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

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

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

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* $\operatorname{argmin}_x \|b - Ax\|_2$, *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

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

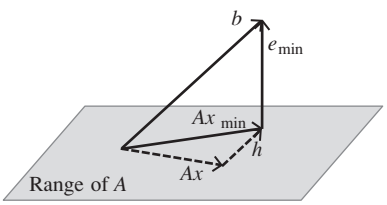

Figure 1.1 Best linear quadratic approximation.

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

$A'A = 2$ 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.

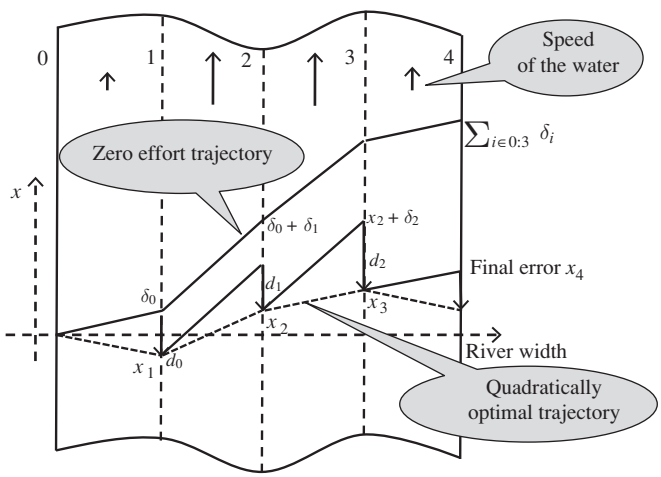

Figure 1.2 Optimal cost trajectory to row over a river with variable water speed.

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

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*

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

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

$$y = \begin{bmatrix} N \\ -ME' \end{bmatrix} d + \begin{bmatrix} 0 \\ M\delta_t \end{bmatrix} \tag{1.11}$$

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, $\begin{bmatrix} N \\ -ME' \end{bmatrix}$ of A and $-\begin{bmatrix} 0 \\ M\delta_t \end{bmatrix}$ of b. The Moore–Penrose inverse of the nonsingular *system matrix* $S := \begin{bmatrix} N \\ -ME' \end{bmatrix}$ is then

$$S^{\dagger} = (N^2 + M^2EE')^{-1} \begin{bmatrix} N & -ME \end{bmatrix} \tag{1.12}$$

and the solution of the optimization problem is given by

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

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).

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.

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}.

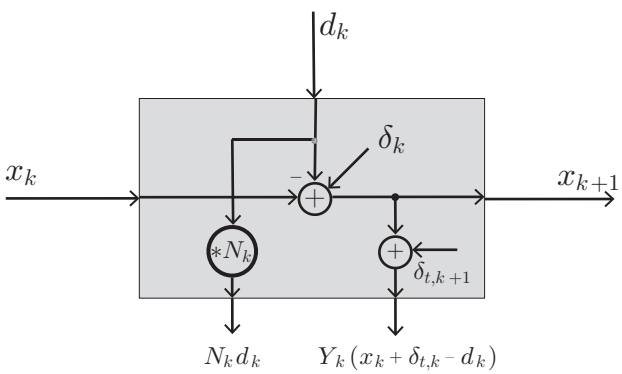

Figure 1.3 The local cost model for our rowing situation: the total cost starting with x_k is the quadratic norm of the output vector in the local model.

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

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