次のコードのファクタ (Factor) 型を用いて，表 2.1 をインスタンス化できる．

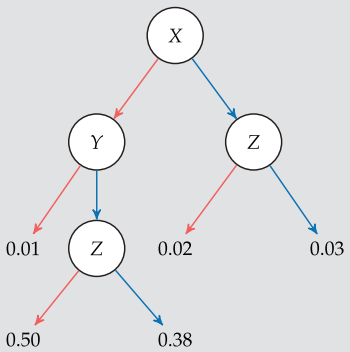
```
# requires convenience functions from appendix G.5
X = Variable(:x, 2)
Y = Variable(:y, 2)
Z = Variable(:z, 2)
ϕ = Factor([X, Y, Z], FactorTable(
    (x=1, y=1, z=1) ⇒ 0.08, (x=1, y=1, z=2) ⇒ 0.31
    (x=1, y=2, z=1) ⇒ 0.09, (x=1, y=2, z=2) ⇒ 0.37
    (x=2, y=1, z=1) ⇒ 0.01, (x=2, y=1, z=2) ⇒ 0.05
    (x=2, y=2, z=1) ⇒ 0.02, (x=2, y=2, z=2) ⇒ 0.07
))
```

例 2.3　離散的ファクタの構成．名前付きタプルを用いたファクタ表を構成して，付録 G.5 で定義された有用な関数を活用する．

繰り返して現れる値をもつ同時分布を表すために必要なストレージを削減するもう 1 つの方法は**決定木** (decision tree) を用いることである．3 つの離散変数を含む決定木を例 2.4 に示す．この例でのパラメータ数の削減はそれほど重要ではないかもしれないが，多くの変数と多くの繰返しの値がある場合にはかなり重要になりうる．

同時確率分布を表す次の表があるとする．表の値をより簡潔に表現するために，表の右にある決定木を用いる．変数が 0 の場合は赤い矢印に進み，変数が 1 の場合は青い矢印に進む．8 つの確率を保存する代わりに，決定木の表現とともに 5 つだけ確率を保存する．

X	Y	Z	$P(X,Y,Z)$
0	0	0	0.01
0	0	1	0.01
0	1	0	0.50
0	1	1	0.38
1	0	0	0.02
1	0	1	0.03
1	1	0	0.02
1	1	1	0.03



例 2.4　決定木は表よりも効率的に同時分布を表現できる．

2.3.2　連続同時分布

連続変数の同時分布を定義することもできる．どちらかといえば単純な分布は**多変量一様分布** (multivariate uniform distribution) で，台があるすべての値に定数の確率密度を割り当てるものである．**箱** (box) 上の一様分布を表現するために $\mathcal{U}(\boldsymbol{a},\boldsymbol{b})$ を用いる．箱とは，i 番目の区間が $[a_i,b_i]$ である区間のデカルト積である．この一様分布の族は**多変量積分布** (multivariate product distribution) の特別な型であり，一変量分布の積に関して定義される分布である．この場合，次のように表現される．

$$\mathcal{U}(\boldsymbol{x} \mid \boldsymbol{a},\boldsymbol{b}) = \prod_i \mathcal{U}(x_i \mid a_i,b_i) \tag{2.19}$$

分離はない．

F が与えられたときの条件付き独立である D と B に対しては，D から B へのすべての無向パスに沿った d 分離があるはずである．この場合，2 つのパスのどちらも d 分離はない．よって，条件付き独立性はネットワーク構造によって結論付けられない．

ノードの値が既知であるとき，X を他のすべてのノードから条件付き独立にするノードの最小の集合を呼ぶのに，ノード X の**マルコフブランケット** (Markov blanket) という用語がしばしば用いられる．特定のノードのマルコフブランケットは，結局その親，子，子の他の親で構成されていることがわかる．

2.7 要　約

- 不確実性を確率分布として表現することは，異なる言明のもっともらしさの比較に関連する一組の公理によって動機付けられる．
- 離散および連続確率分布の両方の族が多数ある．
- 連続確率分布は密度関数で表すことができる．
- 確率分布の族を組み合わせれば，より柔軟な分布を作成できる．
- 同時分布は複数の変数上の分布である．
- 条件付き分布は，証拠変数の値が与えられたときの 1 つ以上の変数上の分布である．
- ベイズネットワークは図的構造と条件付き分布の集合によって定義される．
- ベイズネットワークの構造によっては，条件付き独立性の仮定により，より少ないパラメータで同時分布を表現できる．

2.8 演　習

2.1　パラメータは λ で，非負の台をもち，密度 $p(x \mid \lambda) = \lambda \exp(-\lambda x)$ をもつ**指数分布** (exponential distribution) に従う連続確率変数 X を考えよう．X の累積分布関数を計算せよ．

【解】　累積分布関数の定義から始める．分布の台は $x = 0$ で下に有界なので，区間 $(-\infty, 0)$ には確率質量はなく，積分の下限を 0 にできる．積分を計算した後，次の $\mathrm{cdf}_X(x)$ を得る．

$$\mathrm{cdf}_X(x) = \int_{-\infty}^{x} p(x')\mathrm{d}x'$$

$$\mathrm{cdf}_X(x) = \int_{0}^{x} \lambda e^{-\lambda x'}\mathrm{d}x'$$

$$\mathrm{cdf}_X(x) = -e^{-\lambda x'}\Big|_{0}^{x}$$

$$\mathrm{cdf}_X(x) = 1 - e^{-\lambda x}$$

2.2　図 2.6 の密度関数に対して，混合物の 5 つの要素は何か (複数の有効な解がある)．

【解】　1 つの解は $\mathcal{U}([-10,-10],[-5,10])$，$\mathcal{U}([-5,0],[0,10])$，$\mathcal{U}([-5,-10],[0,0])$，$\mathcal{U}([0,-10],[10,5])$, $\mathcal{U}([0,5],[10,10])$ である．

3.5 計算複雑性

ベイズネットワークの推論が NP 困難であることは，3SAT という NP 完全問題を用いて示すことができる[4]．任意の 3SAT 問題からベイズネットワークを構成することは容易である．たとえば，次のような 3SAT の式を考えてみる[5]．

$$F(x_1,x_2,x_3,x_4) = \begin{pmatrix} x_1 & \vee & x_2 & \vee & x_3 \end{pmatrix} \wedge \begin{pmatrix} \neg x_1 & \vee & \neg x_2 & \vee & x_3 \end{pmatrix} \wedge \begin{pmatrix} x_2 & \vee & \neg x_3 & \vee & x_4 \end{pmatrix} \tag{3.14}$$

ここで，¬ は**論理否定** (logical negation) (“not”)，∧ は**論理積** (logical conjunction) (“and”)，∨ は**論理和** (logical disjunction) (“or”) を表す．この式は，**リテラル** (literal) と呼ばれるものの論理和である**節** (clause) の論理積で構成される．リテラルとは，単に変数もしくはその否定のことである．

図 3.4 は式 (3.14) を対応するベイズネットワークで表現したものである．変数は $X_{1:4}$ で表され，節は $C_{1:3}$ で表される．変数上の分布は一様である．節を表すノードは関連付けられた変数を親としてもつ．これは 3SAT であるから，各節のノードはそれぞれ正確に 3 つの親をもつ．各節のノードに対して，節を満たさない割当てには確率 0 を，節を満たす割当てには確率 1 を割り当てる．残りのノードはその親がすべて真であれば，確率 1 を割り当てる．$P(y') > 0$ であれば，またそのときに限り，元の問題は満足できる．したがって，ベイズネットワークにおける推論は，少なくとも 3SAT と同程度に難しい．

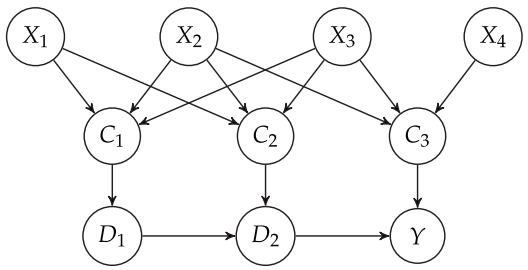


図 3.4　3SAT 問題を表現するベイズネットワーク

ここで，ベイズネットワークの推論が NP 困難であることを明示する理由は，すべてのベイズネットワークで機能する効率的で正確な推論アルゴリズムを探すのに時間を浪費しないようにするためである．過去数十年間の研究では，次に述べるような近似的な推論手法に焦点が当てられてきた．

3.6 直接サンプリング

現実的には正確な推論に関する計算が難しいため，多くの近似手法が開発されてきた．推論の最も単純な方法の 1 つは，**直接サンプリング** (direct sampling) に基づく方法で，同時分布からのランダムなサンプルに基づいて確率の推定を

[4] G. F. Cooper, “The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks,” *Artificial Intelligence,* vol. 42, no. 2–3, pp. 393–405, 1990. 本節におけるベイズネットワークの構築は，この文献に準じている．複雑さのクラスについては付録 C を参照.

[5] この式は，付録 C の例 C.3 にも掲載される.

す．ギブスサンプリングはアルゴリズム 3.10 により実装できる．

```
function blanket(bn, a, i)
    name = bn.vars[i].name
    val = a[name]
    a = delete!(copy(a), name)
    Φ = filter(ϕ → in_scope(name, ϕ), bn.factors)
    ϕ = prod(condition(ϕ, a) for ϕ in Φ)
    return normalize!(ϕ)
end
```

アルゴリズム 3.9　現在の割当て `a` が与えられたベイズネットワーク `bn` に対応する $P(X_i \mid x_{-i})$ を得るためのメソッド

```
function update_gibbs_sample!(a, bn, evidence, ordering)
    for i in ordering
        name = bn.vars[i].name
        if !haskey(evidence, name)
            b = blanket(bn, a, i)
            a[name] = rand(b)[name]
        end
    end
end

function gibbs_sample!(a, bn, evidence, ordering, m)
    for j in 1:m
        update_gibbs_sample!(a, bn, evidence, ordering)
    end
end

struct GibbsSampling
    m_samples # number of samples to use
    m_burnin  # number of samples to discard during burn-in
    m_skip    # number of samples to skip for thinning
    ordering  # array of variable indices
end

function infer(M::GibbsSampling, bn, query, evidence)
    table = FactorTable()
    a = merge(rand(bn), evidence)
    gibbs_sample!(a, bn, evidence, M.ordering, M.m_burnin)
    for i in 1:(M.m_samples)
        gibbs_sample!(a, bn, evidence, M.ordering, M.m_skip)
        b = select(a, query)
        table[b] = get(table, b, 0) + 1
    end
    vars = filter(v⟞v.name ∈ query, bn.vars)
    return normalize!(Factor(vars, table))
end
```

アルゴリズム 3.10　証拠 `evidence` と順序 `ordering` をもつベイズネットワーク `bn` に対して実装されたギブスサンプリング．本メソッドは `m` 回の反復に対して割当て `a` を繰り返し更新する．

ギブスサンプリングはこの実行例に適用することができる．m 個のサンプルを使って，次のように推定することができる．

$$P(b^1 \mid d^1, c^1) \approx \frac{1}{m} \sum_i (b^{(i)} = 1) \tag{3.21}$$

図 3.6 は，化学物質検出ネットワークにおける $P(c^1 \mid d^1)$ の推定値の収束を，直接サンプリング，尤度重み付きサンプリング，ギブスサンプリングで比較したものである．直接サンプリングは収束に最も時間がかかる．直接サンプリン

4.3 ノンパラメトリック学習

4.1, 4.2 節では，固定された形の確率モデルが事前に与えられ，関連付けられたパラメータをデータから学習することが仮定されていた．別の方法として，パラメータの数がデータ量に比例する**ノンパラメトリック** (nonparametric) 法がある．一般的なノンパラメトリック法は，**カーネル密度推定** (kernel density estimation) (アルゴリズム 4.3) である．観測値 $o_{1:m}$ が与えられると，カーネル密度推定により次式のように密度関数が計算される．

$$p(x) = \frac{1}{m}\sum_{i=1}^{m} \phi(x - o_i) \tag{4.35}$$

ここで，ϕ は積分すると 1 となる**カーネル関数** (kernel function) である．カーネル関数は，観測されたデータ点の近くの値により大きな密度を割り当てるために用いられる．一般に，カーネル関数は対称関数であり，$\phi(x) = \phi(-x)$ が成立する．平均がゼロであるガウス関数がカーネル関数として用いられることが多い．このようなカーネルが用いられる場合，標準偏差は**バンド幅** (bandwidth) と呼ばれ，密度関数の滑らかさを調整できる．一般に，バンド幅が大きいほど密度関数が滑らかになる．ベイズ法は，データに基づいて適切なバンド幅を選択するために適用できる．バンド幅の選択の効果を図 4.5 に示す．

```
gaussian_kernel(b) = x→pdf(Normal(0,b), x)

function kernel_density_estimate(ϕ, O)
    return x → sum([ϕ(x - o) for o in O])/length(O)
end
```

アルゴリズム 4.3　メソッド `gaussian_kernel` は，バンド幅 `b` のゼロ平均ガウスカーネル $\phi(x)$ を返す．カーネル `ϕ` と観測値のリスト `O` に対して，カーネル密度推定も実装されている．

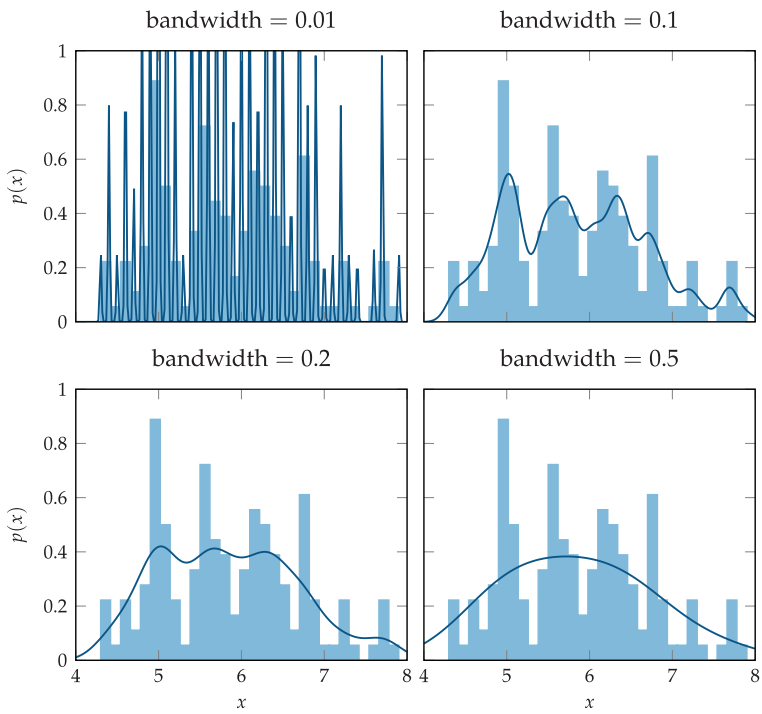


図 4.5　バンド幅が異なるゼロ平均ガウスカーネルを用いた，同じデータ集合に適用されるカーネル密度推定．青色のヒストグラムは基礎となるデータ集合の頻度を示し，黒い線はカーネル密度推定からの確率密度を示す．バンド幅が大きいと推定が平滑化されるが，バンド幅が小さいと特定のサンプルに過適合する可能性がある．

た．式 (6.2) は 6.2 節で与えたフォン=ノイマン–モルゲンシュテルンの公理から直接導かれるので，たとえ B と C を選択する多くの人々がこれらの公理に賛同しているように思えても，公理の少なくとも 1 つを違反しなければならない．

例 6.7 では，人々は損失あるいは利益として示されるかどうかに基づいて選択肢に関する決定を行うという**フレーミング効果** (framing effect) を例示している．他の多くの認知バイアスによって，効用理論によって導かれる結果から逸脱する可能性がある[14]．意思決定支援システムを構築するために，人間のエキスパートから効用関数を引き出そうとするときには，特別に注意しなければならない．意思決定支援システムが推薦することは合理的かもしれないが，特定の状況では人間の選好を正確に反映していないかもしれない．

[14] いくつかの近年の書籍では人間の明らかな非合理性について論じている．D. Ariely, *Predictably Irrational: The Hidden Forces That Shape Our Decisions*. Harper, 2008. J. Lehrer, *How We Decide*. Houghton Mifflin, 2009.

Tversky と Kahneman は伝染病によって 600 人が死亡すると予想される仮説的なシナリオを用いて，**フレーミング効果**を実証した．彼らは学生に次の 2 つの結果を示した．

- E: 200 人の命が救われるだろう．
- F: 1/3 の確率で 600 人が救われ，2/3 の確率で誰も救われないだろう．

学生の大多数は F よりも E を選択した．次に，彼らは学生に次の結果の間の選択を要求した．

- G: 400 人が死ぬだろう．
- H: 1/3 の確率で誰も死なず，2/3 の確率で 600 人が死ぬだろう．

E が G と等価で，F が H と等価であるにもかかわらず，学生の大多数は G よりも H を選択した．この不整合性は質問の構成の仕方のためである．

例 6.7　フレーミング効果を実証する実験．A. Tversky and D. Kahneman, "The Framing of Decisions and the Psychology of Choice," *Science*, vol. 211, no. 4481, pp. 453–458, 1981.

6.8 要　約

- 合理的な意思決定は確率と効用理論を結び付ける．
- 効用関数の存在は合理的な選好に関する制約から導かれる．
- 合理的な意思決定は期待効用を最大化するものである．
- 意思決定問題は意思決定ネットワークを用いてモデル化され，意思決定ネットワークは行動と効用を含むベイズネットワークの拡張である．
- 単純な意思決定を解決することには，ベイズネットワークにおける推論が含まれ，またこのことは NP 困難である．
- 情報の価値は，新しい変数が観測されたときの期待効用の増分を計測する．
- 人は必ずしも合理的ではない．

6.9 演　習

6.1　有限の最大値 $\overline{u}$ と有限の最小値 $\underline{u}$ をもつ効用関数 u があるとする．同じ選好をもつ対応する正規化された効用関数 $\hat{u}$ はどのようなものか．

外乱の分散は期待効用に影響する．アルゴリズム 7.11 により，これを実装できる．例 7.4 は，線形ガウスダイナミクスの単純な問題に対するこの過程を示している．

```
struct LinearQuadraticProblem
    Ts # transition matrix with respect to state
    Ta # transition matrix with respect to action
    Rs # reward matrix with respect to state (negative semidefinite)
    Ra # reward matrix with respect to action (negative definite)
    h_max # horizon
end

function solve(𝒫::LinearQuadraticProblem)
    Ts, Ta, Rs, Ra, h_max = 𝒫.Ts, 𝒫.Ta, 𝒫.Rs, 𝒫.Ra, 𝒫.h_max
    V = zeros(size(Rs))
    πs = Any[s → zeros(size(Ta, 2))]
    for h in 2:h_max
        V = Ts'*(V - V*Ta*((Ta'*V*Ta + Ra) \ Ta'*V))*Ts + Rs
        L = -(Ta'*V*Ta + Ra) \ Ta' * V * Ts
        push!(πs, s → L*s)
    end
    return πs
end
```

アルゴリズム 7.11　行列 `Ts` および `Ta` によってパラメータ化された確率的線形ダイナミクスと，行列 `Rs` および `Ra` によってパラメータ化された二次関数型報酬を用いて，時間区間 `h_max` ステップのマルコフ決定過程の最適方策を計算するメソッド．このメソッドは，要素 h が h ステップ方策で最適な最初の行動を生成する方策ベクトルを返す．

状態がスカラ値の位置と速度 $s = [x, v]$ で構成される連続マルコフ決定過程を考える．各時間ステップ $\Delta t = 1$ においてスカラ値の加速度 a を，行動として選択して実行する．次の二次関数型報酬が与えられた場合，$s_0 = [-10, 0]$ から最適な 5 ステップ方策を発見する．

$$R(\boldsymbol{s}, a) = -x^2 - v^2 - 0.5a^2$$

システムは $\boldsymbol{s} = \boldsymbol{0}$ で静止する傾向がある．

遷移ダイナミクスは次式で与えられる．

$$\begin{bmatrix} x' \\ v' \end{bmatrix} = \begin{bmatrix} x + v\Delta t + \frac{1}{2}a\Delta t^2 + w_1 \\ v + a\Delta t + w_2 \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0.5\Delta t^2 \\ \Delta t \end{bmatrix} \begin{bmatrix} a \end{bmatrix} \boldsymbol{w}$$

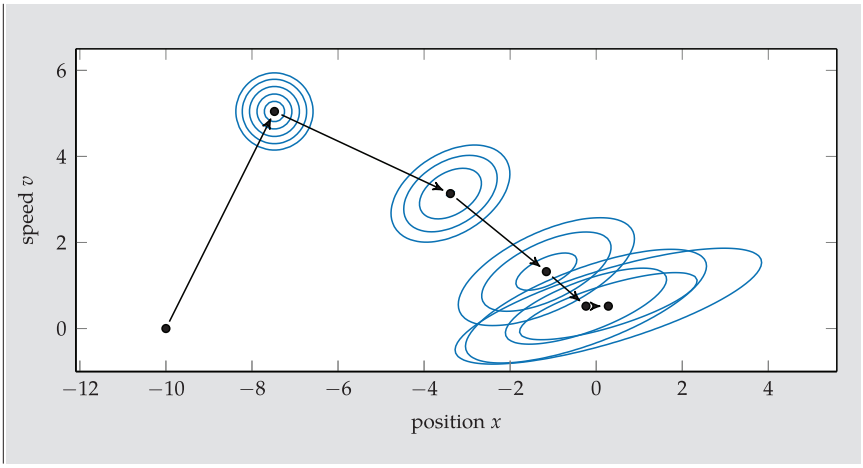
ここで，$\boldsymbol{w}$ は共分散 $0.1\boldsymbol{I}$ の平均ゼロの多変量ガウス分布から得られる．

報酬行列は，$\boldsymbol{R}_s = -\boldsymbol{I}$ および $\boldsymbol{R}_a = -[0.5]$ である．

結果として得られた最適方策は以下の通り．

$$\begin{aligned}
\pi_1(\boldsymbol{s}) &= \begin{bmatrix} 0 & 0 \end{bmatrix} \boldsymbol{s} \\
\pi_2(\boldsymbol{s}) &= \begin{bmatrix} -0.286 & -0.857 \end{bmatrix} \boldsymbol{s} \\
\pi_3(\boldsymbol{s}) &= \begin{bmatrix} -0.462 & -1.077 \end{bmatrix} \boldsymbol{s} \\
\pi_4(\boldsymbol{s}) &= \begin{bmatrix} -0.499 & -1.118 \end{bmatrix} \boldsymbol{s} \\
\pi_5(\boldsymbol{s}) &= \begin{bmatrix} -0.504 & -1.124 \end{bmatrix} \boldsymbol{s}
\end{aligned}$$

例 7.4　線形遷移関数と二次関数型報酬をもつ有限時間区間マルコフ決定過程を解く．この図は，$[-10, 0]$ を初期位置とするシステムの進行を示している．青い等高線は，各反復における状態のガウス分布を示している．最初の信念は円形だが，カルマンフィルタを用いた信念の更新により，形状が非円形に歪んでいる．



7.9 要　約

- 有界の報酬をもつ離散マルコフ決定過程は，動的計画法を用いて厳密に解くことができる．
- このような問題の方策評価は，逆行列による厳密解法，あるいは反復アルゴリズムによる近似解法により求解できる．
- 方策評価と方策改善を繰り返すことで，方策反復は最適方策を導くことができる．
- 価値反復と非同期価値反復は，価値関数を直接的に反復することで計算量を削減できる．
- 最適方策を見つける問題は，線形計画として定式化できるため，多項式時間で解くことができる．
- 線形遷移関数と二次関数型報酬をもつ連続問題は，厳密に解くことができる．

7.10 演　習

7.1　一定の報酬 (すべての t に対して $r_t = r$) がある，定数報酬の無限列に対して，無限時間区間の割引報酬が $r/(1-\gamma)$ に収束することを示せ．

【解】　次の手順で，割引定数報酬の無限列が $r/(1-\gamma)$ に収束することを証明できる．

$$\sum_{t=1}^{\infty} \gamma^{t-1} r_t = r + \gamma^1 r + \gamma^2 r + \cdots$$

$$= r + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} r_t$$

総和を左辺に移動すれば，$(1-\gamma)$ を取り除ける．

$$(1-\gamma) \sum_{t=1}^{\infty} \gamma^{t-1} r = r$$

$$\sum_{t=1}^{\infty} \gamma^{t-1} r = \frac{r}{1-\gamma}$$

7.2　5 つの状態 $s_{1:5}$ と，「止める」 (a_S) と「進める」 (a_C) の 2 種類の行動で構成さ

8.2 最 近 傍

局所的近似に対する単純な方法は，状態 s の**最近傍** (nearest neighbor) である $\mathcal{S}$ 内の状態の価値を用いることである．この方法を用いるためには，**距離尺度** (distance metric) (付録 A.3 参照) が必要である．2 つの状態 s と s' の間の距離を表すために，$d(s,s')$ を用いる．近似価値関数は，$\mathcal{U}_{\boldsymbol{\theta}}(s) = \theta_i, i = \arg\min_{j \in 1:m} d(s_j, s)$ となる．図 8.1 は，最近傍法を用いて表現された価値関数の例を示している．

この方法を一般化することで，***k*-最近傍** (k-nearest neighbors) の価値を平均化できる．このような方法でも，価値関数は階段状となるが，k の値によってはより良い近似が得られることがある．図 8.1 は，いくつかの k の値で近似された価値関数の例を示している．アルゴリズム 8.2 によりこれらを実装できる．

```
mutable struct NearestNeighborValueFunction
    k # number of neighbors
    d # distance function d(s,s')
    S # set of discrete states
    θ # vector of values at states in S
end

function (U0::NearestNeighborValueFunction)(s)
    dists = [U0.d(s,s') for s' in U0.S]
    ind = sortperm(dists)[1:U0.k]
    return mean(U0.θ[i] for i in ind)
end

function fit!(U0::NearestNeighborValueFunction, S, U)
    U0.θ = U
    return U0
end
```

アルゴリズム 8.2　距離関数 `d` によって定義される，`S` 内の `k` 個の最近傍状態に基づいて状態 `s` の価値を近似する k 最近傍法．ベクトル `θ` には，`S` の状態の価値が含まれる．`NearestNeighbors.jl` に実装されている kd ツリーなどの特殊なデータ構造を用いると，効率が向上する．

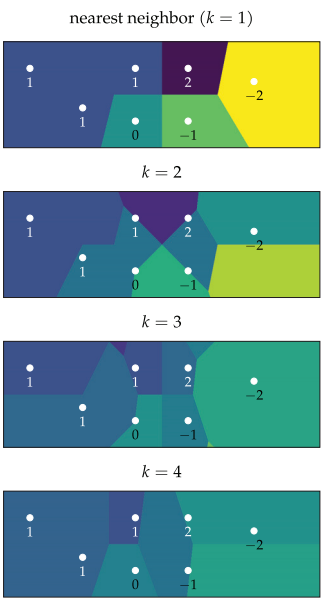


図 8.1　ユークリッド距離に従って k-最近傍の効用の平均を用いた，二次元の連続状態空間内の状態価値の近似．結果の価値関数は区分定数となる．

8.3 カーネル平滑化

もう 1 つの局所的近似法は**カーネル平滑化** (kernel smoothing) で，状態 $\mathcal{S}$ 内の効用が全体の状態空間にわたって平滑化される．この方法では，状態のペア s と s' を関連付ける**カーネル関数** (kernel function) $k(s,s')$ を定義する必要がある．一般に，$k(s,s')$ は，より近い状態に対して大きくなるべきである．なぜなら，それらの値が $\mathcal{S}$ 内の状態に関連付けられた効用をどのように重み付けするかを決定するからである．この方法により，次のように線形近似される：

$$\mathcal{U}_{\boldsymbol{\theta}}(s) = \sum_{i=1}^{m} \theta_i \beta_i(s) = \boldsymbol{\theta}^\top \boldsymbol{\beta}(s) \tag{8.3}$$

ここで，

$$\beta_i(s) = \frac{k(s, s_i)}{\sum_{j=1}^{m} k(s, s_j)} \tag{8.4}$$

である．アルゴリズム 8.3 として，ここまでの実装を示す．

きないため，通常は勾配降下などの最適化手法を用いる必要がある．ニューラルネットワークの勾配は，導関数に対するチェインルールを直接適用することで正確に計算できる．

8.8 要　約

- 大規模または逐次的な問題の場合，価値関数のパラメータ化されたモデルによって表される近似方策を発見できる．
- 本章で紹介した方法は，有限の状態集合で動的計画を反復して適用し，パラメトリックな近似を精緻化するものである．
- 局所的近似法は，既知の価値をもつ近傍の状態に基づいて価値関数を近似する．
- 局所的近似法として，最近傍法，カーネル平滑化，線形補完，シンプレックス補完などがある．
- 大域的近似技術として，線形回帰やニューラルネットワーク回帰がある．
- 非線形効用関数は，非線形基底関数の適切な選択と組み合わせて線形回帰を用いて得ることができる．
- ニューラルネットワーク回帰は基底関数を指定する必要はないが，それらを適合させるのはより複雑で，通常は価値関数のパラメトリックな近似を調整するために勾配降下を用いる．

8.9 演　習

8.1　本章で紹介する価値関数の近似は，ほとんどが連続状態空間を想定している．六角世界問題 (付録 F.1) は離散的であるが，その状態のほとんどは二次元の位置に写像される．ただし，二次元の位置をもたない，報酬をゼロにする終了状態もある．このような状態を処理するには，本章の連続価値関数の近似方法をどのように変更すればよいか．

【解】　六角世界問題では，エージェントは二次元の六角形のグリッド内を移動する．ただし，エージェントは複数の六角グリッドの 1 つから単一の終端状態に移動することができる．単一の終端状態は，状態価値を推論するために近接性に依存することが多く，価値関数近似法にとって課題となる．

終端状態を他の状態と同じ状態空間に投影することができる場合もあるが，たとえ遠く離れていたとしても，終端状態の価値計算に近接性の形式を取り入れることになる．つまり，その終端状態がいくつかの六角グリッドからアクセスできるべきにもかかわらず，複数の先行状態に対して等距離であるべき状態に 1 つの特定の位置に固定することはバイアスを導入することになる．

代わりの方法の 1 つとして，終端状態を特別な状態として扱うことが考えられる．カーネル関数を変更して，終端状態と他の任意の状態との間の距離を無限にすることができる．

別の代わりの方法としては，終端報酬を生成するすべての六角グリッドに対して終端状態をもつように問題を調整することが考えられる．各終端状態は，それに先行す

$$\begin{aligned} \underset{a_{1:d}, s_{2:d}}{\text{maximize}} \quad & \sum_{t=1}^{d} \gamma^t R(s_t, a_t) \\ \text{subject to} \quad & s_{t+1} = T(s_t, a_t), t \in 1:d-1 \end{aligned} \tag{9.4}$$

ここで，s_1 は現在の状態であり，$T(s,a)$ は決定論的な遷移関数，すなわち，状態 s において行動 a をとった結果，遷移する状態を表す．確率的な遷移関数から適切な決定論的な遷移関数を生成する方法として，最も確率の高い状態に遷移すると仮定することが考えられる．式 (9.4) のダイナミクスが線形であり，報酬関数が凸である場合，問題は凸である．

例 9.10 は，障害物を回避しながら目標状態に移動し，加速度の変化を最小化することを目的とした問題である．状態空間と行動空間は連続的であり，解を 1 秒未満で発見できる．各ステップの後に再計画することで，確率的な要素や予期しない事象に対応することができる．たとえば，障害物が移動した場合，計画を再調整することができる．図 9.9 に示されているように，再計画により対応できる．

本問題では，状態 $\boldsymbol{s}$ はエージェントの 2 次元位置と 2 次元速度ベクトルを結合したものであり，$\boldsymbol{s}$ は最初に $[0,0,0,0]$ と設定される．加速度ベクトルを行動 $\boldsymbol{a}$ として，各要素は ± 1 の区間で定義される．各ステップでは，行動に基づいて速度を更新し，速度に基づいて位置を更新する．目標状態 $\boldsymbol{s}_{\text{goal}} = [10,10,0,0]$ に到達することが目的である．割引は考慮せず，最大で $d = 10$ ステップまで計画する．各ステップでは，加速度の変化を最小化するため，$||\boldsymbol{a}_t||_2^2$ のコストを蓄積していく．最後のステップにおいて目標状態にできるだけ近づくことが望ましく，目標状態からの距離に基づくペナルティ $100||\boldsymbol{s}_d - \boldsymbol{s}_{\text{goal}}||_2^2$ を付ける．また，中心が $[3,4]$ で半径 2 の円形の障害物を回避する必要がある．この問題を以下のように定式化し，計画から最初の行動を抽出することができる．

```
model = Model(Ipopt.Optimizer)
d = 10
current_state = zeros(4)
goal = [10,10,0,0]
obstacle = [3,4]
@variables model begin
    s[1:4, 1:d]
    -1 ≤ a[1:2,1:d] ≤ 1
end
# velocity update
@constraint(model, [i=2:d,j=1:2], s[2+j,i] == s[2+j,i-1] + a[j,i-1])
# position update
@constraint(model, [i=2:d,j=1:2], s[j,i] == s[j,i-1] + s[2+j,i-1])
# initial condition
@constraint(model, s[:,1] .== current_state)
# obstacle
@constraint(model, [i=1:d], sum((s[1:2,i] - obstacle).^2) ≥ 4)
@objective(model, Min, 100*sum((s[:,d] - goal).^2) + sum(a.^2))
optimize!(model)
action = value.(a[:,1])
```

例 9.10　決定論的な環境での開ループ計画．円形の障害物を回避する経路を見つけることを目的とする．この実装では，`JuMP.jl` インタフェースを用いて Ipopt ソルバにアクセスしている．A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2005.

9.10 要　約

- オンライン手法は現在の状態から計画を立て，到達できる状態に関して集中的に計算する．
- 後退時間区間計画はある時間区間まで計画を立て，それぞれのステップで再計画を行う．
- ロールアウトを用いた先読みは，ロールアウト方策のシミュレーションによって推定された価値に基づいて貪欲に行動する．他のアルゴリズムと比較して計算効率が良いが，最適性に関するパフォーマンスは保証されない．
- 前方探索は，ある深さまでのすべての状態–行動の遷移の組を考慮し，状態の数と行動の数の両方に関して，計算量が指数的に増加する．
- 分枝限定法は，期待価値でより良い結果につながらない探索ツリーの一部を枝刈りするために，上界関数と下界関数を用いる．
- スパースサンプリングは，各探索ノードからのサンプリングされる遷移の数を制限することで，状態数の指数的な増大を回避できる．
- モンテカルロツリー探索は，探索と知識活用のバランスをとる行動を選択することで，探索空間の有望な領域に探索を誘導する．
- ヒューリスティック探索は，先読みを用いる経路に沿って更新される価値関数に対して貪欲方策のシミュレーションを実行する．
- ラベル付きヒューリスティック探索は，価値が収束した状態を再評価しないことによって計算量を削減する．
- 開ループ計画は，可能な限り最良行動列の発見を目指す．最適化問題が凸であれば計算効率を改善できる．

9.11 演　習

9.1　分枝限定法の計算量が最悪の場合，前方探索と同じになる理由を述べよ．

【解】　最悪の場合，分枝限定法は枝刈りされず，その結果として同じ計算量の前方探索と同じ探索ツリーをたどることになる．

9.2　2つの許容的ヒューリスティック h_1 と h_2 が与えられた場合，ヒューリスティック探索で両方を用いるにはどうすればよいか．

【解】　新しいヒューリスティック関数 $h(s) = \min\{h_1(s), h_2(s)\}$ を作成し，これを代わりに用いる．この新しいヒューリスティック関数は，必ず許容的であり h_1 または h_2 よりも悪い上下限になることはない．$h_1(s) \geq \mathcal{U}^*(s)$ および $h_2(s) \geq \mathcal{U}^*(s)$ の両方が成り立つ場合，$h(s) \geq \mathcal{U}^*(s)$ となる．

9.3　2つの許容的でないヒューリスティック h_1 と h_2 が与えられた場合，ヒューリスティック探索で両方を用いる方法を説明せよ．

【解】　新たなヒューリスティック関数として $h_3(s) = \max\{h_1(s), h_2(s)\}$ を定義することで，許容的，または「許容的でない度合いが低くなる」となる可能性のあるヒューリスティックを得ることができる．この新しいヒューリスティックは収束するまでに時間がかかるかもしれないが，より良い解を見逃さない可能性もある．

$$\nabla_{\boldsymbol{\psi}} \mathop{\mathbb{E}}_{\theta \sim p(\cdot \mid \boldsymbol{\psi})}[U(\theta)] \approx \sum_{i=1}^{m} w^{(i)} \nabla_{\boldsymbol{\psi}} \log p(\theta^{(i)} \mid \boldsymbol{\psi}) \tag{10.12}$$

一般的な重み付けの方法は次式により与えられる[12].

$$w^{(i)} = \frac{\max\left(0, \log(\frac{m}{2}+1) - \log(i)\right)}{\sum_{j=1}^{m} \max\left(0, \log(\frac{m}{2}+1) - \log(j)\right)} - \frac{1}{m} \tag{10.13}$$

これらの重みは図 10.6 に示されており，良好なサンプルを優先するため，ほとんどのサンプルに小さな負の重みを与えることになる．ランク整形により外れ値の影響を軽減することができる．

[12] N. Hansen and A. Ostermeier, "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation," in *IEEE International Conference on Evolutionary Computation*, 1996.

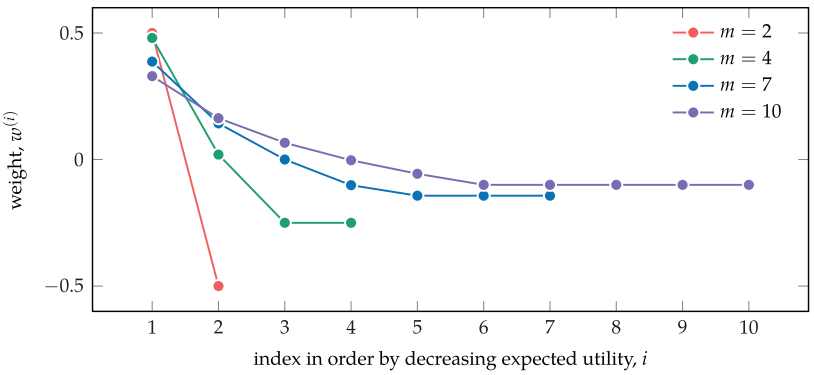


図 10.6　式 (10.13) を用いて構築された複数の重み付け

アルゴリズム 10.5 は進化戦略法の実装である．図 10.7 は探索過程の例を示している．

```
struct EvolutionStrategies
    D     # distribution constuctor
    ψ     # initial distribution parameterization
    ∇logp # log search likelihood gradient
    m     # number of samples
    α     # step factor
    k_max # number of iterations
end

function evolution_strategy_weights(m)
    ws = [max(0, log(m/2+1) - log(i)) for i in 1:m]
    ws ./= sum(ws)
    ws .-= 1/m
    return ws
end

function optimize_dist(M::EvolutionStrategies, π, U)
    D, ψ, m, ∇logp, α = M.D, M.ψ, M.m, M.∇logp, M.α
    ws = evolution_strategy_weights(m)
    for k in 1:M.k_max
        θs = rand(D(ψ), m)
        us = [U(π, θs[:,i]) for i in 1:m]
        sp = sortperm(us, rev=true)
        ∇ = sum(w.*∇logp(ψ, θs[:,i]) for (w,i) in zip(ws,sp))
        ψ += α.*∇
    end
```

アルゴリズム 10.5　方策 `π(θ,s)` に対する方策パラメータ化のための探索分布 `D(ψ)` を更新するための進化戦略法のメソッド．この実装では，初期の探索分布パラメータ化 `ψ`，対数探索尤度勾配 `∇logp(ψ, θ)`，方策評価関数 `U`，および反復回数 `k_max` を指定する必要がある．各反復では，`m` 個のパラメータ化サンプルが抽出され，探索勾配の推定に用いられる．この勾配はステップ係数 `α` とともに用いられる．`Distributions.jl` を用いて `D(ψ)` を定義することができる．たとえば，平均が `ψ` で固定の共分散行列 `Σ` をもつガウス分布を構築するには，`D(ψ) = MvNormal(ψ, Σ)` のように `D` を定義できる．

```
function optimize_dist(M::IsotropicEvolutionStrategies, π, U)
    ψ, σ, m, α, k_max = M.ψ, M.σ, M.m, M.α, M.k_max
    n = length(ψ)
    ws = evolution_strategy_weights(2*div(m,2))
    for k in 1:k_max
        εs = [randn(n) for i in 1:div(m,2)]
        append!(εs, -εs) # weight mirroring
        us = [U(π, ψ + σ.*ε) for ε in εs]
        sp = sortperm(us, rev=true)
        ∇ = sum(w.*εs[i] for (w,i) in zip(ws,sp)) / σ
        ψ += α.*∇
    end
    return MvNormal(ψ, σ)
end
```

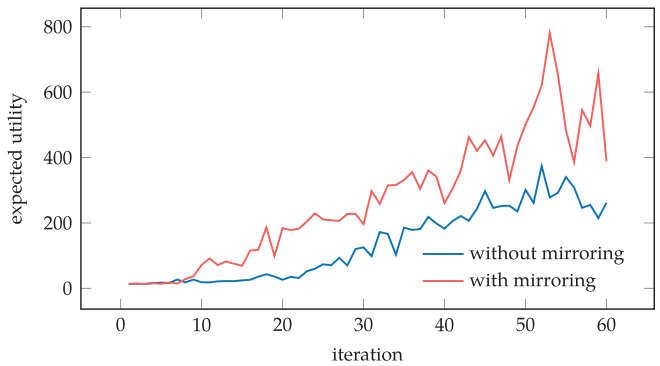


図 10.8　ミラードサンプリングが等方性進化戦略に与える影響．カートポール問題 (付録 F.3) で `m = 10`, `σ = 0.25` の二層ニューラルネットワーク方策を学習する．評価ごとに 6 つのロールアウトを行う．ミラードサンプリングは学習を大幅に高速化し，安定化させる．

10.7　要　約

- モンテカルロ方策評価は，初期状態分布からサンプリングされた状態を用いて大量のロールアウトを行い，方策に関連付けられた期待効用を計算する手法である．
- ホーク–ジーブス法などの局所探索法は，局所的な細かい変更を繰り返すことで方策を改善する．
- 遺伝的アルゴリズムは，パラメータ空間内における点の集合を維持しつつ，いくつかの手続きを用いて探索点の集合を再結合することで大域的最適解に近づけるような手法である．
- 交差エントロピー法は，各反復でエリートサンプルに適合するように探索分布を再調整し，方策パラメータ上の探索分布を繰り返し改善する．
- 進化戦略は，探索分布から抽出されたサンプルからの勾配情報に基づいて，探索分布を改善しようとする手法である．
- 等方性進化戦略は，探索分布が等方性ガウス分布であるという仮定を用いる．

```
struct LikelihoodRatioGradient
    𝒫  # problem
    b  # initial state distribution
    d  # depth
    m  # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::LikelihoodRatioGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇U(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)*R(τ)
    return mean(∇U(simulate(𝒫, rand(b), πθ, d)) for i in 1:m)
end
```

アルゴリズム 11.4　マルコフ決定過程 $\mathcal{P}$ における方策 `π(s)` の方策勾配を推定するためのメソッド．初期状態分布を `b` とし，尤度比トリックを用いる．パラメータ化ベクトル `θ` に関する勾配を，深さ `d` の `m` 回のロールアウトからの対数方策勾配 `∇logπ` を用いて推定する．

例 11.1 からの 1 ステップ 1 状態の問題を考える．ガウス分布 $\mathcal{N}(\theta_1, \theta_2^2)$ に従って，行動をサンプリングする確率的な方策 π_θ があるとする．ここで，θ_2^2 は分散である．

$$\begin{aligned}\log \pi_{\boldsymbol{\theta}}(a \mid s) &= \log\left(\frac{1}{\sqrt{2\pi\theta_2^2}} \exp\left(-\frac{(a-\theta_1)^2}{2\theta_2^2}\right)\right) \\ &= -\frac{(a-\theta_1)^2}{2\theta_2^2} - \frac{1}{2}\log\left(2\pi\theta_2^2\right)\end{aligned}$$

方策の対数尤度の勾配は次のようになる．

$$\begin{aligned}\frac{\partial}{\partial\theta_1}\log \pi_{\boldsymbol{\theta}}(a \mid s) &= \frac{a-\theta_1}{\theta_2^2} \\ \frac{\partial}{\partial\theta_2}\log \pi_{\boldsymbol{\theta}}(a \mid s) &= \frac{(a-\theta_1)^2 - \theta_2^2}{\theta_2^3}\end{aligned}$$

$\boldsymbol{\theta} = [0,1]$ で 3 回のロールアウトを実施し，行動 $\{0.5, -1, 0.7\}$ をとり，同じ報酬 $(R(s,a) = a)$ を受け取るとする．推定された方策勾配は次のようになる．

$$\begin{aligned}\nabla U(\boldsymbol{\theta}) &\approx \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}\left(\tau^{(i)}\right) R\left(\tau^{(i)}\right) \\ &= \frac{1}{3}\left(\begin{bmatrix}0.5\\-0.75\end{bmatrix}0.5 + \begin{bmatrix}-1.0\\0.0\end{bmatrix}(-1) + \begin{bmatrix}0.7\\-0.51\end{bmatrix}0.7\right) \\ &= [0.58, -0.244]\end{aligned}$$

例 11.3　単純な問題において尤度比トリックを用いて方策勾配を推定する

11.4　未来報酬

尤度比方策勾配法はバイアスの影響は受けないが，分散が高くなる傾向がある．例 11.4 では，バイアスと分散について述べる．一般に，ロールアウトの深さが増すと，時間ステップが進むにつれて行動，状態，報酬の相関が高くなり，分散が大きく増加する．未来報酬の方法は，推定値の分散を減らすために導入

勾配に対する貢献は後続の時間ステップに依存すべきではない．次のように，他の因果関係に反する項を削除できる[12]．

$$\nabla U(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\begin{array}{r} f^{(1)} r^{(1)} + f^{(1)} r^{(2)} \gamma + f^{(1)} r^{(3)} \gamma^2 + \cdots + f^{(1)} r^{(d)} \gamma^{d-1} \\ + f^{(2)} r^{(2)} \gamma + f^{(2)} r^{(3)} \gamma^2 + \cdots + f^{(2)} r^{(d)} \gamma^{d-1} \\ + f^{(3)} r^{(3)} \gamma^2 + \cdots + f^{(3)} r^{(d)} \gamma^{d-1} \\ \vdots \\ + f^{(d)} r^{(d)} \gamma^{d-1} \end{array} \right] \tag{11.23}$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \left(\sum_{\ell=k}^{d} r^{(\ell)} \gamma^{\ell-1} \right) \right] \tag{11.24}$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \left(\gamma^{k-1} \sum_{\ell=k}^{d} r^{(\ell)} \gamma^{\ell-k} \right) \right] \tag{11.25}$$

$$= \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \gamma^{k-1} r^{(k)}_{\text{to-go}} \right] \tag{11.26}$$

アルゴリズム 11.5 はこれを実装する．

$\boldsymbol{\theta}$ でパラメータ化された方策における状態–行動のペア (s,a) に対する未来報酬は，その状態からの状態行動価値関数 $Q_{\boldsymbol{\theta}}(s,a)$ の近似値であるということに注意する．行動価値関数が既知である場合，それを用いて方策勾配を得ることができる．

$$\nabla U(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \gamma^{k-1} Q_{\boldsymbol{\theta}} \left(s^{(k)}, a^{(k)} \right) \right] \tag{11.27}$$

```
struct RewardToGoGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::RewardToGoGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    ∇U(τ) = sum(∇logπ(θ, a, s)*R(τ,j) for (j, (s,a,r)) in enumerate(
     τ))
    return mean(∇U(simulate(𝒫, rand(b), πθ, d)) for i in 1:m)
end
```

アルゴリズム 11.5　初期状態分布が `b` であるマルコフ決定過程 $\mathcal{P}$ の方策 `π(s)` の方策勾配を推定するための，未来報酬を用いるメソッド．パラメータ化ベクトル `θ` に対する勾配は，`m` 回の深さ `d` のロールアウトにより，対数方策勾配 `∇logπ` を用いて推定される．

11.5　基準値減算

前節で示した手法をさらに発展させる．未来報酬[13] から**基準値** (baseline) の値を減算することで，勾配の分散を減らすことができる．この減算により，勾配にバイアスはかからない．次のように基準値 $r_{\text{base}}(s^{(k)})$ を減算する．

$$U(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \gamma^{k-1} \left(r^{(k)}_{\text{to-go}} - r_{\text{base}} \left(s^{(k)} \right) \right) \right] \tag{11.28}$$

[12] 項 $\sum_{\ell=k}^{d} r^{(\ell)} \gamma^{\ell-k}$ は，ステップ k からの**未来報酬** (reward-to-go) と呼ばれることがある．

[13] 状態–行動価値から基準値を減算することもできる．

同じ結果が適用される．

分散を最小化するために，勾配の各要素ごとに異なる $r_{\text{base}}(s)$ を選択する．単純化のため，s に関する依存関係を無視して各基準値の要素を定数として取り扱う[14]．導出過程を簡潔に記述するため，以下を定義する．

$$\ell_i(a,s,k) = \gamma^{k-1} \frac{\partial}{\partial \theta_i} \log \pi_{\boldsymbol{\theta}}(a \mid s) \tag{11.39}$$

式 (11.28) の勾配推定における i 番目の要素の分散は次の通り．

$$\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\left(\ell_i(a,s,k)\left(r_{\text{to-go}} - r_{\text{base},i}\right)\right)^2\right] - \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a,s,k)\left(r_{\text{to-go}} - r_{\text{base},i}\right)\right]^2 \tag{11.40}$$

ここで，期待価値は軌跡のサンプルに含まれる 3 項組 $(a,s,r_{\text{to-go}})$ に関するものであり，k は各組の深さを表す．

先ほど示したように，2 つ目の項は 0 である．したがって，1 つ目の項を最小化するために $r_{\text{base},i}$ を選ぶことに焦点を当てることができる．それには，基準値に関する導関数を 0 とすればよい．

$$\begin{aligned}
&\frac{\partial}{\partial r_{\text{base},i}} \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\left(\ell_i(a,s,k)(r_{\text{to-go}} - r_{\text{base},i})\right)^2\right] \\
&\quad = \frac{\partial}{\partial r_{\text{base},i}} \left(\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a,s,k)^2 r_{\text{to-go}}^2\right] \right. \\
&\qquad \left. -2 \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a,s,k)^2 r_{\text{to-go}} r_{\text{base},i}\right] + r_{\text{base},i}^2 \mathop{\mathbb{E}}_{a,s,k}\left[\ell_i(a,s,k)^2\right] \right) && (11.41) \\
&\quad = -2 \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a,s,k)^2 r_{\text{to-go}}\right] + 2 r_{\text{base},i} \mathop{\mathbb{E}}_{a,s,k}\left[\ell_i(a,s,k)^2\right] = 0 && (11.42)
\end{aligned}$$

$r_{\text{base},i}$ に対して解くと，次のように分散を最小化する基準値の要素が得られる．

$$r_{\text{base},i} = \frac{\mathbb{E}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a,s,k)^2 r_{\text{to-go}}\right]}{\mathbb{E}_{a,s,k}\left[\ell_i(a,s,k)^2\right]} \tag{11.43}$$

この基準値減算には，一般に尤度比方策勾配推定が用いられる (アルゴリズム 11.6)[15]．図 11.3 では，ここで議論された手法を比較している．

```
struct BaselineSubtractionGradient
    𝒫 # problem
    b  # initial state distribution
    d  # depth
    m  # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::BaselineSubtractionGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    ℓ(a, s, k) = ∇logπ(θ, a, s)*γ^(k-1)
    R(τ, k) = sum(r*γ^(j-1) for (j,(s,a,r)) in enumerate(τ[k:end]))
    numer(τ) = sum(ℓ(a,s,k).^2*R(τ,k) for (k,(s,a,r)) in enumerate(τ
    ))
    denom(τ) = sum(ℓ(a,s,k).^2 for (k,(s,a)) in enumerate(τ))
    base(τ) = numer(τ) ./ denom(τ)
```

アルゴリズム 11.6　未来報酬と基準値減算を用いたマルコフ決定過程 $\mathcal{P}$，方策 π，初期状態分布 b に対する尤度比勾配を推定するためのメソッド．パラメータ化ベクトル θ に対する勾配は，m 個のロールアウト (深さ d まで) から得られた対数方策勾配 ∇logπ を用いて推定される．

[14] 一部の手法では，$r_{\text{base}}(s^{(k)}) = \boldsymbol{\phi}(s^{(k)})^\top \boldsymbol{w}$ のように，状態に依存する基準値を近似する．適切な基準値関数を選択することが難しい場合がある．J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[15] この組合せは，以下の文献によって導入された**強化** (REINFORCE) と呼ばれるアルゴリズムのクラスで用いられる．R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

```
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    rbase = mean(base(τ) for τ in trajs)
    ∇U(τ) = sum(ℓ(a,s,k).*(R(τ,k).-rbase) for (k,(s,a,r)) in
     enumerate(τ))
    return mean(∇U(τ) for τ in trajs)
end
```

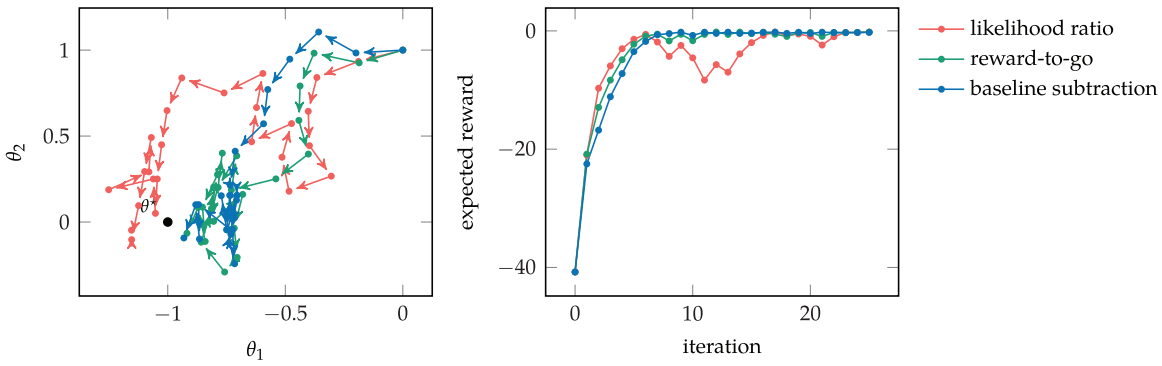


図 11.3　同じ初期パラメータから単純レギュレータ問題に対する方策を最適化するために用いられるいくつかの方策勾配法．各勾配の計算では，深さ 10 の 6 回のロールアウトを実行した．勾配の大きさは 1 に制限され，ステップ更新はステップサイズ 0.2 で適用された．最適な方策パラメータは黒で示されている．

質的には，状態–行動のペアの勾配寄与を考慮するときに興味をもつことは，1 つの行動がもう 1 つよりもどれだけ価値があるかということである．ある状態ですべての行動が同程度の価値をもつ場合，勾配に意味のある情報はなく，そして基準値減算でそれをゼロにすることができる．行動全体の平均に関係なく，他の行動よりも高い価値をもつ行動を識別したい．

行動価値の代わりに**アドバンテージ** (advantage) $A(s,a) = Q(s,a) - \mathcal{U}(s)$ を用いる方法もある．状態価値関数を基準値減算に用いることで，アドバンテージが得られる．アドバンテージを用いた方策勾配はバイアスがなく，通常は分散が低くなる．勾配の計算は以下のようになる．

$$\nabla \mathcal{U}(\boldsymbol{\theta}) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \left(a^{(k)} \mid s^{(k)} \right) \gamma^{k-1} A_{\boldsymbol{\theta}} \left(s^{(k)}, a^{(k)} \right) \right] \tag{11.44}$$

状態価値関数や行動価値関数と同様に，アドバンテージ関数も通常は未知である．それを近似するためには，13 章で説明される他の手法が必要となる．

11.6　要　約

- 有限差分法を用いて，勾配を推定することができる．
- 線形回帰を用いても，方策勾配のより安定した推定値を計算することができる．
- 尤度比を用いて，確率的方策の遷移モデルに依存しない方策勾配を導出することができる．
- 未来報酬と基準値減算を用いることで，方策勾配の分散を大幅に減らすことができる．

```
struct PolicyGradientUpdate
    ∇U # policy gradient estimate
    α  # step factor
end

function update(M::PolicyGradientUpdate, θ)
    return θ + M.α * M.∇U(θ)
end
```

アルゴリズム 12.1　方策最適化のための勾配上昇法．勾配`∇U`の方向にステップ係数`α`で点`θ`からステップをとる．`∇U`を計算するために前章のメソッドの 1 つを用いることができる．

非常に大きな勾配は最適値をオーバーシュートする傾向があり，それはさまざまな理由で発生する可能性がある．2048 問題 (付録 F.2) のように，ある種の問題に対する報酬は桁違いに異なる場合がある．勾配を管理しやすく保つための 1 つの方法は**勾配スケーリング** (gradient scaling) を用いることであり，これは方策のパラメータを更新するために，勾配を用いる前に勾配の推定値の大きさを制限することである．勾配は通常 1 の L_2 ノルムをもつように制限される．別の方法として，**勾配クリッピング** (gradient clipping) があり，これは方策を更新するために勾配を用いる前に，勾配の各要素をある範囲に留めることである．クリッピング操作では通常，要素を ± 1 の間に制限する．両方のメソッドともアルゴリズム 12.2 で実装されている．

```
scale_gradient(∇, L2_max) = min(L2_max/norm(∇), 1)*∇
clip_gradient(∇, a, b) = clamp.(∇, a, b)
```

アルゴリズム 12.2　勾配スケーリングと勾配クリッピング．勾配スケーリングは得られた勾配ベクトル`∇`の大きさを`L2_max`に制限する．勾配クリッピングは得られた勾配ベクトル`∇`を`a`と`b`の間に要素ごとに留める．

図 12.1 に示すように，スケーリングとクリッピングは最終的な勾配の方向に与える影響が異なる．スケーリングでは方向には影響を与えないが，クリッピングでは各要素に個別に影響を与える．この差が有利かどうかは問題によって異なる．たとえば，単一の要素が勾配ベクトルに大きな影響を与えるならば，スケーリングは他の要素をゼロにしてしまう．

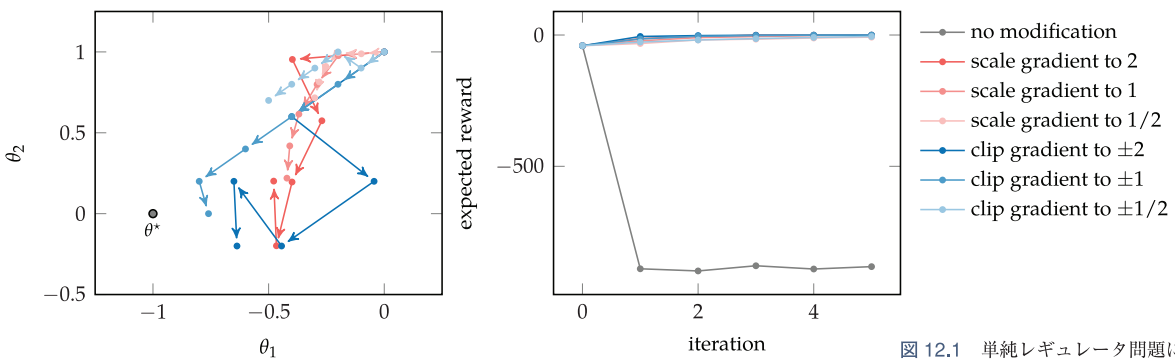


図 12.1　単純レギュレータ問題に勾配スケーリングとクリッピングが適用された効果．各勾配評価は深さ 10 まで 10 回ロールアウトした．ステップ更新は 0.2 のステップ幅で適用された．最適方策のパラメータは黒で表示されている．

12.2　制限された勾配更新

本章の残りのアルゴリズムは，次のステップでの方策パラメータ $\boldsymbol{\theta}'$ が現在のステップでの $\boldsymbol{\theta}$ から離れすぎないという制約のもとで，目的関数 $\mathcal{U}(\boldsymbol{\theta})$ の近

似を最適化しようと試みる．制約は $g(\boldsymbol{\theta}, \boldsymbol{\theta}') \leq \varepsilon$ の形式をとり，$\varepsilon > 0$ はアルゴリズムの自由パラメータである．方法ごとに，$\mathcal{U}(\boldsymbol{\theta})$ の近似と g の形が異なる．本節では，単純な**制限ステップ** (restricted step) 法について記述する．

$\mathcal{U}$ を近似するために，次のように $\boldsymbol{\theta}$ での勾配推定値から得られた一次のテイラー近似を用いる．

$$\mathcal{U}(\boldsymbol{\theta}') \approx \mathcal{U}(\boldsymbol{\theta}) + \nabla \mathcal{U}(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) \tag{12.2}$$

制約に対しては，

$$g(\boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \boldsymbol{I} (\boldsymbol{\theta}' - \boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|_2^2 \tag{12.3}$$

を用いる．この制約式によって，ステップ幅が $\sqrt{2\varepsilon}$ を超えないようにする．言い換えれば，最適化問題における実行可能領域は $\boldsymbol{\theta}$ を中心とする半径 $\sqrt{2\varepsilon}$ の球である．

このとき，最適化問題は

$$\begin{aligned} \underset{\boldsymbol{\theta}'}{\text{maximize}} \quad & \mathcal{U}(\boldsymbol{\theta}) + \nabla \mathcal{U}(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) \\ \text{subject to} \quad & \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \boldsymbol{I} (\boldsymbol{\theta}' - \boldsymbol{\theta}) \leq \varepsilon \end{aligned} \tag{12.4}$$

である．$\mathcal{U}(\boldsymbol{\theta})$ は $\boldsymbol{\theta}'$ に依存しないため，目的関数から削除できる．さらに，線形の目的関数では最適解が実行可能領域の境界上にあるように強制されるので，制約式の不等式を等式に変更できる．これらの変更を考慮すると，次の等価な最適化問題が得られる．

$$\begin{aligned} \underset{\boldsymbol{\theta}'}{\text{maximize}} \quad & \nabla \mathcal{U}(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) \\ \text{subject to} \quad & \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \boldsymbol{I} (\boldsymbol{\theta}' - \boldsymbol{\theta}) = \varepsilon \end{aligned} \tag{12.5}$$

この最適化問題は次のように解析的に解くことができる．

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{u}\sqrt{\frac{2\varepsilon}{\boldsymbol{u}^\top \boldsymbol{u}}} = \boldsymbol{\theta} + \sqrt{2\varepsilon}\frac{\boldsymbol{u}}{\|\boldsymbol{u}\|} \tag{12.6}$$

ここで，正規化していない探索方向 $\boldsymbol{u}$ は単純に $\nabla \mathcal{U}(\boldsymbol{\theta})$ である．もちろん，われわれは $\nabla \mathcal{U}(\boldsymbol{\theta})$ を正確には知らないが，勾配を推定するための前章で記述した方法のいずれかを用いることができる．アルゴリズム 12.3 では，1 つの実装を提供している．

```
struct RestrictedPolicyUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood
    π     # policy
    ε     # divergence bound
end

function update(M::RestrictedPolicyUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫
    .γ
```

アルゴリズム 12.3　初期状態分布 `b` をもつ問題 $\mathcal{P}$ に対する `θ` での制限された方策勾配法の更新関数．勾配は，対数方策勾配 `∇logπ` をもつパラメータ化された方策 `π(θ, s)` の `m` 回のシミュレーションを用いて，初期状態分布 `b` から深さ `d` まで推定される．

```
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    u = mean(∇U(τ) for τ in τs)
    return θ + u*sqrt(2*M.ε/dot(u,u))
end
```

12.3 自然勾配更新

自然勾配 (natural gradient) 法[2] は，パラメータ空間の一部の要素が他の要素よりも敏感である状況をより適切に処理するための，前節で論じた制限ステップ法の 1 つの変形版である．この文脈における**感度** (sensitivity) とは，パラメータの 1 つの小さな変化に対して方策の効用がどの程度変化するかを意味する．勾配法の感度は主として，方策パラメータのスケーリングの選択によって決定される．自然方策勾配法では，探索方向 $\boldsymbol{u}$ がパラメータスケーリングに対して変わらないようにする．図 12.2 では，真の勾配と自然勾配の違いを示している．

[2] S. Amari, "Natural Gradient Works Efficiently in Learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

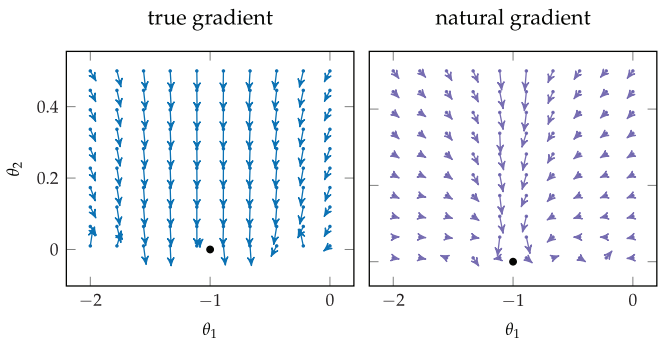


図 12.2 単純レギュレータ問題における真の勾配と自然勾配の比較 (付録 F.5 を参照)．真の勾配は一般に θ_2 の負方向へ強く向くが，自然勾配は一般に $[-1,0]$ で最適解 (黒い点) へ向く．同様な図が次の文献で示されている．J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

自然方策勾配法では，前節と同じ目的関数の一次近似を用いる．ただし，制約は異なる．直観的には，軌道上の分布に大きな変化をもたらす θ の変化を制限したいということである．分布がどの程度変化するかを測定する方法は，**カルバック–ライブラーダイバージェンス** (Kullback–Leibler divergence)，または KL ダイバージェンス (付録 A.10) を用いることである．次のような制約を課すこともできる．

$$g(\boldsymbol{\theta},\boldsymbol{\theta}') = D_{\mathrm{KL}}(p(\cdot \mid \theta) \parallel p(\cdot \mid \theta')) \leq \varepsilon \tag{12.7}$$

しかし，代わりに，二次のテイラー近似

$$g(\boldsymbol{\theta},\boldsymbol{\theta}') = \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{\theta}' - \boldsymbol{\theta}) \leq \varepsilon \tag{12.8}$$

を用いる．ここで，**フィッシャー情報行列** (Fisher information matrix) は次の形式である．

$$\boldsymbol{F}_{\boldsymbol{\theta}} = \int p(\tau \mid \theta) \nabla \log p(\tau \mid \theta) \nabla \log p(\tau \mid \theta)^\top \mathrm{d}\tau \tag{12.9}$$

$$= \mathbb{E}_\tau \left[\nabla \log p(\tau \mid \theta) \nabla \log p(\tau \mid \theta)^\top \right] \tag{12.10}$$

結果として得られた最適化問題は

$$\begin{aligned} \underset{\boldsymbol{\theta}'}{\text{maximize}} \quad & \nabla U(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) \\ \text{subject to} \quad & \frac{1}{2} (\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \boldsymbol{F}_{\boldsymbol{\theta}} (\boldsymbol{\theta}' - \boldsymbol{\theta}) = \varepsilon \end{aligned} \tag{12.11}$$

であり，これは単位行列 $\boldsymbol{I}$ の代わりにフィッシャー行列 $\boldsymbol{F}_{\boldsymbol{\theta}}$ を用いたことを除いて，式 (12.5) と同じように見える．この違いは楕円の実行可能集合となることである．図 12.3 に二次元の例を示す．

この最適化問題は解析的に解けて，次のように前節での更新式と同じ形式である．

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{u} \sqrt{\frac{2\varepsilon}{\nabla U(\boldsymbol{\theta})^\top \boldsymbol{u}}} \tag{12.12}$$

ただし，

$$\boldsymbol{u} = \boldsymbol{F}_{\boldsymbol{\theta}}^{-1} \nabla U(\boldsymbol{\theta}) \tag{12.13}$$

である[3]．サンプリングされた軌跡を用いて $\boldsymbol{F}_{\boldsymbol{\theta}}$ と $\nabla U(\boldsymbol{\theta})$ を推定できる．アルゴリズム 12.4 はこれの実装を提供している．

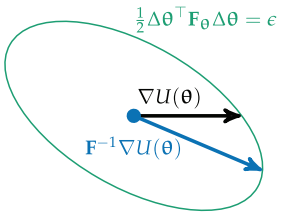


図 12.3　自然方策勾配では，近似されたカルバック–ライブラーダイバージェンスに関する制約が課されている．この制約は楕円の形をとる．楕円は特定の方向に引き延ばされ，勾配を回転させるとより大きなステップが可能になる．

3) この計算は共役勾配降下を用いて実行でき，$\boldsymbol{\theta}$ の次元が大きい場合に計算量が削減される．S. M. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems* (*NIPS*), 2001.

アルゴリズム 12.4　初期状態分布 `b` をもつマルコフ決定過程 $\mathcal{P}$ に対して，`π(θ, s)` が与えられたときの自然方策勾配に対する更新関数．対数方策勾配 `∇logπ` を用いて，パラメータベクトル `θ` に関する自然勾配は深さ `d` までの `m` 回のロールアウトによって推定される．軌跡のリストに対して目的関数の勾配 `∇f(τ)` とフィッシャー行列 `F(τ)` が与えられたとき，`natural_update` 法は式 (12.12) に従って更新する．

```
struct NaturalPolicyUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood
    π     # policy
    ε     # divergence bound
end

function natural_update(θ, ∇f, F, ε, τs)
    ∇fθ = mean(∇f(τ) for τ in τs)
    u = mean(F(τ) for τ in τs) \ ∇fθ
    return θ + u*sqrt(2ε/dot(∇fθ,u))
end

function update(M::NaturalPolicyUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫
    .γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    F(τ) = ∇log(τ)*∇log(τ)'
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return natural_update(θ, ∇U, F, M.ε, τs)
end
```

12.4　信頼領域の更新

本節では，前節の楕円形実行可能領域によって定義された**信頼領域** (trust region) 内で探索する方法について論じる．この方法の範疇は**信頼領域方策最適**

まで，

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \alpha(\boldsymbol{\theta}' - \boldsymbol{\theta}) \tag{12.20}$$

を繰り返し適用する．ステップ係数 $0 < \alpha < 1$ は各反復で $\boldsymbol{\theta}$ と $\boldsymbol{\theta}'$ の間の距離を縮め，通常 0.5 に設定される．アルゴリズム 12.5 はこの方法の実装を与える．図 12.4 では，自然勾配に関連付けられた実行可能領域と直線探索の関係を示している．図 12.5 はレギュレータ問題に対してこの方法を明示し，例 12.1 は単純な問題に対する更新を示している．

```
struct TrustRegionUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    π     # policy
    p     # policy likelihood p(θ, a, s)
    ∇logπ # log likelihood gradient
    KL    # KL divergence KL(θ, θ′, s)
    ε     # divergence bound
    α     # line search reduction factor (e.g., 0.5)
end

function surrogate_objective(M::TrustRegionUpdate, θ, θ′, τs)
    d, p, γ = M.d, M.p, M.𝒫.γ
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    w(a,s) = p(θ′,a,s) / p(θ,a,s)
    f(τ) = mean(w(a,s)*R(τ,k) for (k,(s,a,r)) in enumerate(τ))
    return mean(f(τ) for τ in τs)
end

function surrogate_constraint(M::TrustRegionUpdate, θ, θ′, τs)
    γ = M.𝒫.γ
    KL(τ) = mean(M.KL(θ, θ′, s)*γ^(k-1) for (k,(s,a,r)) in enumerate
     (τ))
    return mean(KL(τ) for τ in τs)
end

function linesearch(M::TrustRegionUpdate, f, g, θ, θ′)
    fθ = f(θ)
    while g(θ′) > M.ε || f(θ′) ≤ fθ
        θ′ = θ + M.α*(θ′ - θ)
    end
    return θ′
end

function update(M::TrustRegionUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫
     .γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    F(τ) = ∇log(τ)*∇log(τ)′
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    θ′ = natural_update(θ, ∇U, F, M.ε, τs)
    f(θ′) = surrogate_objective(M, θ, θ′, τs)
```

アルゴリズム 12.5　直線探索で自然勾配を増大させる信頼領域方策最適化に対する更新手続き．これは初期状態分布 `b` と深さ `d` で，問題 $\mathcal{P}$ において方策 `π` を用いて `m` 個の軌跡を生成する．直線探索の開始点を取得するために，現在の状態から特定の行動を生成する方策の対数確率の勾配を必要とし，これを `∇logπ` と記す．代理目的に関して，方策が現在の状態から特定の行動を生成する確率を与える確率関数 `p` が必要である．代理制約に関しては，$\pi_{\boldsymbol{\theta}}$ と $\pi_{\boldsymbol{\theta}'}$ によって生成される行動分布の間のダイバージェンスが必要である．直線探索の各ステップで，探索方向を維持しながら，点 `θ` と `θ′` の間の距離を縮めていく．

行動価値をもつクランプされていない目的関数 (12.21) の勾配は

$$\nabla_{\boldsymbol{\theta}'} f(\boldsymbol{\theta}, \boldsymbol{\theta}') = \mathop{\mathbb{E}}_{s \sim b_{\gamma,\boldsymbol{\theta}}} \left[\mathop{\mathbb{E}}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)} \left[\frac{\nabla_{\boldsymbol{\theta}'} \pi_{\boldsymbol{\theta}'}(a \mid s)}{\pi_{\boldsymbol{\theta}}(a \mid s)} Q_{\boldsymbol{\theta}}(s,a) \right] \right] \tag{12.23}$$

である．ここで，$Q_{\boldsymbol{\theta}}(s,a)$ は未来報酬から推定できる．(クランピングありの) 目的関数の下限 (12.22) の勾配は目的関数がアクティブにクランプされている経験タプルからの寄与がないことを除いて同じである．つまり，未来報酬が正で，確率の比が $1+\varepsilon$ よりも大きいか，未来報酬が負で，確率の比が $1-\varepsilon$ よりも小さければ，勾配への寄与は 0 である．

TRPO のように，勾配は $\boldsymbol{\theta}$ から生成された経験からのパラメータ $\boldsymbol{\theta}'$ に対して計算される．よって，いくつかの勾配の更新は同じサンプルされた軌跡の集合を用いて続けて実行される．アルゴリズム 12.6 に，これの実装を提供している．

```
struct ClampedSurrogateUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of trajectories
    π     # policy
    p     # policy likelihood
    ∇π    # policy likelihood gradient
    ε     # divergence bound
    α     # step size
    k_max # number of iterations per update
end

function clamped_gradient(M::ClampedSurrogateUpdate, θ, θ′, τs)
    d, p, ∇π, ε, γ = M.d, M.p, M.∇π, M.ε, M.𝒫.γ
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    ∇f(a,s,r_togo) = begin
        P = p(θ, a,s)
        w = p(θ′,a,s) / P
        if (r_togo > 0 && w > 1+ε) || (r_togo < 0 && w < 1-ε)
            return zeros(length(θ))
        end
        return ∇π(θ′, a, s) * r_togo / P
    end
    ∇f(τ) = mean(∇f(a,s,R(τ,k)) for (k,(s,a,r)) in enumerate(τ))
    return mean(∇f(τ) for τ in τs)
end

function update(M::ClampedSurrogateUpdate, θ)
    𝒫, b, d, m, π, α, k_max= M.𝒫, M.b, M.d, M.m, M.π, M.α, M.k_max
    πθ(s) = π(θ, s)
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    θ′ = copy(θ)
    for k in 1:k_max
        θ′ += α*clamped_gradient(M, θ, θ′, τs)
    end
    return θ′
end
```

アルゴリズム 12.6　初期状態分布 `b` をもつマルコフ決定過程 $\mathcal{P}$ の方策 `π(s)` に対する新しい方策パラメータを返すクランプされた代理方策最適化の実装．この実装では，深さ `d` まで `m` 個の軌跡をサンプリングし，`k_max` 回続く更新において方策勾配を推定するためにこれらを用いる．クランピングパラメータ `ε` をもつ方策勾配 `∇p` を用いて，クランプされた目的関数値を用いる方策勾配は構築される．

クランプされた代理目的はいくつかの他の代理目的と図 12.7 において比較される．その中には次の TRPO に関する効果的な目的に対する直線描画が含ま

れる．

$$\mathop{\mathbb{E}}_{\substack{s\sim b_{\gamma,\boldsymbol{\theta}}\\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\frac{\pi_{\boldsymbol{\theta}'}(a\mid s)}{\pi_{\boldsymbol{\theta}}(a\mid s)}A_{\boldsymbol{\theta}}(s,a)-\beta D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\cdot\mid s)\parallel\pi_{\boldsymbol{\theta}'}(\cdot\mid s))\right] \tag{12.24}$$

これは，制約が係数 β に関するペナルティとして実装された信頼領域方策目的関数である．複数の問題にわたるものだけでなく，単一の問題において良好なパフォーマンスを発揮する β の値を選択することは困難であるため，TRPO はペナルティよりもハードな制約を通常は使用する．

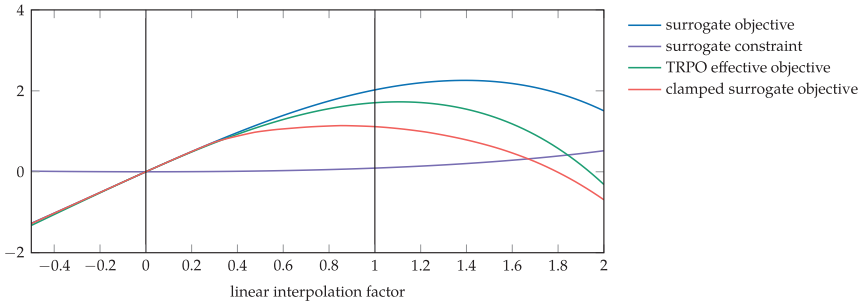


図 12.7　線形二次レギュレータ問題を用いた，クランプされた代理方策最適化に関連付けられた代理目的の比較．1 での自然方策更新を所与として，x 軸は 0 での $\boldsymbol{\theta}$ から $\boldsymbol{\theta}'$ に向かって移動するときの代理目的関数値を示している．$\boldsymbol{\theta}$ に関する代理目的関数値を差し引くことによって，代理目的の中心が 0 に設定された．クランプされた代理目的は，制約を必要としないで，効果的な TRPO の目的に非常に似た振る舞いをしている．ε と β は両方のアルゴリズムで調整され，それぞれの場合の最大値がどこになるかに影響することに注意しよう．

12.6　要　約

- 勾配上昇アルゴリズムは，前章で論じた手法から得られた勾配推定値を使用して，方策を反復的に改善できる．
- 勾配上昇法は，スケーリング，クリッピングまたは改善ステップの大きさをそろえることによって，より堅固なものにすることができる．
- 自然勾配アプローチでは，フィッシャー情報行列の推定値を使用して近似された，各ステップでの軌跡分布間のダイバージェンスに対する制約をもつ目的関数の一次近似を用いる．
- 信頼領域方策の最適化には，追加の軌跡シミュレーションを行わずに方策をさらに改善するために，直線探索による自然勾配法を補強することが含まれる．
- 直線探索の必要なしに，同じように機能するクランプされた代理目的を得るために，TRPO の目的関数の悲観的な下限を用いることができる．

12.7　演　習

12.1　TRPO は自然方策勾配更新によって与えられた新しいパラメータから直線探索を開始する．しかし，TRPO は自然方策勾配とは異なる目的関数値を用いて直線探索を行う．TRPO で用いられる代理目的 (12.18) の勾配が実際に未来報酬方策勾配 (11.26) と同じであることを示せ．

【解】　TRPO の代理目的の勾配は

$$\nabla_{\boldsymbol{\theta}'}\mathcal{U}_{\mathrm{TRPO}}=\mathop{\mathbb{E}}_{s\sim b_{\gamma,\boldsymbol{\theta}}}\left[\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\frac{\nabla_{\boldsymbol{\theta}'}\pi_{\boldsymbol{\theta}'}(a\mid s)}{\pi_{\boldsymbol{\theta}}(a\mid s)}Q_{\boldsymbol{\theta}}(s,a)\right]\right]$$

ことができる．

クリティックも勾配最適化によって更新される．次の損失関数を最小化する $\boldsymbol{\phi}$ を求める．

$$\ell(\boldsymbol{\phi}) = \frac{1}{2}\mathbb{E}_s\left[\left(\mathcal{U}_{\boldsymbol{\phi}}(s) - \mathcal{U}^{\pi_{\boldsymbol{\theta}}}(s)\right)^2\right] \tag{13.4}$$

この目的関数を最小化するために，勾配の逆方向にステップをとる．

$$\nabla\ell(\boldsymbol{\phi}) = \mathbb{E}_s[(\mathcal{U}_{\boldsymbol{\phi}}(s) - \mathcal{U}^{\pi_{\boldsymbol{\theta}}}(s))\nabla_{\boldsymbol{\phi}}\mathcal{U}_{\boldsymbol{\phi}}(s)] \tag{13.5}$$

当然，$\mathcal{U}^{\pi_{\boldsymbol{\theta}}}$ は正確にはわからないが，ロールアウトの軌跡に沿った未来報酬に基づいて推定することができる．これにより，以下の結果を得る．

$$\nabla\ell(\boldsymbol{\phi}) = \mathbb{E}_\tau\left[\sum_{k=1}^{d}\left(\mathcal{U}_{\boldsymbol{\phi}}(s^{(k)}) - r_{\text{to-go}}^{(k)}\right)\nabla_{\boldsymbol{\phi}}\mathcal{U}_{\boldsymbol{\phi}}(s^{(k)})\right] \tag{13.6}$$

ここで，$r_{\text{to-go}}^{(k)}$ は特定の軌跡 τ のステップ k での未来報酬である．

アルゴリズム 13.1 では，ロールアウトから $\nabla\mathcal{U}(\boldsymbol{\theta})$ と $\nabla\ell(\boldsymbol{\phi})$ を推定する方法が示される．各反復において，効用を最大化するために $\boldsymbol{\theta}$ を $\nabla\mathcal{U}(\boldsymbol{\theta})$ の方向に移動させ，損失関数を最小化するために $\boldsymbol{\phi}$ を $\nabla\ell(\boldsymbol{\phi})$ の逆方向に移動させる．この方法は，$\boldsymbol{\theta}$ と $\boldsymbol{\phi}$ の推定の依存関係により不安定になる可能性があるが，さまざまな問題でうまく機能することが知られている．安定性を向上させるため，方策の更新頻度を価値関数よりも多くすることが一般的である．本章における実装では，方策の更新が行われる反復の一部に対してのみ，価値関数を更新するように改造できる．

```
struct ActorCritic
    𝒫     # problem
    b     # depth
    d     # initial state distribution
    m     # number of samples
    ∇logπ # gradient of log likelihood ∇logπ(θ,a,s)
    U     # parameterized value function U(ϕ, s)
    ∇U    # gradient of value function ∇U(ϕ,s)
end

function gradient(M::ActorCritic, π, θ, ϕ)
    𝒫, b, d, m, ∇logπ = M.𝒫, M.b, M.d, M.m, M.∇logπ
    U, ∇U, γ = M.U, M.∇U, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ,j) = sum(r*γ^(k-1) for (k,(s,a,r)) in enumerate(τ[j:end]))
    A(τ,j) = τ[j][3] + γ*U(ϕ,τ[j+1][1]) - U(ϕ,τ[j][1])
    ∇Uθ(τ) = sum(∇logπ(θ,a,s)*A(τ,j)*γ^(j-1) for (j, (s,a,r))
                 in enumerate(τ[1:end-1]))
    ∇ℓϕ(τ) = sum((U(ϕ,s) - R(τ,j))*∇U(ϕ,s) for (j, (s,a,r))
                 in enumerate(τ))
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return mean(∇Uθ(τ) for τ in trajs), mean(∇ℓϕ(τ) for τ in trajs)
end
```

アルゴリズム 13.1　初期状態分布 `b` をもつマルコフ決定過程 $\mathcal{P}$ に対して，方策勾配と価値関数勾配の両者を計算するための基本的なアクター・クリティック法．方策 `π` は `θ` でパラメータ化され，その対数勾配は `∇logπ` である．価値関数 `U` は `ϕ`でパラメータ化され，その目的関数の勾配は`∇U` である．このメソッドでは，`m` 回のロールアウトを深さ `d` まで実行する．その結果を用いて，`θ` と `ϕ` を更新する．方策のパラメータは，期待価値を最大化するために`∇θ` の方向に更新されるが，価値関数のパラメータは価値の損失を最小化するために`∇ϕ` の反対方向に更新される．

13.2　一般化アドバンテージ推定

一般化アドバンテージ推定 (generalized advantage estimation) (アルゴリズム

無限時間区間に対して，一般化アドバンテージ推定は次のように単純化される．

$$\hat{A}^{\text{GAE}}(s,a) = (1-\lambda)\left(\hat{A}^{(1)} + \lambda\hat{A}^{(2)} + \lambda^2\hat{A}^{(3)} + \cdots\right) \tag{13.18}$$

$$= (1-\lambda)\left(\delta_1\left(1+\lambda+\lambda^2+\cdots\right) + \gamma\delta_2\left(\lambda+\lambda^2+\cdots\right) + \gamma^2\delta_3\left(\lambda^2+\cdots\right) + \cdots\right) \tag{13.19}$$

$$= (1-\lambda)\left(\delta_1\frac{1}{1-\lambda} + \gamma\delta_2\frac{\lambda}{1-\lambda} + \gamma^2\delta_3\frac{\lambda^2}{1-\lambda} + \cdots\right) \tag{13.20}$$

$$= \mathbb{E}\left[\sum_{k=1}^{\infty}(\gamma\lambda)^{k-1}\delta_k\right] \tag{13.21}$$

パラメータ λ を調整することで，バイアスと分散のバランスを調整することができる．$\lambda = 0$ の場合，前節の時間差分残差における高いバイアス，低い分散の推定結果となる．$\lambda = 1$ の場合，バイアスのない完全なロールアウトが行われ，分散が増加する．図 13.1 は，異なる λ の値に対する実装である．

```
struct GeneralizedAdvantageEstimation
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood ∇logπ(θ,a,s)
    U     # parameterized value function U(ϕ, s)
    ∇U    # gradient of value function ∇U(ϕ,s)
    λ     # weight ∈ [0,1]
end

function gradient(M::GeneralizedAdvantageEstimation, π, θ, ϕ)
    𝒫, b, d, m, ∇logπ = M.𝒫, M.b, M.d, M.m, M.∇logπ
    U, ∇U, γ, λ = M.U, M.∇U, M.𝒫.γ, M.λ
    πθ(s) = π(θ, s)
    R(τ,j) = sum(r*γ^(k-1) for (k,(s,a,r)) in enumerate(τ[j:end]))
    δ(τ,j) = τ[j][3] + γ*U(ϕ,τ[j+1][1]) - U(ϕ,τ[j][1])
    A(τ,j) = sum((γ*λ)^(ℓ-1)*δ(τ, j+ℓ-1) for ℓ in 1:d-j)
    ∇Uθ(τ) = sum(∇logπ(θ,a,s)*A(τ,j)*γ^(j-1)
                 for (j, (s,a,r)) in enumerate(τ[1:end-1]))
    ∇ℓϕ(τ) = sum((U(ϕ,s) - R(τ,j))*∇U(ϕ,s)
                 for (j, (s,a,r)) in enumerate(τ))
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return mean(∇Uθ(τ) for τ in trajs), mean(∇ℓϕ(τ) for τ in trajs)
end
```

アルゴリズム 13.2　初期状態分布 `b` をもつマルコフ決定過程 $\mathcal{P}$ の方策勾配と価値関数勾配の両者を計算する一般化アドバンテージ推定法．方策は `θ` によってパラメータ化され，対数勾配 `∇logπ` をもつ．価値関数 `U` は `ϕ` によってパラメータ化され，勾配 `∇U` をもつ．このメソッドでは，`m` 回の深さ `d` のロールアウトが実行される．有限の時間区間をもつ式 (13.21) に基づいて，指数的な重み付け `λ` を用いた一般化アドバンテージが計算される．ここでの実装は，ステップ実行の際に信頼領域の側面を含んだ原論文で提案されたものよりも簡略化されたものである．

13.3　決定論的方策勾配

決定論的方策勾配 (deterministic policy gradient) アプローチ[3] は，パラメータ化された行動価値関数 $Q_{\boldsymbol{\phi}}(s,a)$ の形式での，クリティックを用いて連続した行動を生成する決定論的方策 $\pi_{\boldsymbol{\theta}}(s)$ を最適化する．これまでに議論してきたアクター・クリティック法と同様に，パラメータ $\boldsymbol{\phi}$ に関する損失関数を次のように定義する．

$$\ell(\boldsymbol{\phi}) = \frac{1}{2}\mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma Q_{\boldsymbol{\phi}}(s', \pi_{\boldsymbol{\theta}}(s')) - Q_{\boldsymbol{\phi}}(s,a)\right)^2\right] \tag{13.22}$$

[3] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *International Conference on Machine Learning (ICML)*, 2014.

例 13.1　決定論的方策勾配のヤコビ行列の例

以下は，2 次元行動空間と 1 次元状態空間に対する決定論的方策の例である．

$$\pi_{\boldsymbol{\theta}}(s) = \begin{bmatrix} \theta_1 + \theta_2 s + \theta_3 s^2 \\ \theta_1 + \sin(\theta_4 s) + \cos(\theta_5 s) \end{bmatrix}$$

この場合，行列 $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)$ は次の形式をとる．

$$\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) = \begin{bmatrix} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) \mid_{a_1} & \nabla_{\theta} \pi_{\theta}(s) \mid_{a_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ s & 0 \\ s^2 & 0 \\ 0 & \cos(\theta_4 s) s \\ 0 & -\sin(\theta_5 s) s \end{bmatrix}$$

他のアクター・クリティック法と同様に，$\ell(\boldsymbol{\phi})$ に対して勾配降下法を適用し，$\mathcal{U}(\boldsymbol{\theta})$ に対して勾配上昇法を適用する．この手法を実践的に用いるためには，いくつかの追加的なテクニックが必要である．1 つは，探索効率向上のために確率的な方策から経験を生成することである．アルゴリズム 13.3 のように，通常，決定論的方策 $\pi_{\boldsymbol{\theta}}$ によって生成された行動に平均 0 のガウスノイズを加えるだけで十分である．$\boldsymbol{\theta}$ と $\boldsymbol{\phi}$ を学習する際の安定性を確保するため，経験再生を用いることもある[4]．

例 13.2 では，この手法の具体例とパフォーマンスに対する σ の影響が示される．

[4] 経験再生については，強化学習の文脈で 17.7 節で詳しく説明する．学習の安定化における他の技術には，以下の文献によって神経表現の文脈で説明された**目標のパラメータ化** (target parameterization) などがある．T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *International Conference on Learning Representations* (*ICLR*), 2016. arXiv: 1509.02971v6.

アルゴリズム 13.3　初期状態分布 `b` をもつ連続行動のマルコフ決定過程 $\mathcal{P}$ に対して，決定論的方策 `π` の方策勾配 `∇θ` と価値関数勾配 `∇ϕ` を計算するための決定論的方策勾配法．方策は `θ` でパラメータ化され，各連続行動要素に対する勾配 `∇π` が列ベクトルとして含まれる行列として表される．価値関数 `Q` は `ϕ` でパラメータ化され，パラメータ化に対する勾配 `∇Qϕ` と行動に対する勾配 `∇Qa` をもつ．この手法では，`m` 回のロールアウトを深さ `d` で実行し，標準偏差 `σ` をもつ平均 0 のガウスノイズを用いて探索を行う．

```
struct DeterministicPolicyGradient
    𝒫  # problem
    b  # initial state distribution
    d  # depth
    m  # number of samples
    ∇π  # gradient of deterministic policy π(θ, s)
    Q  # parameterized value function Q(ϕ,s,a)
    ∇Qϕ # gradient of value function with respect to ϕ
    ∇Qa # gradient of value function with respect to a
    σ  # policy noise
end

function gradient(M::DeterministicPolicyGradient, π, θ, ϕ)
    𝒫, b, d, m, ∇π = M.𝒫, M.b, M.d, M.m, M.∇π
    Q, ∇Qϕ, ∇Qa, σ, γ = M.Q, M.∇Qϕ, M.∇Qa, M.σ, M.𝒫.γ
    π_rand(s) = π(θ, s) + σ*randn()*I
    ∇Uθ(τ) = sum(∇π(θ,s)*∇Qa(ϕ,s,π(θ,s))*γ^(j-1) for (j,(s,a,r))
                 in enumerate(τ))
    ∇ℓϕ(τ,j) = begin
        s, a, r = τ[j]
        s' = τ[j+1][1]
        a' = π(θ,s')
        δ = r + γ*Q(ϕ,s',a') - Q(ϕ,s,a)
        return δ*(γ*∇Qϕ(ϕ,s',a') - ∇Qϕ(ϕ,s,a))
    end
    ∇ℓϕ(τ) = sum(∇ℓϕ(τ,j) for j in 1:length(τ)-1)
    trajs = [simulate(𝒫, rand(b), π_rand, d) for i in 1:m]
```

$\nabla \ell(\boldsymbol{\theta})$ の逆方向に更新し，$\boldsymbol{\phi}$ を $\nabla \ell(\boldsymbol{\phi})$ の逆方向に更新することで，$\boldsymbol{\theta}$ と $\boldsymbol{\phi}$ を更新する[9]．

[9] AlphaGo Zero の実装においては，本節で説明したような独立したパラメータ化ではなく，1 つのニューラルネットワークを用いて，価値関数と方策の両者を表現する．ネットワークパラメータを更新するために使用される勾配は，式 (13.30) と (13.33) を混合することで計算できる．この改善によって，評価と特徴学習のための時間が大幅に短縮される．

13.5 要　約

- アクター・クリティック法では，アクターがパラメータ化された方策を最適化しようとして，クリティックはアクターを支援するために価値関数のパラメータ化された推定値を与える．
- 一般に，アクター・クリティック法では，方策と価値関数の近似の両者のパラメータを学習するため，勾配に基づき最適化する．
- 基本的なアクター・クリティック法では，アクターに対して方策勾配を用いて，クリティックに対しては二乗の時間差分残差を最小化する．
- 一般化アドバンテージ推定では，複数の時間ステップにおける時間差分残差を蓄積することで，方策勾配の分散を減らす代わりに，一部のバイアスを許容することとなる．
- 決定論的方策勾配では，決定論的な方策をもつアクターと行動価値クリティックを用い，連続した行動空間の問題に適用することができる．
- モンテカルロツリー探索などのオンライン手法は，方策と価値関数の推定を最適化するために用いる．

13.6 演　習

13.1　カートポール問題 (付録 F.3) を解くには，アクター・クリティック法とモンテカルロツリー探索を組み合わせた手法 (13.4 節) は，適切な手法であるか．

【解】　モンテカルロツリー探索は，訪れた状態に基づいてツリーを展開する．カートポール問題は連続した状態空間をもつため，無限の枝分かれをもつ探索ツリーが生成される．このアルゴリズムを用いるためには，状態空間を離散化するなど，問題を調整する必要がある．

13.2　以下のアドバンテージ関数の表現について，正しいものを選択し，それがどのような意味をもつかを説明しなさい．

(a) $\mathbb{E}_{r,s'}[r + \gamma U^{\pi_{\boldsymbol{\theta}}}(s) - U^{\pi_{\boldsymbol{\theta}}}(s')]$

(b) $\mathbb{E}_{r,s'}[r + \gamma U^{\pi_{\boldsymbol{\theta}}}(s') - U^{\pi_{\boldsymbol{\theta}}}(s)]$

(c) $\mathbb{E}_{r_{1:d},s'}\left[-U^{\pi_{\boldsymbol{\theta}}}(s) + \gamma^k U^{\pi_{\boldsymbol{\theta}}}(s') + \sum_{\ell=1}^{k} \gamma^{\ell-1} r_\ell\right]$

(d) $\mathbb{E}_{r_{1:d},s'}\left[-U^{\pi_{\boldsymbol{\theta}}}(s) + \gamma U^{\pi_{\boldsymbol{\theta}}}(s') + \sum_{\ell=1}^{k} \gamma^{\ell-1} r_\ell\right]$

(e) $\mathbb{E}\left[-U^{\pi_{\boldsymbol{\theta}}}(s) + \sum_{\ell=1}^{d} \gamma^{\ell-1} r_\ell\right]$

(f) $\mathbb{E}\left[-\gamma U^{\pi_{\boldsymbol{\theta}}}(s') + \sum_{\ell=1}^{d+1} \gamma^{\ell-1} r_\ell\right]$