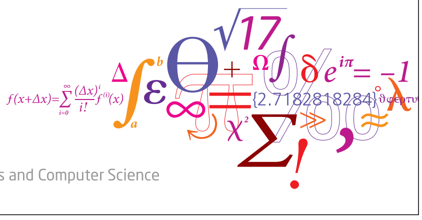
02465: Introduction to reinforcement learning and control

DP reformulations and introduction to Control

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)



DTU Compute
Department of Applied Mathematics and Computer Science

Lecture Schedule

Dynamical programming

1. The finite-horizon decision problem
   2 February
2. Dynamical Programming
   9 February
3. **DP reformulations and introduction to Control**
   16 February

Control

4. Discretization and PID control
   23 February
5. Direct methods and control by optimization
   1 March
6. Linear-quadratic problems in control
   8 March
7. Linearization and iterative LQR
   15 March

Reinforcement learning

8. Exploration and Bandits
   22 March
9. Policy and value iteration
   5 April
10. Monte-carlo methods and TD learning
    12 April
11. Model-Free Control with tabular and linear methods
    19 April
12. Eligibility traces and value-function approximations
    26 April
13. Q-learning and deep-Q learning
    3 May

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

Reading material:

- [Her24, Section 6.3; Chapter 10-11] Alternative formulations of DP

Learning Objectives

- Reformulations of DP
- The control problem
- Simulating a control problem

Recap: Discrete stochastic decision problem

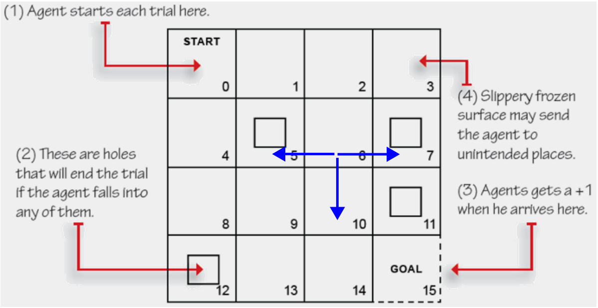
- The states are $x_0, \dots, x_N$, and the controls are $u_0, \dots, u_{N-1}$
- $w_k \sim P_k(W_k = w_k | x_k, u_k)$, $k = 0, \dots, N-1$ are random disturbances
- The system evolves as

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \quad k = 0, \dots, N-1$$

- At time k, the possible states/actions are $x_k \in S_k$ and $u_k \in \mathcal{A}_k(x_k)$

DP Recap: Frozen lake

If agent takes action **down** in square 6, it will slide in either of the blue directions with probability $\frac{1}{3}$



- Implementation: w_k is **'slide forward'**, **'slide left'**, **'slide right'**
- $p(w_k | x_k, u_k) = \frac{1}{3}$ and $f_k(x_k, u_k, w_k)$ computes effect of action + slide

---

The Dynamical Programming algorithm

The Dynamical Programming algorithm

For every initial state x_0, the optimal cost $J^*(x_0)$ is equal to $J_0(x_0)$, and optimal policy π^* is $\pi^* = \{\mu_0, \dots, \mu_{N-1}\}$, computed by the following algorithm, which proceeds backward in time from $k = N$ to $k = 0$ and for each $x_k \in S_k$ computes

$$J_N(x_N) = g_N(x_N) \tag{1}$$

$$J_k(x_k) = \min_{u_k \in \mathcal{A}_k(x_k)} \mathbb{E}_{w_k} \{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\} \tag{2}$$

$$\mu_k(x_k) = u_k^* \quad (u_k^* \textit{ is the } u_k \textit{ which minimizes the above expression}). \tag{3}$$

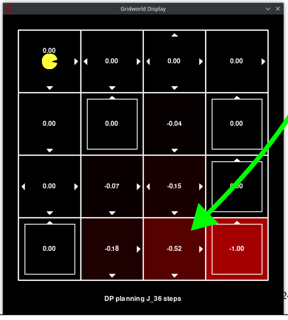
The optimal value function is expected future cost from a given state x_k at a given time k.

Quiz: Frozen lake

Consider the DP update equation:

$$J_k\left(x_k\right)=\min_{u_k \in \mathcal{A}_k\left(x_k\right)} \underset{w_k}{\mathbb{E}}\left\{g_k\left(x_k, u_k, w_k\right)+J_{k+1}\left(f_k\left(x_k, u_k, w_k\right)\right)\right\}$$

What will be the expected cost $J_{35}(x_k)$ of the indicated square? (hint: What action is best at this stage?)



0.00	0.00	0.00	0.00
0.00	0.00	-0.04	0.00
0.00	-0.07	-0.15	0.00
0.00	-0.18	-0.52	-1.00

a. $J_{35}(x_k)=-0.607$

b. $J_{35}(x_k)=-0.587$

c. $J_{35}(x_k)=-0.567$

d. $J_{35}(x_k)=-0.543$

e. Don't know.

Evaluate a policy

- Suppose the policy π is fixed
- We want to now how well it does

$$J_\pi(x_0)=\mathbb{E}_\pi\left[g_N(x_N)+\sum_{k=0}^{N-1} g_k(x_k,u_k,w_k) \mid x_0\right].$$

- Just move expectation:

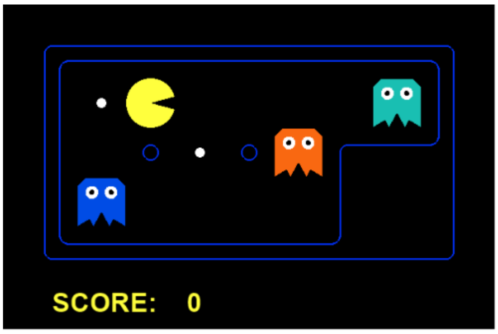
$$\begin{aligned} J_\pi(x_0) &= \mathbb{E}\left[g_0(x_0,u_0,w_0)+\mathbb{E}\left[g_N(x_N)+\sum_{k=1}^{N-1} g_k(x_k,u_k,w_k) \mid x_1\right]\right] \\ &= \mathbb{E}\left[g_0(x_0,u_0,w_0)+J_{1,\pi}(x_1)\right] \end{aligned}$$

- Initialize at $J_{N,\pi}(x_N)=g_N(x_N)$ and iterate:

$$J_{\pi,k}(x_k)=\mathbb{E}\left[g_k(x_k,u_k,w_k)+J_{k+1,\pi}(x_{k+1})\right]$$

- Applications: Many RL algorithms

The DP algorithm is often not practical



- Too many states! $\{\text{tiles}\}^{\{\text{players}\}} \times 2^{\{\text{pellets}\}}$
- We often don't know dynamics/distribution over opponents moves

---

N	J_0	Win pct	Length	$\lvert\mathcal{S}\rvert$
1	0.00	0.00	1.00	12.0
2	0.00	0.00	2.00	41.0
3	0.00	0.00	2.50	155.0
4	0.75	0.72	3.72	278.0
6	0.81	0.81	4.30	1098.0
8	0.82	0.82	4.33	3565.0
12	0.85	0.86	4.54	18956.0
16	0.85	0.84	4.51	37516.0
20	0.85	0.84	4.56	47811.0

Table: Results of the DP algorithm to the pacman level with three ghosts

---

Stationary problem = stationary policy

$$J_k(x_k)=\min_{u_k} \mathbb{E}\left[J_{k+1}\left(f_k(x_k,u_k,w_k)\right)+g_k\left(x_k,u_k,w_k\right)\right]$$

Assume the problem is independent of k:

$$J_k(x)=\min_{u} \mathbb{E}\left[J_{k+1}\left(f(x,u,w)\right)+g\left(x,u,w\right)\right]$$

- Will be true that $J_0 \approx J_1 \approx J_2$ etc.
- Policies will be the same initially $\pi_0 \approx \pi_1$ etc.
- The horizon N is irrelevant assuming it is *long enough*

In fact just iterate to convergence:

$$J(x) \leftarrow \min_{u} \mathbb{E}\left[J\left(f(x,u,w)\right)+g\left(x,u,w\right)\right]$$

Applications: This is nearly always the case.

Action-value formulation

$$J_k(x_k)=\min_{u_k} \mathbb{E}\left[J_{k+1}\left(f_k(x_k,u_k,w_k)\right)+g_k\left(x_k,u_k,w_k\right)\right]$$

Rewrite using $Q(x_k,u_k)$ as the expected cost

- Foundation of Q-learning
- If we know the Q-functions, they give us the policy for free

Robust control

$$J_k(x_k) = \min_{u_k} \mathbb{E}\left[J_{k+1}\left(f_k(x_k, u_k, w_k)\right) + g_k\left(x_k, u_k, w_k\right)\right]$$

- **Problem:** What if we don't know $p(w_k|x_k, u_k)$?
- **Assumes the worst possible thing always happen**

$$J_k(x_k) = \min_{u_k}\left[\arg\max_{w_k}\left[J_{k+1}\left(f_k(x_k, u_k, w_k)\right) + g_k\left(x_k, u_k, w_k\right)\right]\right]$$

RL Most game-playing methods (Alphago-zero, TD-gammon, etc.)

Control Robust control

Games (imperfect information, Nash-equilibrium) are generally a fairly open problem in RL [BBLG20]

Sample-based formulation

$$J_k(x_k) = \min_{u_k} \mathbb{E}\left[J_{k+1}\left(f_k(x_k, u_k, w_k)\right) + g_k\left(x_k, u_k, w_k\right)\right]$$

- **Problem:** What if we **really** don't know $P(w_k|x_k, u_k)$?
- **Idea:** We can sample from it

$$J_k(x_k) \approx \min_{u_k} \frac{1}{S}\sum_{s=1}^{S}\left[J_{k+1}\left(f_k(x_k, u_k, w^{(s)})\right) + g_k\left(x_k, u_k, w^{(s)}\right)\right]$$

Foundation of RL: **Samples can be obtained by just observing what nature does in a state** (x_k, u_k)

Approximate dynamical programming

We solve the following problem at each step k

$$J_k(x_k) = \min_{u_k} \mathbb{E}\left[J_{k+1}\left(x_{k+1}\right) + g_k\left(x_k, u_k, w_k\right)\right]$$

To many damn states! (...although calculation for a single x_k is ok..)

- **Idea:** Use an approximating function $J_k(x_k) \approx \tilde{J}(x_k, \boldsymbol{w})$
- **How?:** The right-hand side gives us a *prediction* y_k for x_k which we use to **train** $\boldsymbol{w}_k$

$$\boldsymbol{w}^* = \min_{\boldsymbol{w}} \sum_{s=1}^{S}\left(y^{(s)} - \tilde{J}(x^{(s)}, \boldsymbol{w})\right)^2$$

This is the idea behind deep RL, and has applications to control and DP-based planning

---

d-step methods

DP applied in the starting state:

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

d-step rollout of DP:

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

Instead of using J_d^*, perhaps use a **really** rough approximation

RL n-step methods (Impala, Alphastar, etc.)

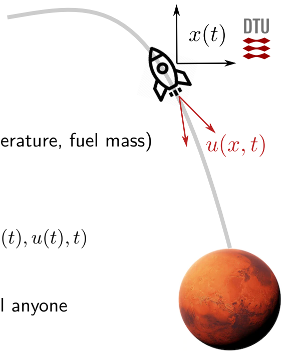
Control Model-predictive control

- Often just ignore the terminal cost
- Often just assume model is deterministic
- Both assumptions are justifiable because the model wrong anyway

---

Control theory

Example: Mars landing



Time Continuous

State/Actions $x(t)$: (Position, velocity, temperature, fuel mass)
$u(t)$: thruster outputs

Dynamics Smooth and time-dependent

$$\dot{x}(t) = f(x(t), u(t), t)$$

Cost Land the right place,
and use little fuel **and** don't kill anyone

Constraints Thrusters deliver limited force,
ship cannot go into mars, etc.

Objective Determine $u(t)$ to minimize final cost

Really important constraints; no learning

`lecture_01_car_random.py`

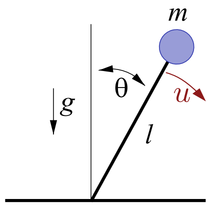
Control theory in general

- Why care?
  - More mature and practically important than RL
  - Ideas in control relevant for RL and beyond
- This course will teach **naive** but **real** control theory:
  - **Don't care about error analysis/analytical properties**
  - **Will emphasize real methods**
  - **Will distinguish between approximate model of environment/actual environment**

Differences and similarities to dynamical programming

- Similarities
  - A time-dependent problem
  - States and actions
  - Goal is still to minimize a cost function
  - Ideas from DP will carry over
- **Complications**
  - Time is continuous $t \in [t_0, t_F]$
  - Dynamics is an ODE
- **Simplifications**
  - No noise
  - Open-loop techniques play a more prominent role

Example: The pendulum environment



If u is a torque applied to the axis of rotation θ then:

$$\ddot{\theta}(t) = \frac{g}{l}\sin(\theta(t)) + \frac{u(t)}{ml^2}$$

If $\boldsymbol{x} = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^T$ this can be written as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \frac{g}{l}\sin(\theta) + \frac{u}{ml^2} \end{bmatrix} = f(\boldsymbol{x}, u) \tag{4}$$

`lecture_04_pendulum_random.py`

Dynamics

We assume the system we wish to control has dynamics of the form

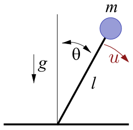
$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$$

- $\boldsymbol{x}(t) \in \mathbb{R}^n$ is a complete description of the system at t
- $\boldsymbol{u}(t) \in \mathbb{R}^d$ are the controls applied to the system at t
- The time t belongs to an interval $[t_0, t_F]$ of interest
- The evolution of the system $\boldsymbol{x}(t), \boldsymbol{u}(t)$ is called a **path** or **trajectory**

---

quiz_pendulum

Quiz: Stopping the pendulum



$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \frac{g}{l}\sin(\theta) + \frac{u}{ml^2} \end{bmatrix} = f(\boldsymbol{x}, u)$$

If the pendulum is at an angle of $\frac{\pi}{4}$ to vertical, how much torque should we apply to keep it still?

a. $u(t) = -\frac{mg}{\sqrt{2}}$

b. $u(t) = -\frac{m}{g\sqrt{2}}$

c. $u(t) = -\frac{mg\sqrt{2}}{l^2}$

d. $u(t) = -\frac{g\sqrt{2}}{ml}$

e. Don't know.

Constraints I

Any realistic physical system has constraints. Examples:

- Simple boundary constraints

$$\boldsymbol{x}_{\text{low}} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\text{upp}}$$
$$\boldsymbol{u}_{\text{low}} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{\text{upp}}$$

Maximal acceleration of a car; that the acceleration of an airplane cannot exceed a certain safety limit

- Problem must terminate within a given time

$$t_{\text{low}} \leq t_0 < t_F \leq t_{\text{upp}}$$

(or we could know t_0 and t_f; note this is different from DP case with x_0 and N!)
Don't take forever

Constraints II

- Boundary constraints

$$\begin{aligned}\boldsymbol{x}_{0,\,\text{low}} &\leq \boldsymbol{x}(t_0) \leq \boldsymbol{x}_{0,\,\text{upp}} \\ \boldsymbol{x}_{F,\,\text{low}} &\leq \boldsymbol{x}(t_F) \leq \boldsymbol{x}_{F,\text{upp}}\end{aligned}$$

I want you to be somewhere when you start or end

- Notice that for some coordinate the two boundaries can be equal to give equality constraints; they can also be ∞ for unconstrained problems

---

Cost and policy

- State/action trajectories $\boldsymbol{x}, \boldsymbol{u}$ which satisfy the constraints are said to be **admissible**
- The cost function will be of this form:

$$J_{\boldsymbol{u}}(\boldsymbol{x}, t_0, t_F) = \underbrace{c_F\left(t_0, t_F, \boldsymbol{x}\left(t_0\right), \boldsymbol{x}\left(t_F\right)\right)}_{\text{Mayer Term}} + \underbrace{\int_{t_0}^{t_F} c(\tau, \boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) d\tau}_{\text{Lagrange Term}}$$

- Note we sometimes write this as $J_{\boldsymbol{u}}(\boldsymbol{x}_0, t_0, t_F)$
- Very often $t_0 = 0$

Special cases

- Minimum time $c_F = 0$, $c = 1$ and

$$\text{cost } = \int_{t_0}^{t_f} 1 d\tau = (t_f - t_0)$$

- Coordinate 3 takes a particular value $c_F(\cdots) = (x_3(t_f) - x_0)^2$, $c = 0$ and

$$\text{cost } = (x_3(t_f) - x_0)^2$$

- Minimize energy used $c(\cdots) =$ force × distance

$$\text{cost } = \int_{t_0}^{t_f} (\text{force} \times \text{velocity}) d\tau = \text{energy}$$

The continuous-time control problem

Given system dynamics for a system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$$

Obtain $\boldsymbol{u} : [t_0; t_F] \rightarrow \mathbb{R}^m$ as solution to

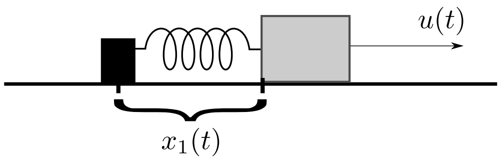
$$\boldsymbol{u}^*, \boldsymbol{x}^*, t_0^*, t_F^* = \underset{\boldsymbol{x}, \boldsymbol{u}, t_0, t_F}{\arg\min}\, J_{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{u}, t_0, t_F).$$

(Minimization subject to all constraints)

Today:

- Simulate the system

Example: The harmonic oscillator



A mass attached to a spring which can move back-and-forth

$$\ddot{x}(t) = -\frac{k}{m} x(t) + \frac{1}{m} u(t) \tag{5}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \tag{6}$$

$$J(\boldsymbol{x}_0) = \int_0^{t_F} \left(\boldsymbol{x}(t)^\top \boldsymbol{x}(t) + u(t)^2 \right) dt. \tag{7}$$

`lecture_04_harmonic.py`

Simulation: Euler integration

Apply a Taylor expansion:

$$\boldsymbol{x}(t + \delta) = \boldsymbol{x}(t) + \dot{\boldsymbol{x}}(t)\delta + \frac{1}{2}\ddot{\boldsymbol{x}}(t)\delta^2 + \mathcal{O}(\delta^3)$$

Define $\Delta = \frac{t_F - t_0}{N}$ and introduce

$$\begin{aligned} t_1 &= t_0 + \Delta \\ t_2 &= t_0 + 2\Delta \\ t_k &= t_0 + k\Delta \\ t_N &= t_0 + N\Delta = t_F \end{aligned}$$

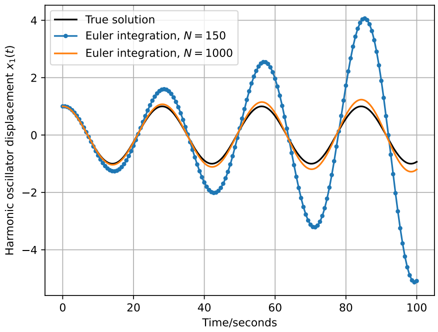
Then we can iteratively update:

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

Practical issues

A harmonic oscillator with no force $\ddot{x} = -\frac{k}{m}x$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \boldsymbol{x}_k, \quad \Delta = \frac{t_F}{N}. \tag{8}$$

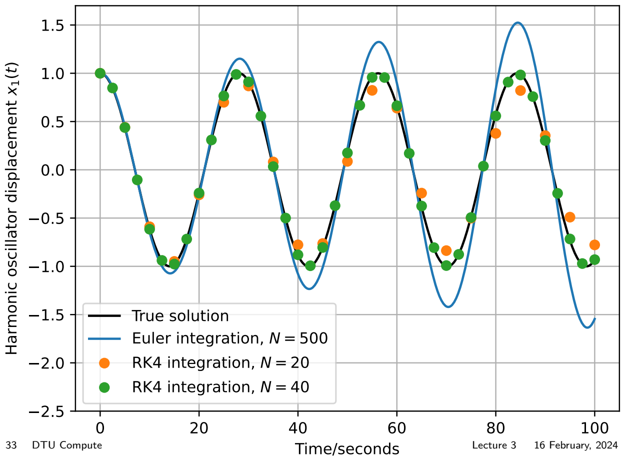


Simulation: Runge-Kutta 4 (RK4)

- Discretize time similar to Euler $t_k = t_0 + k\Delta$
- Compute

$$\begin{aligned} k_1 &= \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) \\ k_2 &= \boldsymbol{f}\left(\boldsymbol{x}_k + \Delta \tfrac{k_1}{2}, \boldsymbol{u}\left(t_k + \tfrac{\Delta}{2}\right), t_k + \tfrac{\Delta}{2}\right) \\ k_3 &= \boldsymbol{f}\left(\boldsymbol{x}_k + \Delta \tfrac{k_2}{2}, \boldsymbol{u}\left(t_k + \tfrac{\Delta}{2}\right), t_k + \tfrac{\Delta}{2}\right) \\ k_4 &= \boldsymbol{f}\left(\boldsymbol{x}_k + \Delta k_3, \boldsymbol{u}(t_{k+1}), t_{k+1}\right) \end{aligned}$$

- Set $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \frac{1}{6}\Delta\left(k_1 + 2k_2 + 2k_3 + k_4\right)$
- Repeat for all k



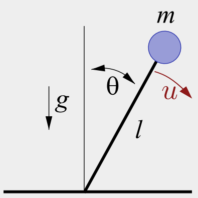
Implementation: Continuous symbolic model

```python
# basic_pendulum.py
class BasicPendulumModel(ControlModel):
    def sym_f(self, x, u, t=None):
        g = 9.82
        l = 1
        m = 2
        theta_dot = x[1]  # Parameterization: x = [theta, theta']
        theta_dot_dot = g / l * sym.sin(x[0]) + 1 / (m * l ** 2) * u[0]
        return [theta_dot, theta_dot_dot]

    def get_cost(self) -> SymbolicQRCost:
        return SymbolicQRCost(Q=np.eye(2), R=np.eye(1))

    def u_bound(self) -> Box:
        return Box(np.asarray([-10]), np.asarray([10]))

    def x0_bound(self) -> Box:
        return Box(np.asarray( [np.pi, 0] ), np.asarray( [np.pi, 0]))
```



Implements:

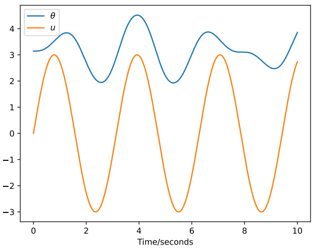
- $\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \boldsymbol{f}\left(\begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, u\right) = \begin{bmatrix} \dot{\theta} \\ \frac{g}{l}\sin(\theta) + \frac{1}{ml^2}u \end{bmatrix}$
- $J(\boldsymbol{x}_0) = \int_{t_0}^{t_F} \left(\frac{1}{2}\boldsymbol{x}(t)^\top Q \boldsymbol{x}(t) + \frac{1}{2}u(t)^\top R u(t)\right) dt = \frac{1}{2}\int_{t_0}^{t_F} \left(\|\boldsymbol{x}(t)\|^2 + u(t)^2\right) dt$
- $-10 \leq u(t) \leq 10$, and $\boldsymbol{x}_0 = \begin{bmatrix} \pi \\ 0 \end{bmatrix}$

Simulation

```python
# chapter7contiuous/model_example_plot.py
cmodel = PendulumModel()
x0 = cmodel.x0_bound().low

def policy(x, t):
    return [3 * np.sin(2 * t)]

xx, uu, tt = cmodel.simulate(x0, policy, t0=0, tF=10)
plt.plot(tt, xx[:, 0], label="$\\theta$")
plt.plot(tt, uu[:, 0], label="$u$")
```



Resources and references

https://en.wikipedia.org Overview of alternative discretization approaches of a ODE to discrete system (https://en.wikipedia.org/wiki/Discretization)

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong.
Combining deep reinforcement learning and search for imperfect-information games, 2020.

Tue Herlau.
Sequential decision making.
(Freely available online), 2024.