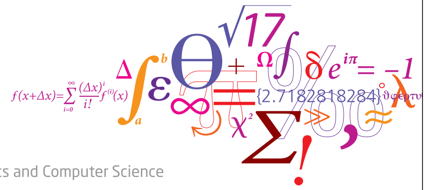
02465: Introduction to reinforcement learning and control

Linearization and iterative LQR

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)



DTU Compute
Department of Applied Mathematics and Computer Science

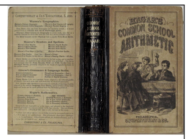
Lecture Schedule

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

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



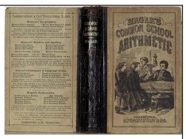
- Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a well-behaved function
- The **gradient** is defined as:

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

- 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



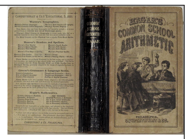
- Let $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a well-behaved multi-variate function defined as

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

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

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

Approximations



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

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

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

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

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

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

- The accumulated cost is:

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

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

Apply dynamical programming:

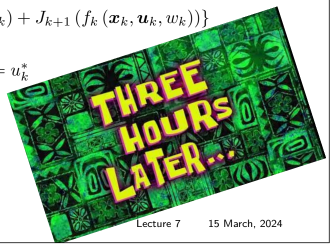
- Define $V_N \equiv Q_N$ and initialize:

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

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

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

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



LQR, simplified form

This gives the controller:

1. $V_N = Q_N$
2. $L_k = -(R_k + B_k^T V_{k+1} B_k)^{-1} (B_k^T V_{k+1} A_k)$
3. $V_k = Q_k + L_k^T R_k L_k + (A_k + B_k L_k)^T V_{k+1} (A_k + B_k L_k)$
4. $\boldsymbol{u}_k^* = L_k \boldsymbol{x}_k$
5. $J_k^*(\boldsymbol{x}_k) = \frac{1}{2} \boldsymbol{x}_k^T V_k \boldsymbol{x}_k$

Double Integrator Example

- True dynamics

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

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

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

- Quadratic cost function:

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

- Where:

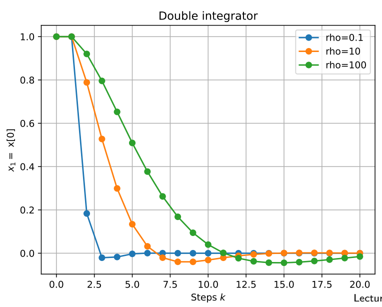
$$Q_k = Q_N = \begin{bmatrix} \frac{1}{\rho} & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1$$

Exponential integrator

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

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

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



The most general form of LQR

- General dynamics:

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

- General quadratic cost:

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

General discrete LQR algorithm

How to start living in luxury and never work again!

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

1. $V_N = Q_N$; $\boldsymbol{v}_N = \boldsymbol{q}_N$; $v_N = q_N$
2. $L_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{ux},k}$, $\quad \boldsymbol{l}_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{u},k}$

   $$S_{\boldsymbol{u},k} = \boldsymbol{r}_k + B_k^T \boldsymbol{v}_{k+1} + B_k^T V_{k+1} \boldsymbol{d}_k$$
   $$S_{\boldsymbol{uu},k} = R_k + B_k^T V_{k+1} B_k$$
   $$S_{\boldsymbol{ux},k} = H_k + B_k^T V_{k+1} A_k.$$
3. $$V_k = Q_k + A_k^T V_{k+1} A_k - L_k^T S_{\boldsymbol{uu},k} L_k$$
   $$\boldsymbol{v}_k = \boldsymbol{q}_k + A_k^T(\boldsymbol{v}_{k+1} + V_{k+1}\boldsymbol{d}_k) + S_{\boldsymbol{ux},k}^T \boldsymbol{l}_k$$
   $$v_k = v_{k+1} + q_k + \boldsymbol{d}_k^T \boldsymbol{v}_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1} \boldsymbol{d}_k + \frac{1}{2}\boldsymbol{l}_k^T S_{\boldsymbol{u},k}$$
4. $\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k$
5. $J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k$.

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

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

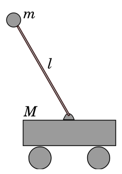
b. The linear-quadratic regulator is an example of model-free control

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

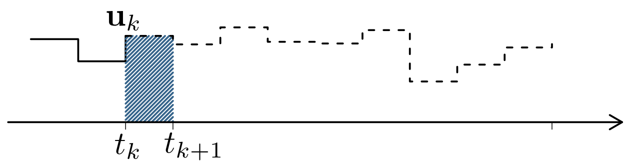
e. Don't know.

Controlling non-linear systems: Cartpole



- Continuous coordinates $\boldsymbol{x}(t) = \begin{bmatrix} x(t) & \dot{x}(t) & \theta(t) & \dot{\theta}(t) \end{bmatrix}$
- Action u is one-dimensional; the force applied to cart

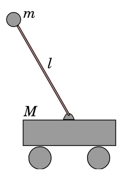
Discretization



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

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

Cartpole cost function



- We also apply a variable transformation:

$$\phi_x : \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix} \mapsto \begin{bmatrix} x & \dot{x} & \sin(\theta) & \cos(\theta) & \dot{\theta} \end{bmatrix}. \qquad (2)$$

- The cost function is of the form:

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

`lecture_06_linearize.py`

Controlling a non-linear system

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

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k$$
$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\boldsymbol{x}_k^\top Q \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^\top R \boldsymbol{u}_k + \cdots$$

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

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

Solution: Linearization!

Assume a general dynamics:

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

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

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

Simplifies to:

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

Linearization and iLQR

Algorithm 1 Linearized LQR

Require: Given a problem horizon N, and an expansion point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ corresponding to where the system should be
Compute $A_k, B_k, \boldsymbol{d}_k$ by expansion
Cost function is the same as usual because it is already quadratic
Use LQR, with dynamics A_k, B_k, d_k and cost matrices $Q_k, R_k, \boldsymbol{q}_k$ to obtain controller $L_k, \boldsymbol{l}_k$ for $k = 0, \dots, N-1$.
In a state $\boldsymbol{x}_k$, the control law is $\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k$

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

`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 $A_k (= J_{\boldsymbol{x}} \boldsymbol{f}_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))$ 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 $\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}$ giving a poor approximation
- **Idea:** Select expansion points $\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}$ near current trajectory $\boldsymbol{x}_k, \boldsymbol{u}_k$
- **How?**
  - Start with initial guess $\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k$ **(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 $\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k$ (**nominal trajectory**) for $k = 1, 2, \dots, N-1$

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

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

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

Back-substituting gives:

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

Expansion of the cost function

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

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

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

$$\begin{aligned} c_k &= c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \\ c_{\boldsymbol{x},k} &= \nabla_{\boldsymbol{x}} c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), \quad c_{\boldsymbol{u},k} = \nabla_{\boldsymbol{u}} c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \\ c_{\boldsymbol{xx},k} &= H_{\boldsymbol{x}} c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), \quad c_{\boldsymbol{uu},k} = H_{\boldsymbol{u}} c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \\ c_{\boldsymbol{ux},k} &= J_{\boldsymbol{x}} \nabla_{\boldsymbol{u}} c_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \end{aligned}$$

Expansion of the cost function

all in all we get a quadratic cost function:

$$c_k(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) = \frac{1}{2}\delta \boldsymbol{x}_k^\top c_{\boldsymbol{xx},k}\delta \boldsymbol{x}_k + c_{\boldsymbol{x},k}^\top \delta \boldsymbol{x}_k$$
$$+ \frac{1}{2}\delta \boldsymbol{u}_k^\top c_{\boldsymbol{uu},k}\delta \boldsymbol{u}_k + c_{\boldsymbol{u},k}^\top \delta \boldsymbol{u}_k + \delta \boldsymbol{u}_k^\top c_{\boldsymbol{ux},k}\delta \boldsymbol{x}_k + c_k$$
$$c_N(\delta \boldsymbol{x}_N) = \frac{1}{2}\delta \boldsymbol{x}_N^\top c_{\boldsymbol{xx},N}\delta \boldsymbol{x}_N + c_{\boldsymbol{x},N}^\top \delta \boldsymbol{x}_N + c_N$$

Linearized solution to actual controls

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

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

- Rearranging

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

- Or

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

Basic iLQR Algorithm

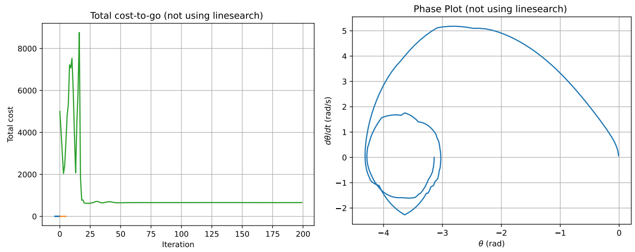
Algorithm 2 Basic iLQR

Require: Given initial state $\boldsymbol{x}_0$

Set $\bar{\boldsymbol{x}}_k = \boldsymbol{x}_0$, $\bar{\boldsymbol{u}}_k = \boldsymbol{0}$ (or a random vector), $L_k = \boldsymbol{0}$ and $\boldsymbol{l}_k = \boldsymbol{0}$
$\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k \leftarrow$ FORWARD-PASS$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, L_k, \boldsymbol{l}_k)$ ▷ Compute initial nominal trajectory using eq. (17.10) .
for $i = 0$ to a pre-specified number of iterations **do**
    $A_k, B_k, c_k, c_{\boldsymbol{x},k}, c_{\boldsymbol{u},k}, c_{\boldsymbol{xx},k}, c_{\boldsymbol{ux},k}, c_{\boldsymbol{uu},k} \leftarrow$ GET-DERIVATIVES$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$
    $L_k, \boldsymbol{l}_k \leftarrow$ BACKWARD-PASS$(A_k, B_k, c_k, c_{\boldsymbol{x},k}, c_{\boldsymbol{u},k}, c_{\boldsymbol{xx},k}, c_{\boldsymbol{ux},k}, c_{\boldsymbol{uu},k}, \mu)$
    $J^{(i)} \leftarrow$ COST-OF-TRAJECTORY$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$
    $\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k \leftarrow$ FORWARD-PASS$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, L_k, \boldsymbol{l}_k)$
end for
Compute control law $\pi_k(\boldsymbol{x}_k) = \bar{\boldsymbol{u}}_k + \bar{\boldsymbol{l}}_k + L_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)$
return $\{\pi_k\}_{k=0}^{N-1}$
function FORWARD-PASS$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, L_k, \boldsymbol{l}_k)$ ▷ Forward-simulation of dynamics
    Set $\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0$
    for all $k = 0, \dots, N-1$ **do**
        $\boldsymbol{u}_k^* \leftarrow \bar{\boldsymbol{u}}_k + L_k(\boldsymbol{x}\_k - \bar{\boldsymbol{x}}_k) + \boldsymbol{l}_k$ ▷ see eq. (17.16)
        $\boldsymbol{x}_{k+1} \leftarrow f_k(\boldsymbol{x}_k, \boldsymbol{u}_k^*)$
    end for
    return $\boldsymbol{x}_k, \boldsymbol{u}_k^*$
end function
function BACKWARD-PASS$(A_k, B_k, c_k, c_{\boldsymbol{x},k}, c_{\boldsymbol{u},k}, c_{\boldsymbol{xx},k}, c_{\boldsymbol{ux},k}, c_{\boldsymbol{uu},k}, \mu)$ eq. (17.14)
    Compute $L_k, \boldsymbol{l}_k$ using dLQR with μ, algorithm 22 ▷ Obtain control law
end function
function COST-OF-TRAJECTORY$(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$
    return $c_N(\bar{\boldsymbol{x}}_N) + \sum_{k=0}^{N-1} c_k(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$
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:

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

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

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

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 $\boldsymbol{x}_0$

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

`lecture_06_pendulum_ilqr_L`

`lecture_06_pendulum_ilqr_ubar`

`lecture_06_cartpole`

Iterative LQR

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

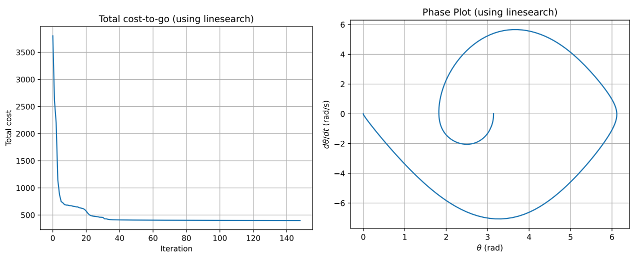
- Simulate $\overline{\boldsymbol{x}}_k$ and compute matrices for linearized problem as well as cost $J_{\overline{\boldsymbol{u}}}(\overline{x}_0)$
- Solve for $\delta \boldsymbol{u}_k^*$ using regularization μ
- Loop over α starting at $\alpha = 1$
    - Obtain controls $\boldsymbol{u}_k^*$ with α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

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

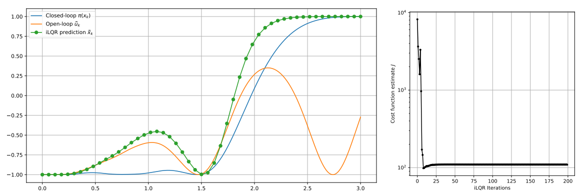
    - If cost $J_{\boldsymbol{u}^*}(\boldsymbol{x}_0) < J_{\overline{\boldsymbol{u}}}(\overline{x}_0)$ accept α/decrease μ
- (On failure to find α increase regularization μ)

Full iLQR: Pendulum swingup task

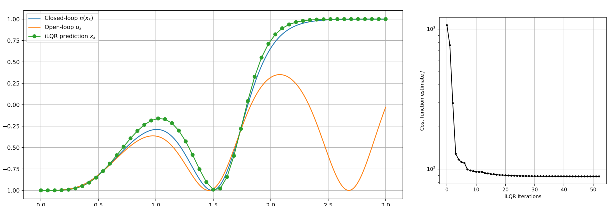
Pendulum starts at $\theta = \pi$ and $\dot{\theta} = 0$ and controller tries to swing it up $\theta = 0$



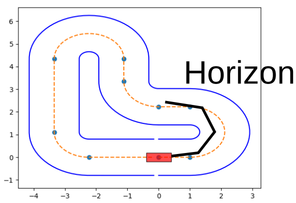
Basic iLQR Algorithm Example



iLQR Algorithm Example



Model Predictive Control



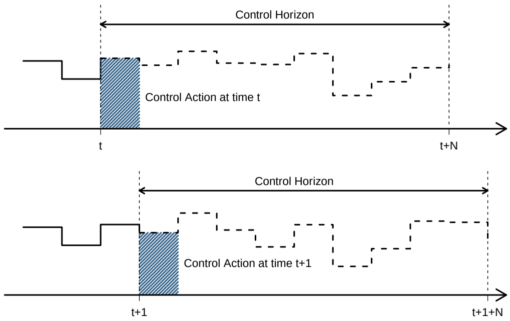
Model-predictive control/receding horizon control

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 $\boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1}$ for a small number of steps N
- Apply control $\boldsymbol{u}_0$ from first step
- Repeat



Appendix: MPC can be understood as dynamical programming

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

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

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

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

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

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

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