INSTRUCTIONS FOR USING THE MONOTONE–ADP MATLAB CODE

DANIEL R. JIANG* AND WARREN B. POWELL*

The provided MATLAB files implement the Monotone–ADP algorithm, adapted for the post–decision case. The two files, `madp.m` and `optimize.m`, provide the majority of the functionality. To adapt the code to a specific problem, one needs to edit the `parameters.m` file and then run `madp_script.m` to begin computation. The `parameters.m` file requires the definition of several variables and functions, as described below. In the example provided, we have chosen a simple enough problem such that all of the contribution/transition functions can be written as anonymous functions; however, in a more complex setting, it would be useful to write the functions as separate `.m` files. These functions can then be referenced in `parameters.m` using the typical anonymous function syntax. For example, one can put the contribution function definition in a file `contributionfunction.m` and then edit the line in `parameters.m` to be

```
params.contribution = @(s,a,t)(contributionfunction(s,a,t));
```

The same can be done for `params.prepost_trans` and `params.postpre_trans` (see (3) below for a discussion regarding these transition functions).

Most of the parameter definitions needed in the file `parameter.m` are self–explanatory and the brief comments in the code illustrate proper usage. Nevertheless, we make the following clarifying remarks:

(1) There are four parameters that define the post–decision (`params.n`, `params.pdstate_min`, `params.pdstate_max`, `params.pdstate_size`) and three parameters that define the pre–decision state space (`params.state_min`, `params.state_max`, `params.state_size`). Each of these parameters are vectors where each component describes a specific dimension of the respective state space. The two state spaces do not need to be of the same dimension, as long the transition functions are defined accordingly. We also do not need to explicitly use the dimension of the pre–decision state space.

(2) The finite action space $\mathcal{A}$ can be enumerated by the one–dimensional index–set, denoted: $\mathcal{I} = \{1, 2, \ldots, \texttt{params.action\_size}\}$. The function `params.action_val` maps from $\mathcal{I} \to \mathcal{A}$. This allows the contribution function and pre–decision to post–decision transition function to be specified as a function of $\mathcal{A}$ rather than the more inconvenient $\mathcal{I}$.

(3) The transition function is typically written, roughly, as $S_{t+1} = f(S_t, a_t, W_{t+1})$, where S_t is the state at time t, a_t is the action taken at time t, and W_{t+1} is the exogenous random information. In this particular version of Monotone–ADP, we take advantage of the post–decision state formulation and break up the transition functions into two parts: $S_t^a = g(S_t, a_t)$ and $S_{t+1} = h(S_t^a, W_{t+1})$, where S_t^a is the post–decision state variable. In the code, `params.prepost_trans` defines g and `params.postpre_trans` defines h.

(4) In order to easily facilitate implementation of the policy, we have separated the Bellman operator part of the algorithm into the file `optimize.m`. The user simply needs to call `optimize.m` with the appropriate state and time period to implement the policy generated by the value function. The function outputs the optimal action index as its third output parameter.

Please email `drjiang@princeton.edu` with any comments, suggestions, or questions.

*Department of Operations Research and Financial Engineering, Princeton University