14 Policy Validation

The methods presented in the earlier chapters show how to construct an optimal or approximately optimal solution with respect to a particular model of the dynamics and reward. However, before deploying a decision making system in the real world, it is generally desirable to validate in simulation that the behavior of the resulting policy is consistent with what is actually desired. This chapter discusses various analytical tools for validating decision strategies.[1] We will start by discussing how to go about evaluating different performance metrics. Accurately computing such metrics can be computationally challenging, especially when they pertain to rare events such as failures. We will discuss methods such as importance sampling that can help address computational efficiency. An important concern is that our systems be robust to differences between the models that we use for analysis and the real world. This chapter suggests methods for analyzing the robustness of our policies. Fundamental to the design of many decision making systems is the tradeoff between multiple objectives, and we will outline ways of analyzing these tradeoffs. The chapter concludes with a discussion of adversarial analysis, which can be used for finding the most likely failure trajectory for a given system.

14.1 Performance Metric Evaluation

Once we have a policy, we are often interested in evaluating it with respect to different *performance metrics*. For example, suppose we constructed a collision avoidance system—either through some form of optimization of a scalar reward function or just heuristically as discussed in example 14.1—and we want to assess its safety by computing the probability of collision when following our policy. Or, if we created a policy for constructing investment portfolios, we might be interested

[1] A more extensive discussion is provided by A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, ''A Survey of Algorithms for Black-Box Safety Validation,'' 2020. arXiv: `2005.02979`.

in understanding the probability that our policy will result in an extreme loss or what the expected return may be.

For the moment, we will consider a single metric f evaluated on a policy π. Often, this metric is defined as the expectation of a metric f_{traj} evaluated on trajectories $\tau = (s_1, a_1, \ldots)$:

$$f(\pi) = \mathbb{E}_\tau[f_{\text{traj}}(\tau)] \tag{14.1}$$

This expectation is over the trajectory distribution. In order to define a trajectory distribution associated with an MDP, we need to specify an *initial state distribution* b. We have the probability of generating a trajectory τ as

$$P(\tau) = P(s_1, a_1, \ldots) = b(s_1) \prod_t T(s_{t+1} \mid s_t, a_t) \tag{14.2}$$

In the collision avoidance context, f_{traj} may be 1 if the trajectory led to a collision and 0 otherwise. The expectation would correspond to the collision probability.

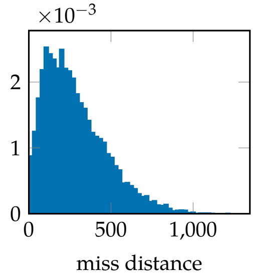


Figure 14.1. Distribution over miss distance estimated from 10^4 simulations when following a simple collision avoidance policy from initial states with:
$h \sim \mathcal{U}(-10, 10)$ (m)
$\dot{h} \sim \mathcal{U}(-200, 200)$ (m/s)
$a_{\text{prev}} = 0\,\text{m/s}$
$t_{\text{col}} = 40\,\text{s}$

In some cases, we are interested in studying the distribution over the output of f_{traj}. Figure 14.1 shows an example of such a distribution. The expectation in equation (14.1) is just one of many different ways to convert a distribution over trajectory metrics to a single value. We will focus primarily on this expectation in our discussion, but examples of other transformations of the distribution to a value include variance, fifth percentile, and mean of the values below the fifth percentile.[2]

The trajectory metric can sometimes be written in this form:

$$f_{\text{traj}}(\tau) = f_{\text{traj}}(s_1, a_1, \ldots) = \sum_t f_{\text{step}}(s_t, a_t) \tag{14.3}$$

where f_{step} is some function that depends on the current state and action, much like the reward function in MDPs. If $f(\pi)$ is defined as the expectation of f_{traj}, the objective is the same as when solving an MDP, where f_{step} is simply the reward function. It turns out that we can use the policy evaluation algorithms introduced in section 7.2 to evaluate our policy with respect to any performance metric of the form in equation (14.3).

Policy evaluation will output a value function that is a function of the state,[3] corresponding to the expected value of the performance metric when starting from that state. Example 14.2 shows slices of this value function for the collision avoidance problem. The overall performance is given by

$$f(\pi) = \sum_b f_{\text{state}}(s) b(s) \tag{14.4}$$

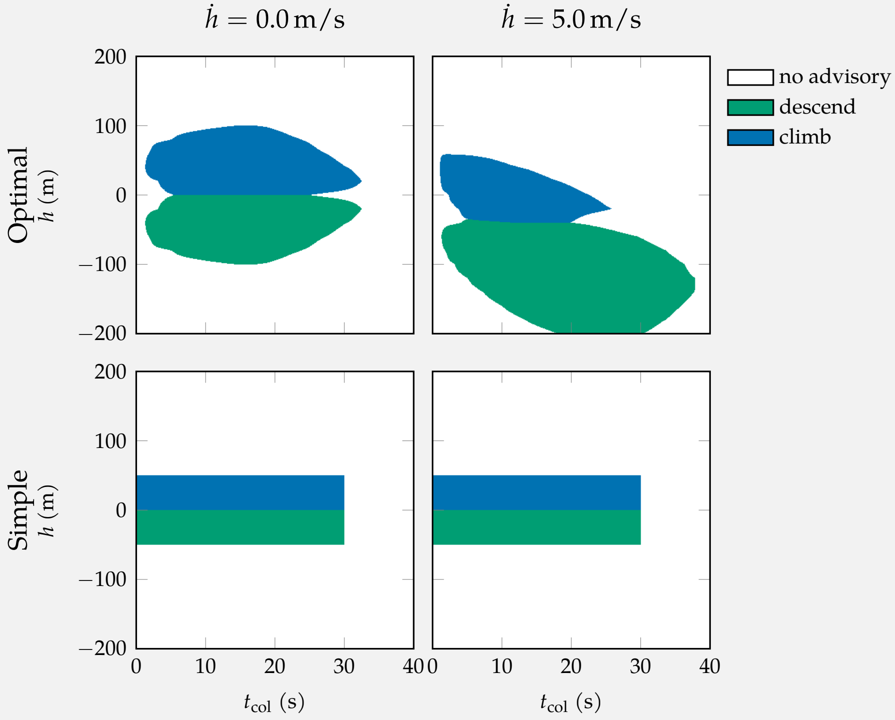
[2] Various risk measures have been discussed in the literature. An overview of some of them that have been used in the context of Markov decision processes are provided by A. Ruszczyński, "Risk-Averse Dynamic Programming for Markov Decision Processes," *Mathematical Programming*, vol. 125, no. 2, pp. 235–261, 2010.

[3] We used U^π to represent the value function associated with policy π in previous chapters.

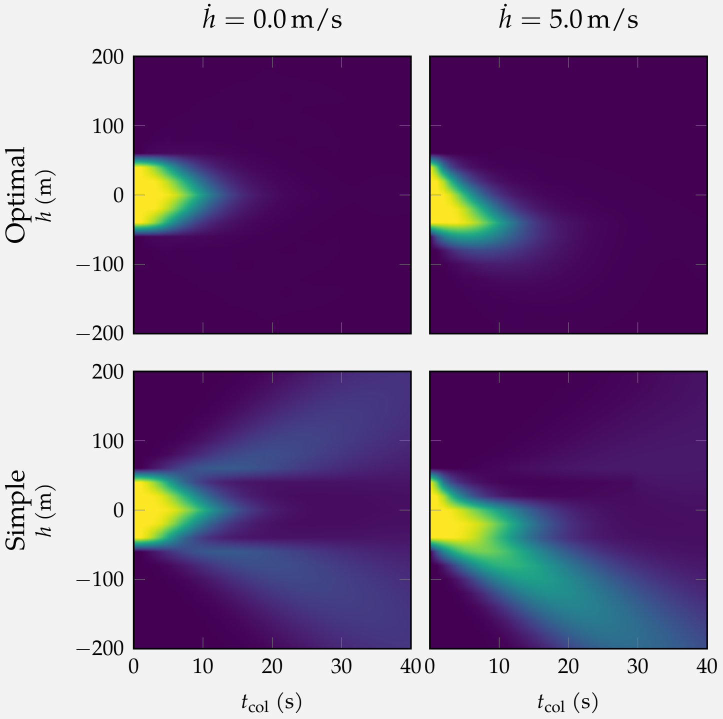
Example 14.1. Optimal and simple collision avoidance policies. Additional details of the problem are in appendix F.6.

In the aircraft collision avoidance problem, we need to decide when to issue a climb or descend advisory to our aircraft to avoid an intruder aircraft. The intruder is approaching us head-on with constant horizontal closing speed. The state is specified by the altitude h of our aircraft measured relative to the intruder aircraft, our vertical rate $\dot{h}$, the previous action a_{prev}, and the time to potential collision t_{col}. There is a penalty of 1 when there is a collision, defined as when the intruder comes within 50 m when $t_{\text{col}} = 0$. Additionally, there is a penalty of 0.01 when $a \neq a_{\text{prev}}$.

We can use dynamic programming with linear interpolation (section 8.4) to derive an optimal policy. Alternatively, we can define a simple heuristic policy parameterized by thresholds on t_{col} and h that works as follows. If $|h| < h_{\text{thresh}}$ and $t_{\text{col}} < t_{\text{thresh}}$ then an advisory is generated. This advisory is a climb if $h > 0$ and a descend otherwise. By default, we use $h_{\text{thresh}} = 50\,\text{m}$ and $t_{\text{thresh}} = 30\,\text{s}$. Below are plots of both the optimal and simple policies for two slices through the state space.



Below is the result of applying policy evaluation to both an optimal policy and the simple policy introduced in example 14.1. Each point in the plot corresponds to the value of the metric conditioned on starting from the associated state. We define $f_{\text{state}}(s, a) = 1$ if s is a collision and 0 otherwise. This plot shows where in the state space there is significant collision risk, indicated by ''hotter'' colors, when following the policy. We can see that the optimal policy is quite safe, especially if $t_{\text{col}} > 20\,\text{s}$. When t_{col} is low, even the optimal policy cannot avoid collision due to the physical acceleration constraints of the vehicle. The simple policy has a much higher level of risk associated with it compared to the optimal policy, especially when $t_{\text{col}} > 20\,\text{s}$, $\dot{h} = 5\,\text{m/s}$, and the intruder is below us—in part because the choice to produce an advisory in the simple strategy does not take $\dot{h}$ into account.



Example 14.2. Probability of collision when following the optimal and simple collision avoidance policies.

where f_{state} is the value function obtained through policy evaluation.

If the state space is discrete, then equation (14.4) can be computed analytically. However, if the state space is large or continuous, we may want to estimate $f(\pi)$ through sampling. We can pull a sample from the initial state distribution and then rollout the policy and compute the trajectory metric. We can then estimate the value of the overall metric from the mean of the trajectory metrics. The quality of the estimate generally improves with more samples. Example 14.3 illustrates this process for estimating different metrics associated with collision avoidance policies.

We often use the *standard error* to measure the quality of our estimate, which is defined as

$$\text{SE} = \hat{\sigma}/\sqrt{n} \tag{14.5}$$

where $\hat{\sigma}$ is the standard deviation of our samples and n is the number of samples. In example 14.3, our the standard deviation of our collision metric is 0.0173, making the standard error of our collision probability metric 0.000173.

We can convert the standard error to a *confidence interval*. For example, a 95 % confidence interval would be $\hat{\mu} \pm 1.96\text{SE}$, where $\hat{\mu}$ is the mean of our samples. For our collision avoidance example, this interval is $(-3.94 \times 10^{-5}, 6.39 \times 10^{-4})$. Alternatively, we can take a Bayesian approach and represent our posterior as a beta distribution as discussed in section 4.2.

For small probabilities, such as failure probabilities in a relatively safe system, we are often interested in the *relative standard error*, which is given by

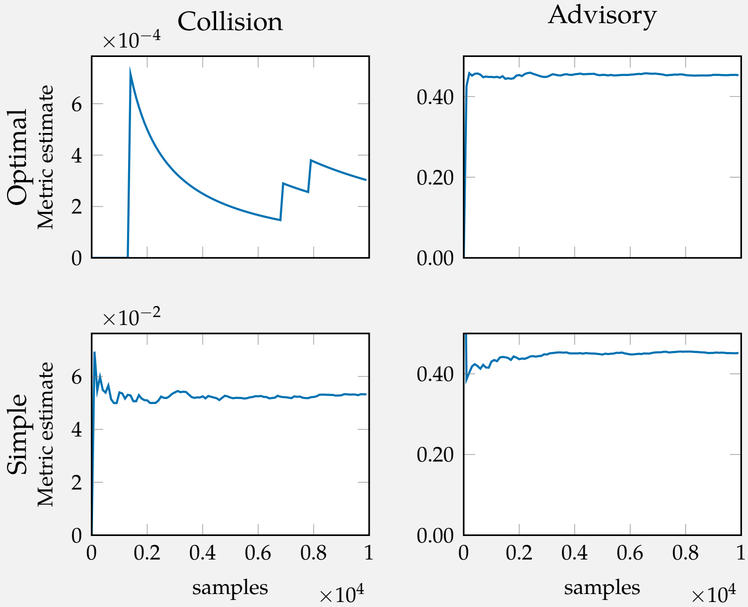
$$\frac{\hat{\sigma}}{\hat{\mu}\sqrt{n}} \tag{14.6}$$

This is equivalent to dividing the standard error by the mean. In our collision avoidance problem, our relative error is 0.578. Although the absolute error might be small, the relative error is quite high since we are trying to estimate a small probability.

14.2 Rare Event Simulation

As we see in example 14.3, we may need many samples to accurately estimate metrics where rare events are very influential, such as estimating collision probability. In the collision avoidance example, there were only three collisions out of our

We want to estimate the probability of collision and the probability of generating an advisory. Here, we will consider the optimal and simple policies introduced in example 14.1. To evaluate these metrics, we use 10^4 samples from the initial state distribution used in figure 14.1 and then perform rollouts. The plots below show the convergence curves.



What we can see is that the optimal policy is much safer than the simple policy while producing advisories at approximately the same frequency. The advisory metric estimate converges much more quickly than the collision estimates. The reason for the faster convergence for the advisory metric is that advisories are more common than collisions. Collisions involving the optimal policy are so rare that even 10^4 samples appear inadequate for an accurate estimate. The curve is very jagged, with large spikes at samples involving collisions, followed by a decay in the collision probability estimate as collision-free samples are simulated.

Example 14.3. Probability of collision and advisory when following the optimal and simple collision avoidance policies.

10^4 samples, as indicated by the three spikes in the plot. When we are designing algorithms for high-stakes systems, such as systems that trade our money or drive our cars, accurately estimating failure probabilities through direct sampling and simulation can be computationally challenging.

A common approach to improve efficiency is called *importance sampling*, which involves sampling from an alternative distribution and weighting the results appropriately to arrive at an unbiased estimate.[4] We used this same kind of approach in the context of inference in Bayesian networks by the name of likelihood weighted sampling (section 3.7). The alternative sampling distribution is often called a *proposal distribution*, and we will use $P'(\tau)$ to represent the probability our proposal distribution assigns to trajectory τ.

We will derive the appropriate way to weight samples from P'. If we have $\tau^{(1)}, \ldots, \tau^{(n)}$ drawn from the true distribution P, then we have

$$
\begin{aligned}
f(\pi) &= \mathbb{E}_h[f_{\text{traj}}(\tau)] && (14.7)\\
&= \sum_\tau f_{\text{traj}}(\tau)P(\tau) && (14.8)\\
&\approx \sum_i f_{\text{traj}}(\tau^{(i)}) \text{ with } \tau^{(i)} \sim P && (14.9)
\end{aligned}
$$

We can multiply equation (14.8) by $P'(\tau)/P'(\tau)$ and obtain the following:

$$
\begin{aligned}
f(\pi) &= \sum_h f_{\text{traj}}(\tau)P(\tau)\frac{P'(\tau)}{P'(\tau)} && (14.10)\\
&= \sum_\tau f_{\text{traj}}(\tau)P'(\tau)\frac{P(\tau)}{P'(\tau)} && (14.11)\\
&\approx \sum_i f_{\text{traj}}(\tau^{(i)})\frac{P(\tau^{(i)})}{P'(\tau^{(i)})} \text{ with } \tau^{(i)} \sim P' && (14.12)
\end{aligned}
$$

In other words, we need to weight the outcomes of the samples from the proposal distribution, where the weight given to sample i is $P(\tau^{(i)})/P'(\tau^{(i)})$.

We want to choose the proposal distribution P' to focus the generation of samples on those that are "important," in the sense that they are more likely to contribute to the overall performance estimate. In the case of collision avoidance, we will want this proposal distribution to encourage collisions so that we have more than just a few collision situations to estimate collision risk. However, we do not want all of our samples to result in collision. In general, assuming that the

[4] A more elaborate introduction to importance sampling and other techniques for rare event simulation is provided by J. A. Bucklew, *Introduction to Rare Event Simulation*. Springer, 2004.

space of histories is discrete, the optimal proposal distribution is

$$P^*(\tau) = \frac{|f_{\text{traj}}(\tau)|P(\tau)}{\sum_{\tau'} |f_{\text{traj}}(\tau')|P(\tau')} \tag{14.13}$$

If f_{traj} is non-negative, then the denominator is exactly the same as the metric we are trying to estimate in equation (14.1).

Although equation (14.13) is generally not practical to compute exactly (this is why we are using importance sampling in the first place), it can provide some intuition as to how to use our domain expertise to construct a proposal distribution. It is common to bias the initial state distribution or the transition model slightly towards more important trajectories, such as towards collision.

To illustrate the construction of an importance distribution, we will use the optimal policy for the collision avoidance problem in example 14.1. Instead of starting at $t_{\text{col}} = 40\,\text{s}$, we will start the aircraft closer with $t_{\text{col}} = 20\,\text{s}$ to make the collision avoidance problem more challenging. The true distribution has $h \sim \mathcal{U}(-10, 10)$ (m) and $\dot{h} \sim \mathcal{U}(-200, 200)$ (m/s). However, certain combinations of h and $\dot{h}$ are more challenging for the optimal policy to resolve. We used dynamic programming on a discrete version of the problem to determine the probability of collision for different values for h and $\dot{h}$. We can take these results and normalize them to turn them into the proposal distribution shown in figure 14.2.

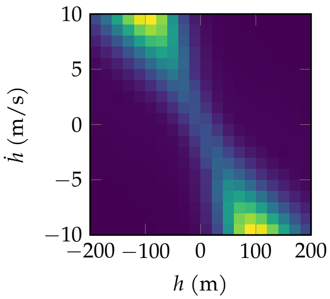


Figure 14.2. Proposal distribution generated from the probability of collision when following the optimal collision avoidance policies from different initial states with $t_{\text{col}} = 20\,\text{s}$ and $a_{\text{prev}} = 0\,\text{m/s}$.

Using the proposal distribution shown in figure 14.2 results in better estimates of the collision probability than direct sampling with the same number of samples. Figure 14.3 shows the convergence curves. By 5×10^4 samples, both sampling methods converge to the same estimate. However, importance sampling converges closely to the true value within 10^4 samples. Using our proposal distribution, importance sampling generated 939 collisions, while direct sampling only generated 246. Even more collisions could be generated if we also biased the transition distribution, rather than solely the initial state distribution.

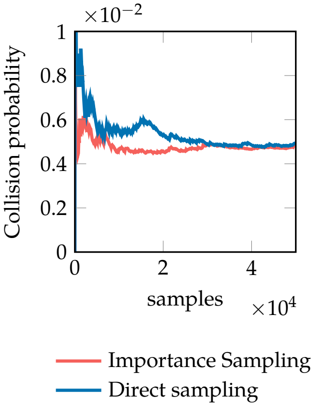


Figure 14.3. Collision probability when following the optimal policy as estimated by importance sampling and direct sampling.

14.3 *Robustness Analysis*

Before deploying a system in the real world, it is important to study its robustness to modeling errors. We can use the tools mentioned in the previous sections, such as policy evaluation and importance sampling, but evaluate our policies on environments that deviate from the model assumed when optimizing the policy. Figure 14.4 shows how performance varies as the true model deviates from the one used for optimization. We can also study the sensitivity of our metrics to modeling assumptions over the state space (example 14.4). If performance on the relevant metrics appears to be preserved under plausible perturbations of the environment model, then we can have greater confidence that our system will behave as planned when deployed.

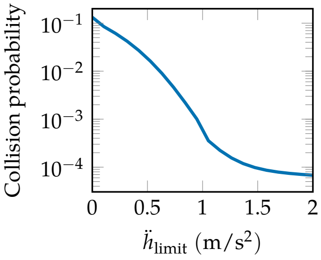


Figure 14.4. Analysis of robustness of a policy optimized for $\ddot{h}_{\text{limit}} = 1\,\text{m/s}^2$ but evaluated in environments with different values for $\ddot{h}_{\text{limit}}$.

We typically want our *planning model*, the model we use for optimizing our policies, to be relatively simple to prevent overfitting to potentially erroneous modeling assumptions that are not representative of the real world. A side benefit of simpler planning models is that they can make planning more computationally efficient. However, our *evaluation model* can be as complex as we can justify. For example, we may use a simple, low-dimensional, discrete model of aircraft dynamics when generating a collision avoidance policy, but then evaluate that policy in a continuous, high fidelity simulation. A simpler planning model is often more robust to perturbations in the evaluation model.

The process of evaluating our policies on a variety of evaluation models is sometimes referred to as *stress testing*, especially if the spectrum of evaluation models includes fairly extreme scenarios. In collision avoidance, extreme scenarios might include those where the aircraft are converging upon each other with extreme climb rates that may not be physically achievable. Understanding what categories of scenarios can lead to system failure can be useful during the design phase, even if we choose not to optimize the behavior of the system for these scenarios because they are deemed unrealistic.

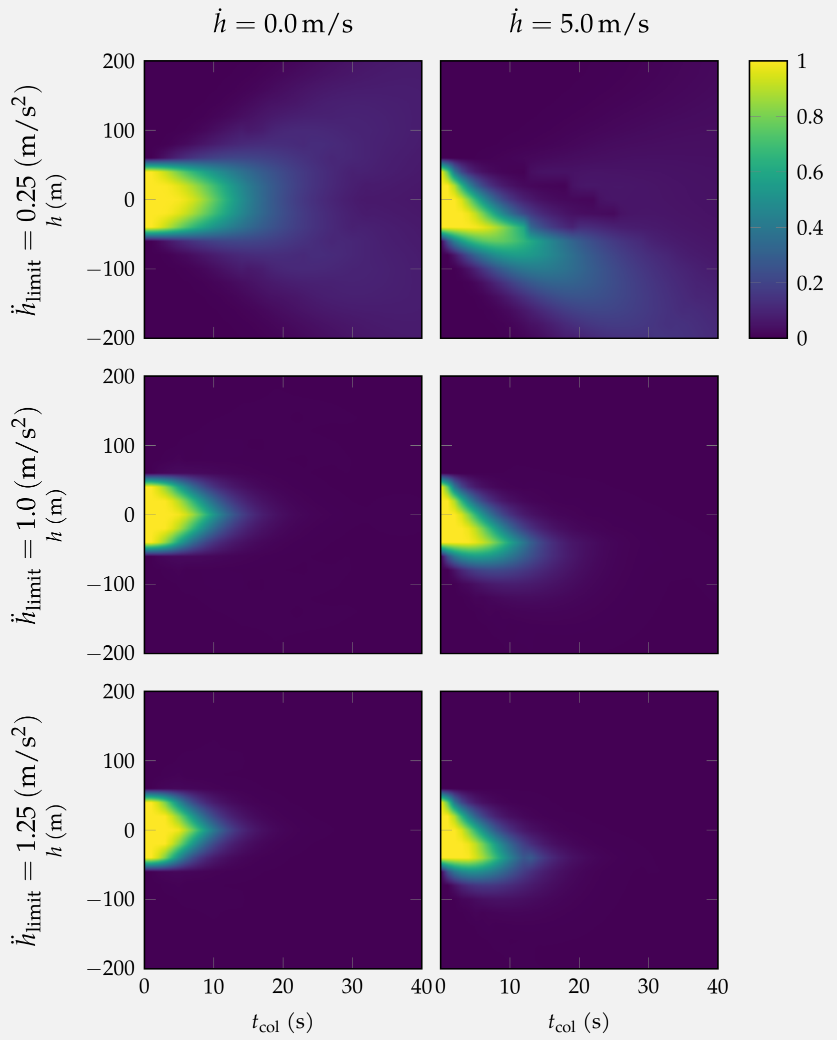
If we find that our policies are overly sensitive to our modeling assumptions, we may consider using a method known as *robust dynamic programming*.[5] Instead of committing to a particular transition model, we have a suite of transition models $T_{1:n}$ and reward models $R_{1:n}$. We can revise the Bellman update equation from equation (7.16) to provide robustness to different models:

$$U_{k+1}(s) = \max_a \min_i \left(R_i(s,a) + \gamma \sum_{s'} T_i(s' \mid s,a) U_k(s') \right) \tag{14.14}$$

The update uses the action that maximizes expected utility when using the model that minimizes our utility.

[5] G. N. Iyengar, ''Robust Dynamic Programming,'' *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005. This approach can improve robustness in the context of collision avoidance. M. J. Kochenderfer, J. P. Chryssanthacopoulos, and P. Radecki, ''Robustness of Optimized Collision Avoidance Logic to Modeling Errors,'' in *Digital Avionics Systems Conference (DASC)*, 2010.

We can plot collision probability when starting from different initial states, similar to example 14.2. Below, we use a policy optimized for the parameters in appendix F.6, but we vary the limit $\ddot{h}_{\text{limit}}$ in the evaluation model.



We optimized the policy with $\ddot{h}_{\text{limit}} = 1\,\text{m/s}^2$. If it was actually $0.25\,\text{m/s}^2$, then the policy performs poorly in some states since it takes longer to achieve a target vertical rate. If the limit was $1.25\,\text{m/s}^2$, we are a bit safer.

Example 14.4. Probability of collision when following the optimal collision avoidance policies when there is a mismatch between the model used for planning and the model used for evaluation.

14.4 *Trade Analysis*

Many interesting tasks involve multiple, often competing, objectives. For autonomous systems, there is often a tradeoff between safety and efficiency. In designing a collision avoidance system, we want to be very safe without making too many unnecessary avoidance maneuvers. A *trade analysis* studies how the various performance metrics are traded as the design parameters are changed.

If we consider only two performance metrics, we can plot a *tradeoff curve* like the one discussed in example 14.5. By varying parameters in the policy, we obtain different values for the two metrics. These curves are useful when comparing different methodologies for generating policies. For example, the curves in example 14.5 suggests that a dynamic programming approach to generating policies can bring significant benefit over simple threshold-based policies—at least in the way we defined them.

For each of the curves in example 14.5, we vary only one parameter at a time, but to arrive at a satisfactory system, we may need to study the effects of varying multiple parameters. As we vary multiple parameters, we obtain a space of possible policies. Some of those policies may perform worse on all performance metrics relative to at least one other policy in that space. We can often eliminate from consideration those policies that are *dominated* by others. A policy is called *Pareto optimal*[6] or *Pareto efficient* if it is not dominated by any other policy in that space. The set of Pareto optimal policies is called the *Pareto frontier* or (in two dimensions) the *Pareto curve*. Figure 14.5 shows an example Pareto curve.

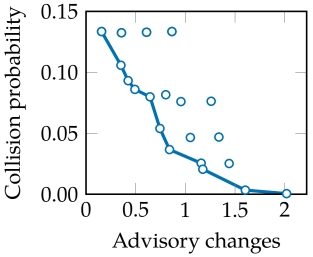


Figure 14.5. Performance of policies generated by varying the parameters of the simple policy from example 14.1. The approximate Pareto curve is highlighted in blue.

14.5 *Adversarial Analysis*

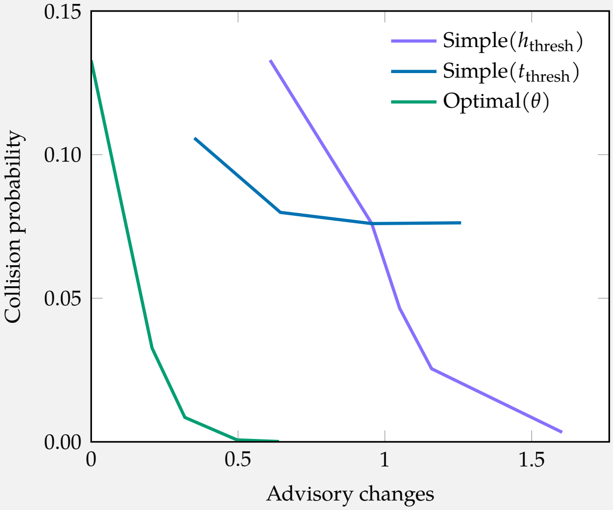
It can be useful to study the robustness of a policy from the perspective of an *adversarial analysis*. At each time step, an *adversary* selects the state that results from applying the action specified by the policy from the current state. The adversary has two objectives to balance: minimizing our return and maximizing the likelihood of the resulting trajectory according to our transition model. We can transform our original problem into an adversarial problem. The adversarial state space is the same as in the original problem, but the adversarial action space is the state space of the original problem. The adversarial reward is

$$R'(s,a) = -R(s,\pi(s)) + \lambda \log(T(a \mid s,\pi(s))) \tag{14.15}$$

[6] Named after the Italian economist Vilfredo Federico Damaso Pareto (1848–1923).

In our aircraft collision avoidance problem, we must balance safety in terms of collision probability with other metrics, such as the expected number of advisory changes. Both of these can be implemented using trajectory metrics that are additively decomposed by steps as done in equation (14.3), allowing us to compute them using exact policy evaluation.

The plot below shows three curves associated with different parameterized versions of the simple and optimal policies. The first curve shows the performance of the simple policy on the two metrics as the h_{thresh} parameter (defined in example 14.1) is varied. The second curve shows the performance of the simple policy as t_{thresh} is varied. The third curve shows the optimal policy as the parameter θ is varied, where the cost of collision is $-\theta$ and the cost of changing advisories is $-(1-\theta)$.



We can see that the optimal policy dominates the curves generated by the parameterized simple policies. When θ is close to 1, then we are very safe, but we have to tolerate more advisory changes. As θ goes to 0, we are less safe but do not produce advisories. Given a particular threshold level of safety, we are able to create an optimized policy that has fewer advisory changes in expectation than either of the simple parametric policies.

Example 14.5. An analysis of the tradeoff between safety and operational efficiency when varying parameters of different collision avoidance systems.

where π is our policy, R is our original reward function, T is our original transition model, and $\lambda \geq 0$ is a parameter that controls the importance of maximizing the resulting likelihood of the trajectory. Since an adversary attempts to maximize the sum of adversarial reward, it is maximizing our expected negative return plus λ times the log-probability of the resulting trajectory.[7] The adversarial transition model is deterministic; the state transitions to exactly what the adversary specifies as its action.

Algorithm 14.1 implements this conversion to an adversarial problem. It assumes a discrete state and action space, which can then be solved using one of the dynamic programming algorithms in chapter 7. The solution is an adversarial policy that maps states to states. Given an initial state, we can generate a trajectory that minimizes our reward given some level of probability. Since the problem is deterministic, it is actually a search problem, and any of the algorithms in appendix E can be used. If our problem is high-dimensional or continuous, we may use one of the approximate solution techniques discussed in chapters 8 and 9.

```julia
function adversarial(𝒫::MDP, π, λ)
    𝒮, 𝒜, T, R, γ = 𝒫.𝒮, 𝒫.𝒜, 𝒫.T, 𝒫.R, 𝒫.γ
    𝒮′ = 𝒜′ = 𝒮
    R′ = zeros(length(𝒮′), length(𝒜′))
    T′ = zeros(length(𝒮′), length(𝒜′), length(𝒮′))
    for s in 𝒮′
        for a in 𝒜′
            R′[s,a] = -R(s, π(s)) + λ*log(T(s, π(s), a))
            T′[s,a,a] = 1
        end
    end
    return MDP(T′, R′, γ)
end
```

Algorithm 14.1. Conversion to an adversarial problem given a policy π. An adversarial agent tries to change the outcomes of our policy actions so as to optimize the dual objectives of minimizing our original utility and maximizing the likelihood of the trajectory. The parameter λ controls how important it is to maximize the likelihood of the resulting trajectory. It returns an MDP whose transition and reward models are represented as matrices.

Sometimes we are interested in finding the *most likely failure* associated with a policy for some definition of failure. In some problems, failure can be defined as entering a particular state. For example, a collision may be considered a failure in our collision avoidance problem. Other problems may require a more complicated definition of failure that goes beyond just entering some subset of the state space. For example, we may want to specify failure using a *temporal logic*, which is a way to represent and reason about propositions qualified in terms of time. In many cases, however, we can use these failure specifications to create an augmented state space that we can then solve.[8]

[7] The log-probability of a trajectory is equal to the sum of the log of the individual state transition probabilities.

[8] M. Bouton, J. Tumova, and M. J. Kochenderfer, "Point-Based Methods for Model Checking in Partially Observable Markov Decision Processes," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

With a definition of failure states, we can solve for the most likely failure trajectory by changing the reward function in equation (14.15) to:

$$R'(s,a) = \begin{cases} -\infty & \text{if } s \text{ is terminal and not a failure} \\ 0 & \text{if } s \text{ is terminal and a failure} \\ \log(T(a \mid s, \pi(s))) & \text{otherwise} \end{cases} \tag{14.16}$$

We can find these most likely failures using a variety of approximation methods. Depending on the approximation method, it may be important to relax the infinite penalty for not reaching a failure at termination so that the search can be guided to failures. If applying Monte Carlo tree search to collision avoidance, the penalty could be related to the miss distance.[9]

We can play back the most likely failure trajectory and gauge whether that trajectory merits concern. If the trajectory is deemed extremely implausible, then we can feel more confident that our policy is safe. If the failure trajectory does merit concern, then we might have a few options:

1. *Change the action space.* We may add more extreme maneuvers to our action set for our collision avoidance problem.

2. *Change the reward function.* We may decrease the cost for changing advisories with the aim of lowering collision risk, as illustrated in the tradeoff curve in example 14.5.

3. *Change the transition function.* We may increase the acceleration limit so that the aircraft can achieve the target vertical rates more quickly when directed by our policy.

4. *Improve the solver.* We may have used a discretization of the state space that is too coarse to capture important features of the optimal policy. In exchange for additional computation time, we may be able to refine the discretization to obtain a better policy. Alternatively, we may adopt a different approximation technique.

5. *Do not deploy the system.* If the policy is unsafe, it may be better to not deploy it in the real world.

[9] This strategy was used by R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, ''Adaptive Stress Testing of Airborne Collision Avoidance Systems,'' in *Digital Avionics Systems Conference (DASC)*, 2015.

14.6 Summary

- Performance metrics for policies may be evaluated using the dynamic programming techniques discussed in earlier chapters or through sampling rollouts.
- We can assess our confidence in our performance metric evaluations using standard error, confidence intervals, or one of the Bayesian approaches discussed earlier.
- Estimating the probability of rare events can be done more efficiently using a method called importance sampling.
- Importance sampling involves sampling from an alternative distribution and weighting the results appropriately.
- Due to the fact that the model used for optimization may not be an accurate representation of the real world, it is important to study the sensitivity of our policy to modeling assumptions.
- Robust dynamic programming can help improve robustness to model uncertainty by optimizing with respect to a set of different transition and reward models.
- Trade analyses can help us determine how to balance multiple performance objectives when optimizing a policy.
- Adversarial analyses involve an adversary that chooses the state to which we transition at each step so as to minimize our objective while maximizing the likelihood of the trajectory.

14.7 Exercises

Exercise 14.1. We have a trajectory τ with

s_1	a_1	s_2	a_2	s_3
6.0	2.2	1.4	0.7	6.0

Our dynamics are linear–Gaussian with $T(s' \mid s, a) = \mathcal{N}(s' \mid 2s + a, 5^2)$, and our initial state distribution is given by $\mathcal{N}(5, 6^2)$. What is the log-likelihood of the trajectory τ?

Solution: The log-likelihood of the trajectory is

$$\log \mathcal{N}(6.0 \mid 5, 6^2) + \log \mathcal{N}(1.4 \mid 2 \cdot 6.0 + 2.2, 5^2) + \log \mathcal{N}(6.0 \mid 2 \cdot 1.4 + 0.7, 5^2) \approx -11.183$$

Exercise 14.2. We ran a million simulations and found that our collision avoidance system resulted in 10 collisions. What is our collision probability estimate and the relative standard error?

Solution: The collision probability estimate is

$$\hat{\mu} = 10/10^6 = 10^{-5}$$

The ith sample x_i is 1 if there is a collision and 0 otherwise. The standard deviation is

$$\hat{\sigma} = \sqrt{\frac{1}{10^6 - 1}\sum_{i=1}^{n}(x_i - \hat{\mu})^2} = \sqrt{\frac{1}{10^6 - 1}\left(10(1-\hat{\mu})^2 + (10^6 - 10)\hat{\mu}^2\right)} \approx 0.00316$$

The relative error is

$$\frac{\hat{\sigma}}{\hat{\mu}\sqrt{n}} \approx \frac{0.00316}{10^{-5}\sqrt{10^6}} = 0.316$$

Exercise 14.3. We want to compute the expectation $\mathbb{E}_{x\sim\mathcal{U}(0,5)}[f(x)]$, where $f(x)$ is -1 if $|x| \leq 1$ and 0 otherwise. What is the optimal proposal distribution?

Solution: The optimal proposal distribution is

$$p^*(x) = \frac{|f(x)|p(x)}{\int |f(x)|p(x)\,\mathrm{d}x}$$

which is equivalent to $\mathcal{U}(0,1)$ because $f(x)$ is only non-zero for $x \in [-1,1]$, $\mathcal{U}(0,5)$ only has support for $x \in [0,5]$, and both $f(x)$ and $p(x)$ produce constant values when non-zero.

Exercise 14.4. Suppose we draw the sample 0.3 from the proposal distribution in the previous exercise. What is its weight? What is the estimate of $\mathbb{E}_{x\sim\mathcal{U}(0,5)}[f(x)]$?

Solution: The weight is $p(x)/p^*(x) = 0.2/1$. Since $f(0.3) = -1$, the estimate is -0.2, which is the exact answer.

Exercise 14.5. Suppose we have the following four policies that have been evaluated on three metrics that we want to maximize:

System	f_1	f_2	f_3
π_1	2.7	1.1	2.8
π_2	1.8	2.8	4.5
π_3	9.0	4.5	2.3
π_4	5.3	6.0	2.8

Which policies are on the Pareto frontier?

Solution: Only π_1 is dominated by other policies. Hence, π_2, π_3, and π_4 are on the Pareto frontier.