Dynamic Programming for Optimal Control

Discrete- and continuous-time versions

Zdeněk Hurák
June 25, 2021

DYNAMIC PROGRAMMING is a powerul framework for solving sequential decision problems, that is, problems where decisions are made one after another. It is based on a simple yet powerful idea now known as *(Bellman's) principle of optimality*. Our first goal in this lecture is to introduce this principle. The problem of optimal control is a perfect (but far from the only) opportunity for application of such framework, therefore our second goal is to formulate a general optimal control problem as a dynamic program. For the popular LQ-optimal problem, dynamic programming serves as a theoretical framework within which the Riccati-equation based solution (now already familiar to us from the previous lecture) can be derived. Although primarily intended for discrete-time systems, the framework can also be extended to continuous-time systems, wherein the principle of optimality is captured by the celebrated *Hamilton-Jacobi-Bellman's (HJB) equation*. For a general nonlinear problem including the popular LQ-optimal control with bound contraints or quantized controls (on-off control), dynamic programming provides a numerical computational procedure yielding a feedback controller in the form of a look-up table. The latter use of dynamic programming is only tractable for simplest textbook problems. For anything more realistic, dynamic programming suffers from the *curse of dimensionality*. A remedy is called *approximate dynamic programming* (ADP), in which the lookup table is approximate by a function. Yet another extension that is gaining popularity within the control field is called *reinforcement learning (RL)*, which can be viewed as an instance of an adaptive control. This text does not include details on ADP and RL but can be used for building prerequisites for introducing ADP and RL.

1 Bellman's principle of optimality and dynamic programming

We start by considering an everyday problem of navigating our car from one city to another. Say, we want to travel from Prague to Ostrava in an optimal way. In particular, we may want to minimize the total time. Fig. 1 shows what the online navigator (`https://en.mapy.cz/`) returns.

Obviously, the optimal route goes through (well, actually around) Brno and then through (around) Olomouc.

Now, let's imagine a situation that we are heading to Ostrava to meet our business partners some of which are travelling from Brno. A natural question pops up: "is our

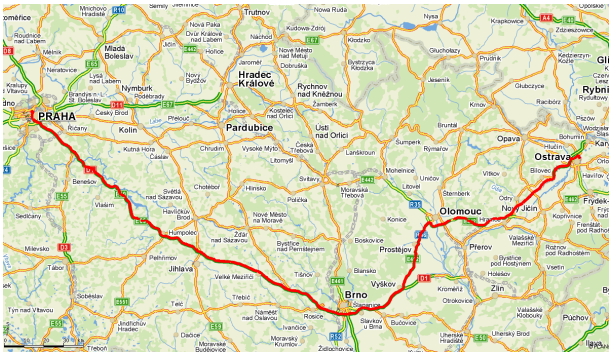

Figure 1: Optimal (shortest time) route from Praha to Ostrava. Reusable aso as an optimal route from Brno to Ostrava.

optimal route reusable for our business partners from Brno?" An answer that is hardly surprising is: "yes!" Knowing that our optimal route from Prague to Ostrava goes through Brno, we can immediately conclude that the optimal route for the colleagues from Brno is identical to our part of the route from Brno to Ostrava. This intuitive and simple reasoning was formalized in 1950s by R. E. Bellman:

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

Let's now explore this idea a bit more quantitatively using a simple computational example of finding a shortest path in a graph in Fig. 2.

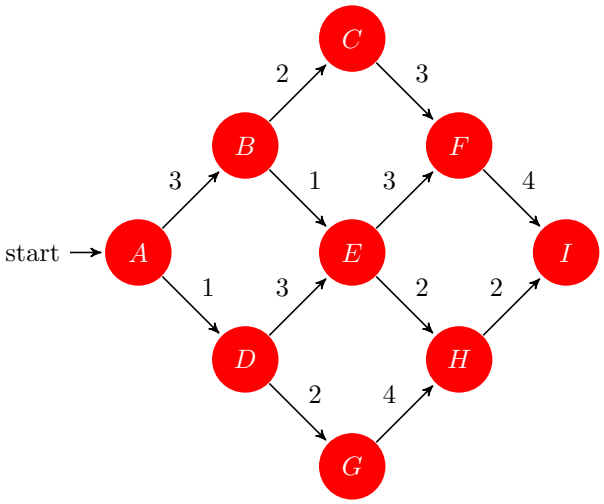

Figure 2: Graph through which a shortest path from the initial node A to the final node I is to be found. The labels on the edges give transition costs (in our case these are the times to get from one node to another).

What are possible solution strategies? First, we can start enumerating all the possible routes (paths) from A to I and calculate their costs (by summing the costs

of the participating edges). Needless to say, this strategy based on enumeration scale very badly with the growing number of nodes.

Alternatively, we solve the problem using dynamic programming and relying on Bellman's principle of optimality. The crucial attribute of this strategy is that we *proceed backwards*. We start at the very final stage. Since it is just one here, there is nothing we can do here but in general, in case of several possible final stages, we associate the terminal costs with them. We then proceed backwards to the last but one (or $(N-1)$th) stage. These are F and H stages. Again, in these two stages there is no freedom as for the actions but for each of them we can record the *cost to go*. There are 4 for F node and 2 for the H node. Things are only getting interesting if we now proceed one more step towards the past: we proceed to the stage $N-2$. We now have to consider three possible states: C, E and G. For the nodes C and G there is still just one action and we can only record their costs to go. The cost for the C node can be computed as the cost for the immediate transition from C to F plus the cost for the F node which we recorded previously, that is, $3+4=7$. We record the value of 7 with the C node. Similarly for the G node. For the E node there are two possible actions—two possible decisions to be made, two paths/routes to take. Either to the left (or, actually, up), which would bring us to the node F, or to the right (or down), which would bring us to the node H. We compute the costs to go for both decisions and choose the decision with a smaller cost. Here the cost of the decision to go to the left is composed of the cost to transition to F plus the cost to go from F, that is, $3+4=7$. The cost to go for the decision to go right is composed of the transition cost from E to H plus the cost to go from H, that is, $2+2=4$. Obviosly, the optimal decision is to go right, that is, to the node H. Here on top of the value of the optimal (smallest) cost to go from the node we also record the optimal decision (go to the right/down). And we proceed one stage backwards a start analyzing the costs to go for the nodes B and D. Again we record the optimal costs to go and the actual optimal decisions. The last step is done by analyzing the same issues for the initial node A. Note that here coincidently both decisions have the same cost to go, hence both possible decisions/actions are optimal and we can just toss a coin.

As a result of this backward sweep we have two arrays of numbers—an array (or table) of optimal costs to go for each node, and an array of optimal decisions (in our case there is just one possible step that could be made or there are two possibilities). Once we are done with the whole graph, we can actually discard the array of costs and only keep the array with optimal decisions.

This artificial looking problem of finding a shortest path in a graph is perfectly matching our original problem of finding an optimal route from Prague to Ostrava. Besides obtaining an optimal route as fixed sequence of cities (nodes) to travel through (around), we get a nice bonus. Since the optimal strategy is realized as a *look-up table*—it can be viewed as a feedback controller because at every modelled intersection we look up an optimal decision from the table. This may come in handy if for some reason or the other we deviate from the optimal route but still find ourselves in a city (node) that has an entry in our table.

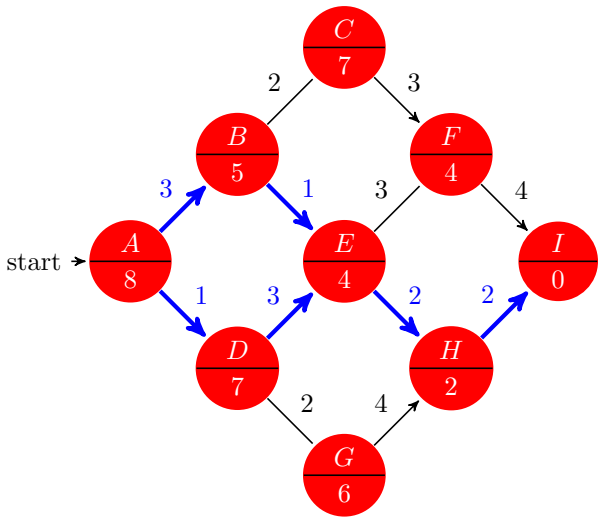

Figure 3: Graph with the optimal path(s) from A to I highlighted. The optimal cost-to-go is shown for each node.

2 Dynamic programming and discrete-time optimal control

2.1 Formulation of the optimal control problem (to be solved using DP)

Let's now use this apply the principle of dynamic programming in the formal setting of discrete-time control systems. In particular, we consider the system modelled by

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) \tag{1}$$

on the interval $k \in [i, N]$, with the initial state $\mathbf{x}_i$ given (say, $\mathbf{x}_i = \mathbf{r}_i$), and we aim at minimizing the cost function

$$J_i\left(\mathbf{x}_i, \left[\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_{N-1}\right]\right) = \phi(\mathbf{x}_N) + \sum_{k=i}^{N-1} L_k(\mathbf{x}_k, \mathbf{u}_k). \tag{2}$$

Three comments are now appropriate

- Note that while looking at the right hand side of the above equation, the cost function is clearly a function of the full sequence $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_N$ of the state vectors, all the states starting with $\mathbf{x}_{i+1}$ can be expressed using the initial state $\mathbf{x}_i$ and the corresponding (sub)sequence of controls $\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots$. Therefore, we can write the cost function as a function of the initial state, initial time, and the sequence of controls.
- The optimization is to be conducted over the sequence $\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_N$ of controls, while the initial state $\mathbf{x}_i$ is typically (but not necessarily always) regarded as fixed (given) as serves as a parameter for the optimization.

- Let's discuss the notation a bit. It turns out that here in this chapter/lecture it is even more important than usually to use the notation conciously to our full advantage. In particular, recall that the lower indices reflect the dependence on the (discrete) time: we read $\mathbf{x}_k$ as the state (vector) at the discrete time k and $\mathbf{u}_k$ as the input (or control) at the discrete time k. But we should then use the same interpretation for the functions $\mathbf{f}_k()$, $L_k()$ and $J_k()$, that is, the *discrete time k is another argument* for these functions. We could perhaps write these as $\mathbf{f}(\cdot,\cdot,k)$, $L(\cdot,\cdot,k)$ and $J(\cdot,\cdot,k)$ to better indicate that k is really an argument for these functions. But instead we typeset the discrete time k as the lower index to make it compatible with the way we indicate the time dependence of the signals $\mathbf{x}_k$ and $\mathbf{u}_k$.

Having introduced the cost function J parameterized by the initial state, initial time and the full sequence of controls, we now introduce the optimal cost function $J*$

$$J_i^*(\mathbf{x}_i) = \min_{\mathbf{u}_i,\, i\in[i,N-1]} J_i\left(\mathbf{x}_i, \left[\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_{N-1}\right]\right) \tag{3}$$

Let summarize the difference between the J and $J*$ functions. Understanding the difference is vital for understanding what is to come next. While J depends on the (initial) state, (initial) time and the sequence of controls applied over the whole interval, $J*$ only depends on the state and time.

2.2 Bellman's principle of optimality aka the principle of dynamic programming

Assume we have already found an optimal control from any given state $\mathbf{x}_{k+1}$ at time $k+1$ on, i.e., we already have $\mathbf{u}_{k+1}, \mathbf{u}_{k+2}, \ldots, \mathbf{u}_{N-1}$ yielding the optimal cost $J_{k+1}^*(\mathbf{x}_{k+1})$. Don't ask me now where we get such optimal sequence, we just *assume* we have it.

Now, if we apply arbitrary (not necessarily optimal) control $\mathbf{u}_k$ at a given state $\mathbf{x}_k$ at time k, the cost is

$$J_k(\mathbf{x}_k, \left[\mathbf{u}_k, \mathbf{u}_{k+1}, \ldots, \mathbf{u}_{N-1}\right]) = L_k(\mathbf{x}_k, \mathbf{u}_k) + J_{k+1}^*(\mathbf{x}_{k+1}). \tag{4}$$

I highlighted the (sub)sequence of controls on the left hand side starting with $\mathbf{u}_{k+1}$ to emphasize that these were already fixed by the assumed availability of the optimal cost function J_{k+1}^*.

According to Bellman, the optimal cost from time k on is

$$\boxed{J_k^*(\mathbf{x}_k) = \min_{\mathbf{u}_k} \left(L_k(\mathbf{x}_k, \mathbf{u}_k) + J_{k+1}^*(\mathbf{x}_{k+1})\right).} \tag{5}$$

Hence, at a given state $\mathbf{x}_k$ at a given time k the optimization is performed over only one control $\mathbf{u}_k$ and not the whole sequence!

What we have got in (5) is a recursion scheme. Is initialized at $k+1 = N$ by invoking $J_N^*(\mathbf{x}_N) = \phi(\mathbf{x}_N)$ and proceeds backwards in time.

Let's emphasize here that the minimization in (5) needs to be performed over the sum $L_k(\mathbf{x}_k, \mathbf{u}_k) + J_{k+1}^*(\mathbf{x}_{k+1})$ because $\mathbf{x}_{k+1}$ is a function of $\mathbf{u}_k$ (recall that $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k)$). We could have written (5) perhaps as

$$J_k^*(\mathbf{x}_k) = \min_{\mathbf{u}_k} \left(L_k(\mathbf{x}_k, \mathbf{u}_k) + J_{k+1}^*(\mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k))\right). \tag{6}$$

We will now show how to actually apply this result to solve an optimal control problem.

2.3 Dynamic programming as a numerical procedure for computing a feedback controller in the form of a look-up table

In the discrete-time control setting, the controls are computed and applied one after another at discrete time instances. This is nicely in alignment with the sequential character of decision making in dynamic programming. However, at a given time, the number of possible states is infinite! In particular, acceptable states constitute a subset of $\mathbb{R}^n$, typically a box given by the minimum and maximum acceptable values for each component of $\mathbf{x}$. Similarly, the number of possible controls (actions, decisions) at a given time and state can be infinite in general, again lower- and upper-bounded real variables (although sometimes quantized controls appear naturally, for instance in on-off control). A "dirty" trick can be applied here—discretization (or gridding or sampling) of the space of states and controls. A sketch is in Fig. 4 for a first-order system, for which the full state space is a real line and the realistically restricted states are in the interval $[x_{\min}, x_{\max}]$.

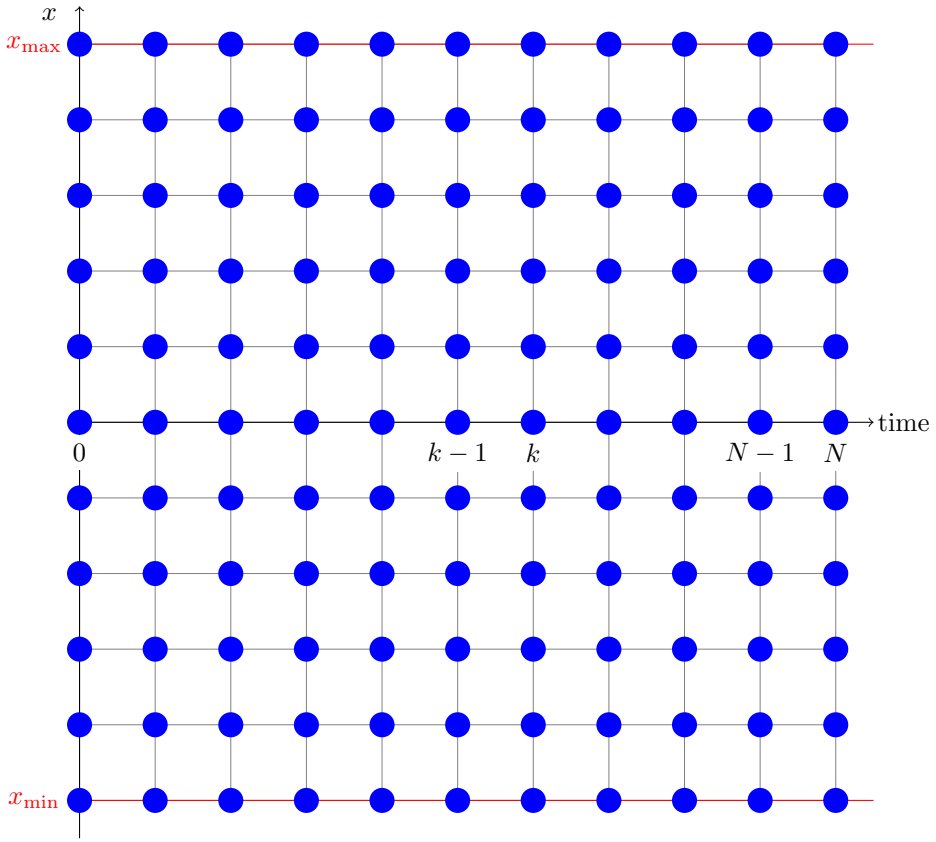

Figure 4: Discretized time and state space for dynamic programming

We now start populating two tables in which entries correspond to the grid points in the state space—one for the optimal costs $J_k^*()$ and the other for the optimal controls $u_k(x_k)$. As in the shortest path problem we start at $k = N$. For all grid points in the state space we evaluate the cost $J_N(x_N^i)$. Since we are at the end of the control interval, there is no decision to be made (no controls to compute) at this

moment (we can just set $u_N(x_N) = 0$ or ignore this step altogether). We record the computed costs, and proceed to $k = N - 1$.

At $k = N - 1$ we again iterate through all the grid points in the state space. Let's label the iterations through the state space with the upper index (i). At each point $x_{N-1}^{(i)}$ we perform another iteration, this time through all the grid points in the space of controls. Let's label the iterations through the controls with the upper index (j). For each control $u_{N-1}^{(j)}(x_{N-1}^{(i)})$ we compute the sum of the cost of the transition from $x_{N-1}^{(i)}$ to the next state x_N and the optimal cost to go from the x_N state on, which we already have from the previous step. After iterating through all possible controls at a given state, we pick the one with minimum cost and and record both this value $J_{N-1}^*(x_{N-1}^{(i)})$ of the optimal cost-to-go and also the corresponding controls $u_{N-1}^*(x_{N-1}^{(i)})$, see (5). We decrease the time again to $k = N - 2 \ldots$ and repeat the whole procedure.

2.3.1 Interpolation often needed

Unless we are particularly lucky, the evolution of the states of a given discrete-time model (1) would not be restricted to our grid points. Some interpolation would then be needed.

...

2.3.2 Combinatorial complexity

Restrict ourselves to first-order and single-input systems (extension of this analysis to higher-order and multiple-input systems is straightforward). That is, the state vector $\mathbf{x}$ actually contains a single a scalar variable and the vector of controls $\mathbf{u}$ is one-dimensional as well. At every discrete time k, the algorithm iterates over the N_x grid points of $\mathbf{x}$ and N_u grid points of $\mathbf{u}$. Hence, the total number of evaluations of the sum insided the bracket on the right hand side of (1) over the whole time interval is $N_x N_u N$. One possible observation is that the computational complexity scales linearly with the length of the control interval.

Contrast this with the strategy based on full enumeration of all possible control sequences. For $N = 1$, starting at a single $\mathbf{x}_0$, there are (trivially) N_u possible control sequences, hence in total $N_x N_u$ possible control sequences if arbitrary initial state is considered. For $N = 2$, from a single $\mathbf{x}_0$, there are now N_u^2 possible control sequences, hence in total $N_x N_u^2$. Obviously, on an interval of lenth N the number of possible control sequences from which we should choose the optimal one scales as $N_x N_u^N$. Dynamic programming offers a significant saving in the computational effort.

2.4 Discrete LQR via dynamic programming

Consider a linear discrete-time system modelled by

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{7}$$

with the performance index

$$J_0(\mathbf{x}_0, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}) = \frac{1}{2}\mathbf{x}_N^{\mathrm{T}}\mathbf{S}_N\mathbf{x}_N + \frac{1}{2}\sum_{k=0}^{N-1}\left(\mathbf{x}_k^{\mathrm{T}}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^{\mathrm{T}}\mathbf{R}\mathbf{u}_k\right) \tag{8}$$

with $\mathbf{S}_N \geq 0, \mathbf{Q} \geq 0, \mathbf{R} > 0$. We now invoke the principle of optimality, that is, we start at the end

$$J_N^*(\mathbf{x}_N) = \frac{1}{2}\mathbf{x}_N^{\mathrm{T}}\mathbf{S}_N\mathbf{x}_N \tag{9}$$

and go backwards (decrement to $k = N-1$)

$$J_{N-1}^*(\mathbf{x}_{N-1}) = \min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \left[\underbrace{L(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) + J_N^*(\mathbf{x}_N)}_{J_{N-1}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1})} \right]. \tag{10}$$

We now expand the expression for the the cost to go

$$\begin{aligned}
J_{N-1} &= \frac{1}{2}\left(\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{Q}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{N-1}\right) + J_N^* \\
&= \frac{1}{2}\left(\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{Q}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{N-1}\right) + \frac{1}{2}\mathbf{x}_N^{\mathrm{T}}\mathbf{S}_N\mathbf{x}_N \\
&= \frac{1}{2}\left(\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{Q}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{N-1} + \mathbf{x}_N^{\mathrm{T}}\mathbf{S}_N\mathbf{x}_N\right) \\
&= \frac{1}{2}\left[\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{Q}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{N-1} + (\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}} + \mathbf{u}_{N-1}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}})\mathbf{S}_N(\mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1})\right] \\
&= \frac{1}{2}\left[\mathbf{x}_{N-1}^{\mathrm{T}}(\mathbf{Q} + \mathbf{A}^{\mathrm{T}}\mathbf{S}_N\mathbf{A})\mathbf{x}_{N-1} + 2\mathbf{x}_{N-1}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{S}_N\mathbf{B}\mathbf{u}_{N-1} + \mathbf{u}_{N-1}^{\mathrm{T}}(\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{S}_n\mathbf{B})\mathbf{u}_{N-1}\right].
\end{aligned}$$

We assumed no constraint on $\mathbf{u}_{N-1}$, hence finding the minimum of J_{N-1} is as easy as setting the gradient of the cost to go to zero

$$\mathbf{0} = \nabla_{\mathbf{u}_{N-1}} J_{N-1} = (\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{S}_n\mathbf{B})\mathbf{u}_{N-1} + \mathbf{B}^{\mathrm{T}}\mathbf{S}_N\mathbf{A}\mathbf{x}_{N-1}, \tag{11}$$

which leads to

$$\mathbf{u}_{N-1}^* = -\underbrace{(\mathbf{B}^{\mathrm{T}}\mathbf{S}_N\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}_N\mathbf{A}}_{\mathbf{K}_{N-1}}\mathbf{x}_{N-1}. \tag{12}$$

The optimal cost can be obtained by substituting $\mathbf{u}_{N-1}^*$ into J_{N-1}

$$J_{N-1}^* = \frac{1}{2}\mathbf{x}_{N-1}^{\mathrm{T}}\underbrace{\left[(\mathbf{A} - \mathbf{B}\mathbf{K}_{N-1})^{\mathrm{T}}\mathbf{S}_N(\mathbf{A} - \mathbf{B}\mathbf{K}_{N-1}) + \mathbf{K}_{N-1}^{\mathrm{T}}\mathbf{R}\mathbf{K}_{N-1} + \mathbf{Q}\right]}_{\mathbf{S}_{N-1}}\mathbf{x}_{N-1}. \tag{13}$$

Decrement to $k = N-2, N-3, \ldots$ the rest of the story is known to you from the previous lecture...

3 Dynamic programming and continuous-time optimal control—HJB equation

Consider the continuous-time system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{14}$$

with the cost function

$$J(\mathbf{x}(t_0), \mathbf{u}(\cdot), t_0) = \phi(\mathbf{x}(T), T) + \int_{t_0}^{T} L(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{15}$$

Optionally we can also consider constraints on the state at the final time (be it a particular value or some set of values)

$$\psi(\mathbf{x}(T), T) = 0. \tag{16}$$

We now split the time interval t, T into two parts and structure the cost function accordingly

$$J(\mathbf{x}(t), \mathbf{u}(\cdot), t) = \int_t^{t+\Delta t} L(\mathbf{x}, \mathbf{u}, \tau) d\tau + \underbrace{\int_{t+\Delta t}^{T} L(\mathbf{x}, \mathbf{u}, \tau) d\tau + \phi(\mathbf{x}(T), T)}_{J(\mathbf{x}(t+\Delta t), \mathbf{u}(t+\Delta t), t+\Delta t)} \tag{17}$$

Bellman's principle of optimality gives

$$J^*(\mathbf{x}, t) = \min_{\mathbf{u}(\tau),\ t \le \tau \le t+\Delta t} \left[\int_t^{t+\Delta t} L(\mathbf{x}, \mathbf{u}, \tau) d\tau + J^*(\mathbf{x} + \Delta\mathbf{x}, t + \Delta t) \right] \tag{18}$$

Now, perform Taylor series expansion of $J^*(\mathbf{x} + \Delta\mathbf{x}, t + \Delta t)$ about $(\mathbf{x}, t)$

$$J^*(\mathbf{x}, t) = \min_{\mathbf{u}(\tau),\ t \le \tau \le t+\Delta t} \left[L\Delta t + J^*(\mathbf{x}, t) + (\nabla_{\mathbf{x}} J^*)^{\mathrm{T}} \Delta\mathbf{x} + \frac{\partial J^*}{\partial t} \Delta t + \mathcal{O}((\Delta t)^2) \right].$$

Use

$$\Delta\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)\Delta t \tag{19}$$

and note that J^* and J_t^* are independent of $\mathbf{u}(\tau),\ t \le \tau \le t + \Delta t$

$$\cancel{J^*(x,t)} = \cancel{J^*(x,t)} + \frac{\partial J^*}{\partial t} \Delta t + \min_{u(\tau),\ t \le \tau \le t+\Delta t} \left(L\Delta t + (\nabla_{\mathbf{x}} J^*)^{\mathrm{T}} f \Delta t \right). \tag{20}$$

Assuming $\Delta t \to 0$ leads to the celebrated Hamilton-Jacobi-Bellman equation

$$\boxed{-\frac{\partial J^*}{\partial t} = \min_{\mathbf{u}(t)} \left(L + (\nabla_{\mathbf{x}} J^*)^{\mathrm{T}} \mathbf{f} \right).} \tag{21}$$

Since this is a differential equation, boundary value(s) must also be specified

$$J^*(\mathbf{x}(T), T) = \phi(\mathbf{x}(T), T), \qquad \text{on the hypersurface } \psi(\mathbf{x}(T), T) = 0. \tag{22}$$

By the way, recall $H(\mathbf{x}, \mathbf{u}, \lambda, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$. Hence, HJB equation can also be written as

$$\boxed{-\frac{\partial J^*}{\partial t} = \min_{\mathbf{u}(t)} H(\mathbf{x}, \mathbf{u}, \nabla_{\mathbf{x}} J^*, t).} \tag{23}$$

What we have just derived is one of the most profound results in optimal control—Hamiltonian must be minimized in order to get optimal control. We will exploit it practically below and come back to the topic in one of the next lectures on Pontryagin's principle of maximum.

4 Deriving continuous-time LQR from HJB equation

Consider the system modelled by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{24}$$

and the cost function

$$J(\mathbf{x}(t_0), \mathbf{u}(\cdot), t_0) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}(T)\mathbf{S}(T)\mathbf{x}(T) + \frac{1}{2}\int_{t_0}^{T} \left(\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}\right) dt. \tag{25}$$

The Hamiltonian is

$$H = \frac{1}{2}\left(\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}\right) + \boldsymbol{\lambda}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}\right) \tag{26}$$

and according to (23) our goal is to minimize H, which enforces the condition on its gradient

$$\mathbf{0} = \nabla_{\mathbf{u}} H = \mathbf{R}\mathbf{u} + \mathbf{B}^{\mathrm{T}}\boldsymbol{\lambda}, \tag{27}$$

so

$$\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\boldsymbol{\lambda} \tag{28}$$

and the Hessian is

$$\nabla^2_{uu}\mathbf{H} = \mathbf{R} > 0. \tag{29}$$

The optimal Hamiltonian is

$$H^* = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \frac{1}{2}\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\boldsymbol{\lambda} \tag{30}$$

Setting $\boldsymbol{\lambda} = (\nabla_{\mathbf{x}} J^*)^{\mathrm{T}}$, the HJB equation is

$$\boxed{-\frac{\partial J^*}{\partial t} = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + (\nabla_{\mathbf{x}} J^*)^{\mathrm{T}}\mathbf{A}\mathbf{x} - \frac{1}{2}(\nabla_{\mathbf{x}} J^*)^{\mathrm{T}}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\nabla_{\mathbf{x}} J^*} \tag{31}$$

and the boundary condition is

$$J^*(x, T) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}(T)\mathbf{S}(T)\mathbf{x}(T). \tag{32}$$

We can now proceed by making an assumption ("sweep") that

$$J^*(x, t) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}(t)\mathbf{S}(t)\mathbf{x}(t) \tag{33}$$

After a few technical steps (see, for example [3]), we finally get the familiar Riccati equation

$$-\dot{\mathbf{S}} = \mathbf{A}^{\mathrm{T}}\mathbf{S} + \mathbf{S}\mathbf{A} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S} + \mathbf{Q} \tag{34}$$

and the formula for the optimal control

$$\mathbf{u}^*(t) = -\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}(t)\mathbf{x}(t). \tag{35}$$

Note that we have just seen (well, I have not documented all the technical steps, I admit) the equivalence between a first-order linear PDE and first-order nonlinear ODE.

5 Approximate dynamic programming and Reinforcement learning

[TBD]

6 Further reading

This lecture was prepared to a major extent with the help of the sixth chapter of [3], but the more affordable (cheaper) [2] follows the same line of exposition and is very recommendable. The continuous-time version of the framework problem is nicely introduced in [1], for which a few copies are available for the students of the course at the NTK library.

References

[1] Brian D. O. Anderson and John B. Moore. *Optimal Control: Linear Quadratic Methods*. Dover Publications, February 2007.

[2] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, April 2004.

[3] Frank L. Lewis and Vassilis L. Syrmos. *Optimal Control*. Wiley-Interscience, 2nd edition, October 1995.