Robot Autonomy

Lecture 3:
Dynamic Programming and LQR

Oliver Kroemer

Example Franka Code

```
franka::RobotState robot_state = cartesian_pose_handle_->getRobotState();
std::array<double, 7> coriolis = model_handle_->getCoriolis();
std::array<double, 7> gravity = model_handle_->getGravity();
```

Get robot state information

```
double alpha = 0.99;
for (size_t i = 0; i < 7; i++) {dq_filtered_[i] = (1 - alpha) * dq_filtered_[i] + alpha * robot_state.dq[i];}
```

Filter velocity signals

```
std::array<double, 7> tau_d_calculated;

for (size_t i = 0; i < 7; ++i) {
  tau_d_calculated[i] = coriolis_factor_ * coriolis[i] +
  k_gains_[i] * (robot_state.q_d[i] - robot_state.q[i]) +
  d_gains_[i] * (robot_state.dq_d[i] - dq_filtered_[i]);
}
// Note: Robot automatically adds gravity compensation
```

Compute Joint Torques

What is still missing?

std::array<double, 49> franka::Model::mass (const franka::RobotState & robot_state) const

Calculates the 7x7 mass matrix.

Unit: $[kg \times m^2]$.

Parameters

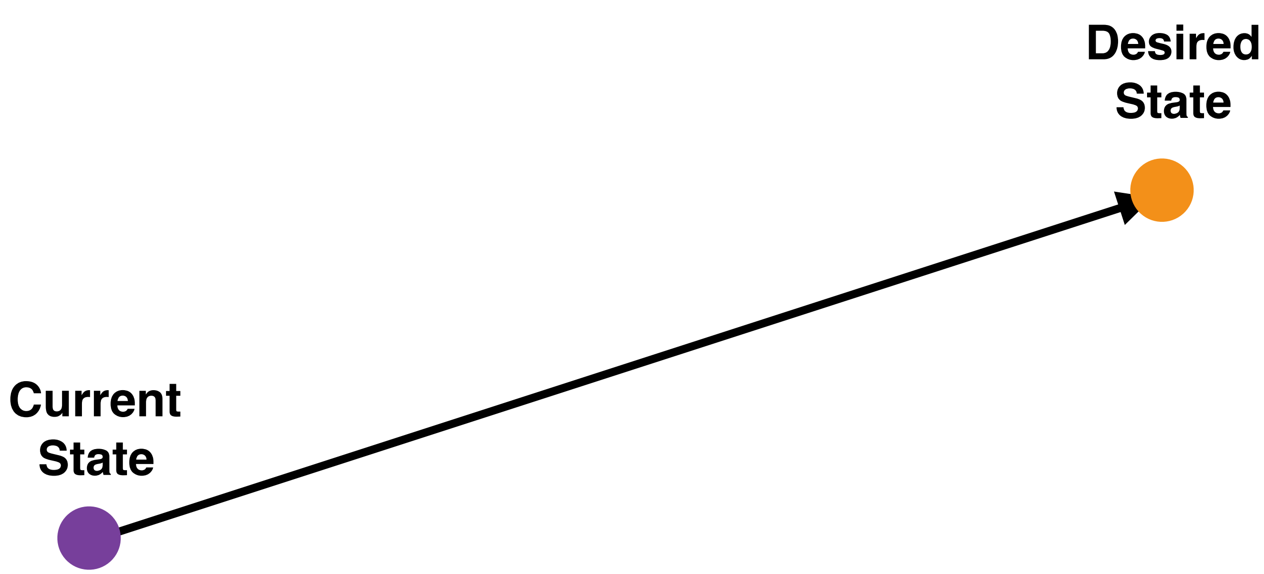
[in] **robot_state** State from which the pose should be calculated.

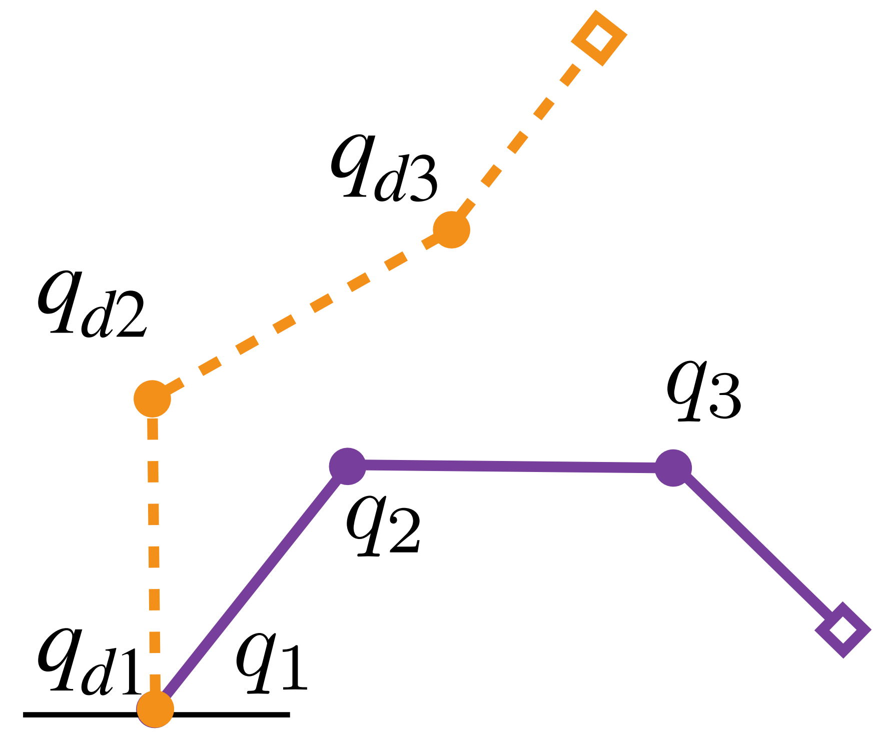
Returns

Vectorized 7x7 mass matrix, column-major.

Motivation

- Want to find a controller that gets us to a desired state





- Different feedback gains result in different performances
  - High/low torques, over/undershoot, etc.
- Rather than define gains, why not define performance?
- Compute the optimal gains for this reward function!

Problem Statement

- State of the robot given by $x_t \in X$ and action by $u_t \in U$
- Given:
  - Transition function $$x_{t+1} = f(x_t, u_t)$$
  - Reward function $$r(x_t, u_t)$$
- Goal: compute a policy/controller

$$u_t = \pi(x_t)$$

- such that we maximise

Horizon

$$J = \sum_{t=0}^{N} r(x_t, u_t)$$

- Good immediate reward may lead to poor future reward!

Dynamic Programming

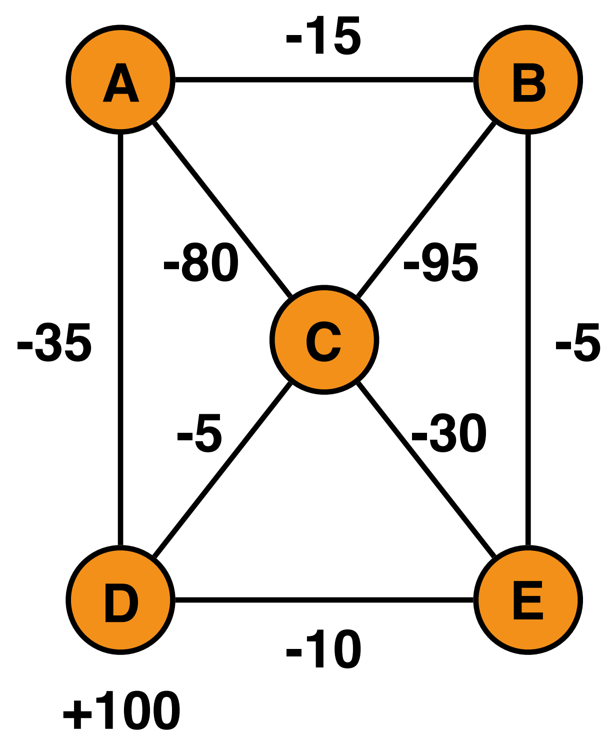
Bellman Optimality Principle

"*An optimal sequence of controls in a multistage optimization problem has the property that whatever the initial stage, state and controls are, the remaining controls must constitute an optimal sequence of decisions for the remaining problem with stage and state resulting from previous controls considered as initial conditions.*"

Richard Bellman, Dynamic Programming, 1957

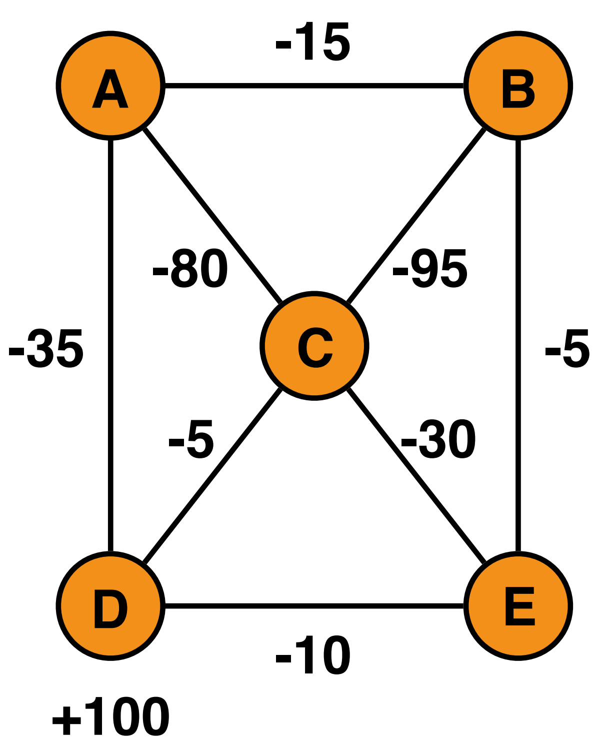
Dynamic Programming Example

- Consider the following discrete state system:



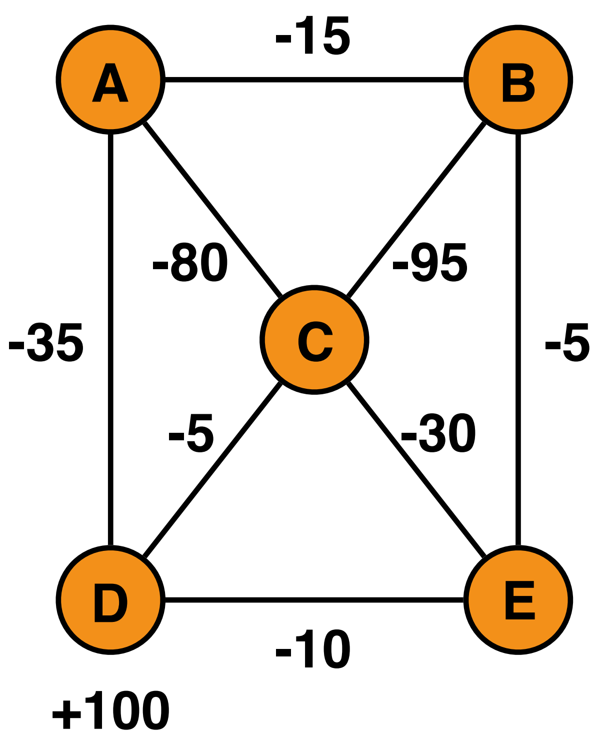
- Transitions: stay in current state or traverse a line
- Rewards: current node plus traversed edge
- Compute a policy: select action for every possible state

Dynamic Programming Example



N-3		N-2		N-1		N		x
								A
								B
								C
								D
								E

Dynamic Programming Example

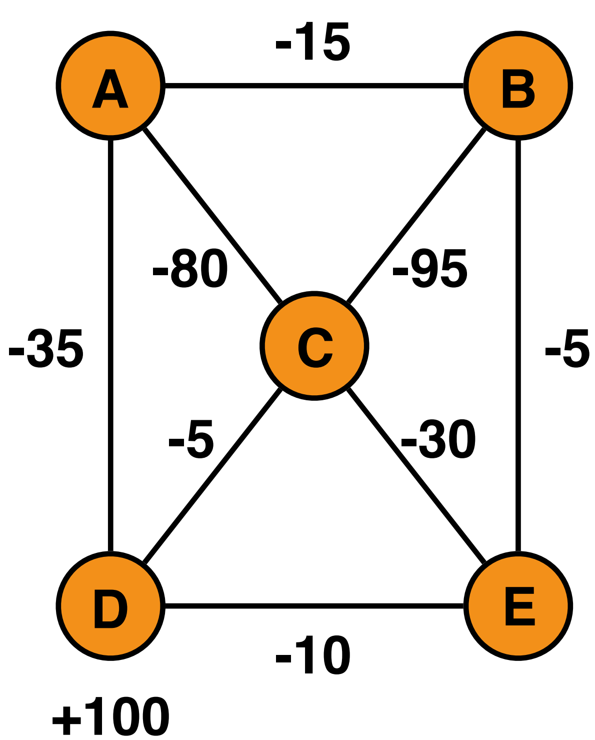


Optimal Action (Policy)

Total Reward (Value)

N-3		N-2		N-1		N		x
								A
								B
								C
								D
								E
$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

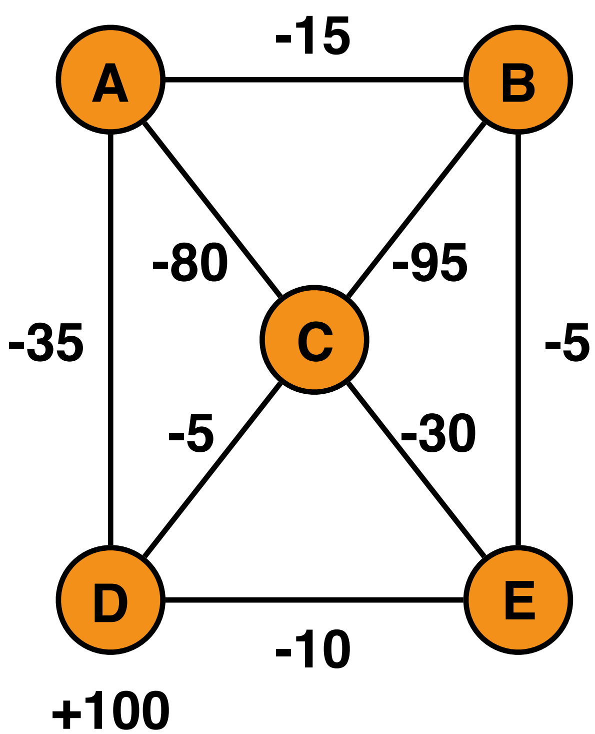
Dynamic Programming Example



N-3		N-2		N-1		N		
								A
								B
								C
								D
								E

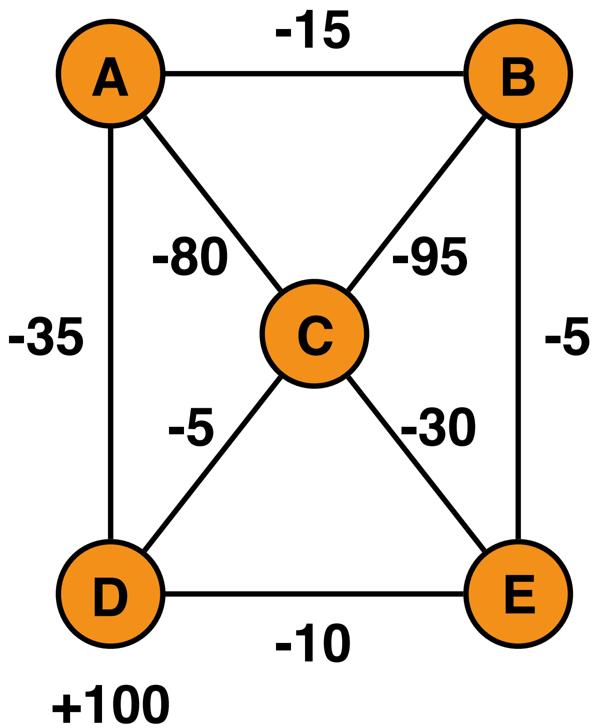
$\pi(x)$ $V(x)$

Dynamic Programming Example



N-3		N-2		N-1		N		
						A	0	A
						B	0	B
						C	0	C
						D	100	D
						E	0	E
				$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

Dynamic Programming Example



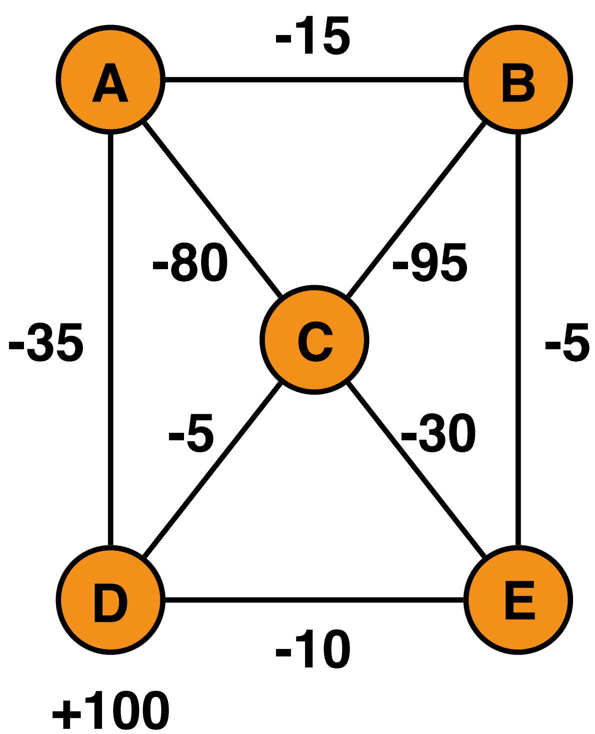
N-3		N-2		N-1		N		
				D	65	A	0	A
				B	0	B	0	B
				D	95	C	0	C
				D	200	D	100	D
				D	90	E	0	E
		$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

-35

-15

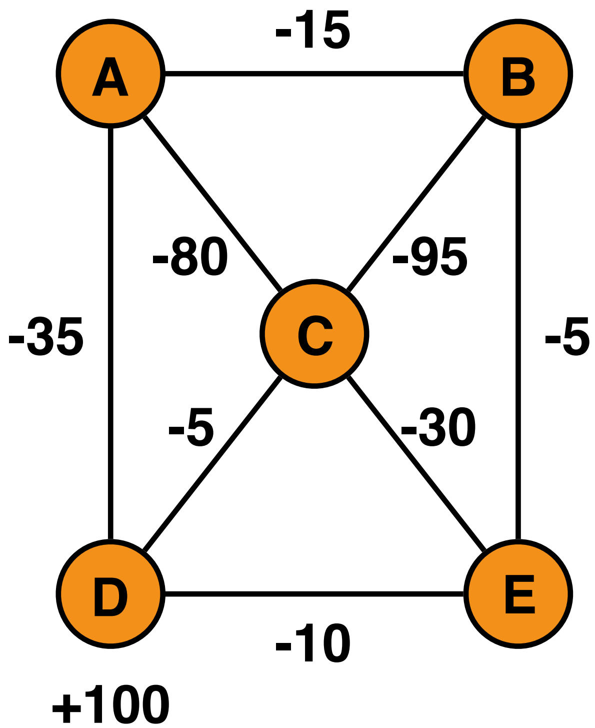
-10

Dynamic Programming Example



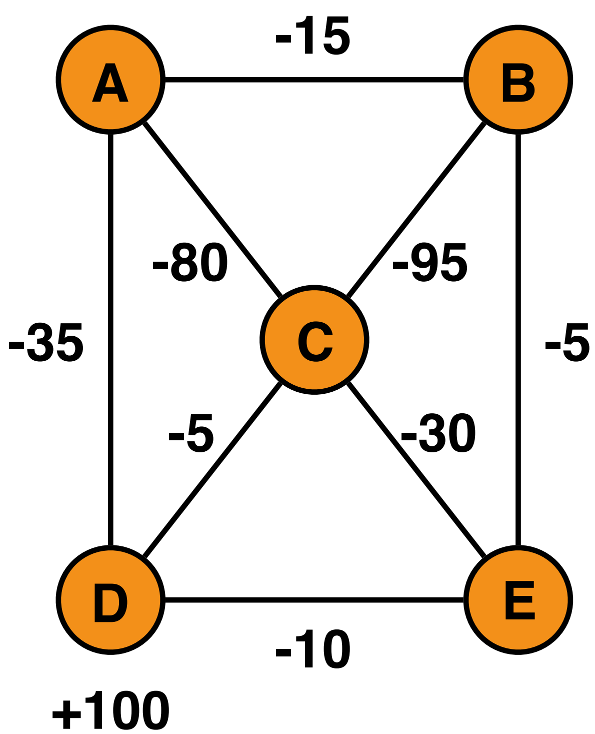
N-3		N-2		N-1		N		
				D	65	A	0	A
		?	-15 0 -95 -5	B	0	B	0	B
				D	95	C	0	C
				D	200	D	100	D
				D	90	E	0	E
		$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

Dynamic Programming Example



N-3		N-2		N-1		N		
				D	65	A	0	A
		E	-5	B	0	B	0	B
				D	95	C	0	C
				D	200	D	100	D
				D	90	E	0	E
		$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

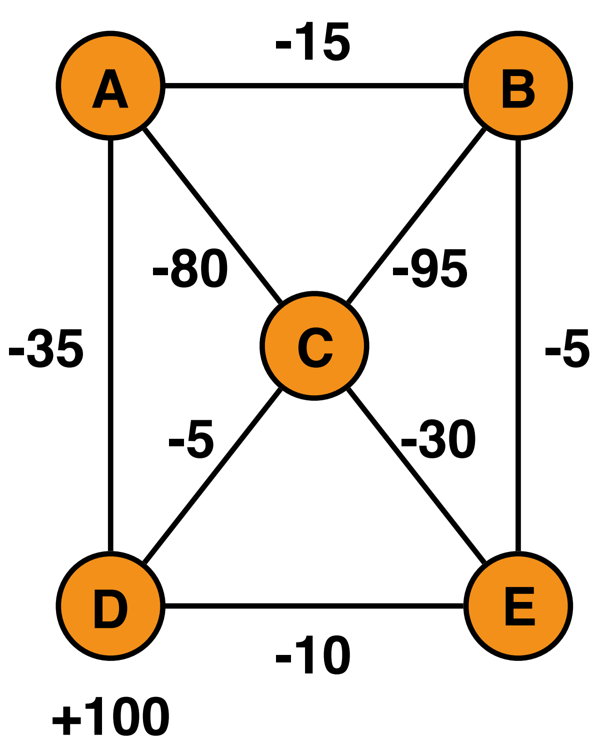
Dynamic Programming Example



N-3		N-2		N-1		N		
		D	165	D	65	A	0	A
		E	85	B	0	B	0	B
		D	195	D	95	C	0	C
		D	300	D	200	D	100	D
		D	190	D	90	E	0	E
$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	

-35 -5 -15 -10

Dynamic Programming Example



	N-3		N-2		N-1		N	
	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$	$\pi(x)$	$V(x)$
A	D	165	D	165	D	65	A	0
B	E	185	E	85	B	0	B	0
C	D	295	D	195	D	95	C	0
D	D	400	D	300	D	200	D	100
E	D	290	D	190	D	90	E	0

-35 -5 -15 -10

Dynamic Programming

- Start at the end (t=N) and move backwards
- (Optimal) value function defines optimal future reward

$$V_t^*(x)$$

- Recursively
  - Compute the optimal action

$$u_t^* = \arg\max_u \{r(x,u) + V_{t+1}^*(f(x,u))\}$$

  - Compute the next optimal value function

$$V_t^*(x) = \max_u \{r(x,u) + V_{t+1}^*(f(x,u))\}$$

- Divides the problem into individual greedy steps (easier)

Linear Quadratic Regulator

Linear Quadratic Regulator

- Continuous states x_t and actions u_t
- Linear (Deterministic) System:

$$x_{t+1} = Ax_t + Bu_t$$

- Quadratic Reward:

$$r(x_t, u_t) = -x_t^T Q x_t - u_t^T R u_t$$

$$Q = Q^T \geq 0 \qquad\qquad R = R^T > 0$$

- What is the optimal feedback controller? $u_t = \pi(x_t)$

LQR

- Start at the end

$$t = N$$

- Optimal final action

$$u^* = 0$$

- Value function

$$V_N^*(x_N) = -x_N^T Q x_N = -x_N^T P_N x_N$$

  - Simply the final state reward

LQR

- Take a step back $\quad t = N - 1$
- Next value function is given by

$$V^*_{N-1}(x_{N-1}) = \max_u (r(x_{N-1}, u) + V^*_N(x_N))$$

$$= \max_u (-x^T_{N-1} Q x_{N-1} - u^T R u - x^T_N P_N x_N)$$

$$= \max_u (-x^T_{N-1} Q x_{N-1} - u^T R u - (A x_{N-1} + B u)^T P_N (A x_{N-1} + B u))$$

- What is the optimal action?

LQR

- Compute optimal action by setting derivative to zero

$$\frac{\mathrm{d}}{\mathrm{d}u}\{-x_{N-1}^T Q x_{N-1} - u^T R u - (A x_{N-1} + B u)^T P_N (A x_{N-1} + B u)\} = 0$$

$$0 = -R u_{N-1}^* - B^T P_N A x_{N-1} - B^T P_N B u_{N-1}^*$$

$$(R + B^T P_N B) u_{N-1}^* = -B^T P_N A x_{N-1}$$

$$u_{N-1}^* = -(R + B^T P_N B)^{-1} B^T P_N A x_{N-1}$$

$$u_{N-1}^* = K_{N-1} x_{N-1}$$

- Linear feedback controller is optimal

LQR

- Plugging our optimal action into our value function

$$V^*_{N-1}(x_{N-1}) = -x^T_{N-1}Qx_{N-1} - x^T_{N-1}K^T_{N-1}RK_{N-1}x_{N-1} - ((A+BK_{N-1})x_{N-1})^T P_N((A+BK_{N-1})x_{N-1})$$

$$V^*_{N-1}(x_{N-1}) = \underbrace{x^T_{N-1}(-Q - K^T_{N-1}RK_{N-1} - (A+BK_{N-1})^T P_N(A+BK_{N-1}))x_{N-1}}$$

$$V^*_{N-1}(x_{N-1}) = -x^T_{N-1}P_{N-1}x_{N-1}$$

- Value function has a quadratic form <u>again</u>

- The pattern repeats itself over and over again

- At each step of the recursion:
  - Compute the optimal feedback gain

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

  - Compute the new value function

$$-P_t = (-Q - K_t^T R K_t - (A + BK_t)^T P_{t+1}(A + BK_t))$$

LQR Example

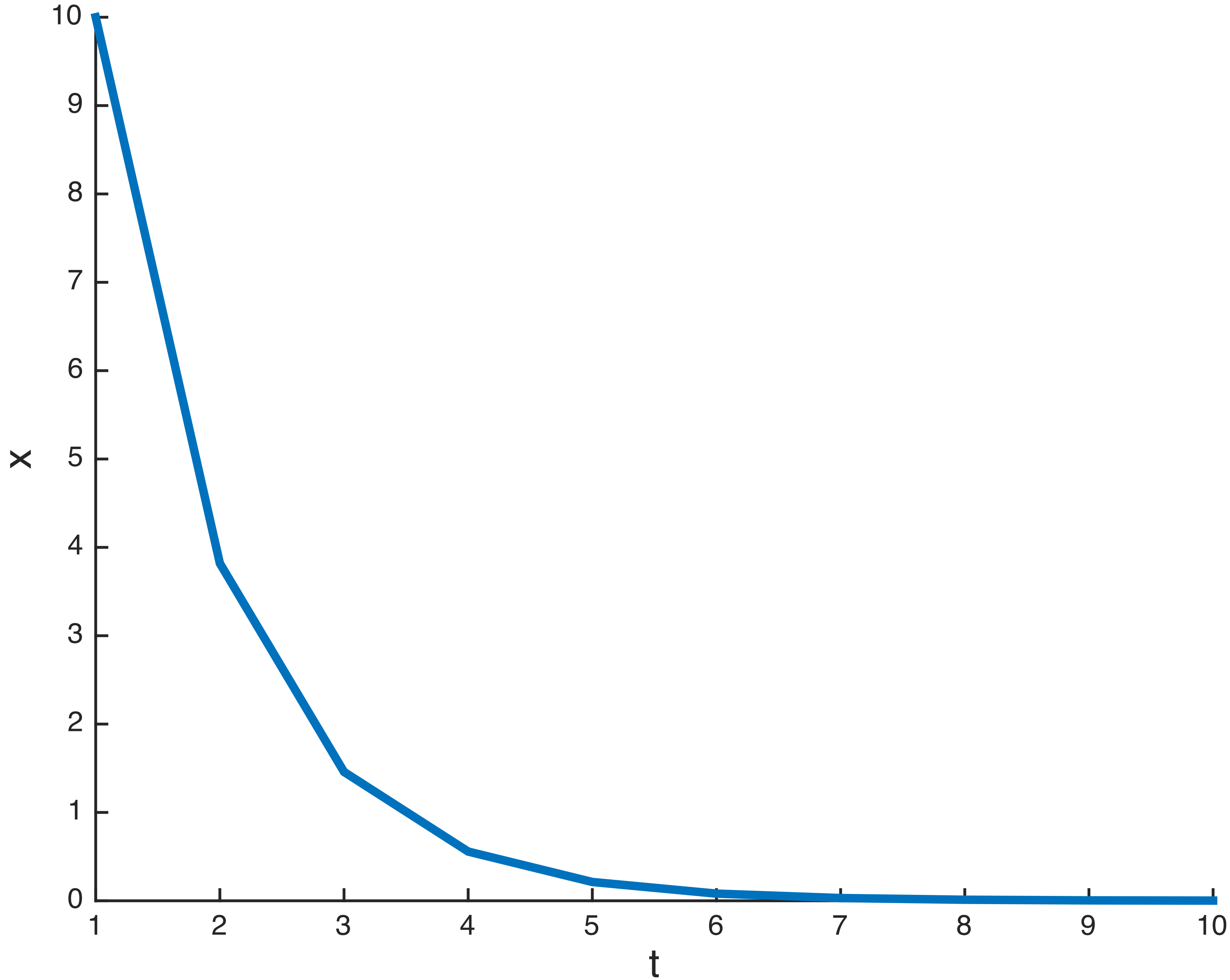
- Consider the following problem

$$x_{t+1} = x_t + u_t \qquad A = 1, B = 1$$

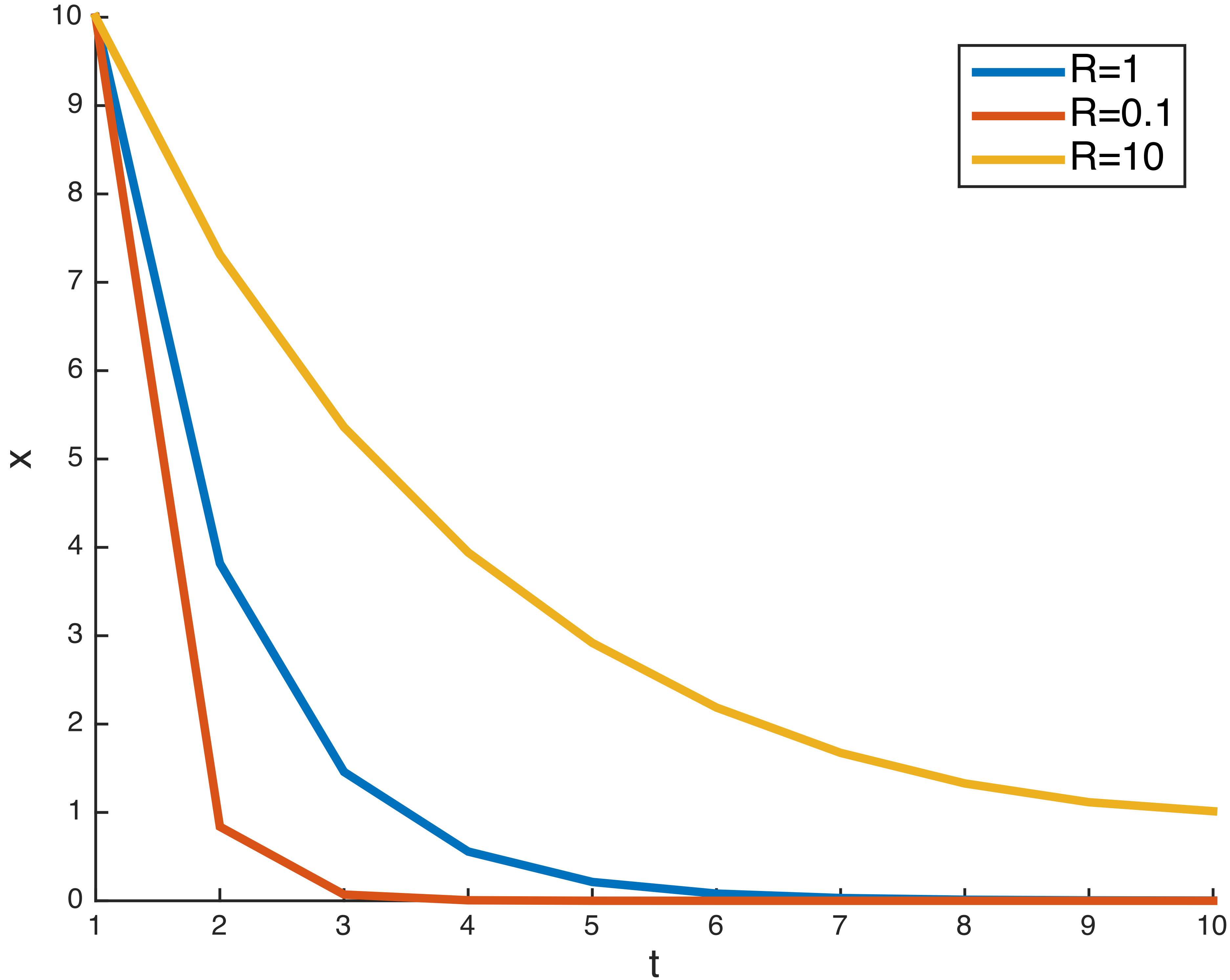
$$Q = 1 \qquad R = 1 \qquad N = 10$$

t	P	K
1	1.6180	-0.6180
2	1.6180	-0.6180
3	1.6180	-0.6180
4	1.6180	-0.6180
5	1.6180	-0.6180
6	1.6176	-0.6176
7	1.6154	-0.6154
8	1.6000	-0.6000
9	1.5000	-0.5000
10	1.0000	0

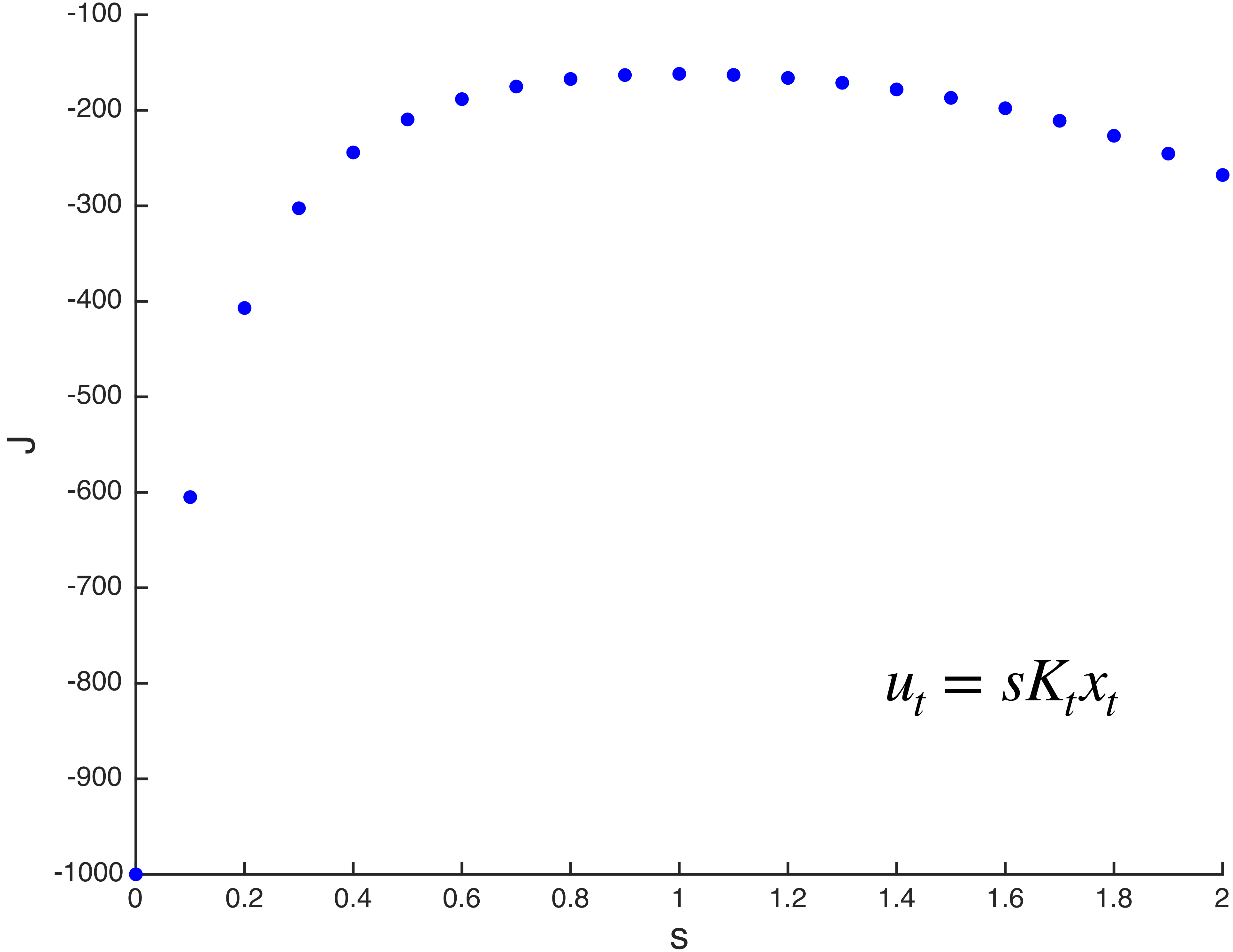
LQR Example



LQR Example

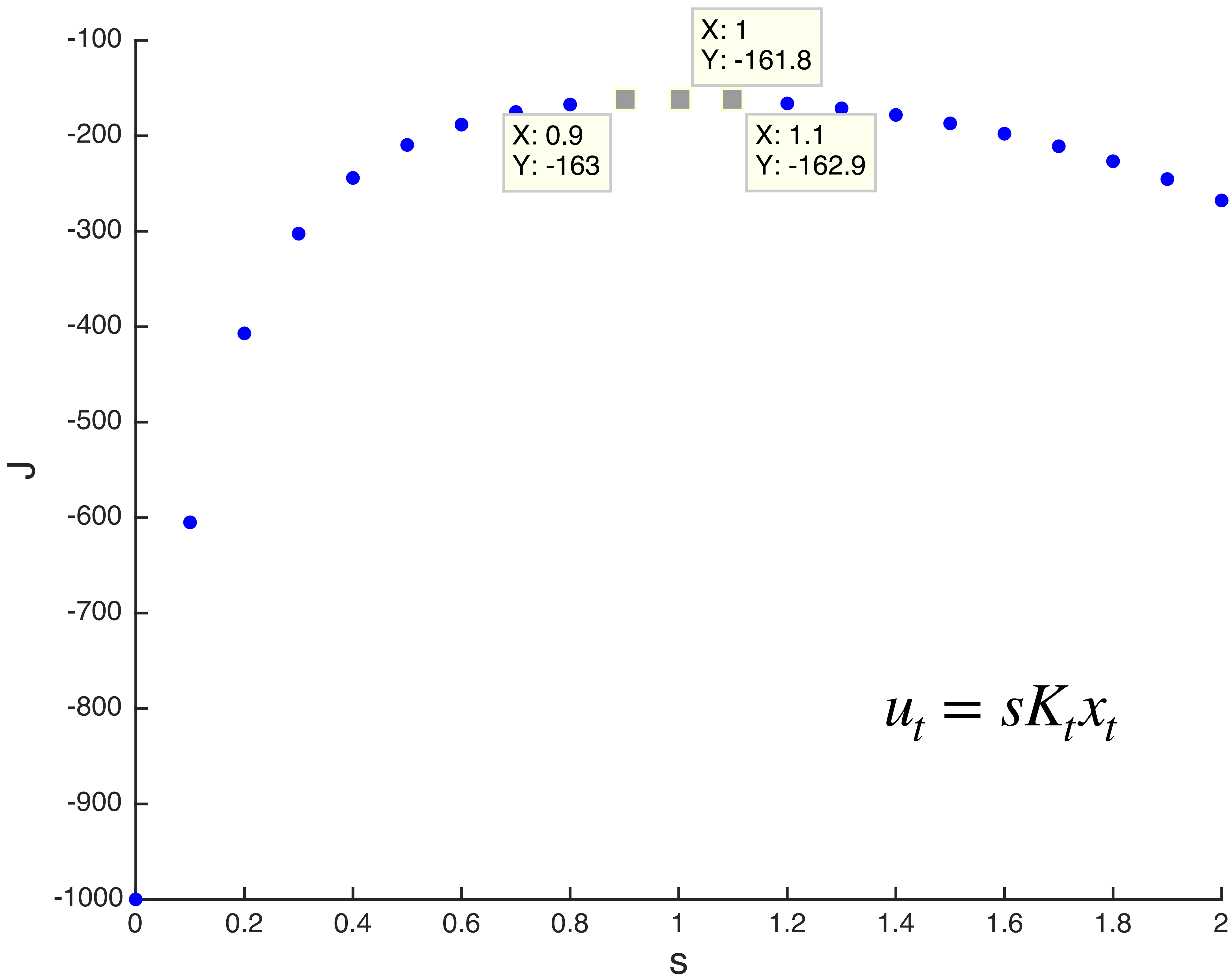


LQR Example



$$u_t = sK_t x_t$$

LQR Example


X: 1
Y: -161.8
X: 0.9
Y: -163
X: 1.1
Y: -162.9
-100
-200
-300
-400
-500
-600
-700
-800
-900
-1000
J
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
s
$u_t = sK_tx_t$

LQR Extensions

Extensions

- Given the basic LQR, lets consider the following:

- What happens if we have a terminal cost?
- What happens if we add noise to the system?
- What to do if we have noisy observations?
- What to do if we have a time-dependent system?
- What to do if we have a nonlinear system?

Terminal cost

- May want to have a different cost for final state

$$\max -x_N^T Q_N x_N + \sum_{t=0}^{N-1} -x_t^T Q x_t - u_t^T R u_t$$

  - Can estimate of future rewards after the horizon
  - Want to reach certain state at t=N, but ignore state before

- Simply set the initial value to terminal reward

$$P_N = Q_N$$

and do the same as before

Process Noise

- What happens when we add Gaussain noise?

$$x_{t+1} = Ax_t + Bu_t + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Does it change the optimal controller?

- Does it change the value function?

Process Noise

- Does it change the optimal controller?

$$\frac{\mathrm{d}}{\mathrm{d}u}\{-x_{N-1}^T Q x_{N-1} - u^T R u - E[(Ax_{N-1} + Bu + \epsilon)^T P_N (Ax_{N-1} + Bu + \epsilon)]\} = 0$$

$$= -Ru - B^T P_N(Ax_{N-1} + Bu + \underbrace{E[\epsilon]}_{0})$$

$$= -Ru - B^T P_N(Ax_{N-1} + Bu)$$

- Nope, the controller is the same

Process Noise

- Does it change the value function?

$$V^*_{N-1}(x_{N-1}) = -x^T_{N-1}Qx_{N-1} - x^T_{N-1}K^T_{N-1}RK_{N-1}x_{N-1}$$
$$-E[((A+BK_{N-1})x_{N-1}+\epsilon)^T P_N((A+BK_{N-1})x_{N-1}+\epsilon)]$$

$$V^*_{N-1}(x_{N-1}) = -x^T_{N-1}Qx_{N-1} - x^T_{N-1}K^T_{N-1}RK_{N-1}x_{N-1}$$
$$-((A+BK_{N-1})x_{N-1})^T P_N((A+BK_{N-1})x_{N-1}) - E[\epsilon^T P_N \epsilon]$$

$$V^*_{N-1}(x_{N-1}) = -x^T_{N-1}P_{N-1}x_{N-1} + const.$$

Constant

Process Noise

- Does it change the value function?

$$V_{N-1}^*(x_{N-1}) = -x_{N-1}^T Q x_{N-1} - x_{N-1}^T K_{N-1}^T R K_{N-1} x_{N-1}$$
$$-E[((A + BK_{N-1})x_{N-1} + \epsilon)^T P_N ((A + BK_{N-1})x_{N-1} + \epsilon)]$$
$$+const.$$

$$V_{N-1}^*(x_{N-1}) = -x_{N-1}^T Q x_{N-1} - x_{N-1}^T K_{N-1}^T R K_{N-1} x_{N-1}$$
$$-((A + BK_{N-1})x_{N-1})^T P_N ((A + BK_{N-1})x_{N-1}) - E[\epsilon^T P_N \epsilon]$$
$$+const.$$

$$V_{N-1}^*(x_{N-1}) = -x_{N-1}^T P_{N-1} x_{N-1} \; +const.$$

- Adds a meaningless offset to the value
- (constant is removed when taking derivative for action)

Process Noise

- What happens when we add Gaussian noise?

$$x_{t+1} = Ax_t + Bu_t + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Does it change the optimal controller?
  - Nope
- Does it change the value function?
  - Adds an offset that we can safely ignore

- The algorithm remains the same!

LQG

- What do we do if we have noisy Gaussian observations?

$$x_{t+1} = Ax_t + Bu_t + \epsilon$$

$$y_t = Cx_t + \delta$$

- Use a Kalman filter to estimate the state distribution
- Use the mean state estimate for the feedback

$$u_t^* = K_t \hat{\mu}_t$$

state estimate from Kalman filter

Time Dependent System

- What if we have a time dependent system?

$$x_{t+1} = A_t x_t + B_t u_t$$

  - Note that A and B are now time dependent

- Same as before, but use the matrices from that time step

$$K_t = -(R_t + B_t^T P_{t+1} B_t)^{-1} B_t^T P_{t+1} A_t$$

$$-P_t = (-Q_t - K_t^T R_t K_t - (A_t + B_t K_t)^T P_{t+1} (A_t + B_t K_t))$$

Non-linear System

- What if the system is non-linear?

$$x_{t+1} = f(x_t, u_t)$$

- Can only control the system about a fixed point $\hat{x}$ if

$$\exists \hat{u} \text{ s. t. } \hat{x} = f(\hat{x}, \hat{u})$$

- Linearize the model about the point

$$x_{t+1} = f(x, u) \approx f(\hat{x}, \hat{u}) + \frac{\partial f(\hat{x}, \hat{u})}{\partial x}(x_t - \hat{x}) + \frac{\partial f(\hat{x}, \hat{u})}{\partial u}(u_t - \hat{u})$$

$$x_{t+1} - \hat{x} = \frac{\partial f(\hat{x}, \hat{u})}{\partial x}(x_t - \hat{x}) + \frac{\partial f(\hat{x}, \hat{u})}{\partial u}(u_t - \hat{u})$$

$$\tilde{x}_{t+1} = A\tilde{x}_t + B\tilde{u}_t$$

- Use LQR algorithm as before with linearised model

Trajectory Tracking

- What if we want to track a sequence of points?

$$\hat{x}_0, \ldots, \hat{x}_N \qquad\qquad x_{t+1} = f(x_t, u_t)$$

- Require that

$$\exists \hat{u}_t \forall t \in \{0, \ldots, N-1\} \text{ s. t. } \hat{x}_{t+1} = f(\hat{x}_t, \hat{u}_t)$$

- Linearize about each point along trajectory

$$x_{t+1} = f(\hat{x}_t, \hat{u}_t) + \underbrace{\frac{\partial f(\hat{x}_t, \hat{u}_t)}{\partial x}}_{A_t}(x_t - \hat{x}_t) + \underbrace{\frac{\partial f(\hat{x}, \hat{u}_t)}{\partial u}}_{B_t}(u_t - \hat{u}_t)$$

- Apply quadratic rewards based on $x_t - \hat{x}_t$ and $u_t - \hat{u}_t$
- Apply algorithm as before

Iterative LQR

Iterative LQR

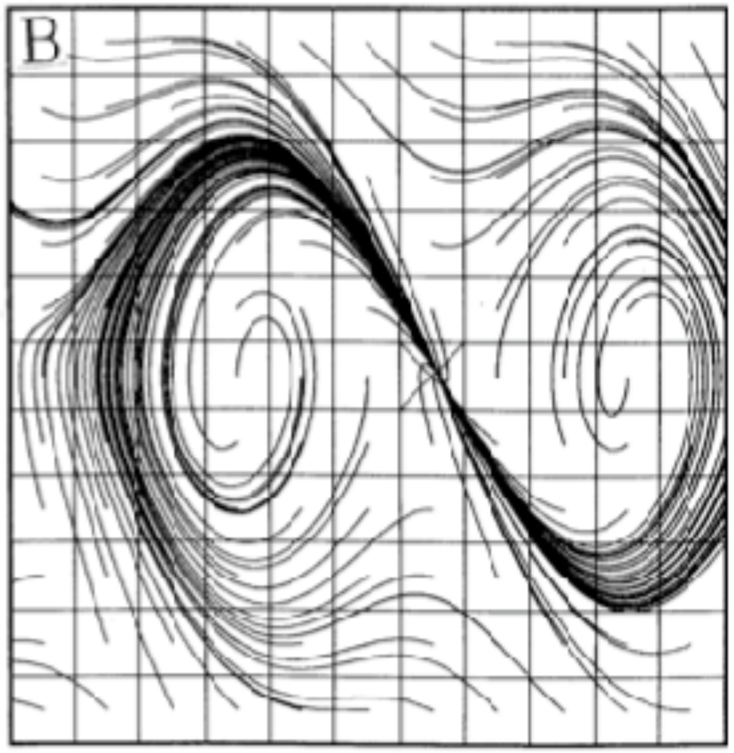
- What if we want to optimise (time-dependent) controllers?
- Iterative Linear Quadratic Regulators
- Given:
  a model, reward function, initial state, initial controller

1. Simulate the current controller using the model
2. Linearize model around visited points
3. Approximate quadratic reward function about visited points
4. Use LQR alg. to compute optimal controllers per time step
5. Repeat until convergence

- Note: Reward function may be non-linear
  Add tracking reward to ensure Q and R are valid

Iterative LQR

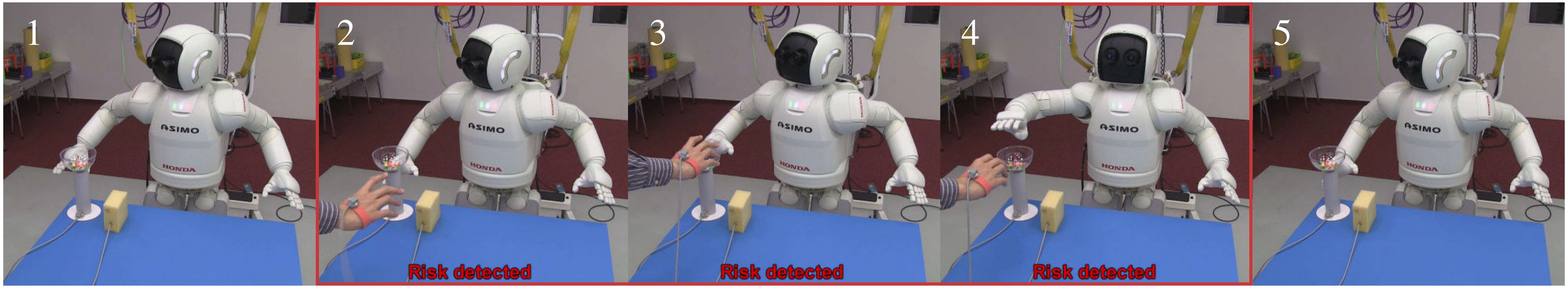
- ILQR is a local optimization method on convex reward
  - The algorithm only finds a local solution
  - The algorithm may not converge
- Can patch together local polices to approximate global
  - Each LQR covers a channel through the space



Model Predictive Control

- Can use ILQR as part of model predictive control
  1. Get current state
  2. Apply optimization for a horizon N
  3. Execute first step
  4. Repeat
- Computing individual actions at each time step
- Allows for look ahead (e.g., avoid obstacles)
- Does not need to consider full horizon
- Still relies on having a reasonably good model

Protect Candy Experiment



Questions?