Solving Continuous MDPs: The Linear Quadratic Regulator (LQR)

Sanjiban Choudhury

Cornell Bowers CIS Computer Science

The Big Picture

Case 1: Known MDP

Planning!

Case 2: Unknown MDP, requires feedback from environment

Reinforcement Learning!

Case 3: Unknown MDP, requires feedback from expert

Imitation Learning!

The Big Picture

Case 1: Known MDP

Planning!

Case 2: Unknown MDP, requires feedback from environment

Reinforcement Learning!

We explored this a bit ...

Case 3: Unknown MDP, requires feedback from expert

Imitation Learning!

The Big Picture

Case 1: Known MDP

Now let's go deeper here!

Planning!

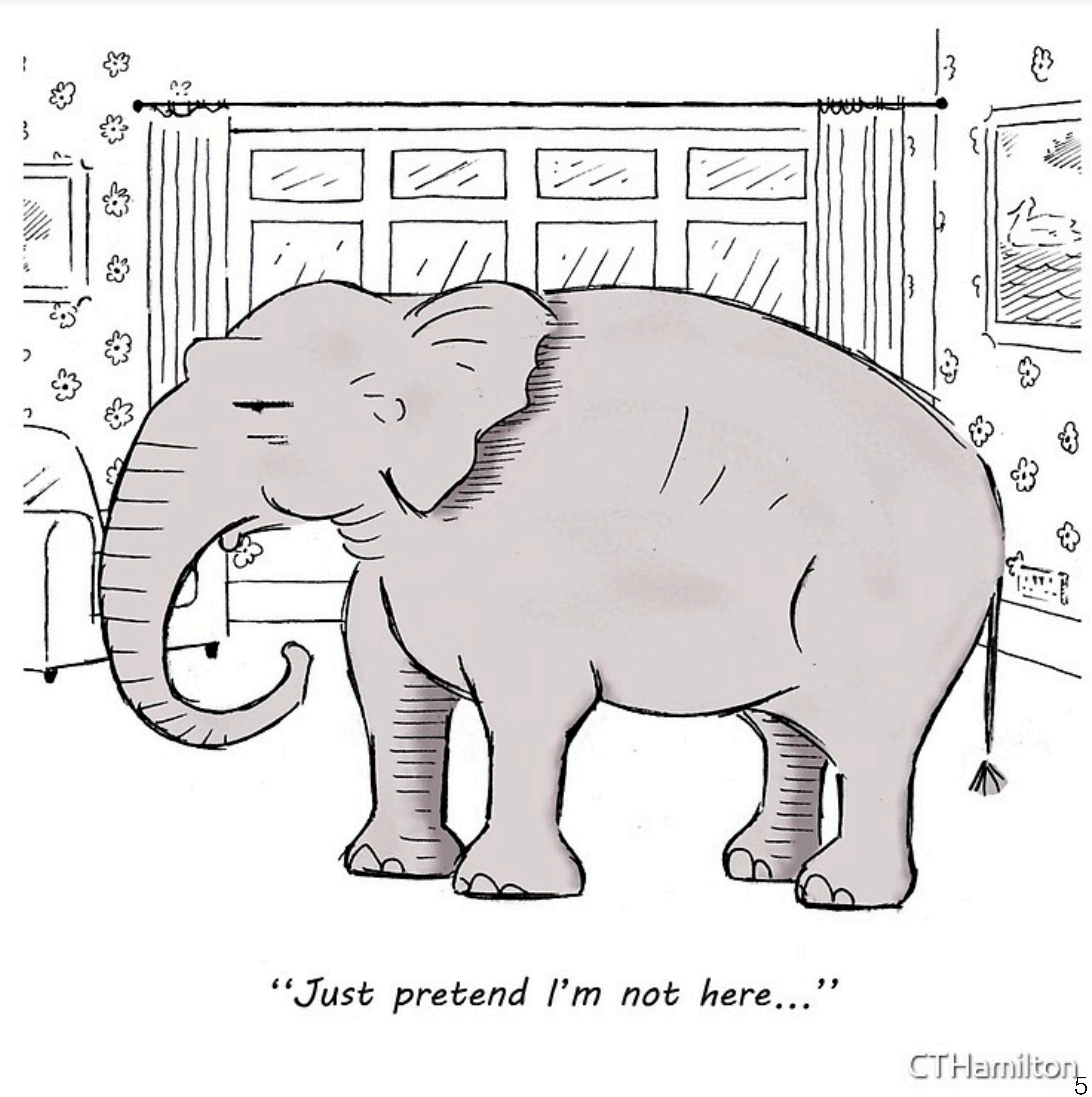
Case 2: Unknown MDP, requires feedback from environment

Reinforcement Learning!

Case 3: Unknown MDP, requires feedback from expert

Imitation Learning!

RL
=
Learn model
+
Plan with model



Model-based Planning

Step 0: Build a robot

Step 1: Collect data of your robot doing stuff in the world

Step 2: Use data to learn a dynamics model for your robot

Step 3: Plan with the model to compute an optimal policy

Today's Challenge!

Step 0: Build a robot

Step 1: Collect data of your robot doing stuff in the world

Step 2: Use data to learn a dynamics model for your robot

$\boxed{\text{Step 3: Plan with the model to compute an optimal policy}}$

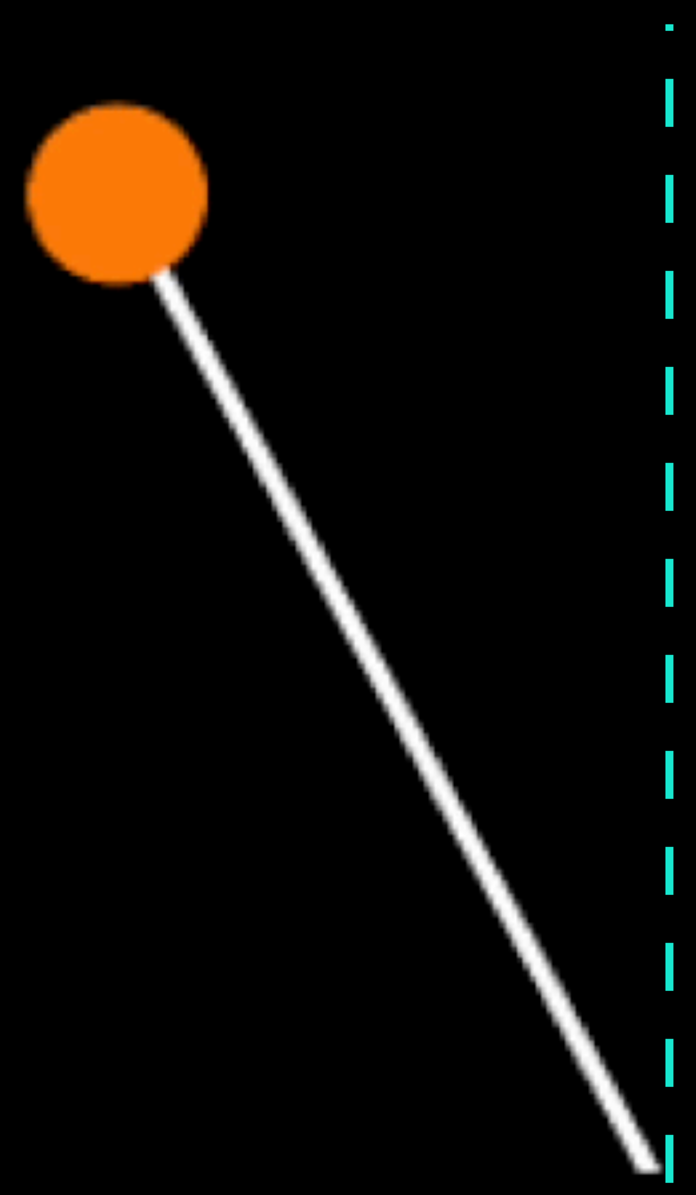
How do we do this for robots with continuous state-actions?


BostonDynamics

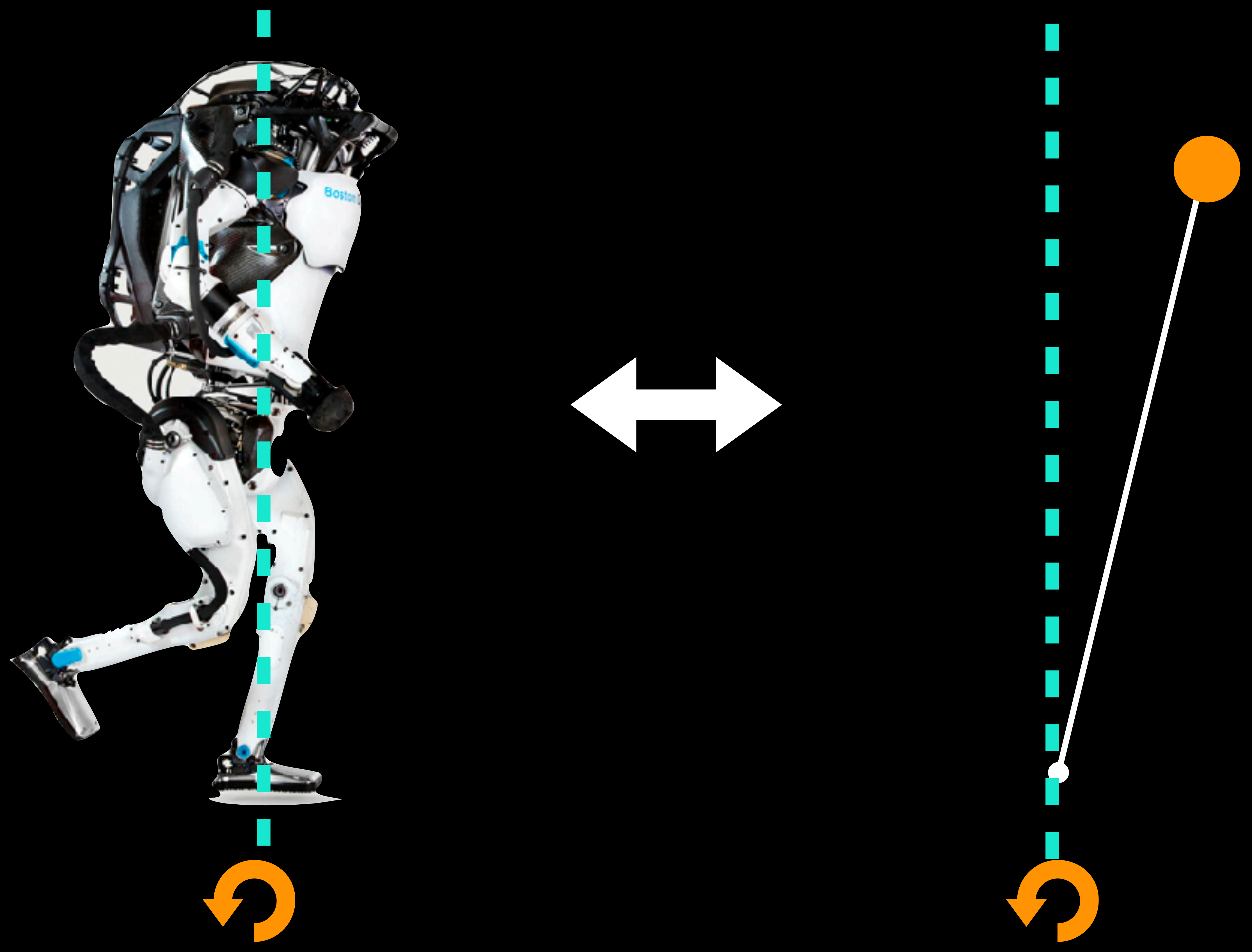
Brainstorm

How do we model the Atlas backflip as a Markov Decision Problem <S, A, C, T>?

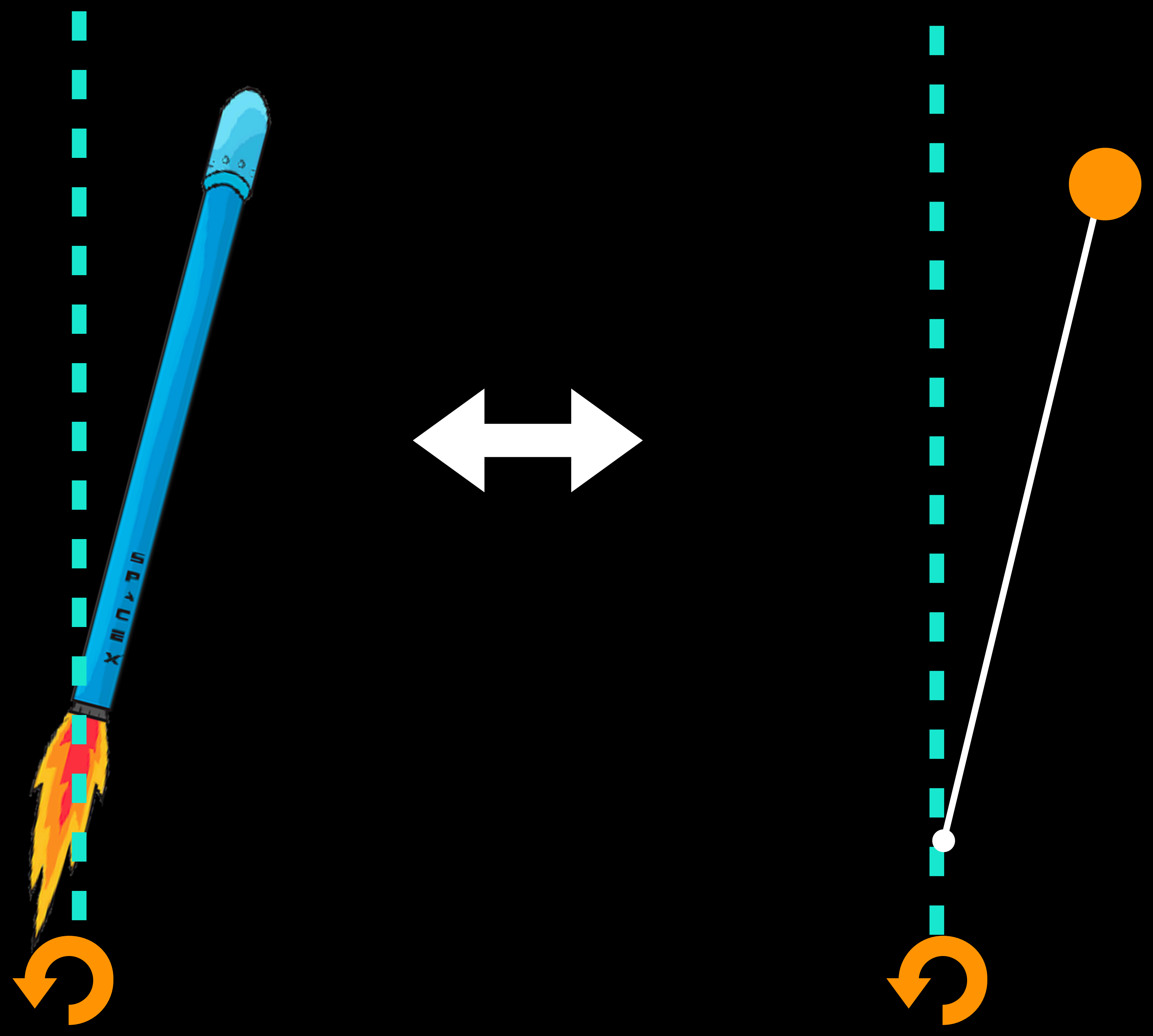
The Inverted Pendulum: A fundamental dynamics model



Humanoid balancing



Rocket landing



Recall: How do we solve a MDP?

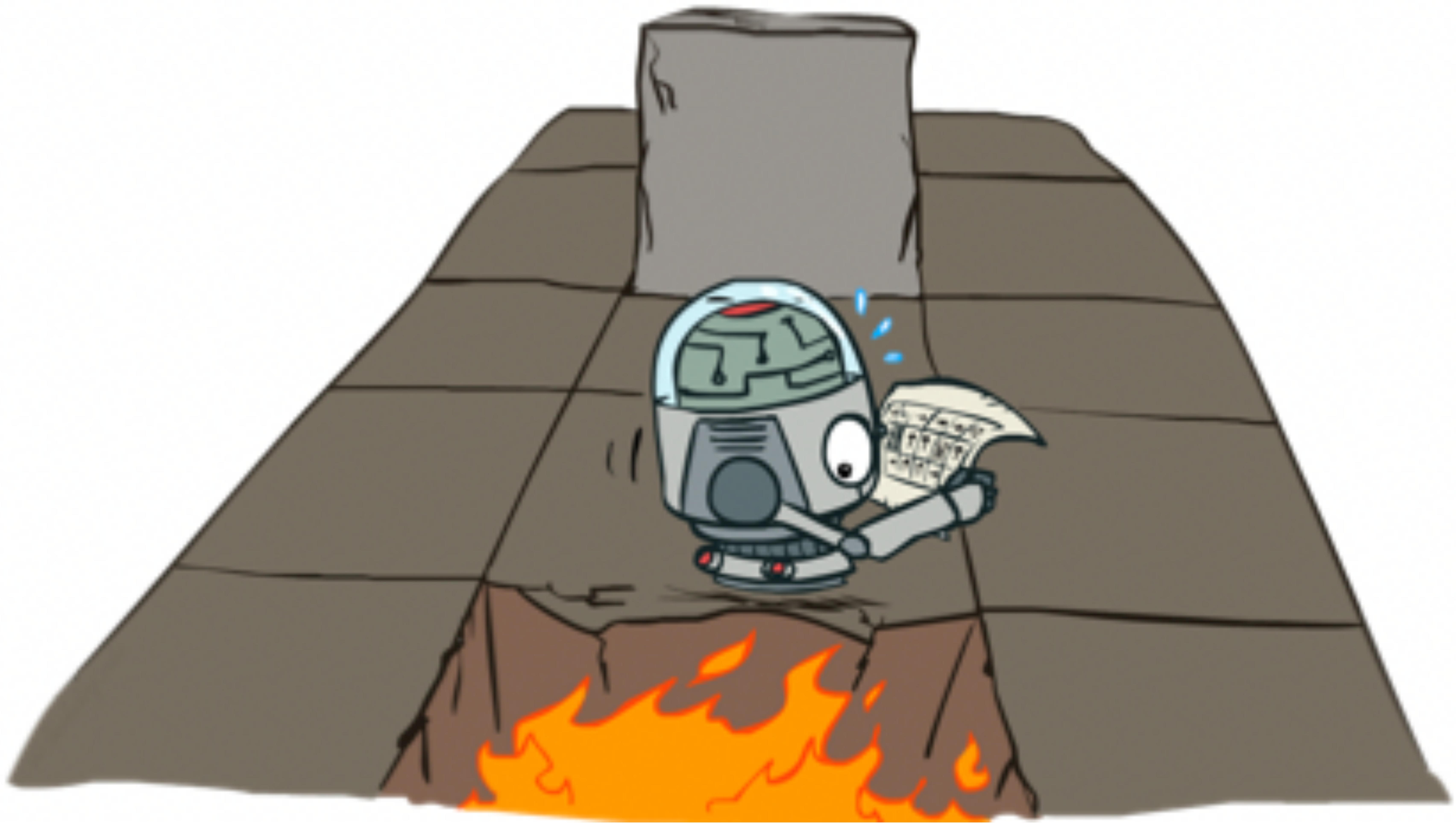


Image courtesy Dan Klein

Value Iteration

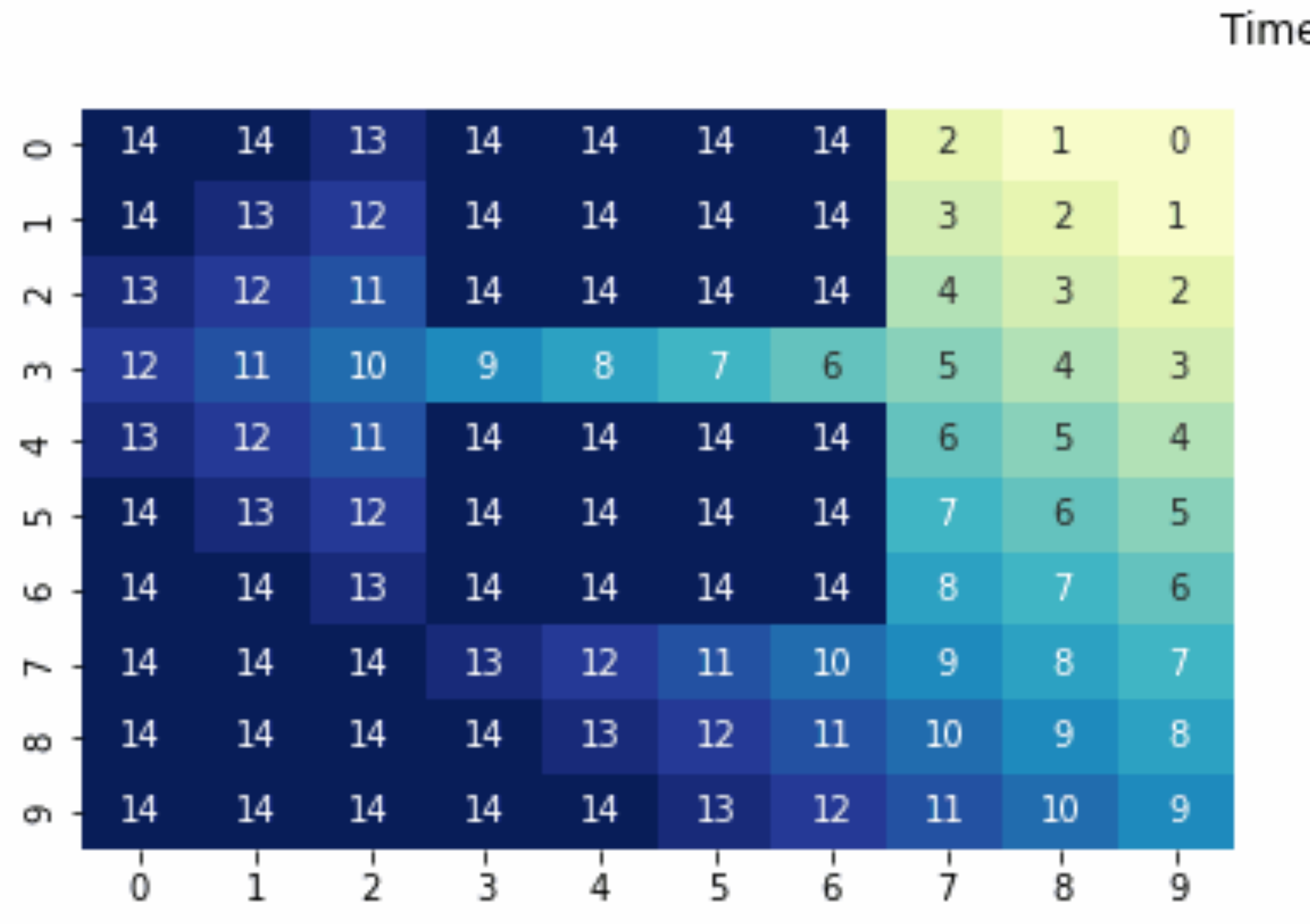
Initialize value function at last time-step

$$V^*(s, T-1) = \min_a c(s, a)$$

for $t = T-2, \ldots, 0$

Compute value function at time-step t

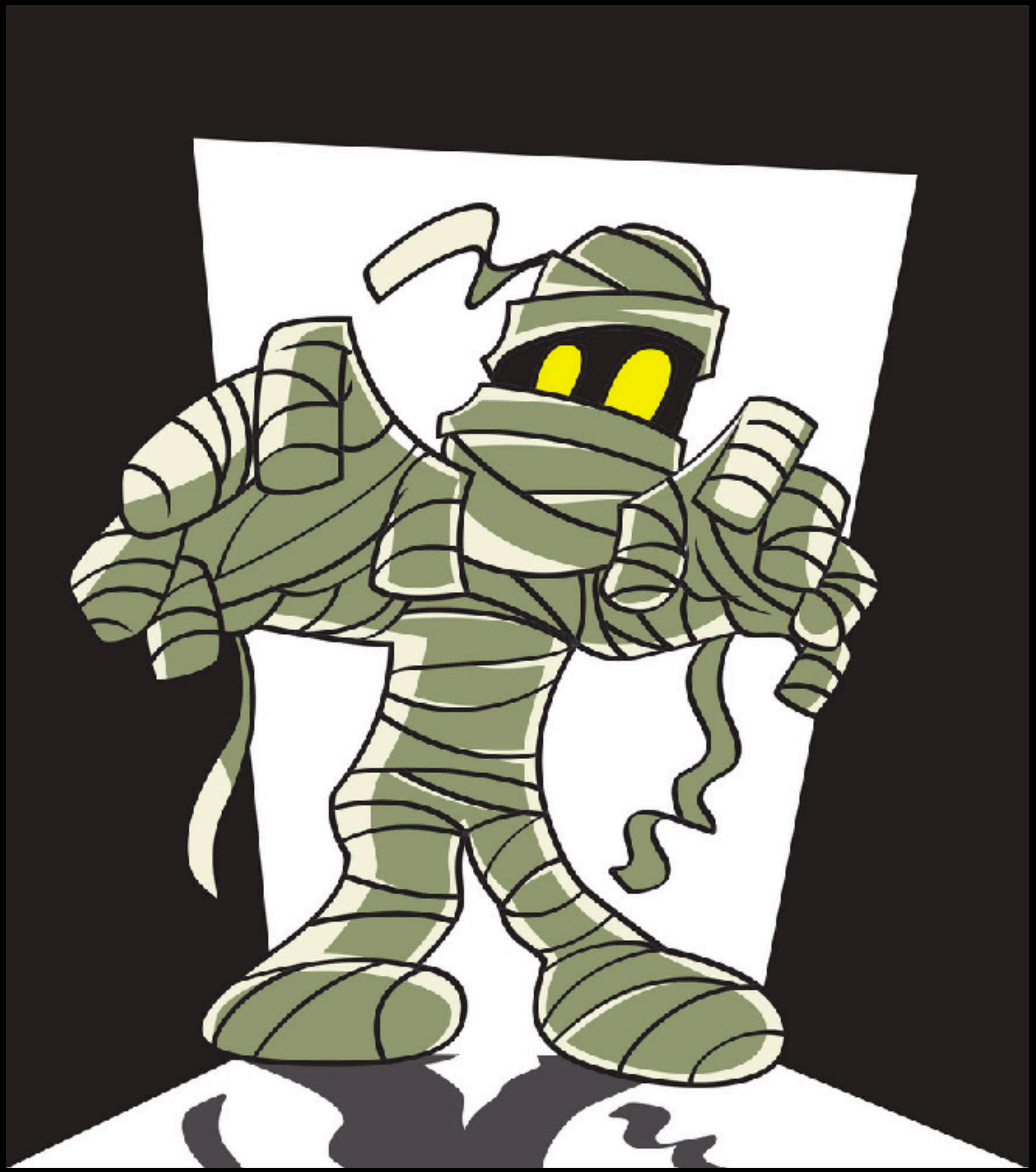
$$V^*(s, t) = \min_a \left[c(s, a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s, a) V^*(s', t+1) \right]$$

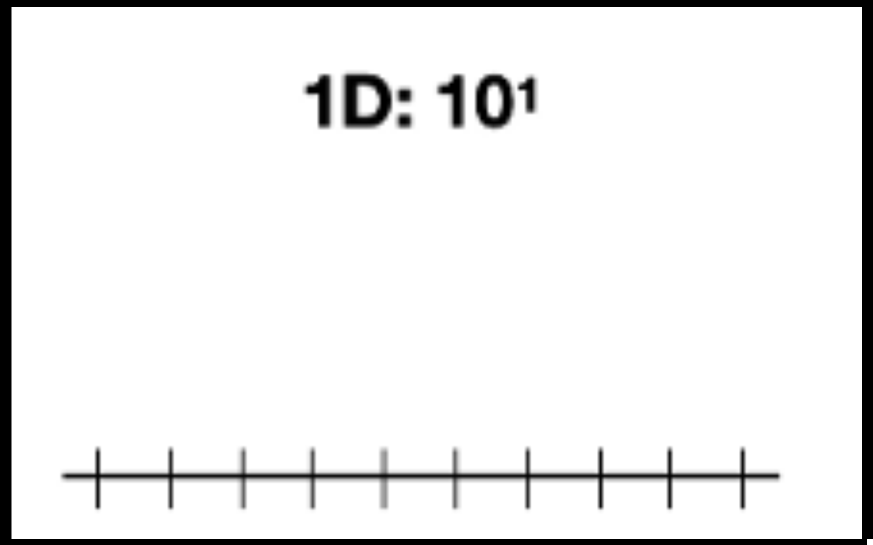


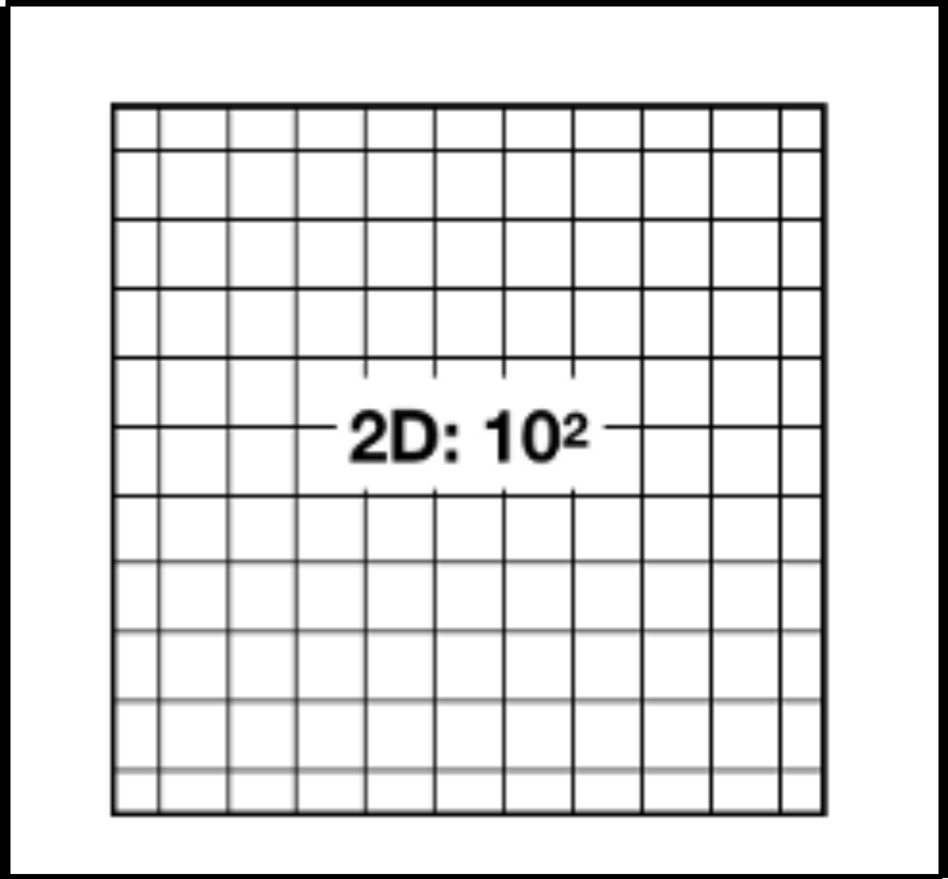
Can we apply value iteration to solve this MDP?

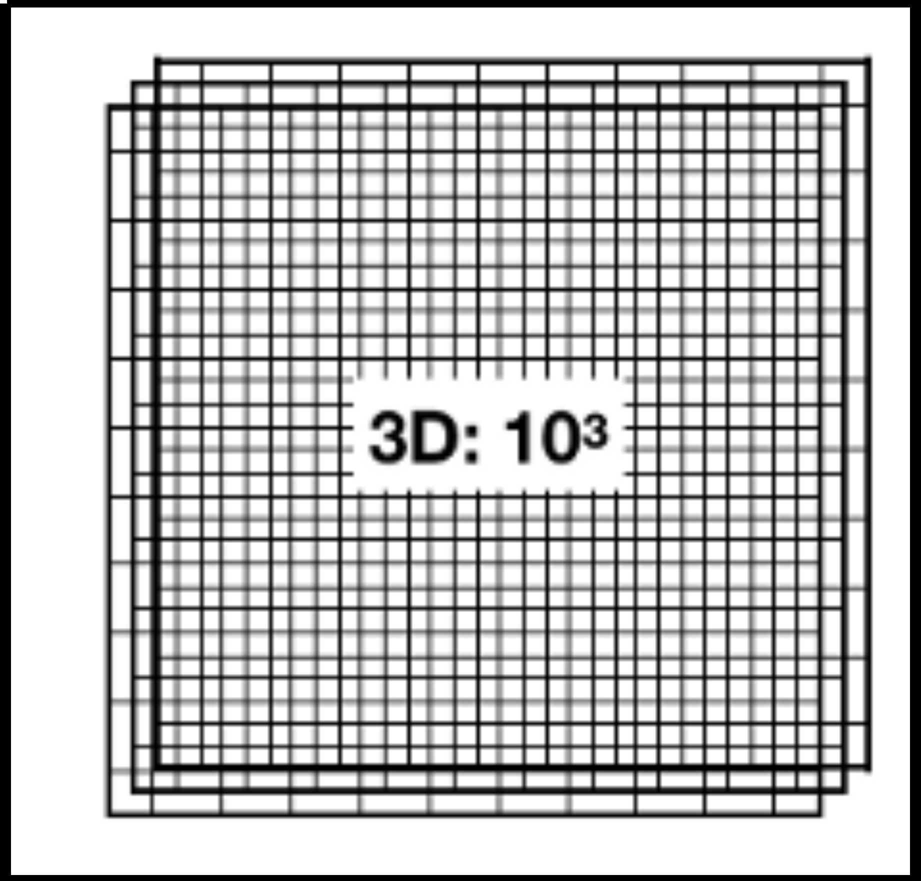
$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right]$$

THE CURSE OF DIMENSIONALITY









Curse of Dimensionality

We cannot discretize continuous states and actions,
because the number of states/action grows exponentially
with dimension

We need some approximation or assumptions!

Can we analytically *represent* and *update* $V^*(s,t)$?

$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s',t+1) \right]$$

What class of functions can we use for $\mathcal{T}(s' \mid s,a)$ and $V^*(s',t+1)$?

Can we analytically *represent* and *update* $V^*(s,t)$?

Yes*

$$V^*(s,t) = \min_a \left[c(s,a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s,a) V^*(s', t+1) \right]$$

(Quadratic) (Quadratic) (Linear) (Quadratic)

Linear Quadratic Regulator (LQR)

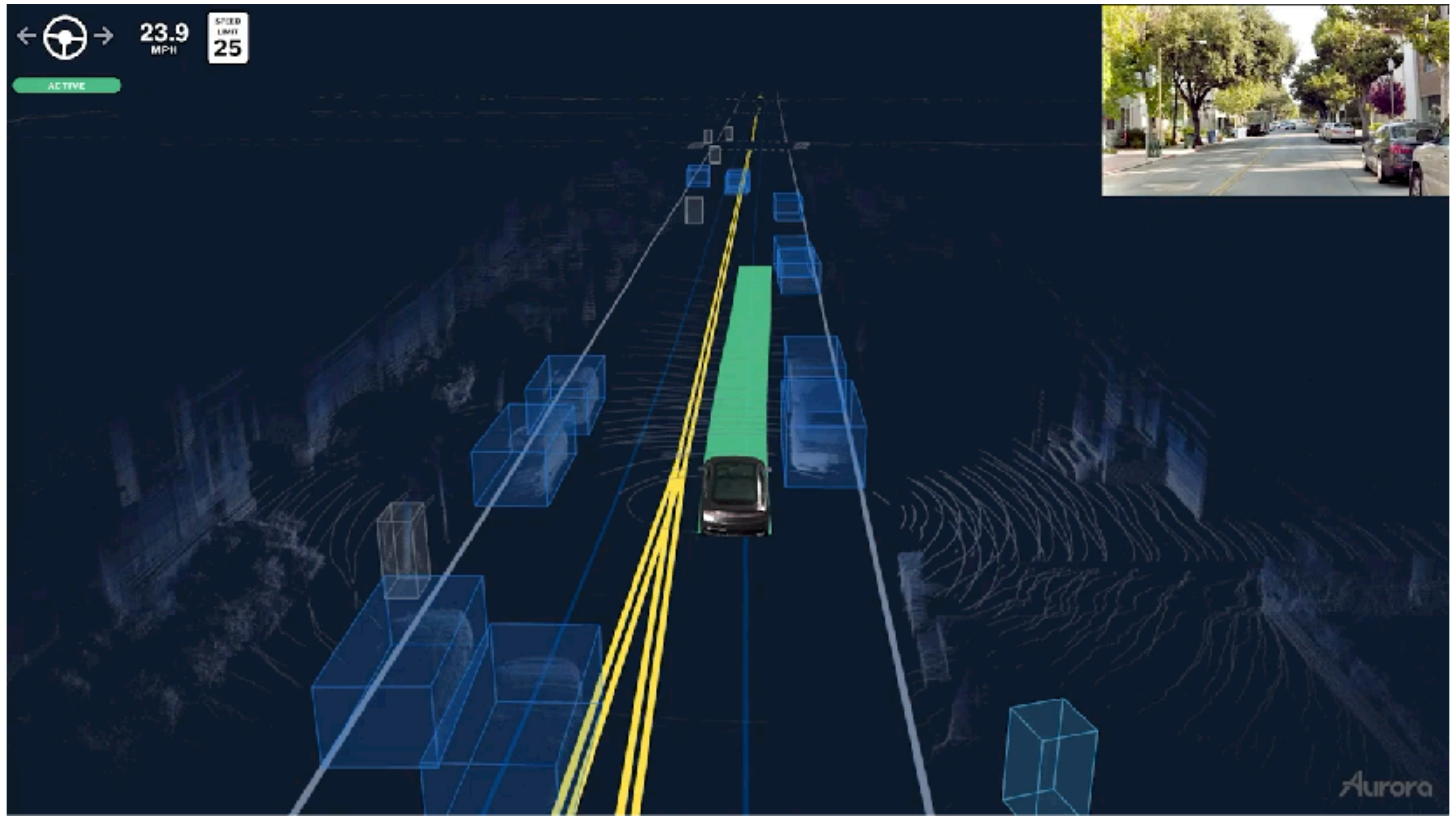
LQR is widely used in real world robotics

But the real world is not linear and quadratic, right?

No, but we can *linearize* dynamics and *quadricize* the costs about some reference

LQR can then be used as a very fast subroutine to compute optimal policy

LQR is widely used in real world robotics



Check out notebook

cs4756_robot_learning / notebooks / inverted_pendulum_lqr.ipynb

jren44 Initial commit

a6c9feb · on Jan 18 History

Preview Code Blame

Raw

Illustrated Linear Quadratic Regulator

Companion to courses lectures from CS6756: Learning for Robot Decision Making and Chapter 2 of Modern Adaptive Control and Reinforcement Learning.

In [3]:

```
import numpy as np
import autograd.numpy as np
from autograd import grad, jacobian
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib import rc
from IPython.display import HTML, Image
from matplotlib.patches import Circle
rc('animation', html='jshtml')
```

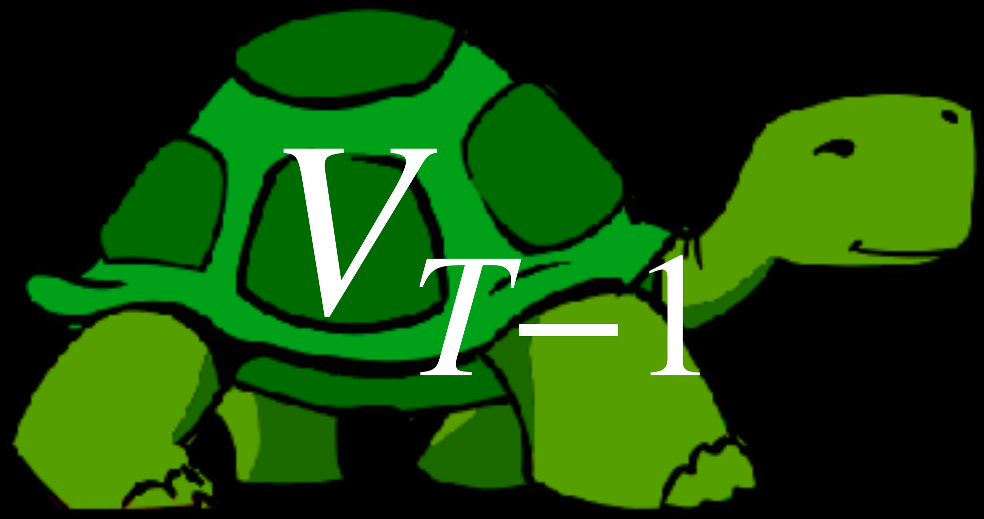
Dynamics of an Inverted Pendulum

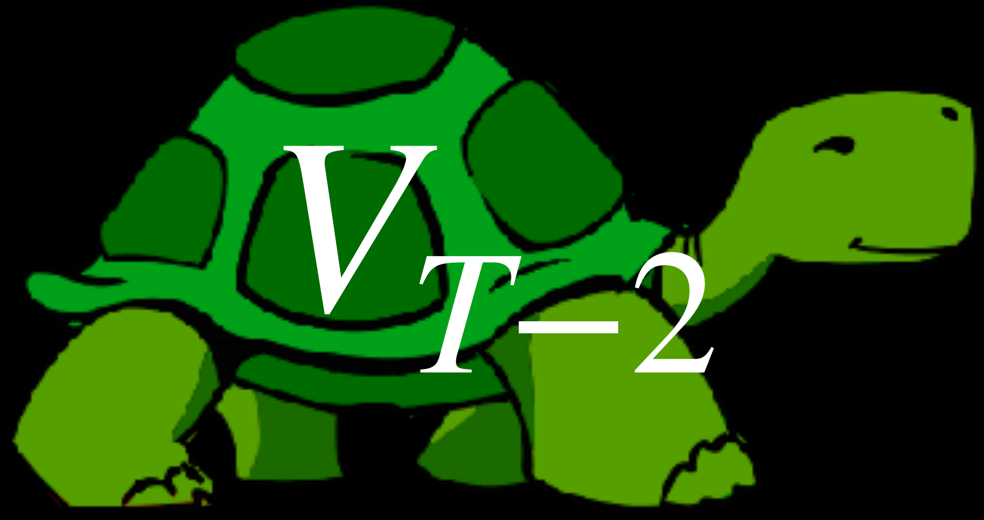
Let’s formalize!

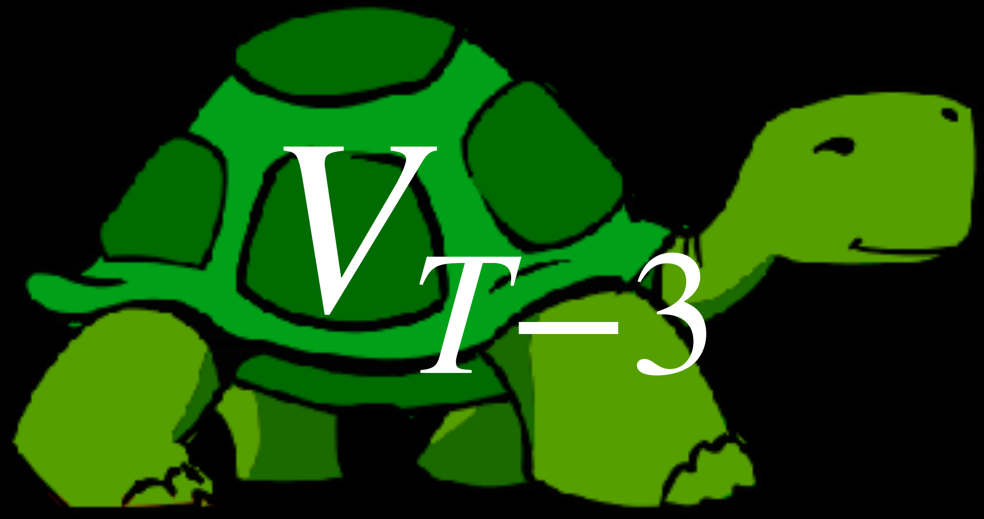
It’s quadratics all the way down!

$$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$$

$$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$$







The LQR Algorithm

Initialize $V_T = Q$

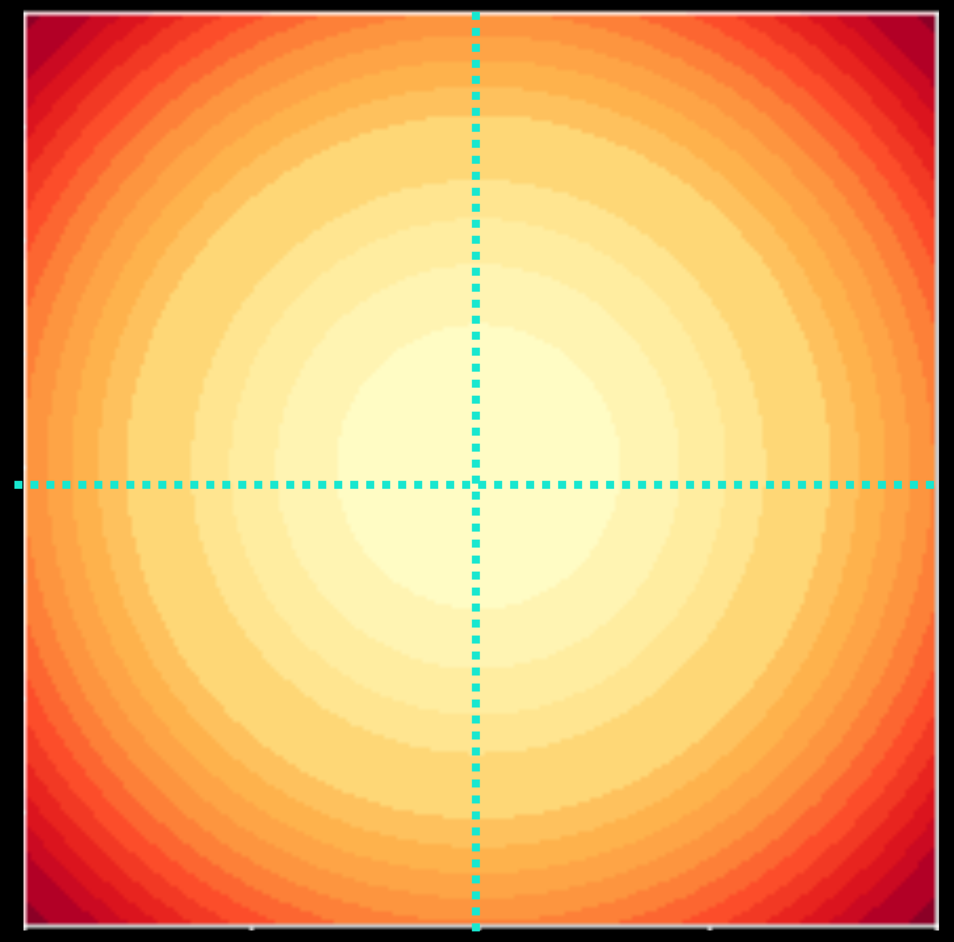
For t = T-1, ..., 1

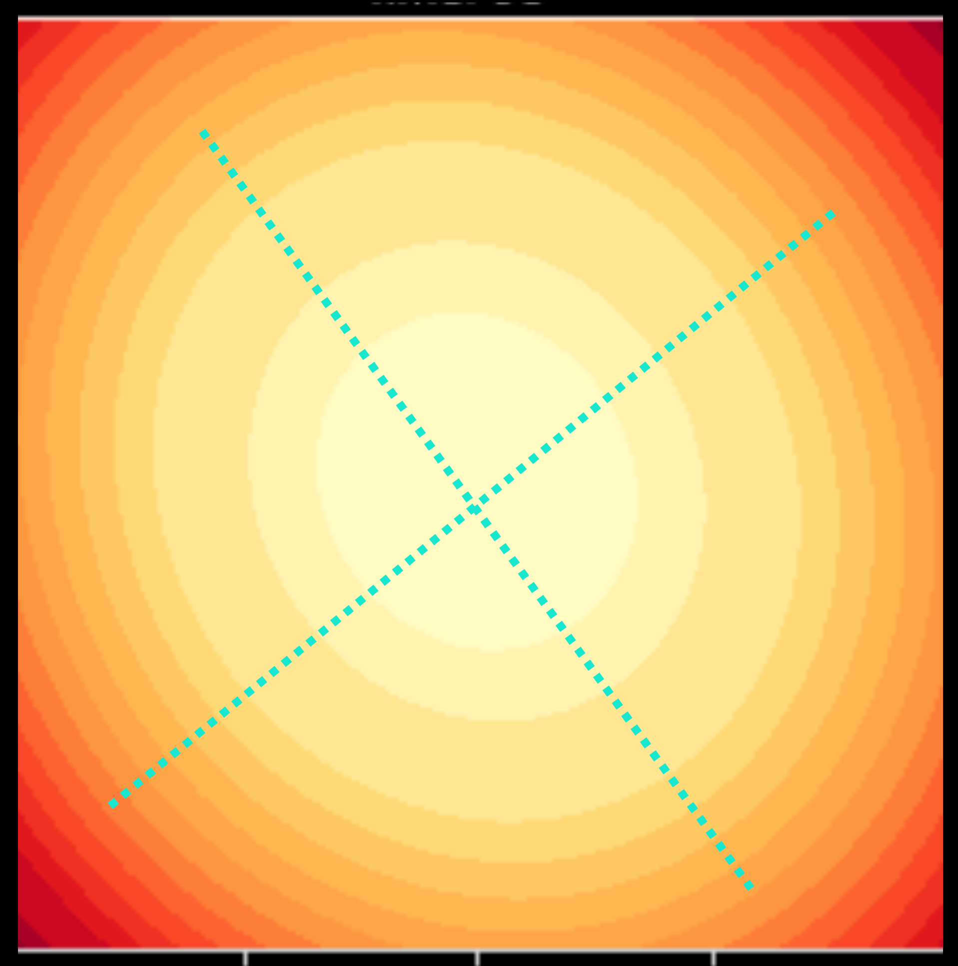
Compute gain matrix

$$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$$

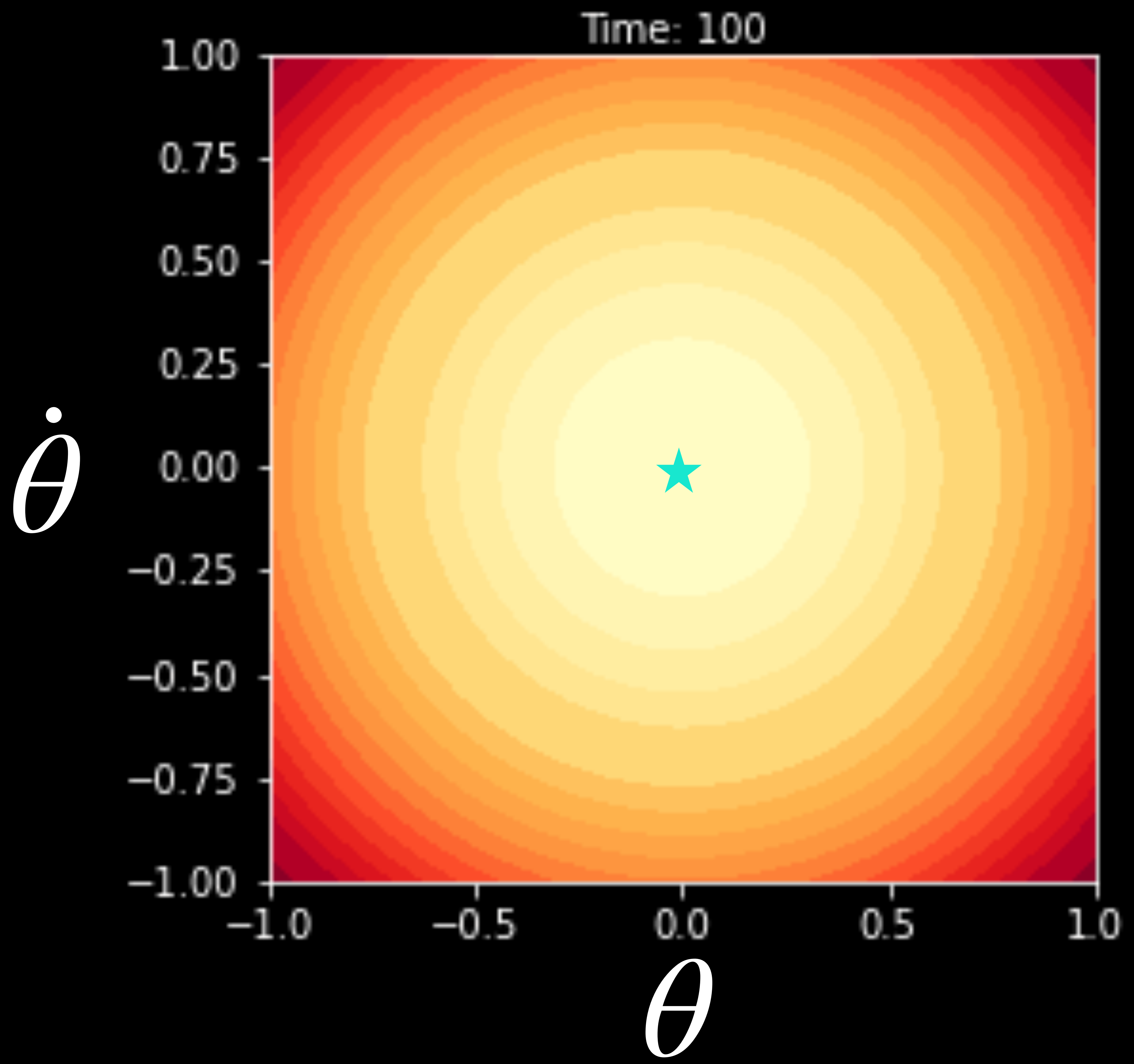
Update value

$$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$$

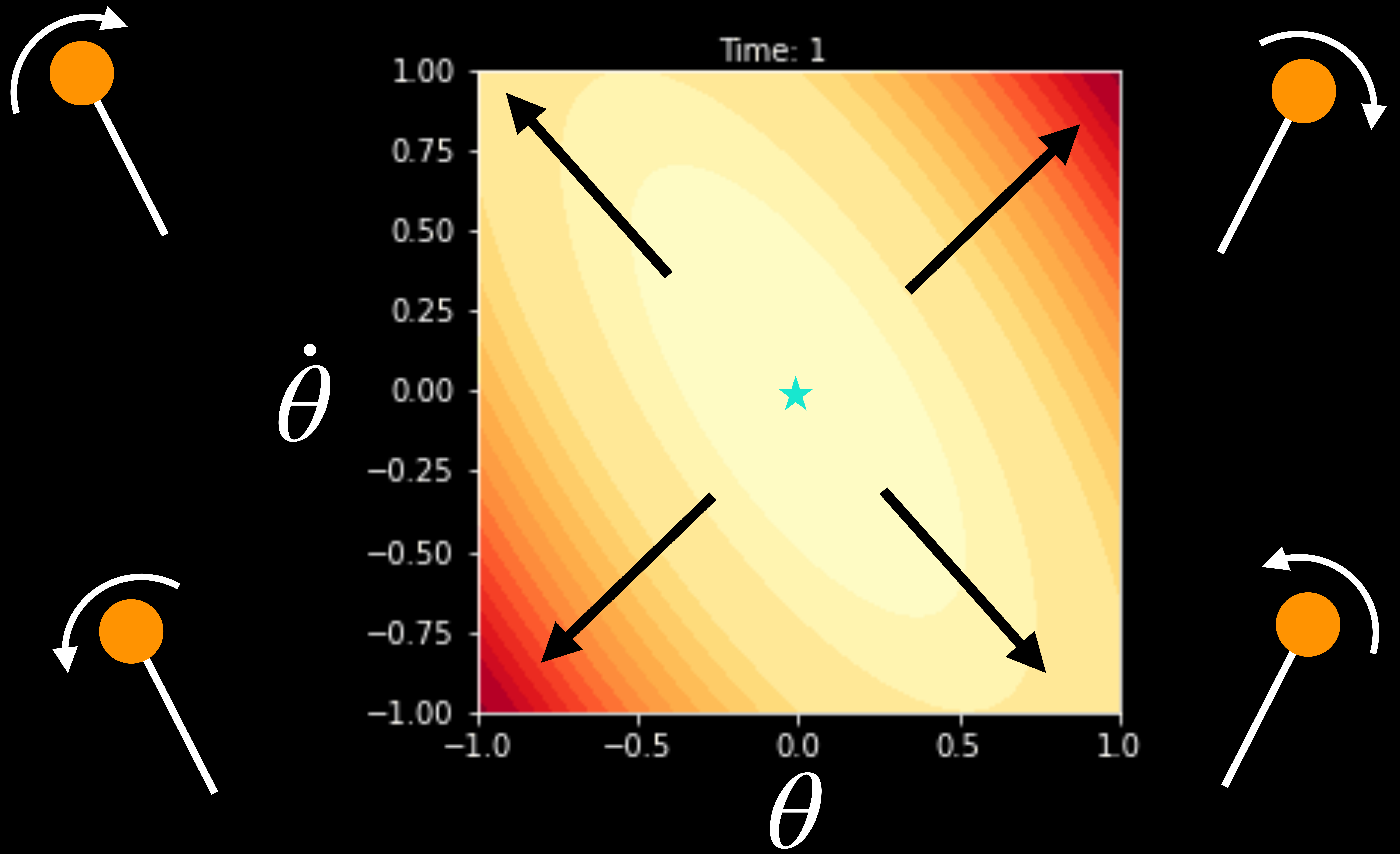




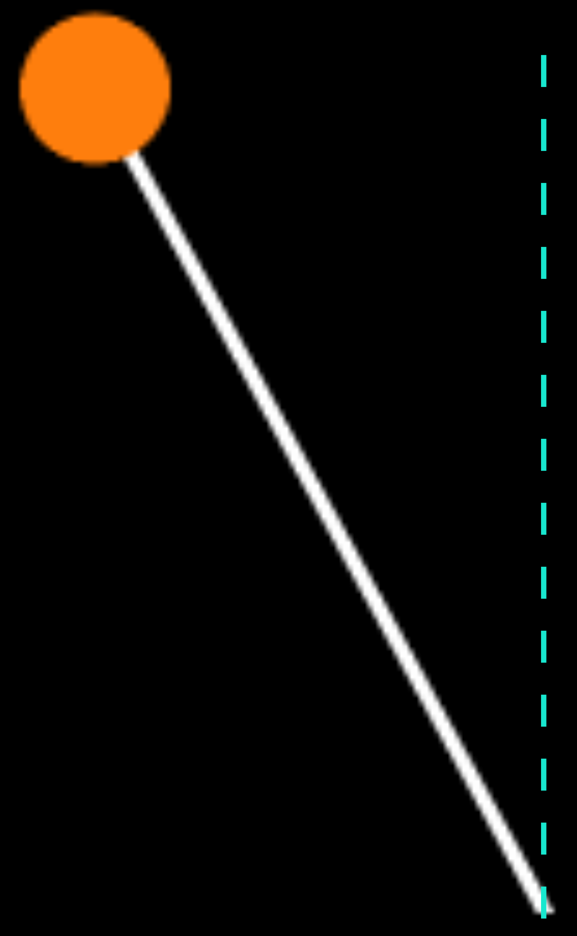
Value Iteration for Inverted Pendulum

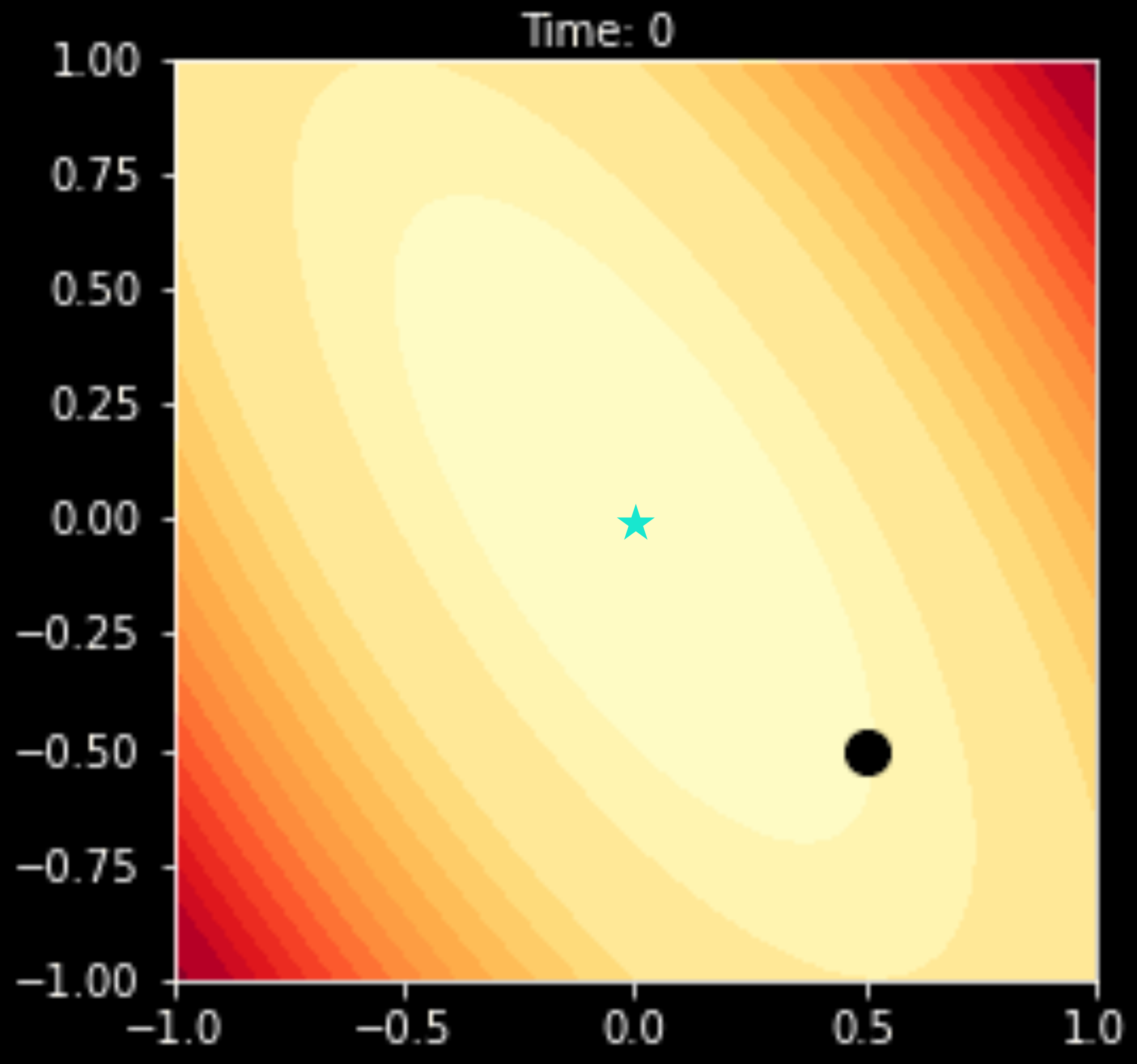


Value Iteration for Inverted Pendulum

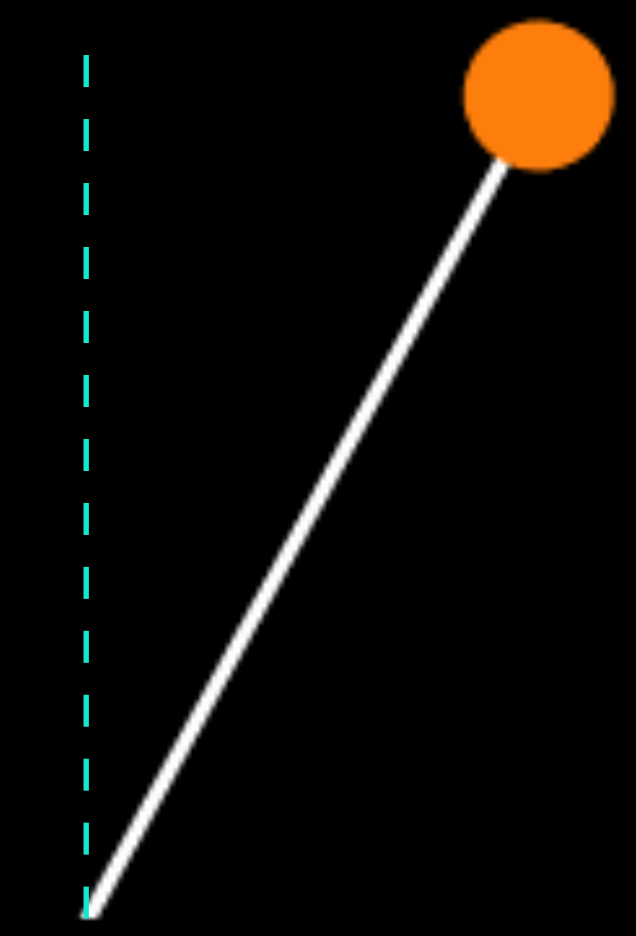


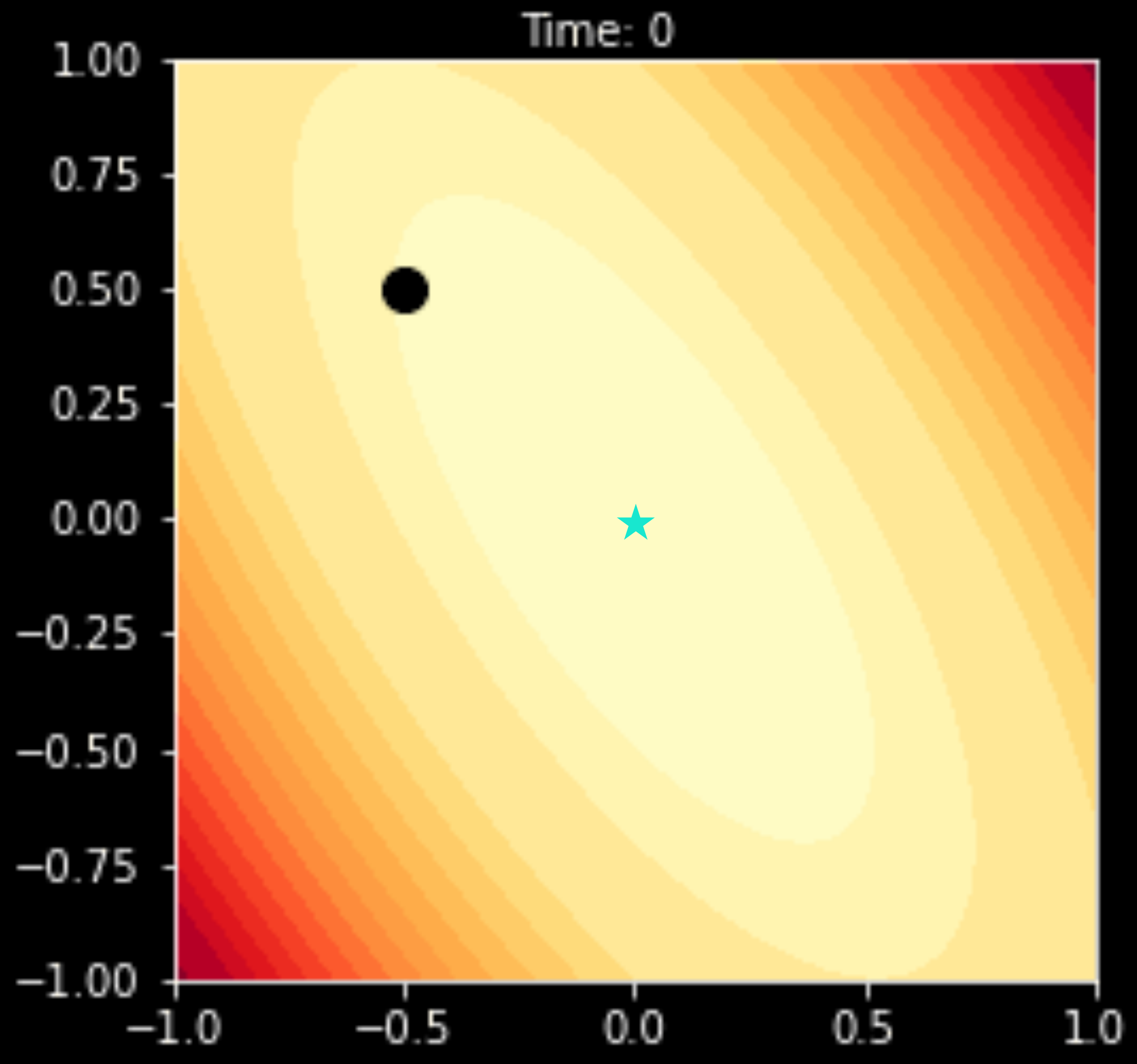
An Easy Starting Point



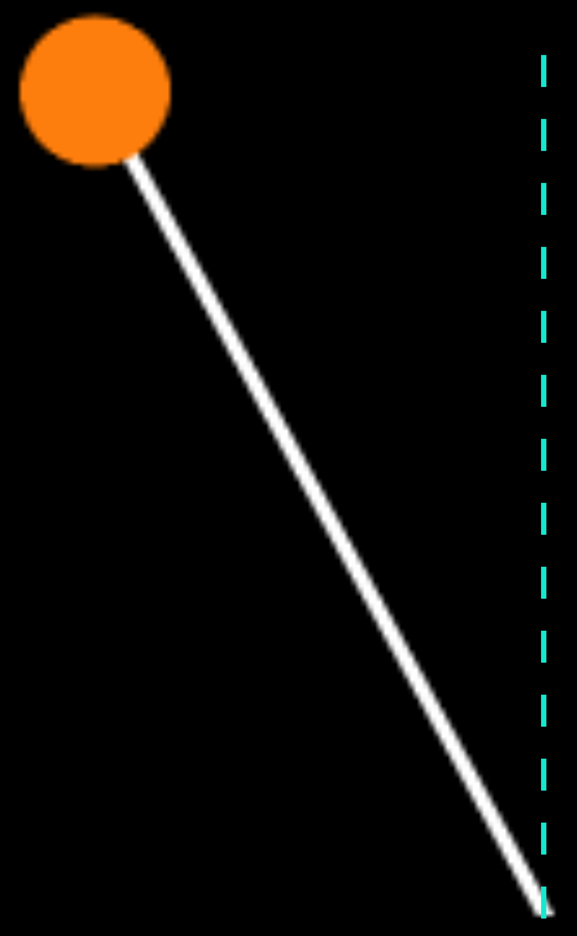


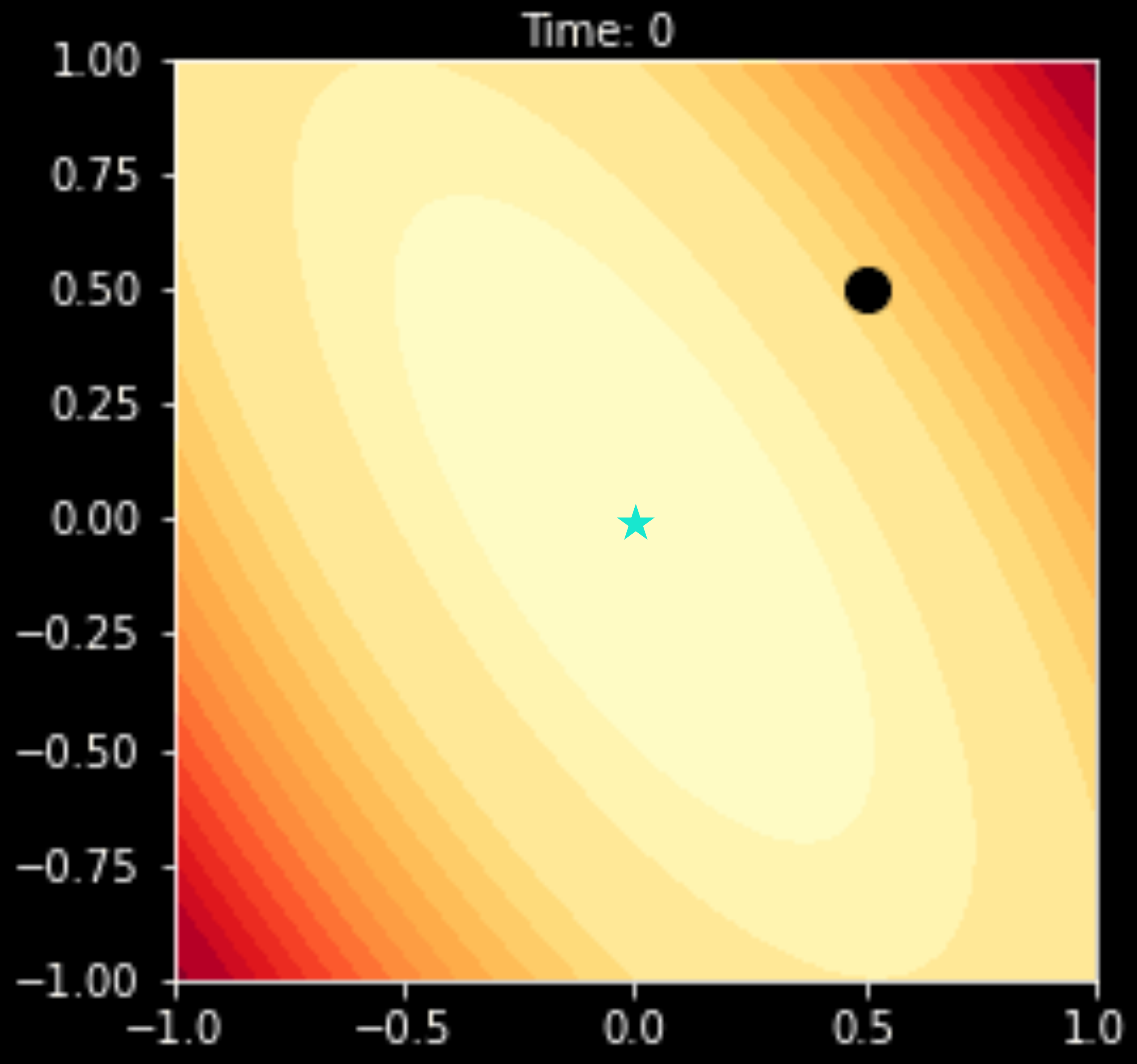
Another Easy Starting Point



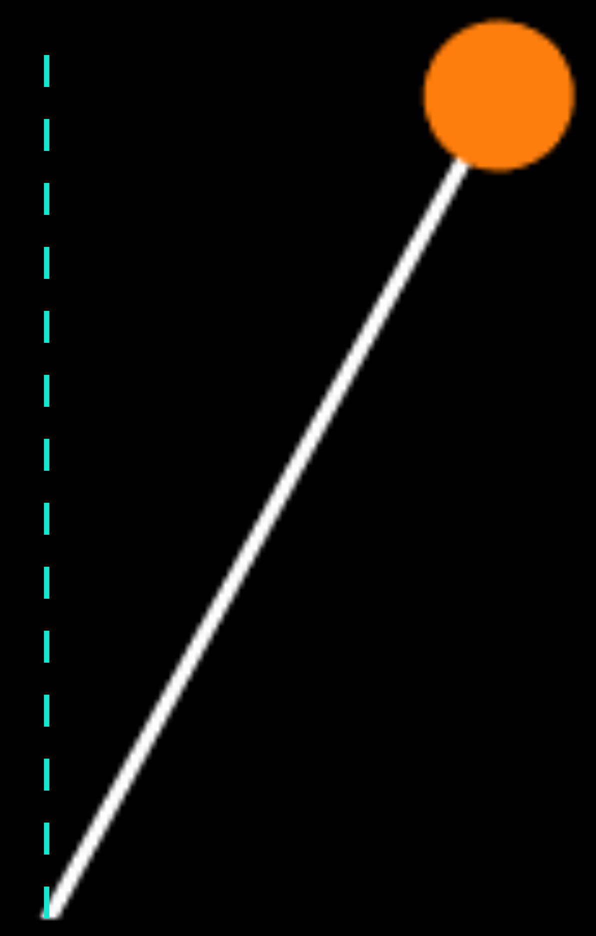


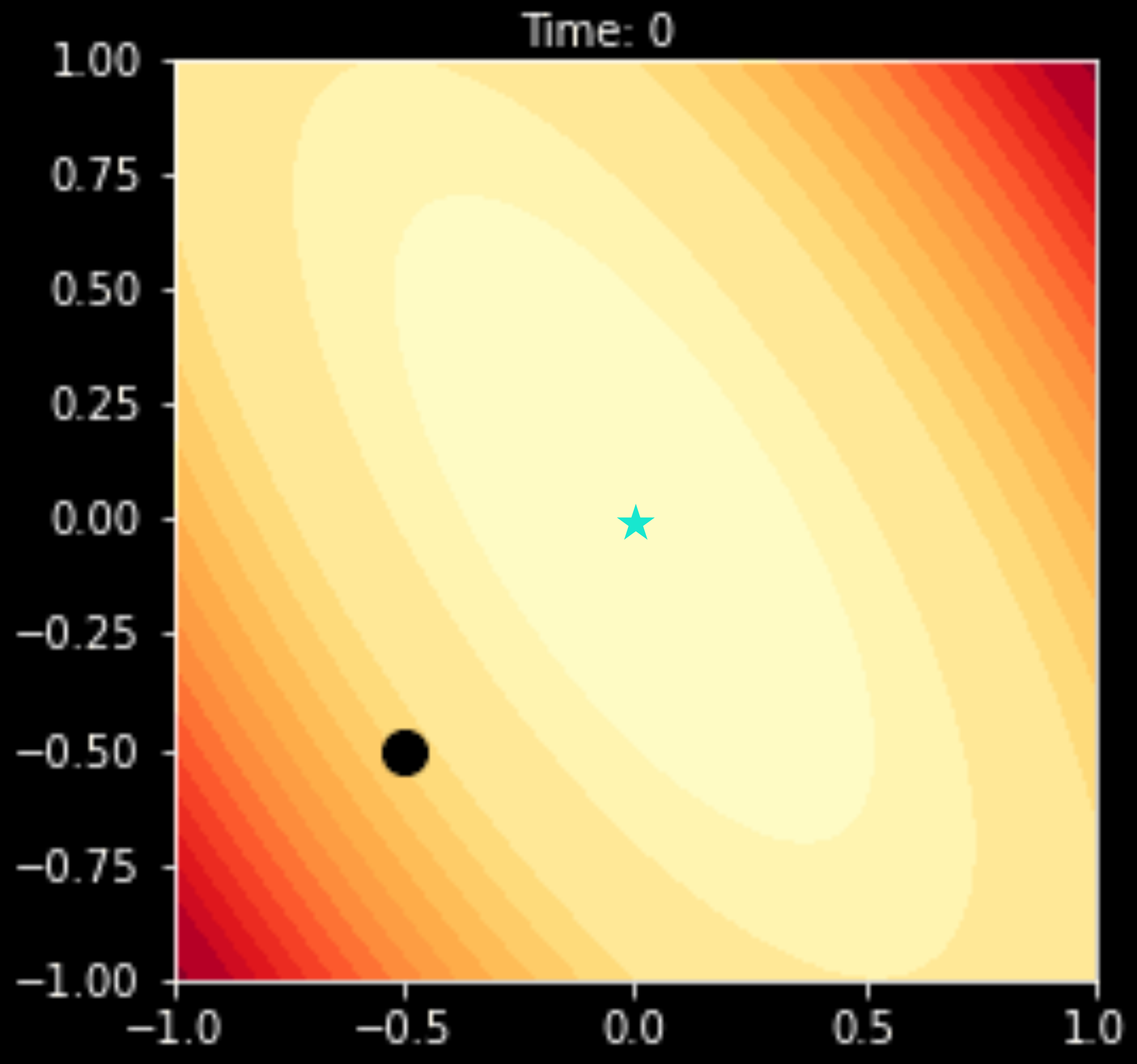
A Hard Starting Point





Another Hard Starting Point





LQR Converges

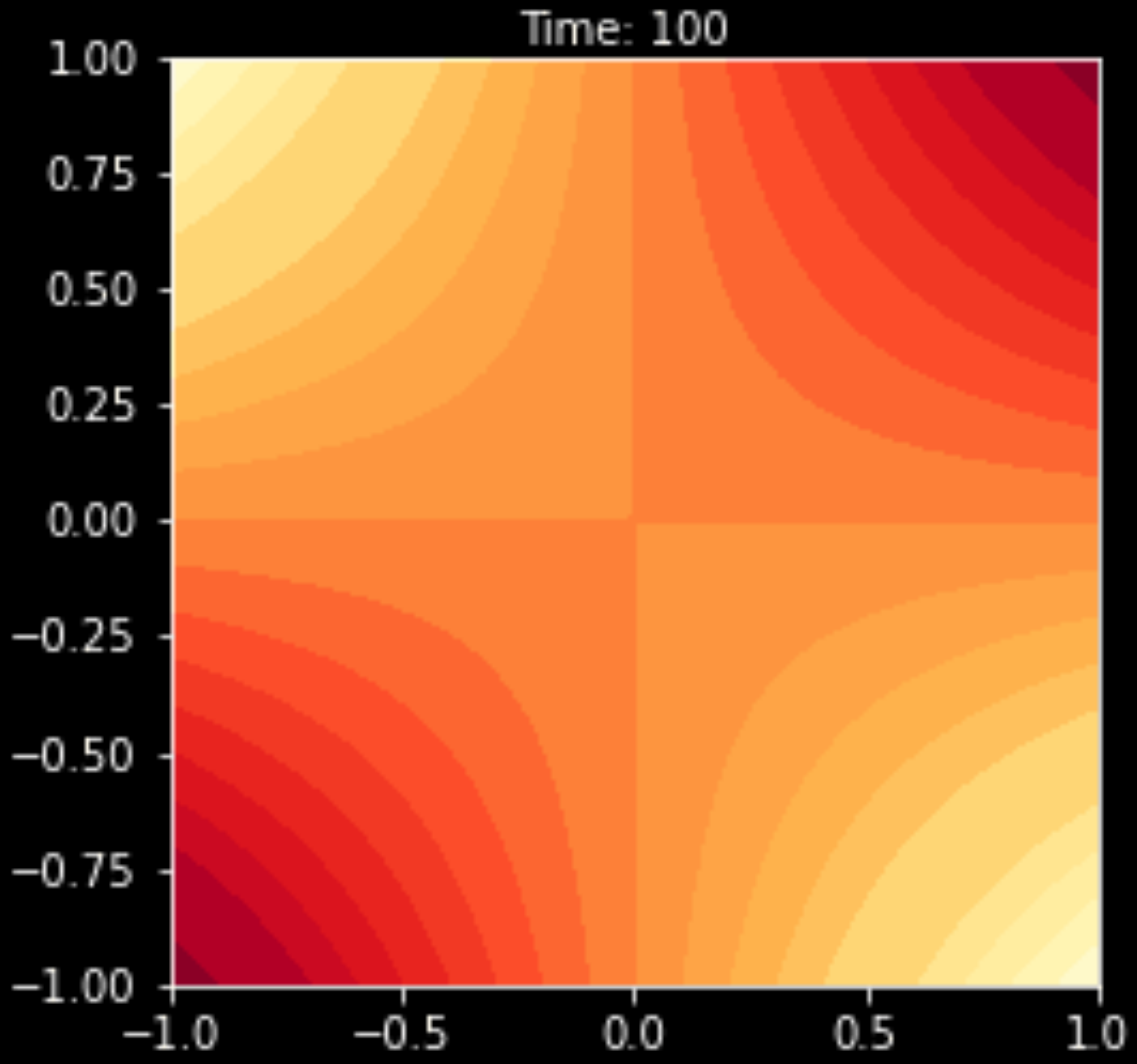
Q is positive semi-definite

$$x^T Q x \geq 0$$

R is positive definite

$$u^T R u > 0$$

What if Q is not PSD?



$$x^T Q x \not\geq 0$$

$$Q = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

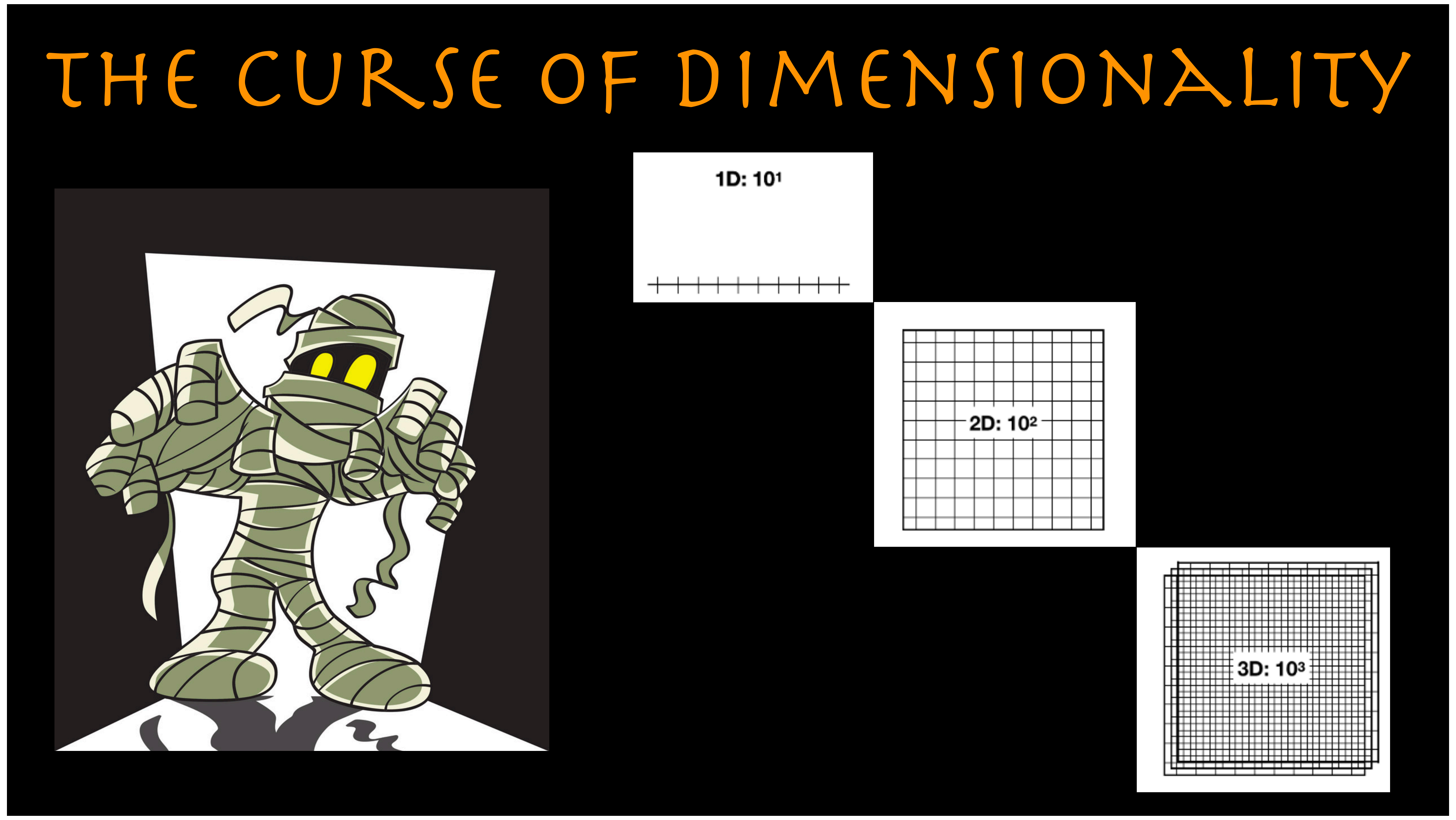
What if R is not positive definite?

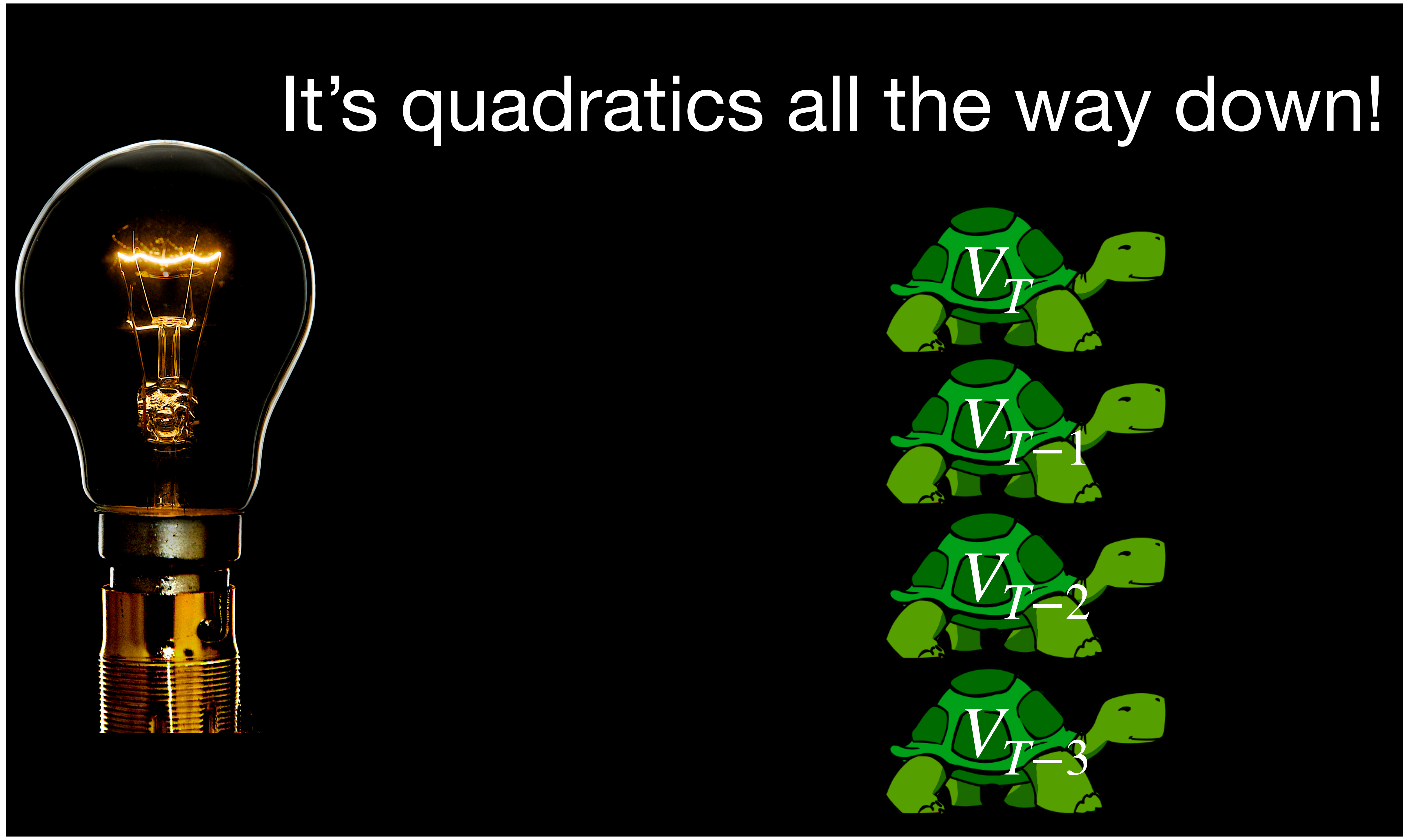
$$u^T R u \not> 0 \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

Hint: Gain matrix update?

$$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$$

tl;dr

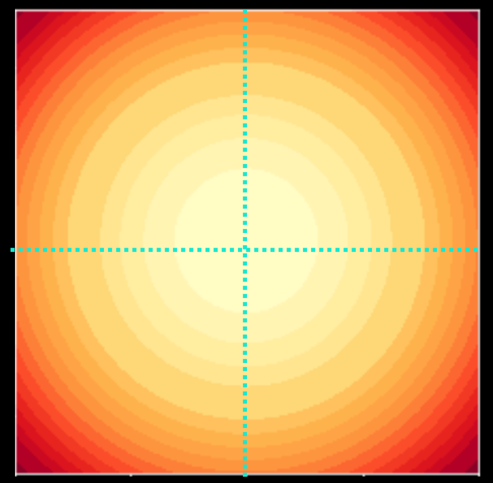




The LQR Algorithm

Initialize $V_T = Q$

For t = T … 1



Compute gain matrix

$K_t = (R + B^T V_{t+1} B)^{-1} B^T V_{t+1} A$

Update value

$V_t = Q + K_t^T R K_t + (A + BK_t)^T V_{t+1} (A + BK_t)$