

Lecture Schedule

Dynamical programming

- 1 The finite-horizon decision problem
31 January
- 2 Dynamical Programming
7 February
- 3 DP reformulations and introduction to Control
14 February

Control

- 4 Discretization and PID control
21 February
- 5 Direct methods and control by optimization
28 February
- 6 Linear-quadratic problems in control
7 March
- 7 **Linearization and iterative LQR**
14 March

Syllabus: <https://02465material.pages.compute.dtu.dk/02465public>
Help improve lecture by giving feedback on DTU learn

Reinforcement learning

- 8 Exploration and Bandits
21 March
- 9 Policy and value iteration
4 April
- 10 Monte-carlo methods and TD learning
11 April
- 11 Model-Free Control with tabular and linear methods
18 April
- 12 Eligibility traces and value-function approximations
25 April
- 13 Q-learning and deep-Q learning
2 May

Housekeeping

- Old exam sets online
- Most of the feedback for project 1 is online on DTU Learn
 - The rest will be available in a few days

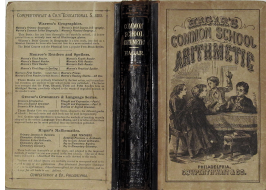
A bit of analysis

- Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a well-behaved function
- The **gradient** is defined as:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

- The **Hessian** is defined as

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$



More analysis



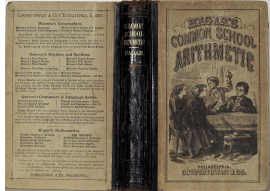
- Let $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a well-behaved multi-variate function defined as

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}$$

- The **Jacobian matrix** is defined as:

$$\mathbf{J}_{\mathbf{f}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Approximations



- Given the gradient and Hessian we can approximate f around \mathbf{x}

$$f(\mathbf{x} + \Delta) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \Delta + \frac{1}{2} \Delta^T \mathbf{H}(\mathbf{x}) \Delta$$

- A similar expression can be obtained for a multi-variate \mathbf{f} :

$$\mathbf{f}(\mathbf{x} + \Delta) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}_{\mathbf{f}}(\mathbf{x}) \Delta$$

Fundamental relations that are the basis for gradient descent, many higher-order optimization methods and all sorts of ML

From last time: The Linear-quadratic regulator

- For $k = 0, 1, \dots, N - 1$

$$\begin{aligned}x_{k+1} &= f_k(x_k, u_k, w_k) = A_k x_k + B_k u_k, \\g_k(x_k, u_k, w_k) &= \frac{1}{2} x_k^\top Q_k x_k + \frac{1}{2} u_k^\top R_k u_k, \\g_N(x_k) &= \frac{1}{2} x_N^\top Q_N x_N\end{aligned}$$

- The accumulated cost is:

$$J_{\mathbf{u}}(\mathbf{x}_0) = g_N(\mathbf{x}_N) + \sum_{k=0}^{N-1} g_k(\mathbf{x}_k, \mathbf{u}_k)$$

- We put this into the dynamical programming algorithm and...

Apply dynamical programming:

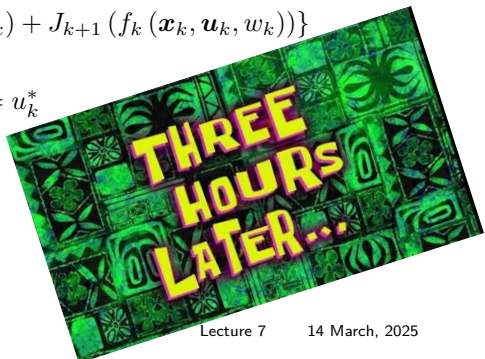
- Define $V_N \equiv Q_N$ and initialize:

$$J_N^*(\mathbf{x}_N) = \frac{1}{2} \mathbf{x}_N^T Q_N \mathbf{x}_N = \frac{1}{2} \mathbf{x}_N^T V_N \mathbf{x}_N$$

- DP iteration (start at $k = N - 1$)

$$J_k(\mathbf{x}_k) = \min_{\mathbf{u}_k} \mathbb{E}_{w_k} \{g_k(\mathbf{x}_k, \mathbf{u}_k, w_k) + J_{k+1}(f_k(\mathbf{x}_k, \mathbf{u}_k, w_k))\}$$

- Remember to store optimal u_k^* as $\pi_k(x_k) = u_k^*$



LQR, simplified form

This gives the controller:

$$\textcircled{1} V_N = Q_N$$

$$\textcircled{2} L_k = -(R_k + B_k^T V_{k+1} B_k)^{-1} (B_k^T V_{k+1} A_k)$$

$$\textcircled{3} V_k = Q_k + L_k^T R_k L_k + (A_k + B_k L_k)^T V_{k+1} (A_k + B_k L_k)$$

$$\textcircled{4} \mathbf{u}_k^* = L_k \mathbf{x}_k$$

$$\textcircled{5} J_k^*(\mathbf{x}_k) = \frac{1}{2} \mathbf{x}_k^T V_k \mathbf{x}_k$$

Double Integrator Example

- True dynamics

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{u}(t) \quad (1)$$

- **Euler discretization** using $\Delta = 1$ System evolves according to:

$$\mathbf{x}_{k+1} = \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}}_{=A} \mathbf{x}_k + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{=B} \mathbf{u}_k$$

- Quadratic cost function:

$$J(\mathbf{x}_0) = \sum_{k=0}^N \mathbf{x}_k^\top Q \mathbf{x}_k + \frac{1}{2} u_k^2$$

- Where:

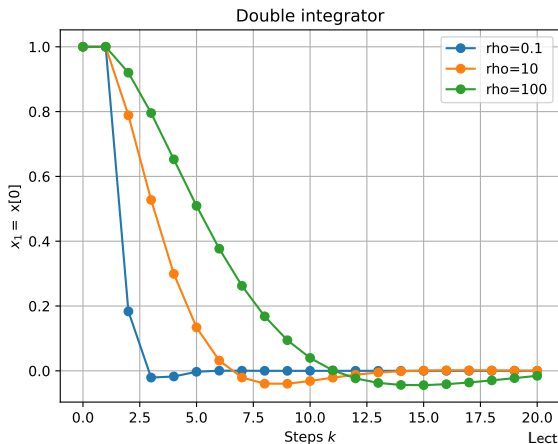
$$Q_k = Q_N = \begin{bmatrix} \frac{1}{\rho} & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1$$

Exponential integrator

- Apply discrete LQR
- Simulate starting in $\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ using policy

$$\pi_k(\mathbf{x}_k) = L_k \mathbf{x}_k$$

- What about the true system $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u})$?



The most general form of LQR

- General dynamics:

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{d}_k$$

- General quadratic cost:

$$c_k(\mathbf{x}_k, \mathbf{u}_k) = \frac{1}{2} \mathbf{x}_k^T Q_k \mathbf{x}_k + \frac{1}{2} \mathbf{u}_k^T R_k \mathbf{u}_k + \mathbf{u}_k^T H_k \mathbf{x}_k + \mathbf{q}_k^T \mathbf{x}_k + \mathbf{r}_k^T \mathbf{u}_k + q_k$$

$$c_N(\mathbf{x}_k) = \frac{1}{2} \mathbf{x}_k^T Q_N \mathbf{x}_k + \mathbf{q}_N^T \mathbf{x}_k + q_N$$



General discrete LQR algorithm

How to start living in luxury and never work again!

$$\dots (V_{k+1} + \mu I) \dots$$

$$1. V_N = Q_N; \mathbf{v}_N = \mathbf{q}_N; v_N = q_N$$

$$2. \begin{aligned} L_k &= -S_{\mathbf{u}\mathbf{u},k}^{-1} S_{\mathbf{u}\mathbf{x},k} & S_{\mathbf{u},k} &= \mathbf{r}_k + B_k^T \mathbf{v}_{k+1} + B_k^T V_{k+1} \mathbf{d}_k \\ \mathbf{l}_k &= -S_{\mathbf{u}\mathbf{u},k}^{-1} S_{\mathbf{u},k} & S_{\mathbf{u}\mathbf{u},k} &= R_k + B_k^T V_{k+1} B_k \\ & & S_{\mathbf{u}\mathbf{x},k} &= H_k + B_k^T V_{k+1} A_k. \end{aligned}$$

$$3. \begin{aligned} V_k &= Q_k + A_k^T V_{k+1} A_k - L_k^T S_{\mathbf{u}\mathbf{u},k} L_k \\ \mathbf{v}_k &= \mathbf{q}_k + A_k^T (\mathbf{v}_{k+1} + V_{k+1} \mathbf{d}_k) + S_{\mathbf{u}\mathbf{x},k}^T \mathbf{l}_k \\ v_k &= v_{k+1} + q_k + \mathbf{d}_k^T \mathbf{v}_{k+1} + \frac{1}{2} \mathbf{d}_k^T V_{k+1} \mathbf{d}_k + \frac{1}{2} \mathbf{l}_k^T S_{\mathbf{u},k} \end{aligned}$$

$$4. \mathbf{u}_k^* = \mathbf{l}_k + L_k \mathbf{x}_k$$

$$5. J_k(\mathbf{x}_k) = \frac{1}{2} \mathbf{x}_k^T V_k \mathbf{x}_k + \mathbf{v}_k^T \mathbf{x}_k + v_k.$$

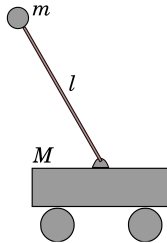
(more seriously μ is a regularization term: $\mu \rightarrow \infty \Rightarrow \mathbf{u} \rightarrow 0$)

Quiz: LQR

Which one of the following statements is correct?

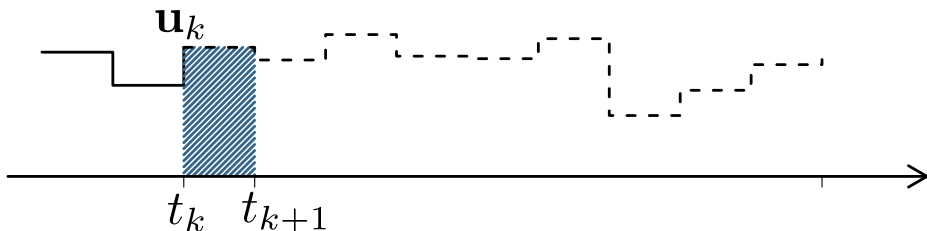
- a.** Control problems where the continuous-time dynamics takes the form $\ddot{x} = a\dot{x} + bx + c + u$ falls outside the scope of the linear quadratic regulator
- b.** The linear-quadratic regulator is an example of model-free control
- c.** In a linear-quadratic control problem of the form $x_{k+1} = Ax_k + Bu_k$, the matrices A and B must both be square.
- d.** The cost-functions suitable for a linear-quadratic regulator can potentially produce negative values
- e.** Don't know.

Controlling non-linear systems: Cartpole



- Continuous coordinates $\mathbf{x}(t) = [x(t) \quad \dot{x}(t) \quad \theta(t) \quad \dot{\theta}(t)]$
- Action u is one-dimensional; the force applied to cart

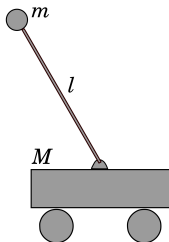
Discretization



- Choose grid size N : $t_0, t_1, \dots, t_N = t_F$, $t_{k+1} - t_k = \Delta$
- $\mathbf{x}_k = \mathbf{x}(t_k)$, $\mathbf{u}_k = \mathbf{u}(t_k)$
- Eulers method $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta f(\mathbf{x}_k, \mathbf{u}_k)$
- Discretized dynamics will have the form:

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k)$$

Cartpole cost function



- We also apply a variable transformation:

$$\phi_x : [x \quad \dot{x} \quad \theta \quad \dot{\theta}] \mapsto [x \quad \dot{x} \quad \sin(\theta) \quad \cos(\theta) \quad \dot{\theta}]. \quad (2)$$

- The cost function is of the form:

$$c(\mathbf{x}_k, \mathbf{u}_k) = \frac{1}{2} \left(\mathbf{x} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right)^\top Q \left(\mathbf{x} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right) + \frac{1}{2} \|\mathbf{u}_k\|^2$$

Controlling a non-linear system

- **We know** how to solve a linear/quadratic control problems of the form

$$\begin{aligned}\mathbf{x}_{k+1} &= A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{d}_k \\ c_k(\mathbf{x}_k, \mathbf{u}_k) &= \frac{1}{2} \mathbf{x}_k^\top Q \mathbf{x}_k + \frac{1}{2} \mathbf{u}_k^\top R \mathbf{u}_k + \dots\end{aligned}$$

- **How** can we use that to solve a problem with non-linear dynamics?

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) \\ c_k(\mathbf{x}_k, \mathbf{u}_k) &= \dots\end{aligned}$$

Solution: Linearization!

Assume a general dynamics:

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k), \quad c(\mathbf{x}_k, \mathbf{u}_k)$$

Assume system is near $\bar{\mathbf{x}}, \bar{\mathbf{u}}$. Expand using **Jacobians**

$$\mathbf{f}_k(\mathbf{x}_k, \mathbf{u}_k) \approx \mathbf{f}_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \underbrace{\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}}(\bar{\mathbf{x}}, \bar{\mathbf{u}})}_{A_k} (\mathbf{x}_k - \bar{\mathbf{x}}) + \underbrace{\frac{\partial \mathbf{f}_k}{\partial \mathbf{u}}(\bar{\mathbf{x}}, \bar{\mathbf{u}})}_{B_k} (\mathbf{u}_k - \bar{\mathbf{u}})$$

Simplifies to:

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{f}_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) - A_k \bar{\mathbf{x}} - B_k \bar{\mathbf{u}}$$

Linearization and iLQR

Algorithm 1 Linearized LQR

Require: Given a problem horizon N , and an expansion point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ corresponding to where the system should be


Compute A_k, B_k, \mathbf{d}_k by expansion

Cost function is the same as usual because it is already quadratic

Use LQR, with dynamics A_k, B_k, \mathbf{d}_k and cost matrices Q_k, R_k, \mathbf{q}_k to obtain controller L_k, \mathbf{l}_k for $k = 0, \dots, N - 1$.

In a state \mathbf{x}_k , the control law is $\mathbf{u}_k^* = \bar{\mathbf{l}}_k + L_k \mathbf{x}_k$

- Select expansion point $\bar{\mathbf{x}}, \bar{\mathbf{u}}$ as desired state
- Usually $A_k = A, B_k = B$ so just choose a large N and use L_0, \mathbf{l}_0

 lecture_06_linearize_b.py

Quiz: Linearized LQR?

Which one of the following statements is **correct**?

- a. We should apply Exponential Integration to the linearized dynamics $A_k (= J_x \mathbf{f}_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}))$ and B_k before applying LQR
- b. Assuming Δ is small enough, the error incurred by Euler discretization can be managed.
- c. Assuming we plan on a sufficiently long horizon, the linear approximation to the dynamics does not result in major issues
- d. This is a computationally inefficient method compared to e.g. Direct control
- e. Don't know

(Note: Quiz changed from lecture due to double-negation of answers being beyond my abilities; In the lecture, option *a* was actually the right option, but I read it incorrectly and thought I had an error. In this formulation quiz, option *b* is correct)

Fixing linearization method

- **Problem:** The system may be far from \bar{x}, \bar{u} giving a poor approximation
- **Idea:** Select expansion points \bar{x}, \bar{u} near current trajectory x_k, u_k
- **How?**
 - Start with initial guess \bar{x}_k, \bar{u}_k (**nominal trajectory**)
 - Approximate around this guess
 - Use LQR on approximation to get initial control law
 - Simulate trajectory based on this control law
 - Use the trajectory as a new guess and repeat

LQR Tracking around Nonlinear Trajectory

Given initial guess $\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k$ (**nominal trajectory**) for $k = 1, 2, \dots, N - 1$

$$\mathbf{x}_{k+1} \approx \underbrace{\mathbf{f}_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)}_{\bar{\mathbf{x}}_{k+1}} + \underbrace{\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}}(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)}_{A_k} \underbrace{(\mathbf{x}_k - \bar{\mathbf{x}}_k)}_{\delta \mathbf{x}} + \underbrace{\frac{\partial \mathbf{f}_k}{\partial \mathbf{u}}(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)}_{B_k} \underbrace{(\mathbf{u}_k - \bar{\mathbf{u}}_k)}_{\delta \mathbf{u}}$$

Introduce new variables signifying deviation around the **nominal trajectory**:

$$\delta \mathbf{x}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k, \quad \delta \mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k.$$

Back-substituting gives:

$$\delta \mathbf{x}_{k+1} = A_k \delta \mathbf{x}_k + B_k \delta \mathbf{u}_k$$

Expansion of the cost function

We then expand the cost-function around: $\mathbf{z}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}$ and $\bar{\mathbf{z}} = \begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\mathbf{u}} \end{bmatrix}$:

$$c_k(\mathbf{x}_k, \mathbf{u}_k) \approx c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + (\nabla_{\mathbf{z}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}))^\top (\mathbf{z}_k - \bar{\mathbf{z}}) + \frac{1}{2} (\mathbf{z}_k - \bar{\mathbf{z}})^\top H_{\bar{\mathbf{z}}} (\mathbf{z}_k - \bar{\mathbf{z}})$$

Multiplying out all the terms gives a quadratic approximation in the δ -coordinates

$$\begin{aligned} c_k &= c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \\ c_{\mathbf{x},k} &= \nabla_{\mathbf{x}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}), & c_{\mathbf{u},k} &= \nabla_{\mathbf{u}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \\ c_{\mathbf{x}\mathbf{x},k} &= H_{\mathbf{x}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}), & c_{\mathbf{u}\mathbf{u},k} &= H_{\mathbf{u}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \\ c_{\mathbf{u}\mathbf{x},k} &= J_{\mathbf{x}} \nabla_{\mathbf{u}} c_k(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \end{aligned}$$

Expansion of the cost function

all in all we get a quadratic cost function:

$$\begin{aligned}c_k(\delta \mathbf{x}_k, \delta \mathbf{u}_k) &= \frac{1}{2} \delta \mathbf{x}_k^\top c_{xx,k} \delta \mathbf{x}_k + c_{\mathbf{x},k}^\top \delta \mathbf{x}_k \\ &\quad + \frac{1}{2} \delta \mathbf{u}_k^\top c_{uu,k} \delta \mathbf{u}_k + c_{\mathbf{u},k}^\top \delta \mathbf{u}_k + \delta \mathbf{u}_k^\top c_{ux,k} \delta \mathbf{x}_k + c_k \\ c_N(\delta \mathbf{x}_N) &= \frac{1}{2} \delta \mathbf{x}_N^\top c_{xx,N} \delta \mathbf{x}_N + c_{\mathbf{x},N}^\top \delta \mathbf{x}_N + c_N\end{aligned}$$

Linearized solution to actual controls

- Put linearized problem into LQR
- Once problem is solved, new control inputs obey

$$\delta \mathbf{u}_k^* = \mathbf{l}_k + L_k \delta \mathbf{x}_k$$

- Rearranging

$$(\mathbf{u}_k^* - \bar{\mathbf{u}}_k) = \mathbf{l}_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)$$

- Or

$$\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \mathbf{l}_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)$$

Basic iLQR Algorithm

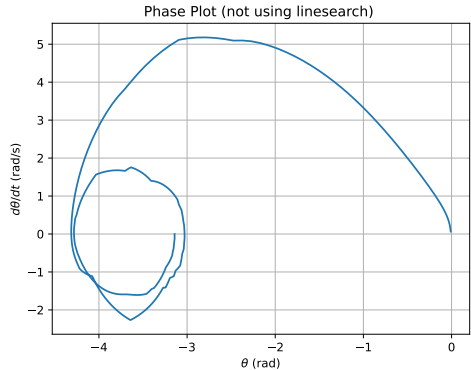
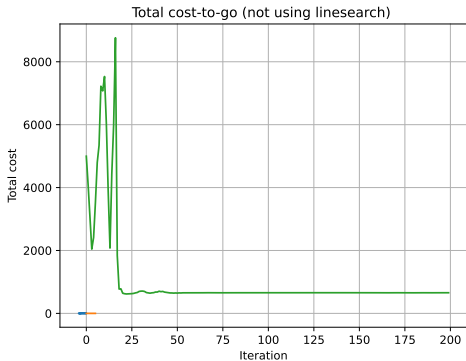
Algorithm 2 Basic iLQR

Require: Given initial state \mathbf{x}_0

- 1: Set $\bar{\mathbf{x}}_k = \mathbf{x}_0$, $\bar{\mathbf{u}}_k = \mathbf{0}$ (or a random vector), $L_k = \mathbf{0}$ and $\mathbf{l}_k = \mathbf{0}$
 - 2: $\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k \leftarrow$ FORWARD-PASS($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, L_k, \mathbf{l}_k$) ▷ Compute initial nominal trajectory using eq. (17.10)
 - 3: **for** $i = 0$ to a pre-specified number of iterations **do**
 - 4: $A_k, B_k, c_k, c_{x,k}, c_{u,k}, c_{xx,k}, c_{ux,k}, c_{uu,k} \leftarrow$ GET-DERIVATIVES($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k$)
 - 5: $L_k, \mathbf{l}_k \leftarrow$ BACKWARD-PASS($A_k, B_k, c_k, c_{x,k}, c_{u,k}, c_{xx,k}, c_{ux,k}, c_{uu,k}, \mu$)
 - 6: $J^{(i)} \leftarrow$ COST-OF-TRAJECTORY($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k$)
 - 7: $\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k \leftarrow$ FORWARD-PASS($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, L_k, \mathbf{l}_k$)
 - 8: **end for**
 - 9: Compute control law $\pi_k(\mathbf{x}_k) = \bar{\mathbf{u}}_k + \bar{\mathbf{l}}_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)$
 - 10: **return** $\{\pi_k\}_{k=0}^{N-1}$
 - 11: **function** FORWARD-PASS($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, L_k, \mathbf{l}_k$) ▷ Forward-simulation of dynamics
 - 12: Set $\mathbf{x}_0 = \bar{\mathbf{x}}_0$
 - 13: **for all** $k = 0, \dots, N - 1$ **do**
 - 14: $\mathbf{u}_k^* \leftarrow \bar{\mathbf{u}}_k + L_k(\mathbf{x}_{k-1} - \bar{\mathbf{x}}_k) + \mathbf{l}_k$ ▷ see eq. (17.16)
 - 15: $\mathbf{x}_{k+1} \leftarrow f_k(\mathbf{x}_k, \mathbf{u}_k^*)$
 - 16: **end for**
 - 17: **return** $\mathbf{x}_k, \mathbf{u}_k^*$
 - 18: **end function**
 - 19: **function** BACKWARD-PASS($A_k, B_k, c_k, c_{x,k}, c_{u,k}, c_{xx,k}, c_{ux,k}, c_{uu,k}, \mu$) eq. (17.14)
 - 20: Compute L_k, \mathbf{l}_k using dLQR with μ , algorithm 22 ▷ Obtain control law
 - 21: **end function**
 - 22: **function** COST-OF-TRAJECTORY($\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k$)
 - 23: **return** $c_N(\bar{\mathbf{x}}_N) + \sum_{k=0}^{N-1} c_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$
 - 24: **end function**
-

Basic iLQR: Pendulum swingup task

Pendulum starts at $\theta = \pi$ and $\dot{\theta} = 0$ and controller tries to swing it up $\theta = 0$



+ 🎮 lecture_06_pendulum_bilqr_L

+ 🎮 lecture_06_pendulum_bilqr_ubar

Iterative LQR

Basic iLQR is not very numerically stable. iLQR adds two ideas:

- Use regularization to stabilize the discrete LQR algorithm (μ)
- Search for policies that are **close** to the old ones. Recall:

$$\mathbf{u}_k^* = \bar{\mathbf{u}}_k + l_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)$$

- Since $(\mathbf{x}_k - \bar{\mathbf{x}}_k)$ assumed small (and L_k stabilized by μ), decreasing l_k means new control closer to old.
- Specifically, introduce $0 \leq \alpha \leq 1$

$$\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \alpha l_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k)$$

Iterative LQR Procedure

- Initialize regularization parameter to a fairly low value μ
- In the forward pass try smaller and smaller changes to trajectory (α -values)
- For each α -value check if the cost $J^{(i)}$ decreases relative to $J^{(i-1)}$. If so, *accept* this α and decrease the regularization parameter μ by a small amount
- If no α -value works, increase the regularization parameter μ by a small amount

iLQR Algorithm

Algorithm 3 iLQR

Require: Given initial state \mathbf{x}_0

- 1: $\mu_{\min} \leftarrow 10^{-6}$, $\mu_{\max} \leftarrow 10^{10}$, $\mu \leftarrow 1$, $\Delta_0 \leftarrow 2$ and $\Delta \leftarrow \Delta_0$
 - 2: Initialize $\tilde{\mathbf{x}}_k, \tilde{\mathbf{u}}_k$ as before
 - 3: **for** $i = 0$ to a pre-specified number of iterations **do**
 - 4: $A_k, B_k, C_k, c_{x,k}, c_{u,k}, c_{xx,k}, c_{ux,k}, c_{uu,k} \leftarrow \text{GET-DERIVATIVES}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{u}}_k)$
 - 5: $L_k, \mathbf{l}_k \leftarrow \text{BACKWARD-PASS}(A_k, B_k, C_k, c_{x,k}, c_{u,k}, c_{xx,k}, c_{ux,k}, c_{uu,k}, \mu)$
 - 6: $J' \leftarrow \text{COST-OF-TRAJECTORY}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{u}}_k)$
 - 7: **for** $\alpha = 1$ to a very low value **do**
 - 8: $\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k \leftarrow \text{FORWARD-PASS}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{u}}_k, L_k, \mathbf{l}_k, \alpha)$
 - 9: $J^{\text{new}} \leftarrow \text{COST-OF-TRAJECTORY}(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k)$
 - 10: **if** $J^{\text{new}} < J'$ **then**
 - 11: **if** $\frac{1}{J'} |J^{\text{new}} - J'| < \text{a small number}$ **then**
 - 12: Method has converged, terminate outer loop and return
 - 13: **end if**
 - 14: $J' \leftarrow J^{\text{new}}$
 - 15: $\tilde{\mathbf{x}}_k \leftarrow \hat{\mathbf{x}}_k$ and $\tilde{\mathbf{u}}_k \leftarrow \hat{\mathbf{u}}_k$
 - 16: **α accepted:** Update Δ and μ using eq. (17.19) ▷ Reduce regularization
 - 17: Break loop over α
 - 18: **end if**
 - 19: **end for**
 - 20: **if No α -value was accepted then**
 - 21: Update Δ and μ using eq. (17.18) ▷ Increase regularization
 - 22: **end if**
 - 23: **end for**
 - 24: Compute controller $\{\pi_k\}_{k=0}^{N-1}$ as before from L_k, \mathbf{l}_k
-

🔊 lecture_06_pendulum_ilqr_L

🔊 lecture_06_pendulum_ilqr_ubar

🔊 DTU Compute
lecture_06_cartpole

Iterative LQR

Given \mathbf{x}_0 and f_k, c_k, c_N ; initialize $\bar{\mathbf{u}}_k$

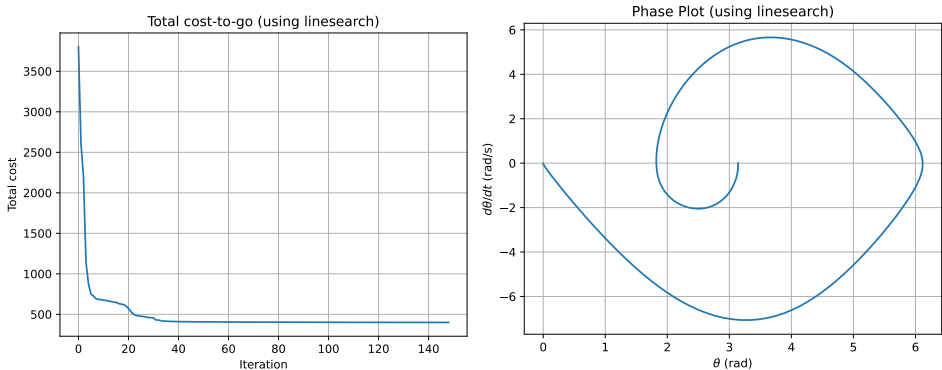
- Simulate $\bar{\mathbf{x}}_k$ and compute matrices for linearized problem as well as cost $J_{\bar{\mathbf{u}}}(\bar{\mathbf{x}}_0)$
- Solve for $\delta \mathbf{u}_k^*$ using regularization μ
- Loop over α starting at $\alpha = 1$
 - Obtain controls \mathbf{u}_k^* with α (see [TET12, Eq.(12)])

$$\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \alpha l_k + L_k(\mathbf{x}_k - \bar{\mathbf{x}}_k) \quad (7)$$

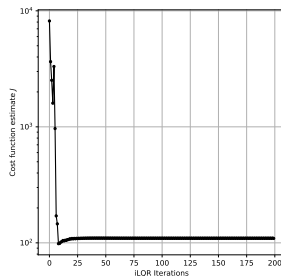
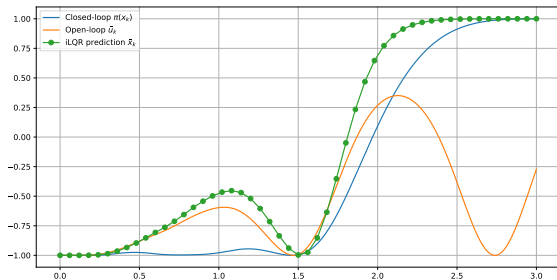
- If cost $J_{\mathbf{u}^*}(\mathbf{x}_0) < J_{\bar{\mathbf{u}}}(\bar{\mathbf{x}}_0)$ accept α /decrease μ
- (On failure to find α increase regularization μ)

Full iLQR: Pendulum swingup task

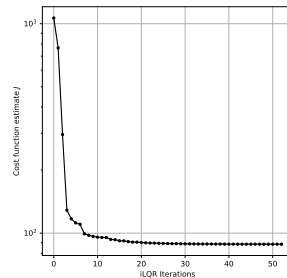
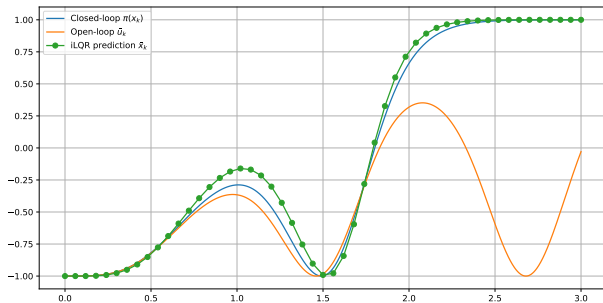
Pendulum starts at $\theta = \pi$ and $\dot{\theta} = 0$ and controller tries to swing it up $\theta = 0$



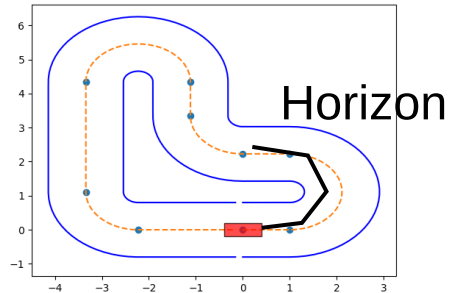
Basic iLQR Algorithm Example



iLQR Algorithm Example



Model Predictive Control



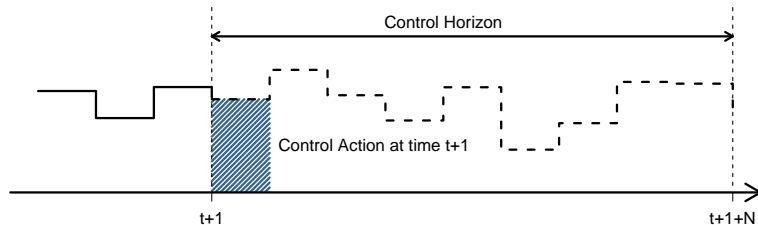
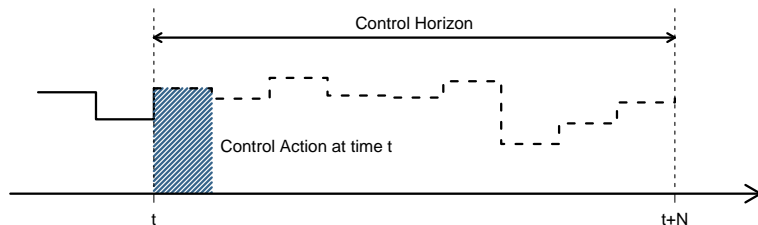
Model-predictive control/receding horizon control

Iteratively solve optimization problem on short time scale

- Long horizon equals great computation, uncertainty
- Solving problem on short horizon often sufficient

Model Predictive Control

- Solve control problem u_0, \dots, u_{N-1} for a **small** number of steps N
- Apply control u_0 from first step
- Repeat



Appendix: MPC can be understood as dynamical programming

DP applied in the starting state (**optimal**):

$$J^*(x_0) = \min_{u_0} \mathbb{E} [J_1^*(x_1) + g_0(x_0, u_0, w_0)]$$

d -step rollout of DP (**optimal**):

$$J^*(x_0) = \min_{\mu_0, \dots, \mu_{d-1}} \mathbb{E} \left[J_d^*(x_{k+d}) + \sum_{k=0}^{d-1} g_k(x_k, \mu_k(x_k), w_k) \right]$$

Deterministic simplification for control (**optimal**):

$$J^*(\mathbf{x}_0) = \min_{\mathbf{u}_0, \dots, \mathbf{u}_{d-1}} \left[J_d^*(\mathbf{x}_{k+d}) + \sum_{k=0}^{d-1} c_k(\mathbf{x}_k, \mathbf{u}_k) \right]$$

- **MPC: Approximate** $J_d^*(\mathbf{x}_{k+d})$ and just plan on d -horizon
- Re-plan at each step