平成 13 年度　博士論文

強化学習のための
自律分散型関数近似法

Function approximation for reinforcement learning
using autonomous-decentralized algorithm

指導教官　　新井 民夫　教授

精密機械工学専攻

学生証番号　　97096

小林 祐一

目 次

第1章　序論

1.1 研究の背景

1.1.1 強化学習とその課題

産業現場や家庭環境などで活動するロボットが，制御則を自律的に獲得あるいは改善することは重要な意味を持っている．その理由としては，ロボットが自律的に制御則を改善することで設計段階でのプログラムの不完全さを補うことができることや，実際のセンサ，アクチュエータの情報に即した自動設計や修正ができることなどを挙げることができる．ある評価関数のもとでの最適な制御則を求める問題は**最適制御問題**として知られ，その解法の一つとして動的計画法 (Dynamic Programming：DP) が 1960 年前後から研究されてきた [ベルマン 62]．動的計画法のうち，系の挙動や評価関数が未知のクラスの問題を扱う方法として，強化学習 (Reiforcement Learning) が定式化されている [Sutton98]．強化学習は動的計画法を理論的背景として関連付けられており，評価関数の部分的な情報 (報酬) をもとに試行錯誤によって制御則を獲得する学習法である．強化学習の扱う問題を定式化したものを **Fig.1.1** に示す．ロボットはセンサ入力を得て行動を出力し，環境から報酬を得る．ロボットは環境に関する情報や評価関数が未知の状態から，報酬をもとに「将来にわたって得られる報酬を最大化する」という目的のもと最適な制御則を探索する．

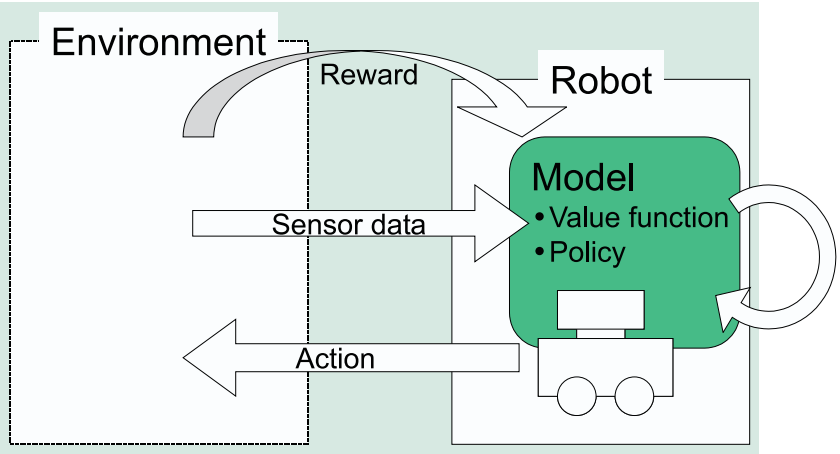

Fig. 1.1 強化学習の扱う問題

強化学習の核となるアルゴリズムは，報酬をもとに価値関数 (value function) [1)]

[1)] 状態価値関数 (state value function)，行動価値関数 (action value function) の両者を総称して価値関数 (value function) と呼ぶ．

と呼ばれる評価関数を更新する TD(Temporal Difference) 学習アルゴリズムである [Sutton88]．価値関数とは，ある状態 (と行動の組) に対して，その状態の後に得られる報酬 (評価) の期待値を表す関数である．強化学習エージェントはこれをオンラインで更新し，これをもとにオンラインで制御則 (policy) を改善する．強化学習は，シンボル処理型人工知能研究から実ロボット制御にわたる幅広い問題に適用されているが，すべての問題に成功を収めているわけではない．複雑な問題への強化学習適用を困難にする原因として，

状態の記述などの情報表現方法の非効率性

を挙げることができる．この問題は，抽象化された記号的表象を基礎としてきた人工知能研究においては軽視されがちであった．その背景には,「強化学習理論の研究はは抽象化された状態空間を扱い，実際の問題への適用の際には，関数近似の既存の手法を組み合わせればよい [Sutton98]」という考え方があったと考えられる．しかし，適用する問題により適切な関数近似方法が変わってくることや，強化学習に特有の関数近似法への制約が存在するため，既存の関数表現手法の単純な組み合わせで全ての問題に適用できるわけではない．

例えば，実際のシステムは状態空間の全域でなく特定の領域のみで充分な精度で制御できることを要求することが多い．**Fig.1.2** に要求精度の不均一性の例を示す．図左のようなマニピュレータによる平面上の物体操作を例に考えると，マニピュレータの各関節角により張られる状態空間中タスクの達成に重要なのは作業平面上に拘束された領域であり，その中でも対象物の周辺で精度高く制御することが要求されるが，それ以外の領域で要求される精度は比較的低い．また，通常の状態空間全域を均一の分解能で表現する方法では，状態変数の次元が高くなるにつれて指数関数的に必要な計算量が増える次元の呪い (the curse of dimensionality) の問題を伴う [Bellman59]．状態空間の設計は多くの場合設計者の経験的な試行錯誤に委ねられることが多いが，この過程を自律的に行うこと，すなわち「問題に即した状態空間の設計を自律的に行う過程 (**状態空間の自律的生成**)」が強化学習の重要な研究テーマである指摘されている [浅田 97]．

このような実用的見地だけでなく，ロボットの知能実現の過程を通じて人間の認知過程を探ろうとするロボット研究の立場 [浅田 99, 川人 00, 銅谷 99] からも，センサやアクチュエータ情報から行動する主体にとって重要な情報を抽出する過程としてこのような問題を重視する考え方もある [マクドーマン 99]．

強化学習においてこのような情報表現の核をなすのは，先に述べた TD 学習に

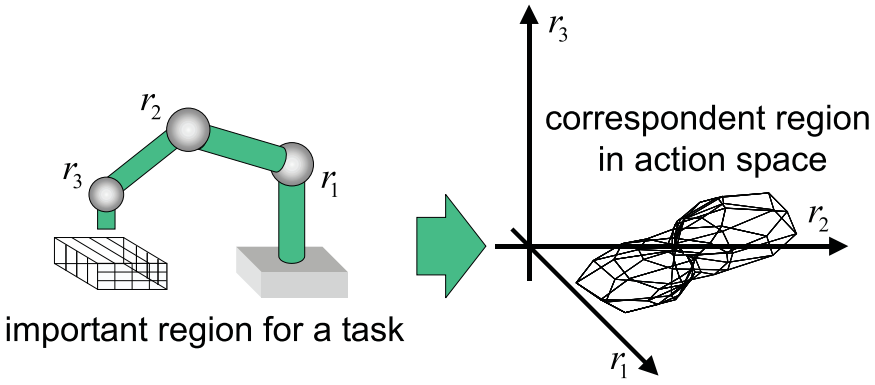

Fig. 1.2　状態空間中分解能の必要な領域の不均一性

よる価値関数である．すなわち，上記の情報表現の問題は強化学習においては価値関数の関数近似問題に対応している．

1.1.2　強化学習の例題：崖のある迷路問題

ここで，強化学習における価値関数近似問題の性質に関する具体的なイメージを提示するために，強化学習の簡単な例題を示す．崖のある迷路問題と呼ばれる問題 [Sutton98] を，**Fig.1.3** に示す．エージェントは，スタート地点 (S) からゴール地点 (G) を目指して 2 次元格子環境を上下左右に格子単位で移動する．各移動にはコスト (-1 の報酬) がかかり，確率的に (1/10 の確率で) 行動指令と異なるランダムな方向に移動する．図中灰色の領域は崖であり，ここに入ると大きな罰 (-100 の報酬) を受け，スタート地点に強制的に戻される．ゴール地点に到達するか 100 ステップの行動をとるかした場合，初期状態 (スタート地点) に戻して状態遷移を再び繰り返す．この一連の状態と行動の遷移系列を，一つの「試行 (trial) [2)]」と呼ぶ．

強化学習の目的は，得られる報酬 (の重み付き期待値) を最大にする行動を獲得することである．この問題では，崖に落ちるリスクを最小にした上で，最低コスト (最小ステップ数) でゴール地点に到達するための制御方策がそれに相当する．状態価値関数は，各状態において，その状態以後最適な行動をとった場合に得られる報酬の期待値を表す関数である．エージェントは，各状態遷移ステップ (行動後) に，得られた報酬および遷移前と遷移後の状態の状態価値から状態価値関数の更

[2)] エピソード (episode) と呼ぶこともあるが，本論では「試行」という呼称で統一する．

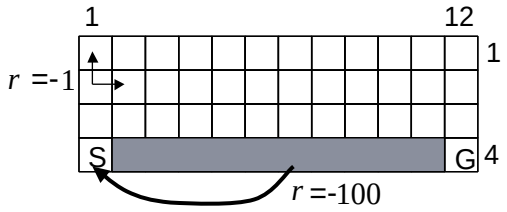

Fig. 1.3　崖のある格子状環境

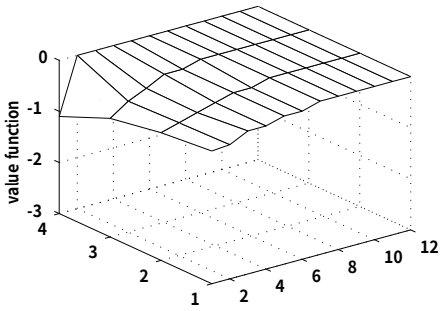

Fig. 1.4　状態価値関数 (1)

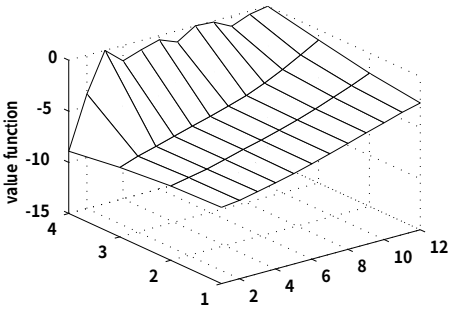

Fig. 1.5　状態価値関数 (2)

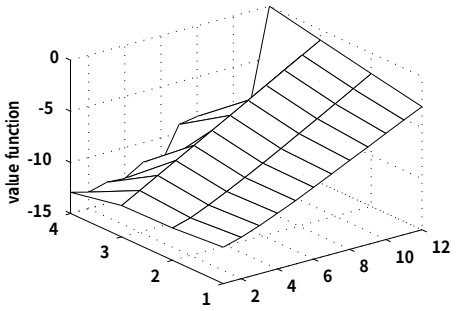

Fig. 1.6　状態価値関数 (3)

新を行う[3)]．崖の影響を無視した場合，状態価値関数は各状態におけるゴール地点までのステップ数を表す．

Fig.1.4，**Fig.1.5**，**Fig.1.6** はそれぞれ，試行回数 1，10，1000 回のときの状態価値関数を表している (**Fig.1.4** のみ縦軸のスケールが異なる)．図中右奥の位置 (12,4) のゴール地点は状態価値 0 で最大値をとる．最適な状態価値関数が獲得されれば，エージェントはこの状態価値関数の山を上るような行動をとることで最適な行動をとることができる．

このように，強化学習では，状態価値関数が漸近的に最適な目標関数に近づいていくことで方策を獲得することができる．この価値関数の推定は，

- 実際に行動をとったときに得られた報酬
- 関数近似器により推定した遷移前の状態価値

に基づいている．この例は価値関数をテーブル参照形式で表現したものであり，連続値空間を表現するためのもっとも単純な方法の一つである．

[3)] アルゴリズムの詳細は 2.3 節で述べる．

1.1.3 強化学習と関数近似

本研究では，強化学習における情報表現として強化学習アルゴリズムの核をなす**価値関数近似に着目**し，タスクに即した**効率の良い情報表現**を**自律的に構成する**ことを目指す．なお，強化学習と関数近似問題の関係については，2 章で詳しく述べる．ここではその概略を示し，強化学習における「効率の良い価値関数近似」について議論する．

一般に関数近似に求められるのは，(1) メモリ消費量および (2) 計算時間の意味での計算効率と (3) 近似精度である．テーブルなどの離散型表現手法ならば分割要素，連続関数近似手法ならば基底関数などの関数近似要素を増やすことで近似精度を高めることができる．一方で，強化学習での価値関数近似問題は，1.1.2 節で示した例からわかるように，**必要データの制約**，**オンライン性**，**非定常関数**という問題を含んでいる．

まず，関数近似要素を増やして近似精度を向上させるためにはより多くのデータを必要とする．強化学習は実際に行動をとって得た情報を用いるため，より多くのデータを得るためにはより多くの行動 (試行錯誤) が必要になる．また，関数近似要素の増加は，メモリ消費量および計算時間の意味で計算効率の低下を招く．この問題は高次元の問題ほど深刻になる．次に，オンラインの非定常関数近似という性質も強化学習の重要な特徴の一つである．ここで定常関数は，時間経過に対して変化しない関数を表す．逆に時間経過に対して変化する関数を非定常関数と呼ぶ．**Fig.1.7** に非定常性のある関数近似例を示す．図左側は TD 学習において逐次的に価値関数予測を行うときに，学習初期から最適な価値関数が得られるわけではなく，学習進行に伴って漸進的に最適価値関数に近づくことを意味している．図右側は，環境の変化により最適価値関数自体が時間経過とともに変化する場合を示している．このような場合，学習進行とともに目標関数が変化する問題にオンラインで対応する適応性能が求められる．前者の非定常性は強化学習に共通の問題であるが，非定常関数への適応性能が高ければ後者の問題に対応できることも期待される．

以上より，強化学習における価値関数近似には

- 学習進行に伴って変化する非定常関数を近似できる
- オンラインで学習にとって重要な領域に**適切な分解能**を得る

という性能が求められる．ここで「適切な」という言葉は，行動学習達成に充分な範囲で関数近似要素数を節約するという意味で用いている．

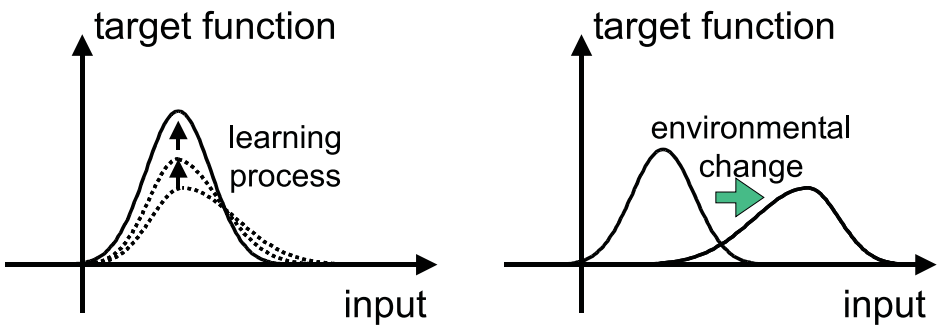

Fig. 1.7　非定常関数近似

このような「適切な分解能」という問題は，これまで積極的には研究されてこなかった．第一の理由として，従来型人工知能の機械学習の研究では，抽象化された表象の中で高度な問題を解く能力を重視する傾向があったことが挙げられる．第二に，実機を想定した強化学習適用例では，タスクの達成を重視するためにその過程の汎用性は重視されないことが多いことがいえる．学習によるタスク達成を最大の目的とすると，人間が試行錯誤によって得た知識をもとに適切な情報表現を設計する方が性能の良いものが得られるためである．

様々な問題に対してどのような情報表現が有効かという知識を蓄積するのも一つの研究のアプローチであるが，この過程を自律的に行う方法を目指す研究も行われている．次節では，上記の「オンラインで必要な領域に適切な分解能を得る」ということを目的とした研究例を示す．

1.2 関連の研究

関連の研究で最も関係の深いのは，すでに述べたように強化学習における価値関数表現を自律的に獲得するものである．一方で，本研究で提案する関数近似法は，後に詳しく述べるように，グラフを用いることや汎関数を最小化するなどの点でニューラルネットワーク理論との関係が深い．また,「効率の良い関数近似」という観点のみから関連の研究を探ると，CAD(Computaer Aided Design) や画像処理など様々な分野に共通点を見出すことができる．CAD などに用いられる形状表現の研究においても，ノードによって曲面を表現することは関数近似との類似性を持っている．そこで本節では，関連の研究を

(1) 強化学習に適用されている関数近似法
(2) ニューラルネットワーク理論
(3) CAD，画像処理などその他の関数近似関連研究

という三つに分類し，それぞれをまとめた上で本研究の目的との関係を論じる．

1.2.1 強化学習における関数近似法

前節で述べたように，強化学習における価値関数近似はオンラインの非定常関数の近似問題であり，関数近似要素が不必要に多すぎない適切な分解能を持っていることが学習性能に大きな影響を与える．従来の強化学習適用例では設計者が各問題に即した関数表現方法を工夫することで対処してきたが，このような関数表現における分解能を自律的に獲得することに焦点を当てた研究も行われるようになった．

Asada は，車輪型サッカーロボットのシュート行動を例にとり，画像特徴からなる状態空間を自律的に分割する手法を提案した [Asada96]．両輪の回転速度を離散化して要素行動とし，同一の要素行動によって到達できる状態群を一つの状態としてカテゴリにまとめるという方法をとった (**Fig.1.8**)．先にゴール状態に到達できる行動により多数の試行を行い，その後要素行動からセンサ入力を統計的に離散化するというオフライン手法である．行動学習法には Q-learning を用いている．Ishiguro も，Asada と同様に先に多数の試行を行った上で状態空間を統計的に分割する方法を提案している [Ishiguro96].

高橋らは，Asada の扱ったのと同様のサッカー行動を対象とし，オンラインでセンサ空間を分割する手法を提案した [高橋 99]．状態変数 $\boldsymbol{x}$，その時間微分を $\dot{\boldsymbol{x}}$,

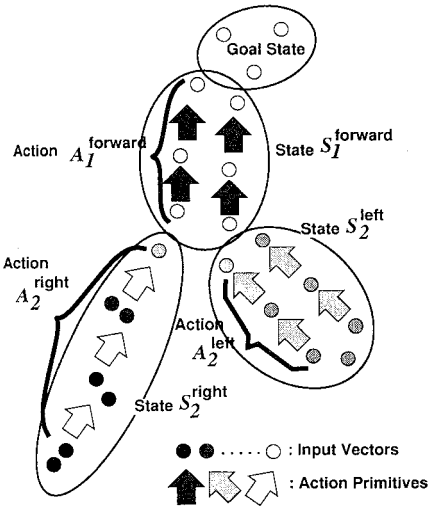

Fig. 1.8 状態遷移に基づく状態分割 [Asada96]

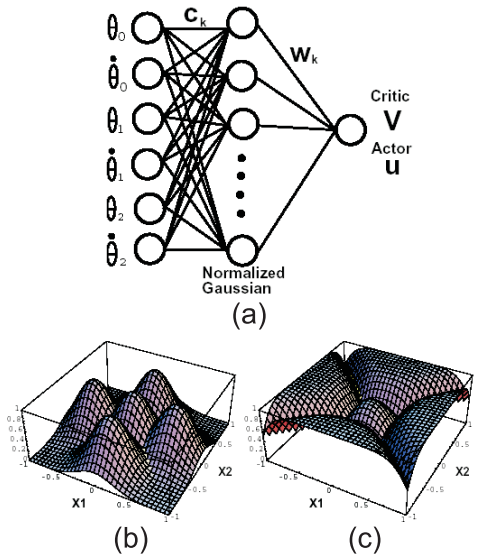

Fig. 1.9 ガウス基底関数による関数近似 [森本 99]

報酬を r として，

$$[\dot{\boldsymbol{x}}^T, r]^T = A\boldsymbol{x} + \boldsymbol{b} \qquad (1.2.1)$$

のような線形予測モデルを状態分割に用いた．上記の線形モデルによる予測 (最小二乗近似) の誤差がしきい値よりも大きければ，新しく局所線形モデルを追加する．**Fig.1.10** に高橋らによる分割法の概念を示す．状態遷移 (報酬値を含む) の関係を局所線形モデルで近似し，近似誤差が大きい領域では局所モデルを追加することで離散化された状態 $s_0, s_1, \cdots$ を得ている．

上記の例は分割による状態の離散化による分解能獲得の例であるが，連続関数近似器において基底を動的に追加することで分解能を獲得する研究例もある．Samejima*et al.* は，移動ロボットの経路探索問題において，動径基底関数 (Radial Basis Function,RBF) ネットワークを用いた状態価値関数近似を行った [Samejima98]．強化学習は Actor-Critic によって行い，得られる報酬のばらつきを判別基準とし，RBF の基底を動的に追加する手法を提案した．RBF は **Fig.1.11** に示すような動径基底関数の重ねあわせによる関数近似手法である．

Morimoto *et al.* は，RBF を正規化した GSBFN(Gaussian Softmax Basis Function Network，**Fig.1.9** 参照) を Samejima *et al.* と同様 Actor-Critic に適用した [Morimoto98]．棒型ロボットの起き上がりタスクを例題として連続値強化学習を実現した．ガウス基底関数の追加の判別基準として近似誤差を用い，基底の変形，移動アルゴリズムを提案した．

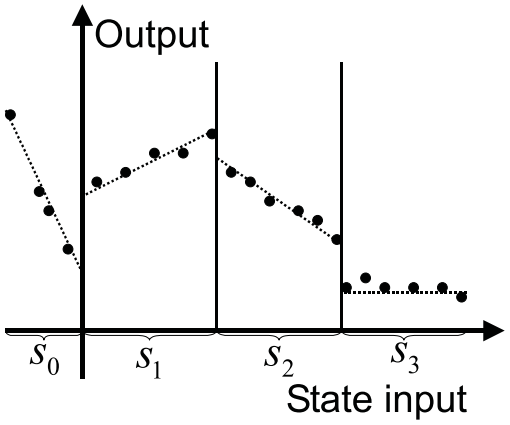

Fig. 1.10 線形モデル分割 [高橋 99]

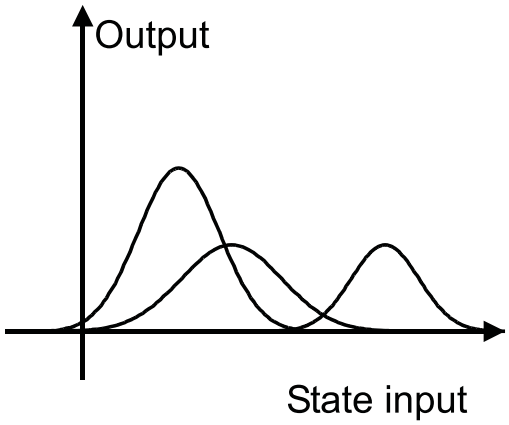

Fig. 1.11 RBFN[Samejima98]

これらの方法に共通するのは，離散化手法ならば分割の判断基準，基底関数による関数近似手法ならば基底追加の判断基準に基づいて逐次的に分割ないし基底追加を行う点である[4)]．高橋らは状態遷移に関する線形モデル当てはめ，Samejima*et al.* は報酬のばらつき，Morimoto*et al.* は近似誤差を用いている．一般に強化学習の進行過程では多くの場合ばらついた報酬信号が入るため，前節で述べたような非定常関数近似問題になる．このため，報酬のばらつきや近似誤差を判断基準とすると過剰な基底追加が起こる可能性がある．また，これらの判別にはしきい値を決める必要があるが，上記の理由から適切なしきい値の設定は経験に依らざるを得ず，かつ学習進行の時間や状態空間内の領域によって適切なしきい値が異なる可能性がある．

また，適切な分解能を獲得するためには，逐次的に分割していくだけでなく，既にある要素の動的な再配置や不必要な要素の削除も重要である．Morimoto *et al.* の方法は基底の移動や変形も扱っているが，一般に RBF ネットワークはパラメータの設計に性能が依存しやすくその過程が経験に依らざるを得ないことが指摘されており [Sutton98]，また基底関数の移動や変形の際の汎化作用が学習の収束性に悪影響を与えることもある [Tsitsiklis97].

Samejima *et al.* で用いられている学習達成度は，学習進行過程での不安定性を持つ．高橋らの方法は，近似価値関数の形状 (線形モデル) に注目している点でそのような不安定性を回避可能であるが，いったん分割すると分割要素の再配置ができない，という欠点がある．そこで，

[4)] Asada，Ishiguro の研究はオフライン手法であり，とるべき行動があらかじめある程度わかっていることが前提になっている．このため，ここでは議論の対象外とする．

- 学習達成度や近似精度以前に，価値関数自身の形状から
- 関数近似要素の動的な再配置，追加，削除の容易な方法で
- 統一的な尺度を導入して

関数表現を行うことができれば，その尺度のもとに近似要素の追加や再配置を行うことで適応的に非定常関数である価値関数を近似できると考えられる．

1.2.2 ニューラルネットワーク理論

ニューラルネットワーク理論の研究は，1943 年の McCulloch-Pitts の形式ニューロンの提案以来，生体の神経細胞のモデル化と最適化・特徴抽出などの工学的機能実現という二つの側面を持って様々な研究が行われてきた．階層型ニューラルネットワークとして知られる誤差逆伝播 (Error Back Propagation) 学習法 [Rumelhart86] はその代表例であるが，写像・特徴抽出は機能的には関数近似の範疇に含めることができる．

誤差逆伝播学習法　McCulloch-Pitts のパーセプトロンの原理に非線形基底関数を適用し非線形分離を可能にしたのが誤差逆伝播学習法 [Rumelhart86] である．**Fig.1.12** に示すように入力層，中間層，出力層からなり，中間層，出力層のニューロンは前の層の入力の 1 次結合に非線形単調増加関数 (しばしばシグモイド関数を用いる) をかけたものを出力する．教師信号として望ましい入出力の組が与えられる．このネットワークの目的は，与えられた入力信号パタン $x_{ic}(i = 1, \cdots, n)$ と教師信号パタン $d_{jc}(j = 1, \cdots, m)$ の組に対し，

$$E_c(t) = \frac{1}{2}\sum_j \{y_{jc}(t) - d_{jc}\}^2 \qquad (1.2.2)$$

で表される近似誤差を最小にすることである．ここで，y_{jc} は，入力信号パタン x_{ic} に対する出力層 j 番目のニューロンの出力である．この近似誤差を減少させるように，最急降下法に基づいて結合係数を変化させるのが誤差逆伝播学習である．出力層の誤差情報を入力層へ向かって伝播させることからこう呼ばれる．

最適化・特徴抽出モデル　ネットワークに定義される量を最適化するモデルとしては，Hopfield 型ネットワーク (**Fig.1.13**) やボルツマンマシンなどを挙げることができる [臼井 95]．Hopfield 型ネットワークは，ネットワーク全体のエネルギーを定義し，それを最小にするように各ニューロン同士の結合係数を変更していくというアルゴリズムをとっている．適用問題の例としては，記銘パタンの想起や，巡回セールスマン問題などが知られている．

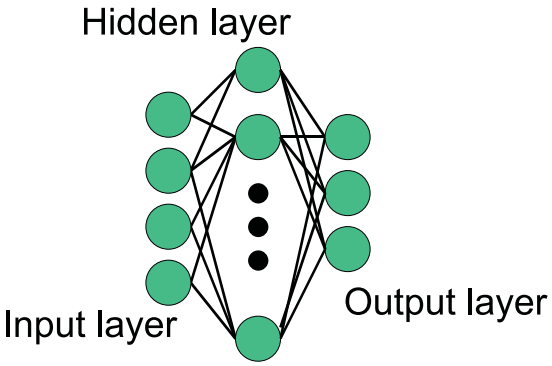

Fig. 1.12　階層型ネットワーク

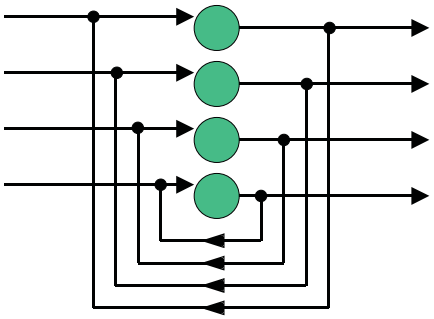

Fig. 1.13　Hopfield 型ネットワーク

また，生体 (特に視覚系) の神経回路における特徴抽出などの処理過程をモデル化したものも数多く提案されている．Kohonen の自己組織化マップ (Self Organizing Maps：SOM) [コホネン 96] は，生体の視覚系の神経細胞群において実現されているトポグラフィ[5] を学習する単純な工学的モデルとして幅広く応用されている．また，Linsker は，同時に活性化したニューロン同士の結合が強化されるという Hebb 則に基づいた学習則を用いて，ランダムな入力ベクトルから，場所[6] や方向選択性を持った特徴検出機構が構成できることを示した [Linsker88]．このモデルは，入力信号の持つ情報量に対して，冗長度がなるべく少なくなるように情報を変換する**冗長度圧縮の原理**にかなっているとされる．この冗長度圧縮を実現する考え方の一つとして，入力 $\boldsymbol{x}$ と出力 $\boldsymbol{y}$ の間の相互情報量ができるだけ大きくなるようにするという情報量最大化 (Information Maximization，Infomax) などが提案されている [Amari96, Bell95].

1.2.3　その他の関数近似関連研究

関数近似は，工学の至る分野に用いられている．それらのうち，適応的な分解能という本研究の目的に比較的近く，代表的なものとして以下の関連研究を挙げることができる．

形状表現における関数表現　　コンピュータ上で形状を表現するためのモデリングでは，メッシュを貼り合わせて曲面を表現する．このメッシュを逐次的に細かくしていく方法として，任意形状のメッシュに対して節点を追加して細かくす

[5] 互いに近くに位置する特徴抽出細胞は良く似た刺激に対して反応するという性質.
[6] 正確には，中心の入力が正で周囲の入力が負であるような入力信号に選択的に反応する性質. center–surround と呼ばれる.

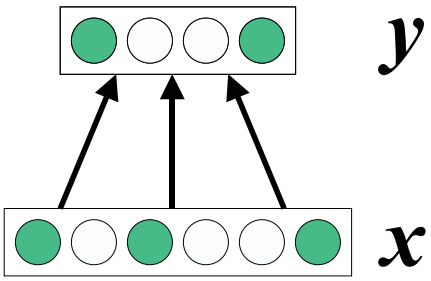

Fig. 1.14　特徴抽出モデル

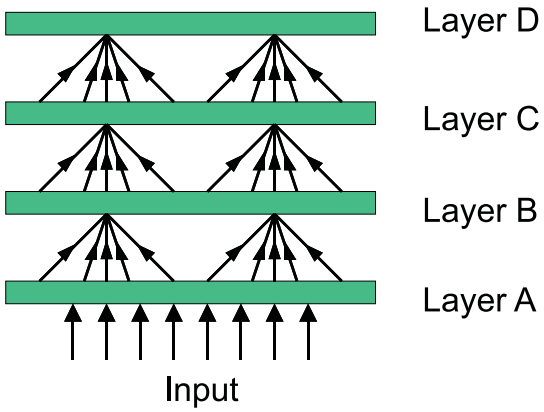

Fig. 1.15　Linsker のモデル

るアルゴリズム (subdivision) が知られている [Doo78, Catmull78]．節点数を節約するための方法としては，表現している対象物の形状に着目し，曲率の大きい領域でメッシュを細かくし，小さい領域では不必要に細かくしないという方法が考えられる．例えば山本らは，節点近傍のガウス曲率 [高木 96] を用いてメッシュを細分割する方法について，近似の連続性や法線間角度による評価を行って非一様の細分割の有効性を検証している [山本 99]．

信号解析・システム同定　　時系列信号をモデルによって当てはめ，信号の性質を把握したり予測に用いたりすることはシステム同定の研究として広く知られている [中溝 88]．自己回帰 (Autoregressive：AR) モデル，自己回帰移動平均 (Autoregressive moving-average：ARMA) モデルなどはその例で，現在の観測値を過去の観測値の 1 次結合で表すものである．時刻 t の観測を x_t と表すと，ARMA モデルは以下のように表される．

$$x_t + \sum_{i=1}^{p} a_i x_{t-i} = e_t + \sum_{i=1}^{q} b_i e_{t-i}, \quad t = 0, 1, 2, \cdots \tag{1.2.3}$$

ただし，e_t は正規分布に従う独立な系列であり，

$$E\{e_t\} = 0, \quad E\{e_t e_\tau\} = \sigma_e^2 \delta_{t\tau} \tag{1.2.4}$$

とする．1 次結合の係数 a_i, b_i がモデルのパラメータに相当し，これを最尤推定をはじめとする種々の推定法により求めることで時系列信号のモデル化および予測を可能にする．一般に，モデルの次数 (パラメータ数) を大きくすることにより，信号の推定の精度 (最尤推定ならば尤度) を大きくすることができるが，逆にモデルが特定の信号の表現に特化しすぎて予測モデルとしての性能が低下することがあ

る．このような問題を含めてモデル次数を評価するための規準として，情報量規準 AIC(Akaike Information Criterion) が提案されている．

本研究で提案する関数近似法は，実際に経験したデータをもとに非定常な価値関数を近似するため，これらの形状表現やシステム同定の枠組みが直接適用可能ではない．しかし，分解能を上げるための判断基準として曲率を用いるという考え方，あるいはモデルの次数をどの程度まで上げるのが適切かという問題と共通する考え方が存在する．その関連については，3 章における関数近似の設計法において考察を行う．

1.3 研究の目的

本研究では，前節で述べた背景のもと

価値関数の形状の複雑さに即した分解能を適応的に得る関数近似を行う

ということを目的とする．アプローチとして，

(1) 位置・勾配情報を持ったノードにより超平面を構成して関数近似を行う
(2) グラフ上の反応拡散方程式 [湯浅 99] に基づいてノードを関数形状の複雑度に応じて移動させる

という方法をとる．グラフ上の反応拡散方程式は自律分散系の適応アルゴリズムと見なすことができ，システム全体としての秩序を形成するように個々の要素の挙動を設計することができる．関数形状の複雑度として勾配変化に着目し，勾配変化の大きいところに密に，小さいところに疎に分布するような設計指針を取る．

提案手法の特色として，自律分散システムのアルゴリズムであることから

- 既にある関数近似の要素 (ノード) を動的に再配置することができること
- 関数近似の要素を動的に追加・削除することが容易であること
- 近似関数の局所的な複雑さと関数近似器全体との関係が陽に記述された方法であること
- 並列アルゴリズムであること
- 入力信号近傍のノードだけでなくネットワーク全体が学習を行うこと

などを挙げることができる．勾配に着目した適応手法である点で Takahashi *et al.* の線形モデルによる分割と共通する部分があるが，逐次分割によらず同数の要素のままでも適応的に再配置できる．これにより要素数の不要な増加を防ぐことができる．また，RBF ネットワークの更新アルゴリズムに対して，局所近傍における複雑度が定義され，それと関数近似全体との関係が明確に記述されているために設計者のパラメータ調整の複雑さを低減できるという利点がある．

ニューラルネットワーク理論との関係　　最後の 2 点は，位相構造をもったノードを基本単位として入力信号の統計的性質を反映した分布を形成する自己組織化マップ (Self Organizing Maps：SOM)[コホネン 96] と関係が深い．一般的な SOM は以下の特徴をもつ．

- 各ノードは隣接関係を表すアークで結ばれ，位相関係を表す
- 位相，ノード数は固定されている
- 入力データに対して，最整合ノードおよびその近傍ノードの結合係数ベクトルを入力データに近づける計算 (fitting) を繰り返す
- 獲得されたマップは，入力信号に対し各ノードが最整合となる確率が一様になるように構成される

本研究の提案するグラフによる関数近似は，並列アルゴリズムである点，アークで結ばれたノードによる情報表現を行う点や入力信号の性質を反映したネットワークを自己組織するという点が SOM と共通する．**Fig.1.16** は一般的な SOM と本研究で提案するグラフによる関数近似方法の比較を概念的に示したものである．行動学習における試行錯誤から得られた情報では，入力信号の確率密度を反映したマップよりも関数の形状 (勾配) を正確に表現していることの方が望まれる．また，マップ形成のアルゴリズムである fitting 計算は，SOM においては最整合ノードとその近傍に対してのみ行われる．これに対し，提案手法は反応拡散方程式の考え方に立脚しているため，グラフ上の全ノードに対して並列的に計算が行われる．このため，より高速な関数近似の効率化が期待できる．

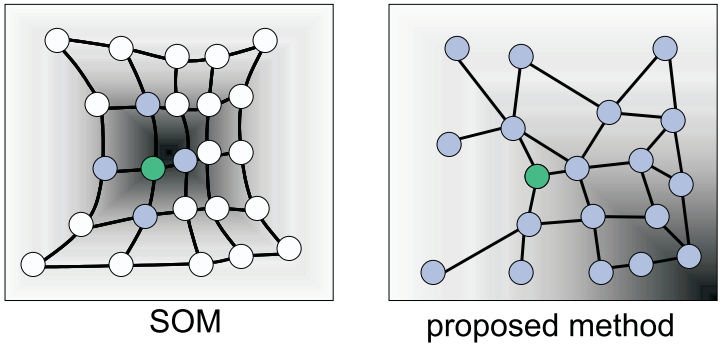

Fig. 1.16　自己組織化マップと提案グラフの相違

1.4 語彙集

本節では，本論文で用いる代表的な用語，数式に関する簡単な説明を行う．

グラフに関する用語・数式

ノード	グラフの構成素，節点．本研究では関数近似のための要素．
アーク	グラフの構成素，辺．近傍との隣接関係を表す．
近傍ノード	あるノードとアークで結ばれたノード
最整合ノード	入力ベクトルとの間のノルムが最小のノード．
ノード u の位置ベクトル	$\boldsymbol{w}_u$
ノード u の勾配ベクトル	$\boldsymbol{a}_u$
ノード u 上の 2 次導関数	B_u
ノード u 上の関数	$f(u)$
ノード u の近傍ノードの集合	$N(u)$
ポテンシャル汎関数	$W(f)$

強化学習に関する用語・数式

ブートストラップ	自力の．ここでは，関数近似のためのデータをその関数近似器自身によって生成することを指す．
試行	初期状態から終端状態までの一連の状態遷移
状態変数	連続系：$\boldsymbol{x}$，離散系：s
行動 (制御入力)	連続系：$\boldsymbol{u}$，離散系：a
方策	連続系：$\boldsymbol{u} = \pi(\boldsymbol{x})$，離散系：$a = \pi(s)$
報酬	時刻 t で実際に得られる評価．r_t
収益	時刻 t 以降に得られる報酬の期待値．R_t
状態遷移	連続系：$\dot{\boldsymbol{x}} = F(\boldsymbol{x}, \boldsymbol{u})$
状態遷移確率	離散系：$\mathcal{P}^a_{ss'}$
獲得報酬期待値	連続系：$R(\boldsymbol{x}, \boldsymbol{u})$，離散系：$\mathcal{R}^a_{ss'}$
状態価値関数	連続系：$V(\boldsymbol{x})$，離散系：$V(s)$
行動価値関数	連続系：$Q(\boldsymbol{x}, \boldsymbol{u})$，離散系：$Q(s, a)$

関数近似一般の数式

関数近似器のパラメータ	$\boldsymbol{w}$

関数近似器の出力	$\Phi^a(\boldsymbol{x})$
近似目標関数	$\Phi^t(\boldsymbol{x})$

頻出略語

CMAC	Cerebellar Model Articulation Cotnroller
DCS	Dynamic Cell Structures
LWR	Locally Weighted Regression
MDP	Markov Decision Process，マルコフ決定過程
POMDP	Partially Observable Markov Decision Process，部分観測マルコフ決定過程
RBFN	Radial Basis Function Networks，動径基底関数
SMDP	Semi Markov Decision Process，セミ (準) マルコフ決定過程
SOM	Self Organizing Maps，自己組織化マップ
TD 学習	Temporal Difference Learning
TRN	Topology Representing Networks

1.5 本論文の構成

本論文の構成を **Fig.1.17** に示す．2 章で提案手法に対する要件をまとめ，3 章，4 章でその要求に対する提案を行う．3 章では関数近似一般の提案を行い，4 章では強化学習への適用を論じる．5 章では提案した内容の評価を行う．

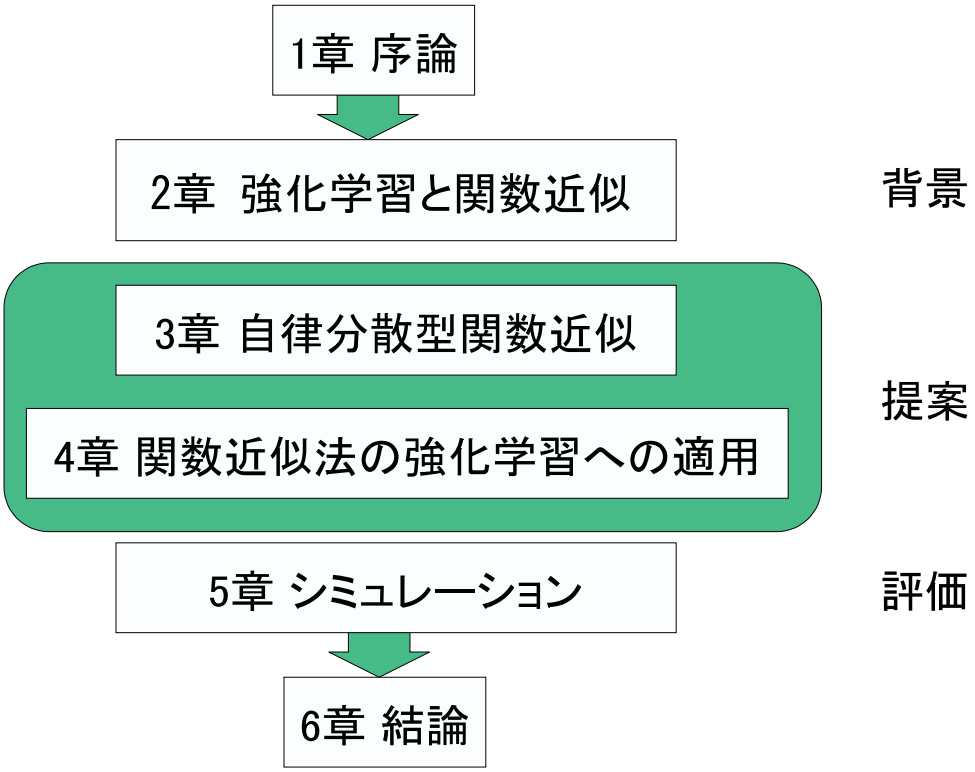

Fig. 1.17 本論文の構成

第 1 章では，強化学習における価値関数の近似問題に着目し，本研究の目的である適応的関数近似の意義について述べた．関連研究として状態空間を逐次的に分割する手法や基底関数を動的に追加する手法を示し，それらに対して，本研究の提案する関数近似法は自律分散型のアルゴリズムを用いたものであり，動的な要素の再配置が可能であることや全体との関係を記述した方法であるため設計自由度の低減が可能であることなどを示した．

第 2 章では，強化学習の基礎とその性質を説明し，関数近似問題に対する要請を述べる．価値関数更新法である TD(Temporal Difference) 学習への適用の観点か

ら関数近似手法を分類し，それらの中での本研究の位置付けを示す．TD 学習における価値関数近似の特色として非定常問題であることやブートストラップ型の近似であることなどを指摘し，同時に関数近似を価値関数近似に利用した場合の学習の収束・発散に関する議論も紹介する．また，適応的分解能の獲得という本研究の目的から見た関連研究の整理を再度行う．

第 3 章では本論文の核となる自律分散型関数近似法について述べる．まず自律分散型の関数近似の情報表現要素となる位置・勾配情報を持ったノードによる超平面の表現について説明する．ノード上に近似関数の複雑度を定義し，その複雑度を均一化するようにノードを移動させるという設計指針をとる．グラフ上の反応拡散方程式とその考えに基づいたノードの挙動設計法を示し，提案手法の妥当性に関する定性的な考察を行う．

第 4 章では，3 章で提案した関数近似手法の具体的な適用方法を示す．一般の関数近似問題と強化学習のための価値関数近似問題との問題の性質の違いを改めて説明し，それぞれについての適用方法，ノードの逐次的な追加・削除方法を示す．強化学習適用例としては Value Gradient と Q-learning(Advantage Updating) を取り上げ，それぞれのアルゴリズムおよびプログラムの実装について述べる．

第 5 章では，本研究で提案する関数近似法の評価を行う．一般的な関数近似手法として利用した例として定常関数近似問題をとりあげ，ノードの挙動設計の妥当性の評価を行う．強化学習適用の評価は連続空間，連続時間の強化学習問題として知られる水たまり問題および 1 自由度振子振り上げ問題を例題として利用する．ノード移動およびノードの逐次的追加により適応的分解能を実現し，その強化学習問題における有効性を検証する．

最後に第 6 章では，本研究の結論および展望を述べ，本論文を総括する．

第2章　強化学習と関数近似

2.1 はじめに

本章では，強化学習の概略および基本的なアルゴリズムについて述べる．強化学習における関数近似に要求されるものを明らかにし，本研究の提案する関数近似手法との関係を示す．

2.2 節では強化学習の歴史的経緯について説明し，対象とする問題の定式化を行う．

2.3 節では，強化学習の基本的アルゴリズムについて状態価値関数更新である TD 学習について述べる．本研究の関数近似手法は，この TD 学習における価値関数近似を対象とする．また，行動学習法として，Q-learning，Actor-Critic，Value Gradient という三つのアプローチを説明する．

2.4 節では，関数近似を強化学習に適用するときの問題点収束発散の議論および状態空間の自律的生成などの関連研究を示し，本研究の提案する関数近似の位置付けを明らかにする．

2.2 強化学習の概略

強化学習 (Reinforcement Learning) の研究は，歴史的には二つの大きな流れを持つ [Sutton98]．一つは学習心理学に端を発する試行錯誤学習を経た記号処理的な人工知能研究の流れであり，もう一つは最適制御問題の解法としての制御理論研究の流れである．

前者は，生物が,「報酬 (と罰) を頼りに，経験を強化して学習する」という心理学的な知見を模倣した試行錯誤学習アルゴリズムとしての研究で，これが強化学習の語源となる．**Fig.2.1** に示すのは「スキナーの箱」として知られる心理学実験の例である．偶然とった「レバーを下げる」という行動の後に報酬 (餌) を得たことで，その行動が強化される．試行錯誤学習の中心的な課題に「信用割当て問題 (credit assingment problem)[Minsky61]」があった．これは，報酬が得られたときに，過去のどの行動にその評価を配分するべきかという問題である．この問題の研究は，時間的に継続した予測の変化を用いた学習則である TD(Temporal Difference) 学習へとつながり，最適制御問題における評価関数更新則と同じ枠組みの議論へと発展していった．

一方，最適制御問題研究の流れでは，1950 年代後半に Bellman が Bellman 方程式と呼ばれる関数方程式を定義した [ベルマン 62]．この方程式を用いて最適制御問題を解く手法のクラスは動的計画法 (Dynamic Programming : DP) として知られるようになった．Watkins による Q-learning[Watkins92] によって TD 学習と動的計画法の流れはまとめられ，強化学習というカテゴリが動的計画法の一種として位置付けられるようになった．このような経緯を経て，強化学習は Back Gammon[Tesauro95] やスケジューリング [Zhang95] などの従来型人工知能の例題から，リンク機構からなる Acrobot の制御 [Sutton96]，小型移動ロボット Khepera のナビゲーション [Touzet97]，サッカーロボットのシュート行動 [浅田 97] などのロボット制御問題まで，幅広い問題に適用されるようになった．

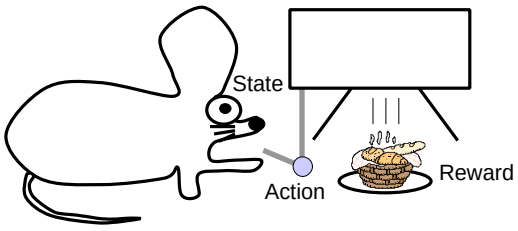

Fig. 2.1 報酬による経験強化の例

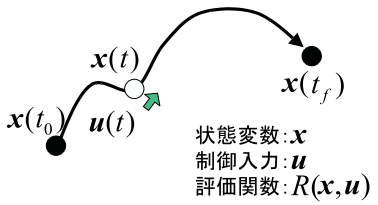

Fig. 2.2 最適制御の問題設定

最適制御問題は，系の状態変数 $\boldsymbol{x}$，制御入力 $\boldsymbol{u}$，評価関数を $J = \int_{t_0}^{t_f} R(\boldsymbol{x}(t), \boldsymbol{u}(t))dt$

としたときに，この評価関数を最大 (最小) にする意味で最適の制御方策を得る問題である．これを解く方法としては，ポントリャーギンの最大原理 [ポントリャーギン 00]，最適性の原理から導かれる Riccati の方程式や動的計画法などがある [坂和 80]．強化学習は，この最適制御問題において，動的計画法を拡張したものととらえることができる．一般的な動的計画法では，

- 系の挙動 $\dot{\boldsymbol{x}} = F(\boldsymbol{x}, \boldsymbol{u})$ (離散表現ならば $\mathcal{P}^a_{ss'}$)
- 評価関数 J

の両者が既知として関数方程式を解き，最適な制御方策を獲得する．これに対し強化学習は，これらが未知の状態から試行錯誤により制御方策を得るものと位置付けることができる．評価関数を環境から与えられる報酬の重み付き総和の期待値として表現するところに特徴がある．

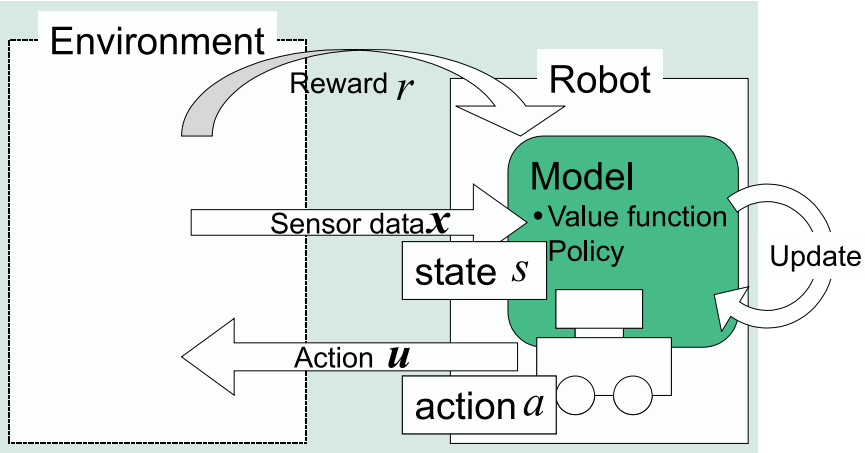

Fig. 2.3　強化学習の問題設定

Fig.2.3 に強化学習の問題設定を示す．エージェント (ロボット) はセンサ入力 $\boldsymbol{x}$ をもとに状態を認識し，行動 $\boldsymbol{u}$ を決定し，実行する．環境との相互作用の結果状態遷移が起こり，エージェントは状態と行動に依存した報酬 r を受け取る．エージェントは報酬を最大化するような行動則を探索する．最適制御問題と対比すると，エージェントの状態認識は環境とエージェントを含めた系全体の状態変数の獲得ととらえることができる．また，エージェントの行動は，系の状態変数をコントロールする制御入力とみなすことができる．

試行錯誤学習研究の流れにおいては，エージェントの認識する状態は離散的な事象として表現されることが多く，その後の研究で連続値表現との対応がとられ

るようになった [Gullapalli90, Baird94, Doya00]．本稿では，連続状態変数には $\boldsymbol{x}$，連続制御入力には $\boldsymbol{u}$，離散状態には s，離散行動 a を用い，アルゴリズムの説明は従来の離散型の表現を基礎とした上で連続値表現との対応について説明する方法をとる．

強化学習研究の基礎である動的計画法では，状態と行動をマルコフ決定過程 (Markov Decision Process：MDP) として扱っている．**Fig.2.4** に MDP の例を示す．MDP とは,「現在の状態と行動にのみ依存して状態遷移と報酬が決まる過程」と表すことができる．形式的には，状態と報酬値の個数は有限であるとした場合，時刻 $t+1$ における環境の応答が

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \qquad (2.2.1)$$

のように時刻 t における状態と行動のみに依存することを「状態信号がマルコフ性を持つ」という．言い換えると，MDP では過去の事象全ての履歴を含めた遷移確率

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \cdots, r_1, s_0, a_0\} \qquad (2.2.2)$$

が (2.2.1) 式と等しくなる．この性質が，次節で述べる学習アルゴリズムの基礎になっている．

逆にこの性質を満たさないクラスの問題は，部分観測マルコフ過程 (Pertially Observable Marcov Decision Process：POMDP) と呼ばれる．POMDP に属する問題では，状態変数，制御入力を過去にさかのぼって用いることで状態を識別することが求められる．

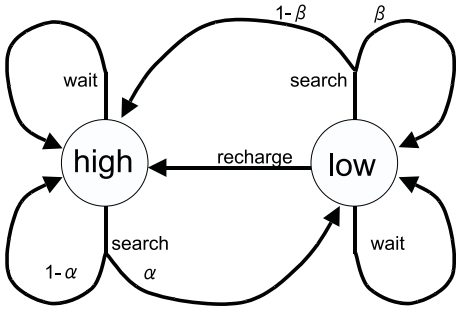

Fig. 2.4　MDP：リサイクルロボットの例

図の説明：エージェントの状態識別は，バッテリの残量が充分にあるか (high) そうでないか (low) の二通りである．各状態において行動の選択肢として (1) 缶を探す (search)，(2) 静止して誰かが缶を持ってくるのを待つ (wait)，(3) バッテリ再充電のため基地に戻る (recharge)，のいずれ

かを選ぶことができる．缶の発見に正の報酬，バッテリ切れによる負の報酬が与えられるとし，各状態遷移に対して期待報酬 $\mathcal{R}^a_{ss'}$ と状態遷移確率 $\mathcal{P}^a_{ss'}$ が定義される．この期待報酬と状態遷移確率は，それより前のステップの状態には依存しない．

2.3 学習アルゴリズム

本節では，強化学習の扱う枠組みの定式化と学習アルゴリズムの具体例を示す．問題の概略，状態価値関数の更新と具体的な行動学習方法の順で述べる．

2.3.1 問題の概略

システムの状態の離散的な表現を s，行動を a とする．決定論的に行動を決定する場合は，行動は $a = \pi(s)$ のように制御方策 (policy)π によって与えられる．状態 s から s' に行動 a によって遷移する確率を $\mathcal{P}^a_{ss'}$ のように表す．この際に得られる報酬の期待値を $\mathcal{R}^a_{ss'}$，実際に時刻 t に得られる報酬を r_t で表す．時刻 t から将来にわたって得られる報酬の (減衰付き) 期待値 R_t を

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad (2.3.1)$$

と定義する．ここで $\gamma(0 < \gamma \leq 1)$ は**時間遅れ**[1] のある報酬に対する割引率で，大きいほど時間遅れのある報酬を重視した評価関数を形成する．強化学習の目的は，この R_t を評価関数として最大にするような制御方策 π を求めることである．

各状態において，その状態から方策 π に基づいて行動を決定したときの得られる報酬の重み付き期待値を状態の関数として $V^\pi(s)$ と表し，これを状態価値関数 (state value function) と呼ぶ．また，状態 s において行動 a をとった後の報酬の期待値を $Q^\pi(s, a)$ と表し，これを行動価値関数 (action value function) と呼ぶ．評価関数 R_t を最大にする方策 π^* を最適な方策とよび，最適な方策のもとでの状態価値関数，行動価値関数をそれぞれ V^*, Q^* と表す．動的計画法は，最適な価値関数 V^*, Q^* が Bellman の最適性の原理を満たすことを利用する．Bellman の最適性の原理とは，

> 最適方策は，初期状態と行動が何であっても，以後の行動は最初の行動から生じた状態に対して最適方策を構成しなければならない

というものである [ベルマン 62]．数学的には以下の Bellman 方程式がそれぞれの

[1] 報酬が得られた場合にその過去の状態にさかのぼってその報酬情報から過去の状態の評価を更新する．このとき，その状態から見ると報酬は未来に (時間的に遅れて) 得られる．これを「時間遅れ」と表現する．

価値関数について成り立つことで表される．

$$\begin{aligned} V^*(s) &= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')] \end{aligned} \tag{2.3.2}$$

または

$$\begin{aligned} Q^*(s,a) &= \max_a E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')]. \end{aligned} \tag{2.3.3}$$

いずれの式においても，状態 s と一つ遷移をさかのぼった状態 s' との間で最適な行動選択をした場合の関係式が最適価値関数において成り立つものとして理解することができる．動的計画法においては状態価値関数は (2.3.2) 式または (2.3.3) 式の繰り返し計算によって漸近的に求めることができる．

連続値空間における Bellman 方程式　　連続値空間においては，状態変数を $\boldsymbol{x} \in R^n$，制御入力を $\boldsymbol{u} \in R^m$ とする．系の挙動は

$$\dot{\boldsymbol{x}}(t) = F(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.3.4}$$

制御方策 π は

$$\boldsymbol{u}(t) = \pi(\boldsymbol{x}(t)) \tag{2.3.5}$$

と表され，状態価値関数は

$$V^\pi(\boldsymbol{x}(t)) = \int_t^\infty e^{-\frac{s-t}{\tau}} r(\boldsymbol{x}(s), \boldsymbol{u}(s)) ds \tag{2.3.6}$$

と定義される．Bellman 方程式の連続時間での定式化は Hamilton–Jacobi–Bellman 方程式と呼ばれ，

$$\frac{1}{\tau} V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}(t)} \left[r(\boldsymbol{x}(t), \boldsymbol{u}(t)) + \frac{\partial V(\boldsymbol{x})}{\partial \boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}) \right] \tag{2.3.7}$$

のように表される [Doya00]．

状態遷移確率 $\mathcal{P}^a_{ss'}$ および報酬関数 $\mathcal{R}^a_{ss'}$ (連続値空間においては $F(\boldsymbol{x}, \boldsymbol{u})$ および $r(\boldsymbol{x}, \boldsymbol{u})$ がこれに対応する) は既知とされているが，この両者が未知の状態で Bellman 方程式による状態価値関数の更新を行うのが TD(Temporal Difference) 学習である．

2.3.2 TD 学習：状態価値関数更新

TD 学習における Bellman 方程式は以下のように表される．

$$\begin{aligned} V^{\pi}(s_t) &= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \\ &= E_{\pi}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s \right\} \\ &= E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \middle| s_t = s \right\} \end{aligned} \tag{2.3.8}$$

時刻 t の状態遷移では，行動 a_t により s_t から s_{t+1} に遷移し，報酬 r_{t+1} を得るものとする．

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.3.9}$$

を TD 誤差とよび，これが 0 になるように状態価値関数 $V(s_t)$ を更新し，それに伴って方策 π を改善していくことで価値関数 $V^{\pi}(s)$ を (2.3.8) 式の Bellman 方程式を満たす最適な価値関数 $V^{*}(s)$ に近づけることができる．具体的な更新式は，

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t \tag{2.3.10}$$

となる．ここで α は $0 < \alpha < 1$ の定数である．この更新則は TD(0) と呼ばれる最も単純なものであり，このような 1 ステップ過去の情報を用いた方法でなく，過去複数ステップの情報をさかのぼって利用する方法として TD(λ) が研究されている [Watkins89].

連続値空間における TD 学習則　　上記の離散時間における TD 学習を連続時間に拡張した議論として，Doya[Doya96] の研究がある．(2.3.6) 式の状態価値関数の定義式の両辺を時刻 t で微分すると，

$$\frac{d}{dt}V^{\pi}(\boldsymbol{x}(t)) = \frac{1}{\tau}V^{\pi}(\boldsymbol{x}) - r(t) \tag{2.3.11}$$

を得る．これより，連続時間での状態価値関数推定誤差は

$$\delta(t) = r(t) - \frac{1}{\tau}V(t) + \dot{V}(t) \tag{2.3.12}$$

と表せる．状態価値関数 $V(\boldsymbol{x})$ の更新は，上記の TD 誤差を用いて

$$V(\boldsymbol{x}) \leftarrow V(\boldsymbol{x}) + \alpha\delta(t) \tag{2.3.13}$$

によって行われる．

先に述べた離散時間における TD 学習則とは，以下のように対応させることができる．(2.3.13) 式に $\dot{V}(t)$ を代入することで，

$$\delta(t) = r(t) - \frac{1}{\Delta t}\left\{(1 - \frac{\Delta t}{\tau})V(t) - V(t - \Delta t)\right\} \qquad (2.3.14)$$

を得る．上式において，

$$1 - \frac{\Delta t}{\tau} \leftarrow \gamma \qquad (2.3.15)$$

$$\frac{1}{\Delta t} \leftarrow V(t)V(s_t) \qquad (2.3.16)$$

とおくことで (2.3.9) 式が得られる．(2.3.15) 式は，1 次の Taylor 展開において $e^{-\frac{\Delta t}{\tau}} \simeq 1 - \frac{\Delta t}{\tau}$ となることに対応している．

2.3.3 行動学習方法

前節の議論で，状態価値関数更新の方法について述べた．この状態価値関数更新をもとに最適な方策 π を学習する代表的な方法として，以下の方法を挙げる．

- Q-learning[Watkins92]，Advantage Updating[Baird94]
- Actor-Critic[Gullapalli90, Williams92]
- Value Gradient[Doya00]

強化学習の理論的な拡張として TD(λ)，Sarsa(方策オン型)，セミマルコフ決定過程 (Semi-Markov Decision Process：SMDP)，部分観測マルコフ決定過程 (Partially Observable Markov Decision Process：POMDP) などの概念があるが，ここでは簡略化のため最も基本的な強化学習アルゴリズムを述べる．以下に各行動学習法の考え方とアルゴリズム，連続時間問題への適用についての説明を行う．

2.3.3.1 Q-learning(Advantage Updating)

Q-learning は Watkins により提案された，行動価値関数 $Q(s, a)$ から行動を学習する方法である．離散状態，離散行動に対する Q-learning の更新式は

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \qquad (2.3.17)$$

で表される．これは (2.3.10) 式の TD 学習式と同様に s_t から s_{t+1} に遷移したときに得た報酬 r_{t+1} を用いた更新則である．各状態では Q 値に基づいて確率的に行動を選択する．その代表例として Boltzmann 分布の式を用いた確率的な行動選択則は

$$\pi(s,a) = \frac{\exp(Q(s,a)/T)}{\sum_b \exp(Q(s,b)/T)} \qquad (2.3.18)$$

のように表される．ここで T は Boltzmann 温度と呼ばれる正の定数であり，これが高い場合には全ての行動が (ほぼ) 同程度に選択される．学習の進行とともに温度を下げていき，ランダムな行動選択の割合を減らしていくという方法がしばしばとられる．以下にアルゴリズムの流れを示す.

(1) 初期化： すべての s, a に対する $Q(s,a)$ の初期化
(2) 各試行の初期化： 各試行における初期状態 s_0 を決定
(3) 行動決定：(2.3.18) 式により得た $\pi(s_t,a)$ に基づいて確率的に行動を決定
(4) 状態 s_{t+1} に遷移し，報酬 r_t を受け取る
(5) (2.3.17) 式による行動価値関数 $Q(s,a)$ の更新
(6) 試行終了判定：(2) に戻る
(7) 繰り返し：(3) に戻る

連続時間への拡張：Advantage Updating　Baird *et al.* は，Q-learning と同様に行動価値関数を基礎とした行動学習法を連続時間，連続空間に拡張したものを Advantage Updating として提案した [Baird94]．(2.3.7) 式の Hamilton–Jacobi–Bellman 方程式に基づいた表現では，

$$A^*(\boldsymbol{x},\boldsymbol{u}) = r(\boldsymbol{x},\boldsymbol{u}) - \frac{1}{\tau}V^*(\boldsymbol{x}) + \frac{\partial V^*(\boldsymbol{x})}{\partial \boldsymbol{x}}F(\boldsymbol{x},\boldsymbol{u}) \qquad (2.3.19)$$

のように表される関数を最大化することで行動出力の最適性を保証することができる．この関数の導入は，離散時間において定義されている Q-learning の更新式を，任意の時間幅に対応することができるようにしていると見ることもできる.

Advantage Updating は，時間，空間に関する連続性を扱える半面，関数近似器に連続値行動に関して最大値探索を要求する点が実際の計算の足かせとなっている.

2.3.3.2　Actor-Critic

Actor-Critic は，行動決定器を状態価値関数とは独立に構成する行動学習方法である．**Fig.2.5** に Actor-Critic の機構を示す．状態を評価する機構 (critic) と行動

を決定する機構 (actor) から構成され，actor は critic からの情報をもとに方策を修正する．critic は，報酬をもとに TD 学習により状態価値関数を更新し，actor へ行動修正指令を送る．

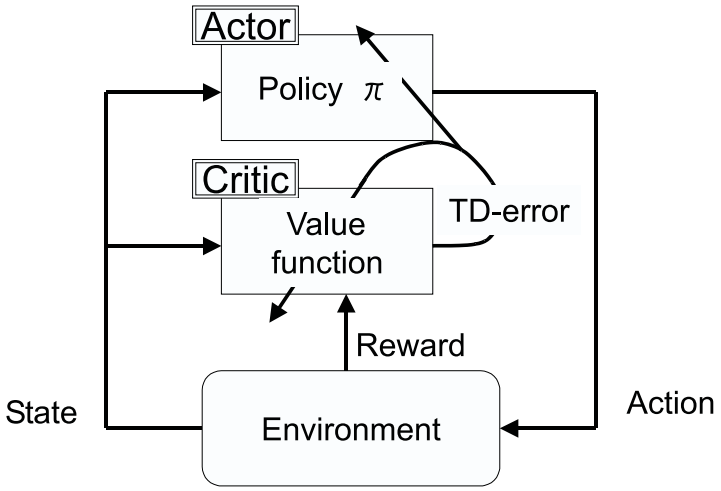

Fig. 2.5　Actor-Critic の機構

Actor における行動選択確率をアルゴリズムの具体例として，連続状態 $\boldsymbol{x}_t$，離散行動 a に対して行動選択確率を与える関数 $\pi(a, \boldsymbol{w}, \boldsymbol{x})$ を用いる木村らの確率的傾斜法の例 [木村 00] を示す．ただしここで $\boldsymbol{w}$ は，行動選択確率関数 π を関数近似器で表現したときの関数近似のパラメータベクトルであり，π は $\boldsymbol{w}$ の各成分で偏微分可能であるとする．

(1) 初期化： $V(\boldsymbol{x})$ の初期化
(2) 各試行の初期化： 各試行における初期状態 $\boldsymbol{x}_0$ を決定
(3) 行動決定 (actor)：$\pi(a, \boldsymbol{w}, \boldsymbol{x}_t)$ に基づいて確率的に行動 a_t を決定
(4) $\boldsymbol{x}_{t+1}$ に状態遷移し，報酬 r_t を受け取る
(5) 状態価値関数 $V(\boldsymbol{x})$ の更新
(6) 行動決定器の更新：

$$\Delta \boldsymbol{w} = \delta_t \boldsymbol{n}(t) \frac{\partial \pi(a_t, \boldsymbol{w}, \boldsymbol{x})}{\partial \boldsymbol{w}} \qquad (2.3.20)$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w} \qquad (2.3.21)$$

(7) 試行終了判定：(2) に戻る
(8) 繰り返し：(3) に戻る

ここで，$\boldsymbol{n}(t)$ は正規分布に従う乱数ベクトルである．Actor-Critic の学習アルゴリズムとしては Gullapalli の stochastic real-valued unit algorithm (SRV) [Gullapalli90] や Williams の提案する REINFORCE アルゴリズム [Williams92] などが提案されているが，強化学習の歴史的には，Q-learning のような状態と行動の組を単位とする行動価値関数を用いたアルゴリズムが盛んに研究されたという経緯があったため，収束性の議論を含めて理論的な研究はまだ十分にはなされていない．Actor-Critic アルゴリズムには方策を陽に学習するために連続値行動を実現しやすい，部分観測マルコフ決定過程に対応しやすい [木村 96] などの利点があり，今後の研究の進展が期待されている．

2.3.3.3 Value Gradient

Value Gradient は，状態価値関数から直接行動を決定する手法である．状態価値関数の勾配 (Gradient) の情報を用いて，山登り法的に行動を決定するものである [Doya00]．行動 (制御入力) と状態遷移の関係が決定論的で既知あるいは容易に獲得できるような問題で有効である．

状態変数を $\boldsymbol{x}$，行動出力を $\boldsymbol{u}$ としたときに，系の挙動が $\dot{\boldsymbol{x}} = F(\boldsymbol{x}, \boldsymbol{u})$ で表されるとすると，行動出力 $\boldsymbol{u}$ の状態変化に対する寄与は $\frac{\partial F(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}}$ となる．この項は各行動に対して,「その行動をとったときにどのように状態が遷移するか」ということを表しており，問題によっては容易に獲得できる場合もある．例えば 1 自由度振子振り上げ問題ではトルク T と角速度 ω の比例関係，2 次元平面上を移動するエージェント (状態変数は平面上の位置) の問題では各行動によりエージェントの位置が上下左右に移動する関係などは既知のものとして扱うことが可能であり，仮に未知だとしても $\boldsymbol{x}$ によらず一定であるため比較的容易に学習により獲得できる．

この場合の行動決定方法は，状態価値関数 $V(\boldsymbol{x})$ を用いて

$$u_j = u_j^{\max} S\left(\frac{\partial F(\boldsymbol{x}, \boldsymbol{u})}{\partial u_j}\frac{\partial V(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) \qquad (2.3.22)$$

のようになる．ただし，ここで $S(x)$ は単調増加関数である．通常はシグモイド関数のような連続出力を用いるが，ステップ関数を用いれば bang-bang 制御則になる．

(1) 初期化： $V(\boldsymbol{x})$ の初期化
(2) 各試行の初期化： 各試行における初期状態 $\boldsymbol{x}_0$ を決定
(3) 行動決定 (ε-greedy 戦略)：
　ε の確率で乱数により行動決定
　それ以外の場合：$j = 1, \cdots, m$ に対して

$$u_j = u_j^{\max} S\left(\frac{\partial F(\boldsymbol{x}_t, \boldsymbol{u})}{\partial u_j} \frac{\partial V(\boldsymbol{x}_t)}{\partial \boldsymbol{x}}\right) \qquad (2.3.23)$$

(4) $\boldsymbol{x}_t$ に状態遷移し，報酬 r_t を受け取る
(5) 状態価値関数 $V(\boldsymbol{x}_t)$ の更新
(6) 試行終了判定：(2) に戻る
(7) 繰り返し：(3) に戻る

2.4 TD 学習と関数近似

前節までで述べたように，強化学習の核となるアルゴリズムは TD 学習と呼ばれる逐次的な価値関数の更新である．この価値関数の表現について，基本的に要求される性質と制約を考察し，その上で本研究で重視する適応的な分解能と設計自由度の低減という要求について改めて考察し，関連研究の整理を行う．

(1) 強化学習のための情報表現に求められる要件
(2) 提示要件のもとでの関数近似手法の分類
(3) 適応的に分解能を得る関数近似手法の研究例と本研究の位置付け

以上が本節の流れである．強化学習に適した情報表現方法について論じる部分では，関数近似と収束証明の議論についても紹介する．ここで，関数近似器の出力を一般に入力ベクトル $\boldsymbol{x}$ として $\Phi^a(\boldsymbol{x})$ で表すこととする．

2.4.1 関数近似の要件

(線形関数などの) モデルを仮定しない価値関数表現の最も単純な方法の一つは，状態変数を離散化し，テーブル参照 (lookup table) と呼ばれる表の形式で値を持つことである．しかし，単純な離散化では，次元の増大や細かい離散化によりテーブルの容量が爆発的に増加し，実用的でなくなる．これを避けるために，状態変数を連続のまま扱う方法や効率の良い離散化を行う方法が研究されている．これらは全て，価値関数の関数近似を行う問題とみることができる．一般の関数近似においては 1 章でも触れたように，

(1) メモリ消費量の節約
(2) 計算時間の短縮
(3) 近似精度の向上

が要求される．価値関数近似の目的は最適な方策の獲得であり，最終的に必要とされるのは必ずしも近似精度でない可能性もあるが，以後の議論では，基本的に一般の場合と同様近似精度と計算効率を基本要求とする．

強化学習における価値関数近似の特徴と制約を以下に列挙し，各項目について説明する．

(1) **オンライン計算**を要すること

(2) **ブート スト ラップ型**関数近似であること
(3) **非定常関数**近似問題であること
(4) **必要データ数**の節約を要すること

オンライン計算　入出力の組が一定数与えられた上でそれを一括して近似するのではなく，データは各学習ステップにおいて逐次的に与えられ，その都度近似を更新することが求められる．これに対して，入出力の組を蓄積しておいてそれらをオフラインで (繰り返し計算などを用いて) 近似するというアプローチも可能である．このように一つの試行が終了するまで (終了状態に至るまで) 状態遷移を繰り返してから価値関数更新を行う方法は Monte Carlo 法と呼ばれる．しかし，この方法はオフライン近似手法を扱いやすいという利点がある反面，オンラインで価値関数や方策を改善するという強化学習の特長の一つを犠牲にしてしまう．

ブート スト ラップ型[2]**近似**　価値関数近似では，(2.3.9) 式，(2.3.10) 式 のように，関数近似を更新するためのデータとして前のステップで更新した関数近似器の出力を用いる．関数近似では汎化や補外などが行うことがあるが，このようなブートストラップ型の近似では発散を招くおそれがある．関数近似を用いた場合の収束と発散については下記補足段落で述べる．

非定常関数　ブートストラップ型近似という性質と関連して，目標近似関数は学習の時間経過に対して常に一定の関数が推定できるわけではなく，学習が進行するにしたがって最適価値関数[3]に近づいていく．特に，学習の初期段階では一般に報酬が入ってきたときに TD 予測誤差は大きくなり，目標関数は不安定である．そのため，関数近似には目標関数の変動に対する柔軟性が求められる．

必要データ数の問題　一般に，分解能の高い関数近似を達成するためにはより多くのデータを必要とする．連続空間の学習においては，データのサンプリングと制御の周期を短くする (価値関数更新を頻繁に行う) ことで状態空間中の軌跡上のデータを多く得ることはできるが，軌跡周辺のデータを十分に得るためには，より多くの試行を行わなくてはならない．この意味で，必要データ数の増加は間接的に学習効率の低下を招くおそれがある．

補足：発散可能性と収束証明　テーブル形式で離散化された状態表現においては，強化学習アルゴリズムの多くは収束証明がなされている [Sutton98]．その一方で，TD 学習を関数近似と組み合わせた場合に発散が起きる例が報告されている [Baird95, Boyan95]．理論的には，Tsitsiklis *et al.* により，MDP の軌跡に沿った更

[2] ブートストラップ (bootstrap) とは自力の，自給の，という意味．ここでは目標関数の値を得るのに関数近似器自身の出力を用いることを表す．
[3] 最適方策 π^* のもとでの価値関数．

新を行わない場合や関数近似器に非線形近似を用いた場合の発散の可能性が指摘されている [Tsitsiklis97]．逆に，収束が証明されている安全な方法として線形近似関数を挙げることができる．線形独立な関数近似要素による

$$\Phi(\boldsymbol{x}) = \sum_{i=1}^{p} \theta_t(i)\phi_i(\boldsymbol{x}) \qquad (2.4.1)$$

のようなパラメータ線形の関数近似手法では，TD 学習の収束が証明されている [Tsitsiklis97]．また，定性的には，安定して価値関数近似を行うためには汎化の影響を大きくしすぎないこと [Kaelbling96]，補外などの誇張を含む近似を行わないこと [Gordon95] などが重要であることが指摘されている．

Baird の発散例を **Fig.2.6** に示す．状態数は 7 で，上の 6 状態の一つを初期状態とし，すぐに下の状態に遷移し，そこで何回かのサイクルを経た後で終端状態にたどり着く．状態遷移に対する報酬は全ての場合で 0 である．すなわち，最適価値関数はすべての状態 s に対して $V^*(s) = 0$ である．この問題を関数近似に組み合わせたとき，ある初期値に対しては DP による方策評価が不安定になることが報告されている．

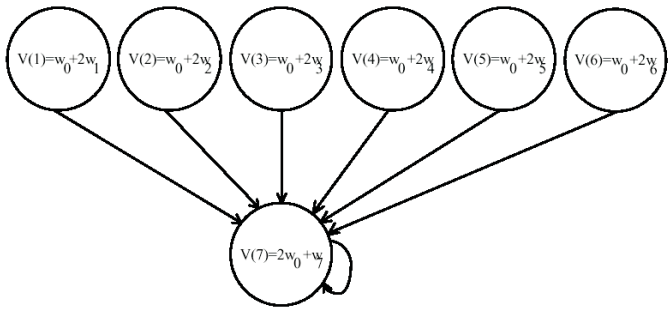

Fig. 2.6　線形関数近似の発散例 [Baird94]

Baird の例は実際には意味のない単純かつ特殊な状況を作り出したものであるが，関数近似の利点が生かされることが期待されるような比較的大きな状態空間を扱う問題での発散を報告する例もある．**Fig.2.7** は自動車による山登り問題 (Mountain car problem) と呼ばれるタスクに階層型ニューラルネットワークを適用して発散した例である [Boyan95]．自動車が山のある地形を前後に移動し，頂上で停止するというタスクである．状態変数は車の位置と速度の 2 次元，行動はアクセルのオンオフである．図左側は最適状態価値関数であるが，階層型ニューラルネットワークを直接的に適用した結果，図右側のように価値関数近似が発散してしまっている．

以上の議論から，強化学習における価値関数近似に求められる性質として，

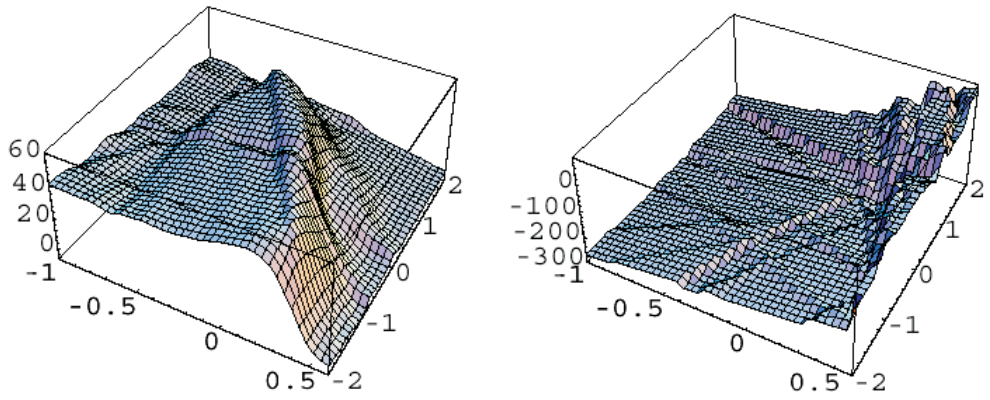

Fig. 2.7　山登り問題での状態価値関数近似の発散例 [Boyan95]

- オンライン更新に適していること
- 非定常目標関数に対応しやすいこと
- 収束条件内にある，もしくは発散の危険性が少ないこと

の三つを挙げることができる．これらは強化学習に適用するために関数近似手法に求められる基本的な要件である．

本研究ではこれらの基本的要件に加え，1 章でも述べたように

(1) 適応的に分解能を獲得できること
(2) 統一的尺度を導入することで設計自由度を減少できること

の二つを重視した関数近似手法を目的としている．(1) の適応的な分解能の獲得は必要データ数を節約するという考え方からも，後に述べるメモリベースの関数近似手法のメモリ節約の観点からも重要な意味を持っている．(2) の設計自由度の主張は，本節での関数近似に対する要件とは離れ，関数近似器の実際の利用の際に問題設定に依存した調整を多く要するという不便さを回避しようとするものである．あらゆる問題に対して完全に自律的に適切な関数近似を達成することは困難であるが，その設計自由度を低減することで試行錯誤の手間や経験に依存した調整を低減できれば非常に有用である．

2.4.2　関数近似手法の分類

強化学習に適用されている手法を中心に，代表的な関数近似手法の分類を **Table 2.1** に示す．

Table 2.1　関数近似手法の分類

分割 (離散化) 手法	代表点による分割 格子 (グリッド) 状分割 木構造 (kd-tree etc.) 統計的手法 (共分散行列，SOM，　etc.)
メモリベース手法	LWR(局所線形回帰)[Atkeson97a] 実例に基づいた強化学習 [Santamaria98]
連続関数近似手法	多項式近似 (Spline 曲線 [鳥谷 91]，多項式 [Bellman63] etc.) CMAC etc. 階層型ニューラルネットワーク [Rumelhart86] RBFN，GSBFN

状態の離散化手法としては，代表点からの距離による分割，グリッド表現，木構造，統計的性質を用いた分割 [水野 96]，自己組織化マップ (Self Organizing Maps, SOM)[コホネン 96] などを挙げることができる．非線形性の強い関数や高次元の状態空間に対しては，統計的な離散化を行う近似方法は有効である．しかし，非定常関数の適切な近似のためには再分割や統合を要するために向いていない．例えば **Fig.2.8** に示す Munos*et al.* による木構造分割手法は，系の挙動と強化信号が既知である (動的計画法に近い) 問題に適用を限定している．特に統計的分割法は，データの特徴抽出能力に優れる反面直接のオンライン適用が難しいという欠点がある．

観測データを直接記憶する手法として代表的なものは LWR(Locally Weighted Regression) である．これは，データをメモリに逐次的に蓄え，入力近傍のデータを用いた回帰直線による近似計算を行う局所線形回帰手法である [Atkeson97a]．これは実例を直接に記憶するために広大な状態空間であっても学習に必要な領域を細かく識別できるという利点がある．しかし，LWR はパラメータ非線形な近似手法であり，ブートストラップ型の価値関数近似には適していない [Boyan95]．実ロボットによる学習例として広く知られる Schaal らの Juggling への LWR の適用例 [Schaal94] は，強化学習の価値関数近似に用いたものではなく，制御入力と状態遷移間の関係における順逆モデル獲得という入出力の組が一方向的に与えられる問題に用いている．

記憶したデータから線形回帰を行わず平均を利用する方法ならば，非線形近似

に起因する発散の危険性を避けることができることが報告されている [Gordon95]. このようなメモリベース手法で問題になるのは，全てのデータを記憶すると計算効率，メモリ消費量の上で非効率的であるため，適当な忘却が必要になることである．記憶したデータの忘却の判断基準には時間や濃度など様々な方法が提案されている [Atkeson97b] が，本研究で提案する方法は，近似関数の複雑度という局所的な形状情報を判断基準にしたメモリベース手法ととらえることもできる．

連続関数近似手法としては，誤差逆伝播学習 [Rumelhart86] を用いる階層型ニューラルネットワーク (Multi-Layered Perceptron：MLP または Feed Forward Neural Networks：FFNN)，動径基底関数 (Radial Basis Function Networks, RBFN) [Fritzke94]，CMAC(Cerebellar Model Articulation Cotnroller)[Albus75] などを挙げることができる．階層型ニューラルネットワークは，Tesauro の Back Gammon への適用例 [Tesauro95] が人間のチャンピオンと互角のレベルの性能を得るという大きな成功を収めたこともあり盛んに研究がなされた [Lin93]．しかし，LWR と同様非線形近似手法であることから，発散する可能性があることが指摘されている [Bertsekas96]．Boyan による山登り問題の発散の例は，LWR と MLP への適用例において報告されている．また，関数近似要素としてシグモイド関数のような単調関数の出力を複数層にわたって重ね合わせる機構を用いているため，適応的に分解能を変更するためにノードの追加や削除を行うことに適していない．

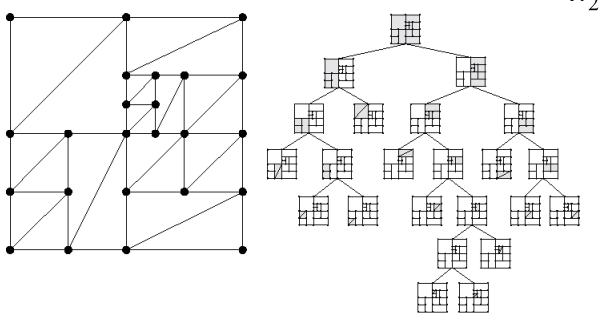

Fig. 2.8 木構造による分割 [Munos01]

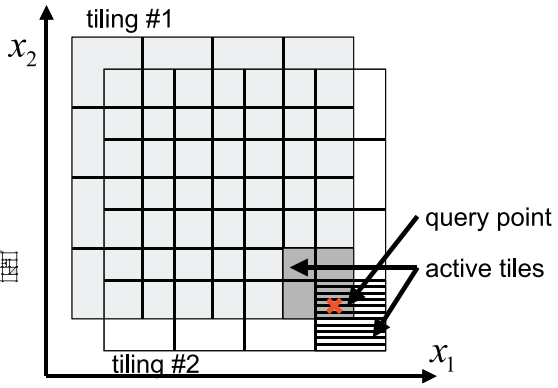

Fig. 2.9 タイルの重ねあわせによる関数近似

CMAC は **Fig.2.9** に示すような分割格子 (tiling と呼ぶ) を重ね合わせる関数近似手法である．単一の分割格子の場合は格子状分割と等価になるが，分割格子を重ねることで，より高い分解能と汎化作用を得ることができる．図は 2 枚の分割格子を重ねた例である．入力点 (query point) に対してその点を含む格子 (tile) は

それぞれの分割格子から一つずつ，計二つ存在する．この二つの格子に対して重み付き和で出力を決定し出力に対する寄与を分担することで，分解能の向上と汎化作用を実現している．CMAC は線形近似手法であり，比較的複雑な問題にも安定的に適用できる例が報告されている [Sutton96]．しかし CMAC は格子の形状や粗さが固定であり，適応的に分解能を獲得するのには適していない．

Table 2.2　関数近似手法の比較

	オンライン	非定常	収束証明	適応的分解能	設計自由度
統計的分割	△	×	○		
木構造分割	○	×	○		
LWR	○	△	×		
MLP	△	○	×		
メモリベース	○	△	○		
RBFN	○	○	○	○	×
CMAC	○	○	○	×	△

以上の議論より，代表的な関数近似手法について **Table 2.2** のようなような性能比較を示すことができる．オンライン性，非定常性，収束証明という基本的要件に対して良い性能を示しているのは RBFN と CMAC である．そのため，本研究の評価および比較は，この RBFN と CMAC を中心に考え，本研究で重視する適応的分解能と設計自由度の項目に関する比較を次小節では詳しく述べる．

また，適応的分解能の獲得という目的でなされた状態空間の逐次分割手法の研究も行われているが，上記の分類には含まれてない．次小節では，RBFN および分割手法を中心に適応的分解能獲得を目標とした研究例について述べる．

2.4.3　適応的分解能の獲得

上記の関数近似手法について，強化学習への適用研究例を紹介し，適応的な分解能の獲得における問題点と本研究の位置付けを述べる．

Samejima*et al.* は，移動ロボットの経路探索問題において，動径基底関数 (Radial Basis Function,RBF) ネットワークを用いた状態価値関数近似を行った [Samejima98]．強化学習は Actor-Critic によって行い，得られる報酬のばらつきを判別基準とし，RBF の基底を動的に追加する手法を提案した．RBFN は **Fig.1.11** に示すような

動径基底関数の重ねあわせによる関数近似手法である．

Morimoto *et al.* は，RBF を正規化した GSBFN(Gaussian Softmax Basis Function Network) を Samejima *et al.* と同様 Actor-Critic に適用した [Morimoto98]．棒型ロボットの起き上がりタスクを例題として連続値強化学習を実現した．ガウス基底関数の追加の判別基準として近似誤差を用い，基底の変形，移動アルゴリズムを提案した．

Bruske *et al.*[Bruske95] は，Williams *et al.* の提案した REINFORCE アルゴリズムを用い，連続値行動出力の枠組み [Williams92] と位相構造を保持したネットワークである DCS(Dynamic Cell Structures) と RBF を組み合わせた手法を提案し，移動ロボットの衝突回避問題に適用している．

浅田ら [Asada96] は，サッカーロボットのシュート行動を例題として用い，同じ要素行動に対する遷移可能性によって状態分割を行った．Ishiguro *et al.*[Ishiguro96] は，移動ロボットの経路探索問題に対し，生の画像情報から統計的手法により状態空間分割を行う方法を提案し，事前にランダムな動作を多数行い，得た入力情報を主成分分析などの統計的手法によって圧縮し，判別関数を用いて状態空間を分割した．また，Murao *et al.*[Murao97] は，Q 値の更新則を用いた状態の分割手法を提案している．高橋ら [高橋 99] は，センサ入力を局所線形モデルに当てはめる状態分割を提案している．

浅田，Ishiguro *et al.* の方法はオフライン手法であり，ある程度行動が達成できるという根本的な矛盾をはらんでいる．Morimoto *et al.*，Samejima *et al.* の方法では価値関数の近似誤差によって基底を追加するかどうかを判断するが，判別しきい値を試行錯誤的に求めなくてはならないことと前節で指摘したように非定常関数近似のために無駄に基底が追加されるという問題がある．

2.5 本研究の位置付け

これらの関連研究から高橋らの逐次分割手法，Morimoto *et al.*，Samejima *et al.* の基底関数追加手法などを有力なものとして挙げることができる．高橋らは状態遷移に関する線形モデル当てはめ，Samejima*et al.* は報酬のばらつき，Morimoto*et al.* は近似誤差を用いている．一般に強化学習の進行過程では多くの場合ばらついた報酬信号が入るため，報酬のばらつきや近似誤差を判断基準とすると過剰な基底追加が起こる可能性がある．また，これらの判別にはしきい値を決める必要があるが，上記の理由から適切なしきい値の設定は経験に依らざるを得ず，かつ学習進行の時間や状態空間内の領域によって適切なしきい値が異なる可能性がある．つまり，設計自由度が高くその調節が設計者の試行錯誤に委ねられてしまう．また，適切な分解能を獲得するためには，逐次的に分割していくだけでなく，既にある要素の動的な再配置や不必要な要素の削除も重要である．

以上の議論より，これまでの関連研究の問題点として

(1) 適応的に分解能を変更するための判断基準の議論が不充分であり，設計者の試行錯誤に委ねられる自由度が高い
(2) 非定常問題に対応するためには関数近似要素の動的再配置も重要であるが，充分に行われていない

ということがいえる．本研究で提案する関数近似手法は，上記の問題点に対して

(1) 関数近似器全体との関係が統一的に記述される尺度として，複雑度の定義を導入する
(2) 関数近似要素の動的再配置，追加，削除の容易な機構を提案する

ということを目的としている．

2.6 おわりに

本章では，強化学習の概要を説明し，関数近似に必要とされる要件をまとめた．強化学習は，動的計画法と理論的な関連性をもっており，オンラインで価値関数を更新する TD 学習アルゴリズムを核とする．価値関数の近似問題には

- 非定常関数 (時間経過とともに目標近似関数が変化する) 近似問題であること
- ブートストラップ型関数近似問題であること
- 上記 2 点の特質に対し発散せず安定的に近似できる必要があること
- 必要データ数の節約を要すること→高すぎる分解能を回避する必要があること

などの特徴がある．これらの基本的要求を満たした上で，本研究では，

(1) 適応的に分解能を変更できること
(2) 設計自由度が低減され，設計者の試行錯誤に頼る過程が少なくてすむこと

という特質を重視する．

分解能を適応的に獲得するという関連の研究に対して，本研究では

(1) 関数近似器全体との関係が統一的に記述される尺度として，複雑度の定義を導入する
(2) 関数近似要素の動的再配置，追加，削除の容易な機構を提案する

ということを目的とする．次章以降では，ここで指摘した特質に対する性質という観点から関数近似手法に対する考察・評価を行う．

第3章　自律分散型関数近似

3.1 はじめに

本章では，本論文の核となる，自律分散的手法に基づいた関数近似法について述べる．

3.2 節では，グラフ上の反応拡散方程式の考え方に基づく関数近似の基本的な設計方針について述べる．

3.3 節では，位置・勾配情報を持つノードによる関数近似方法を示す．一つは，観測データをもとに最小二乗法により位置・勾配を更新する方法，もう一つは，観測データをもとに位置のみを更新し，位置情報から勾配を決定する方法である．

3.4 節では，グラフ上の反応拡散方程式とその背景となる自律分散システムについて説明する．自律分散システムの特徴である耐故障性と拡縮性について，本研究におけるノードを用いた関数近似手法との関係について述べる．

3.5 節では，3.4 節で述べたグラフ上の反応拡散方程式の考えに基づき，ノードの密度を適応的に変化させるための挙動設計について述べる．各ノードに対してその近傍における複雑度を定義し，その複雑度を系全体が均等に負担するようにノードを移動させるという方法をとる．

3.6 節では，3.5 節における挙動設計の妥当性に関する議論を行う．ノードの複雑度の定義に対しては，その妥当性に関する考察および近似誤差最小化問題との関係についての考察を行う．また，本提案手法が自律分散システムである重要な利点であるノードの逐次的な追加や削除について，追加や削除の判断基準に関する考察を行う．

3.2 自律分散型関数近似の方針

本節では，グラフ上の反応拡散方程式の考え方に基づく関数近似の基本的な方針について述べる．本研究で提案する関数近似方法は多次元入力 1 次元出力の陽関数を対象とする．設計の目的は，目標近似関数の形状に着目し，

> 近似関数の複雑な領域に関数近似要素を密に分布させることにより適応的な分解能を実現する

ことである．そのための方法として，以下の過程を経る．

(1) 位置と勾配を持つノードにより平面を表現する (**Fig.3.1**)
(2) 近傍のノード同士をアークで結んでグラフを構成し，超平面の組み合わせで近似面を構成する (**Fig.3.2**)
(3) 各ノードの局所近傍での近似関数形状の複雑度を定義する (**Fig.3.3**)
(4) 各ノードの複雑度を周囲のノードと均一にするようなノードの挙動を設計する (**Fig.3.4**)

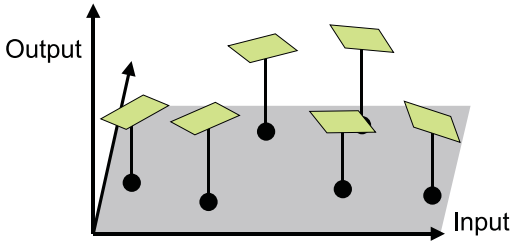

Fig. 3.1 平面の貼り合わせ

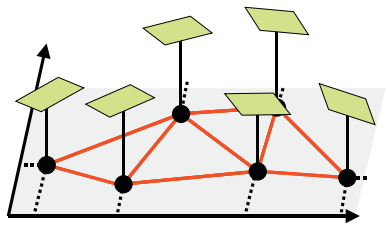

Fig. 3.2 境界付きグラフへの適用

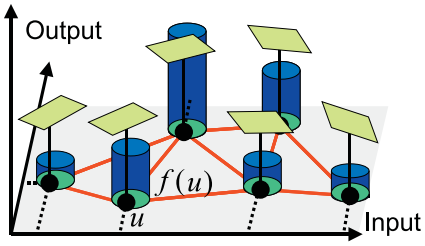

Fig. 3.3 複雑度の定義

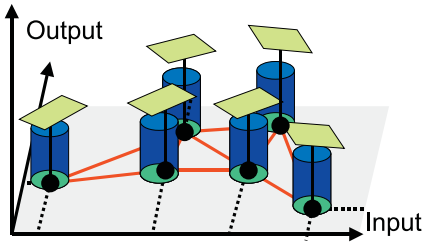

Fig. 3.4 複雑度の均一化

(2) でグラフを構成することにより，(3) におけるノードの近傍という概念を表現する．**Fig.3.3** に示すように，各ノード u は関数近似のための位置・勾配情報を

持つと同時に近似関数形状の複雑度を表す値 (図中円柱で表される $f(u)$) を持っている．この複雑度を均一にするように各ノードを移動させる．つまり，複雑度の高い領域にはノードが密に分布することで複雑度を低減し，複雑度の低い領域にはノードが疎に分布することで複雑度を増大させる．これにより適応的な分解能を実現する．

ここで扱うノードによる関数表現システムは，空間内に同一の性質をもつ動的要素が不均一・不連続に分布しているシステムといえる．このようなシステムにおいて全体として秩序を形成するように個々の要素の挙動を設計する方法としてグラフ上の反応拡散方程式 [湯浅 99] が提案されている．この考え方を用いて，ノード上に表現された複雑度を全体として均一にするためのノードの挙動を設計する．**Fig.3.4** に複雑度を均一にするようにノードを移動させた様子の概念図を示す．ノード移動は，関数近似面の形状を保つような拘束のもとで行う．

なお，本章で提案するアルゴリズムは，自律分散システムの利点である**動的なノードの追加・削除が容易である**という利点を有している．ノードを追加または削除した場合には，**Fig.3.5** のようにその局所近傍のみのアークを構成しなおすことで対応し，ノード移動の挙動設計や関数表現方法などは変更する必要がない．ノードを追加する場所はどこであってもノード移動が行われることで適切な分布を形成することが期待されるが，実用的には複雑な領域に追加する方が効率がよい．

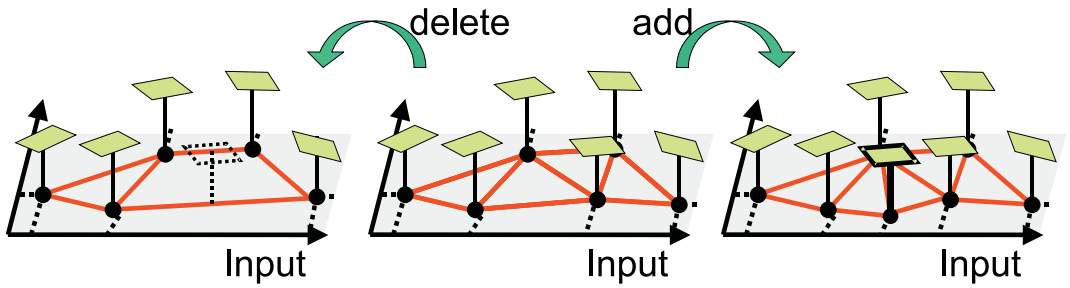

Fig. 3.5　ノードの追加・削除の例

ノードの追加アルゴリズムは提案手法の理論的中心から外れるため 4 章で述べ，本章ではノードを適応的に移動させるための挙動設計を中心に述べる．また，3.6 節においてノードの追加・削除に関連して関数値とノード追加の判断基準の関係について論じる．

以下の節では，(1)，(2) の超平面による関数近似法について述べ，次にグラフ上の反応拡散方程式と自律分散システムについて論じた後で，(3)，(4) の挙動設計に

ついて述べる.

3.3 ノードの位置・勾配更新による関数近似

本節では，関数を表現するための分散配置されたノードの位置および勾配の決定方法を述べる．本研究で提案するノードによる関数近似手法では，近似関数の形状に応じてノードの疎密が動的に変化する．また，強化学習に適用する場合問題によって入力変数の次元が異なるが，次元が異なっても共通のアルゴリズムで対処できることが望ましい．そのため，CAD の分野などでよく用いられる曲面近似理論と異なり，以下の要求がある．

- 2 次元，3 次元などの特定の入力次元に特化したアルゴリズムではなく，任意の次元の空間に共通して適用可能なアルゴリズムであること
- 近傍を表すノード同士の接続は格子状などの一定の形状を想定したアルゴリズムではなく，任意の形状に適用可能なアルゴリズムであること

また，ノードの密度を適応的に変化させた結果，空間的に近くに存在するノードの個数は変化し得る．そのため，近傍ノード[1)]の個数に依存しないアルゴリズムが望ましい．本研究では，1 次元入力 1 次元出力の補間曲線理論を基礎として用い，任意の入力次元，任意の近傍ノード数に適用可能なアルゴリズムを構築した．

入力次元を n とし，ノード u の位置情報を $\boldsymbol{w}_u$ とする．$\boldsymbol{w}_u$ は出力次元を含むため $n+1$ 次元ベクトルであり，

$$\boldsymbol{w}_u \equiv \begin{bmatrix} \boldsymbol{w}_u^x \\ w_u^y \end{bmatrix} \in R^{n+1} \qquad (3.3.1)$$

のように $\boldsymbol{w}_u^x \in R^n$ と w_u^y からなる．また，勾配情報を $\boldsymbol{a}_u \in R^n$ とする．V をノードの集合とする．これらのノードによる (超) 平面の構成を 1 次元，2 次元の場合について表したものが **Fig.3.6**，**Fig.3.7** である．

2 章で述べたように，TD 学習への関数近似の適用において収束性が保証されている関数近似の分類として線形近似手法がある．線形近似手法とは，

$$\Phi^a(\boldsymbol{x}) = \sum_{i=1}^{p} \theta_i \phi_i(\boldsymbol{x})$$

のように関数を表現する方法で，p 個のパラメータ θ_i とそれに対応する p 個の特徴 (feature) と呼ばれる基底関数 $\phi_i(\boldsymbol{x})$ (いずれも $i = 1, \cdots, p$) の線形和で表される．位置と勾配情報をともに更新する関数近似法はこの線形近似手法の分類に属さないため，位置のみを更新する関数近似方法も同時に示す (強化学習への適用の詳細については 4 章で再び詳しく述べる)．以降では，

[1)] 本研究では，近傍ノードをアークで結ばれたノードとして定義する．

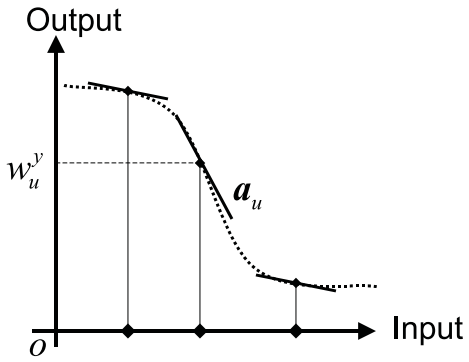

Fig. 3.6 超平面の構成 (1 次元の例)

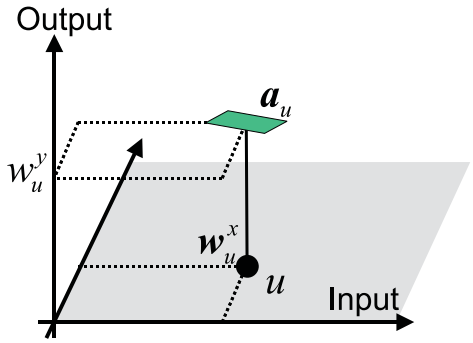

Fig. 3.7 超平面の構成 (2 次元の例)

- 一般の関数近似：位置と勾配を同時に更新
- 強化学習のための関数近似：位置のみの更新，勾配は位置情報から得る

という二つの仕様にあわせ，二通りの方法を各々別に示す．

まずこの関数近似器の出力決定方法として，勾配情報 $\boldsymbol{a}$ を用いる方法と用いない方法の二通りを述べる．入力ベクトル $\boldsymbol{x} \in R^n$ が与えられたとき，

$$b = \arg\min_{u \in V} \|\boldsymbol{x} - \boldsymbol{w}_u^x\| \tag{3.3.2}$$

によって決定される入力ベクトルに最も近いノード b を最整合 (best matching) ノードと呼ぶ．ここで，ノルム $\|\cdot\|$ には，ユークリッドノルム

$$\|\boldsymbol{x}\| \equiv \sqrt{\sum_{i=1}^{n} x_i^2} \tag{3.3.3}$$

を用いる．勾配情報を用いた出力決定は，表現の簡略化のために

$$\tilde{\boldsymbol{x}} \equiv \begin{bmatrix} \boldsymbol{x} - \boldsymbol{w}_b^x \\ 1 \end{bmatrix} \tag{3.3.4}$$

として，出力 y をノード b によって

$$\begin{aligned} y &= \boldsymbol{a}_b^T(\boldsymbol{x} - \boldsymbol{w}_b^x) + w_b^y \\ &= [\boldsymbol{a}_b^T w_b^y]\tilde{\boldsymbol{x}} \end{aligned} \tag{3.3.5}$$

で与える．ここで，T は転置を表す．**Fig.3.8** に 1 次元入力の場合の位置・勾配情報を用いた出力決定方法を示す．勾配情報を用いない出力決定は，単に最整合ノードの位置情報を用いて

$$y = w_b^y \tag{3.3.6}$$

により出力を得る．**Fig.3.9** に同じく 1 次元入力の場合の位置情報のみを用いた出力決定方法を示す．以下に逐次最小二乗法および勾配伝播法について述べ，それぞれの方法について各場合への適用方法を述べる．

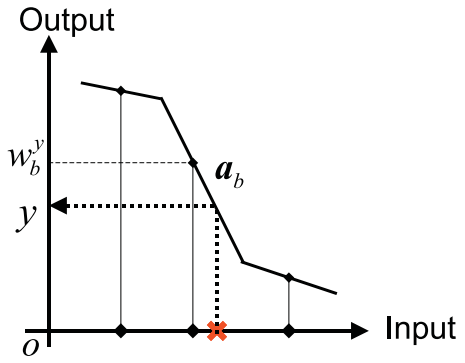

Fig. 3.8 位置･勾配情報による出力

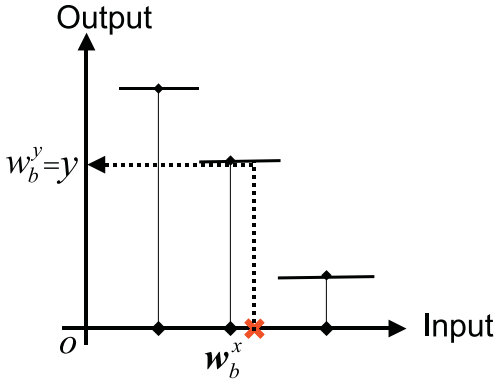

Fig. 3.9 位置情報のみによる出力

3.3.1 逐次最小二乗法

逐次的に入出力情報を得た場合に，逐次最小二乗法によって各データに対する最整合ノードの位置および勾配情報 $w_b^y, \boldsymbol{a}_b$ を更新する方法を示す．

入出力データが繰り返し与えられたときにノード b が N 回最整合になったとする．ノード b が最整合になった N 個のデータを $[\boldsymbol{x}_1^T \ y_1], \cdots, [\boldsymbol{x}_N^T \ y_N]$ とし，行列 A_N とベクトル $\boldsymbol{y}_N$ をそれぞれ

$$A_N = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{w}_b^x & \boldsymbol{x}_2 - \boldsymbol{w}_b^x & \cdots & \boldsymbol{x}_N - \boldsymbol{w}_b^x \\ 1 & 1 & & 1 \end{bmatrix} \tag{3.3.7}$$

$$\equiv [\ \tilde{\boldsymbol{x}}_1 \quad \tilde{\boldsymbol{x}}_2 \quad \cdots \quad \tilde{\boldsymbol{x}}_N]$$

$$\boldsymbol{y}_N = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.3.8}$$

とする．これらの点がノード b で作られる平面上にあるとすると，

$$\boldsymbol{y}_N^T = [\boldsymbol{a}_b^T w_b^y] A_N \tag{3.3.9}$$

が成り立つ．**Fig.3.10** のように点列が同一面上にない場合には，二乗誤差

$$\delta^2 = \|\boldsymbol{y}_N^T - [\boldsymbol{a}_b^T w_b^y] A_N\|^2 \tag{3.3.10}$$

を最小にする w_b^y と $\boldsymbol{a}_b$ を求める．$\mathrm{rank}A_N = n+1$ のとき，A_N の擬似逆行列は $A_N^+ = A^T(AA^T)^{-1}$ のように表される．これを用いて $\boldsymbol{a}_b$ は以下のように計算することができる．

$$[\boldsymbol{a}_b^T w_b^y] = \boldsymbol{y}_N^T A_N^T (A_N A_N^T)^{-1} \quad (3.3.11)$$

この計算は最小二乗法 (LSM) に相当する [柳井 83]．データ行列 A_N とデータベクトル $\boldsymbol{y}_N$ を新しい観測の情報によって逐次的に更新するために，B_N と $\boldsymbol{c}_N$ を以下のように定義する．

$$B_N = A_N A_N^T, \quad \boldsymbol{c}_N^T = \boldsymbol{y}_N^T A_N^T \quad (3.3.12)$$

この両式は，

$$B_N = B_{N-1} + \tilde{\boldsymbol{x}}_N \tilde{\boldsymbol{x}}_N^T, \quad \boldsymbol{c}_N = \boldsymbol{c}_{N-1} + \tilde{\boldsymbol{x}}_N y_N \quad (3.3.13)$$

と逐次表現することができる．各ステップごとに $B_{N-1}, \boldsymbol{c}_{N-1}$ の情報を保存することで B_N および $\boldsymbol{c}_N$ を更新することができる．ただし B_N は $(n+1)\times(n+1)$ 行列，$\boldsymbol{c}_N$ は $(n+1)$ 次元ベクトルであり，そのサイズはデータ数 N に依存しない．このようにして，勾配および位置 $[\boldsymbol{a}_b^T w_b^y]$ は逐次更新パラメータ B_N および $\boldsymbol{c}_N$ を用いて

$$[\boldsymbol{a}_b^T w_b^y] \leftarrow \boldsymbol{c}_N^T B_N^{-1} \quad (3.3.14)$$

のように表現することができる．

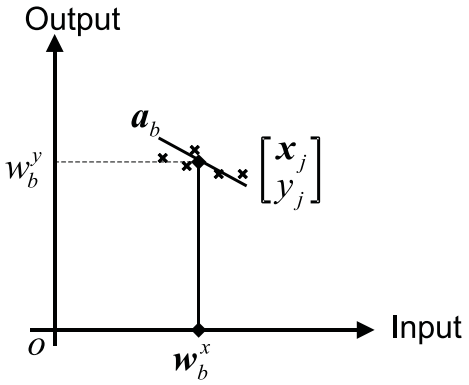

Fig. 3.10　最小二乗法による近似

3.3.2　勾配係数拡散法

与えられた入出力データ $[\boldsymbol{x}^T y]^T$ に対して $[\boldsymbol{a}_b^T w_b^y]$ を更新するのではなく勾配情報を除いた w_b^y のみを更新する方法を述べる．位置のみを更新する場合は，入力 $\boldsymbol{x}$

から (3.3.2) 式に基づいて最整合ノード b を決定し，

$$w_b^y \leftarrow y \tag{3.3.15}$$

として更新する．後に述べるノード移動には勾配情報を用いるため，各ノードの位置情報から勾配を決定する必要がある．ここでは，A.1 節で示す Spline 補間曲線の理論を用いて勾配を決定する方法を示す．

Spline 補間曲線は，各節点の間の 2 次導関数が連続になるような 3 次関数を与える．仮に 1 次元入力 1 次元出力系を想定し，点列 $[x_0\ y_0], \cdots, [x_{i-1}\ y_{i-1}], [x_i\ y_i], [x_{i+1}\ y_{i+1}], \cdots, [x_f\ y_f]$ が $x_0 < \cdots < x_{i-1} < x_i < x_{i+1} < \cdots < x_t$ の順に並んでいるとする．点 i における曲線の 2 次導関数を y_i'' と表すと，点 i と $i+1$ の 2 点間の補間曲線は次式のように表される．

$$y = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}'' \quad (x_i < x < x_{i+1}) \tag{3.3.16}$$

ただし各項は

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i} \tag{3.3.17}$$

$$B = \frac{x - x_i}{x_{i+1} - x_i} \tag{3.3.18}$$

$$C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2 \tag{3.3.19}$$

$$D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2 \tag{3.3.20}$$

のような多項式である．ここから補間曲線上の導関数を導出すると，

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{3A^2 - 1}{6}(x_{i+1} - x_i)y_i'' + \frac{3B^2 - 1}{6}(x_{i+1} - x_i)y_{i+1}'' \tag{3.3.21}$$

$$\frac{d^2y}{dx^2} = Ay_i'' + By_{i+1}'' \tag{3.3.22}$$

となる．複数の点列の間をつなぐときには，1 次導関数が連続になるという拘束条件を用いて各点の 2 次導関数を決定する．例えば 1 次元格子状に配列した点列 $[x_0\ y_0], [x_1\ y_1], \cdots, [x_f\ y_f]$ に対しては，両端点に $y_0' = 0, y_f' = 0$ という境界条件 (あるいは $y_0'' = 0, y_f'' = 0$) を与えることで，$y_0'', y_1'', \cdots, y_f''$ を定めることができる．本研究で用いるノードは，本節冒頭で述べたように，任意の次元，近傍ノード個数に適用できる必要がある．例えば，1 次元入力空間の場合は入力軸正方向と負方向 2 個の近傍ノードが必要になるが，近傍ノードが一つしかない場合は，上記の方法を直接適用することができない．このように近傍の情報が充分に得られない場合にも近似的な解を得る方法を示す．

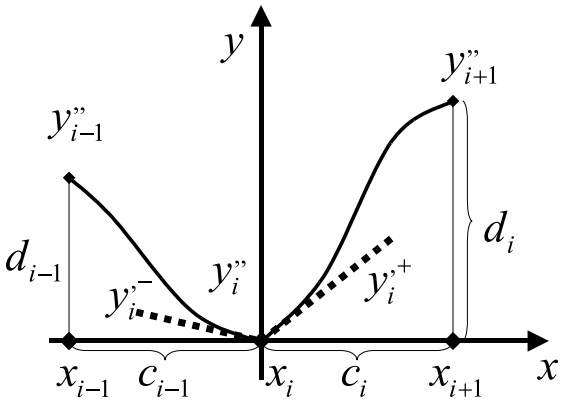

Fig. 3.11　勾配の連続性の利用

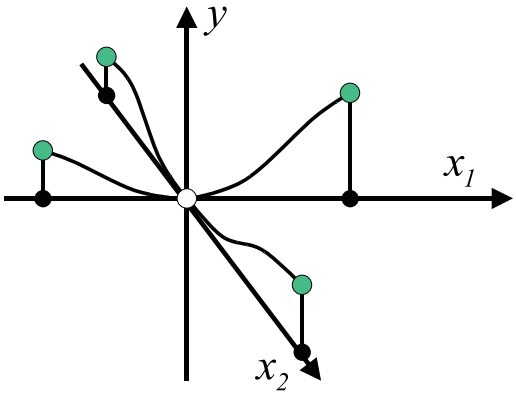

Fig. 3.12　2 次元入力の場合の近傍

まず **Fig.3.11** に示すようなノードが 1 次元格子状に配置されている場合を考える．ここで，$c_i \equiv x_{i+1} - x_i, d_i \equiv y_{i+1} - y_i$ である．正方向と負方向の近傍との関係から構成される各勾配 $y_i'^+, y_i'^-$ はそれぞれ，

$$y_i'^+ = \frac{d_i}{c_i} - \frac{c_i}{3}y_i'' - \frac{c_i}{6}y_{i+1}'' \qquad (3.3.23)$$

$$y_i'^- = \frac{d_{i-1}}{c_{i-1}} + \frac{c_{i-1}}{3}y_i'' + \frac{c_{i-1}}{6}y_{i-1}'' \qquad (3.3.24)$$

のように各近傍の 2 次導関数により表される．点 i における勾配を連続にするためには，二乗誤差 $\delta_i^2 = (y_i'^+ - y_i'^-)^2$ を 0 にすればよい．この 1 次元の例のように各点の正負両方向に近傍の点が存在すれば，$\delta_i = 0$ から導かれる $y_{i-1}'', y_i'', y_{i+1}''$ の関係式を境界から逐次的に解けばよい．多次元の空間に充分なノードが存在しない場合や完全な境界条件が得られない場合には，2 次導関数 y_i'' を逐次的に更新することで二乗誤差 δ_i^2 を減少させる．この二乗誤差は

$$\delta_i^2 = \left(\frac{c_{i-1}+c_i}{3}y_i'' + \frac{c_i}{6}y_{i+1}'' + \frac{c_{i-1}}{6}y_{i-1}'' + \frac{d_{i-1}}{c_{i-1}} - \frac{d_i}{c_i}\right)^2 \qquad (3.3.25)$$

のように表される．これを各点 i の 2 次導関数 y_i'' の関数と見ることで，各点について最急降下法によって二乗誤差の総和を極小に向かわせる．y_i'' の更新式は

$$y_i'' \leftarrow y_i'' + \alpha\frac{\partial}{\partial y_i''}\delta_i^2(y_i'') \qquad (3.3.26)$$

となる．**Fig.3.13** は一次元ガウス関数上に等間隔に 8 点を配置し，逐次更新によって 2 次導関数を更新した結果の各点における勾配と 2 次導関数を線分と縦棒 (点線) で表したものである．この例では $\alpha = 10$ である．初期状態は $y_i'' = 0 (i = 1, \cdots, 8)$ とし，両端点の 2 次導関数は 0 で固定とした．各点における二乗誤差 δ_i^2 の推移を

Fig.3.14 に示す．(3.3.25) 式で表す二乗誤差は，厳密には y_i'' だけでなく y_{i-1}'', y_{i+1}'' の影響も受ける．同じように，y_i'' の変更は $\delta_{i-1}^2, \delta_{i+1}^2$ にも影響を与える．(3.3.26) 式による更新は，この意味で全点における二乗誤差を減らす厳密な最急降下法にはなっていないが，実用上問題がなければこの方法を用いることとする．このような考え方は，微分方程式の解法としては，2 点境界値問題を解くための緩和法 (Relaxation Methods) として知られている [Press92].

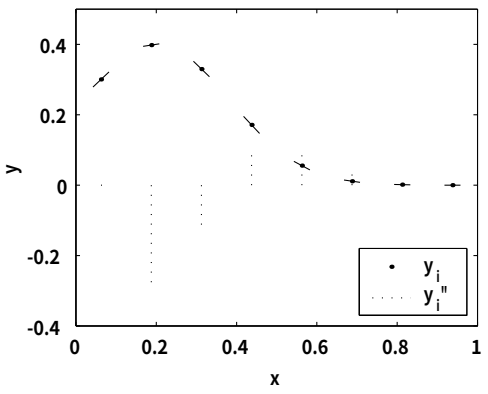

Fig. 3.13 端点の 2 次導関数を 0 とした境界条件での解

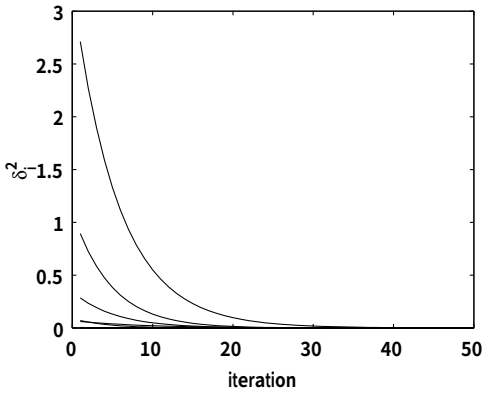

Fig. 3.14 各ノードにおける二乗誤差の推移

この方法を多次元入力に拡張する．近傍が正負両方向に存在する多次元格子のような場合には，**Fig.3.12** のように各入力軸について同様の方法で最急降下法により 2 次偏導関数を決めることができる．**Fig.3.15** はその計算例である．

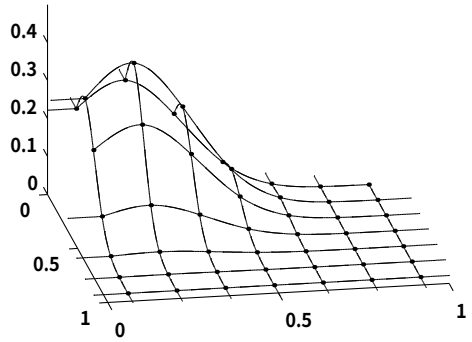

Fig. 3.15 2 次元格子への適用例

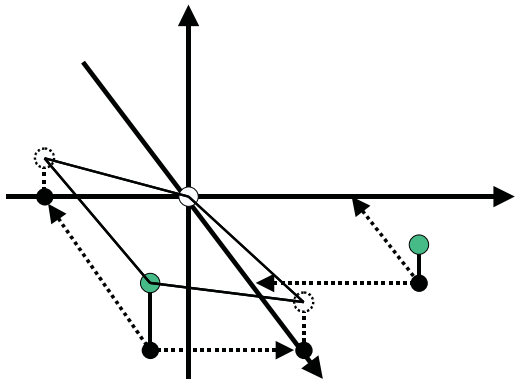

Fig. 3.16 各入力軸への投影

一般的には，近傍ノードが入力軸方向に存在するとは限らない．ここでは，仮想的に近傍ノードが入力軸正負の各方向に存在する状況を作り，その仮想近傍情報をもとに 2 次導関数を更新する．

- 各近傍ノードの位置，2 次導関数情報は各入力軸上の仮想近傍ノードに射影する．位置情報の射影は **Fig.3.16** のように近傍ノード h と u を通る直線と法線ベクトルが平行になる平面によって行う．
- 近傍ノードの無い方向には，位置偏差なし，2 次導関数 0 $(d_I = 0, y''_I = 0)$ の仮想近傍ノードをおく．

という方法をとる． 正負の方向の誤差を 0 に近づけるような繰り返し計算を行う．**Fig.3.18** は，2 次元格子を入力軸座標系に対して $\pi/8$[rad] 回転させた状態での計算例である．

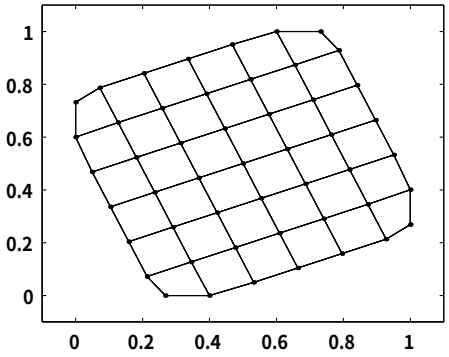

Fig. 3.17　回転した 2 次元格子

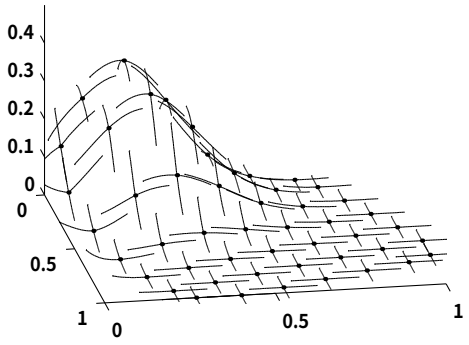

Fig. 3.18　勾配更新の結果

3.4 自律分散システムと反応拡散方程式

前節で述べた位置と勾配情報をもつノードに対して，その局所近傍での複雑度を定義し，各ノードの担う複雑度を均一にするようにノードを移動させることで適応的分解能の達成を目指す．ここで,「局所近傍」「均一化」などの機能を実現する鍵となるのが，自律分散システム理論と勾配系の適用例としてのグラフ上の反応拡散方程式である．本節では，グラフ上の反応拡散方程式とその背景となる自律分散システムについて述べる．

3.4.1 自律分散システムと勾配系

複雑化，大規模化するシステムを設計・制御するためのパラダイムとして自律分散システムが提唱されている [伊藤 95]．自律分散システムとは，集中管理システムの対極に位置する概念で，

全体を統合的に管理するのではなく，システムの構成素が自律的に行動しながら協調・競合的に相互作用しあい，全体として秩序を形成または維持するシステム

と定義されている．システムの構成素 (サブシステム) はシステム全体の情報を管理するのでなく局所的な情報のみから挙動を決定する「局所性」という考え方がある [新 95]．サブシステムが局所的な相互作用から全体的な秩序を形成している場合，サブシステムの追加や削除の対して柔軟に対処しやすい．つまり，自律分散システムは集中管理システムと比較して耐故障性や拡縮性[2)]に優れているという工学的利点を有する．

自律分散系のアルゴリズムの工学的応用研究例としては，伊藤らの自律分散型画像認識法 [伊藤 94] や Luo *et al.* の逆運動学の解法 [Luo98] などを挙げることができる．伊藤らの画像認識法は，画素ごとに完全に分散処理を行うことが可能なモデルであり，結線を近傍の画素に限っている．このため，ハードウェア化ができれば，計算能力を著しく改善させることが期待される．一方，Luo *et al.* の提案する分散計算のアプローチは，

- アルゴリズム・計算方法として分散的並列的なアプローチが複雑・大規模化する問題に有効である

[2)] 拡張・縮小を行うという意味.

- 局所性・並列性を持つ情報処理機構が，生体の情報処理のモデルとして有意義である

という意義を持っていると考えられる．後者については，以下の文脈でとらえることができる．人間の脳神経系には，全体の情報を統合する機能モジュールは存在しないということ，また機能モジュールを個々に設計してそれらをつなぎあわせるようなアプローチは，人間のような高度な知的振る舞いをするロボットを実現する上で限界があることなどが指摘されている [川人 96]．それに対して求められる一つのアプローチは，局所並列性を持った個々の自律的計算要素が全体として秩序を形成し機能を果たすような設計方法であると考えられる．この意味で，後者の意義は広い意味で前者の有効性に含まれると考えられる．

湯浅らは，時空間発展方程式系およびグラフ上の発展方程式系において，ポテンシャル汎関数を極小化する勾配系を構成すると，一般的に (非線形) 反応拡散方程式が得られることを示した [湯浅 99]．これは，システムを設計する立場から見ると，系全体にわたる設計指針 (ポテンシャル汎関数) を定めれば，その指針からみて系の状態を望ましい方向に変化させるための各要素の局所的な挙動を反応拡散方程式に基づいて設計できるということを示している．

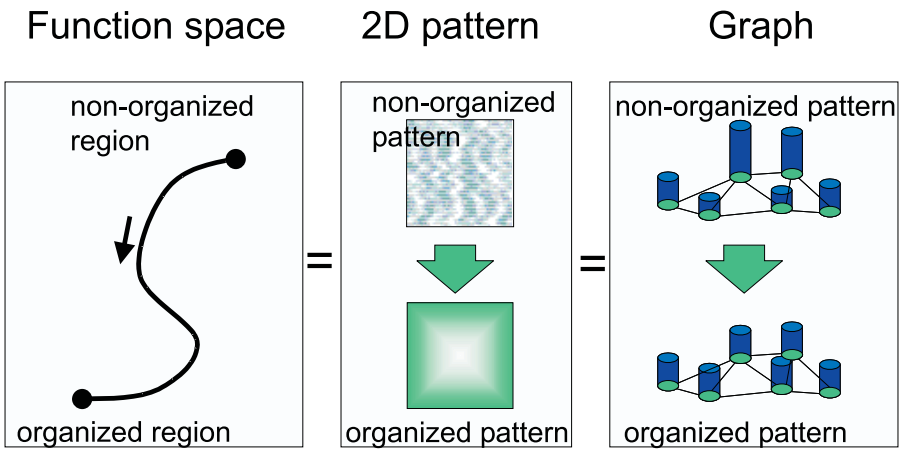

Fig. 3.19　関数空間における勾配系の挙動

Fig.3.19 に関数空間における勾配系の挙動の概念を示す．左図の関数空間において秩序のない状態から秩序のある状態へ遷移する過程は，中図ならば 2 次元のパタン，右図ならばグラフ上の関数値のパタンが無秩序のパタンから何らかの秩序を持ったパタンに遷移することに相当する．この遷移の過程を本研究におけるノードによる関数近似に当てはめると，ノード上に表現された近似関数の「複雑

度が均一である」という秩序を勾配系によって形成するように，各ノードの状態を遷移させる過程がこれに相当する．

3.4.2 グラフ上の反応拡散方程式

有限グラフ $G = (V, E)$ の頂点 (ノード) 集合 V が内部点の集合 $\bar{V}$ と境界点の集合 ∂V からなり，辺の集合 E が内部辺の集合 $\bar{E}$ と境界辺の集合 ∂E からなるとき (**Fig.3.20**)，これを境界付きグラフと呼ぶ[3]．各頂点 u における実数値関数を $f(u)$ と表し，$C(V)$ を V 上の，$C(E)$ を E 上のそれぞれ実数値関数全体の空間とする．

$$df(e) = f(t(e)) - f(o(e)) \tag{3.4.1}$$

を対応づける作用素 $d : C(V) \to C(E)$ を余微分といい，df を f の勾配という．ここで，$o(e)$ は辺 e の始点となる頂点，$t(e)$ は辺 e の終点となる頂点である．**Fig.3.21** にその説明を示す．各ノード上に描かれている濃い色の円柱はノード上の関数値を表し，ノード u とアーク e_i で結ばれたノード上の薄い色の円柱は $df(e_i)$ を表す[4]．

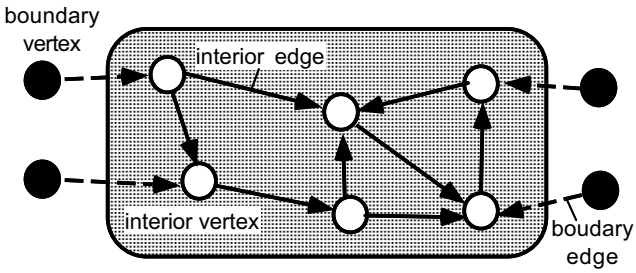

Fig. 3.20 境界付きグラフ

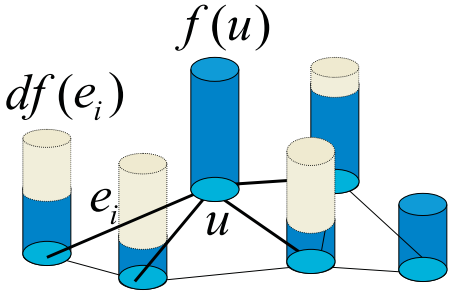

Fig. 3.21 関数 $f(u)$ と $df(e)$

この f の関数である汎関数 $W(f)$ を，系全体の状態を表す関数として定義する．

$$W(f) = W_0(f) + W_1(df) \tag{3.4.2}$$

$$W_0(f) = \alpha_0 \sum_{u \in \bar{V}} F_r(f(u)) \tag{3.4.3}$$

$$W_1(df) = \alpha_1 \sum_{u \in \bar{V}} F_d^{(u)}(\mathbf{df}(u)) \tag{3.4.4}$$

(3.4.5)

[3] 正確な定義は付録 B 章参照．
[4] 図中では有向アークはいずれも u を終点とする向きであるとしている．

ただし，$E(u)$ を u を頂点とする辺の集合とし，$\mathbf{df}(u) = [df(e)]_{e \in E(u)}$ である．これをポテンシャル汎関数とする関数空間での勾配系を構成する．ポテンシャル汎関数 $W(f)$ を極小値に向かわせるための関数 f の変化を以下のように与えることができる．

$$\frac{\partial f}{\partial t} = -\frac{\delta W(f)}{\delta f} \qquad (3.4.6)$$

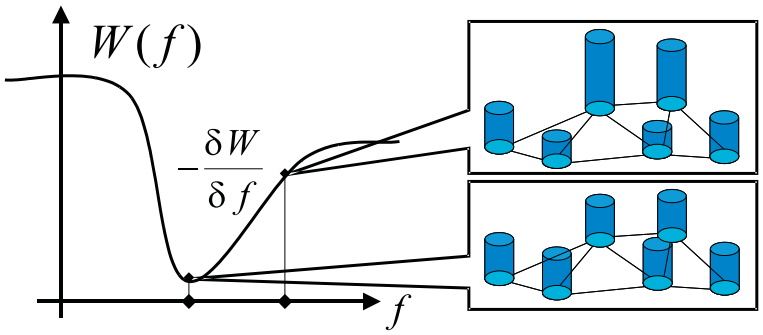

Fig. 3.22　関数空間における最急降下の概念図

Fig.3.22 に関数空間における最急降下法の概念を表す．ここで $W(f)$ の最小値はシステム全体の望ましい状態を表しており，この例では各ノードの関数値が均一になることが望ましい状態に相当する．そこに向かうように関数値 f を変化させることでシステムを望ましい状態に近づけることができる．このときの各関数値 f の変化量は (3.4.6) 式のようにポテンシャル汎関数 W を f で偏微分したものとして与えられる．

3.5 反応拡散方程式に基づいたノードの挙動設計

3.2 節で述べたノードにグラフ上の反応拡散方程式を適用し，前述の複雑度に即した分布を実現するための各ノードの挙動 (移動方向) 設計について述べる．ただしノードを移動したときには，ノードの移動モデルに従って近似関数の形状を保つように位置・勾配を更新する．以下に，

(1) 境界付きグラフへの適用
(2) 各ノード上の関数 $f(u)$ とポテンシャル汎関数 $W(f)$ の定義
(3) 近似関数曲面に沿ったノードの移動モデル
(4) 関数値の位置微分によるノードの挙動設計

の順で述べる．

3.5.1 境界付きグラフへの適用

各ノード u は 3.3 節で述べたように，位置情報 $\boldsymbol{w}_u$ および勾配情報 $\boldsymbol{a}_u$ を持つ．さらに，**Fig.3.23** のように辺によって接続されたノード (近傍ノード) と情報を交換することができる．近傍ノードおよび境界ノードは以下のように定義される．

- 各辺はノードの生成・削除や移動に対して動的に生成，削除される
- 境界に近いノードは境界ノード (境界点) を認識し，境界辺によって結ばれる
- 境界ノードは境界辺で結ばれるノード u と同じ関数値 $f(u)$ をもち，$\boldsymbol{w}_u^x$ を境界上に射影した位置，勾配は $\boldsymbol{a} = \boldsymbol{0}$ であると仮定する (ノイマン条件)

近傍を表すアークを構成する方法には，一定距離内のノードを結ぶ方法，近傍の一定個数のノードを結ぶ方法，Voronoi 図を作成する方法 [杉原 94] などがある．前者の二つは，単純なアルゴリズムで記述できる反面，問題の次元やスケールに依存して距離や個数などを調節しないと適切な近傍関係が構築できないという欠点を持つ．また，Voronoi 図の作成アルゴリズムは理論的な厳密さが保証される反面，空間の次元を特定したアルゴリズムであるために任意の次元に共通して適用可能という要請に応えることができない．本研究では，比較的少ない計算量で任意の次元の空間に対して近傍関係を柔軟に得る方法として TRN(Topology Representing Networks) アルゴリズム [Martinetz94] を用いる．本研究では近傍を構築する部分のアルゴリズムを用いるが，TRN アルゴリズム自身は各ノードが入力ベクトルに

対して移動する過程も含んでいる．そのため，動的にノード同士の位置関係が変わるような問題と相性がよい．TRN の具体的なアルゴリズムについては付録 B.2 節で述べる．

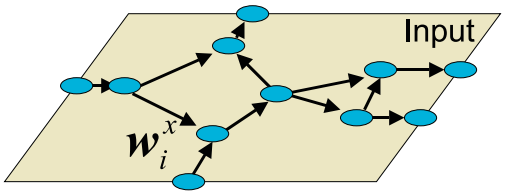

Fig. 3.23 境界付きグラフへの適用

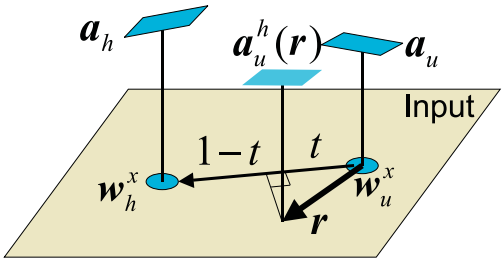

Fig. 3.24 ノードの移動と勾配変化

3.5.2 グラフ上の関数 $f(u)$ とポテンシャル汎関数 $W(f)$ の設計

ノード u の関数値 $f(u)$ は，その近傍における複雑度としてノードの密度を反映した勾配変化を表現するために，

$$f(u) = \frac{1}{m(u)} \sum_{h \in N(u)} \|\boldsymbol{a}_u - \boldsymbol{a}_h\|^2 \|\boldsymbol{w}_u^x - \boldsymbol{w}_h^x\|^2 \qquad (3.5.1)$$

のように定義する[5]．ここで $m(u)$ はノード u の次数 (近傍ノードの数)，$N(u)$ はノード u の近傍ノードの集合である[6]．また，以降の記述の簡略化のために

$$\boldsymbol{b}_{hu} \equiv \boldsymbol{a}_h - \boldsymbol{a}_u \qquad (3.5.2)$$

$$\boldsymbol{c}_{hu} \equiv \boldsymbol{w}_h^x - \boldsymbol{w}_u^x \qquad (3.5.3)$$

を定義する．また，各ノードについて関数値 f を大きくし，近傍との差を減らして均一にするという設計指針に基づき，汎関数を以下のように定める．

$$W_0(f) = \alpha_R \sum_{u \in \bar{V}} f(u) \qquad (3.5.4)$$

$$W_1(df) = \beta_D \sum_{u \in \bar{V}} \| \mathbf{df}(u) \|^2 \qquad (3.5.5)$$

ただし $\alpha_R < 0, \beta_D > 0$ である．ポテンシャル汎関数 $W(f)$ は $W_0(f)$ と $W_1(df)$ の和として表されるが，$W_0(f)$ は関数値 f を大きくすることで，$W_1(df)$ は関数値の

[5] この定義については，3.6 節で考察を行う．

[6] ノード u の次数は $\deg(u)$，ノード u の近傍ノードの集合は $\{v|v \sim u\}$ と表記するのが一般的である [浦川 96] が，本論の後の表記の便宜上このように定義した．

近傍との差を減らして均一化することで $W(f)$ を極小値に導くことができる．実際の挙動設計においては，拡散項 $W_1(df)$ がより重要な役割を持っていると考えられる．

3.5.3 近似関数曲面に沿ったノードの移動モデル

複雑度を表す関数値は，ノードの近傍との位置関係および勾配変化によって決まる．ノードは移動によって近傍との位置関係を変え，勾配を変化させる．これにより関数値を変化させることができる．このとき，ノードが移動したときに近似曲面を維持するように位置と勾配を変化させるような移動モデルが必要になる．ここでは，ノード u が入力空間内を $\boldsymbol{r} \equiv \Delta \boldsymbol{w}_u^x$ 移動したときの勾配を $\boldsymbol{a}_u(\boldsymbol{r})$ と表し，その変化を表すモデルについて述べる．

$$\boldsymbol{a}_u(\boldsymbol{r}) = \sum_{h \in N(u)} q_h^u \boldsymbol{a}_u^h(\boldsymbol{r}) \qquad (3.5.6)$$

のように，各近傍ノード h との関係により決定される勾配 $\boldsymbol{a}_u^h(\boldsymbol{r})$ の重み付き和として表す．ここで重み係数 q_h^u は距離の近い近傍の影響を強くするために距離の二乗に反比例する重みを正規化したものとして，

$$q_h^u = \frac{1/\|\boldsymbol{c}_{hu}\|^2}{\sum_{i \in N(u)} 1/\|\boldsymbol{c}_{iu}\|^2} \qquad (3.5.7)$$

により与える．以下では，特定の近傍ノード $h \in N(u)$ との関係による勾配変化 $\boldsymbol{a}_u^h(\boldsymbol{r})$ について述べる．

位置・勾配の変化は $t = \frac{\boldsymbol{r} \cdot \boldsymbol{c}_{hu}}{\|\boldsymbol{c}_{hu}\|^2}$ の関数として表し (**Fig.3.24**),

(1) $\boldsymbol{c}_{hu}$ 方向成分は，近傍ノード h の位置 w_h^y・勾配 $\boldsymbol{a}_h$ との間を滑らかにつなぐために Ferguson 曲線 [黒瀬 99] で表現する (**Fig.3.25**).

(2) $\boldsymbol{c}_{hu}$ に直交する成分は，t に応じた線形補間によって勾配を $\boldsymbol{a}_h$ に近づける (**Fig.3.26**).

Ferguson 曲線 $P(s) = [P_{x_1}(s) \cdots P_{x_n}(s)\ P_y(s)]^T$ は，位置，方向ベクトルを持った 2 点を s を媒介変数とする 3 次多項式により滑らかに結んだ曲線である．媒介変数を用いていることから，任意の次元へ適用しやすいという利点がある．本研究での適用の詳細については付録 A.2 で述べる．

勾配の方向成分と直交成分を分離するために，ノード u から h へ向かう単位ベクトル $\boldsymbol{n}_c = \frac{\boldsymbol{c}_{hu}}{\|\boldsymbol{c}_{hu}\|}$ に対するヌル空間の直交基底を $\boldsymbol{n}_1 \cdots \boldsymbol{n}_{n-1}$ とし，行列 N を

$$N \equiv [\boldsymbol{n}_1 \cdots \boldsymbol{n}_{n-1}] \qquad (3.5.8)$$

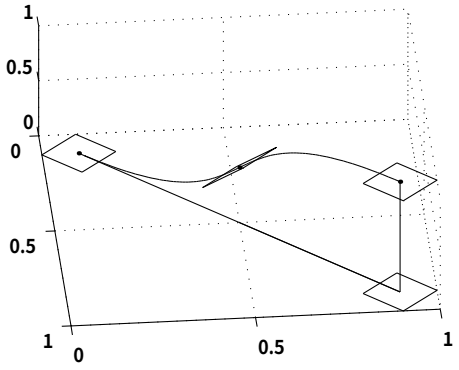

Fig. 3.25 Ferguson 曲線モデル

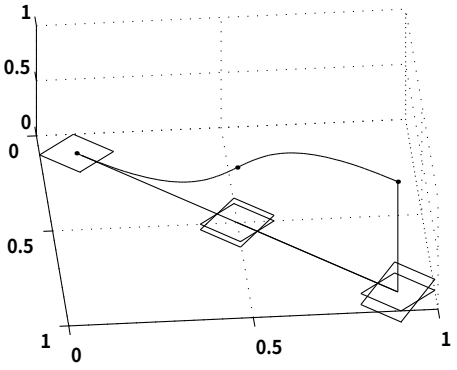

Fig. 3.26 線形補間モデル

と定義する． Ferguson 曲線により決定される勾配を $\boldsymbol{a}_1(s)$，線形補間による勾配を $\boldsymbol{a}_2(t)$ とし，

$$\boldsymbol{a}_u(\boldsymbol{r}) = (\boldsymbol{a}_1(s) \cdot \boldsymbol{n}_c)\boldsymbol{n}_c + NN^T\boldsymbol{a}_2(t)^T \tag{3.5.9}$$

として勾配変化を記述する．各成分の変化は，

$$\boldsymbol{a}_1(s) = \left[\frac{\dot{P}_y}{\dot{P}_{x_1}}, \cdots, \frac{\dot{P}_y}{\dot{P}_{x_n}}\right], \quad \dot{p} = \frac{dP}{ds} \tag{3.5.10}$$

$$\boldsymbol{a}_2(t) = (1-t)\boldsymbol{a}_u + t\boldsymbol{a}_h \tag{3.5.11}$$

のように表される．各項の $\boldsymbol{r}$ による微分は

$$\frac{\partial \boldsymbol{a}_u(\boldsymbol{r})}{\partial \boldsymbol{r}} = \frac{\partial t}{\partial \boldsymbol{r}}\frac{\partial s}{\partial t}\left(\frac{d\boldsymbol{a}_1(s)}{ds} \cdot \boldsymbol{n}_c\right)\boldsymbol{n}_c + \frac{\partial t}{\partial \boldsymbol{r}}N\frac{d\boldsymbol{a}_2(t)}{dt}N^T \tag{3.5.12}$$

と表される．

3.5.4 関数の位置微分によるノードの挙動設計

上記で定義した $\boldsymbol{a}_u(\boldsymbol{r})$ を用いて，ノードが u が $\boldsymbol{r}$ 移動したときの関数値を

$$f_u(\boldsymbol{r}) \equiv \frac{1}{m(u)}\sum_{h \in N(u)} \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \|\boldsymbol{r} - \boldsymbol{c}_h^u\|^2 \tag{3.5.13}$$

と定義する．$f(u)$ を変化させるためのノード u の移動方向は，この $f_u(\boldsymbol{r})$ を $\boldsymbol{r}$ で微分し，$\boldsymbol{r} = \boldsymbol{0}$ を代入することで得られる．ここで，$g_{hu}(\boldsymbol{r})$ を

$$g_{hu}(\boldsymbol{r}) \equiv \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \|\boldsymbol{r} - \boldsymbol{c}_h^u\|^2 \tag{3.5.14}$$

と定義すると，

$$f_u(\boldsymbol{r}) = \frac{1}{m(u)} \sum_{h \in N(u)} g_{hu}(\boldsymbol{r}) \tag{3.5.15}$$

と表せる．$g_{hu}(\boldsymbol{0})$ を得ることで，$\left.\frac{\partial f_u(\boldsymbol{r})}{\partial \boldsymbol{r}}\right|_{\boldsymbol{r}=\boldsymbol{0}}$ は

$$\left.\frac{\partial f_u(\boldsymbol{r})}{\partial \boldsymbol{r}}\right|_{\boldsymbol{r}=\boldsymbol{0}} = \frac{1}{m(u)} \sum_{h \in N(u)} \left(\left.\frac{\partial g_{hu}(\boldsymbol{r})}{\partial \boldsymbol{r}}\right|_{\boldsymbol{r}=\boldsymbol{0}} \right) \tag{3.5.16}$$

と求められる．この方向にノードを移動させることで関数値 $f(u)$ を任意に変化させることができる．なお，(3.5.12) 式を用いた $\frac{\partial g_{hu}(\boldsymbol{r})}{\partial \boldsymbol{r}}$ の計算は付録 B.3 に記す．

(3.4.6) 式 における関数値の勾配法による設計によりポテンシャル汎関数を極小にする解に導くことができるが，これにはノードの関数値を他のノードの関数値とは独立に制御できるという仮定が含まれている．本研究での関数値の定義では，ノード u を移動させて関数値 $f(u)$ を更新すると，(3.5.1) 式にしたがってその近傍の関数値も変化してしまう．このような近傍への影響を考慮するために，近傍も含めたノード u の反応項 $D_u^R(\boldsymbol{r})$ および拡散項 $D_u^D(\boldsymbol{r})$ を

$$D_u^R(\boldsymbol{r}) = f_u(\boldsymbol{r}) + \sum_{d \in N(u)} f_d^u(\boldsymbol{r}) \tag{3.5.17}$$

$$D_u^D(\boldsymbol{r}) = \sum_{d \in N(u)} (f_u(\boldsymbol{r}) - f_d^u(\boldsymbol{r}))^2 \tag{3.5.18}$$

と定義する．ここで $f_d^u(\boldsymbol{r})$ は，近傍ノード u が $\boldsymbol{r}$ 移動したときのノード d の関数値を表す．ポテンシャル汎関数はこの D_u^R, D_u^D を用いて，

$$W(f) = \frac{\alpha_R}{2} \sum_{u \in \bar{V}} D_u^R + \frac{\beta_D}{2} \sum_{u \in \bar{V}} D_u^D \tag{3.5.19}$$

と改める． これらを直接 $\boldsymbol{r}$ で微分し，

$$\frac{\partial D_u^R(\boldsymbol{r})}{\partial \boldsymbol{r}} = \sum_{h \in N(u)} \left(\frac{1}{m(u)} + \frac{1}{m(h)} \right) \frac{\partial g_{hu}(\boldsymbol{r})}{\partial \boldsymbol{r}} \tag{3.5.20}$$

$$\frac{\partial D_u^D(\boldsymbol{r})}{\partial \boldsymbol{r}} = \sum_{d \in N(u)} \left\{ (f(u) - f(d)) \sum_{h \in N(u,d)} m_{hu}^D \frac{\partial}{\partial \boldsymbol{r}} g_{hu}(\boldsymbol{r}) \right\} \tag{3.5.21}$$

を得る．ただし $N(u,d)$ は $N(u)$ からノード d を除いた集合，$m_{hu}^D \equiv \frac{1}{m(u)} - \frac{1}{m(h)}$ である．ポテンシャル汎関数を減少させるための各ノード u の移動方向 $\boldsymbol{v}_u$ は，

$$\boldsymbol{v}_u = \alpha_R \left.\frac{\partial D_u^R(\boldsymbol{r})}{\partial \boldsymbol{r}}\right|_{\boldsymbol{r}=\boldsymbol{0}} + \beta_D \left.\frac{\partial D_u^D(\boldsymbol{r})}{\partial \boldsymbol{r}}\right|_{\boldsymbol{r}=\boldsymbol{0}} \tag{3.5.22}$$

と定めることができる．

3.6 関数値定義に関する考察

本節では，本提案手法において用いている複雑度に相当する関数値の妥当性に関する議論を行う．第一に，関数値定義の直観的理解と多次元空間問題の考察を行う．第二に，関数値の均一化が近似誤差最小化の問題と関係が深いことを示す．第三に，関数値の妥当な水準,「どの程度まで関数値が減衰すれば関数近似にとって十分か」という問題に対する数値解析と考察を行う．

3.6.1 直観的理解と多次元空間近似

ノード 上に定義された関数 $f(u)$ の内容は，

$$f(u) = \frac{1}{m(u)} \sum \Big\{ (\text{勾配変化})^2 \times (\text{位置の差})^2 \Big\} \qquad (3.6.1)$$

というものであった．形状表現の研究では，ガウス曲率に基づいてメッシュの分割を判断するという方法がとられている [山本 99]．曲線 $y = \Phi(x)$ の曲率は，

$$\kappa = \frac{\Phi''(x)}{(1 + \Phi'(x)^2)^{3/2}} \qquad (3.6.2)$$

と定義される．この式から，本手法は，$\Phi'(x)$ の変化の影響を無視して曲率の代わりに勾配変化を基準として用いているものととらえることができる．一方,「勾配変化に応じた密度を実現する」という方針を仮に「勾配変化とノード 密度が比例する」と理解すると，

$$\begin{aligned} (\text{勾配変化}) &\propto (\text{ノード 密度}) \\ &\equiv 1(\text{個})/(\text{ノード 間隔}) \\ (\text{ノード 間隔}) \times (\text{勾配変化}) &= (\text{一定}) \end{aligned}$$

というように，勾配変化とノード 間隔の積が一定になることとノード 密度が勾配変化に比例することが等価であるという直観的理解ができる．ただし，ノード の密度がノード 間隔の逆数になるのは測度がノード 間隔に相当する $n = 1$ の場合のみであり，次に述べる近似誤差最小化問題も 1 次元での議論である．

本来の設計指針に理想的に従うには，ノード 間隔の項がその近傍での (例えば Voronoi 図によって分割された領域の面積や体積に相当する) 測度に対応していることが望ましいが，位置と勾配情報のみをもったノード によるグラフ上に次元に依存した量を直接反映させることは難しい．多次元の場合には，**Fig.3.27** のよう

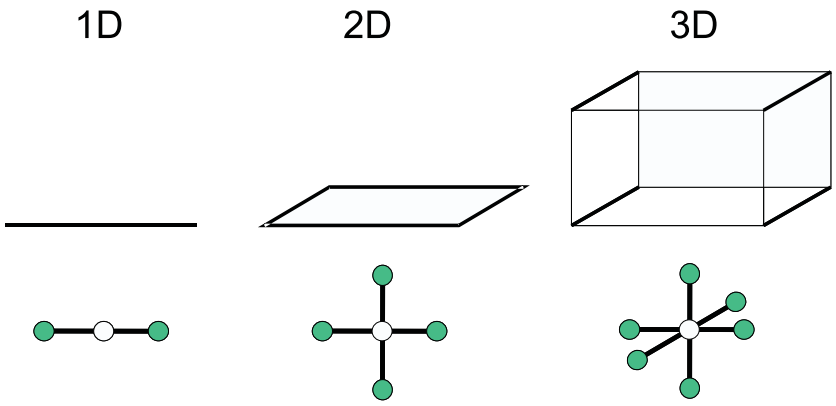

Fig. 3.27　次元の変化に対する近傍ノード数の変化

に次元数の増大に伴って近傍を表すアークが増加する．本研究で近傍関係の構築に用いている TRN アルゴリズムでも，入力ベクトルの次元を反映した数のアークを形成することができる．本研究では，次元の増大に対しては近傍ノード数の増加で対処し，近傍ノードと自身の間で定義される勾配変化と位置の差の積を各近傍について加えて平均を取ることで近似的にノード密度を表現することとする．

3.6.2　近似誤差最小化問題との関係

2 章で考察したように，本提案手法の強化学習価値関数近似への適用の目的は勾配変化を適切に表現できることであって近似精度の向上は必ずしも第一の目的ではないが，1 次元では近似誤差最小化と対応が取れる．位置と勾配を持ったノードによって行う関数近似の区間 $[x_i\ x_{i+1}]$ における近似誤差の積分を求め，各ノードが誤差を最小にするために満たすべき極値の条件から関数値定義との関係を論じる．条件として，**Fig.A.1** に示すように各ノードの節点における位置，1 次，2 次導関数が得られるものとする．

点列 $\{x_i, y_i\}(i = 1, \cdots, N)$ が近似関数 $\Phi^t(x)$ 上に存在するとする．つまり，各ノードは目標近似関数 $\Phi^t(x)$ 上に存在し，各点上における勾配情報を持っているとする．区間 $[x_i\ x_{i+1}]$ における $\Phi^t(x)$ の 2 次までの Taylor 展開は，$\frac{x_i+x_{i+1}}{2}$ を境に以下のように表される．

$$\Phi^t(x) = \begin{cases} y_i + y_i'(x-x_i) + \frac{1}{2}y_i''(x-x_i)^2 + \mathcal{O}^3(x-x_i) & (x_i \le x \le \frac{x_i+x_{i+1}}{2}) \\ y_{i+1} + y_{i+1}'(x-x_{i+1}) + \frac{1}{2}y_{i+1}''(x-x_{i+1})^2 + \mathcal{O}^3(x-x_{i+1}) & (\frac{x_i+x_{i+1}}{2} \le x \le x_{i+1}) \end{cases} \quad (3.6.3)$$

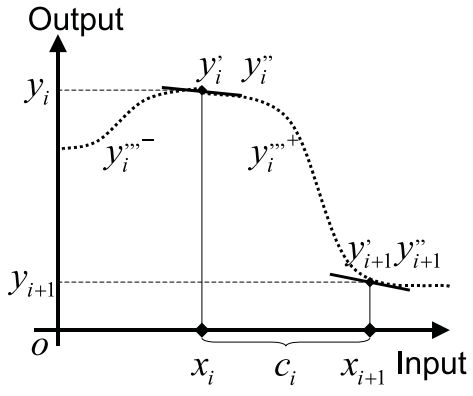

Fig. 3.28　区間 $[x_i\ x_{i+1}]$ における曲線補間

Taylor 展開を用いるのは，次式に示すノードによる近似 $\Phi^a(x)$ との差の計算が容易であるためである．

$$\Phi^a(x) = \begin{cases} y_i + y_i'(x - x_i) & (x_i \le x \le \frac{x_i+x_{i+1}}{2}) \\ y_{i+1} + y_{i+1}'(x - x_{i+1}) & (\frac{x_i+x_{i+1}}{2} \le x \le x_{i+1}) \end{cases} \tag{3.6.4}$$

$\Phi^a(x)$ と $\Phi^t(x)$ の近似誤差を各区間にわたって積分したものの総和を E_p とすると，E_p は以下のように表される．

$$\begin{aligned} E_p &= \sum_{i=1}^{N-1} \left(\int_{x_i}^{\frac{x_i+x_{i+1}}{2}} \frac{y_i''}{2}(x - x_i)^2 dx + \int_{\frac{x_i+x_{i+1}}{2}}^{x_{i+1}} \frac{y_{i+1}''}{2}(x - x_{i+1})^2 dx \right) \\ &= \sum_{i=1}^{N-1} \left\{ \left[\frac{y_i''}{6}(x - x_i)^3 \right]_{x_i}^{\frac{x_i+x_{i+1}}{2}} + \left[\frac{y_{i+1}''}{6}(x - x_{i+1})^3 \right]_{\frac{x_i+x_{i+1}}{2}}^{x_{i+1}} \right\} \\ &= \frac{1}{48} \sum_{i=1}^{N-1} (y_i'' + y_{i+1}'')(x_{i+1} - x_i)^3 \end{aligned} \tag{3.6.5}$$

この積分誤差を各 x_i で偏微分したときに，それぞれについて $\frac{\partial E}{\partial x_i} = 0$ となるような x_i を考える．ここで，$c_i = x_{i+1} - x_i$ とする．

$$\begin{aligned} 48\frac{\partial E}{\partial x_i} &= y'''^-(x_i - x_{i-1})^3 + 3(y_{i-1}'' + y_i'')(x_i - x_{i-1})^2 \\ &\quad - y'''^+(x_{i+1} - x_i)^3 - 3(y_i'' + y_{i+1}'')(x_{i+1} - x_i)^2 \\ &= y_i'''^- c_{i-1}^3 - y_i'''^+ c_i^3 + 3(y_i'' + y_{i-1}'')c_{i-1}^2 - 3(y_i'' + y_{i+1}'')c_i^2 \\ &= 0 \end{aligned} \tag{3.6.6}$$

ただし，y'''^-, y'''^+ は **Fig.3.28** に示すようにそれぞれ区間 $[x_{i-1}\ x_i], [x_i\ x_{i+1}]$ の Spline 補間曲線における 3 次導関数である．Spline 補間曲線の定義から，y'''^-, y'''^+ は両

端の 2 次導関数を用いて

$$y_i'''^{-} = \frac{y_i'' - y_{i-1}''}{c_{i-1}}, \quad y_i'''^{+} = \frac{y_{i+1}'' - y_i''}{c_i} \tag{3.6.7}$$

のように表すことができる．(3.6.6) 式の下 2 段の方程式を i に関する漸化式とみると，

$$3(y_{i-1}'' + y_i'')c_{i-1}^2 + y_i'''^{-}c_{i-1}^3 = 3(y_i'' + y_{i+1}'')c_i^2 + y_i'''^{+}c_i^3 \tag{3.6.8}$$

という等式が得られ，(3.6.7) 式を用いると

$$c_{i-1}^2(2y_i'' + y_{i-1}'') = c_i^2(2y_{i+1}'' + y_i'') \tag{3.6.9}$$

とすることができる．すなわち，各ノードにおける関数値 f_i の定義を

$$f_i = c_i^2(y_i'' + 2y_{i+1}'') \tag{3.6.10}$$

のように各リンクについて定めれば，$f_0 = f_1 = f_2 = \cdots = f_{N-1}$ とすることが近似誤差 E_p が最小となるための必要条件となる．Spline 補間式における 1 次と 2 次導関数の関係式

$$6y_{i+1}' = 6\frac{y_{i+1} - y_i}{c_i} + c_i(y_i'' + 2y_{i+1}'') \tag{3.6.11}$$

を代入すると，(3.6.10) 式は

$$f_i = 6\left(c_i y_{i+1}' - (y_{i+1} - y_i)\right) \tag{3.6.12}$$

のように書き換えることができる．$y_i' \simeq \frac{y_{i+1}-y_i}{c_i}$ ならば[7]，$f(u) = c_i(y_{i+1}' - y_i')$ という幅と勾配変化の積の定義が近似誤差最小の条件に合致することになる．

3.6.3 関数値基準に関する考察

関数値 $f(u)$ はそのノード近傍での近似関数の複雑度を表している．関数近似が充分に達成されていない場合は，この複雑度 f が均一になるようにノードを移動させると同時に，複雑度の大きいノードの近傍に新たにノードの追加を行うことで複雑度を低減することができる．逆に，複雑度の極端に小さい領域では，そのノードを削除しても関数近似には大きな影響は与えないと考えられる．

では，複雑度がどの程度であれば新規にノードを追加すべきかという判断はできないだろうか (**Fig.3.29**)．本章で提案した自律分散型の関数近似手法では，近

[7] あるいは $6\frac{y_{i+1}-y_i}{c_i} \gg c_i(2y_i'' + y_{i+1}'')$ ならば

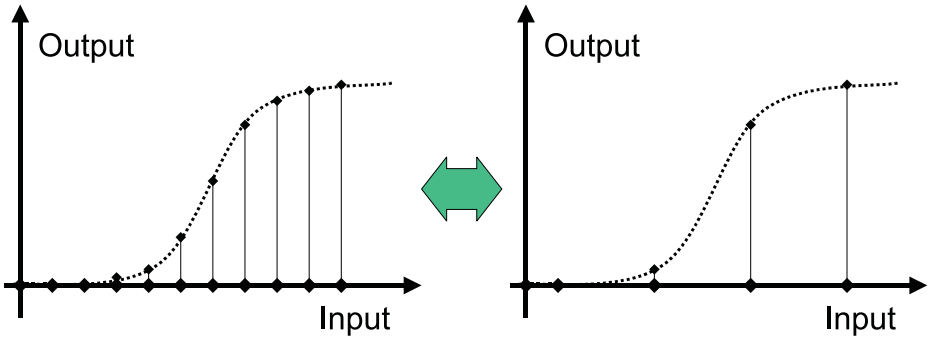

Fig. 3.29　適切なノード密度の目安

似関数の局所的な形状に注目し複雑度を定義している．これは，ノードの動的な挙動設計を可能にすると同時に,「新規ノードを追加するとしたらどこに追加するべきか」という問題に指針を与えるものでもある．ここでは,「どの程度ノードを追加して複雑度を均一にしていれば充分に関数近似が達成されたといえるか」という基準について議論する．

1 章で述べた時系列信号のモデルパラメータ推定との比較では，モデルの次数を上げることが関数値の基準を小さくすることに対応していると考えられる．AR モデルや ARMA モデルなどの時系列信号の線形モデルのパラメータ最尤推定においては，モデル次数の増加とモデルの予測性能の低下の間のトレードオフを吟味した判断基準として，AIC(Akaike Information Criterion) が知られている．

この問題は，関数近似問題独立の問題ならば関数近似をどの程度精度よくしたいかによって異なり，強化学習適用問題ならば，問題によって学習達成に必要な近似精度も異なる．ここでは一つの目安として，サンプリング定理を用いた方法を示し，シミュレーションにおける評価のときの一つの比較基準を提示することを目的とする．

ここで行う考察の仮定として，

- ノードはある程度充分な密度で配置されている

とする．サンプリング定理とは，

周波数領域に変換された信号を回復するためには，周波数の最大帯域の 2 倍のサンプリング周波数でデータをサンプリングしなくてはならない

というものである [美多 84]．サンプリング理論は時間波形とその周波数領域への

変換との関係を論じたものであるが，空間波形と空間周波数の関係についても同じことが言える．本章で述べるノードによる関数近似では，ノード間隔をサンプリング周期に相当するものと考えることができる[8]．ノード間隔が決まっているとして，関数値と適切なノード間隔の関係から「最低限満たすべき関数値の条件」を導く．簡略化のため 1 次元で考え，A.1 節で述べる Spline 補間曲線を用いる．

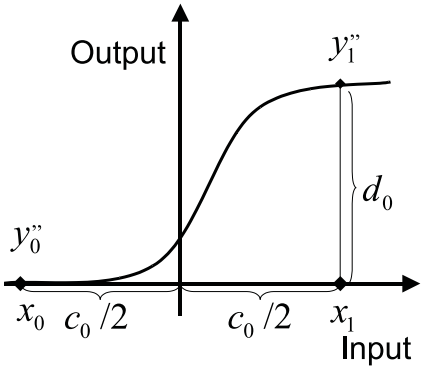

Fig. 3.30 ノード 0 と 1 を結ぶ Spline 曲線

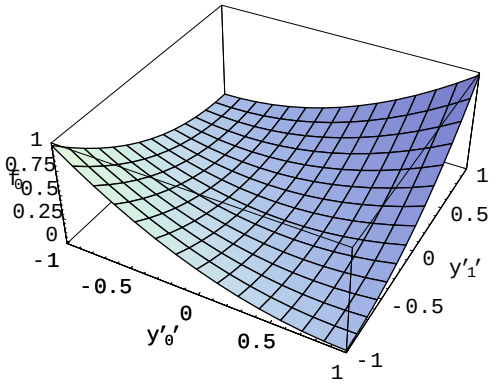

Fig. 3.31 y_0'', y_1'' と関数値 f_0 の関係

各ノードにおける関数の定義式は

$$f(u) = \frac{1}{m(u)} \sum_{h \in N(u)} \|\boldsymbol{a}_u - \boldsymbol{a}_h\|^2 \|\boldsymbol{w}_u^x - \boldsymbol{w}_h^x\|^2 \qquad (3.6.13)$$

であった．ここで $m(u)$ はノード u の次数 (近傍ノードの数)，$N(u)$ はノード u の近傍ノードの集合である．**Fig.3.30** のようにノード 0 と 1 の中点を原点に移動させた曲線を考える．ここで，$c_0 \equiv x_1 - x_0, d_0 \equiv y_1 - y_0$ とする．簡略化のため一方の近傍のみとの関係を用いて関数定義を書き直すと

$$f_0 = (y_1' - y_0')^2 c_0^2 \qquad (3.6.14)$$

となる．Spline 補間曲線における 1 次導関数と 2 次導関数の関係式 (A.1.8) 式 および (A.1.9) 式から，

$$y_0' = \frac{y_1 - y_0}{c_0} - \frac{c_0}{3} y_0'' - \frac{c_0}{6} y_1'' \qquad (3.6.15)$$

$$y_1' = \frac{y_1 - y_0}{c_0} + \frac{c_0}{6} y_0'' + \frac{c_0}{3} y_1'' \qquad (3.6.16)$$

[8] 各ノードが位置だけでなく勾配情報を持っているとするとデータのサンプリングとは条件が異なるが，強化学習に適用する場合は位置情報のみを更新するため実質的にデータのサンプリングと等価であると考える．

となるので，これを用いて y_0', y_1' を消去すると，f_0 は

$$f_0 = \frac{1}{4}(y_0'' + y_1'')^2 c_0^4 \qquad (3.6.17)$$

のように両端点の 2 次導関数 y_0'', y_1'' とノード間隔 c_0 によって表せる．**Fig.3.31** に $c_0 = 1$ としたときの y_0'', y_1'' に対する f_0 の変化を示す.

一方，2 点の 2 次導関数およびその位置関係 c_0, d_0 が決まれば 2 点間を通る Spline 補間曲線を決定することができ，その式は (3.3.16) 式で示したように

$$y = B(x)d_0 + C(x)y_0'' + D(x)y_1'' \qquad (-c_0/2 \le x \le c_0/2) \qquad (3.6.18)$$

となる．ただし，

$$A(x) = \frac{c_0/2 - x}{c_0} \qquad (3.6.19)$$

$$B(x) = \frac{x + c_0/2}{c_0} \qquad (3.6.20)$$

$$C(x) = \frac{1}{6}(A(x)^3 - A(x))c_0^2 \qquad (3.6.21)$$

$$D(x) = \frac{1}{6}(B(x)^3 - B(x))c_0^2 \qquad (3.6.22)$$

である．サンプリング定理を適用すると，この曲線の最大空間周波数を ω_0 としたとき，ノード間隔 c_0 を空間周波数とすると $1/c_0$ となるので

$$2\omega_0 \le 1/c_0 \qquad (3.6.23)$$

であれば，ノードは近似関数を復元するのに充分な密度で存在していることになる.

アナログ信号のスペクトル解析をするためには，通常は Fourier 変換

$$\hat{\Phi}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t}\Phi(t)dt \qquad (3.6.24)$$

を用いる．Fourier 変換は時間領域全域にわたる情報を用いるものであるが，時間的に局所的なスペクトル情報を得るためには Fourier 解析においては時間を局所化するための窓関数を用いる．このように窓関数 $g_w(t)$ をかけて Fourier 変換

$$\hat{\Phi}(\omega) = \int_{-\infty}^{\infty} g_w(t)e^{-i\omega t}\Phi(t)dt \qquad (3.6.25)$$

を行うものを短時間 Fourier 変換 (Short Time Fourier Transform,SFT) とよぶ．この短時間 Fourier 変換において，窓関数にガウス関数

$$g_w(t, \alpha) = \frac{1}{2\sqrt{\pi\alpha^2}} e^{\frac{t^2}{4\alpha^2}} \qquad (3.6.26)$$

を用いたものを Gabor 変換と呼ぶ [チャールズ 94]．変換前の信号は原点に平行移動しているので，窓関数も原点中心とする．

Fig.3.32 は，$c_0 = 1, d_0 = 0.5$ において，$y_0'' = 1, y_1'' = 1$ の曲線を Gabor 変換した結果である．横軸は周波数，縦軸は窓関数の幅を決めるパラメータ α である．窓の幅を広げるにしたがってスペクトルも変化することがわかる．この場合，サンプリング周期は $T = c_0$ であるから，サンプリング定理から許容される周波数帯域は $\omega_0 = 1/2c_0$ となる．窓の幅 α を $c_0/2$ の前後で伸縮したときにこの ω_0 以上の周波数成分が増加しなければ，この曲線は曲率に対して適切なノード間隔を確保しているといえる．

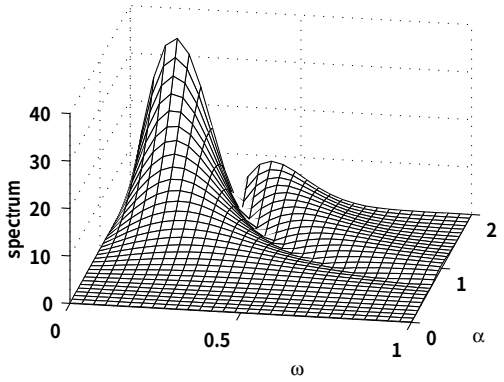

Fig. 3.32 周波数と窓関数の幅に対するスペクトル

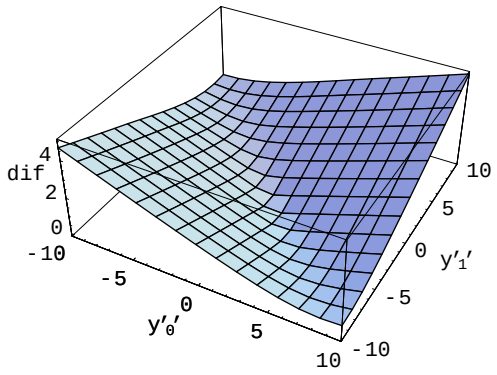

Fig. 3.33 y_0'', y_1'' に対するスペクトル変化

そこで，上記の Gabor 変換について，ω_0 を固定し α を $c_0/2$ の前後で変化させて差分をとった関数を **Fig.3.33** に示す．ここでは α は $c_0/3$ から $c_0/2$ まで変化させて差分をとっている．縦軸および横軸は (y_0'', y_1'') である．**Fig.3.31** の関数値のグラフと対応した形をしており，関数値が大きくなるような曲率の大きい曲線ほど，サンプリング周期付近での周波数成分変化が大きいことがわかる．この曲面を差分が 0 になる平面で切断した切り口を $d_0 = 0.5, -0.5$ の場合についてそれぞれ示したのが **Fig.3.34**，**Fig.3.35** である．縦軸，横軸はいずれも y_0'', y_1'' である．

この切断面に対応する関数値 f_0 を (3.6.17) 式に基づいて計算することにより，各 c_0, d_0 に対してサンプリング定理から導かれる f_0 の水準を求めることができる．c_0, d_0 のスケーリングが異なると対応する関数値も変化するが，基本的な形状は変化しない．5 章において実際の関数近似問題での関係を実際に求めることにより検証を行う．

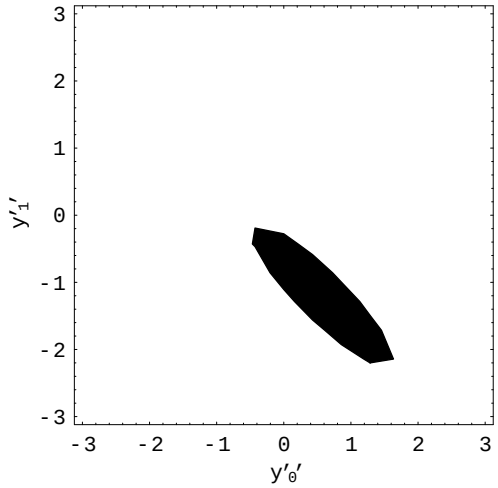

Fig. 3.34 $d_0 = 0.5$ のときの切断面

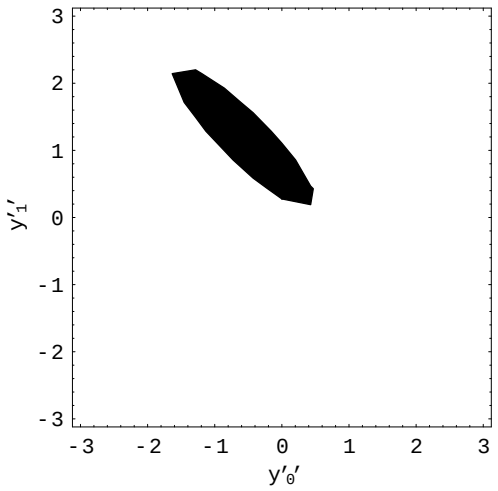

Fig. 3.35 $d_0 = -0.5$ のときの切断面

3.7 おわりに

本章では，本論文の核となる，自律分散的手法に基づいた関数近似法について述べた．

基本的な設計方針としては，位置・勾配情報を持ったノードにより超平面を構成し，関数近似を行う．各ノードの間に TRN アルゴリズムによってアークを張り，局所近傍を表す．各ノードにおいてその周辺の複雑度を関数値 $f(u)$ として定義し，その複雑度を各ノードについて均一にするようにノードを移動させる．本章では特に，ノードを動的に再配置するための挙動設計を中心に述べた (逐次的なノードの追加やアークの再構成については 4 章で述べる)．

関数近似方法には最小二乗法により位置・勾配を更新する方法，位置のみを更新し，位置情報から勾配を決定する方法の二つを述べた．

3.5 節では，3.4 節で述べたグラフ上の反応拡散方程式の考えに基づき，ノードの密度を適応的に変化させるための挙動設計について述べた．各ノードに対してその近傍における複雑度を定義し，その複雑度を系全体が均等に負担するようにノードを移動させるという方法をとった．

3.6 節では，3.5 節における挙動設計の妥当性に関する議論を行った．ノードの複雑度の定義に対しては，その妥当性に関する考察および近似誤差最小化問題との関係についての考察を行った．また，本提案手法が自律分散システムである重要な利点であるノードの逐次的な追加や削除について，追加や削除の判断基準に関する考察を行った．

第4章　関数近似法の強化学習への適用

4.1 はじめに

本章では，3 章で述べた関数近似方法を強化学習問題に適用する方法とその際の問題点について述べる．同時に，本研究の適用に適する問題と適さない問題群の整理を行う．また，強化学習問題の様々な拡張に関する適用可能性とその効用に関する考察を行う．

4.2 節では，まず，本研究の提案する関数近似手法を用いるのに適した問題と適さない問題に関する整理を行う．続いて，2.4 節での議論を踏まえ，本研究で提案する関数近似方法を強化学習に適用する際に問題になることについて述べる．

4.4，4.5 節では，5 章において評価する本研究での強化学習への具体的な適用方法を示す．4.4 節では Value Gradient，4.5 節では Q-learning にそれぞれ適用する方法を述べる．

4.6 節では，プログラム形式でアルゴリズムの具体的な流れを示す．ノード移動や近傍構築などの過程に要する計算時間とノード数，入力次元の関係について考察する．

4.7 節では，本研究で扱わないその他の強化学習の拡張に関して，提案する関数近似を用いた場合の効果，適用可能性についての考察を行う．

4.2 強化学習問題適用時の問題

4.2.1 適用方法の概略

Fig.4.1 に強化学習への関数近似法適用の概略を示す．図中左側の灰色領域は，強化学習の試行錯誤ループである．図中右下の灰色領域は，関数近似器の更新過程を表している．すなわち，強化学習の試行ループとは独立に，一定の間隔で関数近似器を更新するという方法をとっている．

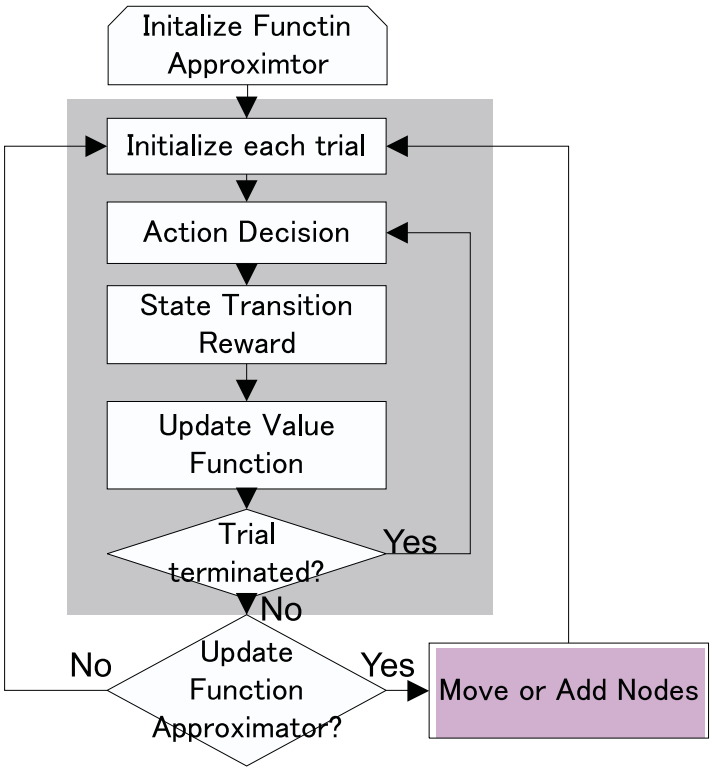

Fig. 4.1 適用アルゴリズムの概略

4.2.2 提案関数近似法の性質

3 章で述べた関数近似法は，グラフ内の各ノード上に定義される複雑度 $f(u)$ に基づいて挙動を設計するものであった．この挙動設計の前提には，目標近似関数 $\Phi^t(\boldsymbol{x})$ について，連続性，微分可能性を仮定している．このため，近似関数 $\Phi^t(\boldsymbol{x})$ が不連続になるような問題には本手法は適していない．

強化学習における状態価値関数における不連続性の例としては，1 章で示した崖のある迷路環境をあげることができる．この環境において崖の境界付近では，価値関数の値が不連続に変化することが予想される．また，行動価値関数の近似に用いる際には，状態空間における価値関数の性質だけでなく，行動空間における行動価値の不連続性がある場合に注意を要する．

4.2.3 各適用方法とその性質

2 章で紹介したように，強化学習の適用方法としては以下のものを挙げることができる．

(1) 状態価値関数 $V(\boldsymbol{x})$ のみの近似：Value Gradient
(2) 行動価値関数 $Q(\boldsymbol{x}, \boldsymbol{u})$ の近似：Q-learning(Advantage Updating)
(3) 状態価値関数 $V(\boldsymbol{x})$ および行動決定器 $\boldsymbol{u} = \pi(\boldsymbol{x})$ の近似：Actor-Critic

Q-learning のように価値関数の中に行動決定情報が埋め込まれる方法に対して，Actor-Critic のように方策を陽に学習する方法にはいくつかの利点があることが指摘されている [Sutton98]．一つには，方策の表現を確率的に行うことにより部分観測マルコフ決定過程 (POMDP) に対応する能力を持たせることができる点 [木村 96] が挙げられる．行動決定器の関数近似問題と TD 学習における価値関数近似問題は若干性質が異なるため [木村 00]，ここでは，TD 学習における価値関数近似の際に起きる問題に対象を絞って考察する．

3.3.1 節で述べた逐次最小二乗法による位置と勾配の逐次推定を行う方法は，関数近似手法としては LWR(Locally Weighted Regression)[Atkeson97a] などと同じ局所線形回帰と等価なものと見なすことができる．**Fig.4.2** に示すように，LWR ではデータを直接記憶し，入力 (query point) が与えられたときには，その近傍の点のいくつかを用いて線形回帰を行う．この過程はデータを直接記憶することと逐次更新パラメータに記憶させることの違いはあるが，データから線形回帰を行うという意味では本研究の逐次最小二乗法による関数近似と等価である．

それに対し，2 章において，

$$\Phi^a(\boldsymbol{x}) = \sum_{i=1}^{p} \theta(i)\phi_i(\boldsymbol{x})$$

のようなパラメータ線形な近似手法は収束が証明されていること [Tsitsiklis97]，および階層型ニューラルネットワークのような非線形の近似手法では発散の可能性があること [Bertsekas96] を紹介した．局所線形回帰の近似方法は，上記の線形近

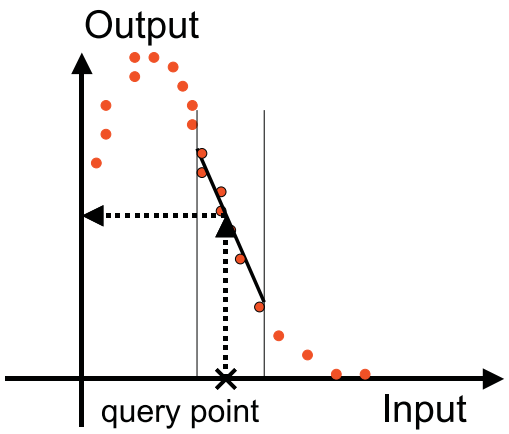

Fig. 4.2 局所線形回帰の例

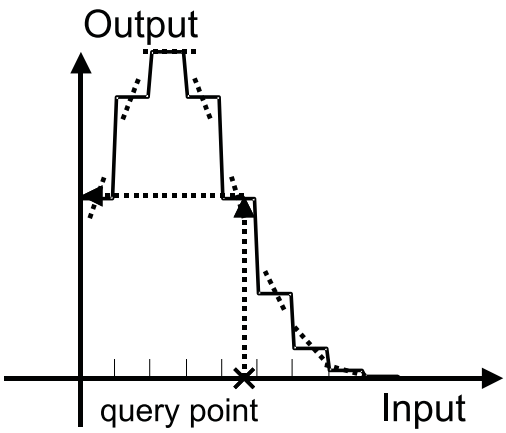

Fig. 4.3 位置更新による関数近似

似手法に当てはまらない．価値関数更新で広く用いられているのはブートストラップ型の近似であり，関数近似器自身の出力を用いた新しい目標関数から勾配を形成し，次の更新にはこの勾配出力を用いることになり，挙動を不安定にさせる．特に外挿を行う近似方法は発散の危険性が高いことが指摘されている [Gordon95]．LWR が直接強化学習に適用されていない [Atkeson97b] のもこのためであると考えられる．

本研究では，このような問題を回避するため，強化学習の価値関数近似に適用するときは勾配情報は更新せず位置情報のみを更新する関数近似を行う．**Fig.4.3** はその 1 次元の例であり，最整合ノードの位置情報を直接出力に用いる．3.3.2 節で示したように，勾配情報は位置情報から決定する．このような近似は，CMAC のタイルコーディング手法において，タイリングを単一にして単純な格子状表現に限り，汎化作用を無くしたものと等価と考えられる．

4.3 適用方法1：一般的な関数近似

強化学習適用時に生じるブートストラップの弊害の無い一般の関数近似適用時には，逐次最小二乗法による (3.3.13) 式の更新，あるいは位置更新と拡散による (3.3.15) 式,(3.3.26) 式 の更新のどちらを用いてもよい．一般に同じノード数ならば，得られるデータ数が多い場合は，位置のみの更新よりも逐次最小二乗法の方が精度良い近似が可能であると考えられる．**Table 4.1** に一般の場合の関数近似の具体的なプロセスを示す.

Table 4.1　一般的な関数近似適用のアルゴリズム

(1) ノードの初期配置：
(2) 関数近似ループ：N_{update} 回繰り返し
　(i) 入出力データ観測：$(\boldsymbol{x}, y)$ を得る
　(ii) 最整合ノード b の計算：
　(iii) ノード b の位置・勾配情報の更新：
(3) ノード移動ループ：各ノード u について N_{move} 回繰り返し
　(i) 反応拡散方程式に基づく挙動決定，微小量移動
　(ii) 自身および近傍の関数値の更新
(4) ノード追加・削除：$f_u > \theta$ ならばノードを追加
(5) (2) に戻る

(1) ノードの初期配置は，入力次元 n に対して n 次元格子状にする．これは，近似関数の形状に関する事前知識がないことを仮定し，状態空間中均一にノードを分布させるためである.

(2),(3) N_{update}, N_{move} は関数近似ループとノード移動ループの回数 (の割合) を決める変数である．前者に対して後者を大きくすると，関数近似結果に敏感にノード移動を行うことができる．しかし，関数近似ループ回数が少なすぎると，ノード移動による関数近似の劣化の影響が大きくなるおそれがある．本提案手法の利点の一つである「(SOM と異なり) 単一の入力に対してグラフ全体のノード移動が行えるため高速な適応が可能である」という性質を生かすためには，後者を大きくすることが求められる．非定常関数近似問題において近似関数の変化に敏感に分布を変化させたい場合は N_{move} を比較的大きくとることで目標近似関数の変化

に対して適応的に分解能を変更する関数近似を行うことができる．逆に，定常関数近似問題で充分にデータが得られる場合は，N_{update} を比較的大きくとることで劣化の影響を小さくとどめることができる．

(4) 新規ノードの追加は関数値を基準にして行うかどうかの判断を行い，追加する場所の決定も関数値をもとに決定する．ノードを追加する際の新規ノードの位置ベクトル $\boldsymbol{w}$ に関しては，原理的には厳密な最適位置を追求する必要はない．ノード移動を行うことで，適応的に適切な位置に移動することが期待されるためである．しかし，複雑度の高い領域に生成する方が必要なノード移動の量を低減できる．そのため，

- すでにあるノードのうち，関数値最大のノードの近傍に新規ノードを生成する
- 関数値最大のノードの近傍ノードのうち，関数値に対する寄与が最大のアーク上にノードを生成する

という方法で新規ノードの位置ベクトルを決定する．具体的には **Table 4.2** のような手続きによる．

Fig.4.4 に，ノードの追加方法の説明を示す．関数値最大のノードを図 (a) 中白ノードとする．この白ノードの接続する近傍ノードは 4 つあるが，そのうち関数値への寄与の最大のものを選ぶ．図 (a) では左側のノードがそれに相当する．新規ノードはこの二つのノードの中間点に新しく追加する．図 (b) の白ノードが新規追加されたノードである．このとき，もとの関数値最大ノードの近傍情報 (アーク) は全消去する．最後に，TRN アルゴリズムによって近傍情報を再構成し，図 (c) のようにアークを張りなおす．

Table 4.2　新規ノード追加のアルゴリズム

(1) 関数値最大のノードを発見する

$$b_f = \arg\max_{u \in \tilde{V}} f(u) \qquad (4.3.1)$$

(2) ノード b_f と接続する近傍ノードのうち，関数値に対する寄与の最大のもの h_f を発見する (**Fig.4.4**(a))

$$h_f = \arg\max_{h \in N(b_f)} \|\boldsymbol{a}_h - \boldsymbol{a}_{b_f}\|^2 \|\boldsymbol{w}_h^x - \boldsymbol{w}_{b_f}^x\|^2 \qquad (4.3.2)$$

(3) 新規ノード b_n をノード b_f と h_f の中点に生成し，ノード b_f の近傍情報を消去する (**Fig.4.4**(b))

$$\boldsymbol{w} = \frac{\boldsymbol{w}_{b_f} + \boldsymbol{w}_{h_f}}{2} \qquad (4.3.3)$$

(4) ノード b_f 周辺にデータ $\boldsymbol{x}$ を複数回ランダムに与え，TRN アルゴリズムにより位相関係を再構成する (**Fig.4.4**(c))

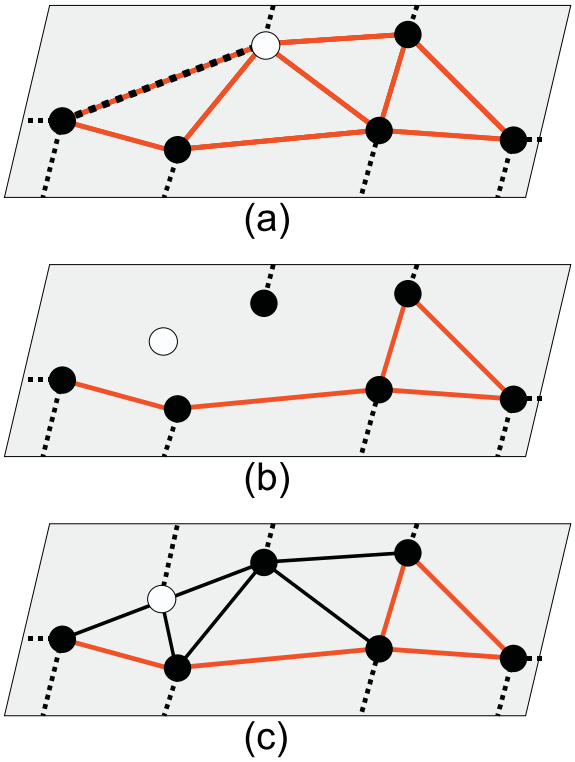

Fig. 4.4　ノード追加の方法

4.4 適用方法2：Value Gradient

強化学習適用の第一例として Value Gradient の例を示す．Value Gradient は，Actor-Critic と異なり，状態価値関数 $V(\boldsymbol{x})$ から直接行動を決定する方法である．つまり，Actor(行動決定器) のアルゴリズムや行動価値関数近似の性能ではなく，状態価値関数近似の適切さが直接行動学習性能に反映される学習法であるといえる．**Table 4.3** にアルゴリズムを示す．ここではタスクを終端状態[1] s_f の存在するエピソード的タスク (episodic tasks) と仮定している．本研究では初期状態から終端状態にいたる状態遷移の系列を試行 (trials) と呼ぶ．行動出力は連続値で m 次元 ($\boldsymbol{u} \in R^m$) とする．

Table 4.3 Value Gradient 適用のアルゴリズム

(1) 状態価値関数 $V(\boldsymbol{x})$ の初期化：ノードの位置情報$\boldsymbol{w}$の初期化
(2) 各試行の初期化： 各試行における初期状態 $\boldsymbol{x}_0$ を決定
(3) 状態観測：$\boldsymbol{x}_t$ を得る
(4) 行動決定：

- ε の確率で乱数により行動決定
- それ以外の場合：$j = 1, \cdots, m$ に対して

$$u_j = u_j^{\max} S\left(\frac{\partial F(\boldsymbol{x}_t, \boldsymbol{u})}{\partial u_j}\frac{\partial V(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) \qquad (4.4.1)$$

(5) 状態遷移：$\boldsymbol{x}_{t+1}$ を得，報酬 r_t を観測
(6) 状態価値関数 $V(\boldsymbol{x}_t)$ の更新：

$$\delta_t = r_t + \gamma V(\boldsymbol{x}_{t+1}) - V(\boldsymbol{x}_t) \qquad (4.4.2)$$

$$V(\boldsymbol{x}_t) \leftarrow V(\boldsymbol{x}_t) + \alpha\delta_t \qquad (4.4.3)$$

(7) 試行終了判定：$\boldsymbol{x}_{t+1} \in s_f$ ならば，ノード移動を行い，(2) に戻る
(8) 試行内繰り返し：(3) に戻る

[1] エピソード的タスクにおいては，終端状態に到達すると開始状態 s_0 へのリセットが行われる．このような特殊な状態遷移を伴わずに状態遷移が無限に続くタスクは連続タスク (continuing tasks) と呼ばれる．

(4) (2.3.22) 式の S は値域 $[0,1]$ の単調増加関数であり，連続値行動の場合は

$$S(x) = \frac{1}{1+\exp(-x)} \qquad (4.4.4)$$

で表されるシグモイド関数を用いることが多い．行動出力が最大値と最小値に限定されるような場合は

$$S(x) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases} \qquad (4.4.5)$$

のようなステップ関数を用いる．

(6) 状態価値関数の参照および更新は全て位置情報に基づく．すなわち，(4.4.2) 式における状態価値関数参照には (3.3.6) 式を用い，(4.4.3) 式における状態価値関数更新には，(3.3.15) 式を用いる．

(7) ノードの移動は，関数近似器の出力を変えないように位置と勾配を変化させるように設計されているが，前節でも述べたように，移動を多数回繰り返したときに完全に同じ出力を維持することが保証されるわけではない．また，4.2 節で述べたような収束証明のなされているパラメータ線形近似の仮定については，ノード移動が起こると

$$\Phi^a(\boldsymbol{x}) = \sum_{i=1}^{p} \theta(i)\phi_i(\boldsymbol{x})$$

という近似式における基底関数 $\phi_i(\boldsymbol{x})$ の形状が変化するため収束証明の仮定を逸脱する．本適用では，ノード移動に関するこの二つの悪影響を避けるために，ノード移動のダイナミクスは強化学習のそれより小さくする．

Table 4.4　ノード移動頻度決定のアルゴリズム

(1) 状態価値関数 $V(\boldsymbol{x})$ の初期化：
(2) 強化学習試行ループ：
Table 4.3 の (2)〜(7),(8) からなる強化学習の試行を $N_{reinforce}$ 回繰り返す
(3) ノード移動ループ：
ノードの位置情報をもとに勾配情報を決定する (3.3.2 節参照)
Table 4.1 のノード移動ループを N_{move} 回繰り返す
(4) (2) に戻る

前節の表記にならってノード移動のダイナミクスを **Table 4.4** に示す．上記のノード移動のダイナミクスは，N_{move} に対して $N_{reinforce}$ を大きくすることで実現される．また，強化学習適用時には前述のように位置情報のみの更新を行っているため，ノード移動を行う前には 3.3.2 節で述べた方法を用いて勾配情報を決定する．

4.5 適用方法3：Q-learning

広く一般に知られている Q-learning は，離散状態 s と離散行動 a の組に対する評価値 $Q(s,a)$ をテーブル形式で保持するものである．これを関数近似によって連続な状態空間に拡張した手法として Lin *et al.* の提案した QCON がある [Lin93]. **Fig.4.5** にその機構を示す．m 個の行動要素 $a_1, a_2, \cdots, a_m$ に対して，Q 値 (行動価値関数) を推定する階層型ニューラルネットワークを別々に用意するという方法である．Rummery *et al.* は，QCON のアルゴリズムを改良し，適格度トレース (eligibility trace) を用いたアルゴリズムである Q(λ) をオンライン計算で行う方法を提案している [Rummery94]．この方法の欠点は，行動の種類が多い場合には多数のネットワークが必要になり，行動空間の連続性が表現できないことである．行動空間に連続性を扱える枠組みとして，Baird の提案する Advantage Updating がある [Baird94]．Advantage Updating では，連続状態 $\boldsymbol{x}$ と連続行動 $\boldsymbol{u}$ の組に対する評価を $Q(\boldsymbol{x},\boldsymbol{u})$ として表す[2)]．関数近似器は連続状態変数，連続行動変数を入力とし行動価値 (スカラ値) を出力とする関数近似を行う．行動決定時には，現在の状態 $\boldsymbol{x}$ を用いて関数近似器を参照し，評価値を最大にする行動を探索して出力する．

$$\boldsymbol{u} = \arg\max_{\boldsymbol{u}'} Q(\boldsymbol{x},\boldsymbol{u}') \qquad (4.5.1)$$

この方法は，連続値行動空間を関数近似で表現するために行動空間の汎化が期待できるが，その反面連続値空間の最大値探索に多くのオンライン処理を要するという問題を抱える．

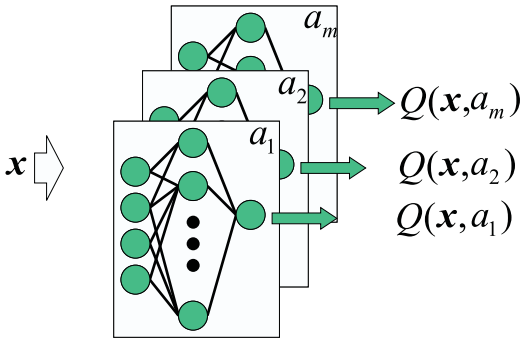

Fig. 4.5 QCON[Lin93]

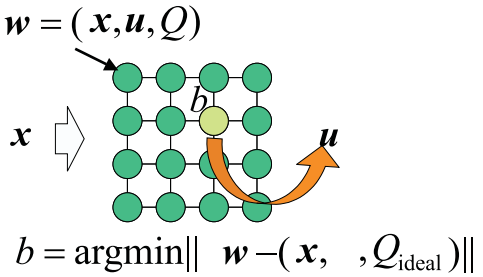

Fig. 4.6 Q-kohon[Touzet97]

Touzet の提案する Q-kohon は，Kohonen の自己組織化マップ (SOM) を用いてこの最大値探索を単純化したものととらえることができる [Touzet97]．**Fig.4.6** に

[2)] 厳密には状態価値関数と行動価値関数に近い意味を持つ Advantage 関数 $A(\boldsymbol{x},\boldsymbol{u})$ の両者を持つが，行動決定時には後者のみを用いるためこのように略記した．

示すように，連続値状態 $\boldsymbol{x}$，連続値行動 $\boldsymbol{u}$ に対する評価値を $Q(\boldsymbol{x}, \boldsymbol{u})$ として，SOM の各結合係数ベクトル $\boldsymbol{w} \in R^{n+m+1}$ を

$$\boldsymbol{w} = [\boldsymbol{x}^T \ \boldsymbol{u}^T \ Q]^T \qquad (4.5.2)$$

と定める．ただし n は状態変数の次元，m は行動出力変数の次元とする．理想的な Q 値を Q_{ideal} とし，最整合計算を

$$b = \arg\min_i \|\boldsymbol{w}_i^{xq} - [\boldsymbol{x}^T \ Q_{ideal}]^T\|^2 \qquad (4.5.3)$$

のように行う．ここで $\boldsymbol{w}_i^{xq} \in R^{n+1}$ は結合係数ベクトルのうち状態と評価値のみを取り出した $n+1$ 次元ベクトルである．この最整合ノード計算が，Advantage Updating における (4.5.1) 式の最大評価値探索に相当する．Q-kohon は，小型車輪型ロボット (Khepera) の反射的行動生成問題に適用され，関連のニューラルネットワーク適用方法と比較して低計算コストと高い学習性能を示している．この方法は，状態空間，行動空間に渡って強い汎化性能を得られる方法であるが，同時に汎化の悪影響をもたらすおそれがある．また，SOM の学習則に従うためノードは入力ベクトルの確率密度を反映した分布を実現するが,「頻繁に経験した情報を細かく識別する」という分布が必ずしも適切であるとは限らない．

Santamaria *et al.* の方法は，実際に観測し，実行した連続値状態と連続値行動のデータを逐次的に記憶して行動価値関数の関数近似を行うものである [Santamaria98]. この方法では，状態空間，行動空間ともに関数近似 (RBF) による汎化を行うが，状態空間と行動空間を (Advantage Updating に近い方法で) 同じ関数近似の枠組みで表現する方法と状態空間の汎化と行動空間の汎化とを独立に扱う方法の二通りを提案し，比較を行っている．

以上，QCON，Q-kohon，メモリベースと RBF の組み合わせ法について，それぞれの利点と欠点を含めて述べた．2 章で述べたように，状態価値関数の関数近似による汎化の影響については適用例や収束証明などがいくつかすでに議論されている．しかし，行動空間の汎化についてはその悪影響に関する議論が未だ充分になされていない．この問題は，Q-learning のみならず Actor-Critic の研究においても今後盛んに議論されるものと考えられる [Sutton00]．そこで，本適用では，これまで比較的広く (安定的な性能が得られる方法として) 用いられている方法を採用し，離散的な各要素行動に対して別々の関数近似器を用いる．**Table 4.5** に具体的な適用アルゴリズムを示す．ここで，$a_i (i = 1, \cdots, m)$ は離散行動要素とする．

Table 4.5　Q-learning 適用のアルゴリズム

(1) 行動価値関数 $Q_{a_i}(\boldsymbol{x})$ の初期化 $(i = 1, \cdots, m)$：
(2) 各試行の初期化： 各試行における初期状態 $\boldsymbol{x}_0$ を決定
(3) 状態観測：$\boldsymbol{x}_t$ を得る
(4) 行動決定：$\pi(\boldsymbol{x}, a_i)$ から確率的に行動 a_t を選択

$$\pi(\boldsymbol{x}, a_i) = \frac{\exp(Q_{a_i}(\boldsymbol{x})/T)}{\sum_{j=1}^{m} \exp(Q_{a_j}(\boldsymbol{x})/T)} \qquad (4.5.4)$$

(5) 状態遷移：$\boldsymbol{x}_{t+1}$ を得，報酬 r_t を観測
(6) 行動価値関数 $Q(\boldsymbol{x}, a_t)$ の更新：

$$\delta_t = r_t + \gamma \max_{a_i} Q_{a_i}(\boldsymbol{x}_{t+1}) - Q_{a_t}(\boldsymbol{x}_t) \qquad (4.5.5)$$

$$Q_{a_t}(\boldsymbol{x}) \leftarrow Q_{a_t}(\boldsymbol{x}) + \alpha\delta_t \qquad (4.5.6)$$

(7) 試行終了判定：$\boldsymbol{x}_{t+1} \in s_f$ ならば，各関数近似器 $Q_{a_i}(\boldsymbol{x})(i = 1, \cdots, m)$ についてノード移動を行い，(2) に戻る
(8) 試行内繰り返し：(3) に戻る

4.6 プログラムの実装

本節では，プログラム実装の具体的なプロセスを示し，実用の際に計算量を要する過程についての考察を行う．各ルーチンの表記方法は擬似的に C++言語に習い，以下のクラスおよびインスタンスを定義する．

- `Node`：ノードのクラス．
- `FunctionApproximator` ：関数近似器のクラス．`Node` のインスタンス `node` を配列の形で[3]保有する．
- `Agent` ：強化学習のエージェントのクラス．`FunctionApproximator` のインスタンス `fa` を保有する．
- `Environment` ：強化学習の環境のクラス．状態遷移，報酬を決定する

Fig.4.7 にクラスのインスタンス同士の関係をまとめる．

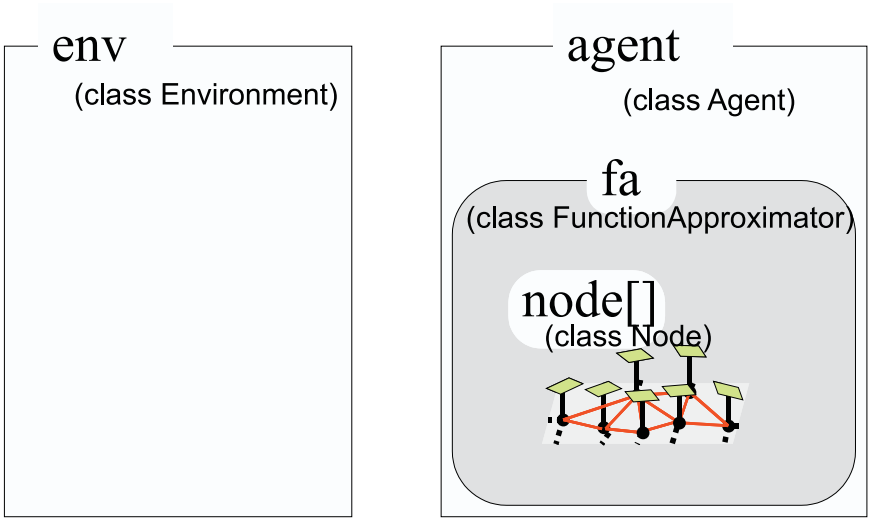

Fig. 4.7　クラスのインスタンス同士の関係

各クラスの代表的なメンバと関数を以下にまとめる．

[3] 正確には動的にサイズを変更できる vector 型配列で

```
class Node {                                  // ノードのクラス
  update_f();                                 // 関数値の更新
  update_df();                                //
  move();                                     // 移動
  lsm_update();                               // 目標関数への近似
};

class FunctionApproximator {                  // 関数近似器のクラス

  class node[NodeNum];                        // NodeNum 個の配列として保持

  UpdateResolution();                         // ノードの追加・削除・移動
  FindBestmatchingNode();                     // 最整合ノードの計算
  UpdateArcs();                               // アークの更新
};

class Agent {                                 // 強化学習におけるエージェントのクラス

  FunctionApproximator fa;                    // 関数近似器（状態価値関数近似）

  DecideAction();                             // 現在の状態から行動決定
  UpdateValueFunction();                      // 価値関数の更新
  UpdateFunctionApproximator();               // 関数近似器の構成を更新

} agent;

class Environment {                           // 強化学習における環境のクラス

  Init();                                     // エージェント初期状態の設定
  TransNextState();                           // 状態と行動から次の状態への遷移を決定

} env;
```

main_general()では4.3節で述べた一般的な関数近似法のアルゴリズムを示す．また，main_reinforce()では4.4節で述べた関数近似手法の強化学習への適用例のアルゴリズムを示す．

```
main_general(){                                  // 一般的な関数近似適用
  init();                                        // 各ノードの初期位置設定
  for(){                                         // 全体のループ
    for(k=0;k<LSMiteration;k++){                 // 最小二乗法ループ
      data = targetfunction(rnd());
      b = bestmatching_nodenum(data);            // 最整合ノードの決定
      node[b].lsm_update(data);                  // 最小二乗法による位置・勾配更新
      UpdateArcs(data);                          // アーク更新
    }

    UpdateResolution();                          // 反応拡散方程式に基づくノード移動
  }
}

main_reinforce(){                                // 強化学習問題への適用
  init();                                        // エージェント，環境の初期化
  for(){                                         // 強化学習ループ
    init_trial();                                // 各試行の初期化
    for(i=0;i<StepNum;i++){                      // 各試行ループ
      a = agent.DecideAction(s);                 // 行動決定
      [ss,r] = env.TransNextState(s,a);          // 状態遷移
      agent.UpdateValueFunction(s,a,ss,r);       // 価値関数の更新
      if(trial_terminated())                     // 試行終了判定
        break;
    }

    if(interval(i))                              // 関数近似器更新判断
      agent.fa.UpdateResolution();               // 関数近似器の更新
  }
}
```

一般的な関数近似の場合は，データ提示と最小二乗法による関数近似更新のループとノードの移動・追加からなる UpdateResolution() のルーチンの交互の繰り返しとなる．強化学習の場合のアルゴリズムの流れは，各試行における行動決定・状態遷移・価値関数更新の繰り返しに対して，一定の間隔で UpdateResolution() を実行する．各関数の出力決定には最整合ノード情報を用いるため，全ノードに対する比較計算を行う．

次に，上記アルゴリズム内で用いられている関数 UpdteResolution()，move()，UpdateArcs() について，大まかな流れを以下に述べる．

```
FunctionApproximator::
              UpdateResolution(){          // ノード移動による分解能の更新

  for(k=0;k<Motioniteration;k++){
    for(i=0;i<NodeNum;i++){
      node[i].move();
    }
  }
}

Node::move(){                              // ノードの移動
  update_f();                              // 複雑度の更新
  update_df();                             // 近傍との差分の更新
  update_w();                              // 位置ベクトルの更新 (移動)
}

FunctionApproximator::UpdateArcs(){        // TRN アルゴリズムによるアーク更新

  for(k=0;k<TRNiteration;k++){
    data = rnd();                          // データの取得
    b1 = bestmatching_nodenum(data);       // 最整合ノードの計算
    b2 = secondbm_nodenum(data);           // 第 2 最整合ノードの計算

    for(i=0;i<node[b1].arcnum;i++){
      if(b2 == end(i)){
        node[b1].refresh_arc(b2);          // アーク年齢初期化
      } else {
        node[i].age_arcs();                // アーク加齢
        node[i].remove_oldarcs();          // 古いアークの消去
      }
    }
  }
}
```

move() では，全ノードについて微小移動を繰り返す．移動方向決定には，アークで結ばれた近傍のノード情報を用いる．UpdateArcs() では，B.2 節で述べる TRN(Topology Representing Networks) のアルゴリズムによる近傍情報の更新を行う．この関数内では，仮想的に入力データを提示し (出力データは必要ない)，最整合ノードおよび第 2 最整合ノードの計算を行う．各アークには age(年齢) が定義されており，最整合と第 2 最整合となったノード間の age は 0 にリセットされ，それ以外の最整合ノードのアークは加齢する．age が一定以上のアークは削除することで，近傍でなくなったノード同士の接続を消去する．

計算量に影響を与えると考えられる主な要因は，入力次元 n とノード数 N である．

- 入力次元の増加の直接的な影響は，最整合ノードの計算 (ノルム計算の次元)，ノード移動計算における次元の増大である．これらの直接的な影響のオーダーは $\mathcal{O}(n)$ であり，全体的な計算量に占める割合は比較的小さいと考えられる．間接的な影響としては，高次元入力空間では，各ノードの近傍ノード数の増加を招く傾向がある．ノード同士が単純な格子状の位置関係にあるとした場合，近傍ノード数は $2n$ 個となる．近傍ノード数の増加はノード移動における計算および近傍更新計算に影響を与えるが，後者の方が影響は大きいと考えられる．
- ノード数の増加の与える影響は，直接的には入力次元の場合と同様に最整合ノードの計算およびノード移動計算の繰り返し回数の増大であると考えられる．間接的な影響としては，TRN アルゴリズムによる近傍関係の更新の際の計算量の増加が挙げられる．TRN アルゴリズムでは最整合と第二最整合[4)]の計算を行うため，この過程はノード数に対して $\mathcal{O}(N^2)$ の計算量を与える．

以上の議論から，計算量に影響を与える要因を出力時の最整合ノード計算および最小二乗法，ノード移動のための計算，近傍情報更新のための計算の 3 つの要素に大別することができる．5 章では関数近似問題および強化学習例題における計算機実験において，提案近似手法の有効性を検証すると同時にそれぞれの要因に関する計算時間の測定を行う．

[4)] 入力ベクトルに対するノルムが 2 番目に小さいノードを第二最整合ノードと呼ぶ．

4.7 強化学習拡張問題への適用

本研究で提案する強化学習への適用例は，TD 学習法の最も単純な TD(0) と呼ばれるものに限定している．しかし，本研究は一般の関数近似手法としても成り立つものであり，適用される強化学習アルゴリズムに大きな制約を加えるものではない．強化学習の学習性能を高めようとする方向性はいくつかあるが，それら拡張研究のうち

(1) 適格度トレース (eligibility trace)
(2) セミマルコフ決定過程問題 (SMDP)
(3) 階層型アーキテクチャ

について，本研究の提案する近似手法の適用可能性とその意義について論じる．

4.7.1 適格度トレース (eligibility trace)

TD(0) は 1 ステップ毎の更新則であり，報酬の伝播は各状態を訪れた時に遷移した状態同士の間で行われる[5)]．適格度トレース (eligibility trace) は時間遅れの大きい報酬の情報を早く伝播させることで学習の加速を図るものである．**Fig.4.8** はその概念を直観的に説明するものである．エージェントが 2 次元格子環境を (a) 図のような経路を経てゴールに到達したとする．TD(0) では，ゴール到達により得られた報酬は (b) 図のようにその 1 ステップ前の状態に対して価値関数更新によって伝播する．図中で矢印の長さはその状態での矢印方向への移動という行動に分配される収益の大きさを表している．それに対し適格度トレースでは，(c) 図のように，過去の状態遷移に対してもその収益の分配を行う．複数ステップにわたって分配を行う割合をパラメータ λ で表し，$0 \leq \lambda \leq 1$ である．$\lambda = 0$ のときは複数ステップにわたる分配は行われず，1 ステップ前後の報酬伝播のみが行われる．

この適格度トレースを Q-learning などに適用した例として，Sarsa(λ) や Q(λ) などが知られている．本研究で提案する関数近似器には，各ノードについて状態遷移の履歴を記憶することで容易に適用できる．Sutton *et al.* の適用例では CMAC のようなパラメータを介する関数近似器のパラメータに状態遷移の情報を割り付けるアルゴリズムが用いられている [Sutton96] が，本研究のノードによる関数近似器は一つの状態に一つのノードが対応しているため，より簡潔なアルゴリズムで同様のアルゴリズムが実現可能であると考えられる．

[5)] (2.3.10) 式参照.

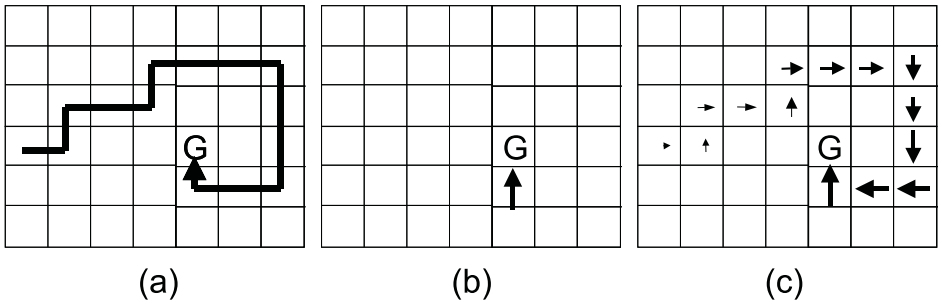

Fig. 4.8　適格度トレース

4.7.2　セミマルコフ決定過程 (SMDP)

連続値状態と連続値行動に対応する理論的枠組みは提案されているが，実用的には広大な (連続の) 状態空間と行動空間を同時に探索するのは難しい．その難しさの一つとして状態空間探索の問題がある．連続状態と連続行動を扱える学習器であっても学習の初期段階で手がかりとなる情報が充分でないにも関わらず細かい周期で意思決定 (探索的行動の選択) を頻繁に行うことは状態空間の探索のためには非効率的であり学習効率の低下を招く．効率の良い探索のためには一度行動決定した後しばらくその行動をとりつづけることも必要であるが，そのためには行動を切り替えるタイミングを知る必要がある．離散状態・離散行動を扱う枠組みでは，同じ離散状態と認識する間は同じ行動をとりつづけ，状態遷移によって行動決定と価値関数更新を行う考え方が提案されている．このような状態遷移を区切りとした行動決定過程は，セミマルコフ決定過程 (Semi Markov Decision Process：SMDP) として MDP と同様の理論的性質を持つことが示されている [Bradtke94].

浅田は，サッカーロボットへの強化学習の適用を通じて，連続値状態・連続値行動を同時に扱う難しさを指摘した [浅田 97]．**Fig.4.9** に示すように,「状態識別を行うためには要素行動が確定していなければならず，逆に意味のある要素行動を生成するためには状態認識が成立していなければならない」という関係として表現されている．このような関係の背後には，(報酬の形で与えられる) 評価がある [小林 00]．つまり，状態変数そのものが行動変数を規定するのではなく，状態識別や行動要素の決定を規定しているのは評価であり，強化学習の文脈では価値関数表現と深い関わりを持っているといえる．

本研究の提案する関数近似法は，連続値状態を扱う枠組みであるが，入力に対して競合的に最整合ノードを決定するという離散的な表象を持つ．そのため，状態識別の変化を行動決定過程に結びつける SMDP の考え方を適用しやすいという利点

を持っている．また，浅田の指摘する問題は，ある評価のもとで価値関数を生成するときに，その関数表現の粒度を決定する困難さを表していると考えることができる．これに対し，本研究の提案する関数近似法は，状態識別の粒度をポテンシャル汎関数 $W(f)$ という一つの量で表現できるという利点を持つ．すなわち，ポテンシャル汎関数 $W(f)$ を極小に保ったまま (各ノードの複雑度が均一であるという条件を保ったまま) ノードを逐次的に追加していき価値関数表現の粒度を上げていくという方法が,「状態空間と行動空間とを同時に生成する」という問題に対する一つのアプローチになるのではないかと考えられる．そのためには，Actor-Critic または Advantage Updating のような連続値行動の枠組みへも本研究を適用する必要がある．

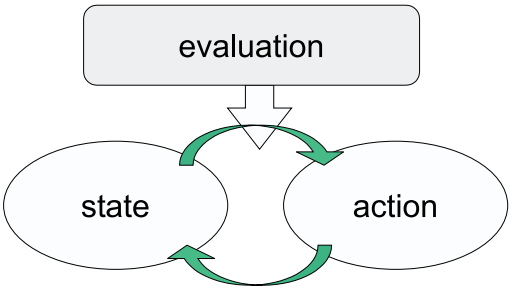

Fig. 4.9 状態・行動・評価の関係

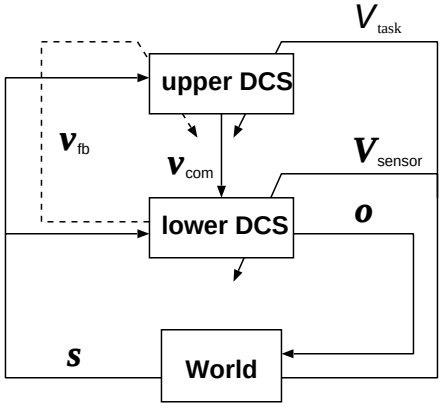

Fig. 4.10 階層型学習の例 [Kobayashi00]

4.7.3 階層型アーキテクチャ

大規模な問題に対するアプローチとして，問題を小さなサブタスクに分割して解くという方法は強化学習枠組みでの適用 [Parr98, Dorigo95, Dayan93] だけでなく，一般的な知能の構成論 [Albus96] から脳の運動学習モデルとしての研究 [Haruno99] など，幅広く研究されている．柔軟性という観点から見て，複数のモジュールを構成するときの重要な要素として，モジュール間でやり取りする情報の柔軟性を挙げることができる．

本研究で提案する関数近似器は，SOM や SOM の位相・ノード数を柔軟に変更できるようにした DCS(Dynamic Cell Structures：DCS)[Bruske95] と同様に，入出力の組を結合係数とするノードから形成される．4.5 節で示した Q-kohon の例か

らもわかるように，このような「入出力の組をノードで表す」という情報表現方法は，

- 入出力関係を動的に入れ替えることが容易である
- 部分的な入力から出力を引き出すことができる

という特質をもっている．**Fig.4.10** はその特質を利用した階層的強化学習の例である [Kobayashi00]．学習器は 2 階層からなり，下位層は状態入力と行動出力および多次元の行動価値関数の組を結合係数ベクトルとするノードからなる．この層は単独で行動価値関数の学習を行い，同時に上位層からの指令に基づいて多次元評価のもとでの行動の出力を行うこともできる．この方法は Q-kohon の汎化の悪影響と同様の欠点を持つが,「多次元の評価のうち一部分のみに着目した行動決定」を実装しやすいという利点を持っている．

本研究の関数近似手法も，ノードを関数近似要素とした上記の手法と同様の利点を持っている．それに加え，一般的な (SOM や DCS などの) ベクトル量子化アルゴリズムに対して，本研究の関数近似手法は

- 入力ベクトルの確率密度を反映したノード分布ではなく，近似関数の形状の複雑さを反映した分布を行う．このため，入力ベクトルの出現頻度の高い領域にノードが集中してしまうことによる不利益を回避できる．
- 入力ベクトルに対して単純に近づけるというノード移動方法ではなく，ノード移動は近似関数の形状を保存するように行われる．このため，ノード移動による関数近似の劣化を食い止めることができる．

という利点を持っている．したがって，強化学習をより複雑な問題に適用とする上記の試みに対して，本研究の関数近似手法は有効な道具になりえると考えられる．

4.8 おわりに

本章では，本研究で提案する関数近似アルゴリズムを実際の問題に適用する方法について，問題点を考察した上で，以下の 3 つの適用アルゴリズムを示した．

(1) 一般的な関数近似問題への適用方法
(2) 強化学習適用例 1：Value Gradient
(3) 強化学習適用例 2：Q-learning

一般的な関数近似問題への適用では，データが充分に得られる場合最小二乗法による逐次更新で関数近似を行う．強化学習における価値関数近似では位置情報のみを更新し，直接に勾配情報は変更しない．ノード移動の際に 3.3.2 節で述べた勾配情報更新法を用いる．また，各関数近似で必要に応じてノードの追加を行う方法としてノード上の関数値を利用した方法を示した．

最後に，本研究の適用例で述べなかった強化学習の他の拡張研究との関連について論じた．

適格度トレースへの適用については，状態遷移が最整合ノードの切り替わりに対応しているためにアルゴリズムの実装が容易になることを指摘した．また，連続値行動探索問題においては，SMDP の考え方と逐次的なノード追加 (とノード移動) を組み合わせる方法が状態と行動の同時生成問題に対する一つのアプローチとなりうることを考察した．さらに，階層型アーキテクチャの議論においては入力の部分的利用によってより柔軟な情報の蓄積と取り出しが可能であるという利点を指摘した．

これらの可能性はいずれも，状態をノードという完全に独立した離散的表象で表現することの利点であり，RBFN や CMAC などの関数近似手法では実現しにくい．5 章における比較とは別に，他の関数近似手法に対する利点を論じた．

第5章　シミュレーション

5.1 はじめに

本章では，3 章で提案した関数近似手法および 4 章で提案したその強化学習への適用方法を計算機実験により評価し，その有効性と検証し，問題点を考察する．

5.2 節では，本章におけるシミュレーションの目的について述べる．1 章で述べた強化学習のための関数近似に対する要請，および本研究で特に着目する

- 適応的に分解能を変更する
- 問題に応じた設計自由度を低減する

という特質について，比較対象と比較方法について改めて説明する．

5.3 節では，定常関数近似問題を例題として，提案手法によるノード移動が近似誤差を低減できることを示す．ノード移動およびノードの逐次的追加が関数近似精度に与える影響について基本的な評価・考察を行う．

5.4 節，5.5 節，各強化学習例題においてノード移動および近似関数の複雑度に基づいたノードの逐次追加の効果を検証する．5.4 節では 2 次元平面上を動くエージェントによるゴール到達問題，5.5 節では 1 自由度の倒立振子振り上げ問題， それぞれの問題の性質に適した行動学習法を適用して評価を行う．

最後に 5.6 節において，本章の内容をまとめるとともにそれぞれのシミュレーション結果を総括的に考察する．

5.2 シミュレーションの目的

本章のシミュレーションでは，提案する関数近似手法を定常関数の近似問題と強化学習例題に適用する．

定常関数近似問題　　定常関数近似問題への適用においては，本研究で提案する関数近似手法の基本的な性質を検証し，3章で述べた設計指針に沿うものであることを確認する．具体的には，

- 反応拡散方程式に基づく挙動設計による複雑度均一化の検証
- ノード移動による関数近似誤差の低減性能の検証
- ノード個数と関数値，近似誤差の関係に関する考察

を目的とする．

強化学習例題への適用　　強化学習においては，1章において述べたように，関数近似手法には以下の基本的な要求が存在する．

(1) 非定常関数近似が可能であること
(2) ブートストラップ型関数近似問題に対して安定的に近似可能であること

この問題に対しては，4.2.3 節で述べたように，一般の関数近似問題では位置と勾配の両方を更新する方法と異なり，位置情報のみを更新する方法を提案している．この方法を導入することにより，ノード移動を伴わない場合は，CMAC [1)]や k-近傍法と等価の枠組みとなる．これらの方法は強化学習への適用事例でももっとも安定的に機能するクラスの手法として知られているため [Sutton96]，これらの問題に対する検証は特に行わない．ノードの移動を行わない状態での各強化学習例題への適用で学習が可能であることによりこれらの基本的要求に応えるものであることは示されると考える．

上記の要求を前提として，本研究で強化学習のための関数近似に関して主張するのは，以下の2点である．

(1) 学習効率化のためには，自律的・適応的に分解能を変更できることが重要である
(2) 自律的な分解能変更のためには，設計者の試行錯誤による調整などの設計自由度を低減できることが重要である

[1)] CMAC の条件設定において，タイリングを単一にし汎化作用を無くしたもの．

(1) の主張に関しては，本章で用いている問題設定において安定的な性能を示している CMAC と同等の格子状関数近似手法との比較を行う．CMAC の利点は，線形近似手法で安定的に強化学習に適用可能であることがすでに示されていることと，tiling と呼ばれる格子状の状態分割要素を重ね合わせることで汎化が期待されることである．本研究の関数近似手法におけるノードを格子状に配置した場合は，CMAC の tiling を単一にし，tiling 重ねあわせによる汎化効果をなくした状態に相当する．厳密に CMAC と等価になるわけではないが，汎化の作用は大きくしない方が安定的な関数近似が可能であるという指摘がなされていることも考え [Kaelbling96]，本章におけるこの主張に関しては，本研究で提案する関数近似手法に関して，

- 格子状に固定して配した場合
- ノード移動を行った場合

の両者を比較することにする．

(2) の主張は，適応的に分解能を獲得することを目指した関連研究に対する位置付けととらえられる．状態空間の自律的分割手法としては，高橋らの手法 [高橋 99]，関数近似器における動的な基底追加手法としては Samejima *et al.* の方法 [Samejima98] と Morimoto *et al.* の方法 [Morimoto98] を挙げることができる．本研究の提案の特徴は，これらの研究では近似誤差 (あるいは報酬のばらつき) に基づいて動的な基底追加を行うのに対し，近似関数に定義される「複雑度」を用いてノード追加を行う点にある．本章では，この複雑度を利用した動的なノード追加が適切に機能することを検証することで関連研究に対する利点を示す．

以上の議論をまとめ，強化学習例題の適用においては，

- ノードを格子状に固定した場合と提案アルゴリズムによりノード移動を行った場合の比較
- 複雑度に判断基準として用いたノードの逐次的追加の有効性の検証

を行うことを目的とする．

5.3 定常関数近似問題

定常関数の入出力データを提示し，ノード移動の際の各ノード上の関数 $f(u)$ の遷移を調べる．関数近似器の出力の評価として一般的なのは，以下の式で表されるような関数近似の二乗誤差の総和である．

$$PE = \sum_i \left| \Phi^t(\boldsymbol{x}_i) - \Phi^a(\boldsymbol{x}_i) \right|^2 \qquad (5.3.1)$$

ここで，$\Phi^t(\boldsymbol{x})$ は目標近似関数 (target function)，$\Phi^a(\boldsymbol{x})$ は関数近似器の出力，$\boldsymbol{x}_i$ は状態空間中から均等に選んだ状態変数とする．2 章において議論したように，強化学習に適用する際の目的は，必ずしも上記の位置の近似誤差 PE を最小化することではない．本研究の関数近似器の「勾配変化を適切に表現できる」という設計指針に対する評価として，以下の式で表される各点における勾配推定の二乗誤差も評価指標として有効であると考えられる．

$$GE = \sum_i \left\| \frac{\partial \Phi^t(\boldsymbol{x}_i)}{\partial \boldsymbol{x}} - \frac{\partial \Phi^a(\boldsymbol{x}_i)}{\partial \boldsymbol{x}} \right\|^2 \qquad (5.3.2)$$

この GE は以後勾配近似誤差と呼ぶ．

近似目標関数には，1 次元および 2 次元のガウス関数

$$\Phi_0^t(\boldsymbol{x}) = \frac{1}{2\sqrt{\pi}} \exp\left(-q\|\boldsymbol{x} - \boldsymbol{p}\|^2\right) \qquad (5.3.3)$$

および以下の二つの関数を用いる．

$$\Phi_1^t(x) = \begin{cases} 0.15 + 0.1\sin(\frac{2}{3}\pi x) & (0 < x < 0.3) \\ 0.5 - 288(x - 0.5)^4 & (0.3 < x < 0.7) \\ \frac{1}{2\sqrt{\pi b}} \exp\left\{-\frac{(x-0.5)^2}{4b}\right\} & (0.7 < x < 1) \end{cases} \qquad (5.3.4)$$

$$\begin{aligned} \Phi_2^t(x_1, x_2) = 0.02x_1^2 &+ \frac{1}{2\sqrt{\pi}} \exp\left[-25\left\{(x_1 - 0.15)^2 + (x_2 - 0.15)^2\right\}\right]^2 \\ &+ \frac{1}{4\sqrt{\pi}} \exp\left[-250\left\{(x_1 - 0.3)^8 + (x_2 - 1.0)^8\right\}\right]^2 \end{aligned} \qquad (5.3.5)$$

で表される．ここで $b = 0.0028$ である．**Fig.5.1** および **Fig.5.2** に各近似目標関数の形状を示す．これらの関数は，状態空間内に勾配変化の小さい領域と大きい領域が存在するように設計した．以降，本節でのノード移動のパラメータはいずれの場合も $\beta_D = 0.001$ である．

1 次元ガウス関数の例　　ノード数 10 個で等間隔のノードに充分なデータを提示した初期状態と，全ノードを微小量移動させるのを 1 ステップとして 300 ステッ

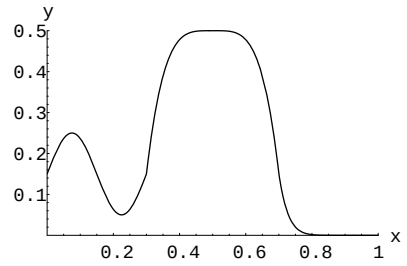

Fig. 5.1　1 次元近似目標関数 Φ_1^t

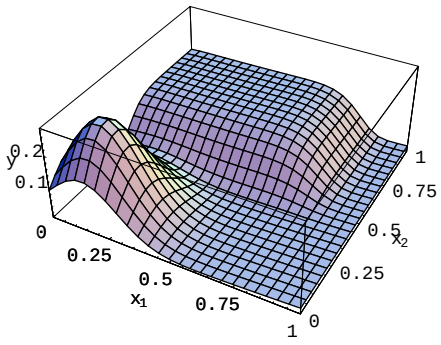

Fig. 5.2　2 次元近似目標関数 Φ_2^t

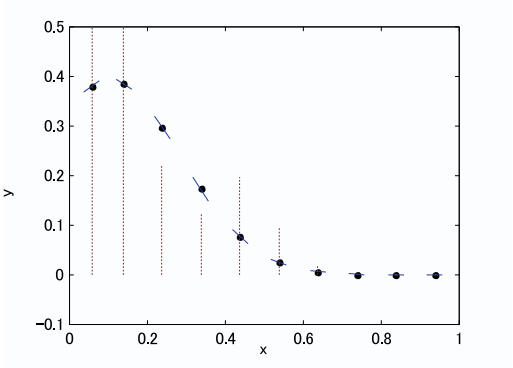

Fig. 5.3　初期状態

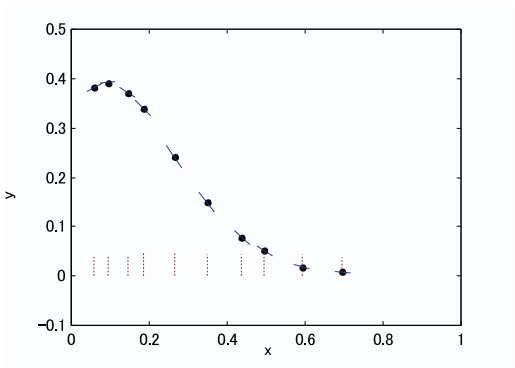

Fig. 5.4　ノード移動後

プ後の状態をそれぞれ **Fig.5.3**，**Fig.5.4** に示す．各ノード上の線分は勾配，鉛直な線分は各ノードにおける関数値 $f(u)$ を示している．

初期状態では 1 次元格子状に等間隔に配置されていたノードが，移動後にはガウス関数の頂点付近，裾野付近の勾配変化の大きい領域に密に分布していることがわかる．同時に，初期状態では頂点および裾野付近のノードの関数値が比較的大きかったが，ノードがその領域に集中することで関数値の均一化が達成されていることがわかる．

次に，このときの各関数値 $f(u)$ およびポテンシャル汎関数 $W(f)$ の推移を **Fig.5.5** および **Fig.5.6** に示す．各関数値は均一化されているのがわかるが，必ずしも滑らかな曲線で推移するわけではない．この原因の一つは，ノード移動の 10 ステップ毎に最小二乗法によるデータの提示を行っていることである．ノード移動の 10 ステップの間は勾配変化モデルにしたがって各ノードの位置と勾配を更新するため，近似目標関数との間に若干のずれが生じる．次のデータ提示によってノードの情報が更新されるときに関数値に変化が生じるものと考えられる．

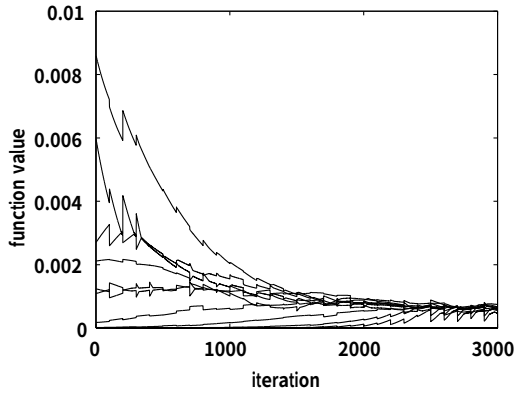

Fig. 5.5　各関数値 f の推移

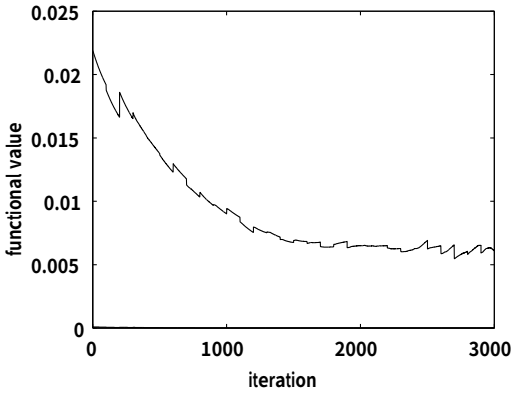

Fig. 5.6　差分 df の推移

Fig.5.7 および **Fig.5.8** に，位置近似誤差 PE および勾配近似誤差 GE のそれぞれの推移を示す．位置近似誤差，勾配近似誤差ともに低減していることがわかる．同じノード数に対して，ノードを移動させることにより関数近似の効率が向上しているといえる．

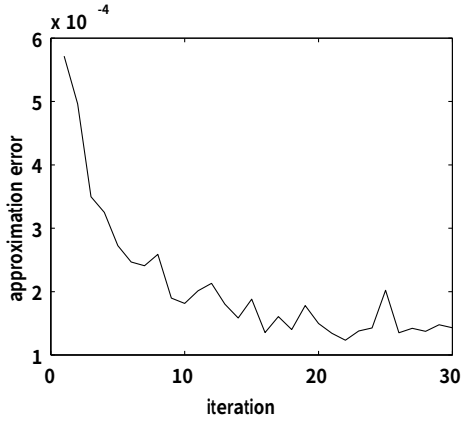

Fig. 5.7　位置近似誤差 PE の推移

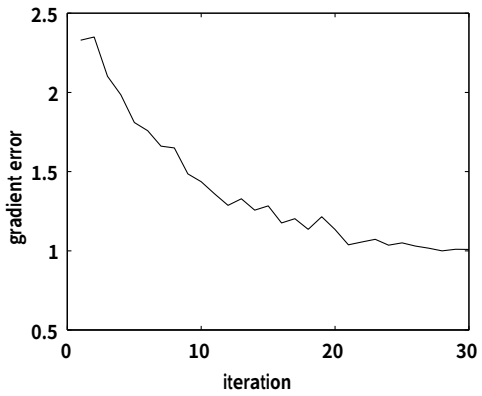

Fig. 5.8　勾配近似誤差 GE の推移

Fig.5.9 は，異なるノード数に対して，ノード移動を行う前と後の近似誤差の比較を行ったものである．いずれのノード数に対しても，ノード移動によって分布密度が変わった結果，関数近似の精度が向上していることがわかる．また，ノード数が少ないほど，ノードの移動による近似精度の向上が顕著である．強化学習適用の際には少ないノード数で効率よく価値関数を近似することが求められるため，この効用は重要な意義をもっている．

関数値とノード追加基準　　次に，3 章で論じた「関数値とノード追加基準の関係」という問題に関する考察を行うために，ノード数を変化させたときの関数値

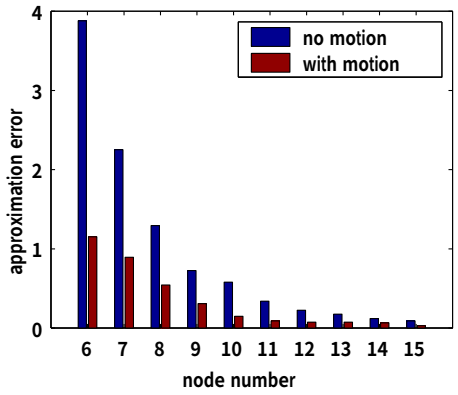

Fig. 5.9　各ノード数でのノード移動による近似誤差の変化

の推移を調べた．**Fig.5.10** は，各ノード数に対して，ノード移動を行う前 (左図) とノード移動を行った後 (右図) の $f(b_f)$ を示したものである．ここで，b_f はグラフ内で関数値が最大のノードである．左図と右図のスケーリングの違いより，それぞれのノード数に対して，ノード移動により最大の関数値は 1/5 から 1/10 程度に減少していることがわかる．

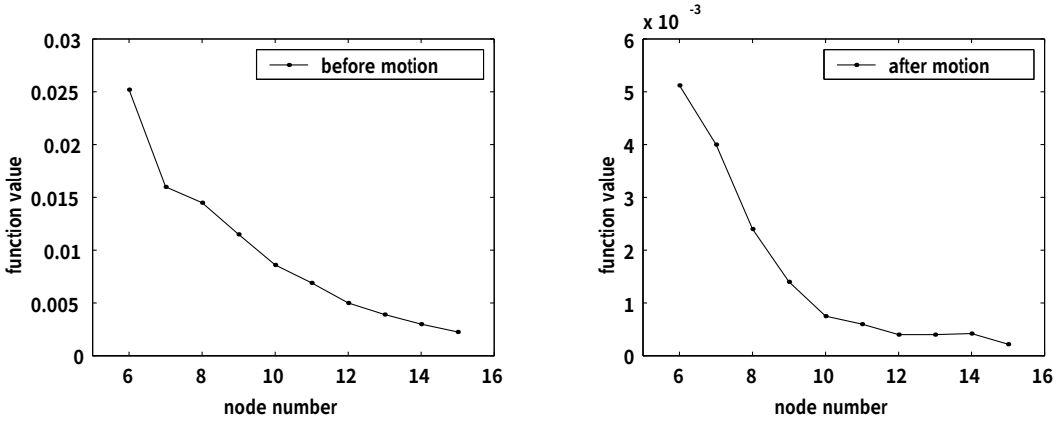

Fig. 5.10　各ノード数に対する関数値の変化

ここで，3 章で考察したノード追加の基準の議論をこのケースに当てはめて考える．**Fig.5.11**，**Fig.5.12** は，3.6.3 節における問題設定において，ノード間隔を $c_0 = 0.1, d_0 = \pm 0.1$ としたときの関数値基準を示す断面である．横軸，縦軸はそれぞれ y_0'', y_1'' である．

Fig.5.13 はこの断面の片側の曲線をプロットしたものであり，この各点に対応

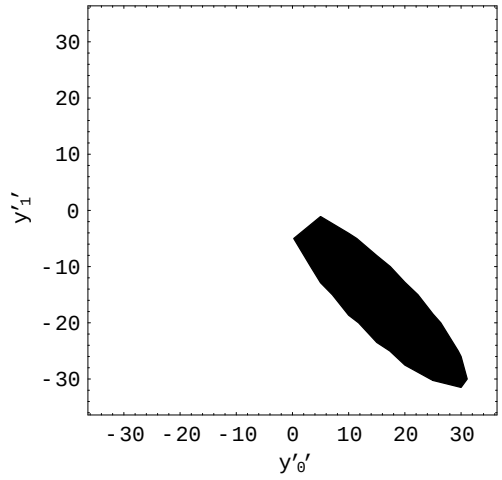

Fig. 5.11　d_0=0.1 の場合の断面

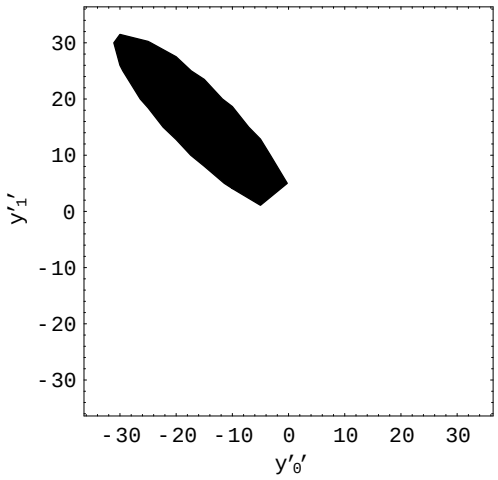

Fig. 5.12　$d_0 = -0.1$ の場合の断面

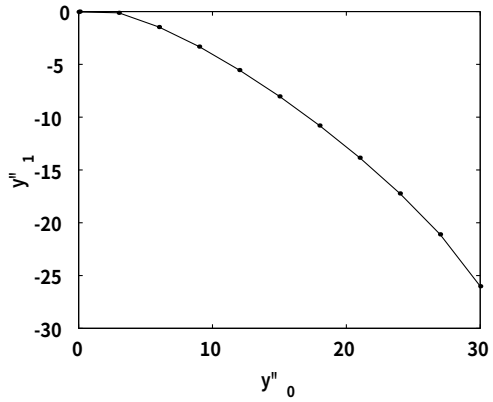

Fig. 5.13　断面の片側の軌跡

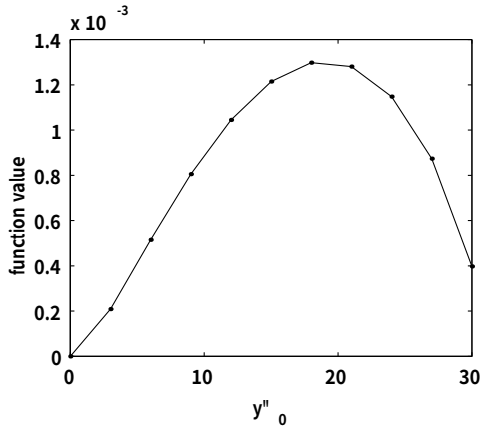

Fig. 5.14　断面の関数値への変換

する関数値を (3.6.17) 式を用いて計算したものが **Fig.5.14** である．この関係から，サンプリング定理に基づいた判断では，信号を回復するために充分なノード密度を保持するための必要条件は $0 \leq f(u) \leq 1.3 \times 10^{-3}$ ということになる．**Fig.5.10** と対応を取ると，ノード数 9 個でノード移動をさせたあとの状態はこの関数値の基準値に近い状態である．しかし，実際のノード間隔は $c_0 = 0.1, d_0 = 0.1$ の条件にちょうど当てはまるわけではないため，厳密にはより高い密度でノードが集中した状態がこの基準値に相当すると考えられる．

1 次元関数 $\Phi_1^t(x)$ の例　　1 次元関数 $\Phi_1^t(x)$ はガウス関数と比較して，平坦な領域と勾配変化の大きい領域の差が大きく，極小・極大点の数も多い．近似方法としては，逐次的ノードの追加を用いた．ノード数 10 から一定周期でノードを追加

し，最終的にノード数 16 の状態で終了した．

Fig.5.15 に移動およびノード追加を行った後のノード分布を示す．勾配変化の少ない領域である最大値付近や最小値付近ではノードは疎に分布している．勾配変化が急激な領域では，ノード同士が接近していることがわかる．接近しすぎたノードは，その点における勾配を正確に表現できないおそれがある．**Fig.5.16** に各関数値の推移を示す．同数のノードを移動させる場合と異なり，ノードの追加により急激に関数値が変化する場合があることがわかる．

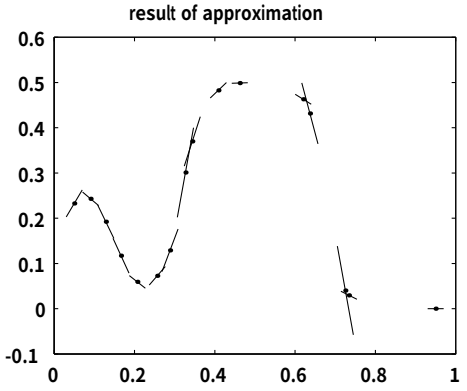

Fig. 5.15 移動・追加後のノード分布

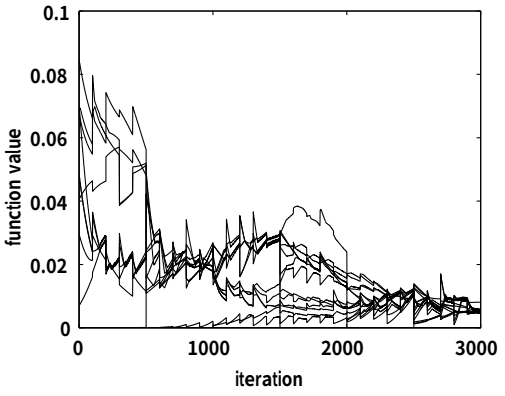

Fig. 5.16 各関数値の推移

Fig.5.17 に位置近似誤差の推移，**Fig.5.18** に勾配近似誤差の推移を示す．1 次元軸上に等間隔に配置したノードによる位置近似誤差は 0.00604，勾配近似誤差は 63.78 であった．いずれの誤差も逐次追加の場合は約 50% 程度に低減されている．ノードを逐次的に追加することで近似誤差が低減できることに加え，等間隔配置の場合と比較しても誤差が低減できることを確認した．

2 次元の例　2 次元問題では，ノードの移動と同時に追加と削除を行った．すなわち，一定周期毎に **Table 4.2** で述べた方法でノード追加を行う．同時に，関数値最小のノードを削除する．これにより，ノード数は結果として初期状態から一定数に保たれる．

まず，2 次元ガウス関数の近似を行った結果を示す．ノード数は 64 個，8×8 の格子状の配置を初期状態とする．**Fig.5.19** に，ノードの追加・削除・移動後のノードの分布を示す．ガウス関数の勾配変化の大きい領域は最大値をとる [0.2 0.2] の周辺の領域であるが，最大値周辺にノードが密に分布し，反対側の領域では疎に分布していることがわかる．**Fig.5.20** にこのときの各ノード上関数値の推移を示す．1 次元の場合と異なり，全ノードの関数値を完全に均一にするのが困難であるこ

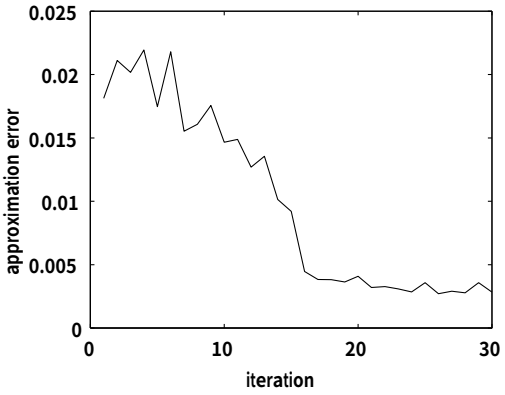

Fig. 5.17 位置近似誤差 PE の推移

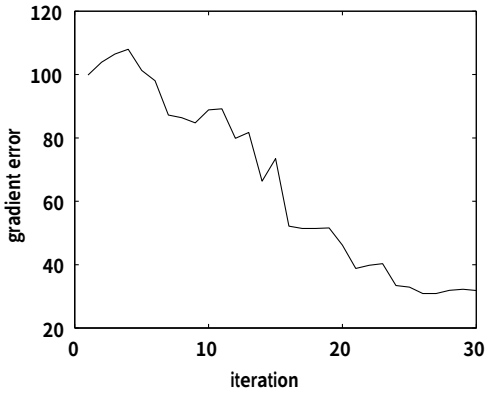

Fig. 5.18 勾配近似誤差 GE の推移

とがわかる．関数値の均一度を表す $W(f)$ は大きく減少しなかったが，**Fig.5.21**, **Fig.5.22** からわかるように，各近似誤差は低減されている．また，関数値の推移は，1 次元の場合と比較して，不連続な変化がより顕著になっていることがわかる．これには，1 次元の場合は (ノード の追加・削除を行わない限り) 近傍関係が固定であるのに対し，2 次元の場合は，ノード の移動に伴ってアークが動的に生成・消去されることの影響が考えられる．

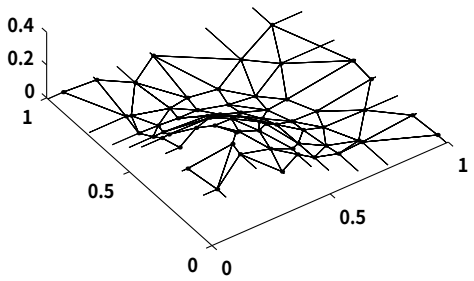

Fig. 5.19 ノード 移動後の分布

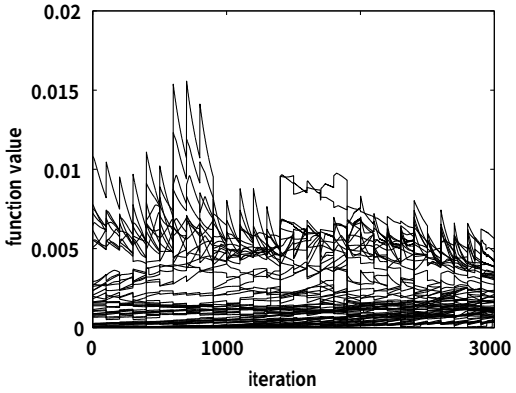

Fig. 5.20 各関数値の推移

次に，2 次元関数近似問題 $\Phi_2^t(\boldsymbol{x})$ の近似を行った結果を示す．ノード の移動・追加・削除は 2 次元ガウス関数近似の場合と同様，64 個のノード を 2 次元格子状に配置し，移動を行いながら一定周期で追加と削除を行った．**Fig.5.23** にノード 移動後の分布を示す．勾配変化の大きい領域である図中手前の山の周辺，奥の台地の周辺に比較的密に分布していることがわかる．**Fig.5.24** に各関数値の推移を示す．位置近似誤差は，これまでと同様状態空間中全域に渡る総和を表した **Fig.5.25** の推

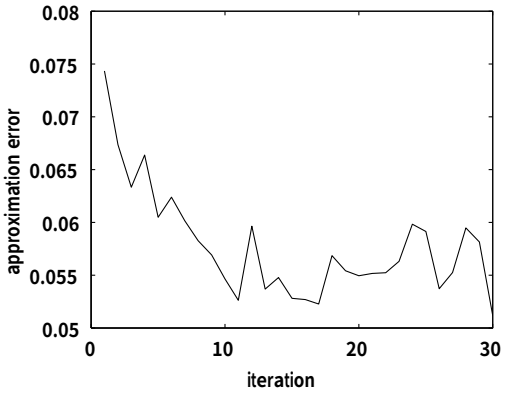

Fig. 5.21　位置近似誤差の推移

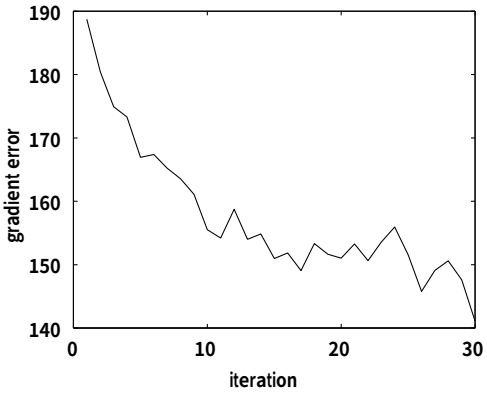

Fig. 5.22　勾配近似誤差の推移

移に加え，比較的勾配変化が大きいと考えられる $\{(x,y)|0 \leq x \leq 0.4, 0 \leq x \leq 0.4\}$ の領域のみにわたって位置近似誤差の総和を求めた推移を **Fig.5.26** に示す．全域に渡る近似誤差と比較して，特定の領域に着目した近似誤差はより大きな近似誤差の低減を達成していることがわかる．この性質が，強化学習適用時に有効に作用することが期待される．**Fig.5.27** に勾配近似誤差の推移を示す.

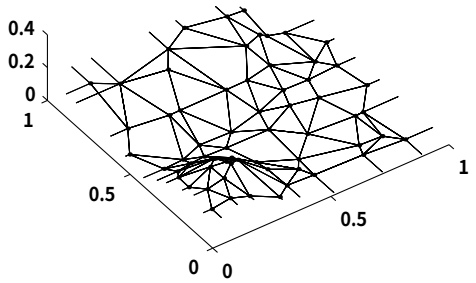

Fig. 5.23　ノード移動後の分布

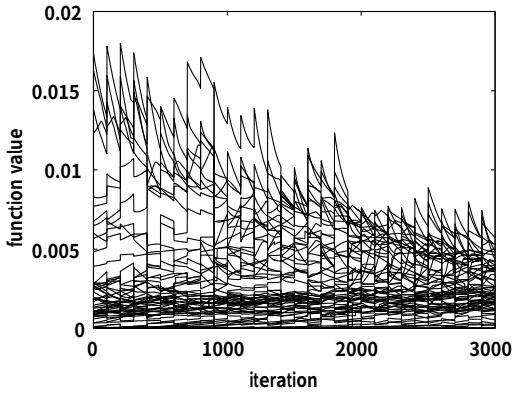

Fig. 5.24　各関数値の推移

計算時間　　ノード移動計算に関する CPU 計算時間の測定を行った．プログラムの実装には，C++言語 (Visual C++ Version 6.0) を用いている．ハードウェアは，PentiumIII 1GHz CPU の PC である．各ノードを微小量移動させるのを 1 ステップとし，100 ステップに要する時間を計測した．近似目標関数には，1 次元の場合は近似関数 $\Phi_1^t(x)$，2 次元の場合は近似関数 $\Phi_2^t(\boldsymbol{x})$ を用いている．**Table 5.1** に各計算時間を示す．計算時間はいずれも 10 回平均値である．

2 次元の場合は，ノード移動が進行するにしたがって計算時間は変化した．これ

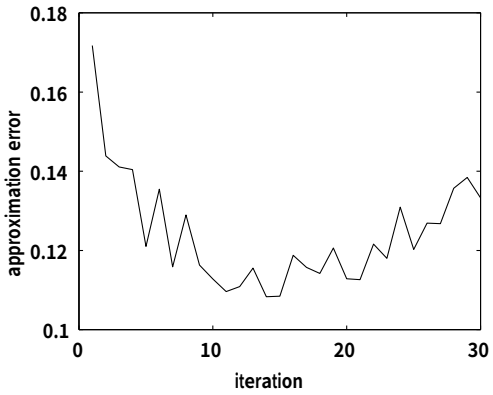

Fig. 5.25 位置近似誤差の推移

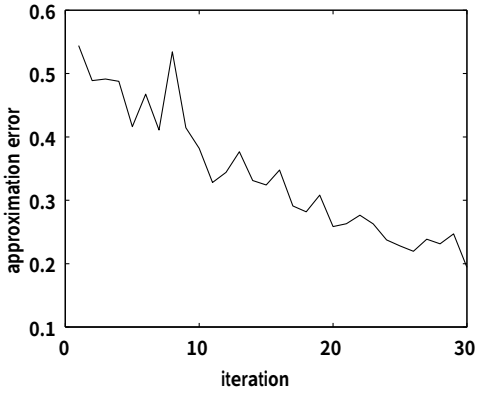

Fig. 5.26 位置近似誤差 (2) の推移

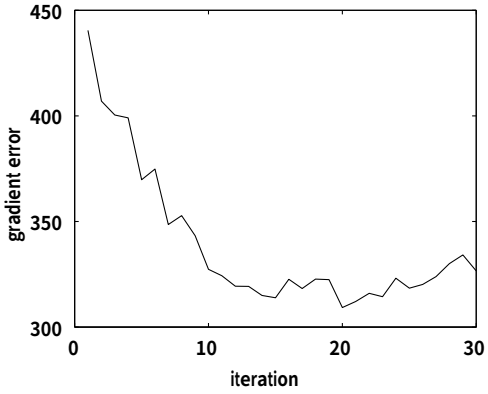

Fig. 5.27 勾配近似誤差の推移

Table 5.1 ノード移動の計算時間 (100 ステップ)

ノード数	1 次元 [sec]	2 次元 [sec]
16	0.025	0.115
64	0.110	0.480
100	0.180	0.852
225	0.480	1.874

は，ノードが移動するとともに近傍関係が変化し，近傍ノード数が変化するためであると考えられる．全般には，ノード数の増加に比例して計算時間が増加していることがわかる．

5.4 強化学習例題1：水たまり問題

水たまり問題 (Puddle World) は，**Fig.5.28** に示すような 2 次元平面上をエージェントが水たまりを避けながら移動し，ゴールを探索するという問題である．[Boyan95] や [Sutton96] などでニューラルネットワークや LWR，CMAC などの関数近似手法の適用例として取り上げられている．ここでは，広く例題として用いられている水たまり問題と，その問題設定の一部を改変した 2 次元環境探索問題の二つを例題として用いる．

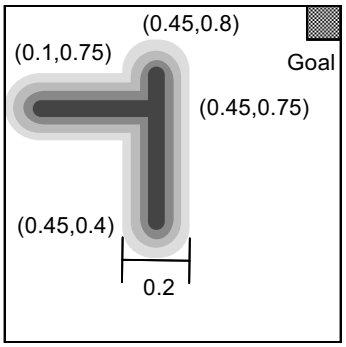

Fig. 5.28 水たまり問題

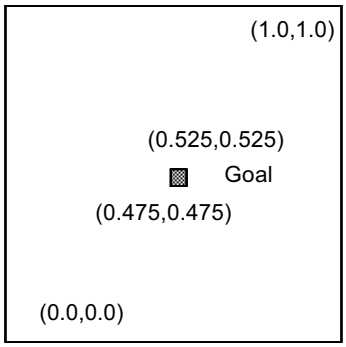

Fig. 5.29 2 次元環境探索問題

問題設定　水たまり問題では，**Fig.5.28** に示すような 2 次元平面上をエージェントが移動し，右上のゴール領域を目指す．2 次元平面領域は $[0, 0]$ から $[1, 1]$ までの正方形領域とし，ゴール領域は，[0.9,0.9] から [1.0,1.0] までの正方形領域である．領域内には水たまり (puddle) が存在し，エージェントは水たまり領域に入ると (中心溝からの距離) × (-400) の負の報酬を得る．行動は上下左右 4 方向への移動で大きさ 0.05 であり，1 ステップで -1 の報酬を得る．また，各移動毎に標準偏差 0.01，平均 0 の正規分布のノイズを受ける．また，2 次元領域境界上に来た場合は，それより外には出ないように位置を修正する．

2 次元環境探索問題では，**Fig.5.29** のように水たまりのない領域を移動してゴール領域を目指す．エージェントの移動に関する設定は同じであるが，ゴール領域の設定を変更した．ゴール領域は中央 [0.475,0.475] から [0.525,0.525] の正方形領域に位置し，領域の大きさは水たまり問題の半分である．

問題の性質　水たまり問題は，行動学習問題としては，以下のような性質を挙げることができる．

- ゴール領域到達時のみに零の報酬を与え移動に小さな負の報酬を与えることで，最小移動量 (ステップ数) でゴールに到達する行動を獲得させる

- 水たまりで大きな負の報酬を得るため，水たまりを避ける行動をとれるかどうかが得られる報酬量を左右する
- 状態空間 (2 次元) の境界領域ではその場にとどまる．そのため，境界外に出たら終了とする場合と比較してゴール領域に到達しやすい

学習性能を決定するのは，状態価値関数の学習でいうとゴール方向への勾配が適切に表現されていることと，水たまりを避けるような (水たまり部分が谷になるような) 勾配が適切に表現されていることであると考えられる．そこで，**Fig.5.29** のような 2 次元環境探索問題を加えて設定した．これは，水たまり問題の水たまりをなくし，ゴール領域を小さくして環境の中心に移動させたものである．

学習条件　　各例題の学習における条件は，強化学習定数は $\alpha = 0.7, \varepsilon = 0.1$ である．終了条件は，ゴール領域に到達するか 10000 ステップ行動時である．

結果　　まず，格子状に固定したノードによる水たまり問題の学習を行った結果の例を **Fig.5.30** に示す．この例は 8×8 のノード数で 100 回試行を行った結果を 10 回分の平均をとったものである．学習の初期段階では大きな負の報酬を得るが，その後の負の報酬は小さくなり，30 回以降は大きくは変化しない．**Fig.5.31** は格子状固定のノードで達成した状態価値関数近似の例である．z 軸は正負を逆転させている．状態空間中右奥の領域のゴール領域での状態価値関数は 0 となり，水たまり以外の領域ではゴール領域に向かう勾配が形成されていることがわかる．水たまり領域では大きい負の報酬の得るために高い山になっている．

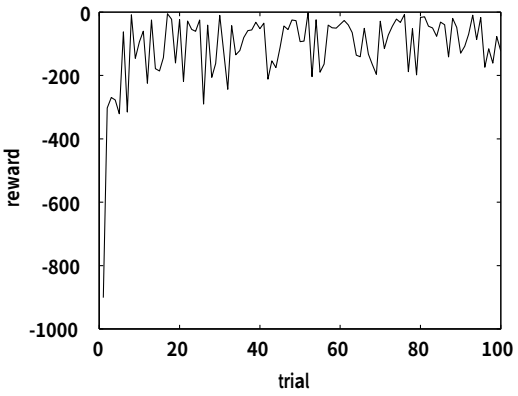

Fig. 5.30 試行回数に対する報酬 (平均)

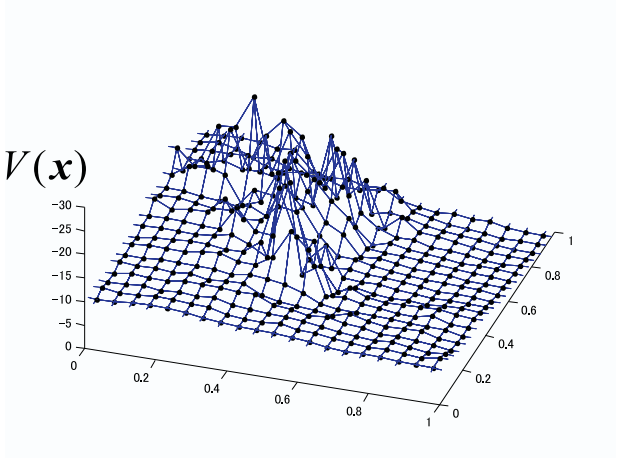

Fig. 5.31　状態価値関数の例

Fig.5.32 に，格子状のノード数を変化させたときの 100 回試行において得られた報酬の積算値 (10 回平均) を示す．横軸は個数の平方根であり，実際のノード数

は 16,25,36,$\cdots$,324(18^2) である．ノード数増加に伴って学習効率が向上していることがわかる．しかし，ノード数が多ければ多いほど学習性能が高いわけではなく，細かすぎる状態識別は逆に学習効率の低下を招いていることがわかる．**Fig.5.33** に，同数の格子状ノードについて，ノードを固定したままの場合と移動した場合との比較を行った結果を示す．ノード移動を行うことにより得られる報酬積算値が向上していることが確認された．しかし，**Fig.5.32** からわかる適切なノード個数を超えた場合にはそれ以上の学習性能が得られるわけではない．

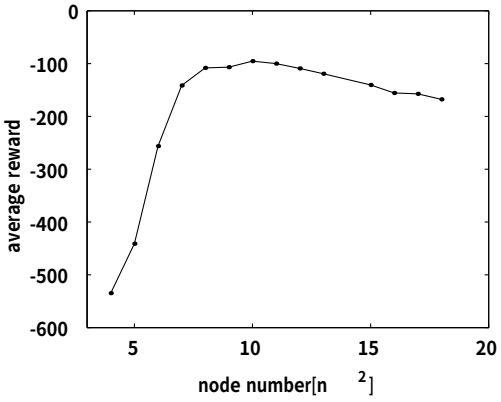

Fig. 5.32 ノード数に対する報酬変化

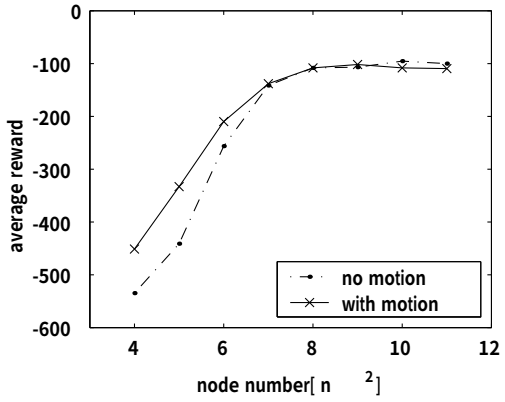

Fig. 5.33 ノード移動と固定の場合の比較

2 次元環境探索問題の場合の，ノード数を変化させたときの報酬積算値の変化を **Fig.5.34** に示す．図中の四角形は，複雑度に基づいて逐次的なノード追加を行った場合の報酬積算値を表す．数値はいずれも 10 回の平均値である．逐次的なノード追加におけるノード数は 45 個である．グラフから，格子状固定の場合と比較して 2 倍程度のノード数の場合と同程度の学習性能を得ていることがわかる．

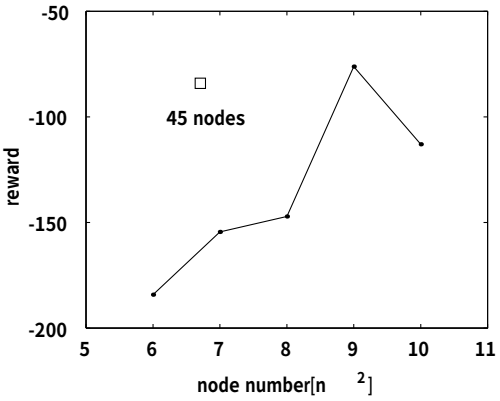

Fig. 5.34　ノード数に対する報酬変化

5.5 強化学習例題2：1自由度振子振り上げ問題

1 自由度振子振り上げ (Pendulum swing up with limited torque) 問題は，連続値強化学習の例題として知られる [Sutton98, Doya00]．**Fig.5.35** に示すような 1 自由度の振子に回転トルクを与え，振り上げて上方で停止させることが目的である．振子の状態変数は $\boldsymbol{x} = [\theta\ \omega]$ であり，以下の運動方程式にしたがって運動する．

$$\dot{\theta} = \omega \tag{5.5.1}$$

$$ml^2\dot{\omega} = -\mu\omega + mgl\sin\theta + u \tag{5.5.2}$$

m は振子の先の錘の質量 [kg]，l は振子の長さ [m]，μ は粘性抵抗係数，g は重力加速度である．$T = u$ は行動出力の回転トルク [N· m] であり，最大トルク $u_{\max}$ を発生させても一度で頂上に到達することはできず，頂上に到達するためには，往復運動を必要とする．さらに，得られる報酬をより多くするためには，振り上げられるようになった後，その状態で静止させる必要がある．シミュレーション条件は，$m = l = 1, g = 9.8, \mu = 0.01, u_{\max} = 5.0$，報酬関数は $R(\boldsymbol{x}) = \cos\theta$，また状態空間の定義域は $-2\pi < \theta < 2\pi, -3\pi < \omega < 3\pi$ である．物理モデル，強化学習ともに計算は 0.02[sec] 毎に行う．

強化学習における 1 試行の終了条件は，1 回の試行時間は 20[sec] 経過時あるいは，定義域外に出たときである．また，強化学習の学習定数は $\alpha = 0.7, \varepsilon = 0.1$ である．

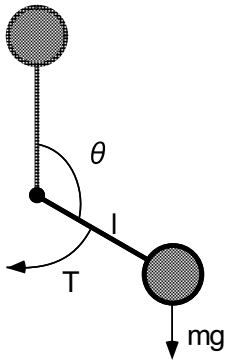

Fig. 5.35　1 自由度振子振り上げ問題

まず，ノード数 15 × 15 個に対してノードを格子状に固定した場合とノードの移動を行った場合との比較を示す．それぞれ 1500 回試行を行い，20 秒間の試行中 $-\pi/4 < \theta < \pi/4$ の範囲にあった時間の総和を t_{up} として比較した．ノード移動な

しの場合は **Fig.5.36** に示すように試行を重ねても一定時間以上 t_{up} の時間を長くすることができていない．実際には，この場合は頂上に到達することはできているが，そこで静止させることができないために 20 秒間頂上の間を往復しつづける．頂上付近で静止するための充分な分解能が得られていないためであると考えられる．500 ステップ時にノード移動を行った場合は **Fig.5.37** のように，静止を維持できるために振り上げている時間が長くなっている．**Fig.5.38**，**Fig.5.39** はノード移動がない場合とある場合についての状態価値関数をメッシュ状にプロットしたものである．

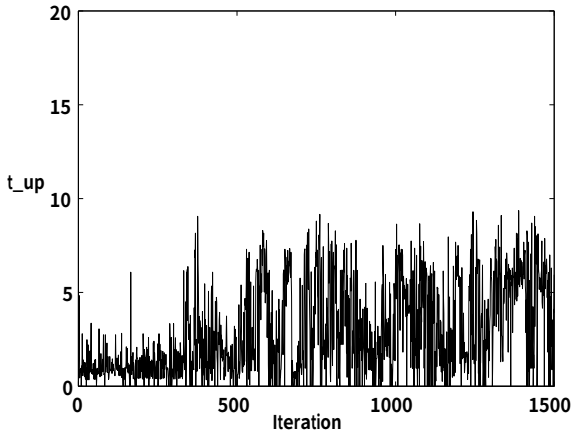

Fig. 5.36 ノード移動なしの場合

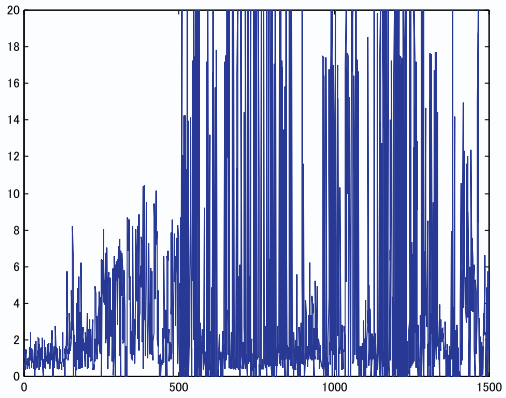

Fig. 5.37 ノード移動ありの場合

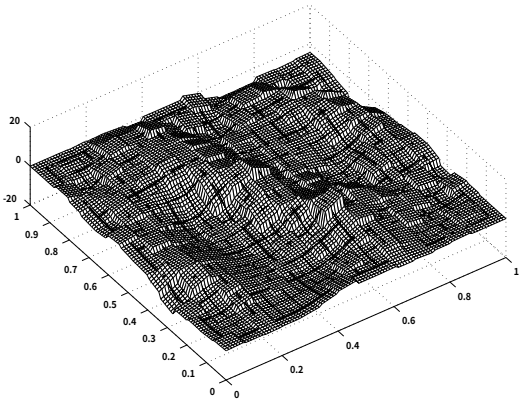

Fig. 5.38 ノード移動なしの場合の $V(\boldsymbol{x})$

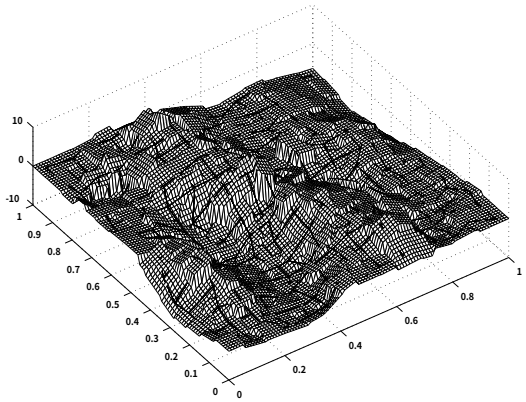

Fig. 5.39 ノード移動ありの場合の $V(\boldsymbol{x})$

Fig.5.40 は振子振り上げを達成したときの状態空間上の軌跡である．最終的に $(\theta, \omega) = (0, 0)$ 付近の領域に到達させられていることがわかる．**Fig.5.41** は同じ軌

跡に対する報酬値を表している．学習初期段階では頂上付近で角速度を充分に小さくすることができず頂上を通過してしまうが，この場合のように学習が達成されると頂上付近にとどまらせることができるようになる．

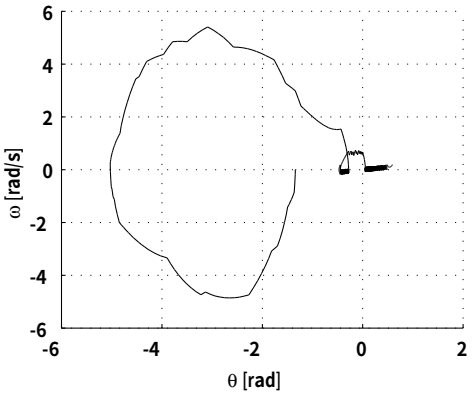

Fig. 5.40　状態空間内の軌跡

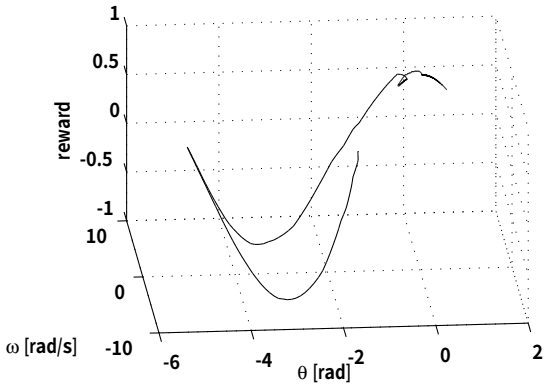

Fig. 5.41　軌跡と報酬値

また，**Fig.5.42** は格子状に固定したノード数での学習結果と逐次的にノード追加を行った場合の報酬の積算値の比較を行ったものである．右端以外のデータは格子状に配置した各ノード数に対して得られる 100 回試行における報酬積算値を示したものである．右端の*210 個のノード数は，196 個の格子状ノードから逐次的な追加とノード移動を組み合わせた結果である．数値はいずれも 10 回の平均値である．格子状のノード数を増大させていく方法と比較して，複雑度を判断基準としてノードを逐次的に追加していく方法が学習効率の上昇に寄与していることが確認された．

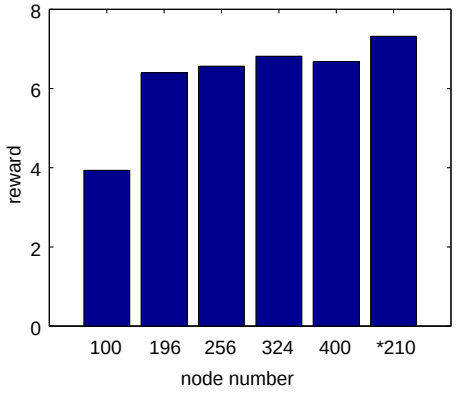

Fig. 5.42　報酬積算値の比較

5.6 おわりに

本章では，提案関数近似手法の有効性を定常関数近似問題と強化学習例題に適用して確認した．特に，

- ノードを格子状に配置した場合とノードを移動させた場合の比較
- 複雑度を表す関数値を判断基準にしたノードの追加 (と削除) の効果の検証

を行った．

定常関数近似問題では，1 次元および 2 次元のガウス関数とより複雑な形状の関数を用いて関数近似精度を中心に提案手法の評価を行った．ノードの移動により複雑度が均一化され，近似誤差が低減されることを確認した．より複雑な問題においては，完全に関数値を均一にすることが困難であるが，ノード移動に加えて逐次的なノードの追加・削除を行うことで同数のノードに対しても関数近似の効率を改善することができることを確認した．

3 章では，サンプリング定理から導かれる「信号を復元するために最低限必要なノード間隔」という考え方から適切な関数値の基準について考察した．1 次元ガウス関数近似において，この基準からみた適切なノード個数を推定した．強化学習適用例においては，水たまり問題の例で確認されたようにノード密度が高すぎる場合に逆に学習効率の低下を招くことがある．このため，この基準を強化学習の際に直接用いるのは適切でない可能性もあるが，この考え方は充分に精度よく関数近似が達成できているかという判断の一つの目安になり得ると考えられる．

強化学習例題には，水たまり問題，2 次元環境探索問題，1 自由度振子振り上げ問題を取り上げ，それぞれの問題の状態価値関数近似に提案関数近似手法を適用し，その有効性を確認した．

水たまり問題においては，同数のノードに対してノード移動を行うことで得られる報酬の積算値が改善できることを示した．しかし，この問題においては，勾配表現の適切さが得られる報酬積算値に与える影響が比較的小さいと考えられるため，ゴール領域を状態空間中中央に移動させた 2 次元環境探索問題を設けた．この問題では，中央に向かう勾配を適切に表現できているかどうかが直接的にゴールへの到達時間に影響を与えるため，水たまり問題と比較して勾配表現の適切さ

の報酬に対する影響が大きいと考えられる．2 次元環境探索問題では，ノードの逐次追加と移動を行うことで，ノード数に対して高い学習性能を得られることを示した．これにより，複雑度に注目したノードの追加が有効に機能していることが確認された．

1 自由度振子振り上げ問題では，トルクの最大値に制限があるため，反動をつけて頂上に到達する必要がある．得られる報酬は，頂上に到達できるようになる段階とさらに頂上にいる状態を維持できるようになる段階によって異なる．まず，同数のノードに対して，ノードを固定した場合と移動させた場合とで頂上で停止させられるようになる例を示した．次に，異なるノード数に対する学習性能を示し，ノードの逐次追加と移動により，約半分のノード数で同等以上の学習性能が得られることを示した．この例では，初期ノード数 196 個に対して関数値を判断基準として 15 個の追加を行った．これにより学習性能の大幅な向上が得られたことより，関数値を判断基準としたノード追加が有効に機能していることが確認された．

このように，問題の性質に依存する側面はあるが，それぞれの問題について，本研究の提案する関数近似手法による

- グラフ上の反応拡散方程式に挙動設計によるノードの移動
- 局所近傍における複雑度を表す関数値を利用したノードの追加

が，状態価値関数近似において有効に機能していることが確認された．

第6章　結論

6.1 結論

本研究では，グラフ上の反応拡散方程式に基づいた自律分散型のアルゴリズムにより，適応的に分解能を変化させる関数近似を実現し，強化学習への適用を行った．

従来の離散的・記号的表象を基礎とした強化学習研究では重視されてこなかったが，強化学習の実問題への適用においては，情報表現の効率が学習性能に大きな影響を与える．つまり，強化学習の核をなす価値関数の更新アルゴリズムにおいて適切な価値関数の関数近似を行うことが重要である．強化学習における価値関数近似には，一般に

- 非定常関数近似に適していること
- ブートストラップ型の関数近似において収束性がある程度保証されること
- 学習に必要なデータを最小限にとどめること

などの要件が存在する．分解能の高い関数近似を行うためには，関数近似要素を増やすため必要メモリが増加することに加え，学習効率の問題が存在する．分解能を高くするためにはより多くの学習データが必要であるが，学習に必要なデータの増大は行動の試行錯誤量の増大につながり学習性能の低下を招く．このような意味から，学習にとって重要な領域のみに高い分解能を持つような表現，すなわち**適応的な分解能**を獲得する関数近似手法が重要な意味を持っている．

適応的な分解能を自律的に獲得するための関連研究として，関数近似手法における逐次的な基底関数の追加方法や状態空間の自律的な分割手法などをあげることができるが，人手による調整が必要な設計要素が多いという問題が存在する．それに対し，本研究では自律的な情報表現獲得を目指し，

設計自由度を低減した適応的分解能獲得型の関数近似手法

を提案した．強化学習における適応的に分解能を獲得する関数近似でこれまで注目されてきたのは近似精度や学習達成度であったが，本研究では価値関数の形状に着目し，勾配変化を適切に表現できるように勾配変化の大きい領域に密に，小さい領域に疎に分布するように適応的に関数近似要素を再配置するアルゴリズムを提案した．系 (関数近似器) 全体の秩序との関係を記述した形での局所的な関数近似要素の挙動を設計する手法としてグラフ上の反応拡散方程式を用いた方法を提案した．

強化学習適用で想定するのは多次元入力 1 次元出力の陽関数であり，本研究では位置と勾配をもったノードによる超平面の組み合わせによって関数近似面を構

成する．このノードを近傍を表すアークで結ぶことでグラフを構成し，各ノード近傍における近似関数の複雑度を定義する．各ノードの複雑度を均一にするようにノードを移動させる．勾配変化を反映した複雑度を定義することにより，ノードが勾配変化の大きい領域に密に，小さい領域に疎に分布するように挙動を設計する．そのための設計方法としてグラフ上の反応拡散方程式を用いた．

グラフ上の反応拡散方程式は自律分散システムの理論であり，不均一に分布するサブシステムからなるシステムにおいて，全体としての秩序を形成するためのサブシステムの挙動を勾配系に基づいて設計する手法である．提案手法は，自律分散系のアルゴリズムの特徴であるシステムの拡縮性に優れるという特質をもつため，ノードの動的な追加・削除が容易であるという利点を有する．位置と勾配情報を持ったノードの近傍をアークで結ぶことによって境界付きグラフに適用を行う．このとき，ノード間のアークを構成する方法として，ユークリッドノルムに基づいた近傍を構成する TRN(Topology Representing Networks) アルゴリズムを用いた．

複雑度の定義には，各ノードの近傍ノードに対する勾配変化と位置変化の積として定義した．この定義については，直観的には勾配変化とノードの密度が比例関係にあることと複雑度が一定になることが 1 次元では等価になることで設計指針に沿ったものであるという理解ができる．さらに，1 次元において Spline 補間曲線の理論を用いて曲線近似誤差最小化問題との対応が取れることを示した．提案手法がノードの動的な追加を容易に行える方法であることに関して，また，近似目標関数を周波数領域に変換しサンプリング定理を適用することで，関数値のとるべき水準についての考察を行った．ノードの逐次的な追加に際しての関数近似がすでに充分な精度で達成されているかどうかの判断基準を従来研究のような近似誤差に基づく方法ではなく関数形状情報に基づく方法で目安を示した．

提案関数近似手法の強化学習への適用方法として，Value Gradient 法および Q-learning 法の状態価値関数および行動価値関数の近似を行う方法を示した．Value Gradient 法では，状態価値関数を TD 誤差学習により推定し，行動決定は近似した状態価値関数の勾配情報を用いて行う．Q-learning 法では，要素行動の数だけ関数近似器を独立に用意し，各関数近似器は特定の要素行動に関する行動価値関数の近似を行う．プログラム実装に際しては，入力次元やノード数の増大に伴う計算量変化について考察を行った．

提案する関数近似手法の性質の検証および強化学習例題を用いた性能向上の確認を行った．定常関数近似問題においては，位置推定および勾配推定の両者についてノード移動を適応的に行うことにより近似誤差を低減できることを確認した．

また，複雑度を表す関数値の大きい領域に逐次的にノードを追加していく方法が複雑な形状の関数を近似する上で有効に機能することを確認した．

強化学習への適用の評価に関しては，CMAC のような格子状に固定された関数近似方法との比較として，入力空間に格子状にノードを分布させ，固定したままの場合と，ノード移動や逐次的なノード追加を行った場合との比較を行った．また，ノードの追加を行うことは関連研究との対比では自律的な状態分割や基底の動的追加に対応しているが,「複雑度 (関数値) の大きなノードの近傍にノードを追加する」という追加指標が有効に機能することを示した．強化学習例題には水たまり問題と 1 自由度振子振り上げ問題を取り上げ，各場合において，ノード数の変化，ノード移動の有無，ノードの逐次的追加による学習効率の変化を検証した．

その結果,

- 固定ノード数の場合，ノード数が多い方が学習効率は改善されるが，一定個数よりも多くなりすぎると逆に学習効率は低下する．これは分解能が高すぎると逆に必要データ数の増加が強化学習性能に悪影響を与える可能性を示唆しているものと考えられる．
- ノード移動により，学習効率が改善できることを示した．この改善の割合は水たまり問題よりも問題設定を変更した 2 次元環境探索問題においてより顕著に表れた．このことから，ノード移動による適応的な分解能の変更が強化学習において有効であること，および勾配変化の表現効率が報酬値に影響を与えやすい問題ほどノード移動の効果が高くなることが確認された．
- ノードの逐次追加によって，学習効率が改善できることを示した．逐次追加を行う場所の判別に関数値 $f(u)$(複雑度) を用いることで，効率の良いノードの追加が可能であることを示した．

という結果を得た．

6.2 今後の課題

今後の課題・展望としては大きく分けて二つの方向性がある．一つは，ノードの挙動設計を改良することによりより安定したノード 移動の実現を目指す方向である．もう一つは，4 章において述べた強化学習拡張問題への適用を図り，強化学習のより複雑な問題への適用を目指す方向である．

前者の問題に関しては，

- 多次元空間 (3 次元以上) の問題に適用し，安定的に動作することを確認する
- あるノード の持つアーク同士の関係を陽に扱う挙動設計を行うことにより，本来の設計指針の要素である「勾配変化」と「測度 (体積)」との関係を的確に反映した枠組みを構築する

などが考えられる．

後者の問題に関しては

- 段階的学習，階層型強化学習問題への適用
- SMDP への適用，状態・行動空間の自律的分割
- 行動空間の汎化．Actor の近似とその評価 [木村 00]．

などが考えられる．

参考文献

[Albus75] J.S. Albus: A new approach to manipulator control: The cerebellar model articulation cotnroller (cmac). *Journal of Dynamic Systems, Measurement, and Control,Trans. ASME*, Vol. 97, No. 3, pp. 220–227, 1975.

[Albus96] J. S. Albus: The engineerng of mind. *Proceedings of Fourth International Conference on Simulation and Adaptive Behavior (From Animals to Animats 4)*, pp. 23–32, 1996.

[Amari96] S. Amari, A. Cichocki, and Y. H. Yang: A new learning algorithm for blind signal separation. *Adavences in Neural Information Processing Systems*, Vol. 8, pp. 757–763, 1996.

[Asada96] M.Asada, S.Noda, and K.Hosoda: Action-based sensor space categorization for robot learning. *Proc. of IEEE/RSJ Int. Conf. IROS96*, pp. 1502–1506, 1996.

[Atkeson97a] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schall: Locally weighted learning. *Artificial Intelligence Review*, Vol. 11, pp. 11–73, 1997.

[Atkeson97b] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schall: Locally weighted learning for control. *Artificial Intelligence Review*, Vol. 11, pp. 75–113, 1997.

[Baird94] Leemon C. Baird III: Reifoncement learning in continuous time:advantage updating. *Proc. of the International Conference on Neural Networks*, pp. 2448–2453, 1994.

[Baird95] Leemon C. Baird III: Residual algorithms: Reiforcement learning with function approximation. *Proc. of the International Conference on Machine Learning*, pp. 30–37, 1995.

[Bell95] Anthony J. Bell and Terrence J. Sejnowski: An information-maximisation approach to blind separation and blind deconvolution. *Neural Computation*, Vol. 7, No. 6, pp. 1004–1034, 1995.

[Bellman59] R. Bellman, S. Dreyfus, and a dynamic: Functional approximations and dynamic programming. *Math Tables and*

Other Aides to Computation, Vol. 13, pp. 247–251, 1959.

[Bellman63] Richard Bellman, Robert Kalaba, and Bella Kotkin: Polynomial approximation–A new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation*, Vol. 17, No. 82, pp. 155–161, 1963.

[Bertsekas96] Dimitri P. Bertsekas and John N. Tsitsiklis: *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Boyan95] Justin A. Boyan and Andrew W. Moore: Generalization in reinforcement learning : Safely approximating the value function. *Advance in Neural Information Processing Systems*, Vol. 7, , 1995.

[Bradtke94] Steven J. Bradtke and Michael O. Duff: Reiforcement learning methods for continuous-time markov decision problems. *Advances in Neural Information Processing Systems*, Vol. 7, pp. 393–400, 1994.

[Bruske95] Joerg Bruske and Gerald Sommer: Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, Vol. 7, pp. 845–865, 1995.

[Carpenter91] G.A. Carpenter, S.Grossberg, and D.B. Rosen: Fuzzy art: Fast stable learning and categorization of analogue patterns by an adaptive resonance system. *Neural Networks*, Vol. 4, pp. 759–771, 1991.

[Catmull78] E. Catmull and J. Clark: Recursively generated B-spline surfaces on arbitary topological meshes. *Computer Aided Design*, Vol. 10, pp. 350–355, 1978.

[Comaniciu02] Dorin Comaniciu and Peter Meer: Mean shift: A robust approach toward feature space analysis. *Trans. Pattern Analysis Machine Intell.*, 2002.

[Connell96] Jonalthan H. Connell and Sridhar Mahadevan: *ROBOT LEARNING*. Kluwer Academic Publishers, 1996.

[Dayan93] P. Dayan and G. E. Hinton: Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, Vol. 5, pp. 271–278, 1993.

[Doo78] D. Doo: A subdivision algorithm for smoothing down irregularly shaped polyhedrons. *Proceedings of Interactive Techniques in Computer Aided Design*, Vol. 157, No. 157, pp. 157–157, 1978.

[Dorigo95] Marco Dorigo: Alecsys and the autonomouse: Learning to control a real robot by distributed classifier systems. *Machine Learning*, Vol. 19, pp. 209–209, 1995.

[Doya96] Kenji Doya: Temporal difference learning in continuous time and space. *Advances in Neural Information Processing Systems*, Vol. 8, pp. 1073–1079, 1996.

[Doya00] Kenji Doya: Reifoncement learning in continuous time and space. *Neural Computation*, Vol. 12, pp. 243–269, 2000.

[Fritzke94] Bernd Fritzke: Fast learning with incremental rbf networks. *Neural Processing Letters*, Vol. 1, No. 1, pp. 2–5, 1994.

[Fritzke95] Bernd Fritzke: A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, Vol. 7, pp. 625–632, 1995.

[Gordon95] Geoffrey J. Gordon: Stable function approximation in dynamic programming. *Proc. of the International Conference on Machine Learning*, pp. 261–268, 1995.

[Gullapalli90] Vijaykumar Gullapalli: A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, Vol. 3, pp. 671–692, 1990.

[Harmon96] Mance E. Harmon and Leemon C. Baird III: Reinforcement learning applied to a differential game. *Adaptive Behavior*, Vol. 4, pp. 3–28, 1996.

[Haruno99] Masahiko Haruno, Daniel Wolpert, and Mitsuo Kawato: Multiple paired forward-inverse models for human motor learning and control. *Advance in Neural Information Processing Systems*, Vol. 11, pp. 31–37, 1999.

[Ishiguro96] H.Ishiguro, R.Sato, and T.Ishida: Robot oriented state space construction. *Proc. of IEEE/RSJ Int. Conf. IROS96*, pp. 1496–1501, 1996.

[Kaelbling96] Leslie pack Kaelbling, Michael L. Littman, and Andrew W. Moore: Reinforcement learning, a survey. *Journal of Artificial Intelligence Research*, No. 4, pp. 237–285, 1996.

[Kobayashi00] Yuichi Kobayashi, Hideo Yuasa, and Tamio Arai: Learning using multidimensional internal rewards. *IEEE/RSJ Int. Conf. on Intelligent Robot and Systems*, 2000.

[Linsker88] Ralph Linsker: Self-organization in a perceptual network. *Computer*, Vol. 21, pp. 105–117, 1988.

[Lin93] Long-Ji Lin and Tom M. Mitchell: Reinforcement learning with hidden states. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 271–280, 1993.

[Luo98] Zhiwei Luo and Masami Ito: Diffusion-based learning theory for organizing visuo-motor coordination. *Biological Cybernetics*, No. 79, pp. 279–289, 1998.

[Mahadevan96] Sridhar Mahadevan: Machine learning for robots: A comparison of different paradigms. *IEEE/RSJ Int. Conf. on Intelligent Robot and Systems*, 1996.

[Martinetz93] Thomas Martinetz, Stanislav Berkovich, and Klaus Schulten: 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE-Transactions on Neural Networks*, Vol. 4, pp. 558–569, 1993.

[Martinetz94] Thomas Martinetz and Klaus Shulten: Topology representing networks. *Neural Networks*, Vol. 7, No. 3, pp. 507–522,

1994.

[Minsky61] Marvin Minsky: Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, Vol. 49:8-30, pp. 406–450, 1961.

[Moore95] Andrew W. Moore: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, Vol. 21, No. 3, pp. 199–233, 1995.

[Morimoto98] Jun Morimoto and Kenji Doya: Reinforcement learning of dynamic motor sequences: learning to stand up. *Proc. of IEEE/RSJ International Conference of Intelligent Robots and Systems(IROS)'98*, Vol. 3, pp. 1721–1726, 1998.

[Munos01] Rémi Munos and Andrew Moore: Variable resolution discretization in optimal control. *Machine Learning*, Vol. 1, pp. 1–31, 2001.

[Murao97] Hajime Murao and Shinzo Kitamura: Q-learning with adaptive state segmentation(qlass). *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation(CIRA)*, pp. 179–184, 1997.

[Parr98] Ron Parr and Stuart Russell: Reiforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*, Vol. 10, pp. 1043–1049, 1998.

[Peng93] Jing Peng and Ronald J. Williams: Efficient learning and planning within the dyna framework. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 281–290, 1993.

[Press92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery: *Neumerical Recipes in C*. Cambridge University Press, 1992.

[Rumelhart86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams: Learning internal representations by error propagation. In D. E.

Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations*, pp. 318–362. Bradford Books/MIT Press, Cambridge, MA., 1986.

[Rummery94] Rummery G. A. and Niranjan M.: On-line q-learning using connectionist systems. *Technical Report CUED/F-INFEG/TR 166*, 1994.

[Samejima98] Kazuyuki Samejima and Takashi Omori: Adaptive state space formation in reinforcement learning. *Proc. of International Conference on Neural Information Processing(ICONIP)'98*, pp. 251–255, 1998.

[Santamaria98] Ashwin Ram Juan C. Santamaria, Richard S. Sutton: Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, Vol. 6, No. 2, pp. 163–218, 1998.

[Sato99] Masa aki Sato and Shin Ishii: Reinforcement learning based on on-line em algorithm. *Advances in Neural Information Processing Systems*, No. 11, 1999.

[Schaal94] Stefan Schall and Christopher G. Atkeson: Robot juggling: An implementation of memory-based learning. *Control Systems*, Vol. 14, No. 1, pp. 57–71, 1994.

[Schaal97] Stefan Schaal. Learning from demonstration: In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, Vol. 9, p. 1040. The MIT Press, 1997.

[Sutton88] Richard S. Sutton: Learning to predict by the methods of temporal differences. *Machine Learning*, No. 3, pp. 9–44, 1988.

[Sutton96] Richard S. Sutton: Generalization in reinforcement learning:successful example using sparse coarse coding. *Advences*

in Neural Information Processing Systems, No. 8, pp. 1038–1044, 1996.

[Sutton98] Richard S.Sutton and Andrew G.Barto: *Reinforcement Learning*. MIT Press, 1998.

[Sutton00] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour: Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, Vol. 12, pp. 1057–1063, 2000.

[Tesauro95] G. Tesauro: Temporal difference learning and td-gammon. *Communication of the ACM*, Vol. 38, pp. 58–68, 1995.

[Touzet97] Claude F. Touzet: Neural reinforcement learning for behaviour synthesis. *Robotics and autonomous systems*, No. 22, pp. 251–281, 1997.

[Tsitsiklis97] John N. Tsitsiklis and Benjamin Van Roy: An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, Vol. 42, No. 5, pp. 674–690, 1997.

[Watkins89] C. Watkins: *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[Watkins92] C. Watkins and P. Dayan: Q–learning. *Machine Learning*, No. 8, pp. 279–292, 1992.

[Williams92] Ronald J. Williams: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, No. 8, pp. 229–256, 1992.

[Zhang95] Wei Zhang and Thomas G. Dietterich: High-performance job-shop scheduling with a time-delay td(λ) network. *Advances in Neural Information Processing Systems*, pp. 1024–1030, 1995.

[浅田 97] 浅田稔: 強化学習の実ロボットへの応用とその課題. 人工知能学会誌, Vol. 12, No. 6, pp. 23–28, 1997.

[浅田 99] 浅田稔, 石黒浩, 國吉康夫: 認知ロボティクスの目指すもの. 日本ロボット学会誌, Vol. 17, No. 1, pp. 2–6, 1999.

[伊藤 94] 伊藤聡, 湯浅秀男, 伊藤正美: 反応拡散方程式を用いた自己想起型連想記憶による画像認 識. 計測自動制御学会論文集, Vol. 30, No. 1, pp. 97–103, 1994.

[伊藤 95] 伊藤正美, 市川淳信, 須田信英: 自律分散システム宣言. オーム社, 1995.

[上山 00] 上山英三, 湯浅秀男, 細江繁幸: 動きの境界の移流と拡散によって実現される動き推定. 電子情報通信学会論文誌, Vol. J83-D-II, No. 2, pp. 653–661, 2000.

[臼井 95] 臼井支朗, 岩田彰, 久間和生, 淺川和雄: 基礎と実践ニューラルネットワーク. コロナ社, 1995.

[浦川 96] 浦川肇: ラプラス作用素とネットワーク. 裳華房, 1996.

[川人 96] 川人光男: 脳の計算理論. 産業図書, 1996.

[川人 00] 川人光男: 川人学習動態脳プロジェクト. 日本ロボット学会誌, Vol. 18, No. 8, pp. 1074–1080, 2000.

[菊地 80] 菊地文雄: 有限要素法概説. サイエンス社, 1980.

[木村 96] 木村元, 山村雅幸, 小林重信: 部分観測マルコフ決定過程下での強化学習：確率的傾斜法 による接近. 人工知能学会誌, Vol. 11, No. 5, pp. 761–768, 1996.

[木村 99] 木村元, 宮崎和光, 小林重信: 強化学習システムの設計指針. 計測と制御, Vol. 38, No. 10, pp. 618–623, 1999.

[木村 00] 木村元, 小林重信: Actor に適正度の履歴を用いた actor-critic アルゴリズム：不完全な value-function のもとでの強化学習. 人工知能学会誌, Vol. 15, No. 2, pp. 267–275, 2000.

[黒瀬 99] 黒瀬能聿: 3 次元図形処理工学. 共立出版, 1999.

[小林 00] 小林祐一, 井上康介, 太田順, 新井民夫: 視覚情報を用いた状態・行動空間の自律的生成. 計測自動制御学会論文集, Vol. 36, No. 11, pp. 1029–1036, 2000.

[コホネン 96] T. コホネン: 自己組織化マップ. シュプリンガー・フェアラーク東京, 1996. 徳高平蔵, 岸田悟, 藤村喜久郎訳.

[坂和 80] 坂和愛幸: 最適化と最適制御. 森北出版, 1980.

[新 95] 新誠一, 池田健司, 湯浅秀男, 藤田博之: 自律分散システム. 朝倉書店, 1995.

[杉原 94] 杉原厚吉: 計算幾何工学. 培風館, 1994.

[高木 96] 高木隆司: ベクトル解析. 岩波書店, 1996.

[高橋 99] 高橋泰岳, 浅田稔: 実ロボットによる行動学習のための状態空間の漸次的構成. 日本ロボット学会誌, Vol. 17, No. 1, pp. 118–124, 1999.

[竹之内 86] 竹之内脩, 西白保敏彦: 近似理論–関数の近似. 培風館, 1986.

[チャールズ 94] チャールズ K. チュウイ: ウェーブレット入門. 電気大出版局, 1994. 桜井明・新井勉共訳.

[鳥谷 91] 鳥谷浩志, 千代倉弘明: 3 次元 CAD の基礎と応用. 共立出版, 1991.

[銅谷 99] 銅谷賢治: 脳科学とロボット. 日本ロボット学会誌, Vol. 17, No. 1, pp. 7–10, 1999.

[中溝 88] 中溝高好: 信号解析とシステム同定. コロナ社, 1988.

[ポントリャーギン 00] L. S. ポントリャーギン: 最適制御理論における最大値原理. 森北出版, 2000. 坂本 實訳.

[美多 84] 美多勉: ディジタル制御理論. 昭晃堂, 1984.

[柳井 83] 柳井晴夫, 竹内啓: 射影行列・一般逆行列・特異値分解. 東京大学出版会, 1983.

[山本 99] 山本徹也, 鈴木宏正, 金井崇, 木村文彦: 三角形メッシュモデル細分割手法の拡張と評価. 精密工学会誌, Vol. 65, No. 3, pp. 386–390, 1999.

[湯浅 99] 湯浅秀男, 伊藤正美: グラフ上の反応拡散方程式と自律分散システム. 計測自動制御学会論文集, Vol. 31, No. 1, pp. 1–7, 1999.

[湯浅 02] 湯浅秀男, 新井民夫: 移動ロボットにおける知能とセンサ–アクチュエータ系の 双対性. 第 14 回自律分散システム・シンポジウム資料, pp. 219–224, 2002.

[ベルマン 62] R. ベルマン, S. ドレイファス: 応用ダイナミック・プログラミング. 日科技連, 1962. 小田中敏男，有水彊訳.

[マクドーマン 99] Karl F. MacDorman: 感覚—運動統合による記号接地. 日本ロボット学会誌, Vol. 17, No. 1, pp. 20–24, 1999.

[森本 99] 森本淳, 銅谷賢治: 強化学習を用いた高次元連続状態空間における時系列運動学習：起き上がり運動の獲得. 電子情報通信学会論文誌 D–II, Vol. J82–D–II, pp. 2119–2131, 1999.

[水野 96] 水野欽司: 多変量データ解析講義. 朝倉書店, 1996.

研究業績

学位論文

- 小林 祐一, “リリース型マニピュレーションによる物体の受け渡し操作,” 東京大学卒業論文,1997.
- 小林 祐一, “視覚情報を利用した状態・行動空間の自律的生成,” 東京大学大学院修士論文,1999.

査読つき学術論文

- 小林祐一，太田順，井上康介，新井民夫, “視覚情報を用いた状態・行動空間の自律的生成,” 計測自動制御学会論文集 第 36 巻 11 号,1029-1036,2000
- 小林 祐一， 湯浅 秀男， 新井 民夫，“自律分散系の適応アルゴリズムによる強化学習のための関数近似,” 計測自動制御学会論文集 第 38 巻 2 号,(項未定),2002

査読つき講演論文

- Yuichi KOBAYASHI, Jun OTA, Kousuke INOUE and Tamio ARAI, “State and Action Space Construction Using Vision Information,” Proc. of IEEE International Conference on Systems, Man and Cybernetics, Tokyo, Japan, 1999.
- Yuichi KOBAYASHI, Hideo YUASA and Tamio ARAI, “Learning using multidimensional internal rewards,” Proc. of IEEE/RSJ International Conference on Intelligent Robot and Systems(IROS) ,Takamatsu, Japan, 2000.
- Masahiro YOKOI, Yuichi KOBAYASHI, Takeshi FUKASE, Hideo YUASA and Tamio ARAI, “Learning Self-ocalization with Teaching System,” Proc. of IEEE/RSJ International Conference on Intelligent Robot and Systems(IROS),Takamatsu, Japan, 2000.
- Takeshi FUKASE, Masahiro YOKOI, Yuichi KOBAYASHI, Hideo YUASA and Tamio ARAI, “Quadruped Robot Navigation Considering the Observational Cost,” Proc. of The RoboCup 2001 International Symposium, Seattle, USA, 2001.

口頭発表

- 小林 祐一，相山 康道，朱 赤， 新井 民夫，“リリース型マニピュレーションを利用した物体の受け渡し操作,” 1996 年度精密機械工学会春期大会学術講演会予稿集，Vol.3, pp.985-986,1997.

- 小林 祐一， 相山 康道，井上 康介， 新井 民夫，“GA による物体の押し・弾き操作の獲得,” 第 15 回 日本ロボット学会学術講演会予稿集，pp.159-160,1997.

- 小林 祐一，相山 康道，井上 康介，太田 順，新井 民夫，“自己組織型ハンドアイシステム,” 日本機会学会ロボティクス・メカトロニクス講演会’98 講演論文集，1CII3-3,1/2,1998.

- 小林 祐一，井上 康介，相山 康道，太田 順，新井 民夫，“押し操作における視覚情報からの状態空間の自律的生成,” 第 16 回日本ロボット学会学術講演会予稿集, Vol.1, pp. 415-416,1998.

- 小林 祐一， 相山 康道，井上 康介，太田 順，新井 民夫，“視覚・接触情報を用いた状態空間の自律的生成,” 第 11 回自律分散システム・シンポジウム資料, pp. 275-280, 1999.

- 井上 康介，太田 順，千葉 龍介，小林 祐一，新井 民夫，“部分観測環境下における強化学習による移動ロボットの行動獲得,” 第 11 回自律分散システム・シンポジウム資料, pp.271-274,1999.

- 千葉龍介，太田順，井上康介，小林祐一，新井民夫，“部分観測 Markov 決定過程における自律的状態分割によるロボットの強化学習,” 日本機械学会ロボティクス・メカトロニクス講演会講演論文集, 2A1-27-041,1/2 ,1999.

- 小林 祐一，太田 順，湯浅 秀男，井上 康介，新井 民夫，“脚型ロボットにおける視覚情報と蹴球動作の関係の獲得,” 日本ロボット学会学術講演会予稿集,Vol.3, pp.1003-1004,1999.

- 小林 祐一，湯浅 秀男，太田 順，新井 民夫，“遅れ報酬を含む多次元評価にもとづく状態と行動の獲得,” 第 12 回自律分散システム・シンポジウム資料,

pp.463-466,2000.

- 井上康介，太田順，小林祐一，新井民夫，“部分観測環境下における自律的状態分割による強化学習,” 第 12 回自律分散システム・シンポジウム資料, pp.223-226 ,2000.

- 上田 隆一，小林 祐一，横井 真浩，深瀬 武，湯浅 秀男，新井民夫，“四脚ロボットにおける Monte Carlo Localization に基づく環境認識, ” 日本機会学会ロボティクス・メカトロニクス講演会’01 講演論文集，2001.

- 小林 祐一，湯浅 秀男，新井 民夫，“多次元遅延信号に基づく強化学習,” 第 13 回自律分散システム・シンポジウム資料，pp.407-410,2001.

- 井上康介，太田順，小林祐一，湯浅秀男，新井民夫，“局所センサ入力に基づく移動ロボットのナビゲーション行動の学習,” 第 13 回自律分散システム・シンポジウム資料, pp.379-382 ,2001.

解説記事

- 小林祐一，湯浅秀男，新井民夫，“脚型ロボットを用いたロボットサッカー,” 精密工学会誌 Vol.2, No.66 pp.185–188, 2000.

付 録A　近似曲線の理論

A.1 Spline補間

Spline 補間曲線は，2 点間の 2 次導関数を線形補間でつなぐ 3 次曲線であり，節点の位置の他に，各点での 2 次導関数 y_i'', y_{i+1}'' をパラメータとする (**Fig.A.1**).

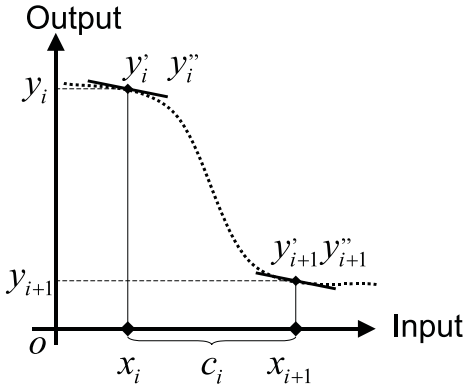

Fig. A.1　2 個の点による補間

式は以下のようになる.

$$y = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}'' \qquad (\text{A.1.1})$$

ただし各項は

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i}, \quad B = \frac{x - x_i}{x_{i+1} - x_i} \qquad (\text{A.1.2})$$

$$C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2, \quad D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2 \qquad (\text{A.1.3})$$

のような多項式である．ここから導関数を導出すると，

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{3A^2 - 1}{6}(x_{i+1} - x_i)y_i'' + \frac{3B^2 - 1}{6}(x_{i+1} - x_i)y_{i+1}'' \qquad (\text{A.1.4})$$

$$\frac{d^2y}{dx^2} = Ay_i'' + By_{i+1}'' \qquad (\text{A.1.5})$$

となる．(3.3.22) 式より，2 次導関数が線形補間されていることがわかる．また，拘束条件は，各節点での 1 次導関数が連続であるという条件から

$$\frac{(x_i - x_{i-1})}{6}y_{i-1}'' + \frac{(x_{i+1} - x_{i-1})}{3}y_i'' + \frac{(x_{i+1} - x_i)}{6}y_{i+1}'' = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \qquad (\text{A.1.6})$$

のように与えられる．各点における 1 次，3 次導関数は

$$y'_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{3} y''_i - \frac{x_{i+1} - x_i}{6} y''_{i+1} \quad (\text{A.1.7})$$

$$y'_{i+1} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{x_{i+1} - x_i}{6} y''_i + \frac{x_{i+1} - x_i}{3} y''_{i+1} \quad (\text{A.1.8})$$

$$y'''_i = y'''_{i+1} = \frac{y''_{i+1} - y''_i}{x_{i+1} - x_i} \quad (\text{A.1.9})$$

となる．

A.2 Ferguson 曲線

Ferguson 曲線は勾配と位置情報をもとに曲線を構成する計算式で，Bezier 曲線, Spline 曲線の基礎である [黒瀬 99]．始点および終点の位置ベクトルを $\boldsymbol{w}_u, \boldsymbol{w}_h$，始点および終点における方向ベクトルを $\boldsymbol{v}_o, \boldsymbol{v}_t$ としたとき，これをつなぐ Ferguson 曲線の軌跡 P は媒介変数 $s(0 \leq s \leq 1)$ を用いて，

$$P(s) = \boldsymbol{p}_3 s^3 + \boldsymbol{p}_2 s^2 + \boldsymbol{p}_1 s + \boldsymbol{p}_0 \qquad (\text{A.2.1})$$

のように表せる．ただし係数ベクトル $(\boldsymbol{p}_0, \cdots, \boldsymbol{p}_3)$ は以下のように表される．

$$\boldsymbol{p}_3 = 2\boldsymbol{w}_u - 2\boldsymbol{w}_h + \boldsymbol{v}_o + \boldsymbol{v}_t \qquad (\text{A.2.2})$$

$$\boldsymbol{p}_2 = -3\boldsymbol{w}_u + 3\boldsymbol{w}_h - 2\boldsymbol{v}_o - \boldsymbol{v}_t \qquad (\text{A.2.3})$$

$$\boldsymbol{p}_1 = \boldsymbol{v}_o \qquad (\text{A.2.4})$$

$$\boldsymbol{p}_0 = \boldsymbol{w}_u \qquad (\text{A.2.5})$$

Fig.A.2 に Ferguson 曲線の概念を示す．始点ベクトル $\boldsymbol{v}_o$ と終点ベクトル $\boldsymbol{v}_t$ の大きさにより曲線の曲率は変化する．

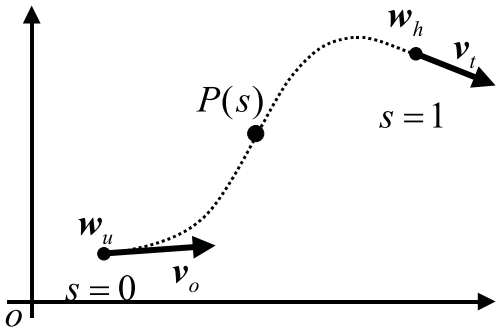

Fig. A.2 Ferguson 曲線

A.3 Spline 補間曲線と Ferguson 曲線の関係

1 次元入力，1 次元出力系において Ferguson 曲線と Spline 補間曲線の関係を考える．ここでは，2 点 $(x_0, y_0), (x_1, y_1)$ を結ぶ両曲線がどのような条件で一致するかについて考える．

まず，前節で述べたように Ferguson 曲線を決定するパラメータは $\boldsymbol{w}_u, \boldsymbol{w}_h, \boldsymbol{v}_o, \boldsymbol{v}_t$ である．このうち 2 点の位置ベクトルは条件から与えられる．2 点における方向ベクトルは，Spline 補間曲線と勾配が等しくなることが両曲線が一致するために必要である．この条件をもとに，両曲線が一致する方向ベクトルの大きさを考える．

Spline 補間曲線では，両端点における勾配は，各点での 2 次導関数 y_0'', y_1'' を用いて

$$y_0' = \frac{y_1 - y_0}{x_1 - x_0} - \frac{x_1 - x_0}{3} y_0'' - \frac{x_1 - x_0}{6} y_1'' \qquad (\text{A.3.1})$$

$$y_1' = \frac{y_1 - y_0}{x_1 - x_0} + \frac{x_1 - x_0}{6} y_0'' + \frac{x_1 - x_0}{3} y_1'' \qquad (\text{A.3.2})$$

と表せる．これより，$p \equiv y_0'', q \equiv y_1''$ として両端点における方向ベクトル $\boldsymbol{v}_0, \boldsymbol{v}_1$ を

$$\boldsymbol{v}_0 = \begin{pmatrix} v_{0x} \\ v_{0x}\left(\frac{y_1 - y_0}{x_1 - x_0} - \frac{x_1 - x_0}{3} p - \frac{x_1 - x_0}{6} q\right) \end{pmatrix} \qquad (\text{A.3.3})$$

$$\boldsymbol{v}_1 = \begin{pmatrix} v_{1x} \\ v_{1x}\left(\frac{y_1 - y_0}{x_1 - x_0} + \frac{x_1 - x_0}{6} p + \frac{x_1 - x_0}{3} q\right) \end{pmatrix} \qquad (\text{A.3.4})$$

のように定める．これにより，Ferguson 曲線を決定するための方向ベクトルは v_{0x}, v_{1x}, p, q の 4 変数により決定できる．この方向ベクトルを用いて Ferguson 曲線の軌跡が媒介変数 s を用いて $P(s, v_{0x}, v_{1x}, p, q)$ のように表せる．Ferguson 曲線上の 1 次導関数は

$$\frac{dy}{dx}(s) = \frac{\frac{\partial P_y(s)}{\partial s}}{\frac{\partial P_x(s)}{\partial s}}, \qquad (\text{A.3.5})$$

2 次導関数は

$$\begin{aligned} \frac{d^2y}{dx^2}(s) &= \frac{d}{dx}\left(\frac{\frac{\partial P_y(s)}{\partial s}}{\frac{\partial P_x(s)}{\partial s}}\right) \\ &= \frac{1}{\frac{\partial P_x(s)}{\partial s}} \cdot \frac{\partial}{\partial s}\left(\frac{\frac{\partial P_y(s)}{\partial s}}{\frac{\partial P_x(s)}{\partial s}}\right) \end{aligned} \qquad (\text{A.3.6})$$

のように表せる．これを上記の $P(s, v_{0x}, v_{1x}, p, q)$ に適用した 2 次導関数を求めたものを $\frac{d^2y}{dx^2}(s, v_{0x}, v_{1x}, p, q)$ と表す．この Ferguson 曲線が Spline 補間曲線と一致す

るための必要条件として，$s = 0, s = 1$ における 2 次導関数が p, q に等しくなることが挙げられる．そこで p, q を未知数として連立方程式

$$\frac{d^2y}{dx^2}(0, v_{0x}, v_{1x}, p, q) = p \qquad (\text{A.3.7})$$

$$\frac{d^2y}{dx^2}(1, v_{0x}, v_{1x}, p, q) = q \qquad (\text{A.3.8})$$

を解くと，

$$v_{0x} = x_1 - x_0, \quad v_{1x} = x_1 - x_0 \qquad (\text{A.3.9})$$

を得る．この解は $\frac{d^3y}{dx^3} = \frac{p-q}{x_1-x_0}(\text{const.})$ を満たし，区間内で 3 次導関数一定の曲線を構成できる．すなわち，Ferguson 曲線のパラメータである方向ベクトルの決定において，(A.3.9) 式のようにすれば，Spline 補間曲線と一致するということがいえる．

付 録B　グラフ上の反応拡散方程式

B.1 グラフ上の反応拡散方程式の導出

ここでは, 湯浅によるグラフ上の反応拡散方程式の導出過程の一部を示す [湯浅 99].

グラフ上の頂点を u として，頂点の集合を V とする．また，頂点を結ぶ辺 (相互作用) を e として，辺の集合を E とする．そして，グラフをこの二項組 $G = (V, E)$ で表現する．G はここで，有限な有向グラフとする．

Definition 1 (境界付きグラフ) 有限グラフ $G = (V, E)$ の頂点の集合 V がが内部点の集合 $\bar{V}$ と境界点の集合 ∂V に互いに素に分かれ，かつ辺の集合 $\bar{E}$ と境界点の集合 ∂E に互いに素に分かれ，かつ次の 2 条件を満たすとき，これを境界付きグラフと呼ぶ.

$$\begin{array}{lll} (1) e \in \bar{E} & \Leftrightarrow & u \in \bar{V} \text{ and } v \in \bar{V} \\ (2) e \in \partial E & \Leftrightarrow & \{u \in \bar{V} \text{ and } v \in \partial V\} \text{ or } \{u \in \partial V \text{ and } v \in \bar{V}\} \end{array} \qquad (\text{B.1.1})$$

ここで，頂点 u, v は辺 e の端点である.

$C(V)$ を V 上の実数値関数全体の空間，$C(E)$ を E 上の実数値関数全体の空間，$f \in C(V)$ に対して，$df(e) = f(t(e)) - f(o(e))$ を対応づける作用素 $d : C(V) \to C(E)$ を余微分といい，df を f の勾配という [浦川 96]．ここで，$o(e)$ は辺 e の始点となる頂点，$t(e)$ は辺 e の終点となる頂点である.

$f_1, f_2 \in C(V)$ に対して，内積を

$$(f_1, f_2) = \sum_{u \in V} f_1(u) f_2(u) \qquad (\text{B.1.2})$$

と定義し，対応する 2 乗ノルムを $\|f\| = \sqrt{(f, f)}$ とする．すると，$C(V)$ を完備化することによりヒルベルト空間 $L_2(V)$ が得られる.

$$L_2(V) = \{f \in C(V) : \|f\| < \infty\} \qquad (\text{B.1.3})$$

同様に，$\psi_1, \psi_2 \in C(E)$ に対して，内積を

$$(\psi_1, \psi_2) = \sum_{u \in E} \psi_1(u) \psi_2(u) \qquad (\text{B.1.4})$$

と定義し，対応する 2 乗ノルムを $\|\psi\| = \sqrt{(\psi, \psi)}$ とする．すると，$C(E)$ を完備化することによりヒルベルト空間 $L_2(E)$ が得られる.

$$L_2(E) = \{\psi \in C(E) : \|\psi\| < \infty\} \qquad (\text{B.1.5})$$

局所有限グラフ G のラプラシアン Δ_A は，$f \in C(V)$ に対して，

$$\begin{aligned}(\Delta_A f)(u) &= \deg(u) f(u) - \sum_{u\sim v} f(v) \\ &= \sum_{u\sim v} (f(u) - f(v))\end{aligned} \tag{B.1.6}$$

と定義される [浦川 96]．ここで，

ポテンシャル汎関数 $W(f)$ を，

$$W(f) = W_0(f) + W_1(df) \tag{B.1.7}$$

$$W_0(f) = \alpha_R \sum_{u\in\bar{V}} f(u) \tag{B.1.8}$$

$$W_1(df) = \beta_D \sum_{u\in\bar{V}} \parallel \mathbf{df}(u) \parallel^2 \tag{B.1.9}$$

とする．

$W_1(df)$ は余微分を含むので，任意の変動関数 $h \in L_2(V)$，微小パラメータ $\varepsilon \in R$ を用いて，$W_1(d(f+\varepsilon h))$ を ε により展開する．

$$\begin{aligned}W_1(d(f+\varepsilon h)) &= \sum_{u\in\bar{V}} F_d^{(u)}(\mathbf{df}(u) + \varepsilon \mathbf{dh}(u)) \\ &= \sum_{u\in\bar{V}} \left\{ F_d^{(u)}(\mathbf{df}(u)) + \varepsilon \sum_{e\in E(u)} \frac{\partial F_d^{(u)}(\mathbf{df}(u))}{\partial df(e)} dh(e) + \mathcal{O}(\varepsilon^2)\mathbf{dh}(u) \right\} \\ &= \sum_{u\in\bar{V}} F_d^{(u)}(\mathbf{df}(u)) + \varepsilon \sum_{u\in\bar{V}} \sum_{e\in E(u)} \frac{\partial F_d^{(u)}(\mathbf{df}(u))}{\partial df(e)} dh(e) \\ &+ \quad \sum_{u\in\bar{V}} \mathcal{O}(\varepsilon^2)\end{aligned} \tag{B.1.10}$$

これにより，

$$\begin{aligned}dW_1(d(f,h)) &= \lim_{\varepsilon\to 0} \frac{W_1(d(f+\epsilon h)) - W_1(df)}{\varepsilon} \\ &= \sum_{u\in\bar{V}} \sum_{e\in E(u)} \frac{\partial F_d^{(u)}(\mathbf{df}(u))}{\partial df(e)} dh(e)\end{aligned} \tag{B.1.11}$$

となる．ここで，任意の内部辺 e において

$$\begin{aligned}\frac{\partial F_d^{(t(e))}(\mathbf{df}(t(e)))}{\partial df(e)} &= \frac{\partial F_d^{(o(e))}(\mathbf{df}(o(e)))}{\partial df(e)} \\ &\equiv dF_d(e)\end{aligned} \tag{B.1.12}$$

が成り立つと仮定する．

$$\sum_{e\in E(u)} \mathrm{sign}(u,e) dF_d(e) \equiv \nabla^{(u)} \cdot \mathbf{dF}_d(u) \qquad (\text{B.1.13})$$

ただし，$\mathbf{dF}_d(u) \equiv (dF_d(e))_{e\in E(u)}$ は，頂点 u につながる辺における $F_d^{(u)}$ の偏微係数をならべたベクトルである．最終的に $W_1(df)$ は，

$$\frac{\delta W_1(df)}{\delta f} = 2\nabla^{(u)} \cdot \mathbf{dF}_d(u) \qquad (\text{B.1.14})$$

のように Fréchet 微分できる．よって，ポテンシャル汎関数 $W_0(f)$ の勾配系は

$$\frac{\partial f}{\partial t} = -\frac{\delta W_0(f)}{\delta f} \qquad (\text{B.1.15})$$

$$= -F_r'(f) - 2\nabla^{(u)} \cdot \mathbf{dF}_d(u) \qquad (\text{B.1.16})$$

のように表現できる．

B.2 グラフの位相構造構築方法:Topology Representing Networks

TRN(Topology Representing Networks)は，SOM(Self Organizing Maps)[コホネン 96]のように格子状に限定されない位相構造を保存してベクトル量子化を行うことを目的として提案されたアルゴリズムである [Martinetz94]．入力ベクトルは，ノードとして表現される代表点によって離散化される．ここでの位相構造の保存とは，入力ベクトルの離散的表現であるノード同士の関係において，隣接する (アークで結ばれている) ノードに対応する入力ベクトルの部分空間同士が隣接していることを表す．ネットワークには N 個のノードが存在し，各ノード i が D 次元の入力に対応する結合係数 $\boldsymbol{w}_i(i = 1, \cdots, N)$ を持っている．入力ベクトル $\boldsymbol{x}$ は R^D の部分空間 M の要素である．ここでは TRN アルゴリズムのうち，位相構造を保存する部分を説明する．

(1) $\boldsymbol{w}_i \in R^D(i = 1, \cdots, N)$，各ノード (ノード i とノード j) 同士の結合 C_{ij} を 0 に初期化する ($C_{ij} = 0$ は結合なし，$C_{ij} = 1$ は結合ありを意味する)
(2) ある入力ベクトル $\boldsymbol{x} \in M(\subset R^D)$ を与える．
(3) 入力 $\boldsymbol{x}$ に対し

$$\| \boldsymbol{x} - \boldsymbol{w}_j \|^2 = \sum_{i=1}^{D} (x_i - w_{ij})^2 \qquad (\text{B.2.1})$$

によってユークリッド距離を計算し，この距離の最小のノード i_0 および 2 番目に小さいノード i_1 を発見する (ノード i_0 を最整合ノードと呼ぶ)
(4) $C_{i_0 i_1} = 0$ ならば $C_{i_0 i_1} = 1, t_{i_0 i_1} = 0$ とし (ノード i_0 とノード i_1 を結合させ)，$C_{i_0 i_1} = 1$ ならば $t_{i_0 i_1} = 0$ とする (結合を refresh する)
(5) ノード i_0 にすでに存在するすべての結合に対して $t_{i_0 j} = t_{i_0 j} + 1$ によって結合を古くする
(6) $C_{i_0 j} = 1$ かつ $t_{i_0 j} > T$ なる j について $C_{i_0 j} = 0$ とする (ノード i_0 の古くなった結合を削除する)
(7) 2. に戻る

ここで t_{ij} はノード i とノード j の間の結合の古さを表し，寿命 T に達すると結合は消滅する．すなわち，結合の生成・消滅を最整合ノードに対して最も近い位置

にいるかどうかによって決定し，持続して最整合ノードの最近傍に存在するノードの結合を最終的に保存する．

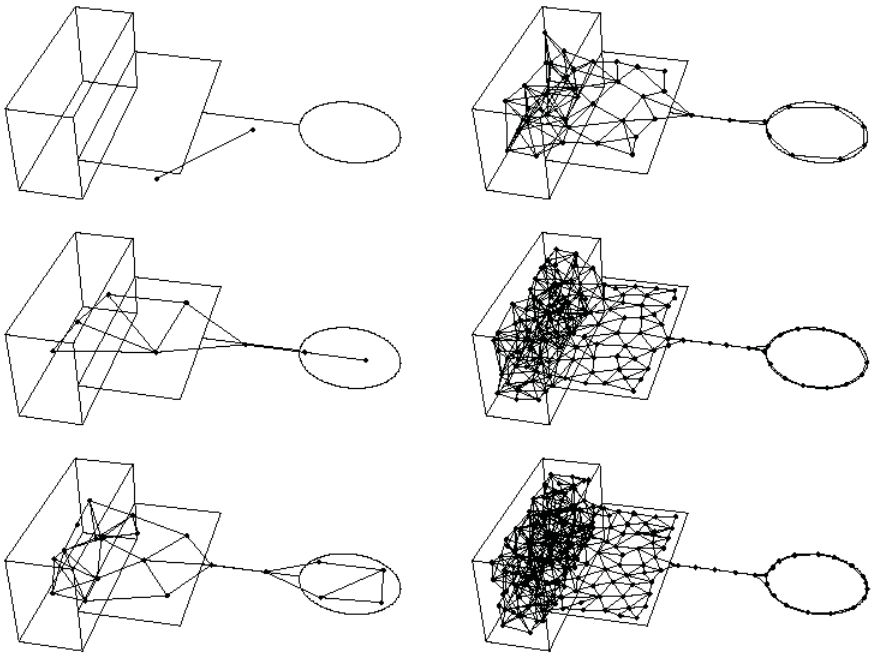

Fig. B.1 Growing neural gas[Fritzke95]

関連の方法として Neural gas と呼ばれるベクトル量子化アルゴリズムがある [Martinetz93]．**Fig.B.1** は，Neural gas のアルゴリズムをノードの逐次生成に適用した Growing neural gas と呼ばれるアルゴリズム [Fritzke95] により 3 次元入力ベクトルを量子化した例である．入力ベクトルは，3 次元空間直方体領域と 2 次元平面上と 1 次元直線上および円弧上に与えられる．ノードは入力空間を覆うように分布し，ノード同士の接続関係はそれぞれの入力ベクトルの形状を反映したものになっていることがわかる．

B.3 ノード 上関数の移動ベクトルによる微分

ノードが u が $\boldsymbol{r}$ 移動したときの関数値を

$$f_u(\boldsymbol{r}) \equiv \frac{1}{m(u)} \sum_{h \in N(u)} \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \|\boldsymbol{r} - \boldsymbol{c}_h^u\|^2 \qquad (\text{B.3.1})$$

と定義する．$f(u)$ を変化させるためのノード u の移動方向は，この $f_u(\boldsymbol{r})$ を $\boldsymbol{r}$ で微分し，$\boldsymbol{r} = \boldsymbol{0}$ を代入することで得られる．ここで，$g_{hu}(\boldsymbol{r})$ を

$$g_{hu}(\boldsymbol{r}) \equiv \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \|\boldsymbol{r} - \boldsymbol{c}_h^u\|^2 \qquad (\text{B.3.2})$$

と定義すると，

$$f_u(\boldsymbol{r}) = \frac{1}{m(u)} \sum_{h \in N(u)} g_{hu}(\boldsymbol{r}) \qquad (\text{B.3.3})$$

と表せる．この $g_{hu}(\boldsymbol{r})$ の $\boldsymbol{r}$ に関する微分は次のように表される．

$$\begin{aligned} \frac{\partial g_{hu}(\boldsymbol{r})}{\partial \boldsymbol{r}} = &\left(\frac{\partial}{\partial \boldsymbol{r}} \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \right) \|\boldsymbol{c}_{hu} - \boldsymbol{r}\|^2 \\ &+ \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 \left(\frac{\partial}{\partial \boldsymbol{r}} \|\boldsymbol{c}_{hu} - \boldsymbol{r}\|^2 \right) \end{aligned} \qquad (\text{B.3.4})$$

$\boldsymbol{r}$ による偏微分の項は以下のように展開できる．

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{r}} \|\boldsymbol{a}_u(\boldsymbol{r}) - \boldsymbol{a}_h\|^2 &= \frac{\partial}{\partial \boldsymbol{r}} \sum_{j=1}^{n} (a_{hj} - a_{uj}(\boldsymbol{r}))^2 \\ &= -2 \sum_{j=1}^{n} (a_{hj} - a_{uj}(\boldsymbol{r})) \frac{\partial}{\partial \boldsymbol{r}} a_{uj}(\boldsymbol{r}) \qquad (\text{B.3.5}) \\ \frac{\partial}{\partial \boldsymbol{r}} \|\boldsymbol{c}_h - \boldsymbol{r}\|^2 &= \frac{\partial}{\partial \boldsymbol{r}} \sum_{j=1}^{n} (c_{hj} - r_j)^2 \\ &= -2(\boldsymbol{c}_h - \boldsymbol{r}) \qquad (\text{B.3.6}) \end{aligned}$$

$\frac{\partial a_{uj}(\boldsymbol{r})}{\partial \boldsymbol{r}}$ は以下の (3.5.12) 式により計算される．

$$\frac{\partial \boldsymbol{a}_u(\boldsymbol{r})}{\partial \boldsymbol{r}} = \frac{\partial t}{\partial \boldsymbol{r}} \frac{\partial s}{\partial t} \left(\frac{d\boldsymbol{a}_1(s)}{ds} \cdot \boldsymbol{n}_c \right) \boldsymbol{n}_c + \frac{\partial t}{\partial \boldsymbol{r}} N \frac{d\boldsymbol{a}_2(t)}{dt} N^T$$