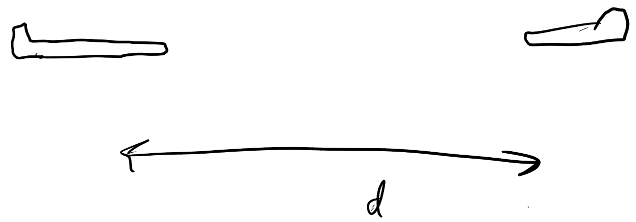


Continuous Space MDPs

Last Time

$U^\pi(s) = V^\pi(s)$ = expected sum of future rewards given start in s and follow π

$$U^\pi(s) = E\left[\sum_t \gamma^t R(s_t, a_t) \mid s_0 = s\right]$$

$U^*(s) = V^*(s)$ = " " under optimal policy

$R(s,a) = \sum_{s'} T(s'|s,a) R(s,a,s')$ = immediate reward for taking a in s

$$E_{s' \sim T(s'|s,a)}[R(s,a,s')]$$

$$Q^\pi(s,a) = E_{s' \sim T(s'|s,a)}\left[R(s,a,s') + \gamma U^\pi(s')\right]$$

Last Time

$Q^\pi(s,a)$ = expected future rewards given start in s, take a, and then follow π

$$\boxed{Q^\pi(s,a) = R(s,a) + \gamma E_{s' \sim T(s'|s,a)}\left[U^\pi(s')\right]}$$

- What are the differences between online and offline solutions?
- Are there solution techniques that are *independent* of the state space size?

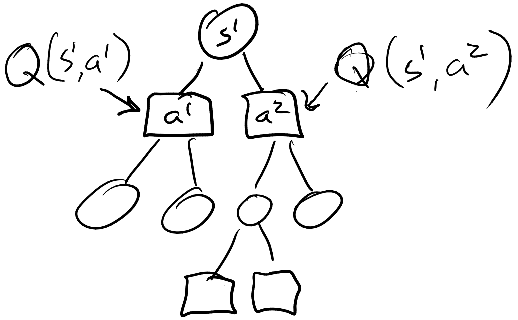
$Q^*(s,a)$ = " " then follow optimal policy

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

$$U^*(s) = \max_a Q^*(s,a)$$

$Q(s,a)$

approximation of Q^*



Guiding Questions

Guiding Questions

- What tools do we have to solve MDPs with continuous S and A?

Current Tool-Belt

Offline: VI

PI

Online: MCTS

Continuous S and A

Continuous S and A

e.g. $S \subseteq \mathbb{R}^n$, $A \subseteq \mathbb{R}^m$

$S = [-1, 1]^4$

Continuous S and A

e.g. $S \subseteq \mathbb{R}^n$, $A \subseteq \mathbb{R}^m$

The old rules still work!

$$U^*(s) = \max_a \left(R(s,a) + \gamma E[U^*(s')]\right)$$

$$U^\pi(s) = \ldots$$

$$B[U](s) = \underbrace{\max_a}_{\text{hard! optimization}} \left(R(s,a) + \gamma \underset{s' \sim T(s'|s,a)}{E}[U(s')]\right)$$

$$\underbrace{\int_{s' \in S} T(s'|s,a)\, U(s')\, ds}_{\text{hard}}$$

(but not as hard as max)

Today: Four Tools

1. LQR
2. Value Function Approximation
3. Sparse Sampling / Prog. Widening
4. MPC

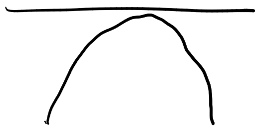
$S = \mathbb{R}^n \quad A = \mathbb{R}^m$ $\quad x_{k+1} = \underset{T_s}{A} x_k + \underset{T_a}{B} u_k$

1. Linear Dynamics, Quadratic Reward

$R_s = \begin{bmatrix} -5 & 0 \\ 0 & -10 \end{bmatrix}$

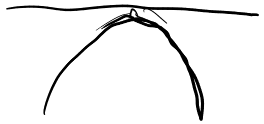
$s' \sim \mathcal{N}(T_s s + T_a a, \Sigma)$ $\quad s_{t+1} = T_s s_t + T_a a_t + w_t \quad w_t \sim \mathcal{N}(0, \Sigma)$

$R(s,a) = s^T R_s s + a^T R_a a$ $\quad$ goal: $s \to 0$ with small a



$R_s = -1$ $R_a = -1$

$R(s,a) = -s^2 - a^2$



$U_h^*(s) = \max_\pi \left[\sum_{t=0}^{h} R(s_t, a_t) \right]$ $\quad \pi_h^*$ "optimal h-step policy"

show that

$$\boxed{U_h^*(s) = s^T V_h s + q_h \qquad \pi_h^*(s) = -K_h s}$$

Proof by induction $\quad$ Base: $U_1(s) = \max_a (s^T R_s s + a^T R_a a) = s^T R_s s \quad \therefore V_1 = R_s \quad q_1 = 0$

Inductive step: if U_t is quadratic, then U_{t+1} is quadratic

$$U_{t+1}(s) = \max_a \left(R(s,a) + \cancel{\gamma} E[U_t(s')] \right)$$

$$= \max_a \left(s^T R_s s + a^T R_a a + \int p(w) U_t(T_s s + T_a a + w) dw \right)$$

$$= s^T R_s s + \max \left(a^T R_a a + \int p(w) \left((T_s s + T_a a + w)^T V_t (T_s s + T_a a + w) + q_t \right) dw \right.$$

$$= s^T R_s s + s^T T_s^T V_t T_s s + \max_a \left(a^T R_a a + 2 s^T T_s^T V_t T_a a + a^T T_a^T V_t T_a a \right) + \int p(w) w^T V_t w \, dw + q_t$$

a^* is where $\nabla_a(\text{max term}) = 0$

$$0 = 2R_a a^* + 2T_a^T V_t T_s s + 2T_a^T V_t T_a a^*$$

$$a^* = \underbrace{-(R_a + T_a^T V_t T_a)^{-1} T_a^T V_t T_s}_{K} s$$

$$U_{t+1}(s) = s^T \underbrace{\left(R_s + T_s^T V_t T_s - (T_a V_t T_s)^T (R_a + T_a^T V_t T_a)^{-1} (T_a^T V_t T_s)\right)}_{V_{t+1}} s + \underbrace{\int p(w)\, w^T V_t w\, dw + q_t}_{q_{t+1}}$$

$$U_{t+1}(s) = s^T V_{t+1} s + q_{t+1} \quad \square$$

as $h \to \infty$

$$V_\infty = T_s^T \left(V_\infty - V_\infty T_a (T_a^T V_\infty T_a + R_a)^{-1} T_a^T V_\infty\right) T_s + R_s$$

$$K_\infty = (T_a^T V_\infty T_a + R_a)^{-1} T_a^T V_\infty T_s$$

$$\boxed{\pi^*_\infty(s) = -K_\infty s}$$

K_∞ has no dependence on Σ

Certainty-Equivalence Principle

For L-Q problems

Optimal policy with noise = Optimal policy w/o noise

2. Value Function Approximation

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

$V_\theta(s) = \theta^\top \beta(s)$ (linear feature)

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

$V_\theta(s) = \theta^\top \beta(s)$ (linear feature)

Fitted Value Iteration

while not converged

$\theta \leftarrow \theta'$

$\hat{V}' \leftarrow B_{\text{approx}}[V_\theta]$

$\theta' \leftarrow \text{fit}(\hat{V}')$

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

$V_\theta(s) = \theta^\top \beta(s)$ (linear feature)

Fitted Value Iteration

while not converged

$\theta \leftarrow \theta'$

$\hat{V}' \leftarrow B_{\text{approx}}[V_\theta]$

$\theta' \leftarrow \text{fit}(\hat{V}')$

$$B_{\text{MC}(N)}[V_\theta](s) = \max_a \left(R(s,a) + \gamma \sum_{i=1}^{N} V_\theta(G(s,a,w_i)) \right)$$

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

$V_\theta(s) = \theta^\top \beta(s)$ (linear feature)

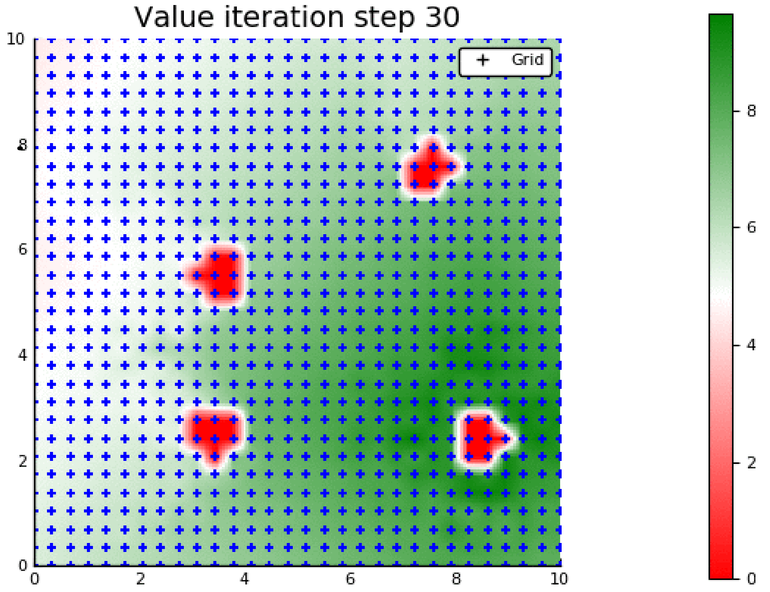
Fitted Value Iteration

while not converged

$\theta \leftarrow \theta'$

$\hat{V}' \leftarrow B_{\text{approx}}[V_\theta]$

$\theta' \leftarrow \text{fit}(\hat{V}')$



$$B_{\text{MC}(N)}[V_\theta](s) = \max_a \left(R(s,a) + \gamma \sum_{i=1}^{N} V_\theta(G(s,a,w_i)) \right)$$

2. Value Function Approximation

$V_\theta(s) = f_\theta(s)$ (e.g. neural network)

$V_\theta(s) = \theta^\top \beta(s)$ (linear feature)

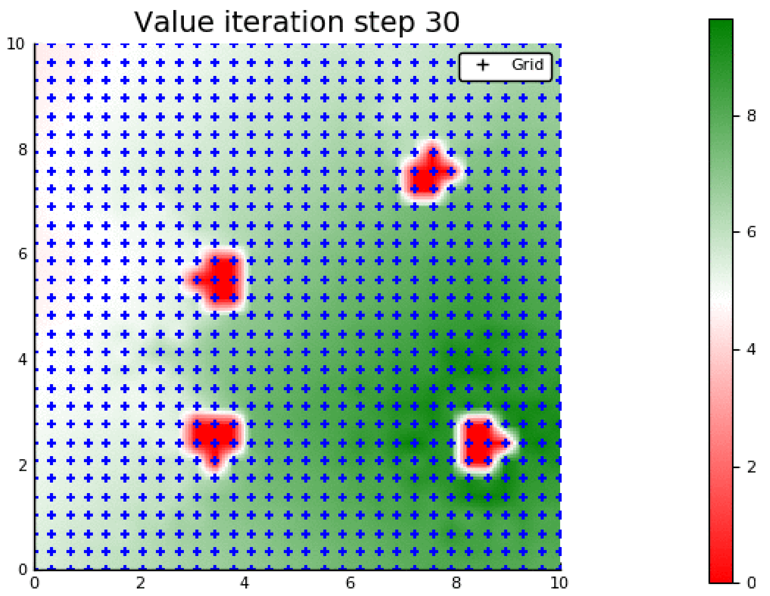
Fitted Value Iteration

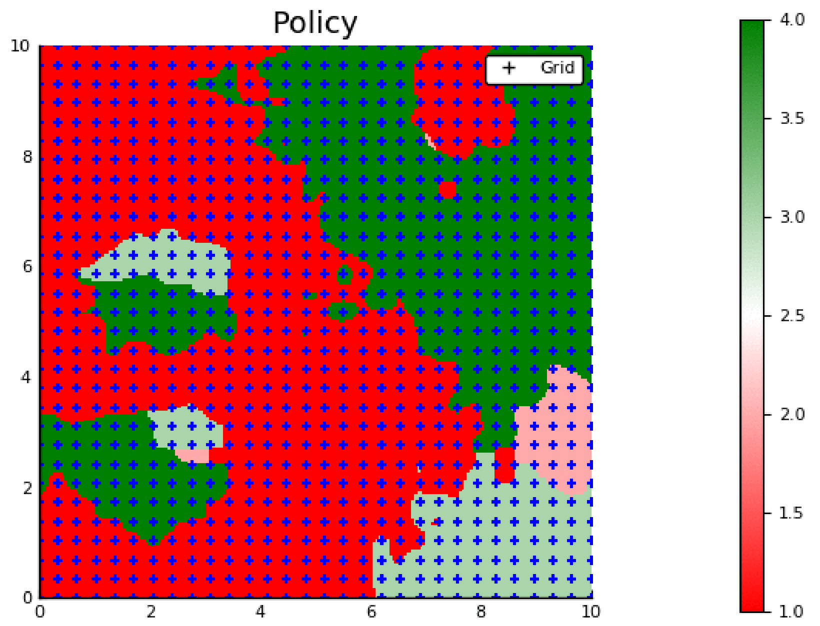
while not converged

$\theta \leftarrow \theta'$

$\hat{V}' \leftarrow B_{\text{approx}}[V_\theta]$

$\theta' \leftarrow \text{fit}(\hat{V}')$





$$B_{\text{MC}(N)}[V_\theta](s) = \max_a \left(R(s,a) + \gamma \sum_{i=1}^{N} V_\theta(G(s,a,w_i)) \right)$$

Function Approximation

Weighting of 2^d points

Weighting of only $d + 1$ points!

Function Approximation

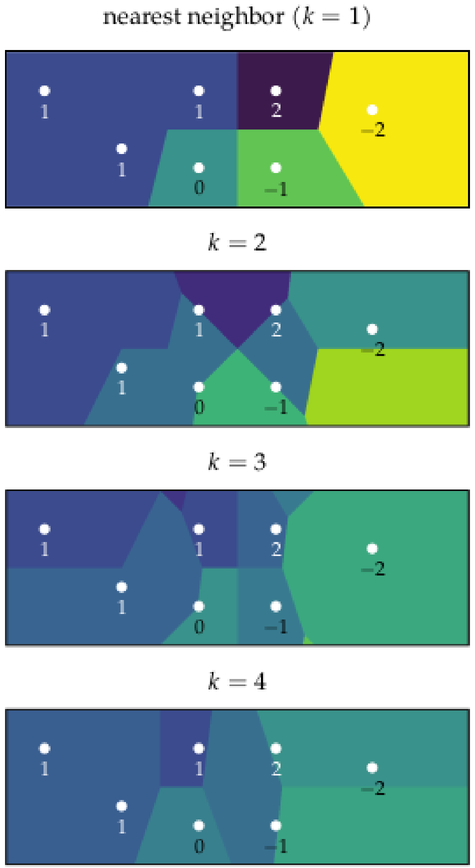
- Global: (e.g. Fourier, neural network)
- Local: (e.g. simplex interpolation)

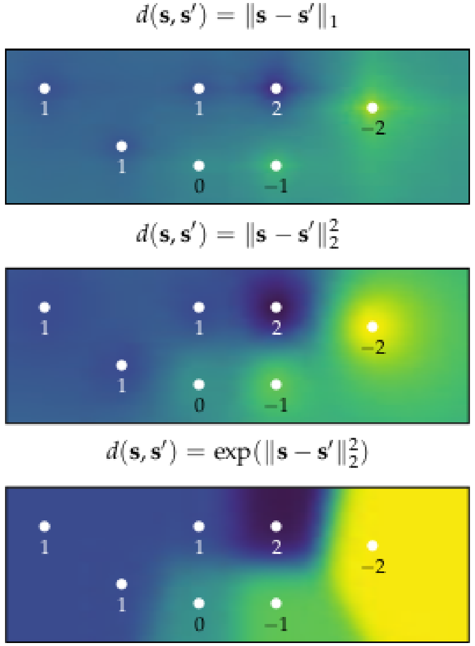
Weighting of 2^d points

Weighting of only $d + 1$ points!

Function Approximation

- Global: (e.g. Fourier, neural network)
- Local: (e.g. simplex interpolation)





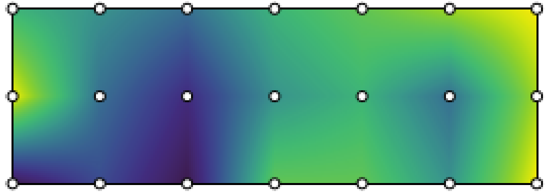


Figure 8.9. Two-dimensional linear interpolation over a 3×7 grid.

Weighting of 2^d points

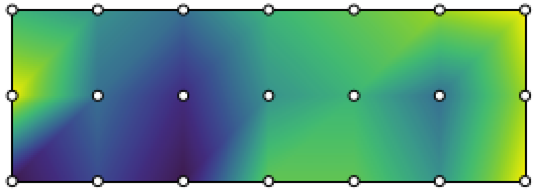
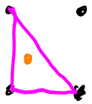


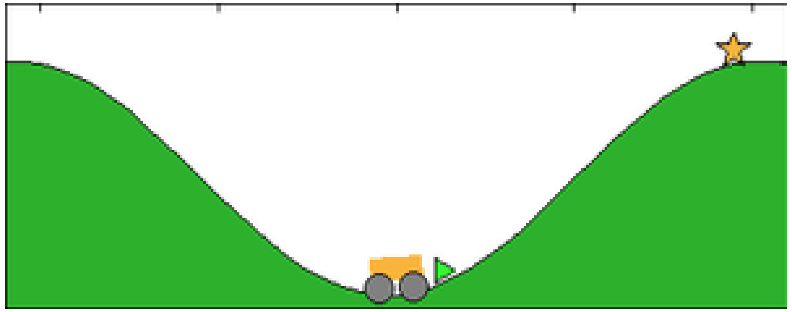
Figure 8.10. Two-dimensional simplex interpolation over a 3×7 grid.



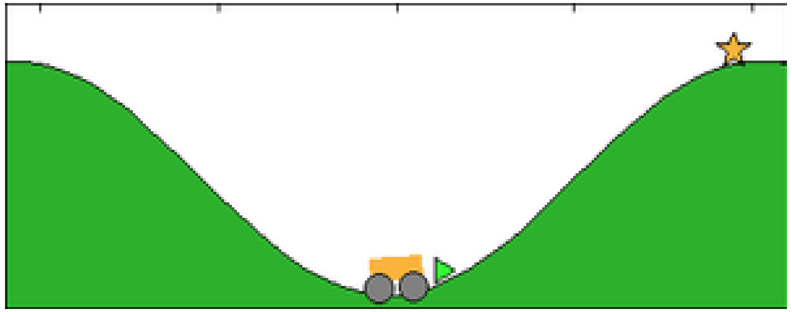
Weighting of only $d + 1$ points!

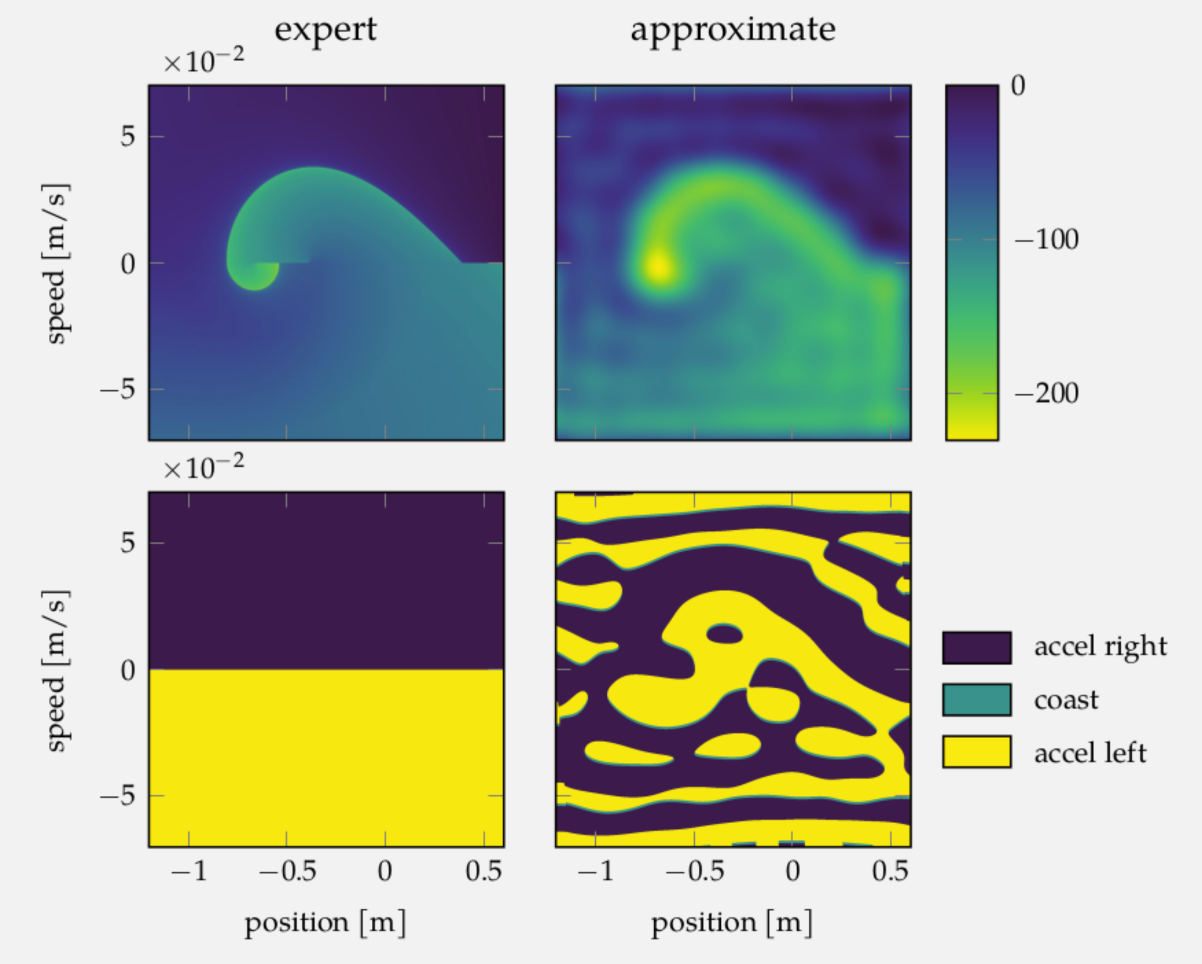
Function Approximation: Mountain Car

Function Approximation: Mountain Car



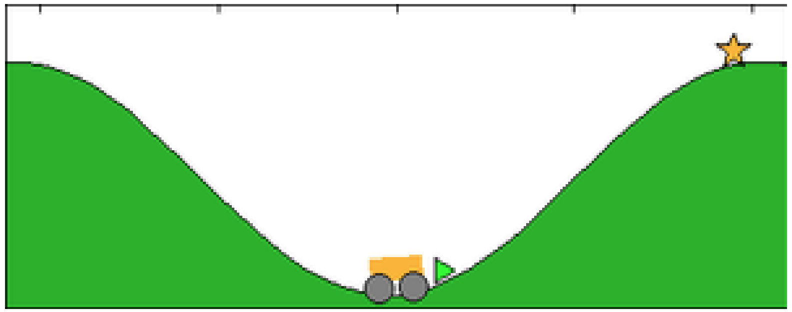
Function Approximation: Mountain Car





(Fourier, 17 params)

Function Approximation: Mountain Car



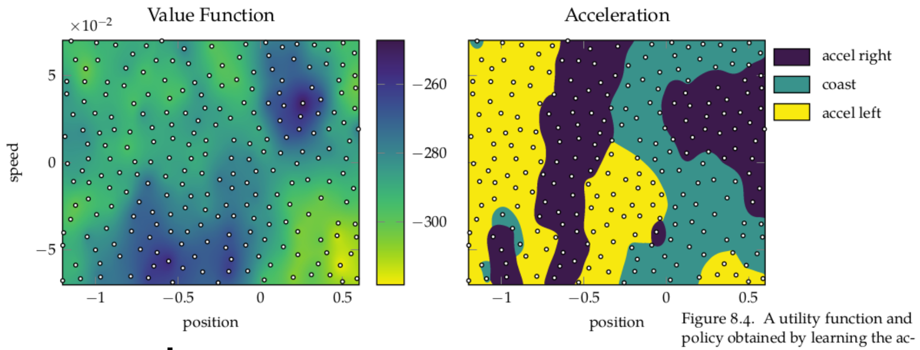
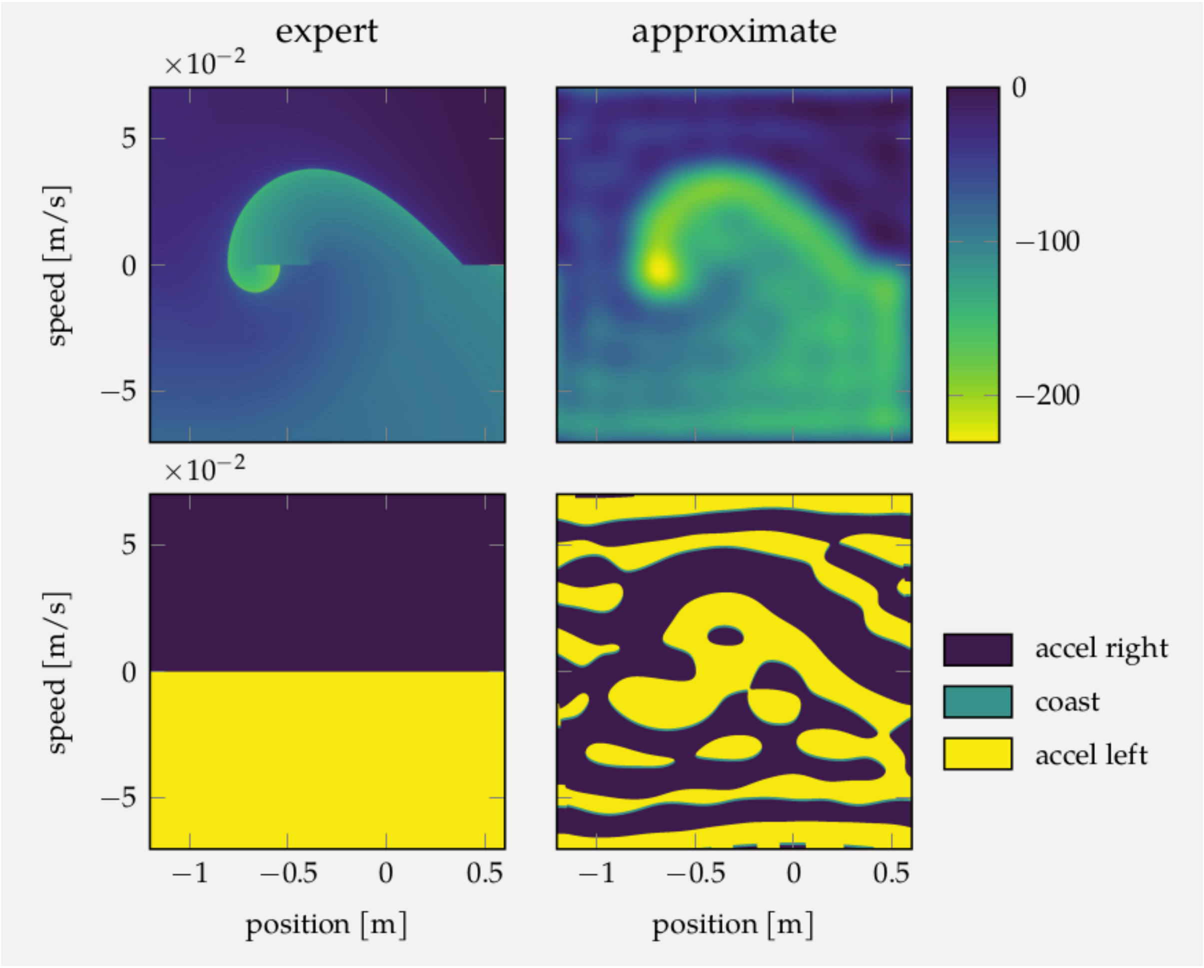


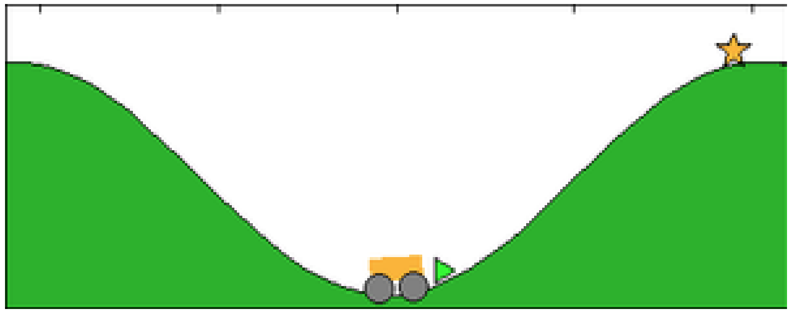
Figure 8.4. A utility function and policy obtained by learning the action values for a finite set of states (white) in the mountain car problem using the distance function $\|\mathbf{s} - \mathbf{s}'\|_2 + 0.1$.

(Kernel, > 100 params)



(Fourier, 17 params)

Function Approximation: Mountain Car



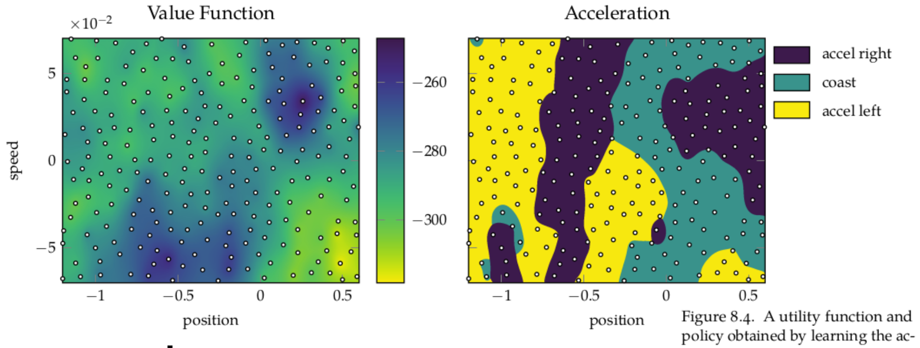
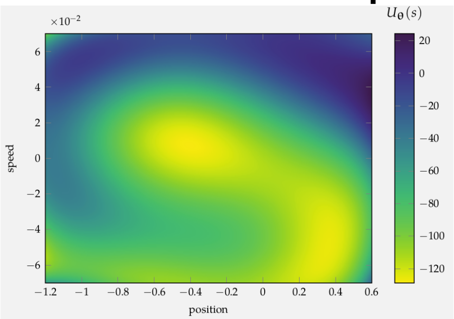
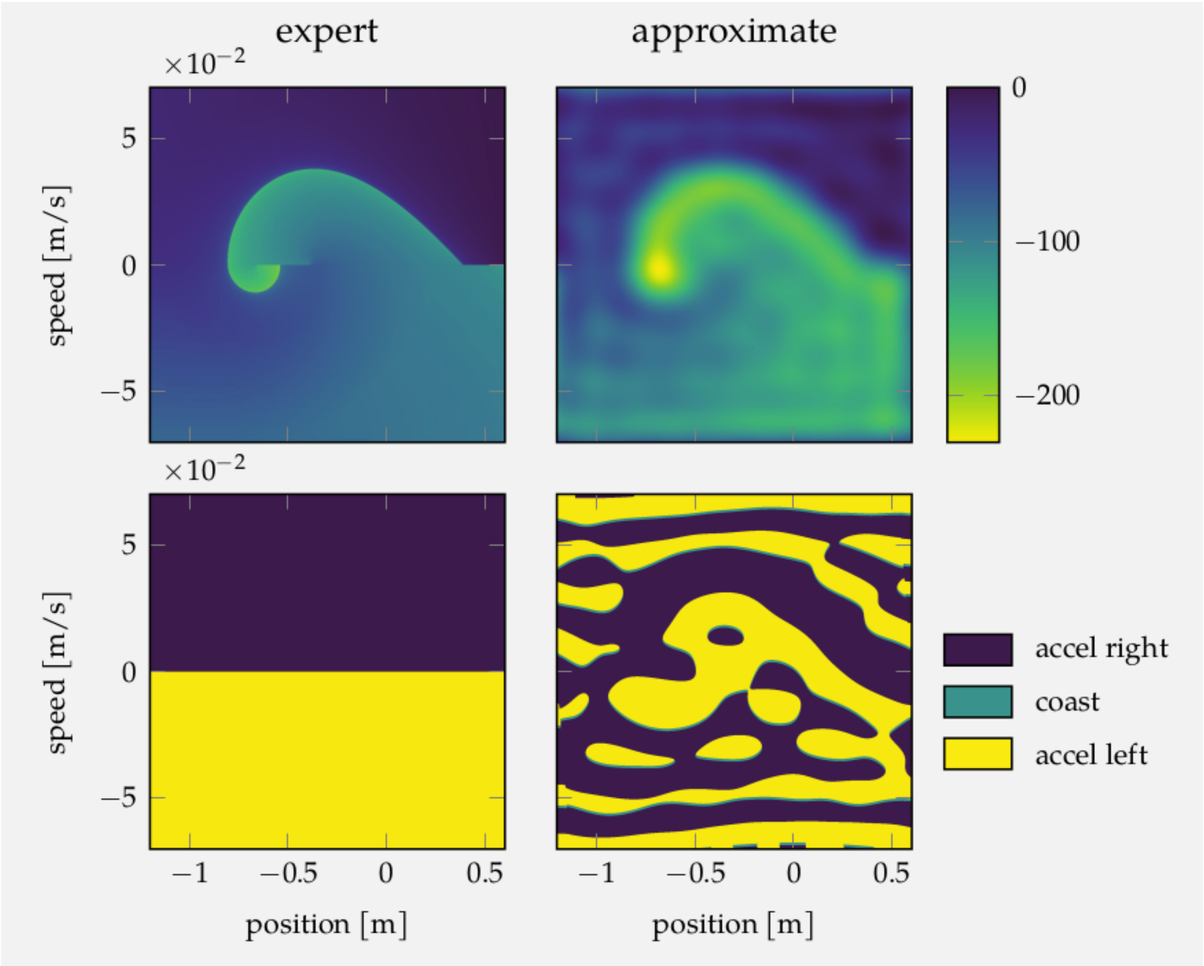


Figure 8.4. A utility function and policy obtained by learning the action values for a finite set of states (white) in the mountain car problem using the distance function $\|\mathbf{s} - \mathbf{s}'\|_2 + 0.1$.

(Kernel, > 100 params)

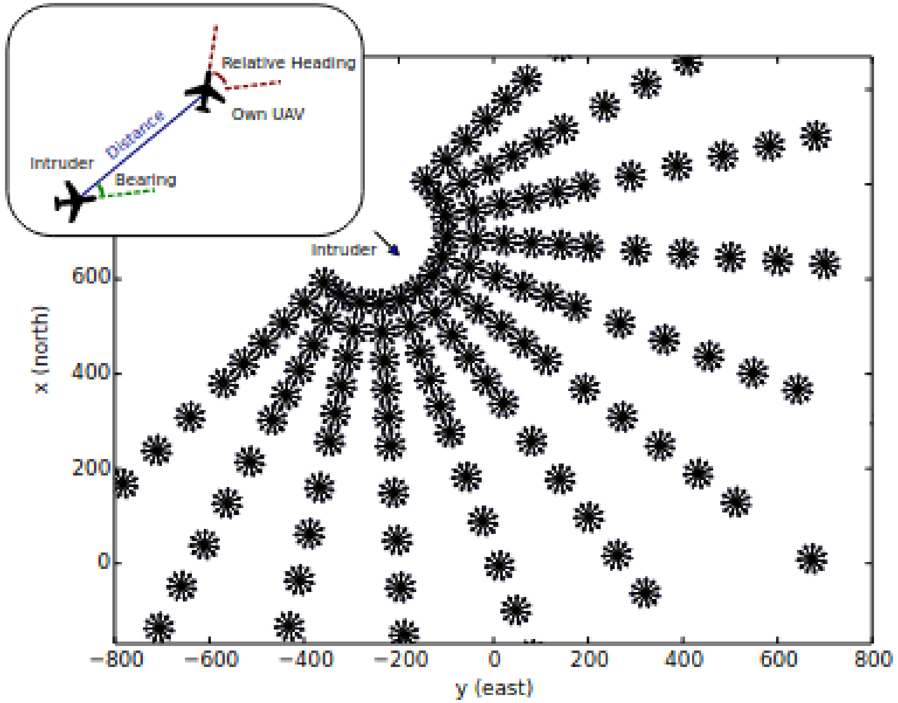


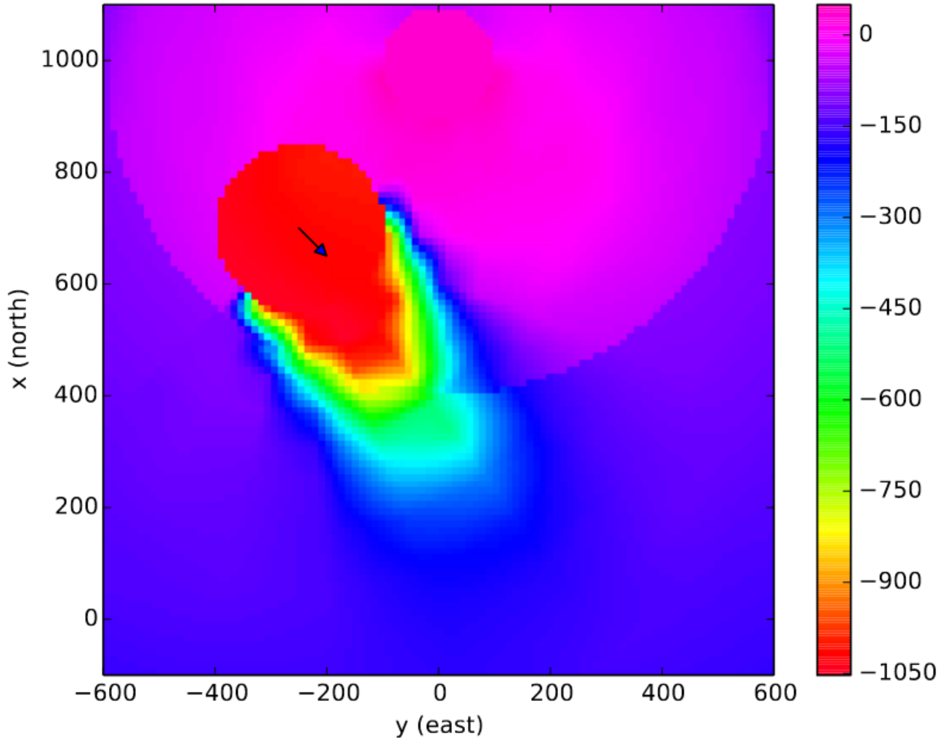
(Polynomial, 28 params)



(Fourier, 17 params)

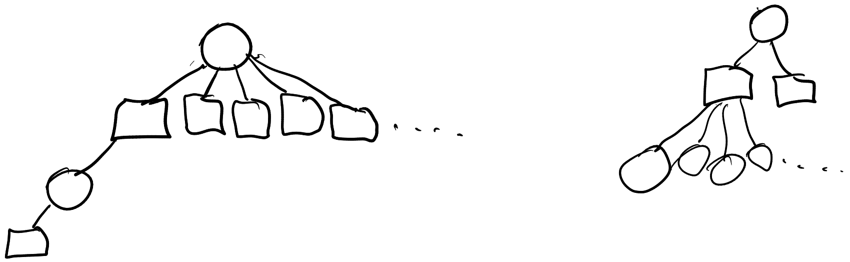
Function Approximation





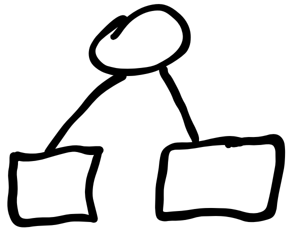
Break

What will a Monte Carlo Tree Search tree look like if run on a problem with continuous spaces?

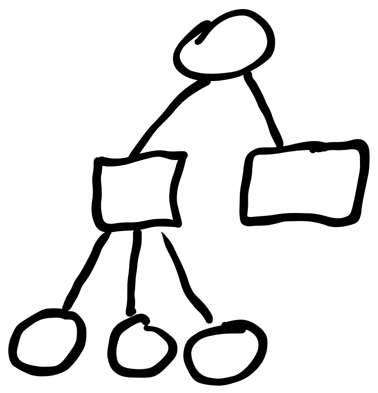


3. Sparse Tree Search/Progressive Widening

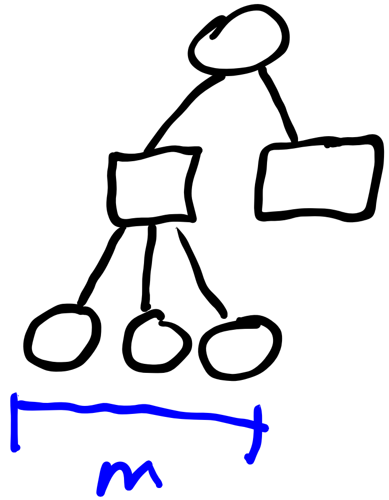
3. Sparse Tree Search/Progressive Widening



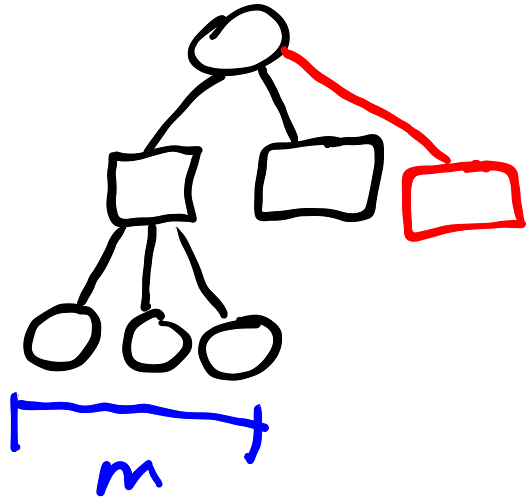
3. Sparse Tree Search/Progressive Widening



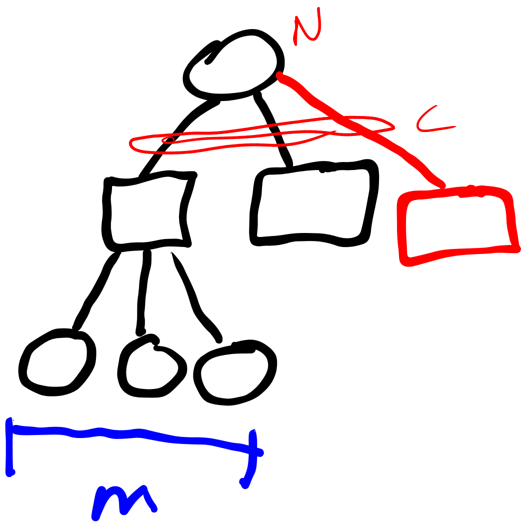
3. Sparse Tree Search/Progressive Widening



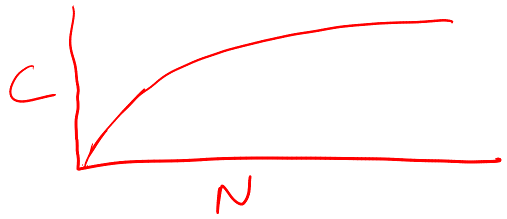
3. Sparse Tree Search/Progressive Widening



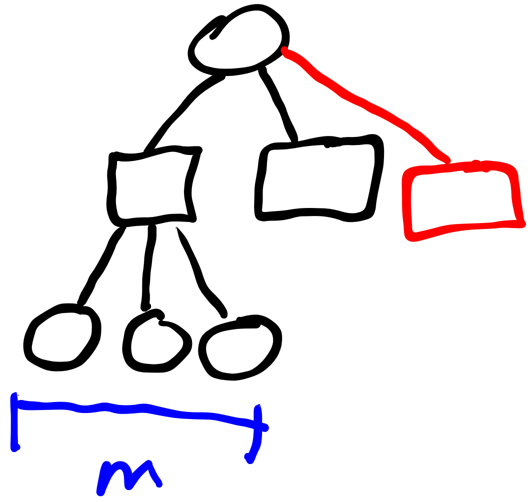
3. Sparse Tree Search/Progressive Widening



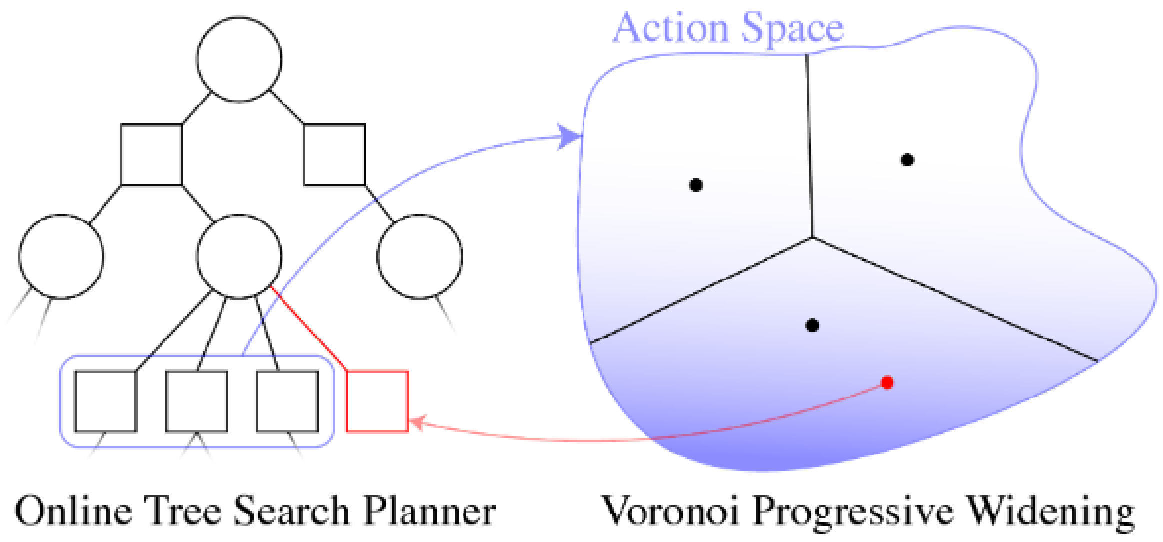
add new branch if $C < kN^{\alpha}$ $\quad (\alpha < 1)$



3. Sparse Tree Search/Progressive Widening



add new branch if $C < kN^\alpha$ $(\alpha < 1)$



4. Model Predictive Control

(Use off-the-shelf optimization software, e.g. Ipopt)

4. Model Predictive Control

(Use off-the-shelf optimization software, e.g. Ipopt)

Certainty-Equivalent

$$
\begin{aligned}
\underset{a_{1:d}, s_{1:d}}{\text{maximize}} \quad & \sum_{t=1}^{d} \gamma^t R(s_t, a_t) \\
\text{subject to} \quad & s_{t+1} = \mathrm{E}[T(s_t, a_t)] \quad \forall t
\end{aligned}
$$

4. Model Predictive Control

(Use off-the-shelf optimization software, e.g. Ipopt)

Certainty-Equivalent

$$\begin{aligned} \underset{a_{1:d}, s_{1:d}}{\text{maximize}} \quad & \sum_{t=1}^{d} \gamma^t R(s_t, a_t) \\ \text{subject to} \quad & s_{t+1} = \mathrm{E}[T(s_t, a_t)] \quad \forall t \end{aligned}$$

Open-Loop

$$\begin{aligned} \underset{a_{1:d}, s_{1:d}^{(1:m)}}{\text{maximize}} \quad & \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{d} \gamma^t R(s_t^{(i)}, a_t) \\ \text{subject to} \quad & s_{t+1} = G(s_t^{(i)}, a_t, w_t^{(i)}) \quad \forall t, i \end{aligned}$$

4. Model Predictive Control

(Use off-the-shelf optimization software, e.g. Ipopt)

Certainty-Equivalent

$$\begin{aligned} \underset{a_{1:d}, s_{1:d}}{\text{maximize}} \quad & \sum_{t=1}^{d} \gamma^t R(s_t, a_t) \\ \text{subject to} \quad & s_{t+1} = \mathrm{E}[T(s_t, a_t)] \quad \forall t \end{aligned}$$

Open-Loop

$$\begin{aligned} \underset{a_{1:d}, s_{1:d}^{(1:m)}}{\text{maximize}} \quad & \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{d} \gamma^t R(s_t^{(i)}, a_t) \\ \text{subject to} \quad & s_{t+1} = G(s_t^{(i)}, a_t, w_t^{(i)}) \quad \forall t, i \end{aligned}$$

Hindsight Optimization

$$\begin{aligned} \underset{a_{1:d}^{(1:m)}, s_{1:d}^{(1:m)}}{\text{maximize}} \quad & \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{d} \gamma^t R(s_t^{(i)}, a_t^{(i)}) \\ \text{subject to} \quad & s_{t+1} = G(s_t^{(i)}, a_t^{(i)}, w_t^{(i)}) \quad \forall t, i \\ & a_1^{(i)} = a_1^{(j)} \quad \forall i, j \end{aligned}$$

Guiding Questions

- What tools do we have to solve MDPs with continuous S and A?