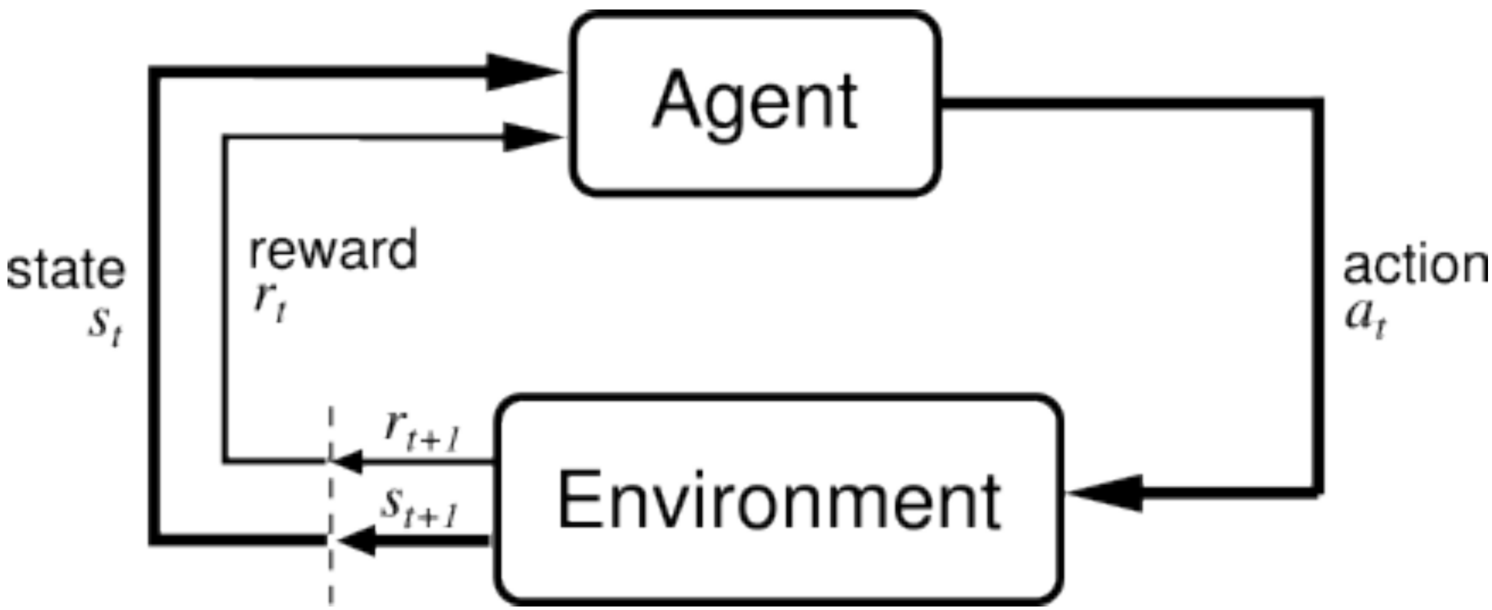
Discretization

Pieter Abbeel
UC Berkeley EECS

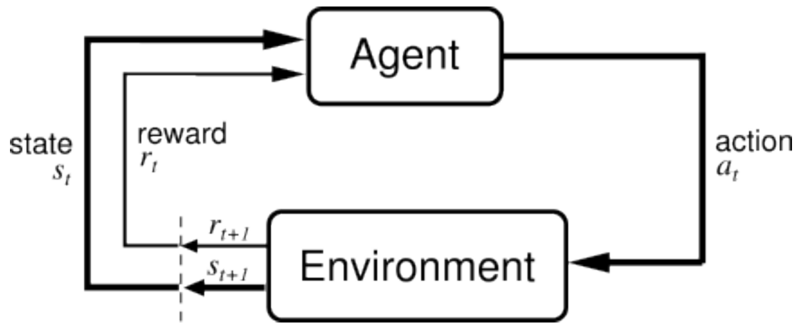
Markov Decision Process



Assumption: agent gets to observe the state

[Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998]

Markov Decision Process (S, A, T, R, H)



Given

- S: set of states
- A: set of actions
- T: S x A x S x {0,1,…,H} → [0,1], $T_t(s,a,s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$
- R: S x A x S x {0, 1, …, H} → $\Re$ $R_t(s,a,s')$ = reward for ($s_{t+1} = s'$, $s_t = s$, $a_t = a$)
- H: horizon over which the agent will act

Goal:

- Find π : S x {0, 1, …, H} → A that maximizes expected sum of rewards, i.e.,

$$\pi^* = \arg\max_{\pi} \mathrm{E}[\sum_{t=0}^{H} R_t(S_t, A_t, S_{t+1}) | \pi]$$

Value Iteration

- Algorithm:
  - Start with $V_0^*(s) = 0$ for all s.
  - For i=1, … , H

    For all states s $\in$ S:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

$$\pi_{i+1}^*(s) \leftarrow \arg\max_{a \in A} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$$

  This is called a value update or Bellman update/back-up

- $V_i^*(s)$ = the expected sum of rewards accumulated when starting from state s and acting optimally for a horizon of i steps
- $\pi_i^*(s)$ = the optimal action when in state s and getting to act for a horizon of i steps

Continuous State Spaces

- S = continuous set

- Value iteration becomes impractical as it requires to compute, for all states s $\in$ S:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + V_i^*(s') \right]$$

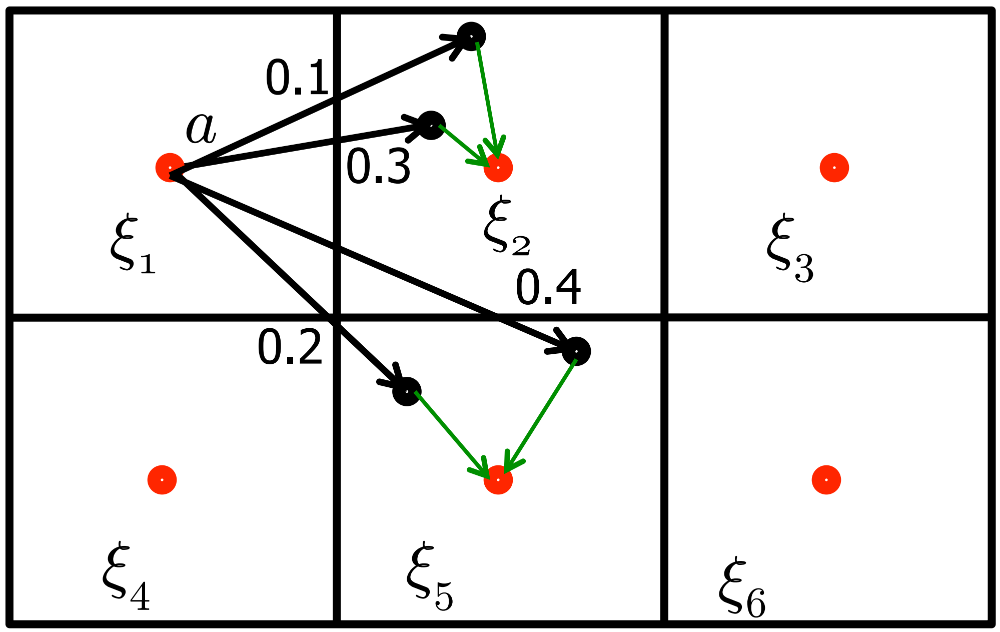
Markov chain approximation to continuous state space dynamics model ("discretization")

- Original MDP (S, A, T, R, H)

  - Grid the state-space: the vertices are the discrete states.
  - Reduce the action space to a finite set.
    - Sometimes not needed:
      - When Bellman back-up can be computed exactly over the continuous action space
      - When we know only certain controls are part of the optimal policy (e.g., when we know the problem has a "bang-bang" optimal solution)
  - Transition function: see next few slides.

- Discretized MDP $(\bar{S}, \bar{A}, \bar{T}, \bar{R}, H)$

Discretization Approach A: Deterministic Transition onto Nearest Vertex --- 0'th Order Approximation



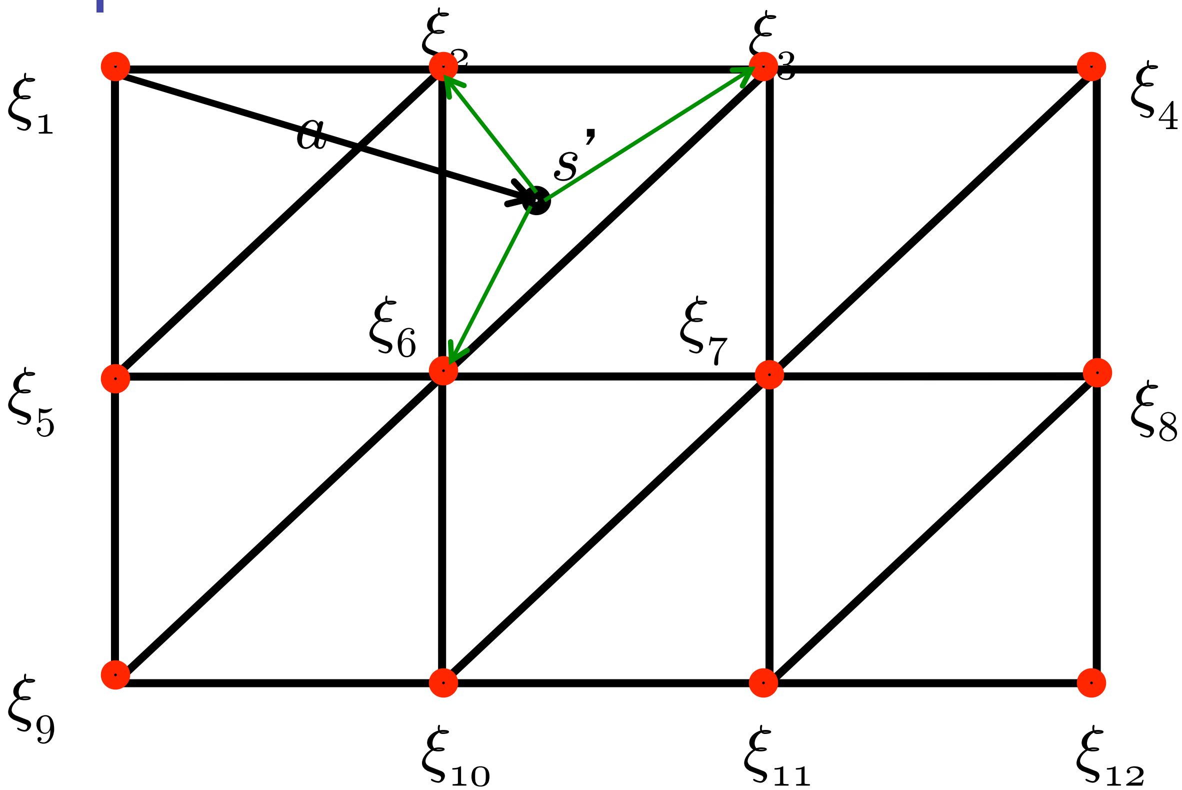
Discrete states: { ξ_1 , …, ξ_6 }

$$P(\xi_2|\xi_1, a) = 0.1 + 0.3 = 0.4;$$
$$P(\xi_5|\xi_1, a) = 0.4 + 0.2 = 0.6$$

Similarly define transition probabilities for all ξ_i

- → Discrete MDP just over the states { ξ_1 , …, ξ_6 }, which we can solve with value iteration
- If a (state, action) pair can results in infinitely many (or very many) different next states: Sample next states from the next-state distribution

Discretization Approach B: Stochastic Transition onto Neighboring Vertices --- 1'st Order Approximation



Discrete states: { ξ_1 , …, ξ_{12} }

$$P(\xi_2|\xi_1, a) = p_A;$$
$$P(\xi_3|\xi_1, a) = p_B;$$
$$P(\xi_6|\xi_1, a) = p_C;$$
$$\text{s.t. } s' = p_A\xi_2 + p_B\xi_3 + p_C\xi_6$$

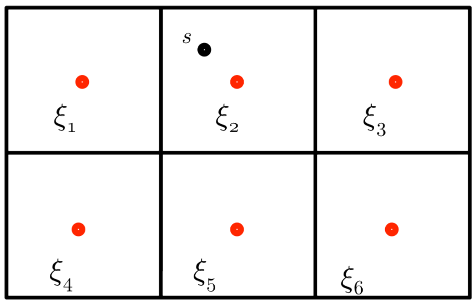
- If stochastic: Repeat procedure to account for all possible transitions and weight accordingly
- Need not be triangular, but could use other ways to select neighbors that contribute. "Kuhn triangulation" is particular choice that allows for efficient computation of the weights p_A, p_B, p_C, also in higher dimensions

Discretization: Our Status

- Have seen two ways to turn a continuous state-space MDP into a discrete state-space MDP
- When we solve the discrete state-space MDP, we find:
  - Policy and value function for the discrete states
  - They are optimal for the discrete MDP, but typically not for the original MDP
- Remaining questions:
  - How to act when in a state that is not in the discrete states set?
  - How close to optimal are the obtained policy and value function?

How to Act (i): 0-step Lookahead

- **For non-discrete state s choose action based on policy in nearby states**
  - **Nearest Neighbor:** $\pi(s) = \pi(\xi_i) \quad \text{for} \quad \xi_i = \arg \min_{\xi \in \{\xi_1, \ldots, \xi_N\}} \|s - \xi\|$

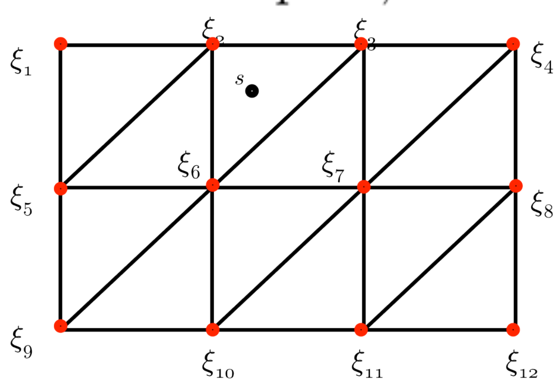


E.g., $\pi(s) = \pi(\xi_2)$

  - **(Stochastic) Interpolation:** Find $p_1, \ldots, p_N$ s.t. $s = \sum_{i=1}^{N} p_i \xi_i$

Policy at s: choose $\pi(\xi_i)$ with probability p_i.

If continuous action space, can interpolate actions and choose $\sum_{i=1}^{N} p_i \pi(\xi_i)$



E.g., let p_2, p_3, p_6 be such that
$s = p_2\xi_2 + p_3\xi_3 + p_6\xi_6$
then choose $\pi(\xi_2)$, $\pi(\xi_3)$, $\pi(\xi_6)$
with probabilities p_2, p_3, p_6 respectively.

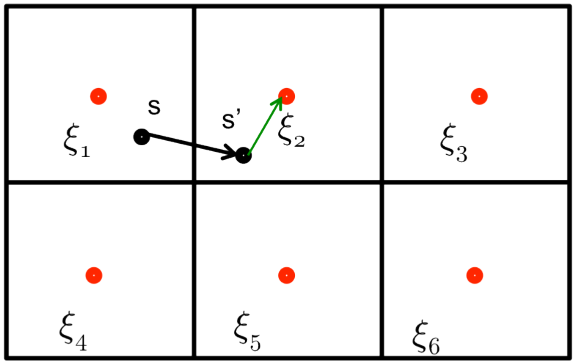
How to Act (ii): 1-step Lookahead

- **Use value function found for discrete MDP**

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \sum_i P(\xi_i; s') V(\xi_i) \right)$$

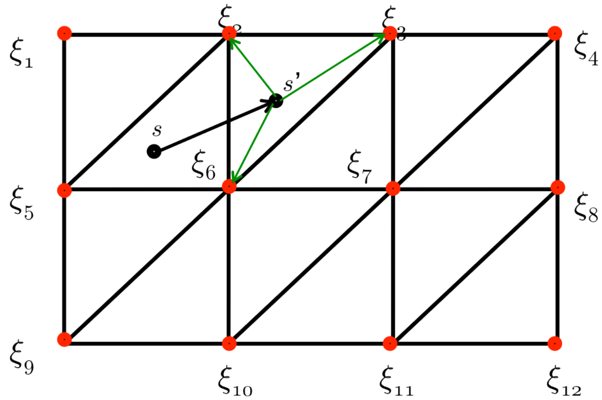
  - **Nearest Neighbor:**

$$P(\xi_i; s') = \begin{cases} 1 & \text{if } \xi_i = \arg\min_{\xi \in \{\xi_1, \ldots, \xi_N\}} \|s - \xi\| \\ 0 & \text{otherwise} \end{cases}$$



  - **(Stochastic) Interpolation:**

$$P(\xi_i; s') \text{ such that } \quad s' = \sum_{i=1}^{N} P(\xi_i; s') \xi_i$$



How to Act (iii): n-step Lookahead

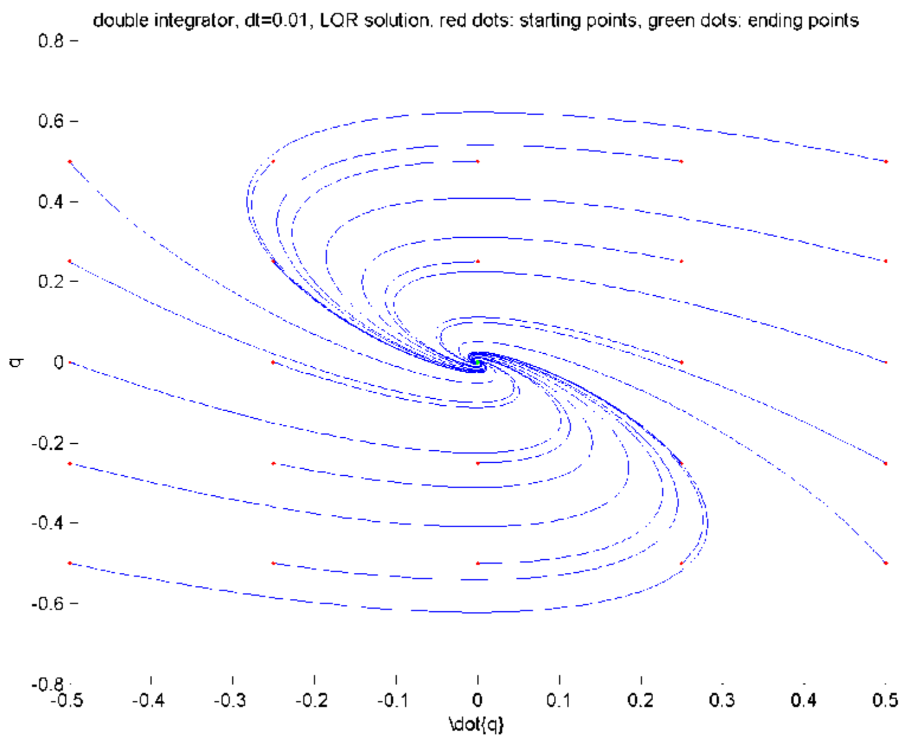
- Think about how you could do this for n-step lookahead
- Why might large n not be practical in most cases?

Example: Double integrator---quadratic cost

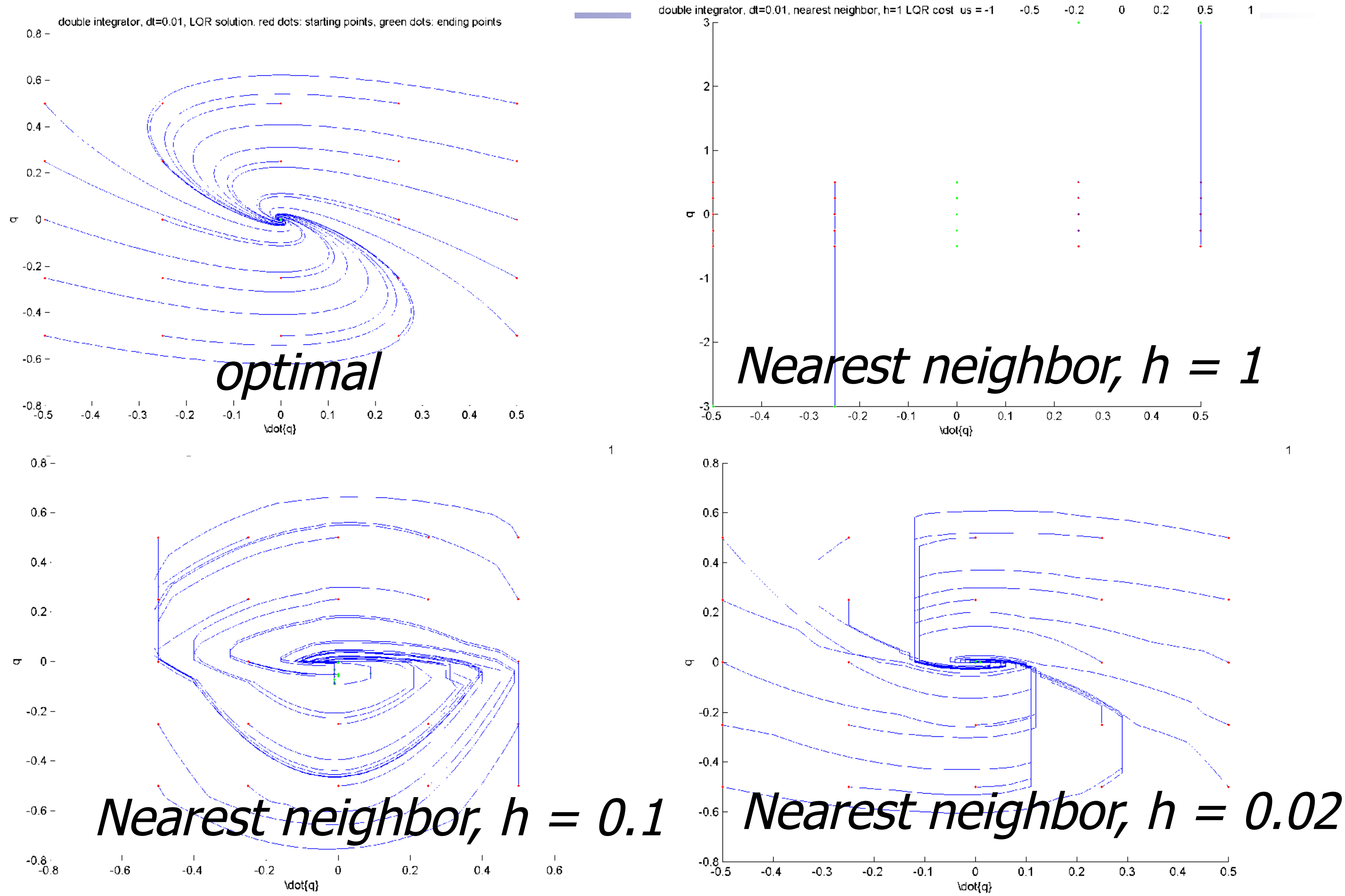
- Dynamics:

$$
\begin{aligned}
q_{t+1} &= q_t + \dot{q}_t \delta t \\
\dot{q}_{t+1} &= \dot{q}_t + u \delta t
\end{aligned}
$$

- Cost function: $g(q, \dot{q}, u) = q^2 + u^2$

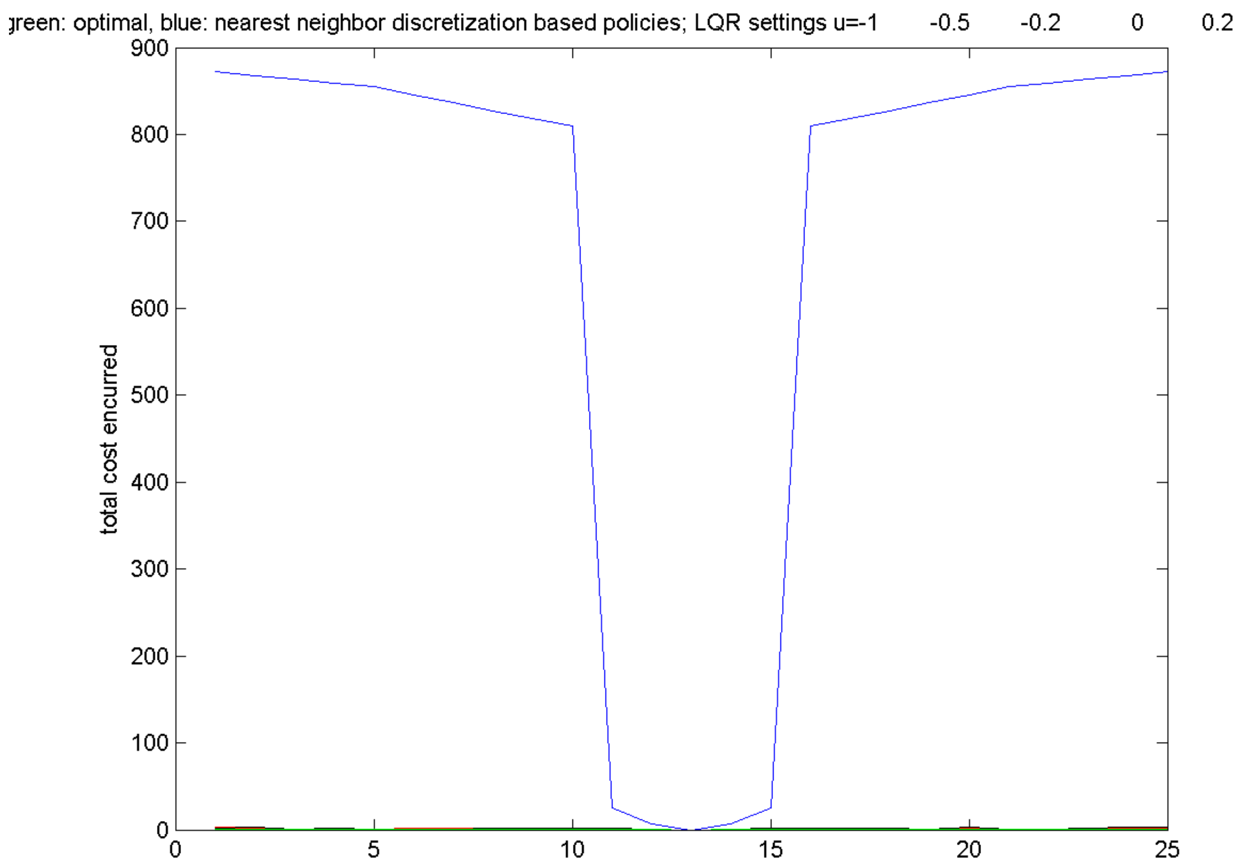


0'th Order Interpolation, 1 Step Lookahead for Action Selection --- Trajectories

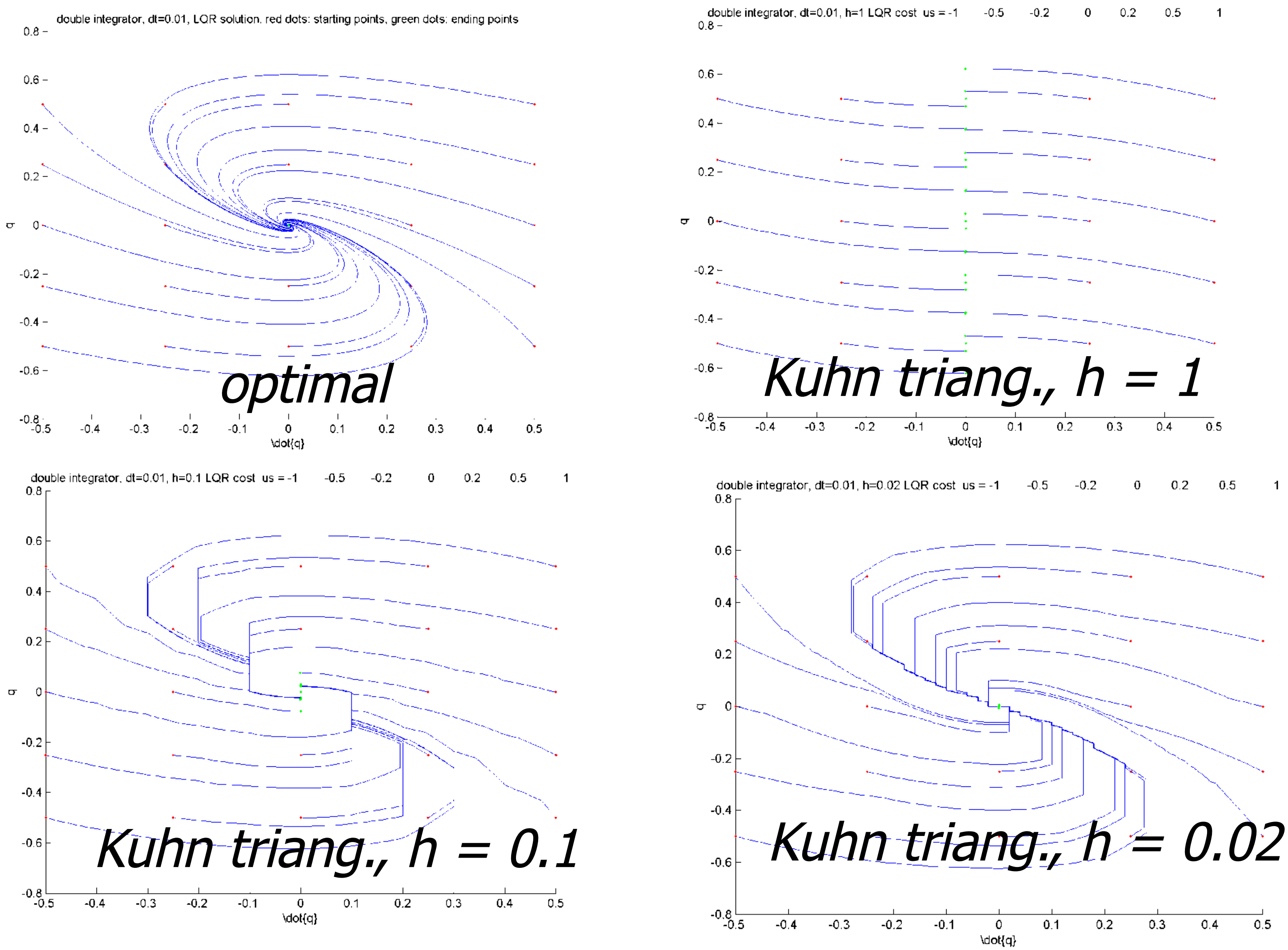


dt=0.1

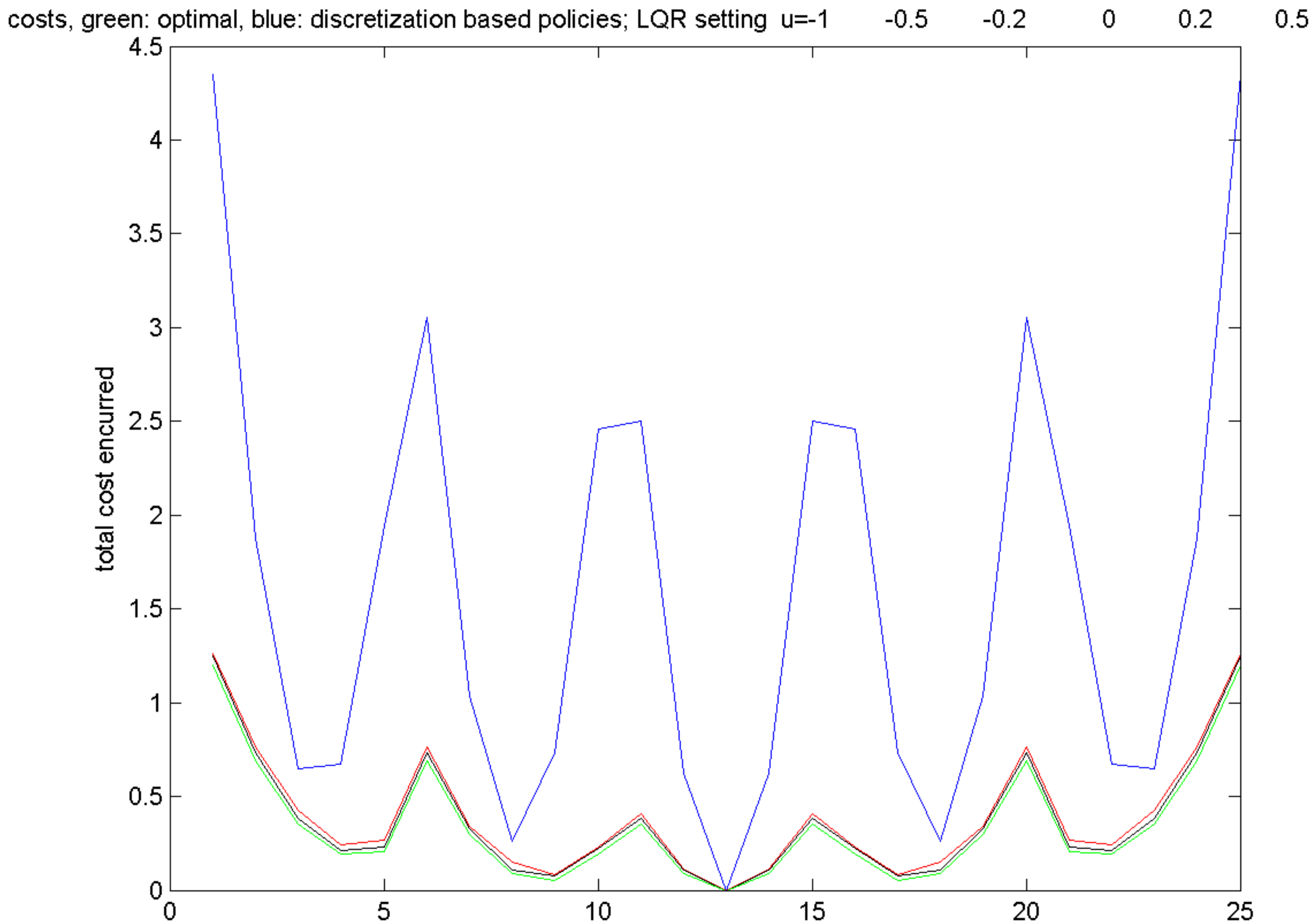
0'th Order Interpolation, 1 Step Lookahead for Action Selection --- Resulting Cost



1st Order Interpolation, 1-Step Lookahead for Action Selection --- Trajectories



1st Order Interpolation, 1-Step Lookahead for Action Selection --- Resulting Cost



Discretization Quality Guarantees

- Typical guarantees:
  - Assume: smoothness of cost function, transition model
  - For $h \rightarrow 0$, the discretized value function will approach the true value function

- To obtain guarantee about resulting policy, combine above with a general result about MDP's:
  - One-step lookahead policy based on value function V which is close to V* is a policy that attains value close to V*

Quality of Value Function Obtained from Discrete MDP: Proof Techniques

- Chow and Tsitsiklis, 1991:
  - Show that one discretized back-up is close to one "complete" back-up + then show sequence of back-ups is also close
- Kushner and Dupuis, 2001:
  - Show that sample paths in discrete stochastic MDP approach sample paths in continuous (deterministic) MDP [also proofs for stochastic continuous, bit more complex]
- Function approximation based proof (see later slides for what is meant with "function approximation")
  - Great descriptions: Gordon, 1995; Tsitsiklis and Van Roy, 1996

Example result (Chow and Tsitsiklis,1991)

A.1: $|g(x, u) - g(x', u')| \le K \|(x, u) - (x', u')\|_\infty$, for all $x, x' \in S$ and $u, u' \in C$;

A.2: $|P(y \mid x, u) - P(y' \mid x', u')| \le K \|(y, x, u) - (y', x', u')\|_\infty$, for all $x, x', y, y' \in S$ and $u, u' \in C$;

A.3: for any $x, x' \in S$ and any $u' \in U(x')$, there exists some $u \in U(x)$ such that $\|u - u'\|_\infty \le K \|x - x'\|_\infty$;

A.4: $0 \le P(y \mid x, u) \le K$ and $\int_S P(y \mid x, u)\, dy = 1$, for all $x, y \in S$ and $u \in C$.

Theorem 3.1: There exist constants K_1 and K_2 (depending only on the constant K of assumptions A.1–A.4) such that for all $h \in (0, 1/2K]$ and all $J \in \mathscr{B}(S)$

$$\|TJ - \tilde{T}_h J\|_\infty \le (K_1 + \alpha K_2 \|J\|_S) h. \qquad (3.6)$$

Furthermore,

$$\|J^* - \tilde{J}_h^*\|_\infty \le \frac{1}{1-\alpha} (K_1 + \alpha K_2 \|J^*\|_S) h. \qquad (3.7)$$

Value Iteration with Function Approximation

Provides alternative derivation and interpretation of the discretization methods we have covered in this set of slides:

- Start with $V_0^*(s) = 0$ for all s.
- For i=0, 1, … , H-1

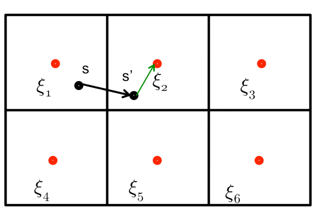
  for all states $s \in \bar{S}$, where $\bar{S}$ is the discrete state set

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \hat{V}_i^*(s') \right]$$

where $\hat{V}_i^*(s') = \sum_j P(\xi_j; s') V_i^*(\xi_j)$

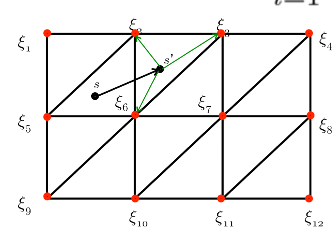
0'th Order Function Approximation

$$P(\xi_i; s') = \begin{cases} 1 & \text{if } \xi_i = \arg\min_{\xi \in \{\xi_1,\ldots,\xi_N\}} \|s - \xi\| \\ 0 & \text{otherwise} \end{cases}$$



1st Order Function Approximation

$P(\xi_i; s')$ such that $\quad s' = \sum_{i=1}^{N} P(\xi_i; s') \xi_i$



Discretization as function approximation

- 0'th order function approximation

  builds piecewise constant approximation of value function

- 1st order function approximatin

  builds piecewise (over "triangles") linear approximation of value function

Kuhn triangulation**

- Allows efficient computation of the vertices participating in a point's barycentric coordinate system and of the convex interpolation weights (aka the barycentric coordinates)

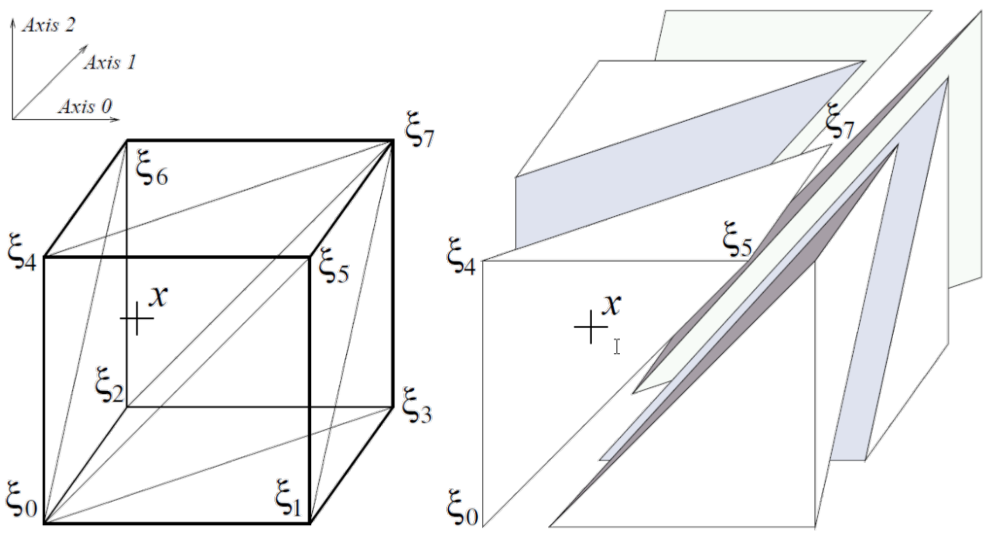


Figure 2. The Kuhn triangulation of a (3d) rectangle. The point x satisfying $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ is in the simplex $(\xi_0, \xi_4, \xi_5, \xi_7)$.

- See Munos and Moore, 2001 for further details.

Kuhn triangulation (from Munos and Moore)**

3.1. Computational issues

Although the number of simplexes inside a rectangle is factorial with the dimension d, the computation time for interpolating the value at any point inside a rectangle is only of order $(d \ln d)$, which corresponds to a sorting of the d relative coordinates $(x_0, ..., x_{d-1})$ of the point inside the rectangle.

Assume we want to compute the indexes $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing a point defined by its relative coordinates $(x_0, ..., x_{d-1})$ with respect to the rectangle in which it belongs to. Let $\{\xi_0, ..., \xi_{2^d}\}$ be the corners of this d-rectangle. The indexes of the corners use the binary decomposition in dimension d, as illustrated in Figure 2. Computing these indexes is achieved by sorting the coordinates from the highest to the smallest: there exist indices $j_0, ..., j_{d-1}$, permutation of $\{0, .., d-1\}$, such that $1 \geq x_{j_0} \geq x_{j_1} \geq ... \geq x_{j_{d-1}} \geq 0$. Then the indices $i_0, ..., i_d$ of the $(d+1)$ vertices of the simplex containing the point are: $i_0 = 0$, $i_1 = i_0 + 2^{j_0}$, ..., $i_k = i_{k-1} + 2^{j_{k-1}}$, ..., $i_d = i_{d-1} + 2^{j_{d-1}} = 2^d - 1$. For example, if the coordinates satisfy: $1 \geq x_2 \geq x_0 \geq x_1 \geq 0$ (illustrated by the point x in Figure 2) then the vertices are: ξ_0 (every simplex contains this vertex, as well as $\xi_{2^d-1} = \xi_7$), ξ_4 (we added 2^2), ξ_5 (we added 2^0) and ξ_7 (we added 2^1).

Let us define the *barycentric coordinates* $\lambda_0, ..., \lambda_d$ of the point x inside the simplex $\xi_{i_0}, ..., \xi_{i_d}$ as the positive coefficients (uniquely) defined by: $\sum_{k=0}^{d} \lambda_k = 1$ and $\sum_{k=0}^{d} \lambda_k \xi_{i_k} = x$. Usually, these barycentric coordinates are expensive to compute; however, in the case of Kuhn triangulation these coefficients are simply: $\lambda_0 = 1 - x_{j_0}$, $\lambda_1 = x_{j_0} - x_{j_1}$, ..., $\lambda_k = x_{j_{k-1}} - x_{j_k}$, ..., $\lambda_d = x_{j_{d-1}} - 0 = x_{j_{d-1}}$. In the previous example, the barycentric coordinates are: $\lambda_0 = 1 - x_2$, $\lambda_1 = x_2 - x_0$, $\lambda_2 = x_0 - x_1$, $\lambda_3 = x_1$.

Continuous time**

- One might want to discretize time in a variable way such that one discrete time transition roughly corresponds to a transition into neighboring grid points/regions
- Discounting: $$\exp(-\beta \delta t)$$

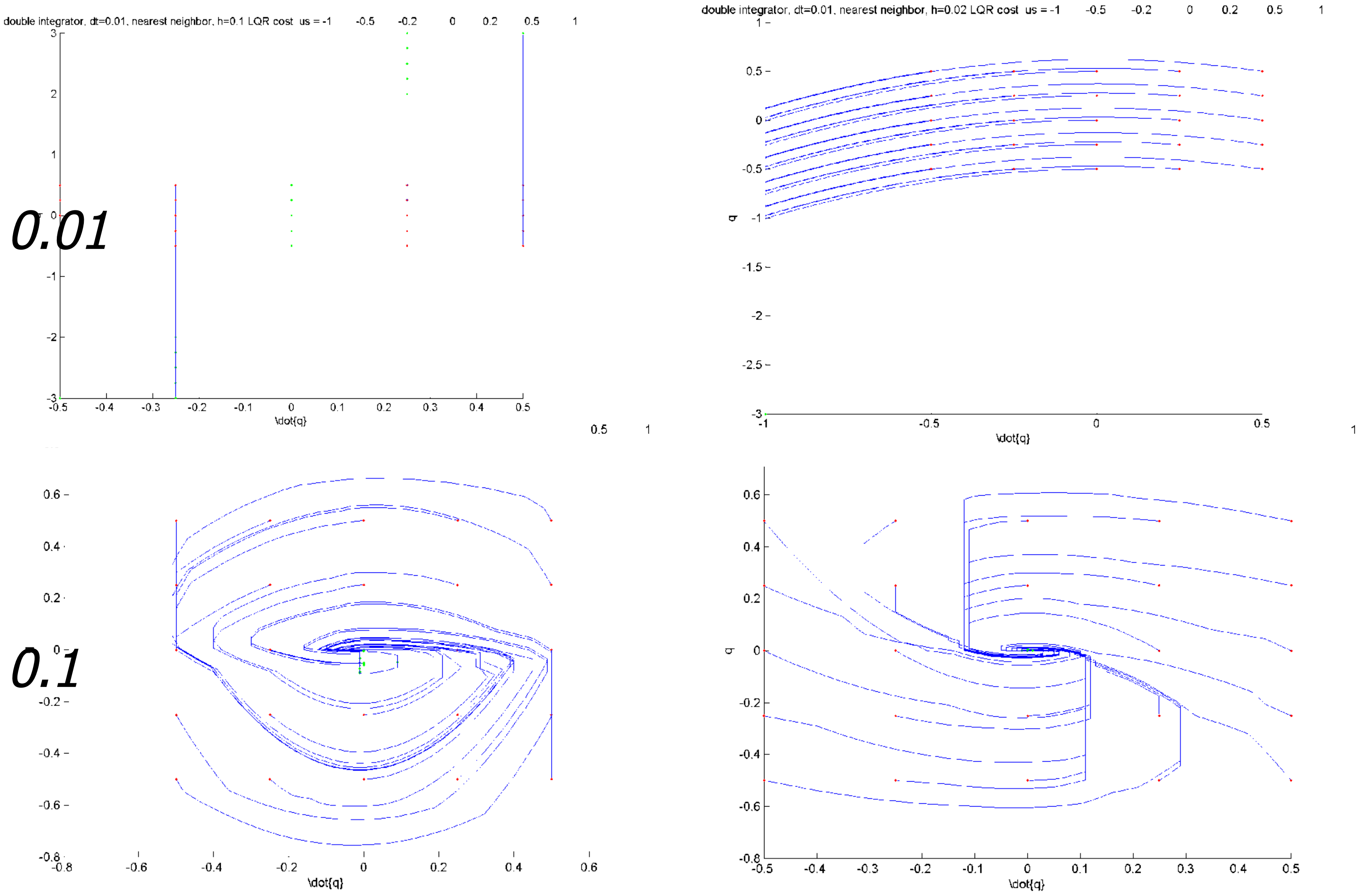
  δt depends on the state and action

See, e.g., Munos and Moore, 2001 for details.

Note: Numerical methods research refers to this connection between time and space as the CFL (Courant Friedrichs Levy) condition. Googling for this term will give you more background info.

!! 1 nearest neighbor tends to be especially sensitive to having the correct match [Indeed, with a mismatch between time and space 1 nearest neighbor might end up mapping many states to only transition to themselves no matter which action is taken.]

Nearest neighbor quickly degrades when time and space scale are mismatched**



dt= 0.01

dt= 0.1

h = 0.1

h = 0.02