AE353: Additional Notes on Dynamic Programming and HJB Equation

(to be treated as an appendix to the presentation)

A. Borum T. Bretl M. Ornik P. Thangeda

1 Dynamic Programming

As introduced in the lecture, dynamic programming is a recursive method to find the optimal solution in discrete systems which have so called *optimal sub-structure*. In the lecture, we used as example where given a graph with rewards on edges and number of time steps T, we find the path that has the maximum reward. We will now look at a more practical example and use dynamic programming in MATLAB to find a solution to this problem.

Given a graph G illustrated in Fig. 1, with a cost on each edge, our objective is to find the cost of the shortest path from origin node a to the destination node g. Note that this is different from the illustration used in lecture in two ways: (i) we are trying to minimize the cost of the path here whereas in lecture we tried to maximize the reward and, (ii) there is no T here; we are supposed to find the shortest (least cost) path from a given node to another node.

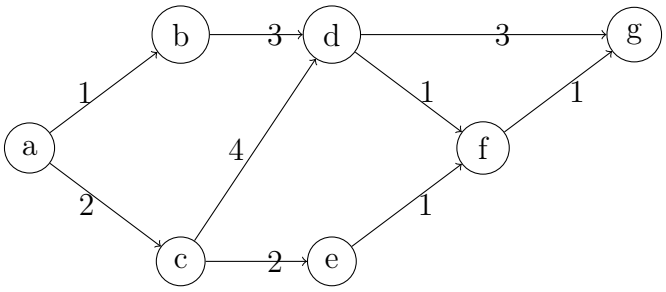

Figure 1: The graph of interest showing the the costs of different edges.

We use dynamic programming to find a solution to this problem. Let $V(n)$ denote the value of a node $n \in G$ where the value is defined as the least cost of a path from node n to destination node g. For example, node d in the graph has a value 3 as there is only a single direct path from node d to node g. Now, exploiting the optimal sub-structure, the value of a node that is not directly connected to destination g can be written in terms of the value of an intermediate node. For example, the value of node d can be written as

$$V(d) = \min\{3, 1 + V(f)\}$$

Updating the value function recursively until we find the value of origin node a, we can find the shortest path from node a to node g. To solve this using MATLAB, we first have to express the graph and the associated costs in a way that is easily accessible in MATLAB. We do this by defining a matrix G where $G(i, j)$ is equal to the cost of the edge between node i an j and 0 no edge between nodes i and j.

```
% represent graph as a matrix where indices are a = 1, b = 2, ..., g = 7.
    %   a,b,c,d,e,f,g
>> G = [0,1,2,0,0,0,0;
        0,0,0,3,0,0,0;
        0,0,0,4,2,0,0;
        0,0,0,0,0,1,3;
        0,0,0,0,0,1,0;
        0,0,0,0,0,0,1;
        0,0,0,0,0,0,0];
```

Now let us initialize the value of destination node g to 0 and all other nodes (whose values are not known yet) to ∞.

```
% initialize values
>>  V = [inf,inf,inf,inf,inf,inf,0];
```

Each iteration in dynamic programming updates the value of nodes backwards starting from the destination node g. We can perform the iteration using the following code:

```
% update the values
>>  for i = 1:7
        for j = 1:7
            if G(i,j) ~= 0
                if G(i,j)+ V(j) < V(i)
                    V(i) = G(i,j)+ V(j);
                end
            end
        end
    end
     % show V
     V

V =

   Inf   Inf   Inf     3   Inf     1     0
```

In the above code, we are iterating over all possible node combinations and updating the value function. The condition *if G(i,j) ~= 0* verifies that there exists an edge in graph G between the node i and node j. As you can see, after the first iteration, we only find the

values at nodes d and f. This is not surprising as these are the only two nodes that are directly connected to destination node g. However, 3 is not cost of the shortest path from d to g as the path $d- > f- > g$ has a cost of 2. We can account for paths involving two edges by performing another iteration:

```
% update the values
>>  for i = 1:7
        for j = 1:7
            if G(i,j) ~= 0
                if G(i,j)+ V(j) < V(i)
                    V(i) = G(i,j)+ V(j);
                end
            end
        end
    end
     % show V
     V

V =

   Inf     6     7     2     2     1     0
```

As you can see, the value of node d now converged to the least possible value of 2. To obtain the value of the shortest path from a to g, we run the value iteration until the values of all the states converge to an optimal value. The following code does exactly that:

```
% update the values
>>  while true
    Vprev = V;
    for i = 1:7
        for j = 1:7
            if G(i,j) ~= 0
                if G(i,j)+ V(j) < V(i)
                     V(i) = G(i,j)+ V(j);
                end
            end
        end
    end
    if V == Vprev
        break
    end
    Vprev = V;
    % show V
    V
end

V =
```

```
22
23      6      5      4      2      2      1      0
```

Hence, as obtained using the above code, the cost of the shortest path from origin node a to destination node g is 6.

2 Optimal Control

2.1 Statement of the problem

As explained in the earlier notes on introduction to optimal control, the following thing is called an *optimal control problem*:

$$\begin{aligned} \underset{u_{[t_0,t_f]}}{\text{minimize}} \quad & h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt \\ \text{subject to} \quad & \frac{dx(t)}{dt} = f(x(t), u(t)), \quad x(t_0) = x_0 \end{aligned} \tag{1}$$

Let's try to understand what it means.

- The statement
$$\underset{u_{[t_0,t_f]}}{\text{minimize}}$$
says that we are being asked to choose an input trajectory u that minimizes something. Unlike in the optimization problems we saw before, the decision variable u in this problem is a function of time. The notation $u_{[t_0,t_f]}$ is one way of indicating this. Given an initial time t_0 and a final time t_f, we are being asked to choose the value of $u(t)$ at all times in between, i.e., for all $t \in [t_0, t_f]$.

- The statement
$$\frac{dx(t)}{dt} = f(x(t), u(t)), \quad x(t_0) = x_0$$
is a constraint. It implies that we are restricted to choices of u for which there exists an x satisfying a given initial condition
$$x(t_0) = x_0$$
and satisfying the ordinary differential equation
$$\frac{dx(t)}{dt} = f(x(t), u(t)).$$
One example of an ordinary differential equation that looks like this is our usual description of a system in state-space form:
$$\dot{x} = Ax + Bu,$$

- The statement
$$h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt$$
says what we are trying to minimize—it is the cost function in this problem. Notice that the cost function depends on both x and u. Part of it—$g(\cdot)$—is integrated (i.e., "added up") over time. Part of it—$h(\cdot)$—is applied only at the final time. One example of a cost function that looks like this is
$$x(t_f)^T M x(t_f) + \int_{t_0}^{t_f} \left(x(t)^T Q x(t) + u(t)^T R u(t)\right) dt$$

2.2 The HJB equation (our new "first-derivative test")

As usual, there are a variety of ways to solve an optimal control problem. One way is by application of what is called the *Hamilton-Jacobi-Bellman Equation*, or "HJB." This equation is to optimal control what the first-derivative test is to optimization. To derive it, we will first rewrite the optimal control problem in "minimum form":
$$\underset{u_{[t_0,t_f]}}{\text{minimum}} \left\{ h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt \;:\; \frac{dx(t)}{dt} = f(x(t), u(t)), \quad x(t_0) = x_0 \right\}$$
Nothing has changed here, we're just asking for the minimum and not the minimizer. Next, rather than solve this problem outright, we will first state a slightly different problem:
$$\underset{\bar{u}_{[t,t_f]}}{\text{minimum}} \left\{ h(\bar{x}(t_f)) + \int_{t}^{t_f} g(\bar{x}(s), \bar{u}(s))ds \;:\; \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s), \bar{u}(s)), \quad \bar{x}(t) = x \right\} \tag{2}$$
The two changes that I made to go from the original problem to this one are:

- Make the initial time arbitrary (calling it t instead of t_0).
- Make the initial state arbitrary (calling it x instead of x_0).

I also made three changes in notation. First, I switched from x to $\bar{x}$ to avoid getting confused between x as initial condition and $\bar{x}$ as state trajectory. Second, I switched from u to $\bar{u}$ to be consistent with the switch from x to $\bar{x}$. Third, I switched from calling time t to calling time s to avoid getting confused with my use of t as a name for the initial time.

You should think of the problem (2) as a function itself. In goes an initial time t and an initial state x, and out comes a minimum value. We can make this explicit by writing
$$v(t,x) = \underset{\bar{u}_{[t,t_f]}}{\text{minimum}} \left\{ h(\bar{x}(t_f)) + \int_{t}^{t_f} g(\bar{x}(s), \bar{u}(s))ds \;:\; \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s), \bar{u}(s)), \quad \bar{x}(t) = x \right\} \tag{3}$$
We call $v(t,x)$ the *value function*. Notice that $v(t_0, x_0)$ is the solution to the original optimal control problem that we wanted to solve—the one where the initial time is t_0 and the initial

state is x_0. More importantly, notice that $v(t,x)$ satisfies the following recursion:

$$v(t,x) = \underset{\bar{u}_{[t,t+\Delta t]}}{\text{minimum}} \left\{ v\left(t+\Delta t, \bar{x}(t+\Delta t)\right) + \int_t^{t+\Delta t} g(\bar{x}(s),\bar{u}(s))ds \; : \right. \\ \left. \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s),\bar{u}(s)), \quad \bar{x}(t) = x \right\} \tag{4}$$

The reason this equation is called a "recursion" is that it expresses the function v in terms of itself. In particular, it splits the optimal control problem into two parts. The first part is from time t to time $t+\Delta t$. The second part is from time $t+\Delta t$ to time t_f. The recursion says that the minimum value $v(t,x)$ is the sum of the cost

$$\underset{\bar{u}_{[t,t+\Delta t]}}{\text{minimum}} \left\{ \int_t^{t+\Delta t} g(\bar{x}(s),\bar{u}(s))ds \; : \; \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s),\bar{u}(s)), \quad \bar{x}(t) = x \right\}$$

from the first part and the cost

$$\underset{\bar{u}_{[t+\Delta t,t_f]}}{\text{minimum}} \left\{ h(\bar{x}(t_f)) + \int_{t+\Delta t}^{t_f} g(\bar{x}(t),\bar{u}(t))dt \; : \; \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s),\bar{u}(s)), \quad \bar{x}(t+\Delta t) = \text{blah} \right\}$$

from the second part (where "blah" is whatever the state turns out to be, starting at time t from start x and applying the input $u_{[t,t+\Delta t]}$), which we recognize as the definition of

$$v\left(t+\Delta t, \bar{x}(t+\Delta t)\right).$$

We now proceed to approximate the terms in (4) by first-order series expansions. In particular, we have

$$\begin{aligned} v\left(t+\Delta t, \bar{x}(t+\Delta t)\right) &\approx v\left(t+\Delta t, \bar{x}(t) + \frac{d\bar{x}(t)}{dt}\Delta t\right) \\ &= v\left(t+\Delta t, x + f(x,\bar{u}(t))\Delta t\right) \\ &\approx v(t,x) + \frac{\partial v(t,x)}{\partial t}\Delta t + \frac{\partial v(t,x)}{\partial x} f(x,\bar{u}(t))\Delta t \end{aligned}$$

and we also have

$$\begin{aligned} \int_t^{t+\Delta t} g(\bar{x}(s),\bar{u}(s))ds &\approx g(\bar{x}(t),\bar{u}(t))\Delta t \\ &= g(x,\bar{u}(t))\Delta t. \end{aligned}$$

If we plug both of these results into (4), we find

$$v(t,x) = \underset{\bar{u}_{[t,t+\Delta t]}}{\text{minimum}} \Bigg\{ v\left(t+\Delta t, \bar{x}(t+\Delta t)\right) + \int_t^{t+\Delta t} g(\bar{x}(s),\bar{u}(s))ds \ : \\ \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s),\bar{u}(s)), \quad \bar{x}(t) = x \Bigg\}$$

$$= \underset{\bar{u}_{[t,t+\Delta t]}}{\text{minimum}} \Bigg\{ v(t,x) + \frac{\partial v(t,x)}{\partial t}\Delta t + \frac{\partial v(t,x)}{\partial x} f(x,\bar{u}(t))\Delta t + g(x,\bar{u}(t))\Delta t \ : \\ \frac{d\bar{x}(s)}{ds} = f(\bar{x}(s),\bar{u}(s)), \quad \bar{x}(t) = x \Bigg\}.$$

Notice that nothing inside the minimum depends on anything other than t, x, and $\bar{u}(t)$. So we can drop the constraint and make $\bar{u}(t)$ the only decision variable. In fact, we might as well replace $\bar{u}(t)$ simply by "u" since we only care about the input at a single instant in time:

$$v(t,x) = \underset{u}{\text{minimum}} \Bigg\{ v(t,x) + \frac{\partial v(t,x)}{\partial t}\Delta t + \frac{\partial v(t,x)}{\partial x} f(x,u)\Delta t + g(x,u)\Delta t \Bigg\}.$$

Also, notice that

$$v(t,x) + \frac{\partial v(t,x)}{\partial t}\Delta t$$

does not depend on u, so it can be brought out of the minimum:

$$v(t,x) = v(t,x) + \frac{\partial v(t,x)}{\partial t}\Delta t + \underset{u}{\text{minimum}} \Bigg\{ \frac{\partial v(t,x)}{\partial x} f(x,u)\Delta t + g(x,\bar{u}(t))\Delta t \Bigg\}.$$

To simplify further, we can subtract $v(t,x)$ from both sides, then divide everything by Δt:

$$0 = \frac{\partial v(t,x)}{\partial t} + \underset{u}{\text{minimum}} \Bigg\{ \frac{\partial v(t,x)}{\partial x} f(x,u) + g(x,u) \Bigg\}. \tag{5}$$

Equation (5) is called the *Hamilton-Jacobi-Bellman Equation*, or simply the HJB equation. As you can see, it is a partial differential equation, so it needs a boundary condition. This is easy to obtain. In particular, going all the way back to the definition (3), we find that

$$v(t_f,x) = h(x). \tag{6}$$

The importance of HJB is that if you can find a solution to (5)-(6)—that is, if you can find a function $v(t,x)$ that satisfies the partial differential equation (5) and the boundary condition (6)—then the minimizer u in (5), evaluated at every time $t \in [t_0, t_f]$, is the solution to the optimal control problem (1). You might object that (5) "came out of nowhere." First of all, it didn't. We derived it just now, from scratch. Second of all, where did the first-derivative test come from? Could you derive that from scratch? (Do you, every time you need to use it? Or do you just use it?)

2.3 Solution approach

The optimal control problem

$$\begin{aligned} \underset{u_{[t_0,t_f]}}{\text{minimize}} \quad & h(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt \\ \text{subject to} \quad & \frac{dx(t)}{dt} = f(x(t), u(t)), \quad x(t_0) = x_0 \end{aligned}$$

can be solved in two steps:

- Find v:

$$0 = \frac{\partial v(t,x)}{\partial t} + \underset{u}{\text{minimum}} \left\{ \frac{\partial v(t,x)}{\partial x} f(x,u) + g(x,u) \right\}, \qquad v(t_f, x) = h(x)$$

- Find u:

$$u(t) = \underset{u}{\text{minimize}} \ \frac{\partial v(t,x)}{\partial x} f(x,u) + g(x,u) \qquad \text{for all } t \in [t_0, t_f]$$