Quick Start: Resolving a Markov decision process problem using the MDPtoolbox in Matlab

Iadine Chadès[*], Guillaume Chapron[†], Marie-Josée Cros[‡], Frédérick Garcia[‡], Régis Sabbadin[‡]

January 2014

1 MDP framework

(From Wikipedia, the free encyclopedia with minor changes)

Markov decision processes (MDP) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning.

More precisely, a Markov Decision Process is a discrete time stochastic control process. At each time step, the process is in some state s, and the decision maker may choose any action a that is available in state s. The process responds at the next time step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R(s, s', a)$.

The probability that the process moves into its new state s' is influenced by the chosen action. Specifically, it is given by the state transition function $P(s, s', a)$. Thus, the next state s' depends on the current state s and the decision maker's action a. But given s and a, it is conditionally independent of all previous states and actions; in other words, the state transitions of an MDP possess the Markov property.

Definition
In its typical definition, a Markov decision process is a 4-tuple $< S, A, P, R >$,

[*]CSIRO Ecosystem Sciences, GPO Box 2583, Brisbane QLD 4001, Australia
[†]Grimsö Wildlife Research Station, Swedish University of Agricultural Sciences, 73091 Riddarhyttan, Sweden
[‡]INRA, UR 875 Applied Mathematics and Computer Science laboratory, F-31326 Castanet Tolosan, France.

where:

- S is a finite set of states,
- A is a finite set of actions,
- $P(s, s', a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time t+1,
- $R(s, s', a)$ or $R(s', a)$ is the immediate reward (or expected) received after transition to state s' from state s with action a.

The core problem of MDPs is to find a "policy" for the decision maker: a function π that specifies the action $\pi(s)$ that the decision maker will choose when in state s. The goal is to choose a policy π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

$\sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1}, \pi(s_t))$

where γ is the discount factor and satisfies $0 \leq \ \gamma \ < 1$.

Algorithms
MDPs can be solved by linear programming or dynamic programming. In what follows we present the latter approach. The standard family of algorithms to calculate this optimal policy requires storage for two arrays indexed by state: value V, which contains real values, and policy π which contains actions. At the end of the algorithm, π will contain the solution and V(s) will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s.

The algorithm has the following two kinds of steps, which are repeated in some order for all the states until no further changes take place. They are defined recursively as follows:

$\pi(s) := \arg\max_a \{\sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V(s')\right)\}$

$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s')\right)$

Their order depends on the variant of the algorithm; one can also do them for all states at once or state by state, and more often to some states than

others. As long as no state is permanently excluded from either of the steps, the algorithm will eventually arrive at the correct solution.
Notable variants: Value iteration, Policy iteration, Modified policy iteration.

2 MDPtoolbox

The MDPtoolbox (http://www7.inra.fr/mia/T/MDPtoolbox) proposes functions related to the resolution of discrete-time Markov Decision Processes: value iteration, policy iteration, linear programming algorithms with some variants.
It is currently available on several environment: MATLAB, GNU Octave, Scilab and R.

Download Matlab version
To download the toolbox for Matlab, follow the toolbox instructions given on the toolbox website: http://www7.inra.fr/mia/T/MDPtoolbox/Install.html

To use the toolbox, just call Matlab and add the MDPtoolbox directory to search path.
For example, go to the MDPtoolbox directory, call Matlab and execute:

```
>> MDPtoolbox_path = pwd;
>> addpath(MDPtoolbox_path)
```

In this Matlab session, it is then possible to use all the MDPtoolbox functions. To acces the HTML documentation, open with a browser the local file: MDPtoolbox/documentation/DOCUMENTATION.html .

3 Description of a tiny forest management problem

The considered problem is to manage a forest stand with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. The forest stand is managed by two possible actions: Wait or Cut. An action is decided and applied each time period of 20 years at the beginning of the period.
Three states are defined, corresponding to 3 age-class of trees: age-class 0-20 years (state 1), 21-40 years (state 2), more than 40 years (state 3). The state 3 correspond to the oldest age-class. At the end of a period t, if the state was s at t and action Wait is choosen, the state at the next time period will be $min(s + 1, 3)$ if no fire occured. But there is a probability p that a fire burns the forest after the application of the action, living the stand at

the youngest age-class (state 1). Let p = 0.1 be the probability of wildfire occurence during a time period.
The problem is howto manage this stand in a long term vision to maximize the γ-discounted reward with $\gamma = 0.95$.

Here is a modelisation of this problem in the MDP framework.
Let Wait be action 1 and Cut action 2.
The transition matrix P(s,s',a) of the problem can then be defined as follows:

$$\text{P}(.,.,1) = \begin{bmatrix} p & 1-p & 0 \\ p & 0 & 1-p \\ p & 0 & 1-p \end{bmatrix}$$

$$\text{P}(.,.,2) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The reward matrix R(s',a) is defined as follows.

$$\text{R}(.,1) = \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix}$$

$$\text{R}(.,2) = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

For example, the probability to be in state 3 at time t+1 (s' = 3), being in state 3 (s = 3) and choosing action Wait (a = 1) at time t, is P(3,3,1) = 1 - p = 0.9 and the associated reward is R(3,1) = 4.

Let define the MDP problem in Matlab.
First, define the matrices P for the transition function and R for the reward function.

```
>> P(:,:,1) = [0.1 0.9 0; ...
              0.1 0 0.9; ...
              0.1 0 0.9]; ...
>> P(:,:,2) = [1 0 0; 1 0 0; 1 0 0];
>> R(:,1) = [0 0 4]';
>> R(:,2) = [0 1 2]';
```

or use the forest example function of the MDP toolbox:

```
>> [P, R] = mdp_example_forest();
```

It is important to check the validity of the description.
For this, use the mdp_check function.

```
>> mdp_check(P, R)
   ans =
     ''
```

When the output is empty, no error was detected.

Finally, define the discount.

```
>> discount = 0.95;
```

4 Resolution of this tiny problem

The problem is now expressed, lets solve it.
For a discounted criterium, several algorithms and their related functions are available in the toolbox (see Functions by category page documentation). For the very simple considered problem, their are quite equivallent. Let call the notable algorithms.

```
>> [V, policy] = mdp_policy_iteration(P, R, discount)
V =
   58.4820
   61.9020
   65.9020
policy =
     1
     1
     1

>>[policy] = mdp_value_iteration(P, R, discount)
policy =
     1
     1
     1

>>[V, policy] = mdp_LP(P, R, discount)
Optimization terminated.
V =
   58.4820
   61.9020
   65.9020
policy =
     1
     1
     1
>> [~, V, policy] = mdp_Q_learning(P, R, discount)
V =
   56.3507
   59.8762
   63.8072
policy =
     1
     1
     1
```

The optimal policy found is to choose action Wait(1) in the 3 defined states, that is in a more understandable way 'never cut'.
Note that mdp_LP function provides the exact expected value function. For

instance, V(2) = 61.9020 is the expected reward value when in state 2 (age-class 21-40 years for trees). For the other algorithms, to better apprehend V, some functions are available in the toolbox. Let call them. Note that mdp_eval_policy_matrix provides also the exact expected value function.

```
>> Vpolicy = mdp_eval_policy_matrix (P, R, discount, policy)
Vpolicy =
   58.4820
   61.9020
   65.9020
>> Vpolicy = mdp_eval_policy_iterative(P, R, discount, policy)
Vpolicy =
   58.4819
   61.9019
   65.9019
>> Vpolicy = mdp_eval_policy_TD_0 (P, R, discount, policy)
Vpolicy =
   54.2553
   58.1873
   62.7556
```

It is often necessary to express and visualise policies in more understandable expressions: decision rules, graphs ... An example of graph, pertinent for this problem, is the representation of the percentage spend in each state (age-class), that is the stationary distribution of state.

In order to compute it, define the get_stationary_distribution as follow and save it in a get_stationary_distribution.m file.

```
function mu = get_stationary_distribution( p )
% Computes the stationary distribution mu of a Markov chain
% described by p (stochastic matrix, ie sum(p,2)=1).
% Input
% p : transition matrix associated with a policy, p(s,s')
% Output
% mu : stationary distribution for each state s ( p*mu'=mu' )
% is_OK_mu : false if p is not a stochatic matrix, else true

s=size(p,1);
mu=zeros(1,s);
if any(abs(sum(p,2)-1)>10^-4) || (size(p,2)~=s)
    disp 'ERROR in get_stationary_distribution: argument p must be a stochastic matrix'
else
    % mu satisfies p*mu'=mu' and mu sums to one
    A=transpose(p)-eye(s);
    A(s,:)=ones(1,s);
    b=zeros(s,1);
    b(s)=1;
    mu=transpose(A\b);
    is_OK_mu = ~isempty(mu) && all(mu>-10^-4) && (sum(mu)-1<10^-4);
    if ~is_OK_mu; mu=[]; end
end
```

Then call this function and plot (Figure 1) the stationary distribution in age-classes.

```
>> mu = get_stationary_distribution( mdp_computePpolicyPRpolicy(P, R, policy) )
mu =
    0.1000 0.0900 0.8100
>> bar(mu,0.4); ylim([0 1]);
>> xlabel('age-class'); ylabel('percentage of time in age-class');
```

Beware that in the Matlab code the caracter ' must be replace by a '.

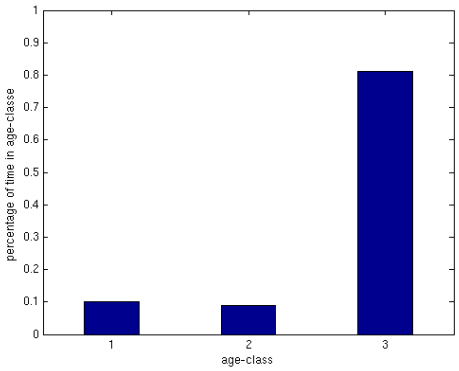

Figure 1: Percentage of time spend in age-classes for the mdp_example_forest() problem, with the optimal policy: never cut

5 Exploration of variant problems

Furthermore, it is also possible to evaluate the impact of model parameters change.

First, what about a lower incitation of conserving the oldest age-class (0.4 instead of 4).

```
>> [P, R] = mdp_example_forest (3, 0.4, 2,.1);
>> [V, policy] = mdp_LP(P, R, discount)
Optimization terminated.
V =
   11.3073
   11.9686
   12.7419
policy =
     1
     1
     2
```

The policy found ask now to cut the oldest-age class.
Lets compute the state stationary distribution and plot it (Figure 2).

```
>> mu = get_stationary_distribution( mdp_computePpolicyPRpolicy(P, R, policy) )
mu =
    0.3690 0.3321 0.2989
>> bar(mu,0.4); ylim([0 1]);
>> xlabel('age-class'); ylabel('percentage of time in age-class');
```

Beware that in the Matlab code the caracter ' must be replace by a '.

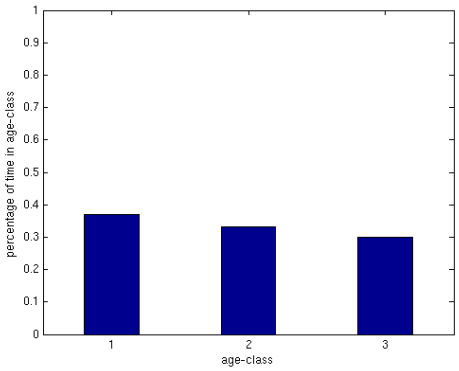

Figure 2: Percentage of time spend in age-classes for the mdp_example_forest(3, 0.4, 2,.1) problem, with the optimal policy: cut the oldest-age class

Second, what about if the probability of wildfire is very high (p = 0.8).

```
>>[P, R] = mdp_example_forest (3, 4, 2,.8);
>>[V, policy] = mdp_LP(P, R, discount)
Optimization terminated.
V =
    3.1933
    4.0336
    7.9344
policy =
     1
     2
     1
```

As expected, the policy found changes and requires to cut at the 2nd age-class with lower expected values.
Lets compute the state stationary distribution and plot it (Figure 3).

```
>> mu = get_stationary_distribution( mdp_computePpolicyPRpolicy(P, R, policy) )
mu =
```

```
    0.8333 0.1667 0
>> bar(mu,0.4); ylim([0 1]);
>> xlabel('age-class'); ylabel('percentage of time in age-class');
```

Beware that in the Matlab code the caracter ' must be replace by a '.

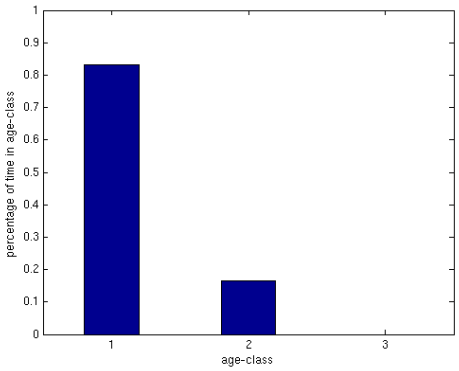

Figure 3: Percentage of time spend in age-classes for the mdp_example_forest(3, 4, 2,.8) problem, with the optimal policy: cut the second age-class