizon and stationary transitions and stationnary costs. A discount factor is also considered

The index of the times varies between 0 and horizon-1.

We assume that the cost at index horizon is null

4.5.2 Constructor & Destructor Documentation

4.5.2.1 `finiteHorizonDiscountedMDP()` [1/2]

```
finiteHorizonDiscountedMDP::finiteHorizonDiscountedMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews,
    int horiz,
    double b )
```

Constructor to create `finiteHorizonDiscountedMDP` object.

Author

EH

Version

0.9

Date

May 2020

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure. The sparseMatrix object (state, action) an * entry of the matrix is the reward for state indexed by line and action indexed by * column.
<i>horiz</i>	: int. The horizon of the problem.
<i>b</i>	: double the discount factor

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector * is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for * state in line and action in column.

4.5.2.2 `finiteHorizonDiscountedMDP()` [2/2]

```
finiteHorizonDiscountedMDP::finiteHorizonDiscountedMDP (
    std::string r,
    marmoteSet * states,
```

```

    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews,
    int horiz,
    double b )

```

Constructor to create [finiteHorizonDiscountedMDP](#) object.

Author

EH

Version

0.9

Date

May 2020

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	vector<sparseMatrix*> vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.
<i>horiz</i>	: int. The horizon of the problem.
<i>b</i>	: double the discount factor

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

`rews` vector of sparse matrix objects: an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.5.2.3 ~finiteHorizonDiscountedMDP()

```
finiteHorizonDiscountedMDP::~~finiteHorizonDiscountedMDP ( ) [inline]
```

destructor to delete `finiteMDP` object.

Author

EH

Version

0.9

Date

jul 2019

4.5.3 Member Function Documentation

4.5.3.1 `getChain()`

```
sparseMatrix * finiteHorizonDiscountedMDP::getChain (  
    solutionMDP * policy ) [virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the policy.
---------------	--

Returns

a sparseMatrix

Implements [genericMDP](#).

4.5.3.2 policyCost()

```
double * finiteHorizonDiscountedMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [virtual]
```

A function to evaluate the cost of a policy.

Author

Hyon

Version

1

Date

May 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the optimal policy.
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

a pointer to the array of the value at the first step.

Note that all the values are filled in.

Implements [genericMDP](#).

4.5.3.3 policyIteration()

```
solutionMDP * finiteHorizonDiscountedMDP::policyIteration (
    int maxIter ) [virtual]
```

A function that has no significance for finite horizon.

Author

Hyon

Version

1

Date

jul 2020

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.5.3.4 policyIterationModified()

```
solutionMDP * finiteHorizonDiscountedMDP::policyIterationModified (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter ) [virtual]
```

A function that has no significance for finite horizon.

Author

EH

Version

1

Date

Jul 2020

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.5.3.5 valueIteration() [1/2]

```
solutionMDP * finiteHorizonDiscountedMDP::valueIteration (
    double epsilon,
    int maxIter ) [virtual]
```

A function to solve (discrete time finite horizon MDP with discounted cost using DP algorithm.

Author

EH

Version

0.9

Date

May 2020

Parameters

<i>epsilon</i>	double : no utility only for be consistent with genericMDP
<i>maxIter</i>	int : no utility only for be consistent with genericMDP

Returns

[solutionMDP](#) object.

Implements [genericMDP](#).

4.5.3.6 valueIteration() [2/2]

```
solutionMDP * finiteHorizonDiscountedMDP::valueIteration ( )
```

A function to solve discrete time finite horizon MDP with discounted cost without parameters.

Author

EH

Version

0.9

Date

May 2020

Returns

[solutionMDP](#) object. Do only a call to the generic value iteration function with dummy parameters

4.5.3.7 valueIterationGS()

```
solutionMDP * finiteHorizonDiscountedMDP::valueIterationGS (
    double epsilon,
    int maxIter ) [virtual]
```

A function that has no significance for finite horizon problem.

Author

EH

Version

1

Date

jul 2020

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.5.3.8 writeMDP()

```
void finiteHorizonDiscountedMDP::writeMDP ( ) [virtual]
```

function providing printing function to stdout.

Author

EH

Version

0.9

Date

May 2020

Reimplemented from [genericMDP](#).

4.5.4 Member Data Documentation**4.5.4.1 beta_**

```
double finiteHorizonDiscountedMDP::beta_ [protected]
```

discount factor

4.5.4.2 horizon_

```
int finiteHorizonDiscountedMDP::horizon_ [protected]
```

the time horizon>

4.5.4.3 type_c

```
std::string finiteHorizonDiscountedMDP::type_c [protected]
```

MDP criteria: here "finiteHorizonDiscounted"

The documentation for this class was generated from the following files:

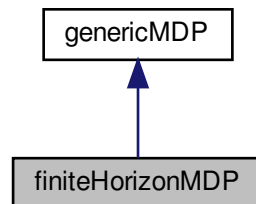
- finiteHorizonDiscountedMDP.h
- finiteHorizonDiscountedMDP.cpp

4.6 finiteHorizonMDP Class Reference

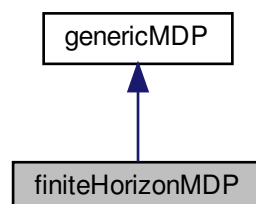
Class [finiteHorizonMDP](#): definition of a finite horizon MDP class.

```
#include <finiteHorizonMDP.h>
```

Inheritance diagram for finiteHorizonMDP:



Collaboration diagram for finiteHorizonMDP:



Public Member Functions

- void [writeMDP](#) ()
function providing printing function to stdout.
- [finiteHorizonMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, sparseMatrix *rews, int horiz)
Constructor to create [finiteHorizonMDP](#) object.
- [finiteHorizonMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews, int horiz)
Constructor to create [finiteHorizonMDP](#) object.
- [~finiteHorizonMDP](#) ()
destructor to delete finiteMDP object.
- [solutionMDP](#) * [valueIteration](#) (double epsilon, int maxIter)
A function to solve (discrete time finite horizon MDP using DP algorithm.
- [solutionMDP](#) * [valueIteration](#) ()
A function to solve discrete time finite horizon MDP without parameters.
- [solutionMDP](#) * [valueIterationGS](#) (double epsilon, int maxIter)
A function that has no significance for finite horizon problem.
- [solutionMDP](#) * [policyIterationModified](#) (double epsilon, int maxIter, double delta, int maxIter)
A function that has no significance for finite horizon.
- [solutionMDP](#) * [policyIteration](#) (int maxIter)
A function that has no significance for finite horizon.
- double * [policyCost](#) ([solutionMDP](#) *policy, double epsilon, int maxIter)
A function to evaluate the cost of a policy.
- sparseMatrix * [getChain](#) ([solutionMDP](#) *policy)
A function to build the transition matrix associated to a policy.

Protected Attributes

- int [horizon_](#)
- std::string [type_c](#)

4.6.1 Detailed Description

Class [finiteHorizonMDP](#): definition of a finite horizon MDP class.

Author

Hyon, lip6.

Version

1.1

Date

12th jan 2020

This class is inherited from the abstract class [genericMDP](#).

It describes a finite horizon MDP model with horizon and stationary transitions and costs.

The index of the times varies between 0 and horizon-1.

We assume that the cost at index horizon is null

4.6.2 Constructor & Destructor Documentation

4.6.2.1 `finiteHorizonMDP()` [1/2]

```
finiteHorizonMDP::finiteHorizonMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews,
    int horiz )
```

Constructor to create `finiteHorizonMDP` object.

Author

EH

Version

0.9

Date

jul 2019

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure. The sparseMatrix object (state, action) an * entry of the matrix is the reward for state indexed by line and action indexed by * column.
<i>horiz</i>	: int. The horizon of the problem.

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector * is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for * state in line and action in column.

4.6.2.2 finiteHorizonMDP() [2/2]

```
finiteHorizonMDP::finiteHorizonMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews,
    int horiz )
```

Constructor to create [finiteHorizonMDP](#) object.

Author

EH

Version

0.9

Date

jul 2019

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	vector<sparseMatrix*> vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.
<i>horiz</i>	: int. The horizon of the problem.

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

`rews` vector of sparse matrix objects: an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.6.2.3 ~finiteHorizonMDP()

```
finiteHorizonMDP::~~finiteHorizonMDP ( ) [inline]
```

destructor to delete `finiteMDP` object.

Author

EH

Version

0.9

Date

jul 2019

4.6.3 Member Function Documentation

4.6.3.1 `getChain()`

```
sparseMatrix * finiteHorizonMDP::getChain (  
    solutionMDP * policy ) [virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the policy.
---------------	--

Returns

a sparseMatrix

Implements [genericMDP](#).

4.6.3.2 policyCost()

```
double * finiteHorizonMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [virtual]
```

A function to evaluate the cost of a policy.

Author

Hyon

Version

1

Date

oct 2019

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the optimal policy. It should be a nonstationarySolutionMDP
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

a pointer to the array of the value at first Step.

Implements [genericMDP](#).

4.6.3.3 policyIteration()

```
solutionMDP * finiteHorizonMDP::policyIteration (
    int maxIter ) [virtual]
```

A function that has no significance for finite horizon.

Author

Hyon

Version

1

Date

jul 2019

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.6.3.4 policyIterationModified()

```
solutionMDP * finiteHorizonMDP::policyIterationModified (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter ) [virtual]
```

A function that has no significance for finite horizon.

Author

EH

Version

1

Date

Jul 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.6.3.5 valueIteration() [1/2]

```
solutionMDP * finiteHorizonMDP::valueIteration (
    double epsilon,
    int maxIter ) [virtual]
```

A function to solve (discrete time finite horizon MDP using DP algorithm.

Author

EH

Version

0.91

Date

jan 2020

Parameters

<i>epsilon</i>	double : no utility only for be consistent with genericMDP
<i>maxIter</i>	int : no utility only for be consistent with genericMDP

Returns

[solutionMDP](#) object.

Implements [genericMDP](#).

4.6.3.6 valueIteration() [2/2]

```
solutionMDP * finiteHorizonMDP::valueIteration ( )
```

A function to solve discrete time finite horizon MDP without parameters.

Author

EH

Version

1

Date

jan 2020

Returns

[solutionMDP](#) object. Do only a call to the generic value iteration function with dummy parameters

4.6.3.7 valueIterationGS()

```
solutionMDP * finiteHorizonMDP::valueIterationGS (
    double epsilon,
    int maxIter ) [virtual]
```

A function that has no significance for finite horizon problem.

Author

EH

Version

1

Date

jul 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Warning

return a NULL

Not implemented Kept only for compatibility

Implements [genericMDP](#).

4.6.3.8 writeMDP()

```
void finiteHorizonMDP::writeMDP ( ) [virtual]
```

function providing printing function to stdout.

Author

EH

Version

0.9

Date

jul 2019

Reimplemented from [genericMDP](#).

4.6.4 Member Data Documentation**4.6.4.1 horizon_**

```
int finiteHorizonMDP::horizon_ [protected]
```

the time horizon>

4.6.4.2 type_c

```
std::string finiteHorizonMDP::type_c [protected]
```

MDP criteria: here "finiteHorizon"

The documentation for this class was generated from the following files:

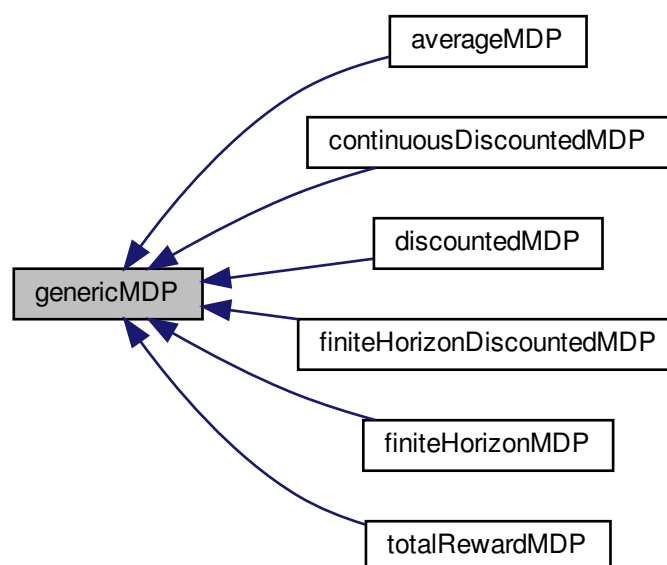
- finiteHorizonMDP.h
- finiteHorizonMDP.cpp

4.7 genericMDP Class Reference

Class [genericMDP](#) header : implementation of an abstract MDP class that represents an MDP object and can be inherited by other classes.

```
#include <genericMDP.h>
```

Inheritance diagram for genericMDP:



Public Member Functions

- virtual void [writeMDP](#) ()
virtual function providing interface framework (printing function to stdout.
- [genericMDP](#) (std::string t, std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, sparseMatrix *rews)
Constructor to create [genericMDP](#) object.
- [genericMDP](#) (std::string t, std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews)
Constructor to create [genericMDP](#) object with a vector of transition structure for rewards.
- [genericMDP](#) (std::string t, std::string r, marmoteSet *states, marmoteSet *actions, sparseMatrix *rews)
Constructor to create [genericMDP](#) object with no vector of transition in parameters.
- sparseMatrix * [cost_perStage](#) (std::vector< sparseMatrix *> rews)
A function to calculate costs (rewards) per stage.
- void [clearRew](#) ()
a function to delete reward when it has been created during the building of the object
- void [addMatrix](#) (int i, sparseMatrix *spM)
a function to add a reference to a transition matrix in a dedicated position in the vector
- void [deleMatrix](#) (int i)
a function to delete a reference in a specified position in the vector
- virtual [solutionMDP](#) * [valueIteration](#) (double epsilon, int maxIter)=0
A function to solve (discrete time) MDP using value iteration algorithm.
- virtual [solutionMDP](#) * [valueIterationGS](#) (double epsilon, int maxIter)=0
A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.
- virtual [solutionMDP](#) * [policyIterationModified](#) (double epsilon, int maxIter, double delta, int moyIter)=0
A function to solve (discrete time) MDP using modified policy iteration algorithm.
- virtual [solutionMDP](#) * [policyIteration](#) (int maxIter)=0
A function to solve (discrete time) MDP using policy iteration algorithm.
- virtual double * [policyCost](#) ([solutionMDP](#) *policy, double epsilon, int maxIter)=0
A function to evaluate the cost of a policy with iteration of the power.
- virtual sparseMatrix * [getChain](#) ([solutionMDP](#) *policy)=0
A function to build the transition matrix associated to a policy.

Protected Attributes

- std::string [type_t](#)
- std::string [type_r](#)
- marmoteSet * [stateSpace](#)
- marmoteSet * [actionSpace](#)
- std::vector< sparseMatrix * > [transitions](#)
- sparseMatrix * [rewards](#)
- bool [opMin](#)

4.7.1 Detailed Description

Class [genericMDP](#) header : implementation of an abstract MDP class that represents an MDP object and can be inherited by other classes.

Author

Hyon, lip6

Date

18 feb 2019

Version

1.4

This class is generic, it contains the general elements of MDP: MDP type : time type, optimization rule, state space, action space, action space, list of transition probabilities, list of transition rewards.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 [genericMDP\(\)](#) [1/3]

```
genericMDP::genericMDP (
    std::string t,
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews )
```

Constructor to create [genericMDP](#) object.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>t</i>	string : type_t value : "discrete" , "continuous".
<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an entry of the matrix is the reward for state line and action column.

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for state line and action column.

4.7.2.2 genericMDP() [2/3]

```
genericMDP::genericMDP (
    std::string t,
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews )
```

Constructor to create [genericMDP](#) object with a vector of transition structure for rewards.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>t</i>	: string, type_t values : "discrete" , "continuous".
<i>r</i>	: string, type_r rule can be "min" , "max".
<i>states</i>	: marmoteSet. State space
<i>actions</i>	: marmoteSet. Action space
<i>trans</i>	: vector<sparseMatrix*> transition structures vector of sparseMatrix.
<i>rews</i>	: vector<sparseMatrix*> : vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.

Returns

none.

this second constructor take an other form reward description. It corresponds with the theoretical model where reward depends on transition $r(i,a,j)$. Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

`rews` : vector of sparse matrix objects an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.7.2.3 genericMDP() [3/3]

```
genericMDP::genericMDP (
    std::string t,
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    sparseMatrix * rews )
```

Constructor to create [genericMDP](#) object with no vector of transition in parameters.

Author

Hyon

Version

1

Date

june 2020

Parameters

<i>t</i>	: string, type_t values : "discrete" , "continuous".
<i>r</i>	: string, type_r rule can be "min" , "max".
<i>states</i>	: marmoteSet. State space
<i>actions</i>	: marmoteSet. Action space
<i>rews</i>	: reward : sparseMatrix object (state, action) an entry of the matrix is the reward for state line and action column.

Returns

none.

this third constructor does not take any vector of transition. The vector is created during this constructor

`rews` : reward structure : (state, action) an entry of the sparseMatrix is the reward for state line and action column.

4.7.3 Member Function Documentation

4.7.3.1 addMatrix()

```
void genericMDP::addMatrix (
    int i,
    sparseMatrix * spM )
```

a function to add a reference to a transition matrix in a dedicated position in the vector

Author

EH

Date

apr 2020

Version

0.1

Parameters

<i>i</i>	an integer which is the index of the position of the sparse matrix
<i>spM</i>	a sparseMatrix

4.7.3.2 clearRew()

```
void genericMDP::clearRew ( )
```

a function to delete reward when it has been created during the building of the object

Author

EH

Date

mar 2018

4.7.3.3 cost_perStage()

```
sparseMatrix * genericMDP::cost_perStage (
    std::vector< sparseMatrix *> rews )
```

A function to calculate costs (rewards) per stage.

Author

Abood Mourad.

Version

2

Date

jan 2016

Parameters

<i>rews</i>	vector<sparseMatrix*> : for reward vector of transitionStructure objects (state, action, state), one reward matrix for each action.
-------------	---

Returns

sparseMatrix pointer.

this method transforms this vector in a transition matrix where first entry is the initial state and second entry is the action.

4.7.3.4 deleMatrix()

```
void genericMDP::deleMatrix (
    int i )
```

a function to delete a reference in a specified position in the vector

Author

EH

Date

oct 2020

Version

0.1

Parameters

<i>i</i>	an integer which is the index of the position of the sparse matrix to delete
----------	--

4.7.3.5 getChain()

```
virtual sparseMatrix* genericMDP::getChain (
    solutionMDP * policy ) [pure virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the policy.
---------------	--

Returns

a sparseMatrix

Implemented in [averageMDP](#), [discountedMDP](#), [finiteHorizonDiscountedMDP](#), [finiteHorizonMDP](#), and [totalRewardMDP](#).

4.7.3.6 policyCost()

```
virtual double* genericMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [pure virtual]
```

A function to evaluate the cost of a policy with iteration of the power.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the optimal policy.
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

double.

Implemented in [discountedMDP](#), [averageMDP](#), [finiteHorizonDiscountedMDP](#), [finiteHorizonMDP](#), and [totalRewardMDP](#).

4.7.3.7 policyIteration()

```
virtual solutionMDP\* genericMDP::policyIteration (
    int maxIter ) [pure virtual]
```

A function to solve (discrete time) MDP using policy iteration algorithm.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns

[solutionMDP](#) object.

Implemented in [discountedMDP](#), [finiteHorizonDiscountedMDP](#), [averageMDP](#), [finiteHorizonMDP](#), and [totalRewardMDP](#).

4.7.3.8 policyIterationModified()

```
virtual solutionMDP* genericMDP::policyIterationModified (
    double epsilon,
    int maxIter,
    double delta,
    int moyIter ) [pure virtual]
```

A function to solve (discrete time) MDP using modified policy iteration algorithm.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>epsilon</i>	double.
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>moyIter</i>	nb iter max in the inner loop

Returns

[solutionMDP](#) object.

Implemented in [averageMDP](#), [finiteHorizonDiscountedMDP](#), [totalRewardMDP](#), [finiteHorizonMDP](#), and [discountedMDP](#).

4.7.3.9 valueIteration()

```
virtual solutionMDP* genericMDP::valueIteration (
    double epsilon,
    int maxIter ) [pure virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm.

Author

Hyon

Version

3

Date

jan 2018

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Implemented in [averageMDP](#), [discountedMDP](#), [totalRewardMDP](#), [finiteHorizonDiscountedMDP](#), and [finiteHorizonMDP](#).

4.7.3.10 valueIterationGS()

```
virtual solutionMDP* genericMDP::valueIterationGS (
    double epsilon,
    int maxIter ) [pure virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.

Author

Hyon

Version

1

Date

feb 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Implemented in [averageMDP](#), [finiteHorizonDiscountedMDP](#), [finiteHorizonMDP](#), [discountedMDP](#), and [totalRewardMDP](#).

4.7.4 Member Data Documentation

4.7.4.1 actionSpace

```
marmoteSet* genericMDP::actionSpace [protected]
```

Action space represented by marmoteSet object.

4.7.4.2 opMin

```
bool genericMDP::opMin [protected]
```

the translation of the type for minimizing or maximizing. Equals true if the aim is to minimizing.

4.7.4.3 rewards

```
sparseMatrix* genericMDP::rewards [protected]
```

a rewards (costs) structure (row : state, column : action)

4.7.4.4 stateSpace

```
marmoteSet* genericMDP::stateSpace [protected]
```

State space represented by marmoteSet object.

4.7.4.5 transitions

```
std::vector<sparseMatrix*> genericMDP::transitions [protected]
```

List (vector) of transitionStructure (one transition matrix for each action).

4.7.4.6 type_r

```
std::string genericMDP::type_r [protected]
```

MDP rule: "min" , "max".

4.7.4.7 type_t

```
std::string genericMDP::type_t [protected]
```

MDP type: "discrete" , "continuous".

The documentation for this class was generated from the following files:

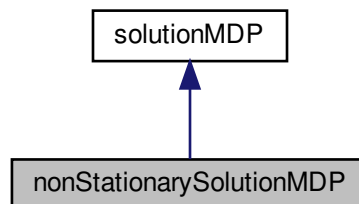
- genericMDP.h
- genericMDP.cpp

4.8 nonStationarySolutionMDP Class Reference

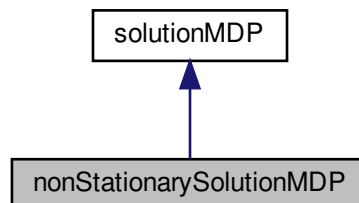
Class [nonStationarySolutionMDP](#): implementation of a [nonStationarySolutionMDP](#) class.

```
#include <nonStationarySolutionMDP.h>
```

Inheritance diagram for nonStationarySolutionMDP:



Collaboration diagram for nonStationarySolutionMDP:



Public Member Functions

- [nonStationarySolutionMDP](#) (int h, int size)
Constructor to create feedbackSolution object.
- [~nonStationarySolutionMDP](#) ()
destructor to delete feedbackSolution object.
- void [writeSolution](#) ()
A function to print the solution object.
- void [setAction](#) (int **a)
setter of the action vector
- void [setActionAtStep](#) (int step, int *t)
setter of the optimal action vector at step t
- void [setActionAtStepIndex](#) (int step, int indice, int action_op)
setter of only one action at index at step t
- int ** [getAction](#) ()
getter of the action vector
- int * [getActionAtStep](#) (int step)
getter of only one action vector at step t
- int [getActionAtStepIndex](#) (int step, int index)
getter of only one action at index k at step t
- void [setValue](#) (double **t)
setter of the value function vector
- void [setValueAtStep](#) (int step, double *t)
setter of the value function vector at step t
- void [setValueAtStepIndex](#) (int step, int indice, double value_op)
setter of only one value at index at step t
- double ** [getValue](#) ()
getter of the value functions vector
- double * [getValueAtStep](#) (int step)
getter of a vector at a given step
- double [getValueAtStepIndex](#) (int step, int index)
getter of only one value at index at step t
- int [getHorizon](#) ()
getter of the horizon
- void [writeSolutionByDim](#) (int d, marmoteSet *set)
A function to print the solution object w.r.t. a dimension.

Protected Attributes

- int [horizon_](#)
- int ** [action](#)
- double ** [value](#)

4.8.1 Detailed Description

Class `nonStationarySolutionMDP`: implementation of a `nonStationarySolutionMDP` class.

Author

Hyon lip6 2019.

Version

0.3

Date

july 2019

This class is inherited from the class `solutionMDP`.

It contains a data structure (here an array) with two entries one for the horizon and the other for the state.

The horizon is the size of one dimension of the array. The index of the time varies between 0 and horizon-1.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 `nonStationarySolutionMDP()`

```
nonStationarySolutionMDP::nonStationarySolutionMDP (
    int h,
    int size )
```

Constructor to create feedbackSolution object.

Author

EH

Version

0.2

Date

jan 2019

Parameters

<i>h</i>	: int l'horizon du probleme fini
<i>size</i>	: int la taille de l'espace d'etat

Constructor which initialize two empty arrays pour les deux attributs. One double dimension array for the action and one double dimension array for the values

4.8.2.2 ~nonStationarySolutionMDP()

```
nonStationarySolutionMDP::~~nonStationarySolutionMDP ( )
```

destructor to delete feedbackSolution object.

Author

AM EH

Version

0.2

Date

jan 2019

4.8.3 Member Function Documentation

4.8.3.1 getAction()

```
int ** nonStationarySolutionMDP::getAction ( )
```

getter of the action vector

Author

EH

Date

july 2019

Version

0.1

Returns

the pointer to the array

This method returns the double entries array

4.8.3.2 `getActionAtStep()`

```
int * nonStationarySolutionMDP::getActionAtStep (
    int step )
```

getter of only one action vector at step t

Author

EH

Date

jan 2019

Version

0.1

Parameters

<i>step</i>	: int the step for getting the vector
-------------	---------------------------------------

Returns

int * the value vector of the index of the actions

4.8.3.3 `getActionAtStepIndex()`

```
int nonStationarySolutionMDP::getActionAtStepIndex (
    int step,
    int index )
```

getter of only one action at index k at step t

Author

EH

Date

May 2020

Version

0.1

Parameters

<i>step</i>	: int the step for getting the action
<i>index</i>	: int the index for

Returns

int the action of the index at step

4.8.3.4 getHorizon()

```
int nonStationarySolutionMDP::getHorizon ( )
```

getter of the horizon

Author

EH

Date

oct 2020

Version

0.1

Returns

int the horizon

4.8.3.5 getValue()

```
double ** nonStationarySolutionMDP::getValue ( )
```

getter of the value functions vector

Author

EH

Date

july 2019

Version

0.1

Returns

the array of value functions

4.8.3.6 `getValueAtStep()`

```
double * nonStationarySolutionMDP::getValueAtStep (
    int step )
```

getter of a vector at a given step

Author

EH

Date

sept 2019

Version

0.2

Parameters

<i>step</i>	: int the step for getting the vector of value at step
-------------	--

Returns

double * the vector of the value at step

4.8.3.7 `getValueAtStepIndex()`

```
double nonStationarySolutionMDP::getValueAtStepIndex (
    int step,
    int index )
```

getter of only one value at index at step t

Author

EH

Date

jan 2020

Version

0.1

Parameters

<i>step</i>	: int the step for getting the vector
<i>index</i>	: int the index for which we want the value

Returns

double the value at step and index

4.8.3.8 setAction()

```
void nonStationarySolutionMDP::setAction (
    int ** a )
```

setter of the action vector

Author

EH

Date

jul 2019

Version

0.2

Parameters

<i>a</i>	: int **. Pointeur sur un tableau de vecteur d'entiers de dimension l'horizon, chaque vecteur etant de dimension le cardinal de l'espace d etat.
----------	--

This method is a setter for the double entries array

4.8.3.9 setActionAtStep()

```
void nonStationarySolutionMDP::setActionAtStep (
    int step,
    int * t )
```

setter of the optimal action vector at step t

Author

EH

Date

sep 2019

Version

0.2

Parameters

<i>step</i>	: int the step for including the vector
<i>t</i>	: int * Pointer on an integer vector of double (whose dimension should be the state space cardinal)

Chaque entree représente l'action optimale à l'entree dont l'index est la coordonnée du vecteur

4.8.3.10 setActionAtStepIndex()

```
void nonStationarySolutionMDP::setActionAtStepIndex (
    int step,
    int indice,
    int action_op )
```

setter of only one action at index at step t

Author

EH

Date

sept 2019

Version

0.2

Parameters

<i>step</i>	: int the step for including the vector
<i>indice</i>	: int index de la valeur de l'action à modifier
<i>action_op</i>	: int nouvelle valeur de la solution optimale à l'index

4.8.3.11 setValue()

```
void nonStationarySolutionMDP::setValue (
    double ** t )
```

setter of the value function vector

Author

EH

Date

july 2019

Version

0.2

Parameters

<i>t</i>	: double ** pointer. Pointer on an double dimension array with size l times h. Where the first dimension is the horizon and the second dimension is the state space cardinal.
----------	---

4.8.3.12 setValueAtStep()

```
void nonStationarySolutionMDP::setValueAtStep (
    int step,
    double * t )
```

setter of the value function vector at step t

Author

EH

Date

jan 2019

Version

0.1

Parameters

<i>step</i>	: int the step for including the vector
<i>t</i>	: double * Pointeur sur un vecteur de double de dimension le cardinal de l'espace d etat.

Chaque entree représente la valeur de la fonction de valeur dans l etat dont l'index est la coordonnée du vecteur

4.8.3.13 setValueAtStepIndex()

```
void nonStationarySolutionMDP::setValueAtStepIndex (
```

```

    int step,
    int indice,
    double value_op )

```

setter of only one value at index at step t

Author

EH

Date

sept 2019

Version

0.2

Parameters

<i>step</i>	: int the step for including the vector
<i>indice</i>	: int index de la valeur de l'action à modifier
<i>value_op</i>	: double nouvelle valeur de l'index de la valeur optimale

4.8.3.14 writeSolution()

```
void nonStationarySolutionMDP::writeSolution ( ) [virtual]
```

A function to print the solution object.

Author

AM EH

Version

0.2

Date

jan 2019

Reimplemented from [solutionMDP](#).

4.8.3.15 writeSolutionByDim()

```
void nonStationarySolutionMDP::writeSolutionByDim (
    int d,
    marmoteSet * set ) [virtual]
```

A function to print the solution object w.r.t. a dimension.

Author

E.H.

Date

Jul 2020

Parameters

<i>d</i>	an integer giving the dimension wrt to the variation
<i>set</i>	is the stateSpace

Returns

none.

Reimplemented from [solutionMDP](#).

4.8.4 Member Data Documentation

4.8.4.1 action

```
int** nonStationarySolutionMDP::action [protected]
```

An array whose columns are the index of actions representing the optimal policy.

4.8.4.2 horizon_

```
int nonStationarySolutionMDP::horizon_ [protected]
```

the time horizon>

4.8.4.3 value

```
double** nonStationarySolutionMDP::value [protected]
```

An array whose columns are the value at each step.

The documentation for this class was generated from the following files:

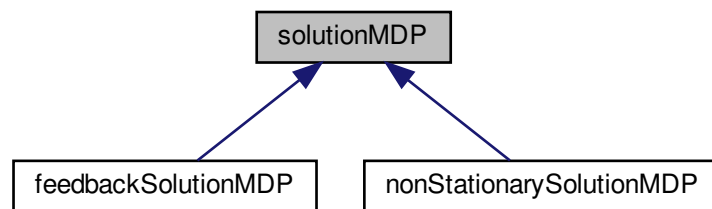
- nonStationarySolutionMDP.h
- nonStationarySolutionMDP.cpp

4.9 solutionMDP Class Reference

Class [solutionMDP](#): implementation of an abstract [solutionMDP](#) class.

```
#include <solutionMDP.h>
```

Inheritance diagram for solutionMDP:



Public Member Functions

- [solutionMDP](#) ()
Constructor to create [solutionMDP](#) object.
- virtual [~solutionMDP](#) ()
the destructor of the object [solutionMDP](#)
- virtual void [writeSolution](#) ()
A function to print the solution object.
- virtual void [writeSolutionByDim](#) (int d, marmoteSet *set)
A function to print the solution object w.r.t. a dimension.
- void [setSize](#) (int s)
setter of the size.

Protected Attributes

- int [size_](#)

4.9.1 Detailed Description

Class [solutionMDP](#): implementation of an abstract [solutionMDP](#) class.

This class is to be inherited by other classes.

Author

Abood Mourad, and E. Hyon, lip6

Date

18 jan 2018

Version

2

4.9.2 Constructor & Destructor Documentation

4.9.2.1 [solutionMDP\(\)](#)

```
solutionMDP::solutionMDP ( )
```

Constructor to create [solutionMDP](#) object.

Author

Hyon

Version

1

Date

feb 2018

4.9.2.2 ~solutionMDP()

```
solutionMDP::~~solutionMDP ( ) [virtual]
```

the destructor of the object [solutionMDP](#)

Author

EH

Version

1

Date

feb 2018

4.9.3 Member Function Documentation

4.9.3.1 setSize()

```
void solutionMDP::setSize (
    int s )
```

setter of the size.

Author

EH

Date

mar 2018

Returns

none.

4.9.3.2 writeSolution()

```
void solutionMDP::writeSolution ( ) [virtual]
```

A function to print the solution object.

Author

Abood Mourad.

Date

feb 2018

Returns

none.

Reimplemented in [feedbackSolutionMDP](#), and [nonStationarySolutionMDP](#).

4.9.3.3 writeSolutionByDim()

```
void solutionMDP::writeSolutionByDim (
    int d,
    marmoteSet * set ) [virtual]
```

A function to print the solution object w.r.t. a dimension.

Author

E.H.

Date

Jul 2020

Parameters

<i>d</i>	an integer giving the dimension
<i>set</i>	a pointer to an object describing the stateSpace w.r.t we enumerate the states

Returns

none.

Reimplemented in [nonStationarySolutionMDP](#), and [feedbackSolutionMDP](#).

4.9.4 Member Data Documentation

4.9.4.1 size_

```
int solutionMDP::size_ [protected]
```

_size of the solution

The documentation for this class was generated from the following files:

- solutionMDP.h
- solutionMDP.cpp

4.10 structuralPropertiesPol Class Reference

Class [structuralPropertiesVF](#): implementation of a class to test structural Properties of Value Function.

```
#include <structuralPropertiesPol.h>
```

Public Member Functions

- [structuralPropertiesPol](#) (marmoteSet *st)
Constructor to create class to check properties of policy.
- [~structuralPropertiesPol](#) ()
destructor to delete feedbackSolution object.
- int [monotonicityPol](#) (feedbackSolutionMDP *solution)
A generic function to evaluate monotonicity of the policy.
- int [monotonicityPol](#) (nonStationarySolutionMDP *solution)
A generic function to evaluate monotonicity of the policy.
- int [monotonicityPolByDim](#) (feedbackSolutionMDP *solution, int dim)
A generic function to evaluate monotonicity of the policy by dimension.
- int [monotonicityPolByDim](#) (nonStationarySolutionMDP *solution, int dim)
A generic function to evaluate monotonicity of the policy by dimension.
- int [thresholdPol](#) (feedbackSolutionMDP *solution)
A generic function to evaluate if the policy has threshold type.
- int [thresholdPol](#) (nonStationarySolutionMDP *solution)
A generic function to evaluate if the policy has threshold type.
- bool [sSPol](#) (feedbackSolutionMDP *solution)
A generic function to test if the policy is sS.
- bool [sSPol](#) (nonStationarySolutionMDP *solution)
A generic function to test if the policy is sS.
- bool [sSPolbyDim](#) (feedbackSolutionMDP *solution)
A generic function to test if the policy is sS following a dimension.
- bool [sSPolbyDim](#) (nonStationarySolutionMDP *solution)
A generic function to test if the policy is sS following a dimension.

Protected Attributes

- marmoteSet * [space_](#)

4.10.1 Detailed Description

Class [structuralPropertiesVF](#): implementation of a class to test structural Properties of Value Function.

Author

Hyon 2020.

Version

0.1

Date

oct 2020

4.10.2 Constructor & Destructor Documentation

4.10.2.1 structuralPropertiesPol()

```
structuralPropertiesPol::structuralPropertiesPol (
    marmoteSet * st )
```

Constructor to create class to check properties of policy.

Author

Hyon

Version

1

Date

jul 2020

Parameters

<i>st</i>	: state space set
-----------	-------------------

Returns

none.

4.10.2.2 ~structuralPropertiesPol()

```
structuralPropertiesPol::~~structuralPropertiesPol ( ) [inline]
```

destructor to delete feedbackSolution object.

Author

EH

Version

0.3

Date

oct 2020

4.10.3 Member Function Documentation**4.10.3.1 monotonicityPol()** [1/2]

```
int structuralPropertiesPol::monotonicityPol (
    feedbackSolutionMDP * solution )
```

A generic function to evaluate monotonicity of the policy.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if the policy is increasing -1 decreasing and 0 otherwise

version for testing feedback solution

4.10.3.2 monotonicityPol() [2/2]

```
int structuralPropertiesPol::monotonicityPol (
    nonStationarySolutionMDP * solution )
```

A generic function to evaluate monotonicity of the policy.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if the policy is increasing -1 decreasing and 0 otherwise

version for testing non stationary solution

4.10.3.3 monotonicityPolByDim() [1/2]

```
int structuralPropertiesPol::monotonicityPolByDim (
    feedbackSolutionMDP * solution,
    int dim )
```

A generic function to evaluate monotonicity of the policy by dimension.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if the policy is increasing.

For feedback solution

4.10.3.4 monotonicityPolByDim() [2/2]

```
int structuralPropertiesPol::monotonicityPolByDim (
    nonStationarySolutionMDP * solution,
    int dim )
```

A generic function to evaluate monotonicity of the policy by dimension.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if the policy is increasing.

For feedback solution

4.10.3.5 sSPol() [1/2]

```
bool structuralPropertiesPol::sSPol (
    feedbackSolutionMDP * solution )
```

A generic function to test if the policy is sS.

Author

Hyon

Version

0.1

Date

dec 2020

Returns

bool

version for testing feedback solution

4.10.3.6 sSPol() [2/2]

```
bool structuralPropertiesPol::sSPol (
    nonStationarySolutionMDP * solution )
```

A generic function to test if the policy is sS.

Author

Hyon

Version

0.1

Date

dec 2020

Returns

bool

version for testing non stationary solution

4.10.3.7 sSPolbyDim() [1/2]

```
bool structuralPropertiesPol::sSPolbyDim (
    feedbackSolutionMDP * solution )
```

A generic function to test if the policy is sS following a dimension.

Author

Hyon

Version

0.1

Date

dec 2020

Returns

bool

version for testing feedback solution

4.10.3.8 sSPolbyDim() [2/2]

```
bool structuralPropertiesPol::sSPolbyDim (
    nonStationarySolutionMDP * solution )
```

A generic function to test if the policy is sS following a dimension.

Author

Hyon

Version

0.1

Date

dec 2020

Returns

bool

version for testing non stationary solution

4.10.3.9 thresholdPol() [1/2]

```
int structuralPropertiesPol::thresholdPol (
    feedbackSolutionMDP * solution )
```

A generic function to evaluate if the policy has threshold type.

Author

Hyon

Version

0.0

Date

nov 2020

Returns

int value

version for testing feedback solution

4.10.3.10 thresholdPol() [2/2]

```
int structuralPropertiesPol::thresholdPol (
    nonStationarySolutionMDP * solution )
```

A generic function to evaluate if the policy has threshold type.

Author

Hyon

Version

0.0

Date

nov 2020

Returns

int value is 1 if the policy is increasing -1 decreasing and 0 otherwise

version for testing non stationary solution

4.10.4 Member Data Documentation

4.10.4.1 space_

```
marmoteSet* structuralPropertiesPol::space_ [protected]
```

State space represented by marmoteSet object.

The documentation for this class was generated from the following files:

- structuralPropertiesPol.h
- structuralPropertiesPol.cpp

4.11 structuralPropertiesVF Class Reference

Class [structuralPropertiesVF](#): implementation of a class to test structural Properties of Value Function.

```
#include <structuralPropertiesVF.h>
```

Public Member Functions

- [structuralPropertiesVF](#) (marmoteSet *st)
Constructor to create class to check properties.
- [~structuralPropertiesVF](#) ()
destructor to delete feedbackSolution object.
- int [monotonicityVF](#) (feedbackSolutionMDP *solution)
A generic function to evaluate monotonicity of value function.
- int [monotonicityVF](#) (nonStationarySolutionMDP *solution)
A generic function to evaluate monotonicity of value function.
- int [monotonicityVFByDim](#) (feedbackSolutionMDP *solution, int dim)
A generic function to evaluate monotonicity of value function by dimension.
- int [monotonicityVFByDim](#) (nonStationarySolutionMDP *solution, int dim)
A generic function to evaluate monotonicity of value function by dimension.
- int [monotonicityCX](#) (feedbackSolutionMDP *solution)
A generic function to evaluate convexity of value function.
- int [monotonicityCX](#) (feedbackSolutionMDP *solution, int dim)
A generic function to evaluate convexity of value function.

Protected Attributes

- marmoteSet * [space_](#)

4.11.1 Detailed Description

Class [structuralPropertiesVF](#): implementation of a class to test structural Properties of Value Function.

Author

Hyon 2020.

Version

0.1

Date

oct 2020

4.11.2 Constructor & Destructor Documentation

4.11.2.1 structuralPropertiesVF()

```
structuralPropertiesVF::structuralPropertiesVF (
    marmoteSet * st )
```

Constructor to create class to check properties.

Author

Hyon

Version

1

Date

jul 2020

Parameters

<i>st</i>	: state space set
-----------	-------------------

Returns

none.

4.11.2.2 ~structuralPropertiesVF()

```
structuralPropertiesVF::~~structuralPropertiesVF ( ) [inline]
```

destructor to delete feedbackSolution object.

Author

EH

Version

0.3

Date

oct 2020

4.11.3 Member Function Documentation

4.11.3.1 monotonicityCX() [1/2]

```
int structuralPropertiesVF::monotonicityCX (
    feedbackSolutionMDP * solution )
```

A generic function to evaluate convexity of value function.

Author

Hyon

Version

0.1

Date

nov 2020

Returns

int value is 1 if value function is convex -1 concave and 0 otherwise

version for testing feedback solution

4.11.3.2 monotonicityCX() [2/2]

```
int structuralPropertiesVF::monotonicityCX (
    feedbackSolutionMDP * solution,
    int dim )
```

A generic function to evaluate convexity of value function.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if value function is convex -1 concave and 0 otherwise

version for testing feedback solution

4.11.3.3 monotonicityVF() [1/2]

```
int structuralPropertiesVF::monotonicityVF (
    feedbackSolutionMDP * solution )
```

A generic function to evaluate monotonicity of value function.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if value function is increasing -1 decreasing and 0 otherwise

version for testing feedback solution

4.11.3.4 monotonicityVF() [2/2]

```
int structuralPropertiesVF::monotonicityVF (
    nonStationarySolutionMDP * solution )
```

A generic function to evaluate monotonicity of value function.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if value function is increasing -1 decreasing and 0 otherwise

version for testing non stationary solution

4.11.3.5 monotonicityVFByDim() [1/2]

```
int structuralPropertiesVF::monotonicityVFByDim (
    feedbackSolutionMDP * solution,
    int dim )
```

A generic function to evaluate monotonicity of value function by dimension.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if value function is increasing.

For feedback solution

4.11.3.6 monotonicityVFByDim() [2/2]

```
int structuralPropertiesVF::monotonicityVFByDim (
    nonStationarySolutionMDP * solution,
    int dim )
```

A generic function to evaluate monotonicity of value function by dimension.

Author

Hyon

Version

0.1

Date

oct 2020

Returns

int value is 1 if value function is increasing.

For feedback solution

4.11.4 Member Data Documentation

4.11.4.1 space_

```
marmoteSet* structuralPropertiesVF::space_ [protected]
```

State space represented by marmoteSet object.

The documentation for this class was generated from the following files:

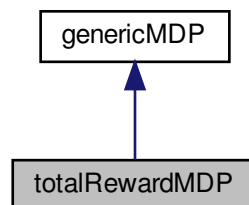
- structuralPropertiesVF.h
- structuralPropertiesVF.cpp

4.12 totalRewardMDP Class Reference

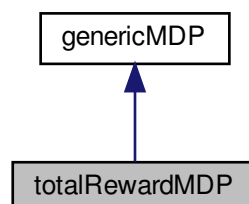
Class [totalRewardMDP](#) header: definition of an infinite horizon total reward MDP class.

```
#include <totalRewardMDP.h>
```

Inheritance diagram for totalRewardMDP:



Collaboration diagram for totalRewardMDP:



Public Member Functions

- [totalRewardMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, sparseMatrix *rews)
Constructor to create a [totalRewardMDP](#) object.
- [totalRewardMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, std::vector< sparseMatrix *> trans, std::vector< sparseMatrix *> rews)
Constructor to create [totalRewardMDP](#) object.
- [totalRewardMDP](#) (std::string r, marmoteSet *states, marmoteSet *actions, sparseMatrix *rews)
Constructor to create [totalRewardMDP](#) object with no vector of transition in parameters.
- virtual [~totalRewardMDP](#) ()
destructor to delete [discountedMDP](#) object.
- void [writeMDP](#) ()
function providing printing function to stdout.
- [solutionMDP](#) * [valueIteration](#) (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm.
- [solutionMDP](#) * [valueIterationGS](#) (double epsilon, int maxIter)
A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.
- [solutionMDP](#) * [policyIteration](#) (int maxIter)
A function to solve (discrete time) MDP using policy iteration algorithm.
- [solutionMDP](#) * [policyIterationModified](#) (double epsilon, int maxIter, double delta, int maxIter)
A function to solve (discrete time) MDP using modified policy iteration algorithm.
- double * [policyCost](#) ([solutionMDP](#) *policy, double epsilon, int maxIter)
A function to evaluate the cost of a policy with iteration of the power.
- sparseMatrix * [getChain](#) ([solutionMDP](#) *policy)
A function to build the transition matrix associated to a policy.

Protected Attributes

- std::string [type_c](#)

4.12.1 Detailed Description

Class [totalRewardMDP](#) header: definition of an infinite horizon total reward MDP class.

Author

Emmanuel Hyon

Version

0.1

Date

jan 2019

This class is inherited from the abstract class [genericMDP](#).

4.12.2 Constructor & Destructor Documentation

4.12.2.1 totalRewardMDP() [1/3]

```
totalRewardMDP::totalRewardMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    sparseMatrix * rews )
```

Constructor to create a [totalRewardMDP](#) object.

Author

EH

Version

0.1

Date

dec 2018

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	sparseMatrix : reward structure : sparseMatrix object (state, action) an entry of the matrix is the reward for state indexed by line and action indexed by * column.

Returns

none.

trans : transition structures vector of sparseMatrix objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

reward structure : (state, action) an entry of the sparseMatrix is the reward for state in line and action in column.

4.12.2.2 totalRewardMDP() [2/3]

```
totalRewardMDP::totalRewardMDP (
    std::string r,
```

```

    marmoteSet * states,
    marmoteSet * actions,
    std::vector< sparseMatrix *> trans,
    std::vector< sparseMatrix *> rews )

```

Constructor to create [totalRewardMDP](#) object.

Author

EH

Version

1

Date

dec 2018

Parameters

<i>r</i>	string : type_r rule value can be "min" , "max".
<i>states</i>	marmoteSet : State space
<i>actions</i>	marmoteSet : Action space
<i>trans</i>	vector<sparseMatrix*> : transition structures vector of sparseMatrix.
<i>rews</i>	vector<sparseMatrix*> : vector of sparse matrix objects an entry of the vector is the reward structure from state to state for the action.

Returns

none.

This second constructor take an other form of reward description. It corresponds with the theoretical model where reward depends on transition and reached state : $r(i,a,j)$ Then `cost_perStage` method is used to be consistent with the reward attribute

`trans` : transition structures vector of `sparseMatrix` objects an entry of the vector is a transition structure matrix from state to state for the action at vector index.

`rews` vector of `sparse matrix` objects an entry of the vector is the reward structure from state to state when trigger action (action is the index of the vector).

4.12.2.3 totalRewardMDP() [3/3]

```

totalRewardMDP::totalRewardMDP (
    std::string r,
    marmoteSet * states,
    marmoteSet * actions,
    sparseMatrix * rews )

```

Constructor to create [totalRewardMDP](#) object with no vector of transition in parameters.

Author

Hyon

Version

1

Date

june 2020

Parameters

<i>r</i>	: string, type_r rule can be "min" , "max".
<i>states</i>	: marmoteSet. State space
<i>actions</i>	: marmoteSet. Action space
<i>rewards</i>	: reward : sparseMatrix object (state, action) an entry of the matrix is the reward for state line and action column.

Returns

none.

this third constructor does not take any vector of transition.

rewards : reward structure : (state, action) an entry of the sparseMatrix is the reward for state line and action column.

4.12.2.4 ~totalRewardMDP()

```
totalRewardMDP::~~totalRewardMDP ( ) [virtual]
```

destructor to delete [discountedMDP](#) object.

Author

EH

4.12.3 Member Function Documentation

4.12.3.1 `getChain()`

```
sparseMatrix * totalRewardMDP::getChain (
    solutionMDP * policy ) [virtual]
```

A function to build the transition matrix associated to a policy.

Author

Hyon

Version

0.1

Date

dec 2020

Parameters

<i>policy</i>	solutionMDP* : pointer to a solution that contains the policy.
---------------	--

Returns

a sparseMatrix

Implements [genericMDP](#).

4.12.3.2 `policyCost()`

```
double * totalRewardMDP::policyCost (
    solutionMDP * policy,
    double epsilon,
    int maxIter ) [virtual]
```

A function to evaluate the cost of a policy with iteration of the power.

Author

Hyon

Version

0.1

Date

april 2019

Parameters

<i>policy</i>	solutionMDP : object to get of the action
<i>maxIter</i>	int : the maximum number of iterations.
<i>epsilon</i>	double.

Returns

double.

Implements [genericMDP](#).

4.12.3.3 policyIteration()

```
solutionMDP * totalRewardMDP::policyIteration (  
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using policy iteration algorithm.

Author

Hyon

Version

0.1

Date

feb 2019

Parameters

<i>maxIter</i>	int : the maximum number of iterations.
----------------	---

Returns

[solutionMDP](#) object.

Warning

not fully implemented

Implements [genericMDP](#).

4.12.3.4 policyIterationModified()

```
solutionMDP * totalRewardMDP::policyIterationModified (
    double epsilon,
    int maxIter,
    double delta,
    int maxInIter ) [virtual]
```

A function to solve (discrete time) MDP using modified policy iteration algorithm.

Author

Hyon

Version

0.1

Date

Feb 2019

Parameters

<i>epsilon</i>	double the precision in the outer loop
<i>maxIter</i>	int : the maximum number of iterations.
<i>delta</i>	precision in the inner loop
<i>maxInIter</i>	nb iter max in the inner loop

Returns

`solutionMDP` object.

also called hybrid value iteration in Powell

Implements `genericMDP`.

4.12.3.5 valueIteration()

```
solutionMDP * totalRewardMDP::valueIteration (
    double epsilon,
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm.

Author

Hyon

Version

0.1

Date

feb 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.Implements [genericMDP](#).

4.12.3.6 valueIterationGS()

```
solutionMDP * totalRewardMDP::valueIterationGS (  
    double epsilon,  
    int maxIter ) [virtual]
```

A function to solve (discrete time) MDP using value iteration algorithm with Gauss Seidel improvement.

Author

Hyon

Version

0.1

Date

feb 2019

Parameters

<i>epsilon</i>	double precision of the solution
<i>maxIter</i>	int : the maximum number of iterations.

Returns

[solutionMDP](#) object.

Implements [genericMDP](#).

4.12.4 Member Data Documentation

4.12.4.1 type_c

```
std::string totalRewardMDP::type_c [protected]
```

MDP criteria: here "total reward"

The documentation for this class was generated from the following files:

- totalRewardMDP.h
- totalRewardMDP.cpp

Chapter 5

File Documentation

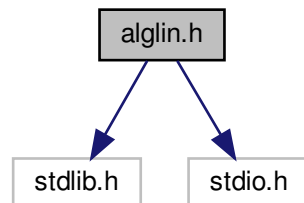
5.1 alglin.h File Reference

library of linear algebra functions (root programming)

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

Include dependency graph for alglin.h:



Functions

- void `verifVd` (double *Vecteur, int dim)
function to print a vector of double
- void `verifVi` (int *Vecteur, int dim)
function to print a vector of integer
- void `verifMat` (double **M, int dim)
function to print a matrix of double
- void `verifMat2D` (double **M, int dim1, int dim2)
function to print a no square matrix
- void `verifMatP` (double **M, int dim)
function to print a matrix of double (each double is printed with a large number of digits)
- double * `produitMatVect` (double **A, double **B, int dim1)

- function to compute the product of a matrix and a vector*
- double ** [produitMatMat](#) (double **A, double **B, int dim1)
- function to compute the product of two square matrices*
- double [prodScalV](#) (double *U, int Size)
- function to sum all the term of a vector.*
- double [Norm](#) (double *U, double *V, int Size)
- function to calculate the norm 1 of the difference of two vectors of doubles.*
- double [Span](#) (double *U, double *V, int Size)
- function to calculate the span of the difference of two vectors of doubles.*
- double [SpanRecup](#) (double *U, double *V, int Size, double *max, double *min)
- function to calculate the span of the difference of two vectors of doubles and to get the .*
- void [Transpose](#) (double **Matrice, int dimension)
- function to transpose a matrix*
- double * [ResolutionSysLin](#) (double **, double *, int)
- void [Inversion](#) (double **, double **, int)

5.1.1 Detailed Description

library of linear algebra functions (root programming)

header of linear algebra functions

Author

Hyon

Version

1

Date

jan 2018

5.1.2 Function Documentation

5.1.2.1 Inversion()

```
void Inversion (
    double ** ,
    double ** ,
    int )
```

Nettoyage des malloc

5.1.2.2 Norm()

```
double Norm (
    double * U,
    double * V,
    int Size )
```

function to calculate the norm 1 of the difference of two vectors of doubles.

Author

Hyon

Parameters

<i>U</i>	double* vector1
<i>V</i>	double* vector2
<i>Size</i>	int size of the vectors

Returns

the difference

5.1.2.3 prodScalV()

```
double prodScalV (  
    double * U,  
    int Size )
```

function to sum all the term of a vector.

Author

Hyon

Parameters

<i>U</i>	double* vector
<i>Size</i>	int size of vector

Returns

the difference

5.1.2.4 produitMatMat()

```
double** produitMatMat (  
    double ** A,  
    double ** B,  
    int dim1 )
```

function to compute the product of two square matrices

Author

Hyon

Parameters

<i>A</i>	double ** the first matrix
<i>B</i>	double ** the second matrix
<i>dim1</i>	the dimension of the matrices

Returns

a pointer to the created matrice that stores the product

5.1.2.5 produitMatVect()

```
double* produitMatVect (
    double ** A,
    double ** B,
    int dim1 )
```

function to compute the product of a matrix and a vector

Author

Hyon

Parameters

<i>A</i>	double ** the matrix
<i>B</i>	double ** the vector
<i>dim1</i>	the dimension of the matrix

Returns

a pointer to the created matrice that stores the product

5.1.2.6 ResolutionSysLin()

```
double* ResolutionSysLin (
    double ** ,
    double * ,
    int )
```

Nettoyage des malloc

5.1.2.7 `Span()`

```
double Span (
    double * U,
    double * V,
    int Size )
```

function to calculate the span of the difference of two vectors of doubles.

Author

Hyon

Parameters

<i>U</i>	double* vector1
<i>V</i>	double* vector2
<i>Size</i>	of the vectors

Returns

the difference of the maximum gap minus the minimum gap

5.1.2.8 `SpanRecup()`

```
double SpanRecup (
    double * U,
    double * V,
    int Size,
    double * max,
    double * min )
```

function to calculate the span of the difference of two vectors of doubles and to get the .

Author

Hyon

Parameters

<i>U</i>	double* vector 1
<i>V</i>	double* vector 2
<i>Size</i>	of the vectors
<i>max</i>	pointer on the calue of the max
<i>min</i>	pointer on the value of the min

Returns

the difference of the maximum gap minus the minimum gap

5.1.2.9 Transpose()

```
void Transpose (
    double ** Matrice,
    int dimension )
```

function to transpose a matrix

Author

Hyon

Parameters

<i>Matrice</i>	Matrix to be transposed
<i>dimension</i>	int

Returns

none

5.1.2.10 verifMat()

```
void verifMat (
    double ** M,
    int dim )
```

function to print a matrix of double

Author

Hyon

Parameters

<i>M</i>	double ** the matrix
<i>dim</i>	the dimension of the matrix

5.1.2.11 verifMat2D()

```
void verifMat2D (
    double ** M,
    int dim1,
    int dim2 )
```

function to print a no square matrix

Author

Hyon

Parameters

<i>M</i>	double ** the matrix
<i>dim1</i>	number of rows
<i>dim2</i>	number of columns

5.1.2.12 verifMatP()

```
void verifMatP (
    double ** M,
    int dim )
```

function to print a matrix of double (each double is printed with a large number of digits)

Author

Hyon

Parameters

<i>M</i>	double ** the matrix
<i>dim</i>	the dimension of the matrix

5.1.2.13 verifVd()

```
void verifVd (
    double * Vecteur,
    int dim )
```

function to print a vector of double

Author

Hyon

Parameters

<i>Vecteur</i>	double * the vector
<i>dim</i>	the dimension of the vector

5.1.2.14 verifVi()

```
void verifVi (
    int * Vecteur,
    int dim )
```

function to print a vector of integer

Author

Hyon

Parameters

<i>Vecteur</i>	int * the vector
<i>dim</i>	the dimension of the vector

Index

- ~averageMDP
 - averageMDP, [11](#)
- ~discountedMDP
 - discountedMDP, [27](#)
- ~feedbackSolutionMDP
 - feedbackSolutionMDP, [35](#)
- ~finiteHorizonDiscountedMDP
 - finiteHorizonDiscountedMDP, [43](#)
- ~finiteHorizonMDP
 - finiteHorizonMDP, [53](#)
- ~nonStationarySolutionMDP
 - nonStationarySolutionMDP, [75](#)
- ~solutionMDP
 - solutionMDP, [85](#)
- ~structuralPropertiesPol
 - structuralPropertiesPol, [90](#)
- ~structuralPropertiesVF
 - structuralPropertiesVF, [97](#)
- ~totalRewardMDP
 - totalRewardMDP, [105](#)
- accumulateRewards
 - continuousDiscountedMDP, [22](#)
- action
 - feedbackSolutionMDP, [39](#)
 - nonStationarySolutionMDP, [83](#)
- actionSpace
 - genericMDP, [71](#)
- addMatrix
 - genericMDP, [65](#)
- algin.h, [111](#)
 - Inversion, [112](#)
 - Norm, [112](#)
 - prodScalV, [113](#)
 - produitMatMat, [113](#)
 - produitMatVect, [114](#)
 - ResolutionSysLin, [114](#)
 - Span, [114](#)
 - SpanRecup, [115](#)
 - Transpose, [116](#)
 - verifMat, [116](#)
 - verifMat2D, [116](#)
 - verifMatP, [117](#)
 - verifVd, [117](#)
 - verifVi, [118](#)
- averageMDP, [7](#)
 - ~averageMDP, [11](#)
 - averageMDP, [9](#), [10](#)
 - changeIndex, [11](#)
 - getChain, [12](#)
 - indexEtatSpecifique, [19](#)
 - policyCost, [13](#)
 - policyIteration, [13](#)
 - policyIterationModified, [14](#)
 - policyIterationModifiedAA, [15](#)
 - relativeValueIteration, [16](#)
 - rho, [19](#)
 - type_c, [19](#)
 - valueIteration, [16](#)
 - valueIterationGS, [17](#)
 - valueIterationMC, [18](#)
- beta
 - continuousDiscountedMDP, [22](#)
 - discountedMDP, [32](#)
- beta_
 - finiteHorizonDiscountedMDP, [49](#)
- changeIndex
 - averageMDP, [11](#)
- clearRew
 - genericMDP, [65](#)
- continuousDiscountedMDP, [19](#)
 - accumulateRewards, [22](#)
 - beta, [22](#)
 - discountedMDP, [21](#)
 - type_c, [22](#)
- cost_perStage
 - genericMDP, [65](#)
- deleteMatrix
 - genericMDP, [66](#)
- discountedMDP, [22](#)
 - ~discountedMDP, [27](#)
 - beta, [32](#)
 - continuousDiscountedMDP, [21](#)
 - discountedMDP, [24–26](#)
 - getChain, [27](#)
 - policyCost, [27](#)
 - policyCostbyIndex, [28](#)
 - policyIteration, [29](#)
 - policyIterationModified, [29](#)
 - policyIterationModifiedGS, [30](#)
 - type_c, [33](#)
 - valueIteration, [31](#)
 - valueIterationGS, [32](#)
- feedbackSolutionMDP, [33](#)
 - ~feedbackSolutionMDP, [35](#)
 - action, [39](#)

- feedbackSolutionMDP, 35
- getAction, 35
- getActionIndex, 35
- getValue, 36
- getValueIndex, 36
- setAction, 37
- setActionIndex, 37
- setValue, 38
- value, 39
- writeSolution, 38
- writeSolutionByDim, 38
- finiteHorizonDiscountedMDP, 40
 - ~finiteHorizonDiscountedMDP, 43
 - beta_, 49
 - finiteHorizonDiscountedMDP, 41, 42
 - getChain, 44
 - horizon_, 49
 - policyCost, 44
 - policyIteration, 45
 - policyIterationModified, 46
 - type_c, 49
 - valueIteration, 47
 - valueIterationGS, 48
 - writeMDP, 49
- finiteHorizonMDP, 50
 - ~finiteHorizonMDP, 53
 - finiteHorizonMDP, 52
 - getChain, 54
 - horizon_, 59
 - policyCost, 54
 - policyIteration, 55
 - policyIterationModified, 56
 - type_c, 59
 - valueIteration, 57
 - valueIterationGS, 58
 - writeMDP, 59
- genericMDP, 60
 - actionSpace, 71
 - addMatrix, 65
 - clearRew, 65
 - cost_perStage, 65
 - deleteMatrix, 66
 - genericMDP, 62–64
 - getChain, 67
 - opMin, 71
 - policyCost, 67
 - policyIteration, 68
 - policyIterationModified, 69
 - rewards, 71
 - stateSpace, 71
 - transitions, 71
 - type_r, 71
 - type_t, 72
 - valueIteration, 69
 - valueIterationGS, 70
- getAction
 - feedbackSolutionMDP, 35
 - nonStationarySolutionMDP, 75
- getActionAtStep
 - nonStationarySolutionMDP, 75
- getActionAtStepIndex
 - nonStationarySolutionMDP, 76
- getActionIndex
 - feedbackSolutionMDP, 35
- getChain
 - averageMDP, 12
 - discountedMDP, 27
 - finiteHorizonDiscountedMDP, 44
 - finiteHorizonMDP, 54
 - genericMDP, 67
 - totalRewardMDP, 105
- getHorizon
 - nonStationarySolutionMDP, 77
- getValue
 - feedbackSolutionMDP, 36
 - nonStationarySolutionMDP, 77
- getValueAtStep
 - nonStationarySolutionMDP, 77
- getValueAtStepIndex
 - nonStationarySolutionMDP, 78
- getValueIndex
 - feedbackSolutionMDP, 36
- horizon_
 - finiteHorizonDiscountedMDP, 49
 - finiteHorizonMDP, 59
 - nonStationarySolutionMDP, 83
- indexEtatSpecifique
 - averageMDP, 19
- Inversion
 - algin.h, 112
- monotonicityCX
 - structuralPropertiesVF, 97, 98
- monotonicityPol
 - structuralPropertiesPol, 90
- monotonicityPolByDim
 - structuralPropertiesPol, 91
- monotonicityVFByDim
 - structuralPropertiesVF, 99, 100
- monotonicityVF
 - structuralPropertiesVF, 98, 99
- nonStationarySolutionMDP, 72
 - ~nonStationarySolutionMDP, 75
 - action, 83
 - getAction, 75
 - getActionAtStep, 75
 - getActionAtStepIndex, 76
 - getHorizon, 77
 - getValue, 77
 - getValueAtStep, 77
 - getValueAtStepIndex, 78
 - horizon_, 83
 - nonStationarySolutionMDP, 74
 - setAction, 79

- setActionAtStep, 79
 - setActionAtStepIndex, 80
 - setValue, 80
 - setValueAtStep, 81
 - setValueAtStepIndex, 81
 - value, 83
 - writeSolution, 82
 - writeSolutionByDim, 82
- Norm
 - alglin.h, 112
- opMin
 - genericMDP, 71
- policyCost
 - averageMDP, 13
 - discountedMDP, 27
 - finiteHorizonDiscountedMDP, 44
 - finiteHorizonMDP, 54
 - genericMDP, 67
 - totalRewardMDP, 106
- policyCostbyIndex
 - discountedMDP, 28
- policyIteration
 - averageMDP, 13
 - discountedMDP, 29
 - finiteHorizonDiscountedMDP, 45
 - finiteHorizonMDP, 55
 - genericMDP, 68
 - totalRewardMDP, 107
- policyIterationModified
 - averageMDP, 14
 - discountedMDP, 29
 - finiteHorizonDiscountedMDP, 46
 - finiteHorizonMDP, 56
 - genericMDP, 69
 - totalRewardMDP, 107
- policyIterationModifiedAA
 - averageMDP, 15
- policyIterationModifiedGS
 - discountedMDP, 30
- prodScalV
 - alglin.h, 113
- produitMatMat
 - alglin.h, 113
- produitMatVect
 - alglin.h, 114
- relativeValueIteration
 - averageMDP, 16
- ResolutionSysLin
 - alglin.h, 114
- rewards
 - genericMDP, 71
- rho
 - averageMDP, 19
- sSPol
 - structuralPropertiesPol, 92
- sSPolbyDim
 - structuralPropertiesPol, 93
- setAction
 - feedbackSolutionMDP, 37
 - nonStationarySolutionMDP, 79
- setActionAtStep
 - nonStationarySolutionMDP, 79
- setActionAtStepIndex
 - nonStationarySolutionMDP, 80
- setActionIndex
 - feedbackSolutionMDP, 37
- setSize
 - solutionMDP, 86
- setValue
 - feedbackSolutionMDP, 38
 - nonStationarySolutionMDP, 80
- setValueAtStep
 - nonStationarySolutionMDP, 81
- setValueAtStepIndex
 - nonStationarySolutionMDP, 81
- size_
 - solutionMDP, 88
- solutionMDP, 84
 - ~solutionMDP, 85
 - setSize, 86
 - size_, 88
 - solutionMDP, 85
 - writeSolution, 86
 - writeSolutionByDim, 87
- space_
 - structuralPropertiesPol, 95
 - structuralPropertiesVF, 101
- Span
 - alglin.h, 114
- SpanRecup
 - alglin.h, 115
- stateSpace
 - genericMDP, 71
- structuralPropertiesPol, 88
 - ~structuralPropertiesPol, 90
 - monotonicityPol, 90
 - monotonicityPolByDim, 91
 - sSPol, 92
 - sSPolbyDim, 93
 - space_, 95
 - structuralPropertiesPol, 89
 - thresholdPol, 94
- structuralPropertiesVF, 95
 - ~structuralPropertiesVF, 97
 - monotonicityCX, 97, 98
 - monotonicityVFByDim, 99, 100
 - monotonicityVF, 98, 99
 - space_, 101
 - structuralPropertiesVF, 96
- thresholdPol
 - structuralPropertiesPol, 94
- totalRewardMDP, 101
 - ~totalRewardMDP, 105

- getChain, [105](#)
 - policyCost, [106](#)
 - policyIteration, [107](#)
 - policyIterationModified, [107](#)
 - totalRewardMDP, [103](#), [104](#)
 - type_c, [110](#)
 - valueIteration, [108](#)
 - valueIterationGS, [109](#)
- transitions
 - genericMDP, [71](#)
- Transpose
 - algin.h, [116](#)
- type_c
 - averageMDP, [19](#)
 - continuousDiscountedMDP, [22](#)
 - discountedMDP, [33](#)
 - finiteHorizonDiscountedMDP, [49](#)
 - finiteHorizonMDP, [59](#)
 - totalRewardMDP, [110](#)
- type_r
 - genericMDP, [71](#)
- type_t
 - genericMDP, [72](#)
- value
 - feedbackSolutionMDP, [39](#)
 - nonStationarySolutionMDP, [83](#)
- valueIteration
 - averageMDP, [16](#)
 - discountedMDP, [31](#)
 - finiteHorizonDiscountedMDP, [47](#)
 - finiteHorizonMDP, [57](#)
 - genericMDP, [69](#)
 - totalRewardMDP, [108](#)
- valueIterationGS
 - averageMDP, [17](#)
 - discountedMDP, [32](#)
 - finiteHorizonDiscountedMDP, [48](#)
 - finiteHorizonMDP, [58](#)
 - genericMDP, [70](#)
 - totalRewardMDP, [109](#)
- valueIterationMC
 - averageMDP, [18](#)
- verifMat
 - algin.h, [116](#)
- verifMat2D
 - algin.h, [116](#)
- verifMatP
 - algin.h, [117](#)
- verifVd
 - algin.h, [117](#)
- verifVi
 - algin.h, [118](#)
- writeMDP
 - finiteHorizonDiscountedMDP, [49](#)
 - finiteHorizonMDP, [59](#)
- writeSolution
 - feedbackSolutionMDP, [38](#)
 - nonStationarySolutionMDP, [82](#)
 - solutionMDP, [86](#)
- writeSolutionByDim
 - feedbackSolutionMDP, [38](#)
 - nonStationarySolutionMDP, [82](#)
 - solutionMDP, [87](#)