TR-AC-0049

034

高次元アルゴリズムの原理と応用

斎藤 茂　新上 和正　種田 和正　大田原 一成
下川 信祐　平田 和貴*　北川 美宏
倉持 裕　恩田 和幸　小松崎 彰　野口 孝明**

* 現在、富士ゼロックス（株）　** 現在、関西日本電気ソフトウェア（株）

2000.12.13

ATR環境適応通信研究所

目次

第1章 まえがき

『高次元アルゴリズム（Hamiltonian Algorithm)』は最適化問題を解く手法の一つであり，ＡＴＲにおいて考案された．まず，多くの変数を持つハミルトン系の運動が，最適化問題の解法として使用できることが発見，指摘された[1]．次いで，解法としての基本的なアイデアも明らかにされ，いくつかの適用事例で確認の後，『高次元アルゴリズム（Hamiltonian Algorithm)』の名のもとに発表された[2],[3]．

ＡＴＲ環境適応通信研究所第二研究室では，この『高次元アルゴリズム』を様々な問題に適用し，それぞれの課題について研究を進めている．本テクニカルレポートは，『高次元アルゴリズム』の原理を解説すると共に，それらの適用例を最適化手法の観点から紹介している．

本アルゴリズムについては，かねてから
- プログラムの具体的な作り方，
- プログラムそのもの（サンプルも含め）の入手法，
- 良く知られている問題への適用結果，
- 他の最適化法との比較結果，
- アルゴリズムの使用料（権利関係），

などに関する問い合わせをしばしば受けている．

このような問い合わせに答え，その原理と使用法を理解していただくため，「高次元アルゴリズムの原理と応用」と題し，平成 12 年 9 月にセミナーを開催した．本テクニカルレポートはそのセミナーでの配付資料をもとに編集したものであり，上記の問い合わせのいくつかに回答を与えている．なお，現状では，具体的なプログラムを最適化ツールとして提供しているわけではないことをお断りしておく．また，本アルゴリズムには特別な権利が設定されているわけではなく，従って，その使用に対して料金が課されることもない．

『高次元アルゴリズム（Hamiltonian Algorithm)』は，冒頭でも少しふれたように，(1)基本的なアイデアと(2)具体的な方法の二要素から成り立っている．

「基本的なアイデア」とは次のようなものである．

最適解を求めることの難しさは，解領域（解とその近傍を含む領域）が変数の定義域に比較して非常に小さいことにあると考えられる．従って，何らかの方法によって，定義域の中に占める解領域の割合を拡大できるとすれば，解が求め易くなるはずである．その方法とは定義域を変形することであり，新しい変数を問題に付け加えることではないか．言い換えれば，問題の次元数を増やすことではないだろうか．

このような「基本的アイデア」を実現する「具体的方法」としてハミルトン力学系と呼ばれる力学系を採用する．すなわち，最適化問題を力学系の運動に置き換え，運動の持つ性質を利用して解こうとする．

最適化問題におけるコスト関数を力学系のポテンシャルエネルギーに，最適化されるべき変数を位置に対応させる．新しい変数として運動量を導入し，運動エネルギーを考える．ポテンシャルエネルギーと運動エネルギーの和で与えられるハミルトニアンの下で運動を解き，当初の問題の最適解を探索する．このようにすると，ポテンシャルエネルギーが小さい所（位置）で力学系の存在確率が大きくなり，最適解が求め易くなっている．

このような二つの要素が，本アルゴリズムの和名『高次元アルゴリズム』と英名『Hamiltonian Algorithm』の由来となっている．

通常の考えと異なり，敢えて変数を増やして問題を解こうとすることは，本アルゴリズムの特徴と言えるであろう．また，シミュレーテッド・アニーリング（ＳＡ）や遺伝的アルゴリズム（ＧＡ）など，多くの最適化手法が外国で提案されたのに対し，高次元アルゴリズムが我が国において考案されたことにもその意義があると思われる．

本テクニカルレポートの構成は以下の通りである．

第２章は，「高次元アルゴリズムの概要」と題し，アルゴリズムの原理，使用法，特徴，他手法との比較，適用例を述べる．特に，使用法については多くの紙面をさいている．本アルゴリズムを使うときの作業は次の５段階から成る．(1) 最適化問題を設定する，(2) ハミルトン力学系を構成する，(3) 運動方程式を明らかにする，(4) 運動を解くための計算プログラムを作成する，(5) 計算を実行して解を得る．本章ではこれらの各段階における考え方，注意事項を詳細に解説する．

第３章から第１１章は高次元アルゴリズムの適用例を紹介する．適用例は，情報通信ネットワークの設計・制御（第３，４章），情報通信システムに関連する装置，方式の最適化（第５～７章），一般的なシステムの最適設計・制御（第８章），物理，数学の分野（第９，１０章），および膨大なデータの整理法（第１１章）に及んでいる．しかし，各章では，それぞれの領域において得られた新しい知見を述べることよりも，それぞれの問題への高次元アルゴリズムの適用法を具体的に示すことを主眼にまとめられている．従って，まず解こうとする最適化問題を説明し，次いで上記の５段階に従って議論を進めていく．この中で，問題の特徴，取扱い方の特徴が明らかにされる．

第３章では，「通信ネットワークのリソース配分の最適化」が述べられる．トラフィックの経路配分，ノード速度の配分，リンク帯域の配分を変数として，平均遅延時間と最大リソース使用率からなるコスト関数を最小にするような最適化を高次元アルゴリズムを用いて行っている．最適化される変数の数が１万６千以上となる計算例が示される．

第４章の話題は「動的ルーティング政策」であり，複数の並列伝送サーバを使ってパケットを伝送するとき，遅延時間を最小にするパケット振り分け法を明らかにするものである．ここでは，伝送すべきパケットの生起時刻と長さが前もってわかっているという仮定の下で，それぞれのパケットをどのサーバに割り当てるかという問題を解く．このため，割り当てられるサーバを表す離散変数（整数変数）が最適化の対象となっている点に特徴がある．

第５章は「可変指向性アンテナのパラメータ最適化」を紹介する．給電素子の回りの無給電素子に装荷された可変リアクタを制御することで，所望の指向性を得るという課題である．ここでは，所望の方向の利得を最大にするという目的の下で，最適なリアクタンス値を探索する．ある方向の利得を求めるために特別な数値計算が必要である，換言すれば，利得がリアクタンス値の関数として明な形で与えられていない，というのがこの問題の特徴である．

第６章は，静止画像の符号化・複合化に関し，「JPEG 量子化テーブルの最適化」を行っている．標準化された量子化テーブルに比較して，画像品質の劣化を招くことなく，圧縮率を大きくすることを目的としている．離散変数を連続変数として取扱っていること，画像品質と圧縮率という相反する評価尺度をコスト関数に取り入れている点に特徴がある．

第７章は「ニューラルネットワークの学習」を取扱っている．ニューラルネットワークの結合荷重を所望の出力が得られるように高次元アルゴリズムで決定する．コスト関数には一意性はなく，ここではニューラルネットワーク出力と所望の出力の二乗誤差和をコスト関数としている．従って，最適点におけるコスト関数の値が０であることが前もってわかっている．

第８章は「コージェネレーションシステムの最適設計・制御」について述べている．電力のみを入力エネルギーとする冷暖房システムに比較し，都市ガスをも入力エネルギーとするシステムで投資回収年数を最小にすることを目的としている．エネルギーの流れを表す連続変数（実数変数）と冷暖房機器の冷暖の切替を表す離散変数（整数変数）が混在したシステムの最適化を行っている．

第９章は，熱力学的に安定な内部エネルギーを持つ分子構造を探すという「分子構造の決定」を説明する．すなわち，エネルギーをコスト関数として分子内の各原子座標を決定する問題である．解の探索を効率良く行うための工夫がなされており，その方法が詳しく論じられる．

第１０章の「行列の固有値の探索」は，対称行列の固有値と固有ベクトルを求める問題である．コスト関数は一意ではなく，ここでは，変数が固有値と固有ベクトルに一致するとき最小値０になるようにコスト関数を設定している．代数的な解法が決まりきった手順で，従って，決まりきった計算量で厳密解を求めるのに対し，要素が時間的に少しずつ変化していくような行列について，短い探索時

間で近似解を次々と求めていくような場合に有効である．

第１１章の「どんなモノがブレイクしてゆくのか？」は，これまでの例とは異なり，必ずしも具体的に適用した結果を解説するものではない．むしろ，今後高次元アルゴリズムを適用しようとしている例，このような問題にも適用できるという例である．多くの複雑なデータの中に隠れた関係（写像）を探し出す手段としての適用が考えられている．

第１２章はむすびである．本テクニカルレポートの内容を簡単に振り返り，読者へのお願いを述べるとともに，問い合わせ先を列挙する．

なお，各章の執筆者は以下の通りである．

第１章　斎藤　茂，	第２章　新上　和正，	第３章　恩田　和幸，
第４章　種田　和正，	第５章　小松崎　彰，	第６章　平田　和貴(*)，
第７章　倉持　裕，	第８章　北川　美宏，	第９章　大田原　一成，
第１０章　野口　孝明(**)，	第１１章　下川　信祐，	第１２章　斎藤　茂

(*)　現在，富士ゼロックス株式会社
(**)　現在，関西日本電気ソフトウェア株式会社

文献

(1) 新上，佐々田，"ハミルトニアンダイナミクスの功利性"，日本物理学会全国大会予稿集，13p-F2，1993．
(2) 新上和正，"高次元アルゴリズム"，bit，Vol.31，No.7，pp.2-8，July 1999．
(3) 新上和正，"高次元アルゴリズム：最適化問題を解く一つの方法"，日本ファジィ学会誌，Vol.11，No.3，pp.382-395，1999．

第 2 章 高次元アルゴリズムの概要
(HA: Hamiltonian Algorithm)

概要

多くの変数を持つハミルトン系の運動を調べる研究から, 最適化問題を解く 1 つの方法を考案した. この方法は, 問題を記述するのに必要な変数 (意味変数と呼ぶ) に新しい変数を加えた高次元空間で構成された力学系の自律的運動を利用して最適化問題を解こうとするものである. 自律的運動とは, 解に向かう傾向を持つ運動という意味である. 以下ではこの方法の原理を説明する. また, このテクニカルレポートで紹介する 9 つの応用例について高次元アルゴリズムと最適化問題との関係を解説する.

内容

HA の適用例 1 ～ 17 を示す. また, 適用例 1 ～ 9 が HA や最適化問題の内容とどのような関係を持っているかを付録で明示する.

(1) 通信ネットワークのリソース配分の最適化.
(2) 動的ルーチングの政策.
(3) 可変指向性アンテナのパラメータ設計.
(4) JPEG 量子化テーブルの最適化.
(5) ニューラルネットの学習.
(6) コージェネレーションシステムの最適設計・制御.
(7) 分子構造の決定.
(8) 行列の固有値の探索.
(9) どんなモノがブレイクして行くのか?
(10) 光ファイバーの軸合わせ問題: 向かい合うファイバーの一つの出力を最大 [1]

(このレポートで紹介していない他の適用例)

(11) 画像復元: 動きボケにより劣化した画像の復元, 劣化した観測画像の復元 [1]
(12) トラベリングセールスマン問題 [1]
(13) 多要素の制御問題の一つのアクチーブ制御問題 [1]
(14) スケジューリングの問題: s 個の仕事を m 個の機械で分担し, 終了時間を最小 [1]
(15) ルービックキューブの自動解法: 色合わせ [1]
(16) 音声合成 [2]
(17) 装置に組み込む制御アルゴリズムの設計問題 [3]
など.

(文献は, 他の適用例では引用されないと思われるものを付ける)

1. 高次元アルゴリズムの原理と特徴

1.1. 高次元アルゴリズムとは. 高次元アルゴリズムは,[4]-[7]

(I) 基本的アイデア

(II) そのアイデアをどのように現実化させるか

という二つの要素から創られる. (I) はニューラルネットやオートマトンやファジィーのような具体的な道具立てを使わない形式で述べる. しかし, (I) を実体化させるには具体的な道具が必要となる. この道具として私が日頃慣れ親んでいるハミルトン力学系と呼ばれる力学系[8]-[11]を採用する – 他の舞台でもこれを実行することが可能である –.

基本的アイデアを図 1 に示す. 実際には多次元空間であるが, 簡単のためにアイデアを 2, 3 次元空間で描いている. 最適化問題は (図で左端に示された) 意味空間で設定されている. 普通, 直接的に意味を持つ多くの変数で表現されので, この空間を意味空間と呼ぶことにする. 一般的に, 最適化問題を解くのに相当な困難がある. 意味空間で解に相当する (図で • に相当する) 領域は微小で, 誰もがこの領域を見つけるのは難しいと言える. この困難さを, 次のようなアイデアで克服しようとする.

仮りに, 高次元空間では意味空間の解でない領域は小さく, 逆に解に相当する領域はより大きく拡大されるとする (図 1 で解である • は球で囲まれている領域ぐらい拡大される). 高次元空間での運動を想定すると, その運動は最初球から外れた適当な点から出発すると, 時間が経過するとやがて球の中に入いる. これにより, 意味空間では難しい問題も高次元空間で運動を追跡して行けば解を容易く探すことが期待される. このことを意味空間で言い替えれば, 運動は解でない領域を素早く通り過ぎ, 解に相当する領域に入り易くなっていることに相当する. 高次元空間で構成された, このような性質を持つ運動を自律的運動と呼んでいる. HA は, 高次元空間を設定してその空間上で自律的運動を利用して問題を解く考えに基づいている.

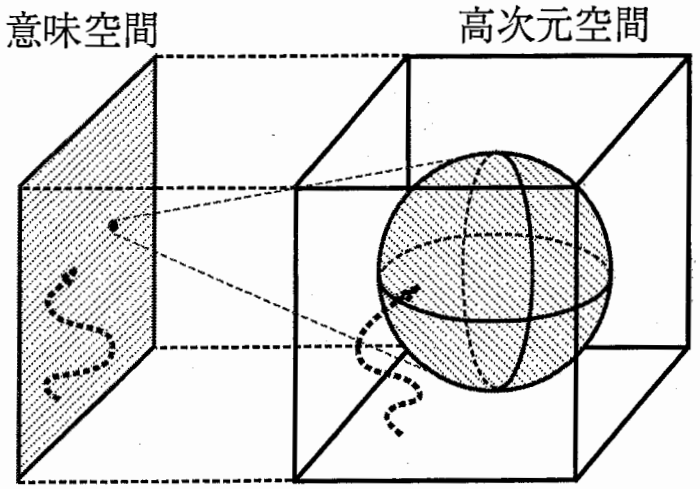


図 1. 基本的アイデア.

以下では, ハミルトン系を舞台に話を進める. HA は,

◎ 力学系 (運動) を利用 : 位置 (意味空間の変数に相当する) と速度を持つことが特徴.

◎ 巧く解く仕組み: 運動に “混合性” を持たせる. 混合性を持たない運動もあれば持つ運動もある. この混合性によって, 高いコストを持つ領域は素早くスキップし, 低いコスト値を持つ領域へ集中する運動となる (6 節で説明).[8]

1.2. 高次元アルゴリズムの基本的原理[6]. 最適化問題は, コスト関数の最小値を求める. HA は, コスト関数をポテンシャル関数と見なして力学系を構成し, 以下の 2 つの操作を利用して最適化問題を解く. (何でもポテンシャル関数となる訳ではないが, 多くの場合見なせる)

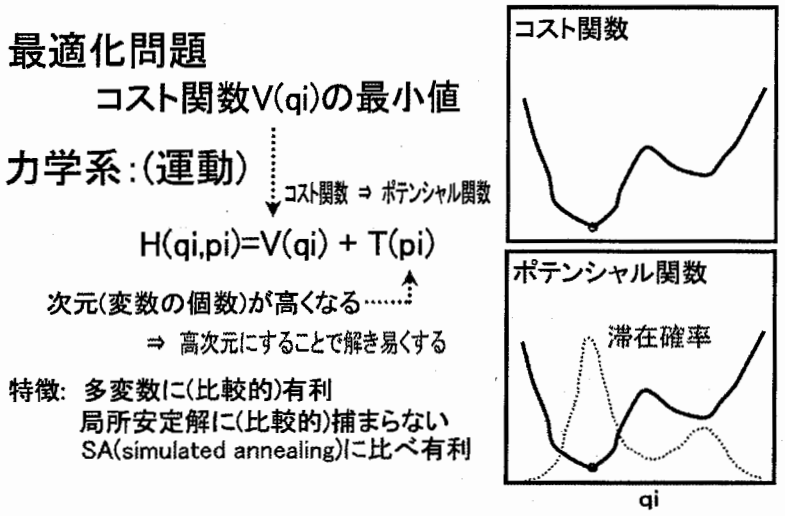


図 2. 高次元アルゴリズムの基本的原理.

(1) 自律的運動を利用する. その力学系の運動に付与された (解のある領域の方向に動く傾向をもつ) 自律的運動を利用して解を求める. 自律的運動の中身は, コスト関数値が低ければ低いほど運動の滞在する時間 (確率) が大きいことを意味する. これは運動の性質であって, コスト関数を直接下げるという直感的に理解し易い操作を持つ遺伝子アルゴリズム (以下, GA と略記する) や焼き鈍し法 (以下, SA と略記する) と違ったやり方である.

(2) エネルギーの減少. (1) で, 運動する領域を狭める効果を持つ. (後で説明するが) このことを具体的に実行する方法は, 式(14)のダンピング定数 $\kappa(t)$ と式(31)での $\xi(t)$ を導入することである. 運動エネルギーを減少させることを通じて, コスト関数を下げる. この操作は, 直接的にコスト関数を下げる GA や SA の操作と同様に, 直観的に判り易い. (これは SA の温度を下げることに相当する)

“コスト関数値が低ければ低いほど運動の滞在する時間 (確率) が大きい” には, 次のような機構が働いている.

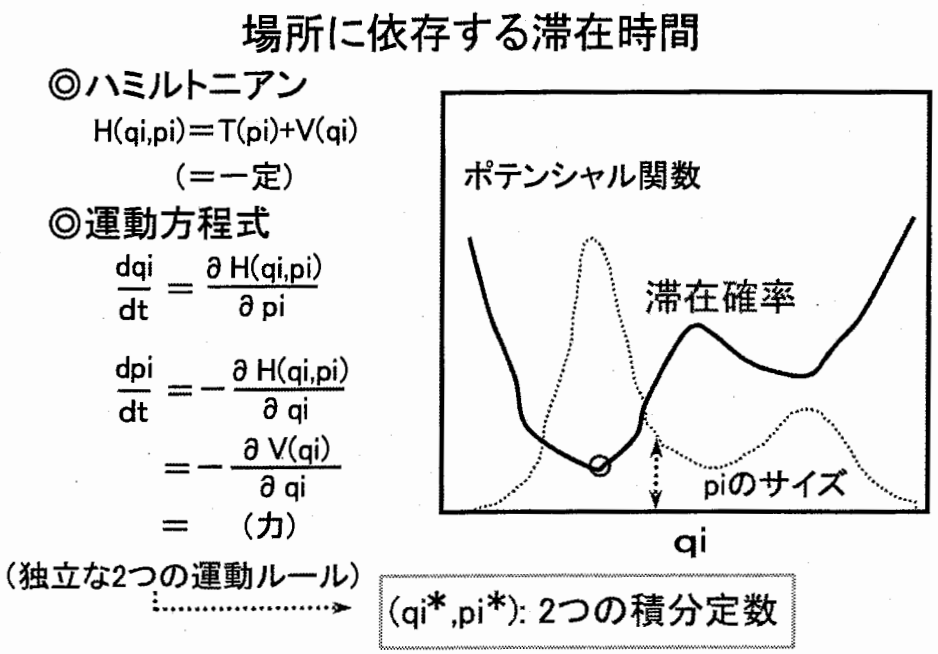


図 3. 場所に依存する滞在時間.

ハミルトン系の特徴は, 位置 q_i とそれに共役な (新しい変数に対応する) 運動量 p_i の二つの量で運動が定まることである. そして, その運動が一つのハミルトン関数 $H(q_i, p_i)$ から $-H(q_i, p_i)$ は任意に与えられるが, 全微分可能である –, 図 3 のように決まる. ここには q_i と p_i に対して, 二つの独立な運動ルールがあり, q_i が一つ決められても異なる p_i の値を取り得る. このことが q_i に依存した滞在時間を生じる. この q_i に依存した滞在時間を, 「コスト関数値を低くする領域で滞在時間 (確率) が長くなる (大きくなる)」 ように構成することが可能である (計算を後で行なう).

より正確には, q_i だけからなる運動方程式 – 例えば, 散逸系と呼ばれる力学系 – の場合には, q_i に依存して滞在時間が異なるが, 「コスト関数値を低くする領域で滞在時間 (確率) が長くなる (大きくなる)」 ような性質を持つ運動を一般的に構成することは困難である.

1.3. その特徴. HA は以下の特徴を持っている:

(1) 局所的な安定解に捕まり難い,

(2) パラメータの注意深い制御を必要としない,

(3) (この方法が使えるように) 問題を変形する必要がない,

(4) ベクトル・パラレル計算処理に向いている,

(5) フラットなコスト関数を持つ場合には, 焼き鈍し法と比べ有利である

(6) 多変数の最適化問題に向く

などのメリットと

(7) (後で説明するが) 力の計算の反復回数が多くなる場合には, 計算時間を短くするために, 反復処理回数を少なくするなどの工夫 (適用例 4) を行う必要が生じる

などのデメリットを持っている (詳細は 8 節を参照).

1.4. アルゴリズムの作業の流れ図. (1)→(2)→(3)→(4)→(5) と進む. 以下では, この流れ図 (図 4) を理解できることを目標に説明する.

(1) 最適化問題の設定 (2 節を参照)

(a) コスト関数と最適化変数

($a1$) 具体的な関数形がある場合.

一意的.

任意性.

($a2$) 具体的な関数形がない場合.

($a3$) コスト関数が一つか複数か.

(b) 拘束条件の扱い

($b1$) プログラムで処理: 境界内での運動, エネルギー減衰, 保存則など (5.1 節参照)).

($b2$) コスト関数へ: $L(q_i)$.

($b21$) 整数変数 → 実数変数化を行う (5.2 節参照).

($b3$) 拘束条件を満たす運動を構成する (5.3 節参照).

(2) 力学形の構成: ($b1$) ~ ($b3$) に対応して

($b1$)→ $H = T(p_i) + V(q_i)$.

($b2$)→ $H = T(p_i) + V(q_i) + L(q_i)$.

($b3$)→ $H = S(q_i, p_i) + V(q_i)$.

混合性の工夫 (4 節参照)

($c1$) $T = (1/2)\sum_i p_i^2$ (通常, これを採用. 運動が混合性を持つという目的が達成されない場合には, 以下の ($c2$)($c3$)($c4$) を).

($c2$) $T = (1/2)\sum_{ij} p_i A_{ij} p_j$, ($A_{ij}$: 正定値対称行列, 4.3 節参照).

($c3$) $T = (1/2)(\sum_{ij} p_i A_{ij} p_j)^\gamma$ など (4.3 節参照).

($c4$) $S =$ (複雑な関数形) (5.3 節を参照).

(3) 運動方程式

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \qquad \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}.$$

(4) 計算プログラム: 運動方程式の積分

($d1$) Runge-Kutta 法 → 力の計算が多い.

($d2$) Verlet 法 → 力の計算が少ない (可能な限り, こちらを利用する. 10 節を参照)

など

プログラムのチェック

バグ ← 「時間を陽に含まない H で, $H(q_i(t), p_i(t)) =$ 一定」を確認 (3.2 節参照).

運動に混合性があるかどうかの確認 (4.2 節参照) ← $(q_i(t), p_i(t))$ や $(q_i(t), q_{i+1}(t))$ など時間変化をプロット.

- 複雑になっていれば混合性あり.
- 規則性があれば混合性なし → ($c2$)($c3$) に戻る.

(5) 解を得る (7 節を参照)

コスト関数の最小値が既知の場合

($e1$) 最小値に到達したとき.

コスト関数の最小値が判らない場合: 全ての安定解を調べる以外, 真の解を得るという保証はない.

($e2$) 運動で通過したコスト関数の最小値を記録. 運動を止めないで続行.

($e3$) エネルギーを減少させ, (疑似) 安定解を得る.

動かせるパラメータ: 特に (f_1)(f_5)(f_7) が重要? (11 節を参照)

(f_1) 初期値: 運動方程式を解くための出発とする (q_i, p_i) $(i = 1, 2, .., N)$ の値.

→ これは同時に初期 $H(q_i, p_i)$ の値も決めている.

(f_2) 式(37)などの γ の値.

(f_3) 式(24)の正定値対称行列 A_{ij} の設定.

(f_4) 拘束条件の処理の仕方： コスト関数に導入した場合.

(f_5) 式(14)のダンピング定数 (κ) と式(31)での $\xi(t)$ の大きさ (5.1 節参照).

(f_6) コスト関数の設定の仕方: 任意性がある場合.

(f_7) 拘束条件の制御: 式(35)の $\beta(t)$(5.2 節参照)

(f_8) 式(57)の ζ: $\zeta V(q_i)$ を新たなコスト関数とする.

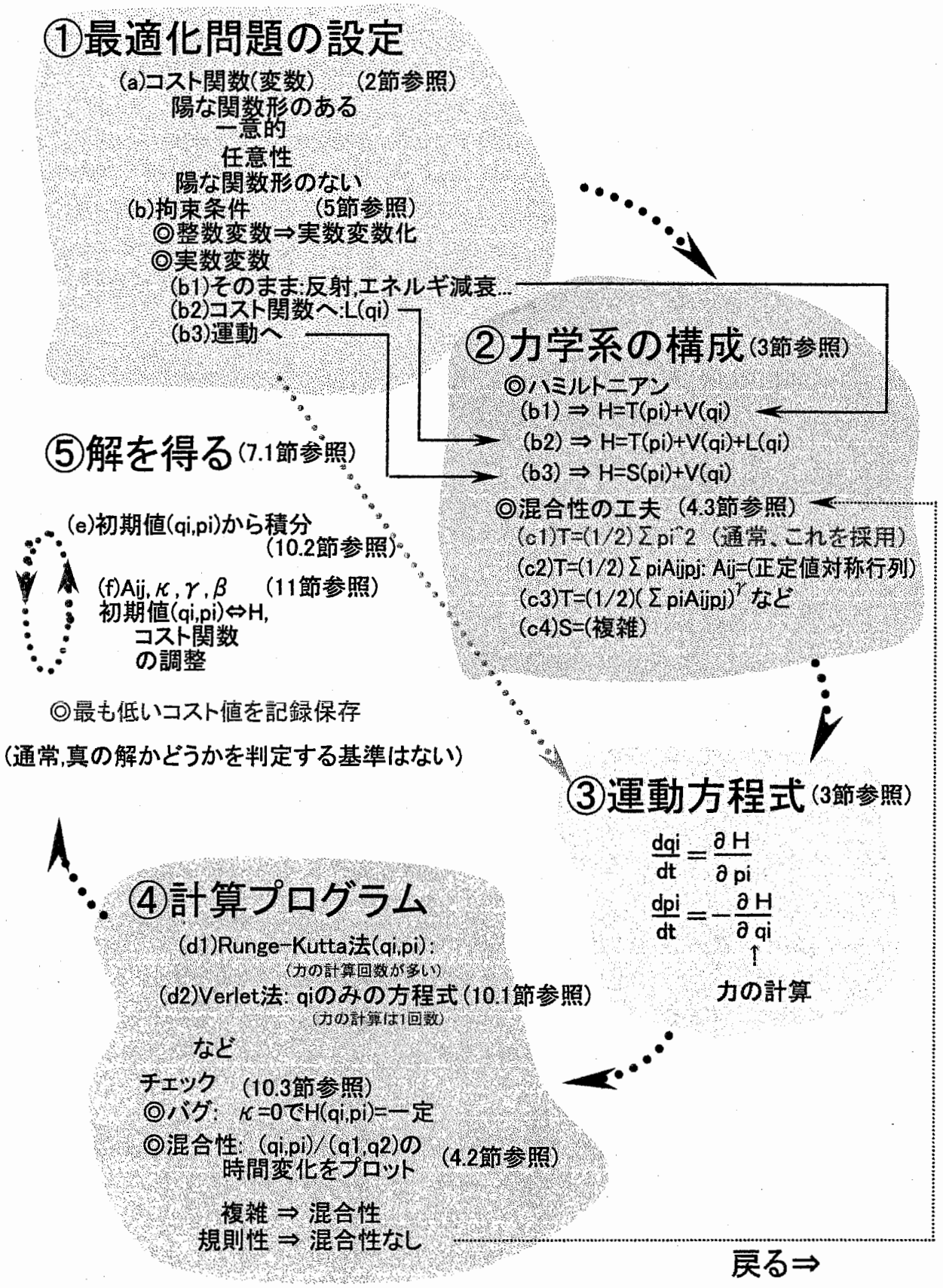


図4. アルゴリズムの作業流れ図

2. 最適化問題の設定

2.1. 最適化問題とは. 最適化問題は, "合理性" や "効率" を重んじる工学や経済学などの広い分野で随所に出会う問題である. 私が今実際に取り組んでいる問題を例に取りあげる. 物質工学では, ある物性 (例えば, 硬さ) を最大とする物質材料を創りたい. 通信工学では, インターネットで, ある人から他の人に送信するデータパケットの到着時間を最小とする通信経路を選びたい. 設計工学では, エネルギー効率を最大にするように建物を決めたい. 設計機械の制御効率を最大にしたい. 等々. また, 経済学では, 例えば, 完全競争の下で "見えざる手" によって全体にとって最善の状態 – 最大多数の最大幸福 – が追及されてきた. これらの全て「…の制約条件の下で…を最大 (最小) にしたい」という要求が最適化の原理である. 最適化の原理に基づいて設定される問題が最適化問題である.

2.2. 三つの要素. 最適化問題とは,[12]–[14] 多くの候補の中からある評価値が最適であるような解を求める問題をいう. 通常, 三つの要素から設定される.

[1] 最適化される変数:

$$x_1, x_2, \cdots, x_N \quad (N : \text{変数の個数})$$

[2] 拘束条件(変数に対する): 例えば,

$$x_1 : \text{実数変数(連続な値を取る)},$$
$$x_2 : \text{整数変数(整数な値を取る)},$$
$$0 \leq x_1, x_2 \leq 1,$$
$$x_1^2 + x_2^2 = 1,$$
$$x_3 \leq x_4 x_5,$$

など.

◎ 拘束条件は, 最適化変数が必ず満たす条件である.

◎ 変数が整数か実数かが重要である. HA では常に整数変数は実数変数化を行う (適用例 2,4,6). 実数化はコスト関数に新たな拘束条件をコスト関数で表現したものを加えることで実行する (5 節を参照). 適用例 2 や 6 がこの問題に属する.

[3] コスト関数(目的関数, 評価関数)

多くの解の候補は一組の最適化変数で表され, 最適化変数の評価を定める関数を, 普通評価関数と言う. その他にもコスト関数, 目的関数などと呼んでいる. 最適化問題の解は, コスト関数の最小値, 評価関数の最大値が対応する. HA は, 関数の最小値を探す. 評価関数にマイナス (負) 記号を付ければ, 評価関数の最小値が解になる. "HA は, マイナス (負) 記号のついた評価関数の最小値を探す' といった表現を一々するのは面倒でなので, この解説全体では, 実際に問題を解くときには, コスト関数という呼び方を統一して使用する.

2.3. コスト関数の例と分類. (コスト関数への注意を含めて)

[A] 具体的な関数が与えられている場合

$a)$ $V(x_i) = x_1 + x_2 + x_3$,

$b)$ $V(x_i) = \mathrm{Max}\{x_1, x_2, x_3\}$,

$c_1)$ $V(f) = \sum_{i=1,..,M} \Big(a_i - f(b_i)\Big)^2$,

$c_2)$ $V(f) = \mathrm{Max}_{i=1,..,M}\{|a_i - f(b_i)|\}$,

$d)$ $V(x_i) = |x_i|$ (微分不可能な所をもつような場合にもHAは使える),

などがある.

$a)$ と $b)$ は, 判り易い例である. 一方, $c_1)$ と $c_2)$ は f という関数自身が変数となっている. これは, 既知である幾つかの事実 (a_i, b_i) のペアから, 事実の背後に潜む相関関係 – $a_i = f(b_i)$ $(i = 1, ..., M)$ となる – を求める問題 (適用例 9,5) で現れる. この時のコスト関数の最小値は, $V = 0$ である.

また, このように具体的な関数が与えられている以外に,

[B] 変数$x_1, x_2, \cdots, x_N$からコスト関数$V(x_i)$を計算するアルゴリズムが与えられている場合

(特に適用例 3.) この場合には, コスト関数が具体的に与えられた場合に比べ, ずっと複雑な関数関係を持つコスト関数を表現することが可能である.

[C]コスト関数が上記の例のように, 単一の場合の他に, 複数ある場合

特に, 複数ある場合 (適用例 1,4) には, 複数のコスト関数をどう扱うかが新たな問題として生じる. この場合には, ある基準となる (複数の) コスト関数値があれば (図 5), その値より全てのコスト関数値を低くなる変数が求めることが可能なら, その変数を解と見なすことができる – パレート最適化 – (適用例 4).

一方, そのような基準点が無ければ, もっと違うことを考えなければない (適用例 1).

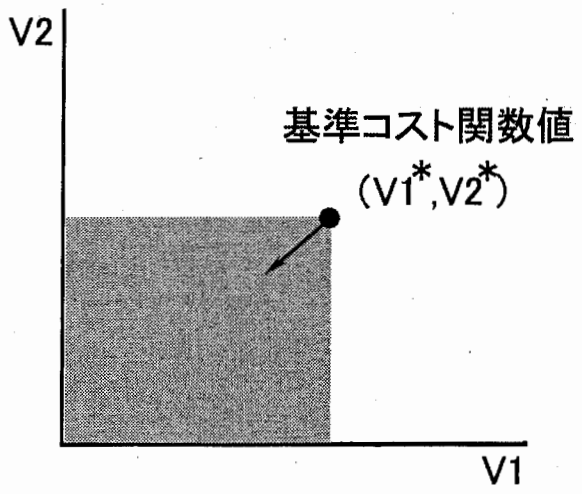


図 5. 複数のコスト関数の一例. 2 つのコスト関数 V_1, V_2 がある場合には, 既知の基準点 (V_1^* と V_2^*) に比べ V_1 と V_2 両方がより小さくなる薄黒い領域へ.

[D]コスト関数が一意的に定まらない場合

コスト関数の最小値以外はある程度自由に設定できる (適用例 5,8,9). [A] の c_1) と c_2) がこの例である. 何故なら, $i = 1, 2, \cdots, M$ の全てに対して $a_i = f(b_i)$ となる関数 f を求めることが問題であるので, c_1) と c_2) のコスト関数はその最小値 $V(f) = 0$ を与える f 以外の関数形には関心が無いからである. 関心のない所では, ある程度自由にコスト関数の形を変形できる.

最適化問題を解くとは,

(1) コスト関数$V(x_i)$の最小値,
(2) その時のx_i $(i = 1, .., N)$の値,

を求めることを言う. 但し, 上記のコスト関数の例で判るように最小値が既知な場合がある. この時には, (2) のみが問題となる.

更に, コスト関数の最小値が既知の場合 (適用例 5,8,9) には, これを解の判定に使えるが, 反対にその最小値が分からない場合には, 一般に全ての解の候補に当たる以外に真の解であるかどうかを決めるきちんとした判定基準はない.

3. ハミルトン力学系の主な性質

必要となるハミルトン系の基本的な性質を簡単に述べる.[15],[16]

3.1. ハミルトン関数 (H) と運動方程式. "運動は, 位置 q_i とそれに共役な運動量 p_i $(i = 1, ..., N)$ で定まる" というのが大きな出発点である. 散逸系と呼ばれる q_i のみで運動が定まる場合もある. 運動方程式は, ハミルトン関数 (ハミルトニアンとも呼ぶ)

$$H(q_i, p_i) = T(p_i) + V(q_i) \tag{1}$$

から導かれる. 一般的には, このように q_i と p_i の関数は独立する理由は全く無く, $q_i p_i$ のように混ざった項があっても構わない. また, $H(q_i, p_i)$ は全微分可能であることが条件である.

運動方程式は,

$$\frac{dq_i}{dt} = \frac{\partial H(q_i, p_i)}{\partial p_i}, \tag{2}$$

$$\frac{dp_i}{dt} = -\frac{\partial H(q_i, p_i)}{\partial q_i} \quad (\text{通常これを "力" と呼び, } f_i \text{で表記する}) \tag{3}$$

と導かれる.

特に,

$$T(p_i) = \frac{1}{2}\sum_i p_i^2 \tag{4}$$

の時には,

$$\frac{dq_i}{dt} = \frac{\partial H(q_i, p_i)}{\partial p_i} = p_i, \tag{5}$$

$$\frac{dp_i}{dt} = -\frac{\partial H(q_i, p_i)}{\partial q_i} = f_i \tag{6}$$

を得る. これから p_i を消去すれば,

$$\frac{d^2 q_i}{dt^2} = f_i. \tag{7}$$

"位置の加速度 = 力" というニュートンの運動方程式が得られる.

3.2. H が陽に時間依存性を持たない場合の運動. H が陽に時間依存性を含まなければ, 式(2)と(3)を解いて得られる各時間 t での $q_i(t)$ と $p_i(t)$ を使うと, ハミルトンニアン $H(q_i(t), p_i(t))$ は時間的に一定値を取る:

$$H(q_i(t), p_i(t)) = \text{一定}. \tag{8}$$

この性質を使えば, 運動方程式を作ったときに, H を評価すればプログラムのバグが発見することが可能となる. 一定ならばバグ無し, 変化するならばバグあり.

[証明]

$$\frac{dH}{dt} = \frac{\partial H(q_i, p_i)}{\partial t} + \sum_i \left\{ \frac{\partial H(q_i, p_i)}{\partial p_i}\frac{dp_i}{dt} + \frac{\partial H(q_i, p_i)}{\partial q_i}\frac{dq_i}{dt} \right\}, \tag{9}$$

$$= 0 + \sum_i \left\{ \frac{dq_i}{dt}\frac{dp_i}{dt} - \frac{dp_i}{dt}\frac{dq_i}{dt} \right\}, \tag{10}$$

$$= 0. \tag{11}$$

式(9)と(10)を導くとき, "H が陽に時間依存性を持たない" という条件である

$$\frac{\partial H(q_i,p_i)}{\partial t}=0 \tag{12}$$

と式(2)と(3)を使った.

従って, 式(11)より, H が陽に時間依存性を持たなければ Hは時間によらず一定 となる.

3.3. Hが陽に時間依存性を持つ場合の運動の一つの例. 7 頁の 1.2 節で述べた (2) エネルギーの減衰を実行する一つの方法 (他の方法は 5.1 節参照) に相当する.

「古典力学で, ダンピング項 (摩擦項) のある場合のハミルトニアンは?」

ダンピング項のある系の運動方程式は

$$\frac{d^2q_i}{dt^2}=f_i-\kappa(t)v_i(t). \tag{13}$$

である. ここで, $v_i(t)=dq_i/dt$ は速度, f_i は力, $\kappa(t)$ は (通常時間に依存しない) ダンピングの割合を左右する定数.

この運動は, 速度が増大すると進む方向とは逆方向の力が ($\kappa>0$ の場合) 生じる場合で, 例えば, 石が大気中を落下する時の運動に対応する.

この時のハミルトニアンは

$$H(q_i,p_i)=\frac{1}{2}\sum_i\left[p_i-\int^t\kappa(t')v_i(t')dt'\right]^2+V(q_i) \tag{14}$$

と設定する. これは陽に時間依存性を持つ.

式(2)と(3)の運動方程式に代入すれば,

$$\frac{dq_i}{dt}=p_i-\int^t\kappa(t')v_i(t')dt', \tag{15}$$

$$\frac{dp_i}{dt}=f_i. \tag{16}$$

式(15)をもう一回時間で微分すれば,

$$\frac{d^2q_i}{dt^2}=\frac{dp_i}{dt}-\kappa(t)v_i(t). \tag{17}$$

これに式(16)を代入して dp_i/dt を消去すれば, 式(13)を得る.

方程式(17)をしばしば HA では利用する (適用例 7,9).

エネルギー減少率. H が陽に時間依存性を持てば, H は一定でなくなる. 単位時間当たりのエネルギー減少率は,

$$\frac{dH}{dt}=\frac{\partial H}{\partial t} \tag{18}$$

より計算される. これに式(14)を代入すれば

$$=-\sum_i\kappa(t)v_i(t)\left[p_i-\int^t\kappa(t')v_i(t')dt'\right], \tag{19}$$

$$=-\sum_i\kappa(t)v_i(t)\frac{dq_i}{dt}, \tag{20}$$

$$=-\sum_i\kappa(t)\left[\frac{dq_i}{dt}\right]^2. \tag{21}$$

ここで, 式(20)から式(21)を得るために, $v_i(t)=dq_i/dt$ の関係を使った. この式からエネルギーは $\kappa(t)>0$ で減少し, $\kappa(t)<0$ で増大する.

4. 運動の混合性と混合性の導入の仕方

4.1. 混合性とは.

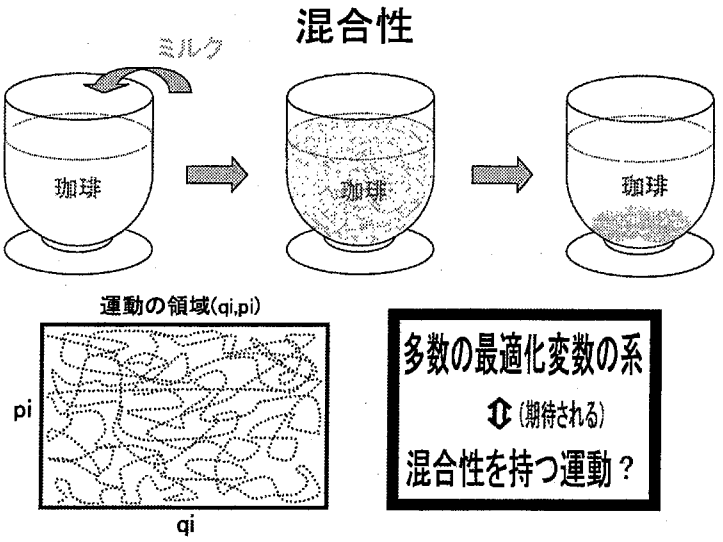


図 6. 混合性の説明.

混合性の説明で,[18] 良く使われる例は, コーヒーに入れたミルクである. 初期には コーヒーとミルクはあまり混ざっていなくても, 十分時間が経つと, 混ざり合う. 混ざり合うという内容は, 液体のどの部分 (その容積は任意である) をすくいあげても, その中のコーヒーとミルクの割合は一定 (もとのコーヒーとミルクの割合に等しい) である. この例で, 言いたいことは, 運動できる空間 (コーヒーの入っている空間に対応する) に多数の (ミルクの分子数だけの) 初期値を持つ運動を解いて行くと – この時, ミルクの分子の相互作用を考えないで, つまり, 異なる初期値を持つ運動の間の相互作用を考えないで - その運動の点はバラバラに一様に分布するということである. これをイメージ的に描くと, 図 6 の左下の図になる. ある所から出発した運動は, 時間が経つにつれて動ける空間を隈無く一様に経巡る. (ミルクの例では, 多数のミルク分子を考えながら, 左下のは一つの点 (ミルク分子) だけを想定したので, 論理的な矛盾があるのではあるが)[20],[16],[19]

通常力学系を扱っている (物理学を研究している) 人達は, "(対象とする) 自然系が混合性を持っている" と期待している. しかし, この期待は, 最適化問題を扱う限り, 必ずしも成り立たない. 例えば, 2500 万もの変数を含む系の運動 (適用例 9) を調べたことがあるが, 混合性を持っていなかった.

4.2. 混合性の確認. 混合性があるかどうかを厳密に示したり, 正確に確認することは通常不可能である. このため, 不正確であるが簡単に行う方法を述べる. これは, 幾つかの $(q_i(t), p_i(t))$ $(i = 1, 2, ...)$, 又は $(q_1(t), q_2(t))$ などをプロットすることで経験的に得られている.

(a) 混合性の無い運動

混合性のない場合にはリサージュ図形を描いたり, 何らかの規則性を持つ. この場合には, 運動している領域の内部で行けない場所 (例えば, 原点の近傍) が見える.

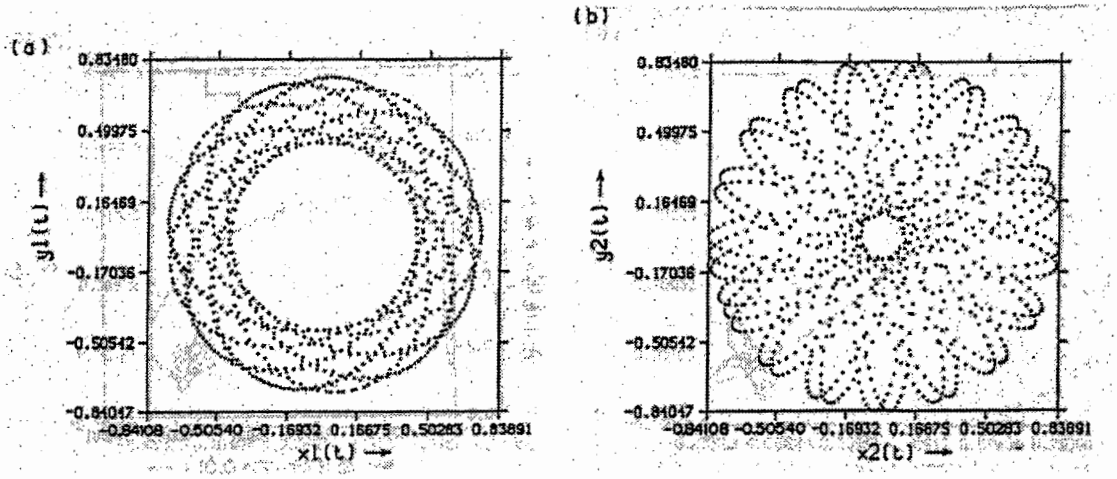


図 7. 混合性の無い運動.

(b) 混合性がある(と思える)運動

混合性がある運動では, (a) の見られるような規則性は無くなる. この場合には, 運動できる領域では色々な場所に行くように見える. また, 軌道にカスプ (鋤型) が見える.

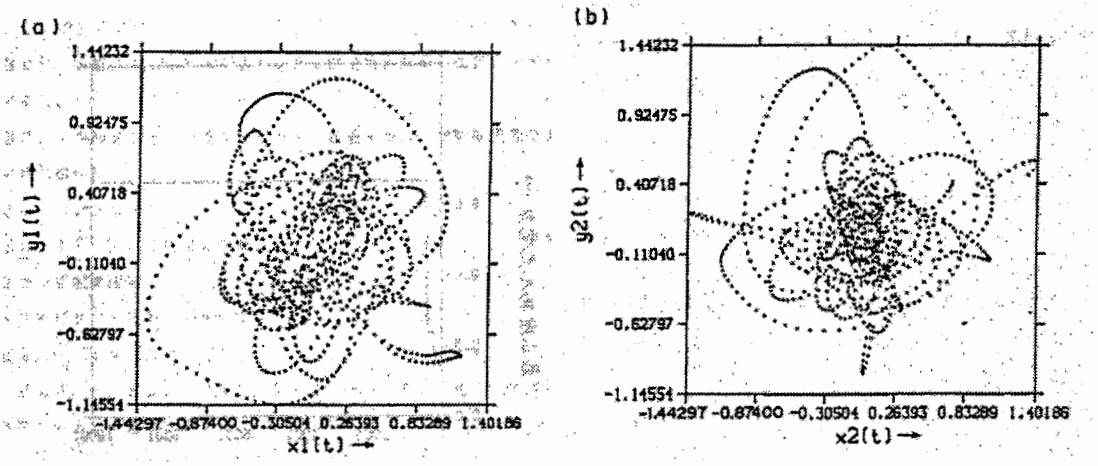


図 8. 混合性のある運動.

このように, 混合性のない場合には規則性が見えるが, 混合性がある場合は, 複雑な運動となる. このような運動をプロットすることによって混合性の有無を (経験上的に) 確認する. 混合性が無い運動では, コスト関数はある程度以下には減少しないが, 混合性を導入 (その仕方は次節で述べる) すると減少して行く.

4.3. **混合性の導入の方法.** 混合性の導入の仕方は, 大まかに分けると, 式(1)で

[I] 運動エネルギーの項 $T(p_i)$ を変形するか,

[II] コスト関数 $V(q_i)$ を変形するか,

[III] 両方を変形するか

である. 多くの場合には, コスト関数は一意的に決まっている場合が多いので, (I) を使う.

★この頁の $(c1)(c2)(c3)$ は, "アルゴリズムの作業流れ図 (図 4)" に現れるものに対応する★

$[I_1]$ 通常, $T(p_i)$ は $(c1)$ を採用する.

$[I_2]$ それで混合性が得られなければ, $(c2)$ を試みる. A_{ij} は (その全ての固有値が正となる) 正定値対称行列である.

正定値対称行列を得る一つの方法:

正定値対称行列 A を得る方法は, いろいろあるが一つの方法を紹介する.

$$C = I + \lambda B. \tag{22}$$

I は単位行列, B はその行列要素をランダム (必ずしもランダムである必要はない) に与えた対称行列, λ は適当な定数 (負でも正でも構わない) とする. この行列 C から, シュミットの直行化によって N 個の直行するベクトルを計算する:

$$|i>:=(c_{1i}, c_{2i}, .., c_{Ni}) \quad (i = 1, \cdots, N). \tag{23}$$

これから, $|i><i|$ を作ると行列になり, $<i|i>$ は内積を与える. 正定値対称行列は

$$A = \sum_i |i> \epsilon_i <i|, \quad (\text{但し}, \epsilon_i > 0) \tag{24}$$

から得られる. ϵ_i は外から適当に与える: 例えば (よく利用する), $0.6 < \epsilon_i \leq 1.4$ の範囲で一様に値を持つ

$$\epsilon_i = 0.6 + (1.4 - 0.6)\frac{i}{N}.$$

行列 A の固有値は, ϵ_i で, その固有ベクトルは $|i>$ となる. その逆行列 A^{-1} は,

$$A^{-1} = \sum_i |i> \frac{1}{\epsilon_i} <i|. \tag{25}$$

となる (その固有値は $1/\epsilon_i$, 固有ベクトルは $|i>$). A^{-1} は, H を計算する時に必要となる (適用例 7).

$[I_3]$ それで得られなければ, $(c3)$ を用いる.

A_{ij} は正定値であるので, $(...)^\gamma$ の括弧の中の項は任意の p_i に対して正かゼロである. γ の運動へ効果は, γ が小さくなると, γ の大きな運動に比べ, ゆったりした運動となる (10.3 節の式(52)).

$[I_1] \sim [I_3]$ で混合性が得られるとは限らないが, 経験では (十数個の最適化問題では) この範囲で達成された. それらの方法で得られなければ, 更に工夫する必要がある. 例えば, [I] の範囲で

$$H(q_i, p_i) = \frac{1}{2}\sum_i p_i^{\gamma/2} A_{ij} p_j^{\gamma/2} + V(q_i) \tag{26}$$

など. 経験では, $i = j$ で $A_{ij} = 0$ となる場合の方が, 混合性は入り易い.

また, [II] は, コスト関数に任意性がある場合には, コスト関数を変形することも出来る. 例えば, (適用例 9 で言えば)

$$V(f) = c_1 \mathrm{Max}_i \Big\{ |a_i - f(b_i)| \Big\} + c_2 \sum_i \Big(a_i - f(b_i) \Big)^2 \tag{27}$$

など. c_1 と c_2 は制御パラメータ.

補足: 上記の式(22)の行列 C の直行化は, その固有ベクトルを使っても構わない. そのとき, $\lambda \neq 0$ と $\lambda = 0$ の固有ベクトルは λ の関数として見たときに一般的に不連続となる. $\lambda = 0$ の場合には, 全ての固有値は等しく (=1) なり, 固有ベクトルに任意性が生じるため (任意のそれらの線形和 – 規格化条件と相互に直行化した – も再び固有ベクトルとなる).

前出の事柄をまとめると, 図 9 が得られる.

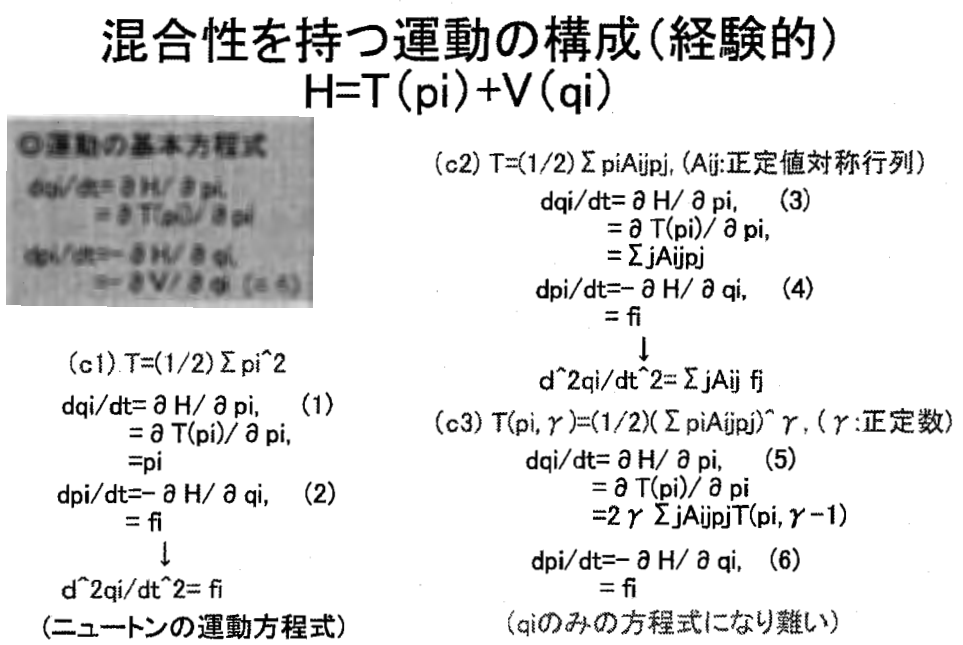


図 9. 混合性を持つ運動の構成.

5. 拘束条件の取り扱い

拘束条件を扱う方法に三種類ある.

5.1. プログラムで細工する場合. 運動に反映させないで, 手軽に計算プログラムで処理する場合.

(a)反射 : 変数を常に境界内に留まらせる

例えば, $0 \leq q_i \leq 1$ の拘束条件では, 運動で境界を外れたときには (図 10), プログラムで q_i が負 ($q_i \leq 0$) であることを認識し, (q_i, p_i) を $(-q_i, -p_i)$ で計算を続ける. つまり, 位置 q_i を領域内に入れて, 同時に速度 p_i を反対にする.

同様に, $1 < q_i$ の時には, $(q_i, p_i) \rightarrow (2 - q_i, -p_i)$ とする.

その時に, この操作を行って

$$H(q_i, p_i) \neq H(-q_i, -p_i) \tag{28}$$

のとき,

$$H(q_i, p_i) = H(-q_i, -\alpha p_i) \tag{29}$$

が成立するように α を決め, αp_i を新たな p_i を見なす必要がある (適用例 9). この場合は, 特に正定値対称行列がゼロでない非対角行列要素を持つとき起こる: $A_{ij} \neq 0$ $(i \neq j)$.

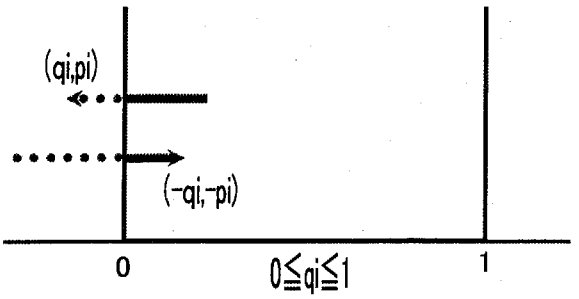


図 10. プログラムで細工する例: 反射.

(b)総和を一定に保つ

変数 q_i は

$$q_1 + q_2 + \cdots + q_N = 1 \tag{30}$$

を満たす必要がある場合, 運動によりこの条件は一般に満たされなくなる. そのときに強制的にスケール変換を行って満足させる場合もある (適用例 1,6).

(c)運動エネルギーを減衰させる

拘束条件と違うが, プログラム上で簡単に処理する例を. これは 1.2 節の「エネルギーの減少」を簡便に達成する一つの方法でもある. 運動方程式を積分する際に, 速度を

$$v_i(t) \rightarrow \xi(t) v_i(t) \tag{31}$$

と $\xi(t)$ 倍するする. これにより速度を減少させたり, 増大させたりする (適用例 2,6,8). これは運動エネルギーを増大 $(1 < \xi)$ させたり, 減少させたり $(\xi < 1)$ することと同じである. 例えば, $\xi = 0.99$ ならば, (31)の操作を 1 回実行すると速度は, 0.99 $(= (1 - 0.01))$ 倍になり, 2 回実行す

ると 0.99^2 $(=(1-0.01)^2)$ 倍になり, $\cdots$. 大体, (31)の操作を k 回実行すると, 速度は $e^{-0.01k}=e^{-(1-\xi)k}$ と減少する.

5.2. **拘束条件をコスト関数で表す** $L(q_i)$ は拘束条件を q_i が破る時増大する関数とする (図 11). このコスト関数をもともとのコスト関数 $V(q_i)$ に加えたものを新たなコスト関数と見なす: $V(q_i)+L(q_i)$.

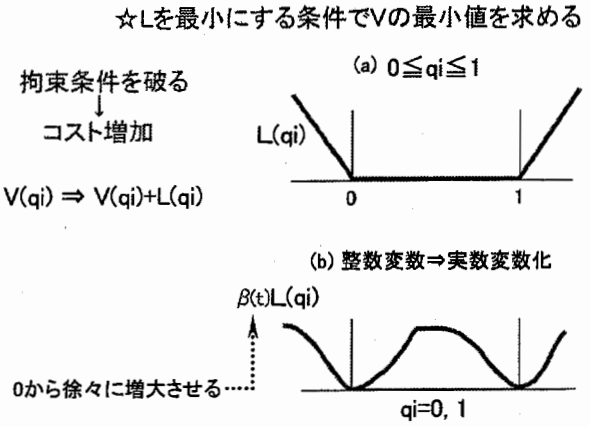


図 11. 拘束条件をコスト関数として組み込む場合.

(a) 変数がある領域内にある場合. 領域の外に変数が飛び出ると $L(q_i)$ は増大する (適用例 5,9 など). 拘束条件の入れ方には任意性がある. 上記の例 $0\le q_i\le 1$ では

$$L_1(q_i)=c_1\sum_i\Big\{(q_i-1)\theta(q_i-1)-q_i\theta(-q_i)\Big\}. \tag{32}$$

ここで $\theta(x)$ は, $x<0$ で $\theta(x)$=0, $0\le x$ で $\theta(x)$=1 となるステップ関数. c_1 は適当な正定数. この他に

$$L_2(q_i)=c_1\sum_i\Big\{(q_i-1)^2\theta(q_i-1)+q_i^2\theta(-q_i)\Big\}. \tag{33}$$

L_1 と L_2 の違いは, q_i の変化で力 $(=f_i)$ は不連続的か連続的に変わるかの点である.

(b) 整数変数を実数変数化する場合. (適用例 2,6)
例えば, $q_i=0,1$ なら

$$L(q_i)=\sum_i\Big\{(2q_i-1)^4-2(2q_i-1)^2+1\Big\}. \tag{34}$$

これは, $q_i=0,1$ で最小値をとる. (a) と異なるのは, この $L(q_i)$ を 最初からは導入せず,

$$\beta(t)L(q_i) \tag{35}$$

と置いて (これを改めて $L(q_i)$ と見なして), $\beta(t)$ を 0 から徐々に増大させることにする. 最初から $\beta(t)$ を大きくして置くと, 例えば初期の $q_i\simeq 0$ から出発するとき $q_i\simeq 0$ の近傍に留まったままになる場合がある.

この $L(q_i)$ は, $q_i=0,1$ で最小値 0 を取る.

5.3. 拘束条件を運動に組み込む場合. (適用例 7 のより進んだ問題で現れる) 例えば, $q_1^2+q_2^2+q_3^2+\cdots q_N^2=1$ のような拘束条件が与えられた場合に起こる.

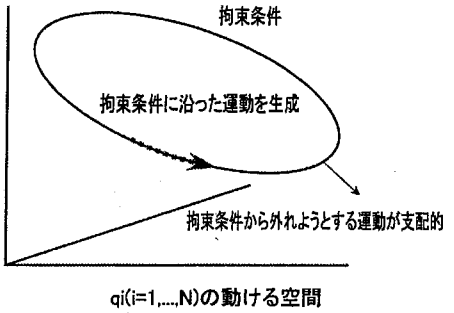


図 12(a). 拘束条件を組み込む運動.

拘束条件は運動できる領域を著しく制限する. このため, コスト関数に拘束条件を組み込む方法だと拘束条件を満足し, 且つ, 色々な q_i を取る (拘束条件に沿った) 運動が殆ど生成されないことが生じる. この時, 運動を作る必要が出る. (図 12a)

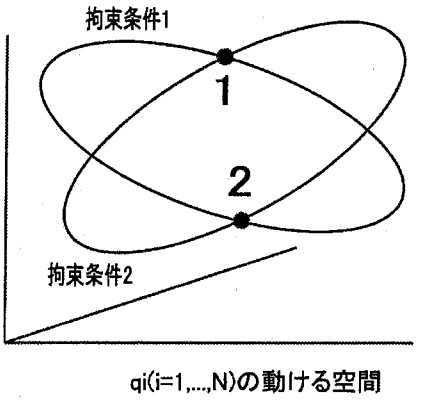


図 12(b). 拘束条件が複数ある場合の動ける領域 1,2 で (起こり得る) 運動.

一方, 複数の拘束条件がある場合には, 動ける領域が離れる不都合を生じる場合がある. 初期値が 1 の領域から運動が出発すると, 2 の領域に行くことが無い. (図 12(b) では, 領域 1, 2 は点として描かれているが, 最適化変数の多い問題では必ずしも点ではない.) 一方, 拘束条件をコスト関数に組み込む方法で導入すると, 領域 1 と 2 以外の領域を運動する場合が生じるので, 拘束条件により領域が分割されるときには, 拘束条件をコスト関数に組み込む方法が良い場合がある. 拘束条件を組み込んだ運動の作り方に付いては, 文献 [15] を参照. 特に拘束条件が沢山ある場合には不向きと思える. 例えば, 適用例 8 では (被) 最適化変数の個数の約 1/2 程度の個数の拘束条件がある.

5.4. **コスト関数の形状**. コスト関数の形状は, 図 13 のようなイメージがある. 拘束条件をコスト関数で表現した場合を含む. 図のコスト関数の (中央の • に対応する) 最小値は最適化問題の解である. 他に多くの擬似安定解がある – しかし, 拘束条件をプログラムで処理する方法の場合では, 必ずしもこのようなイメージではでない –. このように擬似安定解が (必ずしも, そうではないが) 多くあることが問題を解き難くしている.[21]

コスト関数は, 微分不可能な点を含んでていても構わない. 例えば, $V(x_i) = |x_i|$.

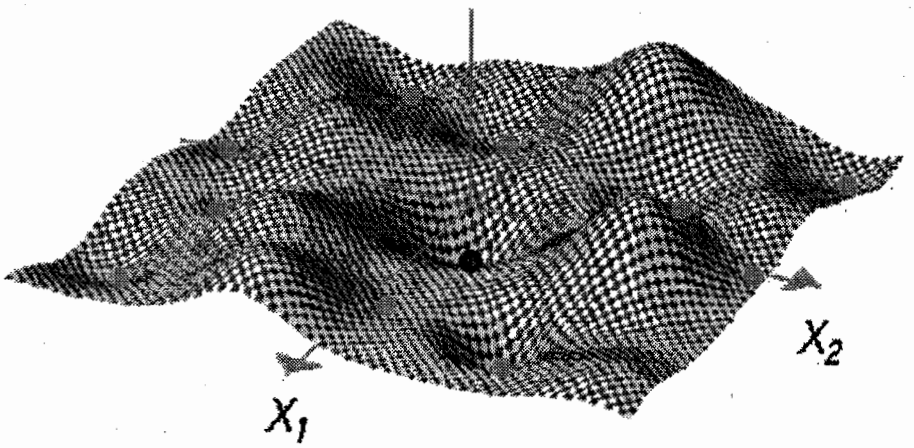


図 13. コスト関数の形状のイメージ.

6. 運動のイメージと滞在時間

6.1. 運動のイメージ. エネルギー減衰の無い場合を述べる. (エネルギー減衰のある場合はそのイメージを少し修正すれば得られる)

(A) $H(q_i, p_i)$=一定となる. (この一定値を E で表記)

つまり, $H(q_i, p_i) = T(p_i) + V(q_i)$ とすれば, 場所 q_i に依存して $V(q_i)$ が増減すれば, それを打ち消すように $T(p_i)$ が変化する.

更に, 混合性のある運動では, ((1) は混合性がある場合もない場合も成立する)

(B) q_iとp_iの空間では一様に運動する.

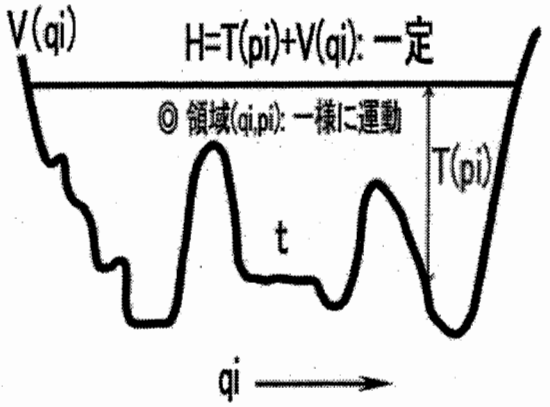


図 14. 運動のイメージ.

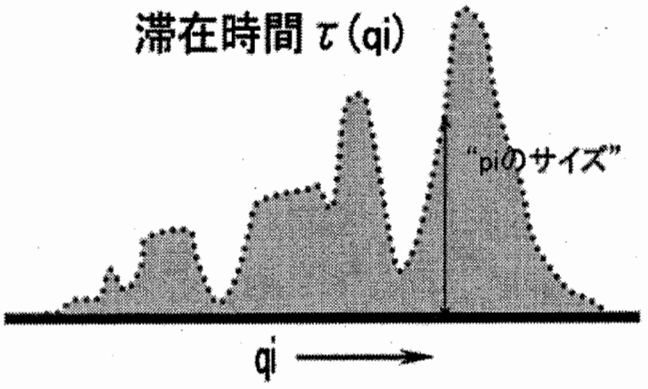


図 15. 場所に依存する滞在時間.

6.2. 場所に依存した滞在時間 $\tau(q_i)$の計算[6]. ここで, 滞在時間の計算を行う. 出発は, (A)と(B)である. 滞在時間 $\tau(q_i)$ は, $H(q_i,p_i)$ を一定にし, q_iとp_i で一様な運動するのであるから, $\tau(q_i)$ は $H(q_i,p_i)$ を一定にする条件で, 色々な p_i の値を取るサイズに比例する. 一定の条件は, $\delta(E-H(q_i,p_i)$ ($\delta(x)$: Dirac のデルタ関数) で導入できる. また, p_i のサイズは積分によって達成できる. 従って,

$$\tau(q_i) \propto \int \cdots \int dp_1 dp_2..dp_N \delta(E-H(q_i,p_i)). \tag{36}$$

ハミルトン関数は滞在時間を計算するのに必要であった.

更に進むには H の具体形が必要となる.

$$H(q_i,p_i) = \frac{1}{2}\sum_{i=1,..,N} p_i^\gamma + V(q_i), \tag{37}$$

$$= T(p_i) + V(q_i). \tag{38}$$

と置く (通常は $\gamma=2$).[25]-[27] 後は積分を行うだけである.

$$\tau_q \propto \int \cdots \int \Pi_{i=1}^N dp_i \delta(E-H[q_i,p_i]),$$

$$= \frac{d}{dE}\int \cdots \int \Pi_{i=1}^N dp_i \theta(E-H[q_i,p_i]), \tag{39}$$

$$= \frac{d}{dE}\int \cdots \int \Pi_{i=1}^N dp_i \theta(E-[\frac{1}{2}\sum_{i=1,\cdots,N} p_i^\gamma + V[q_i]]), \tag{40}$$

$$= \frac{d}{dE}\int \cdots \int \Pi_{i=1}^N dp_i \theta(E-V[q_i]-\frac{1}{2}\sum_{i=1,\cdots,N} p_i^\gamma),$$

$$= \frac{d}{dE}(E-V[q_i])^{N/\gamma}\int \cdots \int \Pi_{i=1}^N dp_i \theta(1-\frac{1}{2}\sum_{i=1,\cdots,N} p_i^\gamma), \tag{41}$$

$$= C(N,\gamma)(E-V[q_i])^{N/\gamma-1} \tag{42}$$

$$= C(N,\gamma)T(p_i)^{N/\gamma-1}, \tag{43}$$

ここで

$$C(N,\gamma) = \frac{N}{\gamma}\int \cdots \int \Pi_{i=1}^N dp_i \theta(1-\frac{1}{2}\sum_{i=1,\cdots,N} p_i^\gamma). \tag{44}$$

$E-V\geq 0$ とする. 式 (39) ではデルタ関数をステップ関数の微分で置き換えた. そして, 積分と微分の計算の順番を入れ替えた. 式 (40) では式(37)のハミルトン関数を代入した. 式 (41) は p_i を $(E-V)^{1/\gamma}$ を単位として変数変換した. 式 (42) は E で微分し, E を含まない項を $C(N,\gamma)$ と置いた. 但し, $C(N,\gamma)$ の具体的な関数形と運動とは一切関係しない. 式(43)は式(38)を使った.

式(42)と(43)から判るように, $N/\gamma-1>0$, 即ち $N>\gamma$ で, $\tau(q_i)$ はコスト関数の低い領域で増大する. 不思議なことに, コスト関数の低い領域では運動エネルギーは大きくなり (速度は速くなり), 素早く通り過ぎると思えるのに, 滞在時間は増大している.

★式(42)と(43)の具体的な関数形は, 式(37)のハミルトン関数の形に依存していることに注意.

★混合性の概念や式(42)は, 実は系を非常に長時間観察している場合に適用できる事柄である. 変数が多数あれば, "長時間" というのは途方もない長い時間である. しかし, ある力学的条件が満たされると, 途方もない長い時間は短時間で構わないというのが文献 [8] の一つの結論である.

6.3. $\tau(q_i)$ の計算：振り子の例.

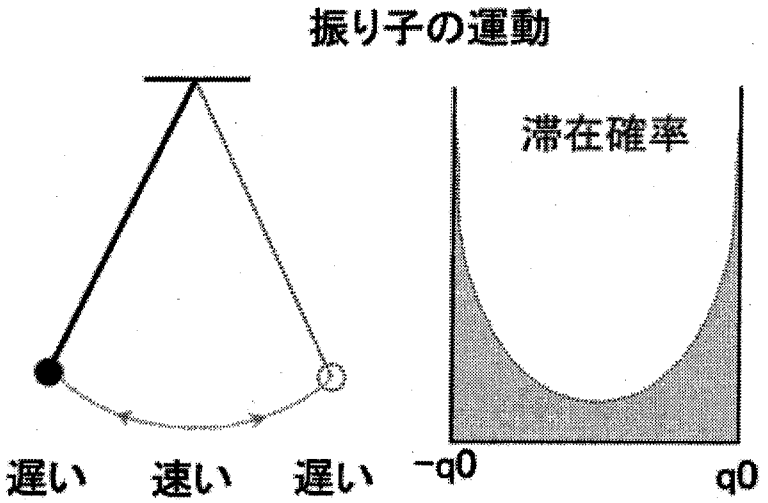


図 16. 振り子の滞在時間.

N=1, $\gamma = 2$ より,

$$\tau(q_i) \propto (E - V(q_i))^{-1/2}. \tag{45}$$

つまり, 振り子の真下では, $V(q_i)$ はもっとも小さくなるので $(E - V(q_i))$ はもっとも大きくなる. 従って, $\tau(q_i)$ はもっとも小さくなる (図 16 の右図). また, 両端で $\tau(q_i)$ は発散する. この結果は直感的に理解できる. 真下では速度が速く素早く通過する. 一方, 両端では 速度は小さくなるので, 滞在時間は増大する.

6.4. $\tau(q_i)$ の計算： 箱の中のボールの運動の例.

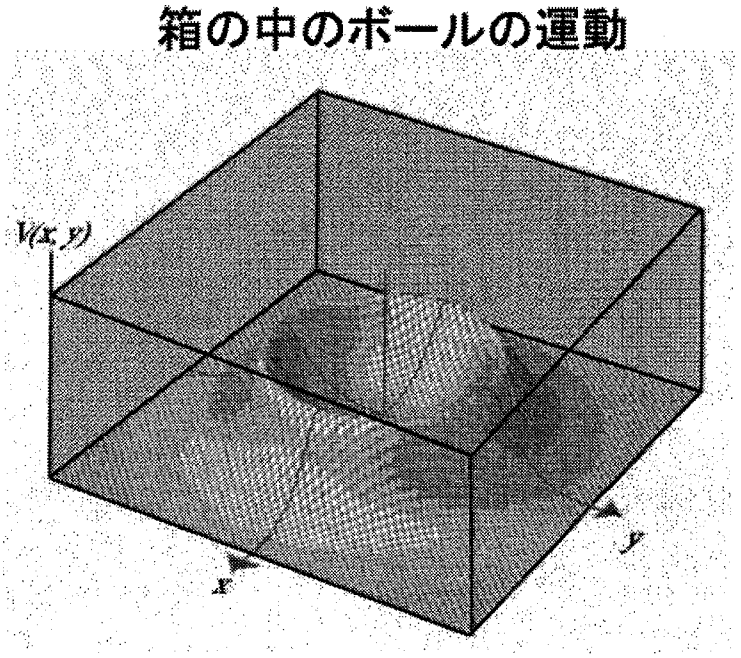


図 17. 箱の中のポテンシャルの形状

二つの場合を示す:

(a) $y=$一定でx方向のみに行ったり来たりする場合.

(b) (x,y)平面を$V(x,y)$のポテンシャルを感じながら運動する場合.

どちらの場合も, ポテンシャル関数の最高値V_0に比べ, $H(q_i,p_i)>V_0$となる条件で運動すると仮定する. $H(q_i,p_i)$は運動の過程で, 一定の値を取る. 運動は, 箱の中に限られ, 箱の境界で完全反射をすると仮定されている. (a)は, $y=$一定であるがその一定値を変えながら, x方向のみに行ったり来たりする場合の滞在時間をプロットする. ボールの運動のイメージは, 山や谷の上を一定の高度で飛んでいる飛行機に例えられる. 山や谷の起伏がポテンシャルに対応する. 運動エネルギーは, (飛行機の高度—ポテンシャルの値) に等しい.

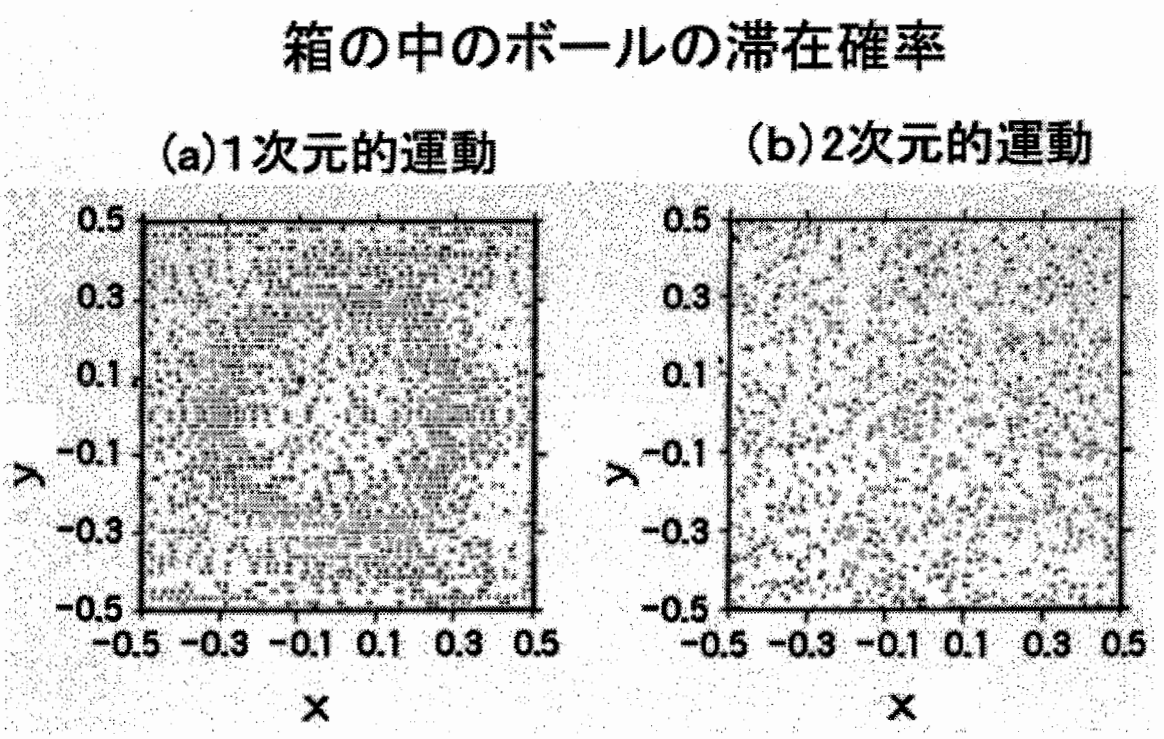


図 18. 滞在時間の$N(=1,2)$依存性

(この図は, 実際に運動方程式を計算機で解いて得た)

(a)の場合には, $N=1$と$\gamma=2$より, 図16と同様の結果である.

(b)の場合には$N=2, \gamma=2$より,

$$\tau(q_i) \propto (E-V(q_i))^0, \tag{46}$$

$$= (q_i\text{に依らず一定}). \tag{47}$$

従って, 滞在時間び分布はq_iに依らない. このことは図18の右図の観察と一致する.

7. 解を得る方法

7.1. 三つの方法.

コスト関数の最小値が判っている場合

(1) コスト関数がその値に到達したとき. 運動では, そのような最小値を与える解が幾つかある場合には, その一つを見つける. 他の解を見つけるには, 他の初期値などから計算を行う必要がある.

コスト関数の最小値が判らない場合: (全ての擬似安定解を知る以外に真の解に到着したという保証はない)

(2) 初期値から出発して長時間運動方程式を積分する. その間通過した最も最小のコスト関数を保存し記録しておく. 必ずしも, 小さなコスト関数値に到達したからといって運動を止めることは行わない.

(3) 適当に運動エネルギーを減少させて行き (式(14)のダンピング定数κや式(31)での$\xi(t)$の大きさ調節), 何処かの (擬似?) 安定解に収束させる.

(1) コスト関数の最小値が既知 ⇒ 最小値に達した時

(2) コスト関数の最小値が判らない ⇒
運動で通過したコスト関数の最小値を記録保存

(3) コスト関数の最小値が判らない ⇒
運動エネルギーを小さくして(擬似)安定解に至る

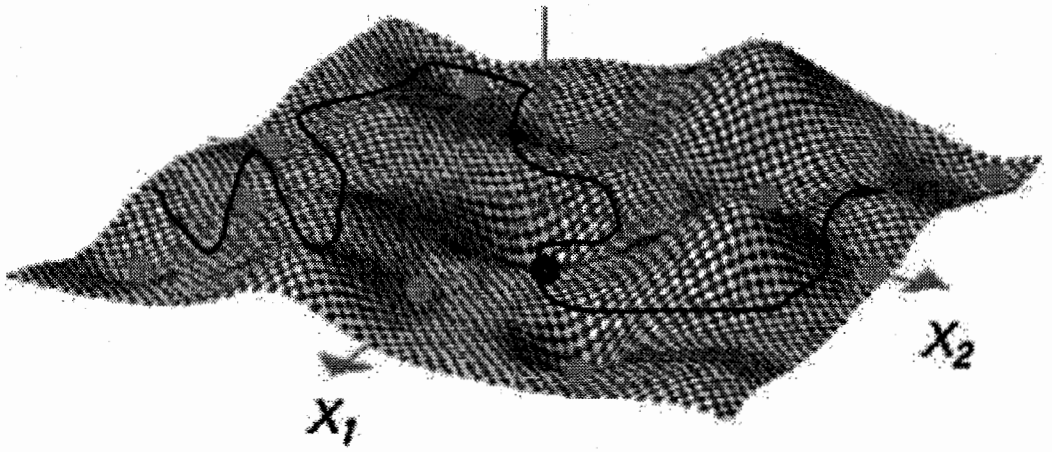


図 19. 解を得る方法.

7.2. **解を得た一つの具体例.** せいぜい数本の光ファイバーの軸合せが, 現実に出くわす問題である. ここでは全部で 100 組の軸合せ問題を解く. 向かい合ったファイバーの一つは固定され, 他方が (x, y) 平面で自由に動くと仮定する. 通常は, 異なるファイバー間には相互作用は無いが, コントロラー (controller) で創り出している (図 20).

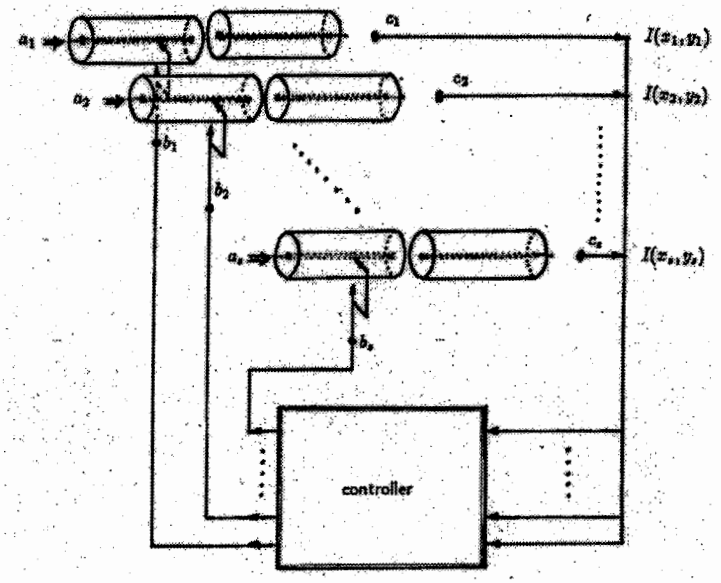


図 20. ファイバーの軸合せ問題 [22],[23].

下図は, 100 組の中で最も悪い収束を示すファイバーの光出力の時間変化を示す. ほぼ 8000 ステップで収束している. この運動は混合性のみで, エネルギーの減少は無いシステムである. エネルギーの減少を入れると $(\kappa > 0)$ を入れると, 一気に 260 ステップ程度で収束する.

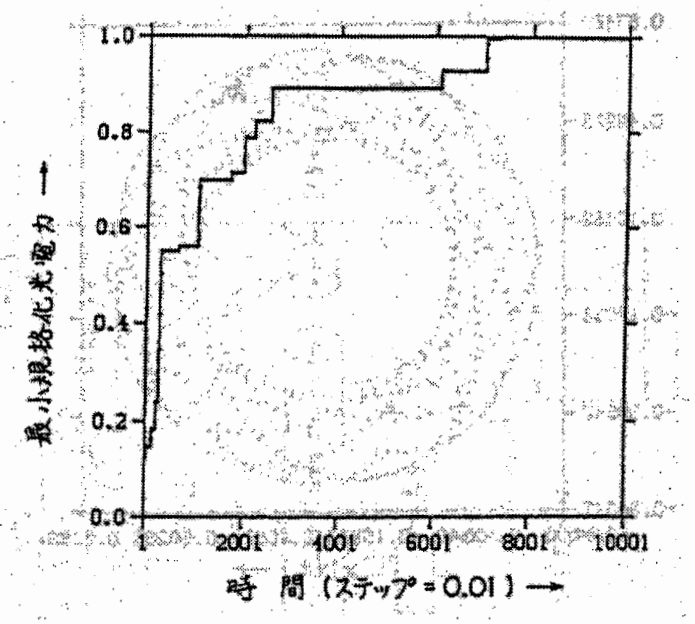


図 21. 出力光強度の時間変化

8. 高次元アルゴリズムの特徴

(1) 発見的方法 (heuristic search) である.

これは GA や SA と同じ. 真の解に到達したかどうか確認する方法がコスト関数値の最小値が判っている場合以外に一般にない.

(2) コーディングが簡単: コスト関数がそのまま使える.

自律的運動を利用して, 問題を解く. コスト関数を下げる判り易い操作を持つ GA や SA に比べ, 問題が何故解けるかが判り難い.

(3) 整数変数を実数変数化する必要.

(4) SA より優れている (と思える)

(41) コスト関数のフラットな領域を素早く通過する.

(42) 擬似安定解に捕まり難い.

(次節で説明する)

(5) GA と比べ, コーディングが簡単.

(6) ベクトルパラレル計算を実行し易い.

(7) 準ニュートン法 (共役勾配法) と比べて

コスト関数の局所的形状を利用する準ニュートン法 (共役勾配法) に比べ, (擬似) 安定解を安全に計算する. 準ニュートン法で失敗するが, HA は常に成功 (適用例 7).

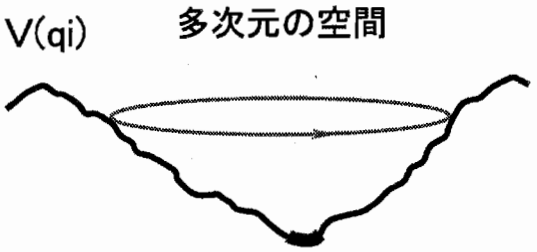


図 22. 準ニュートン法での失敗

準ニュートン法と HA との擬似安定解の捜し方の違い. 準ニュートン法と共役勾配法ではコスト関数の局所的な形状 (2 次関数で近似できること) を使う. 安定解を得るまでに 2 次関数近似が成り立たない場所を通過する場合がしばしば起こる. このような場合には, 準ニュートン法などでは失敗する. 一方, HA は特にそのような局所的性質を使わないため, 常に成功する. (適用例 8)

(8) 拘束条件に沿った運動を作り易い.

(9) (擬似) 安定解を精度良く計算するのに時間が掛かる.

HA を使って, 厳密に安定解を計算するには非常に計算時間が掛かる. 安定解の近似値 ($\sim 10^{-3}$ 程度離れた) を計算するのは速い. それ以上の近似値 (10^{-12} から 10^{-6} 程度) を得るには, 適当なところで, HA から準ニュートン法に切りかえるのが賢明である. (適用例 8)

適当なところで HA から準ニュートン法にスイッチ.

(10) コスト関数の形が具体的に与えられていない場合には, 力の計算の反復回数が多くなる.
反復回数を減らす工夫が必要 (適用例 3).

(11) 運動を積分する主要部分 (プログラムで) のが短い： 10 から 20 行程度.

(12) プログラムにバグがあるかどうかをチェックする方法がある.

(13) 外乱に対して強い.[23]

(14) 性能比較： GA や SA と比べ. (感想のような)
問題の特徴を見出すことが重要で, GA や SA や HA に組み込み方に本質的な問題は無いと思っている. これまで, HA を検討するというより, 先ず解きたい問題があって, それを解くのに専念していた. これまで色々な問題を解くことを試みたが, それほど困ったことは生じなかった.

(向いている問題: コスト関数の形状との関係で[6])

(15) コスト関数の平坦な部分が多い問題
コストがステップ型に変化する部分が多い問題. 広いゴルフコースの中のボールを探すような, コスト関数が一面平坦な中に解に当たる特異点があるような問題. しかし, 現実に直面する問題を思いつかない. コスト関数を構成する一部分がステップ型になっている建物設計問題がある (適用例 6 の以前の問題).

(16) それほど疑似局所安定解が深くない問題: (前頁で番号を付けた特徴 (41) の言い替え)

(17) ある局所安定解の周りにより安定な解が存在するという繰り返しで, 真の解に繋がるようなコスト関数の形状を持つ最適問題
具体的に挙げることは出来ない. 一般的に, コスト関数の形状に付いてはそれほどの知識を期待出来ない. 反対に, そのようなコスト関数の形状を持ちながら最後に行き着くのは偽の解であるような最適化問題は HA に向いていない. もっとも他の全ての方法も同様である.

9. SA と HA との比較 [24]

9.1. フラットなコスト関数での運動. HA では, フラットな領域を, SA に比べて素早く通過する. これは, SA に比べ HA の優れた特徴である.

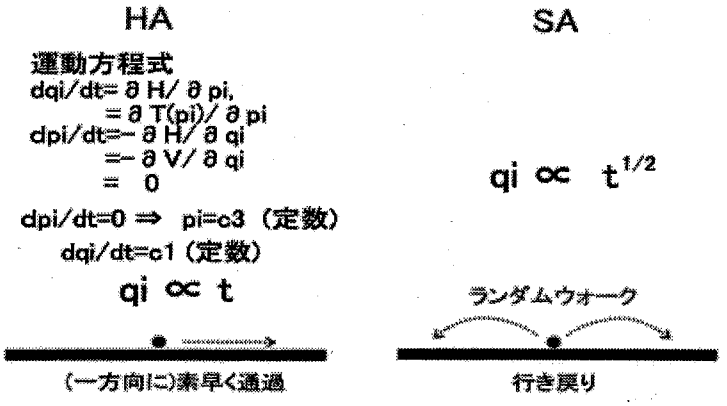


図 23. HA と SA のコスト関数のフラットな領域での運動の比較.

9.2. 擬似安定解に捕まり難い?. HA では温度に相当する量は $T(p_i)$[6] である (次節の式(52)参照). 一方, SA は, 温度 T は外部パラメータである. SA での T は q_i に依存せず一定であるのに対して, HA での $T(p_i)$ はコスト関数の低い領域に入れば増加する. このため, 深い領域の所で運動が活発になり, 抜け易くなる (と考えられる).

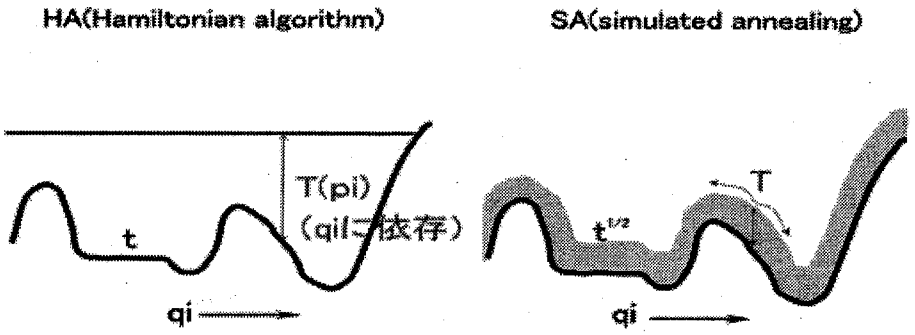


図 24. HA と SA と擬似安定解に入った時の運動の違い.

9.3. SAの温度とHA. HAはSAと類似している．しかし，SAが初めから“確率の概念”を導入する点でHAと大きく違う．SAはコスト関数のより低くなる方向により大きな確率を付けて変数q_iを変化させる．確率は温度Tで制御する．他方，HAは運動を扱っているので確率の概念は全く入ってこない．しかし，滞在時間τ_qを通して“見かけ上”入いる．見かけ上という言葉の内容を正確に言い当てることは難しいが，大体次のことを指す．隣接したq_iと$q_i+\Delta q_i$の2点での滞在時間の比ηは

$$\eta = \frac{\tau_{(q_i+\Delta q_i)}}{\tau_{q_i}}, \tag{48}$$

$$= \left\{ \frac{E - V[q_i + \Delta q_i]}{E - V[q_i]} \right\}^{N/\gamma - 1}, \tag{49}$$

$$= \left\{ 1 - \frac{V[q_i + \Delta q_i] - V[q_i]}{E - V[q_i]} \right\}^{N/\gamma - 1}, \tag{50}$$

$$\simeq e^{-(V[q_i+\Delta q_i]-V[q_i])/T_h}. \tag{51}$$

$$T_h = \frac{E - V[q_i]}{N/\gamma - 1}, \tag{52}$$

$$= \frac{T(p_i)}{N/\gamma - 1}. \tag{53}$$

式(51)を導くのに，$N/\gamma - 1 \gg 1$を使った．この式を見ると，SAで使う確率測度と類似している．しかし，次の2つの点で異なっている．SAの温度Tはq_i依存性を持ないが，T_hは最後の式が示すようにq_iに依存する．そして，T_hが温度Tと同じなのはγ=1の時であるが，T_hのγはこれ以外の値も取る．この滞在時間の比を導くのに式(42)が前提となっている．この前提が成立するための条件が混合性である．

HAはコスト関数の平坦な領域を素早く通り過ぎて，平坦でない領域で，ある極限でSAに一致するようなアルゴリズムであると言える．“ある極限”というのは，γ=2と混合性が成立する場合である．混合性はコスト関数の形状に依存する．

HAは，コスト関数の形状に依存して混合性を適宜に－平坦な領域では混合性のない(運動としては素早く通り過ぎて)，また複雑な領域ではSAと類似した動きをする－使い分けながら解を探すアルゴリズムだというのが私の大雑把なHAのイメージである．

10. 計算実行時での指針

10.1. ベルレ法: 運動方程式を解く一つの方法. 運動方程式を積分するとき良く使う方法にベルレ法がある. 比較的大きな時間ステップを取ることが出きるといわれている. また, プログラムが簡単で, しかも時間ステップあたり, 一回の力の計算で済むところが良い. 一方, Runge − Kutta 法では, 4 回の力の計算を行わなくてはならないので, 可能な限り, ベルレ法を使うようにしている. ベルレ法の特徴は, 速度の計算精度が良い (次の項が時間ステップの 1 乗ではなく 2 乗のため) と考えられている. そのため, 通常のハミルトニアンは速度の 2 乗を含むので, ハミルトニアンも精度が良い (次の補正が時間ステップの 2 乗).

時間ステップを Δ とする.

$$q_i(t \pm \Delta) = q_i(t) \pm \Delta \frac{dq_i}{dt} + \frac{1}{2}\Delta^2 \frac{d^2 q_i}{dt^2} + \cdots \tag{54}$$

とテーラー展開. 時刻 t での速度 $v_i(t)$ を計算するのに, 将来の時刻 $t+\Delta$ と過去の時刻 $t-\Delta$ での位置座標を使って,

$$v_i(t) = \frac{q_i(t+\Delta) - q_i(t-\Delta)}{2\Delta} \tag{55}$$

とするのがミソである. $q_i(t+\Delta) - q_i(t-\Delta) = 2\Delta dq_i/dt + O(\Delta^3)$ であるから ((54)の展開で奇数次の項のみが残る !),

$$v_i(t) = \frac{dq_i}{dt} + O(\Delta^2). \tag{56}$$

H は, 通常の場合 $v_i(t)^2$ であるので, H への補正は Δ の項からではなくて Δ^2 の項から現れる.

このため, ベルレ法は比較的大きな時間幅 (~ 0.01) を取っても良いと言われている.

10.2. 計算の指針：初期値設定など. ハミルトニアンを $H(q_i, p_i) = T(p_i) + V(q_i)$ とする.

運動方程式は,

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i},$$
$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \quad (= f_i).$$

(1) q_iのスケール.

q_i は, 通常 1〜10 程度を動くようにスケール変換を行っておく. 余りにも広い領域だと計算に時間が掛かりすぎる.

(2) コスト関数いついて.

コスト関数 V に適当な正定数 ζ を掛けた

$$H(q_i, p_i, \zeta) = T(p_i) + \zeta V(q_i) \tag{57}$$

をハミルトン関数として設定することが出来る. 新たなコスト関数 ζV の最小値を探すことは, 元のコスト関数 V の最小値を探すことを意味する. 力 f_i は大体 $\partial \zeta V/\partial q_i = f_i <$ 500程度 に収まるように, そして, 計算を行っている間では (解の近傍以外で), $10 < f_i <$ 500程度 になるぐらいに, ζ を調節して固定する. 多くの場合 $\zeta = 1$. 特にこの操作を行うのは, f_i が非常に大きい場合や, 非常に小さい場合である.

(3) p_iのスケール.

初期 p_i は, $|\zeta V(q_i)| \sim T(p_i)$ 程度に設定. $10|\zeta V(q_i)| \sim T(p_i)$ 程度に設定する場合もある.

(4) Δの大きさ.

時間ステップ Δ は, $H(q_i, p_i)$ がほぼ一定になるように選択. 10% 程度揺らぎがあっても良い (と思っている).

"より広い領域を運動して欲しい" ので, 可能なら Δ を大きくしたいという要求はある.

$\Rightarrow H(q_i, p_i)$ がある程度一定という条件の下で, Δ はある程度大きくしても構わないと思える. 何故なら, 滞在時間は Δ の知識ではなく, $H(q_i, p_i)$= 一定という条件を使っているため.

10.3. **プログラムのバグのチェックの方法.** (計算機プログラムのバグの取り方の一つの方法)

(*a*) $H(q_i, p_i)$ に陽な時間に依存する項を除いておく. 例えば, 式(14)で $\kappa = 0$ と置く, 式(31)で $\xi(t) = 1$ と置くなど.

(*b*) 運動方程式が得られているか, $H(q_i, p_i)$ がきちんと微分が出来ているかどうか確認.

(*c*) 運動の各時間で $H(q_i, p_i)$ が一定かどうか (通常, H は一定値の回りを揺らぐ. その揺れ幅が Δ を小さくすれば小さくなること) を確認する. 運動方程式を解く時間ステップ Δ を小さくして行けば行くほど, $H(q_i, p_i)$ が一定になる. 但し, Δ が 1/10 になれば計算時間も 10 倍にする必要ある.

11. 高次元アルゴリズムで動かせるパラメータ

これまで出てきた動かせるパラメータをまとめておく. (設定方法などは, 経験的)

(f_1) 初期値: 運動方程式を解くための出発とする (q_i, p_i) $(i = 1, 2, \cdots, N)$ の値.

q_iとp_i の設定の仕方については 11.2 節を参照. 通常,

$$\sum_i p_i = 0$$

で, 全ての $p_i = 0$ とならないように初期値を設定する. 例えば, $i = 1, \cdots, N$ で N が偶数の時には, 偶数番目の i に対して $p_i = c_i$, 奇数番目の i に対して $p_i = -c_i$ など (c_i は定数). 例えば, 全ての i に対して $c_i = 0.1$ と取る. 或は, c_i を i ごとにランダムに取る場合もある.

(f_2) γ の値.

通常, $\gamma \leq 2$ とする. これまでの例では, $\gamma \simeq 0.6$ が解への収束が良かった.

(f_3) 正定値対称行列 A の設定.

(1) 式(22)の λ は, 経験上, 余り大きくないほうが良かった. 例えば, $\lambda \simeq 0.1 \sim 0.5$.

(2) また, 固有値 ϵ_i $(i = 1, \cdots, N)$ は, $\epsilon_i > 0$ で, 且つ, 全てが同じ値を持ってはいけない. また, その最大値と最小値の比が余りかけ離れていないほうが良い. 例えば, $\mathrm{Max}_i\{\epsilon_i\}/\mathrm{Min}_i\{\epsilon_i\} \simeq 2 \sim 3$.

(f_4) 拘束条件の処理の仕方： 特にコスト関数として導入した場合.

変数がある領域内を動く場合は, 式(32)でも式(33)でもどちらでも構わない. 後者の方がコスト関数の微分係数が連続になるため, 好みであるが.

(f_5) ダンピング定数 (κ) の大きさ. 式(31)での $\xi(t)$ の値 (速度のスケール).

(1) 式(14)の κ は, 適用例 7 では比較的大きな値 (=10) を設定している. 問題に応じて対処するというのが今の知見.

(2) 式(31)の $\xi(t)$ の値の設定は, 時間ステップ Δ を単位にして, $1/(1-\xi(t))$ 回で運動エネルギーが殆んどゼロとなるのを目安にする. 例えば, $\xi(t) = 0.999$ だと, 1000 ステップ目で運動エネルギーが殆んどゼロとなる. 5.1. 節参照.

(f_6) コスト関数の設定の仕方: 任意性がある場合がある

コスト関数は一意的に決まっている場合でも, 実際にはコスト関数のある領域を変形することが出来る. ある領域とは, コスト関数が明らかに大きな領域などで, 最早そのように大きいコスト関数値を与える変数には興味がない場合である. そのような場合には, コスト関数値はある程度以上 $(= V_c)$ にならないように新しいコスト関数を, 例えば,

$$V + \omega(V - V_c)\theta(V - V_c)$$

又は,

$$V + \omega(V - V_c)^2\theta(V - V_c)$$

などと取ることが出来る. ω は, 大きな正定数. これに類似したことを, 光ファイバーの軸あわせ問題で行っている. θ はステップ関数: $0 \leq x$ なら $\theta(x) = 1$, それ以外で $\theta(x) = 0$.

また, 式(27)のように設定する場合もある (適用例 9).

(f_7) 拘束条件の制御: 式(35)の $\beta(t)$

問題に応じて対処する.

(f_8) 式(57)の ζ: $\zeta V(q_i)$ を新たなコスト関数とする.

10.2 節参照.

(f_9) 方程式を解くための時間ステップ (Δ の大きさ).

11.2. 節を参照. f_i が大きくなると逆に Δ を調節して (小さくして), 運動方程式(2)と(3)がきちんと計算できるようにする方法もある. この時には, 常に (f_8) の $\zeta = 1$ としてよい.

12. まとめ

適用例から分かるように, アルゴリズムを研究すると言うより, 実際に出会った (出会う) 問題を解決する過程の 1 つの部分として HA を手軽に利用している. この方法は, このような立場で利用するなら便利な道具の 1 つとなると信じている. 特に,

(1) コーディングが簡単,

(2) プログラムが書ける: (バグチェックの手段がある)

(3) 運動を解くプログラムの主要部分は, 10 行程度である

(4) これまで十数例が解けた

などのため気楽に使えるのではないかと思っている.

文献

(1) 新上 和正, "多自由度複雑系ダイナミクスの研究: その理解と応用" ATR テクニカルレポート, (TR-O-0101, 1996); 新上 和正, 佐々田 友平, "最適化問題の 1 考察: 高次元化と自律運動", 物性研究 Vol. 56, No. 3, 862-870(1996).
(2) Ohya K. and Shinjo K., "Hamiltonian Algorithm for Sound Synthesis", Proc. of the 7th Inter. Colloquim ed. by Kajiwara J., Li Z, and Shon Ho K., pp401-407 (1999).
(3) Shinjo K., Shimogawa S., Yamada J., and Oida K., "A Strategy of Designing Routing Algorithms Based on Ideal Routings" International Journal of Modern Physics, Vol. 10, No. 1, 63-94(1999).
(4) 新上 和正, 佐々田 友平, "ハミルトニアンダイナミクスの功利性", 日本物理学会全国大会予稿集, 13p-F2, (1993).
(5) Shinjo K., "Hamiltonian Algorithm: A Method to Solve Optimization Problems" ATR Journal 1, 48-49, (1998).
(6) 新上和正, "高次元アルゴリズム - 最適化問題を解く一つの方法 -", ファジー学会誌 11(2), pp382-395(1999).
(7) 新上和正, "高次元アルゴリズム", Bit 31(7), pp2-8(1999).
(8) Shinjo K. and Sasada T., "Hamiltonian Systems with Many Degrees of Freedom: Asymmetric Motion and Intensity of Motion in Phase Space" Phys. Rev. E54, 4686-4700 (1996); Shinjo K., "Formation of a glassy solid by computer simulation", Phys. Rev. B40, 9147 (1989).
(9) 新上 和正, "高次元系のダイナミクス: 相空間における運動の強度", 大阪大学大型計算機センターニュース (スーパーコンピューターシンポジューム), 29-38(1997).
(10) 新上 和正, 佐々田 友平, "相空間における運動の強度", 物性研究 Vol. 64, No. 4, 390-395(1996).
(11) 新上 和正, "高次元系のハミルトンダイナミクス", 物性研究 12 月号「複雑系」研究会報告, 269-282 (1993).
(12) 島内剛一, 有澤誠, 野下浩平, 浜田穂積, 伏見正則 編集, 「アルゴリズム辞典」共立出版社 1994. この解説ではアルゴリズムの名称をこの辞典で使われているものを参考にする.
(13) J. van Leeuwen 編 (広瀬健, 野崎昭弘, 小林孝次郎監訳) 「コンピュータ基礎理論ハンドブック」丸善株式会社 1994.
(14) ここでは, 幾らでも時間を掛ければ解けるような最適化問題を対象とする.
(15) 力学系の教科書は沢山ある. 例えば, E. T. ホイーテッカー (多田 政忠, 薮下 信訳), 「解析力学 (上下)」講談社 (1979).
(16) 湯川 秀樹, 戸田 盛和, 久保 亮五監修, 「岩波講座 現在物理学の基礎, 統計物理学」岩波書店 (1978) の 10 章のエルゴード問題.
(18) アーノルド, アベズ (吉田耕作訳), 「古典力学のエルゴード問題」吉岡書店 (1972).
(19) Zaslavsky G. M. (三島信彦 等訳), 「カオス – 古典および量子力学系 –」現代工学社 (1989).
(20) これはエルゴード仮説と言われ, 物理学の 1 つの分野である統計力学でよく知られている最も重要な基本問題である. 興味のある方は文献 [19] を読むことをお勧めする. 事の起りは, 「1 つの運動の軌跡が, 時間の概念の消失した, 平均化した量と等しくなる」ということである. 例えば, 運動の軌跡といえばただ 1 つの軌道をイメージするが, 数学的には多数の軌道を同時に扱う測度論を必要とする. そのギャップをどう埋めるのでしょうか. この問題は解決していません.
(21) 正確には, 解に相当する所を除いて殆どの領域で凸凹が全く無いような問題も, 難問であることが知られている. Baum E., Phys. Rev. Lett., 59, 374 (1987).
(22) 水上 雅人, 平野 元久, 新上 和正, "ハミルトンアルゴリズムを用いて多自由度光軸調整法", 計測自動制御学会論文集, Vol. 33, No. 7, 709-715 (1997).
(23) Mizukami M., Hirano M., and Shinjo K., "Alignment of multiple axes in a multistage optical system by using Hamiltonian algorithm", Proc. 46th IEEE Elect. Comp. & Tech Conf., 1284-1288(1996).
(24) SA での温度のスケジューリングを考えに入れていない. 最もプライマリーな形で, HA と SA を比べている.
(25) 自然現象で現れる場合には $\gamma = 2$ である. 例えば, ニュートン方程式では γ =2 である. γ をこのように拡張して扱うのは, 初めてである. しかし, この拡張によって運動は非常に奇妙なものとなる.
(26) 新上 和正, 下川 信祐, 佐々田 友平, "ハミルトンダイナミクス: 非 2 次運動量形式を持つ系", 日本物理学会 1997 年秋の分科会 6p-YE-3(1997.10.5).
(27) 新上 和正, 下川 信祐, 佐々田 友平, "ハミルトンダイナミクス II: 非 2 次運動量形式を持つ系", 日本物理学会 53 回年会 31a-YA-12(1998.3.30).

付録. 適用例と HA との関連

これから紹介する適用例で, HA と最適化問題としての特徴を, 簡単に解説.

(1) 通信ネットワークのリソース配分の最適化.
実数変数の問題. コスト関数が複数 (4 つ) あるのが特徴. 複数あるときに, 複数のコスト関数の関係をどう考えるかが重要となる. しかし, この例では, この点について明らかにされていない.

(2) 動的ルーティングの政策.
整数変数. よって実数変数化を行うように拘束条件をコスト関数に繰り込む. コスト関数は一つ.

(3) 可変指向性アンテナのパラメータ設計.
実数変数. コスト関数は, 陽な関数形は無い. 最適化変数からコスト関数を計算するアルゴリズムが与えられている. このため力の計算を反復回数が多いため, それに消耗する時間を短くするように工夫している点に意義がある. 混合性を導入するのに, γ を導入している稀有な一つの例.

(4) JPEG 量子化テーブルの最適化.
整数変数問題だが, 変数が, 0 から 255 までの整数であるため, 常道である整数変数を実数変数化する手続きを取っていない. 整数を実数変数と考え, 実数変数に最も近い整数値を使ってコスト関数などを計算している. コスト関数は 2 つある. 量子化テーブルには, 標準的なものがあるので, 本文で述べたパレート最適化の考えを使って処理している.

(5) ニューラルネットの学習.
実数変数. コスト関数の最小値 (=0) が判っている例. コスト関数形を, 任意に取って良い例でもある. あまり意味の無い領域を運動しないように境界条件を勝手に導入している. 境界条件をコスト関数で運動に組み入れている例でもある. 混合性導入のため, 正定値対称行列を導入する.

(6) コージェネレーションシステムの最適設計・制御.
整数変数と実数変数が混合する簡単な様で割に複雑な実用的な例. コスト関数は一つ.

(7) 分子構造の決定.
ここで紹介した範囲を少し進むと, 拘束条件を運動に組み入れる必要が出てくる例. また, 準ニュートン法や共役勾配法が失敗する問題である. 混合性導入のため, 正定値対称行列を使っている.

(8) 行列の固有値の探索.
対称行列の固有値と固有ベクトルを求める古い問題である. 古い解き方の殆どは, 代数的な解法なので, 決まった処理時間が掛かる. ここの方法は, 近似的に解く代わりに, 処理時間をセーブする. コスト関数の最小値が判っていて関数形に任意性がある. また, 最適化変数の約半数ぐらいの多数の拘束条件がある問題となっている. 多くの拘束条件をコスト関数として導入したために, コスト関数は, 最小値 (=0) の近傍でも複雑になっているように見える.

(9) どんなモノがブレイクして行くのか?
世界で起こる様々な出来事の因果関係 (相関関係) を計算する例. これもコスト関数の最小値 (=0) が判っている例. 言語も対象にした, 上記の例と比べ異色. 勿論, HA や最適化問題を単に解けば, 直ぐにブレイクするモノが判るわけではない.

(次ページに表に纏める)

問題	(被)最適化変数			拘束条件			コスト関数				
	整数	実数	混合	そのまま (プログラムで処理)	コスト関数へ	運動へ	単一	複数 (競合)	陽な関数形		陽な関数形なし
									一意的	任意性	(アルゴリズムあり)
①通信ネットワークのリソース配分の最適化		○		○				○	○		
②動的ルーティングの政策	○			○	○		○		○		
③可変指向性アンテナのパラメータ最適化		○		○			○				○
④JPEG量子化テーブルの最適化	○			○				○			○
⑤ニュラルネットワークの学習		○		○			○			○	
⑥コージェネレーションシステムの最適設計・制御			○	○	○		○		○	○	
⑦分子構造の最適化		○				○	○				○
⑧行列の固有値の探索		○			○					○	
⑨“どんなモノがブレイクしてゆくのか?”		○		○	○		○			○	
制御アルゴリズムの設計		○		○				○	○		
光ファイバーの軸合せ		○					○		○		

第3章　通信ネットワークのリソース配分の最適化

概要

通信ネットワークの最適設計は古くて新しい課題である．この課題に対する一つのアプローチとして，我々は通信ネットワークのリソース配分の最適化に取り組んでいる．具体的には，予測される通信トラフィック条件及びリソース制約条件の下でネットワーク性能を最大化するための最適なネットワークリソース配分及び通信トラフィック経路配分を求める検討を進めている．

通信ネットワークのリソース配分の最適化問題は，対象とするネットワークの規模が大きいほど，あるいはトポロジーが複雑であるほど最適化変数が増加する．もし解法にスケーラビリティがあれば，非常に多変数の場合でも問題を部分分解することなく最適解を導くことができる．言い換えれば，通信ネットワークの総合設計の道が開ける．我々は多変数に強い高次元アルゴリズムを本問題の解法として利用する．

1．問題の説明

1.1. 通信ネットワークの最適資源配分問題. 近年，ユーザ（企業，個人等）がパケット通信ネットワークを構築するケースが増えている．ユーザネットワークの管理者にとって，最初の大きな課題は「限られた予算（資源）の中で，いかに性能の良いネットワークを設計するか」である．我々は，この課題を最適資源配分問題として解くことに取り組む．

通信ネットワークは，予測される通信トラフィック需要に対して，(1)できるだけ通信品質を良くするように，(2)できるだけネットワークリソースが有効利用されるように，設計されるべきである（その他にも設計要求はあるが，ここでは省略する）．(1)の例として，“送受信間の遅延時間をできるだけ短くする”，“パケットロスを少なくする”等々が挙げられる．(2)は，具体的には“ネットワークが持つ潜在的能力を最大限活かすことによってネットワーク全体の通信容量の拡大を図る”ことである．ここでは(1)及び(2)をネットワーク性能の評価基準とみなす．なお，本問題ではネットワーク設備（ノードやリンク）が持つパケットの処理能力をネットワークリソースと考える．例えば，ノードの単位時間あたりのルーティング能力，リンクの帯域等々である．

通信ネットワークは，投資可能な予算の中で設計・構築される．問題を簡単化するため，この予算制約条件を「利用可能なネットワークリソースが限られている」と置き換える．前提条件として，通信ネットワークに入力するトラフィックは与えられるものとする．我々は，ネットワークリソースが予測トラフィックに対して必ずしも十分ではない場合を想定する（現実の通信ネットワークでは，通信トラフィックは右上がりに増加し，ネットワークリソースが不足する傾向にある）．この場合，ネットワークリソースは競合あるいは相互作用し，そのふるまいはネットワークの規模が大きいほど複雑である．

このリソース制約の下で，ネットワークに対する要求を最大限満たすためには，ネットワークリソースをどう配分したらよいか？また，トラフィックをどう経路配分したらよいかを問題としてとりあつかう．最終目標は，本問題の解を通信ネットワークの基本設計に適用することである．

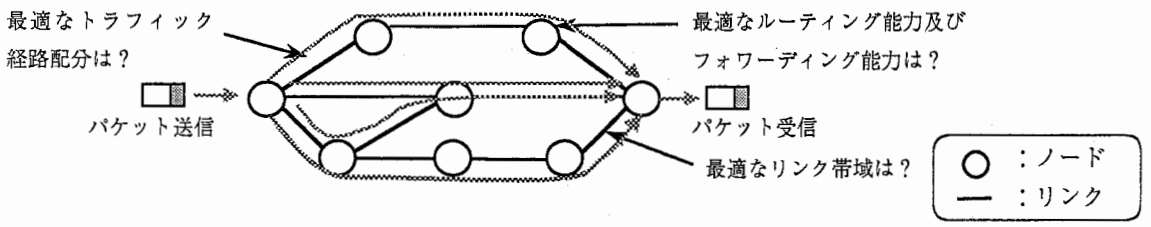


図1．通信ネットワークのリソース配分の最適化

1.2. 通信ネットワークモデル．我々の最適化問題を設定する前に，まず通信ネットワークをモデル化する．現実の通信ネットワークは様々な構成要素から成り立つが，ここでは取り扱いやすさを考慮して，基本構成要素をモデル化の対象とする．

ネットワークは “ノード”と“リンク”から構成されるものとする．そして任意のノード間でパケット通信が行われるものとする．パケットは，送信元ノードにおいてある確率分布に従って生起し，宛先ノードに向けて送信される．

中継ノードは，あるルーティングレートで到着パケットの経路を選択する（ルーティング）．経路決定されたパケットは，あるフォワーディングレートで出力リンクに転送される（フォワーディング）．

リンクは，そのリンクに設定された帯域の範囲内でパケットを次のノードへ転送する．このパケット転送はリンク長に比例した伝送遅延をうけるものとする．

本ネットワークモデルにおいて以下の３つをネットワークリソースとして取り扱う．

(1)各ノードのルーティングレート（単位時間あたりに経路選択可能なパケット数）
(2)各ノードのフォワーディングレート（単位時間あたりに転送可能なパケット数）
(3)各ノードのリンク帯域（単位時間あたりに転送可能なパケット数）

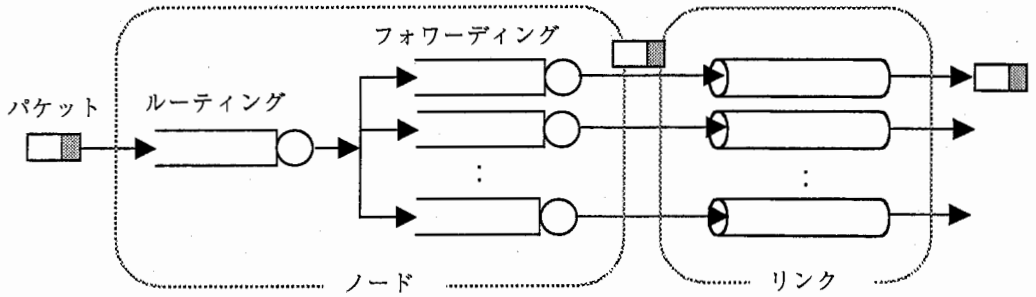


図２．通信ネットワークモデル

２．最適化問題の設定

まず，通信ネットワークのリソース配分の最適化問題を簡単に整理する．

(1)目的：ネットワークリソースの配分及びトラフィック経路配分の最適化
(2)要求：ネットワーク性能の最大化
(3)制約：利用可能なネットワークリソースが限られている．

これらに沿って，最適化問題（最適資源配分問題）を設定する．

2.1. 最適化変数．次に示す４種類の配分率を最適化変数として定義する．

・ルーティングレート配分率
・フォワーディングレート配分率
・リンク帯域配分率
・トラフィック経路配分率

これらの配分率に従ってネットワークリソース及びトラフィック経路を配分するものとする．以下，各々について具体的に説明する．

(1)ルーティングレート配分率

ルーティングレート配分率は，ネットワーク内の総ルーティングレートを各ルーティング処理部に配分する割合である．ネットワーク内のルーティングレートの総和を M_r として，そのうちあるルーティング処理部 r に配分されるレートを μ_r とすると，r のルーティングレート配分率 x_r は次式で表わされる．

$$x_r = \frac{\mu_r}{M_r}, \quad \sum_{r \in N} x_r = 1, \quad 0 \le x_r \le 1 \tag{1}$$

(2)フォワーディングレート配分率

フォワーディングレート配分率は，ネットワーク内の総フォワーディングレートを各フォワーディング処理部に配分する割合である．ネットワーク内のフォワーディングレートの総和を M_f として，そのうちあるフォワーディング処理部 f に配分されるレートを μ_f とすると，f のフォワーディングレート配分率 x_f は次式で表わされる．

$$x_f = \frac{\mu_f}{M_f}, \quad \sum_{f \in L} x_f = 1, \quad 0 \le x_f \le 1 \tag{2}$$

(3)リンク帯域配分率

リンク帯域配分率は，ネットワーク内の総帯域を各リンクに配分する割合である．ネットワーク内の総リンク帯域を B として，そのうちリンク l に配分される帯域を b_l とすると，l のリンク帯域配分率 x_l は次式で表わされる．

$$x_l = \frac{b_l}{B}, \quad \sum_{l \in L} x_l = 1, \quad 0 \le x_l \le 1 \tag{3}$$

(4)トラフィック経路配分率

トラフィック経路配分率は，送受信間パケットトラフィックをとりうる経路に配分する割合である．ノード s からノード d への送受信間パケット生起率を λ_{sd}，s～d 間でとりうる経路のうち，ある経路 p へ配分されるパケット生起率を λ_p とすると， p のトラフィック配分率 x_p は次式で表わされる．

$$x_p = \frac{\lambda_p}{\lambda_{sd}}, \quad \sum_{p \in Psd} x_p = 1, \quad 0 \le x_p \le 1 \tag{4}$$

2.2. コスト関数. 通信ネットワーク性能をコスト関数として定義する．通信ネットワーク性能の定義の仕方には任意性がある．ここでは，送受信間のパケット転送遅延時間及びネットワークリソースの使用率の最大値をネットワーク性能と定義し，それらの値が小さいほどネットワーク性能が高いと考える．具体的には，送受信間パケット転送遅延時間の平均値D，ルーティング部使用率の最大値A_r，フォワーディング部使用率の最大値A_f，及びリンク帯域使用率の最大値A_lの計4項をネットワーク性能のコスト項と考え，これらの重み付け加算をコスト関数Vとして定義する．

$$V = w_1 D + w_2 A_1 + w_3 A_2 + w_4 A_3 = f\left(x_r, x_f, x_l, x_p\right) \tag{5}$$

ここで W_1～W_4 は重み係数である．V は，x_p, x_r, x_f 及び x_l の陽な関数として記述できる．D, A_r, A_f 及び A_l の詳細は，付録1及び付録2に示す．

2.3. 拘束条件. 最適化変数 x_r, x_f, x_l 及び x_p は全て実数変数として取り扱う．各最適化変数の定義上，それらのとりうる範囲は 0 以上 1 以内である．各変数がこの値域で運動するように反射操作を導入する．また，（疑似）安定解を得るためにエネルギー減衰を加える．

その他の制約条件として，ルーティングレートの総和 M_r，フォワーディングレートの総和 M_f，及びリンク帯域の総和 B が与えられる．

3．力学系の構成

3.1. ハミルトニアン. ハミルトニアンHは，(b1)の形式で定義する．

$$H = V\left(x_r, x_f, x_l, x_p\right) + T\left(p_r, p_f, p_l, p_p\right) \tag{6}$$

ここで，$V(x_r, x_f, x_l, x_p)$は，2.2節に定義したコスト関数であり，$T(p_r, p_f, p_l, p_p)$ は運動項である．p_r, p_f, p_l及び p_pは，各々最適化変数x_r, x_f, x_l及びx_pの仮想的運動量である．運動項については3.2.節で説明する．

3.2. 混合性. 運動の混合性に特別な工夫はせず，(c1)の形式で定義する．

$$T\left(p_r, p_f, p_l, p_p\right) = \frac{1}{2}\left(\sum p_r^{\,2} + \sum p_f^{\,2} + \sum p_l^{\,2} + \sum p_p^{\,2}\right) \tag{7}$$

4．運動方程式

コスト関数Vをルーティングレート配分率x_rによって偏微分することによってx_pに加わる力を計算する．同様にして，フォワーディングレート配分率x_fに加わる力，リンク帯域配分率x_lに加わる力及びトラフィック経路配分率x_pに加わる力を計算する．

$$\frac{d^2x_r}{dt^2} = -\frac{\partial V}{\partial x_r} = w_1\frac{\partial D}{\partial x_r} + w_2\frac{\partial u_r}{\partial x_r} + w_3\frac{\partial u_f}{\partial x_r} + w_4\frac{\partial u_l}{\partial x_r} \tag{8}$$

$$\frac{d^2x_f}{dt^2} = -\frac{\partial V}{\partial x_f} = w_1\frac{\partial D}{\partial x_f} + w_2\frac{\partial u_r}{\partial x_f} + w_3\frac{\partial u_f}{\partial x_f} + w_4\frac{\partial u_l}{\partial x_f} \tag{9}$$

$$\frac{d^2x_l}{dt^2} = -\frac{\partial V}{\partial x_l} = w_1\frac{\partial D}{\partial x_l} + w_2\frac{\partial u_r}{\partial x_l} + w_3\frac{\partial u_f}{\partial x_l} + w_4\frac{\partial u_l}{\partial x_l} \tag{10}$$

$$\frac{d^2x_p}{dt^2} = -\frac{\partial V}{\partial x_p} = w_1\frac{\partial D}{\partial x_p} + w_2\frac{\partial u_r}{\partial x_p} + w_3\frac{\partial u_f}{\partial x_p} + w_4\frac{\partial u_l}{\partial x_p} \tag{11}$$

式(8)～(11)は解析的に求めることができる．計算の詳細を付録３に示す．

5．計算プログラム

5.1. 最適化変数の更新． Verlet法を用いて最適化変数を更新する．時刻tにおける最適化変数値を$x(t)$，その速度を$v_x(t)$とすると，$x(t+\Delta t)$は次式に従って計算される．

$$x(t+\Delta t) = 2\Delta t v_x(t)k + x(t-\Delta t) \tag{12}$$

ここでkは減衰定数である．

Verlet法によって最適化変数x_pを更新するソースプログラム（C++関数）の抜粋を以下に示す．

```
E_okNg C_networkDesigner::calculatePathX()
{
 (略)
        for(p = 1; p <= this->pathCount; p++) {
                // get path
                L_pathPtr = &G_pathTable[p];
                // get traffic pointer
                L_trafficPtr = L_pathPtr->trafficPtr;
                // Verlet 法による最適化変数 xp の更新
                L_work = L_trafficPtr->x[0];
                L_trafficPtr->x[0] = L_trafficPtr->x[1];
                L_trafficPtr->x[1] = 2.0 * this->dt * L_trafficPtr->vx * this->dampingRate + L_work;
        }
 (略)
        return OK;
}
```

5.2. 最適化変数の拘束． 最適化変数の運動域を[0, 1]に拘束するために反射操作を導入する．

反射によって最適化変数x_pの運動域を[0, 1]に拘束するソースプログラム（C++関数）の抜粋を示す．

```
E_okNg C_networkDesigner::calculatePathX()
{
 (略)
        for(p = 1; p <= this->pathCount; p++) {
                // get path
                L_pathPtr = &G_pathTable[p];
                // get traffic pointer
```

```
                L_trafficPtr = L_pathPtr->trafficPtr;
(略)
                //反射によって最適化変数 x_p の運動域を[0, 1]に拘束する
                if(L_trafficPtr->x[1] >= 1.0) {
                        L_trafficPtr->vx = -1.0 * L_trafficPtr->vx;
                        L_trafficPtr->x[0] = 2.0 - L_trafficPtr->x[0];
                        L_trafficPtr->x[1] = 2.0 - L_trafficPtr->x[1];
                }
                if(L_trafficPtr->x[1] <= 0.0) {
                        L_trafficPtr->vx = -1.0 * L_trafficPtr->vx;
                        L_trafficPtr->x[0] = -1.0 * L_trafficPtr->x[0];
                        L_trafficPtr->x[1] = -1.0 * L_trafficPtr->x[1];
                }
        }
(略)
        return OK;
}
```

6．計算結果

6.1. 計算環境. 計算機は，HP社のワークステーション（VISUALIZE Work Station C240）を用いた．計算機のCPUはPA-8200(200/236MHz)×１である．計算プログラムはC++にてコーディングし，コンパイルにはgcc 2.95.2 を用いた．

6.2. 計算条件.

(1)ネットワーク条件

25個のノードを格子状に配置し，隣接ノード間をリンク接続したネットワーク条件を設定した．この時リンク数は144，トラフィック経路数は16,752，そして最適化変数の総数は16,921(＝25+144+16,752)である．

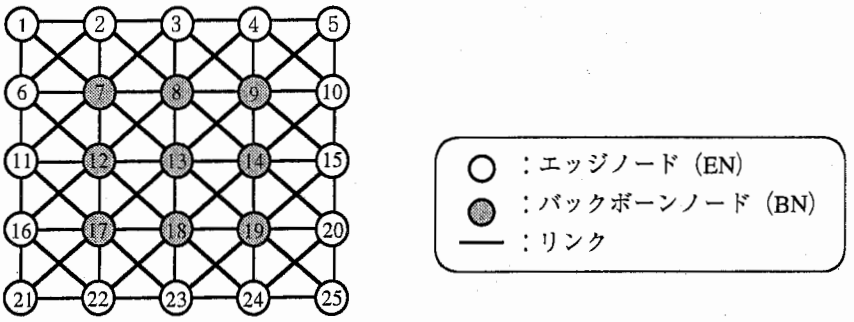


図３．ネットワークトポロジー

(2)トラフィック条件

全てのノード間に，ポアソン過程にしたがうパケットトラフィックが生起する仮定した．また，ネットワークのバックボーン側にトラフィックが集中する状況を想定して，以下のようなパケット生起率条件を与えた．

EN間：EN～BN間：BN間＝1：2：4

ここで， ENはエッジノード， BNはバックボーンノードを表わし，右辺は各ノード間トラフィックのパケット生起率の総和の比率である．

6.3. 計算結果. 計算時間は，約40分であった．計算結果を図4から図7に示す．
「通信トラフィックが集中するバックボーン部分により多くのネットワークリソース（ルーティングレート，フォワーディングレート及びリンク帯域）が配分される」という我々の直感とよく一致する結果を得た．

本リソース配分結果にはほぼ対称性がみられる．これは妥当な結果であるといえる．何故なら，計算に用いたネットワークのトポロジーに対称性があること，そしてそのネットワークに対称なトラフィックがかかっているからである．

本例は，最適化変数の総数が16,921の最適資源問題であるが，高次元アルゴリズムを適用することによって，適切なネットワークリソース配分結果が得ることができた．

6.4. 今後の課題. 今後の課題としては，以下のようなものが挙げられる．
(1)バーストトラフィックを前提としたネットワーク設計

バーストトラフィックを前提としてコスト関数 V を算出する．バーストトラフィックは2状態MMPPによってモデル化し，コスト項の算出には近似計算を用いる．
(2)トポロジー設計

リンク帯域の最適配分結果から支配的リンクを抽出する．この支配的リンクを利用してトポロジーを設計する．
(3)コスト関数の評価

コスト項間の相互関係を明らかにする．またコスト項の重み付けについて評価する．

文献

(1)恩田，山田，北川，“高次元アルゴリズムによるコネクションレス網の最適設計の一検討”，1999年電子情報通信学会総合大会，B-7-47.
(2)恩田，斎藤，北川，“高次元アルゴリズムによる通信網の最適容量設計の検討”，1999年電子情報通信学会ソサイエティ大会，B-7-3.
(3)恩田，斎藤，北川，“高次元アルゴリズムによる通信網設計”，1999年情報処理学会第59回全国大会，1U-02.
(4)恩田，斎藤，北川，“高次元アルゴリズムによる網トポロジー設計”， 2000年情報処理学会第60会全国大会，1S-03.
(5)恩田，斎藤，北川，“高次元アルゴリズムによる通信ネットワークのトポロジー設計の検討”，2000年電子情報通信学会総合大会，B-7-12.
(6)恩田，仲村，種田，斎藤，“通信ネットワークの最適資源配分問題における設計パラメータの選択的最適化に対する考察”，2000年電子情報通信学会ソサイティ大会，B-7-1.
(7)恩田，仲村，種田，斎藤，“通信ネットワークの最適資源配分問題における最適化変数評価”， 2000年情報処理学会第60会全国大会，3H-05 .
(8)K.Onda and S.Saito,"Resource Allocation Adaptive to Traffic Changes in Communication Networks", IEEE Symposium 2000 on AS-SPCC.

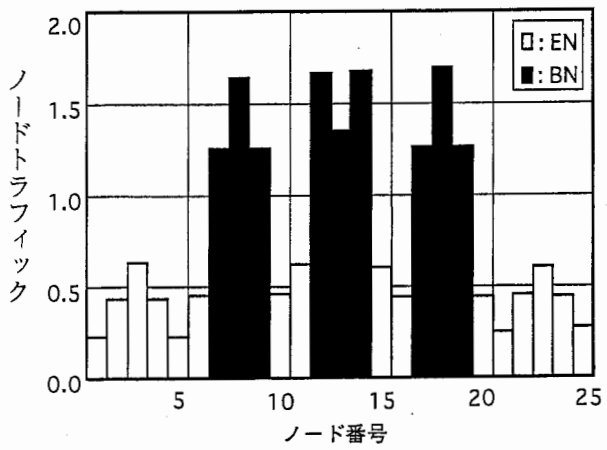


図４．ノードトラフィック配分

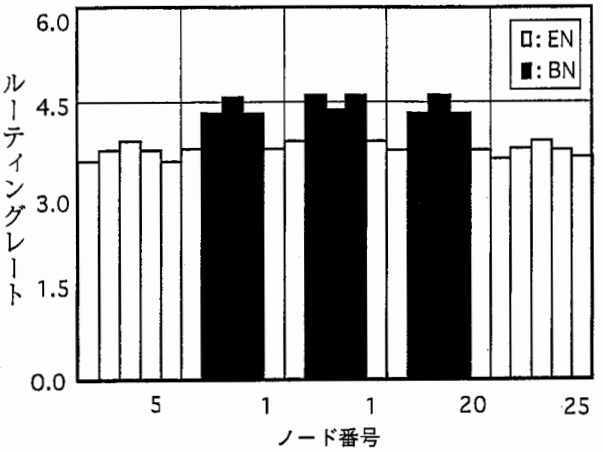


図５．ルーティングレート配分

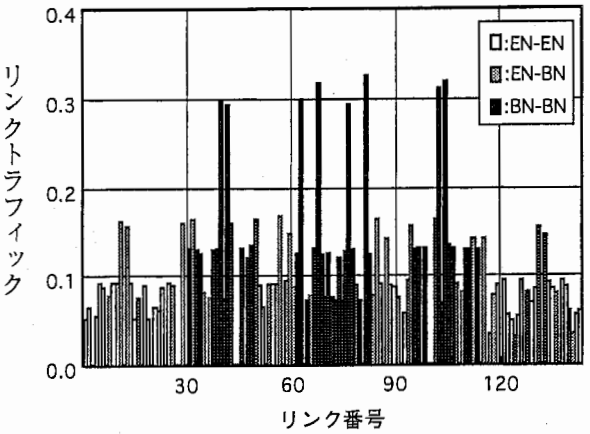


図６．リンクトラフィック配分

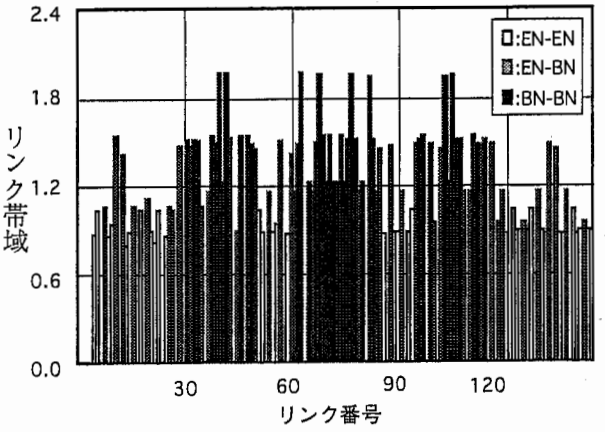


図７．リンク帯域配分

付録1　リソース使用率の計算

(1)ルーティング処理部の使用率

ノード s からノード d に向かう経路の集合を P_{sd}，P_{sd} に含まれるある経路 p へのトラフィック経路配分率を x_p，s～d 間のトラフィック生起率 λ_{sd} とすると，p に配分されるパケット生起率 λ_p について次式が成り立つ．

$$\lambda_p = x_p \lambda_{sd}, \quad \sum_{p \in Psd} x_p = 1, \quad 0 \le x_p \le 1 \tag{A-1}$$

あるルーティング処理部 r に流入する経路の集合を P_r，P_r に含まれるある経路 p から r へのパケット到着率 w_p とすると，r のパケット到着率 λ_r は次式によって計算される．

$$\lambda_r = \sum_{p \in P_r} w_p \tag{A-2}$$

r のルーティングレート μ_r とすると，r の使用率 a_r は次式によって計算される．

$$a_r = \frac{\lambda_r}{\mu_r} = \frac{1}{\mu_r} \sum_{p \in P_r} w_p \tag{A-3}$$

任意の経路 p でのパケット損失がない，すなわち $w_p = \lambda_p$ とすると，a_r は次式によって計算される．

$$a_r = \frac{\lambda_r}{\mu_r} = \frac{1}{\mu_r} \sum_{p \in P_r} \lambda_p \tag{A-4}$$

(2)フォワーディング処理部の使用率

フォワーディング処理部 f を通過する経路の集合を P_f，P_f に含まれるある経路 p のパケット到着率 w_p とすると，f のパケット到着率 λ_f は次式によって計算される．

$$\lambda_f = \sum_{p \in P_f} w_p \tag{A-5}$$

f のフォワーディングレートを μ_f とすると，f の使用率 a_f は次式によって計算される．

$$a_f = \frac{\lambda_f}{\mu_f} = \frac{1}{\mu_f} \sum_{p \in P_f} w_p \tag{A-6}$$

任意の経路 p でのパケット損失がない，すなわち $w_p = \lambda_p$ とすると，a_f は次式によって計算される．

$$a_f = \frac{\lambda_f}{\mu_f} = \frac{1}{\mu_f} \sum_{p \in P_f} \lambda_p \tag{A-7}$$

(3)リンク帯域の使用率

リンク l を通過する経路の集合を P_l，P_l に含まれるある経路 p のパケット到着率 w_p とすると，l のパケット到着率 λ_l は次式によって計算される．

$$\lambda_l = \sum_{p \in P_l} w_p \tag{A-8}$$

l の帯域を b_l とすると，l の使用率 a_l は次式によって計算される．

$$a_l = \frac{\lambda_l}{b_l} = \frac{1}{b_l} \sum_{p \in P_l} w_p \tag{A-9}$$

任意の経路 p でのパケット損失がない，すなわち $w_p = \lambda_p$ とすると，a_l は次式によって計算される．

$$a_l = \frac{\lambda_l}{b_l} = \frac{1}{b_l} \sum_{p \in P_l} \lambda_p \tag{A-10}$$

付録2　パケット転送遅延時間の平均値の計算

経路 p の中継ノードの集合を N_p，中継リンクの集合を L_p，ルーティング処理部 r の平均処理待時間を W_r，ルーティング処理時間を μ_r，フォワーディング処理部 f の平均処理待時間を W_f，ルーティング処理時間を μ_f，及びリンク l の伝送遅延を d_l とすると，経路 p のパケット遅延時間の平均 d_p は次式によって計算される．

$$d_p = \sum_{r=1}^{N_p}\left(W_r + \frac{1}{\mu_r}\right) + \sum_{f=1}^{L_p}\left(W_f + \frac{1}{\mu_f}\right) + \sum_{l=1}^{L_p} d_l \tag{A-11}$$

ルーティング処理部及びフォワーディング処理部におけるパケット処理を各々M／M／1 でモデル化する場合，W_r及び W_fは次式によって計算される．

$$w_r = \frac{1}{\mu_r}\frac{1}{1-a_r} = \frac{1}{\mu_r - \lambda_r} \tag{A-12}$$

$$w_f = \frac{1}{\mu_f}\frac{1}{1-a_f} = \frac{1}{\mu_f - \lambda_f} \tag{A-13}$$

従って，

$$d_p = \sum_{r=1}^{N_p}\left(\frac{1}{\mu_r}\frac{1}{1-a_r} + \frac{1}{\mu_r}\right) + \sum_{f=1}^{L_p}\left(\frac{1}{\mu_f}\frac{1}{1-a_f} + \frac{1}{\mu_f}\right) + \sum_{l=1}^{L_p} d_l = \sum_{r=1}^{N_p}\frac{1}{\mu_r}\left(\frac{1}{1-a_r}+1\right) + \sum_{f=1}^{L_p}\frac{1}{\mu_f}\left(\frac{1}{1-a_f}+1\right) + \sum_{l=1}^{L_p} d_l \tag{A-14}$$

ノード s～ノード d 間経路の集合を P_{sd}，P_{sd} に含まれるある経路 p のトラフィック経路配分率を x_p とすると， s から d までのパケット遅延時間の平均 d_{sd}は次式によって計算される．

$$d_{sd} = \sum_{p \in P_{sd}} x_p d_p \tag{A-15}$$

ルーティング処理部及びフォワーディング処理部におけるパケット処理を各々M／M／1 でモデル化する場合，d_{sd}は次式によって計算される．

$$d_{sd} = \sum_{p \in P_{sd}} x_p \left\{ \sum_{r=1}^{N_p}\frac{1}{\mu_r}\left(\frac{1}{1-a_r}+1\right) + \sum_{f=1}^{L_p}\frac{1}{\mu_f}\left(\frac{1}{1-a_f}+1\right) + \sum_{l=1}^{L_p} d_l \right\} \tag{A-16}$$

ネットワーク内のノード数を N とすると，ネットワーク内のパケット遅延時間の平均値 D は次式によって計算される．

$$D = \frac{1}{E}\sum_{s}\sum_{d} d_{sd}, \quad s \neq d \tag{A-17}$$

ルーティング処理部及びフォワーディング処理部におけるパケット処理を各々M／M／1 でモデル化する場合，D は次式によって計算される．

$$D = \frac{1}{N(N-1)}\sum_{s=1}^{N}\sum_{d=1}^{N}\left\{ \sum_{p \in P_{sd}} x_p \left\{ \sum_{r=1}^{N_p}\frac{1}{\mu_r}\left(\frac{1}{1-a_r}+1\right) + \sum_{f=1}^{L_p}\frac{1}{\mu_f}\left(\frac{1}{1-a_f}+1\right) + \sum_{l=1}^{L_p} d_l \right\} \right\}, \quad s \neq d \tag{A-18}$$

付録3　最適化変数に加わる力の計算

(1)ルーティングレート配分率に加わる力

ノードサービスレート配分率 x_r に加わる力は，目的関数 V を x_r で偏微分することで得られる．

$$\frac{\partial V}{\partial x_r} = \frac{\partial}{\partial x_r}\left\{w_1 D + w_2 A_r + w_3 A_f + w_4 A_l\right\} = w_1 \frac{\partial D}{\partial x_r} + w_2 \frac{\partial A_r}{\partial x_r} + w_3 \frac{\partial A_f}{\partial x_r} + w_4 \frac{\partial A_l}{\partial x_r} \tag{A-19}$$

各項別の計算は以下のとおりである．なお A_r，A_f 及び A_l は，各々 a_r，a_f 及び a_l の最大値なので，第 2～4 項の計算 a_r，a_f 及び a_l の場合に置き換えている．

$$\frac{\partial D}{\partial x_r} = -\frac{M_r}{N(N-1)}\left(\frac{1}{\left(\mu_r - \lambda_r\right)^2} + \frac{1}{\mu_r^{\,2}}\right)\sum_{s=1}^{N}\sum_{d=1}^{N}\sum_{p\in P_{sd}} x_p, \quad s \neq d \tag{A-20}$$

$$\frac{\partial a_r}{\partial x_r} = -\frac{\lambda_r}{\mu_r^{\,2}} M_r \tag{A-21}$$

$$\frac{\partial a_f}{\partial x_r} = 0 \tag{A-22}$$

$$\frac{\partial a_l}{\partial x_r} = 0 \tag{A-23}$$

(2)フォワーディングレート配分率に加わる力

フォワーディングレート配分率 x_f に加わる力は，目的関数 V を x_f で偏微分することで得られる．

$$\frac{\partial V}{\partial x_f} = \frac{\partial}{\partial x_f}\left\{w_1 D + w_2 A_r + w_3 A_f + w_4 A_l\right\} = w_1 \frac{\partial D}{\partial x_f} + w_2 \frac{\partial A_r}{\partial x_f} + w_3 \frac{\partial A_f}{\partial x_f} + w_4 \frac{\partial A_l}{\partial x_f} \tag{A-24}$$

各項別の計算は以下のとおりである．なお A_r，A_f 及び A_l は，各々 a_r，a_f 及び a_l の最大値なので，第 2～4 項の計算 a_r，a_f 及び a_l の場合に置き換えている．

$$\frac{\partial D}{\partial x_f} = -\frac{M_f}{N(N-1)}\left(\frac{1}{\left(\mu_f - \lambda_f\right)^2} + \frac{1}{\mu_f^{\,2}}\right)\sum_{s=1}^{N}\sum_{d=1}^{N}\sum_{p\in P_{sd}} x_p, \quad s \neq d \tag{A-25}$$

$$\frac{\partial a_r}{\partial x_f} = 0 \tag{A-26}$$

$$\frac{\partial a_f}{\partial x_f} = -\frac{\lambda_f}{\mu_f^{\,2}} M_f \tag{A-27}$$

$$\frac{\partial a_l}{\partial x_f} = 0 \tag{A-28}$$

(3)リンク帯域配分率に加わる力

リンク帯域配分率 x_l に加わる力は，目的関数 V を x_l で偏微分することで得られる．

$$\frac{\partial V}{\partial x_l} = \frac{\partial}{\partial x_l}\left\{w_1 D + w_2 A_r + w_3 A_f + w_4 A_l\right\} = w_1 \frac{\partial D}{\partial x_l} + w_2 \frac{\partial U_r}{\partial x_l} + w_3 \frac{\partial U_f}{\partial x_l} + w_4 \frac{\partial U_l}{\partial x_l} \tag{A-29}$$

各項別の計算は以下のとおりである．なお A_r，A_f 及び A_l は，各々 a_r，a_f 及び a_l の最大値なので，第 2～4 項の計算 a_r，a_f 及び a_l の場合に置き換えている．

$$\frac{\partial D}{\partial x_l} = 0 \tag{A-30}$$

$$\frac{\partial a_r}{\partial x_l} = 0 \tag{A-31}$$

$$\frac{\partial a_f}{\partial x_l} = 0 \tag{A-32}$$

$$\frac{\partial a_l}{\partial x_l} = -\frac{\lambda_l}{b_l^2} B \tag{A-33}$$

(4)トラフィック経路配分率に加わる力

トラフィック経路配分率 x_p に加わる力は，目的関数 V を x_p で偏微分することで得られる．

$$\begin{aligned} \frac{\partial V}{\partial x_p} &= \frac{\partial}{\partial x_p}\left\{w_1 D + w_2 A_r + w_3 A_f + w_4 A_l\right\} \\ &= w_1 \frac{\partial D}{\partial x_p} + w_2 \frac{\partial A_r}{\partial x_p} + w_3 \frac{\partial A_f}{\partial x_p} + w_4 \frac{\partial A_l}{\partial x_p} \end{aligned} \tag{A-34}$$

各項別の計算は以下のとおりである．なお A_r，A_f 及び A_l は，各々 a_r，a_f 及び a_l の最大値なので，第 2～4 項の計算 a_r，a_f 及び a_l の場合に置き換えている．λ_{sd} は p の送受信間パケット生起率である．

$$\frac{\partial D}{\partial x_p} = \frac{1}{N(N-1)}\left\{\left(\sum_{r=1}^{N_p}\frac{1}{\mu_r}\left(\frac{1}{1-a_r}+1\right)+\sum_{f=1}^{L_p}\frac{1}{\mu_f}\left(\frac{1}{1-a_f}+1\right)+\sum_{l=1}^{L_p} d_l\right)+x_p\lambda_{sd}\left(\sum_{r=1}^{N_p}\frac{1}{\mu_r^{\,2}}\frac{1}{(1-a_r)^2}+\sum_{f=1}^{L_p}\frac{1}{\mu_f^{\,2}}\frac{1}{(1-a_f)^2}\right)\right\} \tag{A-35}$$

$$\frac{\partial a_r}{\partial x_p} = \frac{\lambda_{sd}}{\mu_r} \tag{A-36}$$

$$\frac{\partial a_f}{\partial x_p} = \frac{\lambda_{sd}}{\mu_f} \tag{A-37}$$

$$\frac{\partial a_l}{\partial x_p} = \frac{\lambda_{sd}}{b_l} \tag{A-38}$$

第4章 動的ルーティング政策

概要

動的ルーティング政策とは、複数の並列パケット伝送サーバを使ってパケットを伝送する時、遅延時間を最小にするパケット振分け政策である．この問題には、次の3つの特徴がある．

（1）従来、この問題への前提条件として、パケット到着が確率過程に基づいていると仮定されていた。ここでは、パケット到着が決定論的であると仮定する。

（2）この問題は、組合せ最適化問題である。つまり、最適化変数がとる値が整数であるような組合せ的な問題である．

（3）変数の数（これは、パケット数に相当する）が多い．

高次元アルゴリズムは、これらの特徴を持つ問題に適していると考えられる．

1. 問題の説明

以下の条件で，全パケットの遅延時間の平均値が最小となるように，パケットをサーバに割り当てるルーティング政策を求める．

（1）二つのサーバがある．

（2）サーバの処理方法は到着順である．

（3）パケットは到着後に直ちに待ち行列に入る．

（4）待ち行列の長さは無限大である．

（5）待ち行列間を移動することは出来ない．

（6）全パケットの長さと到着時間は与えられている．

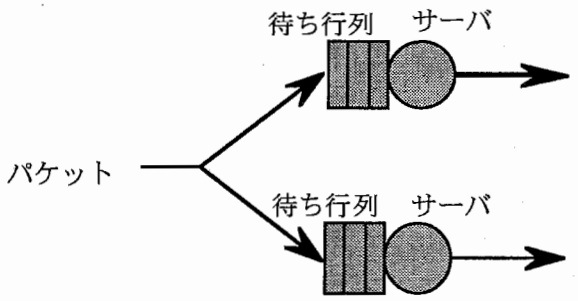


図1．並列サーバモデル

2. 最適化問題の設定

第 i 番目に到着するパケットを e_i とする．パケット e_i の長さと到着時刻を各々，x_i と t_i とする．到着するパケットの総数は n である．サーバ k の処理能力を C_k とする．ここで，パケット e_i が割り当てられるサーバ（最適化変数）を u_k と表すと，時刻 t でのサーバ k の残留処理時間は，次のように与えられる．

$$W_k(t_{i+1}) = \max(W_k(t_i) + \frac{\theta_k^i x_i}{C_k} - (t_{i+1} - t_i), 0),\ k = 1,2$$

$$\theta_k^i = \begin{cases} 1, & \text{if } u_i = k, \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

以上から，残留処理時間から，平均遅延時間（コスト関数）は次のように表すことが出来る．

$$D_n = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{2}\theta_k^i\left(W_k(t_i) + \frac{x_i}{C_k}\right) \tag{2}$$

ここで，最適化問題の制約条件は，以下の通りである．

$u_i \in \{1,2\},\ i = 1,..,n$

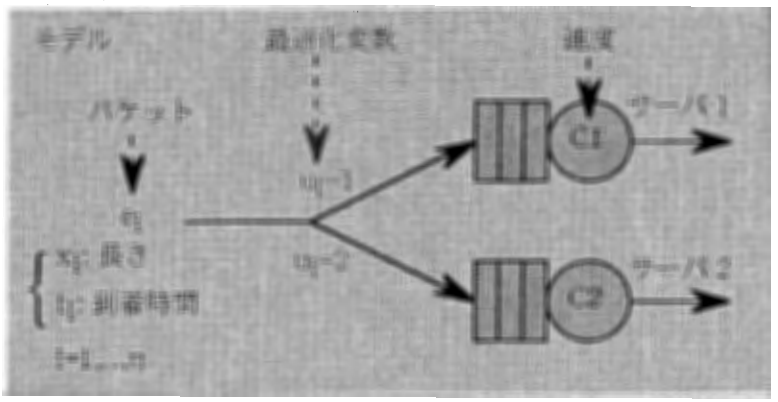


図２．定式化のための変数

3. 力学系の構成

最適化変数をバイナリの値をとる u_i から[0,1]の実数値をとる変数 θ_i で置き換えて最適化する．

$$H = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{2}\left[p_k^i - \int_0^t \kappa(t')\frac{d\theta_k^i}{dt'}dt'\right]^2 + V\left[\theta_k^i\right], \tag{3}$$

$$V\left[\theta_k^i\right] = D_n\left[\theta_k^i\right] + L\left[\theta_k^i\right] + \left(L'\left[\theta_k^i\right]\right), \tag{4}$$

$$L\left[\theta_k^i\right] = v\sum_{i=1}^{n}\left(1 - \sum_{k=1}^{2}\theta_k^i\right)^2 + v'\sum_{i=1}^{n}\sum_{k=1}^{2}\left\{-\theta_k^i\theta(\theta_k^i) + (\theta_k^i - 1)\theta(\theta_k^i - 1)\right\} \tag{5}$$

力学系の構成を以下に示す．

式(4)のL'は，θ_i をバイナリにするための関数である．L'については，後程説明する．式(5)の $\theta()$ は，ステップ関数である．

4. 運動方程式

運動方程式を以下に示す．

$$\frac{d\theta_k^i}{dt} = \frac{\partial H}{\partial p_k^i} = p_k^i - \int_0^t \kappa(t') \frac{d\theta_k^i}{dt'} dt', \tag{6}$$

$$\frac{dp_k^i}{dt} = -\frac{\partial H}{\partial \theta_k^i} \left(= -\frac{dV}{d\theta_k^i} \right), \tag{7}$$

$$\frac{d^2\theta_k^i}{dt^2} = -\frac{dV}{d\theta_k^i} - \kappa(t) \frac{d\theta_k^i}{dt} \tag{8}$$

5. プログラム

（1）はじめに，[0,1]の実数値をとる変数θ_iの最適解を求める．

（2）次に，コスト関数にL'を追加して，変数θ_iを強制的にバイナリにする．図3は，式(9)の中で，サーバ数 p＝2の場合の L'の例であるが，p>3 の場合も同様である．最適化の繰返し回数が増加する毎にαの値を少しづつ大きくする．

$$L'(\theta_k^i) = \begin{cases} \alpha\left(\frac{1}{p} - \left|\theta_k^i - \frac{1}{p}\right|\right), & 0 \le \theta_k^i \le 1 \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

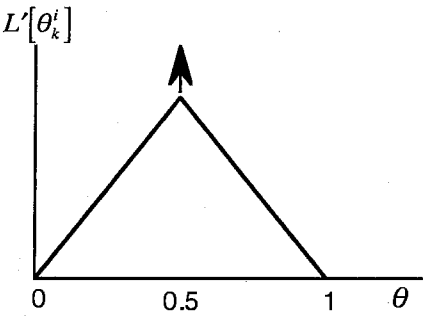


図3．関数L'の例

6. 計算結果

（1）計算時間

計算時間は，変数の数（パケット数）の増加によって，急激に増加する．表1はパケット数と計算時間（繰返し回数１００万回）の関係を表している．

表1．パケット数と計算時間

パケット数(n)	計算時間(h)
256	16.6
512	50
1024	138

（2）最適ルーティングの特徴

図4は，パケット長と割り当てられるサーバの関係を表している．図にしめすように，入力トラヒック強度Iに依存して，最適ルーティング政策が異なることが解る[1].

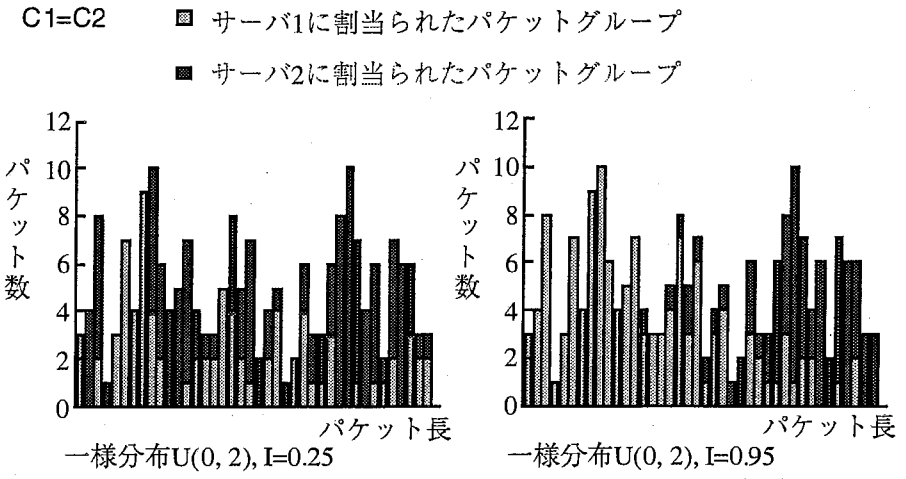


図4．パケット長と割り当てられるサーバとの関係

文献

(1) Oida et al., “Characteristics of Deterministic Optimal Routing for a Simple Traffic Control Problem,” IPCCC’99, pp. 386-392.

第5章 可変指向性アンテナのパラメータ最適化

概要

エスパアンテナと呼ばれる可変指向性アンテナを制御する．そのために与えるべきパラメータの最適値を求める問題である．

適用においては，コスト関数が解析的な関数形としては表現されず，数値解析によって求められることが特徴である．そのために運動方程式における力の計算に工夫をしている．また，混合性の工夫として運動エネルギーに相当する項に定数γを導入している．

1．問題の説明

エスパアンテナ (Electronically Steerable Passive Array Radiator antenna，電子制御導波器アレーアンテナ) はATRで考案された可変指向性アンテナである．水平方向の指向性を適応的に変化させることができる．従来のアダプティブアンテナに比べて小型で低コストにできるため，ユーザ端末用のアンテナへ搭載するための発展性も見込まれている[1]．

図1に5素子エスパアンテナの構造を示す．素子のうち1本は給電素子で，他のn本は無給電素子である．この場合は$n=4$である．接地導体板の中央に給電素子を置き，円周上に無給電素子を配置する．無給電素子それぞれには可変リアクタを装荷する．この可変リアクタはそれぞれ独立にリアクタンス値を変化させることができ，指向性パターンを制御することが可能である．

そのため，望ましい指向性パターンを得るためにはエスパアンテナをいかに制御すべきかという問題，すなわちそれぞれに与えるべきリアクタンス値を知りたいという最適化問題が生じる．

指向性はモーメント法という数値解析を通して求められる．総当りにすべてのリアクタンス値の組み合わせを調べるとすると，計算回数と無給電素子数 n について以下の関係となる．

計算回数 $\propto$ (リアクタンス値のとる範囲)^n

モーメント法1回の計算には1秒もかからないが，くり返す回数が非常に多いため，結果として計算時間が長くなってしまう．そのため高次元アルゴリズムによって短い時間で最適解を探索してゆく[2]-[4]．

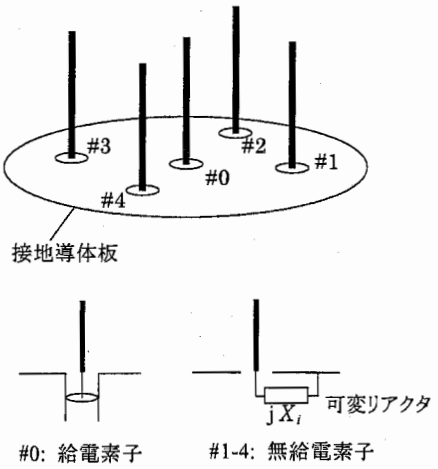


図1．5素子エスパアンテナの構造

2．最適化問題の設定

最適化したい変数q_iはリアクタンス値x_iである$(i = 1, ..., n)$．コスト関数Vは，望ましい指向性パターンを実現するような関数形を考える．図2に示すように，今回の目的とする指向性パターンは指定した方位角ϕについての利得$G(\phi)$が最大となるパターンとする．そのためコスト関数を，$V = -G(\phi)$とする．Gが最大のときVは最小値をとり，最適状態となる．なお，他の角度の方向は考慮しないため，指向性パターンが必ずしも

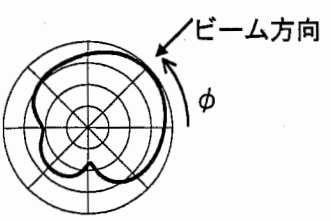


図2．目的とする指向性パターン

ϕ にピークを持つとは限らない．ここで，G は数値解析によって得られる値であるため，V が q_i の陽な関数として書くことはできないことに注意を要する．

リアクタンス値の範囲は−250Ω～＋250Ωとする．そのため拘束条件として，$q_i = \pm 250$ で反射させる．

3．力学系の構成

ハミルトニアン $H(q_i, p_i)$ は次の形とする．

$$H(q_i, p_i) = T(p_i) + V(q_i) = \frac{1}{2}\left(\sum_{i=1}^{n} {p_i}^2\right)^{\gamma} + V(q_i) \tag{1}$$

ここで，$p_i\ (i = 1, ..., n)$ は運動量であり，$T(p_i)$ は運動エネルギーに相当する項である．混合性を与える工夫として正の定数 γ を導入している．

4．運動方程式

運動方程式は式(1)のハミルトニアンから次の形となる．なお，エネルギー減衰は入れていない．

$$\text{速度：}\quad \frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} = \frac{\partial T}{\partial p_i} = \gamma \cdot p_i \left(\sum_{k=1}^{n} {p_k}^2\right)^{\gamma-1} \equiv F_i(p_1, \ldots, p_n) \tag{2}$$

$$\text{力：}\quad \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} = -\frac{\partial V}{\partial q_i} \tag{3}$$

速度は式(2)のように p_i の関数として与えられるため，直接計算ができる．一方，力については q_i の関数として陽に書くことはできず，数値計算が必要である．ここでは，偏微分を差分近似して求める．

$$-\frac{\partial V}{\partial q_i} \fallingdotseq -\frac{\Delta V_i}{\Delta q} = -\frac{V(q_1, \ldots, q_i + \Delta q, \ldots, q_n) - V(q_1, \ldots, q_i, \ldots, q_n)}{\Delta q} \tag{4}$$

力の n 個の成分それぞれについて差分近似をおこなうと，ある $q_1, ..., q_n$ の組における V と，q_i だけが $q_i + \Delta q$ へ微小変化したときの V がそれぞれの i について必要である．微小変化後の V は $i = 1, ..., n$ の全体として n 個必要である．つまり V の計算が（$n+1$）回となり，多くの計算時間が必要となる．そこで正確に求めるのは微小変化前の 1 回だけ，すなわち $V(q_1, ..., q_i, ..., q_n)$ にとどめ，他の微小変化後の n 回，すなわち $V(q_1, ..., q_i + \Delta q, ..., q_n), (i = 1, ..., n)$ は近似によって求め，計算時間の短縮を図る．近似のためには変化前に求めた値を再利用する．

まず，V を求めるための数値解析手法として用いるモーメント法を要約すると次の手順である．

(a1) アンテナを微小な素辺に分割し，その構造と装荷リアクタンス値によって決定されるインピーダンス行列 $[Z]$ を求める．

(a2) 印加電圧を $\{v\}$ とする．各素辺の電流分布 $\{I\}$ を未知数とする連立方程式 $\{v\} = [Z]\{I\}$ を組み立てて解き，$\{I\} = [Z]^{-1}\{v\}$ を求める．

(a3) 電流分布 $\{I\}$ をもとに遠方の電界計算をし，利得 G を求める．(なお，本稿では $V = -G$ である．)

このうち計算時間を占める主要な部分は連立方程式を解く (a2) である．すなわち逆行列 $[Z]^{-1}$ を求めるところに最も時間がかかっている．

次に，q_i が $q_i + \Delta q$ へ微小変化したときの $V(q_1, ..., q_i + \Delta q, ..., q_n)$については次の手順とする．

(a'1) 装荷リアクタンスの一つである q_i の微小変化によって $[Z]$ は $[Z+\Delta Z_i]$ へ変化するため，$[\Delta Z_i]$ を求める．$[\Delta Z_i]$ は i と Δq によって定まる微小な行列である．

(a'2) 連立方程式 は $\{v\}=[Z+\Delta Zi]\{I+\Delta Ii\}$ となる．ただし $\{I+\Delta I_i\}$ は $\{I\}$ が微小変化したものであり，$\{v\}$ は微小変化の前後で不変である．これを解くためには逆行列 $[Z+\Delta Zi]^{-1}$ を直接計算することなく，$[Z+\Delta Z_i]^{-1} \fallingdotseq [Z]^{-1}-[Z]^{-1}[\Delta Z_i]\,[Z]^{-1}$ なる近似を用いる．すると連立方程式の解の近似値として $\{I+\Delta I_i\} \fallingdotseq \{I\}-[Z]^{-1}[\Delta Z_i]\{I\}$ を得る．

(a'3) $\{I+\Delta I_i\}$ の近似値から利得を求め，V の近似値とする．

(a'1) における$[\Delta Z_i]$ の計算は簡単なものである．また (a'2) では $\{I\}, [Z]^{-1}$ は微小変化前の計算ですでに求めた値を再利用できる．そして単純な乗算，減算であるから逆行列の計算に比べれば時間がかからない．

以上の手順による近似を用い，式(3)の力を求める計算時間を短縮する．

5．プログラム

計算プログラムでは Verlet 法によって各時刻の q_i, p_i を求めていく．ただし，通常の Verlet 法では運動方程式から p_i を消去して q_i についてだけの式とし計算するが，ここでは γ を導入したために p_i の消去ができない．そこで以下の手順で計算を進める．

(b1) 式(3)により力 $\dfrac{dp_i}{dt}=-\dfrac{\Delta V_i}{\Delta q}$ を求める．

(b2) 加速度 $\dfrac{d^2q_i}{dt^2}=\dfrac{dF_i(p_i)}{dt}\equiv F_i'\left(p_i,\dfrac{dp_i}{dt}\right)$ を式(2)から次のように求める．

$$F_i'\left(p_i,\frac{dp_i}{dt}\right)=\gamma\left\{\frac{dp_i}{dt}\left(\sum_{k=1}^{n}{p_k}^2\right)^{\gamma-1}+2(\gamma-1)p_i\left(\sum_{k=1}^{n}{p_k}^2\right)^{\gamma-2}\sum_{k=1}^{n}\left(p_k\frac{dp_k}{dt}\right)\right\} \tag{5}$$

(b3) 次の時刻の q_i, p_i をそれぞれ次の式により求める．

$$q_i(t+\Delta t)=2q_i(t)-q_i(t-\Delta t)+\frac{d^2q_i(t)}{dt^2}\cdot(\Delta t)^2 \tag{6}$$

$$p_i(t+\Delta t)=p_i(t-\Delta t)+2\frac{dp_i(t)}{dt}\cdot(\Delta t) \tag{7}$$

この方法では，通常の Verlet 法よりも近似精度が劣るが，Runge-Kutta 法に比べて力の計算が少なくて済む利点がある．その誤差については，計算中において H がほぼ一定となることを確認すれば問題ない．ここでは揺れの振幅が初期値の 10% 未満となるようにした．

6．計算結果

ϕ を 0° から 45° まで 5° 刻みに変化させ，それぞれについて最適化をおこなう．素子の配置の対称性から，この角度の範囲について調べれば十分である．運動方程式の計算回数，すなわち最適化変数を更新する回数は 10,000 回とし，その間にコスト関数が最小だったところを最適値としている．q_i, p_i の初期値は事前に概算して与え，γは 0.3 と定める．

結果として得られる最適リアクタンス値を図 3 に示す．比較のために，モンテカルロ法 50,000 回の

計算により得たリアクタンス値も示している．モンテカルロ法はランダムにリアクタンス値を与えてくり返す方法である．どちらもほぼ同様の値であり，高次元アルゴリズムによって正しく最適解が探索できたとみなすことができる．計算結果として生じる最大利得を図4に示す．全角度に対して 9.1 dBi ± 0.5 dB の高い値を得ている．また，この図には比較としてモンテカルロ法 10,000 回および 50,000 回の計算による最大利得も示している．高次元アルゴリズムでは，同じ回数である 10,000 回のモンテカルロ法よりも高い利得が得られ，5倍の回数である 50,000 回のモンテカルロ法とほぼ同等の利得が得られる．図3，図4の示す結果から，高次元アルゴリズムではモンテカルロ法に比べて 1/5 の計算回数で同等の成果を挙げることがわかる．関連する放射パターンの例を図5に示す．いずれも目的通りに，指定した角度において高い利得となるパターンを得ている．

使用計算機は HP ワークステーション C360 (CPU: PA-8500, 367MHz)である．この計算機および作成したプログラムでは，第4節で述べた近似によって V を求めるための計算時間は近似しない場合に比べておよそ 1/6 であった．運動方程式の1ステップあたりでは $(n+1)$ 回 の V の計算のうち n 回分を近似によって求めるため，およそ $(n/6+1)/(n+1)$ の計算時間となり，$n=4$ の場合は約 1/3 である．一つの ϕ についての最適化計算 10,000 回に要した時間は V を近似せずに約 120 分，近似を用いて約 40 分である．

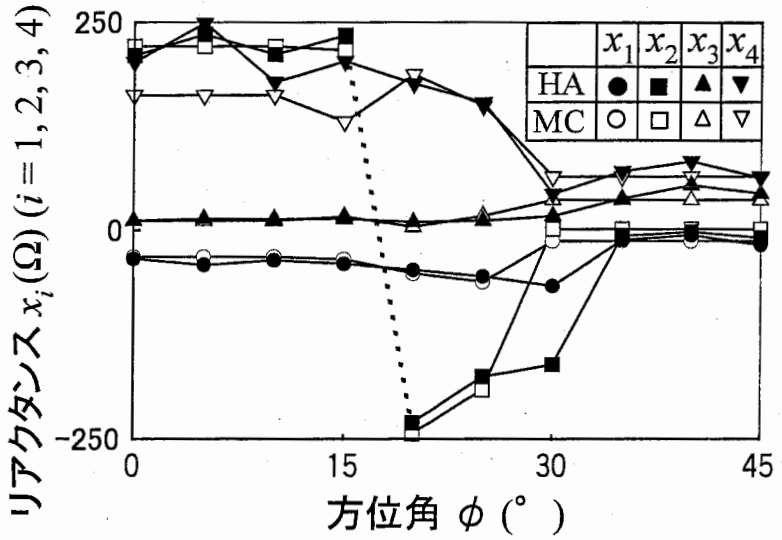


図3．方位角 ϕ において最大利得を与えるリアクタンス値
HA: 高次元アルゴリズム, MC: モンテカルロ法

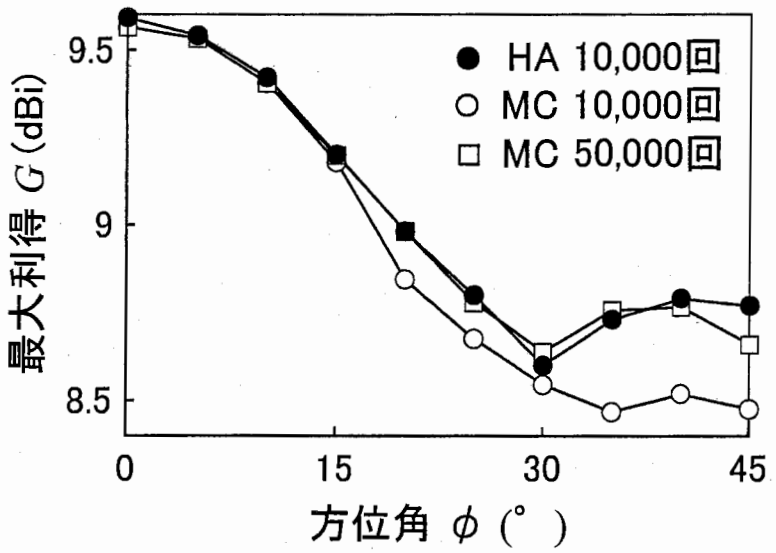


図4．方位角 ϕ に対する最大利得の計算結果
HA: 高次元アルゴリズム, MC: モンテカルロ法

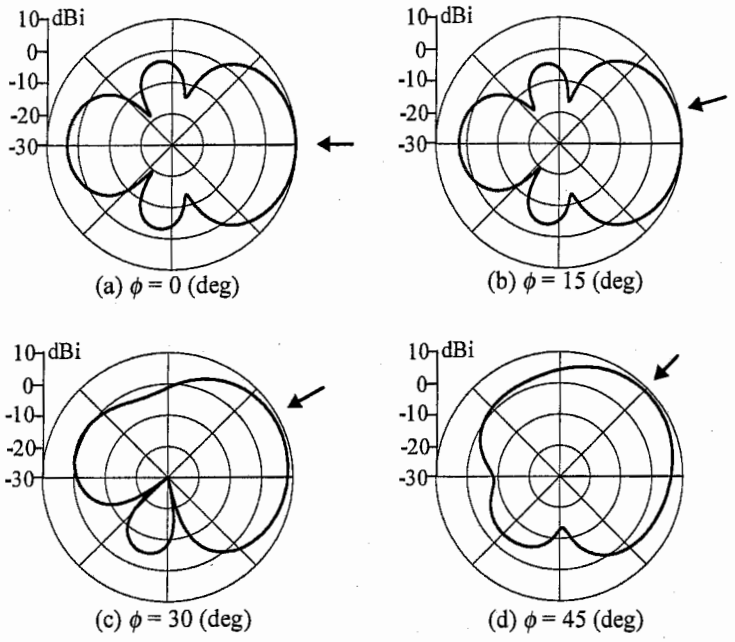


図５．方位角 ϕ において最大利得を得るように最適化した放射パターン

文献

(1) T. Ohira and K. Gyoda, "Electronically steerable passive array radiator antennas for low-cost analog adaptive beamforming", Proc. IEEE International Conf. Phased Array Syst. Tech. 2000, MP-1-2, Dana Point, May 2000.

(2) 小松崎，斎藤，行田，大平，"高次元アルゴリズムによるESPARアンテナのリアクタンス最適化"，信学技報，A・P2000-13, SAT2000-4, pp.17-22

(3) 小松崎，斎藤，行田，大平，"高次元アルゴリズムによるエスパアンテナのリアクタンス最適化"，信学会ソサイエティ大会2000, B-116.

(4) A. Komatsuzaki, S. Saito, K. Gyoda, and T. Ohira, "Hamiltonian approach to Reactance Optimization in ESPAR Antennas", Asia Pacific Microwave Conference 2000, No. 352.

第 6 章 JPEG 量子化テーブルの最適化

概要

本章では，静止画像の圧縮符号化方式の一つである JPEG 符号化形式において，圧縮率と画像の品質を制御するためのパラメータである量子化テーブルに対して，高次元アルゴリズムを利用して最適化する方法について述べる．

高次元アルゴリズムとしては，(1) JPEG 量子化テーブルは整数の値の組であり，(被)最適化変数が整数であること，(2)JPEG 量子化テーブルの値が 1 から 255 までの整数に制約されるため，最適化のプログラム実装上，反射処理(強制的な境界処理)を行っていること，(3)画像の圧縮率と画像の品質を用いて，コスト関数を構成していることに特徴がある．

1． 問題の説明

1.1. JPEG 画像圧縮符号化形式． JPEG 画像圧縮方式は，国際標準化団体の JPEG(Joint Photographic Coding Experts Group)が規定したカラー静止画像のための圧縮形式である．インターネットを通した画像データ通信やデジタル・カメラなどでの静止画像データの記録では，ベースライン・システム注(1)(脚注は本章末を参照されたい)に分類される JPEG 画像圧縮符号化形式[1] (以下，単に JPEG と略すこともある．)が主に利用される．この JPEG 画像圧縮符号化形式は，画像の品質の劣化を伴う圧縮符号化方式であるが，オリジナルの画像(原画像)のデータ・サイズを 16 分の 1～24 分の 1 に圧縮しても，視覚的な劣化を伴わないと言われている．

一般的に，画像圧縮符号化形式では，画像の圧縮率と画像の品質は相反する．すなわち，画像の圧縮率を上げる(符号化された画像のデータ・サイズを小さくする)ほど画像の品質は低下する．逆に，画像の圧縮率を下げる(符号化された画像のデータ・サイズを大きくする)ほど画像の品質は向上する．つまり，画像の圧縮率を上げれば，通信コストや画像保存のためのディスク・スペースをさらに削減することが可能であるが，画像の品質は劣化する．そこで，画像の視覚的な劣化を伴うことなく，圧縮率を上げる JPEG 画像圧縮符号化形式のパラメータ(ここでは JPEG 量子化テーブル注(2))を求めること(最適化)が問題となる．

1.2. JPEG 符号化処理/復号化処理と JPEG 量子化テーブル． 図 1 の上段および下段は，それぞれ符号器および復号器の処理の流れを示す[2]~[5]．符号器では，符号化される画像を入力し，色変換，画素の間引き，DCT(Discrete Cosine Transform:離散コサイン変換)，量子化およびハフマン符号化の過程を経て，JPEG データ・ストリームを生成する．一方，復号器では，JPEG データ・ストリームを入力し，ハフマン復号化，逆量子化，IDCT(Inverse DCT)，画素の補間および色変換の過程を経て，復号画像を生成する．前述したように，この復号画像は，原画像，すなわちもとの符号化される画像に対して劣化を伴っている．

ここで，符号器の量子化および復号器の逆量子化の過程で JPEG 量子化テーブルと呼ばれる量子化ステップのパラメータの組が用いられる．JPEG 画像圧縮方式では，この量子化テーブルの値を変更することによって，画像(JPEG データ・ストリーム)の圧縮率と(復号)画像の品質とを制御することができる．

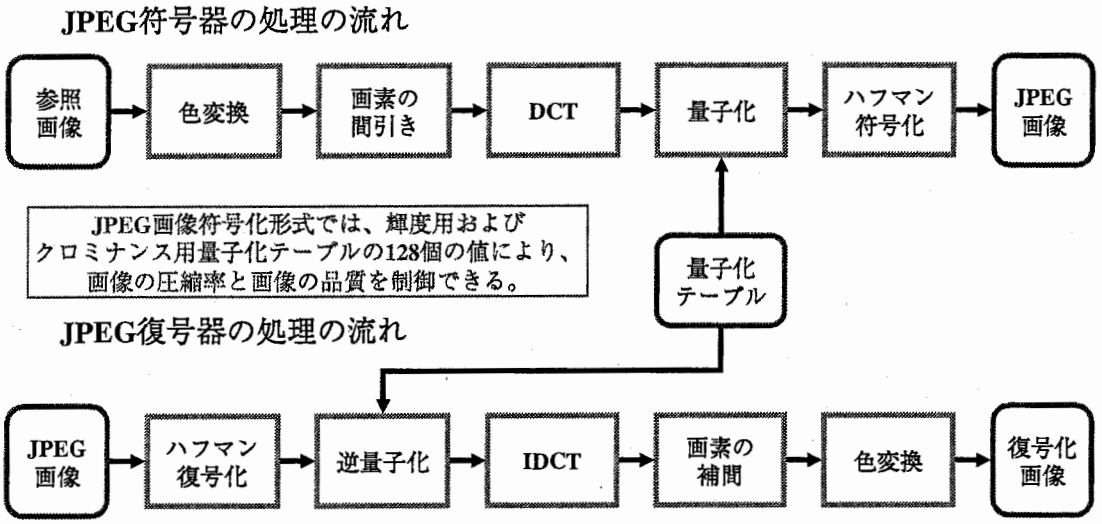


図 1. JPEG 符号化処理と復号化処理

2. 最適化問題の設定

2.1. 量子化テーブルの値と仮想粒子の位置. JPEG 量子化テーブルは，輝度用量子化テーブルとクロミナンス用量子化テーブル[注(3)]の二つに分けることができる．それぞれ輝度に対する 64 個の DCT 係数および言わば色成分であるクロミナンスに対する 64 個の DCT 係数をどのように量子化するかの量子化ステップ[注(4)]を保持している．量子化ステップは，1 から 255 までの値で設定され，単一の画像を符号化する場合，量子化テーブルの設定方法は 255^{128} 個の場合の数がある．圧縮率と画像の品質によりコスト関数を構成し，高次元アルゴリズムを使って，これらの JPEG 量子化テーブルの値を仮想的な粒子の位置とみなして最適化を試みる[(6)~(11)]．

つまり，JPEG 量子化テーブルの各要素，すなわち量子化ステップを仮想的な粒子の各次元における位置とみなして，その仮想粒子の運動を追跡することによって，最適な量子化テーブルを得るように高次元アルゴリズムを実装した．具体的には，128 個の量子化テーブルの要素を仮想粒子の 128 次元の位置に対応させる．高次元アルゴリズムのコスト関数 V は，画像の圧縮率を表現するレート[注(5)]と画像の品質を表現する SNR で構成し，各コスト関数における二つの仮想粒子の運動を追跡する．つまり，それらのコスト関数の下で，128 次元運動をする 2 個の仮想粒子に対応するそれぞれ 128 次元の運動量を考える．すなわち，2 個の状態空間を考え，そこでの仮想粒子の運動の軌跡を追跡する(図 2 参照)．

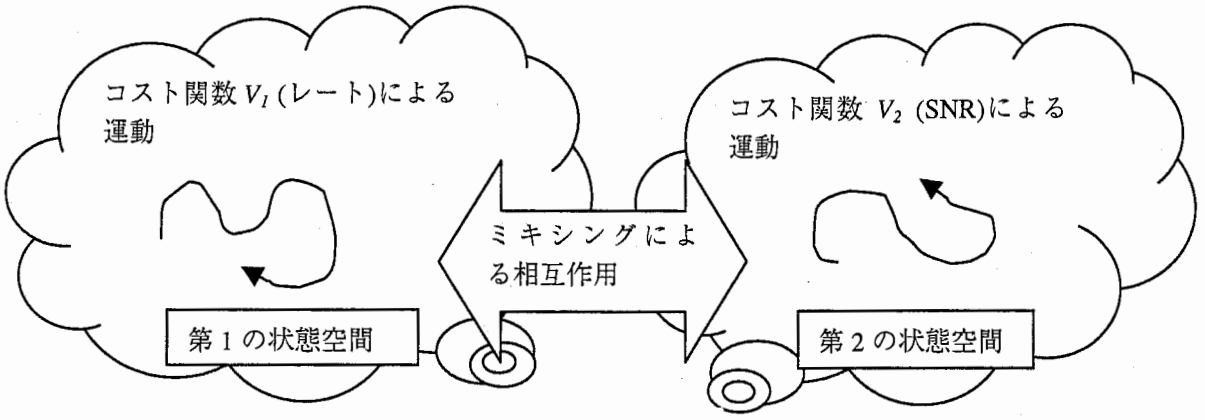


図 2. 二つのコスト関数

2.2. **コスト関数.** 圧縮率の尺度については, 1 画素あたりのビット数で表現したレート計量結果を用いた. 一方, 画像品質の尺度については, 画像における信号対雑音比(Signal to Noise Ratio: SNR)を用いた[注(6)]. SNR で表現されるコスト関数 V_2 は, 各 RGB 成分の SNR(SNR_r, SNR_g および SNR_b)の平均値とした(式(1)参照).

$$V_2 = -\frac{SNR_r + SNR_g + SNR_b}{3} \tag{1}$$

3. 力学系の構成

3.1. **ハミルトニアンの構成.** 高次元アルゴリズムを構成するハミルトニアン H は, 式(2)で表現されるものを用いた. ハミルトニアンは仮想粒子の運動を規定する. 次の節では, ハミルトニアンからプログラムで使用する運動方程式(反復公式)を導出する. γ_1 および γ_2 で表現されるシステム・パラメータは, 共に 0.5 として高次元アルゴリズムを実装した. 式(2)において, m は仮想的な粒子の質量を意味する定数[注(7)], V_i は i 番目の状態空間に対応するコスト関数を示す. 式(2)においては, i 番目の状態空間の仮想的な粒子の位置および運動量の第 j 次元の要素は, それぞれ $q_{i,j}$ および $p_{i,j}$ で表現される. 以下の式でも同様の記述方法を用いる.

$$\begin{aligned} H = & \frac{1}{2m}\left(p_{1,1}^2 + p_{1,2}^2 + \Lambda + p_{1,128}^2\right)^{\gamma_1} \\ & + \frac{1}{2m}\left(p_{2,1}^2 + p_{2,2}^2 + \Lambda + p_{2,128}^2\right)^{\gamma_2} \\ & + V_1(q_{1,1}, q_{1,2}, \Lambda, q_{1,128}) \\ & + V_2(q_{2,1}, q_{2,2}, \Lambda, q_{2,128}) \\ & + M(p_{1,1}, p_{2,1}, \Lambda, p_{1,128}, p_{2,128}) \end{aligned} \tag{2}$$

式(2)において, コスト関数 V_1 は圧縮率(レート)の尺度に対応する. 一方, コスト関数 V_2 は画像品質の尺度(SNR)に対応する.

3.2. **ミキシング.** ここで, ミキシング M の項を式(3)のように定義する.

$$\begin{aligned} M(p_{1,1}, p_{2,1}, \Lambda, p_{1,128}, p_{2,128}) &= \sum_{j=1}^{128} p_{1,j}p_{2,j} \\ &= p_{1,1}p_{2,1} + p_{1,2}p_{2,2} + \Lambda + p_{1,128}p_{2,128} \end{aligned} \tag{3}$$

概念的には, 式(3)のミキシング M を通して, 二つの状態空間の仮想的な粒子は, 相互にエネルギーを交換し, 各粒子の運動に影響を与えると考えられる.

4. 運動方程式

前述の各状態空間における仮想粒子の位置と運動量を計算する計算過程に対応する反復公式を導出する.

まず, 式(2)のハミルトニアン H に基づく, 式(4)の二つの連立偏微分方程式で表現される 128 次元の 2 個の状態空間を考える.

$$\begin{cases} \dfrac{dp_{1,j}}{dt} = -\dfrac{\partial H}{\partial q_{1,j}} \\ \dfrac{dq_{1,j}}{dt} = \dfrac{\partial H}{\partial p_{1,j}} \end{cases} \qquad \begin{cases} \dfrac{dp_{2,j}}{dt} = -\dfrac{\partial H}{\partial q_{2,j}} \\ \dfrac{dq_{2,j}}{dt} = \dfrac{\partial H}{\partial p_{2,j}} \end{cases} \tag{4}$$

ここで, 一番目の仮想粒子に対応する状態空間について, 式(4)から Δt を数値計算の刻み幅とすることによって, 式(5)および式(6)で表現される反復公式を得る[注(8)].

$$p_{1,j(t+\Delta t)} = p_{1,j(t)} - \frac{\partial V_{1(q_{1,1},\ q_{1,2},\ \Lambda,\ q_{1,128})}}{\partial q_{1,j}}\Delta t \tag{5}$$

$$q_{1,j(t+\Delta t)} = q_{1,j(t)} + \frac{\gamma_1 p_{1,j}}{m}\left(p_{1,1}^2 + p_{1,2}^2 + \Lambda + p_{1,128}^2\right)^{\gamma_1 - 1}\Delta t + p_{2,j}\Delta t \tag{6}$$

これらの連立した反復公式は, 第 2 の状態空間に対しても同様に成立する.

5. プログラム

5.1. 概念的システム構成. 図 3 は, JPEG 量子化テーブル最適化システムのシステム構成図を示す[注(9)]. 同一の輝度用およびクロミナンス用量子化テーブルが JPEG 符号器および復号器に参照される. JPEG 符号器は, 試験画像を入力し, JPEG データ・ストリームを出力する. JPEG 復号器は, JPEG データ・ストリームを入力し, 復号化画像を出力する[注(10)]. JPEG データ・ストリームを用いて圧縮率(レート)を算出する. また, 試験画像および復号化画像を用いて画像品質を算出する. 算出された圧縮率および画像品質は高次元アルゴリズムのコスト関数の値を算出するために用いられる. 図 3 の高次元アルゴリズム部分は, コスト関数の値を算出し, その値によって量子化テーブルの値を変更する. その変更された量子化テーブルを用いて符号化および復号化が行われる. このように処理が繰り返される.

今回の実装においては, 反復ごとの JPEG 量子化テーブルの値はすべてファイルとして記録し, 既定の反復回数終了後に, 適切な JPEG 量子化テーブルの値を検索することによって, 最適化された JPEG 量子化テーブルを得るようにした.

また, JPEG 符号化処理ないし復号化処理の際は, 量子化テーブルの値は 1 から 255 までの整数である必要があるため, その前後において, 浮動小数点数から整数へのキャスティング(型変換)による整数化, および整数から浮動小数点数へのキャスティングの処理を行う.

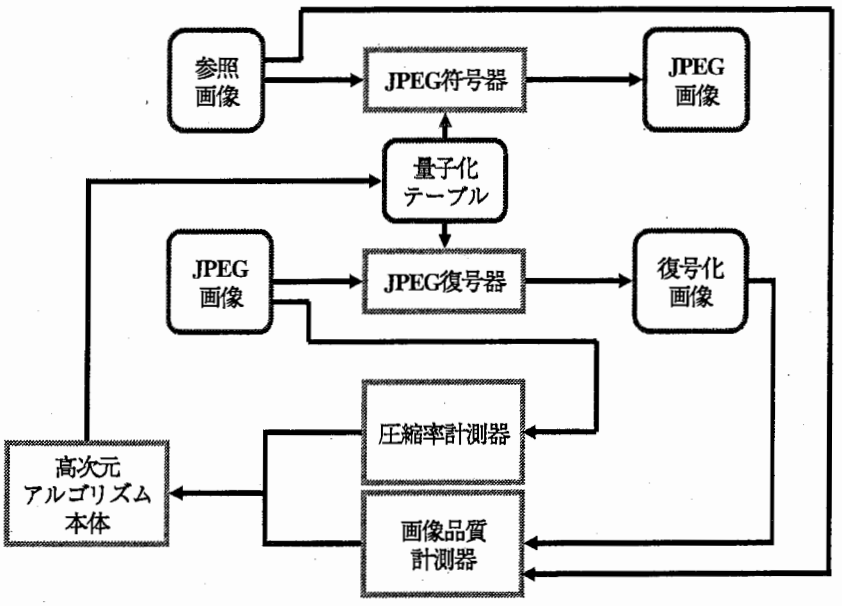


図 3. システム構成図

5.2. 処理の流れ. 図 4 は, フローチャートであり, 左部分は主な処理の流れを示す. 右部分は, メイン・ループの各処理を示す. 主な処理の流れは以下のステップから成る.

(S1) 初期化処理
　パラメータの入力, 運動量および位置の各初期値の設定などを行う.
(S2) メイン・ループ
　反復公式を実行する.
(S3) 終了処理
　メモリの解放などを行う.

メイン・ループは以下のステップから成る.

(S2-1) 終了判定

所望の回数反復を行った否かを判定し, 継続する場合, 次のステップ(S2-2)から(S2-7)までを繰り返す.

(S2-2) 結果の保存

中間結果をファイルに記録する. なお, 最適値は結果を保存したファイルを検索して見つけ出す.

(S2-3) 第 1 の状態空間の運動量の計算(5.3.節参照)

(S2-4) 第 2 の状態空間の運動量の計算(5.3.節参照)

(S2-5) 第 1 の状態空間の位置の計算(5.4.節参照)

(S2-6) 第 2 の状態空間の位置の計算(5.4.節参照)

(S2-7) 更新処理

各変数の更新処理を行う.

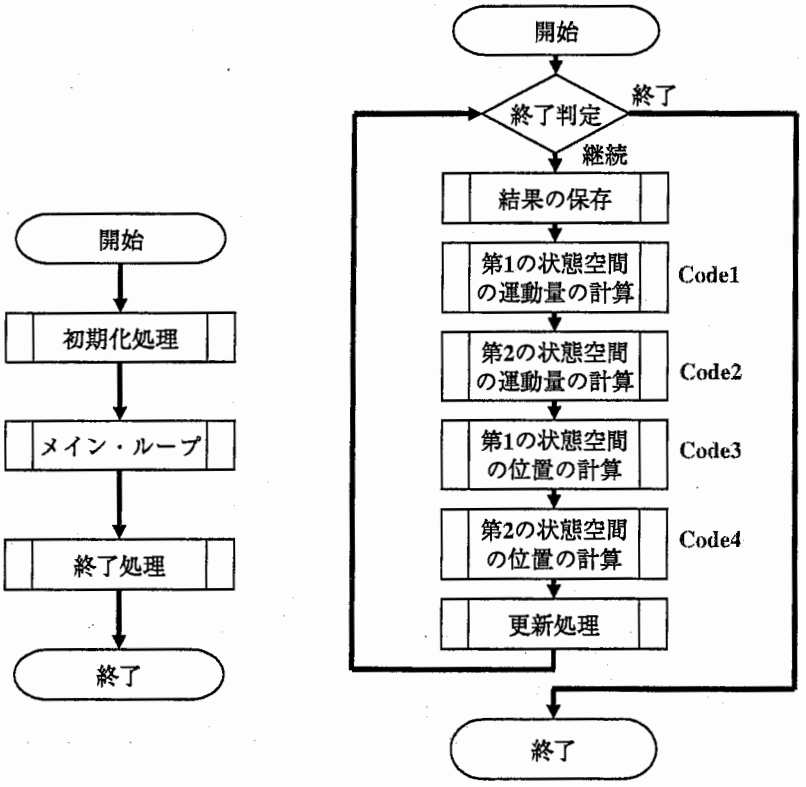


図 4. フローチャート

5.3. 運動量の計算. Code1 および Code2 は, それぞれ第 1 および第 2 の状態空間の運動量を計算するためのプログラム・コードの抜粋である. Code1 は, 式(5)の計算に対応する.

Code1

```
/* 量子化テーブル q1 の準備 */
encoder (...);// 符号化
rate = 0.0;
rate = calcRate (bmiHeader); // レートの計測
v1_cur = rate;        // コスト関数 1
for (i = 0; i<BLOCK_SIZE_2; i++) {
                for (j = 0; j<BLOCK_SIZE_2; j++) {
                                q1tmp[j] = q1[j];
                }
                q1tmp[i] = q1[i] + delta_q;
                /* q1tmp[i]の範囲検査 */
                /* 量子化テーブル(最小変位を加えたもの)の準備 */
                encoder (...);   // 符号化
                rate = 0.0;
                rate = calcRate (bmiHeader); // レートの計測
                v1_tmp = rate;        // コスト関数 1
                p1new[i] = p1[i] - (v1_tmp - v1_cur) * dt;
}
```

Code2

```
/* 量子化テーブル q2 の準備 */
encoder (...); // 符号化
decoder (...); // 復号化
snrR = 0.0; snrG = 0.0; snrB = 0.0;
bmpRGBSNR (...); // 各 RGB の SNR 計測
v2_cur = 0.0;
v2_cur = - (snrR + snrG + snrB)/3.0; // コスト関数 2
for (i = 0; i<BLOCK_SIZE_2; i++) {
                for (j = 0; j<BLOCK_SIZE_2; j++) {
                                q2tmp[j] = q2[j];
                }
                q2tmp[i] = q2[i] + delta_q;
                /* q2tmp[i]の範囲検査 */
                /* 量子化テーブルの準備(微小変位を加えたもの) */
                encoder (...); // 符号化
                decoder (...); // 復号化
                snrR = 0.0; snrG = 0.0; snrB = 0.0;
                bmpRGBSNR (...); // 各 RGB の SNR 計測
                v2_tmp = 0.0;
                v2_tmp = - (snrR + snrG + snrB)/3.0; // コスト関数 2
                p2new[i] = p2[i] - (v2_tmp - v2_cur) * dt;
}
```

Code の解説 1

関数
- encoder () //JPEG 符号化関数
- decoder () //JPEG 復号化関数
- calcRate () // レートを計測する関数
- bmpRGBSNR () // SNR を計測する関数

変数
- rate // レート
- snrR, snrG, snrB // 各 RGB 成分の SNR
- i, j // 量子化テーブルのインデックス
- bmpHeader // 画像の情報
- v1_cur // コスト関数 1
- v1_tmp // コスト関数 1(偏微分計算用)
- v2_cur // コスト関数 2
- v2_tmp // コスト関数 2(偏微分計算用)
- q1[], q2[] // 位置(量子化テーブル)の配列
- q1tmp[], q2tmp[] // 位置(量子化テーブル)の配列
- p1[], p2[] // 更新前の運動量
- p1new[], p2new[] // 更新後の運動量

マクロ
- BLOCK_SIZE_2 // 量子化テーブルの個数(128)

定数
- delta_q // 最小変位(1.0)
- dt // 時間の刻み(1.0)

Code1 は, 第 1 の状態空間の運動量を計算する. つまり, レートのコスト関数により仮想粒子の運動量を計算する. まず, 量子化テーブル q1[]を前もって準備する. 次に, レートを求めるために encoder ()関数を使って JPEG 符号化を行う. レートは calcRate ()関数により算出する. 式(5)右辺第 2 項の偏微分を計算するために, 前もって準備した量子化テーブルでのコスト関数(レート)の値 v1_cur とある次元 i の量子化テーブルの値(量子化ステップ)を微小変位 delta_q だけ加えた値 q1tmp[i]を使って, JPEG 符号化し, その量子化テーブルでのレートを計測する. その計測結果を v1_tmp に保持する. 続いて, v1_cur および v1_tmp の値を使って, 式(5)を計算する.

Code2 は, 第 2 の状態空間の運動量を計算する. つまり, SNR のコスト関数により仮想粒子の運動量を計算する. まず, 量子化テーブル q2[]を前もって準備する. 次に, SNR を求めるために encoder ()関数および decoder ()関数を使って JPEG 符号化および JPEG 復号化を行う. SNR は bmpRGBSNR ()関数により算出する. 式(5)右辺第 2 項の偏微分に対応する計算するために, 前もって準備した量子化テーブルでのコスト関数(SNR)の値 v2_cur とある次元 i の量子化テーブルの値(量子化ステップ)を微小変位 delta_q だけ加えた値 q2tmp[i]を使って, JPEG 符号化および JPEG 復号化し, その量子化テーブルでの SNR を計測する. その計測結果を v2_tmp に保持する. 続いて, v2_cur および v2_tmp の値を使って, 式(5)に対応する計算する.

5.4. 位置の計算．Code3 および Code4 は，それぞれ第 1 および第 2 の状態空間の運動量を計算するためのプログラム・コードの抜粋である. Code3 は式(6)の計算に対応する.

Code3

```
sum_p = 0.0;
for (i = 0; i<BLOCK_SIZE_2; i++) {
        sum_p += p1[i] * p1[i];
}
sg = 0.0;
gsd = 0.0;
sg = pow (sum_p, g1m1);
gsd = gamma1 * sg * dt;
for (i = 0; i<BLOCK_SIZE_2; i++) {
        mixing = 0.0;
        mixing = p2[i];
        rq_tmp = (double)q1[i]
                    + gsd * p1[i] + mixing * dt;
        /* 境界の処理(反射処理) */
        /* rq_tmp の範囲検査と
              q1new[i]への代入 */
}
```

Code の解説 2

関数

- pow () // べき乗を計算する関数

変数

- sum_p, sg, gsd, mixing, g1m1, g2m1 // ミキシングを計算するための変数
- rq_tmp // 位置
- q1new[], q2new[] // 更新後の位置

定数

- gamma1, gamma2 // システム・パラメータ γ_1 および γ_2 (共に 0.5)

Code4

```
sum_p = 0.0;
for (i = 0; i<BLOCK_SIZE_2; i++) {
        sum_p += p2[i] * p2[i];
}
sg = 0.0;
gsd = 0.0;
sg = pow (sum_p, g2m1);
gsd = gamma2 * sg * dt;
for (i = 0; i<BLOCK_SIZE_2; i++) {
        mixing = 0.0;
        mixing = p1[i];
        rq_tmp = (double)q2[i]
                    + gsd * p2[i] + mixing * dt;
        /* 境界の処理(反射処理) */
        /* rq_tmp の範囲検査と
             q2new[i]への代入 */
}
```

Code3 は，第 1 の状態空間の位置を計算する．つまり，レートのコスト関数により仮想粒子の位置を計算する．まず，式(6)右辺第 2 項の $p_{1,j}$ の 2 乗和を計算する. gsd 変数は式(6)右辺第 2 項を計算する．次に, p2[]および p1[]の値を使って rq_tmp 変数にミキシングが作用した値を計算する. rq_tmp 変数の値に対して境界の処理(反射処理)を行い，範囲検査をした後, q1new[]に代入する.

Code4 は，第 2 の状態空間の位置を計算する．つまり, SNR のコスト関数により仮想粒子の位置を計算する．まず，式(6)右辺第 2 項に対応する $p_{2,j}$ の 2 乗和を計算する．gsd 変数は式(6)右辺第 2 項に対応する値を計算する．次に, p1[]および p2[]の値を使って rq_tmp 変数にミキシングが作用した値を計算する. rq_tmp 変数の値に対して境界の処理(反射処理)を行い，範囲検査をした後, q2new[]に代入する.

5.5. 境界の処理(反射処理). Code5 は，量子化テーブルの値を適正にするための境界の処理(反射処理と呼ぶこともある.)である．反射処理で量子化テーブルの値を 1 から 255 までの値にする場合，運動量の符号を反対にすることによって仮想粒子の移動方向を反対にする．mapFrom1To255 ()関数は，改めて rq_tmp に保存されている量子化テーブルの値を 1 から 255 までの値に正当化する．

Code5

```
#define DQMIN           1.0
#define DQMAX           255.0
double          reflection = 1.0;

if ( DQMAX<rq_tmp ) {
                rq_tmp = DQMAX - (rq_tmp - DQMAX);
                p1new[i] = - p1new[i] * reflection;
}
if ( DQMIN>rq_tmp ) {
                rq_tmp = DQMIN + (DQMIN - rq_tmp);
                p1new[i] = - p1new[i] * reflection;
}
mapFrom1To255Double (&rq_tmp);
```

コードの解説 3

関数
- mapFrom1To255Double () // 量子化テーブルの値を正当化する関数

マクロ
- DQMIN // 量子化ステップの最小値 1.0
- DQMAX // 量子化ステップの最大値 255.0

変数
- reflection // 反射時の減衰率 (1.0)

5.6. 計算機実験.

(a) 参照画像

参照画像(最適化において参照する原画像)として，映像メディア学会(旧テレビジョン学会)が編成したディジタル標準画像を用いた.

(b) パラメータ

表 1 のパラメータで量子化テーブルの最適化を試みた.

表 1. パラメータ

指数	$\gamma_1=\gamma_2=0.5$
質量	$m=1$
時間の刻み	$\Delta t=1.0$

(c) 反復回数

反復回数は，各参照画像に対して 10000 回とした.

(d) 減衰

今回の高次元アルゴリズムの実装では，減衰は行っていない[注(11)].

(e) 初期値

表 2 および表 3 は，それぞれ第 1 の状態空間および第 2 の状態空間における仮想粒子の位置に対する初期値をそれぞれ示す[注(12)]．左の表が輝度用 JPEG 量子化テーブルに対応する．また，右の表がクロミナンス用 JPEG 量子化テーブルに対応する．

表 2. 第 1 の状態空間における仮想粒子の位置

4	2	2	4	6	10	12	15
3	3	3	4	6	14	15	13
3	3	4	6	10	14	17	14
3	4	5	7	12	21	20	15
4	5	9	14	17	27	25	19
6	8	13	16	20	26	28	23
12	16	19	21	25	30	30	25
18	23	23	24	28	25	25	24

4	4	6	11	24	24	24	24
4	5	6	16	24	24	24	24
6	6	14	24	24	24	24	24
11	16	24	24	24	24	24	24
24	24	24	24	24	24	24	24
24	24	24	24	24	24	24	24
24	24	24	24	24	24	24	24
24	24	24	24	24	24	24	24

表 3. 第 2 の状態空間における仮想粒子の位置

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

6. 計算結果

6.1. **計算速度.** 表 4 のもとにおける反復 1 回あたりに要する計算時間は, 35.23 秒である.

表 4. 計算速度の計測のための計算機環境

CPU	Intel Mobile Pentium III 500MHz
メモリー	328MB
OS	Microsoft Windows 2000
開発環境	cygwin-1.1.12
コンパイラ	gcc-2.95.2 (-O4 フラグの最適化)

6.2. **最適解発見までの反復回数.** 表 5 は, 最適解発見までの反復回数を示す.

表 5. 反復回数

参照画像	試行回数(回)	採用回数(回)
ヘアバンドの女性	10000	21

6.3. **得られた量子化テーブルの例.** 表 6 は, 参照画像として「ヘアバンドの女性」を用いた場合の得られた量子化テーブルである[注(13)].

表 6. 得られた量子化テーブル

ヘアバンドの女性(輝度用)

13	22	22	24	26	30	32	35
21	23	23	24	26	34	35	33
23	23	24	26	30	34	37	34
23	24	25	27	32	41	40	35
24	25	29	34	37	47	45	39
26	28	33	36	40	46	48	43
32	36	39	41	45	50	50	45
38	43	43	44	48	45	45	44

ヘアバンドの女性(クロミナンス用)

11	18	26	31	44	44	44	44
19	23	26	36	44	44	44	44
26	26	34	44	44	44	44	44
31	36	44	44	44	44	44	44
44	44	44	44	44	44	44	44
44	44	44	44	44	44	44	44
44	44	44	44	44	44	44	44
44	44	44	44	44	44	44	44

表7は, 参照画像に対するレートと各RGB色成分に対するSNRを示す. レートの値が小さければ小さいほど, 圧縮率は高くなる. 一方, SNR の値が大きければ大きいほど, 一般的に画像の品質が良い. 改善率は 64 ビット倍精度浮動小数点で計算し, 表中は四捨五入表示していることに注意すること. SNR の改善率は, 0.07%～0.80%と低いものの, レートについては, 9.12%の改善がみられる[注(14)].

表 7. レートと SNR

参照画像	国際標準例示			本方式			改善率(%)		
	レート	SNR		レート	SNR		レート	SNR	
ヘアバンドの女性	0.603	R	30.3	0.548	R	30.5	9.12	R	0.79
		G	34.6		G	34.6		G	0.07
		B	29.3		B	29.5		B	0.80

6.4. 画像の比較. 図 5 は, 左から参照画像, 国際標準例示の量子化テーブルの場合の画像, 本手法により得られた量子化テーブルの場合の画像である. ディスプレイで目視した場合, 本手法の画像は, 国際標準例示の画像に比較して著しい劣化は認められない.

図 5. 画像の比較

文献

(1) "Information technology-Digital compression and coding of continuous-tone sitll images: Requirements and guidelines", ISO/IEC10918-1(1994)

(2) 八木伸行, 井上誠喜, 林正樹ほか:"C 言語で学ぶ実践ディジタル映像処理",オーム社(平成 7 年),p.239-257

(3) 荻原剛志, 山口英(訳):"データ圧縮ハンドブック改訂第 2 版",トッパン(1996),p.280-326,ISBN4-8101-8605-9

(4) 稲村浩:"C で実現する JPEG 準拠画像データ圧縮/伸張システム",インターフェース(Dec. 1991), p.183-203

(5) 藤原洋:"最新 MPEG 教科書",アスキー出版局(1994),p.53-57,ISBN4-7561-0247-6

(6) 平田和貴, 山田順一, 新上和正:"高次元アルゴリズムによる JPEG 量子化テーブルの最適化", 電子情報通信学会技術研究報告 IE98-87(1998-11),p.31-38

(7) 平田和貴, 山田順一, 新上和正:"高次元アルゴリズムの JPEG 量子化テーブル最適化への応用",1998 年電子情報通信学会総合大会 D-11-70

(8) HIRATA Kazutaka, YAMADA Jun-ichi, SHINJO Kazumasa:"Applying the Hamiltonian algorithm to

optimize JPEG quantization tables", SPIE Vol. 3648 (Jan. 1999), pp.344-351
(9) 濱本和彦, 梅村英昭, 平田和貴, 山田順一, 新上和正:"医療用超音波画像のための JPEG 量子化テーブルの最適化に対する高次元アルゴリズムの応用",第 19 回超音波エレクトロニクスの基礎と応用に関するシンポジウム講演予稿集, OH1, (1998 年 11 月), p263
(10) 平田和貴, 山田順一:"JPEG 量子化テーブル最適化のための客観的画質評価法の検討",電子情報通信学会 1998 年情報・システムソサイエティ大会 D-11-11
(11) 平田和貴, 山田順一:"高次元アルゴリズムの JPEG 量子化テーブル最適化への応用",1999 年電子情報通信学会総合大会 D-11-25

注釈

注(1) 本来の JPEG の規定では, もとの画像品質が保たれる可逆符号化方式ともとの画像品質が保たれない不可逆符号化方式とを含んでいるが, 本報告では, 不可逆符号化方式の必須機能であるベースライン・システムを対象とする. つまり, 本報告で取り扱う JPEG 画像圧縮方式は, 本質的に画像の劣化を伴う方式であることに留意されたい.

注(2) JPEG 画像のパラメータには, JPEG 量子化テーブルのほかに, 画像の幅(水平方向の画素数)あるいは画像の高さ(垂直方向の画素数)などがある. JPEG の規定においては, 輝度用量子化テーブルとクロミナンス用量子化テーブルが例示されている. 本文中の表3は, JPEG に例示されている量子化テーブルを示す.

注(3) ここでは, 「クロミナンス」の用語を国際規格 CCIR601 で知られる色成分 YC_bC_r の内, C_bC_r 成分の意味で用いる. 色差成分と呼ばれることもある.

注(4) 量子化ステップとは, JPEG 量子化テーブルの各要素の別名ないし値のことである. 輝度用量子化テーブルは 64 個の量子化ステップの組であり, 同じく, クロミナンス用量子化テーブルも 64 個の量子化ステップの組である.

注(5) 2.2.で説明するように, 本報告では, 式(A-1)の 1 画素あたりのビット数で表現したいわゆるビット・レートを画像の圧縮率の尺度として用いている. ここで, W および H は参照画像および試験画像の幅および高さを示す.

$$rate(bit) = \frac{1}{WH}\left(\begin{array}{l}\text{復号化画像のデータ・}\\ \text{サイズ}\end{array}\right) \qquad \text{(A-1)}$$

注(6) 画像における信号対雑音比(SNR)は, 画像の品質を評価する合に典型的に用いられる尺度であり, 式(A-1)によって表現される. この派生版として視覚の周波数特性を考慮したものも多用される. 式(A-1)において各色成分を8ビットで構成することを仮定している. また, 式(A-2)の $R(x,y)$および $T(x,y)$は, それぞれ参照画像および復号化画像の座標(x,y)におけるある色成分の値を示す. また, W および H は参照画像および試験画像の幅および高さを示す. 例えば, 参照画像および試験画像の各画素の色成分を RGB の三色で構成した場合, R 成分に対する SNR, G 成分に対する SNR および B 成分に対する SNR のように, 三つの SNR を求める.

$$SNR(dB) = 20\log\frac{255}{\sqrt{MSE}} \qquad \text{(A-2)}$$

$$MSE = \frac{1}{WH}\sum_{y=1}^{H}\sum_{x=1}^{W}\left(R_{(x,y)} - T_{(x,y)}\right)^2 \qquad \text{(A-3)}$$

注(7)　ハミルトニアン H の式(2)中で表現される仮想粒子の質量 m は，高次元アルゴリズムで最適化されるパラメータではない．

注(8)　式(5)の右辺第 2 項並びに式(6)の右辺第 2 項および第 3 項の位置 q および運動量 p は現在の時刻(反復前)t における値であり，(t)の表記を割愛していることに注意されたい．

注(9)　図 3 中の二つの JPEG 画像は同一の JPEG データ・ストリームである．図 3 においては煩雑さを避けるため，JPEG データ・ストリームを JPEG 画像として 2 か所に分離して描いた．

注(10)　ソフトウェアにおける JPEG の符号化および復号化は，Independent JPEG Group の JPEG ソフトウェアのライブラリ libjpg を用いて実装されている．

注(11)　本報告の最適化においては，減衰は行っていない．高次元アルゴリズムの減衰については，第 2 章を参照されたい．

注(12)　第 1 の状態空間における仮想粒子の位置の初期値は，国際標準例示の JPEG 量子化テーブルの各量子化ステップの約 4 分の 1 の値としたものである．一方，第 2 の状態空間における仮想粒子の位置の初期値は，国際標準例示の JPEG 量子化テーブルと同じ値である．第 1 の状態空間および第 2 の状態空間における仮想粒子の運動量に対する初期値は，すべて 1 として設定した．

ここで，第 1 の状態空間の仮想粒子は，概念的には，レートに関するコスト関数によって運動が規定され，第 2 の状態空間の仮想粒子は，画像の品質に関するコスト関数によって運動が規定され，各量子化テーブルの値が変更される．

なお，初期値を乱数で生成し，JPEG 量子化テーブルの最適化をした場合に(国際標準例示のような量子化テーブルが得られるか否か)ついては研究課題である．

注(13)　他の画像に対する結果は割愛する．いろいろな画像に対する高次元アルゴリズムによる JPEG 量子化テーブルの最適化については文献(6), (7)などを参照されたい(ただし，異なるコスト関数を用いていることに注意すること)．

注(14)　この場合の得られた結果は，画像の品質を維持したまま，あるいは視覚的な画像を劣化を伴うことなく，圧縮率を約 1 割改善するものである．この結果に対して計算コストが見合うかの議論があるが，本報告は，高次元アルゴリズムが他の最適化アルゴリズムに対して(計算コストを含めた)優位性を示すものではなく，高次元アルゴリズムの実施(高次元アルゴリズムによる最適化ないしプログラミング)が容易であることを示すためにあり，計算コストに対する議論は他の機会に譲る．

第７章　ニューラルネットワークの学習

概要

本章では，ニューラルネットワークの結合荷重を決定する学習問題を扱う[(1)]．対象とするニューラルネットワークは階層型ネットワーク構造である．最適化の手順で各層内のユニットが持つ非線形入出力関数に拘束条件を導入する方法を示し，その後，計算機実験による最適化の様子を例示する．

高次元アルゴリズムによる学習法の特徴は，他の手法と比較して初期値依存性が小さく，大局的解探索を行うことである．この特徴を活かして他の局所解探索法とハイブリッド化することで，経験的なパラメータ設定への依存度を減少させた効率の良い学習を行うことができる．

１．問題の説明

最適化の対象とするニューラルネットワークの構成と各変数について説明する．図１に対象とするニューラルネットワークの構成を示す．３層の階層構造を持つニューラルネットワークで，各層のユニット数（入力：i=1,…,I，中間：j=1,…,J，出力：k=1,…,K）はあらかじめ決定されているものとする．入力層ユニット i と中間層ユニット j の結合を q_{ij}，中間層ユニット j と出力層ユニット k の結合を r_{jk} とする．各層のユニット間を結合する結合荷重 q_{ij}，r_{jk} が最適化の対象である．

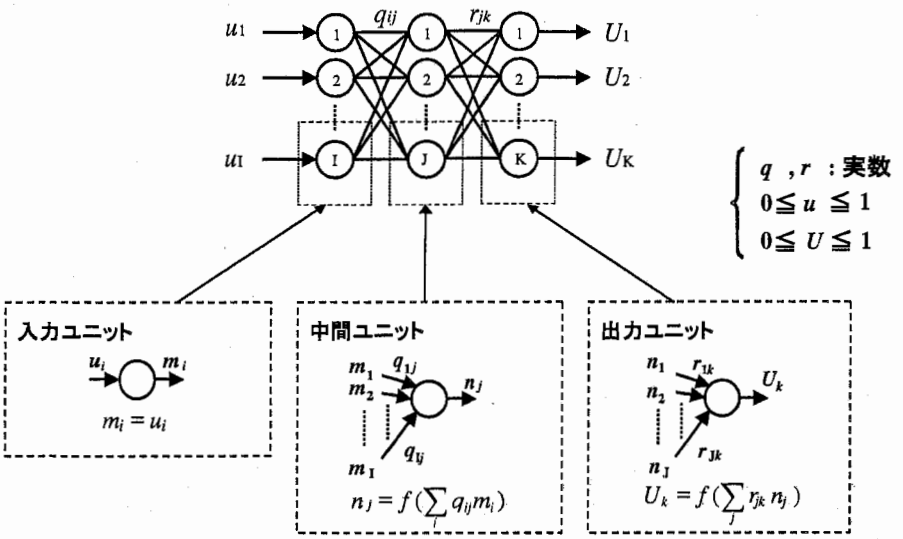


図１．最適化対象のニューラルネットワーク構成

また，各ユニットの入出力関数 $f(x)$ は，

$$f(x)=(1+\exp(-x/T))^{-1} \tag{1}$$

で示すシグモイド関数とする．このとき，入力層のユニット i へ入力値 u_i が与えられたとき，出力層のユニット k から出力される値 U_k は，

$$U_k = f(\sum_j f(\sum_i u_i q_{ij})\cdot r_{jk}) \tag{2}$$

である．

最適化は，s 組の入力ベクトル $\bar{u}_s=(u_{s1},\cdots,u_{sI})$ と出力として期待されるベクトル $\bar{S}_s=(S_{s1},\cdots,S_{sK})$ の組

$(\bar{u}_s, \bar{S}_s)$ を教師セットとして，$\bar{u}_s$ が入力されたときに $\bar{S}_s$ が出力されるように結合荷重 q_{ij}, r_{jk} を決定することである．

2．最適化問題の設定

2.1. コスト関数. はじめに，コスト関数を定義する．コスト関数は任意に決定できるが，ここでは，入力ベクトル$\bar{u}_s$が与えられた時の出力値 $\bar{U}_s = (U_{s1},\cdots,U_{sK})$ と出力期待値 $\bar{S}_s$ の２乗誤差和

$$V_1 = \sum_s \sum_k (S_{sk} - U_{sk})^2 \tag{3}$$

をコスト関数として用いる．式(3)に学習用教師データセットを与えて，ニューラルネットワークの内部自由度で表現できる最小値にコスト関数 V_1 をすることが目的である．理想的には，コスト関数 V_1 が零となることである．

先に述べたように，コスト関数は任意に決定できる．例えば、出力値の最大誤差を最小にする方法や、内部自由度の表現を組み込む方法などが考えられる．

2.2. 拘束条件. 次に，ニューラルネットワークの各ユニットにおける入出力関数に起因する問題を回避するために拘束条件を導入する．解探索の運動の方向はユニットの入出力関数を微分することで求められるが，シグモイド関数（図２）のような入出力関数の場合，丸点で示すようにユニットの入力が原点からある程度遠ざかるとほとんど力が働かなくなり，無限遠へ向かって運動する．すなわち，局所解から抜け出せなくなってしまう．この問題を回避する目的で，拘束条件を導入する．拘束条件は，原点からある程度遠ざかると原点方向へと力が働くようにするものである．これは，図３のようなスロープを持つポテンシャルをコスト関数 V_1 へ付加することにより、無限遠へ向かう運動を阻害する．

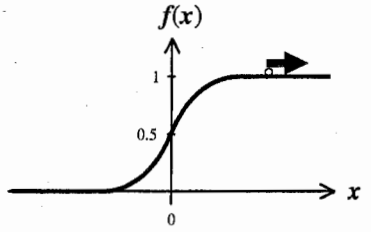


図２．ユニットの入出力関数

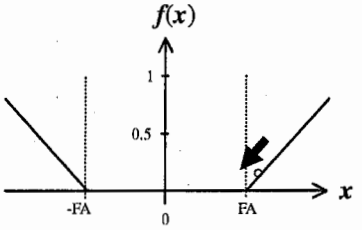


図３．拘束条件

このスロープが拘束条件であり，

$$V_2 = \alpha \sum_s \sum_j \left\{ (-FA \pm \sum_i q_{ij} m_{si}) \theta(-FA \pm \sum_i q_{ij} m_{si}) \right\} \tag{4}$$

$$V_3 = \alpha \sum_s \sum_k \left\{ (-FA \pm \sum_j r_{jk} n_{sj}) \theta(-FA \pm \sum_j r_{jk} n_{sj}) \right\} \tag{5}$$

$$\begin{cases} \theta(x) = 1 : x > 0 \\ \theta(x) = 0 : x \le 0 \end{cases}$$

で示される．式(4)の V_2 は中間層ユニットに対する拘束条件であり，式(5)の V_3 は出力層ユニットに対する拘束条件である．$\theta(x)$はステップ関数で，定数$\pm FA$ は拘束条件の付加される入出力関数の入力位置を決定する．また，α は拘束条件の傾きを決める定数である．ここでは，定数$\pm FA$，α は経験的に決定されている．しかし，未知入力への応答を考慮して，過剰学習や内部自由度などの表現をコスト関数へ盛り込む場合には，何らかの検討が必要となるであろう．

3．力学系の構成

コスト関数と拘束条件が決定されたので，次の手順としてハミルトニアンを決定する．基本型として，最も単純な

$$H = \tfrac{1}{2} p^2 + V(q) \tag{6}$$

が考えられる．$V(q)$は最適化の対象となるコスト関数で q が求める変数ベクトルである．p は q に正準共役な変数ベクトルとして新たに追加される変数である．式(6)から導かれる運動方程式に従って最適解が探索される．仮に，式(6)のハミルトニアンで解探索の軌道に混合性が観測されない場合には，混合性を導入する

$$H = \tfrac{1}{2}(Ap, p) + V(q) \tag{7}$$

を用いる必要がある．A は混合性を導入する正定値対称行列（行列の固有値が全て正）である．行列 A の掛けられたベクトル p の内積により p の各要素間のミキシング項を決定する．ただし，式(7)であっても必ずしも混合性が発生するとは限らない．したがって，行列 A をパラメータとして式(7)を調整する必要がある（ 式(7)の詳細については第９章を参照のこと ）．混合性の例として参考文献(2)がある．また，式(7)により混合性を発生させたニューラルネットワークの学習例として，参考文献(3)(4)がある．本章では，式(6)の基本型を採用する．最適化の対象となる結合荷重を $\bar{q} = (q_{11},\cdots,q_{IJ})$，$\bar{r} = (r_{11},\cdots,r_{JK})$ として，それぞれに共役な変数を $\bar{p} = (p_{11},\cdots,p_{IJ})$，$\bar{w} = (w_{11},\cdots,w_{JK})$ と定義すると、コスト関数 V_1，拘束条件 V_2，V_3 により式(6)は，

$$H(\bar{q},\bar{r},\bar{p},\bar{w}) = \tfrac{1}{2}(\bar{p}^2 + \bar{w}^2) + V_1(\bar{q},\bar{r}) + V_2(\bar{q}) + V_3(\bar{r}) \tag{8}$$

のように書ける．

4．運動方程式

解探索の軌道は，式(8)から運動方程式(9)～(12)を求めることで決定される．

$$\begin{cases} \dot{q}_{ij} = \dfrac{\partial H}{\partial p_{ij}} = p_{ij} & (9) \\ \dot{p}_{ij} = -\dfrac{\partial H}{\partial q_{ij}} = -\left(\dfrac{\partial V_1}{\partial q_{ij}} + \dfrac{\partial V_2}{\partial q_{ij}}\right) & (10) \end{cases}$$

$$\begin{cases} \dot{r}_{jk} = \dfrac{\partial H}{\partial w_{jk}} = w_{jk} & (11) \\ \dot{w}_{jk} = -\dfrac{\partial H}{\partial r_{jk}} = -\left(\dfrac{\partial V_1}{\partial r_{jk}} + \dfrac{\partial V_3}{\partial r_{jk}}\right) & (12) \end{cases}$$

ここで，シグモイド関数の微分は，

$$f'(x) = \tfrac{1}{T} f(x)\big(1 - f(x)\big) \tag{13}$$

で与えられることから，$\frac{\partial V_1}{\partial q_{ij}}, \frac{\partial V_1}{\partial r_{jk}}$ は，

$$\frac{\partial V_1}{\partial q_{ij}} = -\frac{2}{T^2}\sum_s\sum_k\left[\{S_{sk} - f(X_{sk})\}f(X_{sk})\{1-f(X_{sk})\}f(Y_{sj})\{1-f(Y_{sj})\}m_i r_{jk}\right] \tag{14}$$

$$\frac{\partial V_1}{\partial r_{jk}} = -\frac{2}{T}\sum_s\left[\{S_{sk} - f(X_{sk})\}f(X_{sk})\{1-f(X_{sk})\}f(Y_{sj})\right] \tag{15}$$

$$X_{sk} = X(s,k) = \sum_j f(\sum_i m_{si} q_{ij}) r_{jk} \tag{16}$$

$$Y_{sj} = Y(s,j) = \sum_i m_{si} q_{ij} \tag{17}$$

のように求められる．

5．プログラム

ハミルトニアンが決定されたので，実際に問題を解くためのＣ言語によるプログラム例を示す．解探索は，式(9)～(12)の運動方程式に従って行われる．図４に全体のフローを示す．初めに，初期値として結合荷重$\bar{q}$，$\bar{r}$及び，共役変数$\bar{p}$，$\bar{w}$をランダムに設定する．ただし，入出力関数（シグモイド関数）の取り得る変化域等を鑑みて，初期値を常識的な値に制限することは当然必要である．その後，運動方程式に従って各変数を更新する．このとき，現在までの更新で求められたコスト関数値V_1の最小値とその時の結合荷重$\bar{q}$，$\bar{r}$を保存しておき，一回更新が行われる度にコスト関数値V_1を求め，前回までに求められたコスト値の最小値よりも更に小さなコスト値であれば，保存されている結合荷重$\bar{q}$，$\bar{r}$を更新する．最適化の終了条件としては，更新回数を指定してその中で最小のコスト値を与えた結合荷重$\bar{q}$,$\bar{r}$を解とする方法やコスト値が指定された値以下になった時点で終了する方法などが考えられる．ここでは，前者の方法を取ることにする．

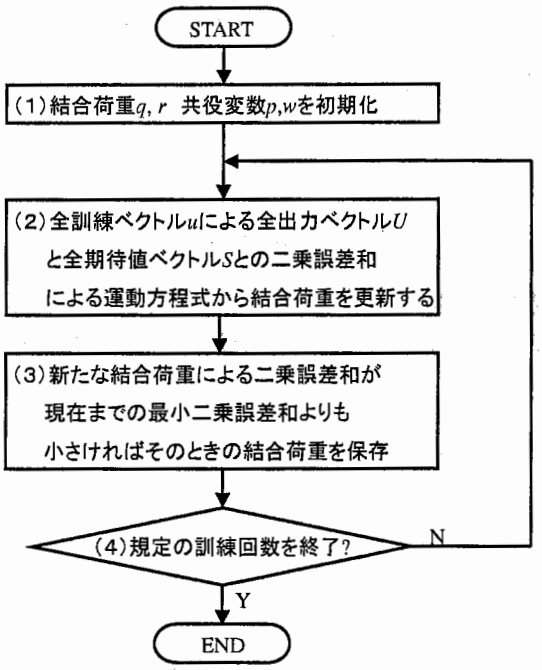


図４．解探索プログラムの流れ

本節での説明は，図4中の(2)結合荷重の更新についてである．プログラム中では各変数を更新するために微分方程式の解にRunge-Kutta法を用いている．Runge-Kutta法による各変数の更新は，

$$\begin{cases} q_{ij}(t+dt) = q_{ij}(t) + \frac{1}{6}(kq_{1ij} + 2kq_{2ij} + 2kq_{3ij} + kq_{4ij}) & (18) \\ r_{jk}(t+dt) = r_{jk}(t) + \frac{1}{6}(kr_{1jk} + 2kr_{2jk} + 2kr_{3jk} + kr_{4jk}) & (19) \\ p_{ij}(t+dt) = p_{ij}(t) + \frac{1}{6}(hp_{1ij} + 2hp_{2ij} + 2hp_{3ij} + hp_{4ij}) & (20) \\ w_{jk}(t+dt) = w_{jk}(t) + \frac{1}{6}(hw_{1jk} + 2hw_{2jk} + 2hw_{3jk} + hw_{4jk}) & (21) \end{cases}$$

を計算すればよい．ただし，各変数は以下で求められる．

$$\begin{cases} kq_1 = \dot{q}(p(t))dt \\ kr_1 = \dot{r}(w(t))dt \\ hp_1 = \dot{p}(q(t), r(t))dt \\ hw_1 = \dot{w}(q(t), r(t))dt \end{cases} \quad \begin{cases} kq_2 = \dot{q}(p(t) + hp_1/2)dt \\ kr_2 = \dot{r}(w(t) + hw_1/2)dt \\ hp_2 = \dot{p}(q(t) + kq_1/2, r(t) + kr_1/2)dt \\ hw_2 = \dot{w}(q(t) + kq_1/2, r(t) + kq_1/2)dt \end{cases}$$

$$\begin{cases} kq_3 = \dot{q}(p(t) + hp_2/2)dt \\ kr_3 = \dot{r}(w(t) + hw_2/2)dt \\ hp_3 = \dot{p}(q(t) + kq_2/2, r(t) + kr_2/2)dt \\ hw_3 = \dot{w}(q(t) + kq_2/2, r(t) + kq_2/2)dt \end{cases} \quad \begin{cases} kq_4 = \dot{q}(p(t) + hp_3)dt \\ kr_4 = \dot{r}(w(t) + hw_3)dt \\ hp_4 = \dot{p}(q(t) + kq_3, r(t) + kr_3)dt \\ hw_4 = \dot{w}(q(t) + kq_3, r(t) + kq_3)dt \end{cases}$$

この部分のプログラムを付録にしているので，参照願いたい．注意点として，拘束条件を入れた場合に、局所解から脱出する際に力が不連続に働くためエネルギーが保存されない点である．したがって，バグチェック等でエネルギーの保存を確認する場合には拘束条件を外しておく必要がある．

以下では，付録のプログラムの一部について変数の説明と簡単な補足を記す．本プログラムでは，引数の解り易さため2次元配列のサイズを固定として記述している．

5.1. 共役変数 p_{ij}, w_{jk} の偏微分. 図5は Runge-Kutta 法で用いる kq_{nij}, kr_{njk} (n = 1,...,4) を求める関数を示している. ForceDq()関数は, $\dot{q}_{ij}$ の引数となる p_{ij} へのポインタと結果である kq_{nij} を格納するための配列へのポインタを引数とする. 各2次元配列は i=16, j=16 として確保されている. ForceDr()関数は, $\dot{r}_{jk}$ の引数となる w_{jk} へのポインタと結果である kr_{njk} を格納するための配列へのポインタを引数とする. 各2次元配列は j=16, k=2 として確保されている. それぞれ, 時間刻み dt が掛けられている.

```
001  void CNeuroDoc::ForceDq(double kq[][16], double p[][16])
002  {                                          q̇ij = ∂H/∂pij = pij
003  UINT  i, j;
004    for( i=0; i<m_imax; i++) {
005      for( j=0; j<m_jmax; j++) kq[i][j] = p[i][j]*m_dt;
006    }
007  }

001  void CNeuroDoc::ForceDr(double kr[][2], double w[][2])
002  {                                          ṙjk = ∂H/∂wjk = wjk
003  UINT  j, k;
004    for( j=0; j<m_jmax; j++) {
005      for( k=0; k<m_kmax; k++) kr[j][k] = w[j][k]*m_dt;
006    }
007  }
```

図5. 共役変数 p_{ij}, w_{jk} の偏微分

▼変数の説明

kq[][16] : kq1[i][j], kq2[i][j], …, kq4[i][j] へのポインタ
p[][16] : m_p[i][j], m_p[i][j]+ hp1[i][j]/2., …, m_p[i][j]+ hp3[i][j] へのポインタ
kr[][2] : kr1[j][k], kr2[j][k], …, kr4[j][k] へのポインタ
w[][2] : m_w[j][k], m_w[j][k]+ hw1[j][k]/2., …, m_w[j][k]+ hw3[j][k] へのポインタ
m_imax : 入力層ユニットの数
m_jmax : 中間層ユニットの数
m_kmax : 出力層ユニットの数
m_dt : 時間刻み

5.2. w_{jk} を更新する $-\frac{\partial V_1}{\partial r_{jk}}$ 関数. 図6はRunge-Kutta法で用いる hw_{njk} (n = 1,...,4) を求める関数を示している. ForceDw()関数は, $\dot{w}_{jk}$ の引数となる q_{ij} と r_{jk} へのポインタと結果である hw_{njk} を格納するための配列へのポインタを引数とする. 各2次元配列は i=16, j=16, k=2 として確保されている. それぞれ, 時間刻み dt が掛けられている.

```
001  void CNeuroDoc::ForceDw(double hw[][2], double q[][16], double r[][2])
002  {
003  UINT j, k, s;
004  double thw, fxsk, fysj;
005    for( j=0; j<m_jmax; j++) {
006      for( k=0; k<m_kmax; k++) {
007        for( s=0, thw=0; s<m_smax; s++) {
008          fxsk = CalXsk( s, k, q, r);
009          fysj = CalYsj( s, j, q);
010          thw += ((m_s[s][k]-fxsk)*(1.-fxsk)*fxsk*fysj);
011        }
012        hw[j][k] = (2./m_T)*thw*m_dt;
013      }
014    }
015  }
```

$$-\frac{\partial V_1}{\partial r_{jk}} = \frac{2}{T}\sum_s \left[\{S_{sk} - f(X_{sk})\}\{1 - f(X_{sk})\}f(X_{sk})f(Y_{sj})\right]$$

図6. w_{jk} を更新する $-\frac{\partial V_1}{\partial r_{jk}}$ 関数

▼変数の説明

hw[][2] : kw1[j][k], kw2[j][k], …, kw4[j][k] へのポインタ
q[][16] : m_q[i][j], m_q[i][j]+ kq1[i][j]/2., …, m_q[i][j]+ kq3[i][j] へのポインタ
r[][2] : m_r[j][k], m_r[j][k]+ kr1[j][k]/2., …, m_r[j][k]+ kr3[j][k] へのポインタ
m_jmax : 中間層ユニットの数
m_kmax : 出力層ユニットの数
m_smax : 訓練データの数
m_T : シグモイド関数のパラメータ
m_dt : 時間刻み
m_s[s][k] : 各訓練データ[s]毎に出力ユニット[k]から出力されることを期待される値

▼関数の説明

double CalXsk(UINT s, UINT k, double q[][16], double r[][2]);

$$X_{sk} = X(s,k) = \sum_j f(\sum_i m_{si} q_{ij}) r_{jk}$$ → X_{sk} をシグモイド関数で変換した値を返す.

double CalYsj(UINT s, UINT j, double q[][16]);

$$Y_{sj} = Y(s,j) = \sum_i m_{si} q_{ij}$$ → Y_{sj} をシグモイド関数で変換した値を返す.

注) $-\frac{\partial V_3}{\partial r}$ は別途求める.

5.3. p_{ij} を更新する $-\frac{\partial V_1}{\partial q_{ij}}$ 関数．図7は Runge-Kutta 法で用いる hp_{nij}（n = 1,...,4）を求める関数を示している．ForceDp()関数は，p_{ij} の引数となる q_{ij} と r_{jk} へのポインタと結果である hp_{nij} を格納するための配列へのポインタを引数とする．各2次元配列は i=16, j=16, k=2 として確保されている．それぞれ，時間刻み dt が掛けられている．

```
001  void CNeuroDoc::ForceDp(double hp[][16], double q[][16], double r[][2])
002  { UINT i, j, s, k; double fxsk, fysj, thp;
003    for( i=0; i<m_imax; i++) {
004      for( j=0; j<m_jmax; j++) {
005        for( s=0, thp=0; s<m_smax; s++) {
006          for( k=0; k<m_kmax; k++) {
007            fxsk = CalXsk( s, k, q, r); fysj = CalYsj( s, j, q);
008            thp += ((m_s[s][k]-fxsk)*(1.-fxsk)*fxsk
009                                  *(1.-fysj)*fysj*m_m[s][i]*r[j][k]);
010          }
011        }
012        hp[i][j] = (2./(m_T*m_T))*thp*m_dt;
013      }
014    }
015  }
```

$$-\frac{\partial V_1}{\partial q_{ij}} = \frac{2}{T^2}\sum_s\sum_k \left[\{S_{sk} - f(X_{sk})\}\{1 - f(X_{sk})\}f(X_{sk})\{1 - f(Y_{sj})\}f(Y_{sj})m_{si}\omega_{jk}\right]$$

図7．p_{ij} を更新する $-\frac{\partial V_1}{\partial q_{ij}}$ 関数

▼変数の説明

hp[][16] : kp1[i][j], kp2[i][j], …, kp4[i][j] へのポインタ
q[][16] : m_q[i][j], m_q[i][j]+ kq1[i][j]/2., …, m_q[i][j]+ kq3[i][j] へのポインタ
r[][2] : m_r[j][k], m_r[j][k]+ kr1[j][k]/2., …, m_r[j][k]+ kr3[j][k] へのポインタ
m_imax : 入力層ユニットの数
m_jmax : 中間層ユニットの数
m_kmax : 出力層ユニットの数
m_smax : 訓練データの数
m_T : シグモイド関数のパラメータ
m_dt : 時間刻み
m_S[s][k] : 各訓練データ[s]毎に出力ユニット[k]から出力されることを期待される値
m_m[s][i] : 各訓練データ[s]毎の入力ユニット[i]の出力(0 ≦ m_m[s][i] ≦ 1.)

▼関数の説明

```
double CalXsk( UINT s, UINT k, double q[][16], double r[][2]);
```
前ページと同一

```
double CalYsj( UINT s, UINT j, double q[][16]);
```
前ページと同一

注) $-\frac{\partial V_2}{\partial q}$ は別途求める．

6．計算結果

作成したプログラムを使用してニューラルネットワークの学習を行う．

6.1. 訓練データセットと初期化． 学習のための教師として，入力データと出力データを組にした訓練データセットを図８に定義する．データセットは s=8 組用意し、入力$\bar{u}_s$ を添付プログラム中の変数 m_m[s][i]にセットする．また，出力期待値$\bar{S}_s$ を変数 m_s[s][k]にセットする．

	入力ベクトル	最適化後に期待する出力
s	$\bar{u}_s = (u_{s1}, u_{s2}, \ldots, u_{sI})$, (I=16)	$\bar{S}_s = (S_{s1}, \ldots, S_{sK})$, (K=2)
1	(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	(1, 0)
2	(0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)	(1, 0)
3	(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0)	(0, 1)
4	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1)	(1, 0)
5	(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)	(0, 1)
6	(0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0)	(0, 1)
7	(0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0)	(0, 1)
8	(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1)	(0, 1)

図８．訓練データセット

変数 q_{ij} = m_q[i][j], r_{jk} = m_r[j][k], p_{ij} = m_p[i][j], w_{jk} = m_w[j][k]は±0.1 以内でランダムに初期化する．また，学習の進み具合を考慮して，ここでは時間刻み m_dt = 0.001，シグモイド関数の温度 m_T = 0.02 とする．拘束条件は，ステップ関数の閾値 m_fa = 0.3，傾き m_fax = 4 とする．

6.2. バグチェック． トータルエネルギー$\mathbf{H}(\bar{q},\bar{r},\bar{p},\bar{w})$が保存されていることを確認する．ただし，拘束条件が q_{ij}，r_{jk} ではなく $\Sigma q_{ij}m_{si}$，$\Sigma r_{jk}n_{sj}$ で制御されるためエネルギーが保存されない．そこで，拘束条件を外して確認する．添付プログラムでは ForceBp()，ForceBw()をコメントアウトしておく．図９は，初期エネルギーと各学習回毎のエネルギーとの比を確認した一例である．

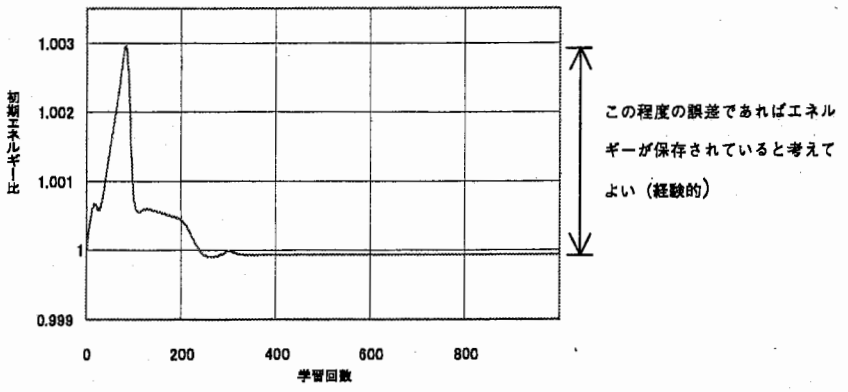


図９．各学習毎エネルギー／初期エネルギー比（拘束条件なし）

6.3. 最適化の様子. 学習の様子をコスト関数 H の最適化の様子で示す. 図１０では，拘束条件の有無を比較している. どちらも同一の初期値から始めているが，拘束条件がない場合は局所解から脱出できず，解が求められない.

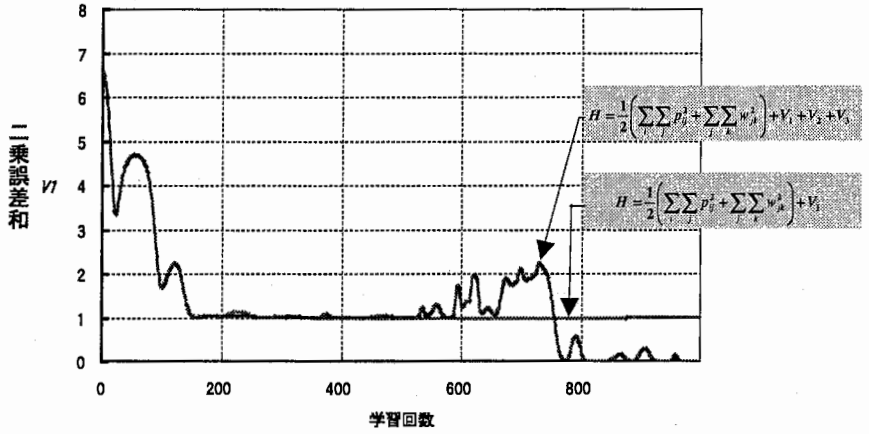


図１０. 最適化の様子

また、学習においてコスト値 H が零近傍の値を取るときを解として考えると，解候補は複数存在するが，図１１，１２に1000回の学習で求められた結合荷重 q_{ij}, r_{ij} の一例を示す.

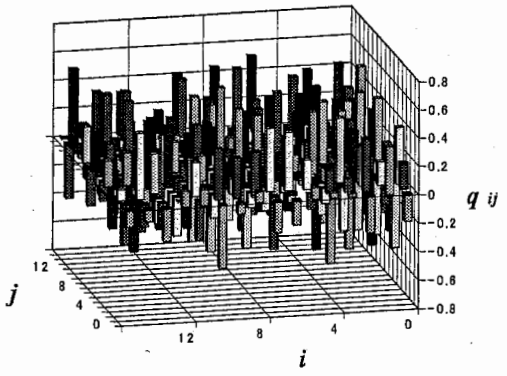


図１１. 1000回の探索で求められた結合荷重 q_{ij}

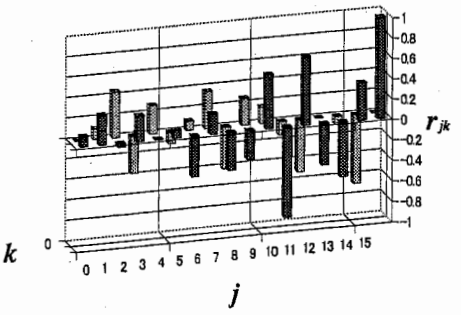


図１２. 1000回の探索で求められた結合荷重 r_{ij}

6.4. まとめ. 本章では高次元アルゴリズムをニューラルネットワークの学習問題へ適用した. 高次元アルゴリズムによる学習は，以下の特徴を持つ.

1）初期値やパラメータの設定が容易

2）更新回数が比較的少なく済む

また，演算量は，

$$\begin{cases} \dot{q}_{ij} = \dfrac{\partial H}{\partial p_{ij}} = p_{ij} \end{cases} \tag{22}$$

$$\dot{p}_{ij} = -\frac{\partial V_1}{\partial q_{ij}} + \alpha n_i \tag{23}$$

$$\begin{cases} \dot{r}_{jk} = \dfrac{\partial H}{\partial w_{jk}} = w_{jk} \end{cases} \tag{24}$$

$$\dot{w}_{jk} = -\frac{\partial V_1}{\partial r_{jk}} + \alpha n_j \tag{25}$$

の下線で示す共役変数と拘束条件の演算が逆誤差伝播法に対して増加する．しかし，式から分かるように演算量の増加はほとんど無視できると考えてよい．ただし，添付プログラムで使用しているRunge-kutta 法では，一回の更新で $\dot{q}_{ij}$, $\dot{p}_{ij}$, $\dot{r}_{jk}$, $\dot{w}_{jk}$ ともに 4 回演算される．

最後に，添付プログラムを用いた場合の参考計算時間を表 1 に示す．

表 1．参考計算時間

計算機	PentiumⅢ 733MHz ×2
OS	Win2000
使用言語	Visual C++ 6.0
パラメータ更新回数	1000 回
計算時間	約 210 秒

文献

(1) 倉持,新上：” 高次元アルゴリズムによるニューラルネットワークの学習,”
情報処理学会第 60 回全国大会，2 分冊 p. 143-144 (2000).

(2) 倉持,新上,下川：” 高次元アルゴリズムの高速化に関する検討,”
信学会総合大会 A-1-9，p. 9 (1999).

(3) 倉持,新上,下川,河野：” 拡張型結合を持つニューラルネットワークの HA 学習,”
情報処理学会第 61 回全国大会，2 分冊 p. 57-58 (2000).

(4) 河野,下川,新上：” 高次元アルゴリズムによる効率的な大局解探索法：
ニューラルネットワークの学習問題への適用,” 情報処理学会第 61 回全国大会，
2 分冊 p. 63-64 (2000).

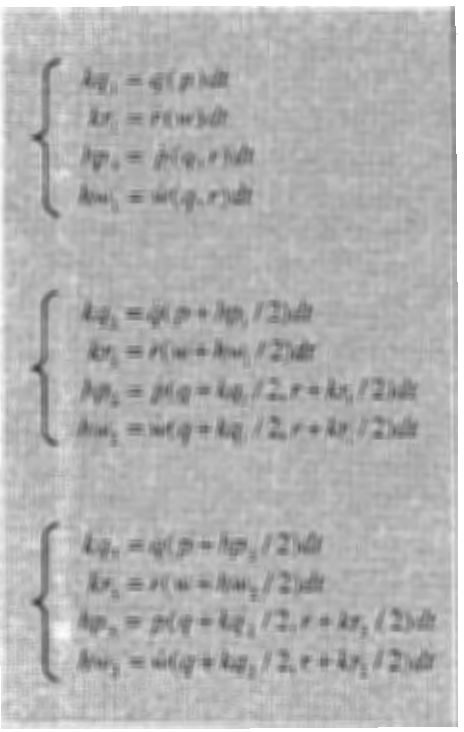
（付録）C言語によるプログラム例

```
// 図4のフローチャートで示した(2)部分に対応するプログラム
// 下記変数を初期化後、RungeKutta()関数を1回呼び出す毎に結合荷重が1回更新される。

//  初期化の必要な変数の種類とタイプ（以下記載の2次元配列は型宣言ではありません）
double m_T;   // シグモイド関数のパラメータ
double m_dt;  // 刻み時間
double m_fa;  // 拘束条件（ステップ関数閾値）
double m_fax; // 拘束条件（係数）
UINT m_smax;  // 訓練データ数
UINT m_imax;  // 入力ユニット数
UINT m_jmax;  // 中間ユニット数
UINT m_kmax;  // 出力ユニット数
double m_m[m_smax][m_imax];  // 訓練データ（入力）
double m_s[m_smax][m_kmax];  // 訓練データ（出力期待値）
double m_q[m_imax][m_jmax];  // 入力層－中間層間結合荷重
double m_p[m_imax][m_jmax];  // 入力層－中間層間運動量
double m_r[m_jmax][m_kmax];  // 中間層－出力層間結合荷重
double m_w[m_jmax][m_kmax];  // 中間層－出力層間運動量

//  テンポラル変数の種類とタイプ（以下記載の2次元配列は型宣言ではありません）
double  tq[m_imax][m_jmax];          // ForceDp(),ForceDw() の引数として使用（m_q[][]+kq1[][]/2. などの値が入いる）
double kq1[m_imax][m_jmax], kq2[m_imax][m_jmax], kq3[m_imax][m_jmax], kq4[m_imax][m_jmax];
double  tr[m_jmax][m_kmax];          // ForceDp(),ForceDw() の引数として使用（m_r[][]+kr1[][]/2. などの値が入いる）
double kr1[m_jmax][m_kmax], kr2[m_jmax][m_kmax], kr3[m_jmax][m_kmax], kr4[m_jmax][m_kmax];
double  tp[m_imax][m_jmax];          // ForceDq() の引数として使用（m_p[][]+hp1[][]/2. などの値が入いる）
double hp1[m_imax][m_jmax], hp2[m_imax][m_jmax], hp3[m_imax][m_jmax], hp4[m_imax][m_jmax];
double  tw[m_jmax][m_kmax];          // ForceDr() の引数として使用（m_w[][]+hw1[][]/2. などの値が入いる）
double hw1[m_jmax][m_kmax], hw2[m_jmax][m_kmax], hw3[m_jmax][m_kmax], hw4[m_jmax][m_kmax];
```

```
1  #include <math.h>
2  ////////////////////////////////////////////////////////////////////////////
3  // Runge Kutta 補間
4  void CNeuroDoc::RungeKutta()
5  {
6  UINT i, j, k, s;
7  // 1
8      SetDqrpw( tq, kq1, tr, kr1, tp, hp1, tw, hw1, 0);
9      ForceDq( kq1, tp);
10     ForceDr( kr1, tw);
11     ForceDw( hw1, tq, tr);
12     ForceBw( hw1, tq, tr);
13     ForceDp( hp1, tq, tr);
14     ForceBp( hp1, tq);
15 // 2
16     SetDqrpw( tq, kq1, tr, kr1, tp, hp1, tw, hw1, .5);
17     ForceDq( kq2, tp);
18     ForceDr( kr2, tw);
19     ForceDw( hw2, tq, tr);
20     ForceBw( hw2, tq, tr);
21     ForceDp( hp2, tq, tr);
22     ForceBp( hp2, tq);
23 // 3
24     SetDqrpw( tq, kq2, tr, kr2, tp, hp2, tw, hw2, .5);
25     ForceDq( kq3, tp);
26     ForceDr( kr3, tw);
27     ForceDw( hw3, tq, tr);
28     ForceBw( hw3, tq, tr);
29     ForceDp( hp3, tq, tr);
30     ForceBp( hp3, tq);
```



```
    // 4
        SetDqrpw( tq, kq3, tr, kr3, tp, hp3, tw, hw3, 1.);
        ForceDq( kq4, tp);
        ForceDr( kr4, tw);
        ForceDw( hw4, tq, tr);
        ForceBw( hw4, tq, tr);
        ForceDp( hp4, tq, tr);
        ForceBp( hp4, tq);
    //
        for( i=0; i<m_imax; i++) {
            for( j=0; j<m_jmax; j++) {
                m_p[i][j] += ((hp1[i][j]+2.*hp2[i][j]+2.*hp3[i][j]+hp4[i][j])/6.);
                m_q[i][j] += ((kq1[i][j]+2.*kq2[i][j]+2.*kq3[i][j]+kq4[i][j])/6.);
            }
        }
        for( j=0; j<m_jmax; j++) {
            for( k=0; k<m_kmax; k++) {
                m_w[j][k] += ((hw1[j][k]+2.*hw2[j][k]+2.*hw3[j][k]+hw4[j][k])/6.);
                m_r[j][k] += ((kr1[j][k]+2.*kr2[j][k]+2.*kr3[j][k]+kr4[j][k])/6.);
            }
        }
    }
    ////////////////////////////////////////////////////////////////////////////
    // 補間
    void CNeuroDoc::SetDqrpw( double q[][16], double kq[][16], double r[][2], double kr[][2],
                              double p[][16], double hp[][16], double w[][2], double hw[][2], double x)
    {
    UINT  i, j, k;
        for( i=0; i<m_imax; i++) {
          for( j=0; j<m_jmax; j++) {
                p[i][j] = m_p[i][j]+hp[i][j]*x;
                q[i][j] = m_q[i][j]+kq[i][j]*x;
            }
        }
        for( j=0; j<m_jmax; j++) {
            for( k=0; k<m_kmax; k++) {
                w[j][k] = m_w[j][k]+hw[j][k]*x;
                r[j][k] = m_r[j][k]+kr[j][k]*x;
            }
        }
    }
    ////////////////////////////////////////////////////////////////////////////
    // q̇ij  (本文参照)
    void CNeuroDoc::ForceDq(double kq[][16], double p[][16])
    {
    UINT  i, j;
        for( i=0; i<m_imax; i++) {
            for( j=0; j<m_jmax; j++) kq[i][j] = p[i][j]*m_dt;
        }
    }
    ////////////////////////////////////////////////////////////////////////////
    // ṙjk  (本文参照)
    void CNeuroDoc::ForceDr(double kr[][2], double w[][2])
    {
    UINT  j, k;
        for( j=0; j<m_jmax; j++) {
            for( k=0; k<m_kmax; k++) kr[j][k] = w[j][k]*m_dt;
        }
    }
```

$$\begin{cases} p(t+1) = p(t) + \frac{1}{6}(hp_1 + 2hp_2 + 2hp_3 + hp_4) \\ q(t+1) = q(t) + \frac{1}{6}(kq_1 + 2kq_2 + 2kq_3 + kq_4) \end{cases}$$

$$\begin{cases} w(t+1) = w(t) + \frac{1}{6}(hw_1 + 2hw_2 + 2hw_3 + hw_4) \\ r(t+1) = r(t) + \frac{1}{6}(kr_1 + 2kr_2 + 2kr_3 + kr_4) \end{cases}$$

```
91  ////////////////////////////////////////////////////////////////////////////
92  // ẇjk  (本文参照)
93  void CNeuroDoc::ForceDw(double hw[][2], double q[][16], double r[][2])
94  {
95  UINT j, k, s;
96  double thw, fxsk, fysj;
97      for( j=0; j<m_jmax; j++) {
98          for( k=0; k<m_kmax; k++) {
99              for( s=0, thw=0; s<m_smax; s++) {
100                 fxsk = CalXsk( s, k, q, r);
101                 fysj = CalYsj( s, j, q);
102                 thw += ((m_s[s][k]-fxsk)*(1.-fxsk)*fxsk*fysj);
103             }
104             hw[j][k] = (2./m_T)*thw*m_dt;
105         }
106     }
107 }
108 ////////////////////////////////////////////////////////////////////////////
109 // 拘束条件 V3
110 void CNeuroDoc::ForceBw(double hw[][2], double q[][16], double r[][2])
111 {
112 UINT i, j, k, s;
113 double con, nst;
114     for( s=0; s<m_smax; s++) {
115         for( k=0; k<m_kmax; k++) {
116             con = 0;
117             for( j=0; j<m_jmax; j++) {
118                 nst=0;
119                 for( i=0; i<m_imax; i++) nst += m_m[s][i]*q[i][j];
120                 con += Calc_Sig( nst, m_T)*r[j][k];
121             }
122             if( con >=  m_fa ) {
123                 for( j=0; j<m_jmax; j++) {
124                     nst=0;
125                     for( i=0; i<m_imax; i++) nst += m_m[s][i]*q[i][j];
126                     hw[j][k] += (-m_fax * Calc_Sig( nst, m_T) * m_dt);
127                 }
128             }
129             if( con <= -m_fa ) {
130                 for( j=0; j<m_jmax; j++) {
131                     nst=0;
132                     for( i=0; i<m_imax; i++) nst += m_m[s][i]*q[i][j];
133                     hw[j][k] += ( m_fax * Calc_Sig( nst, m_T) * m_dt);
134                 }
135             }
136         }
137     }
138 }
139 ////////////////////////////////////////////////////////////////////////////
140 // ṗij  (本文参照)
141 void CNeuroDoc::ForceDp(double hp[][16], double q[][16], double r[][2])
142 {
143 UINT i, j, s, k;
144 double fxsk, fysj, thp;
145     for( i=0; i<m_imax; i++) {
146         for( j=0; j<m_jmax; j++) {
147             for( s=0, thp=0; s<m_smax; s++) {
148                 for( k=0; k<m_kmax; k++) {
149                     fxsk = CalXsk( s, k, q, r);
150                     fysj = CalYsj( s, j, q);
```

$$-\frac{\partial V_3}{\partial r_{jk}}$$

$$V_3 = \alpha \sum_s \sum_k \left\{ \left(-FA \pm \sum_j r_{jk} n_{sj} \right) \theta \left(-FA \pm \sum_j r_{jk} n_{sj} \right) \right\}$$

RungeKutta( ) 内での本関数呼出しをコメントアウトすることで拘束条件を削除可能.

$$\theta\left(-FA + \sum_j r_{jk} n_{sj} \right)$$

$$\theta\left(-FA - \sum_j r_{jk} n_{sj} \right)$$

```
                    thp += ((m_s[s][k]-fxsk)*(1.-fxsk)*fxsk*(1.-fysj)*fysj*m_m[s][i]*r[j][k]);
                }
            }
            hp[i][j] = (2./(m_T*m_T))*thp*m_dt;
        }
    }
}
////////////////////////////////////////////////////////////////////////////
// 拘束条件 V2
void CNeuroDoc::ForceBp(double hp[][16], double q[][16])
{
double con;
UINT i, j, s;
    for( s=0; s<m_smax; s++) {
        for( j=0; j<m_jmax; j++) {
            con=0;
            for( i=0; i<m_imax; i++) con += m_m[s][i]*q[i][j];
            if( con >=  m_fa ) {
                for( i=0; i<m_imax; i++) hp[i][j] += ( -m_fax * m_m[s][i] * m_dt);
            }
            if( con <= -m_fa ) {
                for( i=0; i<m_imax; i++) hp[i][j] += (  m_fax * m_m[s][i] * m_dt);
            }
        }
    }
}
////////////////////////////////////////////////////////////////////////////
// f(Xsk)
double CNeuroDoc::CalXsk( UINT s, UINT k, double q[][16], double r[][2])
{
UINT i, j;
double  tqin, trin;
    trin = 0;
    for( j=0; j<m_jmax; j++) {
        tqin = 0;
        for( i=0; i<m_imax; i++) tqin += (m_m[s][i]*q[i][j]);
        trin += (Calc_Sig( tqin, m_T)*r[j][k]);
    }
    return( Calc_Sig( trin, m_T));
}
////////////////////////////////////////////////////////////////////////////
// f(Ysj)
double CNeuroDoc::CalYsj( UINT s, UINT j, double q[][16])
{
UINT i;
double  tqin;
    tqin = 0;
    for( i=0; i<m_imax; i++) tqin += (m_m[s][i]*q[i][j]);
    return( Calc_Sig( tqin, m_T));
}
////////////////////////////////////////////////////////////////////////////
// シグモイド関数
double CNeuroDoc::Calc_Sig(double x, double beta)
{
    sig = return( 1./(1.+exp(-x/beta)) );
}
```

$$-\frac{\partial V_2}{\partial q_{ij}}$$

$$V_2 = \alpha \sum_s \sum_i \left\{ \left(-FA \pm \sum_j q_{ij} m_{sj} \right) \theta \left(-FA \pm \sum_j q_{ij} m_{sj} \right) \right\}$$

RungeKutta( ) 内での本関数呼出しをコメントアウトすることで拘束条件を削除可能。

$$\theta\left(-FA + \sum_i q_{ij} m_{si}\right)$$

$$\theta\left(-FA - \sum_i q_{ij} m_{si}\right)$$

$$X_{sk} = X(s,k) = \sum_j f\left(\sum_i m_{si} q_{ij}\right) r_{jk}$$

$$Y_{sj} = Y(s,j) = \sum_i m_{si} q_{ij}$$

第８章　コージェネレーションシステムの最適設計・制御

概要

コージェネレーションシステム(Co-Generation System：以下 CGS)は，単一燃料から２種類のエネルギー（電気と熱）を供給するシステムであり，エネルギー効率が高く，経済性に優れ，地球環境問題への対策としても期待されている．しかしながら，実際の建物設計において，CGS の最適な装置容量や運転方法を求めることは，かなり難しい問題となる．このため従来手法では，想定されるエネルギー負荷について，運転方法に条件（例えば電力負荷をすべて発電により賄う等）を設けることにより問題を単純化して装置容量を求めるのが一般的であった．

これを CGS の中を流れる複数のエネルギー量が各々288（24 時間×12 月）の値を持つ多変数の最適化問題と考え，高次元アルゴリズムを適用すると，設定したコスト関数の下，想定されるエネルギー負荷について，最適な装置容量と運転方法を同時に求めることが可能となる．本章では，経済性についての指標である投資回収年数をコスト関数とした最適化について述べる．また特に，冷房と暖房の切り替えを表す整数変数とエネルギー量を表す実数変数が混在する最適化問題を高次元アルゴリズムで解く場合の一例として，整数変数の取り扱いについて説明する．

なお，本適用例は，株式会社 NTT ファシリティーズとの共同研究の結果である．

１．問題の説明

近年，経済性，環境性の両面で CGS が評価され，いくつかのタイプの CGS が実用化されてきた．その中でも，最も普及しているのは，都市ガスを入力エネルギーとし，原動機にガスエンジン（Gas Engine：以下 GE）を使って発電を行うタイプである（図１）．このタイプでは，発電時に生じる排熱を利用して，排熱吸収式冷凍機（Absorption Refrigerator：以下 AR）による冷房や暖房を供給する．エネルギー負荷に対して不足がある場合は，購入電力や補助熱源であるヒートポンプ（Heat Pump：HP）を使って不足分を賄う．HP は，家庭用のエアコン同様，運転切替により冷房／暖房が可能である．

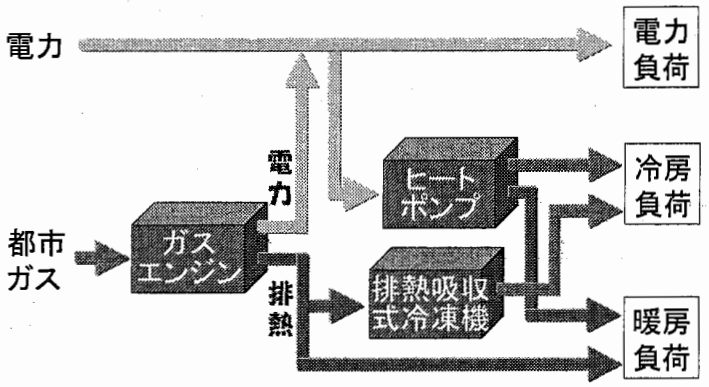


図１．一般的な CGS モデル

経済性を考える場合，もし CGS の装置容量がエネルギー負荷に対して十分大きければ，電力に比べて安価な都市ガスですべてのエネルギー負荷を賄うことによりランニングコストを最も安くできる．しかし現実には，十分大きな装置を導入すると，初期コストが多大になり経済性は悪くなる．以下では，経済性についての一般的な指標である「投資回収年数」を最適化（最小化）する装置容量と運転方法を問題として考える．もちろん，環境性の指標である「一次エネルギーの消費量」や「二酸化炭素の排出量」を最適化する問題も取り扱い可能であるし，経済性と環境性を適当に重み付けして両立させる問題も興味深いがここでは取り扱わない．

さて，投資回収年数の比較対象となるのは，購入電力と HP のみでエネルギー負荷を賄うシステム

である（図２）．比較モデルでは，想定されるエネルギー負荷に対して，２台もしくは１台（各月の熱負荷が冷房または暖房いずれかのみの場合は１台でよい）の HP の必要容量および購入電力量は簡単に計算できる．

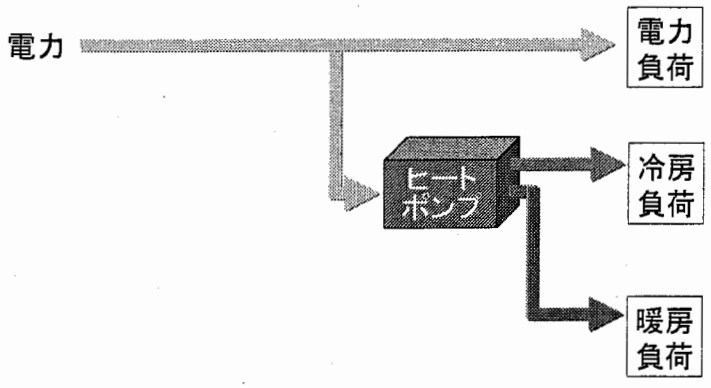


図２．比較モデル

投資回収年数は，CGS と比較モデルの初期コストの差分を年間ランニングコストの差分で除した値である．建物設計時に想定されるエネルギー負荷に対して，適当な容量の CGS 装置を導入し，最適な運転を行うことにより，最小の投資回収年数が得られる．

エネルギー負荷は，ホテル，事務所，住宅，通信機械室，店舗といった建物用途別に，単位面積あたりの電力，冷房，暖房負荷データ[1]として与えられる．負荷データは，各々24 時間×12 月（すなわち，月内の負荷パターンは同じ）の個数となる．

２．最適化問題の設定

2.1. 投資回収年数. コスト関数となる投資回収年数を最適化変数で記述するために，図１の CGS モデルにおけるエネルギー量の一部を最適化変数として定義する．図３において，x_{ct}, x_{hc}, x_{hw}, c_h が最適化変数であり，それ以外のエネルギー量は最適化変数および定数（負荷データ）により記述できる．図１に対して，より現実的な CGS を扱うために，エネルギーの流れに廃棄排熱を加えた．また，装置台数は GE，AR，HP とも複数台を可能とするが，HP のみ非均等な装置容量を許し，GE と AR は均等な装置容量とする．

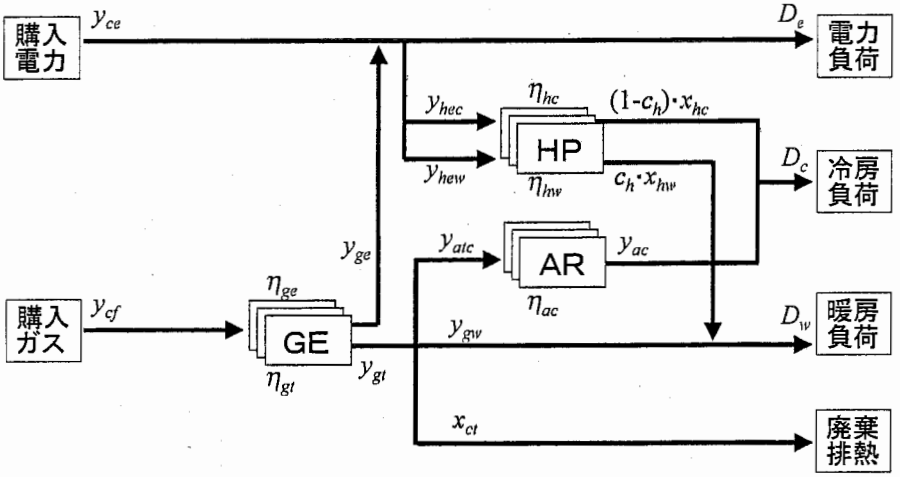


図３．最適化変数

最適化変数 c_h は，HP の運転切替を表し，冷房なら 0，暖房なら 1 となる整数変数である．c_h についての詳細は後述する．図３における各記号の意味を表１に整理する．

表１．変数の表記と意味

記号	意味	配列サイズ(注)
	最適化変数	
x_{ct}	廃棄排熱量	(ih,im)
x_{hc}	HP の冷房出力量	(ih,im,Nh)
x_{hw}	HP の暖房出力量	(ih,im,Nh)
c_h	HP の運転切替（0=冷房，1=暖房）	(im,Nh)
	定数（負荷データ）	
D_e	電力負荷量	(ih,im)
D_c	冷房負荷量	(ih,im)
D_w	暖房負荷量	(ih,im)
	その他のエネルギー量	
y_{ce}	購入電力量	(ih,im)
y_{cf}	購入ガス量	(ih,im)
y_{ge}	総 GE 発電量	(ih,im)
y_{gt}	総 GE 排熱量	(ih,im)
y_{hec}	HP 冷房用電力量（全 HP 積算）	(ih,im)
y_{hew}	HP 暖房用電力量（全 HP 積算）	(ih,im)
y_{atc}	総 AR 冷房用排熱量	(ih,im)
y_{gw}	排熱利用暖房量	(ih,im)
y_{ac}	総 AR 冷房出力量	(ih,im)
	エネルギー効率	
η_{ge}	GE 発電効率	(ih,im)
η_{gt}	GE 排熱効率	(ih,im)
η_{hc}	HP 冷房効率	(ih,im,Nh)
η_{hw}	HP 暖房効率	(ih,im,Nh)
η_{ac}	AR 冷房効率	(ih,im)

（注）プログラム（Fortran）での記述にあわせた．Nh：HP 台数，im：月，ih：時．

以上の準備により，コスト関数である投資回収年数を最適化変数により記述できる．なお，最適化変数は，上で定義したもの以外を採ることも差し支えない．最適化変数によりその他のエネルギー量をすべて記述できることが肝要である．

まず，冷房負荷についての関係から，AR の冷房出力量を記述できる．

$$y_{ac} = D_c - \sum_{N_h} \left((1 - c_h) \cdot x_{hc} \right) \qquad \text{総 AR 冷房量＝冷房負荷量－総 HP 冷房量} \qquad (1)$$

同じく，暖房負荷についての関係から，排熱による暖房量を記述できる．

$$y_{gw} = D_w - \sum_{N_h} \left(c_h \cdot x_{hw} \right) \qquad \text{排熱利用暖房量＝暖房負荷量－総 HP 暖房量} \qquad (2)$$

次に，排熱についての関係から，必要な排熱量を記述できる．

$$y_{gt} = y_{atc} + y_{gw} + x_{ct} \qquad \text{総 GE 排熱量} \qquad (3)$$

$$= \frac{y_{ac}}{\eta_{ac}} + y_{gw} + x_{ct}$$

＝総 AR 冷房用排熱量＋排熱利用暖房量＋廃棄排熱量

式(3)より，購入ガス量は GE 排熱量を排熱効率で除して求められる．また，GE 発電量はその購入ガス量に発電効率を乗じる．

$$y_{cf} = \frac{y_{gt}}{\eta_{gt}} = \frac{1}{\eta_{gt}} \cdot \left(\frac{y_{ac}}{\eta_{ac}} + y_{gw} + x_{ct} \right)$$

購入ガス量 (4)

$$y_{ge} = \eta_{ge} \cdot y_{cf}$$

総 GE 発電量 (5)

最後に，電力についての関係から，

$$y_{ce} = D_e - y_{ge} + y_{hec} + y_{hew} = D_e - y_{ge} + \sum_{N_h} \left(\frac{1}{\eta_{hc}} \cdot (1 - c_h) \cdot x_{hc} \right) + \sum_{N_h} \left(\frac{1}{\eta_{hw}} \cdot c_h \cdot x_{hw} \right)$$

(6)

購入電力量＝電力負荷量－総 GE 発電量＋HP 冷房用電力量＋HP 暖房用電力量

以上で，すべてのエネルギー量を最適化変数および定数で記述できた．

各装置の容量についても，年間のピーク出力量がすなわち装置容量となるので，上述のエネルギー量を使って記述できる．

$$s_g = \frac{\max_{ih,im}(y_{ge})}{N_g}$$

GE 容量（ピーク発電量，N_g は GE 台数） (7)

$$s_a = \frac{\max_{ih,im}(y_{ac})}{N_a}$$

AR 容量（ピーク冷房出力量，N_a は AR 台数） (8)

$$s_h = \max_{ih,im} \left((1 - c_h) \cdot x_{hc} + \frac{\eta_{hc}}{\eta_{hw}} \cdot c_h \cdot x_{hw} \right)$$

HP 容量（ピーク冷房出力量，年間のピーク出力が暖房の場合は冷房換算） (9)

各装置の初期コストは装置容量により決定するので，式(7)-(9)をもとに，CGS の初期コストを記述できる．ここでは関数表記しておくが，実際は市場の実勢価格から定めた近似式である．

$$y_{ci} = f_g(s_g) \cdot N_g + f_a(s_a) \cdot N_a + \sum_{N_h} f_h(s_h)$$

CGS 初期コスト (10)

同様にランニングコストも関数表記しておくが，電力の基本料金，従量料金は購入電力量から，都市ガスの定額基本料金，流量基本料金，従量料金は購入ガス量をもとに記述できる．実際には，関東の電力会社，ガス会社の標準的な料金体系を使用している．

$$y_{cr} = f_e(y_{ce}) + f_f(y_{cf})$$

CGS 年間ランニングコスト (11)

以上により，比較モデルの初期コストを B_i，年間ランニングコストを B_r とすると，投資回収年数は以下のように記述できる．コスト関数 U_{econo} は，最適化変数 x_{ct}，x_{hc}，x_{hw}，c_h の関数となる．

$$U_{econo}(x_{ct}, x_{hc}, x_{hw}, c_h) = \frac{y_{ci} - B_i}{B_r - y_{cr}} \qquad \text{投資回収年数} \tag{12}$$

2.2. 整数変数 c_h の実数変数化. 最適化変数のうち，整数変数である c_h は，HP の運転切替を表し，値が 0 なら冷房，1 なら暖房となる．HP の冷房または暖房の出力量は，c_h と x_{hc} または x_{hw} との積で表す（図 4）．

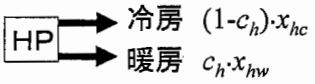


図 4．整数変数 c_h

高次元アルゴリズムで整数変数を最適化変数として扱う場合，通常は整数変数の実数変数化を行う．ここでも，一旦 c_h を実数変数とし，$0 \leqq c_h \leqq 1$ の値を取ることを許す．これはすなわち，1 台の HP から冷房と暖房が同時に出力されている状態を許すことになる（現実的にはありえない）が，最終的に c_h を整数値に収束させることにより，現実と整合する．

ここでは，c_h を 0 か 1 に収束させるために，新たにコスト関数 U_{digit} を追加した．簡単に言うと，このコスト関数は，ある HP のある月について，24 時間積算した冷房量と暖房量のより多い方へ c_h を運動させる力を生じる．

$$U_{digit}(c_h) = \cos\left(\frac{\pi}{2} \cdot \frac{\text{冷房日積算量} - \text{暖房日積算量}}{\text{冷房日積算量} + \text{暖房日積算量}}\right) \qquad c_h \text{用コスト関数} \tag{13}$$

U_{digit} は，最小値 0 のコスト関数であり，冷房日積算量，暖房日積算量のいずれかが 0 になると最小値を取る．すなわち，HP の運転切替が決定する．

なお，計算プログラムの中では，U_{digit} の U_{econo} に対する重みを図 5 のように調節している．高次元アルゴリズムの繰返し計算のうち，N1 回目までは U_{digit} の重みを 0 とし，N1 回目から線形に重みを大きくし N2 回目で重み w とする．実際の計算では，N2 回目の時点で c_h が 0 または 1 に収束していない場合もあるが，計算時間（実用上）の制約から，強制的に 0 または 1 に確定させる．このときの判定基準にも日積算量の多寡を利用している．これまで試行した負荷データでは，N1，N2，w のパラメータは固定した値でよく（負荷データにあまり依存しない），強制的な確定操作も―最適化計算の厳密性とは反する面もあるが―問題は生じていない．

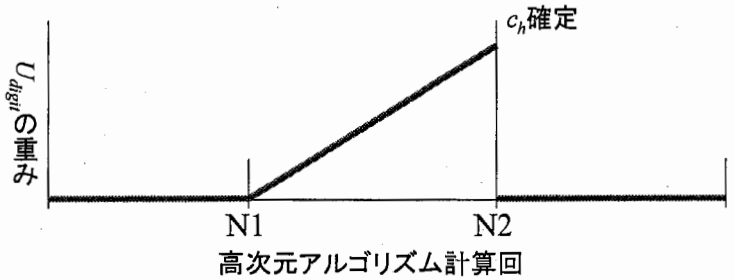


図 5．U_{digit} の重み調節

3．力学系の構成

ハミルトニアンは次の形式となる．ここで，q_iは最適化変数x_{ct}，x_{hc}，x_{hw}，c_hの位置，p_iは最適化変数の速度を表す．また，Tは運動エネルギー，V，Lはそれぞれコスト関数U_{econo}，U_{digit}に相当するポテンシャルエネルギーである．なお，混合性についての特別な考慮はしていない．

$$H = T(p_i) + V(q_i) + L(q_i) \tag{14}$$

$$T = \frac{1}{2}\sum p_i^2 \tag{15}$$

4．運動方程式

運動方程式も標準的なニュートンの運動方程式に沿っている．

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \tag{16}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{17}$$

具体的には，各最適化変数はコスト関数から以下のような力を受けて運動する．

（例１）最適化変数x_{hc}がコスト関数U_{econo}から受ける力

$$\frac{\partial U_{econo}}{\partial x_{hc}} = \frac{1}{B_r - y_{cr}} \cdot \frac{\partial y_{ci}}{\partial x_{hc}} + \frac{y_{ci} - B_i}{(B_r - y_{cr})^2} \cdot \frac{\partial y_{cr}}{\partial x_{hc}} \tag{18}$$

$$\frac{\partial y_{ci}}{\partial x_{hc}} = \frac{\partial f_g}{\partial x_{hc}} + \frac{\partial f_h}{\partial x_{hc}} + \frac{\partial f_a}{\partial x_{hc}} \tag{19}$$

$$\frac{\partial y_{cr}}{\partial x_{hc}} = \frac{\partial f_e}{\partial x_{hc}} + \frac{\partial f_f}{\partial x_{hc}} \tag{20}$$

（例２）最適化変数x_{hc}がGEの初期コスト分から受ける力（a_{g1}，a_{g2}は初期コスト近似式の係数）

$$\frac{\partial f_g}{\partial x_{hc}} = \left(\frac{2 \cdot a_{g1} \cdot s_g}{N_g} + a_{g2} \right) \cdot \eta_{ge} \cdot \left(-\frac{(1 - c_h)}{\eta_{gt} \cdot \eta_{ac}} \right) \tag{21}$$

5．プログラム

高次元アルゴリズムのプログラムは，既定回数もしくは収束判定されるまで繰返し計算を行う．おおまかに言うと，ある初期値から始めて，コスト関数値（ポテンシャル）の計算，力の計算，最適化変数の運動計算を繰り返す．

5.1．コスト関数値の計算．式(1)-(12)をもとに，順次，各エネルギー量，各コストを計算し，最後に投資回収年数を計算する．各エネルギー量はすべて，エネルギーの流れや装置の入出力の関係から最適化変数により記述されているので，プログラムはごく単純なものとなる．

```
C-----------------------------------------------------------------------
C     CGS諸量の計算
C-----------------------------------------------------------------------
      subroutine Condition()
      include 'CGSCOMMON.fi'
      (中略)
C     Yhec        HP冷房用総電力量 (N_Hour,N_Month)
C     Yhew        HP暖房用総電力量 (N_Hour,N_Month)
C     Yac         ARの冷房出力量 (N_Hour,N_Month)
C     Ygw         排熱利用暖房量 (N_Hour,N_Month)
C     Ygtw        暖房に使われるGE排熱量 (N_Hour,N_Month)
      do im=1,N_Month
         do ih=1,N_Hour
            wXhc=0
            wXhw=0
            do iHP=1,Nh
               wXhc=wXhc+Xhc(ih,im,iHP)*(1-Ch(im,iHP))
               wXhw=wXhw+Xhw(ih,im,iHP)*Ch(im,iHP)
            enddo
            Yhec(ih,im)=wXhc/Hhc
            Yhew(ih,im)=wXhw/Hhw
            Yac(ih,im)=Dc(ih,im)-wXhc
            Ygw(ih,im)=Dw(ih,im)-wXhw
            Ygtw(ih,im)=Ygw(ih,im)/Hgw
         enddo
      enddo
      (中略)
C     Ygi         GE初期コスト
C     Yhi         HP初期コスト (N_HP)
C     Yai         AR初期コスト
C     Yci         CGS初期コスト
      Ygi=flCge(Sg*SWatt)/SYen
      wYhi=0
      do iHP=1,Nh
         Yhi(iHP)=flChp(Sh(iHP))/SYen
         wYhi=wYhi+Yhi(iHP)
      enddo
      Yai=flCar(Sa)/SYen
      Yci=Ygi+wYhi+Yai
C     Yceb        電力基本料金
C     Ycev        電力従量料金 (N_Hour,N_Month)
C     Ycfd        ガス定額基本料金
C     Ycfb        ガス流量基本料金
C     Ycfv        ガス従量料金 (N_Hour,N_Month)
C     Ygr         GEメンテナンスコスト (N_Hour,N_Month)
C     Ycr         CGSランニングコスト
      (中略)
      Ycr=Yceb+wYcev+Ycfd+Ycfb+wYcfv+wYgr
C     Uecono      投資回収年数
      Uecono=(Yci-Bi)/(Br-Ycr)
      return
      end
```

5.2. 力の計算. 式(18)-(21)のように，コスト関数 U_{econo} について，各最適化変数による偏微分を行い，それを各最適化変数に対するコスト関数からの力として計算する．なお，実際のプログラム中では，エネルギー量は負荷データの最大値で規格化し，料金等の金額は百万円単位で計算している．

以下のプログラム例において，配列 FXct の値は最適化変数 x_{ct} にかかる力であり，サブルーチン dYci_Xct（初期コスト分）と dYcr_Xct（ランニングコスト分）により，各コストから受ける力がこの配列に加算される．

```
C-----------------------------------------------------------------------
C     力の計算
C-----------------------------------------------------------------------
      subroutine Force()
```

```
      include 'CGSCOMMON.fi'
      do im=1,N_Month              ! force をゼロクリア
         do ih=1,N_Hour
            FXct(ih,im)=0
     (中略)
      enddo
      w1=1/(Br-Ycr)
      w2=(Yci-Bi)/((Br-Ycr)*(Br-Ycr))
      call dYci_Xct(w1)
      call dYcr_Xct(w2)
      call dYci_Xhc(w1)
      call dYcr_Xhc(w2)
      call dYci_Xhw(w1)
      call dYcr_Xhw(w2)
      if(IDigital) then            ! N2 回目まで計算
         call dYci_Ch(w1)
         call dYcr_Ch(w2)
         if(IDigitalForce) then ! N1 回目から計算
            call dUCh_Ch()
         endif
      endif
      return
      end
C-----------------------------------------------------------------------
C     微係数 fg/Xhc : XhcがGE初期コストから受ける力
C-----------------------------------------------------------------------
      subroutine dfg_Xhc(factor)
      include 'CGSCOMMON.fi'
         w=(2*Ag1*(Sg*SWatt)/Ng+Ag2)
     &         *Hge(MHg,MMg)/(-Hgt(MHg,MMg))/Hac(MHg,MMg)
     &         /SYen*factor       ! GE出力のピーク月:MMg, 時:MHg
         do iHP=1,Nh
            FXhc(MHg,MMg,iHP)=FXhc(MHg,MMg,iHP)+w*(1-Ch(MMg,iHP))
         enddo
      return
      end
```

5.3. 最適化変数の運動計算. 各最適化変数にかかる力を計算した後，その力を加速度とする単位時間 dt 分の運動を Verlet 法を用いて計算する．運動計算後は，最適化変数の新しい値を使って，コスト関数値を再計算する（5.1.に戻る）.

5.3.1. 反射：変数を常に境界内に留まらせる. 冷房等のエネルギー量である最適化変数 x_{ct}, x_{hc}, x_{hw} は 0 以上でないといけない．また，実数変数化した最適化変数 c_h も 0 以上 1 以下の範囲に運動を留まらせる必要がある．ここでは，単位時間 dt の運動の結果，最適化変数が境界を外れる場合に，プログラム上で境界を壁に見立て反射させる.

5.3.2. 総和を一定以下に保つ. HP の冷房出力の総和が冷房負荷を超えてはいけない．また，暖房出力の総和も暖房負荷を超えてはいけない．ここでは，単位時間 dt の運動の結果，その総和が負荷量を超える場合に，プログラムにより強制的にスケール変換を行う．実際の計算では，その時点の最適化変数 x_{hc}, x_{hw} の値に比例して負荷量を配分している.

$$D_c \geq \sum_{N_h}\left((1-c_h)\cdot x_{hc}\right) \tag{22}$$

$$D_w \geq \sum_{N_h}\left(c_h \cdot x_{hw}\right) \tag{23}$$

5.3.3. 運動エネルギーを減衰させる．第2章5.1.節の(c)にあるように，プログラム上で運動エネルギーの減衰処理を行っている．ここで減衰に用いる係数は，0.9995 である．

本プログラムは建物設計の現場での実用化を予定しており，できるだけ短時間で結果を返す必要がある．ここでの減衰処理は，徐々に運動エネルギーを減少させ運動領域を狭めることにより，短時間で最適解（らしきもの）へ収束することを期待したものであり，今のところ期待どおりの効果が得られている．

以下は運動計算のプログラムの抜粋である．網掛け部分が反射等それぞれの処理に対応している．中ほどで，新しい x_{hc} 値を計算している行の右辺で使っている dK 項が減衰処理のための係数である．

```
C-------------------------------------------
C     dt 分の運動
C-------------------------------------------
      subroutine Move()
      include 'CGSCOMMON.fi'
      call MoveXct()
      call MoveXhc()
      call MoveXhw()
      if(lDigitalMove) then
        call MoveCh()
      endif
      call Redivide()     ! 負荷≧HP 出力総和の確認 5.3.2.
      return
      end
C-------------------------------------------
C     変数 Xhc の dT 分の運動
C-------------------------------------------
      subroutine MoveXhc()
      include 'CGSCOMMON.fi'
      do iHP=1,Nh
        do im=1,N_Month
          do ih=1,N_Hour
            VXhc(ih,im,iHP)=(Xhc(ih,im,iHP)-pXhc(ih,im,iHP))/dt
     &                -FXhc(ih,im,iHP)*dt
            wXhc=pXhc(ih,im,iHP)
            pXhc(ih,im,iHP)=Xhc(ih,im,iHP)
            Xhc(ih,im,iHP)=2d0*VXhc(ih,im,iHP)*dt*dK+wXhc 5.3.3.
          enddo
        enddo
      enddo
C     値域の制限による反射 5.3.1.
      do iHP=1,Nh                    ! Xhc>=0
        do im=1,N_Month
          do ih=1,N_Hour
            if(Xhc(ih,im,iHP).lt.0) then
              Xhc(ih,im,iHP)=-Xhc(ih,im,iHP)
              VXhc(ih,im,iHP)=-VXhc(ih,im,iHP)
              pXhc(ih,im,iHP)=Xhc(ih,im,iHP)-VXhc(ih,im,iHP)*dt
            endif
          enddo
        enddo
      enddo
      return
      end
```

6．計算結果

標準的なホテル 1 万㎡の負荷データ（給湯負荷を暖房負荷に加算）について， 最適化計算を行った結果を表2にまとめる．図6は本プログラムのデモシステムの画面例である．

なお，本プログラムによって得られる装置の運転方法や投資回収年数についての評価は，文献(2)-(6)に詳しいので，そちらを参照されたい．

表２．計算結果

計算機	Pentium Ⅱ/450MHz Windows98, DigitalFortran6.0
計算条件	高次元アルゴリズム計算を 20 万回 GE1 台，HP2 台，AR2 台
計算時間	約 10 分
負荷データ	ホテル 10,000 ㎡（給湯負荷込） ～空気調和・衛生工学会
投資回収年数	2.15 年

図６．デモシステム画面例

文献

(1)空気調和・衛生工学会編，“都市ガスによるコージェネレーションシステム計画・設計と評価”丸善，1994.
(2)山田，新上，北川，柳，植草，“ハミルトニアンアルゴリズムによるコージェネレーションシステムの最適化”，信学技報 EE98-35，pp.33-40，1998.
(3)柳，植草，山田，新上，“Optimal Design of Cogeneration System by Using Hamiltonian Algorithm”，Building Simulation '99, pp.699-706, Kyoto
(4)柳，植草，寺前，新上，“ハミルトニアンアルゴリズムによるコージェネレーションシステム最適設計　第１報：システムのモデル化”，日本建築学会大会梗概集，pp.871-872，1999.9.
(5)柳，植草，寺前，新上，“ハミルトニアンアルゴリズムによるコージェネレーションシステム最適

設計　第2報：有効性の評価”，日本建築学会大会梗概集，pp.873-874，1999.9.
(6)柳，植草，山田，新上，“高次元アルゴリズムによるコージェネレーションシステムの最適設計”，日本建築学会計画系論文集2000年12月掲載予定

第 9 章 分子構造の決定

概要

熱力学的に安定な内部エネルギーを持つ分子構造を探すことが目的である．ここではエネルギーが極小となる分子内の各原子の位置を求めている．

一般に構造最適化で用いられる非線形最適化手法の Newton 法が成功しない場合についても，高次元アルゴリズムで最適化することができることを述べる．また，原子の位置座標を高精度，且つ，少ない繰り返し計算回数で求めるために，エネルギーの勾配ベクトルが大きい領域を，高次元アルゴリズムで探索した後，準 Newton 法に切り替えることで，収束性を向上できることも示す．

1．問題の説明

分子の 3 次元構造は，材料の物性や機能と直接的に結び付く．身近な話題では，水道管や卵パックに使われるポリ塩化ビニルは，長い炭素鎖にぶら下がる格好で付く塩素が，隣の塩素に対して同じ方向に結合しているのか，上下の向きが反対に結合しているのかという違いで，ガラス転移温度が変わり，加熱した時の粘弾性が異なる．また，人の視覚は，シス型のレチナールという折れ曲がった分子が，光反応で直線形の全トランス型のレチナールに構造変化することがトリガーとなって光を知覚する．生体内の情報伝達に関わる分子は，構造に敏感に反応するゲスト - ホスト (鍵と鍵穴) の関係から機能発現する機構を多く持ち，工業的には医薬品に関わっている．

このように分子構造を知ることは，機能材料を始め，プラスチック，医薬品，農薬などの化学工業全般，それを応用する電子工業などに渡って重要となっている．分子構造は，実験的には核磁気共鳴スペクトル，赤外スペクトルや X 線回折などで測定・解析されるが，測定が困難な化学反応過程の動的解析のみならず，新規物質の探索や物性の予測という場面で，計算によって構造を求めることが役に立つ．

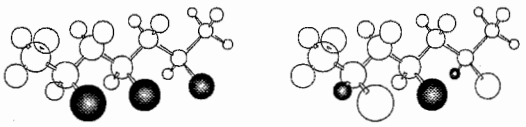


図 1．ポリ塩化ビニルの構造

長い炭素鎖に塩素原子 (黒い球で示す) が付いているが，同じポリ塩化ビニルでも，各々塩素原子の付き方が異なるために物性値が異なる．

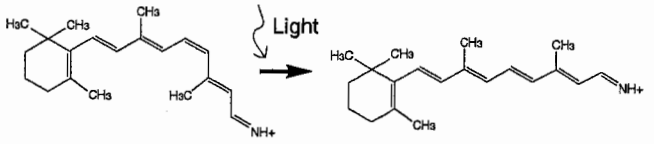
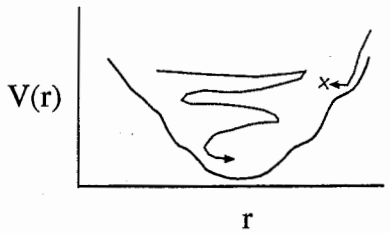


図 2．視物質の発色分子の光化学反応

11-cis-retinal は光異性化反応で all-trans-retinal へ構造変化することで，光受容たんぱく質の構造を変化させ，最終的に電気信号として伝わる．

物質探索のめざす一つの方向は，目標とする物性値や機能を持つ物質を計算機によって探索することである．このような方法で新しい物質を探索する場合，その分子構造を計算によって求めたいことが生じる．しかし，予め，最適化構造に近い，良い初期座標を与えることは困難なため，一般に分子構造の最適化計算で用いられる Newton 法では，最適化を失敗することを実際に多く経験する．局所解に捕らわれ難く，多変数の非線形最適化に対して強い高次元アルゴリズムは，新しい物質の探索における分子構造の最適化に向いている．



最適化変数 r について，コスト関数 $V(r)$ の極小値の探索で，Newton 法は，コスト関数を探索点近傍で 2 次関数に近似して極小点を求めるのに対し，高次元アルゴリズムは大域的な探索を行うため，局所的な関数形状の影響を受け難い．

図 3．高次元アルゴリズムのよる探索

2．最適化問題の設定

ここでは，分子を構成する原子種と，任意に各原子の位置 (初期座標) を与えた時，それら原子がどう配置するのかを最適化問題として解いていく．分子を構成する原子の位置によって，各々の核や電子による斥力と引力が異なるので，分子の内部エネルギーが異なる．安定な分子構造は，原子の位置座標を最適化変数とし，分子のエネルギーをコスト関数とした最適化問題を解くことで決定する．分子のエネルギーは分子軌道法 [1] で数値的に求める．

3．力学系の構成

3.1. ハミルトニアン：混合性とエネルギー減衰. ハミルトニアンは，運動項，運動に混合性を導入する正定値対称行列とコスト関数から構成する (式 (1))．運動によって大域的に探索する中，局所領域に留まらせ，そこでの収束解を得るために，ハミルトニアンに時間とともに運動量を減衰する項を導入する (式 (2))．

$$H = \frac{1}{2}\sum a_{ij}P_iP_j + V(r_i) \tag{1}$$

$$P_i = p_i - \int^t \kappa(t')\ \dot{r}(t')\ dt' \tag{2}$$

ここで P_i, P_j は運動項，a_{ij} は混合性を導入する正定値対称行列 (全ての固有値が正である対称行列)，$V(r_i)$ はコスト関数 (分子のエネルギー)，r_i は最適化変数である分子の原子座標 (x_i, y_i, z_i)，κ はエネルギー減衰係数．コスト関数について最適化を行うため，$a_{ij}P_iP_j \geq 0$ とする (詳細は **3.3** 節に示す)．

3.2. 混合性の導入. 運動に混合性を持たせるために，次の手順で混合性行列 (混合性を導入する正定値対称行列を混合性行列と呼ぶことにする) を作る．この方法は混合性を与えるための 1 つの例である．まず，混合性行列 $\{a_{ij}\}$ を適当に与えるため，単位行列と非対角項に適当な乱数からなる対称行列を作り，これに対してシュミットの直交化を行う．次いで，直交化で得られた直交ベクトルを用い，正定値 ε_μ を適当に与え ($\{a_{ij}\}$ の固有値に負の要素があるとポテンシャル面が反転する不都合を生じるため: **3.3** 節を参照)，混合性行列 $\{a_{ij}\}$ を作る．

混合性を導入する行列 $\{a_{ij}\}$ を作る手順

① 次の行列 $\mathbf{E}$ を準備する．

$$\mathbf{E} = \mathbf{I} + \mathbf{R} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix} + \begin{pmatrix} 0 & & \text{乱数} & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix} \tag{3}$$

ここで $\mathbf{I}$ は単位行列，$\mathbf{R}$ は非対角要素が適当な乱数からなる対称行列．

② この行列 $\mathbf{E}$ をシュミットの直交化を行い，直交ベクトル $|\mu\rangle$ $(\mu = 1, 2, \cdots, N)$ を得る．

③ 正定値 ε_μ を適当に与え (例えば 0.5 ～ 1.5)，求めた直交ベクトルから a_{ij} を得る．

$$\mathbf{A} = \{a_{ij}\} = \sum_\mu |\mu\rangle\ \varepsilon_\mu\ \langle\mu| \tag{4}$$

④ $\mathbf{AB} = 1$ となる b_{ij} を用意する．

$$\mathbf{B} = \{b_{ij}\} = \sum_\mu |\mu\rangle\ \frac{1}{\varepsilon_\mu}\ \langle\mu| \tag{5}$$

⑤ ハミルトニアンは次の便利な形に書き換えできる．

$$H = \frac{1}{2}\sum b_{ij}\dot{r}_i\dot{r}_j + V(r_i) \tag{6}$$

このような対称行列 $\{a_{ij}\}$ を使うことで，多くの場合，運動の混合性を得ることができる．シュミットの直交化の代りに，対角化で固有値，固有ベクトルを求める方法もある．対角化の場合には，非対角項の要素が 0 の場合，固有ベクトルに値が残ってしまうので，シュミット直交化の方が非対角項の要素が 0 の場合とそうでない場合で連続的に繋がる．

3.3. 正定値対称行列. 運動の混合性を導入するための正定値対称行列は，その固有値 ε_μ の全てが正値となる対称行列である．仮に負の ε_μ がある場合には，以下のような不都合が生じる．

例えば $\{a_{ij}\}$ に負の固有値がある場合で，$i=1$, $j=1$ の簡単な場合を考える．

この時のハミルトニアンは

$$H = -\frac{1}{2}P^2 + V(r) \tag{7}$$

これから導かれる運動方程式は

$$\dot{r} = \frac{\partial H}{\partial P} = -P \tag{8}$$

$$\dot{P} = -\frac{\partial H}{\partial r} = -\frac{\partial V}{\partial r} \tag{9}$$

式 (8), 式 (9) から

$$\ddot{r} = -\frac{\partial(-V)}{\partial r}\ . \tag{10}$$

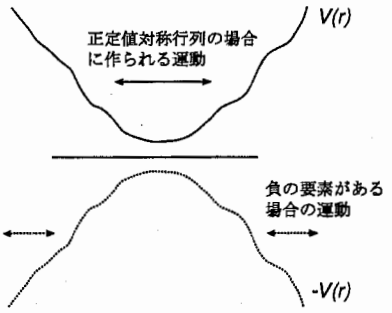


図 4．正定値であることの必要性

式 (10) の運動は，ハミルトニアンが $H = \frac{1}{2}P^2 - V(r)$ の時の運動と同じである．従って，本来，コスト関数 $V(r)$ を極小化するのが目的であるが，対称行列が負の固有値を持つ場合には，$-V(r)$ の極小化を解いていることになる．

3.4. 拘束条件がある場合の注意：混合性行列と反射. 本章の「分子構造の決定」では反射は使用していないが，混合性行列を用いて，拘束条件付きの問題を解くために「反射」を導入する場合には，次の取り扱いをする．

混合性行列を用いたハミルトニアンは次のように構成した．

$$H = \frac{1}{2}\sum b_{ij}\dot{r}_i\dot{r}_j + V(r_i) \tag{11}$$

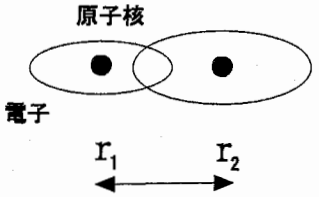
ここで拘束条件のために、反射 $\dot{r}_i \rightarrow -\dot{r}_i$ を導入すると，混合性行列の非対角項によりハミルトニアンが一定にならない．

$$H(\dot{r}_i, r_i) \neq H(-\dot{r}_i, r_i) \tag{12}$$

これを避けるために，反射を入れない場合と入れた場合の運動量を各々計算し，ハミルトニアンに係数をかけることで，運動量を一定に保つ．

$$H(\dot{r}_i, r_i) = \alpha H(-\dot{r}_i, r_i) \tag{13}$$

3.5. コスト関数. コスト関数は分子軌道計算から求める分子のエネルギーで与える．分子軌道法 (1) の詳細は省略するが，分子のエネルギーは，原子核へ作用するエネルギーとして，他の核からの反発エネルギーと周囲の電子から受けるエネルギーから構成され，それらのエネルギーは，最適化変数 r_i (各原子の位置) が動くことで変化する．コスト関数は「陽に関数形のない場合」に分類されるが，コスト関数である分子のエネルギーは，r_i が決まれば数値的に求まる．



核へ作用するエネルギーは，
① 他の核からの反発エネルギー
② 周囲の電子から受けるエネルギー
からなり，各原子の位置によって分子のエネルギーが変化する．

図5．最適化変数 r_i と分子のエネルギー

分子軌道は，次の Hartree-Fock-Roothaan 方程式 (式 14) を分子軌道の規格直交条件下 (式 15) で解くことで得られる．

$$\mathbf{FC} = \mathbf{C}\varepsilon_D \tag{14}$$

$$\mathbf{C^T SC} = \mathbf{I} \tag{15}$$

ここで $\mathbf{F}$ は Fock 行列，$\mathbf{C}$ は固有ベクトル (分子軌道の係数)，ε_D は固有値 (分子軌道のエネルギー)，$\mathbf{S}$ は重なり積分，$\mathbf{I}$ は単位行列．

Fock 行列を対角化し，固有値と固有ベクトルを得るが，Fock 行列を作るためには，固有ベクトルから求められる密度行列が必要となる．この非線形性のため (Fock 行列の中に，求める固有ベクトルの知識が使われている)，適当な固有ベクトルから作った Fock 行列から出発し，前回に求めた固有ベクトルとの差が十分に小さくなった SCF 状態 (Self Consistent Field: 自己無頓着な場) になるまで，繰り返す．

4．運動方程式

前述の通り，ハミルトニアンは次のように構成した．

$$H = \frac{1}{2}\sum a_{ij}P_iP_j + V(r_i) \qquad \left(= \frac{1}{2}\sum b_{ij}\dot{r}_i\dot{r}_j + V(r_i) \right) \tag{16}$$

$$P_i = p_i - \int^t \kappa(t')\ \dot{r}(t')\ dt' \tag{17}$$

これから次の運動方程式を得る．

$$\dot{r}_i = \frac{\partial H}{\partial p_i} = \sum_j a_{ij}P_j \tag{18}$$

$$\dot{p}_i = -\frac{\partial H}{\partial r_i} = f_i \tag{19}$$

$$\ddot{r}_i = \sum_j a_{ij}(f_j - \kappa\dot{r}_j) \tag{20}$$

ここで，f_i は r_i についての力，κ はエネルギー減衰項．分子構造の最適化では，$\kappa = 5.0$ とした．
解の探索は，初期値 $r_i, \dot{r}_i$ を与え，それによって生じる r_i の変化から次の運動を得ることを繰り返すことで行う．r_i に対する力は，分子軌道計算で求める分子のエネルギーであるコスト関数から求める．
運動量はエネルギー減衰項 $\kappa\dot{r}$ によって徐々に下がり，やがて収束に達する．

5．計算プログラム

分子構造最適化に用いた計算プログラムの高次元アルゴリズムによる探索エンジン部分を FORTRAN 言語で示す．実装するのにプログラムの行数が少なくて済むのも高次元アルゴリズムの特長である．

5.1 混合性を導入する行列の作成.

```
subroutine mixmat
gamma=0.5d0                              (1)
do  i=1,3*numatm
amix(i,i)=1.0d0                          (2)
do j=1,i-1
xg=gamma*ran(ii)
amix(j,i)=xg                             (3)
amix(i,j)=xg
enddo
enddo
call schmit(amix,3*numatm)               (4)
do i=1,3*numatm
do j=1,3*numatm
v(j,i)=amix(j,i)                         (5)
enddo
enddo
x0=0.6d0
x1=1.4d0
estep=(x1-x0)/float(3*numatm-1)          (6)
do i=1,3*numatm
e(i)=estep*float(i-1)+x0
enddo
do i=1,3*numatm
do j=1,3*numatm
amix(i,j)=0.0d0
bmix(i,j)=0.0d0
do k=1,3*numatm
amix(i,j)=amix(i,j)+v(i,k)*v(j,k)*e(k)   (7)
bmix(i,j)=bmix(i,j)+v(i,k)*v(j,k)/e(k)   (8)
enddo
enddo
enddo
```

(1) 乱数に掛ける係数 $\gamma = 0.5$
($\gamma = 0$の時、混合性なし)
(2) 対角要素 ($= 1$)
(3) 非対角要素：乱数による設定
(4) シュミットの直交化サブルーチンを CALL
(5) 直交性ベクトルの代入
(6) 正定値を適当に与える ($0 < \varepsilon_\mu$)
(7) 混合性行列 $\mathbf{A}$ を計算する
(8) 混合性行列 $\mathbf{B}$ を計算する

5.2 力の計算とエネルギー減衰.

```
subroutine force_r
call force                               (1)
do n=1,numatm
f_r0(n,1)=f_r(n,1)-xkappa*vr(n,1)
f_r0(n,2)=f_r(n,2)-xkappa*vr(n,2)        (2)
f_r0(n,3)=f_r(n,3)-xkappa*vr(n,3)
enddo
do i=1,numatm
f_r(i,1)=0.0d0
f_r(i,2)=0.0d0
f_r(i,3)=0.0d0
i1=3*(i-1)+1
i2=3*(i-1)+2
i3=3*(i-1)+3
do j=1,numatm
j1=3*(j-1)+1
j2=3*(j-1)+2
j3=3*(j-1)+3
a11=amix(i1,j1)
a12=amix(i1,j2)
a13=amix(i1,j3)
a21=amix(i2,j1)
a22=amix(i2,j2)
a23=amix(i2,j3)
a31=amix(i3,j1)
a32=amix(i3,j2)                          (3)
a33=amix(i3,j3)
f_r(i,1)=f_r(i,1)+a11*f_r0(j,1)+a12*f_r0(j,2)+a13*f_r0(j,3)
f_r(i,2)=f_r(i,2)+a21*f_r0(j,1)+a22*f_r0(j,2)+a23*f_r0(j,3)
f_r(i,3)=f_r(i,3)+a31*f_r0(j,1)+a32*f_r0(j,2)+a33*f_r0(j,3)
enddo
enddo
```

(1) 原子座標 r_i に対する微分 (力) を計算するサブルーチンを CALL
(分子軌道法による電子状態計算)
(2) 求めた力に減衰を入れる
(3) 混合性行列 $\mathbf{A}$ から $\ddot{r}_i$ を計算する

5.3 運動とプログラム全体の流れ. 運動は *Verlet* 法によって，時刻 t における位置 $r(t)$, 前回の位置 $r(t-\Delta t)$ と，t における力から $t+\Delta t$ の位置を，次式で計算する.

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \ddot{\mathbf{r}}_i(t)\Delta t^2 \tag{21}$$

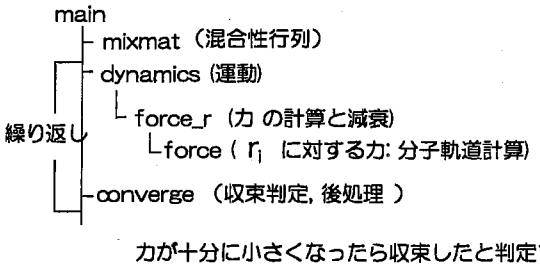
$$\mathbf{V}_i(t) = \Big(\mathbf{r}_i(t+\Delta t) - \mathbf{r}_i(t-\Delta t)\Big) \,/\, 2\Delta t \tag{22}$$

運動のプログラムを次に示す.

```
subroutine dynamics
call force_r
do n=1,numatm                                          (1)
r2(n,1)=2.0d0*r1(n,1)-r0(n,1)+dt*dt*f_r(n,1)
r2(n,2)=2.0d0*r1(n,2)-r0(n,2)+dt*dt*f_r(n,2)
r2(n,3)=2.0d0*r1(n,3)-r0(n,3)+dt*dt*f_r(n,3)
vr(n,1)=(r2(n,1)-r0(n,1))/(2*dt)
vr(n,2)=(r2(n,2)-r0(n,2))/(2*dt)                       (2)
vr(n,3)=(r2(n,3)-r0(n,3))/(2*dt)
enddo
```

(1) 前回の位置 $r_i(t-\Delta t)$ と現在の位置 r_i と力から，次回に動かす位置 $r_i(t+\Delta t)$ を計算する
(2) 運動の速度を計算する

プログラムの全体流れを次に示す.



プログラムの動きは，解を探索する運動を繰り返す中で，徐々に運動量が下がり，力も小さくなっていく．最終的に，力が十分に小さくなった時点で，収束解とする．その際，分子のエネルギーについての収束性や，各原子の移動距離についての収束性を含めた判定条件を課しても良い．また，力が十分に小さくなって来たら，収束精度を向上するため，Newton 法に切り替えても良い.

6．計算結果

6.1 最適化した分子構造 (最適化過程). 適当な初期構造から出発して，高次元アルゴリズムで構造が最適化されるまでの過程を示す.

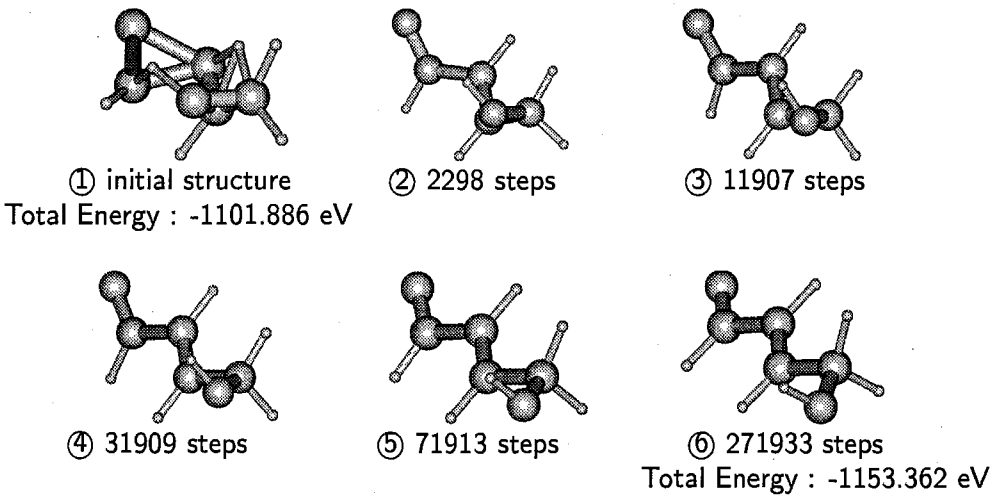


図 6．分子構造の最適化過程

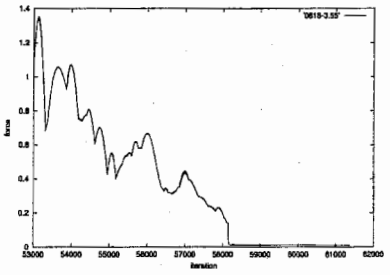
この例では運動のステップが約 70000 回で収束解を得た．最適化までの計算量を表 1 に示す．

表 1．構造最適化の計算量

繰り返し数	71913	111917	201926	271933
CPU Time [min]	34.1	45.9	67.4	80.6

計算機：Compaq Alpha station XP-1000　CPU：Alpha21264　667MHz(EV67)

6.2 高次元アルゴリズムと準 Newton 法のハイブリッド 高精度な収束解をより早く得るために，大域的な探索は高次元アルゴリズムで行い，収束に近づいたら，2 次関数で近似できる領域で探索効率が良い準 Newton 法に切り替えた．このハイブリッドを行うことで，互いの特長を相補する効率的な探索が行える．力の最大値を収束の判定基準にして，最終過程で準 Newton 法に切り替えた場合の収束過程の例を示す．



運動の繰り返し回数とともに力の最大値が減少する．最終段階で準 Newton 法に切り替えることで高精度に収束する．

図 7．ハイブリッドした場合の収束過程

文献

(1) 例えば，米沢 貞次郎ら，“量子化学入門 (上)”，化学同人．
(2) 大田原 一成，下川 信祐，新上 和正，“新物質の安定構造の探索”，情報処理学会第 60 回全国大会，6H-04(2000)．
(3) 大田原 一成，下川 信祐，新上 和正，“分子の安定構造の探索における収束性”，情報処理学会第 61 回全国大会，2Q-03(2000)．
(4) 大田原 一成，下川 信祐，新上 和正，“(新) 物質設計の手法: 双極子モーメントへの適用”，情報処理学会第 59 回全国大会，1M-4(2000)．
(5) Ohtawara,K., Shimogawa,S., and Shinjo,K., “ A Design Theory of Material System”, submitted.

第１０章　行列の固有値の探索

概要

対象とする系（＝行列）が時間発展を遂げており，系の変動が比較的小さい時に各時間ステップにおける固有値・固有ベクトルを精度を制御しながら，高速に求めるためのアルゴリズム（ＳＤ：Simulated Diagonalization）[(1)] を提案する．

ＨＡはもともと共有メモリ型ベクトル・並列計算機に向いたアルゴリズムを備えており，容易に効率の良い計算を行うことが出来，ベクトル化・並列化で短縮した時間内で，固有値・固有ベクトルを求めることが出来れば，十分に適用する価値があると思われる．

1. 問題の説明

分子の構造最適化問題においては，原子に掛かる力は電子状態から計算され，また電子状態は分子の構造から導かれる行列の固有値問題を解いて得られる．この時にもし構造の変化が微小であれば，それに伴って得られる行列の時間変化も微小である．しかし，従来からある固有値解法においては，与えられた行列の代数的変形を行うため，例え，正解に近い固有値・固有ベクトルの情報があってもそれらを生かして計算量を減らすことはできない．この時，固有値問題を最適化問題に置き換えることができれば，一つ前の時刻における系の固有値・固有ベクトルが次の時刻の系における初期値として正解の近傍に取ることができるため早い収束が期待できる．

2. 最適化問題の設定

2.1. コスト関数. コスト関数を定義するにあたり，固有値の探索問題を最適化問題に帰着させることを考える．

対称行列 F は適当な直交行列 $C(=\{\boldsymbol{c_\mu}\})$ を用いて対角化できる．${}^tC\cdot C = I$（単位行列）より，

$$\begin{cases} {}^t\boldsymbol{c_\mu}\,\boldsymbol{c_\nu} = 0, \quad ({}^t\boldsymbol{c_\mu}\,F\,\boldsymbol{c_\nu} = 0) & \text{as} \quad \nu \neq \mu. \\ {}^t\boldsymbol{c_\mu}\,\boldsymbol{c_\mu} = 1, & \text{as} \quad \nu = \mu. \end{cases} \tag{1}$$

つまり，対称行列 F の固有ベクトル $\{\boldsymbol{c_\mu}\}(=\{c_{i\mu}\})$ は正規直交性を持つ．そこで，それをそのままコスト関数 $(=V(\{c_{i\mu}\}))$ として採用し，

$$V(\{c_{i\mu}\}) \;=\; \sum_{\nu\neq\mu}\left\{\left|{}^t\boldsymbol{c_\mu}\,F(t)\,\boldsymbol{c_\nu}\right|^2 + \left|{}^t\boldsymbol{c_\mu}\,\boldsymbol{c_\nu}\right|^2\right\} + \sum_{\mu}\left|{}^t\boldsymbol{c_\mu}\,\boldsymbol{c_\mu} - 1\right|^2 \tag{2}$$

と定義する．第１項は，固有ベクトル系でない正規直交系を導かないようにするためである．

式（２）で与えたコスト関数の値を下げることにより，正規直交性を満たす真の固有ベクトルを探索することができるようになった．最適化変数は，すべての固有ベクトルのすべての成分である．また，$F(t)$ は，対象となる行列が時間発展することを示す．

上記コスト関数の値は，一般の $\{c'_{i\mu}\}$ に対して $0 \leq V(\{c'_{i\mu}\})$ となり（常に０か正を取るように右辺各項が２乗されている），$\{c_{i\mu}\}$ が $F(t)$ の真の固有ベクトルとなるそのときに限り，

$$V(\{c_{i\mu}\}) = 0$$

を満たす．

逆に，０という最小値を設定することにより，現在得られている固有値・固有ベクトルの精度が分かる．

その時，μ 番目の固有値 λ_μ は

$$\lambda_\mu = {}^t\boldsymbol{c_\mu}\,F(t)\,\boldsymbol{c_\mu}$$

で得られる．

実は，固有ベクトルが満たす条件を生かしたコスト関数の形には任意性がある．最小値を確定値（この場合は0）とすることができれば，右辺各項の巾には何を与えても良い．

さらに，部分的な固有値だけを求めることもできる．例えば，コスト関数を

$$V(\{c_{i\mu}\}) = \sum_{\substack{\nu\neq\mu \\ \mu,\nu=1}}^{m} \{|{}^t\boldsymbol{c}_\mu F(t)\,\boldsymbol{c}_\nu| + |{}^t\boldsymbol{c}_\mu\,\boldsymbol{c}_\nu|\} + \sum_{\mu=1}^{m} |{}^t\boldsymbol{c}_\mu\,\boldsymbol{c}_\mu - 1| \pm \underbrace{\sum_{\mu=1}^{m} {}^t\boldsymbol{c}_\mu F(t)\,\boldsymbol{c}_\mu} \tag{3}$$

とすれば，固有値を近似する項（最終項）を加える（差し引く）ことにより，最小（最大）の固有値からm番目までの固有値を求めることができる．また，それに対応するm個の固有ベクトルも求めることができる．この時の最適化変数は，m個の固有ベクトルの各成分だけからなっている．

最後に，式（2），（3）が示すように，コスト関数が単純な線型計算で構成されているため，高いベクトル性能と並列性能を容易に引き出すことができる．

2.2. 拘束条件. 拘束条件は，対称行列の固有ベクトルが持つ正規直交性（式（1））であり，一般に多い．それらは，既にコスト関数に組み込まれている．拘束条件をコスト関数に組み込むことによって生じる利点は，探索空間（拘束条件を満たす空間）を破る運動を生成することができることである．つまり，任意に与えた初期ベクトルに対して，真の固有ベクトルが，すべての拘束条件を満たすつながった空間に存在するとは限らないからである．

3. 力学系の構成

3.1. ハミルトニアン. 力学系を構成するハミルトニアンを

$$H(\{c_{i\mu}\},\{v_{i\mu}\}) = T(\{v_{i\mu}\}) + V(\{c_{i\mu}\})$$

で定義する．$c_{i\mu}$ は，求めるべき μ 番目の固有ベクトルの第 i 成分．$v_{i\mu}$ は $c_{i\mu}$ に対する運動の速度．右辺の $T(\{v_{i\mu}\})$ は生成された運動のエネルギー，右辺の $V(\{c_{i\mu}\})$ はポテンシャルエネルギーで，ここでは式（2）（または式（3））で定義した $\{c_{i\mu}\}$ によるコスト関数を用いる．運動エネルギー $T(v_{i\mu})$ が，$1/2\sum v_{i\mu}^2$ で与えられるため，ニュートンの運動方程式に変形される．

3.2. 混合性に対する工夫. 本適用例においては，混合性に対する工夫は特に行わず，

$$T(v_{i\mu}) = 1/2\sum v_{i\mu}^2$$

の基本系を用いた．特に混合性を生み出す工夫を行なわなくても収束させることができたためである．

4. 運動方程式

ハミルトニアンから導かれたニュートン方程式は

$$\frac{d^2c_{i\mu}}{dt'^2} = -\frac{\partial V(\{c_{i\mu}\})}{\partial c_{i\mu}}$$

で与えられる．実際の運動は行列 $F(t)$ が従う時間とは別の時間 t' を導入して，式（2）のコスト関数を各 $c_{i\mu}$ で偏微分することにより

$$\frac{d^2c_{i\mu}(t')}{dt'^2} = -4\sum_{\nu(\neq\mu)}\left\{\sum_j F_{ij}c_{j\nu}\left(\sum_{lm} F_{lm}c_{l\mu}c_{m\nu}\right) + c_{i\nu}\left(\sum_l c_{l\mu}c_{l\nu}\right)\right\} - 4c_{i\mu}\left(\sum_l c_{l\mu}^2 - 1\right) \tag{4}$$

で得ることができる．

式（2），（3）同様，各 $c_{i\mu}$ で偏微分された式（4）も単純な線型計算で構成されているため，高いベクトル性能・並列性能が期待できる．

5. プログラム

5.1. Verlet 法. ニュートンの運動方程式は，分子動力学法でよく用いられるヴェルレ法を用いて解く．ヴェルレ法で用いた式は，

$$c_{i\mu}(t'_+) = 2c_{i\mu}(t') - c_{i\mu}(t'_-) + \frac{d^2c_{i\mu}(t')}{dt'^2}(\Delta t')^2 \quad (5)$$

$$v_{i\mu}(t') = \frac{1}{2\Delta t'}[c_{i\mu}(t'_+) - c_{i\mu}(t'_-)] \quad (6)$$

で与えられる．ここで，今の時刻 t' に対して，次の時刻，前の時刻を t'_+，t'_- で表している．式（5）（6）における運動方程式を繰り返し解くことによって，コスト関数値がより低くなる最適化変数（固有ベクトル）を探索する．

ヴェルレ法はテーラー展開を2次までしか取っていないので一見あまり良い近似には見えないが，公式は $c_{i\mu}(t' \pm \Delta t')$ を展開して両式の和から座標を計算する．この時 $\Delta t'$ の1次の項は打ち消し合っている．そこで，$c_{i\mu}(t' \pm \Delta t')$ を4次まで展開して和を取っても $\Delta t'$ の3次の項は打ち消し合う．ゆえにヴェルレ法では4次まで展開して4次の項を無視した近似になっている．この意味で良い近似と言える．

式（5），（6）に対応するプログラムを図1に示す．

```
      do 100 mu=1,n1
      do 100 i=1,n1
      c2(i,mu)=2.0d0*c1(i,mu)-c0(i,mu)
     &          -(ff1(i,mu)*v1+ff2(i,mu)*v2+ff3(i,mu)*v3)
     &          *(dt*dt)
  100 vx(i,mu)=(c2(i,mu)-c0(i,mu))/(2.0d0*dt)
```

図 1: Verlet 法

ここで，ff1,ff2,ff3 は，コスト関数の3項に対応する力であり，式（4）の偏微分の計算から得られる値である．v1,v2,v3 は，力を正規化するための変数で，本固有値問題においては，

```
v1=50.0d0/f1max
v2=600.0d0/f2max
v3=600.0d0/f3max
```

で与えている．

5.2. 運動の減衰性. ヴェルレ法だけで運動方程式を解いていると，コスト関数値が低いところでは，エネルギーの保存則から運動エネルギーが増大し，うまく解に収束しない．そこで，式（7）における β の導入は，運動における摩擦のイメージに相当し，結果的にエネルギーが減衰していく運動を生成させることができる．

$$c_{i\mu}(t'_+) = c_{i\mu}(t'_-) + (2\Delta t')v_{i\mu}(t')\underline{\beta} \quad (7)$$

式（7）に対応するプログラムを図2に示す．

本固有値の探索問題では $\beta = 0.995$ を採用している．

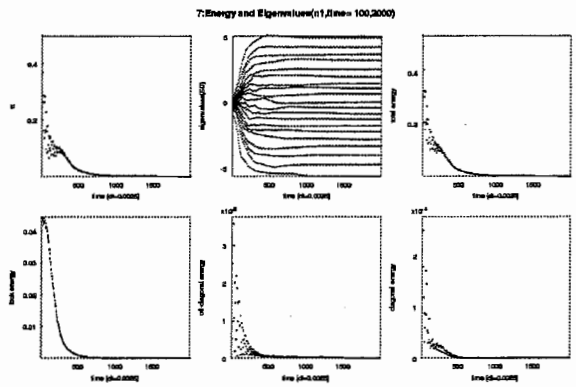
```
      do 200 mu=1,n1
      do 200 i=1,n1
      c2(i,mu)=c0(i,mu)+2.0d0*dt*vx(i,mu)*beta
  200 continue
```

図 2: 運動エネルギーの減衰

6. 計算結果

6.1. 計算例. ここでは，上記の方法を実際の行列に適用して固有値と固有ベクトルが得られることを示す．例として用いた行列は，一様乱数で得られた実対称密行列を用い，行列サイズ（次数）は１００とした．

図３左側は，コスト関数の値が減少していくに従って固有値が収束していく様子を示している．右側は，それぞれ左側のグラフが表す物理量を示している．



運動エネルギー　固有値　トータルエネルギー

コスト関数の第一項　コスト関数の第二項　コスト関数の第三項

図 3: コスト関数と固有値

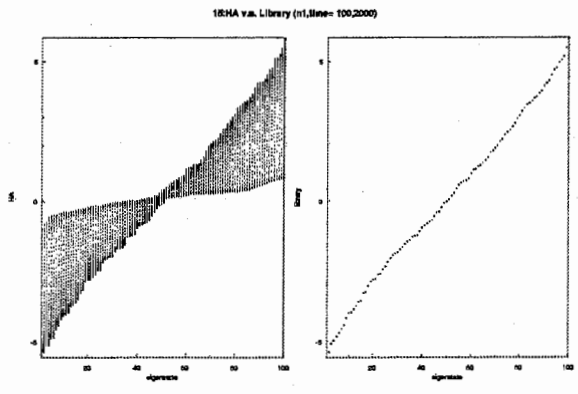


図４は，［－１，１］の一様乱数による初期ベクトル $\{c_{i\mu}\}$ から出発した近似固有値が，真の固有値（右図：ハウスホルダー法）に収束していくときの時間変化を示す（左図：ＳＤ）．

図 4: 固有値の運動の軌跡

6.2. 計算時間. ここでは，式（４）の右辺の１回の計算が並列化によりどれ程時間短縮されるかを報告する．用いた行列は，６．１で用いた行列と同じ行列で，行列サイズ（次数）は５００から４０００までの範囲で測定した．用いた計算機は，スーパーコンピュータＳＸ－４／８（ＮＥＣ社製）．参

照ソフトウェアとして科学技術計算ライブラリＡＳＬ（ＮＥＣ社製）を参考に比較を行なった．参照ソフトウェアのA.ＳＬは１ＣＰＵでのみ動く（プログラムの詳細は公開されていない）．

１ＣＰＵではＡＳＬのライブラリが良い性能を示しているが，ＳＤアルゴリズムでは全体の９９％を占める行列演算に並列化とベクトル化を実行させることにより，表１の様な性能が得られた．[注1]

解法＼次数	500	1000	2000	4000
SD	0.67 秒	5.3 秒	42 秒	337 秒
SD(自動並列化 (8cpu))	0.12 秒	0.79 秒	6.2 秒	49 秒
ASL[*1]	0.84 秒	5.4 秒	38 秒	285 秒

[*1]Householder 法 (メーカ製ライブラリ)

表 1: 計算時間

ＡＳＬは１ＣＰＵで，どのアルゴリズムに比べても高速である．これは，ベクトル機で高速になるようにチューニングされているためと思われる．ＳＤは１ＣＰＵではＡＳＬと同程度であるが，８ＣＰＵを使って計算を行なうと並列化が可能であるためにほぼＣＰＵ台数に比例した高速化が実現されている．また，行列の次数に依らず高速化が達成されている．例えば，行列の次数１０００～４０００で６．７～６．９倍である．これらは８ＣＰＵでの計算であることを考えると理想的な並列化率に近い値である．従って，ＳＤでは固有ベクトルを求めるのに逐次回数を高々６．７～６．９回以内で計算出来れば他のアルゴリズムに比べ高速に計算したことになる（固有値と固有ベクトルの計算の近似程度に依る）．

6.3 計算精度.(2) ＡＳＬとＳＤで計算した固有値と固有ベクトルに対して各々二つの差の２乗和の平方根を使って相対誤差（$r_\lambda(t)$：固有値；$r_c(t)$：固有ベクトル）で判定する．$r_\lambda(t)$ は時間を t_0 から始めて $t = t_0 + 64\Delta$ まではゆっくり増大する（$r_\lambda(t) \leq 10^{-3}$）がその時間を境に再び減少する．一方，$r_c(t)$ は t_0 から始めて $t = t_0 + 64\Delta$ を超えてもゆっくりと増大し続ける．時間ステップ Δ に対して増加分はほぼ $2 \cdot 10^{-4}$ 程度ある．

文献

(1) 野口孝明, 新上和正,“行列の固有値と固有ベクトルの近似解法について”, 情報処理学会第 55 回全国大会,3G-03

(2) 新上和正, 野口孝明,“時間変動する対称行列に適した近似固有値解法の適用”, 情報処理学会第 55 回全国大会,3G-04

注１ ＡＳＬは固有値と固有ベクトルを実際に計算しているのに対して，此所でのＳＤは式（7）の右辺だけを１回だけ計算していることに注意せよ．

第１１章　どんなモノがブレイクしてゆくのか?
(What kind of goods will be preferred?)

概要

高次元アルゴリズムは，簡単なプログラムで極小解や遷移状態に捕まりりにくい最適化計算を行なうことができる．手軽で強力なので，いろんな問題に使えてゆる可能性がある．実際，なんらかの問題を取り上げて，モデル化，定式化とすすめてゆけば，最適化問題が生じることがよくある．ここで，最適化手法が制約を持っていたり，プログラムを作るのが易しくないといったことがあると，最適化問題に定式化するときに，これらの事を考慮して工夫する必要があり，障害となってしまう．しかし，高次元アルゴリズムを利用する場合，プログラムが簡単でコスト関数に関する制約も最低限なので，後に続くプロセスと切り離して問題を定式化できる．このため，一見すると捉えどころの無いような問題でも，気軽に定式化に取り組める．

ここでは，一見つかみどころ無く曖昧に見える，『どんなモノがブレイクしてゆくのか』という研究に，高次元アルゴリズムを利用していく目論見を紹介する．

１．はじめに

『どんなモノがブレイクしてゆくのか?』－いくら高次元アルゴリズムが強力といっても，データを与えればこの答えを自動的に出してくれるといった，一般的なアルゴリズムが作れるわけではない．ブレイクしそうなモノを考えだしたりイメージすることを強力に支援してくれるツールを，高次元アルゴリズムを使って開発することを目論んでいる．ブレイクするモノを考え出してゆくためのツールとは，一体どのようなものであるべきだろうか? その手がかりを知るために，まず，『どんなモノがブレイクしてゆくのか?』という問題の背景について理解しておこう [3,4].

今日，どんなモノがブレイクするのかなど，永遠普遍の関心をあつめてきた難問のように思われるかもしれない．しかし，ちょっと思い起こしてみれば，バブル景気真最中には，関心の無かった分野も多かった．この点から，今度は逆に単に景気の問題で一時的で偶然の問題だと思われるかもしれない．しかし，これも正しくない．何故なら，携帯電話など不景気にも関らず著しい普及をみせているモノも少なくない．それでは，永遠普遍でも一時的な偶然でも無いこの問題は，一体なにものだろうか? それは，時代が進むと共に熟成されて来た歴史的必然である人とモノの関係の複雑化が表面化したものであり，将来に向かってなげだされた問題である．以下では，この歴史的必然について説明しよう (図１).

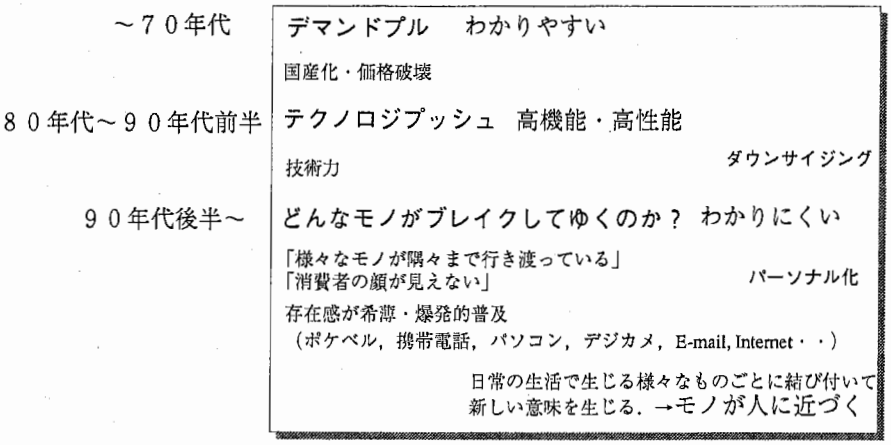


図１．人とモノの関係の複雑化.

まず，70 年代までは，ブレイクしてゆくことが十分に予期できるモノが多い．冷蔵庫，洗濯機，掃除機，(固定) 電話，テレビ，自家用車など，消費者にニーズがあるのはあきらかだった．問題は，資

本力と一定の質を出せる人の確保，改良を重ね価格と品質を消費に耐える水準にすることなどだった(デマンドプルの時代)．この時代，人とモノの関係は消費者・生産者のいずれの立場からも見通せた．80年代に入ると，既存のモノを技術力で高度化・多様化させ，おびただしい種類のモノが溢れるようになった(テクノロジプッシュ)．ここでは，モノと人の関係が生産者・供給者の立場から作られて行った．

テクノロジプッシュはバブル経済によって絶頂期となり，その崩壊と共に長いモノ余りの不況が訪れて終わりを告げる．『必要なモノが隅々まで行き渡っている』，『消費者の顔が見えない』などと言われるようになった．その一方で，女子高生などによるポケベルの爆発的な利用が出現するなど，携帯電話普及の流れが形作られた．今日，人とモノの関係は，技術者があらかじめ想定する類のものではない．この関係は，例えばダウンサイジングなどによって接近し深まってきており，実際に利用することを通して初めて知り得るという複雑なものとなっている．こうして『どんなモノがブレイクしてゆくのか?』という問題が，一般的なものとなって来た．高度な技術で良いものを作ることではなく，人とモノの複雑な関係というジャングルに分け入って将来の市場をイメージすることが，新たな関係を作りだす時代になっている．

2．現地調査，デザインコンセプト，ツール

私達は，複雑になってきた人とモノの関係を，現地調査を通じて分析し[3]，ブレイクするためのデザインコンセプト[1,4]を抽出した．その上でこれを実現して行くことを支援するツールを検討している．高次元アルゴリズムは，強力でプログラムが容易なので，インプリメントする手法を気にせずに構想することができる．

図2．現地調査．

ショッピングセンター，および，コミュニケーションツールの利用の2点を重点的に調査した(図2)．ショッピングセンターは，人とモノが結び付くか否かの最前線である．日常に特化した従来の手法が行き詰まり，超大規模化やゲーム感覚など様々な試みによって大きな変化が繰り返されている．個別の店舗の視察を中心に調査した．ポケベルに始まるコミュニケーションツールの新たな潮流は，女子高生を中心にブレイクし，携帯電話の隆盛を築くと共に，インターネットやITの将来との関わりは，いまや万人の関心の向かう所となった．アンケート調査を中心に，彼女らの利用イメージを分析した．

調査・分析を通じて，私達は，ブレイクするためのデザインコンセプトとして，

“驚き”　　“鳥瞰”

“プライベート性”(3)　　“儀式性”(2)

の４つを提案する．

次に，これらのデザインコンセプトを，未来へ開かれた形で実現するイメージを得るためのツールが必要である．その一つは人とモノの関係を含む隠れた様々な相互作用を過去，現在，未来に沿って取り出す道具立てである(図３)．例えば，個人のメンタルバリアを左右する要因を知ることは，プラベート性を備えたモノのイメージを想起するための方法になってゆく．

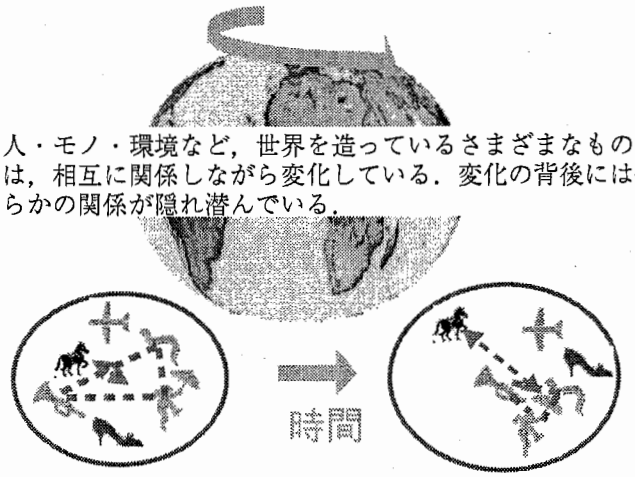


図３．隠れた関係を取り出すツール．

３．ツールの構成

具体的には，隠れた関係を時間軸に沿った関数関係として取り出す．これらの要素のどれかの将来値を，様々な必ずしもここに乗っていない要素（言葉，絵，映像，事件発生など）を表わす変数の現在までの値の写像として取り出す(図４)．つまり，予測関数として取り出す．（多くの場合Ｘは Ｙを含む）

このような事が可能と考えられるのは，インターネット・ITなどにより，大量のデータが収集可能となったこと，大規模なデータ解析をして予測関係を取り出すのに必要な強力で簡便な最適化アルゴリズム(高次元アルゴリズム)を利用できること，計算機の性能が向上してきたことなどがあげられる．

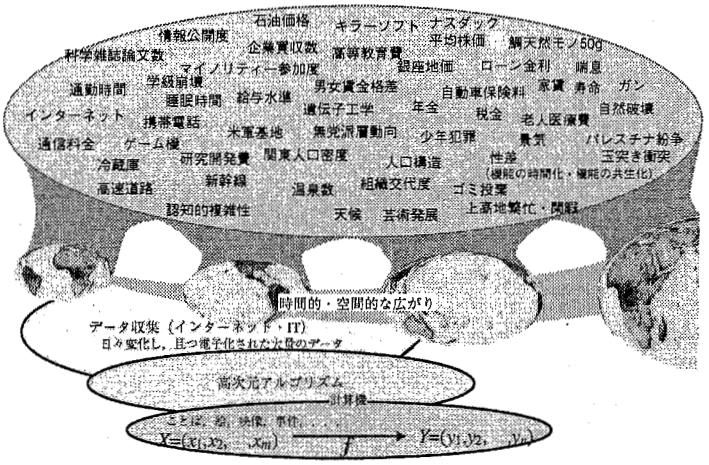


図４．隠れた関係を写像として取り出す．

４． 高次元アルゴリズムの適用

コスト関数の計算は，おおよそ，次のようになる．まず，変数の選択をする．一見すると数量に表現しにくいようなイベント情報なども，言葉を文章データベースで数値化する手法などを適用すれば，数量として表現が利用できる．関連する変数を設定するのにも同様の手法が利用できる．次に，関数を表現する計算モデルを決める．関数としての表現力に明らかな制約がないことが重要であり，問題に応じて適宜モデルを選ぶ．データは，現在の値と過去の値からなるが，これを，将来の値と現在までの履歴値と見なして，関数を２乗誤差最小などにより最適化する．

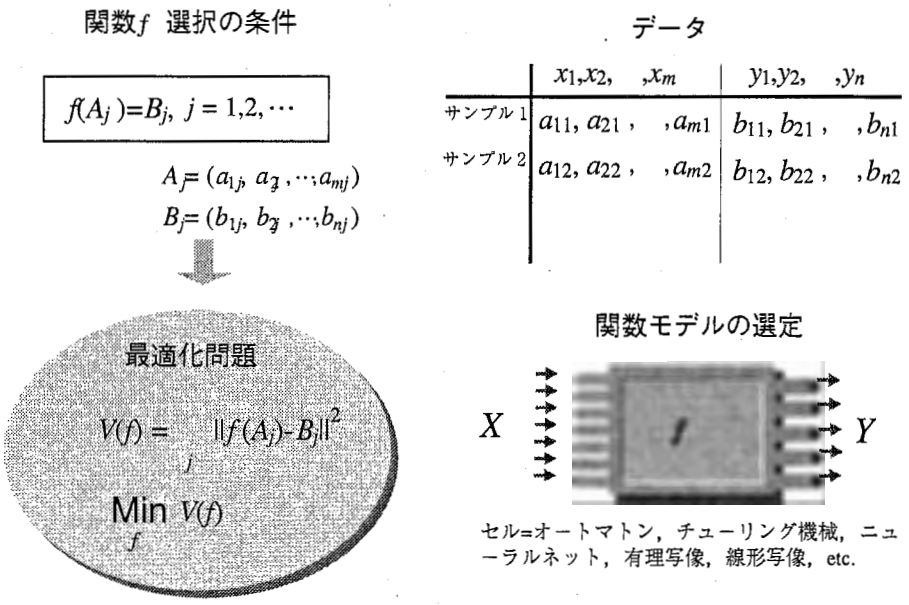


図５．高次元アルゴリズムによる写像の構成．

大規模で不揃いのデータを対象にすることになるが，これは，最適化において，大規模化と多重極小解の出現を意味し，高次元アルゴリズムのような強力な手法が必要である．

なお，以下の点が課題となって来る．まず，一つの変数の取る各数値それぞれが物理的な次元などを越えた異質な内容を表すため，コード化の違いに敏感な要因を取り除くことが必要となる．また，利用できるデータは取り出すべき関係に関する可能な相互関係から見たとき，おおよそ偏ったものであり，

偏りに対する予測値劣化の依存性を把握する必要がある.

5. まとめ

高次元アルゴリズムを用いて，隠れた関数を抽出するツールを開発し，その支援により，人とモノの関係を含む隠れた様々な相互作用を過去，現在，未来に沿って取り出して，デザインコンセプト - “驚き” “鳥瞰” “プライベート性” “儀式性” - を，未来へ開かれた形で備えるモノ・サービスイメージの開発を目指す.

文献

(1) 新上和正，“横断するデザイン：どんなモノがブレイクするか？，” NEC HPC 第 1 1 回研究会，6 月 12 日，東京・田町，2000.

(2) 下川，大田原，“人とモノの新たな関係のためのデザイン戦略，ーどんなモノがブレイクするのか？，” 情報処理学会第 6 1 回（平成 12 年後期）全国大会講演論文集 (4), pp.223-224(2000).

(3) 下川，新上，大田原，“プライベート性，新時代のデザイン戦略ー，” 情報処理学会第 6 0 回（平成 12 年前期）全国大会講演論文集 (4)，pp. 247-248(2000).

(4) 下川，大田原，“インターネットを含むコミュニケーションツールの利用調査とユーザー意識 – 人とモノの新たな関係を築くために –，” 電子情報通信学会誌，Vol.83, No.10, pp.740-742(2000).

第１２章 むすび

本テクニカルレポートは，『高次元アルゴリズム』の原理，使用法を解説し，応用例を紹介した．本アルゴリズムは，(1) 多くの変数を持つ問題に有効である，(2) 局所的な安定解に捕まり難い，(3) パラメータの注意深い制御を必要としない，(4) 問題を変形する必要がない，(5) ベクトル・パラレル計算処理に向いている，などの利点を持つ．我々はこれまで，このアルゴリズムをいろいろな領域の様々な問題に適用してみるという姿勢で研究を進めてきており，それぞれの領域で新しい知見を得ることができた．しかし一方では，他の最適化手法との定量的な比較については必ずしも十分な知見を得ているわけではない．

本レポートの読者が，『高次元アルゴリズム』の原理，使用法を理解し，具体的な問題に適用していただければ，本レポートが意図する所の大部分は達成されたと思われる．さらには，本アルゴリズムを他の手法と公平な立場で比較，評価していただければ幸いである．本アルゴリズムについての御意見，御批判をお待ちする次第である．

なお，本テクニカルレポート，本アルゴリズムに関する御質問，御意見，御批判については，

haeg@acr.atr.co.jp

宛およせいただきたい．また，各論についての問い合わせ先も列挙しておく．

第１，１２章	斎藤　茂	saito@acr.atr.co.jp
第２章	新上　和正	shinjo@acr.atr.co.jp
第３章	恩田　和幸	onda@acr.atr.co.jp
第４章	種田　和正	oida@acr.atr.co.jp
第５章	小松崎　彰	zaki@acr.atr.co.jp
第６章	平田　和貴(*)	haeg@acr.atr.co.jp
第７章	倉持　裕	kuramo@acr.atr.co.jp
第８章	北川　美宏	kitagawa@acr.atr.co.jp
第９章	大田原　一成	ohtawara@acr.atr.co.jp
第１０章	野口　孝明(**)	noguchi@acr.atr.co.jp
第１１章	下川　信祐	simogawa@acr.atr.co.jp

(*)　現在，富士ゼロックス株式会社

(**)　現在，関西日本電気ソフトウェア株式会社