Controlling Stochastic Systems

• What happens when the system we are trying to control is stochastic?

\[ x_{t+1} = f(x_t, u_t) + \epsilon_t \]

• Remember, stochasticity can arise from “actual” noise in the system, errors in modeling, inaccurate predictions, etc.

• We no longer know which state we will end up in, how do we define cost?
• A natural extension, pick $u_{1:T}$ to minimize the expected cost

$$J = \mathbb{E} \left[ \sum_{t=1}^{H} C(x_t, u_t) \right] = \sum_{t=1}^{H} \mathbb{E}[C(x_t, u_t)]$$

• Not the only possibility, for example might want to minimize the probability that cost exceeds some threshold

$$J = P \left( \sum_{t=1}^{H} C(x_t, u_t) > c \right)$$

• However, expected cost is quite common, as it is often (relatively) easy to optimize
• **The bad news:** The general stochastic control problem can be extremely hard to solve optimally.

• **The good news:** Certain special cases can be solved exactly (LQ control with Gaussian noise, Markov Decision Processes); an optimization-based approach known as *Model Predictive Control* (MPC) works well in many cases (even though it is sub-optimal).
Linear Quadratic Stochastic Control

• Linear quadratic setting: linear dynamics and quadratic costs, now with (zero-mean Gaussian) random noise

\[ x_{t+1} = Ax_t + Bu_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Sigma) \]

\[ J = \mathbb{E} \left[ \sum_{t=1}^{H} \left( \|Q x_t\|_2^2 + \|R u_t\|_2^2 \right) \right] \]

• A remarkable fact: optimal stochastic solution is the same as LQR for deterministic setting (just ignore noise entirely)

\[ \texttt{K} = \texttt{dlqr(A, B, Q'*Q, R'*R);} \]
• Basic idea: as discussed before for linear-Gaussian systems, expected state of system is unaffected by noise

\[ \mathbf{E}[x_{t+1}] = A\mathbf{E}[x_t] + Bu_t \]

• Further, expected value of quadratic cost is just quadratic cost of expected state plus a constant

\[ \mathbf{E}[\|Qx_t\|_2^2] = \|QE[x_t]\|_2^2 + c \]

(constant term does depend on the distribution of \( x_t \))

• Derivation of LQR solution ends up not depending on this constant term at all, just adds a fixed amount to the total expected cost
Example: generator control with random disturbances to generator angular velocities (i.e., random changes to loads)
- Can regulate generators more closely around desired point (at the expense of larger inputs)
• Extension: partially observable stochastic linear quadratic Gaussian:

\[ x_{t+1} = Ax_t + Bu_t + \epsilon_t \quad \epsilon_t \sim \mathcal{N}(0, \Sigma) \]
\[ y_t = Cx_t + q_t \quad q_t \sim N(0, Q) \]

• In this setting the following procedure is optimal:

1. Estimate current state \( \hat{x}_t \) using Kalman filter (recall, like a least-squares estimate of \( x_t \))

2. Apply LQR controller to state estimate \( u_t = K\hat{x}_t \)

• The fact that this is optimal is known as the separation principle, resulting algorithm called LQG

• Very rare for such procedures to be optimal
Aside: Markov Decision Processes

- Recall setting for Markov Decision Process:
  - **State:** \( x_t \in \{1, 2, \ldots, n\} \)
  - **Control:** \( u_t \in \{1, 2, \ldots, m\} \)
  - **Transition probabilities:** \( P^u \in \mathbb{R}^{n \times n}, \ u = 1, \ldots, m \)
    \[
P(x_{t+1} = j | x_t = i, u_t = k) = P_{i,j}^k
    \]
  - **Cost:** \( C : \{1, \ldots, n\} \times \{1, \ldots, m\} \rightarrow \mathbb{R} \)
• Approach (Value Iteration): iteratively build optimal cost-to-go function $J^*$ (also called value function)
  
  – Initialize

  $$J^*_T(x) = \min_u C(x, u), \ \forall x = 1, \ldots, n$$

  – Repeat for $t = T - 1, \ldots, 1$

  $$J^*_t(x) = \min_u \left\{ C(x, u) + \sum_{x'=1}^{n} P_{x,x'}^u J^*_{t+1}(x') \right\}$$

  $$\forall x = 1, \ldots, n$$

  – Optimal policy is then

  $$\pi^*_t(x) = \arg \min_u J^*_t(x)$$
• Again, method is only tractable for relatively small systems, so that total number of states $n$ and $m$ can be enumerated

• For continuous-state systems, typically discretizations will require a number of states that grows exponentially in system dimension

• Unlike LQG setting, partially observable systems are *much* more challenging; setting known as Partially Observable MDPs (POMDPs)
Model Predictive Control

- How can we use the optimization approach to control in stochastic systems?
- Executing the controls we get back from our optimization problem can cause system to diverge
• Executing controls

![Graph showing generator power over time for Gen 1 to Gen 5.]

• What optimization method predicts

![Graph showing the difference between θ and θ^d over time for Gen 1 to Gen 5.]

Time (s)
Generator power

θ  − θ^d

0 2 4 6 8 10
−2
−1
0
1
2
3
Time (s)
• Executing controls

![Graph showing generator power against time](image1)

• What actually happens

![Graph showing angle deviation against time](image2)
• The model predictive control (MPC) solution: re-run our optimization-based control approach at every time point, execute just the first returned control $u_1$

• Repeat for $t = 1, \ldots, T$:

1. Observe state of system $x_t$

2. Solve optimization problem

\[
\begin{align*}
\text{minimize} & \quad \sum_{\tau=1}^{H} C(\tilde{x}_\tau, \tilde{u}_\tau) \\
\text{subject to} & \quad \tilde{x}_{\tau+1} = A\tilde{x}_\tau + B\tilde{u}_\tau \\
& \quad \tilde{x}_1 = x_t \\
& \quad u \leq \tilde{u}_\tau \leq \bar{u}, \quad x \leq \tilde{x}_\tau \leq \bar{x}
\end{align*}
\]

3. Execute the first control input $u_t = \tilde{u}_1$
• Resulting trajectories from MPC

![Plot of \( \theta - \theta^d \) against time (s) for different generations.]

- Gen 1
- Gen 2
- Gen 3
- Gen 4
- Gen 5

• Control inputs applied

![Plot of generator power against time (s) for different generations.]

- Gen 1
- Gen 2
- Gen 3
- Gen 4
- Gen 5
Resolving optimization problem at each time point allows the system to adjust for actual evolution, deviations from expected states.

But remember, this requires solving an entire optimization problem at every time step (compare to LQR, which requires just a matrix multiplication)

- Five generator system (very small as control tasks go)

- For integration time step of 0.01 seconds, optimization over horizon of 10 seconds takes 0.25 second (using CPLEX solver directly)

- We couldn’t run this in real time even on this small system
Speeding up MPC

• Speed-up #1: shorter optimization horizon

• Remember that MPC effectively had two horizons:

  1. $T$: actual horizon of control task

  2. $H$: horizon we use for optimization

• Often, we can use a much shorter horizon in the optimization problem (intuition: states far in the future won’t affect what we do now very much)
• $H = 10$ seconds (same as $T$)

![Graph showing system response with $H = 10$ seconds]

• $H = 2$ seconds

![Graph showing system response with $H = 2$ seconds]
• $H = 0.5$ seconds

\[ \begin{array}{c}
\theta - \theta^d \\
\end{array} \]

\[ \begin{array}{c}
\text{Time (s)} \\
0 & 2 & 4 & 6 & 8 & 10 \\
-0.1 & 0 & 0.1 & 0.2 & 0.3 \\
\end{array} \]

• $H = 0.1$ seconds

\[ \begin{array}{c}
\theta - \theta^d \\
\end{array} \]

\[ \begin{array}{c}
\text{Time (s)} \\
0 & 2 & 4 & 6 & 8 & 10 \\
-0.1 & 0 & 0.1 & 0.2 & 0.3 \\
\end{array} \]
• Speed-up #2: less frequent updating

• In MPC we recomputе the controls based upon observed states, but there is no need to do this at every single time step

• Instead, compute updates only every $k$ time steps

• Intuition: over short time periods, effects of stochasticity aren’t that bad
• Update every 0.01 seconds (same as integration time step)

![Graph showing time (s) vs. \( \theta - \theta_d \) with Gen 1 to Gen 5.

• Update every 0.1 seconds

![Graph showing time (s) vs. \( \theta - \theta_d \) with Gen 1 to Gen 5.]
• Update every 0.5 seconds

![Graph 1](image1)

• Update every 2 seconds

![Graph 2](image2)
• On my desktop (Core i7, 4-core, 3.4Ghz), $H = 2$ seconds, updating every 0.1 seconds takes 0.086 seconds per optimization (faster than real time)

• But admittedly, this is still a lot of computation for a relatively simple task like this

• MPC finds the most application in control or planning tasks where there is significantly longer time constants (e.g., the scheduling of generation and storage you will work on in the homework)
• Speed-up #3: faster optimization

• For MPC, using “standard form” solvers over YALMIP can be a big win

• Some customization can also pay off here (e.g., initializing subsequent optimization iterations with solutions from previous ones)

• Some recent research in code generation for small custom optimization solvers
Aside: Learning dynamical systems

- All control discussion thus far assumes that we know the dynamical system
  \[ x_{t+1} = Ax_t + Bu_t \]

- What if we don’t know the $A$ and $B$ matrices?

- If we can observe a collection of data points generated by the system, we can use machine learning to estimate the model

  - Typically will want to use stochastic control methods for such models, because learning algorithms may not make perfect predictions
• In some detail: suppose we can observe a collection of data points from the system

\[ x_1, u_1, x_2, u_2, \ldots, x_{T-1}, u_{T-1}, x_T \]

• Then we want to find \( A \) and \( B \) that solve optimization problem

\[
\min_{A,B} \sum_{t=1}^{T-1} \| x_{t+1} - Ax_t - Bu_t \|_2^2
\]

(or use a different loss function from ML lectures)

• This is a linear least-squares problem (actually \( n \) least squares problems, since \( x_t \) is multivariate)

• If the true system is linear, this works well, but if the true system is not quite linear, the distribution of states/control in the training set becomes a big issue