Chapter 11

Introduction to Approximate DP and Model Predictive Control

[1]In this chapter, we start our investigation of approximation methods or "sub-optimal control". When DP is used as a computational technique (i.e., not to study theoretical properties of the optimal policy as in inventory control or linear quadratic control), we encounter in most application what Bellman called the "curse of dimensionality". DP is an efficient way to solve an optimal control problem, but the curse says that the modeling step leads to state-space explosion, i.e., a number of states that grows exponentially with the parameters of the problem. For example, suppose the state space is $\mathsf{X} = \mathbb{R}^d$. To apply the dynamic programming over a finite horizon, or value iteration for an infinite-horizon problem, some form of discretization is necessary. Suppose we quantize real values to represent them by only n values. Then the number of states in the problem is n^d, which grows exponentially with the dimension of X. For another example, consider a small queueing network (e.g. a communication network, or a production line) with 10 queues. The service rates at the queues, and the routing between the queues can be controller, and such problems can be formulated as optimal control problems. But assume say that all queues have a finite buffer which can contain at most 20 jobs. The number of states for this problem is then $20^{10} \approx 10^{13}$, and the model is useless for a direct optimal control approach.

One important point that I would like to emphasize is that modeling large-scale systems for control must usually be done differently than modeling for simulation or analysis. For example, if a control policy is fixed, one can usually simulate large queueing systems more or less directly using discrete-event simulators (see e.g. `ns-2` - here I'm brushing aside the difficulty of simulating certain stochastic systems, but let's say that it's still much easier than controlling them). Consider also the exploding research on simulation and inference for large-scale networks in physics, biology, the social sciences, or for the electric grid. For control, much simpler models than those encountered in

[1]This version: October 11 2009.

these fields are generally needed to allow computations. And developing the right models that capture the essential behavior of the system without adding extra complexity is a challenge in itself. The reward is that using the control laws developed for such good simplified models can usually provide enormous benefits for the real-world system as well. Currently we have some ideas on how to develop these simplified models for certain categories of problems. For example, we can use model-order reduction for linear state-space systems, aggregation for large Markov chains, and fluid (and Brownian) models (mostly used for queueing networks). This chapter does not consider model simplification techniques, as does most of Bertsekas' book (aggregation is mentioned on p.319). Instead, we focus on approximation methods for the control of a complex model. But it is good to keep the remarks above in mind for your research, because there is not one approach that is clearly preferable to an other in terms of performance, and good approximate models usually offer more insight into the control problem.

In this chapter we consider certainty equivalent control, rollout policies, and model predictive control (MPC). Some references are [Ber07, chapter 6] and Stephen Boyd's EE364II 2008 lectures 15 and 16 [Boyb] on fast MPC.

11.1 Certainty Equivalent and Open-Loop Feedback Control

CEC: fixing process disturbances

The certainty equivalent controller (CEC) is a heuristic controller that applies the certainty equivalence and separation principles of linear quadratic control theory, even if these do not formally hold for the problem. For a finite horizon problem with horizon N, and possibly partial informations, the heuristic can follow the following recipe at stage k:

1. Given the information vector $\mathcal{I}_k$, compute an estimate $\hat{x}_k(\mathcal{I}_k)$, for example the conditional mean estimate $\hat{x}_k(\mathcal{I}_k) = E[x_k|\mathcal{I}_k]$.

2. Fix the process disturbances $\{w_j\}_{j \geq k}$ at some typical deterministic value, for example their mean $\bar{w}_j(x_j, u_j) = E[w_k|x_j, u_j]$. Hence if we know that w_j are i.i.d. and zero mean, we could just fix them to 0 below.

3. Solve the *deterministic* (perfect information) optimal control problem

$$\min \;\; c_N(x_N) + \sum_{j=k}^{N-1} c_j(x_j, u_j, \bar{w}_j(x_j, u_j)),$$

   with the initial condition $x_k = \hat{x}_k(\mathcal{I}_k)$ and the constraints $u_j \in \mathsf{U}_j$, $x_{j+1} = f_j(x_j, u_j, \bar{w}_j(x_j, u_j))$. Note that because this is a deterministic control problem, optimization (over open-loop policies) can be used, see chapter 2.

4. Use only the first element $\bar{u}_k = \bar{\mu}_k(\mathcal{I}_k)$ in the control sequence found. We land in a new state x_{k+1} make a new observation y_{k+1} and now repeat from step 1.

Due to the equivalence between open-loop and closed-loop policies for deterministic problems in step 3, we have two ways of computing the CEC, one offline and one online method. In the offline method, we solve once the full horizon deterministic optimal control, using a method such as DP that provides a *feedback* policy, denoted $\pi^d = \{\mu_0^d, \dots, \mu_{N-1}^d\}$

$$\min \; c_N(x_N) + \sum_{j=0}^{N-1} c_j(x_j, \mu_j(x_j), \bar{w}_j(x_j, u_j)) \tag{11.1}$$

$$\text{subject to} \quad \mu_j(x_j) \in \mathsf{U}_j,\; x_{j+1} = f_j(x_j, \mu_j(x_j), \bar{w}_j(x_j, u_j)),\; j \geq 0. \tag{11.2}$$

Then we store the functions $\mu_0^d, \dots \mu_{N-1}^d$. In step 3, there is no need to solve any optimization problem any more, so we can skip this step, and we just apply $\bar{\mu}_k(\mathcal{I}_k) = \mu_k^d(\hat{x}_k(\mathcal{I}_k))$ in step 4. In the online approach, we do not compute the feedback policy, and we use optimization in step 3. Note that this requires solving a new optimization problem at every stage, for a new initial condition $x_k = \hat{x}_k(\mathcal{I}_k)$ (in fact, among other obvious variants, you could use more that one control input before reoptimizing). If the problem has some interesting structure, e.g. if it is convex (convex stage cost functions, linear dynamics), there are powerful methods for doing this. Note that linear constraints of the form $x_{k+1} - A_k x_k - B_k u_k = \bar{w}_k(x_k, u_k)$ correspond to a *sparse and banded* constraint matrix in the optimization problem, and this can be exploited in numerical methods. Moreover, we also want to take advantage of the fact that the optimization problems at successive stages are quite similar (see the *warm start* methods in optimization).

Note that the CEC is optimal for linear quadratic problems, as shown in chapters 4 and 5. Often, it performs well. However, it is possible to construct examples where the CEC performs strictly worse than the optimal open-loop controller (i.e., which does not observe the trajectory and never recompute the controls - see problem B.3.4) !

Open-loop feedback control

In the CEC, since we fix the process disturbances and consider a perfect information problem in step 3, we can equivalently solve over open-loop or closed loop policies. A variant, called Open-Loop Feedback Control (OLFC), does not fix the process disturbances, but still solves at each stage an open-loop control problem, which is now more complicated than step 3 for CEC because it involves the computation of some expected values. Let us assume that the observation noise v_k only depends on x_k, u_k, and w_k, so that $P_{x_k|\mathcal{I}_k}$ is a sufficient statistic for the partial information problem (see section 5.1). The OLFC proceeds as follows at stage k:

1. Given $\mathcal{I}_k$, compute $P_{x_k|\mathcal{I}_k}$ (skip this step for a perfect information problem). In general, this step can be difficult, and some form of approximation might be necessary.

2. Compute an optimal *open-loop* control sequence $\{\bar{u}_k, \ldots, \bar{u}_{N-1}\}$ for the problem

$$\begin{aligned} \min \; & E\Big[c_N(x_N) + \sum_{j=k}^{N-1} c_j(x_j, u_j, w_j)\Big|\mathcal{I}_k\Big] \\ \text{s.t. } & x_{j+1} = f_j(x_j, u_j, w_j), \; u_j \in \mathsf{U}_j, \; j \geq k. \end{aligned}$$

   Note here that in contrast to the CEC, because of the stochastic disturbances $\{w_j\}_{j \geq k}$ there are in general closed-loop policies that would perform better than the sequence $\bar{u}_{k:N-1}$. Also the optimization problem here is a stochastic optimization problem, which is in general harder to deal with than the CEC optimization problem with fixed disturbances. Computing the conditional expectations appearing in this optimization problem makes use of $P_{x_k|\mathcal{I}_k}$. The simplification in the OLFC comes from the fact that we ignore the future observations in the computation of $\bar{u}_{k:N-1}$.

3. As in the CEC, apply only the first input $\bar{\mu}_k(\mathcal{I}_k) = \bar{u}_k$. Then go back to step 1, with the new control $\bar{u}_k$ and observation y_{k+1} included in the information vector $\mathcal{I}_{k+1}$.

Although the OLFC is harder to compute than the CEC, it does always perform at least as well than the pure open-loop controller. So the OLFC controller uses the measurements advantageously.

Proposition 11.1.1. *The cost $J_{\bar{\pi}}$ of an OLFC and J_{ol} of an optimal open-loop policy satisfy $J_{\bar{\pi}} \leq J_{ol}$.*

This proposition is proved in [Ber07, proposition 6.2.1]. It's good to know that the open-loop cost provides a bound on the OLFC cost, but since the open-loop control law is often very bad for control problems with disturbances (see problem B.1.7), I don't think that the result is terribly exciting in general. You can read about more variants of CEC and OLFC in [Ber07, sections 6.1 and 6.2].

11.2 Limited Lookahead and Rollout Policies

Limited lookahead and rollout policies, as well as many of the approximation techniques we will encounter later on, rely on an *approximation $\tilde{J}_k(\cdot)$ of the cost-to-go in the DP algorithm.* Different algorithms differ essentially by the way the approximation is obtained. For a perfect information problem, the *one-step lookahead policy* takes at stage k and in state x_k the control $\bar{\mu}_k(x_k)$ which attains the minimum in

$$\min_{u_k \in \mathsf{U}_k(x_k)} E\Big[c_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\Big]. \tag{11.3}$$

That is, we just replace the optimal cost-to-go J^*_{k+1} by its approximation $\tilde{J}_{k+1}$ in the DP recursion step, with in addition $\tilde{J}_N = c_N$. Similarly, we can consider a *two-step lookahead policy*, where $\tilde{J}_{k+1}$ in (11.3) itself is obtained using a one-step lookahead

$$\begin{aligned}\tilde{J}_{k+1}(x_{k+1}) = \min_{u_{k+1} \in \mathsf{U}_{k+1}(x_{k+1})} E\Big[&c_{k+1}(x_{k+1}, u_{k+1}, w_{k+1}) \\ &+ \breve{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1}, w_{k+1}))\Big],\end{aligned}$$

and $\breve{J}_{k+2}$ is some approximation of the optimal cost-to-go J^*_{k+2}. You can define similarly *k-step lookahead policies.* Note that you can apply this method to infinite horizon as well, where $\tilde{J}(x_{k+1})$ in the one-step lookahead policy just represents an approximation of the optimal infinite-horizon cost starting at state x_{k+1}. If the minimization problem over $\mathsf{U}_k(x_k)$ is too hard, say for the one-step lookahead policy, we can also consider only a promising set of controls $\overline{\mathsf{U}}_k(x_k) \subset \mathsf{U}_k(x_k)$

$$\bar{\mu}_k(x_k) \in \arg \min_{u_k \in \overline{\mathsf{U}}_k(x_k)} E\Big[c_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\Big]. \tag{11.4}$$

Even more generally, the minimization over $\overline{\mathsf{U}}_k(x_k)$ could be done approximately. Consider the following proposition.

Proposition 11.2.1. *Let $\pi = \{\bar{\mu}_0, \bar{\mu}_1, \ldots, \bar{\mu}_{N-1}\}$ be a policy such that for all k and x_k we have*

$$E\Big[c_k(x_k, \bar{\mu}_k(x_k), w_k) + \tilde{J}_{k+1}(f_k(x_k, \bar{\mu}_k(x_k), w_k))\Big] \leq \tilde{J}_k(x_k) + \delta_k, \tag{11.5}$$

for some scalars $\delta_0, \ldots, \delta_{N-1}$. Then for all x_k and k, we have

$$J_{\pi,k}(x_k) \leq \tilde{J}_k(x_k) + \sum_{j=k}^{N-1} \delta_j. \tag{11.6}$$

Note that in this proposition $\hat{J}_k$, just like $\tilde{J}_k$, need not represent the cost of an actual policy of the problem (we have left the question of how to obtain $\tilde{J}_k$ completely open at this point). The inequalities (11.6) provide a bound on the performance of the policy π. This proposition can be specialized to the following result regarding the performance of one-step lookahead policies, with additionally a refined performance bound.

Corollary 11.2.2. *Consider the one-step lookahead policy π defined by (11.4), and a cost approximation $\tilde{J}_k$ which satisfies (11.5) for all k and x_k, with $\delta_0 = \ldots = \delta_{N-1} = 0$*

$$\hat{J}_k(x_k) := \min_{u_k \in \bar{\mathsf{U}}_k(x_k)} E\Big[c_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\Big] \leq \tilde{J}_k(x_k). \quad (11.7)$$

Then we have the following bound on the performance of the policy π

$$J_{\pi,k} \leq \hat{J}_k \leq \tilde{J}_k.$$

Note that the result of the proposition could replace the result of the corollary to characterize the performance of a one-step lookahead policy which performs the minimization (11.4) only approximately. One way to find a cost approximation that satisfies condition (11.7) will be described in the next section.

Proof of the proposition. By backward induction. We have $J_{\pi,N} = \tilde{J}_N = c_N$. Next assume that for all x_{k+1}, we have

$$J_{\pi,k+1}(x_{k+1}) \leq \tilde{J}_{k+1}(x_{k+1}) + \sum_{j=k+1}^{N-1} \delta_j.$$

Then

$$\begin{aligned} J_{\pi,k}(x_k) &\overset{(DP)}{=} E\Big[c_k(x_k, \bar{\mu}_k(x_k), w_k) + J_{\pi,k+1}(f_k(x_k, \bar{\mu}_k(x_k), w_k))\Big] \\ &\overset{(induction)}{\leq} E\Big[c_k(x_k, \bar{\mu}_k(x_k), w_k) + \tilde{J}_{k+1}(f_k(x_k, \bar{\mu}_k(x_k), w_k))\Big] + \sum_{j=k+1}^{N-1} \delta_j \\ &\overset{(11.5)}{\leq} \tilde{J}_k(x_k) + \delta_k + \sum_{j=k+1}^{N-1} \delta_j. \end{aligned}$$

This concludes the induction step. □

Exercise 16. Prove corollary 11.2.2. You need to redo the proof above in order to get the improved bound $J_{\pi,k} \leq \hat{J}_k$.

Rollout Policies

The rollout algorithm is a special name given to the one-step lookahead algorithm when $\tilde{J}_k$ is the cost of an actual (suboptimal) policy π for the problem. This initial policy π is called the *base policy.* This heuristic policy could be for example a myopic policy (minimizing at each state only the stage cost). Since $\tilde{J}_k$ is the cost of a policy, we can use the DP algorithm to see that in this case $\tilde{J}_k$ satisfies

$$\tilde{J}_k(x_k) = E\Big[c_k(x_k, \tilde{\mu}_k(x_k), w_k) + \tilde{J}_{k+1}(f_k(x_k, \tilde{\mu}_k(x_k), w_k))\Big],$$

and so

$$\tilde{J}_k(x_k) \geq \min_{u_k \in \overline{\mathsf{U}}_k(x_k)} E\Big[c_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\Big],$$

at least if the base policy satisfies $\tilde{\mu}_k(x_k) \in \overline{\mathsf{U}}_k(x_k)$, which we assume. In other word, inequality (11.7) is verified. The *rollout policy* based on π is the one-step lookahead policy which uses the cost $\tilde{J}_k$ of the base policy π as approximation of the optimal cost-to-go. From the preceding discussion and corollary 11.2.2, we see that the performance of the rollout policy is always at least as good as the performance of the base policy. Often in practice, one can obtain significant improvements over the base policy. If we have several base heuristics $\pi_1, \dots, \pi_m$, we can also obtain a one-step lookahead policy that performs at least as well as any of them by considering the cost approximation ([Ber07, p.307])

$$\tilde{J}_k(x_k) = \min\{J_{\pi_1,k}(x_k), \dots, J_{\pi_m,k}(x_k)\}.$$

Note that the rollout policy is based on the same idea as policy improvement in the policy iteration algorithm. We can also define rollout policies based on l-step lookahead.

Cost-to-Go Approximations

There is no general way of selecting a good cost-to-go approximation for limited lookahead policies, and the choices are usually highly problem specifics. This will be discussed in more details in the following chapters, but here are a few initial remarks.

1. First, we do not necessarily need $\tilde{J}_k$ to be a good approximation of the true optimal cost J_k^*, but only that the relative values are approximated well, i.e., for an l-step lookahead policy

$$\tilde{J}_{k+l}(x) - \tilde{J}_{k+l}(x') \approx J_{k+l}^*(x) - J_{k+l}^*(x'),$$

   for all states x, x' that can be reached in l steps from the current step. For example, if $\tilde{J}_{k+l}$ and J_{k+l}^* differed by a constant, then the l-step policy would be optimal.

2. If the rollout approach is used, the cost of the base policy need not be computed analytically, but can be obtained by simulation (Monte-Carlo simulations if the problem is not deterministic). In order to compute a rollout policy with such a method, say for perfect information problems, we only need a simulator (or a way to perform experiments) that, given a state and control, can generate a value for the next state and for the corresponding stage cost. In an online control approach, we can in state x_k at stage k use the simulator to evaluate approximately the expected

value

$$Q_{\pi,k}(x_k, u_k) = E\Big[c_k(x_k, u_k, w_k) + J_{\pi,k+1}(f_k, u_k, w_k)\Big] \tag{11.8}$$
$$= E\left[c_k(x_k, u_k, w_k) + \sum_{j=k+1}^{N-1} c_j(x_j, \tilde{\mu}_j(x_j), w_j)\right],$$

for each possible control u_k, where $\pi = \{\tilde{\mu}_0, \dots, \tilde{\mu}_{N-1}\}$ is the base policy. The expected value on the second line is calculated approximately for example by taking the empirical average of the cost obtained for several trajectories (one trajectory suffices if the problem is deterministic). To speed-up calculations, we might also want to approximate $J_{\pi,k+1}$ by simpler function $\hat{J}_{k+1}$, truncate the horizon, etc.

3. Related to the remarks in the introduction, we can choose as $\tilde{J}_k$ the optimal cost-to-go for a simplified but related problem ([Ber07, section 6.3.3. and 6.3.4.]). We will discuss fluid approximations, where $\tilde{J}_k$ is obtained as the cost of a related deterministic problem, later in the course. It is hard to obtain rigorous performance bounds with these methods, but they usually perform well if the simplified problem is chosen adequately.

4. We can try to obtain $\tilde{J}_k$ within a parametrized family of functions and develop a scheme to tune the parameters. Such a parametric family of functions is often called an *approximation architecture*, and we write $\tilde{J}_k(x, r)$ where r is a finite-dimensional parameter (its coordinates are called "weights" in machine learning parlance). We will cover ways of tuning these weights later on for infinite-horizon time-homogeneous problems, in order to obtain a good approximation of the optimal cost-to-go. Clearly however to have any hope of obtaining a good approximation, we also need a rich enough family of functions. Sometimes it is intuitively possible to characterize the state x of the problem by a set of real-valued *features* $\phi_1(x), \dots, \phi_m(x)$, which are typically handcrafted based on human experience and intuition. For example, the state of a chess game is very high dimensional, but the features of a board configuration could score material balance, mobility, etc. Then a popular method is to use a linearly parametrized family of functions

$$\tilde{J}_k(x) = \hat{J}(\phi(x), r) = \sum_{i=1}^{m} \phi_i(x) r_i. \tag{11.9}$$

The features should encode the nonlinearity present in J. These features partition the state space X into subsets of states that have the same features

$$S_v = \{x \mid \phi(x) = v\},$$

and will have the same values for the cost-to-go approximation under the parameterization. Hence the features should capture a form of similarity between states. This is a type of state aggregation. If the state space X is finite with cardinality $|\mathsf{X}| = n$, then a function $\mathsf{X} \to \mathbb{R}$ is just a real n-dimensional vector and so $J^* \in \mathbb{R}^n$, an finite but high-dimensional space in practice. A linear approximation architecture as in (11.9) can be viewed as projecting functions $\mathsf{X} \to \mathbb{R}$ such as J^* on a space of dimension m, where m is generally chosen so that $m << n$. Representing without loss of generality X by the set $\{1, \dots, n\}$, we can use the notation J to also represent the vector $J = [J(1), \dots, J(n)]^T$, and similarly for the features $\phi_i = [\phi_i(1), \dots, \phi_i(n)]^T$. Then, droping the time index k, we can rewrite (11.9) as

$$\tilde{J} = \Phi r,$$

where Φ is the matrix $n \times m$ matrix

$$\Phi = \Big[\phi_1 \Big| \dots \Big| \phi_m\Big].$$

Hence J^* is approximated by a function in the subspace

$$S = \text{im } \Phi = \{\Phi r | r \in \mathbb{R}^m\},$$

an m-dimensional subspace of $\mathbb{R}^n$ (unless the features are poorly chosen and some are linearly dependent).

5. Replacing $J_{\pi,k+1}$ by J^*_{k+1} in (11.8), we define

$$Q^*_k(x_k, u_k) = E\Big[c_k(x_k, u_k, w_k) + J^*_{k+1}(f_k, u_k, w_k)],$$

known as the *Q-factor* of the state-control pair (x_k, u_k) at time k (no particular reason for that name, it's just that the first person to introduce it used the letter Q). The quantity $Q_k(x_k, u_k)$ is the cost-to-go of the policy that takes action u_k at times k in state s_k and then follows the optimal policy. $Q_{\pi,k}(x_k, u_k)$ has the same interpretation with the optimal policy replaced by policy π. We have said above that rollout can be performed using approximations $\tilde{Q}_{\pi,k}(\cdot, \cdot)$ of the Q-factors obtained by Monte-Carlo simulations. The approximate rollout control is

$$\bar{\mu}_k(x_k) \in \arg \min_{u_k \in \mathsf{U}_k(x_k)} \tilde{Q}_{\pi,k}(x_k, u_k). \tag{11.10}$$

To compare two controls u_k, u'_k using a simulation based methods, it is usually more accurate to work with the differences $Q_{\pi,k}(x_k, u_k) - Q_{\pi,k}(x_k, u'_k)$ and estimate it by sampling the quantity

$$C_k(x_k, u_k, \mathbf{w}_{k:N-1}) - C_k(x_k, u'_k, \mathbf{w}_{k:N-1}), \tag{11.11}$$

where

$$C_k(x_k, u_k, \mathbf{w}_k) = c_N(x_N) + c_k(x_k, u_k, w_k) + \sum_{j=k+1}^{N-1} c_j(x_j, \tilde{\mu}_j(x_j), w_j),$$

if the policy π after stage $k+1$ is $\tilde{\mu}_{k+1}, \tilde{\mu}_{k+2}, \ldots$. Note that in (11.11), the same noise realization is used in $C_k(x_k, u_k, \mathbf{w}_{k:N-1})$ and $C_k(x_k, u'_k, \mathbf{w}_{k:N-1})$. See the discussion in [Ber07, p.361]. Also, we can develop methods that approximate the Q-factors directly, e.g. using approximation architectures $\tilde{Q}_k(x, u; r) \approx Q^*_k(x, u)$, instead of approximating the cost-to-go. For example, we can consider a parametric architecture of the form

$$\tilde{Q}(x, u; r) = \sum_{k=1}^{m} r_k \phi_k(x, u).$$

Methods based on Q-factors are interesting in reinforcement learning, because the minimization (11.10) does not require the knowledge of the transition probabilities of the controlled Markov chain. More on this later.

Example 11.2.1 (polynomial approximation)**.** Suppose that the state space decomposes as the product $\mathsf{X} = \mathsf{X}_1 \times \ldots \times X_q$, a common situation that arises for example in the control of queueing networks. Assume $\mathsf{X}_i \subset \mathbb{R}$, for example $\mathsf{X}_i = \{0, 1, \ldots n_i\}$ (e.g., a finite capacity queue). Then a choice of architecture could consist in approximating J^* using polynomials in the variables $x_1, \ldots, x_q$ of degree at most d. The parameter r consists of the coefficients of the polynomial. For example for $d = 2$ (quadratic polynomials), this space has dimension $1 + q + q(q+1)/2$, and the features can be taken to be

$$\phi_0(x) = 1, \ \phi_i(x) = x_k, \ \phi_{ij}(x) = x_i x_j, \ \ 1 \leq i < j \leq q.$$

Then

$$\tilde{J}(x; r) = r_0 + \sum_{i=1}^{q} r_i x_i + \sum_{1 \leq i < j \leq q} r_{ij} x_i x_j.$$

11.3 Receding Horizon and Model Predictive Control

Receding Horizon Control

Receding (or rolling) horizon control is essentially l-step lookahead control with the cost-to-go approximation $\tilde{J}_{k+l}$ equal to a fixed function V. For example, we can take $V \equiv 0$, in which case the choice of control u_k is made by ignoring the cost incurred after $l-1$ stages (with $l = 1$ this is usually called the greedy policy). Another variant takes $V \equiv c_N$, if the terminal cost is important. We can use the receding horizon approach for infinite-horizon problems, keeping the length of the horizon of the problems solved at each stage always the same (this produces a stationary policy for time-homogeneous problems). The receding horizon approach can also be used in rollout to obtain an approximation of the cost of the base policy, and it is actually possible that this results in a better performance of the rollout policy than when using the full horizon to evaluate $J_{k,\pi}$ [Ber07, p.368]. It can be combined with CEC and OLFC, where the optimization steps only compute control sequences for a fixed horizon N (which is kept the same at all steps in an infinite horizon problem).

Remark. See [Ber07, p.367] for an example that shows that increasing the length of the horizon can decrease the performance of the controller!

Stability Issues in Model Predictive Control

Model predictive control (MPC) is a combination of some of the techniques mentioned earlier, such as CEC and the receding horizon approach. Initially, it was motivated by the desire to introduce nonlinearities and control and/or state constraints in the linear-quadratic framework, and obtain a suboptimal but stable closed-loop system (although we haven't covered this, it turns out that one big advantage of the LQR/LQG controllers for infinite horizon problems is that they insure stability of the closed-loop system - under reasonable controllability conditions, of course). Let us consider MPC for the time-homogeneous deterministic system

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

controlled over an infinite horizon, with quadratic stage cost

$$x_k^T Q x_k + u_k^T R u_k,\ \forall k \geq 0,$$

with $Q \succ 0$ and $R \succ 0$. There are now state and control constraints

$$x_k \in \mathsf{X} \subset \mathbb{R}^n,\ u_k \in \mathsf{U}_k(x_k),\ \forall k \geq 0,$$

with $0 \in \mathsf{X}$. Moreover the origin is an equilibrium point of the system with zero input, i.e., $0 \in \mathsf{U}(0)$ and $f(0,0) = 0$. We want to find a stationary controller (i.e., the such that control law μ_k is independent of k) that applies the control $\bar{\mu}(x) \in \mathsf{U}(x)$ in state x so that the trajectory of the closed-loop system

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

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

$$\sum_{k=0}^{\infty} x_K^T Q x_k + \bar{\mu}(x_k)^T R\, \bar{\mu}(x_k) < \infty. \tag{11.12}$$

Because Q and R are positive definite, this last condition implies $x_k \to 0$ and $\bar{\mu}(x_k) \to 0$ as $k \to \infty$, for all initial states $x_0 \in \mathsf{X}$. A sufficient condition for such a controller to exist is that for any initial condition x_0 there exist an integer m and a sequence of controls $u_{0:m-1}$ that brings x_0 to $x_m = 0$ in m stages while satisfying the constraints $x_1, \ldots, x_{m-1} \in \mathsf{X}$ (then letting $u_k = 0$ for $k \geq m$ leaves the system at the origin since $f(0,0) = 0$). This is a sort of constrained controllability assumption.

Assuming that this condition is satisfied, MPC can proceed as follows, if the system is in state x_k at stage k

1. Solve the m-stage problem

$$\begin{aligned}\hat{J}(x_k) = \min\ & \sum_{j=k}^{k+m-1} x_j^T Q x_j + u_j^T R u_j \\ \text{subject to}\ & x_{j+1} = f(x_j, u_j), j \geq k \\ & x_j \in \mathsf{X}, u_j \in \mathsf{U}_j(x_j), j \geq k \\ & x_{k+m} = 0.\end{aligned} \tag{11.13}$$

   Note the presence of the additional terminal constraint $x_{k+m} = 0$. The problem has a feasible solution by the constrained controllability assumption. Since the problem is deterministic, we can solve it using optimization methods, and find an optimal control sequence $\{\bar{u}_k, \ldots, \bar{u}_{k+m-1}\}$.

2. Apply the first control input $\bar{\mu}_k(x_k) = \bar{u}_k$, and discard the rest of the control sequence. Then observe the new state x_{k+1} and repeat. Note that x_{k+1} can be predicted with certainty for such a deterministic problem, but in practice the MPC control law is applied on a real-world system, which is subject to disturbances.

Let us now see that the MPC controller satisfies the stability condition (11.12). We show this by establishing a sort of dissipativity or Lyapunov condition for $\hat{J}$ defined in (11.13). Note that

$$\hat{J}(x_k) = x_k^T Q x_k + \bar{u}_k^T R \bar{u}_k + \tilde{J}(x_{k+1}), \tag{11.14}$$

where

$$\begin{aligned}\tilde{J}(x) = \min\ & \sum_{j=0}^{m-2} x_j^T Q x_j + u_j^T R u_j \\ \text{subject to}\ & x_{j+1} = f(x_j, u_j), j \geq 0 \\ & x_j \in \mathsf{X}, u_j \in \mathsf{U}_j(x_j), j \geq 0 \\ & x_{m-1} = 0,\end{aligned}$$

with the usual convention $\tilde{J}(x) = +\infty$ if this optimization problem is infeasible. Hence $\tilde{J}$ is defined exactly like $\hat{J}$, except that the horizon is changed from m to $m-1$. Clearly $\hat{J}(x) \leq \tilde{J}(x)$ for all $x \in \mathsf{X}$, because we allow one more step in the definition of $\hat{J}$. Hence from (11.14) we get

$$\hat{J}(x_k) \geq x_k^T Q x_k + \bar{u}_k^T R \bar{u}_k + \hat{J}(x_{k+1}).$$

By immediate recursion,

$$\hat{J}(x_0) \geq \sum_{k=0}^{K} (x_k^T Q x_k + u_k^T R u_k) + \hat{J}(x_{K+1}),$$

where the control inputs and states u_k, x_k in this expression are generated by the MPC algorithm. Then noting that $J(x) \geq 0$, and letting $K \to \infty$, we get

$$\sum_{k=0}^{\infty}(x_k^T Q x_k + u_k^T R u_k) \leq \hat{J}(x_0) < \infty. \tag{11.15}$$

Note that (11.15) provides an upper bound on the performance of the MPC algorithm.

In the scheme above, one must balance two effects for the choice of an adequate value of the horizon m. The requirement of constrained controllability is more easily established for larger values of m. Moreover, the performance upper bound $\hat{J}(x)$ (see 11.15) decreases with larger m, which provides an additional incentive for larger values of m. However, as m increases it becomes harder to solve the optimization problems (11.13) at each stage, which is an important consideration in real-time applications.

Remark. Note that you can also interpret the MPC scheme above as a rollout policy using the base policy defining $\tilde{J}$ (driving the state to 0 in $m-1$ stages and keeping it there afterwards using 0 inputs). Indeed we have

$$\hat{J}(x_k) = \min_{u \in \mathsf{U}(x_k)} \{x_k^T Q x_k + u^T R u + \tilde{J}(f(x_k, u))\},$$

see (11.14) above.

There are many variations on the basic MPC scheme, which generally use several of the ideas described in this chapter. For example, instead to adding a terminal constraint $x_{k+m} = 0$ in the problem (11.13), one can add a large penalty term in the cost function for nonzero values of the state x_{k+m}. The main theme of the literature on MPC is to maintain stability of the closed-loop system, at least before including the effect of disturbances.