特集 科学技術計算におけるソフトウェア自動チューニング





# 3 ソフトウェア自動チューニングの数理

須田　礼仁　東京大学情報理工学系研究科コンピュータ科学専攻／ JST, CREST

## ソフトウェア自動チューニングとは

ソフトウェア自動チューニングとは，ソフトウェアがその実行環境に自分自身を適応させることを目指す技術的パラダイムである．ソフトウェア自動チューニングの一般論や実現方法などについては，本特集の他の原稿において詳述されているところであるが，ソフトウェア自動チューニングの数理的側面について紹介する本稿においては，ソフトウェア自動チューニングとは何なのかを，一般的な枠組みでとらえておく必要がある．本稿で仮定するソフトウェア自動チューニングの模式図を図 -1 に示す．以下ではこの図に沿って，一般的なソフトウェア自動チューニングの概念を整理する．なお，本稿は数理を論じながら，読みやすさを優先して厳密さを欠く表現を用いる場合があることをあらかじめお断りしておく．

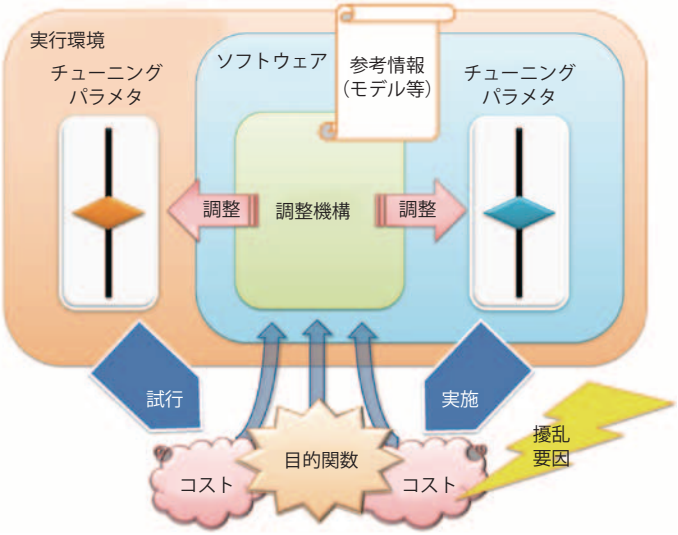


図 -1　ソフトウェア自動チューニングの抽象モデル

### ■ ソフトウェア自動チューニングの抽象モデル

ソフトウェアの自己適応能力の実現方法にはさまざまなものが考えられる．たとえば，アルゴリズムが本来的にデータに適応する機能を有していて，それを実装することで自然に適応性を有する場合がある．他方，調整可能なパラメタを持つソフトウェアがあり，そのパラメタを調整するようなソフトウェアを付加的に組み合わせることにより，全体として自己適応的なソフトウェアを実現するという方法もある．さらには，同じ機能を持つ複数のライブラリが実装されていて，その中から実行環境に適したものを選択する機能を有するソフトウェアを経由してライブラリを利用することによっても，自己適応的なソフトウェアが実現できる．このようにさまざまな実現方法があるが，本質的な点をまとめると，「ソフトウェア」の中に変更可能な「チューニングパラメタ」があり，ソフトウェアが内包する「調整機構」がそのチューニングパラメタを調整するという共通点がある．

ソフトウェアの「実行環境」には，ソフトウェアが実行されるハードウェア，入力となるデータ，当該のソフトウェアとともに動作する関連ソフトウェアなどが総合的に含まれる．そして，「チューニングパラメタ」は，実行環境にも含まれる可能性がある．たとえば，OS やコンパイラ，ライブラリなどのオプションなどである．実行環境とチューニングパラメタが与えられたとき，ソフトウェアの動作はほぼ一定であるが，各種の「擾乱要因」のために完全に同一の結果になるとはかぎらない．擾乱要因の例としては，他のプロセスの影響や，キャッシュの状態，所要時間や消費電力の測定に伴う不可避の測定誤差などがある．擾乱要因を実行環境の一部と考えることも可能だが，本稿では別物として扱う．

調整機構がチューニングパラメタを調整するにあたっては，「目的関数」が必要である．従来多くの研究では所要時間を暗黙の目的関数としていたが，消費電力や課金，計算精度など，また，それらを組み合わせたものも目的関数になり得る．本稿では，所要時間や消費電力などの個々の要因を「コスト」と呼び，「目的関数」はいずれかの

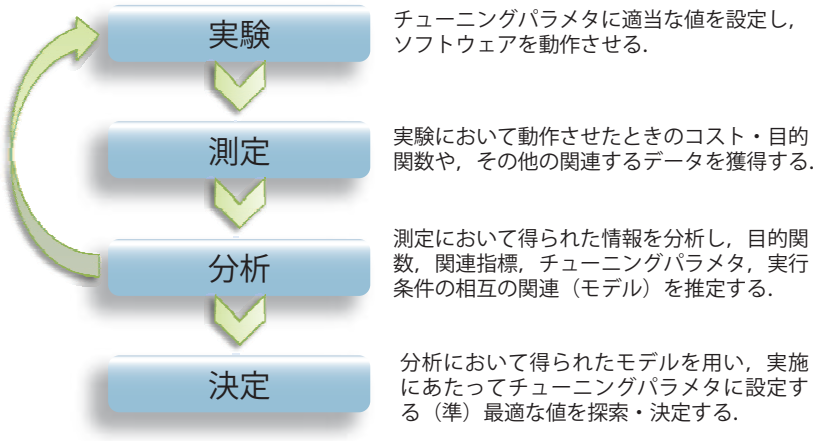


図 -2　ソフトウェア自動チューニングの 4 つの処理

コスト，あるいは複数のコストを複合したものであるとし，チューニングにおいてはこの目的関数を最小化することを目指すものとする．また，各種のパラメタやコストに関して「制約条件」が課される場合もある．一般に，制約条件には，それを破ってはいけないハードな制約条件と，できるだけ破らないほうがよいソフトな制約条件とがある．ソフトな制約条件を扱うには，目的関数と分けて制約条件を個別に扱うほかに，ペナルティとして目的関数に取り込むこともできる．しかし，ソフトウェア自動チューニングはあくまで経験に基づいてパラメタを調整するものであるため，ハードな制約条件を満たすために用いることは適当でない．ハードな制約条件は，自動チューニング機構とは別の手法で満たされるようになっていなければならない．

ソフトウェア自動チューニングにおいては，調整機構自身が何らかの実行環境（評価用のサンプルデータなど）を準備し，そこでソフトウェアを実行することにより，ソフトウェアの特性を分析するということがよく行われる．本稿ではこのようなソフトウェアの実行を「試行」と呼ぶ．これに対して，ソフトウェアの本来の目的である実用的な実行のことを「実施」と呼んで区別する．

ソフトウェア自動チューニングは，実行を通して獲得したデータ（コストの測定値など）を用いてチューニングパラメタを調整するものであるが，これに加えて，プログラマやユーザから「参考情報」が与えられているのが一般的である．参考情報には，「1 次キャッシュのアクセス遅延は何サイクルである」といった定量的なものから，「正方行列の積の計算量は行列サイズの 3 乗のオーダである」といった定性的なものまでさまざまである．これらの参考情報は，チューニングには役立つものの，一般に不完全であり（完全であれば実験の必要がない），場合によっては間違っている（実際の実行条件に対して不適切）かもしれない．たとえば行列積の計算量オーダの例について考えると，この情報は数学的には正しいが，所要時間が正確に 3 次多項式で表現できるわけではないし，行列のサイズが小さい範囲では多くの計算機において所要時間は行列サイズの 3 乗に比例しない．

参考情報の中で最も重要な要素は，先に挙げた行列サイズと行列積の計算時間との関係などのように，コスト，目的関数，チューニングパラメタ，その他の定量的・定性的指標などの間で近似的に成立すると仮定される数学的関係である．本稿ではこれを「モデル」と呼ぶ．行列積の計算時間の例の場合，モデルは「3 乗」といっているだけで，それにかかる係数は与えられていない．このように未知パラメタを含むものも含めてモデルと呼び，モデルの中の未知パラメタを推定する処理を「フィッティング」と呼ぶ．

## ソフトウェア自動チューニングにおける数理の役割

ソフトウェア自動チューニングを実現するとき，多くの場合において次の 4 つの処理が必要となる（図 -2）．

1. 実験：チューニングパラメタに適当な値を設定し，ソフトウェアを動作させる．
2. 測定：実験において動作させたときのコスト・目的関数や，その他の関連するデータを獲得する．
3. 分析：測定において得られた情報を分析し，目的関数や関連する指標の挙動と，チューニングパラメタやその他の条件との関連（モデル）を推定する．
4. 決定：分析において得られたモデルを用い，実施にあたってチューニングパラメタに設定する（準）最適な値を探索・決定する．

以下ではこれら4つの処理における数理の役割について，既存手法の概観も含めつつ考察する．

### ■ソフトウェア自動チューニングの実験の数理

実験は自動チューニングの基礎である．パラメタに値を与えて実行し，性能を実測することにより，パラメタと性能との関連についての情報を獲得する．このとき，パラメタをどう選んで実験するかによって情報獲得の効率が変わってくるので，実験におけるパラメタの選択は自動チューニングにとって非常に重要な問題である．古典的には実験計画という分野が直接関係している．実験そのもののコストを無視すれば，とり得るすべてのパラメタの組合せについて，十分な回数，ランダムな順序で実験を行えば，チューニングに必要な情報が十分かつ確実に得られる．しかし一般にこれは現実的ではない．実験のコストを抑えつつ，いかに効果的に準最適解を得るための情報を獲得するかが，実験計画の問題である．

実験計画に大きな影響を与えるのが，参考情報が含む誤差と擾乱要因の影響の2つである．しかし既存のソフトウェア自動チューニングの研究においては，これらが定量的に扱われていることはほとんどない．測定を反復して統計的処理を行ったり，モデルが示唆する最適値の付近を探索したりする研究も少なくないが，定量的で数理的な根拠がない場合が多い．機械学習[1]やNelder-MeadのSimplex法などの最適化アルゴリズムを用いる研究もあるが，これらの多くは擾乱要因を十分に考慮していない．

### ■ソフトウェア自動チューニングの測定の数理

測定とは，設定したパラメタ値で実行した際の，性能に関する情報を獲得することである．測定を実行するのはシステムの役割であるが，数理に関係するのは測定に対する擾乱要因の影響の扱いである．これに関して，数理に期待されている点は2つある．

第1は，擾乱の大きさを分析し，測定値にどれだけの誤差が混入しているかを定量的に推定することである．

第2は，擾乱の特性を分析し，測定値から擾乱要因を除去して，本来求めたい値に近づけることである．これらの処理に関係が深い古典的分野は統計的データ解析や時系列解析などである．しかし，これらの古典的な知見を十分に反映させたソフトウェア自動チューニングの研究はほとんど見当たらない．

### ■ソフトウェア自動チューニングの分析の数理

分析とは，実験で得られた性能を情報としてまとめる処理で，その中でも重要なのがモデルの構築である．通常，未知パラメタを含むモデルは参考情報として事前に与えられており，チューニングにおいて必要となるのはフィッティングである．すなわち，統計学で推定と呼ばれる処理である．

フィッティングは，自動チューニング以前から長らく行われている性能評価・性能予測における基本的問題であり，多くの知見がある．古典的なフィッティングではモデルと実験データとの乖離(モデル誤差)を最小にするように未知パラメタを選択するという最適化問題となるが，この最適化問題における目的関数を何に設定するかは意外と難しい．安易に最小二乗法などを用いると，期待したようなフィッティング結果にならないことがある．たとえば，所要時間に関して最小二乗法でフィッティングすると，所要時間が長いデータに影響されて，所要時間が短いところでのフィッティングが悪くなる．

従来の性能評価の研究のように，フィッティングの結果を人間が見て判断する機会があれば，適切なフィッティングができていないときに，フィッティングの手法を修正することができる．しかしソフトウェア自動チューニングでは人間が介在することなくフィッティングが行われ，結果が利用されることになる．このような条件下でも安定してよい結果を導くようなフィッティング手法が必要である．

### ■ソフトウェア自動チューニングの決定の数理

決定は自動チューニングの最終ステップであって，これ以降はパラメタの変更は行われない．それまでに得られた情報を総合し，チューニングパラメタに具体的な値を代入して，できるだけ最適に近い状態を実現するのがこのステップの目的である．

これは一種の最適化であるので，目的関数をいかに定めるかがきわめて重要である．それにもかかわらず，従来のソフトウェア自動チューニングの研究の多くでは，目的関数が明確にされていない．たとえばATLASのような行列計算の場合，行列のサイズによって最適なパラメタが異なるが，どのような行列サイズ分布を仮定して最適化してあるかが明らかとなっていない．目的関数とユーザが実際に利用する際のコストとが一致しているかどうか，つまり設定した目的関数が適切かという議論も行われていない．たとえば，そもそも大規模行列計算をしないユーザにはATLASのようなライブラリはむしろ邪魔であるが，ユーザが大規模行列計算をするかどうかの確認はATLASでは行われていないのである．

## Bayes 統計に基づくソフトウェア自動チューニングのための数理基盤

前章で見たように，ソフトウェア自動チューニングにおける 4 つの処理のいずれにおいても，数理は重要な役割を果たすが，既存研究では十分な取り扱いがなされていない．これに対し我々は，ソフトウェア自動チューニングに必要な数理的な基盤技術の整理と整備を目的として研究を推進してきている．本章では我々の提案と成果をごく簡単に紹介する．

我々の提案は大きく分けると次の 4 項目になる．

1. ソフトウェア自動チューニングの最適化問題としての定式化
2. Bayes 統計に基づくソフトウェア自動チューニングのための数理コアの提案
3. ソフトウェア自動チューニングのための準最適な逐次実験計画法
4. ソフトウェア自動チューニングの数理的手法に求められる性質の定式化

### ■ソフトウェア自動チューニングの最適化問題としての定式化

これまでのソフトウェア自動チューニングのほとんどの実装においては，目的関数が何であるかが明確になっていなかった．従来研究ではハードウェアやシステムソフトウェアなどのプラットフォームに対する意識が強く，ユーザがどのようにソフトウェアを利用するかという視点が欠けていた．たとえば計算化学や量子計算のシミュレーションにおいては，サイズが 3 あるいは 4 の大量の行列の固有系を求める必要があるが，このような用途には ATLAS のような大規模行列を意識した自動チューニング行列ソフトウェアは適さない．巨大行列の計算はそれ自身で時間がかかる処理であるので，それを高速化することは非常に有用であり，これをターゲットにチューニングすることには合理性がある．しかし，ユーザが大規模行列の計算をするかどうかを確認さえせずに最適化するという点は，不親切というべきである．

ソフトウェア自動チューニングの従来研究におけるもう 1 つの問題点として，試行のコストについての制御ができるようになっていないことが挙げられる．インストール時に自動チューニングを行うソフトウェアは，インストールが完了するまで半日ほどかかってしまうこともある．自動チューニングソフトウェアが 1 つだけであればよいが，多数のソフトウェアがそれぞれ自動チューニング機構を有するようになれば，ソフトウェアのインストールに何週間もかかってしまいかねない．計算機センタのスーパーコンピュータであれば，そもそも設置に長期間を有するので，インストールに多少時間がかかってもよいかもしれないが，個人の PC では長いインストール時間は問題である．

これらの考察から我々は，今後ソフトウェア自動チューニングを考える際には，次の 2 点が必要だと考える．

1）ユーザの実際に利用する実行条件での実施時コストを目的関数の主たる要素とすること．
2）チューニングの目的関数あるいは制約条件として，試行のコストを明示的に含めること．

ここで，第 1 点で述べているユーザが実際に利用する実行条件はどのようにして知ることができるであろうか．我々は次の 2 通りを考えている．

- 計画ベースの手法：ユーザから利用の予定についての情報を得るもの．特定のソフトウェアを稼働させるための計算機や，組込みシステムなどで使えると考えられる．
- 履歴ベースの手法：ユーザの履歴を記録し，そこから将来の需要を予測する．個人の PC など汎用の計算機で用いる．

計画ベースの手法では実施前にチューニングをしてしまうことができるが，履歴ベースの手法では，最初は（あまり）チューニングされていない状態で使い始め，履歴の蓄積に応じて徐々にチューニングされることになる．両者の融合として，計画内容を履歴により修正する方法も考えられる（図 -3）．

以上のような何らかの方法で将来の実施を予測し，試行のコストを含めて目的関数とすることで，ソフトウェア自動チューニングを最適化問題として定式化できる．ここで最も単純な目的関数として，試行と実施のコストの総和を考えてみよう．近年地球温暖化に関連して消費エネルギー削減の重要性の認識が高まっているが，実行に伴う総エネルギーを最小にするには，試行と実施のエネルギーの総和を目的関数にすべきであり，この枠組みで扱うのが適切である．さて，チューニングのためにはコストの情報が必要であり，情報の多寡がチューニング結果の良し悪しに直結する．このため，実施においても試行と同様に可能なかぎり情報を収集すべきである．実施においても試行と同様に情報が得られるのであれば，コストをかけて実施と別に試行を行う理由はほとんどなくなってしまう．このような考察から我々が提案したのが「オンライン自動チューニング」である．すなわち，試行はまったく行わず，実施時に情報を収集してチューニングを行うという，新しいソフトウェア自動チューニン

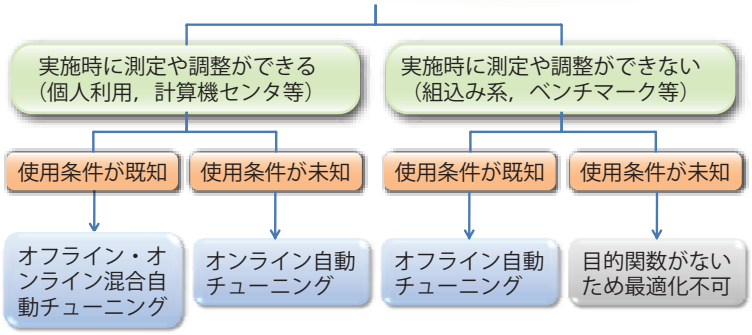


図 -3　使用条件に基づく自動チューニング方式の違い

グの手法である．これに対し，試行を用いた自動チューニングは「オフライン自動チューニング」と呼ぶ．オンライン自動チューニングの利点の 1 つは，インストール時のコストが最小限で済むということである．また，実施時の環境で情報が収集されるため，自然にユーザが多く利用する条件に対してチューニングされる．欠点としては，実施を実験として利用するために，実施時にパラメタが動的に変更され，実施時の実行性能が安定しない．このため，ベンチマーキングなどでは注意が必要である．なお，オンライン自動チューニングでは，実験結果を見て次の実験を決めることになるが，これには逐次統計 [2] と呼ばれる統計数理が必要である．

## ■ Bayes 統計によるソフトウェア自動チューニングのための数理コア

ソフトウェア自動チューニングは実験を重視するが，測定データには擾乱要因によるばらつきが含まれるのが一般的である．擾乱要因を無視してしまうと，真の最適解に到達できなくなる危険性が高い．そのため，ソフトウェア自動チューニングの数理においては，擾乱要因を適切に考慮することが必要である．

他方で，参考情報の誤差もまた同様に問題である．参考情報は一般に誤差を含むが，誤差が小さければ参考情報をもとにきわめて効率的にチューニングを行うことができる．しかし，参考情報と現状との不一致が激しい場合には，参考情報を用いたために低い性能にとどまってしまう危険性もある．このため，参考情報の精度を評価しながらチューニングを行うことが必須である．

我々はこれらの 2 種類の未知の誤差を定量的に扱うことができる数理的手法として，Bayes 統計を用いることを提案している．参考情報は Bayes 統計の「事前分布」として取り込み，事前分布と測定値を合わせて「事後分布」を得ることにより，参考情報と測定値の不確定性を定量的に扱うことが可能となる．

## ■ オンライン自動チューニングのための準最適な逐次実験計画法

Bayes 統計を用いた数理コアにおいてオンライン自動チューニングの最適化問題を定式化すると，bandit problem として知られる問題に帰着される．Bandit problem を直感的に説明すると次のようになる．いくつかの選択肢があり，各試行ではその選択に応じた報酬が与えられる．各選択肢に対する報酬は未知の確率分布に従うのだが，試行を繰り返すことで確率分布に関する情報が得られる．このような状況下で報酬の総和を最大にするという問題が bandit problem である [3]．数学的にはこの問題の最適解は既知であるが，それをそのまま計算しようとすると爆発的な計算量が必要となるため，ごく限られた場合を除き，実用的ではない．

そこで我々は，いくつかの重要な性質を保持する近似最適解を提案した [4]．詳細はここでは省略するが，我々の提案する近似解は(大雑把な表現であるが)次の性質を有している．

a. コストの期待値が大きく，不確定性が小さいパラメタ値については，選択されることはない．
b. コストの期待値が大きくても，不確定性が大きいパラメタ値については，選択される可能性がある．
c. 未来に実行される回数が多いほど，b で述べたようなコストの期待値が大きなパラメタ値が選択される可能性が高くなる．

上記の b で述べたように，コストが大きいと期待されるパラメタ値が選択されるのは，実行することにより得られる情報が，実行コストよりも価値が高いと判断されるためである．このように，我々の手法は，実験により得られる情報量と実験に伴うコストとのバランスを定量的に評価できるものとなっている．

図 -4 に提案手法を既存手法と比較した事例を示す．処理内容は行列積ルーチンのアンローリング段数を最適化するものであるが，実験と評価の詳細は論文 [4] を参照

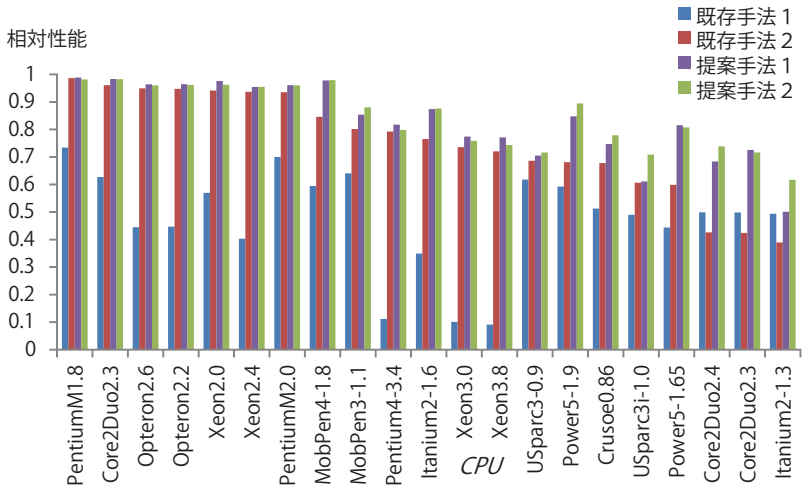


図 -4　提案した逐次実験計画の評価事例

してほしい．グラフでは横軸にさまざまな計算機（CPU とクロック（GHz）のみ示す）を取り，縦軸には以下のように定義した「相対性能」を取った．ここでの「相対性能」とは，各手法との総実行コストに対して，実験をせずに同じ回数だけコスト最小の選択肢だけを実行した場合のコストの比を取ったもので，1 に近いほど少ない実験コストでよいパラメタが得られたことを表す．比較手法を略述すると，既存手法 1 はすべての選択肢を 1 度ずつ試す，既存手法 2 はモデルが最適と示すものを使う，というもので，提案手法の 1 と 2 は Bayes 統計に用いる手法が異なるものである（正確なアルゴリズムは文献 4）を見られたい）．グラフ横軸の計算機は既存手法 2 の相対性能でソートしてあるので，左の計算機ほどモデルと現実との一致がよく，右のほうではモデルが現実をうまく表していない．グラフより，提案手法はいずれの計算機でも既存手法よりもよい結果を示しているが，これは，提案手法が，モデルが正確であればそれを利用して実験を効率的に行っているだけではなく，モデルが不正確であってもそれなりに情報を抽出して実験を効率化していることを示している．

## ■ソフトウェア自動チューニングの数理的手法に求められる性質の定式化

Bayes 統計という枠組みはきわめて一般的なもので，事前分布や擾乱要因に伴う測定のばらつきの分布として具体的に何を選ぶかについては，非常に大きな自由度がある．また，分布の種類を固定したとしても，そこに含まれる各種パラメタをどのように推定するかについても各種の方法（参考情報の一部として与える，経験 Bayes 法や階層 Bayes 法により推定するなど）が考えられる．前述の我々の逐次実験計画を用いる場合を考えても，これらの分布や手法の選択によって，どの程度効率的に最適化されるかが影響される．そして，最悪の場合には，ほとんど最適化が進まないうちに，実験を行わなくなってしまうことがあることが分かってきた．

その原因を追及したところ，統計学の基本的難問である「分散成分」にあることが判明した．ソフトウェア自動チューニングにおいては，参考情報の誤差と，擾乱要因によるばらつきという 2 種類の誤差要因がある．不都合が生じるのは参考情報の誤差に比べて擾乱要因の影響が大きい場合で，このとき標準的な統計的推定手法のいずれを用いても，正の確率で「参考情報の誤差が 0 である」という推定をしてしまう．このときアルゴリズムは参考情報が完全に正しいと推定してしまい，参考情報（モデル）が最適と推定するパラメタ値より優れたものが存在する可能性はないと判断してしまうのである．しかし，使用する分布や統計的手法を適切に選ぶことにより，与えられた参考情報や擾乱要因の影響の大小にかかわらず，決してこのような最悪の状況には陥らないようにできる．このとき，「無限に実施を繰り返す」という極限において，真の最適解を必ず見つけ出すことを保証することができる．このような保証がある数理的手法は，「漸近最適性」を持つという．

上記の状況とは逆に，参考情報の精度をあまりに低く見積もってしまうと，取り得るあらゆるパラメタ値について実験をしてみないと何の判断もできないという状況に陥ってしまう．連続値パラメタの場合には，すべての値で実験するということは原理的に不可能だから，実質的にまったく最適化がなされないことになる．このような状況に陥らないという性質を「初期実験の有限性」と呼

ぶ．初期実験の有限性も，分布や手法を適切に選ぶことにより避けることができる．

我々は，ソフトウェア自動チューニングに用いられる数理手法は，これら 2 つの性質，つまり「漸近最適性」と「初期実験の有限性」を有しているべきであると考えている．また，これらの性質を満たす具体的な手法もいくつか示している．

## 今後の展望

この章では，ソフトウェア自動チューニングにおける数理の役割と，我々の研究提案の概要を紹介してきた．ソフトウェア自動チューニングの数理的側面に焦点を当てた研究は世界的に類例がなく，独自性が高い研究であるが，それだけに取り組まなければならない課題も多い．また，ソフトウェア自動チューニングは，数理だけでとどまる話ではないので，システム，プログラミング言語，具体的な場面におけるチューニング技術の研究開発とも連携し，これから一層研究に注力し，ソフトウェア自動チューニングの数理的基盤の確立を目指したい．

**参考文献**


1) Eijkhout, V. : A Self-Adapting System for Linear Solver Selection, Proc. 1st International Workshop on Automatic Performance Tuning (iWAPT2006), pp.44-53 (2006).
2) Govindarajulu, Z. : Sequential Statistics, World Scientific (2004).
3) Berry, D. A. and Fristedt, B. : Bandit Problem, Chapman and Hall (1985).
4) 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 HPCS 2008, pp.73-80 (2008).


（平成 21 年 3 月 31 日受付）


**須田　礼仁（正会員）** reiji@is.s.u-tokyo.ac.jp

東京大学准教授．東京大学助手，名古屋大学講師・助教授を経て現職．
http://olab.is.s.u-tokyo.ac.jp/~reiji/