CS294-40 Learning for Robotics and Control Lecture 7 - 9/18/2008

Differential Dynamic Programing (DDP)

Lecturer: Pieter Abbeel *Scribe: Brandon Basso*

1 Lecture outline

- Setup
- DDP
- Trajectory following
- Runtime and offline solutions

2 Recap: Problem Setup

We are considering LTI systems

$$x_{t+1} = A_t x_t + B_t u_t \tag{1}$$

We want to solve the finite horizon optimization:

$$\min_{u_0 \dots u_{H-1}} \sum_{t=0}^{H-1} (x_t^\top Q_t x_t + u_t^\top R_t u_t) + x_H^\top P_H x_H \tag{2}$$

for $t = H-1, H-2 \ldots 0$

$$\begin{aligned} K_t &= -(B_t P_t B_t + R_t)^{-1} B_t^T P_{t+1} A_t & (3)\\ P_t &= Q_t + K_t^\top R_t K_t + (A_t + B_t K_t)^\top P_{t+1} (A_t + B_t K_t) & (4)\\ \mu_t^*(x) &= K_t x & (5) \end{aligned}$$

where the cost to go is $x_t^T P_t x_t$

To formulate the problem in a more general (nonlinear, for example) setting, consider $\mu_i : S \rightarrow \mathcal{A}$, which maps states to actions. We want to solve the following minimization:

$$\min_{\mu_0 \dots \mu_{H-1}} \sum_{t=0}^{H-1} g_t(x_t, u_t) + g_H(x_t) \tag{6}$$

subject to

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) & (7)\\ u_t &= \mu_t(x_t) & (8) \end{aligned}$$

and the state at time zero equals x_0.

3 Differential Dynamic Programming (DDP)

3.1 Algorithm:

Assume we are given $\pi(0)$

1. Set $i = 0$
2. Run π^i, record state and input sequence $x_0^i, u_0^i, ...$
3. Compute A_t, B_t, a_t $\forall t$ linearization about x_t^i, u_t^i
   ie. $x_{t+1} = A_t x_t + B_t u_t + a_t$ (Aside: linearization is a big assumption!)
4. Compute Q_t, q_t, R_t, r_t by quadratic approximation about x_t^i, u_t^i
   $\min_{\mu_1 \ldots \mu_H} \sum (x_t^\top Q_t x_t + a q_t^T P_H x_t + u_t^\top R_t u_t)$
5. Run LQR, which gives us $\pi^{i+1} : \mu_t^i(x) = K_t(x) + k_t$

3.2 Considerations

Some issues to consider:

- LQR is only optimal for linear systems with a quadratic cost function
- Deviation from the linearization point can result in very poor performance
- If Q or R are not positive definite, set negative eigenvalues to 0

Some options:

- Skip the linearization. (How?) (Discretizing the state-space results in algorithms growing exponentially in the dimensionality of the state space—typically only feasible up to 6-dim, assuming very good implementation.)
- Force the trajectory to stay close to the linearization point

3.3 Practical Solution

Quadraticize the cost function

$$\bar{g}_t(x_t, u_t) = \alpha g_t(x_t, u_t) + (1 - \alpha)[(x_t - x_t^i)^\top \bar{Q}_t (x_t - x_t^i) + (u_t - u_t^i)^\top \bar{R}_t (u_t - u_t^i)] \tag{9}$$

where $\alpha \in (0, 1)$

Now we can perform a line search over α for the one that gives the best control performance.
Note: finding π^0 can be hard! Pick what makes sense based on problem-specific knowledge.

4 Trajectory Following

In the trajectory following problem, we want to track some desired "reference" state sequence

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ \min_{u_0 \ldots u_{H-1}} \sum_{t=0}^{H-1} & (x_t - x_t^*)^\top Q_t (x_t - x_t^*) + (u_t - u_t^*)^\top R_t (u_t - u_t^*) + (x_H - x_H^*)^\top P_H (x_H - x_H^*) \end{aligned} \tag{10}$$

where $x_0^* ... x_H^*$ is the target sequence

4.1 Algorithm

Based on linearization about target sequence

1. $x_t^0 = x_t^*, u_t^0 = u_t^* \ \forall t$
2. Set $i = 0$
3. For $\beta = .999, \ .95, \ ...0$

   Set the dynamics model to: $x_{t+1} = \bar{f}(x_t, u_t) = \beta x_t^* + (1-\beta) f(x_t, u_t)$.
   Linearize the dynamics around x_t^i, u_t^i
   Run LQR, get π^{i+1}
   Run π^{i+1} in simulation with $\bar{f}^i$
   Iterate

In practice it can be a bit of a dark art to pick the sequence of values that β takes on. Spreading the values too far apart can result in the algorithm not converging to a good solution.

5 Receding horizon

5.1 Solving the entire control problem at every time-step at run-time

At time t we want to run DDP on the following

$$\min_{u_t \ldots u_{H-1}} \sum_{k=t}^{H-1} g_k(x_k, u_k) \tag{11}$$

with the constraint $x_{k+1} = f_k(x_k, u_k) \forall k \geq t$, where x_k is the current state.

5.2 Receding horizon DDP

It is impractical in general to solve DDP at runtime, mainly because of the time spent in computing the linearization: A_t, B_t, C_t, D_t. A much more practical cost function is the following "receding-horizon" cost function:

$$\sum_{k=t}^{t+h} g_k(x_k, u_k) + G_{t+h}(x_{t+h}) \tag{12}$$

Here the choice of G_{t+h} can be crucial to the success of the algorithm.

We could run DDP offline to obtain a decent cost-to-go estimate G and then use this in the following algorithm:

1. Run DDP offline
2. Store the cost to go: $G_t(x_t) = (x_t - x_t *^) {}^\top P_t(x_t - x_t^*)$
3. Online, run the receding-horizon method above using $G_t(x_t)$ as the cost-to-go.