Approximation in Value Space

Contents

   2.1.1. Methods for Computing Approximations in . . . . . p. 6
         Value Space . . . . . . . . . . . . . . . . . . . . . . . . . . . p. 6
   2.1.2. Off-Line and On-Line Methods . . . . . . . . . . . p. 7
   2.1.3. Model-Based Simplification of the Lookahead . . . . p. 8
       Minimization . . . . . . . . . . . . . . . . . . . . . . . . . . . p. 8
   2.1.4. Model-Free Q-Factor Approximation in Value Space p. 9
   2.1.5. Approximation in Policy Space on Top of . . . . . . p. 12
       Approximation in Value Space . . . . . . . . . . . . . . p. 12
   2.1.6. When is Approximation in Value Space Effective? . p. 13

2.2. Multistep Lookahead . . . . . . . . . . . . . . . . p. 14
   2.2.1. Multistep Lookahead and Rolling Horizon . . . . . . p. 16
   2.2.2. Multistep Lookahead and Deterministic Problems . . p. 17

2.3. Problem Approximation . . . . . . . . . . . . . . . . p. 19
   2.3.1. Enforced Decomposition . . . . . . . . . . . . . . . p. 19
   2.3.2. Probabilistic Approximation - Certainty . . . . . . . p. 26
       Equivalent Control . . . . . . . . . . . . . . . . . . . . p. 26

2.4. Rollout and Model Predictive Control . . . . . . . . . p. 32
   2.4.1. Rollout for Deterministic Problems . . . . . . . . p. 33
   2.4.2. Stochastic Rollout and Monte Carlo Tree Search . . p. 43
   2.4.3. Model Predictive Control . . . . . . . . . . . . . . p. 51

2.5. Notes and Sources . . . . . . . . . . . . . . . . . . . p. 60
As we noted in Chapter 1, the exact solution of optimal control problems by DP is often impossible. To a great extent, the reason lies in what Bellman has called the “curse of dimensionality.” This refers to a rapid increase of the required computation and memory storage as the problem’s size increases. Moreover, there are many circumstances where the structure of the given problem is known well in advance, but some of the problem data, such as various system parameters, may be unknown until shortly before control is needed, thus seriously constraining the amount of time available for the DP computation. These difficulties motivate suboptimal control schemes that strike a reasonable balance between convenient implementation and adequate performance.

**Approximation in Value Space**

There are two general approaches for DP-based suboptimal control. The first is approximation in value space, where we approximate the optimal cost-to-go functions $J_k^*$ with some other functions $\tilde{J}_k$. We then replace $J_k^*$ in the DP equation with $\tilde{J}_k$. In particular, at state $x_k$, we use the control obtained from the minimization

$$\hat{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E\left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right\}. \quad (2.1)$$

This defines a suboptimal policy $\{\hat{\mu}_0, \ldots, \hat{\mu}_{N-1}\}$. There are several possibilities for selecting or computing the functions $\tilde{J}_k$, which are discussed in this chapter, and also in subsequent chapters.

Note that the expected value expression appearing in the right-hand side of Eq. (2.1) can be viewed as an approximate Q-factor

$$\tilde{Q}_k(x_k, u_k) = E\left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right\},$$

and the minimization in Eq. (2.1) can be written as

$$\hat{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k),$$

(cf. Section 1.2). This also suggests a variant of approximation in value space, which is based on using Q-factor approximations that may be obtained directly, i.e., without the intermediate step of obtaining the cost function approximations $\tilde{J}_k$. In what follows in this chapter, we will focus on cost function approximation, but we will occasionally digress to discuss direct Q-factor approximation.

Approximation in value space based on the minimization (2.1) is commonly referred to as one-step lookahead, because the future costs are approximated by $\tilde{J}_{k+1}$, after a single step. An important variation is multistep lookahead, whereby we minimize over $\ell > 1$ stages with the future costs
approximated by a function $\tilde{J}_{k+\ell}$. For example, in two-step lookahead the function $\tilde{J}_{k+1}$ is given by

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{u_{k+1} \in U_{k+1}(x_{k+1})} E\left\{ g_{k+1}(x_{k+1}, u_{k+1}, w_{k+1}) + \tilde{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1}, w_{k+1})) \right\},$$

where $\tilde{J}_{k+2}$ is some approximation of the optimal cost-to-go function $J^*_{k+2}$.

Actually, as the preceding two-step lookahead case illustrates, one may view $\ell$-step lookahead as the special case of one-step lookahead where the lookahead function is the optimal cost function of an $(\ell - 1)$-stage DP problem with a terminal cost at the end of the $\ell - 1$ stages, which is equal to $\tilde{J}_{k+\ell}$. However, it is often important to discuss $\ell$-step lookahead separately, in order to address special implementation issues that do not arise in the context of one-step lookahead.

In our initial discussion of approximation in value space of Section 2.1, we will focus on one-step lookahead. There are straightforward extensions of the main ideas to the multistep context, which we will discuss in Section 2.2.

**Approximation in Policy Space**

The major alternative to approximation in value space is approximation in policy space, whereby we select the policy by using optimization over a suitably restricted class of policies, usually a parametric family of some form. An important advantage of this approach is that the computation of controls during on-line operation of the system is often much easier compared with the minimization (2.1). However, this advantage can also be gained by combining approximation in value space with policy approximation in a two-stage scheme:

(a) Obtain the approximately optimal cost-to-go functions $\tilde{J}_k$, thereby defining a corresponding suboptimal policy $\tilde{\mu}_k$, $k = 0, \ldots, N - 1$, via Eq. (2.1).

(b) Approximate $\tilde{\mu}_k$, $k = 0, \ldots, N - 1$, using a training set consisting of a large number $q$ of sample pairs $(x_k^s, u_k^s)$, $s = 1, \ldots, q$, where $u_k^s = \tilde{\mu}_k(x_k^s)$. For example, introduce a parametric family of policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N - 1$, of some form, where $r_k$ is a parameter, such as a family represented by a neural net, and then estimate the parameters $r_k$ using some form of regression, e.g.,

$$r_k \in \arg \min_r \sum_{s=1}^q \| u_k^s - \mu_k(x_k^s, r) \|^2.$$ 

In this chapter we discuss primarily approximation in value space, although some of the ideas are also relevant to approximation in policy space.
We focus on finite horizon problems, postponing the discussion of infinite horizon problems for Chapter 4 and later. However, the finite horizon ideas are relevant to the infinite horizon setting, and many of the methods of the present chapter and Chapter 3 also apply with small modifications to infinite horizon problems.

**Model-Based Versus Model-Free Implementation**

An important attribute of the solution process is whether an analytical model or Monte Carlo simulation is used to compute expected values such as those arising in one-step and multistep lookahead expressions. We distinguish between two cases:

(a) In the *model-based* case, we assume that the conditional probability distribution of \( w_k \), given \( (x_k, u_k) \), is available in essentially closed form. By this we mean that the value of \( p_k(w_k | x_k, u_k) \) is available for any triplet \( (x_k, u_k, w_k) \). Moreover, the functions \( g_k \) and \( f_k \) are also available. In a model-based solution process, expected values, such as the one in the lookahead expression of Eq. (2.1), are obtained with algebraic calculations as opposed to Monte Carlo simulation.

(b) In the *model-free* case, the calculation of the expected value in the expression of Eq. (2.1), and other related expressions, is done with Monte Carlo simulation. There may be two reasons for this.

1. A mathematical model of the probabilities \( p_k(w_k | x_k, u_k) \) is not available, but instead there is a computer program/simulator that for any given state \( x_k \) and control \( u_k \), simulates sample probabilistic transitions to a successor state \( x_{k+1} \), and generates the corresponding transition costs. In this case the expected value can be computed approximately by Monte Carlo simulation.†

2. The probabilities \( p_k(w_k | x_k, u_k) \) are available for any triplet \( (x_k, u_k, w_k) \), but for reasons of computational efficiency we prefer to compute the expected value in the expression (2.1) by using sampling and Monte Carlo simulation. Thus the expected value is computed as in the case where there is no mathematical model, but instead there is a computer simulator.‡

† The term Monte Carlo simulation mostly refers to the use of a software simulator. However, a hardware or a combined hardware/software simulator may also be used in some practical situations to generate samples that are used for Monte Carlo averaging.

‡ The idea of using Monte Carlo simulation to compute complicated integrals or even sums of many numbers is used widely in various types of numerical computations. It encompasses efficient Monte Carlo techniques known as *Monte Carlo*...
Sec. 2.1 General Issues of Approximation in Value Space

There are two major issues in a value space approximation scheme, and each of the two can be considered separately from the other:

1. **Obtaining $\hat{J}_k$**, i.e., the method to compute the lookahead functions $\hat{J}_k$ that are involved in the lookahead minimization (2.1). There are integration and importance sampling; see textbooks such as [Liu01], [AsG10], [RoC10] for detailed developments.

In summary, the use of sampling and Monte Carlo simulation is the defining attribute for a method to be model-based or model-free in the terminology of this book. In particular, a model-free method is one that can be applied both when there is a mathematical model and when there is not.

Note that for deterministic problems there is no expected value to compute, so these problems typically come under the model-based category, even if values of the functions $g_k$ and $f_k$ become available through complicated computer calculations. Still however, Monte Carlo simulation may enter the solution process of a deterministic problem for a variety of reasons. For example the games of chess and Go are perfectly deterministic, but the AlphaGo and AlphaZero programs (Silver et al. [SHM16], [SHS17]) use randomized policies and rely heavily on Monte Carlo tree search techniques, which will be discussed later in Section 2.4. The same is true for some policy gradient methods, which will be discussed in Chapter 4.

Figure 2.1.1 Schematic illustration of various options for approximation in value space with one-step lookahead. The lookahead function values $\hat{J}_{k+1}(x_{k+1})$ approximate the optimal cost-to-go values $J_{k+1}^*(x_{k+1})$, and can be computed by a variety of methods. There may be additional approximations in the minimization over $u_k$ and in the computation of the expected value over $w_k$; see Section 2.1.1.

In summary, the use of sampling and Monte Carlo simulation is the defining attribute for a method to be model-based or model-free in the terminology of this book. In particular, a model-free method is one that can be applied both when there is a mathematical model and when there is not.
quite a few approaches here (see Fig. 2.1.1). Several of them are discussed in this chapter, and more will be discussed in subsequent chapters.

(2) Control selection, i.e., the method to perform the minimization (2.1) and implement the suboptimal policy $\tilde{\mu}_k$. Again there are several exact and approximate methods for control selection, some of which will be discussed in this chapter (see Fig. 2.1.1).

In this section we will provide a high level discussion of these issues.

2.1.1 Methods for Computing Approximations in Value Space

Regarding the computation of $\tilde{J}_k$, we will consider four types of methods:

(a) Problem approximation (Section 2.3): Here the functions $\tilde{J}_k$ in Eq. (2.1) are obtained as the optimal or nearly optimal cost functions of a simplified optimization problem, which is more convenient for computation. Simplifications may include, exploiting decomposable structure, ignoring various types of uncertainties, and reducing the size of the state space. The latter form of simplification is known as aggregation, and is discussed separately in Chapter 5.

(b) On-line approximate optimization (Section 2.4): These methods often involve the use of a suboptimal policy or heuristic, which is applied on-line when needed to approximate the true optimal cost-to-go values. The suboptimal policy may be obtained by any other method, e.g., problem approximation. Rollout algorithms and model predictive control are prime examples of these methods.

(c) Parametric cost approximation (Chapter 3): Here the functions $\tilde{J}_k$ in Eq. (2.1) are obtained from a given parametric class of functions $\tilde{J}_k(x_k, r_k)$, where $r_k$ is a parameter vector, selected by a suitable algorithm. The parametric class is typically obtained by using prominent characteristics of $x_k$ called features, which can be obtained either through insight into the problem at hand, or by using training data and some form of neural network.

(d) Aggregation (Chapter 5): This is a special but rather sophisticated form of problem approximation. A simple example is to select a set of representative states for each stage, restrict the DP algorithm to these states only, and approximate the costs-to-go of other states by interpolation between the optimal costs-to-go of the representative states. In another example of aggregation, the state space is divided into subsets, and each subset is viewed as a state of an “aggregate DP problem.” The functions $\tilde{J}_k$ are then derived from the optimal cost functions of the aggregate problem. The state space partition can be arbitrary, but is often determined by using features (states with “similar” features are grouped together). Moreover, aggregation can be
combined in complementary fashion with the methods (a)-(c) above, and can use as a starting point an approximate cost-to-go function produced by any one of these methods; e.g., apply a parametric approximation method and enhance the resulting cost function through local corrections obtained by aggregation.

Additional variations of the above methods are obtained when used in combination with approximate minimization over $u_k$ in Eq. (2.1), and also when the expected value over $w_k$ is computed approximately via a certainty equivalence approximation (cf. Section 2.3.2) or adaptive simulation and Monte Carlo tree search (Sections 2.1.2 and 2.4.2).

2.1.2 Off-Line and On-Line Methods

In approximation in value space an important consideration is whether the cost-to-go functions $\tilde{J}_{k+1}$ and the suboptimal control functions $\tilde{\mu}_k$, $k = 0, \ldots, N - 1$, of Eq. (2.1) are computed off-line (i.e., before the control process begins, and for all $x_k$ and $k$), or on-line (i.e., after the control process begins, when needed, and for just the states $x_k$ to be encountered).

Usually, for challenging problems, the controls $\tilde{\mu}_k(x_k)$ are computed on-line, since their storage may be difficult for a large state space. However, the on-line or off-line computation of $\tilde{J}_{k+1}$ is an important design choice. We thus distinguish between:

(i) **Off-line methods**, where the entire function $\tilde{J}_{k+1}$ in Eq. (2.1) is computed for every $k$, before the control process begins. The values $\tilde{J}_{k+1}(x_{k+1})$ are either stored in memory or can be obtained with a simple and fast computation, as needed in order to compute controls by one-step lookahead. The advantage of this is that most of the computation is done off-line, before the first control is applied at time 0. Once the control process starts, no extra computation is needed to obtain $\tilde{J}_{k+1}(x_{k+1})$ for implementing the corresponding suboptimal policy.

(ii) **On-line methods**, where most of the computation is performed just after the current state $x_k$ becomes known, the values $\tilde{J}_k(x_{k+1})$ are computed only at the relevant next states $x_{k+1}$, and are used to compute the control to be applied via Eq. (2.1). These methods require the computation of a control only for the $N$ states actually encountered in the control process. In contrast with the off-line approximation methods, these methods are well-suited for on-line replanning, whereby the problem data may change over time.

Examples of typically off-line schemes are neural network and other parametric approximations, as well as aggregation. Examples of typically online schemes are rollout and model predictive control. Schemes based on problem approximation may be either on-line or off-line depending on other
problem-related factors. Of course there are also problem-dependent hybrid methods, where significant computation is done off-line to expedite the on-line computation of needed values of $\tilde{J}_{k+1}$.

### 2.1.3 Model-Based Simplification of the Lookahead Minimization

We will now consider ways to facilitate the calculation of the suboptimal control $\tilde{\mu}_k(x_k)$ at state $x_k$ via the minimization of the one-step lookahead expression

$$E\left\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, \tilde{w}_k))\right\}, \tag{2.2}$$

once the cost-to-go approximating functions $\tilde{J}_{k+1}$ have been selected. In this section, we will assume that we have a mathematical model, i.e., that the functions $g_k$ and $f_k$ are available in essentially closed form, and that the conditional probability distribution of $w_k$, given $(x_k, u_k)$, is also available. In particular, Monte Carlo simulation is not used in Eq. (2.2). We will address the model-free case in the next section.

Important issues here are the computation of the expected value (if the problem is stochastic) and the minimization over $u_k \in U_k(x_k)$ in Eq. (2.2). Both of these operations may involve substantial work, which is of particular concern when the minimization is to be performed on-line.

One possibility to eliminate the expected value from the expression (2.2) is (assumed) certainty equivalence. Here we choose a typical value $\tilde{w}_k$ of $w_k$, and use the control $\tilde{\mu}_k(x_k)$ that solves the deterministic problem

$$\min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k, \tilde{w}_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, \tilde{w}_k)) \right]. \tag{2.3}$$

The approach of turning a stochastic problem into a deterministic one by replacing uncertain quantities with single typical values highlights the possibility that $\tilde{J}_k$ may itself be obtained by using deterministic methods. We will discuss this approach and its variations in greater detail later in this chapter (see Section 2.3).

Let us now consider the issue of algorithmic minimization over $U_k(x_k)$ in Eqs. (2.2) and (2.3). If $U_k(x_k)$ is a finite set, the minimization can be done by brute force, through exhaustive computation and comparison of the relevant cost expressions. This of course can be very time consuming, particularly for multistep lookahead, but parallel computation can be used with great effect for this purpose [as well as for the calculation of the expected value in the expression (2.2)]. For some discrete control problems, integer programming techniques may also be used. Moreover, for deterministic problems with multistep lookahead, sophisticated exact or approximate shortest path methods may be considered; several methods of this type are available, such as label correcting methods, $A^*$ methods, and their variants (see the author’s books [Ber98] and [Ber17] for detailed accounts, which are consistent with the context of this chapter).
When the control constraint set is infinite, it may be replaced by a finite set through discretization. However, a more efficient alternative may be to use continuous space nonlinear programming techniques. This possibility can be attractive for deterministic problems, which lend themselves better to continuous space optimization; an example is the model predictive control context (see Section 2.4.3).

For stochastic problems and either one-step or multistep lookahead and continuous control spaces, the methodology of stochastic programming, which bears a close connection with linear and nonlinear programming methods, may be useful. We refer to the textbook [Ber17] for a discussion of its application to the approximate DP context, and references to the relevant literature. Still another possibility to simplify the one-step lookahead minimization (2.2) is based on Q-factor approximation, which is also suitable for model-free policy implementation, as we discuss next.

2.1.4 Model-Free Q-Factor Approximation in Value Space

One of the major aims of this book is to address stochastic model-free situations, i.e., methods where a mathematical model [the system functions $f_k$, the probability distribution of $w_k$, and the one-stage cost functions $g_k$] is not used because it is either hard to construct, or simply inconvenient. We assume instead that the system and cost structure can be simulated in software far more easily (think, for example, of a queueing network with complicated but well-defined service disciplines at the queues).

In this section, we will review some of the high-level ideas of passing from model-based to model-free policy implementations for stochastic problems. In particular, we assume that:

(a) There is a computer program/simulator that for any given state $x_k$ and control $u_k$, simulates sample probabilistic transitions to a successor state $x_{k+1}$, and generates the corresponding transition costs.

(b) A cost function approximation $\tilde{J}_{k+1}$ is available. Approaches to obtain $\tilde{J}_{k+1}$ in model-free fashion will be discussed in the context of specific methods later. For example $\tilde{J}_{k+1}$ may be obtained by solving a simpler problem for which a model is available, or it may be separately obtained without a mathematical model, by using a simulator.

We want to use the functions $\tilde{J}_{k+1}$ and the simulator to compute or approximate the Q-factors

$$E \left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right\},$$

for all $u_k \in U_k(x_k)$, and then find the minimal Q-factor and corresponding one-step lookahead control.

† Another possibility is to use the real system to provide the next state and transition cost, but we will not deal explicitly with this case in this book.
Given a state $x_k$, we may use the simulator to compute these Q-factors for all the pairs $(x_k, u_k)$, $u_k \in U_k(x_k)$, and then select the minimizing control. However, in many cases this can be very time-consuming. To deal with this difficulty, we may introduce a parametric family/approximation architecture of Q-factor functions, $\tilde{Q}_k(x_k, u_k, r_k)$, where $r_k$ is the parameter vector and use a least squares fit/regression to approximate the expected value that is minimized in Eq. (2.2). One possibility is to use a neural network parametric architecture; see Chapter 3, where we will discuss methods for selecting and training parametric architectures. The steps are as follows:

**Summary of Q-Factor Approximation Based on Approximation in Value Space**

Assume that the value of $\tilde{J}_{k+1}(x_{k+1})$ is available for any given $x_{k+1}$:

(a) Use the simulator to collect a large number of “representative” sample state-control-successor states-stage cost quadruplets

$$(x_s^k, u_s^k, x_{k+1}^s, g_s^k),$$

and corresponding sample Q-factors

$$\beta_s^k = g_s^k + \tilde{J}_{k+1}(x_{k+1}^s), \quad s = 1, \ldots, q. \quad (2.5)$$

Here $x_{k+1}^s$ is the simulator’s output of the next state

$$x_{k+1}^s = f_k(x_s^k, u_s^k, w_s^k)$$

that corresponds to some disturbance $w_s^k$. This disturbance also determines the one-stage-cost sample

$$g_s^k = g_k(x_s^k, u_s^k, w_s^k).$$

The simulator need not output $w_s^k$; only the sample next state $x_{k+1}^s$ and sample cost $g_s^k$ are needed (see Fig. 2.1.2).

(b) Determine the parameter vector $\tilde{r}_k$ by the least-squares regression

$$\tilde{r}_k \in \arg \min_{r_k} \sum_{s=1}^q (\tilde{Q}_k(x_s^k, u_s^k, r_k) - \beta_s^k)^2. \quad (2.6)$$
Sec. 2.1 General Issues of Approximation in Value Space

11

\[ \beta_k = g_k^s + \hat{J}_{k+1}(x_{k+1}^s) \]

Figure 2.1.2 Schematic illustration of the simulator used for a model-free Q-factor approximation, assuming approximate cost functions \( \hat{J}_{k+1} \) are known. The input to the simulator are sample state-control pairs \((x_k^s, u_k^s)\), and the outputs are a next state sample \( x_{k+1}^s \) and cost sample \( g_k^s \). These correspond to a disturbance \( w_k^s \) according to

\[ x_{k+1}^s = f_k(x_k^s, u_k^s, w_k^s), \quad g_k^s = g_k(x_k^s, u_k^s, w_k^s). \]

The actual value of \( w_k^s \) need not be output by the simulator. The sample Q-factors \( \beta_k^s \) are generated according to Eq. (2.5), and are used in the least squares regression (2.6) to yield a parametric Q-factor approximation \( \hat{Q}_k \) and the policy implementation (2.7).

\[ \tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \hat{Q}_k(x_k, u_k, \bar{r}_k). \quad (2.7) \]

Note some important points about the preceding procedure:

(1) It is model-free in the sense that it is based on Monte Carlo simulation. Moreover, it does not need the functions \( f_k \) and \( g_k \), and the probability distribution of \( w_k \) to generate the policy \( \tilde{\mu}_k \) through the least squares regression (2.6) and the Q-factor minimization (2.7). The simulator to collect the samples (2.5) and the cost function approximation \( \hat{J}_{k+1} \) suffice.

(2) Two approximations are potentially required: One to compute \( \hat{J}_{k+1} \), which is needed for the samples \( \beta_k^s \) [cf. Eq. (2.5)], and another to compute \( \hat{Q}_k \) through the regression (2.6). The approximation methods to obtain \( \hat{J}_{k+1} \) and \( \hat{Q}_k \) may be unrelated.

(3) The policy \( \tilde{\mu}_k \) obtained through the minimization (2.7) is not the same as the one obtained through the minimization (2.2). There are two reasons for this. One is the approximation error introduced by the Q-factor architecture \( \hat{Q}_k \), and the other is the simulation error introduced by the finite-sample regression (2.6). We have to accept these sources of error as the price to pay for the convenience of not requiring a mathematical model for policy implementation.
Let us also mention a variant of the least squares minimization in Eq. (2.6), which is to use a regularized minimization where a quadratic regularization term is added to the least squares objective. This term is a multiple of the squared deviation \( \| r - \hat{r} \|^2 \) of \( r \) from some initial guess \( \hat{r} \). Moreover, in some cases, a nonquadratic minimization may be used in place of Eq. (2.6) to determine \( \tilde{r}_k \), but in this book we will focus on least squares exclusively.

### 2.1.5 Approximation in Policy Space on Top of Approximation in Value Space

A common approach for approximation in policy space, is to introduce a parametric family of policies \( \tilde{\mu}_k(x, r_k) \), where \( r_k \) is a parameter vector. The parametrization may involve a neural network as we will discuss in Chapter 3. Alternatively, the parametrization may involve problem-specific features, exploiting the special structure of the problem at hand.

A general scheme for parametric approximation in policy space is to obtain a large number of sample state-control pairs \((x^s_k, u^s_k)\), \(s = 1, \ldots, q\), such that for each \( s \), \( u^s_k \) is a “good” control at state \( x^s_k \). We can then choose the parameter \( r_k \) by solving the least squares/regression problem

\[
\min_{r_k} \sum_{s=1}^{q} \| u^s_k - \tilde{\mu}_k(x^s_k, r_k) \|^2
\]

(possibly with added regularization). In particular, we may determine \( u^s_k \) using a human or a software “expert” that can choose “near-optimal” controls at given states, so \( \tilde{\mu}_k \) is trained to match the behavior of the expert. Methods of this type are commonly referred to as supervised learning in artificial intelligence (see also the discussion in Section 4.11).

A special case of the above procedure, which connects with approximation in value space, is to generate the sample state-control pairs \((x^s_k, u^s_k)\) through a one-step lookahead minimization of the form

\[
u^s_k \in \arg \min_{u \in U_k(x_k)} E \left\{ \frac{1}{2} \sum_{k=1}^{q} \sum_{s=1}^{q} \left( g_k(x^s_k, u, w_k) + \tilde{J}_{k+1}(f_k(x^s_k, u, w_k)) \right) \right\},
\]

where \( \tilde{J}_{k+1} \) is a suitable (separately obtained) approximation in value space; cf. Eq. (2.2), or an approximate Q-factor based minimization

\[
u^s_k \in \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x^s_k, u, \tilde{r}_k),
\]

[cf. Eq. (2.7)]. In this case, we collect the sample state-control pairs \((x^s_k, u^s_k)\), \(s = 1, \ldots, q\), by using approximation in value space through Eq. (2.9) or Eq. (2.10), and then apply approximation in policy space through
Sec. 2.1 General Issues of Approximation in Value Space

Eq. (2.8) (i.e., approximation in policy space is built on top of approximation in value space).

A major advantage of schemes based on the minimization (2.8) is that once the parametrized policy is obtained, the on-line implementation of the policy is fast and does not involve extensive calculations such as minimizations of the form (2.9) or (2.10). This advantage is generally shared by schemes that are based on approximation in policy space.

2.1.6 When is Approximation in Value Space Effective?

An important question is what constitutes good approximating functions \( \tilde{J}_k \) in a one-step lookahead scheme. An answer that suggests itself is that \( \tilde{J}_k \) should be “close” to the optimal cost-to-go function \( J^*_k \) for all \( k \). This guarantees a certain degree of quality of the approximation scheme, but is neither necessary nor is it satisfied by all or even most good practical schemes.

For example if the approximating values \( \tilde{J}_k(x_k) \) differ from the optimal values \( J^*_k(x_k) \) uniformly by the same constant, the policy obtained by the approximation in value space scheme is optimal. This suggests that a better condition might be that relative values of \( \tilde{J}_k \) and \( J^*_k \) should be “close” to each other, i.e.,

\[
\tilde{J}_k(x_k) - J^*_k(x_k) - J^*_k(x_k') - \tilde{J}_k(x_k') \approx 0,
\]

for all pairs of states \( x_k \) and \( x'_k \). Still, however, this guideline neglects the role of the first stage cost (or the cost of the first \( \ell \) stages in the case of \( \ell \)-step lookahead).

A more accurate predictor of good quality of the suboptimal policy obtained is that the Q-factor approximation error \( Q_k(x_k, u) - \tilde{Q}_k(x_k, u) \) changes gradually as \( u \) changes, where \( Q_k \) and \( \tilde{Q}_k \) denote the exactly optimal Q-factor and its approximation, respectively. For a heuristic explanation, suppose that approximation in value space generates a control \( \tilde{u}_k \) at a state \( x_k \) where another control \( u_k \) is optimal. Then we have

\[
\tilde{Q}_k(x_k, u_k) - \tilde{Q}_k(x_k, \tilde{u}_k) \geq 0, \tag{2.11}
\]

since \( \tilde{u}_k \) minimizes \( \tilde{Q}_k(x_k, \cdot) \), and

\[
Q_k(x_k, u_k) - Q_k(x_k, \tilde{u}_k) \geq 0, \tag{2.12}
\]

since \( u_k \) minimizes \( Q_k(x_k, \cdot) \). If \( \tilde{u}_k \) is far from optimal, the Q-factor difference in Eq. (2.12) will be large, and by adding Eq. (2.11), it follows that the expression

\[
(Q_k(x_k, \tilde{u}_k) - \tilde{Q}_k(x_k, \tilde{u}_k)) - (Q_k(x_k, u_k) - \tilde{Q}_k(x_k, u_k))
\]

is large.
Approximation in Value Space  Chap. 2

Figure 2.1.3 Schematic illustration of the “slope” of the Q-factor approximation error as a predictor of quality of an approximation in value space scheme. At a given state $x_k$, let $u_k$ be optimal, so that it minimizes $Q_k(x_k, u)$ over $u \in U_k(x_k)$, and let $\tilde{u}_k$ be generated by approximation in value space, so that it minimizes $\tilde{Q}_k(x_k, u)$ over $u \in U_k(x_k)$. In the figure on the right the approximation error $Q_k(x_k, u) - \tilde{Q}_k(x_k, u)$ changes gradually (i.e., has small “slope”), and $\tilde{u}_k$ is a good choice, because $Q_k(x_k, \tilde{u}_k)$ is close the optimal $Q_k(x_k, u_k)$. In the figure on the left the approximation error $Q_k(x_k, u) - \tilde{Q}_k(x_k, u)$ changes fast (i.e., has large “slope”), and $\tilde{u}_k$ is a poor choice. In the extreme case where the Q-factors $Q_k(x_k, u)$ and $\tilde{Q}_k(x_k, u)$ differ by a constant, minimization of either one of them yields the same result.

will be even larger. This is not likely to happen if the approximation error $Q_k(x_k, u) - \tilde{Q}_k(x_k, u)$ changes gradually (i.e., has small “slope”) for $u$ in a neighborhood that includes $u_k$ and $\tilde{u}_k$ (cf. Fig. 2.1.3). In many practical settings, as $u$ changes, the corresponding changes in the approximate Q-factors $\tilde{Q}_k(x_k, u)$ tend to have “similar” form to the changes in the exact Q-factors $Q_k(x_k, u)$, thus providing some explanation for the observed success of approximation in value space in practice.

Of course, one would like to have quantitative tests to check the quality of either the approximate cost functions $\tilde{J}_k$ and Q-factors $\tilde{Q}_k$, or the suboptimal policies obtained. However, general tests of this type are not available, and it is often hard to assess how a particular suboptimal policy compares to the optimal, except on a heuristic, problem-dependent basis. Unfortunately, this is a recurring difficulty in approximate DP/RL.

2.2 MULTISTEP LOOKAHEAD

The approximation in value space scheme that we have discussed so far is known as one-step lookahead, since it involves solving a one-step minimization problem at each time $k$ [cf. Eq. (2.1)]. A more ambitious, but also computationally more intensive scheme is multistep lookahead.

As an example, in two-step lookahead we apply at time $k$ and state $x_k$, the control $\tilde{\mu}_k(x_k)$ attaining the minimum in Eq. (2.1), where now
Sec. 2.2  Multistep Lookahead

At State $x_k$

DP minimization

\[
\min_{u_k, \mu_{k+1} \ldots \mu_{k+\ell-1}} E \left\{ g_k(x_k, u_k, w_k) + \sum_{m=k+1}^{k+\ell-1} g_k(x_m, \mu_m(x_m), w_m) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}
\]

\[
\text{Lookahead Minimization} \quad \text{Cost-to-go Approximation}
\]

First $\ell$ Steps “Future” Steps “Future”

**Figure 2.2.1** Illustration of approximation in value space with $\ell$-step lookahead.

$\tilde{J}_{k+1}$ is obtained itself on the basis of a one-step lookahead approximation. In other words, for all possible states $x_{k+1}$ that can be generated via the system equation starting from $x_k$,

\[
x_{k+1} = f_k(x_k, u_k, w_k),
\]

we have

\[
\tilde{J}_{k+1}(x_{k+1}) = \min_{u_k+1 \in U_{k+1}(x_{k+1})} E \left\{ g_{k+1}(x_{k+1}, u_{k+1}, w_{k+1}) + \tilde{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1}, w_{k+1})) \right\},
\]

where $\tilde{J}_{k+2}$ is some approximation of the optimal cost-to-go function $J^*_k$.

Thus two-step lookahead amounts to solving a two-stage version of the DP problem with $x_k$ as the initial state and $\tilde{J}_{k+2}$ as the terminal cost function. Given $x_k$, the solution of this DP problem yields a two-stage policy that consists of the single control $u_k$ for the first lookahead stage, plus a control $\mu_{k+1}(x_{k+1})$ for each value of $x_{k+1} = f_k(x_k, u_k, w_k)$ that can occur at the second lookahead stage $k + 1$. However, once this two-stage policy is computed, the controls $\mu_{k+1}(x_{k+1})$ are discarded, and only $u_k$ is used as the control applied by the two-step lookahead policy at $x_k$. At the next stage, this process is repeated, i.e., we solve a two-stage DP problem with $x_{k+1}$ as the initial state and $\tilde{J}_{k+3}$ as the terminal cost function.

Policies with lookahead of $\ell > 2$ stages are similarly defined: at state $x_k$, we solve an $\ell$-stage version of the DP problem with $x_k$ as the initial state and $\tilde{J}_{k+\ell}$ as the terminal cost function, and use the first control of the $\ell$-stage policy thus obtained, while discarding the others; see Fig. 2.2.1. Of course, in the final stages where $k > N - \ell$, we should shorten the size of lookahead to $N - k$. Note that the simplifications in the one-step lookahead minimization discussed in Section 2.1.2 (assumed certainty equivalence, adaptive sampling, etc), and model-free policy implementation (Section 2.1.3) extend to multistep lookahead.
2.2.1 Multistep Lookahead and Rolling Horizon

There are several ways to compute the lookahead functions $\hat{J}_{k+\ell}$ in $\ell$-step lookahead, similar to the one-step lookahead case. However, there is also another possibility: with sufficiently long lookahead, we may capture enough of the character of the DP problem at hand so that a sophisticated choice of $\hat{J}_{k+\ell}$ may not be needed.

In particular, we may set $\hat{J}_{k+\ell}(x_{k+\ell}) \equiv 0$, or set

$$\hat{J}_{k+\ell}(x_{k+\ell}) = g_N(x_{k+\ell}).$$

The idea is to use a sufficiently large number of lookahead steps $\ell$ to ensure a reasonably faithful approximation of the optimal Q-factors $Q_k$ or cost-to-go functions $J^*_k$ within a constant.† This is also referred to as the rolling horizon approach, but essentially it is the same as multistep lookahead with a simplified cost-to-go approximation. Note that the idea of a rolling horizon is well-suited and applies with few modifications to infinite horizon problems as well.‡

Typically, as the size $\ell$ of the lookahead is chosen larger, the need for a good choice of $\hat{J}_{k+\ell}$ tends to diminish. The reason is that the effective cost-to-go approximation in $\ell$-step lookahead consists of two components:

(a) The cost of an $(\ell - 1)$ step problem involving the last $(\ell - 1)$ stages of the $\ell$-step lookahead.

(b) The terminal cost approximation $\hat{J}_{k+\ell}$.

Since the $(\ell - 1)$-step problem is treated by exact optimization, the overall approximation will be accurate if the contribution of the terminal cost approximation is relatively insignificant. This is likely to be true with large enough $\ell$.

Thus, one is tempted to conjecture that if $\ell$ is increased, then the performance of the lookahead policy is improved. This, however, need not be true always, essentially because beyond the next $\ell$ stages, the policy may be “blind” to the presence of particularly “favorable” or “unfavorable” states. The following example is an illustration.

† See the discussion in Section 2.1.4. Generally, rolling horizon schemes tend to work well if the probability distribution of the state $k+\ell$ steps ahead is roughly independent of the current state and control, or is concentrated around “low cost” states.

‡ For infinite horizon problems the cost-to-go approximations $\hat{J}_k$ will typically be the same at all stages $k$, i.e., $\hat{J}_k \equiv \hat{J}$ for some $\hat{J}$. As a result, the limited lookahead approach produces a stationary policy. In the case of discounted problems with an infinite horizon (see Chapter 4), a simple approach is to use a rolling horizon that is long enough so that the tail cost is negligible and can be replaced by zero, but it is also possible to use a small number of lookahead stages $\ell$, as long as we compensate with a terminal cost function approximation $\hat{J}$. 

Sec. 2.2 Multistep Lookahead

Figure 2.2.2 The 4-stage deterministic shortest problem of Example 2.2.1, illustrating how using a longer lookahead with a cost function approximation \( \tilde{J}_k(x_k) \equiv 0 \) may degrade the performance of the policy obtained.

Example 2.2.1

This is an oversimplified example that demonstrates a pitfall of the multistep lookahead and rolling horizon approaches with approximate cost-to-go functions

\[ \tilde{J}_k(x_k) \equiv 0. \]

It may happen that with longer lookahead the quality of the suboptimal control obtained is degraded.

Consider the 4-stage deterministic shortest problem of Fig. 2.2.2. At the initial state there are two possible controls, denoted \( u \) and \( u' \). At all other states there is only one control available, so a policy is specified by just the initial choice between controls \( u \) and \( u' \). The costs of the four transitions on the upper and the lower path are shown next to the corresponding arcs (0, 1, 2, 1 for the upper path and 0, 2, 0, 10 on the lower path). From the initial state, 2-step lookahead with terminal cost approximation \( \tilde{J}_2 = 0 \), compares 0 + 1 with 0 + 2 and prefers the optimal control \( u \), while 3-step lookahead with terminal cost approximation \( \tilde{J}_3 = 0 \), compares 0 + 1 + 2 with 0 + 2 + 0 and prefers the suboptimal control \( u' \). Thus using a longer lookahead yields worse performance. The problem here has to do with large cost changes at the “edge” of the lookahead (a cost of 0 just after the 2-step lookahead, followed by a cost of 10 just after the 3-step lookahead).

2.2.2 Multistep Lookahead and Deterministic Problems

Generally, the implementation of multistep lookahead can be prohibitively time-consuming for stochastic problems, because it requires at each step the solution of a stochastic DP problem with a horizon that is equal to the size of the lookahead. However, when the problem is deterministic, the lookahead problems are also deterministic, and can be solved by shortest
path methods for a finite spaces problem, or even for an infinite spaces problem after some form of discretization. This makes deterministic problems particularly good candidates for the use of long-step lookahead in conjunction with the rolling horizon approach that we discussed in the preceding section.

Similarly, for a continuous-spaces deterministic optimal control problem, the lookahead minimization may be conveniently solvable by nonlinear programming methods. This idea finds wide application in the context of model predictive control (see the discussion in Section 2.4.3).

**Partially Deterministic Form of Multistep Lookahead**

When the problem is stochastic, one may consider a hybrid approach, partially deterministic approach: at state $x_k$, allow for a stochastic disturbance $w_k$ at the current stage, but fix the future disturbances $w_{k+1}, \ldots, w_{k+\ell-1}$, up to the end of the lookahead horizon, to some typical values. This allows us to bring to bear deterministic methods in the computation of approximate costs-to-go beyond the first stage.

In particular, with this approach, the needed values $\hat{J}_{k+1}(x_{k+1})$ will be computed by deterministic shortest path methods, and will be used to compute the approximate Q-factors of pairs $(x_k, u_k)$ using the formula

$$\hat{Q}_k(x_k, u_k) = E\left\{ g_k(x_k, u_k, w_k) + \hat{J}_{k+1}\left(f_k(x_k, u_k, w_k)\right) \right\},$$
which incorporates the first stage uncertainty. Finally, the control chosen by such a scheme at time $k$ will be

$$\hat{\mu}_k(x_k) \in \arg\min_{u_k \in \mathcal{U}_k(x_k)} \tilde{Q}_k(x_k, u_k);$$

cf. Eq. (2.7).

The idea of fixing uncertain quantities to typical values for approximation purposes is generally referred to as (assumed) certainty equivalence, and will be discussed at length in Section 2.3.2. The idea of using multi-step lookahead for deterministic problems will also be reexamined in the context of the rollout algorithm in Section 2.4.1.

### 2.3 Problem Approximation

A key issue in implementing a limited lookahead policy is the choice of the cost-to-go approximation at the end of the lookahead. In this section, we discuss the problem approximation approach whereby we approximate the optimal cost function $J^*_k$ with some function $\tilde{J}_k$ derived from a related but simpler problem (for example the optimal cost-to-go function of that problem). In the following subsections we consider two approaches:

1. **Simplifying the structure of the problem through enforced decomposition,** e.g., replacing coupling constraints with simpler decoupled constraints or with Lagrange multiplier-related penalties.

2. **Simplifying the probabilistic structure of the problem,** e.g., replacing stochastic disturbances with deterministic ones.

Another approach that can be viewed as problem approximation is aggregation, whereby the original problem is approximated with a related “aggregate” problem that has smaller dimension or fewer states. This problem is solved exactly to yield a cost-to-go approximation for the original problem. Aggregation is also related to the feature-based parametric approximation ideas of Chapter 3, and will be discussed in Chapter 5.

#### 2.3.1 Enforced Decomposition

The simplification/approximation approach is often well-suited for problems involving a number of subsystems that are coupled through the system equation, or the cost function, or the control constraints, but the degree of coupling is “relatively weak.” It is difficult to define precisely what constitutes “weak coupling,” but in specific problem contexts, usually this type of structure is easily recognized. For such problems it is often sensible to introduce approximations by artificially decoupling the subsystems in some way, thereby creating either a simpler problem or a simpler cost calculation, where subsystems can be dealt with in isolation.
There are a number of different ways to effect enforced decomposition, and the best approach is often problem-dependent. Generally, for a deterministic problem, enforced decomposition can be applied both off-line and on-line to produce a suboptimal control sequence. For a stochastic problem, it can be applied with off-line computation of the approximate cost-to-go functions $\tilde{J}_k$ and on-line computation of the corresponding suboptimal policy. We will illustrate these two possibilities with various application contexts in what follows.

**Optimization of One Subsystem at a Time**

When a problem involves multiple subsystems, a potentially interesting approximation approach is to optimize one subsystem at a time. In this way the control computation at time $k$ may become simpler.

As an example consider an $N$-stage deterministic problem, where the control $u_k$ at state $x_k$ consists of $n$ components, $u_k = \{u_k^1, \ldots, u_k^n\}$, with $u_k^i$ corresponding to the $i$th subsystem. Then to compute a cost-to-go approximation at a given state $x_k$, one may optimize over the control sequence of a single subsystem, while keeping the controls of the remaining subsystems at some nominal values. Thus, upon arriving at $x_k$, we first optimize over the control sequence $\{u_k^1, u_{k+1}^1, \ldots, u_{N-1}^1\}$ of the first subsystem, then optimize over the controls of the second subsystem, and so on, while keeping the controls of the other subsystem at the latest “optimal” values computed.

There are several possible variations; for example to make the order in which the subsystems are considered subject to optimization as well, or to repeat cycling through the subsystems multiple times, each time using the results of the latest computation as nominal values of subsystem controls. This is similar to a “coordinate descent” approach, used in other optimization contexts.

Additional variations are obtained when we use approximate minimization over $u_k$ in Eq. (2.1), and also when the expected value over $w_k$ is computed approximately via adaptive simulation or a certainty equivalence approximation (cf. Section 2.1.2).

**Example 2.3.1 (Vehicle Routing)**

Consider $n$ vehicles that move along the arcs of a given graph. Each node of the graph has a known “value” and the first vehicle that will pass through the node will collect its value, while vehicles that pass subsequently through the node do not collect any value. This may serve as a model of a situation where there are various valuable tasks to be performed at the nodes of a transportation network, and each task can be performed at most once and by a single vehicle. We assume that each vehicle starts at a given node and after at most a given number of arc moves, it must return to some other
Sec. 2.3 \hspace{1cm} Problem Approximation \hspace{1cm} 21

Figure 2.3.1 Schematic illustration of the vehicle routing problem and the one-vehicle-at-a-time approach. As an example, given the position pair $x_k = (1, 4)$ of the two vehicles and the current valuable tasks at positions 6 and 9, we consider moves to all possible positions pairs $x_{k+1}^*$:

$$(2, 2), (2, 3), (2, 6), (2, 7), (3, 2), (3, 3), (3, 6), (3, 7).$$

From each of these pairs, we first compute the best route of vehicle 1 assuming vehicle 2 does not move, and then the best route vehicle 2, taking into account the previously computed route of vehicle 1. We then select the pair $x_{k+1}^*$ that results in optimal value, and move the vehicles to the corresponding positions.

The problem is to find a route for each vehicle satisfying these constraints, so that the total value collected by the vehicles is maximized.

This is a difficult combinatorial problem that in principle can be approached by DP. In particular, we can view as state the $n$-tuple of current positions of the vehicles together with the list of nodes that have been visited by some vehicle in the past, and have thus “lost” their value. Unfortunately, the number of these states is enormous (it increases exponentially with the number of nodes and the number of vehicles). The version of the problem that involves a single vehicle, while still difficult in principle, can often be solved in reasonable time either exactly by DP or fairly accurately using a suitable heuristic. Thus a one-step lookahead policy suggests itself, with the value-to-go approximation obtained by solving single vehicle problems.

In particular, in a one-step lookahead scheme, at a given time $k$ and from a given state $x_k$ we consider all possible $n$-tuples of moves by the $n$ vehicles. At the resulting state $x_{k+1}$ corresponding to each $n$-tuple of vehicle moves, we approximate the optimal value-to-go with the value corresponding to a suboptimal set of paths. These paths are obtained as follows: we fix an order of the vehicles and we calculate a path for the first vehicle, starting at $x_{k+1}$, assuming the other vehicles do not move. (This is done either optimally by DP, or near optimally using some heuristic.) Then we calculate a path for the second vehicle, taking into account the value collected by the first vehicle,
and we similarly continue: for each vehicle, we calculate in the given order a path, taking into account the value collected by the preceding vehicles. We end up with a set of paths that have a certain total value associated with them. This is the value \( \hat{J}_{k+1}(x_{k+1}) \) associated with the successor state \( x_{k+1} \). We repeat with all successor states \( x_{k+1} \) corresponding to all the \( n \)-tuples of vehicle moves that are possible at \( x_k \). We then use as suboptimal control at \( x_k \) the \( n \)-tuple of moves that yields the best value; see Fig. 2.3.1.

There are several enhancements and variations of the scheme just described. For example, we can consider multiple alternative orders for optimizing paths one-at-a-time, and choose the \( n \)-tuple of moves that corresponds to the best value obtained. Other variations may include travel costs between nodes of the graph, and constraints on how many tasks can be performed by each vehicle.

### Constraint Decoupling by Constraint Relaxation

Let us now consider problems involving coupled subsystems where the coupling comes only through the control constraint. Typical cases involve the allocation of a limited resource to a set of subsystems whose system equations are completely decoupled from each other. We will illustrate with examples a few enforced decomposition approaches to deal with such situations. The first approach is constraint relaxation, whereby the constraint set is replaced by another constraint set that does not involve coupling.

**Example 2.3.2 (Restless Multiarmed Bandit Problems)**

An interesting DP model, generally referred to as the multiarmed bandit problem, involves \( n \) projects of which only one can be worked on at any time period. Each project \( i \) is characterized at time \( k \) by its state \( x_i^k \). If project \( i \) is worked on at time \( k \), one receives an expected reward \( R^i(x_i^k) \), and the state \( x_i^k \) then evolves according to the equation

\[
x_{i+1} = f_i(x_i^k, w_i^k),
\]

where \( w_i^k \) is a random disturbance with probability distribution depending on \( x_i^k \) but not on prior disturbances. If project \( i \) is not worked on, its state changes according to

\[
x_{i+1} = \bar{f}(x_i^k, \bar{w}_i^k),
\]

where \( \bar{f} \) is a given function and \( \bar{w}_i^k \) is a random disturbance with distribution depending on \( x_i^k \) but not on prior disturbances. Furthermore, a reward \( R^i(x_i^k) \) is earned, where \( R^i \) is a given function. The projects are coupled through the control constraint (only one of the projects may be worked on at any one
A suboptimal enforced decomposition approach is to consider the single project problems where a single project is worked on through the entire remaining horizon, and add the contributions of the $n$ problems to form an optimal reward approximation.

In particular, suppose that the optimal reward function $J^*_k(x^1, \ldots, x^n)$ is approximated by a separable function of the form

$$\sum_{i=1}^n \tilde{J}_k^i(x^i),$$

where each $\tilde{J}_k^i$ is a function that quantifies the contribution of the $i$th project to the total reward. The corresponding one-step lookahead policy selects at time $k$ the project $i$ that maximizes

$$R_i^k(x^i) + \sum_{j \neq i} \mathcal{R}^i(x^j) + E \left\{ \tilde{J}_{k+1}^i (f^i(x^i, w^i)) \right\} + \sum_{j \neq i} E \left\{ \tilde{J}_{k+1}^j (\mathcal{T}^j(x^j, w^j)) \right\},$$

which can also be written as

$$R_i^k(x^i) - \mathcal{R}^i(x^i) + E \left\{ \tilde{J}_{k+1}^i (f^i(x^i, w^i)) - \tilde{J}_{k+1}^i (\mathcal{T}^i(x^i, w^i)) \right\}$$

$$+ \sum_{j=1}^n \left\{ \mathcal{R}^i(x^j) + E \left\{ \tilde{J}_{k+1}^j (\mathcal{T}^j(x^j, w^j)) \right\} \right\}. $$

Noting that the last term in the above expression does not depend on $i$, it follows that the one-step lookahead policy takes the form

$$\text{work on project } i \quad \text{if} \quad \tilde{m}_k^i(x^i) = \max_j \{ \tilde{m}_k^j(x^j) \},$$

where for all $i$,

$$\tilde{m}_k^i(x^i) = R_i^k(x^i) - \mathcal{R}^i(x^i) + E \left\{ \tilde{J}_{k+1}^i (f^i(x^i, w^i)) - \tilde{J}_{k+1}^i (\mathcal{T}^i(x^i, w^i)) \right\}. $$

† In the classical and simplest version of the problem, the state of a project that is not worked on remains unchanged and produces no reward, i.e.,

$$x_{k+1}^i = x_k^i, \quad \mathcal{R}^i(x_k^i) = 0, \quad \text{if } i \text{ is not worked on at time } k.$$

This problem admits optimal policies with a nice structure that can be computationally exploited. The problem has a long history and is discussed in many sources; we refer to [Ber12] and the references quoted there. In particular, in favorable instances of the problem, optimal policies have the character of an index rule, which is structurally similar to the decoupled suboptimal decision rules discussed in this section, and has been analyzed extensively, together with its variations and special cases. The term “restless” in the title of the present example, introduced by Whittle [Whi88], refers to the fact that the states of the projects that are not worked on may change.
An important question for the implementation of the preceding suboptimal control scheme is the determination of the separable reward function terms $\tilde{J}_{i}^{k+1}$. There are several possibilities here, and the best choice may strongly depend on the problem’s structure. One possibility is to compute $\tilde{J}_{i}^{k+1}$ as the optimal cost-to-go function for a problem involving just project $i$, i.e., assuming that none of the other projects $j \neq i$ will be worked on for the remaining periods $k+1, \ldots, N-1$. This corresponds to restricting the control constraint set of the problem, and involves a single-project optimization that may be tractable.

An alternative possibility is to use a separable parametric approximation of the form
\[
\sum_{i=1}^{n} \tilde{J}_{i}^{k+1}(x_{i}^{k+1}, r_{i}^{k+1}),
\]
where $r_{i}^{k+1}$ are vectors of “tunable” parameters. The values of $r_{i}^{k+1}$ can be obtained by some “training” algorithm such as the ones to be discussed in Chapter 3.

**Constraint Decoupling by Lagrangian Relaxation**

Another approach to deal with coupled constraints is to replace them with linear Lagrange multiplier-related penalty functions that are added to the cost function. We illustrate this approach with an extension of the preceding multiarmed bandit Example 2.3.2.

**Example 2.3.3 (Separable Lower Bound Approximation in Multiarmed Bandit Problems)**

Let us consider a version of the multiple projects Example 2.3.2 involving a more general form of control. Here there are $n$ subsystems, with a control signal $u_{k}^{i}$ applied to subsystem $i$ at time $k$. Instead of the requirement that only one subsystem is worked on at any one time, we assume a control constraint of the form
\[
u_{k} = (u_{k}^{1}, \ldots, u_{k}^{n}) \in U, \quad k = 0, 1, \ldots,
\]
where the set $U$ is given (Example 2.3.2 is obtained as the special case where $U$ consists of the union of the coordinate vectors, i.e., those whose components are equal to 0, except for one component that is equal to 1). The $i$th subsystem is described by
\[
x_{i}^{k+1} = f'(x_{i}^{k}, u_{k}^{i}, w_{i}^{k}), \quad i = 1, \ldots, n, \quad k = 0, 1, \ldots,
\]
where $x_{i}^{k}$ is the state taking values in some space, $u_{k}^{i}$ is the control, $w_{i}^{k}$ is a random disturbance, and $f'$ is a given function. We assume that $w_{k}^{i}$ is selected according to a probability distribution that may depend on $x_{k}^{i}$ and $u_{k}^{i}$, but not on prior disturbances or the disturbances $w_{k}^{j}$ of the other subsystems $j \neq i$. The cost incurred at the $k$th stage by the $i$th subsystem is
\[
g^{i}(x_{k}^{i}, u_{k}^{i}, w_{k}^{i}), \quad (2.13)
\]
where \( g^i \) is a given one-stage cost function. For notational convenience, we assume stationarity of the system equation and the cost per stage, but the approach to be discussed applies to the nonstationary case as well.

One possibility for a separable approximation of the problem is to replace the constraint \( u_k \in U \) by a smaller or larger decoupled constraint, i.e., requiring that

\[
u_k^i \in U^i, \quad i = 1, \ldots, n, \quad k = 0, 1, \ldots,
\]

where the subsets \( U^1, \ldots, U^n \) satisfy \( U^1 \times \cdots \times U^n \subset U \) or \( U \subset U^1 \times \cdots \times U^n \), respectively.

We discuss another possibility for the case where the constraint set \( U \) includes linear inequality constraints. As a simple example, let us focus on a constraint set \( U \) of the form

\[
U = \left\{(u^i, \ldots, u^n) \mid u^i \in U^i \subset \mathbb{R}, \ i = 1, \ldots, n, \ \sum_{i=1}^{n} c^i u^i_k \leq b\right\}, \quad (2.14)
\]

where \( c^1, \ldots, c^n \), and \( b \) are some scalars.† Here we replace the coupling constraint

\[
\sum_{i=1}^{n} c^i u^i_k \leq b, \quad k = 0, 1, \ldots, \quad (2.15)
\]

by a “relaxed” (larger) constraint whereby we require that

\[
\sum_{k=0}^{N-1} \sum_{i=1}^{n} c^i u^i_k \leq N b. \quad (2.16)
\]

Roughly speaking, the constraint (2.16) requires that the coupling constraint (2.15) is satisfied “on the average,” over the \( N \) stages.

We may now obtain a lower bound approximation of the optimal cost of our problem by assigning a scalar Lagrange multiplier \( \lambda \geq 0 \) to the constraint (2.16), and add a Lagrangian term

\[
\lambda \left( \sum_{k=0}^{N-1} \sum_{i=1}^{n} c^i u^i_k - N b \right) \quad (2.17)
\]

to the cost function. This amounts to replacing the \( k \)th stage cost (2.13) by

\[
g^i(x^i_k, u^i_k, w^i_k) + \lambda c^i u^i_k,
\]

while replacing the coupling constraint (2.14) with the decoupled constraint

\[
u_k^i \in U^i, \quad i = 1, \ldots, n,
\]

† More general cases, where \( u^i \) and \( b \) are multi-dimensional, and \( c^i \) are replaced by matrices of appropriate dimension, can be handled similarly, albeit with greater computational complications.
cf. Eq. (2.14). This is a lower bound approximation, as is typical in Lagrange multiplier-based decomposition approaches in linear and nonlinear programming (see e.g., [BeT97], [Ber16]). To see this, note that for every feasible solution of the original problem, the Lagrangian term \((2.17)\) makes a nonpositive contribution when added to the cost function, while with the constraint relaxed, the resulting optimal cost can only be reduced further.

With the subsystems now decoupled, we may solve each single subsystem problem separately, thereby obtaining a separable lower bound approximation

\[
\sum_{i=1}^{n} \tilde{J}_{k}^{i}(x_{k}^{i}, \lambda)
\]

for every \(k = 1, \ldots, N - 1\). This approximation can in turn be used to obtain a suboptimal one-step lookahead policy. Note that we may also try to optimize the approximation over \(\lambda\), either by ad hoc experimentation or by a more systematic optimization method.† Another possibility is to use a more general Lagrangian term of the form

\[
\left(\sum_{k=0}^{N-1} \lambda_{k} \sum_{i=1}^{n} c^{i} u_{k}^{i} - Nb\right),
\]

in place of the term \((2.17)\), where \(\lambda_{0}, \ldots, \lambda_{N-1} \geq 0\) are time-varying scalar multipliers.

### 2.3.2 Probabilistic Approximation - Certainty Equivalent Control

We will now consider problem approximation based on modifying the underlying probabilistic structure. The most common example of this type is the certainty equivalent controller (CEC). It replaces the stochastic disturbances with deterministic variables that are fixed at some “typical” values. Thus it acts as if a form of the certainty equivalence principle were holding, cf. the discussion of linear quadratic problems in Section 1.3.7.

The advantage of the CEC is that it involves a much less demanding computation than the stochastic DP algorithm: it requires the solution of a deterministic optimal control problem at each stage. This problem yields an optimal control sequence, the first component of which is used at the current stage, while the remaining components are discarded. Thus the CEC is able to deal with stochastic and even partial information problems by using the more flexible and potent methodology of deterministic optimal control.

† Maximization of the lower bound approximation over \(\lambda\) is an interesting possibility. This is common in duality-based optimization. Generally the approach of this example falls within the framework of Lagrangian relaxation, a decomposition method that is based on the use of Lagrange multipliers and duality theory; see e.g., [BeT97], [Ber15], [Ber16].
We will describe the CEC for the stochastic DP problem of Section 1.2. Suppose that for every state-control pair \((x_k, u_k)\) we have selected a “typical” value of the disturbance, which we denote by \(\tilde{w}_k(x_k, u_k)\). For example the expected values
\[
\tilde{w}_k(x_k, u_k) = E\{w_k \mid x_k, u_k\},
\]
can serve as typical values, if the disturbance spaces are convex subsets of Euclidean spaces [so that they include \(\tilde{w}_k(x_k, u_k)\)].

To implement the CEC at state \(x_k\) and stage \(k\) we solve a deterministic optimal control problem obtained from the original problem by replacing all uncertain quantities by their typical values. In particular, we solve the problem
\[
\min_{x_{k+1} = f_k(x_k, u_k, \tilde{w}_k(x_k, u_k)) \atop u_j \in U_j(x_j), \ j = k, \ldots, N-1} \left[ g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, u_i, \tilde{w}_i(x_i, u_i)) \right]. \tag{2.18}
\]
If \(\{\tilde{u}_k, \ldots, \tilde{u}_{N-1}\}\) is the optimal control sequence for this problem, we use the first control in this sequence and discard the remaining controls:
\[
\tilde{\mu}_k(x_k) = \tilde{u}_k.
\]

An alternative implementation of the CEC is to compute off-line an optimal policy
\[
\{\mu^d_0(x_0), \ldots, \mu^d_{N-1}(x_{N-1})\}
\]
for the deterministic problem
\[
\begin{align*}
\text{minimize} & \quad g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), \tilde{w}_k(x_k, u_k)) \\
\text{subject to} & \quad x_{k+1} = f_k(x_k, \mu_k(x_k), \tilde{w}_k(x_k, u_k)), \quad \mu_k(x_k) \in U_k, \quad k \geq 0, \tag{2.19}
\end{align*}
\]
by using the DP algorithm. Then the control input \(\tilde{\mu}_k(I_k)\) applied by the CEC at time \(k\) is given by
\[
\tilde{\mu}_k(I_k) = \mu^d_k(x_k).
\]
The two variants of the CEC just given are equivalent in terms of performance. The main difference is that the first variant is well-suited for on-line replanning, while the second variant is more suitable for an off-line implementation.

Finally, let us note that the CEC can be extended to imperfect state observation problems, where the state \(x_k\) is not known at time \(k\), but instead an estimate of \(x_k\) is available, which is based on measurements that have been obtained up to time \(x_k\). In this case, we find a suboptimal control similarly, as in Eqs. (2.18) and (2.19), but with \(x_k\) replaced by the estimate, as if this estimate were exact.
Certainty Equivalent Control with Heuristics

Even though the CEC approach simplifies a great deal the computations, it still requires the optimal solution of a deterministic tail subproblem at each stage [cf. Eq. (2.18)]. This problem may still be difficult, and a more convenient approach may be to solve it suboptimally using a heuristic algorithm. In particular, given the state \( x_k \) at time \( k \), we may use some (easily implementable) heuristic to find a suboptimal control sequence \( \{ \tilde{u}_k, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1} \} \) for the problem of Eq. (2.18), and then use \( \tilde{u}_k \) as the control for stage \( k \).

An important enhancement of this idea is to use minimization over the first control \( u_k \) and to use the heuristic only for the remaining stages \( k+1, \ldots, N-1 \). To implement this variant of the CEC, we must apply at time \( k \) a control \( \tilde{u}_k \) that minimizes over \( u_k \in U_k(x_k) \) the expression

\[
g_k(x_k, u_k, \tilde{w}_k(x_k, u_k)) + H_{k+1}(x_{k+1}), \tag{2.20}
\]

where

\[
x_{k+1} = f_k(x_k, u_k, \tilde{w}_k(x_k, u_k)), \tag{2.21}
\]

and \( H_{k+1} \) is the cost-to-go function corresponding to the heuristic, i.e., \( H_{k+1}(x_{k+1}) \) is the cost incurred over the remaining stages \( k+1, \ldots, N-1 \) starting from a state \( x_{k+1} \), using the heuristic. This is a hybrid approach: it resembles one-step lookahead with lookahead function \( H_{k+1} \), and it resembles certainty equivalence in that the uncertain quantities have been replaced by their typical values.

Note that for any next state \( x_{k+1} \), it is not necessary to have a closed-form expression for the heuristic cost-to-go \( H_{k+1}(x_{k+1}) \). Instead we can generate this cost by running the heuristic from \( x_{k+1} \) and computing the corresponding cost. Thus all the possible next states \( x_{k+1} \) must be computed for all possible values of the control \( u_k \), and then the heuristic must be run from each of these \( x_{k+1} \) to calculate \( H_{k+1}(x_{k+1}) \), which is needed in the minimization of the expression (2.20).

Example 2.3.4 (Parking with Probability Estimates)

Consider the one-directional parking problem of Example 1.3.3, where a driver is looking for a parking space along a line of \( N \) spaces, with a garage at the end of the line (position \( N \)). The driver starts at space 0 and traverses the parking spaces sequentially, i.e., continues to subsequent spaces, up to a decision to park in space \( k \) at cost \( c(k) \), if space \( k \) is free, or upon reaching the garage where parking is mandatory at cost \( C \). In Example 1.3.3, we assumed that space \( k \) is free with a given and fixed probability \( p(k) \), independently of whether other parking spaces are free or not.

Assume instead that \( p(k) \) is an estimate that is based on the driver’s observations of the status of preceding spaces. For example, this estimation process may involve exploiting probabilistic relations that may exist between
the parking statuses of different spaces. In particular, let us assume that the driver, upon arrival at space \( k \), can estimate the belief state of the spaces that lie ahead, i.e., the vector of conditional probabilities \( \{p(k+1), \ldots, p(N)\} \) given the observations of the spaces \( 0, \ldots, k \).

The problem now becomes very hard to solve by exact DP, because the state space is infinite: at time \( k \) the state consists of the free/taken status of the current position \( k \), plus the belief state of the spaces that lie ahead. However, a simple suboptimal approach to the problem can be based on certainty equivalence: at time \( k \), we fix the free/taken probabilities of the spaces that lie ahead to their current belief values, and act as if these values will not change in the future. Then upon reaching space \( k \), the fast DP algorithm of Example 1.3.3 can be used to solve on-line the corresponding fixed probabilities problem, and to find the corresponding suboptimal decision.

As an illustration, let \( p(k) \) be estimated by using the ratio \( R(k) \) of the number of free spaces encountered up to reaching space \( k \), divided by the total number \( k + 1 \). Knowing \( R(k) \), the driver adjusts the probabilities \( p(m) \) for \( m > k \) to the level

\[
\hat{p}(m, R(k)) = \gamma p(m) + (1 - \gamma) R(k),
\]

where \( \gamma \) is a known constant between 0 and 1. The problem can then be solved by exact DP, by using as state at time \( k \) the free/taken status of space \( k \) together with the ratio \( R(k) \) [in the terminology of Section 1.3, \( R(k) \) is a sufficient statistic, which contains all the relevant information for the purposes of control]. The number of possible values of \( R(k) \) grows exponentially with \( k \), so the solution by exact DP may become intractable for large \( N \). However, by applying the probabilistic approximation approach of this example, the corresponding suboptimal policy may be easily obtained and implemented on-line.

**Partial Certainty Equivalent Control**

In the preceding descriptions of the CEC all future and present uncertain quantities are fixed at their typical values. A useful variation is to fix at typical values only *some* of these quantities. For example, a partial state information problem may be treated as one of perfect state information, using an estimate \( \tilde{x}_k \) of \( x_k \) as if it were exact, while fully taking into account the stochastic nature of the disturbances. Thus, if \( \{\mu^p_0(x_0), \ldots, \mu^p_{N-1}(x_{N-1})\} \) is an optimal policy obtained from the DP algorithm for the stochastic perfect state information problem

\[
\min E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}
\]

subject to \( x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \quad \mu_k(x_k) \in U_k, \quad k = 0, \ldots, N - 1, \)

then the control input applied by this variant of CEC at time \( k \) is \( \mu^p_k(\tilde{x}_k) \), where \( \tilde{x}_k \) is the estimate of \( x_k \) given the information available up to time \( k \). Let us provide an example.
Example 2.3.5 (The Unscrupulous Innkeeper)

Consider an innkeeper who charges one of \( m \) different rates \( r_1, \ldots, r_m \) for a room as the day progresses, depending on whether he has many or few vacancies, so as to maximize his expected total income during the day. A quote of a rate \( r_i \) is accepted with probability \( p_i \) and is rejected with probability \( 1 - p_i \), in which case the customer departs, never to return during that day. When the number \( y \) of customers that will ask for a room during the rest of the day (including the customer currently asking for a room) is known and the number of vacancies is \( x \), the optimal expected income \( \tilde{J}(x, y) \) of the innkeeper is given by the DP algorithm

\[
\tilde{J}(x, y) = \max_{i=1, \ldots, m} \left[ p_i (r_i + \tilde{J}(x-1, y-1)) + (1 - p_i) \tilde{J}(x, y-1) \right], \quad (2.22)
\]

for all \( x \geq 1 \) and \( y \geq 1 \), with initial conditions

\[
\tilde{J}(x, 0) = \tilde{J}(0, y) = 0, \quad \text{for all } x \text{ and } y.
\]

This algorithm can be used to obtain the values of \( \tilde{J}(x, y) \) for all pairs \((x, y)\).

Consider now the case where the innkeeper does not know \( y \) at the times of decision, but instead only maintains a probability distribution for \( y \). Then, it can be seen that the problem becomes a difficult partial state information problem. The exact DP algorithm should then be executed over the set of the pairs of \( x \) and the belief state. Yet a reasonable partially stochastic CEC is based on approximating the optimal cost-to-go of subsequent decisions with \( \tilde{J}(x-1, \tilde{y}-1) \) or \( \tilde{J}(x, \tilde{y}-1) \), where the function \( \tilde{J} \) is calculated by the preceding recursion (2.22) and \( \tilde{y} \) is an estimate of \( y \), such as the closest integer to the expected value of \( y \). In particular, according to this one-step lookahead policy, when the innkeeper has a number of vacancies \( x \geq 1 \), he quotes to the current customer the rate that maximizes

\[
p_i (r_i + \tilde{J}(x-1, \tilde{y}-1) - \tilde{J}(x, \tilde{y}-1)).
\]

Thus in this suboptimal scheme, the innkeeper acts as if the estimate \( \tilde{y} \) were exact.

Other Variations of Certainty Equivalent Control

It is also possible to use more general approaches to implement one-step lookahead control based on simplification of the probabilistic structure of the problem. The possibilities are highly problem-dependent by nature, but we may distinguish a few general techniques, which we illustrate through examples.
Example 2.3.6 (Decoupling Disturbance Distributions)

Let us consider a CEC approach in the context of enforced decomposition (cf. Section 2.3.1) when the subsystems are coupled only through the disturbance. In particular, consider \( n \) subsystems of the form

\[
x_{k+1}^i = f_i(x_k^i, u_k^i, w_k^i), \quad i = 1, \ldots, n.
\]

Here the \( i \)th subsystem has its own state \( x_k^i \), control \( u_k^i \), and cost per stage \( g_i(x_k^i, u_k^i, w_k^i) \), but the probability distribution of \( w_k^i \) depends on the full state \( x_k = (x_1^k, \ldots, x_n^k) \).

A natural form of suboptimal control is to solve at each stage \( k \) and for each \( i \), the \( i \)th subsystem optimization problem where the probability distribution of the future disturbances \( w_{k+1}^i, \ldots, w_{N-1}^i \) is “decoupled,” in the sense that it depends only on the corresponding “local” states \( x_{k+1}^i, \ldots, x_{N-1}^i \). This distribution may be derived by using some nominal values \( \tilde{x}_{j+1}^i, \ldots, \tilde{x}_{N-1}^i \), \( j \neq i \), of the future states of the other subsystems, and these nominal values may in turn depend on the full current state \( x_k \). The first control \( u_k^i \) in the optimal policy thus obtained is applied at the \( i \)th subsystem in stage \( k \), and the remaining portion of this policy is discarded.

Example 2.3.7 (Approximation Using Scenarios)

We noted earlier the possibility to approximate the optimal cost-to-go with a CEC approach, whereby for a given state \( x_{k+1} \) at time \( k + 1 \), we fix the remaining disturbances at some nominal values \( \tilde{w}_{k+1}, \ldots, \tilde{w}_{N-1} \), and we compute an optimal control or heuristic-based trajectory starting from \( x_{k+1} \) at time \( k + 1 \).

This CEC approximation involves a single nominal trajectory of the remaining uncertainty. To strengthen this approach, it is natural to consider multiple trajectories of the uncertainty, called scenarios, and to construct an approximation to the optimal cost-to-go that involves, for every one of the scenarios, the cost of either an optimal or a heuristic policy.

Mathematically, we assume that we have a method, which at a given state \( x_{k+1} \), generates \( q \) uncertainty sequences

\[
w^s(x_{k+1}) = (w_{k+1}^s, \ldots, w_{N-1}^s), \quad s = 1, \ldots, q.
\]

These are the scenarios considered at state \( x_{k+1} \). The optimal cost \( J^*_k(x_{k+1}) \) is then approximated by

\[
J_{k+1}(x_{k+1}, r) = \sum_{s=1}^q r_s C_s(x_{k+1}), \quad (2.23)
\]

where \( r = (r_1, \ldots, r_q) \) is a probability distribution, i.e., a vector of nonnegative parameters that add to 1, and \( C_s(x_{k+1}) \) is the cost corresponding to an
occurrence of the scenario $w^s(x_{k+1})$, when starting from state $x_{k+1}$ and using either an optimal or a given heuristic policy.

The parameters $r_1, \ldots, r_q$ may depend on the time index, and may be interpreted as “aggregate probabilities” that encode the aggregate effect on the cost-to-go function of uncertainty sequences that are similar to the scenario $w^s(x_{k+1})$. They may be computed using some ad hoc scheme, or some more systematic approach. This idea is also related to the rollout approach that is the subject of the next section.

\section*{2.4 ROLLOUT AND MODEL PREDICTIVE CONTROL}

The principal aim of rollout is policy improvement, i.e., start with a suboptimal/heuristic policy, called the base policy (or sometimes, the default policy), and produce an improved policy by limited lookahead minimization with use of the heuristic at the end. This policy is called the rollout policy, and the fact that it is indeed “improved” will be established, under various conditions, in what follows in this section and also in Chapter 4.

In its purest one-step lookahead form, rollout can be defined very simply: it is approximation in value space with the approximate cost-to-go values $\tilde{J}_{k+1}(x_{k+1})$ calculated by running the base policy, starting from each possible next state $x_{k+1}$. There is also an $\ell$-step lookahead generalization, where the heuristic is used to obtain the approximate cost-to-go values $\tilde{J}_{k+\ell}(x_{k+\ell})$ from each possible next state $x_{k+\ell}$ (see Fig. 2.4.1). In a variant for problems involving a long horizon, the run of the base policy may be “truncated,” i.e., it may be used for a limited number of steps, with some cost function approximation at the end to take into account the cost of the remaining steps.

The choice of base policy is of course important for the performance of the rollout approach. However, experimental evidence has shown that the choice of base policy may not be crucial for many contexts, and in
fact surprisingly good rollout performance may be attained even with a relatively poor base heuristic, particularly with a long lookahead.

Note also here a connection between the rollout and problem approximation approaches. Suppose that we use as base heuristic an optimal policy for the approximating problem. Then the one-step (or multistep) rollout policy is the same as the one obtained by one-step (or multistep, respectively) lookahead with terminal cost function approximation equal to the optimal cost of the approximating problem.

In Chapter 4 within the context of infinite horizon problems. We will view there the method of policy iteration as a perpetual rollout algorithm, whereby a sequence of suboptimal policies is generated, each one being a rollout policy obtained by using the preceding one as a base policy. In this chapter, however, we focus on a one-time policy improvement starting from a given base policy.

We will describe rollout first for deterministic problems and one-step lookahead, and then for stochastic problems, in Sections 2.4.1 and 2.4.2, respectively. In Section 2.4.3, we discuss model predictive control, a dominant control system design methodology at present, which is a special case of rollout.

In Section 2.4.2, we will also discuss the use of rollout in an enhanced implementation of any given suboptimal policy, possibly obtained through some other approximation scheme. This is to use one-step or multistep lookahead with the cost function approximation consisting of two parts:

(a) Rollout with the given base policy over a limited horizon.

(b) A terminal cost function approximation at the end of the rollout horizon, such as an estimate of the true cost function of the policy.

We will also discuss schemes of this type in Chapter 4, Section 4.5.3, in the context of infinite horizon problems.

2.4.1 Rollout for Deterministic Problems

Let us consider a deterministic DP problem with a finite number of controls. Given a state \( x_k \) at time \( k \), rollout considers all the tail subproblems that start at every possible next state \( x_{k+1} \), and solves them suboptimally by using some algorithm, called base heuristic.† Thus when at \( x_k \), rollout generates on-line the next states \( x_{k+1} \) that correspond to all \( u_k \in U_k(x_k) \), and uses the base heuristic to compute the sequence of states \( \{x_{k+1}, \ldots, x_N\} \) and controls \( \{u_{k+1}, \ldots, u_{N-1}\} \) such that

\[
x_{i+1} = f_i(x_i, u_i), \quad i = k, \ldots, N - 1.
\]

† For deterministic problems we prefer to use the term “base heuristic” rather than “base policy” for reasons to be explained later in this section, in the context of the notion of sequential consistency.
Figure 2.4.2 Schematic illustration of rollout with one-step lookahead for a deterministic problem. At state $x_k$, for every pair $(x_k, u_k)$, $u_k \in U_k(x_k)$, the base heuristic generates a Q-factor $\tilde{Q}_k(x_k, u_k)$ [cf. Eq. (2.25)], and selects the control $\tilde{\mu}_k(x_k)$ with minimal Q-factor.

The rollout algorithm then applies the control that minimizes over $u_k \in U_k(x_k)$ the cost expression for stages $k$ to $N$:

$$g_k(x_k, u_k) + g_{k+1}(x_{k+1}, u_{k+1}) + \cdots + g_{N-1}(x_{N-1}, u_{N-1}) + g_N(x_N). \quad (2.24)$$

More succinctly, the rollout algorithm applies at state $x_k$ the control $\tilde{\mu}_k(x_k)$ given by the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k),$$

where $\tilde{Q}_k(x_k, u_k)$ is the approximate Q-factor defined by

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k)), \quad (2.25)$$

with $H_{k+1}(x_{k+1})$ denoting the cost of the base heuristic starting from state $x_{k+1}$ [i.e., $H_{k+1}(x_{k+1})$ is the sum of all the terms in Eq. (2.24), except the first]; see Fig. 2.4.2. The rollout process defines a suboptimal policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, referred to as the rollout policy.

**Example 2.4.1 (Traveling Salesman Problem)**

Let us consider the traveling salesman problem, whereby a salesman wants to find a minimum mileage/cost tour that visits each of $N$ given cities $c = 0, \ldots, N-1$ exactly once and returns to the city he started from (cf. Example 1.3.1). With each pair of distinct cities $c, c'$, we associate a traversal cost $a_{cc'}$. Note that we assume that we can go directly from every city to every other city. There is no loss of generality in doing so because we can assign a very high cost $a_{cc'}$ to any pair of cities $(c, c')$ that is precluded from participation.
in the solution. The problem is to find a visit order that goes through each city exactly once and whose sum of costs is minimum.

There are many heuristic approaches for solving the traveling salesman problem. For illustration purposes, let us focus on the simple nearest neighbor heuristic. Here, we start from a path consisting of just a single city $c_0$ and at each iteration, we enlarge the path with a city that does not close a cycle and minimizes the cost of the enlargement. In particular, after $k$ iterations, we have a sequence $\{c_0, \ldots, c_k\}$ consisting of distinct cities, and at the next iteration, we add a new city $c_{k+1}$ that minimizes $a_{c_k c_{k+1}}$ over all cities $c_{k+1} \neq c_0, \ldots, c_k$. After the nearest neighbor heuristic selects city $c_N$, a complete tour is formed with total cost

$$a_{c_0 c_1} + \cdots + a_{c_{N-2} c_{N-1}} + a_{c_{N-1} c_0}. \quad (2.26)$$

We can formulate the traveling salesman problem as a DP problem as discussed in Example 1.3.1. We choose a starting city, say $c_0$, as the initial state $x_0$. Each state $x_k$ corresponds to a city sequence/partial tour $(c_0, c_1, \ldots, c_k)$ consisting of distinct cities. The state $x_{k+1}$, next to $x_k$, are sequences of the form $(c_0, c_1, \ldots, c_k, c_{k+1})$ which correspond to adding one more unvisited city $c_{k+1} \neq c_0, c_1, \ldots, c_k$. The terminal states $x_N$ are the complete tours of the form $(c_0, c_1, \ldots, c_{N-1}, c_0)$, and the cost of the corresponding sequence of city choices is the cost of the corresponding complete tour. Thus a state trajectory yields a complete tour and its total cost is the cost of the tour (2.26).

Let us now use as a base heuristic the nearest neighbor method. The corresponding rollout algorithm operates as follows: After $k < N$ iterations, we have a state $x_k$, i.e., sequence $\{c_0, \ldots, c_k\}$ consisting of distinct cities. At the next iteration, we add one more city as follows: We run the nearest neighbor heuristic starting from each of the sequences of the form $\{c_0, \ldots, c_k, c\}$ where $c \neq c_1, \ldots, c_k$. We then select as next city $c_{k+1}$ the city $c$ that yielded the minimum cost tour under the nearest neighbor heuristic.

Cost Improvement with a Rollout Algorithm - Sequential Consistency

The definition of the rollout algorithm leaves open the choice of the base heuristic. There are several types of suboptimal solution methods that can be used as base heuristics, such as greedy algorithms, local search, genetic algorithms, tabu search, and others. Clearly we want to choose a base heuristic that strikes a good balance between quality of solutions produced and computational tractability. However, some special conditions must hold in order to guarantee that the rollout policy has no worse performance than the base heuristic. We provide two such conditions, sequential consistency and sequential improvement, and then show how to modify the algorithm to deal with the case where these conditions are not satisfied.

We say that the base heuristic is sequentially consistent if it has the property that when it generates the sequence

$$\{x_k, x_{k+1}, \ldots, x_N\}$$
starting from state $x_k$, it also generates the sequence

$$\{x_{k+1}, \ldots, x_N\}$$

starting from state $x_{k+1}$. In other words, the base heuristic is sequentially consistent if it “stays the course”: when the starting state $x_k$ is moved forward to the next state $x_{k+1}$ of its state trajectory, the heuristic will not deviate from the remainder of the trajectory.

As an example, the reader may verify that the nearest neighbor heuristic described in the traveling salesman Example 2.4.1 is sequentially consistent. Similar examples include the use of many types of greedy heuristics (see [Ber17], Section 6.4). However, not all heuristics of interest are sequentially consistent.

Conceptually, it is important to note that sequential consistency is equivalent to the heuristic being a legitimate DP policy. By this we mean that there exists an admissible policy $\{\mu_0, \ldots, \mu_{N-1}\}$ such that the sequence generated by the base heuristic starting from any state $x_k$ is the same as the one generated by $\{\mu_0, \ldots, \mu_{N-1}\}$ starting from the same state $x_k$. To see this, note that a policy clearly has the sequential consistency property, and conversely, a sequentially consistent base heuristic defines a policy: the one that moves from $x_k$ to the state $x_{k+1}$ that lies on the path $\{x_k, x_{k+1}, \ldots, x_N\}$ generated by the base heuristic.

Based on this fact, we can show that the rollout algorithm obtained with a sequentially consistent base heuristic yields an improved cost over the base heuristic. In particular, let us consider the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, and let $J_{k,\tilde{\pi}}(x_k)$ denote the cost obtained with the base heuristic starting from state $x_k$. We claim that

$$J_{k,\tilde{\pi}}(x_k) \leq H_k(x_k), \quad \text{for all } x_k \text{ and } k,$$

where $H_k(x_k)$ denotes the cost of the heuristic starting from $x_k$.

We prove this inequality by induction. Clearly it holds for $k = N$, since $J_{N,\tilde{\pi}} = H_N = g_N$. Assume that it holds for index $k + 1$. For any state $x_k$, let $\overline{u}_k$ be the control applied by the base heuristic at $x_k$. Then we have

$$J_{k,\tilde{\pi}}(x_k) = g_k(x_k, \tilde{\mu}_k(x_k)) + J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{\mu}_k(x_k)))$$

$$\leq g_k(x_k, \tilde{\mu}_k(x_k)) + H_{k+1}(f_k(x_k, \tilde{\mu}_k(x_k)))$$

$$= \min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k)) \right]$$

$$\leq g_k(x_k, \overline{u}_k) + H_{k+1}(f_k(x_k, \overline{u}_k))$$

$$= H_k(x_k),$$

where:
(a) The first equality is the DP equation for the rollout policy \( \bar{\pi} \).

(b) The first inequality holds by the induction hypothesis.

(c) The second equality holds by the definition of the rollout algorithm.

(d) The third equality is the DP equation for the policy that corresponds to the base heuristic (this is the step where we need sequential consistency).

The induction proof of the cost improvement property (2.27) is thus complete.

**Sequential Improvement**

We will now show that the rollout policy has no worse performance than its base heuristic under a condition that is weaker than sequential consistency. Let us recall that the rollout algorithm \( \bar{\pi} = \{\bar{\mu}_0, \ldots, \bar{\mu}_{N-1}\} \) is defined by the minimization

\[
\bar{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k)
\]

where \( \tilde{Q}_k(x_k, u_k) \) is the approximate Q-factor defined by

\[
\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k)),
\]

[cf. Eqs. (2.25)], and \( H_{k+1}(x_{k+1}) \) denotes the cost of the base heuristic starting from state \( x_{k+1} \).

Let us assume that for all \( x_k \) and \( k \), we have

\[
\min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k)) \right] \leq H_k(x_k).
\]  \hspace{1cm} (2.29)

Then, from the calculation of Eq. (2.28), replacing the last two steps (that rely on sequential consistency) with Eq. (2.29), we have

\[
J_{k, \bar{\pi}}(x_k) \leq H_k(x_k), \quad \text{for all } x_k \text{ and } k.
\]

If the base heuristic has the property (2.29) we say that it is *sequentially improving*. Thus the rollout algorithm obtained with a sequentially improving base heuristic, will improve or at least will perform no worse than the base heuristic, for every starting state \( x_k \). Note that when the heuristic is sequentially consistent it is also sequentially improving, since in this case Eq. (2.29) is satisfied with equality.

Empirically, it has been observed that the cost improvement obtained by rollout is typically considerable and often dramatic. Generally, however, it is hard to provide solid theoretical support for this observation. The price for the performance improvement is extra computation that is typically
equal to the computation time of the base heuristic times a factor that is a low order polynomial of the problem size. Several case studies (see the end of chapter references) support this assessment. The textbook [Ber17], Section 6.4, provides some detailed worked-out examples.

On the other hand the sequential improvement condition may not hold for a given base heuristic. It is thus important to know that any heuristic can be made to be sequentially improving with a simple modification, as we explain next.

**The Fortified Rollout Algorithm**

We will describe a variant of the rollout algorithm that implicitly uses a sequentially improving base heuristic, so that it has the sequential improvement property (2.29). This variant, called the **fortified rollout algorithm**, starts at $x_0$, and generates step-by-step a sequence of states $\{x_0, x_1, \ldots, x_N\}$ and corresponding sequence of controls. Upon reaching state $x_k$ it stores the trajectory

$$\{x_0, u_0, \ldots, u_{k-1}, x_k\}$$

that has been constructed up to stage $k$, called *permanent* trajectory, and it also stores a *tentative* trajectory

$$\overline{T}_k = \{x_k, \overline{u}_k, \overline{x}_{k+1}, \overline{u}_{k+1}, \ldots, \overline{x}_{N-1}, \overline{x}_N\}$$

with corresponding cost

$$C(\overline{T}_k) = g_k(x_k, \overline{u}_k) + g_{k+1}(\overline{x}_{k+1}, \overline{u}_{k+1}) + \cdots + g_{N-1}(\overline{x}_{N-1}, \overline{u}_{N-1}) + g_N(\overline{x}_N).$$

The tentative trajectory is such that $\overline{T}_k \cup T_k$ is the best end-to-end trajectory computed up to stage $k$ of the algorithm. Initially, $\overline{T}_0$ is the trajectory generated by the base heuristic starting at the initial state $x_0$. The idea now is to deviate from the rollout algorithm at every state $x_k$ where the base heuristic produces a trajectory with larger cost than $T_k$, and use $\overline{T}_k$ instead.

In particular, upon reaching state $x_k$, we run the rollout algorithm as earlier, i.e., for every $u_k \in U_k(x_k)$ and next state $x_{k+1} = f_k(x_k, u_k)$, we run the base heuristic from $x_{k+1}$, and find the control $\tilde{u}_k$ that gives the best trajectory, denoted

$$\tilde{T}_k = \{x_k, \tilde{u}_k, \tilde{x}_{k+1}, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1}, \tilde{x}_N\}$$

with corresponding cost

$$C(\tilde{T}_k) = g_k(x_k, \tilde{u}_k) + g_{k+1}(\tilde{x}_{k+1}, \tilde{u}_{k+1}) + \cdots + g_{N-1}(\tilde{x}_{N-1}, \tilde{u}_{N-1}) + g_N(\tilde{x}_N).$$
Sec. 2.4 Rollout and Model Predictive Control

39

Factors
Current State
\( x_k \)

\( \hat{x}_{k+1} \)

\( \hat{x}_N \)

Heuristic
Permanent trajectory \( P_k \)

Tentative trajectory \( T_k \)

Figure 2.4.3 Schematic illustration of fortified rollout. After \( k \) steps, we have constructed the permanent trajectory

\[ P_k = \{ x_0, u_0, \ldots, u_{k-1}, x_k \} \]

and the tentative trajectory

\[ T_k = \{ x_k, \bar{u}_k, \bar{x}_{k+1}, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1}, \bar{x}_N \} \]

such that \( P_k \cup T_k \) is the best end-to-end trajectory computed so far. We now run the rollout algorithm at \( x_k \), i.e., we find the control \( \hat{u}_k \) that optimizes the sum of \( g_k(x_k, \bar{u}_k) \) plus the heuristic cost from the state \( \hat{x}_{k+1} = f_k(x_k, \hat{u}_k) \), and the corresponding trajectory

\[ \hat{T}_k = \{ x_k, \hat{u}_k, \hat{x}_{k+1}, \hat{u}_{k+1}, \ldots, \hat{u}_{N-1}, \hat{x}_N \} \]

If the cost of the end-to-end trajectory \( P_k \cup T_k \) is lower than the cost of \( P_k \cup \hat{T}_k \), we use add \((\bar{u}_k, \bar{x}_{k+1})\) to the permanent trajectory and set the tentative trajectory to

\[ T_{k+1} = \{ \bar{x}_{k+1}, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1}, \bar{x}_N \} \]

Otherwise we add \((\bar{u}_k, \bar{x}_{k+1})\) to the permanent trajectory and set the tentative trajectory to

\[ \hat{T}_{k+1} = \{ \hat{x}_{k+1}, \hat{u}_{k+1}, \ldots, \hat{u}_{N-1}, \hat{x}_N \} \]

Note that the fortified rollout will produce a different result than the ordinary rollout if the heuristic when started from \( x_{k+1} \) constructs a trajectory that is different than the tail portion of the tentative trajectory that starts at \( x_{k+1} \).

Whereas the ordinary rollout algorithm would choose control \( \hat{u}_k \) and move to \( \hat{x}_{k+1} \), the fortified algorithm compares \( C(\hat{T}_k) \) and \( C(\hat{T}_k) \), and depending on which of the two is smaller, chooses \( \bar{u}_k \) or \( \hat{u}_k \) and moves to \( \bar{x}_{k+1} \) or to \( \hat{x}_{k+1} \), respectively. In particular, if \( C(\hat{T}_k) \leq C(\hat{T}_k) \) the algorithm sets the next state and corresponding tentative trajectory to

\[ x_{k+1} = \bar{x}_{k+1}, \quad \hat{T}_{k+1} = \{ \bar{x}_{k+1}, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1}, \bar{x}_N \} \]

and if \( C(\hat{T}_k) > C(\hat{T}_k) \) it sets the next state and corresponding tentative
trajectory to

\[ x_{k+1} = \tilde{x}_{k+1}, \quad T_{k+1} = \{ \tilde{x}_{k+1}, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1}, \tilde{x}_N \}. \]

In other words the fortified rollout at \( x_k \) follows the current tentative trajectory \( T_k \) unless a lower cost trajectory \( \tilde{T}_k \) is discovered by running the base heuristic from all possible next states \( x_{k+1} \). It follows that at every state the trajectory that consists of the union of the permanent and the tentative trajectories, has lower cost than the initial tentative trajectory, which is the one produced by the base heuristic starting from \( x_0 \). Moreover, it can be seen that if the base heuristic is sequentially improving, the rollout algorithm and its fortified version coincide. Experimental evidence suggests that it is important to use the fortified version if the base heuristic is not sequentially improving.

Finally we note that the fortified rollout may be viewed as the ordinary rollout algorithm applied to a modified version of the original problem and modified base heuristic that has the sequential improvement property. The corresponding construction is somewhat tedious and will not be given; we refer to [BTW97] and [Ber17], Section 6.4.2.

Using Multiple Heuristics

In many problems, several promising heuristics may be available. It is then possible to use all of these heuristics in the rollout framework. The idea is to construct a superheuristic, which selects the best trajectory produced by all the base heuristic trajectories. The superheuristic can then be used as the base heuristic for a rollout algorithm.

In particular, let us assume that we have \( M \) base heuristics, and that the \( m \)th of these, given a state \( x_{k+1} \), produces a trajectory

\[ \tilde{T}_{k+1}^m = \{ x_{k+1}, \tilde{u}_{k+1}^m, \ldots, \tilde{u}_{N-1}^m, \tilde{x}_N^m \}, \]

and corresponding cost \( C(\tilde{T}_{k+1}^m) \). The superheuristic then produces at \( x_{k+1} \) the trajectory \( \tilde{T}_{k+1}^m \) for which \( C(\tilde{T}_{k+1}^m) \) is minimum.

An interesting property, which can be readily verified by using the definitions, is that if all the base heuristics are sequentially improving, the same is true for the superheuristic. Moreover, there is a fortified version of the rollout algorithm, which has the property that it produces a trajectory with no worse cost than all the trajectories produced by the \( M \) base heuristics when started at the initial state \( x_0 \).

Rollout Algorithms with Multistep Lookahead

We may incorporate multistep lookahead into the deterministic rollout framework. To describe two-step lookahead in its most straightforward
Sec. 2.4 Rollout and Model Predictive Control

Figure 2.4.4 Illustration of truncated rollout with two-step lookahead and a terminal cost function approximation $\tilde{J}$. The base heuristic is used for a limited number of steps and the terminal cost is added to compensate for the costs of the remaining steps.

implementation, suppose that after $k$ steps we have reached state $x_k$. We then consider the set of all two-step-ahead states $x_{k+2}$ we run the base heuristic starting from each of them, and compute the two-stage cost to get from $x_k$ to $x_{k+2}$, plus the cost of the base heuristic from $x_{k+2}$. We select the state, say $\tilde{x}_{k+2}$, that is associated with minimum cost, compute the controls $\tilde{u}_k$ and $\tilde{u}_{k+1}$ that lead from $x_k$ to $\tilde{x}_{k+2}$, and choose $\tilde{u}_k$ as the next rollout control and $x_{k+1} = f_k(x_k, \tilde{u}_k)$ as the next state.

The extension of the algorithm to lookahead of more than two steps is straightforward: instead of the two-step-ahead states $x_{k+2}$ we run the base heuristic starting from all the possible $\ell$-step ahead states $x_{k+\ell}$, etc. For problems with a large number of stages, we can consider truncated rollout with terminal cost approximation. Here the rollout trajectories are obtained by running the base heuristic from the leaf nodes of the lookahead tree, and they are truncated after a given number of steps, while a terminal cost approximation is added to the heuristic cost to compensate for the resulting error; see Fig. 2.4.4.

Among other variations of deterministic multistep rollout, let us mention a fortified version, which maintains a tentative trajectory, along the lines described earlier for the one-step lookahead case. In another version of $\ell$-step lookahead rollout, we may consider disregarding some of the states that are $\ell$ steps or less ahead, which are judged less promising accord-
Approximation in Value Space Chap. 2

Selective Depth Tree Projections of Leaves of the Tree

$\mathbf{\times} 0$

Current State

$\mathbf{\times} \mathbf{k}$

Factors

States

Base Heuristic

Figure 2.4.5 Illustration of deterministic rollout with selective depth lookahead.

After $k$ steps of the algorithm, we have a trajectory that states at the initial state $x_0$ and ends at state $x_k$. We then generate the set of all possible next states (states $x_{k+1}^1$, $x_{k+1}^2$, $x_{k+1}^3$, $x_{k+1}^4$ in the figure). We “evaluate” these states using the base heuristic, and select some of them for “expansion,” i.e., we generate their next states $x_{k+2}$, evaluate them using the base heuristic, and continue. In the end we have a selective depth tree of next states, and the base heuristic costs from the leaves of the tree. The state $x_{k+1}$ that corresponds to smallest overall cost is chosen by the selective depth lookahead rollout algorithm. For problems with a large number of stages, we can also truncate the rollout trajectories and add a terminal cost function approximation as compensation for the resulting error; cf. Fig. 2.4.4.

Finally, let us mention a variant of rollout that maintains multiple...
trajectories, extending from a given state $x_k$ to possibly multiple next states $x_{k+1}$. These states are the ones considered “most promising” based on the current results of the base heuristic (like being “$\epsilon$-best”), but may be discarded later based on subsequent computations. Such extended forms of rollout are restricted to deterministic problems, and tend to be problem-dependent. We will not consider them further in this book.

2.4.2 Stochastic Rollout and Monte Carlo Tree Search

We will now discuss the extension of the rollout algorithm to stochastic problems. We will restrict ourselves to the case where the base heuristic is a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ (i.e., is sequentially consistent, in the context of deterministic problems). It is possible to consider more general rollout algorithms that involve base heuristics with a sequential improvement property, but we will not pursue this idea, as it does not seem to have been applied so far in interesting stochastic contexts.

We first note that the cost improvement property that we showed for deterministic problems under the sequential consistency condition carries through for stochastic problems. In particular, let us denote by $J_k,\bar{\pi}(x_k)$ the cost corresponding to starting the base policy at state $x_k$, and by $J_k,\tilde{\pi}(x_k)$ the cost corresponding to starting the rollout algorithm at state $x_k$. We claim that

$$J_k,\tilde{\pi}(x_k) \leq J_k,\pi(x_k), \quad \text{for all } x_k \text{ and } k.$$

We prove this inequality by induction similar to the deterministic case. Clearly it holds for $k = N$, since $J_N,\tilde{\pi} = J_N,\pi = g_N$. Assuming that it holds for index $k + 1$, we have for all $x_k$,

$$J_k,\tilde{\pi}(x_k) = E\left\{g_k(x_k, \tilde{\mu}_k(x_k), w_k) + J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{\mu}_k(x_k), w_k))\right\}$$

$$\leq E\left\{g_k(x_k, \mu_k(x_k), w_k) + J_{k+1,\pi}(f_k(x_k, \mu_k(x_k), w_k))\right\}$$

$$= \min_{u_k \in U_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + J_{k+1,\pi}(f_k(x_k, u_k, w_k))\right\}$$

$$\leq E\left\{g_k(x_k, \mu_k(x_k), w_k) + J_{k+1,\pi}(f_k(x_k, u_k, w_k))\right\}$$

$$= J_k,\pi(x_k),$$

where:

(a) The first equality is the DP equation for the rollout policy $\tilde{\pi}$.
(b) The first inequality holds by the induction hypothesis.
(c) The second equality holds by the definition of the rollout algorithm.
(d) The third equality is the DP equation for the policy $\pi$ that corresponds to the base heuristic.
The induction proof of the cost improvement property (2.27) is thus complete.

Similar to deterministic problems, it has been observed empirically that for stochastic problems the rollout policy not only does not deteriorate the performance of the base policy, but also typically produces substantial cost improvement; see the case studies referenced at the end of the chapter.

Simulation-Based Implementation of the Rollout Algorithm

A conceptually straightforward way to compute the rollout control at a given state $x_k$ and time $k$ is to consider each possible control $u_k \in U_k(x_k)$ and to generate a “large” number of simulated trajectories of the system starting from $(x_k, u_k)$. Thus a simulated trajectory has the form

$$x_{i+1} = f_i(x_i, \mu_i(x_i), w_i), \quad i = k + 1, \ldots, N - 1,$$

where $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$ is the tail portion of the base policy, the first generated state is

$$x_{k+1} = f_k(x_k, u_k, w_k),$$

and the disturbance sequences $\{w_k, \ldots, w_{N-1}\}$ are independent random samples. The costs of the trajectories corresponding to a pair $(x_k, u_k)$ can be viewed as samples of the Q-factor

$$Q_k(x_k, u_k) = E\left\{g_k(x_k, u_k, w_k) + J_{k+1, \tilde{\pi}}(f_k(x_k, u_k, w_k))\right\},$$

where $J_{k+1, \tilde{\pi}}$ is the cost-to-go function of the base policy, i.e., $J_{k+1, \tilde{\pi}}(x_{k+1})$ is the cost of using the base policy starting from $x_{k+1}$. For problems with a large number of stages, it is also common to truncate the rollout trajectories and add a terminal cost function approximation as compensation for the resulting error.

By Monte Carlo averaging of the costs of the sample trajectories plus the terminal cost (if any), we obtain an approximation to the Q-factor $Q_k(x_k, u_k)$ for each control $u_k \in U_k(x_k)$, which we denote by $\hat{Q}_k(x_k, u_k)$. We then compute the (approximate) rollout control $\hat{\mu}_k(x_k)$ with the minimization

$$\hat{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \hat{Q}_k(x_k, u_k). \quad (2.30)$$

Example 2.4.2 (Backgammon)

The first impressive application of rollout was in the ancient two-player game of backgammon by Tesauro and Galperin [TeG96], see Fig. 2.4.6. They implemented a rollout algorithm, which attained a level of play that was better than all computer backgammon programs, and eventually better than the best humans. Tesauro had proposed earlier the use of neural networks as cost
Sec. 2.4 Rollout and Model Predictive Control

Figure 2.4.6 Illustration of rollout for backgammon. At a given position and roll of the dice, the set of all possible moves is generated, and the outcome of the game for each move is evaluated by “rolling out” (simulating to the end) many games using a suboptimal/heuristic backgammon player (the TD-Gammon player was used for this purpose in [TeG96]), and by Monte-Carlo averaging the scores. The move that results in the best average score is selected for play.

function approximators for backgammon, resulting in a program called TD-Gammon [Tes89a], [Tes89b], [Tes92], [Tes94], [Tes95], [Tes02]. TD-Gammon was trained with the use of the TD($\lambda$) algorithm that will be discussed in Section 4.9, and was used as the base heuristic (for both players) to simulate game trajectories. The rollout algorithm also involved multistep lookahead of a small number of stages, as well as truncation of long game trajectories, using a terminal cost function approximation based on TD-Gammon. Game trajectories are of course random, since they involve the use of dice at each player’s turn. Thus the scores of many trajectories have to be generated and averaged to assess the probability of a win from a given position.

An important issue to consider here is that backgammon is a two-player game and not an optimal control problem that involves a single decision maker. While there is a DP theory for sequential zero-sum games, this theory has not been covered in this book. Thus how are we to interpret rollout algorithms in the context of two-player games? The answer is to treat the two players unequally: one player uses the heuristic policy exclusively (TD-Gammon in the present example). The other player takes the role of the optimizer, and tries to improve on the heuristic policy (TD-Gammon) by using rollout. Thus “policy improvement” in the context of the present example means that when playing against a TD-Gammon opponent, the rollout player achieves a better score on the average than if he/she were to play with the TD-Gammon strategy. In particular, the theory does not guarantee that a rollout player that is trained using TD-Gammon for both players will do better than TD-Gammon would against a non-TD-Gammon opponent. This is a plausible hypothesis, albeit one that can only be tested experimentally.

At present, most computer backgammon programs descend from TD-Gammon. Rollout-based backgammon programs are the most powerful in terms of performance, consistent with the principle that a rollout algorithm performs better than its base heuristic. However, they are too time-consuming for real-time play at present, because of the extensive on-line simulation re-
requirement at each move (backgammon also has a high branching factor, i.e., for a given position, the number of possible successor positions is quite large, as compared for example with chess). They have been used in a limited diagnostic way to assess the quality of neural network-based programs (a large number of articles and empirical works on computer backgammon are posted on-line; see e.g., http://www.bkgm.com/articles/page07.html).

Monte Carlo Tree Search

In the rollout implementation just described, we implicitly assumed that once we reach state $x_k$, we generate the same large number of trajectories starting from each pair $(x_k, u_k)$, with $u_k \in U(x_k)$, to the end of the horizon. The drawback of this is threefold:

(a) The trajectories may be too long because the horizon length $N$ is large (or infinite, in an infinite horizon context).

(b) Some of the controls $u_k$ may be clearly inferior to others, and may not be worth as much sampling effort.

(c) Some of the controls $u_k$ that appear to be promising, may be worth exploring better through multistep lookahead.

This has motivated variants, generally referred to as Monte Carlo tree search (MCTS for short), which aim to trade off computational economy with a hopefully small risk of degradation in performance. Variants of this type involve, among others, early discarding of controls deemed to be inferior based on the results of preliminary calculations, and simulation that is limited in scope (either because of a reduced number of simulation samples, or because of a shortened horizon of simulation, or both); see Fig. 2.4.7.

In particular, a simple remedy for (a) above is to use rollout trajectories of reasonably limited length, with some terminal cost approximation at the end (in an extreme case, the rollout may be skipped altogether, i.e., rollout trajectories have zero length). The terminal cost function may be very simple (such as zero) or may be obtained through some auxiliary calculation. In fact the base policy used for rollout may be the one that provides the terminal cost function approximation, as noted for the rollout-based backgammon algorithm of Example 2.4.2.

A simple but less straightforward remedy for (b) is to use some heuristic or statistical test to discard some of the controls $u_k$, as soon as this is suggested by the early results of simulation. Similarly, to implement (c) one may use some heuristic to increase the length of lookahead selectively for some of the controls $u_k$. This is similar to the selective depth lookahead procedure for deterministic rollout that we illustrated in Fig. 2.4.5.

The MCTS approach can be based on sophisticated procedures for implementing and combining the ideas just described. The implementa-
Figure 2.4.7 Illustration of a tree used by MCTS at a state $x_k$. We assume for simplicity two available controls $u^1_k$ and $u^2_k$, and three values of disturbance at each stage. For some of the possible states $x_{k+1}$ we use an additional step lookahead and/or discard one of the two controls, thus generating the selective depth tree shown. We approximate the cost-to-go of the base policy from the leaf-states of the tree by generating a variable (perhaps adaptively determined) number of rollout trajectories, using the base policy, with possibly limited horizon, combined with terminal cost function approximation. Based on these results, we evaluate the approximate Q-factors corresponding to the two controls $u^1_k$ and $u^2_k$, and we select the one with smallest Q-factor.

Selection is often adapted to the problem at hand, but the general idea is to use the interim results of the computation and statistical (confidence interval) tests to focus the simulation effort along the most promising directions. Thus to implement MCTS one needs to maintain a lookahead tree, which is expanded by balancing the competing desires of exploitation and exploration (generate and evaluate controls that seem most promising versus assessing the potential of inadequately explored controls). Ideas that were developed in the context of multiarmed bandit problems have played an important role in the construction of this type of MCTS procedures (see the end-of-chapter references).

Example 2.4.3 (Statistical Tests for Adaptive Sampling with a One-Step Lookahead)
Let us consider the typical one-step lookahead selection by adaptive sampling. We are at a state $x_k$ and trying to find a control $\tilde{u}_k$ that minimizes an approximate Q-factor $\tilde{Q}_k(x_k, \tilde{u}_k)$ over $u_k \in U_k(x_k)$, by averaging samples of $\tilde{Q}_k(x_k, u_k)$. We assume that $U_k(x_k)$ contains $m$ elements, which for simplicity are denoted $1, \ldots, m$. At the $\ell$th sampling period, knowing the outcomes of the preceding sampling periods, we select one of the $m$ controls, say $i_\ell$, and we draw a sample of $\tilde{Q}_k(x_k, i_\ell)$, which is denoted $S_{i_\ell}$. Thus after the $n$th sampling period we have an estimate $Q_{i,n}$ of the Q-factor of each control $i = 1, \ldots, m$ that has been sampled at least once, given by

$$Q_{i,n} = \frac{\sum_{\ell=1}^{n} \delta(i_\ell = i)S_{i_\ell}}{\sum_{\ell=1}^{n} \delta(i_\ell = i)}$$

where

$$\delta(i_\ell = i) = \begin{cases} 1 & \text{if } i_\ell = i, \\ 0 & \text{if } i_\ell \neq i. \end{cases}$$

Thus $Q_{i,n}$ is the empirical mean of the Q-factor of control $i$ (total sample value divided by total number of samples), assuming that $i$ has been sampled at least once.

After $n$ samples have been collected, with each control sampled at least once, we may declare the control $i$ that minimizes $Q_{i,n}$ as the “best” one, i.e., the one that truly minimizes the Q-factor $Q_k(x_k, i)$. However, there is a positive probability that there is an error: the selected control may not minimize the true Q-factor. In adaptive sampling, roughly speaking, we want to design the sample selection strategy and the criterion to stop the sampling, in a way that keeps the probability of error small (by allocating some sampling effort to all controls), and the number of samples limited (by not wasting samples on controls $i$ that appear inferior based on their empirical mean $Q_{i,n}$).

Intuitively, a good sampling policy will balance at time $n$ the desires of exploitation and exploration (i.e., sample controls that seem most promising, in the sense that they have a small running average $Q_{i,n}$, versus assessing the potential of inadequately explored controls, those $i$ that have been sampled a small number of times). Thus it makes sense to sample at time $n$ the control $i$ that minimizes the sum $T_{i,n} + R_{i,n}$ of two indexes: an exploitation index $T_{i,n}$ and an exploration index $R_{i,n}$. Usually the exploitation index is chosen to be the empirical mean $Q_{i,n}$. The exploration index is based on a confidence interval formula and depends on the sample count $s_i = \sum_{\ell=1}^{n} \delta(i_\ell = i)$ of control $i$. A frequently suggested choice is the UCB rule (upper confidence bound), which sets $R_{i,n} = -c\sqrt{\log n/s_i}$, where $c = \sqrt{2}$ or some other positive constant that is selected empirically.

The UCB rule has been extensively discussed in the literature both for one-step and for multistep lookahead (where it is called UCT (UCB applied to trees; see [KoS16])). Its justification is based on probabilistic analyses that relate to the multiarmed bandit problem, and is beyond our scope.

Sampling policies for MCTS with multistep lookahead are based on similar sampling ideas to the case of one-step lookahead. A simulated
Sec. 2.4  Rollout and Model Predictive Control

A trajectory is run from a node \( i \) of the lookahead tree that minimizes the sum \( T_{i,n} + R_{i,n} \) of an exploitation index and an exploration index. There are many schemes of this type, but the details are beyond our scope (see the end-of-chapter references). A major success has been the use of MCTS in two-player game contexts, such as the AlphaGo computer program (Silver et al. [SHM16]), which performs better than the best humans in the game of Go. This program integrates several of the techniques discussed in this chapter, and in Chapters 3 and 4, including MCTS and rollout using a base policy that is trained off-line using a deep neural network. We will discuss neural network training techniques in Chapter 3. The AlphaZero program, which has performed spectacularly well against humans and other programs in the games of Go and chess (Silver et al. [SHS17]), bears some similarity with AlphaGo, and critically relies on MCTS, but does not use rollout.

Randomized Policy Improvement by MCTS

We have described rollout and MCTS as schemes for policy improvement: start with a base policy, and compute an improved policy based on the results of one-step lookahead or multistep lookahead followed by simulation with the base policy. We have implicitly assumed that both the base policy and the rollout policy are deterministic in the sense that they map each state \( x_k \) into a unique control \( \tilde{\mu}_k(x_k) \) [cf. Eq. (2.30)]. In some contexts, success has been achieved with randomized policies, which map a state \( x_k \) to a probability distribution over the set of controls \( U_k(x_k) \), rather than mapping onto a single control. In particular, the AlphaGo and AlphaZero programs use MCTS to generate and use for training purposes randomized policies, which specify at each board position the probabilities with which the various moves are selected.

The use of a randomized policy as a base policy is simple to understand: for a given state \( x_k \), we just generate sample trajectories and associated sample Q-factors, starting from each leaf-state of the lookahead tree that is rooted at \( x_k \). We then average the corresponding Q-factor samples. However, the rollout/improved policy, as we have described it, is a deterministic policy, i.e., it applies at \( x_k \) the control \( \tilde{\mu}_k(x_k) \) that is “best” according to the results of the rollout [cf. Eq. (2.30)]. Still, however, if we wish to generate an improved policy that is randomized, we can simply change the probabilities of different controls in the direction of the deterministic rollout policy. This can be done by increasing the probability of the “best” control \( \tilde{\mu}_k(x_k) \) from its base policy level, while decreasing the probabilities of the other controls.

The use of MCTS provides a related method to “improve” a randomized policy. In the process of adaptive simulation that is used in MCTS, we generate frequency counts of the different controls, i.e., the proportion of rollout trajectories associated each \( u_k \in U_k(x_k) \). We can then obtain the
rollout randomized policy by moving the probabilities of the base policy in the direction suggested by the frequency counts, i.e., increase the probability of high-count controls and reduce the probability of the others. This type of policy improvement is reminiscent of gradient-type methods, and has been successful in a number of applications; see the end-of-chapter literature such policy improvement implementations in AlphaGo, AlphaZero, and other application contexts.

### Variance Reduction in Rollout

When using simulation, sampling is often organized to effect variance reduction. By this we mean that for a given problem, the collection and use of samples is structured so that the variance of the simulation error is made smaller, with the same amount of simulation effort. There are several methods of this type for which we refer to textbooks on simulation (see, e.g., Ross [Ros12], Rubinstein and Kroese [RuK17]).

In this section we discuss a method to reduce the effects of the simulation error in the calculation of the Q-factors in the context of rollout. The key idea is that the selection of the rollout control depends on the values of the Q-factor differences

\[
\hat{Q}_k(x_k, u_k) - \hat{Q}_k(x_k, \hat{u}_k)
\]

for all pairs of controls \((u_k, \hat{u}_k)\). These values must be computed accurately, so that the controls \(u_k\) and \(\hat{u}_k\) can be accurately compared. On the other hand, the simulation/approximation errors in the computation of the individual Q-factors \(\hat{Q}_k(x_k, u_k)\) may be magnified through the preceding differencing operation.

An alternative approach is possible in the case where the probability distribution of each disturbance \(w_k\) does not depend on \(x_k\) and \(u_k\). In this case, we may approximate by simulation the Q-factor difference \(\hat{Q}_k(x_k, u_k) - \hat{Q}_k(x_k, \hat{u}_k)\) by sampling the difference

\[
C_k(x_k, u_k, w_k) - C_k(x_k, \hat{u}_k, w_k),
\]

where \(w_k = (w_k, w_{k+1}, \ldots, w_{N-1})\) and

\[
C_k(x_k, u_k, w_k) = g_N(x_N) + g_k(x_k, u_k, w_k) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i),
\]

where \(\{\mu_{k+1}, \ldots, \mu_{N-1}\}\) is the tail portion of the base policy.

This approximation may be far more accurate than the one obtained by differencing independent samples \(C_k(x_k, u_k, w_k)\) and \(C_k(x_k, \hat{u}_k, \hat{w}_k)\). Indeed, by introducing the zero mean sample errors

\[
D_k(x_k, u_k, w_k) = C_k(x_k, u_k, w_k) - \hat{Q}_k(x_k, u_k),
\]
it can be seen that the variance of the error in estimating \( \hat{Q}_k(x_k, u_k) - \hat{Q}_k(x_k, \hat{u}_k) \) with the former method will be smaller than with the latter method if and only if

\[
E_{w_k, \hat{w}_k} \left\{ \left| D_k(x_k, u_k, w_k) - D_k(x_k, \hat{u}_k, \hat{w}_k) \right|^2 \right\} > E_w \left\{ \left| D_k(x_k, u_k, w_k) - D_k(x_k, \hat{u}_k, \hat{w}_k) \right|^2 \right\},
\]

or equivalently

\[
E \left\{ D_k(x_k, u_k, w_k) D_k(x_k, \hat{u}_k, \hat{w}_k) \right\} > 0; \tag{2.31}
\]

i.e., if and only if the correlation between the errors \( D_k(x_k, u_k, w_k) \) and \( D_k(x_k, \hat{u}_k, \hat{w}_k) \) is positive. A little thought should convince the reader that this property is likely to hold in many types of problems. Roughly speaking, the relation (2.31) holds if changes in the value of \( u_k \) (at the first stage) have little effect on the value of the error \( D_k(x_k, u_k, w_k) \) relative to the effect induced by the randomness of \( w_k \). To see this, suppose that there exists a scalar \( \gamma < 1 \) such that, for all \( x_k, u_k, \) and \( \hat{u}_k \), there holds

\[
E \left\{ \left| D_k(x_k, u_k, w_k) - D_k(x_k, \hat{u}_k, \hat{w}_k) \right|^2 \right\} \leq \gamma E \left\{ \left| D_k(x_k, u_k, w_k) \right|^2 \right\}.
\]

(2.32)

Then we have

\[
D_k(x_k, u_k, w_k) D_k(x_k, \hat{u}_k, \hat{w}_k)
= \left| D_k(x_k, u_k, w_k) \right|^2
+ D_k(x_k, u_k, w_k) (D_k(x_k, \hat{u}_k, \hat{w}_k) - D_k(x_k, u_k, w_k))
\geq \left| D_k(x_k, u_k, w_k) \right|^2
- \left| D_k(x_k, u_k, w_k) \right| \cdot \left| D_k(x_k, \hat{u}_k, \hat{w}_k) - D_k(x_k, u_k, w_k) \right|
\]

from which we obtain

\[
E \left\{ D_k(x_k, u_k, w_k) D_k(x_k, \hat{u}_k, \hat{w}_k) \right\}
\geq E \left\{ \left| D_k(x_k, u_k, w_k) \right|^2 \right\}
- E \left\{ \left| D_k(x_k, u_k, w_k) \right| \cdot \left| D_k(x_k, \hat{u}_k, \hat{w}_k) - D_k(x_k, u_k, w_k) \right| \right\}
\geq E \left\{ \left| D_k(x_k, u_k, w_k) \right|^2 \right\} - \frac{1}{2} E \left\{ \left| D_k(x_k, u_k, w_k) \right|^2 \right\}
- \frac{1}{2} E \left\{ \left| D_k(x_k, \hat{u}_k, \hat{w}_k) - D_k(x_k, u_k, w_k) \right|^2 \right\}
\geq \frac{1 - \gamma}{2} E \left\{ \left| D_k(x_k, u_k, w_k) \right|^2 \right\},
\]
where for the first inequality we use the generic relation $ab \geq a^2 - |a| \cdot |b - a|$ for two scalars $a$ and $b$, for the second inequality we use the generic relation $|a| \cdot |b| \geq -\frac{1}{2}(a^2 + b^2)$ for two scalars $a$ and $b$, and for the third inequality we use Eq. (2.32).

Thus, under the assumption (2.32) and the assumption

$$E \left\{ |D_k(x_k, u_k, w_k)|^2 \right\} > 0,$$

the condition (2.31) holds and guarantees that by averaging cost difference samples rather than differencing (independently obtained) averages of cost samples, the simulation error variance decreases.

### 2.4.3 Model Predictive Control

In many control problems where the objective is to keep the state of a system near some desired point or trajectory, linear-quadratic models are not satisfactory. There are two main reasons for this:

(a) The system may be nonlinear, and using for control purposes a model that is linearized around the desired point or trajectory may be inappropriate.

(b) There may be control and/or state constraints, which are not handled adequately through a quadratic penalty on state and control. In particular, there may be special structure of the problem dictating that, for efficiency purposes, the system should often be operated at the boundary of its constraints. The solution obtained from a linear-quadratic model is not suitable for this, because it does not impose directly states and control constraints, and the quadratic penalty tends to “blur” the boundaries of the constraints.

These inadequacies of the linear-quadratic model have motivated a form of approximation method, called model predictive control (MPC), which combines elements of several ideas that we have discussed so far: certainty equivalence, multistep lookahead, and rollout algorithms. We will focus primarily on the most common form of MPC, where the system is either deterministic, or else it is stochastic, but it is replaced with a deterministic version by using typical values in place of all uncertain quantities, as in the certainty equivalent control approach. At each stage, a (deterministic) optimal control problem is solved over a fixed length horizon, starting from the current state. The first component of the corresponding optimal policy is then used as the control of the current stage, while the remaining components are discarded. The process is then repeated at the next stage, once the next state is revealed.

The primary objective in MPC, aside from fulfilling the state and control constraints of the problem, is to obtain a stable closed-loop system.
Note here that we may only be able to guarantee the stability of the deterministic model that forms the basis for the calculations of the MPC. This is consistent with a common practice in control theory: design a stable controller for a deterministic model of the system, and expect that it will provide some form of stability in a realistic stochastic environment as well.

To explain the mechanism by which stability is achieved by MPC, let us focus on the special case of a stationary, possibly nonlinear, deterministic system, where state and control belong to some Euclidean spaces. The methodology, however, can be extended to more general contexts (see the end-of-chapter references). The system is

$$x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \ldots,$$

and the cost per stage is quadratic:

$$x_k'Qx_k + u_k'Ru_k, \quad k = 0, 1, \ldots,$$

where $Q$ and $R$ are positive definite symmetric matrices, $x_k$ and $u_k$ are column vectors, and a prime denotes transposition. We impose state and control constraints

$$x_k \in X, \quad u_k \in U(x_k), \quad k = 0, 1, \ldots,$$

and we assume that the set $X$ contains the origin of the corresponding Euclidean space. Furthermore, we assume that if the system is at the origin, it can be kept there at no cost with control equal to 0, i.e., $0 \in U(0)$ and $f(0,0) = 0$. We want to derive a stationary feedback controller that applies control $\tilde{\mu}(x)$ at state $x$, and is such that, for all initial states $x_0 \in X$, the state of the closed-loop system

$$x_{k+1} = f(x_k, \tilde{\mu}(x_k)),$$

satisfies the state and control constraints, and the total cost over an infinite number of stages is finite:

$$\sum_{k=0}^{\infty} (x_k'Qx_k + \tilde{\mu}(x_k)'^R\tilde{\mu}(x_k)) < \infty.$$  \hspace{1cm} (2.33)

This finite cost condition guarantees that the feedback controller $\tilde{\mu}$ is stable in the sense that $x_k \to 0$ and $\tilde{\mu}(x_k) \to 0$ for all initial states $x_0 \in X$.

In order for such a controller to exist, it is evidently sufficient [in view of the assumption that $f(0,0) = 0$] that there exists an integer $m > 1$ such that for every initial state $x_0 \in X$, we can find a sequence of controls $u_k$, $k = 0, \ldots, m - 1$, which drive to 0 the state $x_m$ of the system at time $m$, while keeping all the preceding states $x_1, x_2, \ldots, x_{m-1}$ within $X$ and satisfying the control constraints $u_0 \in U(x_0), \ldots, u_{m-1} \in U(x_{m-1})$. We
refer to this as the \textit{constrained controllability assumption}. We will discuss how to check this assumption, and maintain its validity throughout the horizon in what follows.

Let us now describe a form of MPC under the preceding assumption. At each stage \( k \) and state \( x_k \in X \), it solves an \( m \)-stage deterministic optimal control problem involving the same quadratic cost and the requirement that the state after \( m \) stages is exactly equal to 0. This is the problem

\[
\min_{u_i, i = k, \ldots, k+m-1} \sum_{i=k}^{k+m-1} (x_i'Qx_i + u_i'Ru_i), \tag{2.34}
\]

subject to the system equation constraints

\[
x_{i+1} = f(x_i, u_i), \quad i = k, k+1, \ldots, k + m - 1,
\]

the state and control constraints

\[
x_i \in X, \quad u_i \in U(x_i), \quad i = k, k+1, \ldots, k + m - 1,
\]

and the terminal state constraint

\[
x_{k+m} = 0.
\]

By the constrained controllability assumption, this problem has a feasible solution. Let \( \{ \hat{u}_k, \hat{u}_{k+1}, \ldots, \hat{u}_{k+m-1} \} \) be a corresponding optimal control sequence. The MPC applies at stage \( k \) the first component of this sequence,

\[
\hat{\mu}(x_k) = \hat{u}_k,
\]

and discards the remaining components.

We can make the connection of MPC with rollout by noting that \textit{the one-step lookahead function} \( \hat{J} \) implicitly used by MPC [cf. Eq. (2.34)] \textit{is the cost-to-go function of a certain base policy}. This is the policy that drives to 0 the state after \( m - 1 \) stages (\textit{not} \( m \) stages) and keeps the state at 0 thereafter, while observing the state and control constraints, and minimizing the quadratic cost. Thus, we can also think of MPC as a rollout algorithm with the policy just described viewed as the base heuristic.

\textbf{Stability Analysis}

We now show that the MPC satisfies the stability condition (2.33). Let \( x_0, u_0, x_1, u_1, \ldots \) be the state and control sequence generated by MPC, so that

\[
u_k = \hat{\mu}(x_k), \quad x_{k+1} = f(x_k, \hat{\mu}(x_k)), \quad k = 0, 1, \ldots
\]
Denote by $\hat{J}(x)$ the optimal cost of the $m$-stage problem solved by MPC when at a state $x \in X$. Let also $\tilde{J}(x)$ be the optimal cost starting at $x$ of a corresponding $(m-1)$-stage problem, i.e., the optimal value of the quadratic cost
\[ \sum_{k=0}^{m-2} (x'_k Q x_k + u'_k R u_k), \]
where $x_0 = x$, subject to the constraints
\[ x_k \in X, \quad u_k \in U(x_k), \quad k = 0, \ldots, m-2, \]
and
\[ x_{m-1} = 0. \]

[For states $x \in X$ for which this problem does not have a feasible solution, we write $\tilde{J}(x) = \infty$.] Since having one less stage in our disposal to drive the state to 0 cannot decrease the optimal cost, we have for all $x \in X$
\[ \hat{J}(x) \leq \tilde{J}(x). \quad (2.35) \]

From the definitions of $\hat{J}$ and $\tilde{J}$, we have for all $k$,
\[ \min_{u \in U(x)} \left[ x'_k Q x_k + u' R u + \tilde{J}(f(x_k, u)) \right] = x'_k Q x_k + u'_k R u_k + \tilde{J}(x_{k+1}) = \tilde{J}(x_k), \]
so using Eq. (2.35), we obtain
\[ x'_k Q x_k + u'_k R u_k + \tilde{J}(x_{k+1}) \leq \tilde{J}(x_k), \quad k = 0, 1, \ldots \]

Adding this equation for all $k$ in a range $[0, K]$, where $K = 0, 1, \ldots$, we obtain
\[ \hat{J}(x_{K+1}) + \sum_{k=0}^{K} (x'_k Q x_k + u'_k R u_k) \leq \tilde{J}(x_0). \]

Since $\hat{J}(x_{K+1}) \geq 0$, it follows that
\[ \sum_{k=0}^{K} (x'_k Q x_k + u'_k R u_k) \leq \tilde{J}(x_0), \quad K = 0, 1, \ldots, \quad (2.36) \]
and taking the limit as $K \to \infty$,
\[ \sum_{k=0}^{\infty} (x'_k Q x_k + u'_k R u_k) \leq \tilde{J}(x_0) < \infty. \]

This shows the stability condition (2.33).
Example 2.4.4

Consider the scalar linear system
\[ x_{k+1} = x_k + u_k, \]
and the state and control constraints
\[ x_k \in X = \{ x \mid |x| \leq 1.5 \}, \quad u_k \in U(x_k) = \{ u \mid |u| \leq 1 \}. \]

Let also \( Q = R = 1 \). We select \( m = 2 \). For this value of \( m \), the constrained controllability assumption is satisfied, since the 2-step sequence of controls \( u_0 = -\text{sgn}(x_0) \), \( u_1 = -x_1 = -x_0 + \text{sgn}(x_0) \) drives the state \( x_2 \) to 0.

When at state \( x_k \in X \), the MPC minimizes the two-stage cost
\[ x_k^2 + u_k^2 + (x_k + u_k)^2 + u_{k+1}^2, \]
subject to the control constraints
\[ |u_k| \leq 1, \quad |u_{k+1}| \leq 1, \]
and the state constraints
\[ x_{k+1} \in X, \quad x_{k+2} = x_k + u_k + u_{k+1} = 0. \]

It is easily verified that this minimization yields \( u_{k+1} = -(x_k + u_k) \) and \( u_k = -(2/3)x_k \). Thus the MPC takes the form
\[ \mu(x_k) = -\frac{2}{3}x_k, \]
and the closed-loop system is
\[ x_{k+1} = \frac{1}{3}x_k, \quad k = 0, 1, \ldots \]

Note that while the closed-loop system with this MPC is stable, its state is never driven to 0 if started from \( x_0 \neq 0 \).

Target Tube Reachability and Constrained Controllability

We now return to the constrained controllability condition, which asserts that the state constraint set \( X \) is such that starting from anywhere within it, it is possible to drive to 0 the state of the system within some number of steps \( m \), while staying within \( X \) at each intermediate step. Unfortunately, this assumption is flawed and masks major complications. In particular, the control constraint set may not be sufficiently rich to compensate for natural instability tendencies of the system. As a result it may be impossible to keep the state within \( X \) over a sufficiently long period of time. Here is an example.
Example 2.4.5

Consider the scalar linear system
\[ x_{k+1} = 2x_k + u_k, \]
which is unstable, and the control constraint
\[ |u_k| \leq 1. \]
Then if \( 0 \leq x_0 < 1, \) it can be seen that by using the control \( u_0 = -1, \) the next state satisfies,
\[ x_1 = 2x_0 - 1 < x_0, \]
and is closer to 0 than the preceding state \( x_0. \) Similarly, using controls \( u_k = -1, \) every subsequent state \( x_{k+1} \) will get closer to 0 that \( x_k. \) Eventually, after a sufficient number of steps \( \bar{k}, \) with controls \( u_k = -1 \) for \( k < \bar{k}, \) the state \( x_{\bar{k}} \) will satisfy
\[ 0 \leq x_{\bar{k}} \leq \frac{1}{2}. \]
Once this happens, the feasible control \( u_{\bar{k}} = -2x_{\bar{k}} \) will drive the state \( x_{\bar{k}+1} \) to 0.
Similarly, when \( -1 < x_0 \leq 0, \) by applying control \( u_k = 1 \) for a sufficiently large number of steps \( \bar{k}, \) the state \( x_{\bar{k}} \) will be driven into the region \((-1/2, 0], \) and then the feasible control \( u_{\bar{k}} = -2x_{\bar{k}} \) will drive the state \( x_{\bar{k}+1} \) to 0.
Suppose now that the control constraint is \( |u_k| \leq 1 \) and the state constraint is of the form \( X = [-\beta, \beta]. \) Then the preceding discussion shows that if \( 0 < \beta < 1 \) the constrained controllability assumption is satisfied, and for every state initial state \( x_0 \in X \) the state can be kept within \( X \) and can be driven to 0 in a finite number \( m \) of steps. The number \( m \) depends on \( \beta, \) and in particular of \( 0 < \beta < 1/2, m \) can be taken equal to 1. On the other hand, if \( \beta \geq 1, \) it is impossible to drive the state to 0 from every initial state \( x_0 \in [1, \beta] \) without violating the constraint \( |u_k| \leq 1, \) and the constrained controllability assumption is violated.

The critical construction in the preceding example is to identify a set of states \( X \) such that starting from within \( X \) we are guaranteed to stay within the “tube” \( \{X, X, \ldots\} \) for all subsequent times with appropriate choices of control. Such sets are called infinitely reachable and can serve as state constraints in MPC. Formally, a set \( X \) is infinitely reachable if it has the property that for every \( x \in X \) there exists a \( u \in U(x) \) such that \( f(x, u) \in X. \) If \( X \) is not infinitely reachable, the constrained controlla-
Approximation in Value Space Chap. 2

...probability condition cannot be satisfied, since then there will be states within $X$ starting from which we can never reenter the tube $\{X, X, \ldots\}$. Thus ensuring infinite time reachability of $X$ is a prerequisite towards satisfying the constrained controllability assumption, and serves as the first step in the analysis and design of MPC schemes with state constraints.

Regarding the choice of the horizon length $m$ for the MPC calculations, note that if the constrained controllability assumption is satisfied for some value of $m$, it is also satisfied for all larger values of $m$. Furthermore, it can be seen that the $m$-stage cost $\hat{J}(x)$, which by Eq. (2.36), is an upper bound to the cost of MPC, cannot increase with $m$. This argues for a larger value of $m$. On the other hand, the optimal control problem solved at each stage by the MPC becomes larger and hence more difficult as $m$ increases. Thus, the horizon length is usually chosen on the basis of some experimentation: first ensure that $m$ is large enough for the constrained controllability assumption to hold, and then by further experimentation to ensure overall satisfactory performance.

Variants of MPC

The MPC scheme that we have described is just the starting point for a broad methodology with many variations, which often relate to the suboptimal control methods that we have discussed so far in this chapter. For example, in the problem solved by MPC at each stage, instead of the requirement of driving the system state to $0$ in $m$ steps, one may use a large penalty for the state being nonzero after $m$ steps. Then, the preceding analysis goes through, as long as the terminal penalty is chosen so that Eq. (2.35) is satisfied.

In another variant one may use a nonquadratic cost function, which is everywhere positive except at $(x, u) = (0, 0)$. In still another variant, instead of aiming to drive the state to $0$ after $m$ steps, one aims to reach a sufficiently small neighborhood of the origin, within which a stabilizing controller, designed by other methods, may be used.

We finally mention variants of MPC methods, which combine rollout and terminal cost function approximation, and which can deal with uncertainty and system disturbances. A method of this type will be described in Section 4.6.1 in the context of infinite horizon problems, but can be adapted to finite horizon problems as well; see also the end-of-chapter references.

As an illustration, let us provide an example of a scheme that combines MPC with certainty equivalent control ideas (cf. Section 2.3.2).

**Example 2.4.6 (MPC for Stochastic Systems and Certainty Equivalent Approximation)**

Consider the stationary stochastic system

$$x_{k+1} = f(x_k, u_k, w_k),$$

...
with the expected cost of a policy \( \pi = \{ \mu_0, \ldots, \mu_{N-1} \} \) starting at \( x_0 \) defined by

\[
J_\pi(x_0) = E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g(x_k, \mu_k(x_k), w_k) \right\} ;
\]

cf. the framework of Section 1.2. We assume that for all \( k \), there are state and control constraints of the form

\[
x_k \in X, \quad u_k \in U,
\]

and that the stochastic disturbances \( w_k \) take values within some known set \( W \).

An important characteristic of this problem is that a policy must maintain reachability of the target tube \( \{ X, X, \ldots \} \), even under worst-case disturbance values. For this it is necessary that for each state \( x_k \in X \), the control \( u_k \) is chosen from within the subset \( \tilde{U}(x_k) \) given by

\[
\tilde{U}(x_k) = \{ u_k \in U \mid f(x_k, u_k, w_k) \in X, \text{ for all } w_k \in W \}.
\]

We assume that \( \tilde{U}(x_k) \) is nonempty for all \( x_k \in X \). This property is not automatically satisfied; similar to the deterministic case discussed earlier, the target tube \( \{ X, X, \ldots \} \) must be properly constructed using infinite time reachability methods, and \( U \) must be sufficiently “rich” to ensure that this is possible.

We will now describe a rollout/MPC method that generalizes the one given earlier for deterministic problems. It satisfies the state and control constraints, and uses assumed certainty equivalence to define the base heuristic over \( m - 1 \) steps, where \( m > 1 \) is some integer. In particular, at a given state \( x_k \in X \), this method:

(a) Generates, for every \( u_k \in \tilde{U}(x_k) \), all the possible next states

\[
x_{k+1} = f(x_k, u_k, w_k), \quad w_k \in W.
\]

Then for each of these states, it solves the problem of optimally driving the state \( x_{k+m} \) to 0 after \( m - 1 \) steps, starting from \( x_{k+1} \) and assuming that the future disturbances \( w_{k+1}, \ldots, w_{k+m-1} \) take some known typical values.

(b) Applies the control \( \tilde{u}_k \) that minimizes over \( u_k \in \tilde{U}(x_k) \) the Q-factor

\[
\tilde{Q}(x_k, u_k) = E \left\{ g(x_k, u_k, w_k) + \tilde{J}(f(x_k, u_k, w_k)) \right\} , \quad (2.37)
\]

where \( \tilde{J}(x_{k+1}) \) is the optimal cost of the transfer from \( x_{k+1} \) to 0 of part (a).

Here we require a constrained controllability condition that guarantees that the transfer of part (a) from \( x_{k+1} \) to 0 is possible with controls from the original constraint set \( U \). The control constraint set \( U \) must be sufficiently
rich to ensure this, and also to ensure that the sets $\tilde{U}(x_k)$ are nonempty for all $x_k \in X$.

A drawback of the preceding MPC method is that it may not be well suited for on-line implementation because of the substantial amount of computation required at each state $x_k$. An alternative is to generate a large number of sample states $x^s_k$, $s = 1, \ldots, q$, and to calculate the corresponding controls $u^s_k$ using the Q-factor minimization

$$u^*_k \in \arg \min_{u_k \in \tilde{U}(x^*_k)} E \left\{ g(x^*_k, u_k, w_k) + \tilde{J}(f(x^*_k, u_k, w_k)) \right\},$$

[cf. Eq. (2.37)]. We can then use the pairs $(x^s_k, u^s_k)$, $s = 1, \ldots, q$, and some form of regression to train a Q-factor parametric architecture $\tilde{Q}(x_k, u_k, \tilde{r}_k)$ such as a neural network [cf. the approximation in policy space approach of Eq. (2.6)]. Once this is done, the MPC controls can be generated on-line using the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in \tilde{U}(x_k)} \tilde{Q}(x_k, u_k, \tilde{r}_k);$$

cf. Eq. (2.7).

2.5 NOTES AND SOURCES

Approximation in value space has been considered in an ad hoc manner since the early days of DP, motivated by the curse of dimensionality. The idea was reframed and coupled with model-free simulation methods that originated in the late 1980s in artificial intelligence. Since that time, approximation in value space has been one of the two pillars of approximate DP/RL. The other pillar is approximation in policy space, which was discussed briefly in Section 2.1.4 and will be revisited in Section 4.11.

The problem approximation approach has a long history in optimal control and operations research. The author’s paper [Ber07] describes a few enforced decomposition schemes based on constraint relaxation. Constraint decoupling in multiarmed bandit problems using Lagrangian relaxation was proposed by Whittle [Whi88].

The main idea of rollout algorithms, obtaining an improved policy starting from some other suboptimal policy, has appeared in several DP application contexts. The name “rollout” was coined by Tesauro in specific reference to rolling the dice in the game of backgammon [TeG96]. In Tesauro’s proposal, a given backgammon position is evaluated by “rolling out” many games starting from that position, using a simulator, and the results are averaged to provide a “score” for the position; see Example 2.4.2. The use of the name “rollout” has gradually expanded beyond its original context; for example the samples collected through simulated trajectories are referred to as “rollouts” by some authors.
The application of rollout algorithms to discrete deterministic optimization problems, and the notions of sequential consistency and sequential improvement were given in the paper by Bertsekas, Tsitsiklis, and Wu [BTW97], and also in the neuro-dynamic book by Bertsekas and Tsitsiklis [BeT96]. It was further formalized in Bertsekas [Ber97], and Bertsekas and Castanon [BeC99]. A discussion of rollout algorithms as applied to network optimization problems may be found in the author’s textbook [Ber98].

For more recent work on rollout algorithms and related methods, see Secomandi [Sec00], [Sec01], [Sec03], Ferris and Voelker [FeV02], [FeV04], McGovern, Moss, and Barto [MMB02], Savagaonkar, Givan, and Chong [SGC02], Bertsimas and Popescu [BeP03], Guerriero and Mancini [GuM03], Tu and Pattipati [TuP03], Wu, Chong, and Givan [WCG03], Chang, Givan, and Chong [CGC04], Meloni, Pacciarelli, and Pranzo [MPP04], Yan, Diaconis, Rusmevichientong, and Van Roy [YDR04], Besse and Chaib-draa [BCC08], Sun et al. [SZL08], Bertazzi et al. [BBG13], Sun et al. [SLJ13], Tesauro et al. [TGL13], Beyne and Leung [BeL14], Goodson, Thomas, and Ohlmann [GTO15], Li and Womer [LiW15], Mastin and Jaillet [Maj15], Huang, Jia, and Guan [HJJ16], Simroth, Holfeld, and Brunsch [SHB15], and Lan, Guan, and Wu [LGW16]. For a recent survey by the author, see [Ber13b]. These works discuss a broad variety of applications and case studies, and generally report positive computational experience.

The idea of rollout that uses limited lookahead, adaptive pruning of the lookahead tree, and cost function approximation at the end of the lookahead horizon was suggested by Tesauro and Galperin [TeG96] in the context of backgammon. Related ideas appeared earlier in the paper by Abramson [Abr90], in a game playing context. The paper and monograph by Chang, Fu, and Marcus [CFH05], [CFH13] proposed and analyzed adaptive sampling in connection with DP, including statistical tests to control the sampling process. The name “Monte Carlo tree search” (Section 6.5.1) has become popular, and in its current use, it encompasses a broad range of methods that involve adaptive sampling, rollout, extensions to sequential games, and the use and analysis of various statistical tests. We refer to the papers by Coulom [Cou06], the survey by Browne et al. [BPW12], and the discussion by Fu [Fu17]. The development of statistical tests for adaptive sampling has been influenced by works on multiarmed bandit problems; see the papers by Lai and Robbins [LaR85], Agrawal [Agr95], Burnetas and Katehakis [BuK97], Meuleau and Bourgine [MB99], Auer, Cesa-Bianchi, and Fischer [ACF02], Peret and Garcia [PeG04], Kocsis and Szepesvari [KoS06], and the monograph by Munos [Mun14]. The technique for variance reduction in the calculation of Q-factor differences (Section 6.5.2) was given in the author’s paper [Ber97].

The model predictive control approach is popular in a variety of control system design contexts, and particularly in chemical process control, where meeting explicit control and state constraints is an important practical issue. The connection of MPC with rollout algorithms was made in
the author’s paper [Ber05a]. The stability analysis given here is based on
the work of Keerthi and Gilbert [KeG88]. For an early survey of the field,
which gives many of the early references, see Morari and Lee [MoL99], and
for a more recent survey see Mayne [May14]. For related textbooks, see
Maciejowski [Mac02], Camacho and Bordons [CaB04], Kouvaritakis and
Cannon [KoC15], and Borelli, Bemporad, and Morari [BBM17].

In our account of model predictive control, we have restricted ou-
ourselves to deterministic problems possibly involving tight state constraints
as well as control constraints. Problems with stochastic uncertainty and
state constraints are more challenging because of the difficulty of guaran-
teeing that the constraints are satisfied; see the survey by Mayne [May14]
for a review of various approaches that have been used in this context.
The textbook [Ber17], Section 6.4, describes model predictive control for
problems with set membership uncertainty and state constraints, using
the concept of a target tube, which originated in the author’s PhD thesis
[Ber71], [Ber72a], [BeR71a], [BeR71b], [BeR71c], [BeR73]. Target tubes
were used subsequently in MPC and other contexts by several authors; see
the surveys by Blanchini [Bla99] and Mayne [May14].