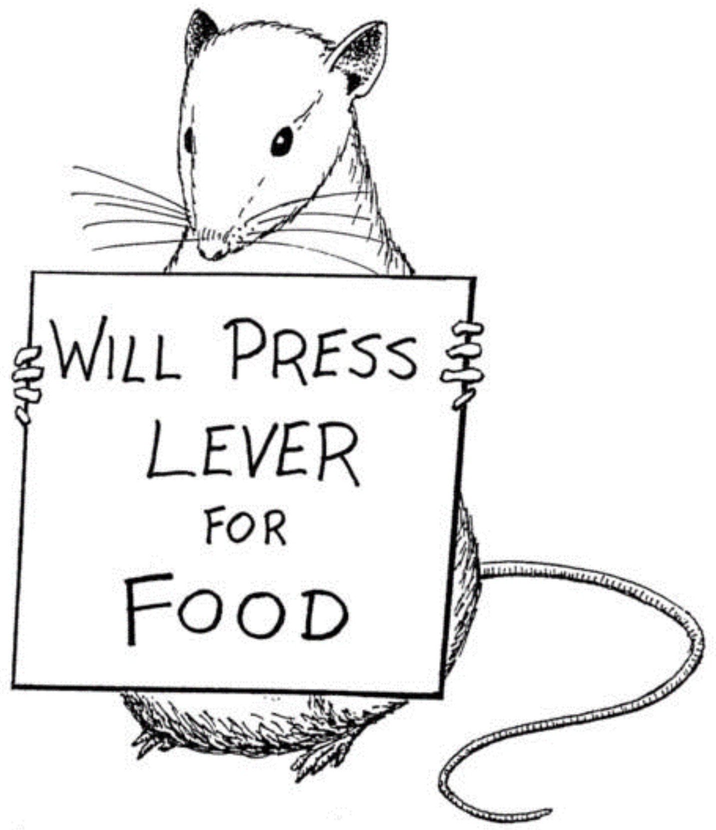
CS 277: Control and Reinforcement Learning

Winter 2026

Lecture 11: Planning



Roy Fox
Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine

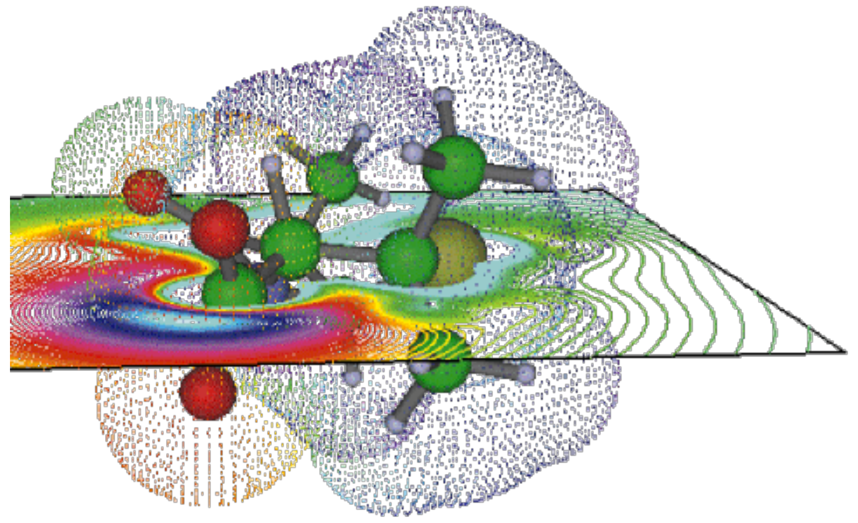
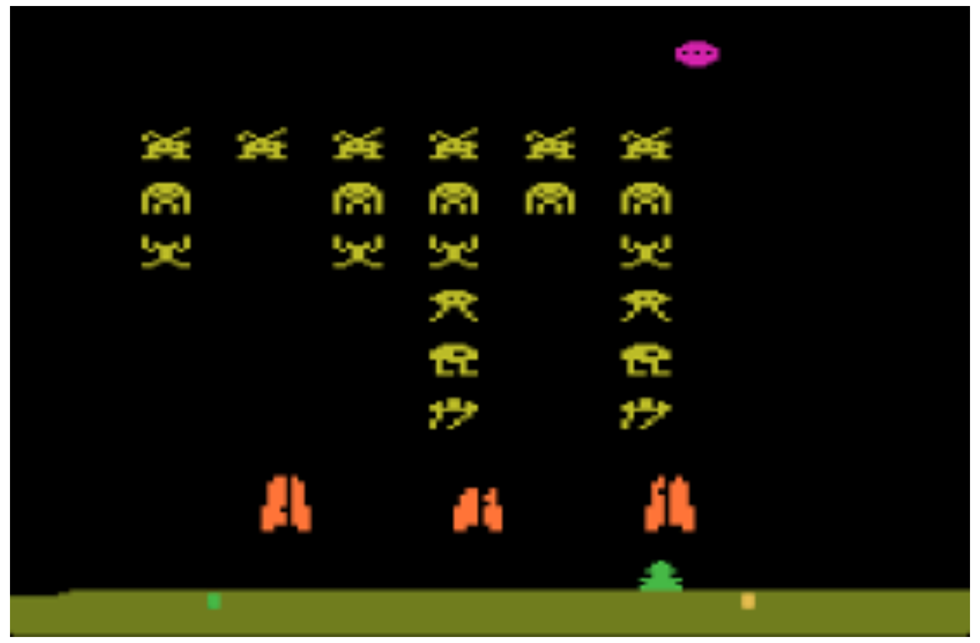
Logistics

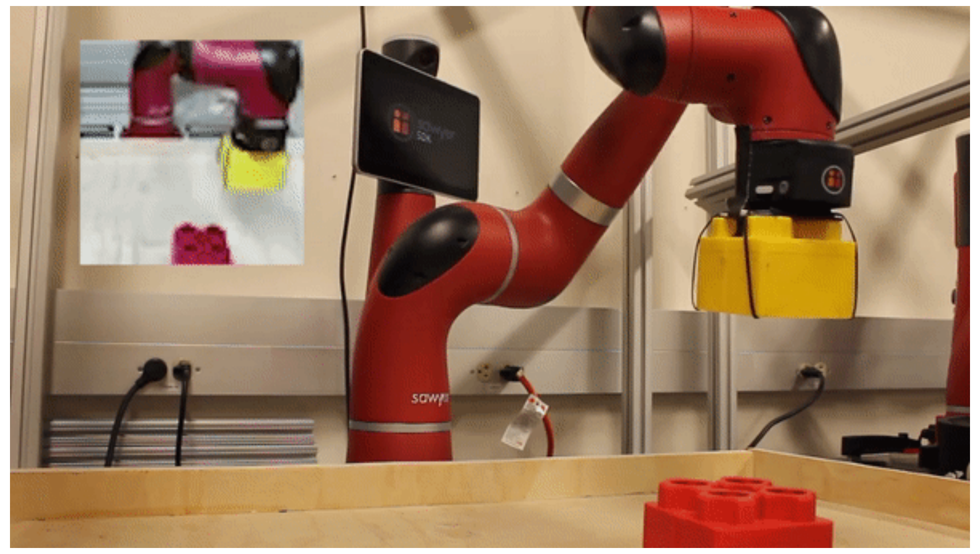
assignments

- Exercise 3 due Monday
- Quiz 6 due next Friday

Planning

- Planning: finding a good policy π when we "know" the MDP model
  - MDP = dynamics + reward function
- When do we "know" the model?
  - Well-modeled environments
    - Dynamics equations
  - Simulators
  - Learned models
  - System identification: the agent itself learns a model



Levels of "knowing" a model

- What does it mean to have a "known" model?
  - A really fast simulator
    - Analytic model, fast implementation, parallelization, approximate (high-level) model
  - A simulator that can be reset to any given state ("teleporting robot")
    - Sample $p(s' | s, a)$ for any (s, a), rather than an entire trajectory $p_\pi(\xi)$ with $s \sim p_\pi$
  - An analytic model (e.g. equations) that can be manipulated symbolically
  - A differentiable model
    - Backprop gradients through p

- Fast
- Resettable
- Differentiable

How to use a really fast simulator

- Any RL algorithm can benefit from more data

Algorithm MC model-free RL

Initialize some policy π
repeat
  Initialize some value function Q
  repeat to convergence
    Sample $\xi \sim p_\pi$
    Update $Q(s_t, a_t) \to R_{\geq t}(\xi)$ for all $t \geq 0$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ for all s

- Fast
- Resettable
- Differentiable

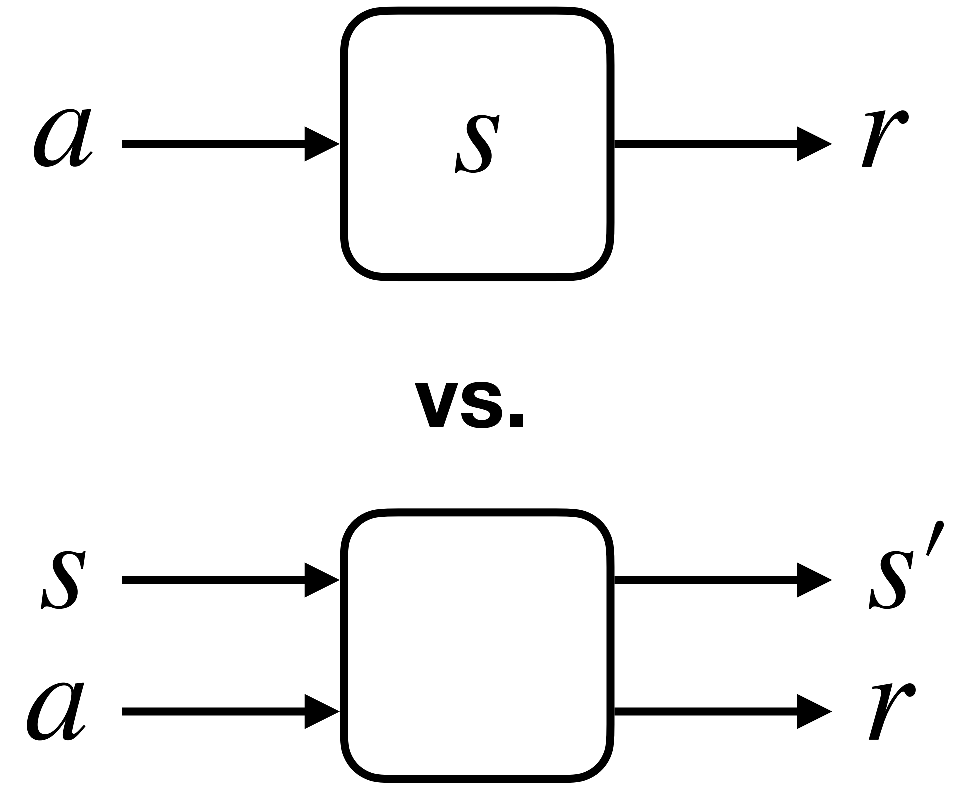
- Simple, unbiased, consistent algorithm
- High variance $\Rightarrow$ with fast simulator, can sample many trajectories

How to use an arbitrary-reset simulator (1)

- Arbitrary-reset simulator allows sampling from $(s' | s, a) \sim p$ for any (s, a) we want

- Small state space — can run Value Iteration with tabular parametrization:

$$V(s) \leftarrow \max_a r(s, a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V(s')]$$



- Large state space — should we use Fitted Value Iteration?

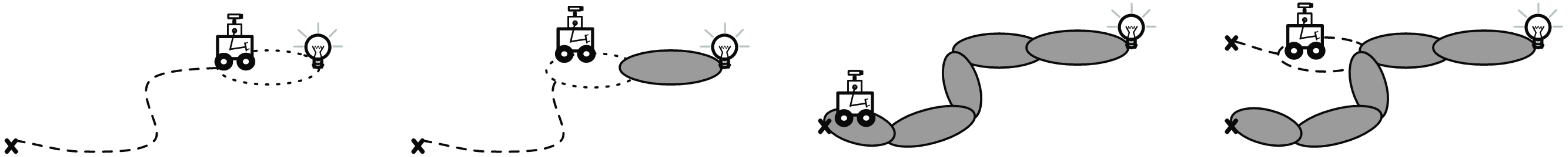
$$\mathcal{L}_\theta(s) = (\min_a r(s, a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V_{\bar{\theta}}(s')] - V_\theta(s))^2$$

  - Arbitrary state distribution can help exploration, variance reduction
  - Problem: must have $s \sim p_\theta(\xi)$, or suffer covariate shift (train–test mismatch)
    - Solution: use importance sampling

- Fast
- Resettable
- Differentiable

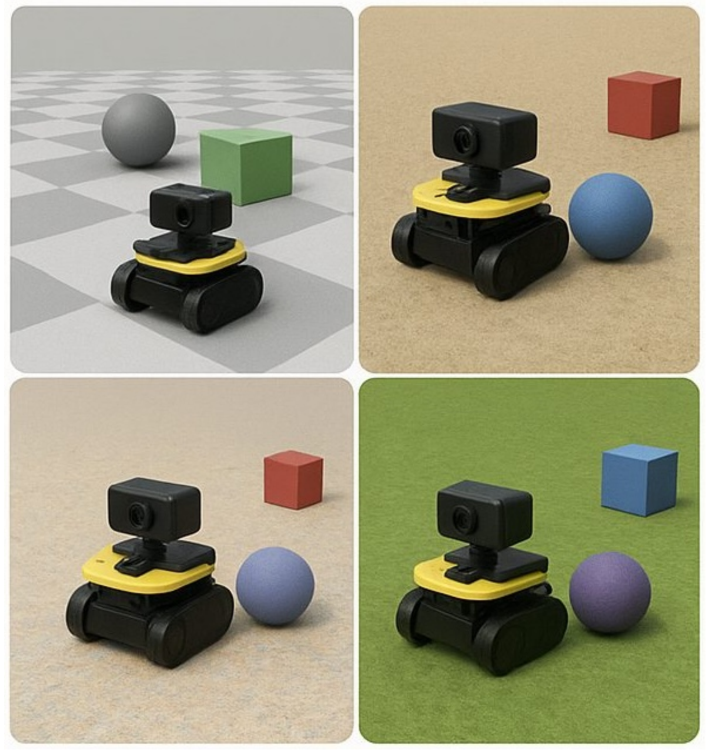
How to use an arbitrary-reset simulator (2)

- Curriculum learning



(a) (b) (c) (d)

  - Idea: reset to increasingly hard initial states



- Data augmentation
  - Idea: perturb the initial state and/or transitions to diversify data
  - Usually requires tight integration with the simulator

- Fast
- Resettable
- Differentiable

[Konidaris et al., Robot learning from demonstration by constructing skill trees, IJRR 2012]

Deterministic dynamics: stochastic optimization

- With deterministic dynamics, we can fully predict future states
  - Open-loop control: policy doesn't depend on observations = sequence of actions

$$\max_{\vec{a}} R(\vec{a}) = \max_{\vec{a}} r(s_0, a_0) + \gamma r(f(s_0, a_0), a_1) + \gamma^2 r(f(f(s_0, a_0), a_1), a_2) + \cdots$$

- Use any black-box optimizer; e.g. stochastic optimization:

Algorithm Stochastic optimization

Initialize π
repeat
  Sample $\vec{a}_1, \dots, \vec{a}_k \sim \pi$
  Run model to get returns $R_1, \dots, R_k$
  Select k/c top returns
  Fit π to these "elites"

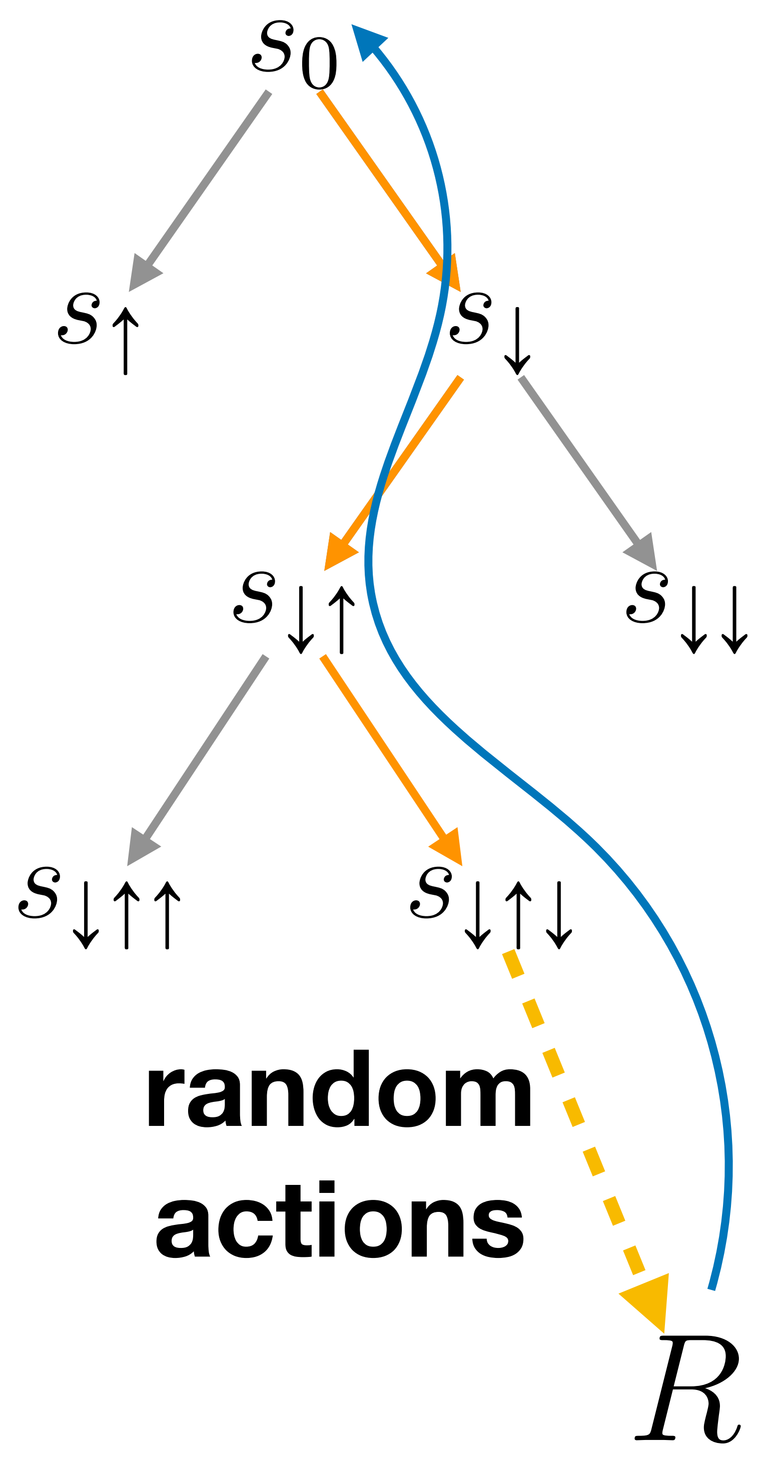
- Scales poorly with the dimension of $\vec{a}$

Discrete + deterministic: search

- Search can work in discrete action spaces with deterministic dynamics
- A*:
  - Maintain priority queue of visited states with return $R(s)$ to reach them
  - Pop state s with largest total predicted return $R(s) + \hat{V}(s)$
  - For each action a:
    - Compute new return $R(s) + r(s, a)$ for $s' = f(s, a)$
    - Update $R(s')$ if new path is better
- Improvements using learning, e.g.: (1) learn $\hat{V}$, (2) use $\hat{Q}$ to not expand all a

Discrete action space: optimal exploration

- Action sequences have a tree structure
  - Shallow (short) prefixes are visited often ⇒ possible to learn their value
  - Deep (long) sequences are visited rarely ⇒ we can only explore
- Monte Carlo Tree Search (MCTS):
  - Select leaf of the already-learned subtree
  - Explore to end of episode
  - Update nodes along branch to leaf



- Selecting a leaf: recursively maximize

$$\begin{cases} \infty & \text{if } N_{\text{visits}}(\text{child}) = 0 \\ V(\text{child}) + C\underbrace{\sqrt{\frac{\log N_{\text{visits}}(\text{self})}{N_{\text{visits}}(\text{child})}}}_{\text{exploration bonus}} & \text{otherwise} \end{cases}$$

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

interesting dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

- Fast
- Resettable
- Differentiable

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

captures linear dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

- Fast
- Resettable
- Differentiable

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

interesting dependence on x_t and u_t

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$
$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t + O(\epsilon^2)$$

linear dependence on x_t and u_t

optimal control: ∞

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

NOW we can neglect these

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t$$

$$+\frac{1}{2}(\delta x_t^\intercal (\nabla_x^2 \hat{c}_t)\delta x_t + \delta u_t^\intercal (\nabla_u^2 \hat{c}_t)\delta u_t + 2\delta x_t^\intercal (\nabla_{xu} \hat{c}_t)\delta u_t) + O(\epsilon^3)$$

Iterative LQR (iLQR)

Algorithm iLQR

Initialize $\hat{x}, \hat{u}$

repeat

Set $A, B \leftarrow \nabla_x \hat{f}_t, \nabla_u \hat{f}_t$ linearize dynamics around current trajectory $(\hat{x}, \hat{u})$

Set $Q, R, N, q, r \leftarrow \nabla_x^2 \hat{c}_t, \nabla_u^2 \hat{c}_t, \nabla_{xu} \hat{c}_t, \nabla_x \hat{c}_t, \nabla_u \hat{c}_t$ quadratic cost approximation around $(\hat{x}, \hat{u})$

$\hat{L}_t, \hat{\ell}_t \leftarrow$ LQR on $\delta x_t = x_t - \hat{x}_t$, $\delta u_t = u_t - \hat{u}_t$ ← place “origin” at $(\hat{x}, \hat{u})$

$\delta\hat{x}, \delta\hat{u} \leftarrow$ execute policy $\delta u_t = \hat{L}_t \delta x_t + \hat{\ell}_t$ in env

$\hat{x} \leftarrow \hat{x} + \delta\hat{x}$, $\hat{u} \leftarrow \hat{u} + \delta\hat{u}$

roll out to get new trajectory $(\hat{x}, \hat{u})$

- Fast
- Resettable
- Differentiable

Newton's method

- Compare iLQR with Newton's method for optimizing $\min_x f(x)$

Algorithm Newton's method

---

repeat

$$g \leftarrow \nabla_x f(\hat{x})$$
$$H \leftarrow \nabla_x^2 f(\hat{x})$$
$$\hat{x} \leftarrow \operatorname{argmin}_x \tfrac{1}{2}\delta x^\intercal H \delta x + g^\intercal \delta x$$

- iLQR approximates this method for $\min_u \mathcal{J}(u)$
- This would be exact if we expanded the dynamics to 2nd order
  - Similar to iLQR, called Differential Dynamic Programming (DDP)

- Fast
- Resettable
- Differentiable