MATH4406 – Assignment 4

Patrick Laub (ID: 42051392)

September 26, 2014

1 Real-world motivation

A simple example of this type of control system is the thermostat for an air-conditioning unit. Once set to a desired temperature, the thermostat must enable and disable the air-conditioning over time to try to meet the set-point.

Consider the case where the thermostat is keeping an expensive server room at the correct temperature; if the temperature becomes too hot or too cold then the equipment could suffer costly damages, or operate inefficiently (hence the large negative rewards in state ± 5). Perhaps the other $r(\cdot,\cdot)$ effects can be explained by additional costs accrued when the unit is swapped between heating and cooling mode.

When the air-conditioning is turned on ($a = 1$) then the system is likely to cool down. Denote the random states $X_i \in \{-5, \dots, 5\}$, corresponding to system temperatures, then the previous statement means

$$\mathbb{P}(X_{n+1} > X_n | a = 1) \geq \mathbb{P}(X_{n+1} < X_n | a = 1)$$

and conversely the temperature is likely to drop when the air-condition is turned off ($a = -1$):

$$\mathbb{P}(X_{n+1} < X_n | a = -1) \geq \mathbb{P}(X_{n+1} > X_n | a = -1)\,.$$

2 Expected behaviour of optimal policies

Optimal policies will likely be symmetric about the state 0. For symmetric policies then the selected action in 0 will have no effect on overall system performance (as $r(0, \cdot) = 0$).

When $\lambda \downarrow 0$ then this means that future states become increasingly unimportant, and so the optimal policy should try to maximised the one-step expected payoff. If only looking at one-step ahead then a maximising policy should be (ignoring states $\{-5, 0, 5\}$)

$$d(s) = \begin{cases} -1, & s < 0 \\ 1, & s > 0 \end{cases}$$

However when $\lambda \uparrow 1$ then the policy should be very conservative; the effects of the distant future are extremely important so the system should do everything it can to stick around $s = 0$ (every $X_t \in \{-5, 5\}$ will have a huge negative cost which should be avoided). A hypothesed optimal policy here is (ignoring states $\{-5, 0, 5\}$)

$$d(s) = \begin{cases} 1, & s < 0 \\ -1, & s > 0 \end{cases}$$

3 Optimal Policies: Discussion and Results

Brute-force enumeration, value iteration and policy iteration were used to solve the MDPs with $\lambda \in \{0.01, 0.02, \dots, 0.99\}$. Value iteration used a tolerance value of $\epsilon = 10^{-9}$. Each of the algorithms gave the same optimal policies (as expected). Fig. 1 visually[1] shows the optimal decision to take at every state s for every discounting factor λ. The policies with $\lambda \approx 0$ and $\lambda \approx 1$ match the expected/intuitive optimal policies outlined in Section 2.

[1] If a table is required, then the code in Section 4.4 constructs a table of the results in the matrix *dRules* which can easily be printed.

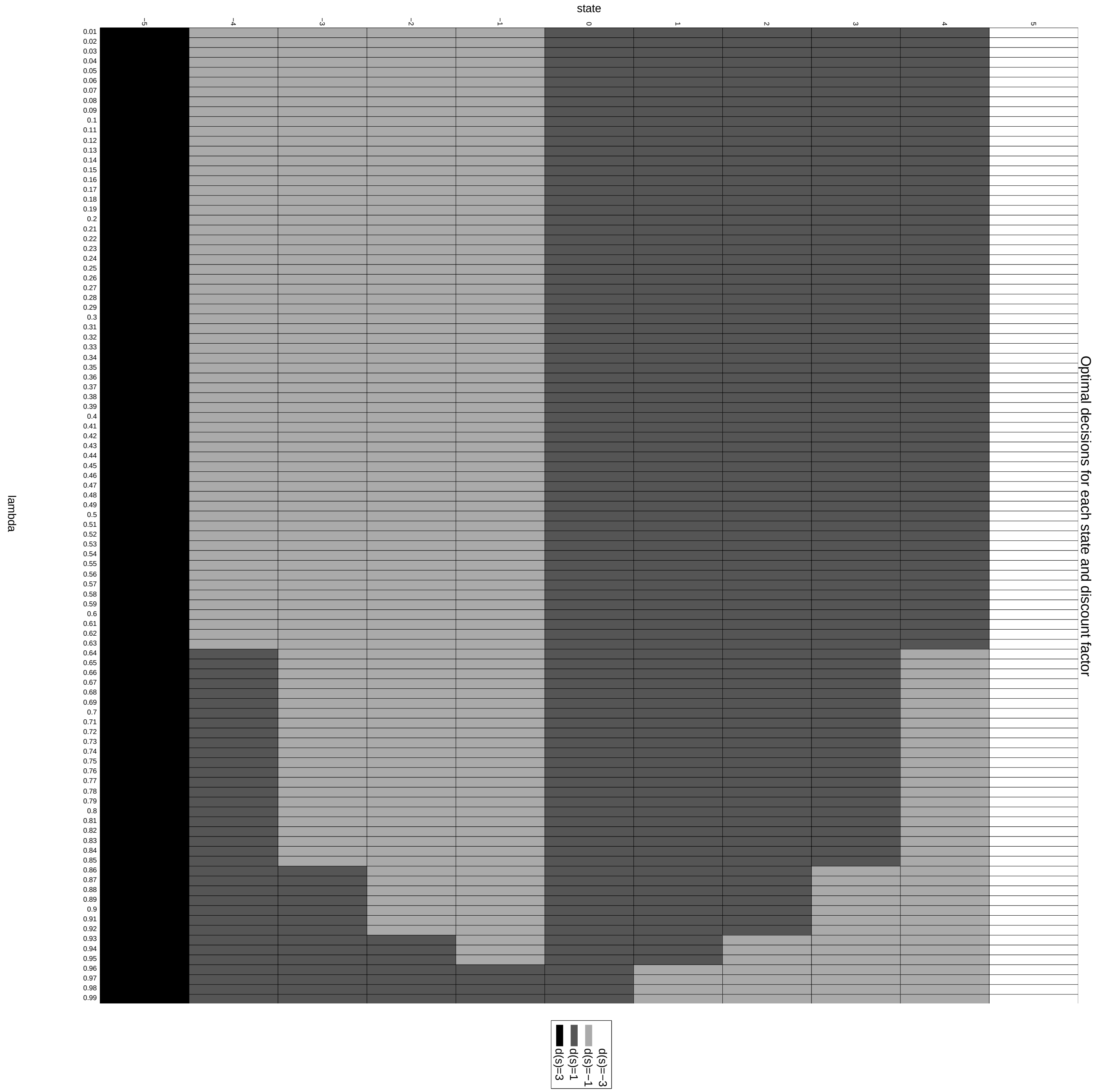

Figure 1: Optimal policies

4 Appendix: MATLAB Implementation

4.1 Brute force enumeration

```
function [v, d] = brute_force(lambda)
    % Problem data.
    S = (-5:5)'; numS = numel(S);

    % Generate all decision rules.
    allDecRules = allcomb(3, [-1,1], [-1,1], [-1,1], [-1,1], 1, ...
        [-1,1], [-1,1], [-1,1], [-1,1], -3);

    % Store the maximising value and decision rule so far.
    bestV = -Inf .* ones(numS, 1);
    bestD = NaN;

    % Perform policy evaluation for every decision rule.
    for ruleNum = 1:size(allDecRules, 1)
        d = allDecRules(ruleNum, :)';

        % Construct reward and transition vectors.
        rd = S .* d;
        Pd = zeros(numS);
        Pd(1,1) = 0.5; Pd(1,2) = 0.5;
        Pd(end,end) = 0.5; Pd(end,end-1) = 0.5;
        for i = 2:(numS-1)
            Pd(i,i-1) = 0.75 * (d(i) == -1) + 0.25 * (d(i) == 1);
            Pd(i,i+1) = 0.25 * (d(i) == -1) + 0.75 * (d(i) == 1);
        end

        % Compute the value of the MDP.
        v = (eye(numS) - lambda .* Pd) \ rd;

        % Store if better (or as good as) the previous best.
        if all((v - bestV) >= 0)
            bestV = v;
            bestD = d;
        end
    end

    v = bestV; d = bestD;
end
```

4.2 Value iteration

```matlab
function [v, d] = value_iteration(lambda)
    % Problem data.
    S = (-5:5)'; numS = numel(S);
    eps = 1e-9;

    % Start with any guess at the value.
    v = zeros(numS, 1);
    nextV = zeros(numS, 1);

    % Construct decision rule at each step.
    d = zeros(numS, 1); d(1) = 3; d(end) = -3; d(S == 0) = 1;

    % Iterate over the value space until convergence (or give up).
    for iters = 1:1e5
        % For each state compute V^(n+1)(s) from V^n.
        for i = 1:numS
            % Handle edge cases separately to the middle states.
            if i == 1
                nextV(1) = S(1)*3 + lambda*(0.5*v(1) + 0.5*v(2));
            elseif i == numS
                nextV(end) = S(end)*-3 + lambda*(0.5*v(end) + 0.5*v(end-1));
            else
                up = S(i)*1 + lambda*(0.25*v(i-1) + 0.75*v(i+1));
                down = S(i)*-1 + lambda*(0.75*v(i-1) + 0.25*v(i+1));

                % Take the action which maximises this quantity. Note that
                % when many maximisers exist, then this will select the
                % larger one every time.
                if up >= down
                    nextV(i) = up; d(i) = 1;
                else
                    nextV(i) = down; d(i) = -1;
                end
            end
        end

        % Check for convergence.
        if all(abs(nextV - v) < eps*(1-lambda)/(2*lambda))
            v = nextV;
            break;
        end

        v = nextV;
    end

    if iters == 1e5
        warning('Did not converge!');
    end
end
```

4.3 Policy iteration

```
function [v, d] = policy_iteration(lambda)
    % Problem data.
    S = (-5:5)'; numS = numel(S);

    % Start with any guess decision rule.
    d = ones(numS, 1); d(1) = 3; d(end) = -3;
    nextD = d;

    % Construct decision rule at each step.
    for iters = 1:1e5
        % Construct reward and transition vectors.
        rd = S .* d;
        Pd = zeros(numS);
        Pd(1,1) = 0.5; Pd(1,2) = 0.5;
        Pd(end,end) = 0.5; Pd(end,end-1) = 0.5;
        for i = 2:(numS-1)
            Pd(i,i-1) = 0.75 * (d(i) == -1) + 0.25 * (d(i) == 1);
            Pd(i,i+1) = 0.25 * (d(i) == -1) + 0.75 * (d(i) == 1);
        end

        % Do policy evalutation.
        v = (eye(numS) - lambda .* Pd) \ rd;

        % Do policy improvement.
        for i = 1:numS
            if any(S(i) == [-5, 0, 5]), continue; end;

            up = S(i)*1 + lambda*(0.25*v(i-1) + 0.75*v(i+1));
            down = S(i)*-1 + lambda*(0.75*v(i-1) + 0.25*v(i+1));
            if up >= down
                nextD(i) = 1;
            else
                nextD(i) = -1;
            end
        end

        % If the decision rule has converged then stop iterating.
        if all(nextD == d)
            break;
        end

        d = nextD;
    end

    if iters == 1e5
        warning('Did not converge!');
    end
end
```

4.4 Tests and plotting

```matlab

dRules = [];

for lambda=0.01:0.01:0.99

    [vb, db] = brute_force(lambda);
    [vv, dv] = value_iteration(lambda);
    [vp, dp] = policy_iteration(lambda);

    if any(db ~= dv) || any(dv ~= dp)
        error('Difference!!');
    end

    dRules = [dRules, db];
end

% Some grid code from: http://stackoverflow.com/questions/8711971
figure(1); clf;

% plot dummy objects
hold on;
for a=[-3, -1, 1, 3]
    plot(1, 1, 'LineWidth', 10, 'Color',  [(3-a)/6,(3-a)/6,(3-a)/6]);
end

% Draw large bounding box:
xstart = 0.01-0.01/2; ystart = -5-0.5;
xlen = 0.99; ylen = 11;

rectangle('position', [xstart, ystart, xlen, ylen])
dx = 0.01; dy = 1;
nx = floor(xlen/dx); ny = floor(ylen/dy);

for i = 1:nx
    x = xstart + (i-1)*dx;
    for j = 1:ny
        y = ystart + (j-1)*dy;

        a = dRules(j,i);
        rectangle('position', [x, y, dx, dy], 'FaceColor', [(3-a)/6,(3-a)/6,(3-a)/6]);
    end
end

axis([0.005, 0.994, -5.5, 5.499]);
set(gca,'XTick',0.01:0.01:0.99)
set(gca,'YTick',-5:5)
xlabel('lambda'); ylabel('state');
title('Optimal decisions for each state and discount factor');
rotateXLabels( gca(), 90 )
legend('d(s)=-3','d(s)=-1','d(s)=1', 'd(s)=3')
```