Vol.2011-HPC-129 No.14
2011/3/16

自動チューニング数理基盤ライブラリ ATMathCoreLib

須田 礼仁[†1,†2]

本論文では自動チューニングの数理コアライブラリについて論ずる．まず，自動チューニングの数理的構成要素について分析し，それを明確化する手法として 4DAC を提案する．本論文で論ずる 4DAC は，著者が以前に提案したものを改良したものである．4DAC は，4 つの D，4 つの A，4 つの C からなる．4 つの D は，コスト，チューニングパラメタ，特徴量など，チューニング対象となるソフトウェアの具体的な要素を定める．4 つの A は，コスト推定関数モデル，条件予測の仮定，目的関数など，チューニング対象の数理的・理論的モデルを定める．4 つの C は，コスト推定方式，条件予測，実験計画など，チューニングの数理手法を実装する．自動チューニング数理ライブラリは，仮定する 4 つの A を明確化した上で，4 つの C を実装する．利用者は，チューニング対象のソフトウェアの 4 つの D を特定し，ソフトウェアの特性を分析し，適合する 4 つの A の仮定をもつ自動チューニング数理ライブラリを用いる．

次に，上記 4DAC に基づいて構築された自動チューニング数理ライブラリ ATMathCoreLib を提案する．現在の ATMathCoreLib は，オンライン自動チューニングのための線形モデル構築とワンステップ近似による実験計画を提供する．本論文では，ATMathCoreLib が仮定する 4 つの A を述べ，4 つの C の実装の概略を説明する．また，ATMathCoreLib の基本的な使い方とともに，無限希釈，ランダムサブセット法への応用方法を説明する．

ATMathCoreLib: Mathematical Core Library for Automatic Tuning

Reiji Suda[†1,†2]

This paper discusses mathematical core library for automatic tuning. First, mathematical elements of automatic tuning are analyzed, and 4DAC, which is a methodology of clarification of those elements, is proposed. This paper proposes a new 4DAC, improved based on what the author has proposed. 4DAC consists of 4 Ds, 4 As and 4 Cs. 4Ds define concrete elements of the target software of tuning, such as costs, tuning parameters, and features. 4As define mathematical and theoretical model of the tuning target, such as model of cost estimation functions, assumptions of future conditions, and the objective function. 4Cs define the mathematical methods of automatic tuning, such as construction of cost estimation functions, prediction of future conditions, and experimental design. Mathematical library for automatic tuning should clarify assumed 4 As, and implement 4Cs. Users of the library should define 4Ds of the target software of tuning, analyze the properties of the software, and use mathematical library which assumes matching 4As.

Next, this paper proposes ATMathCoreLib, a mathematical core library for automatic tuning designed based on the 4DAC model discussed above. The current implementation of ATMathCoreLib provides methods of construction of linear models and experimental design with one-step approximation for online automatic tuning. This paper clarifies 4As which ATMathCoreLib assumes, and overviews implementations of 4Cs. The basic usage of functions of ATMathCoreLib is explained, as well as applications to infinite dilution and random subset method.

1. はじめに

1.1 自動チューニングとは

近年，ハードウェアの多様性が高まっている．主流 CPU アーキテクチャでは，およそ半年ごとに新しい機種が出る．マルチコアにおける階層的なキャッシュの共有の仕方，複数 CPU のメモリへの接続の仕方などには，複数の方式がある．GPGPU のような CPU とは異なるアーキテクチャのアクセラレータプロセッサを用いた計算も広く行われている．並列計算機になると，規模や相互接続網は千差万別であるうえ，同一のシステムであっても，システムの一部が割り当てられる場合に，どのような部分が割り当てられるかによって，性能等に影響を及ぼす場合がある．HPC の主流が COTS で構成されるようになったことで，HPC アーキテクチャの変化速度が加速しているのである．

このような HPC 計算機アーキテクチャの多様性のため，HPC ソフトウェアを手動でチューニングするということに限界が来ている．重要なソフトウェアを少数選定し，特定の計算機をターゲットとして，特定のアロケーションを想定し，その前提のもとで知識と経験のある専門的なプログラマがソフトウェアをチューニングすれば，非常に高い性能を達成

†1 東京大学情報理工学系研究科 / Graduate School of Information Science and Technology
†2 JST, CREST / CREST, JST

することができるであろう．しかし，よく知られているように，世界トップのスーパーコンピュータの性能は 10 年で約 1000 倍， 1 年で約 2 倍という速度で向上している．専門家を投入して多大な人的コストをかけてチューニングしても，特定のハードウェアだけに有効なのであれば，1 年後に 2 倍，2 年後に 4 倍という性能ペナルティを追うことになる．

今後は，消費電力や発熱・冷却の問題が HPC の最重要課題となることが予想されている．また耐故障性への本格的な対応が必要になるとも考えられている．これらの問題に対してソフトウェアがどのような対応をすることになるのか，我々は確固たる見通しを持っていない．これらの問題が最も顕著に表れる世界トップクラスの HPC ソフトウェアでは特に，その構成方式について大きな変更が余儀なくされる可能性がある．

このような複雑，不確定で変動するハードウェア条件に対応するためには，**ソフトウェアが適応性を持つ**ということが望まれる．

この適応性を，我々は**自動チューニング**という言葉で呼んでいる．**自動チューニングはパラダイムである**．ソフトウェアがソフトウェアをチューニングするという方針のもとで，様々な技術がありうる．自動チューニングは 1990 年代ごろからこの名称により研究されてきた若い分野である[1)–3)]．これから一層の技術開拓が必要とされている．

この適応性は，さまざまなところに実装されうる．例えば，それぞれの CPU に対してコードを生成する最適化コンパイラ，システムごとにチューニングされたライブラリ，あるいは，アプリケーションプログラムのソースコードなどにおいて，適応性を実装することができる．これらの実装方法は排他的ではなく，これらの適応性が相互補完的に協調動作することが最も望ましい．

この適応性は，**全自動的**にソフトウェアだけで人手を介さず動作するようにも実装できるし，**半自動的**にユーザやプログラマが介入することで，人間が持つ知識を援用するようにも実装できる．全自動的な適応は人手の介入を必要としない利点があるし，半自動的な適応は人間の知識を利用できるという利点がある．

この適応性は，それが動作するシステムの外で発生する情報を利用することができる．例えば同一のアーキテクチャに基づく他のシステムにおいて観測された性能情報を参照することにより，効率的な最適化が可能になると期待される．また，この適応性は，一回的に発揮されるものとは限らない．例えばソースコードのコンパイルは一度だけとは限らず，実行時情報を収集しつつ何度もコンパイルをしなおしてもよい．また，この適応性は，適応性の実装自身が固定的なものであるとは限らない．新たな情報と知見に基づいて，適応性自身が更新されることが望ましい．

この適応性は，必ずしも一致しない複数の目的関数を持ちうる．例えば，各ユーザは自分のタスクの実行時間（待ち時間）を最小化したいと思う一方で，計算システムのスループットを最適にするスケジューリングとは一致しない可能性がある．また，消費電力，エネルギー，冷却限界には相互関連があるが，これらを目的関数とする最適解は異なる可能性がある．また，特定の目的関数を定めたとしても，最適解は，ハードウェアとソフトウェアだけではなく，データにも依存しうる．ユーザが扱うデータが変化すると，違うところに最適解が来る可能性がある．

1.2 自動チューニングの利点と可能性

さまざまな条件に対応した，適応性を有するソフトウェアと，特定の条件を仮定して人手でチューニングされたソフトウェアを比較する場合には，比較の前提となる仮定条件をどう選ぶかが重要である．特定の条件に対して最適化された手法は，より広い条件で適用できる一般的な手法に比べて，必然的に有利である．平たく言えば，「自動チューニングが手動チューニングにかなうはずがない」．なぜなら，自動チューニングが生成するコードを，人手でコーディングすることは常に可能だからである．人手でチューニングされたソフトウェアがどれだけの条件下でどれだけの性能を発揮するのか，それにどれだけの人的コストが割かれたのか，などの要因を含めて多面的に評価することが必要である．

人手でチューニングしたものよりも自動チューニングが高い性能を挙げる場合がある．多くの場合，そのような事例では，単純な探索をひたすら繰り返すのは人手よりも計算機の方が有利であるという特徴が表れている．別の言い方をすれば，実際には人的コストが有限であるということから生じている．他方で，ひらめきや勘により優秀な解を発見するとか，まったく未知の新しい環境に適応して解を導く能力は，計算機よりも人間の方が優れている．手動チューニングのほうが自動チューニングよりも原理的な適応能力が高く，事例を限れば自動チューニングより高い性能を出すはずである．これに対して自動チューニングの利点は以下の 2 つが考えられる．

(1) 自動チューニングは適応性がソフトウェアに内在しており，異なる条件が与えられたときに人手を介さずに適応できる．
(2) 機械的で網羅的な探索は計算機の得意とするところであり，人手で探索しきれていない選択肢を見出す可能性がある．

著者は利点 (1) に重点をおいて研究を行っているが，利点 (2) に注目する研究者も多い．ただし利点 (2) では次の 4 点について注意しておく必要がある．第 1 に，網羅的探索もコストがゼロではないので，探索には限界がある．第 2 に，人手で探索できる範囲というのは，

人的コスト（社会的要因）に影響される．自動チューニングの評価においては，自動チューニング機構をソフトウェアに組み込む人的コストも含めて評価が必要である．第 3 に，人間が見落としていた解が発見されたとしても，その時点で既知になる．第 4 に，人間がどのような探索をするかは未知であるため，利点 (2) を一般的かつ定量的に論ずることは難しく，個別の事案についての議論にならざるを得ない．

さて，利点 (1) については，2 通りの使い方が考えられる．

- 変動する条件に対して，その都度自動的に適応できる．
- 未知の条件に対して，即座に適応できる．

著者は主に前者を想定して研究を行ってきたが，後者についても期待されるところである．しかし後者については技術的に容易でないし，具体的な技術もほぼ未開拓と言ってよい．段落を改めて少し論ずる．

ハードウェアアーキテクチャの可能性は無限にある．その中で限られたアーキテクチャのみに対象を絞れば，それに対して有効性の高いチューニング手法の集合というものを考えることができる．今後出てくる未来のハードウェアにおいて非常に有効なチューニングが，現在のハードウェアではまったく性能改善に寄与しないという可能性もある．そのようなチューニングを現時点で考慮に入れても，探索空間を無駄に広くするだけで，効果に乏しい．どのような適応性を組み込むべきかは，対象とするハードウェアアーキテクチャなどの現在の条件に依存するのである．組み込む適応性を選択するという作業は，これまで人手で行われており，この点が自動チューニングの手動チューニング的要素であり，またシステム依存的な要素となっている．未知の環境における**組み込むべき適応性と探索方式の自動構築**という可能性はあるが，未開拓の領域である．別な方式として，新たなアーキテクチャが登場したときに，新たな適応性と探索手法を組み込むことができるような枠組みをあらかじめ準備しておくというものが考えられる．すなわち**事後拡張性のある自動チューニングの枠組み**であるが，こちらも未開拓の技術課題である．

1.3 自動チューニングの技術的課題

自動チューニングの基本的アイデアは，ソフトウェアに可変性を実装し，その可変性を適切に調整することで，もっとも高い性能を確保しようとするものである．従って，自動チューニングの技術的課題は，**可変性の実装**と**可変性の制御**という 2 つの部門に分けて考えることができる．

可変性の実装は，高い性能を導くプログラムを開発する**個別のチューニング技術**と，その**プログラミング実装**の 2 点に分けて考えることができる．自動チューニングはソフトウェアの処理内容を変えずに，性能のみを改善することを，基本的に目指している．このため，個別のチューニング技術の開発は，同一の処理をしながら記述と性能の異なる**プログラム変種**を開発することとなる．このようなプログラム変種は，プログラマが直接記述することもできるし，ABCLibScript[3] のような自動チューニング記述言語によって半自動的に生成することもできるし，最適化コンパイラやプログラム変換器などが自動的に生成することもできる．変種コードの自動的な生成はプログラミングの負担を軽減するが，システムが提供する以外の変種を生成できないという欠点を常に持つ．完全に自動ではない変種コードの生成を許容することにより，ドメインの知識や新しい知見をチューニングに取り込むことができる．一方で，このようなプログラム変種の記述と生成をプログラミング言語システムがサポートすることも否定しない．ソフトウェアの開発コストや再利用性という問題は自動チューニングの重大課題として取り組みながら，他方で手動によるチューニングの必要性も主張する点で，自動チューニングは最適化コンパイラとは異なる．開発の容易なプロトタイプ的なものから，高性能を追求した本格的実装まで，逐次に詳細化できるようなプログラミングシステムが望まれる．

可変性の制御は，性能を測定しながら，実装された可変性を調整して，高い性能を実現することを目指すものである．可変性の制御の実現には，可変性を調整する機構および性能を測定する機構をソフトウェアシステムとして実装する**ソフトウェア基盤システム**と，測定した性能から高性能な変種を効率的に発見する**数理**とが必要である．

本研究はこれらの課題のうち，数理の部分に焦点を絞り，数理手法のライブラリとしての実装方式について論ずるものである．

2. 自動チューニングの数理技術の定式化

2.1 旧 4DAC[4]

著者は発表 4) において自動チューニングの数理の実現のための具体的構成について論じた．そこでは著者は，それまで主に扱ってきたオンライン自動チューニングについて考察し，自動チューニングの数理的構成要素を **4DAC** という形でまとめた．それをここで引用すると，以下の通りである．

4 つの D D は Definitions の D である．これら 4 つの D は，自動チューニングの数理モデルが対称とする実体に対応している．これらは可変性の実装において定義あるいは抽出されるもので，可変性の制御においては与えられたものとなる．例えばシステム特徴量やデータ特徴量は，多数のものがありうるが，それらをすべて利用するのがよいとは

限らない．コスト関数やチューニングパラメタも同様である．すでに定義されているものから,「何を選択するか」が問題ということもできる．4 つの D は以下の通りである．

D1 コスト関数
D2 チューニングパラメタ
D3 システム特徴量
D4 データ特徴量

4 つの A A は Assumptions または Analysis の A である．これら 4 つの A は，自動チューニングの数理的モデル化に対応している．これらはチューニングの対象となるものの中に内在しないものであり，自動チューニング機構の実装者あるいはユーザが定義や選択をしなければならないものである．著者らの研究以外では，A2 のコストモデリング以外は明確に論じられていないことが多いが，これらを明確にしないことには，自動チューニングの問題が数理的に明確に定式化されない．4 つの A は以下の通りである．

A1 擾乱（特徴量を固定したときに現れるコストの変化）に関する仮定
A2 コストモデリング（チューニングパラメタ，特徴量とコストの関係）
A3 条件の予測（未来の実行条件，実行回数などの予測）
A4 目的関数（オンライン，オフラインなど）

4 つの C C は Computation または Components の C である．これら 4 つの C は，自動チューニングの数理的手法（解法）に対応している．上記の 4 つの D と 4 つの A が定義されると，自動チューニング問題が数理的問題として定式化される．この数理的問題に解を与えるための構成要素が以下の 4 つの C である．すなわち，自動チューニングの数理コアライブラリが提供するものにほからならない．4 つの C は，数理的定式化である 4 つの A に対して，その解を与えるものであるため，4 つの A のうちひとつでも異なれば，4 つの C がすべて異なるものが必要となる可能性がある．4 つの C は以下の通りである．

C1 データ解析（擾乱に対応）
C2 コストモデルの学習
C3 実験計画
C4 最適化（実験計画の中で使われる）

上記 4DAC の提案ののち，著者[5] はチューニングパラメタ制御関数を用いた自動チューニングやオフライン自動チューニングについて考察した．特に，同論文では 3 つの自動チューニングの方式があることを指摘している．**C2P0** は，コスト推定関数に基づいてチューニングパラメタを設定するもので，著者が数年前より取り組んでいるモデルである．**C0P2** は，コスト推定関数なしにチューニングパラメタ制御関数を直接設定するもので，機械学習に近い．**C2P2** は，コスト推定関数を構築し，それにあわせてチューニングパラメタ制御関数を調整する．このような数理的定式化の展開のため，前節で論じた 4DAC も再検討が必要となっている．また，この 5) でも議論が不十分なものとして，特徴量や性能相関，敵対的モデルと中断，時間変化などの問題が残っている．

本論文では，特徴量とチューニングパラメタ制御関数について注意して，定式化を再検討し，4 つの D と A と C を修正再定義する．

2.2 4 つ の D

4 つの D は自動チューニングのソフトウェアとしての実装に実在するものであり，かつ最適化の数理に直接かかわる部分である．項目としては，従来の D3 と D4 が両方とも特徴量であるため，まとめた．さらに，システムに関する各種条件を加えて 4 つとする．

2.2.1 D1: コスト関数

自動チューニングの目的は，何らかのコストを最小化することであると仮定する．何か最大化したい指標がある場合には，逆数を取る・符号を変えるなどして最小化に帰着させる．これらのコストは，そのまま目的関数になるとは限らない．チューニングは，ソフトウェアの未来の利用のために行うものである．このため，未来においてどのような条件でソフトウェアが使われるかについて何らかの仮定をおき，チューニングコストと実行コストのバランスが取れるように両者を含むような目的関数を設定するべきである．そうしないと，無限にコストをかけて最大性能を出すようなチューニングをするべし，という無意味な解が最適解となってしまう．

2.2.2 D2: チューニングパラメタ

自動チューニングは，ソフトウェアの処理内容を変えず，性能を変えるようなパラメタ t を調整することにより，コストに関する目的関数を最小にすることである．D2 においては，パラメタ t と，その値が取り得る範囲 T とを定義する．

チューニングパラメタが複数 $t_1, t_2, \ldots, t_n$ のようにある場合，それらを並べたベクトル $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$ をチューニングパラメタとみなし，取りうる範囲は $T = T_1 \times T_2 \times \cdots \times T_n$ のように積空間とすれば，単一のチューニングパラメタの場合に帰着できる．もしチューニングパラメタの取りうる範囲に依存関係がある場合，例えば $t_1 < t_2$ のような場合には，$\boldsymbol{t}$ の取りうる範囲 T の中にこの制約条件を込める．

一般に制約条件には，ハードな制約条件と，ソフトな制約条件とがある．自動チューニングによってハードな制約条件を満たすことはできないと思われる．それゆえ，ハードな制約条件は，チューニングパラメタの取りうる範囲 T の制約条件の中で実現する必要がある．

著者の前論文[5] では，関数的適応とチューニングパラメタ制御関数について論じた．例えば密行列演算のブロックサイズをチューニングパラメタ t とすると，行列サイズ x によって最適なブロックサイズは異なるであろう．そのような可変的な適応を**関数的適応**と呼ぶ．そしてこのとき行列サイズという情報 x を**特徴量**と呼ぶ．特徴量 x のそれぞれの値に対して，チューニングパラメタ t をどのように決めるかは，**チューニングパラメタ制御関数** $t = \tilde{t}(x)$ により表現することができる．自動チューニングの目的は，チューニングパラメタ制御関数をいかにうまく構成するかという問題に帰着される．

ここで，チューニングパラメタ制御関数として取り得る関数全体の集合 $\mathcal{T}$ を考える．すると，自動チューニングの問題は，$\mathcal{T}$ の中から，優れたチューニングパラメタ制御関数 $\tilde{t}(x)$ をいかに選択するか，という問題に帰着される．チューニングパラメタ制御関数の集合 $\mathcal{T}$ の各要素を特定できるようなインデクス（可算とは限らない）i を設けよう．インデクス i の集合を $\mathcal{I}$ とする．すなわち

$$\mathcal{T} = \{\tilde{t}_i\}_{i \in \mathcal{I}}$$

である．すると，自動チューニングの問題は，インデクスの集合 $\mathcal{I}$ の中から最適な i を選択するという問題に帰着される．ここで i の記号を t と改め，$\mathcal{I}$ の記号を T と改めれば，単一のチューニングパラメタを持つ自動チューニング問題になることが分かる．すなわち，**チューニングパラメタ制御関数の選択は，チューニングパラメタの選択に帰着することができる**．

ここで，チューニングパラメタ制御関数を扱う場合のコスト関数について考察する．特徴量 x，チューニングパラメタ t のときのコストを $c(x,t)$ と書くことにする．このとき，チューニングパラメタ制御関数 $\tilde{t}_i(x) \in \mathcal{T}$ を選択したときの「コスト」は $c(x, \tilde{t}_i(x))$ であり，特徴量 x に依存する．例えば，行列サイズ x が小さい場合に高い性能を引き出すチューニングパラメタ制御関数 $\tilde{t}_0(x)$ と，行列サイズ x が大きい場合に高い性能を引き出すチューニングパラメタ制御関数 $\tilde{t}_1(x)$ があるとすると，大きい行列を扱う人にとっては $\tilde{t}_1(x)$ が優れており，小さい行列を扱う人にとっては $\tilde{t}_0(x)$ が優れている．すなわち，利用条件によって優劣が変わるということである．無論，もし上記のような特性が既知であれば，

$$\tilde{t}_2(x) = \begin{cases} \tilde{t}_0(x) & x < x_0 \\ \tilde{t}_1(x) & x \geq x_0 \end{cases}$$

のような新たなチューニングパラメタ制御関数を定義することは有用であろう．ここで x_0 はチューニングパラメタである．

2.2.3 D3: 特徴量

特徴量とは，ソフトウェアが実行される環境的な条件のうち，観測可能であり，自動チューニングに利用するものを言う．前著[5] で論じたように，条件には**観測される条件**と**観測されない条件**があり，観測される条件のうち特徴量とならなかったものおよび観測されない条件は**非特徴量**と呼ばれる．非特徴量が性能に与える影響は**擾乱**となる．

オフライン自動チューニングでは実験のために条件を与えてやる必要がある．これを**制御される条件**と呼び，その他の条件を**制御されない条件**と呼ぶ．実験時の制御される条件は，実施時の条件とは完全には一致しない．このため，実験時の性能情報を実施時の性能改善にいかに利用するかは，自明でない問題である．

2.2.4 D4: システムとタスクに関するその他の条件

著者はこれまでの発表の中で，様々な自動チューニング問題の設定について考察してきた．**オンライン自動チューニング**[6] は，実施するたびにチューニングパラメタを設定でき，コストが測定できると仮定している．システムあるいはソフトウェアがこのような機能を有していなければ，オンライン自動チューニングの数理手法は利用できない．このような前提条件としてどれだけの種類があるかはわかっていないが，これらを D4 としてまとめて記述することにする．

オフライン自動チューニング[5] を実現するには，それを実現するプラットフォームや，データなどの仮の実行条件を与えることができる機構が備わっている必要がある．敵対的コストモデル[7] や並列試行[8] では，所定の時間をオーバーした実行を**中断**し，別のチューニングパラメタを用いて計算をやり直すことが必要である．このほか，問題サイズの縮小[5] や履歴情報[9] なども，システムあるいはソフトウェアとしてそのような機能を有していなければ利用できないので，その可否を明らかにしておく必要がある．

2.3 4 つ の A

4 つの A は，自動チューニングという問題を最適化問題として数理的に定式化する部分である．

2.3.1 A1: コスト推定関数に関する仮定

コスト推定関数は，D1 で定義されるコストを事前に推定するものである．自動チューニングにおいては，これを実測情報に基づいて構成するのが普通である．特徴量が x, 非特徴量が y, チューニングパラメタが t のときのコストを $c(x,y,t)$ と書く．非特徴量 y はコストの推定に用いられないので，擾乱として扱われる．特徴量 x とチューニングパラメタ t だけからは，コストを正確に予測することは一般にはできない．このため，コスト推定関数が何を推定するか，というのは議論の余地がある．

自動チューニングの手法によっては，コストの平均値

$$\bar{c}(x,t) = \int c(x,y,t)p(y)dy$$

を計算すれば足りるという場合も考えられる．しかし，著者の提案する手法では，特徴量が x, チューニングパラメタが t のときに，コストが c となる確率

$$\tilde{p}(c,x,t) \approx \int p(y)dy\Big|_{c(x,y,t)=c}$$

を推定する．単純さと安定性のため，著者はパラメトリックな手法で推定を行っている．以下，コスト推定関数を定義するのに必要な要件を検討する．

擾乱の仮定　擾乱を含むような問題の場合は，測定データに含まれる擾乱成分を考慮しなければならない．擾乱があると仮定するか否か，また，擾乱がどのような分布に従うと仮定するかにより，データ解析の手法が異なってくる．特に，擾乱が有限か，あるいは無限のコストを取り得るかが重要な分岐点になる．著者はこれまで擾乱が有限の場合を考察している．その場合は，平均成分と誤差成分

$$c(x,y,t) = \bar{c}(x,t) + e(x,y,t)$$

に分けることができ，誤差成分 $e(x,y,t)$ が擾乱を表すと考えられる．著者はこれまで擾乱について正規分布

$$e(x,y,t) \sim N(0,\sigma^2(x,t))$$

と仮定してきた．実際には，これほど簡単なモデルであっても，後述のように，擾乱の正確な推定は難しい．

コスト推定モデル　コストの推定をどのような形で表現するかという問題である．著者が提案しているベイズ統計に基づく推定手法では，先験的知識を含むような事前情報と，擾乱の混じった実測情報とから，事後分布 $\tilde{p}(c,x,t)$ を推定する．著者の手法では，各 (x,t) につき独立に事後分布を求めており，パラメタ (x,t) とコストとの依存性は事前情報にのみ含まれている．正規分布を用いているので，事前情報として必要なのは期待値 $\mu(x,t)$ と分散 $\tau^2(x,t)$ である．

期待値推定モデル　特徴量 x とチューニングパラメタ t に対して，コストの期待値 $\bar{c}(x,t)$ を推定するモデルである．著者が用いているのは**線形モデル**と呼ばれるもので，既知の関数の集合 $\{f_i(x,t)\}$ と未知の係数 β_i を用いて

$$\bar{c}(x,t) = \sum_i \beta_i f_i(x,t) + e(x,t)$$

のように推定するものである．ここで $e(x,t)$ は誤差成分であり，既知の関数 $w(x,t)$ を用いて

$$e(x,t) \sim N(0,\tau^2 w(x,t))$$

に従うと仮定する．ここで τ^2 は未知のパラメタである．

このほか，例えば疎行列の格納形式を実行時に選択するような問題では，チューニングパラメタの値を変更することに伴い格納形式の変換が必要となり，オーバーヘッドが生じる．一旦作った格納形式を保存しておくことも考えられるが，最近使っていない格納形式のデータはキャッシュから外れているためにやはりオーバーヘッドが生じる．このような効果はコスト推定モデルの中に含めることができる．

また，計算を複数の部分に分解して，それぞれの部分に対してコスト推定を行い，それらを結合して全体のコストを推定することも考えられる．それぞれの部分に対してコストの測定ができる場合には，このような分解により推定精度が改善する場合がある．

2.3.2 A2: 条件予測に関する仮定

自動チューニングは未来の実行のために行うものである（今チューニングしても，過去の実行の性能を変えることはできない）．このため，自動チューニングを数理的に定式化するためには，未来の実行における条件を予測するということが必須となる．

具体的には，特徴量 x と非特徴量 y とが未来においてどのように与えられるかが問題である．その実行順序が問題になることもあるし，回数だけが問題のこともある．

特徴量 x と非特徴量 y の分布だけであれば，過去からの履歴から推定することはある程度妥当であろう．対象のソフトウェアがあと何回実行されるかという情報は極めて重要な意味を持つが，正確な推定が難しい．

2.3.3 A3: 目的関数と制約条件

自動チューニングを数理的に定式化するためには，目的関数を適切に定めることが必須である．とりわけ，実験のコストと実施のコストのバランスがどのように達成されるかが，目

的関数の設定に依存する．この点では，実施と独立した試行を持つオフライン自動チューニングは特に注意が必要で，目的関数の設定によって様々な異なる解が得られることになる．試行コストと実施コストの重みつき和として目的関数を設定すると，**逐次サンプリング**[10] として知られる問題となる．実施と独立した試行を持たないオンライン自動チューニングではこのような問題はなく，ほぼ自明である．このとき，**multi-armed bandit problem**[11] と呼ばれる問題として定式化される．

このほか，各種の制約条件が課される可能性がある．一般的には制約条件をペナルティ関数として目的関数に付加することが考えられる．ある種の制約条件は，実験のやり方などを制約する場合がある．例えば，オンラインにチューニングパラメタを変更してはいけないとか，オフラインの試行ができないとか，実施時のコストはある一定値以下でないといけないとか，中断・再開を行ってはいけないとかいった制約条件が考えられる．並列処理における自動チューニング[7] に関する条件も，ここに含まれる．

2.3.4 A4: その他の数理的仮定

上記のほかに，サイズや種類の異なるタスクにおける性能との相関などが，数理的仮定として追加されうる．

2.4 4 つ の C

4 つの C は，4 つの A により数理的に定式化された自動チューニングの最適化問題に，具体的に解法を与える．4 つの C は 4 つの A に対応する．

2.4.1 C1: コスト推定関数の構築

A1 で定義したコスト推定関数のモデルに従って，コスト推定関数を構築する．ベイズ統計と線形モデルを用いた著者の手法では，以下のアルゴリズムが必要となる．

- 擾乱成分 $\sigma^2(x,t)$ の推定
- 線形モデルの係数 β_i の推定
- 線形モデルの誤差係数 τ^2 の推定[12]
- コストの推定（事後分布の計算）
- 事前事後分布の計算

擾乱成分 $\sigma^2(x,t)$ の推定は，実際のところ非常に難しい．期待値の推定よりも分散の推定の方が収束が遅いうえに，性能の悪い選択肢については，期待値も十分に収束しないような回数しか実験されないからである．このため，詳細なモデルを構築するのは現実的ではなく，非常にシンプルなモデルで近似するのがよいと考えられる．

これらの推定は相互に関連している．ベイズ的な定式化においては，階層ベイズ的な扱いと経験ベイズ的な扱いとが考えられる[13]．著者は，線形モデルに関する部分では，アルゴリズムが単純な経験ベイズを用いている．擾乱成分に関する部分では，階層ベイズ的な「分散未知の正規分布」および経験ベイズ的な「分散既知の正規分布」を用いている．分散未知の正規分布モデルにおいても，事前分布は経験ベイズ的な手法で構築した[14]．

事前事後分布は，次の測定値が c と仮定したときに，その測定の後の事後分布をあらかじめ計算するもので，下記の実験計画において必要になるものである．このように，4 つの C は相互に関連していて，一部が変わると他の部分に影響が及ぶ可能性がある．

2.4.2 C2: 条件予測

条件の予測には 2 つの方法がある．第 1 の方法は**計画ベース**の手法で，どのような条件で実行するかユーザが決めておく．第 2 の方法は**履歴ベース**の手法で，過去にどのような条件で実行されたかを記録しておき，将来も同様な条件で実行されるものと推定する．

2.4.3 C3: 実験計画

実験計画は，各回の実行においてチューニングパラメタ t にどの値を代入するかを決定する．オフライン自動チューニングでは，試行を継続するかどうかも決定する．中断が可能な場合には，どのような条件で中断をするかを決定する．

著者は，オンライン自動チューニング[6]，オフライン自動チューニング[5]，並列自動チューニング[15], ランダマイズド自動チューニング[16] などに適用できる，**ワンステップ近似**を提案してきた．

ワンステップ近似を簡単に説明すると，A1 でモデル化され，C1 により推定されたコストに基づき，各選択肢について，次のステップでそれを選択した場合のコストを推定する．さらに，実際にその選択肢を用いてコストが測定された後に，どのようにコスト推定が変化するかをあらかじめ計算する．これを事前事後分布という．C2 で予測された条件に基づき，事前事後分布において最適と推定される選択肢が常に選択されるという仮定のもとで，2 ステップ先以降のすべての実行のコストを推定する．ここでは，事後分布が逐次修正されるということを無視して，事前事後分布を固定したものとして最適な選択肢を決定する．このようにして，次のステップで各選択肢が選択された場合の，次ステップ以降の総コストを推定する．そして，この総コストが最小になる選択肢を，次のステップの選択肢として選択する．

ワンステップ近似の実験計画で重要な計算は，事前事後分布における最適選択肢の決定である．次ステップで観測されるコストに依存して，選択肢が変わる．選択肢が変わるところで期待値の挙動が変わるため，積分区間を分ける必要がある．

ワンステップ近似はベイズ的に推定されたコストを必要とする．ここで経験ベイズ的手法

を用いているため，最初のコスト推定モデルが構築されるまでは，ワンステップ近似が適用できない．そのため，最初の数回の実験においては，コスト推定モデルを構築するための実験計画を別途定めている．

そのほか，著者が提案してきた実験計画として，無限希釈[17] や並列自動チューニングのための SEO, PEO, MEO[18] などがある．

2.4.4 C4: その他の数理的手法

このほか，サイズや種類の異なるタスクの性能情報の利用などが数理的手法として含まれうる．

2.5 新 4DAC

以上のように，この節では，4) で提案した 4DAC を再検討し，4DAC を修正再定義した．項目を列挙すると，以下のようになる．

D1 コスト関数
D2 チューニングパラメタ
D3 特徴量
D4 その他の条件
A1 コスト推定関数に関する仮定
A2 条件予測に関する仮定
A3 目的関数と制約条件
A4 その他の仮定
C1 コスト推定関数の構築
C2 条件予測
C3 実験計画
C4 その他の数理的手法

4 つの D はチューニング対象に内在する項目，4 つの A はチューニング対象の数理的なモデル化，そして 4 つの C は数理的な計算手法である．4 つの A はチューニング対象と，その 4 つの D を分析することを通して定義される．すなわち，4 つの A はチューニング対象の数理的な性質を反映するものである．これに対して 4 つの C は 4 つの A にそれぞれ対応した計算を実現するもので，チューニング対象とは分離して定義される．このように，4 つの A はチューニング対象の具体性と数理的手法を分離し，またその間をつなぐインタフェースとなっている．

3. 自動チューニング数理コアライブラリ ATMathCoreLib

3.1 自動チューニング数理ライブラリの構築・利用方式の提案

前節の考察に基づき，本論文では自動チューニング数理ライブラリを以下のように構築および利用することを提案する．

- 自動チューニング数理ライブラリは，4 つの C の実装を提供する．
- ひとつひとつのライブラリは，4 つの A としてある範囲のものを想定して構築され，どのような 4 つの A を想定したか明記される．
- 自動チューニング数理ライブラリの利用者は，チューニング対象の 4 つの D を決定し，4 つの A に相当する分析を行う．その結果，自動チューニング数理ライブラリが仮定する 4 つの A と一致するか，適当に近似されると判断した場合，それに対応するライブラリを使用する．

これにより，自動チューニング数理ライブラリは，チューニング対象の具体性から独立して開発することが可能になり，また，その利用者は自動チューニング数理ライブラリを適切に利用することが可能になる．

3.2 ATMathCoreLib の構成

本研究では，上記の方針に基づいて自動チューニングのための数理コアライブラリ ATMathCoreLib を構築した．

ATMathCoreLib は現在プロトタイプとして Scilab 関数で実装されている．今年度の研究で，もともとオンライン自動チューニングのために提案したワンステップ近似が，オフライン自動チューニングやランダマイズド自動チューニングにも使えることが分かってきた．このため，使用法を限定するようなデータ構造を作らず，一般的なデータ構造の組み合わせで計算部分だけをライブラリ化している．

以下，ATMathCoreLib が仮定する 4 つの D と 4 つの A, および 4 つの C の実装の概要，使用法を説明する．

3.2.1 仮定する 4 つの D

4 つの D に関連する仮定は以下の通りである．(1) チューニングパラメタが取り得る値は離散的で有限個である．(2) 特徴量はない．(3) オンライン自動チューニングが可能である．

3.2.2 仮定する 4 つの A

A1 コスト推定関数に関する仮定は，以下の通りである．(1) 擾乱は分散が既知の正規分布に従う．分布は静的である．すなわち

$c(y,t) = \bar{c}(t) + e(y,t), \qquad e(y,t) \sim N(0, \sigma^2(t)).$

(2) コストの期待値は既知の線形モデルに従う．すなわち

$$\bar{c}(t) = \sum_i \beta_i f_i(t) + e(t), \qquad e(t) \sim N(0, \tau^2 w(t)).$$

A2 条件予測に関する仮定は特定していない．しかし，オンライン自動チューニングのための実験計画には，ターゲットソフトウェアが今後実行される回数が必要であり，開発者が何らかの推定方法を提供すると仮定する．

A3 目的関数と制約条件は，オンライン自動チューニングの定式化に従う．すなわち，試行はなく，実施時にチューニングパラメタを変更し，その都度コストを観測することができる．実施回数は既知であり，目的関数は合計実施コストである．

A4 その他の仮定はない．

3.2.3 提供される 4 つの C

ATMathCoreLib は以下の関数により 4 つの C の実装を提供する．

C1 コスト推定関数の構築は，以下の関数からなる．

[n, mu, var] = update_muvar(n, mu, var, y) 新しいデータ y を与え，サンプル数 n, 平均値 mu, 分散 var を更新する．

[tau2, mu0, t02] = linmodel_est(n, mu, X, s2, w, tau2) 線形モデルを推定する．引数の n はサンプル数ベクトル，mu は平均値ベクトル，X は $X_{tj} = f_j(t)$ で計画行列と呼ばれる．引数 s2 は既知の擾乱の分散 $\sigma^2(t)$ を並べたベクトル，w は線形モデルの誤差モデル $w(t)$ を並べたベクトル，tau2 は線形モデルの誤差係数 τ^2 の初期推定値である．出力の tau2 は線形モデルの誤差係数 τ^2 の新しい推定値，mu0 は期待値の推定値 μ_0 のベクトル，t02 は mu0 の誤差分散の推定値 τ_0^2 のベクトルである．すなわち

$$\bar{c}(t) \sim N(\mu_0(t), \tau_0^2(t))$$

と推定されている．

[mun, tn2] = nkv_posterior(mu0, t02, n, mu, s2) 分散既知の正規分布（期待値 mu0, 分散 t02）に対し，データ（サンプル数 n, 標本平均 mu, 標本分散 s2）を与えたときの事後分布（期待値 mun，分散 tn2）を返す．

[mup, tp2] = nkv_predict(mu0, t02, n, mu, s2) 分散既知の正規分布（期待値 mu0, 分散 t02）に対し，データ（サンプル数 n, 標本平均 mu, 標本分散 s2）を与えたときの，次の観測値の予測分布（期待値 mup, 分散 tp2）を返す．

[mupp, tpp2] = nkv_prepost(mu0, t02, n, mu, s2, y) 分散既知の正規分布（期待値 mu0, 分散 t02）に対し，データ（サンプル数 n, 標本平均 mu, 標本分散 s2）および次の観測値 y を与えたときの事前事後分布（期待値 mupp, 分散 tpp2）を返す．

なお，擾乱の分散 $\sigma^2(t)$ は既知と仮定しており，推定ルーチンは提供していない．今後の課題である．

C2 条件予測は実装されていない．ターゲットソフトウェアが今後実行される実行回数が情報として必要であるが，これは例えば (1) 既知である（計画ベース），(2) 以前の実行回数と同数だけ実行される（履歴ベース），のように推定される．

C3 実験計画は，以下の関数からなる．

choice = nkv_online(n, mun, tn2, s2, rem) オンライン自動チューニングにおける，次の選択肢を返す．引数 n は選択肢数，mun と tn2 はベイズモデル（事後分布）の期待値と分散のベクトル，s2 は擾乱の分散のベクトル，rem は残り実行回数である．この中で，以下の関数を用いる．

w = nkv_online_w(n, mun, tn2, s2, mumin, rem) オンライン自動チューニングのある選択肢について，今後の実行コストの予測値を返す[6]．引数 n はその選択肢の既実行回数，mun と tn2 はベイズモデルの期待値と分散，s2 は擾乱の分散，mumin はこの選択肢以外でコストが最小の選択肢のコストの期待値，rem は残り実行回数．計算には以下の関数を用いる．

v = nkv_muppmin(n, mun, tn2, s2, mumin) オンライン自動チューニングのある選択肢について，2 回先以降の 1 回当たりの実行コストの予測値を返す．すなわち，次ステップで当該選択肢を測定後に，その選択肢のコストが最小になればその選択肢を，そうでなければ他の最適選択肢を選ぶとして，コストの期待値を計算する．引数は nkv_online_w と同様である．計算には以下の関数を用いる．

eta = nkv_preposteq(n, mun, tn2, s2, mumin) オンライン自動チューングのある選択肢が次ステップで選択されるとして，観測されるコストがいくらであれば，他の最適選択肢 mumin よりもコストの期待値が小さくなるかを返す．すなわち，次ステップで観測されるコストが eta 以下であれば，当該選択肢のコストの期待値が mumin 以下となるような eta を返す．

なお，C2 で提供される線形モデルが構築されるまでの初期実験計画については未提供である．利用者の知見に基づいてヒューリスティックな初期実験計画が構築されることが望ましいが，一般的な手法としては，既存の最適実験計画[19] などが考えられる．

C4 その他の数理的手法は実装されていない．

3.3 ATMathCoreLib の利用方法

3.3.1 オンライン自動チューニング[6)]

前節の関数を用いたオンライン自動チューニングの方法を以下に示す.

(1) 事前情報: 実行前に，コスト推定モデルの計画行列 X と誤差モデルベクトル w, 誤差係数の初期値 tau2, それに擾乱分散ベクトル s2 を設定する.

(2) 初期設定: 実行回数ベクトル n, 標本平均ベクトル mu, 標本分散ベクトル var をゼロベクトルで初期化する.

(3) 初期実験計画: モデルが構成され，誤差の推定ができるためには，rank X + 1 回の初期実験が必要である．この回数までは，別途与えられた初期実験計画に基づいてチューニングパラメタを設定する．選択肢 i でコスト v が測定されたら，

[n(i), mu(i), var(i)] = update_muvar(n(i), mu(i), var(i), v)

により選択肢 i の情報を更新する.

(4) オンライン実験計画: まず，

[tau2, mu0, t02] = linmodel_est(n, mu, X, s2, w, tau2)

により線形モデルの推定を行う．次に

[mun, tn2] = nkv_posterior(mu0, t02, n, mu, s2)

により，線形モデルと測定データを結合して事後分布を得る．次に

i = nkv_online(n, mun, tn2, s2, rem)

により選択肢 i を得る．ここで，rem は適当な方法で推定した残り実行回数（今回を含む）である．選ばれた選択肢 i で対象ソフトウェアを実行しコスト v を観測する．そして

[n(i), mu(i), var(i)] = update_muvar(n(i), mu(i), var(i), v)

により選択肢 i の情報を更新する.

ATMathCoreLib に含まれるファイル sampleonline.sci には，本ライブラリを用いたシミュレーション（乱数を用いた仮想的チューニング実験）が実装されている．**図 1** は，このファイルで定義されている test 関数を実行した結果の例である．横軸は実施回数，縦軸はコストである．プラスマークは実際に観測されたコストを表す．ダイヤマークは選ばれた選択肢のコストの真の期待値を示す．Scilab コンソールには，リグレットとロスが表示される.

3.3.2 数理モデル構築のヒント

ATMathCoreLib のコストモデルは線形モデルである．ハイパフォーマンスコンピューティングでは性能モデルの構築ということが古くから行われており，多くの知見があり，その中には線形モデルも多い．本節では，ATMathCoreLib に適した数理モデルについて，著者の経験に基づく考察を述べる.

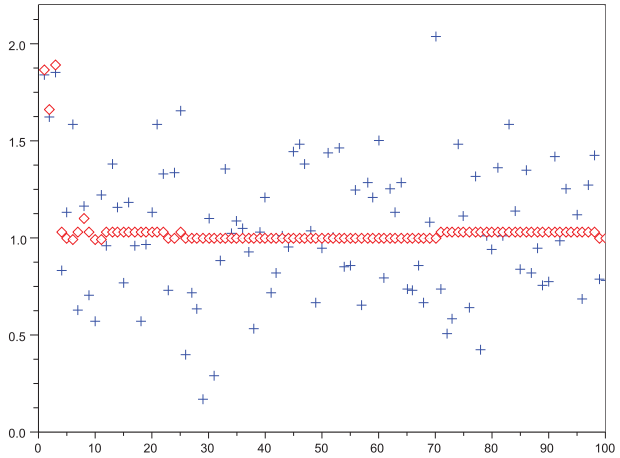

図 1 オンライン自動チューニングのサンプルプログラムの実行結果
Fig. 1 Execution of sample program of online automatic tuning

モデルはシンプルな方がよい．パラメタが多いモデルは，データに過適応しがちである．非線形なモデルはフィッティングが不安定になることがある．多少フィッティングの精度を犠牲にしても，シンプルな線形モデルの方が安定して動作する.

パラメタの範囲によって挙動が異なるような場合には，それぞれの範囲についてモデルを構築してもよい．ATMathCoreLib では，線形モデルの推定の関数と実験計画の関数は分離しているから，線形モデルの構築に手を加えても，実験計画の関数をそのまま利用することができる.

現在の ATMathCoreLib は，特徴量がないと仮定している．特徴量がある場合は以下のように対応することが考えられる．特徴量が取り得る値の集合が，離散的で比較的少数の場合には，その値ごとにモデルを構築することができる．特徴量の取り得る値の集合が，多数もしくは連続の場合には，値の集合をいくつかのクラスに分け，そのクラスごとに（特徴量

を固定した）モデルを構築することが考えられる．

コストモデルはある方がよい．コストモデルがあれば，どのあたりの選択肢がよさそうかという推測が立つ．モデルが実際のコストを適切に反映しているかどうかは自動チューニングにおいて問題になるが，ATMathCoreLib の線形モデルの構築関数はモデルの精度を定量的に評価しているので，モデルが現実に近ければ自動的に効率的な探索が行われ，モデルが現実と離れていればより広範囲の探索が自動的に行われる．この意味において，ATMathCoreLib の利用者は，モデルの精度について重大な注意を払わなくてもよい．両端ではよくなく，真ん中あたりでよいという程度の知識であっても，2 次関数で近似するということも考えられる．

どうしてもコストモデルが構築できない，すなわち，パラメタとコストとの関係がまったく分からない場合であっても，ATMathCoreLib は利用できる．この場合には計画行列 X および誤差モデルとして，すべてが 1 の列ベクトルを入力すればよい．これにより，選択肢のコストがランダムに分布していると仮定して実験計画が行われる．この実験計画は，コストモデルがないにも関わらず，有用な実験計画である．具体的には，平均的なコストに比べて優れた選択肢が見つかれば探索を減らし，あまりよい選択肢が見つからない場合には探索を広げるような実験計画が得られる．

3.3.3 無限希釈[20)]

ATMathCoreLib は，基本的なオンライン自動チューニング以外にも利用することができる．第 1 の応用例として，無限希釈を用いたオンライン自動チューニングの方法を以下に示す．

無限希釈は，希釈関数 $f(n)$ を用いて実験計画を希釈する．希釈関数は $n \to \infty$ で $f(n) \to \infty$ および $f(n)/n \to 0$ を満たすものとする．例えば c を適当な定数として $f(n) = c \log n$ などが考えられる．

無限希釈を用いたオンライン自動チューニングは，(1) の事前情報において，希釈関数 $f(n)$ を定める．(2) と (3) はオンライン自動チューニングと同じである．(4) を以下のように修正する．

第 n 回の実施において，これまでの実験計画計算数 m が $m < f(n)$ を満たすときには**情報探索的**実施とする．このときは，オンライン自動チューニングの実験計画に基づいて，前節の (4) と同様に次ステップの選択肢を決定する．それ以外の場合には**情報活用的**実施とする．このときは，ベイズ統計に基づく事後分布までは同様に計算し，もっともコストの期待値が小さい選択肢を次ステップに用いる．

3.3.4 ランダムサブセット法[16)]

第 2 の応用例として，ランダムサブセット法によるオンライン自動チューニングの方法を以下に示す．

ランダムサブセット法では，(1) の事前情報において，適当な正整数 N_{new} を決めておく．(2) と (3) はオンライン自動チューニングと同じである．(4) を以下のように修正する．

第 $n-1$ 回までの実施において選択された選択肢の集合を C_{n-1} とする． C_{n-1} に含まれない選択肢からランダムに N_{new} 個を選んだ集合を D_n とする．ただし，もし C_{n-1} 以外の選択肢の数が N_{new} 以下の場合は，残りの選択肢すべてを D_n とする．第 n 回の実施においては，$C_{n-1} \cup D_n$ を候補集合とみなして，オンライン自動チューニングの (4) と同様の計算により，選択肢を決定する．

4. おわりに

本論文では，自動チューニングの数理ライブラリについて論じた．まず，著者が以前に提案していた 4DAC[4)] を改良した新 4DAC を提案した．4DAC は，自動チューニングの数理的側面を明確化する道具である．4 つの D が対象のソフトウェアの具体性を記述し，4 つの A はそれを数理的にモデル化し，4 つの C が数理的手法を定義する．数理ライブラリはある 4 つの A を仮定して，それを解決する 4 つの C を提供する．ライブラリの利用者は自分の問題の 4 つの D を明確化し，ソフトウェアの特性を分析して 4 つの A を設定し，それに適したライブラリを選択して用いる．

また，上記 4DAC モデルに基づく自動チューニング数理コアライブラリ ATMathCoreLib を論じた．現在の実装は線形モデルを用いたオンライン自動チューニングを実装している．本論文では仮定する 4 つの A を記述し，4 つの C の実装の概要を説明した．また，基本的な使い方および応用例を示した．本研究により，自動チューニングの数理的諸要素を明確化し，汎用的に使える自動チューニングのための数理ライブラリの構成方式とインタフェースが明らかにできた．

今後の課題は多岐にわたる．本稿執筆時点での ATMathCoreLib に実装されていない基本的機能として，擾乱の分散の推定方式と初期実験計画方式がある．また，現在の ATMathCoreLib は Scilab スクリプトで実装されており，より高速な実装が必要である． QR 分解の downdating のような高速アルゴリズムの実装も望まれる．また，チューニングメーター[20)]やオフライン自動チューニング，並列自動チューニングのための実験計画などの既存の成果のライブラリ化も必要である．このほか，モデル選択，敵対的コストモデル，問題サイズの

縮小，相関情報の利用など，開発が必要な数理的問題も多数ある．

本研究は，自動チューニングの数理的側面に焦点を絞って進めてきた．自動チューニングの実現のためには，数理だけではなく，可変性を実装するプログラミング言語システム，数理ライブラリと対象ソフトウェアをつないで実際に可変性を駆動する基盤ソフトウェアシステム，そして個々の問題で有効性を発揮する個別のチューニング技術の開発が必要である．今後は自動チューニングの数理手法の研究にとどまらず，これら 4 分野の知見の総合的な整備に努力したい．

謝辞 本研究の一部は JST CREST「ULP-HPC: 次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」，科学研究費新学術領域研究「コンピューティクスによる物質デザイン」計画研究「超高速・超低消費電力物質科学シミュレーション方式の研究開発」の支援を受けています．

参 考 文 献

1) Whaley, R.C. and Dongarra, J.J.: Automatically Tuned Linear Algebra Software, *Proceedings of SC98* (1998).
2) Frigo, M. and Johnson, S.G.: FFTW: an adaptive software architecture for the FFT, *Proceedings of ICASSP '98*, Vol.3, pp.1381–1384 (1998).
3) Katagiri, T., Kise, K., Honda, H., and Yuba, T.: ABCLibScript: A directive to support specification of an auto-tuning facility for numerical software, *Parallel Computing*, Vol.32, No.1, pp.92–112 (2006).
4) Suda, R.: Automatic Tuning Math Core Library, *Workshop on Advanced Auto-tuning on Numerical Software (AANS2010)* (2010).
5) 須田礼仁：オフライン自動チューニングの数理手法，情報処理学会研究報告 HPC-125-3, pp.1–9 (2010).
6) Suda, R.: A Bayesian Method for Online Code Selection: Toward Efficient and Robust Methods of Automatic Tuning, *Proc. iWAPT 2007*, pp.23–31 (2007).
7) 須田礼仁：並列計算機におけるソフトウェア自動チューニングのための数理モデル，日本応用数理学会 2009 年度年会講演予稿集，pp.13–14 (2009).
8) 須田礼仁：並列試行による並列処理のためのオンライン自動チューニング，第 15 回計算工学講演会論文集，Vol.15, No.1, pp.89–92 (2010).
9) 須田礼仁：ソフトウェア自動チューニングの数理，情報処理，Vol.50, No.6, pp.487–493 (2009).
10) Cailori, R. and Dalang, R.C.: *Sequential Stochastic Optimization*, John Wiley and Sons, Inc. (1996).
11) Berry, D.A. and Fristedt, B.: *Bandit Problem*, Chapman and Hall (1985).
12) 須田礼仁：頑健で効率的なオンライン自動チューニングのための統計モデル，情報処理学会研究報告 HPC-116，pp.109–114 (2008).
13) Carlin, B.P. and Louis, T.A.: *Bayes and Empirical Bayes Methods for Data Analysis*, Chapman and Hall (2000).
14) 須田礼仁：オンライン自動チューニングのための Bayes 統計に基づく逐次実験計画法，情報処理学会 2008 年ハイパフォーマンスコンピューティングと計算科学シンポジウム（HPCS2008），pp.73–80 (2008).
15) 須田礼仁：並列ソフトウェアのオンライン自動チューニングのための Bayes 的手法，情報処理学会 研究報告 (2010).
16) 須田礼仁：自動チューニングのための実験計画におけるランダマイズの効用，日本応用数理学会 2010 年度年会講演予稿集，pp.295–296 (2010).
17) Suda, R.: *A Bayesian Method of Online Automatic Tuning*, Software Automatic Tuning: Concepts and State-of-the-Art Results, chapter16, Springer (2010).
18) Suda, R.: Methods of Parallel Experimental Design of Online Automatic Tuning and their Application to Parallel Sparse Matrix Data Structure, *Proc. iWAPT 2010 (in Proc. VECPAR'10)* (2010).
19) Atkinson, A. and Donev, A.: *Optimum experimental designs*, Oxford University Press (1992).
20) 須田礼仁：自動チューニングのための数理基盤技術，応用数理， Vol.20, No.3, pp. 5–14 (2010).