Vol.2012-HPC-134 No.9
2012/6/1

収束障害(Fault Convergence)：数値計算ソフトウェアにおける新しい安全性の概念

片桐孝洋†　櫻井隆雄††　伊藤祥司†　猪貝光祥†††　大島聡史†
黒田久泰††††,†　直野健††　中島研吾†

本論文では，数値計算ソフトウェアで多く用いられている数値反復解法において生じると考えられる収束障害（Fault Convergence）の概念を提案する．数値計算分野で用いられている偽収束（False Convergence）との違いを議論する．Laprie により定義されたディペンダブルコンピューティング実現のための 3 つの脅威―障害(fault)－異常(error)－故障(failure)モデルーを用いて数値反復解法での収束問題を議論することにより，収束障害の一例を示す．

Fault Convergence: A New Concept of Safety for Numerical Computation Software

TAKAHIRO KATAGIRI†　TAKAO SAKURAI††　MITUYOSHI IGAI†††
SATOSHI OHSHIMA†　HISAYASU KURODA††††,†
KEN NAONO†† and KENGO NAKAJIMA†

In this paper, we propose a concept of “Fault Convergence” for numerical iteration methods, which are widely used methods in numerical software. With respect to the difference to the concept of “False” convergence on numerical computation field, we explain a situation that fault convergence occurs. By using the model proposed by Laprie with the 3 kinds of threats to dependable computing—the fault-error-failure model—, we discuss an example of fault convergence situation in convergence problem to numerical iterative methods.

1. はじめに

本論文では，数値計算ソフトウェアで多く用いられている数値反復解法において生じると考えられる収束障害（Fault Convergence）の概念を提案する．数値計算分野で用いられている偽収束（False Convergence）との違いを議論することで，収束障害の状況を説明する．

1985年Laprieにより定義されたディペンダブルコンピューティング実現のための 3 つの脅威[1][2]である，障害(fault)－異常(error)－故障(failure)モデルを基にし，数値反復解法を用いる数値計算ソフトウェアに対するディペンダビリティとは何かを定義する．また，ディペンダビリティ実現のための手段である，フォールトトレラントを実現する方法の 1 つを，数値計算反復解法を用いたシミュレーションソフトウェアの事例で紹介する．そのため，数値計算ソフトウェアにおける収束問題を定義することから始める．

なお本論文では，ディペンダブルコンピューティングで頻出する以下の用語を以下のように訳した：

障害(fault)，異常(error)，故障(failure)[a].

† 東京大学情報基盤センター スーパーコンピューティング研究部門
Supercomputing Research Division, Information Technology Center, The University of Tokyo
†† 日立製作所 中央研究所
Central Research Laboratory, Hitachi Ltd.
††† （株）日立超 LSI システムズ
Hitachi ULSI Systems Co.,Ltd.
†††† 愛媛大学大学院理工学研究科
Graduate School of Science and Engineering, Ehime University

1.1 数値計算ソフトウェアにおける収束問題

本論文における数値計算ソフトウェアにおける収束問題を以下に定義する．

【定義】数値計算ソフトウェアにおける収束問題

数値計算アルゴリズム上，数値計算ライブラリの実装上，もしくは，その双方に起因する事項により，エンドユーザーからの要求誤差を満たさない状態で収束と判断されること．

上記の定義では，数値反復解法の理論上は正しく収束するが，数値計算ライブラリの実装の都合で収束問題を生じることも含んでいる．たとえば，丸め誤差の累積，数値計算上必要となるパラメタを理論上誤った指定で実行すること，なども含まれる．

本稿では，フォールトトレランスで通常想定する，ハードウェアに起因するが異常検出ができない状態の「ソフトエラー」は取り扱わない．つまり，計算の途中で何らかの原因によりデータの一部が壊れるが，異常検知が出来ない状態で，かつ計算が進んでいく状況は考慮しない．ソフトエラーの考慮は，フォールトトレランスを考慮した反復解法の構築上きわめて重要であるが，本原稿ではソフトエラーが生じない状況で生じる収束問題を取り扱う[b].

a　たとえば参考文献[2]では，以下のように訳している：欠陥(fault)，誤り(error)，障害(failure)．ここでは，障害がおきてもシステムは動作しているという意味にとり，サービスが停止する状態 failure を故障と定義した．
b 場合により，本稿で提案する機構により，ソフトエラーに対する対策に

1.2 本論文の構成

本稿は以下の構成からなる．2 節で数値計算ソフトウェアにおける収束問題に対するディペンダブルコンピューティングとフォールトトレランスについて議論する．3 節は，収束障害の提案である．4 節では，収束障害を回避する機構の一実装について予備評価を行う．5 節は関連研究の紹介である．最後にまとめを行う．

2. 数値計算ソフトウェアにおける収束問題に対するディペンダブルコンピューティングとフォールトトレランス

2.1 収束問題に対するディペンダブルコンピューティングとフォールトトレランス

Laprie の論文[1]で定義されたディペンダビリティのための手段(Means)は，以下の 4 つである：
（１）障害回避(Fault Avoidance)；（２）フォールトトレランス(Fault Tolerance)；（３）異常除去（Error Removal）；（４）異常予測(Error Forecasting).

これらディペンダブルコンピューティング実現のための手段を，数値計算における収束問題の観点から検討すると，以下の事項になる．

- **障害回避（Fault Avoidance）**
  収束問題が生じないように，数値反復解法，もしくは，数値ライブラリの構成要素（手続き，関数など）を作り直す．たとえば，偽収束(false convergence)が理論上発生しない数値解法を新たに開発すること，適切な前処理を施し収束性能を改善すること，知られている堅牢な数値解法を実装すること，などが相当する．

- **フォールトトレランス（Fault Tolerance）**
  冗長実行，もしくは複数実行などにより，障害がある状況，もしくは障害が起こっている状況でも，継続して求解を続行できるようにすること．障害が生じても，数値計算ライブラリとしての求解サービスの継続ができるようにすること．

- **異常除去（Error Removal）**
  例えば，真の解からの差（残差）を検証(verification)することにより，起こる異常の回数を最小化すること．

- **異常予測（Error Forecasting）**
  真の解からの差（残差）を評価(evaluation)することにより，異常の存在，異常が発生する閾値，および将来における異常の連続発生を予測すること．

計算機システムと同様に，数値計算ソフトウェアにおいても障害回避のアプローチには限界がある．その理由は，提案されている数値解法において障害を防ぐための理論的もしくは技術的な方法が提案されていない場合があるからである．また仮に，障害を防ぐ数値解法が理論的に提案されていても，利用している数値計算ライブラリに有効な既存方法が実装されている保証がない．エンドユーザー自身がシミュレーションコードを開発している場合は解決が容易であるが，フリーソフトウェアを利用している場合には障害を防ぐ実装を行うことは困難である．

シミュレーション実施者のエンドユーザーが解くべき問題の特性を熟知している場合，適切な前処理方式の指定により収束性を改善させ，障害を生じなくすることが可能である．ただしこの場合でも，適する前処理方式を理論的に解明できていない場合や，利用している数値計算ライブラリに有効な前処理が実装されておらず，かつ，エンドユーザー自身でその有効な前処理を実装できない場合は，障害が回避できるかは未知である．

また，エンドユーザーが予期せぬ，実行時に生じる何らかの数値的な不安定性から生じる障害については，事前知識のみでは対応できない．

以上の議論から，本原稿では障害の回避は実現できないという立場を置く．この状況下では，計算機システムのディペンダビリティと同様の立場で，数値計算ソフトウェアにおける収束問題を解決するフォールトトレランスのアプローチを採用する．

2.2 数値計算ソフトウェアにおける収束問題に対する障害(fault)－異常(error)－故障(failure)モデル

Laprie の論文[1]で定義された，ディペンダブル実現のための脅威は，以下の 3 種類である：
（１）障害(fault)；（２）異常(error)；（３）故障(failure)；この 3 種は，独立に生じるのではなく時系列がある．

障害を起因として，異常が生じる．異常は障害が顕在化した状態であり，異常は正しい状態からのズレが定義され判明した状態である．異常が顕在化すると故障を生じさせる．故障に至る連鎖の結果，事故（重大なサービス障害）を生じさせる．この障害－異常－故障の連鎖は，一般的に階層性をなす．以上を**障害－異常－故障モデル**と呼ぶ．

障害－異常－故障モデルを，数値計算ソフトウェアにおける収束問題に適用する．図 1 に適用の一例を示す．

なることがある．本提案によるソフトエラー対応の可能性があることを認識しているが，本題からずれるため説明はしない．

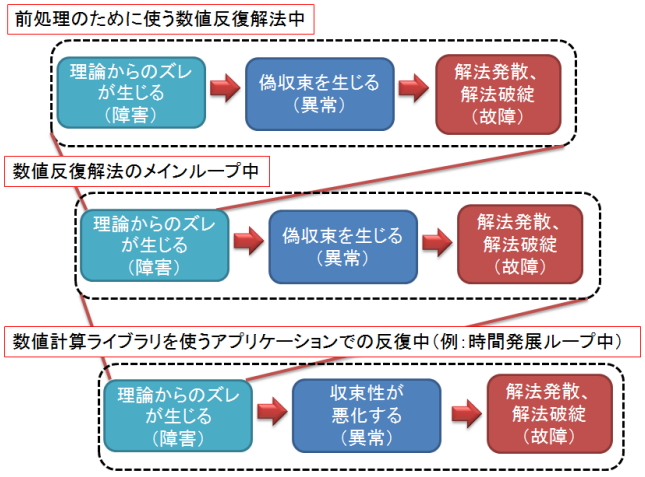

図 1　数値計算ソフトウェアにおける
収束問題に対する障害－異常－故障モデルの例

図 1 では，数値シミュレーションは複数の反復からなることを前提とする．

最下層は，数値計算ライブラリなどを使っているアプリケーション(シミュレーションコード)中での反復である．

中階層は，数値計算ライブラリ内部の反復である．これは，数値反復解法中のメインループを指す．

最上位階層は，数値反復解法中のメインループ中の前処理で使われる反復などである．これはたとえば，前処理として近似逆行列を求める場合，なんらかの数値反復解法のループがあり，これが相当する．

以上は構成の一例である．したがって，図 1 よりも多い階層性があってもよいし，逆に階層の数が少なくてもよい．また図 1 の例では，最上位と中階層の処理は数値計算ライブラリ（線形反復解法）のため，障害－異常－故障モデルの要因は同一となっている．一般的に各階層における，障害，異常，故障の要因は同一でなくてもよい．

3. 収束障害（Fault Convergence）の提案

3.1 収束障害（Fault Convergence）とは

前節の議論から，数値計算ソフトウェアにおける収束問題において，障害－異常－故障モデルにより求解サービスが出来なくなる状況を収束障害（Fault Convergence）と呼ぶ．特に，「障害」段階が存在するということに注目した概念である．

本論文は，収束障害の概念を提案することにある．収束障害の概念を認識することで，解法発散・解法破綻という故障に至る故障連鎖を早期の段階で防ぐ，もしくは，減少させる機構の提案をすることにある．

3.2 偽収束（False Convergence）との関係

収束障害を認識するにあたり，数値解析分野で用いられている**偽収束（False Convergence）**との違いを確認することは重要である．

偽収束とは，「真の残差ベクトルを用いた相対残差ノルム」では停滞しているが，「アルゴリズム中の残差ベクトルを用いた相対残差ノルム」では収束している事象である．またこれから転じ，反復解法中の理論的に得られる残差(相対的な残差を含む）が真の残差と異なる状況も指す．

図 1 の障害－異常－故障モデルにおいて，偽収束は「異常」の状態といえる．なぜなら異常とは，仕様上正しい状況からの差分を定量的に検出した状態と定義できるからである[c]．偽収束は，＜数値解を基に算出して得られる真の残差ベクトル＞と＜計算解から得られる残差＞の差分について定量的に検出できる状況である．したがって，＜異常＞に分類される．

ここで提案する**収束障害は，偽収束と判定されないが，しかし偽収束につながる以前の状態**である．また収束障害時，当該階層より上位階層が偽収束状態になることも含んでいる（上位階層が障害状況になることも含む.）.

図 2 に，収束障害が生じた一例を示す．

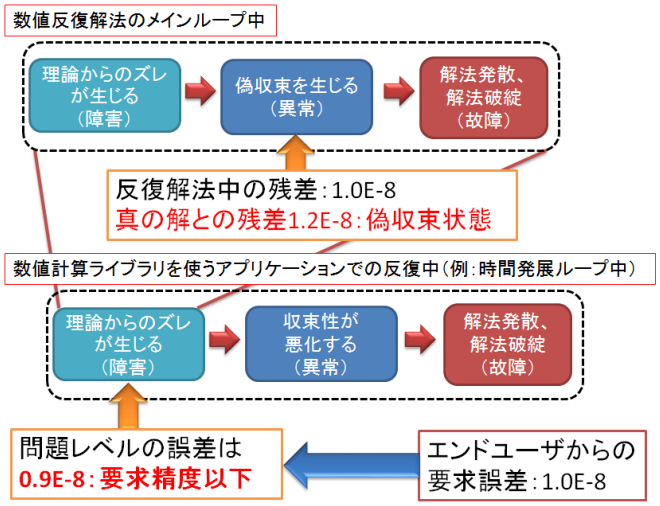

図 2　最下位層で収束障害が生した例
（上位階層では異常が生じている）

図 2 は，あるシミュレーションソフトウェアにおいて数値計算ライブラリを使っている例である．シミュレーションソフトウェア開発者（数値計算ライブラリのエンドユーザ）の要求誤差を 1.0E-8 とする．このとき，何らかの手段により，問題レベルの誤差がシミュレーション中で評価され，シミュレーションの終了を判断する．

一方で，利用している数値計算ライブラリの数値反復解法の収束基準も，シミュレーションでの収束基準と同じ基準で実装されていると仮定する（ただし，この仮定を置く必要はなく，何らかの値が設定されているとしてよい）.

c プログラム上の「エラー」とは，仕様上定義された機能と，実行における挙動が明確に（定量的に）異なると判明した時の状態である．仕様上の機能と，実行時の機能との違いがあるかどうかわからないが何らかの欠陥があるかもしれない状態はエラー（異常）ではなく，障害(fault)に相当する．

以上の前提で，数値計算ライブラリ上で反復終了時に偽収束を生じているとする．利用している数値計算ライブラリには偽収束の検出機能や修正機能が無い場合や，仮に偽収束を検出しても処理を続行する仕様となっている場合，下位層のシミュレーションソフトウェア上では正常な解として扱われる．この解を利用しても，問題レベルの誤差は要求精度以下になったとしよう．

この状況では，シミュレーションソフトウェアの仕様上は正常動作である．しかしながら，数値計算ライブラリ上では偽収束状態（異常）である．シミュレーションソフトウェア上の誤差計算による異常は検出されないので，異常ではない．しかしこの状況は，何らかの理由により障害の度合いが悪化すると，シミュレーションソフトウェア階層で異常が発生するかもしれない．つまり，異常の前段階の状態と考えられる．すなわち＜障害＞状況にある．

これが，収束障害を生じる一例である．

別の角度から見ると，最下層であるシミュレーションソフトウェア階層の障害検知をするため，より上位階層で生じる異常検知を用いるといえる．この意味において図 2 の例は，上位階層においては異常除去，下位階層においては異常予測を行っているとみなすことができる．

3.3 収束障害（Fault Convergence）を回避する機構の一構成例

図 2 の収束障害を回避する，もしくは，収束障害発生回数を削減するためには，シミュレーションソフトウェア階層より**上位階層の「異常」状態を早期に検知する機構を導入する**ことが肝要である．シミュレーション階層における障害検知(Fault Detection)を行い，収束障害を防ぐ．

この概念に基づき，回避のための機構の一構成を提案する．図 3 にその機構の構成をのせる．

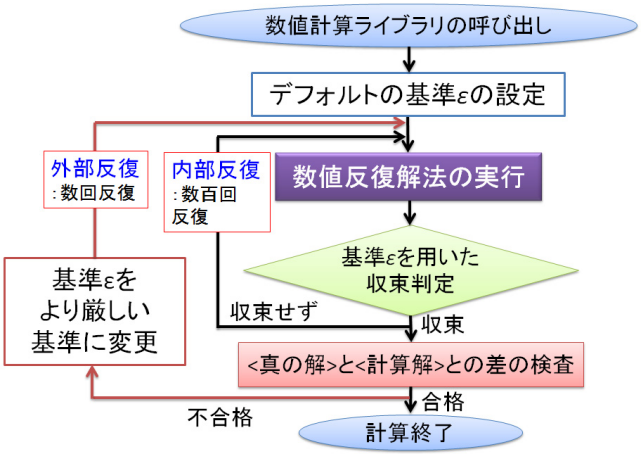

図 3　収束障害を回避する機構の構成例
（数値計算ライブラリ階層）

図 3 では，シミュレーションソフトウェア階層より上層の数値計算ライブラリ階層で偽収束を検知し（すなわち，異常検知を行い），自動的により厳しい収束条件 ε を与えることで，シミュレーションソフトウェア階層での障害を予測し自己修復する機構といえる．図 3 では，数値反復解法に相当する構成要素について，理論上や実装上の工夫により偽収束の回避が保証できない状況を想定していることに注意する．

図 3 では，内部反復と外部反復の 2 種が存在する．

内部反復は，数値反復解法での主ループであり，数百～数千回の反復が想定されるループである．一方で外部反復とは，数値計算ライブラリ階層での異常の修復のためのループであり，数回の反復を想定する．

図 3 の，基準 ε をより厳しい基準に変更する方法は多種存在する．またこの方法は，エンドユーザーから与えられる最適化方針（ポリシー）に依存する．

たとえば，演算時間が演算精度より重要である状況下でのポリシーと，演算精度が演算時間より重要である状況下でのポリシーは全く異なるはずである．そのポリシーを定めないと，適切な基準 ε の設定はできない．そこで，数値計算ポリシー[3]の概念により，AT 機構に最適化のポリシーを与える機構の提案がなされている．図 3 の機構と数値計算ポリシーの併用が望ましい．

図 3 のより厳しい基準に変更する処理は，収束判定基準 ε の変更にこだわる必要はない．別の前処理の適用や数値反復解法の選択も，広い意味での収束基準の変更に相当する．したがって，数値計算法自体の変更も，本機構の範疇であることに注意する．

3.4 自己安定化（Self-Stabilization）との関係

主に分散システムにおけるフォールトトレラントの概念として使われる自己安定化(Self-Stabilization)[4]では，開始された状態（state）にかかわらず，定めた最終状態まで，有限の状態遷移で到達できることを収束(Convergence)と呼ぶ．この収束までの時間を Convergence Time と呼ぶ．特に Convergence Time は，ネットワーク機構におけるフォールトトレランスで用いられていることが多い．

数値反復解法における収束は，ある許容誤差範囲に解が収束することである．したがって両者の違いは，自己安定化における収束では離散状態を取り扱うが，数値反復解法では連続量の変化を取り扱うといえる．取り扱う対象の性質が異なり，最適化のアプローチも異なる．

一方，離散と連続の違いはあるにせよ，ある目的まで反復的に処理を繰り返し，自ら収束点に近づけるという安定化を追究する点は，両者で共通といえる．

本論文で示した収束障害を回避するため，収束時間(convergence time)を最小化することは重要な研究項目である．具体的には，図 3 の外部反復の回数を最小限に抑えることは重要である．この点において両者は共通である．

3.5 ソフトウェア劣化（Software Aging）との関係

収束障害は､異常の要因が徐々に累積して故障を生じるソフトウェア劣化 (Software Aging)[2]の範疇といえる．

また，収束障害を回避する機構の導入は，異常検出後，異常処理なしに直ちに障害処理を行うことといえる．すなわちこれは，故障に至る前にその部品（手続き）に内在する問題因子を取り除くことである．これは，ソフトウェア若化（Software Rejuvenation）[2]に相当する．

3.6 品質管理との関係

収束障害を防ぐことは，数値計算ソフトウェアにおける安定性を高めることに貢献する．すなわち，不安定な状態によるサービス停止を防ぐためのソフトウェア安全性の改善に寄与する．これは，数値計算における品質管理の側面からみたことと同値である．

伊藤[5]は数値反復解法における品質管理の手段として，前処理方式と数値反復解法の組合せを自動的に調べるサーベーチャートを考案した[5]．このサーベーチャートは，反復解法の専門知識のない素人にも，偽収束に陥る数値解法の組合せを知ることができる観点から有益である．従来よりも広範なエンドユーザーのプログラムに対して，収束障害に陥る事態を減少させることができる．この意味において，品質管理手法も収束障害の回避に寄与する．

3.7 自動チューニング（AT）との関係

収束障害を防ぐ目的を達成するには，反復解法の専門知識を習得しエンドユーザーが手動でそれをおこなうことも１つの方法である．また，3.6 節のサーベーチャートを用いたツールを利用することで，専門知識のないユーザーにおいても手動でおこなうことが１つの手法である．ただしこれら手動の方法は，少なからず収束障害回避のためのコスト―すなわち,理論の基礎学習時間,収束性の調査時間,収束障害回避のためのプログラミングおよびデバックの時間―の増大が生じる．

工学的な意味合いにおいて，コストの側面が効果の側面を凌駕すると，適用不能となり空疎な技術となる．一方，効果の側面がコストの側面を凌駕するのは理想だが，一般的になんらかの厳しい制約を置かないと実現できないので,殆ういう技術となる．したがって，両者のトレードオフとなる中庸的な技術が望ましい．

専門的な知識を利用せず，全自動に収束障害を防ぐ技術があれば，工学上有益である．その技術の 1 つが自動チューニング（Auto-tuning, AT）技術[6]である．図 3 の機構において，自動的にチューニングパラメタである反復解法の収束基準 ε を調整する機能を提供すれば AT 機構の範疇となる．実際，後述のようにこの実装は可能である．

3.8 適用制限

図 3 の機構では，収束基準 ε をきつくしても偽収束を防げない状況では効果を奏しない．

しかし再度記述するが，図 3 の収束基準の変更を，収束基準 ε の変更のみに限定してとらえる必要はない．より一般な偽収束を防ぐ手段ととらえるべきである．

たとえば，前処理や数値反復解法の自動変更が，＜より広範な＞偽収束を防ぐ手段となろう．ただし 3.7 節で議論したように，この変更を自動で行うことがコストの観点から望ましい．AT 機能として実現されるべきである．

この前処理や数値反復解法の自動選択のための AT 機能は，OpenATLib ver.1.0[7][8]に実装されている．

4. 収束障害を回避する機構の予備評価

4.1 目的

本節では，図 3 で提案した機構を基にし，数値計算ライブラリ階層において異常検知し，異常を自己修復する機構の予備評価を行う．この機能により，数値計算ライブラリ階層より下位階層のシミュレーションソフトウェア上での収束障害を回避に寄与することを狙う．

4.2 評価環境

図 3 の機構の効果を検証するため，図 3 と等価の機構を OpenATLib ver.2011 の数値計算ポリシー機構における「演算精度ポリシー」に実装した．

対象となる数値計算ライブラリは，固有値ソルバーである．エンドユーザーによる要求精度は 1.0E-8 である．また実行が 600 秒を超えると，強制的に反復を打ち切る．具体的な処理は以下である．

- OpenATI_EIGENSOLVE(LANCZOS)：
  対称実数疎行列の標準固有値問題の反復解法である Lanczos 法による少数の固有値，固有ベクトルの求解ルーチン．
- OpenATI_EIGENSOLVE(ARNORDI)：
  非対称実数疎行列の標準固有値問題の反復解法である Arnordi 法による少数の固有値，固有ベクトルの求解ルーチン．

求める固有値，固有ベクトルの個数は，絶対値最大の固有値から数えて 10 個である

従来の数値計算ライブラリによる実行を，OpenATLib の「演算速度」ポリシーでの実行と仮定する．すなわち，計算速度の最適化しか行われない数値計算ライブラリとする．一方で,「メモリ量」ポリシーとも比較する．これは，速度ではなくメモリ量を最小化した最適化機能を提供する場合であり，演算速度ポリシーとの比較として興味深い．

図 3 の機構では実装パラメタが存在する．本予備評価に

おける実装の詳細を以下に載せる．

- **収束基準 ε のデフォルト値および修正方法**
  - **デフォルト基準 ε ：**
    エンドユーザーからの要求精度を直接設定する．
  - **より厳しい基準に変更する方式：**
    以下の式により，内部反復における収束基準を外部反復で直接設定．

$$\text{新}\ \varepsilon \ = \ \text{旧}\ \varepsilon \ / \ 10 \qquad \ldots (1)$$

なお，真の解と計算解との差の検証は，残差ベクトル $Ax-b$ を直接計算し，その 2 ノルムから判定する．

本評価で利用した，テスト行列および計算機環境を以下に載せる．

- **テスト行列：**
  テスト行列として，フロリダ大学疎行列コレクション[9]から，固有値ソルバーに対して，21 種（対称行列），および 22 種（非対称行列）を選んだ．

- **計算機環境：**
  評価対象は東京大学情報基盤センターが所有する T2K オープンスパコン（東大版）の 1 ノード（16 コア）である．最大のスレッド実行数は，1 ノードあたり 16 スレッドである．計算ノードは，4 ソケットの Quad-Core AMD Opteron Processor 8356（2.3GHz）で構成され，ノードあたりのメモリ量は 32GB である．OpenATLib は OpenMP を利用して並列化されている．コンパイラは Intel Fortran Compiler Professional Version 11.0 であり，コンパイラオプションは -O3 -m64 -openmp -mcmodel=medium である．

4.3 実験結果

図 4 に，対称固有値ソルバーに図 3 の収束障害回避機構をいれた場合の効果を示す．

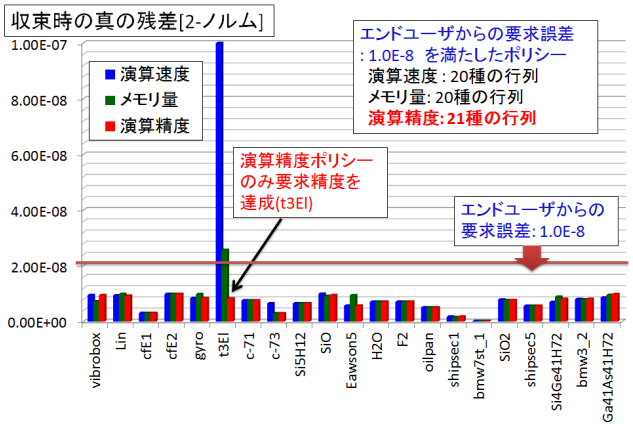

図 4　対称固有値ソルバーにおける
収束障害回避機構の効果

図 4 から，21 種の行列のうち，t3El のみが，速度ポリシーとメモリ量ポリシーでエンドユーザーからの要求精度を満たさない偽収束状態（異常）を生じた．

一方で，演算精度ポリシーに図 3 の機構を導入すると，偽収束状態を回避できた．特に，速度ポリシーにおける真の残差に対する誤差が 7.96E-6 と大きく，実行速度だけ優先すると，偽収束時の演算精度に対するペナルティも大きくなることを示唆している．

図 5 に，非対称固有値ソルバーに図 3 の収束障害回避機構をいれた場合の効果を示す．

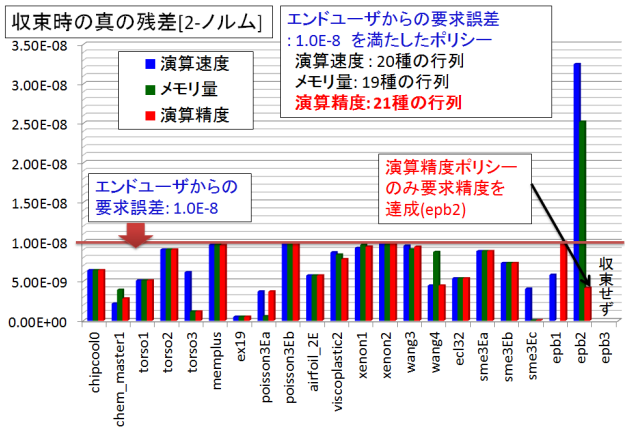

図 5　非対称固有値ソルバーにおける
収束障害回避機構の効果

図 5 から，22 種の行列のうち，epb2 が演算精度ポリシーとメモリ量ポリシーでエンドユーザーからの要求精度を満たさない偽収束状態（異常）を生じた．また，ebp1 では，メモリ量ポリシーで，制限時間以内に収束ができなくなった．epb3 では，すべてのポリシーで制限時間以内に収束できなかった．

一方で，演算精度ポリシーに実装した図 3 の機構を導入すると，21 種の行列すべてで偽収束状態（異常）を回避できた．対称の場合と同様に，速度ポリシーでは精度が 3.25E-8 と一番低い．また，epb1 のように，メモリ量ポリシーだと収束しない事態を生じた．これは，故障相当のサ

ービス劣化となる悪いケースでもある．

4.4 考察

今回用いた行列の問題集，および収束条件は解きやすく収束性が高いものであると判断されるため，偽収束を生じる状況が少ない．それでもこの予備評価を通じ，いくつか有益な知見が得られた．

- **速度ポリシー**は，偽収束時の精度劣化が最も悪い．この理由は，直交化方式など演算精度を高める数値計算の部品レベル，および，数値反復解法の内部ループにおける収束条件を甘めにしても反復回数を削減するなど，設計方針が速度重視になっていることが原因と推定される．
- **メモリ量ポリシー**は，演算精度ポリシーよりも精度が良い場合がある．これは，メモリ量を増やしてスレッド並列化するなどの高速化の方針に対し，逐次実行にしてもメモリ量を削減し，メモリ量が少ないが精度が高い数値計算の部品を選択する（たとえば，直交化処理における修正グラムーシュミッド法の採用など）による設計方針の違いと推定される．
- **メモリ量ポリシー**は収束しないなど，重大なサービス停止の事態（故障）を生じることがある．この主な理由は，リスタート付きの数値反復解法では，メモリ量を削減するためにリスタート周期を小さく取りすぎると，線形代数の理論上において収束性の悪化，もしくは，収束しない状況になることがあるからである．

一般に，収束障害を回避する機構を入れると，演算速度の低下やメモリ量の増大を招く．これらのデメリットが許容できるかは，エンドユーザーの置かれた状況に基づく判断しかない．したがって，数値計算ポリシーとフォールトトレランスの関係は強く，かつ技術的に連携しないと実用に耐える技術は創成できない．

5. 関連研究

以下に数値反復解法におけるフォールトトレランスに関連する研究のいくつかを列挙する．

ハードウェア故障に対するソフトウェア上のフォールトトレランス

アルゴリズムベース（Algorithm-based Fault Tolerance）で，ハードウェアレベルの故障に対応するフォールトトレラントを数値反復解法に適用する流れがある．

たとえば，Mark Hoemmen ら[10]は，フォールトトレランス機能を，連立一次方程式求解のための反復解法 GMRES に適用する研究（FT-GMRES）をしている．この研究では，前処理を反復ごとに変える Frexible GMRES の考え方をフォールトトレランスに拡張し，内部ループ（ディペンダブルでない計算）と外部ループ（ディペンダブルな計算）に分けている．この，内部ループと外部ループの構成は本提案における構成方式と類似している．本提案との違いは，異常検知の方法である．FT-GMRES では GMRES 内部の直交化処理の情報を利用するのに対し，我々のものは残差情報を利用する．したがって，我々の方式のほうが数値解法に依存しないという意味で汎用性がある．

一方，同じくアルゴリズムベースで，サイレントエラーの回避を目的とした固有値ソルバーの開発（櫻井-杉浦法）が，白砂と櫻井[11]により進められている．

特定の反復解法で偽収束を防ぐ AT 方式

連立一次方程式求解のための反復解法 IDR(s)法において，偽収束を防ぐ AT 方式の研究が櫻井ら[12]によりなされている．提案方式は，AT 方式として実装されている．櫻井らの AT 方式は，一種の異常予測（Error Forecasting）方式と分類できる．この観点から，収束障害の回避に貢献できるアルゴリズムの 1 つといえる．

6. おわりに

本稿では，数値計算ソフトウェアで多く用いられている数値反復解法において生じると考えられる収束障害（Fault Convergence）の概念を提案した．数値計算分野で用いられている偽収束（False Convergence）との違いを議論することで，収束障害の状況を説明した．

ディペンダブルコンピューティング実現のための 3 つの脅威を用いた障害(fault)‐異常(error)‐故障(failure)モデルを基にし，数値計算ソフトウェアにおける収束問題に対するディペンダビリティとは何かを定義した．

シミュレーションソフトウェア階層に対して上位階層に当たる数値計算ライブラリ階層における異常を検知し，自己修復する機構を導入することで，下位階層に当たるシミュレーションソフトウェア階層の収束障害を回避する機構の一例を示した．

本予備評価は，数値計算ライブラリ階層における異常検知と異常回避のみ評価したものである．シミュレーションソフトウェア上の収束障害を実際に回避した評価となっていない．一般的に困難と予想されるが，収束障害がおこるケースを同定すること，および，提案機構を組み込み実際にそれが回避できるかどうか検証することが重要な今後の課題となる．

謝辞 本研究は文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」，シームレス高生産・高性能プログラミング環境，の支援により行われた．また，ディペンダブルコンピューティングの資料に関

し教授いただいた松野裕特任講師，フォールトトレランスについて議論いただいた實本英之助教に感謝する．

参考文献

1) Laprie, J-C.: Dependable Computing and Fault Tolerance: Concepts and Terminology, Proceedings of FTCS-15, pp.2-11, 1985.
2) 木下佳樹、松野裕、高村博紀、武山誠訳：対訳 ディペンダブル・セキュアコンピューティングの基本概念と用語, 算譜科学研究速報，独立行政法人産業技術総合研究所システム検証研究センター，AIST-PS-2009-008, 2009.
3) 直野健, 猪貝光祥, 木立啓之: 数値計算ポリシー入力型自動チューニング方式, 情報処理学会研究報告, 2005-HPC-81, 31-36, 2005.
4) Schneider, M.: Self-Stabilization, ACM Computing Surveys, Vol. 25, No.1, pp.45-67, 1993.
5) 伊藤祥司: 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:<ソフトウェア自動チューニングを支える基盤>，5．ソフトウェア自動チューニングのための支援ツール，情報処理，Vol.50 No.6, pp.499-504, 2009.
6) 片桐孝洋: ソフトウエア自動チューニング―数値計算ソフトウエアへの適用とその可能性―，慧文社，ISBN4-905849-18-7, 2004.
7) 片桐孝洋，櫻井隆雄，黒田久泰，直野健，中島研吾：Xabclib：汎用的自動チューニングインターフェース OpenATLib を利用した反復解法ライブラリの開発，応用数理，20 巻 4 号，pp.25-37，2010.
8) Xabclib プロジェクトホームページ，http://www.abc-lib.org/Xabclib/index-j.html
9) University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/index.html
10) Hoemmen, M. and Heroux, A.M.: Fault-tolerant Iterative Methods via Selective Reliability, WEB documentation. http://www.sandia.gov/~maherou/docs/FTGMRES.pdf
11) 白砂渓，櫻井鉄也：周回積分を用いた固有値解法の耐障害性について，日本応用数理学会年会講演予稿集，pp.93-94, 2011.
12) 櫻井隆雄，直野健，猪貝光祥：反復解法 IDR(s)法における偽収束問題と自動チューニング(<特集>数値計算のための自動チューニング(続)), 応用数理，Vol.20, No. 4, pp. 287-296, 2010.