1 A First Example

Optimal Quadratic Control

Our book starts with a motivating chapter to answer the question: *Why is it worthwhile to develop system theory?* To do so, we jump fearlessly into the very center of our methods, using a simple and straightforward example of *optimal control*. Although optimization is not our main subject – that is system theory – it provides for one of the main application areas, namely the optimization of the performance of a dynamical system in a time-variant environment (think of driving a car or sending a rocket to the moon). The chapter starts out with a review of the *Moore–Penrose pseudo-inverse*, which is a central concept of matrix algebra, used throughout the book. Next it describes a simple case of optimal control, which is first solved in a global way and then in the much more attractive recursive way called *dynamic programming*. The chapter then ends by showing how the method generalizes to linear, discrete-time, time-variant models.

Menu

Hors d'oeuvre
The Moore–Penrose Inverse

First Course
Discovering the Power of
Dynamic Programming by Rowing

Second Course
The Bellman Problem: Optimal Quadratic Control
of a Linear Dynamical System

Dessert
Notes

1.1 Matrix Algebra Preliminary: The Moore–Penrose Inverse

Solving a system of linear equations $Ax = b$ is perhaps the first motivation for studying linear algebra. Here, A is a square $m \times m$ matrix with scalar entries, b is a given vector of dimension m, and x is an unknown vector of the same dimension m. When A has independent columns, these columns span the full real m-dimensional vector space $\mathcal{R}^m$ (or in the complex case $\mathcal{C}^m$), and there exists a unique solution $x = A^{-1}b$. In this case, the *range* of A, as an operator acting on x, is the full space $\mathcal{R}^m$, and there is a

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

unique linear combination of columns of A that generates the given b of dimension m. Many other situations are, of course, conceivable: there may be fewer equations than unknowns (A has dimensions $n \times m$ with $n < m$) or just more ($n > m$), and the equations given may turn out to be contradictory. The result is that an infinite number of solutions might exist or just no solution at all. Hence, a more general approach is needed, and it is provided by the Moore–Penrose inverse.

An overdetermined situation ($n > m$) often arises as a result of many (similar or different) measurements involving the same unknown quantities in x, and then one wonders what to do about the resulting incompatibilities (in measurement practice involving more than one unknown variable, one should use a variety of measurement methods to obtain a nonsingular system of equations with more equations n than unknowns m). Let us look at such an overdetermined case in more detail.

Typically, when there are too many equations for the unknown quantities, these equations will be contradictory, and no exact solution for $Ax = b$ will exist. Rather, for each trial x, there will be an associated error $e_x = b - Ax$, and, assuming all measurements to be equally important, one may want to find x's that minimize the quadratic error $e_x' e_x = \sum_{i=1:n} [e_x]_i^2$. More generally, one might give weight to the importance of individual measurements, particularly when they lead to quantities with different dimensions.

Therefore, we consider the error equation

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,m} \\ \vdots & \vdots & \vdots \\ A_{m,1} & \cdots & A_{m,m} \\ \vdots & \vdots & \vdots \\ A_{n,1} & \cdots & A_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} - \begin{bmatrix} b_1 \\ \vdots \\ b_m \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} [e_x]_1 \\ \vdots \\ [e_x]_m \\ \vdots \\ [e_x]_n \end{bmatrix} \tag{1.1}$$

and try to minimize the error e_x in the least-squares sense. Matrix A has dimensions $n \times m$ with $n \geq m$, and let us assume that the columns of A are linearly independent, but since $n \geq m$, they span only a subspace of dimension m, and not the whole space $\mathcal{R}^n$ to which b belongs, unless $n = m$.

Using the quadratic norm $\|a\|_2 = \sqrt{a'a}$ for any vector a, we may write

$$x_{\min} = \text{argmin}_x \|b - Ax\|_2, \tag{1.2}$$

meaning $x_{\min}$ is an argument x that minimizes the expression (notice: the square root does not matter for the minimization). We show:

Proposition 1.1 *The solution to the minimization problem* $\text{argmin}_x \|b - Ax\|_2$, *where A is an $n \times m$ matrix ($n \geq m$) with independent columns, is unique and is given by*

$$x_{\min} = A^\dagger b, \tag{1.3}$$

in which $A^\dagger := (A'A)^{-1}A'$.

Moreover, the minimal error vector $e_{\min}$ is given by

$$e_{\min} = (I - \Pi_A)b, \tag{1.4}$$

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

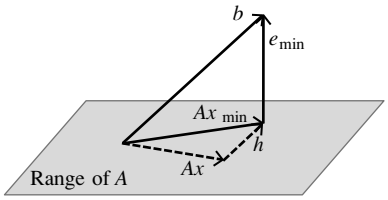

Figure 1.1 Best linear quadratic approximation.

in which $\Pi_A := AA^\dagger$ *is the orthogonal projection on the range of* A *in* $\mathcal{R}^n$*, and* $(I - AA^\dagger)$ *is the projection on the orthogonal complement of the range of* A*.*

Proof

(We follow the traditional orthogonality argument.) For any x of dimension m, Ax will lie in the linear subspace generated by the columns of A, that is, the *range* of A. The best $x_{\min}$ in a least-squares sense will then be such that the least-squares error $e_{\min} = b - Ax_{\min}$ is orthogonal on the range space of A. Expressing the orthogonality of the error vector on the columns of A, we require

$$A'(b - Ax_{\min}) = 0, \tag{1.5}$$

and hence $x_{\min} = (A'A)^{-1}A'b$ since $A'A$ is an $m \times m$ nonsingular matrix thanks to the assumed independence of the columns of A. The solution is unique, because for any x we have $b - Ax = (b - Ax_{\min}) + h$ with $h = A(x_{\min} - x) \perp e_{\min}$ since $e_{\min}$ is orthogonal to the range of A, see Fig. 1.1 for an illustration, and hence $\|b - Ax\|^2 = \|e_{\min}\|^2 + h^2 > \|e_{\min}\|^2$ when $h \neq 0$.

Next, one checks that $\Pi_A = AA^\dagger = A(A'A)^{-1}A'$ is indeed an orthogonal projection operator, for

1. it is a projection operator because $\Pi_A^2 = \Pi_A$, and
2. it is an orthogonal projection because $\Pi_A' = \Pi_A$

(these being the two necessary and sufficient properties for an operator to be an orthogonal projection), and, finally, the range of Π_A is the range of A as well, because, for any Ax whatever x may be, $\Pi_A Ax = A(A'A)^{-1}A'Ax = Ax$. $(I - \Pi_A)$ is then evidently the projection on the orthogonal complement of the range of A. □

Definition 1.2 Given a matrix A with independent columns, the matrix $A^\dagger = (A'A)^{-1}A'$ is called the Moore–Penrose inverse of A.

Example Suppose we have two measurements of a quantity x, the first giving $x = 9$ and the second $x = 11$. What is the "best" x in the least-squares sense? Writing the measurements in matrix form gives $b - Ax = e$ with $A = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $b = \begin{bmatrix} 9 \\ 11 \end{bmatrix}$. We find

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

$A'A = 2$ and $A^\dagger = \frac{1}{2}\begin{bmatrix} 1 & 1 \end{bmatrix}$, and hence $x_{\min} = 10$, with $e_{\min} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and the overall square-root error being $\sqrt{e'_{\min} e_{\min}} = \sqrt{2}$ as one would expect. □

This is the basic "geometric" result used in most quadratic optimization problems. Still, a number of remarks and/or refinements can be made:

1. A' is an $m \times n$ matrix, so the dimension of $A'b$ is the same as that of x. $\Pi_A := AA^\dagger = A(A'A)^{-1}A'$ is the orthogonal projection operator on the range of A, and we often write $\widehat{b} := \Pi_A b$. $\widehat{b}$ is the llse or *linear least-squares estimate of b in the range of A*.
2. Where the columns of A are not linearly independent, more work has to be done to solve the minimization problem, which typically will no longer have a unique solution. We shall treat such cases when they occur.

The QR Solution

The expression $A^\dagger = A(A'A)^{-1}A'$ is unwieldy and certainly not well suited to computations: not only is it largely inefficient, it is also computationally inaccurate – it is only mathematically satisfying because it is a closed-form solution. An adequate, first-hand, efficient and accurate solution is provided by the *upper QR algorithm* applied to A, which produces a factorization of the form

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{1.6}$$

in which $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is an $n \times n$ orthogonal matrix and R a nonsingular $m \times m$ upper-triangular matrix. The columns of Q_1 form an orthonormal basis for the range of A, while the columns of Q_2 form an orthonormal basis for the kernel of A', also known as the *co-kernel* of A. When we dispose of such a QR factorization, then we can immediately write

$$A^\dagger = R^{-1}Q'_1. \tag{1.7}$$

Upper QR is not the only possibility for such a result; we could (and will) also use a *lower QR* version of the same type of algorithm, writing $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} 0 \\ L \end{bmatrix}$, in which Q (different from the previous version!) is also an orthogonal matrix and L is a nonsingular lower triangular matrix. In this latter case, we will still have $A^\dagger = L^{-1}Q'_2$. Both R and L can be seen as "compressed" versions of the rows of A with a special (upper or lower) structure.

Remark: upper or lower QR are not the only possibilities to obtain the range basis. A numerically more refined method is the *singular value decomposition – SVD*. We refer to the linear algebra literature for a more extensive explanation.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

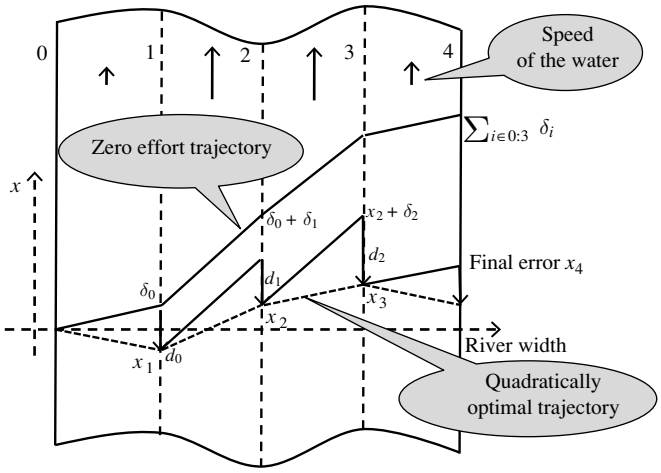

Figure 1.2 Optimal cost trajectory to row over a river with variable water speed.

Example In the previous example, we have $A = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}$, giving orthonormal bases for both the range of A and the *co-kernel of* A, which is the kernel of A'.

1.2 A Toy Example of System Optimization: Row, Row, Row Your Boat

Suppose you want to cross a river in a rowing boat. The current in the river has variable velocities depending on the distance from the shore. You can let your boat drift, and with careful handling of the rudder or the oars, you can reach the other side without any effort on your part. However, you will drift too far downstream doing this, so instead, you would row against the current with the aim of reaching a point on the other side of the river that is close to the opposite of your starting point. You would try to do the best possible job by minimizing the effort you have to exert, while trying to get close to your intended destination.

The Modeling Phase

We start out by making a simplified model of the situation. Here are the assumptions (see Fig. 1.2):

- we subdivide the river into four segments enumerated 0:3, each segment having a uniform (actually average) speed of water $v_{0:3}$. We let the current flow in the (vertical) x-direction; the model will easily generalize to more segments;
- the "natural drift" in each segment (i.e., the drift of the boat with no rowing effort but keeping the boat going to the opposite shore as well as possible) is denoted by δ_i, $i \in 0{:}3$. For example, we assume the natural drift δ_i to be proportional to

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

the current flow v_i with some constant, which we have to specify further; we may assume that the water that pushes on the boat also pushes the boat to the other side when the rudder and/or oars are correctly set – we only need to assume that we know the no-effort drift a priori;
- rowing provides for an improvement on the drift of $d_i \geq 0$ in segment i, and the rowing effort is pegged at $N_i^2 d_i^2$ for some constant N_i solely dependent on v_i, motivated in the following paragraph.

A motivation to estimate the rowing effort in segment i to be proportional to d_i^2 is that two main effects combine to increase d_i, namely the fact that more force has to be used by the rower given the local push by the river, and, second, that that force has to be exercised over a longer relative distance due to the greater drift (energy = force times distance). That makes the effort in the first instance proportional to d_i^2 (an alternative argument is based on a power expansion, the observation that $d_i = 0$ means no effort, and any deviation requires effort.) The proportionality, in turn, is dependent on the local circumstances, and hence on v_i, perhaps proportional (this assumption is not used, but is not unreasonable). We write this constant, which is positive, as a square number N_i^2, for convenience, as will appear soon.

The total cost to be minimized hence becomes

$$C_4 = \sum_{i=0}^{3} N_i^2 d_i^2 + M^2 x_4^2, \tag{1.8}$$

in which the offset at destination x_4 is penalized as $M^2 x_4^2$ for some M, which one may choose: the larger the M is, the closer to the ultimate goal the rower will end up at. All the "modeling quantities" N_i and M are assumed known (this is the big "physics work" to be done before boarding!).

The dynamic model is very simple in this case. We take the position $x_i, i = 0{:}3$ of the boat as the state at position i, and its evolution is

$$x_{i+1} = x_i + \delta_i - d_i. \tag{1.9}$$

Notice that the model is not linear: it is *affine* because of the drift term δ_i, but we shall soon see that it can be handled with linear methods just as well.

Our *optimization strategy* now consists in writing down the complete *cost model* for this situation and then performing optimization on it. The cost model has to relate the control quantities that drive the dynamic model – the d_i – to their contribution in the cost function. It will soon appear that it is best to define the components in the cost model as squares of linear quantities, namely of $y_i = N_i d_i$ for $i = 0{:}3$ and $y_4 = M x_4$ – this will make the model linear or affine. Notice that these quantities are function of either the inputs (the d_i) or the states, in this case just x_4. x_4 can be expressed in terms of the input quantities by integrating the state equations $x_4 = \delta_t - \sum_{i=0:3} d_i$, where $\delta_t := \sum_{i=0:3} \delta_i$ is the total drift (assumed to be known).

Writing this out in matrix language and using the d_i as controlling inputs, we obtain the *global cost equation*

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

$$\left[\begin{array}{c} y_{0:3} \\ \hline y_4 \end{array}\right] = \left[\begin{array}{ccc} N_0 & & \\ & \ddots & \\ & & N_3 \\ \hline -M & \cdots & -M \end{array}\right] \left[\begin{array}{c} d_0 \\ \vdots \\ d_3 \end{array}\right] + \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline M\delta_t \end{array}\right]. \tag{1.10}$$

Defining $N := \mathrm{diag}[N_i]$, $E = \mathrm{col}\left[\begin{array}{ccc} 1 & \cdots & 1 \end{array}\right]$ a column vector of 1's, and using vectors for the other quantities, the equations summarize as

$$y = \left[\begin{array}{c} N \\ -ME' \end{array}\right] d + \left[\begin{array}{c} 0 \\ M\delta_t \end{array}\right] \tag{1.11}$$

and the goal is to find the vector d that minimize $C = y'y$.

The Global Solution

As discussed in Section 1.1, the Moore–Penrose inverse produces the solution: in Eq. (1.11) and referring to the original Moore–Penrose equation $Ax - b = e_x$, d plays the role of x, y of e_x, $\left[\begin{array}{c} N \\ -ME' \end{array}\right]$ of A and $-\left[\begin{array}{c} 0 \\ M\delta_t \end{array}\right]$ of b. The Moore–Penrose inverse of the nonsingular *system matrix* $S := \left[\begin{array}{c} N \\ -ME' \end{array}\right]$ is then

$$S^{\dagger} = (N^2 + M^2EE')^{-1}\left[\begin{array}{cc} N & -ME \end{array}\right] \tag{1.12}$$

and the solution of the optimization problem is given by

$$\widehat{d}_{0:3} := (N^2 + M^2EE')^{-1}M^2E\delta_t. \tag{1.13}$$

This expression can be computed explicitly, using the inversion rule for a low rank perturbation of a nonsingular matrix (sometimes called the "Sherman–Morrison formula": suppose that some low-dimensional (rectangular) matrices A and B of same dimensions are such that $I+B'A$ is nonsingular, then $(I+AB')^{-1} = I-A(I+B'A)^{-1}B'$ – proof is by direct verification; the simplest case is when A and B are just vectors – we leave details to the interested reader). The result is

$$\widehat{d}_i = \left(\frac{\frac{1}{N_i^2}}{\frac{1}{M^2} + \sum\left(\frac{1}{N_i^2}\right)}\right)\delta_t. \tag{1.14}$$

This result is a globally computed a priori control (not a *state-dependent* control), to be computed before boarding the boat. Notice that $\widehat{d}_i = K\frac{1}{N_i^2}$ with constant $K = \frac{\delta_t}{\left(\frac{1}{M^2}+\sum\frac{1}{N_i^2}\right)}$, so and assuming all N_i equal, the optimal efforts $N_i^2\widehat{d}_i^2$ to be spent at each step are equal (which is not unreasonable altogether: you distribute the energy to be exerted evenly over the sections – a pretty generally valid "principle" in optimization theory; notice also that in the limiting case $M \to \infty$, $\sum \widehat{d}_i = \delta_t$, forcing the rower to get at the destination point exactly).

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

Dynamic Programming

The global character of this solution can easily be seen as a problem: there is no adaptivity. Many things can happen when one is plodding in the river, and it pays to figure out a recursive solution that can adapt to the perspective from a local state of affairs, reached somewhere in the middle of the river. It turns out that the global solution can be converted to a local solution, by making the controls a function of the local state. But there is another advantage to a local solution (given the validity of the model of course): at any local position *only information on the cost of the next move* is needed to determine the optimal local move. The *reduction to minimal sufficient information* is what makes the recursive computation attractive and efficient. This we derive now. It is known as *dynamic programming* or *dynamic optimization*.

Let us therefore see how to do the local recursive optimization and derive the control law at stage k, which we shall see to be just a function of the local state x_k. The principle of dynamic optimization, or Bellman principle, is based on the observation that:

once a state x_k has been reached, the cost must be optimal from that point on up to the final state, for if it were not so, there could be a lower total cost obtained by a modification of the final part of the trajectory. It follows that the optimal cost to reach the destination starting at a state x_k depends exclusively on that state x_k, that is, *all dependence on past history or controls $d_{0:k-1}$ go via the state x_k, which also determines what the optimal future controls are supposed to be*.

An important consequence of the principle is that local optimization can be done, provided one disposes of an expression for the cost of the trajectory following the current step, expressed in terms of the next state.

Concretely:
Suppose you have reached state x_k and you are ready to determine the optimal control $\widehat{d}_k$ to move to state x_{k+1} (using the state evolution equation, in this case $x_{k+1} = x_k + \delta_k - d_k$). What you need is the expression for the optimal cost of the trajectory starting at x_{k+1}, which by assumption depends only on x_{k+1}: $\widehat{C}_{k+1}(x_{k+1})$. You have to determine

$$\widehat{d}_k = \text{argmin}_{d_k} \left(N_k^2 d_k^2 + \widehat{C}_{k+1}(x_k + \delta_k - d_k) \right), \tag{1.15}$$

which optimizes the cost from x_k on. Notice that $\widehat{d}_k$ depends solely on the state x_k, since x_{k+1} depends on x_k, so we should actually write $\widehat{d}_k(x_k)$. The minimum in the expression is the optimal cost from x_k on. It is

$$\widehat{C}_k(x_k) = N_k^2 \widehat{d}_k(x_k)^2 + \widehat{C}_{k+1}(x_k + \delta_k - \widehat{d}_k(x_k)). \tag{1.16}$$

So, knowing $\widehat{C}_{k+1}$, one can determine the optimal local control $\widehat{d}_k$, and the cost information needed for the step starting at x_{k-1}. This is the minimal sufficient information needed at step k: $\widehat{C}_{k+1}$, which has to be determined as a function of x_{k+1} by a backward recursion from the end point.

The key to dynamic optimization is therefore the recursive determination of the optimal cost $\widehat{C}_k(x_k)$ to reach the destination after having reached the state x_k and this to be done with a backward recursion, for all relevant k.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

Let n be the index of the last stage for more generality (in this case, $n = 3$); then the dynamic programming equation (or Bellman equation) starts at n with $\widehat{C}_{n+1}(x_{n+1}) = M^2 x_{n+1}^2$, the cost of the deviation from the final goal, and then recurses back to $k = 0$, producing a control that is solely dependent on the x_k reached at each stage k, *provided the physical model does not change from the original assumptions*. If, after reaching x_k, one suddenly realizes that the model is not any more valid, then one would have to redo the backward recursive calculation using a new model, but of course only up to stage k, and derive a new control law, valid from stage k on.

Conclusion: for the optimization at stage k, what you need is

1. the optimal cost $\widehat{C}_{k+1}(x_{k+1})$ for each relevant x_{k+1}, and
2. the *cost model* at stage k, which is a function (still to be determined) of the effect produced by the choice of the optimal local control d_k and the optimal cost $\widehat{C}_{k+1}(x_{k+1})$ you will incur when transiting from x_{k+1} to x_{n+1}. Let us now see how this works out for our case.

The Local Cost Model

The main difficulty in dynamic programming is finding an expression for the cost $\widehat{C}_k(x_k)$, and this for each k. It turns out that, in the case of a linear or affine state cost model and a quadratic cost function, there is a simple solution to this problem. Concerning the optimal cost $\widehat{C}_k(x_k)$ for any k, we observe that

1. the cost is minimally zero when $x_k = -\delta_{t,k}$, where $\delta_{t,k} = \sum_{k=i}^{n} \delta_i$, because in that case we reach the ideal destination $x_{n+1} = 0$ with zero effort (all $d_{i:} = 0$, for $i \geq k$), and
2. the optimal cost expression in function of any x_i will likely be a quadratic expression (we shall prove this hypothesis recursively), has to be positive for all values of x_i, and has the correct minimum value zero when no cost is incurred. Assuming all this, the optimal cost then necessarily has the form

$$\widehat{C}_i(x_i) = Y_{i-1}^2 (x_i + \delta_{t,i})^2, \tag{1.17}$$

with Y_{i-1} being a new coefficient to be determined recursively (the choice for the index $i-1$ in Y_{i-1} instead of i is historical and motivated by the position of Y_i in the local cost model – see below). The proposed choice is quadratic in x_i and is zero for $x_i = -\delta_{t,i}$ as is satisfied by Eq. (1.17).

With this hypothesis, the (Bellman) *dynamic programming* equation at stage k becomes

$$\widehat{d}(x_k) = \text{argmin}_{d_k} \left[N_k^2 d_k^2 + Y_k^2 (x_k + \delta_{t,k} - d_k)^2 \right]. \tag{1.18}$$

Noticing that $x_{k+1} + \delta_{t,k+1} = x_k + \delta_k - d_k + \delta_{t,k+1} = x_k + \delta_{t,k} - d_k$, where all δ_t are the properties of the river and hence known a priori, and introducing the optimal $\widehat{d}(x_k)$ found in the local cost expression, we should find

$$\widehat{C}_k(x_k) \; !=! \; Y_{k-1}^2 (x_k + \delta_{t,k})^2 \tag{1.19}$$

for a new Y_{k-1}.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

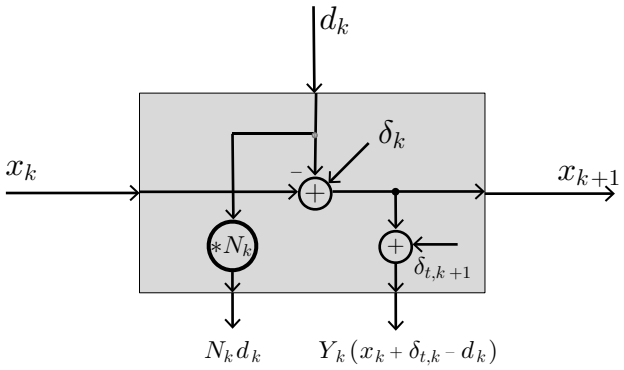

Figure 1.3 The local cost model for our rowing situation: the total cost starting with x_k is the quadratic norm of the output vector in the local model.

Equation (1.18) defines the optimization, and Eq. (1.19) says that the *expression* we guessed for $\widehat{C}_{k+1}(x_{k+1})$ is reproduced for any $\widehat{C}_k(x_k)$, thereby determining Y_{k-1} recursively (still to be proven: see the following remark).

Remark: The introduction of a recursive expression for $\widehat{C}_k$ is necessary. One cannot just add local quadratic costs, because it is not true that $(\sum(\delta_k - d_k))^2 = \sum(\delta_k - d_k)^2$! This illustrates the difficulty of general dynamic programming: one either has to guess the form of the recursive optimal cost function somehow, or else figure out some other method to find $\widehat{C}_k$ recursively in a meaningful way. In the quadratic cost case, as defined for an affine or linear model, it is easy to make the guess. One then shows correctness by recursive verification. This does not work for general norms, although even in such cases, dynamic programming remains interesting, at the cost of a more complex optimization strategy.

The local cost model at stage k, using the proposed cost expression, is shown in Figure 1.3. This cost model is affine, with steering (input) vector d_k and the cost vector expressed in terms of the present state x_k, the control d_k, and the known quantities, is then

$$\begin{bmatrix} Y_k(x_{k+1} + \delta_{t,k+1}) \\ N_k d_k \end{bmatrix} = \begin{bmatrix} Y_k(x_k + \delta_{t,k}) \\ 0 \end{bmatrix} + \begin{bmatrix} -Y_k \\ N_k \end{bmatrix} d_k. \tag{1.20}$$

Using the Moore–Penrose inverse, it follows that

$$\widehat{d}_k = -\begin{bmatrix} -Y_k \\ N_k \end{bmatrix}^{\dagger} \begin{bmatrix} Y_k(x_k + \delta_{t,k}) \\ 0 \end{bmatrix} = \frac{Y_k^2}{N_k^2 + Y_k^2}(x_k + \delta_{t,k}). \tag{1.21}$$

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

This is an affine control, partly proportional to x_k, with a constant *plus* an a priori known driving term $\delta_{t,k}$. For the cost, we find, after a small calculation,

$$\widehat{C}_k(x_k) = \frac{N_k^2 Y_k^2}{N_k^2 + Y_k^2}(x_k + \delta_{t,k})^2, \tag{1.22}$$

thereby proving the recursive hypothesis, with $Y_{k-1} = \frac{N_k Y_k}{\sqrt{N_k^2+Y_k^2}}$. Of course, the cost at n (which initializes the recursion) is simply $Y_n^2 x_{n+1}^2 = M^2 x_{n+1}^2$, and it cannot be optimized, since the other shore has been reached.

The (Lower) QR Way

A slightly different viewpoint, which will prove very effective in larger problems, works with orthogonalization and is, in the present case, particularly simple. For the cost vector (let us call it y_k, with the quadratic cost $y_k' y_k$), we have

$$y_k := \begin{bmatrix} Y_k(x_{k+1} + \delta_{t,k+1}) \\ N_k d_k \end{bmatrix} = \begin{bmatrix} Y_k & -Y_k \\ 0 & N_k \end{bmatrix} \begin{bmatrix} x_k + \delta_{t,k} \\ d_k \end{bmatrix}, \tag{1.23}$$

where d_k is the driving term. A lower QR factorization on the "system matrix" produces

$$\begin{bmatrix} Y_k & -Y_k \\ 0 & N_k \end{bmatrix} = Q_k \begin{bmatrix} \frac{Y_k N_k}{\sqrt{Y_k^2+N_k^2}} & 0 \\ \frac{-Y_k^2}{\sqrt{Y_k^2+N_k^2}} & \sqrt{Y_k^2 + N_k^2} \end{bmatrix}, \tag{1.24}$$

with $Q_k = \frac{1}{\sqrt{Y_k^2+N_k^2}} \begin{bmatrix} N_k & -Y_k \\ Y_k & N_k \end{bmatrix}$ being a single rotation in this case. (The lower QR factorization first starts with compressing the last column downward, in this case rotating $\begin{bmatrix} -Y_k \\ N_k \end{bmatrix}$ to $\begin{bmatrix} 0 \\ \sqrt{Y_k^2 + N_k^2} \end{bmatrix}$, and next moves to what remains of the next to last column, which in this case does not need any further compression as it is already a single scalar.)

Surprisingly perhaps, *the R-factor contains the result directly*! To see this, let $c_k := Q_k' y_k$ be the rotated cost vector over Q_k'; then $c_k' c_k = y_k' y_k$ (i.e., the quadratic cost is preserved after rotation), and

$$\begin{bmatrix} c_{k,1} \\ c_{k,2} \end{bmatrix} = \begin{bmatrix} \frac{Y_k N_k}{\sqrt{Y_k^2+N_k^2}} & 0 \\ \frac{-Y_k^2}{\sqrt{Y_k^2+N_k^2}} & \sqrt{Y_k^2 + N_k^2} \end{bmatrix} \begin{bmatrix} x_k + \delta_{t,k} \\ d_k \end{bmatrix}, \tag{1.25}$$

while the quadratic cost is now written as $c_{k,1}^2 + c_{k,2}^2$. Remembering that x_k is given at the beginning of each recursion, we see that $c_{k,1}$ is fixed, and the minimal cost is obtained by making $c_{k,2} = 0$, which as an optimal driving input gives

$$\widehat{d}_k = \frac{Y_k^2}{Y_k^2 + N_k^2}(x_k + \delta_{t,k}), \tag{1.26}$$

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

and the corresponding cost is $\widehat{C}_k(x_k) = c^2_{k,1} = \frac{Y_k^2 N_k^2}{Y_k^2+N_k^2}(x_k + \delta_{t,k})^2$ as announced and as derived before.

Remarks

- The solution presented in Eq. (1.26) is a *feedback law*: it gives the required control in function of the state reached; see Fig. 1.4.
- The simplicity of the recursive solution should be obvious, but it requires some additional modeling effort to obtain it. It also has the great advantage that the optimization criteria may be modified adaptively as one proceeds (e.g., when one gets better estimates of the properties of the river, the boat, and/or of future costs). This is an issue we shall not address here but which may come up when we discuss optimization problems in later chapters.
- If one has to reach the exact destination ($x_{n+1} = 0$), then one may let M tend to infinity, with some care. This will not lead to much change in the derivation, except at the last stretch. It is a good exercise to perform!
- From the control formula, it is clear that the effort to be made at any stage k has N_k^2 in the denominator, meaning that the stronger the current, the less one should row against it. The wisdom to profit, mostly from the least resistance or the low hanging fruit, turns out to be a pretty general result. The general physical principle of "least action" has some of this flavor as well.
- It should be clear that the Moore–Penrose inverse method only works on linear or affine models and costs of quadratic type. With a different overall cost function or nonlinear model, the whole procedure becomes considerably more complex, but the principle of dynamic programming will still apply, at the cost of added numerical complexity.
- Our example was restricted to $n = 3$, but the treatment is sufficiently general (provided the model is valid, of course) and could also be utilized for a continuous-time

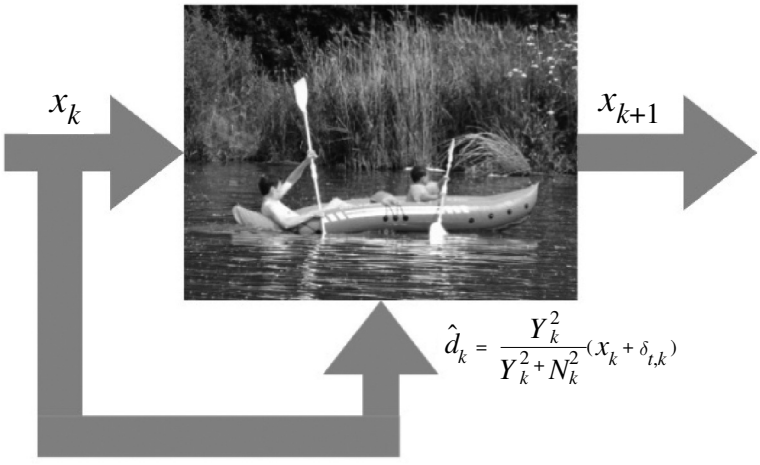

Figure 1.4 The optimal control feedback law.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

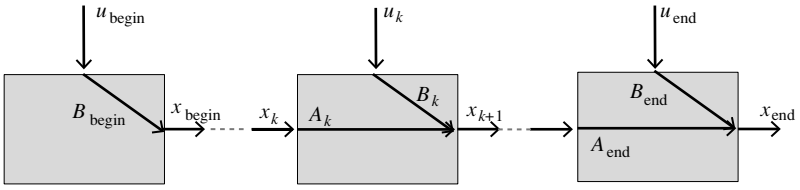

Figure 1.5 Signal flow model of the state evolution considered in the linear tracking model. At index k, $x_{k+1} = A_k x_k + B_k u_k$.

situation after discretization or, conversely, to derive the continuous-time treatment from the discretized, which is what is often done in the literature.

- It is easy to see that the recursive solution gives the same result as the global one. A good exercise!

1.3 The General Linear Model for Optimal Quadratic Control

Given the insights we have developed in our toy example, it is not difficult to derive the matrix algebra for the classical problem of *optimal quadratic state control* of a *linear dynamical system*, as originally proposed by Bellman [13], usually done for the continuous-time case, but done here for the discrete-time case. The goal is to show (1) how the conversion from a physical system environment to matrix algebra is done, and (2) how an upper or lower QR factorization greatly simplifies the procedure, thanks to orthogonalization. It also provides a good example of an important general method we shall develop later: inner–outer factorization.

We assume that the dynamics of our discrete-time system are represented by a real *state vector* x_k of dimension η_k, which is a function of a time index k (an integer) and characterizes the state of the system at the time instant k. Let the system be driven by an input u_k at index k, where u_k is a vector of dimension m_k, and let us assume (inspired by the example) that the evolution of the state of the system from the time point k to the time point $k + 1$ is given by the linear (time-variant) state equations

$$x_{k+1} = A_k x_k + B_k u_k, \tag{1.27}$$

in which A_k is an $\eta_{k+1} \times \eta_k$ matrix and B_k is an $\eta_{k+1} \times m_k$ matrix. A simple *signal flow diagram* as shown in Fig. 1.5 is often used to represent such a model. (In Appendix, we present the "Functional Data Model" we typically use to represent computations graphically. It allows for an easy transformation of numerical computations into a hardware architecture or a computer program.)

Let us assume for illustrative purposes that our system starts at index 0 with a given initial state x_0 and that we wish to control the evolution of our system in the interval $[0, n+1]$, where n is the index of the final stage, so as to minimize a positive quadratic cost function keeping states and inputs small, by putting quadratically increasing costs on their values.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

For simplicity and in order to represent a quadratic cost adequately, we write the cost of a state x_k as $x_k' M_k' M_k x_k$, in which M_k is a matrix of appropriate dimensions and the prime indicates real or complex conjugation, making x_k' a row vector. (Often the cost is defined by a strictly positive definite matrix C_k, and one may take $M_k = C_k^{1/2}$ or just a Cholesky factor, but C_k may be nonstrictly positive definite and M_k may be a rectangular factor accordingly.) Similarly, we write the cost of an input u_k as $u_k' N_k' N_k u_k$, and we assume N_k to be a square, nonsingular matrix (so that arbitrarily large inputs will not be possible).

Dynamic Programming

The problem of optimal quadratic control in the given set up is solved by *dynamic programming*. The basic idea of dynamic programming is: *once the system reaches the state x_k at the time index k, the trajectory has to be optimal from there on*; for, if that were not the case, there would be a better overall trajectory just by replacing the segment from the current position k to the end with a less costly path. This means, in particular, that all optimal inputs and costs, when started from x_k at the time index k, are only dependent on x_k and not directly on previous states; previous states will influence x_k, but can only influence later quantities via x_k, given the model, of course. To put it differently, all the future inputs have to be chosen so as to optimize the trajectory from the index k on and hence are only dependent on x_k and the model from that point in time on. Let us now prove the following recursive hypothesis for the quadratic cost case and linear system model:

> *The total optimal cost starting from any x_k in the remaining interval $[k, n+1]$ is given by a quadratic form $\widehat{C}_k(x_k) := x_k' Y_{k-1}' Y_{k-1} x_k$, in which Y_{k-1} is a (to be computed recursively) $\eta_k \times \eta_k$ matrix.*

The recursive hypothesis will be verified if (1) it is valid at the end point $k = n+1$ and (2) when valid for $k+1$ it is valid for k:

1. The first statement is true because at the end point $n+1$ the cost is $x_{n+1}' M_{n+1}' M_{n+1} x_{n+1}$, so $Y_n = M_{n+1}$.
2. The second statement is to be derived now.

Key to the derivation is the determination of the *cost model*: put the "square roots" $Y_k x_{k+1}$, $M_k x_k$ and $N_k u_k$ of the cost terms as outputs in the cost model, so that the cost equals their total square norm and the model itself is linear. Next, assume recursively that the optimal cost, from $k+1$ on, and for any x_{k+1}, is given by $x_{k+1}' Y_k' Y_k x_{k+1}$, the cost model at k (shown in Fig. 1.6) gives, after the multiplication of the first block row with Y_k,

$$y_k = \left[\begin{array}{c} Y_k x_{k+1} \\ \hline M_k x_k \\ N_k u_k \end{array}\right] = \left[\begin{array}{c} Y_k A_k \\ \hline M_k \\ 0 \end{array}\right] x_k + \left[\begin{array}{c} Y_k B_k \\ \hline 0 \\ N_k \end{array}\right] u_k = \left[\begin{array}{c|c} Y_k A_k & Y_k B_k \\ \hline M_k & 0 \\ 0 & N_k \end{array}\right] \left[\begin{array}{c} x_k \\ \hline u_k \end{array}\right], \quad (1.28)$$

and the optimization problem specializes to the following: *find the u_k that minimizes the cost $y_k' y_k = x_{k+1}' Y_k' Y_k x_{k+1} + x_k' M_k' M_k x_k + u_k' N_k' N_k u_k$, for given x_k.*

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

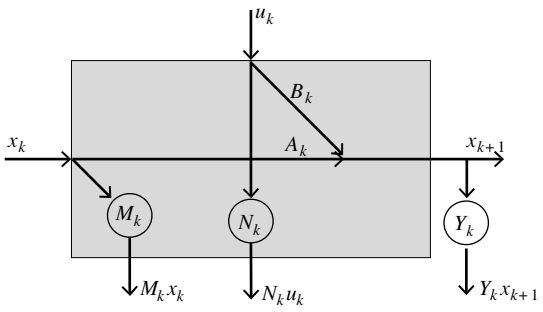

Figure 1.6 The full local cost model for optimal quadratic control, including cost outputs.

Notice: In this phase of the recursion, x_k is the only remaining "variable," whereas the others, namely u_k and x_{k+1}, will be "optimized out" as a function of x_k.

(Lower) QR Orthogonalization

The propagation of the cost function is now easily found using orthogonalization of the system matrix. Following the tradition in the literature, we use a lower QR factorization of the "system matrix" in Eq. (1.28), to produce the block decomposition

$$\left[\begin{array}{c|c} Y_k A_k & Y_k B_k \\ \hline M_k & 0 \\ 0 & N_k \end{array}\right] = Q_k \left[\begin{array}{c|c} 0 & 0 \\ \hline Y_{k-1} & 0 \\ C_{o,k} & D_{o,k} \end{array}\right], \tag{1.29}$$

in which

1. Q_k is an orthogonal (or, in the complex case, unitary) matrix that produces the block staircase in the right factor;
2. $D_{o,k}$ is square invertible, thanks to the assumption that N_k is square invertible (it is not hard to show this);
3. Y_{k-1} is, by construction, in the "row echelon form": it has independent rows and hence is right invertible (flat), which makes it of minimal dimension;
4. all $Q_k, Y_{k-1}, C_{o,k}$ and $D_{o,k}$ are new matrices computed from the entries in the system matrix and whose meaning will soon be clear.

The orthogonal transformation matrix Q_k' transforms the cost vector y_k to $c_k = Q_k' y_k$ in three orthogonal components, each with its own significance:

$$\begin{bmatrix} c_{k,1} \\ c_{k,2} \\ c_{k,3} \end{bmatrix} := Q_k' \begin{bmatrix} Y_k x_{k+1} \\ M_k x_k \\ N_k u_k \end{bmatrix} = \left[\begin{array}{c|c} 0 & 0 \\ \hline Y_{k-1} & 0 \\ C_{o,k} & D_{o,k} \end{array}\right] \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \begin{bmatrix} 0 \\ \hline Y_{k-1} x_k \\ C_{o,k} x_k + D_{o,k} u_k \end{bmatrix} \tag{1.30}$$

valid for any input u_k. It immediately follows from $Q_k Q_k' = I$ that the quadratic cost $y_k' y_k = c_k' c_k$ generated by x_k and u_k can now be expressed as

$$y_k' y_k = \|Y_{k-1} x_k\|^2 + \|(C_{o,k} x_k + D_{o,k} u_k)\|^2, \tag{1.31}$$

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

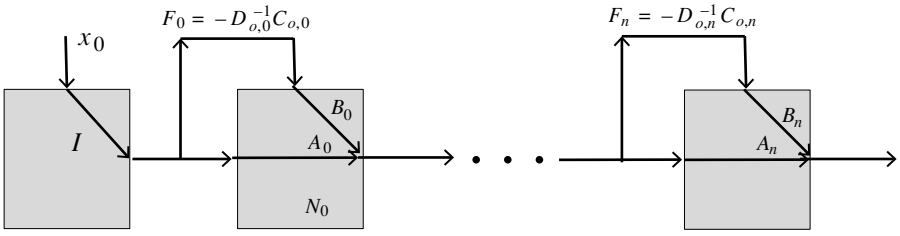

Figure 1.7 The optimal dynamic (i.e., state-dependent) Bellman controller.

in which $D_{o,k}$ is square invertible. It follows that, given x_k, the cost is minimized by choosing the optimal

$$\widehat{u}_k := -D_{o,k}^{-1} C_{o,k} x_k, \tag{1.32}$$

whereupon the minimal cost follows:

$$\widehat{C}_k(x_k) = \|Y_{k-1} x_k\|^2 = x_k' Y_{k-1}' Y_{k-1} x_k, \tag{1.33}$$

which had to be shown. □

The computation of new matrices from a given matrix, as is done in Eq. (1.29), has been termed *array processing* in the literature, because it is a computation on a given matrix that generates new matrices as a result. It is sometimes called a "square-root method" because it works directly on the data given rather than on the quadratic form that represents their cost.

In conclusion, we can state that a simple QR factorization gives the complete recursive solution of the quadratic tracking problem, whereby the orthogonal Q-factor filters the input data to produce both the optimal control and the new global cost function, all in function of the actual state x_k. Formula (1.32) shows that the optimal control consists of a simple state feedback. This realization is shown in Fig. 1.7.

1.4 A Question to Be Researched

Thinking about optimizing the behavior of a system evolving in time, under what conditions would dynamic programming be possible? Or, expressed negatively, when is dynamic programming definitely not possible?

1.5 Notes

1. One of the original motivations for considering a quadratic optimization model was the Apollo mission: How to get a rocket to the moon with minimal expenditure of fuel? It takes, of course, some work to reduce the Apollo mission problem to the simple model presented by discrete-time linear model in this book. One must first find a potentially optimal trajectory that respects gravity laws and that requires a minimal nominal amount of fuel to reach the goal within a domain of

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

feasibility defined by various limits in time and fuel needed. Once settled on such an optimal trajectory, the control problem is to keep the rocket close to the optimal trajectory with minimal expenditure of fuel, even though various inaccuracies may have occurred producing (stochastic) deviations. This is achieved by controlling the *deviation* of the state from the desired optimal and using the control to bring the rocket closer to the nominal optimal trajectory without spending too much fuel. When sufficiently small, the deviation of the state will satisfy a linear differential model derived from the optimal trajectory. After discretization, a model of the type given above is obtained. We leave the study of discretizations to the appropriate literature.

2. Dynamic programming is only one example of the use of system theory. There are many more examples, several of which will be discussed in the following chapters, and many more have to be relegated to further treatments. State estimation (Kalman filtering), control theory, system modeling and model reduction, approximation and interpolation of matrices, efficient computations with some types of structured matrix, and data filter design all rely on system-theoretic concepts. The value of dynamic programming as a prime example is the intimate connection it exhibits between the dynamic model and the recursion that leads to the optimal control. Further instructive examples of optimization problems on systems can be found in the book of Luenberger [50]. A classical textbook on linear optimization is due to Anderson and Moore [3].
3. There is a host of methods one can use to orthogonalize a set of commensurable vectors, to wit:

    1. global multidimensional rotations (to be preferred) or Householder transformations;
    2. Gram–Schmidt orthogonalization;
    3. bi-orthogonalization using hyperbolic transformations [18, 22].

    These methods are discussed in further chapters, when they come up, and in many textbooks on linear algebra.
4. The functional representation and data model used in this chapter, and described in the Appendix, was proposed and analyzed in [6]. It allows for an unbiased transformation of mathematical operations to a computer architecture at the functional and data transfer levels. It hides organizational details such as the conditional sequencing of functions, the partitioning, and storage and transfer of data in such a way that this information can easily be generated from the data model when architectural decisions like the localization of data in memories and the assignment of functions to processors has been made (such design phases will not concern us in this book). In particular, the model easily accommodates hierarchical representations and parallel processing. In this book, we shall only worry about practical implementation aspects as far as numerical properties (numerical accuracy and complexity) are concerned.
5. It is not easy to find a simple direct example for a deterministic quadratic optimization problem on a linear model. Most such problems are of the type "tracking a

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press

nonlinear trajectory to counteract stochastic disturbances." In most practical examples of tracking with a linear model, the model has an extra disturbing noise term, which has to be controlled. Using the time-variant linear model found by differentiating a nonlinear trajectory, adding noise terms and using average quantities, one can, without too much effort, extend the model used in this chapter to handle the more general case.

6. Quite a few famous algorithms that go by specific names are, in fact, examples of dynamic programming. Take for example the famous Viterbi algorithm to decode a bitstream that has been coded with a convolutional code. This is an excellent example of the use of dynamic programming in a p-adic number system.

https://doi.org/10.1017/9781009455640.003 Published online by Cambridge University Press