24 *Multiagent Reasoning*

So far, we have focused on making rational decisions for a single agent. These models have natural extensions to multiple agents. New challenges emerge as agents interact; agents can aid each other or act in their own best interests. Multiagent reasoning is a subject of *game theory*.[1] This chapter builds on the concepts introduced earlier, extending them to multiagent contexts. We will discuss the foundational game theoretic approaches to compute decision strategies and multiagent equilibria.

[1] Game theory is a broad field. Several standard introductory books include D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991. R. B. Myerson, *Game Theory: Analysis of Conflict*. Harvard University Press, 1997. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

24.1 *Simple Games*

A *simple game* (algorithm 24.1) is a fundamental model for multiagent reasoning.[2] Each agent $i \in \mathcal{I}$ selects an action a^i to maximize their own accumulation of reward r^i. The *joint action space* $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^k$ consists of all possible combinations of the actions $\mathcal{A}^i$ available to each agent. The actions selected simultaneously across agents can be combined to form a *joint action* $\mathbf{a} = (a^1, \ldots, a^k)$ from this joint action space.[3] The *joint reward function* $\mathbf{R}(\mathbf{a}) = (R^1(\mathbf{a}), \ldots, R^k(\mathbf{a}))$ represents the reward produced by the joint action $\mathbf{a}$. The *joint reward* is written $\mathbf{r} = (r^1, \ldots, r^k)$. Simple games do not include states or transition functions. Example 24.1 introduces a simple game.

[2] Simple games encompass *normal form games* (also called *standard form games* or *matrix games*), finite-horizon *repeated games*, and infinite-horizon discounted repeated games. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[3] A joint action is also called an *action profile*.

```
struct SimpleGame
    γ  # discount factor
    ℐ  # agents
    𝒜  # joint action space
    R  # joint reward function
end
```

Algorithm 24.1. Data structure for a simple game.

The *prisoner's dilemma* is a two-agent, two-action game involving two prisoners that are on trial. They can choose to *cooperate* and remain silent about their shared crime, or *defect* and blame the other for their crime. If they both cooperate, they both serve a sentence of one year. If agent i cooperates and the other agent defects, then i serves four years and the other serves no time. If both defect, then they both serve three years.

Two-agent simple games can be represented by a table. Rows represent actions for agent 1. Columns represent actions for agent 2. The rewards for agent 1 and 2 are shown in each cell.

Example 24.1. A simple game known as the prisoner's dilemma. Additional detail is provided in appendix F.10.

		agent 2	
		cooperate	defect
agent 1	cooperate	$-1, -1$	$-4, 0$
	defect	$0, -4$	$-3, -3$

A *joint policy* $\boldsymbol{\pi}$ specifies a probability distribution over joint actions taken by the agents. Joint policies can be decomposed into individual agent policies. The probability that agent i selects action a is given by $\pi^i(a)$. In game theory, a deterministic policy is called a *pure strategy* and a stochastic policy is called a *mixed strategy*. The utility of a joint policy $\boldsymbol{\pi}$ from the perspective of agent i is

$$U^i(\boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathcal{A}} R^i(\mathbf{a}) \prod_{j \in \mathcal{I}} \pi^j(a^j) \tag{24.1}$$

Algorithm 24.2 implements routines for representing policies and computing their utility.

A *zero-sum game* is a type of simple game where the sum of rewards across agents is zero. Here, any gain of an agent results as a loss to the other agents. A zero-sum game with two agents $\mathcal{I} = \{1, 2\}$ has opposing reward functions $R^1(\mathbf{a}) = -R^2(\mathbf{a})$. They are typically solved with algorithms specialized for this reward structure. Example 24.2 describes such a game.

24.2 *Response Models*

Before exploring different concepts for solving for a joint policy, we will begin by discussing how to model the *response* of a single agent i, given fixed policies for the other agents. We will use the notation $-i$ as shorthand for $(1, \ldots, i-1, i+1, \ldots, k)$. Using this notation, a joint action is written as $\mathbf{a} = (a^i, \mathbf{a}^{-i})$, a joint reward is written as $\mathbf{R}(a^i, \mathbf{a}^{-i})$, and a joint policy is written as $\boldsymbol{\pi} = (\pi^i, \boldsymbol{\pi}^{-i})$. This section discusses various approaches for computing a response to a known $\boldsymbol{\pi}^{-i}$.

Rock-paper-scissors is a zero-sum game for two agents. Each agent selects *rock*, *paper*, or *scissors*. Rock wins against scissors, paper wins against rock, and scissors wins against paper, with a reward of 1 for the winner and -1 for the loser. If the agents select the same action, both receive 0 reward. Generally, two-agent repeated games can be represented as a sequence of payoff matrices, as shown here:

$t = 1$

		agent 2		
		rock	paper	scissors
agent 1	rock	$0, 0$	$-1, 1$	$1, -1$
	paper	$1, -1$	$0, 0$	$-1, 1$
	scissors	$-1, 1$	$1, -1$	$0, 0$

$t = 2$

		agent 2		
		rock	paper	scissors
agent 1	rock	$0, 0$	$-1, 1$	$1, -1$
	paper	$1, -1$	$0, 0$	$-1, 1$
	scissors	$-1, 1$	$1, -1$	$0, 0$

$\cdots$

$\cdots$

Example 24.2. The well-known game of rock-paper-scissors is an example of a zero-sum game. Appendix F.11 provides additional details.

24.2.1 Best Response

A *best response* of agent i to the policies of the other agents $\boldsymbol{\pi}^{-i}$ is a policy π^i that satisfies

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \tag{24.2}$$

for all other policies $\pi^{i\prime} \neq \pi^i$. In other words, a best response for an agent is a policy where there is no incentive for them to change their policy, given a fixed set of policies for the other agents. There may be multiple best responses.

If we restrict ourselves to deterministic policies, a *deterministic best response* to opponent policies $\boldsymbol{\pi}^{-i}$ is straightforward to compute. We simply iterate over all of agent i's actions and return the one that maximizes the utility as follows:

$$\underset{a^i \in \mathcal{A}^i}{\arg\max}\, U^i(a^i, \boldsymbol{\pi}^{-i}) \tag{24.3}$$

Algorithm 24.3 provides an implementation of this.

```
struct SimpleGamePolicy
    p # dictionary mapping actions to probabilities

    function SimpleGamePolicy(p::Base.Generator)
        return SimpleGamePolicy(Dict(p))
    end

    function SimpleGamePolicy(p::Dict)
        vs = collect(values(p))
        vs ./= sum(vs)
        return new(Dict(k => v for (k,v) in zip(keys(p), vs)))
    end

    SimpleGamePolicy(ai) = new(Dict(ai => 1.0))
end

(πi::SimpleGamePolicy)(ai) = get(πi.p, ai, 0.0)

function (πi::SimpleGamePolicy)()
    D = SetCategorical(collect(keys(πi.p)), collect(values(πi.p)))
    return rand(D)
end

joint(X) = vec(collect(product(X...)))

joint(π, πi, i) = [i == j ? πi : πj for (j, πj) in enumerate(π)]

function utility(𝒫::SimpleGame, π, i)
    𝒜, R = 𝒫.𝒜, 𝒫.R
    p(a) = prod(πj(aj) for (πj, aj) in zip(π, a))
    return sum(R(a)[i]*p(a) for a in joint(𝒜))
end
```

Algorithm 24.2. A policy associated with an agent is represented by a dictionary that maps actions to probabilities. There are different ways to construct a policy. One way is to pass in a dictionary directory, in which case the probabilities are normalized. Another way is to pass in a generator that creates this dictionary. We can also construct a policy by passing in an action, in which case it assigns probability 1 to that action. If we have an individual policy `πi`, we can call `πi(ai)` to compute the probability the policy associates with action `ai`. If we call `πi()`, then it will return a random action according to that policy. We can use `joint(𝒜)` to construct the joint action space from `𝒜`. We can use `utility(𝒫, π, i)` to compute the utility associated with executing joint policy `π` in the game `𝒫` from the perspective of agent `i`.

```
function best_response(𝒫::SimpleGame, π, i)
    U(ai) = utility(𝒫, joint(π, SimpleGamePolicy(ai), i), i)
    ai = argmax(U, 𝒫.𝒜[i])
    return SimpleGamePolicy(ai)
end
```

Algorithm 24.3. For a simple game `𝒫`, we can compute a deterministic best response for agent `i`, given that the other agents are playing the policies in `π`.

24.2.2 Softmax Response

We can use a *softmax response* to model how agent i will select their action.[4] As discussed in section 6.7, humans are often not perfectly rational optimizers of expected utility. The principle underlying the softmax response model is that (typically human) agents are more likely to make errors in their optimization when those errors are less costly. Given a *precision parameter* $\lambda \geq 0$, this model selects action a^i according to

$$\pi^i(a^i) \propto \exp(\lambda U^i(a^i, \boldsymbol{\pi}^{-i})) \tag{24.4}$$

As $\lambda \to 0$, the agent is insensitive to differences in utility, and selects actions uniformly at random. As $\lambda \to \infty$, the policy converges to a deterministic best response. We can treat λ as a parameter that can be learned from data using, for example, maximum likelihood estimation (section 4.1). This learning-based approach aims to be predictive of behavior rather than prescriptive of behavior, though having a predictive model of other human agents can be useful in building a system that prescribes optimal behavior. Algorithm 24.4 provides an implementation of a softmax response.

[4] This kind of model is sometimes referred to as a *logit response* or *quantal response*. We introduced similar softmax models earlier in this book, in the context of directed exploration strategies for reinforcement learning (section 15.4).

```
function softmax_response(𝒫::SimpleGame, π, i, λ)
    𝒜i = 𝒫.𝒜[i]
    U(ai) = utility(𝒫, joint(π, SimpleGamePolicy(ai), i), i)
    return SimpleGamePolicy(ai => exp(λ*U(ai)) for ai in 𝒜i)
end
```

Algorithm 24.4. For a simple game 𝒫 and a particular agent `i`, we can compute the softmax response policy `πi`, given that the other agents are playing the policies in `π`. This computation requires specifying the precision parameter `λ`.

24.3 Dominant Strategy Equilibrium

In some games, an agent has a *dominant strategy*, which is a policy that is a best response against all other possible agent policies. For example, in the prisoner's dilemma (example 24.1), the best response of agent 1 is to defect regardless of the policy of agent 2, making defect a dominant strategy for agent 1. A joint policy where all the agents use dominant strategies is called a *dominant strategy equilibrium*. In the prisoner's dilemma, a joint policy where both agents defect is a dominant strategy equilibrium.[5] Many games do not have a dominant strategy equilibrium. For example, in rock-paper-scissors (example 24.2), the best response of agent 1 depends on the strategy of agent 2.

[5] Interestingly, having both agents act greedily with respect to their own utility function results in a worse outcome for both of them. If they had both cooperated, then they would both get a sentence of one year instead of three years.

24.4 Nash Equilibrium

In contrast with the dominant strategy equilibrium concept, a Nash equilibrium[6] always exists for games with a finite action space.[7] A *Nash equilibrium* is a joint policy $\boldsymbol{\pi}$ in which all agents are following a best response. In other words, a Nash equilibrium is a joint policy in which no agents have an incentive to unilaterally switch their policy.

Multiple Nash equilibria can exist in a single game (exercise 24.2). Sometimes Nash equilibria may involve deterministic policies, but this is not always the case (see example 24.3). Computing a Nash equilibrium is *PPAD-complete*, a class that is distinct from NP-complete (appendix C.2) but also has no known polynomial time algorithm.[8]

[6] Named for the American mathematician John Forbes Nash, Jr. (1928–2015) who formalized the concept. J. Nash, "Non-Cooperative Games," *Annals of Mathematics*, pp. 286–295, 1951.

[7] Exercise 24.1 explores the case where the action space is infinite.

[8] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The Complexity of Computing a Nash Equilibrium," *Communications of the ACM*, vol. 52, no. 2, pp. 89–97, 2009.

The problem of finding a Nash equilibrium can be framed as an optimization problem:

$$\begin{aligned}
\underset{\boldsymbol{\pi},U}{\text{minimize}} \quad & \sum_i \Big(U^i - U^i(\boldsymbol{\pi})\Big) \\
\text{subject to} \quad & U^i \geq U^i(a^i, \boldsymbol{\pi}^{-i}) \text{ for all } i, a^i \\
& \sum_{a^i} \pi^i(a^i) = 1 \text{ for all } i \\
& \pi^i(a^i) \geq 0 \text{ for all } i, a^i
\end{aligned} \tag{24.5}$$

The optimization variables correspond to the parameters of $\boldsymbol{\pi}$ and U. At convergence, the objective will be 0, with U^i matching the utilities associated with policy $\boldsymbol{\pi}$ as computed in equation (24.1) for each agent i. The first constraint ensures that no agent will do better by unilaterally changing their action. Like the objective, this first constraint is nonlinear because it involves a product of the parameters in the optimization variable $\boldsymbol{\pi}$. The last two constraints are linear, ensuring that $\boldsymbol{\pi}$ represents a valid set of probability distributions over actions. Algorithm 24.5 implements this optimization procedure.

24.5 Correlated Equilibrium

The *correlated equilibrium* generalizes the Nash equilibrium concept by relaxing the assumption that the agents act independently. The joint action in this case comes from a full joint distribution. A *correlated joint policy* $\pi(\mathbf{a})$ is a single distribution over the joint actions of all agents. Consequently, the actions of the various agents

Example 24.3. Deterministic and stochastic Nash equilibria.

Suppose that we wish to find a Nash equilibrium for the prisoner's dilemma from example 24.1. If both agents always defect, both receive -3 reward. Any deviation by any agent will result in a -4 reward for that agent; hence, there is no incentive to deviate. Having both agents defect is thus a Nash equilibrium for the prisoner's dilemma.

Suppose that we now wish to find a Nash equilibrium for the rock-paper-scissors scenario from example 24.2. Any deterministic strategy by one agent can be easily countered by the other agent. For example, if agent 1 plays rock, then agent 2's best response is paper. Because there is no deterministic Nash equilibrium for rock-paper-scissors, we know that there must be one involving stochastic policies. Suppose that each agent selects from the actions uniformly at random. This solution produces an expected utility of 0 for both agents:

$$\begin{aligned} U^i(\boldsymbol{\pi}) &= 0\frac{1}{3}\frac{1}{3} - 1\frac{1}{3}\frac{1}{3} + 1\frac{1}{3}\frac{1}{3} \\ &\quad + 1\frac{1}{3}\frac{1}{3} + 0\frac{1}{3}\frac{1}{3} - 1\frac{1}{3}\frac{1}{3} \\ &\quad - 1\frac{1}{3}\frac{1}{3} + 1\frac{1}{3}\frac{1}{3} + 0\frac{1}{3}\frac{1}{3} \\ &= 0 \end{aligned}$$

Any deviation by an agent would decrease their expected payoff, meaning that we have found a Nash equilibrium.

```
struct NashEquilibrium end

function tensorform(𝒫::SimpleGame)
    ℐ, 𝒜, R = 𝒫.ℐ, 𝒫.𝒜, 𝒫.R
    ℐ′ = eachindex(ℐ)
    𝒜′ = [eachindex(𝒜[i]) for i in ℐ]
    R′ = [R(a) for a in joint(𝒜)]
    return ℐ′, 𝒜′, R′
end

function solve(M::NashEquilibrium, 𝒫::SimpleGame)
    ℐ, 𝒜, R = tensorform(𝒫)
    model = Model(Ipopt.Optimizer)
    @variable(model, U[ℐ])
    @variable(model, π[i=ℐ, 𝒜[i]] ≥ 0)
    @NLobjective(model, Min,
        sum(U[i] - sum(prod(π[j,a[j]] for j in ℐ) * R[y][i]
            for (y,a) in enumerate(joint(𝒜))) for i in ℐ))
    @NLconstraint(model, [i=ℐ, ai=𝒜[i]],
        U[i] ≥ sum(
            prod(j==i ? (a[j]==ai ? 1.0 : 0.0) : π[j,a[j]] for j in ℐ)
            * R[y][i] for (y,a) in enumerate(joint(𝒜))))
    @constraint(model, [i=ℐ], sum(π[i,ai] for ai in 𝒜[i]) == 1)
    optimize!(model)
    πi′(i) = SimpleGamePolicy(𝒫.𝒜[i][ai] => value(π[i,ai]) for ai in 𝒜[i])
    return [πi′(i) for i in ℐ]
end
```

Algorithm 24.5. This nonlinear program computes a Nash equilibrium for a simple game $\mathcal{P}$.

may be correlated, preventing the policies from being decoupled into individual policies $\pi^i(a^i)$. Algorithm 24.6 shows how to represent such a policy.

```
mutable struct JointCorrelatedPolicy
    p # dictionary mapping from joint actions to probabilities
    JointCorrelatedPolicy(p::Base.Generator) = new(Dict(p))
end

(π::JointCorrelatedPolicy)(a) = get(π.p, a, 0.0)

function (π::JointCorrelatedPolicy)()
    D = SetCategorical(collect(keys(π.p)), collect(values(π.p)))
    return rand(D)
end
```

Algorithm 24.6. A joint correlated policy is represented by a dictionary that maps joint actions to probabilities. If `π` is a joint correlated policy, evaluating `π(a)` will return the probability associated with the joint action `a`.

A *correlated equilibrium* is a correlated joint policy where no agent i can increase their expected utility by deviating from their current action a^i to another action $a^{i\prime}$:

$$\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \tag{24.6}$$

Example 24.4 demonstrates this concept.

Every Nash equilibrium is a correlated equilibrium because we can always form a joint policy from independent policies:

$$\pi(\mathbf{a}) = \prod_{i=1}^{k} \pi^i(a^i) \tag{24.7}$$

If the individual policies satisfy equation (24.2), then the joint policy will satisfy equation (24.6). Not all correlated equilibria, however, are Nash equilibria.

A correlated equilibrium can be computed using linear programming (algorithm 24.7):

$$\begin{aligned} \underset{\boldsymbol{\pi}}{\text{maximize}} \quad & \sum_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a}) \\ \text{subject to} \quad & \sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \quad \text{for all } i, a^i, a^{i\prime} \\ & \sum_{\mathbf{a}} \pi(\mathbf{a}) = 1 \\ & \pi(\mathbf{a}) \geq 0 \quad \text{for all } \mathbf{a} \end{aligned} \tag{24.8}$$

Example 24.4. Computing correlated equilibria in rock-paper-scissors.

Consider again the rock-paper-scissors scenario from example 24.2. In example 24.3, we found that a Nash equilibrium involves both agents selecting their actions uniformly at random. In correlated equilibria, we use a correlated joint policy $\pi(\mathbf{a})$, meaning that we need to find a distribution over (rock, rock), (rock, paper), (rock, scissors), (paper, rock), and so on. There are nine possible joint actions.

First, consider the joint policy in which agent 1 selects rock and agent 2 selects scissors. The utilities are

$$U^1(\pi) = 0\frac{0}{9} - 1\frac{0}{9} + 1\frac{9}{9} + 1\frac{0}{9} + \cdots = 1$$
$$U^2(\pi) = 0\frac{0}{9} + 1\frac{0}{9} - 1\frac{9}{9} - 1\frac{0}{9} + \cdots = -1$$

If agent 2 switched to paper, it would receive a utility of 1. Hence, this is not a correlated equilibrium.

Consider instead a correlated joint policy in which the joint action was chosen uniformly at random, with $\pi(\mathbf{a}) = 1/9$:

$$U^1(\pi) = 0\frac{1}{9} - 1\frac{1}{9} + 1\frac{1}{9} + 1\frac{1}{9} + \cdots = 0$$
$$U^2(\pi) = 0\frac{1}{9} + 1\frac{1}{9} - 1\frac{1}{9} - 1\frac{1}{9} + \cdots = 0$$

Any deviation from this results in one agent gaining utility and the other losing utility. This is a correlated equilibrium for rock-paper-scissors.

Although linear programs can be solved in polynomial time, the size of the joint action space grows exponentially with the number of agents. The constraints enforce a correlated equilibrium. The objective, however, can be used to select among different valid correlated equilibria. Table 24.1 provides several common choices for the objective function.

```
struct CorrelatedEquilibrium end

function solve(M::CorrelatedEquilibrium, 𝒫::SimpleGame)
    ℐ, 𝒜, R = 𝒫.ℐ, 𝒫.𝒜, 𝒫.R
    model = Model(Ipopt.Optimizer)
    @variable(model, π[joint(𝒜)] ≥ 0)
    @objective(model, Max, sum(sum(π[a]*R(a) for a in joint(𝒜))))
    @constraint(model, [i=ℐ, ai=𝒜[i], ai′=𝒜[i]],
        sum(R(a)[i]*π[a] for a in joint(𝒜) if a[i]==ai)
        ≥ sum(R(joint(a,ai′,i))[i]*π[a] for a in joint(𝒜) if a[i]==ai))
    @constraint(model, sum(π) == 1)
    optimize!(model)
    return JointCorrelatedPolicy(a ⇒ value(π[a]) for a in joint(𝒜))
end
```

Algorithm 24.7. Correlated equilibria are a more general notion of optimality for a simple game $\mathcal{P}$ than a Nash equilibrium. They can be computed using a linear program. The resulting policies are correlated, meaning that the agents stochastically select their joint actions.

Name	Description	Objective Function
Utilitarian	Maximize the net utility.	$\text{maximize}_\pi \sum_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Egalitarian	Maximize the minimum of all agents' utilities.	$\text{maximize}_\pi \, \text{minimize}_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Plutocratic	Maximize the maximum of all agents' utilities.	$\text{maximize}_\pi \, \text{maximize}_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Dictatorial	Maximize agent i's utility.	$\text{maximize}_\pi \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$

Table 24.1. Alternative objective functions for equation (24.8), which select for various correlated equilibria. These descriptions were adapted from A. Greenwald and K. Hall, "Correlated Q-Learning," in *International Conference on Machine Learning (ICML)*, 2003.

24.6 *Iterated Best Response*

Because computing a Nash equilibrium can be computationally expensive, an alternative approach is to iteratively apply best responses in a series of repeated games. In *iterated best response* (algorithm 24.8), we randomly cycle between agents, solving for each agent's best response policy in turn. This process may converge to a Nash equilibrium, but there are guarantees only for certain classes of games.[9] In many problems, it is common to observe cycles.

[9] Iterated best response will converge, for example, for a class known as *potential games*, as discussed in Theorem 19.12 of the textbook by N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, eds., *Algorithmic Game Theory*. Cambridge University Press, 2007.

```
struct IteratedBestResponse
    k_max # number of iterations
    π     # initial policy
end

function IteratedBestResponse(𝒫::SimpleGame, k_max)
    π = [SimpleGamePolicy(ai => 1.0 for ai in 𝒜i) for 𝒜i in 𝒫.𝒜]
    return IteratedBestResponse(k_max, π)
end

function solve(M::IteratedBestResponse, 𝒫)
    π = M.π
    for k in 1:M.k_max
        π = [best_response(𝒫, π, i) for i in 𝒫.ℐ]
    end
    return π
end
```

Algorithm 24.8. Iterated best response involves cycling through the agents and applying their best response to the other agents. The algorithm starts with some initial policy and stops after `k_max` iterations. For convenience, we have a constructor that takes as input a simple game and creates an initial policy that has each agent select actions uniformly at random. The same solve function will be reused in the next chapter in the context of more complicated forms of games.

24.7 *Hierarchical Softmax*

An area known as *behavioral game theory* aims to model human agents. When building decision-making systems that must interact with humans, computing the Nash equilibrium is not always helpful. Humans often do not play a Nash equilibrium strategy. First, it may be unclear which equilibrium to adopt if there are many different equilibria in the game. For games with only one equilibrium, it may be difficult for a human to compute the Nash equilibrium because of cognitive limitations. Even if human agents can compute the Nash equilibrium, they may doubt that their opponents can perform that computation.

There are many behavioral models in the literature,[10] but one approach is to combine the iterated approach from the previous section with a softmax model. This *hierarchical softmax* approach (algorithm 24.9)[11] models the *depth of rationality* of an agent by a level of $k \geq 0$. A level 0 agent plays its actions uniformly at random. A level 1 agent assumes the other players adopt level 0 strategies and selects actions according to a softmax response with precision λ. A level k agent selects actions according to a softmax model of the other players playing level $k-1$. Figure 24.1 illustrates this approach for a simple game.

[10] C. F. Camerer, *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press, 2003.

[11] This approach is sometimes called *quantal-level-k* or *logit-level-k*. D. O. Stahl and P. W. Wilson, "Experimental Evidence on Players' Models of Other Players," *Journal of Economic Behavior & Organization*, vol. 25, no. 3, pp. 309–327, 1994.

We can learn the k and λ parameters of this behavioral model from data. If we have a collection of joint actions played by different agents, we can compute the associated likelihood for a given k and λ. We can then use an optimization algorithm to attempt to find values of k and λ that maximize likelihood. This

```
struct HierarchicalSoftmax
    λ # precision parameter
    k # level
    π # initial policy
end

function HierarchicalSoftmax(𝒫::SimpleGame, λ, k)
    π = [SimpleGamePolicy(ai => 1.0 for ai in 𝒜i) for 𝒜i in 𝒫.𝒜]
    return HierarchicalSoftmax(λ, k, π)
end

function solve(M::HierarchicalSoftmax, 𝒫)
    π = M.π
    for k in 1:M.k
        π = [softmax_response(𝒫, π, i, M.λ) for i in 𝒫.ℐ]
    end
    return π
end
```

Algorithm 24.9. The hierarchical softmax model with precision parameter `λ` and level `k`. By default, it starts with an initial joint policy that assigns uniform probability to all individual actions.

optimization typically cannot be done analytically, but we can use numerical methods to perform this optimization.[12] Alternatively, we can use a Bayesian approach to parameter learning.[13]

[12] J. R. Wright and K. Leyton-Brown, "Beyond Equilibrium: Predicting Human Behavior in Normal Form Games," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[13] J. R. Wright and K. Leyton-Brown, "Behavioral Game Theoretic Models: A Bayesian Framework for Parameter Analysis," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

24.8 Fictitious Play

An alternative approach for computing policies for different agents is to have them play each other in simulation and learn how to best respond. Algorithm 24.10 provides an implementation of the simulation loop. At each iteration, we evaluate the various policies to obtain a joint action, and then this joint action is used by the agents to update their policies. We can use a number of ways to update the policies in response to observed joint actions. This section focuses on *fictitious play*, where the agents use maximum likelihood estimates (as described in section 16.1) of the policies followed by the other agents. Each agent follows its own best response, assuming that the other agents act according to those estimates.[14]

[14] G. W. Brown, "Iterative Solution of Games by Fictitious Play," *Activity Analysis of Production and Allocation*, vol. 13, no. 1, pp. 374–376, 1951. J. Robinson, "An Iterative Method of Solving a Game," *Annals of Mathematics*, pp. 296–301, 1951.

To compute a maximum likelihood estimate, agent i tracks the number of times that agent j takes action a^j, storing it in table $N^i(j, a^j)$. These counts can be initialized to any value, but they are often initialized to 1 to create initial uniform uncertainty. Agent i computes its best response, assuming that each agent j follows the stochastic policy:

$$\pi^j(a^j) \propto N^i(j, a^j) \tag{24.9}$$

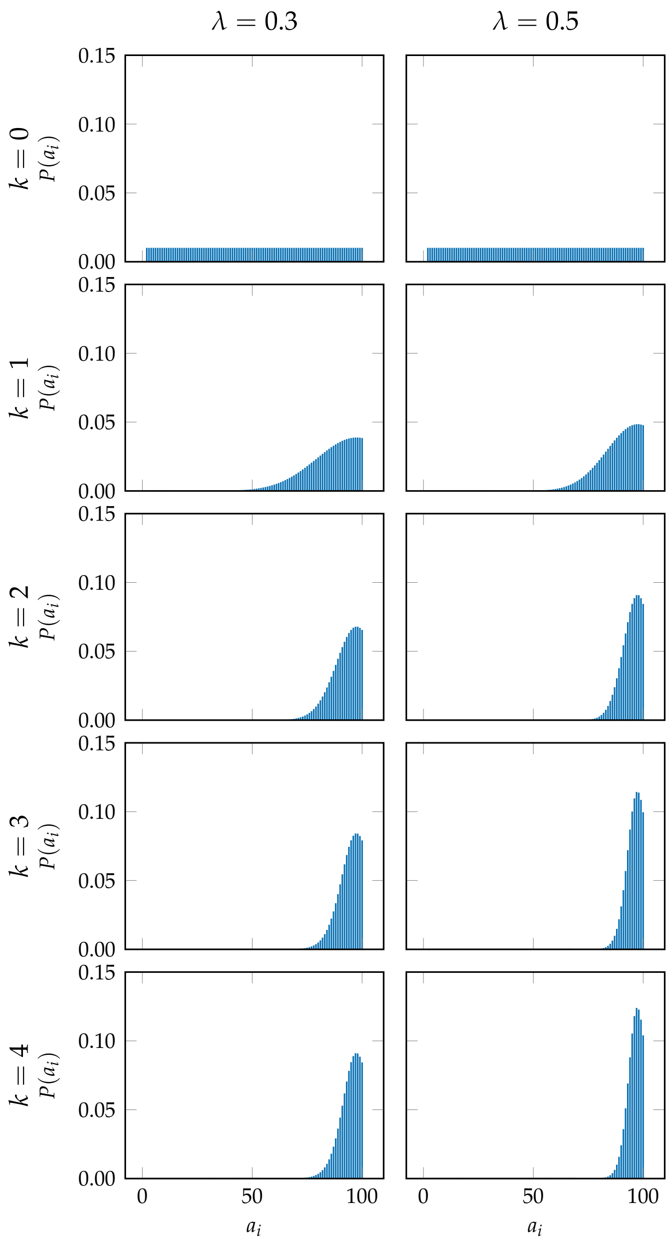


Figure 24.1. The hierarchical softmax model applied to the traveler's dilemma (described in appendix F.12) for various depths of rationality k and precision parameters λ. People tend to select actions between \$97 and \$100, even though the Nash equilibrium is only \$2.

```
function simulate(𝒫::SimpleGame, π, k_max)
    for k = 1:k_max
        a = [πi() for πi in π]
        for πi in π
            update!(πi, a)
        end
    end
    return π
end
```

Algorithm 24.10. A simulation of a joint policy in simple game 𝒫 for `k_max` iterations. The joint policy π is a vector of policies that can be individually updated through calls to `update!(πi, a)`.

At each iteration, we have each agent act according to a best response, assuming these stochastic count-based policies for the other agents. We then update the action counts for the actions taken. Algorithm 24.11 implements this simple adaptive procedure. Figures 24.2 and 24.3 show how the policies evolve over time using fictitious play. Fictitious play is not guaranteed to converge to a Nash equilibrium.[15]

[15] A concise background is provided by U. Berger, "Brown's Original Fictitious Play," *Journal of Economic Theory*, vol. 135, no. 1, pp. 572–578, 2007.

```
mutable struct FictitiousPlay
    𝒫  # simple game
    i  # agent index
    N  # array of action count dictionaries
    πi # current policy
end

function FictitiousPlay(𝒫::SimpleGame, i)
    N = [Dict(aj => 1 for aj in 𝒫.𝒜[j]) for j in 𝒫.ℐ]
    πi = SimpleGamePolicy(ai => 1.0 for ai in 𝒫.𝒜[i])
    return FictitiousPlay(𝒫, i, N, πi)
end

(πi::FictitiousPlay)() = πi.πi()

(πi::FictitiousPlay)(ai) = πi.πi(ai)

function update!(πi::FictitiousPlay, a)
    N, 𝒫, ℐ, i = πi.N, πi.𝒫, πi.𝒫.ℐ, πi.i
    for (j, aj) in enumerate(a)
        N[j][aj] += 1
    end
    p(j) = SimpleGamePolicy(aj => u/sum(values(N[j])) for (aj, u) in N[j])
    π = [p(j) for j in ℐ]
    πi.πi = best_response(𝒫, π, i)
end
```

Algorithm 24.11. Fictitious play is a simple learning algorithm for an agent `i` of a simple game 𝒫 that maintains `counts` of other agent action selections over time and averages them, assuming that this is their stochastic policy. It then computes a best response to this policy and performs the corresponding utility-maximizing action.

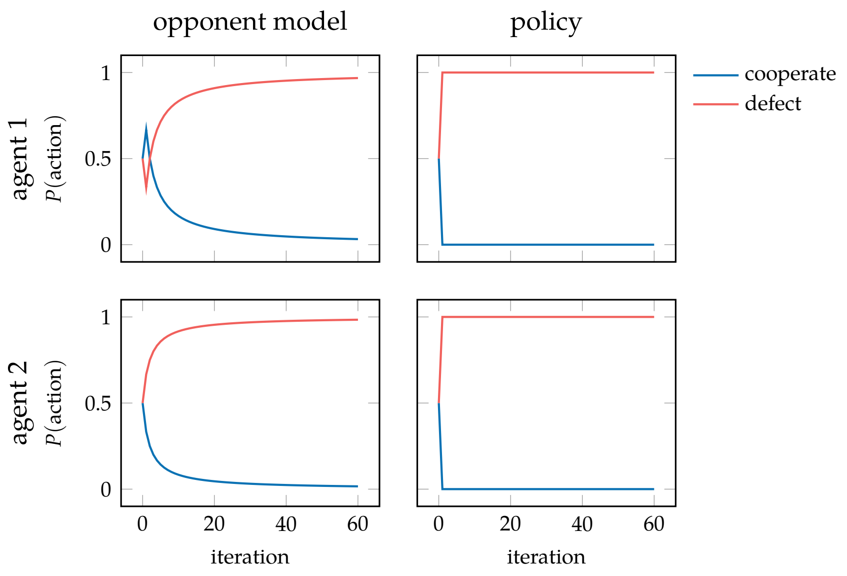


Figure 24.2. Two fictitious play agents learning and adapting to one another in a prisoner's dilemma game. The first row illustrates agent 1's learned model of 2 (left) and agent 1's policy (right) over iteration. The second row follows the same pattern, but for agent 2. To illustrate variation in learning behavior, the initial counts for each agent's model over the other agent's action were assigned to a random number between 1 and 10.

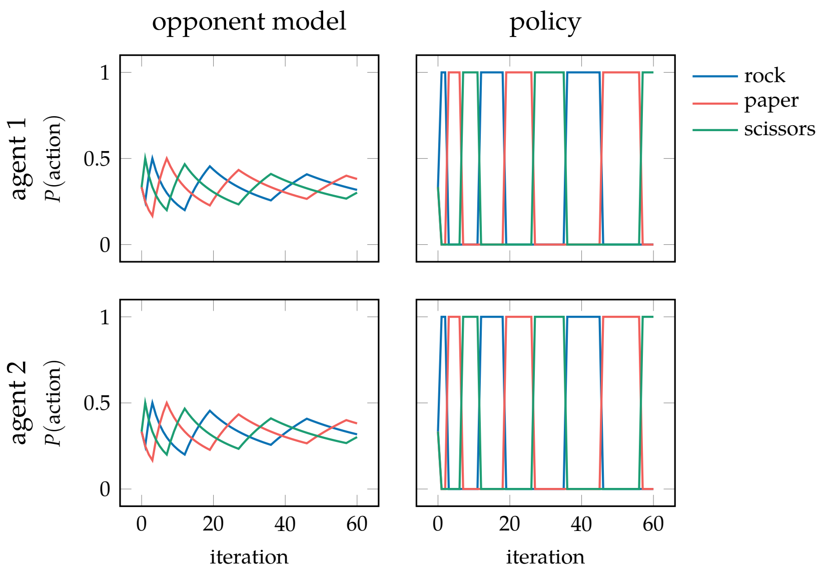


Figure 24.3. A visualization of two fictitious play agents learning and adapting to one another in a rock-paper-scissors game. The first row illustrates agent 1's learned model of 2 (left) and agent 1's policy (right) over time. The second row follows the same pattern, but for agent 2. To illustrate variation in learning behavior, the initial counts for each agent's model over the other agent's action were assigned to a random number between 1 and 10. In this zero-sum game, fictitious play agents approach convergence to their stochastic policy Nash equilibrium.

There are many variants of fictitious play. One variant, called *smooth fictitious play*,[16] selects a best response using expected utility plus a smoothing function, such as the entropy of the policy. Another variant is called *rational learning* or *Bayesian learning*. Rational learning expands the model of fictitious play to be any belief over other agents' actions, formulated as a Bayesian prior. Bayes' rule is then used to update the beliefs, given the history of joint actions. Traditional fictitious play can be seen as rational learning with a Dirichlet prior (section 4.2.2).

[16] D. Fudenberg and D. Levine, "Consistency and Cautious Fictitious Play," *Journal of Economic Dynamics and Control*, vol. 19, no. 5–7, pp. 1065–1089, 1995.

24.9 Gradient Ascent

Gradient ascent (algorithm 24.12) incrementally adjusts the agent's policy in the gradient with respect to its utility. At time t, the gradient for agent i is

$$\frac{\partial U^i(\boldsymbol{\pi}_t)}{\partial \pi_t^i(a^i)} = \frac{\partial}{\partial \pi_t^i}\left(\sum_{\mathbf{a}} R^i(\mathbf{a}) \prod_j \pi_t^j(a^j)\right) = \sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \prod_{j \neq i} \pi_t^j(a^j) \tag{24.10}$$

We can then use standard gradient ascent with

$$\pi_{t+1}^i(a^i) = \pi_t^i(a^i) + \alpha_t^i \frac{\partial U^i(\boldsymbol{\pi}_t)}{\partial \pi_t^i(a^i)} \tag{24.11}$$

with learning rate α_t^i.[17] This π_{t+1}^i may need to be projected back to a valid probability distribution, just as in section 23.4 for POMDP policies.

In practice, however, an agent i knows only its own policy π_t^i, not the policies of the others, making the computation of the gradient difficult. But agents do observe the joint actions $\mathbf{a}_t$ that are performed. Although we could try to estimate their policies as done in fictitious play, one simple approach is to assume the policy of the other agents is to replay their most recent action.[18] The gradient then simplifies to

$$\frac{\partial U^i(\boldsymbol{\pi}_t)}{\partial \pi_t^i(a^i)} = R^i(a^i, \mathbf{a}^{-i}) \tag{24.12}$$

Figure 24.4 demonstrates this approach for a simple rock-paper-scissors game.

[17] The *infinitesimal gradient ascent* method uses an inverse square root learning rate of $\alpha_t^i = 1/\sqrt{t}$. It is referred to as infinitesimal because $\alpha_t^i \to 0$ as $t \to \infty$. We use this learning rate in our implementation. S. Singh, M. Kearns, and Y. Mansour, "Nash Convergence of Gradient Dynamics in General-Sum Games," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[18] This approach is used in *generalized infinitesimal gradient ascent (GIGA)*. M. Zinkevich, "Online Convex Programming and Generalized Infinitesimal Gradient Ascent," in *International Conference on Machine Learning (ICML)*, 2003. A variation of the gradient update rule to encourage convergence is proposed by M. Bowling, "Convergence and No-Regret in Multiagent Learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

24.10 Summary

- In simple games, multiple agents compete to maximize expected reward.

```
mutable struct GradientAscent
    𝒫  # simple game
    i  # agent index
    t  # time step
    πi # current policy
end

function GradientAscent(𝒫::SimpleGame, i)
    uniform() = SimpleGamePolicy(ai => 1.0 for ai in 𝒫.𝒜[i])
    return GradientAscent(𝒫, i, 1, uniform())
end

(πi::GradientAscent)() = πi.πi()

(πi::GradientAscent)(ai) = πi.πi(ai)

function update!(πi::GradientAscent, a)
    𝒫, ℐ, 𝒜i, i, t = πi.𝒫, πi.𝒫.ℐ, πi.𝒫.𝒜[πi.i], πi.i, πi.t
    jointπ(ai) = [SimpleGamePolicy(j == i ? ai : a[j]) for j in ℐ]
    r = [utility(𝒫, jointπ(ai), i) for ai in 𝒜i]
    π′ = [πi.πi(ai) for ai in 𝒜i]
    π = project_to_simplex(π′ + r / sqrt(t))
    πi.t = t + 1
    πi.πi = SimpleGamePolicy(ai => p for (ai, p) in zip(𝒜i, π))
end
```

Algorithm 24.12. An implementation of gradient ascent for an agent `i` of a simple game 𝒫. The algorithm updates its distribution over actions incrementally following gradient ascent to improve the expected utility. The projection function from algorithm 23.6 is used to ensure that the resulting policy remains a valid probability distribution.

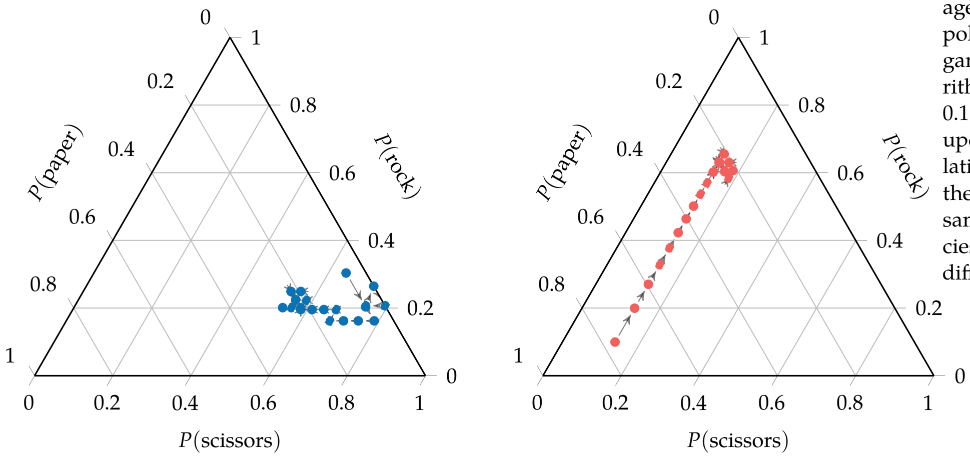


Figure 24.4. Two gradient ascent agents with randomly initialized policies in a rock-paper-scissors game. We use a variation of algorithm 24.12 with a learning rate of $0.1/\sqrt{t}$. Shown here are 20 policy updates. Although different simulation traces will converge because the step size goes to 0, different samples from the stochastic policies may result in convergence to different policies.

- Optimality is not as straightforward in the multiagent setting, with multiple possible solution concepts for extracting policies from a reward specification.
- A best response of an agent to a fixed set of policies of the other agents is one where there is no incentive to deviate.
- A Nash equilibrium is a joint policy where all agents follow a best response.
- A correlated equilibrium is the same as a Nash equilibrium, except that all the agents follow a single joint action distribution that allows correlation between agents.
- Iterated best response can quickly optimize a joint policy by iteratively applying best responses, but there are no general guarantees of convergence.
- Hierarchical softmax attempts to model agents in terms of their depth of rationality and precision, which can be learned from past joint actions.
- Fictitious play is a learning algorithm that uses maximum-likelihood action models for other agents to find best response policies, with the potential to converge to a Nash equilibrium.
- Gradient ascent, followed by projection onto the probability simplex, can be used to learn policies.

24.11 *Exercises*

Exercise 24.1. Give an example of a game with two agents and an infinite number of actions such that a Nash equilibrium does not exist.

Solution: Suppose that the action space of each agent consists of the negative real numbers and their reward is equal to their action. Since no greatest negative number exists, a Nash equilibrium cannot exist.

Exercise 24.2. Give an example of a game with two agents, two actions, and two Nash equilibria involving deterministic policies.

Solution: Here is one example.[19] Suppose that we have two aircraft on a collision course, and the pilots of each aircraft must choose between climb or descend to avoid collision. If the pilots both choose the same maneuver, then there is a crash, with utility -4 to both pilots. Because climbing requires more fuel than descending, there is an additional penalty of -1 to any pilot who decides to climb.

[19] This example comes from M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

		agent 2	
		climb	descend
agent 1	climb	$-5, -5$	$-1, 0$
	descend	$0, -1$	$-4, -4$

Exercise 24.3. Given a stationary joint policy $\boldsymbol{\pi}$ that is a Nash equilibrium for a simple game with a horizon of 1, prove that it is also a Nash equilibrium for the same simple game repeated to any finite or infinite horizon.

Solution: By definition of a Nash equilibrium, all agents i are performing a best response π^i to all other policies $\pi^{i\prime} \neq \pi^i$ following equation (24.2):

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i})$$

By definition of U^i, we have

$$U^i(\boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j)$$

The joint policy remains constant over time for all agents. Apply any horizon n, with any discount factor ($\gamma = 1$ for $n < \infty$; $\gamma < 1$ for $n \to \infty$). The utility of agent i after n steps is

$$\begin{aligned}
U^{i,n}(\boldsymbol{\pi}) &= \sum_{t=1}^{n} \gamma^{t-1} \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j) \\
&= \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j) \sum_{t=1}^{n} \gamma^{t-1} \\
&= U^i(\boldsymbol{\pi}) \sum_{t=1}^{n} \gamma^{t-1} \\
&= U^i(\boldsymbol{\pi}) c
\end{aligned}$$

The discount factor becomes a constant multiplier $c > 0$. Therefore, any constant multiplication of equation (24.2) on both sides results in the same inequality, completing the proof:

$$\begin{aligned}
U^i(\pi^i, \boldsymbol{\pi}^{-i}) &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \\
U^i(\pi^i, \boldsymbol{\pi}^{-i})c &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i})c \\
U^i(\pi^i, \boldsymbol{\pi}^{-i}) \sum_{t=1}^{n} \gamma^{t-1} &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \sum_{t=1}^{n} \gamma^{t-1} \\
\sum_{t=1}^{n} \gamma^{t-1} U^i(\pi^i, \boldsymbol{\pi}^{-i}) &\geq \sum_{t=1}^{n} \gamma^{t-1} U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \\
U^{i,n}(\pi^i, \boldsymbol{\pi}^{-i}) &\geq U^{i,n}(\pi^{i\prime}, \boldsymbol{\pi}^{-i})
\end{aligned}$$

Exercise 24.4. Prove that a Nash equilibrium is a correlated equilibrium.

Solution: Consider any uncorrelated joint policy $\pi(\mathbf{a})$. For any agent i:

$$\pi(\mathbf{a}) = \prod_{j=1}^{k} \pi^j(a^j) = \pi^i(a^i) \prod_{j \neq i} \pi^j(a^j) \tag{24.13}$$

It is sufficient to show that a correlated equilibrium under this constraint forms the exact definition of Nash equilibrium. Begin by applying equation (24.13) to the definition of a correlated equilibrium. For all i, any a^i with nonzero probability[20] in π, and all $a^{i\prime}$:

[20] That is, $\sum_{\mathbf{a}^{-i}} \pi(a^i, \mathbf{a}^{-i}) > 0$. If it is zero, then the inequality trivially becomes true with $0 \geq 0$.

$$\begin{aligned}
\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \pi(a^i, \mathbf{a}^{-i}) &\geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \pi(a^i, \mathbf{a}^{-i}) \\
\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \pi^i(a^i) \prod_{j \neq i} \pi^j(a^j) &\geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \pi^i(a^i) \prod_{j \neq i} \pi^j(a^j) \\
\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) &\geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) \qquad (24.14)
\end{aligned}$$

Now consider the definition of utility:

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) = \sum_{\mathbf{a}} R^i(a^i, \mathbf{a}^{-i}) \prod_{j=1}^{k} \pi^j(a^j) = \sum_{a^i} \pi^i(a^i) \left(\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) \right)$$

Next apply equation (24.14) to the terms inside the parentheses:

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq \sum_{a^i} \pi^i(a^i) \left(\sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) \right) = \left(\sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) \right) \sum_{a^i} \pi^i(a^i) = \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j)$$

This equation holds for any action $a^{i\prime}$. Consequently, applying any probability weighting preserves the right side of this inequality. Consider any other policy $\pi^{i\prime}$ as a weighting:

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq \sum_{a^i} \pi^{i\prime}(a^i) \sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \prod_{j \neq i} \pi^j(a^j) = U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i})$$

This inequality is the definition of a best response. It must hold for all agents i and thus forms the definition of a Nash equilibrium. In summary, a Nash equilibrium is a special kind of correlated equilibrium that is constrained to an uncorrelated joint policy.

Exercise 24.5. Give an example of a two-agent game, each with two actions, for which the correlated equilibria cannot be represented as a Nash equilibrium.

Solution: Consider the following game, in which two people want to go on a date but have a conflicting preference on what kind of date (in this case, a dinner or a movie):

		agent 2	
		dinner	movie
agent 1	dinner	2, 1	0, 0
	movie	0, 0	1, 2

There is a stochastic Nash equilibrium. Agent 1 follows $\pi^1(\text{dinner}) = 2/3$ and $\pi^1(\text{movie}) = 1/3$. Agent 2 follows $\pi^2(\text{dinner}) = 1/3$ and $\pi^2(\text{movie}) = 2/3$. The utilities are:

$$U^1(\boldsymbol{\pi}) = \frac{2}{3} \cdot \frac{1}{3} \cdot 2 + \frac{2}{3} \cdot \frac{2}{3} \cdot 0 + \frac{1}{3} \cdot \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot \frac{2}{3} \cdot 1 = \frac{2}{9} \cdot 2 + \frac{2}{9} \cdot 1 = \frac{2}{3}$$
$$U^2(\boldsymbol{\pi}) = \frac{2}{3} \cdot \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \frac{2}{3} \cdot 0 + \frac{1}{3} \cdot \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot \frac{2}{3} \cdot 2 = \frac{2}{9} \cdot 1 + \frac{2}{9} \cdot 2 = \frac{2}{3}$$

However, if the two agents correlated their actions on a fair coin flip $\pi(\text{movie}, \text{movie}) = \pi(\text{dinner}, \text{dinner}) = 0.5$, then they could coordinate either both going to dinner or both going to the movie. The utilities are:

$$U^1(\boldsymbol{\pi}) = 0.5 \cdot 2 + 0.0 \cdot 0 + 0.0 \cdot 0 + 0.5 \cdot 1 = 0.5 \cdot 2 + 0.5 \cdot 1 = \frac{3}{2}$$
$$U^2(\boldsymbol{\pi}) = 0.5 \cdot 1 + 0.0 \cdot 0 + 0.0 \cdot 0 + 0.5 \cdot 2 = 0.5 \cdot 1 + 0.5 \cdot 2 = \frac{3}{2}$$

This is not possible with a Nash equilibrium. Intuitively, in this example, this is because the probabilistic weight is spread out over each row independently for each player. Conversely, a correlated equilibrium can be targeted toward a specific cell (in this case, with a higher payoff).

Exercise 24.6. Algorithms such as iterated best response and fictitious play do not converge in every game. Construct a game that demonstrates this nonconvergence.

Solution: Iterated best response diverges in rock-paper-scissors. Here is an example of the first 10 iterations with random initialization:

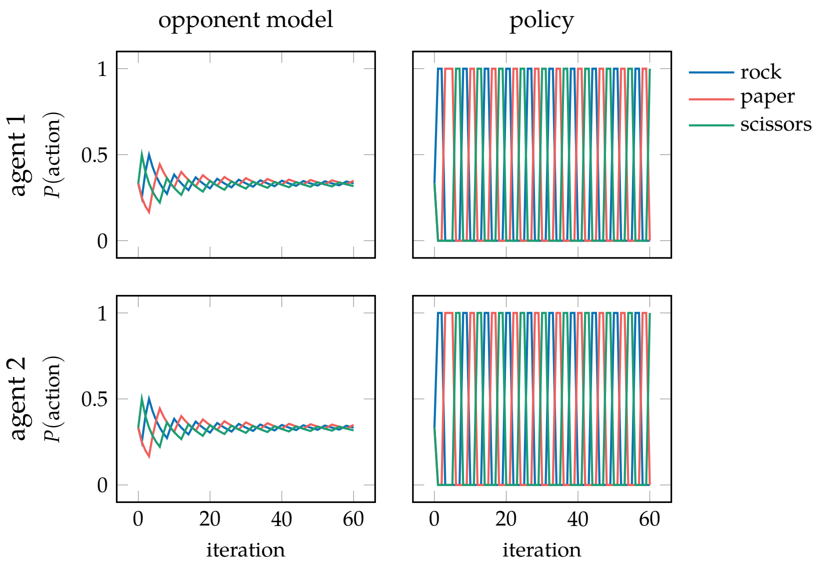
Iteration	Agent 1's Action	Agent 2's Action	Rewards
1	paper	rock	1.0, −1.0
2	paper	scissors	−1.0, 1.0
3	rock	scissors	1.0, −1.0
4	rock	paper	−1.0, 1.0
5	scissors	paper	1.0, −1.0
6	scissors	rock	−1.0, 1.0
7	paper	rock	1.0, −1.0
8	paper	scissors	−1.0, 1.0
9	rock	scissors	1.0, −1.0
10	rock	paper	−1.0, 1.0

Fictitious play also will not converge in *almost-rock-paper-scissors*:[21]

		agent 2		
		rock	paper	scissors
agent 1	rock	0, 0	0, 1	1, 0
	paper	1, 0	0, 0	0, 1
	scissors	0, 1	1, 0	0, 0

[21] This game and many others are discussed in greater detail by Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

Here is an example of fictitious play agents playing this game for 60 iterations:



Exercise 24.7. What does iterated best response converge to in the traveler's dilemma (appendix F.12)?

Solution: It converges to the Nash equilibrium of \$2.