Vol.2012-HPC-135 No.21
2012/8/2

自動チューニングのための相関モデル：行列積における行列サイズパラメタ

須田礼仁[1,a)]

概要：本研究では，性能の相関を利用した自動チューニングに向けて，空間統計で用いられている variogram 解析と kriging 推定を利用することを提案する．例題として正方行列積のアンロールを取り上げ，ある行列サイズでの性能モデルから，別の行列サイズでの性能を推定，またその推定精度を推定する問題を考える．行列サイズの対数の差を用いてパラメトリックなモデルを構築し，外れ値処理を加え，バイアスのキャンセル，誤差モデルおよび variogram モデルを構築し，性能推定と性能推定精度の推定に用いた．これにより性能および性能推定精度が評価できた．

1. はじめに

半導体技術の進歩は，計算機アーキテクチャの高度化・複雑化という形で性能に貢献するようになっている．発熱・冷却の問題から，今後はクロック周波数の大幅な改善は期待できない．一方でスーパーコンピュータのコア数は 100 万を超え，デスクトップでも 10 以上のコアが容易に入手できるようになっている．すなわち並列処理を用いることで計算の高性能化が可能であるともいえ，また同時に，並列処理を用いなければソフトウェアの高性能化が期待できない．また，現代のプロセッサの性能は，小容量・低遅延・広帯域のキャッシュメモリによるデータ供給に依存している．すなわち主記憶は容量・帯域とも継続して改善されているが，遅延の改善は難しいことと，プロセッサの高性能化のペースが速いことから，プロセッサが要求するデータ供給能力を十分に出せない．実際にはこれは設計上のバランスの問題で，発熱量や製造コストをプロセッサとメモリシステムにどのように配分するかという選択において，プロセッサへの投資の比重がまだ多いということである．また標準化・製造コスト・システム構築の柔軟性や保守性等の観点も含まれているものと思われる．さらに容量と遅延のトレードオフから，多段のキャッシュが広く用いられている．今後はさらに主記憶と演算器間の遅延と帯域のバランスの悪化が予想されていることから，これらのキャッシュを十分に活用しなければ，ソフトウェアはプロセッサの性能を十分に引き出せないことになる．さらに GPU や Xeon Phi のような演算加速器が広く用いられるようになってきており，CPU と演算加速器という異機種複合型の計算機を使いこなす技術も必要となってきている．これはタブレットやスマートフォンなどの携帯端末でもすでに導入されており，あらゆる場面で異機種複合型計算機の利用技術が必要となってきている．さらにネットワークで結合された複数のノード計算機を活用するクラスタ型の並列計算，仮想化された多数の演算コアを活用するクラウド型の並列計算，複数のシステムにまたがって計算資源を活用するグリッド型の並列計算など，高性能計算のあり方は多様化している．

このように複雑化した計算機アーキテクチャの性能を引き出すためには，ソフトウェアの**チューニング**が必要となる．チューニングは，プログラムの論理的な挙動は変えずに，ソフトウェアの実行性能を向上させるような，プログラムに施す一種の変換である．上記のようなハードウェア的な視点でのチューニングのほかに，データやユーザの特徴にあわせたソフトウェア的な視点でのチューニングも必要である．しかし，上述のように計算機アーキテクチャが多様化し，さらに急速に進歩を続けているために，個別の条件に対してプログラムをチューニングするのが困難になっている．並列計算においては，使用するノード数および各ノード内で使用するコア数によっても最適なチューニングが異なるため，ノード数・コア数といったパラメタに対して適応するチューニングが望まれる．また多様なユーザが使用して多様なデータが与えられると考えられるライブラリやミドルウェアは，ユーザやデータの多様性にも適応することが望ましい．これらの要請から，異なる実行条

[1] 東京大学情報理工学系研究科 / Graduate School of Information Science and Technology, the University of Tokyo, 113–8656 東京都文京区本郷 7-3-1
[a)] reiji@is.s.u-tokyo.ac.jp

件に対して適応的にソフトウェアをチューニングする**自動チューニング**の実現が望まれている．

自動チューニングの基本的なアイデアは簡単である．ソフトウェアの中に**可変性**を仕込んでおく．この可変性を適切に調整することにより，異なる実行条件に対して適応できるソフトウェアを実現する．可変性を調整するつまみを**チューニングパラメタ**と呼ぶ．また，チューニングパラメタに代入される個々の値を**候補**あるいは**選択肢**と呼ぶ．可変性の実現にはいくつかの方法がある．たとえば，(1) プログラムのソースコードの中に開発者が組み込む，(2) さまざまなバージョンのプログラムを生成する生成プログラムを開発者が構築する，(3) さまざまなバージョンのプログラムを生成するようなプログラミング言語あるいはプログラミング環境の機能を用いる，(4) コンパイラなどが自動的に生成する，などである．さまざまなバージョンが生成される場合，そのすべてが正しい動作をするようにデバッグすること，あるいは正しく動作するバージョンのみを選択することは，容易でない作業である．自動チューニングのためにはプログラムは極端に複雑になりがちであり，プログラムの管理・保守の点でも課題がある．このような**プログラミング**にまつわる課題は，自動チューニングを実現するための課題の中でも大きなもののひとつである．

可変性とつまみが実装されたら，実行条件においてソフトウェアを実行し，チューニングパラメタを変えつつ性能を測定し，良好な性能を実現する選択肢を選ぶ．これを**チューニング作業**と呼ぶことにする．チューニング作業を行うタイミングはさまざまなものがありうる．たとえば，(1) ソフトウェア開発時に，仮定として与えた実行環境と仮想的なデータを与えてチューニングを行う，(2) ソフトウェアを個別のシステムにインストールする際に，実際のハードウェアを用いてサンプルデータを与えてチューニングを行う，(3) ソフトウェアを実際に使う際に，実際のハードウェアと実際のデータを用いてチューニングを行う，(4) ソフトウェアが実行されたときに情報を確保しておき，プロセッサがアイドルなときにチューニングを行う，などである．性能測定のためだけに行われ，計算結果が実際には利用されない実行を**試行**と呼び，実際の計算を**実施**と呼ぶ．実施の前に，試行だけによって行う自動チューニングを**オフライン自動チューニング**，試行を全く用いず，実施時に性能を測ってチューニングする場合を**オンライン自動チューニング**と呼んでいる．当然，試行と実施の両方を活用して行うことも考えられる．

自動チューニングを行うためには，性能測定が欠かせない．これまで自動チューニングの主な目標は所要時間の短縮であったが，精度，消費電力，並列化効率，課金など，多様な要因が目的関数となりうる．これら個別の要因を**コスト**と呼び，トータルとして最適化したいものを**目的関数**と呼ぶ．正しい目的関数は，実施時のコストだけではなく，チューニングのためのコストも含まれなければならない（そうでないと無限に試行をしてしまう）．性能の測定，性能情報の保存と検索などは自動チューニングのためのコストの一部であり，これらは少ないほど良い．性能の測定と性能情報の保存・抽出を低オーバーヘッドで実現すること，またこれらに対する効率的で統一的なインタフェースを与えることは，自動チューニングのための**システムソフトウェア**の課題である．また，複数のソフトウェアモジュールが相互に関連するような自動チューニングのためには，モジュール間を調整するための共通の仕組みも必要である．これもシステムソフトウェアの課題と考えている．一方，測定された性能情報に対するデータ分析，最高の性能に近づけるように候補を選択する最適化，効率的に性能情報を獲得するためにどのようなチューニングパラメタで性能測定するかを決定する実験計画は，自動チューニングにおける**数理**の課題である．

このように，自動チューニングを実現するためには，(1) プログラミング，(2) システム，(3) 数理という 3 つの領域において課題がある．さらに忘れてならないのは，**実際に高い性能を達成するような選択肢がソフトウェアに組み込まれていなければならない**ことである．すなわち，それぞれの実行条件で良好な性能を達成する (4) **個別のチューニング技術**も開発することは大前提である．これら 4 つの領域が自動チューニング研究における重要な課題領域であり，我々はこれらの領域の課題に対して取り組みを進めている．

著者はこれまで主に数理領域において研究を進めてきた．我々は，自動チューニングを，未知情報を含む最適化問題として定式化している．これまで HPC の研究者はチューニングパラメタと性能との関係を近似的に表現する**性能モデル**を構築することで手動のチューニングを行ってきた．性能モデルを構築することで，チューニングパラメタと性能との関係が明らかになり，最適な選択肢を効率よく決定することができていた．この知見を活用するべく，我々は性能モデルに基づくチューニングを目指しているが，性能モデルはいつでも正しいわけではなく，誤差を含んでいる．この誤差は，(1) 測定が不十分であるなどの原因で性能モデル内に含まれるパラメタのフィッティングが最適でないことによるもの，(2) 性能モデルで表現されない性能の違いが実際には存在すること，(3) 同一のパラメタであっても常に同じ性能が得られるとは限らないことなどに起因する．これらの誤差を含む性能モデルを活用しながら，実際に最適な選択肢ができるだけ選ばれるような最適化手法が必要である．このために我々はベイズ推定に基づくデータ解析と実験計画の手法を提案してきた．

本稿では，前回に続いて，相関のある性能情報をどのようにして我々の枠組みにおいて活用するかについて考察する．

ここでいう相関とは，異なる条件や異なるソフトウェアにおける性能における相関である．自動チューニングにおいて相関が期待されるのは，実行条件に相違点と共通点がある場合である．例えば，行列ライブラリをサイズの異なる行列で計算させれば，所要時間に関連性があるであろう．あるいは，アルゴリズム中にパラメタが存在し，値が近ければ性能が近いことが期待される場合もある．あるいは，繰り返し実行してゆくに従い実行条件が変化する場合，近接する実行の間では性能に相関があるであろう．あるいは，同じデータの一部を用いた計算や，同種のデータを用いた計算とは性能の相関があると考えられる．さらに，若干ハードウェアパラメタが異なる類似のハードウェアアーキテクチャにおいては性能に相関があると期待される．これらの相関は，むしろ手動チューニングやベンチマーキングにおける基本的な仮定である．このような相関があるからこそ，一定の方針でチューニングすることに意味があり，厳選されたベンチマークソフトウェアの性能をもってシステムの性能を近似的に表現できると考えられる．このように性能の相関はチューニングにおいて極めて重要なファクターであるにも関わらず，相関のある性能を定量性のある方式でどのように活用すればよいか，論じられてこなかった．本研究はこのような手法の開発を目指している．

次節では・・・

2. ベイズ推定を用いた自動チューニング数理手法

まず，今回の解析の前提となっている手法について説明する．前節で述べたように，我々の自動チューニング数理手法は，ベイズ推定にもとづいている．

2.1 ベイズ推定と自動チューニング数理手法

ベイズ推定は，ベイズの定理

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \qquad (1)$$

にもとづいている．ベイズの定理のこの式は，結合確率 $P(X,Y)$ と条件付き確率 $P(X|Y)$，さらに周辺確率 $P(X)$, $P(Y)$ との関係

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

から導かれる．ベイズ推定の利用の仕方は多様であるが，我々の枠組みでは，X が真の性能を示すパラメタであり，Y が観測される性能である．想定として，真のパラメタ X は未知であるが，我々は（性能モデルなどにより）どのぐらいの確からしさでどのぐらいになりそうかという推定を持っている[*1]．そこで，どのような値がどの程度期待されるかという程度を，確率分布 $P(X)$ という形で定量化する．我々は主に正規分布を扱っていて，その場合，一番ありそうな値を「期待値」とし，どのぐらい誤差がありそうかを「分散」として，確率分布 $P(X)$ を決定する．このように，我々の枠組みでは確率 $P(X)$ を「部分的な知識」を表現するために用いており，ランダムに事象が発生する古典的な「確率」とは異なるものである．しかしながら，真のパラメタが実際に確率 $P(X)$ で分布するように，たとえばハードウェアの母集団からランダムにサンプリングする，あるいはデータの母集団からランダムにサンプリングする，ということをすれば，真のパラメタが $P(X)$ に従って確率的に発生するような状況も**シミュレーション**することができる[*2]．もし実際の実行条件がそのような確率的な実行条件発生過程における一つの実現値であると思うならば，統計的な手法を用いて，「性能の平均値」が最適になるようなチューニングが可能である．性能の平均値が最適というとき，個々の実行条件に対して得られるチューニング結果がそれぞれ最適なのではないということに注意が必要である．個々の実行条件に対しては，うまく最適化される場合もあり，うまく最適化されない場合もある．なお，事前分布 $P(X)$ の期待値が X の真値に近いということは必ずしも必要ない．それはそもそも難しい．むしろ大切なのは，以下に述べるベイズ推定により，事前分布と観測値と組み合わせた時に，残存する不確定性を的確に表現する事後分布が得られることである．それが実現できれば，事前分布が多少ずれていても，観測値により事後分布は矯正されてゆく．ただし，そうであっても，真の X の事前確率が極点に低い場合には，それを矯正するための観測値が多数必要であり，実験計画の効率性は下がる．

また，パラメタが X であったとしたときに，観測される性能 Y がどのような分布で発生するかを，確率分布 $P(Y|X)$ であらわす． $P(Y|X)$ はパラメトリックなモデルであって，そのパラメタは X である．X 以外のパラメタを含む場合は，別にパラメトリックなモデルを作っておき，実験的にパラメタを決定する．我々のモデルでは，計算を反復しつつ観測することで，実際に確率的な挙動を示す性能が得られることを想定している．ただしほぼ決定的に動作するように作られた計算機の性能挙動であるから，性能がばらつくのは何らかの具体的な要因があってのはずである．しかしそのような要因はあまりにも多数あり，計算機の状態をすべて入手して，それに基づき性能の違いを解析・表現することは現実的ではない．計算機の状態の複雑性を忘れて，観測してみると性能が確率的にばらつくとみなしているだけであるとも考えることができる．ちなみに，$P(Y|X)$ が 1 点で確率 1，他の点で確率 0 となる，デルタ関数のような分布とすれば，性能が決定的に決まり，計算を反復したときに毎回同一の性能を示すような状況に対

[*1] この推定も実験的に定める

[*2] 乱数等を用いて仮想的な性能測定データを発生させる

応する．そのような場合も我々の手法は変更なしに適用することができるが，確率分布としては極限になっているので実装の際に用いる式は異なるものが必要である．複数の観測 $Y_1, Y_2, \ldots, Y_n$ がある場合は，$Y = (Y_1, Y_2, \ldots, Y_n)$ のように Y を観測値のベクトルと考え，$P(Y|X)$ はこのベクトル Y が発生する確率であるとする．すなわち多変量分布である．ただし各観測値が確率的に独立

$$P(Y_1, Y_2|X) = P(Y_1|X)P(Y_2|X)$$

であれば，各観測に対する分布 $P(Y_i|X)$ があればよい．

ベイズ推定で得られる $P(X|Y)$ は，Y という観測値が得られた後で，パラメタ X に関して残っている不確定性を表している．上述のように，これはベイズの定理 (1) により，$P(X)$ と $P(Y|X)$ とから求まる．観測値 Y が与えられれば $P(Y)$ は定数であるので，

$$P(X|Y) \propto P(Y|X)P(X)$$

とも書かれる．ここで，任意の X, Y に対して的確な $P(X|Y)$ が得られることが重要である．正規分布を用いる場合，正規分布は期待値と分散だけで決まることから，(1) 期待値が，事前知識と観測値を総合することにより推定されるパラメタ X の値を的確に反映していること，(2) 分散が，事前知識と観測値を総合することにより推定されるパラメタ X に残されている不確定性の大きさを的確に反映していること，の 2 点が実現されていればよい．漸近的には，期待値がパラメタ値の真値に漸近し，分散が 0 に漸近することが望ましいと言える．

我々は，以上のベイズ推定をもとにして，逐次実験計画の準最適解である**ワンステップ近似**を提案した．この手法は，オンライン自動チューニング，オフライン自動チューニング，並列自動チューニング，変化点のある自動チューニング，選択肢変更コストを伴う自動チューニング，モンテカルロシミュレーションの最適化等において，主にシミュレーション評価により，効率的なチューニングができるということが示されてきた．これらの結果からわかったことは，おおよそ以下のようなことである．(1) まず，ワンステップ近似は多様な場面において有効性を発揮するということが確認された．観測値のばらつきモデル $P(Y|X)$ が真の分布に一致し，パラメタの真値 X を想定されている事前分布 $P(X)$ に従ってランダムに生成してやると，チューニングコストの期待値が（もっと単純な実験計画に比べて）小さく抑えられる．(2) 状況にあわせた的確な目的関数を設定すると，ワンステップ近似はそれを最小化しようとし，結果的に効率的なチューニングが実現できる．逆に言うと，ワンステップ近似は目的関数を最小化しようとするので，目的関数が不適切に設定されるとチューニングの効率が悪化する危険性がある．(3) ワンステップ近似と正規分布は相性がよいらしい．著者は正規分布以外のいくつかの分布を試してみたが，うまくゆかなかった．ワンステップ近似は（本稿では詳述しないが）1 ステップ先しか読んでいないので，複数ステップ先になってはじめて生じる事象に依存するような実験計画の最適化はできない．正規分布にはそのような事象がなく，1 ステップの先読みによって，複数ステップ先を考慮した実験計画を近似するものができるようである．ただしこの点についての理論的な解析は済んでいない．それに比べると，パラメタの事前分布 $P(X)$ や観測値の分布 $P(Y|X)$ に多少のずれがあっても，平均値と分散をあわせておけば，そこそこよい実験計画が達成されるという印象を得ている．

ワンステップ近似による効率的な実験計画のためには，3 つのものを与えてやる必要がある．1 つ目は観測値とそのばらつきの分布モデル $P(Y|X)$，2 つめはそのパラメタの事前分布 $P(X)$，3 つめは目的関数である．事前分布と観測値分布からベイズ推定が導かれ，ベイズ推定と目的関数から実験計画が導かれる．ベイズ推定とワンステップ近似は式が確定していて，いわば自動的に導かれる．そして，事前分布と観測値分布に従うサンプルを発生させてシミュレーションをすると，目的関数を小さくすることができた．これらのシミュレーションは，統計手法としてのワンステップ近似の有効性を評価するものである．今のところ，ワンステップ近似の有効性を理論的に証明することはできていない．

これらのシミュレーション結果を認めて，ワンステップ近似の有効性を認めると仮定すると，問題は事前分布をどのように構築するか，観測値分布をどのように推定するか，ということになる．観測値のばらつきは，実際に確率的なばらつきが発生することを想定しているので，標準的な統計手法が使えるものと考えられる．これに対してソフトウェア性能パラメタの推定は確率の問題ではなく，性能モデリングの範疇である．性能モデリングについてはこれまでに無数の研究があるが，我々の手法を用いる際には，誤差を定量的に推定し，それを実験計画に使うというところが異なっている．また，ワンステップ近似を使うということにすると，モデリング誤差の分布を正確に求めるということが必ずしも最適ではなく，ワンステップ近似でうまく実験計画ができなければ「よくないモデル」である．この点で正規分布はこれまで大きな問題なく適用できており，とりあえず使うのに適したモデルとなっている．

性能モデルは仮定であって，証明することはできない．いくら多数の実験によって有効性を示したとしても，新たな計算機アーキテクチャや新たなアプリケーションが登場したときに，その性能モデルが必ずうまく働くという保証はない．このため，性能モデルの評価は実験的にならざるを得ない．これに対して，ワンステップ近似のような統計的なアルゴリズムは，ある程度一般的な仮定のもとで一定の数学的性質を持っているということを証明することがで

きる．一定の仮定をおいて，そのうえでソフトウェアが正しく動作することを示すのを verification と呼ぼう．これに対し，実際のデータを与えた時にそのソフトウェアが適切な動作をすることを示すのを validation という．通常 verification は validation の前提となっている．この関係を**図 1** に示す．ワンステップ近似の評価は verification であり，性能モデルの評価は validation である．著者が提案する 4DAC モデルに従えば，ワンステップ近似は 4C (数学的手法) のひとつであり，性能モデルは 4A (数学的仮定) のひとつである．本稿で論ずるのは後者である．

2.2 相関情報の自動チューニングでの利用

相関がある性能情報を利用して自動チューニングを行うとき，その相関を適切にモデリングすることが必要である．相関はチューニングに役立つ情報であるが，以下に述べるように，その相関ゆえに情報量が薄まるのである．

たとえば，相関は，性能のばらつき $P(Y|X)$ においても考慮しなければならない．もし 2 回の測定値 Y_1 と Y_2 に相関がある場合，これら 2 つを合わせても 1 回の測定の 2 倍の性能情報が得られるわけではない．

極端な話として，2 回ずつ同じ性能値が出る特殊な条件を仮定する．測定の 1 回目と 2 回目ではいつも同じ値になり，3 回目は 2 回目と独立だが，4 回目は 3 回目と同じ性能，等々である．このとき，2 回の測定あたり 1 回分の性能情報しか得られないことが明らかである．もしこれを 2 回分の性能情報と思って処理すれば，測定後に残っている性能の不確定性を不当に低く評価してしまうことになる．

このように，相関が存在すると，複数の性能情報が運ぶ合計の情報量が低下することになる．測定ごとの性能に相関がある場合には，それを考慮した手法に改めなければならない．本稿ではこの問題はこれ以上論じないが，今後は，現実の問題でどのような相関が観測されるのかを分析し，相関があると判断される場合には，それに適切に対応した手法を開発してゆかなければならない．

このことは，事前情報 $P(X)$ を構築するために性能情報を利用する際にも同様である．前回の報告 [4] で，変化点のあるオンライン自動チューニングについて報告し，相関のある性能情報を活用することで自動チューニングの効率化が達成できることを示した．このときは変化点が 1 カ所しかなかったので，相関性能情報が 1 つだけ（変化点前の性能が変化点後の性能に相関する）であり，複数の相関性能情報の間の相関について論ずる必要がなかった．しかし変化点が複数になると，複数の定常区間における性能の相関を考慮に入れて手法を開発する必要がある．こちらも今後の課題である．

本稿では，相関情報として特徴量がある場合の性能モデリングについて考える．例題として，行列積のアンローリングパラメタを取り上げる．これは著者が約 5 年前に取り上げたトピックであるが，そのときは性能モデルを行列サイズごとに構築していた．以下に述べるように，行列サイズによって性能の特性が異なり，特性の変化を定量的にモデル化することが難しかったためである．これに対し，今回は性能の相関を計算し，行列サイズの異なる計算での性能値を参照して性能の推定を試みる．

3. Kriging による相関性能の扱い

3.1 空間統計の初歩事項

特徴量を持つ統計解析・推定として，時系列解析と空間統計解析がある．このうち時系列解析は時間の非可逆性を中心に構成されていて，自動チューニングにおいては，前述のように観測ごとの測定値に相関がある場合や，実行条件が時間に従って変化してゆくような場合に適用できる可能性があると思われる．

一方，空間統計解析は方向に関する制限はなく，自動チューニングにおける一般的な特徴量の扱いに近い．この分野では Noel A. C. Cressie の著書 Statistics for Spatial Data[1] がバイブル的な教科書である．実例に関しては現実のデータを扱う際のいろいろな注意点まで，理論に関しては理論の詳細とともに理論の弱点や理論の仮定の適否などまで，詳細かつ具体的に述べられている．かなり古いが，様々な疑問に答えてくれる良書である．空間統計では 2 次元のデータが扱われることが多いが，同書の理論は一般の次元での議論を展開しているので，自動チューニングでも役に立つ．

以下では Cressie の上記著書の記述にもとづき，variogram と kriging によるモデリングと推定について説明する．

3.2 Stationary process と variogram

空間を d 次元空間 $\mathbb{R}^d$ とし，座標を $s \in \mathbb{R}^d$ であらわす．分析の対象とする確率変数を $Z(s)$ とし，分析の対象となる座標の集合を $D \subseteq \mathbb{R}^d$ であらわす．すなわち $Z(s) : s \in D$ を考える．

確率変数 Z が

$$E(Z(s+h) - Z(s)) = 0$$
$$var(Z(s+h) - Z(s)) = 2\gamma(h)$$

を満たすとき，**intrinsic stationary process** という．上記に出てくる関数 2γ を **variogram** という（ときどき定数 2 を落とした γ も使われるが，これは semivariogram という）．第 1 式から，**定数平均の仮定** (constant-mean assumption)

$$E(Z(s)) = \mu$$

が導かれる（μ は未知の実数）．Intrinsic stationary process は平均値が空間的に変動する場合に拡張できるが，今回

は平均値の空間的な変動（**トレンド**という）は別途消去（**detrend** という）しておき，上記の定数平均モデルを用いる．第 2 式はベクトル h だけ位置が異なる点での値の違いが $2\gamma(h)$ という h だけの関数であらわされるということを述べている．さらに，

$$2\gamma(h) = 2\gamma(\|h\|)$$

のように variogram が 2 点の距離だけに依存するときに**等方的** (isotropic) という．少し拡張して，ある行列 A とある関数 γ^O に対して

$$2\gamma(h) = 2\gamma^O(\|Ah\|)$$

となるときに geometric anisotropy という．

このほかに **second-order stationary process** という仮説がある．これも定数平均で

$$\begin{aligned} E(Z(s)) &= \mu \\ cov(Z(s+h), Z(s)) &= C(h) \end{aligned}$$

という仮定である．関数 $C(h)$ は **covariogram** と呼ばれる．ここでも

$$C(h) = C(\|h\|)$$

を満たすと等方的という．Second-order stationarity は intrinsic stationarity よりも弱い仮定で, intrinsic stationarity を仮定した方がより広い問題をカバーできる．

$C(0) > 0$ とすると，

$$\rho(h) = C(h)/C(0)$$

が定義でき，**correlogram** と呼ばれる．これは相関係数に相当する．明らかに $\rho(-h) = \rho(h)$ および $\rho(0) = 1$ を満たす．

Variogram と covariogram は直接的に関係がある．

$$\begin{aligned} var(Z(s+h) - Z(s)) &= var(Z(s+h)) + var(Z(s)) \\ &\quad -2cov(Z(s+h), Z(s)) \end{aligned}$$

であるが，右辺は平均値 $E(Z(s)) = \mu$ を知っていなければ計算できないのに対し，左辺は μ が要らないという利点がある．いずれにせよ，これから，second-order stationary process であれば

$$2\gamma(h) = 2(C(0) - C(h))$$

となることがわかる．

Variogram $2\gamma(h)$ は明らかに $\gamma(-h) = \gamma(h)$ および $\gamma(0) = 0$ を満たす．2 点の距離 h が 0 以外から 0 に近づく（$h \to 0$）ときに $\gamma(h) \to c_0 > 0$ となる場合，c_0 を nugget effect と呼ぶ．このとき γ は原点で不連続である（著者が前回述べた「間接的相関」に相当する）．たとえば，Z が白色雑音のとき，$h = 0$ 以外で $2\gamma(h) = const$ となる．任意の関数が有効な variogram になるわけではなく，**条件付き負定値性** (conditional negative definiteness)

$$\sum_{i=1}^{m}\sum_{j=1}^{m} a_i a_j 2\gamma(s_i - s_j) \le 0, \quad \text{for any } \sum a_i = 0$$

を満たさなければならない．同様に，covariogram は正定値でなければならない．

もし $\|h\| \to \infty$ のときに $C(h) \to 0$ になるなら（空間統計ではよく仮定される），$2\gamma(h) \to 2C(0)$ ($\|h\| \to \infty$) である．この $C(0)$ は **sill** と呼ばれている．Nugget effect がある場合，$C(0) - c_0$ を **partial sill** と呼ぶ．

3.3 Ordinary kriging による推定

Kriging という言葉は，修士論文で南アフリカの金鉱の分析を行った Daniel Gerhardus Krigge の名前にちなみ，空間統計の先駆者 Georges Matheron が命名したという．ここでは Cressie に従い，文頭以外は小文字で綴ることにする（to krige という動詞もあるという）．

以下では測定していない点 $s_0 \in D$ を考え，$Z(s_0)$ をその他の測定値 $Z(s_1), Z(s_2), \ldots, Z(s_m)$ から推定する問題を考える．

モデルの仮定として，

$$Z(s) = \mu + \delta(s)$$

とおく．平均値 μ は未知定数である．推定値は

$$p(Z(s_0)) = \sum_i \lambda_i Z(s_i)$$

の形とする．ここで係数 λ_i は

$$\sum_i \lambda_i = 1$$

を満たすとする．これは「補間」あるいは「補外」を意味しており，一様にバイアスがない推定値 $E(p(Z(s))) = E(Z(s))$ を得るために必要である．また variogram の存在を仮定する．

Lagrange 未定係数を m として

$$E\big(p(Z(s_0)) - \sum_i \lambda_i Z(s_i)\big)^2 - 2m(\sum_i \lambda_i - 1)$$

を最小化する．ここで仮定である $\sum_i \lambda_i = 1$ を使うと

$$\begin{aligned} &(Z(s_0) - \sum_i \lambda_i Z(s_i))^2 \\ &= -\sum_i \sum_j \lambda_i \lambda_j (Z(s_i) - Z(s_j))^2/2 \\ &+ \sum_i \lambda_i (Z(s_0) - Z(s_i))^2 \end{aligned}$$

が恒等的に成り立つ．これより目的関数は

$$-\sum_i\sum_j \lambda_i\lambda_j\gamma(s_i-s_j)$$
$$+2\sum_i \lambda_i\gamma(s_0-s_i) - 2m(\sum_i \lambda_i - 1)$$

となる．これを微分することで，

$$\sum_j \lambda_j\gamma(s_i-s_j) + m = \gamma(s_0-s_i), \quad \forall i$$
$$\sum_i \lambda_i = 1$$

が得られる．これは λ_i と m に関する連立一次方程式であり，これを解くことにより係数 λ_i が決定できる．このとき推定誤差である **kriging variance** は

$$\sigma_k^2(s_0) = 2\sum_i \lambda_i\gamma(s_0-s_i) - \sum_i\sum_j \lambda_i\lambda_j\gamma(s_i-s_j)$$
$$= \sum_i \lambda_i\gamma(s_0-s_i) + m$$

となる．これにより，未観測の s_0 における値 $Z(s_0)$ の推定値および推定値がどのぐらい正確かという推定値が得られる．

4. 行列積における行列サイズに関する相関の分析

4.1 計算内容と性能モデル

本稿で取り上げる計算は正方形の行列積で，チューニングパラメタにはアンローリング段数を取る．この例題は著者が以前取り上げたもの [3] で，以下の説明はその際の説明と重複するが便宜のため再掲する．

行列積は

```
for (i=0; i< n; i++)
  for (j=0; j< n; j++)
    for (k=0; k< n; k++)
      c[i][j] += a[i][k] * b[k][j];
```

のような形をしたもので，外側 `i` および `j` のループをアンローリングする．アンロールは片桐らによる ABCLibScript を用いて自動的に生成した．外側ループのアンロール段数を u_i, 中間ループのアンロール段数を u_j としたとき，

$$(u_i,u_j) \in [1,128]\times[1,1] \cup [1,64]\times[1,2]$$
$$\cup [1,32]\times[1,4] \cup [1,16]\times[1,8]$$
$$\cup [1,8]\times[1,16] \cup [1,4]\times[1,32]$$
$$\cup [1,2]\times[1,64] \cup [1,1]\times[1,128]$$

という範囲でアンロールを行った．これにより，アンロールされていないコードも含めて 576 通りの異なるサブルーチンが得られた．

次に上記のコードに対する性能モデルを説明する．ループをアンロールすると，アンロール段数で割り切れる「商」のループと，割り切れない「剰余」のループが生じる．反復回数を N, アンロール段数を u とすると，商ループは N/u 回 (但し整数除算の商を表すものとする，以下同様)，剰余ループは $N \bmod u$ 回まわる．これを用いれば，コードの各部分でロード・ストア・演算などの命令が何回行われるか計算することができる．さらにループ制御の命令などを反復回数の定数倍で見込む． ABCLibScript がアンロールしたコードをこのように分析すると，以下の 15 項が得られる (演算回数：演算内容の形で示す)．なお，外側の商ループを「外商」などのように略記する．

- 1: サブルーチン呼び出し
- N/u_i: 外商ループの制御
- $(N/u_i)(N/u_j)$: 外商・中商ループの制御
- $(N/u_i)(N/u_j)u_iu_j$: 外商・中商の A のロード・ストア
- $(N/u_i)(N/u_j)N$: 外商・中商・最内ループの制御
- $(N/u_i)(N/u_j)Nu_i$: 外商・中商の B のロード
- $(N/u_i)(N/u_j)Nu_j$: 外商・中商の C のロード
- $(N/u_i)(N/u_j)Nu_iu_j$: 外商・中商の積和演算
- $(N/u_i)(N \bmod u_j)$: 外商・中剰余ループの制御
- $(N/u_i)(N \bmod u_j)u_i$: 外商・中剰余の A のロード・ストア
- $(N/u_i)(N \bmod u_j)N$: 外商・中剰余・最内ループの制御, C のロード
- $(N/u_i)(N \bmod u_j)Nu_i$: 外商・中剰余の B のロード，積和演算
- $(N \bmod u_i)$: 外剰余ループの制御
- $(N \bmod u_i)N$: 外剰余・中ループの制御，A のロード・ストア
- $(N \bmod u_i)N^2$: 外剰余・中・最内ループの制御，B と C のロード，積和演算

このモデルを使えば，あとは演算などの所要時間がわかれば所要時間が推定できることになる．以下では最小二乗法（所要時間が短いものほど当てはめがよくなるように所要時間の逆数を重みとした）で推定している．但し，推定しているのは命令ごとの時間ではなく，上記 15 項の係数である．

このモデルに，複数の行列サイズをまたがった性能データを入れてみたが，明らかに負になる係数などが生じた．上記のモデルの構築法からすると，各項の係数は対応する計算や処理にかかる時間に相当し，それが負になることは不自然である．このような結果になってしまったのは，モデルが実性能にフィットしない誤差の部分が影響しているため，あるいは速いものほど当てはめがよくなるように重みをかけているためと推測される．さらに，特定の行列サイズで特異な性能を示す場合があり，複数の行列サイズに対してまとめてモデル化することではよい推定が得られなかった．

そこで，当時は行列サイズ N ごとに係数を決めること

とした．行列サイズ N を定数とすると，上記の 15 項は線形独立にはならず，階数は 7 となる．ランク落ちが発生しているので，SVD を用いた最小二乗最小ノルム解を求めた．最小二乗最小ノルム解を求めているため，各項の係数が対応する計算の処理にかかる時間に対応するという意味はもたない．しかし各項を計算して所要時間の推定値を出すと，これはそこそこうまく働き（計算環境による），論文 [3] で紹介したような自動チューニングの効率化が達成された．今回はこれをベースとして，行列サイズ N が異なる性能情報に関する相関の分析と，それを用いた性能の推定をする．

4.2 何を推定するか

研究は最初から行き詰った．行列サイズの異なる性能の情報を用いて，何を推定すればよいのかが問題である．前節で説明した kriging における $Z(s)$ にあたるものが何かという問題である．

最初に検討したのは，性能モデルがもっている 15 個の係数であった．しかしここで問題が発生する．ランク落ちをしているために，線形従属な項に関する係数は最小ノルム条件に従って値が分配されてしまう．この分配は行列サイズ N によって異なるので，異なる行列サイズで得られた各項の係数は完全に対応する意味を持たない．このため，異なる行列サイズの性能モデルの係数にどのような関連があると期待できるのかが説明できなかった．

次に検討したのは，個々のサイズ n において，個々のチューニングパラメタ i に対する所要時間 t_{ni} を推定することである．ここで問題となるのが，t_{ni} は我々の実験では最大 $512 \times 576 = 294912$ もあり，これらについてひとつひとつ推定モデルを構築することの妥当性である．しかし最終的には t_{ni} が推定したいので，目的に対して直接的なアプローチであるともいえる．

別の問題として，性能モデルの係数を推定することの適切性に疑問が生じるのと全く同じ理由で，ある行列サイズで得られた性能モデルの係数を，別の行列サイズの性能モデルに代入することは適切ではない．しかし，とりあえず代入してみたところ，行列サイズが近いところでは，ある程度の精度の推定値を得ることができた．ただし行列サイズが遠ざかると，明らかに精度が低下した．

上記の問題の原因のひとつは，行列サイズごとにモデルをフィッティングするときにランクが落ちているため，係数に自由度が残っている点にある．そこで，3 つから 4 つの隣接する行列サイズの性能情報を集めて，ランクが落ちないようにして係数を推定させてみた．ところが，これでも負の係数は多数発生し，推定誤差もほとんど改善されなかった．これは，そもそものモデルがそれぞれの N に対して性能モデルを構築するのに適切なものになっていないことが疑われる．例えば，N が大きな問題では計算量の主要項が性能のほとんどを決定し，それ以外はばらつきやモデル外要因に埋もれてしまうと思われる．このような状況では，所要時間への貢献の大きくない項の係数は雑音で決まってしまう．

しかし，ほかによい案がなかったので，行列サイズ n，チューニングパラメタ i による所要時間 t_{ni} を推定することとした．以下，行列サイズ m で構築して得られた性能モデルのパラメタを，行列サイズ n チューニングパラメタ i の性能モデルに代入して得られる推定所要時間を t_{mni} とする．なお，同一サイズで構築した性能モデルでの推定値 t_{nni} は t_{ni} と必ずしも一致しない．

これにより，推定するものが非常に多数になった．すなわち，行列サイズが（今回用いた一番大きい場合）512 通り，チューニングパラメタが 576 通りあるので，294,912 個の値がそれぞれ「推定」される．各点は自分自身以外の 511 種類の行列サイズで構築されたモデルから推定されるので，推定値は 150,700,032 個ある．このため常に計算量を気にしながら計算を進めることとなった．

以下ではかなり古い 2.0GHz の Xeon での測定データを分析している．他のマシンのデータでも分析をしているが，ファイルサイズが大きくなるので省略する．

また，以下では今後の研究の参考になるように，試行錯誤をしながら分析をしたことを順次やや詳しく述べてゆく．通常の研究論文の書き方ではないが，ほかの事例で分析をする際に参考になればと考えている．

4.3 推定誤差の分析

まず，行列サイズ n チューニングパラメタ i の所要時間 t_{ni} が，行列サイズ m のモデルから推定した t_{mni} とどのぐらい離れているのかを分析した．最初に平均二乗相対誤差を

$$MSE_{mn} = \frac{1}{M} \sum_i \frac{(t_{mni} - t_{ni})^2}{(t_{ni})^2}$$

として計算した．ここで M は候補の数で，今回は 576 である．

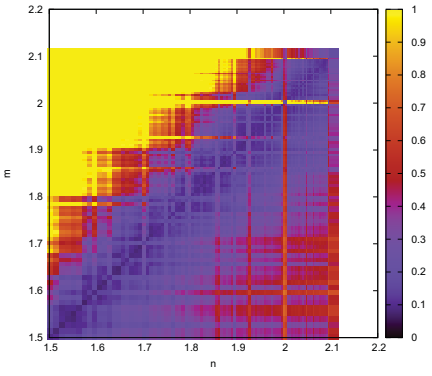

図 1 当初得られた誤差分布

その結果，大きいサイズの性能で小さいサイズの性能を

推定する $n < m$ では精度が悪く，逆に小さいサイズの性能で大きいサイズの性能を推定する $n > m$ では精度がそれほど悪くならなかった．これは当初の予想の逆であった．

そこでさらに調べてみたところ，これは推定精度の良し悪しというよりも，サイズによって推定値にバイアスがかかっている（トレンドがある）ことが原因であることが分かった．そこで次に推定値のトレンドについて分析することとした．

4.4 推定値のトレンドの分析

行列サイズ n チューニングパラメタ i の所要時間 t_{ni} と，行列サイズ m のモデルから推定した t_{mni} との差を分析した．すなわち符号を含めた平均相対バイアス

$$BIAS_{mn} = \frac{1}{M}\sum_i \frac{t_{mni} - t_{ni}}{t_{ni}}$$

を計算した．

その結果，前節で観測された「大きいサイズの性能で小さいサイズの性能を推定する $n < m$ では精度が悪い」という現象は，大きいサイズの性能で推定した値が真値より系統的に大きいことと関係があり，推定値がいくらでも真値から遠く離れうることにより，大きな精度低下を被っていることが分かった．また「小さいサイズの性能で多きサイズの性能を推定する $n > m$ がそれほど悪くない」という現象は，小さいサイズの性能で推定した値が真値より系統的に小さいことと関係があり，（所要時間を意味する）推定値が負の値を取ることはあまりないことにより，真値そのものよりも遠く離れることがなかったことによっていた．

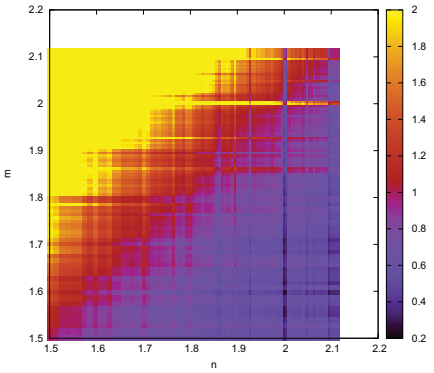

図 2 推定値（真値に対する相対値，両対数）

一定の傾向のあるバイアスは，データを一方向に引っ張ってしまうので，推定結果もバイアスがかかってしまう．このためバイアスの傾向（トレンド）を調査し，トレンドを可能な限り差っ引く (detrend という) ことにした．そこで n と m を両軸に，バイアスを色でプロットすると**図 2** のようになった．図のように，$\log_{10} n$ と $\log_{10} m$ を両軸として，バイアスを色でプロットすると，ほぼ斜め 45° の模様が現れた．そこで，バイアス値が主に

$$l_{nm} = \log_{10} n - \log_{10} m$$

の関数であると仮定した．そして，l_{nm} を x 軸に，バイアス値 $BIAS_{mn}$ を y 軸にプロットすると，**図 3** のようになった．このように明確で一定な傾向がみられたので，バイアス値 $BIAS_{mn}$ を l_{mn} に関する簡単な式でフィッティングし，このトレンドを記述することにした．

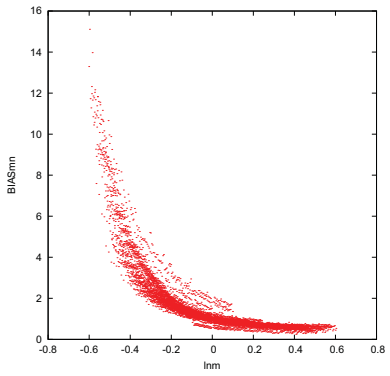

図 3 バイアス vs l_{mn}

最初は単純な最小二乗を試みたが，これはうまくゆかなかった．図 3 のプロットでは，点の数が膨大なので一見してそうとわからないが，実は中央付近に大量の点があり，両端の方は一見密な部分でも，中心付近に比べるとずっと点の数が少ない．このように x 軸方向に著しく不均一なデータに対して最小二乗を適用すると，点の数が多い中心付近にのみ強くフィットし，両端では全然フィットしない関数が得られてしまう．

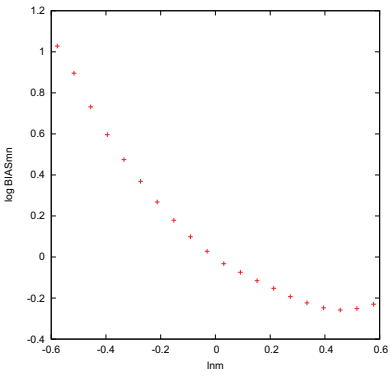

図 4 抽出されたバイアスの傾向

このような状況に対応する方法はいくつか考えられる．ここでは高い精度が必要ではなく，安定的に結果が得られることを優先し，以下のような手法を用いた．まず x 軸方向の範囲を決定し，それを 20 個の区間に分けて，各区間で $BIAS_{nm}$ の平均値をまず出した．これが**図 4** である．この 20 個の平均値を 4 次の多項式に最小二乗法でフィッティングし，これをトレンドのモデル $r(l_{mn})$ とした．ここで「20 個の区間」「4 次の多項式」といった選択はデータ

を見て決定したもので，他の条件でも適用できるという性質のものではない．このため，本手法を完全に自動で適用するというのは適切ではない．

得られた相対トレンドのモデル $r(l_{mn})$ を用いて推定値 t_{mni} を

$$\hat{t}_{mni} = \frac{t_{mni}}{r(l_{mn})}$$

に補正する．望むらくは，$\hat{t}_{mni}$ はバイアスがなくなり t_{mni} をよりよく近似していることが期待される．

4.5 Detrend した推定値の誤差

Detrend した推定値の相対誤差

$$MSE(\hat{t})_{mn} = \frac{1}{M}\sum \frac{\hat{t}_{mni} - t_{ni}}{t_{ni}}$$

を，x 軸に l_{mn} を取ってプロットしたのが**図 5** である．このように対数スケールで比較的広く分布しており，少数のものがほかのものよりもかなり大きいことがわかる．このような場合に単純に平均値を取ると，少数の大きな値に大きく引きずられた値になる．そこで値の対数を取って，それの平均値を取る．これを 20 個の区間ごとに行ったものを**図 6** に示す．これにより誤差の傾向が見て取れる．

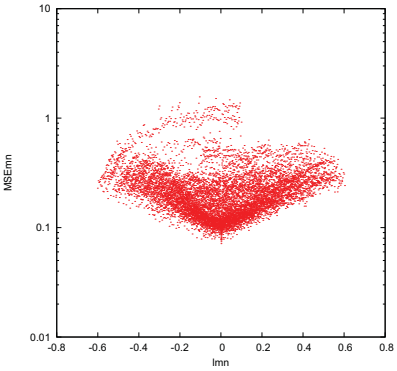

図 5 バイアス補正後の誤差

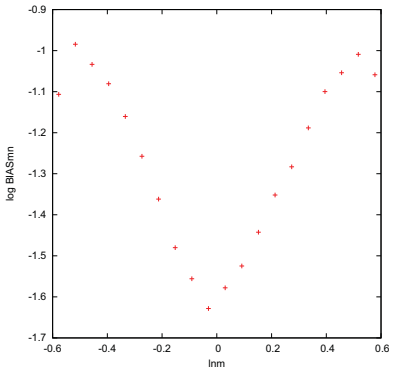

図 6 バイアス補正後の誤差（対数）の傾向

これより，n と m が離れると徐々に誤差が拡大する傾向が見える．また，誤差は原点で最小値を取るが，l_{mn} が正の方向と負の方向とで傾向が違うことがわかる．そこで，l_{mn} が正の部分と負の部分それぞれに関して，$\log MSE'_{nm}$ を近似する 3 次式を作成することにした．

以上により，行列サイズ m で構築された性能モデルから，異なる行列サイズ n の性能を推定する際のおよその誤差が見積もられた．

これらの図に示すように，かなり大きな誤差がみられる．詳しく調べてみると，これは大きく推定を外すときがあって，それら一部の値が，全体の誤差を引き上げていることが分かった．行列積の性能は，いくつかの行列サイズで特殊な性能を示す．これらの行列サイズは特異点であって，その行列サイズで構築された性能モデルは他の行列サイズの性能にほとんどマッチしないし，逆に他の行列サイズで作られた性能モデルは特異的な行列サイズの性能を精度よく推定しない．原因としては，例えば行列サイズ (leading dimension) がキャッシュラインサイズあるいはキャッシュサイズに一致するような場合に，ライン衝突が特異的に頻発するなどが考えられる．

そこで，今回はこのような特異的な情報は外れ値 (outliner) として処理対象から外すことにする．

4.6 外れ値処理をした推定値

外れ値処理の前に，前節で求めたトレンド $r(l_{mn})$ の絶対値が 0.2 以上のものは相関性能として利用しないことにした．これによりもともと精度の悪いと期待されるデータを取り込むことを避ける効果が期待できる．ここで選んだ 0.2 という値は，この程度でも十分な数の相関情報が残っていたからであり，特段の根拠はない．大きく取り過ぎれば，精度の悪い情報を多く取り込むことが懸念され，小さく取り過ぎれば，参照できるデータ数が少なくなってしまう危険性がある．このため適度な値というものがあると思われる．

外れ値の扱いとしては，以下のような単純な方法を用いた．参照できる性能推定値（これらはさまざまな行列サイズ m の性能モデルから得られる）のうちから，値が大きなものから 1/4 と，値が小さいものから 1/4 を捨てて，約半分の値だけを残した．

行列サイズ n の性能を推定するために，上記 2 つの処理をして残った行列サイズ m の値の集合を M_n とする．選ばれた推定値と真値との差の二乗和を

$$MSE'_{ni} = \frac{1}{|M_n|}\sum_{m\in M_n}(1 - \hat{t}_{mni}/t_{ni})^2$$

として評価した．得られた MSE'_{nm} の対数値を，n を x 軸，i を y 軸としてカラーでプロットしたのが**図 7** である．対数を取っているため負の値が多くなっており，それなりの精度で性能が推定できることがわかる．しかしこの値は m ごとの相関を考慮に入れていない単純な誤差の平均値で

あり，これが推定精度なのではない．また，推定精度を推定することも必要である．

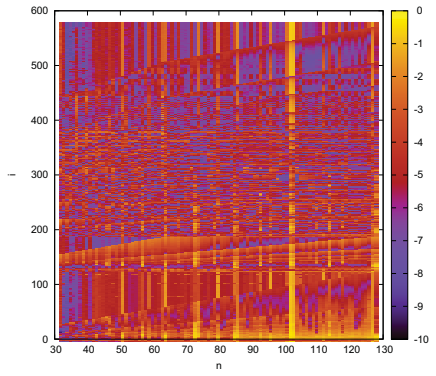

図 7 外れ値処理後の相対誤差（対数）

4.7 Variogram モデルの構築

次に，variogram モデルの構築に取り組んだ．前述のとおり variogram の定義は

$$2\gamma(h) = var(Z(s+h) - Z(s))$$

である．

ここで問題となるのが，この分散，あるいは今回テーマとしている相関とは何を表すかである．相関とは，A と B という 2 つのものがあって，背景として何かが変動したときに，A と B が同じような傾向で変動するということである．本稿の設定では，A と B は，異なる行列サイズ m_A, m_B における性能モデルから推定される性能である．しかし，「背景として変動する何か」とは何であろうか．これを特定しないと，variogram を推定することができない．

今回の設定で，変動するものはいくつか考えられる．例えば，(1) 行列サイズ n とチューニングパラメタ i を固定しても実行ごとに観測される性能値の違い（ばらつき），(2) 推定対象となる行列サイズ n，(3) 推定対象となるチューニングパラメタ i，(4) 推定に用いる性能モデルを作る行列サイズ m，(5) 計算機ハードウェアによる性能の違い，などである．これらのうち，どれを変動させたときに観測されるばらつきが性能相関を引き起こすのだろうか．今回はどれに着目して variogram を導出すればよいのだろうか．

これらのうち (1) は，推定の誤差を生む成分であり，本来は variogram の評価に入れるべきである．ただこれを含めた評価は大変計算量がかかりそうなので，今回は含めなかった．多くの場合，所要時間のばらつきは測定値そのものに対して小さいので，この近似はある程度適切なものと思われる．(2) と (3) は variogram の評価に入らない．確かに異なる n, i に対して異なる性能値が出るが，この違いから見えてくるのは性能モデルの関数形であって，推定の確からしさではない．(4) は一見不思議であるが，空間統計で用いられるものである．位置に依存しない一定の傾向で相関があるとする intrinsic stationary process の仮定そのものである．しかし計算機が出す性能の要因を考えると，intrinsic stationary process ではなさそうである．そこをあえて適用するところがこの手法のミソである．(5) は一見するとこれが我々の期待する変動のように見えるが，今回の設定では variogram に入れるものではない．(5) を必要とするのは，計算機ハードウェアをパラメタ化して，それに関して variogram を構築する場合である．

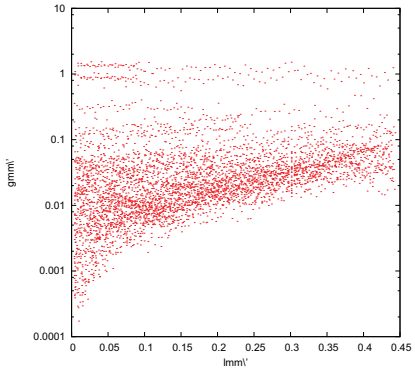

図 8 Variogram 推定値

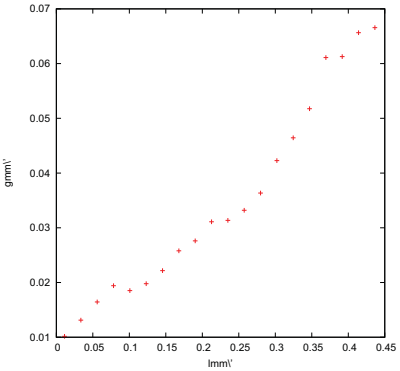

図 9 Variogram 推定値の傾向

そこで，(4) に着目して性能の相関を観測する．正確には (n, i) ごとに相関が定義できるのだが，すべての (n, i) についてまとめて計算をすることにする．所要時間そのままでは (n, i) ごとにスケールが異なってしまうため，およそ 1 になると期待される推定性能比

$$\rho_{mni} = \hat{t}_{mni}/t_{ni}$$

の variogram を推定する．すなわち

$$g_{mm'} = \langle(\rho_{mni} - \rho_{m'ni})^2\rangle$$

（$\langle\cdot\rangle$ は標本平均を表す）をすべての m, m' に対して計算した．定義より $g_{mm'}$ は m と m' に対して対称である．そこで $l_{mm'} = \log_{10} m - \log_{10} m'$ として $l_{mm'} > 0$ のものの

み $g_{mm'}$ をプロットすると**図 8** のようになった．さらに，$l_{mm'}$ の区間を 20 等分して，区間ごとに平均を取ったものは**図 9** のようになった．

Variogram はマシンごとにかなり異なる様相を呈した．今回実験した多くの計算機では，図 9 のように $l_{mm'}$ と $g_{mm'}$ は直線的なグラフとなった．またいくつかの計算機では，**図 10** のように途中で折れ曲がったように見えるグラフとなった．これは縦軸を対数にしてみると直線的に見える．さらに，いくつかの計算機では**図 11** のように，$g_{mm'}$ に上限があるようなグラフとなった．

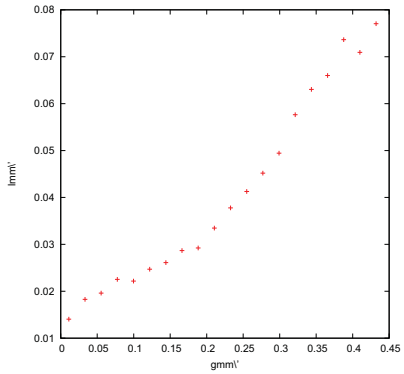

図 10 Variogram 推定値の傾向：折れ曲がる場合

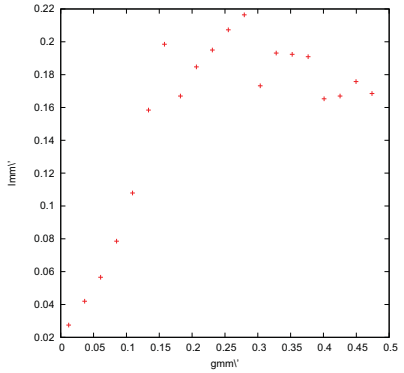

図 11 Variogram 推定値の傾向：上限がある場合

そこで，variogram をフィッティングするのに次の 4 つのモデルを考えた．

(1) 線形モデル．このモデルでは

$$g_{mm'} \approx c_0 + c_1 |l_{mm'}|$$

と近似する．

(2) 上限付き線形モデル．このモデルでは，

$$g_{mm'} \approx \max\{c_0 + c_1 |l_{mm'}|, c_2\}$$

と近似する．

(3) 指数偽モデル．このモデルでは

$$g_{mm'} \approx \exp(c_0 + c_1 |l_{mm'}|)$$

と近似する．

(4) 上限付き指数モデル．このモデルでは

$$g_{mm'} \approx \max\{\exp(c_0 + c_1 |l_{mm'}|), c_2\}$$

と近似する．

(3) を「偽モデル」としたのは，このモデルは variogram の条件を満たさないからである．有効な variogram であるためには，$2\gamma(h)/\|h\|^2 \to 0$ $(\|h\| \to \infty)$ でなければならないが，(3) はこれを満たさないからである．このうちどれを採用するかはグラフを目視して確認することとした．

4.8 性能の推定

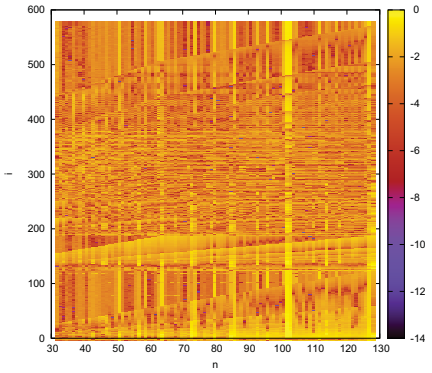

図 12 最終的な性能推定の相対誤差（対数）

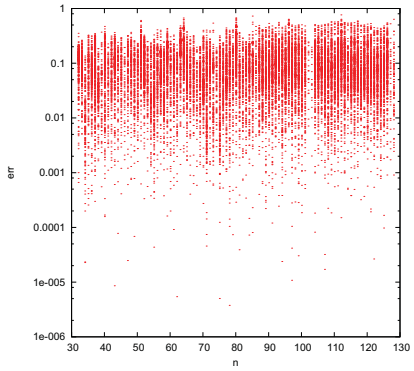

図 13 最終的な性能推定の精度の分布

以上により，行列サイズ m で構築した性能モデルが相互にどのような相関を持つかが分析でき，推定に必要な情報が集約された．そこで，これらの情報を用いて，3.3 節で説明した ordinary kriging により性能推定を行った．すべての (n, i) に対して行った推定の精度を**図 12** にプロットした．また誤差を n ごとにプロットしたものが**図 13** である．

また，推定性能の相対誤差を推定する kriging variance を x 軸に，実際の相対誤差を y 軸に取ったプロットが**図 14** である．実際の相対誤差は広く分布していて，kriging variance との関連は容易には見えない．Kriging variance σ_K^2

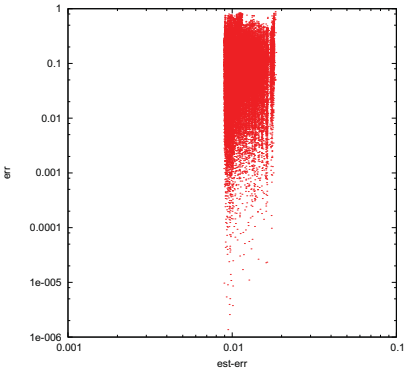

図 14 Kriging variance (x 軸) と実際の相対誤差（y 軸）

は，誤差が σ_K 以下となる確率がおよそ 68%，誤差が $2\sigma_k$ 以下となる確率がおよそ 95%，等々という意味を持つ．そこで誤差が σ_k の整数倍以下となった割合について調べたところ，σ_k 以下のものが 63%，$2\sigma_k$ 以下のものが 86%，$3\sigma_k$ 以下のものが 94%，$4\sigma_k$ 以下のものが 97%などとなった．これより，誤差の分布がやや右にすそが長いものの，およその誤差を推定できていることがわかった．

この結果は計算機によって異なる．計算機アーキテクチャとコンパイラの特性から，性能モデルがよくフィットする場合と，そうでない場合とがある．性能モデルがフィットしない場合には誤差を underestimate しがちである．図 11 の計算機では，$2\sigma_k$ 以下の誤差が 62% で，およそ 2 倍の underestimate になった．さらに右の裾は大変長くなった．

すそが右に長い原因として，特異な性能を示す行列サイズが考えられる．推定に使うデータは外れ値処理をしたが，今回は外れ値にあたる行列サイズでも性能推定をしている．特異な性能を示す行列サイズは，推定性能と実際の性能が大きく違うことで，現実には判別ができると考えられる．このような場合には異なる行列サイズのモデルによる推定値を利用しないことにより，誤った推定を減らすことができる可能性がある．

5. まとめ

本稿では，ある行列サイズ m での性能値から，別の行列サイズ n での性能を推定する問題を考えた．空間統計で用いられている variogram と kriging を用いて，性能の推定と，その推定された性能の精度を推定することができた．

今回，ある程度のもっともらしい値が得られたのは，variogram を与える関数がうまく見つかったことによっている．これ以外の問題でも variogram に相当するものが見つかるかどうか，どのようにすれば見つけられるか，知見を重ねる必要がある．

また，今回は性能と性能推定誤差を推定しただけであった．今後はこのデータを自動チューニングに取り込んで，オンライン・オフラインの自動チューニングの効率化が達成されるかどうか検証する必要がある．

謝辞 本研究は文部科学省科学研究費「汎用自動チューニング機構を実現するためのソフトウェア基盤の研究」，JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」の援助を受けています．

参考文献

[1] Noel A.C. Cressie, “Statistics for Spacial Data,” Wiley, 1993.
[2] R. Suda, A Bayesian Method of Online Automatic Tuning, in: Software Automatic Tuning: From Concepts to the State-of-the-Art Results, Springer, 2010, pp.275–294.
[3] 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）, 2008, pp. 73-80
[4] 須田礼仁，「相関を利用した自動チューニング数理手法」，情報処理学会研究報告 vol.2012-HPC-134, No.10, 2012.