行列演算用言語 LAMAX-S の設計とその処理系の最適化について

1995 年 7 月

内田智史

目 次

第 1 章 序論

さまざまな分野の科学技術の研究・開発においてコンピュータを用いた数値計算は必要不可欠な技術である。そこで計算される数学モデルは行列の形で表現されることが多い。ところが、現在の主要な汎用プログラミング言語である FORTRAN や C などで、これらのプログラムを作成するには、実にさまざまな問題点がある。

まず、多くの常用プログラミング言語では、行列をはじめとする多くの数学的概念が取り入れられていない点である。一般的には、単一の値を代入することのできる変数と同一の型のデータを連続して配置して使用する配列が用意されているだけである。行列やベクトルは、この配列の機能を利用してプログラムの作成者がプログラム化しなければならない。この作業は、プログラミング作業としては比較的単純なものであるが、煩雑でミスが入りやすい。また、作成されたプログラムからもとの数式モデルを読み取ることも複雑な作業となる。

また、連立一次方程式を解いたり、固有値を求める場合、たいていはライブラリ中のサブルーチンを利用する。しかし、サブルーチンコールは、サブルーチン名の命名規則の制限 (たとえば、FORTRAN77 では 6 文字) により短縮化された機能名のあとに、必要なパラメタを引数として羅列しただけの記述形式であるので、もとの数学的表現からはかなり逸脱する。

さらに、多くのプログラミング言語の処理系には最適化機能が備わっているが、これにも指摘すべき点がある。最適化機能には、たとえば、「ループ内の同一計算をループの初期化部分へ移動させる」といったものが多く、手順的な命令の実行順序の変更が中心である。プログラミング対象のモデルのもともとの数学的な特性などを反映した最適化ではない。

このような問題点があるので、多くの研究者がプログラミング作業に多大の開発時間を費やさざるを得ない状況である。結果的にプログラム開発作業の非効率性が本来の研究の進展の効率を低下させている要因の一つになっている。これを改善するため、数多くの行列演算用言語が作成され、プログラム開発作業の圧縮にある程度の効果を上げてきた。しかし、スーパーコンピュータを用いて自然現象を解析するためのプログラム作成、あるいは、オペレーションズ・リサーチなどにおける大規模問題を解くためのプログラム作成などでは、プログラムの効率 (記憶容量・実行速度) が重要な要因となってくる。従来の行列演算用言語はインタプリタを中心とするものが多く、その記述性や読解性に特に重点が置かれ、どちらかというとプログラムの実行効率は軽視される傾向にあり、このようなシステムの利点が十分に活かせなかった。

このような背景から、著者と著者の研究グループは、数値計算の研究やプログラム開発あるいは教育のためのプログラミング環境を整えるために、**LAMAX** (LAnguage for MAtriX) と呼ばれる一連の行列演算用言語の設計とその処理系の開発を行ってきた。

まず、1981 年 3 月にミニ **LAMAX**[51] と呼ばれる行列演算用言語を作成した。ミニ LAMAX は、青山学院大学情報科学研究センターで約 1 年間一般公開され、主に計量経済などのプログラミング実習や経営学部・経済学部の卒業研究などに用いられた。この経験をふまえて、1981 年秋から次の LAMAX の文法の設計 [52] が始まり、1982 年 9 月にその処理系が完成した [54]。これは **LAMAX-E**(E は Extended の略) と呼ばれている。LAMAX-E は、ミニ LAMAX を根本的に改善したもので、ミニ LAMAX とはまったく互換性がない。ミニ LAMAX および LAMAX-E の処理系は、ともに FORTRAN77 プログラムを生成するプリプロセッサとして実現されている。LAMAX-E は、1982 年 10 月から青山学院大学情報科学研究センターで一般公開され、十分な運用経験を積んだのち、1983 年 11 月に東京大学大型計算機センター [20]、1984 年 1 月に北海道大学大型計算機センター [21] で一般公開した。

LAMAX-E の公開後、スーパーコンピュータやエンジニアリングワークステーション、高性能パーソナルコンピュータの登場など、コンピュータの利用環境が大きく変化した。新しいコンピューティング環境に

合わせたLAMAXの設計が1987年から開始され[56]、筆者は1989年に**LAMAX-S**(Sは Supercomputerの略)と呼ばれる行列演算用言語の文法を設計した。

筆者の設計したLAMAX-Sの文法が実際に有効であることを示すために、筆者と(株)システム計画研究所(本社、東京都渋谷区)はLAMAX-Sの処理系のプロトタイプの開発を1989年秋に始め、約2年後の1991年秋に完成した。この処理系はLAMAX-Sの文法から、スパース行列とその演算機能を除き、また、処理系の機能として最適化機能を除いたものでNECのパーソナルコンピュータPC9801シリーズ上で動作するようになっていた。このため、**プロトタイプPC98版LAMAX-S処理系**と呼ばれた。LAMAX-S処理系もまた、FORTRAN77プログラムを生成するプリプロセッサとして実現されている。実行時の行列演算のライブラリとしてLINPACKを前提にしている。この処理系を用いて、さまざまな分野の数値計算のLAMAX-Sのプログラムを試作し、LAMAX-Sの記述性の実験を行った。対象分野としては、固有値問題、連立一次方程式、非線形連立方程式、重回帰分析、因子分析、線形計画法、非線形最適化問題、計量経済などであり、数値計算の多くの分野を包含している。この結果、LAMAX-Sの文法の数カ所の改善点が指摘され、筆者と井上美明(システム計画研究所)と共同でLAMAX-Sの文法の一部を改善した。なお、このプロトタイプ版処理系(PC98版)は、1992年春に、第7回日本オペレーションズ・リサーチ学会事例研究奨励賞(ソフトウェア部門)を授賞した[64]。その後、3回のバージョンアップおよびEWSへの移植などを行った。

信頼性の高められたLAMAX-Sプロトタイプ処理系(筆者らは、この処理系をLAMAX-S (Ver. 1.5)と呼んでいる)が、**LAMAX-I**(Iは Intermediateの略)という製品名で1994年12月に同社より発売が発表され、1995年4月より出荷が開始された。

本論文は、1970年代後半から始めた、一連の行列演算用言語の著者の研究の成果をまとめたものである。

1.1 本研究の概要と本論文の構成

本研究の目的は、次の2点に集約することができる。

- 実用的なレベルの行列演算用言語の文法の設計を行うこと。
- その処理系を作成するための実現技術を開発すること。

本研究の概要を示す前に、まず、言語文法の設計の困難さについて簡単に言及する。ミニLAMAXを実際に利用者に提供した際に多くの不具合が指摘されたが、その中のいくつかはまったく筆者らの予想外のものであった。一般に、新しいプログラミング言語を演繹的に設計することは、非常に困難である。

本研究では、ミニLAMAX・LAMAX-E・LAMAX-Sと3つの互換性を無視した言語文法を設計した。さらに、ミニLAMAXとLAMAX-Eに関しては、実際に動作する処理系を作成し、一般利用者に公開してきた。すなわち、ミニLAMAX、LAMAX-Eは筆者らの研究グループ外の実際のユーザを得て、多くの運用経験を積むことができた。LAMAX-Sに関しては、一部の機能を除いたプロトタイプ版ではあるが、実際に動作する処理系を作成した。このプロトタイプ処理系のユーザは筆者の関係者が中心となっているが、例題を数多く解くことによってある程度の運用経験を積んだ。このような背景から、数値計算のプログラミング言語の設計に関してある程度の指針を得ることができた。本論文では、まず、第2章で、LAMAXの開発の歴史を通して、この点について論じる。

ところで、たとえ立派なプログラミング言語の文法を設計しても、その処理系を実現する技術がなければ、工学としての価値を失う。本研究では、LAMAX-Sの実現技術について次の観点から研究を進めた。

1. 一般のコンパイラで行われている最適化に加えて、数学的な特性を利用してプログラムの計算量を低減化する最適化とその実現法に関する提案。
2. さまざまなアーキテクチャを持つコンピュータに対し、プログラムのチューニングを対象ハードウェアに合わせて自動的に行う機構の開発。

3. 行列演算のための効率的な実行時ルーチンの管理法の開発。

第 1 に示した最適化を、本研究では計算量低減化と呼んでいる。第 2 に示した機能を、本研究では自動チューニング機能と呼んでいる。第 3 に示した手法では、行列フォルダモデルと行列構造階層モデルと呼ばれる行列データの管理法を提案している。

本研究の流れとその研究対象の範囲について、図 1.1 に示す。

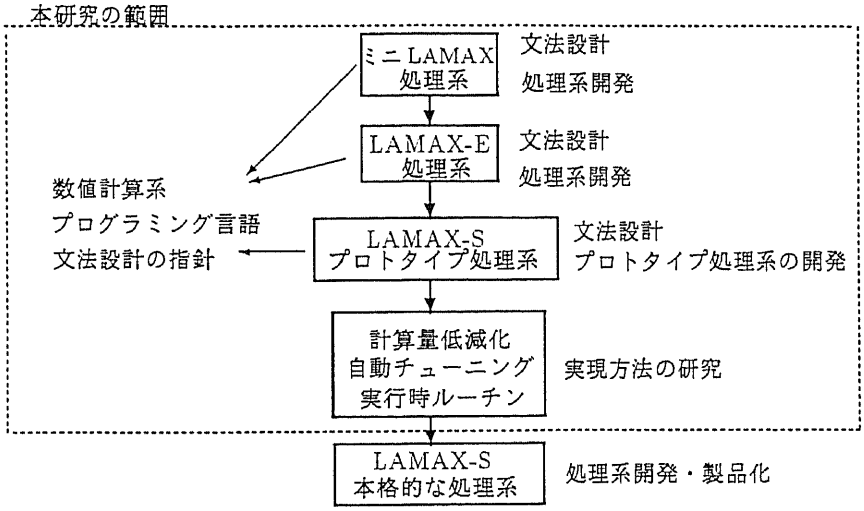


図 1.1: 本研究の流れとその研究対象の範囲

本研究では、図 1.1 に示すように、LAMAX-S の文法の設計、およびそのプロトタイプ処理系の開発による LAMAX-S 文法の検証、そして本格的な LAMAX-S(筆者らはこれを LAMAX-S (Ver. 2.0)、あるいはフルセット LAMAX-S と呼んでいる) を開発するための実現技術の研究がその対象となる。本研究で示された実現技術を適用しスーパーコンピュータ用にチューニングされた LAMAX-S を開発する作業が、システム計画研究所と筆者らによって行われている。

具体的な本論文の構成について示す。本章の次節 (第 1.2 節) で、LAMAX の研究に関連する他の研究や実在するプログラミング言語について概観し、LAMAX の研究と他の研究との立場の違いを明らかにする。

第 2 章では、LAMAX 開発の歴史について述べ、LAMAX-S 研究の背景を明らかにする。第 3 章では、設計された LAMAX-S の文法について述べる。計算量低減化については第 4 章で、自動チューニング機能については第 5 章で、行列フォルダモデルと行列構造階層モデルについては第 6 章で、それぞれ論じる。第 7 章で、LAMAX-S のプロトタイプ処理系を用いた応用と評価について述べる。第 8 章で、本研究を総括する。

1.2 数値計算のためのプログラミング言語とその発展

本節では、数値計算を主目的とする既存のプログラミング言語の発展形態を LAMAX-S と比較しながら説明し、LAMAX-S の立場を明らかにする。

1.2.1 数値計算用プログラミング言語 FORTRAN とその後の発展

多くの人が常用的に用いるようになった数値計算用の最初のプログラミング言語は、1950 年代中ごろに開発された FORTRAN[5] であることは誰もが認めるところであろう。FORTRAN の目標は、数式の記述と効率のよい機械語の生成であったが、十分にその目標を達成した。この考え方は、プログラムの最適化技

術としてその後のコンパイラの設計に大きな影響を与えた。FORTRAN の成功に刺激され、実に多くのプログラミング言語が登場し消えていった。その最も象徴的なものは、ALGOL[35] である。ALGOL は実に洗練されたプログラミング言語であったが、メーカのスポンサシップがなかったなどの理由により、しだいに使われなくなっていった。FORTRAN は、その後、FORTRAN66[1]、FORTRAN77[9]、Fortran90[34] と発展し、現在でも数値計算の主要な常用プログラミング言語として利用されている。FORTRAN では、行列演算を直接記述することはできないが、行列演算を支援する機能がいくつか用意されている。たとえば、任意の寸法の配列をサブルーチンに引き渡すことのできる整合配列は古くから FORTRAN に取り入れられていた。この機能のおかげで、任意の寸法の行列のサブルーチンライブラリを構築することが可能となっていた。**Fortran90** では、スーパーコンピュータへの対応が計られ、行列の乗算などの組込み関数が用意されたが、行列そのものの概念は組み込まれていない。近年では、FORTRAN の代わりに、C[29] を用いるケースも多くなっている。これは、UNIX オペレーティングシステムの普及によるものであると考えられる。この言語はもともと数値計算を目的として設計された言語ではなく、UNIX を記述するために設計されたシステム記述言語である。C で行列を表現するには 2 次元配列を用いるが、この扱いが FORTRAN に比べて多少やっかいである。しかし、数値計算の記述能力としては C よりも Fortran90 の方が強力であると筆者は考えているが、UNIX に C コンパイラが付属することが多いことや、C コンパイラの価格が安いこと、グラフィックスルーチンやシステムコール呼び出しなどが C では記述しやすいこと、などから今後も C が数値計算用として使われることが多いであろう。もちろん、Fortran90 からは、複数のデータを 1 つにカプセル化するための文法機能である構造体が導入されているので、グラフィックスルーチンやシステムコール呼び出しの相性は、FORTRAN77 よりも格段に向上している。したがって、Fortran90 がこの分野で今後も使われていく可能性も十分にある。また、FORTRAN の強みとして、今までに作成されている多くのライブラリが FORTRAN で記述されており、これが 1 つの大きな文化資産と呼べるほどになっているという事実があることである。

ところで、C にオブジェクト指向の機能を導入した C++[10] では、加算 (+) や乗算 (*) などの演算子のオーバーローディング (多重定義) ができる。この機能を用いることにより、自分で行列演算の機能やその他のいろいろな機能を組み込むことができる。たとえば、平山はこの機能を利用して、漸近展開級数の演算が可能なシステムを開発している [12]。本研究でも、第 6 章で論じているモデルを実現するために C++を用い、この中でこのオーバーローディングの機能を利用している。

1.2.2 プログラミング言語の文法上の設計

プログラミング言語を設計する際には、さまざまな注意事項が存在する。数値計算用プログラミング言語であってもこれは例外ではなく、「数多くの失敗事例から多くの教訓を得て改良を続ける」という作業が根気よく行われている。また、プログラミング言語の文法を規定する際には、記述性や直交性、無矛盾性などが要求され、さらにプログラマのヒューマンエラーに対する配慮も必要になる。

本項では、FORTRAN を例にとって、プログラミング言語の文法がその記述性やヒューマンエラーに与える問題点とそれに対する FORTRAN の改良の経緯 (FORTRAN66 から FORTRAN77) について述べる。たとえば、次の数式をプログラミングする事を考えてみる。

$$S = \sum_{k=1}^{n} x_k$$

これを FORTRAN でプログラム化すると次のようになる。ただし、変数 N, K を整数型、変数 S、配列 X を倍精度実数型とする。

```
      S = 0.0D0
      DO 10 K=1, N
          S = S + X(K)
   10 CONTINUE
```

一般には、N=0 のとき、数式から判断すれば、$S = \sum_{k=1}^{0} x_k$ となり、S の値は 0 になるはずである。ところが、FORTRAN66 の DO 文では、最初のループを無条件に実行してしまうという規則であった。したがって、上記のプログラムを FORTRAN66 のコンパイラでコンパイルして実行すると、S の値は X(1) の値となってしまう。これでは、数学的意味付けと異なるということで、FORTRAN77 では、この場合、N が 0 であるならば、このループを一回も実行しないという規則に変更された。つまり、上記のプログラムを FORTRAN77 のコンパイラでコンパイルして実行すると、S の値は 0 となる。これは、プログラミング言語の表現と実際の意味がより数学的になったといえる例である。しかし、これは、FORTRAN66 から FORTRAN77 へプログラムを移植する際の大きな障害の一つとなった。DO 文のループを「無条件に 1 度実行する」ことを前提に作成されたプログラムは、その部分を書き直さなければならなかったのである[†1]。プログラミング言語の文法の変更は、どうしてもこのようなペナルティを伴う。

文法の構文上の問題点を FORTRAN の DO 文を再び例にとって説明する。「1 から 10 までの和を求める」次のプログラム例について考える。ただし、S、K を整数型の変数とする。

```
      S = 0
      DO 10 K=1, 10
          S = S + K
   10 CONTINUE
```

ここで、2 行目の DO 文のコンマを間違えてピリオドにしてしまったとする。

```
      S = 0.0D0
      DO 10 K=1. 10
          S = S + K
   10 CONTINUE
```

FORTRAN には、「プログラム中の空白を無視する」という規則があるため、2 行目の文は、次のように解釈される。

```
      DO10K = 1.10
```

つまり、「1.10 という数値を DO10K という変数に代入する」代入文になってしまう。つまり、このような単純な記述ミスが正しく指摘されることなく、まったく別の意味に解釈されるという危険な設計になっている。実際に、FORTRAN で作成されたプログラムの中にこの原因によるミスがあり、無人火星探査宇宙船が失敗してしまったという例が報告されている [25](p. 71)。このため、FORTRAN77 からは、文番号のあとにコンマを入れてもよいように文法が変更された[†2]。

```
      DO 10,  K=1, 10
```

このようにすれば、たとえ間違えてコンマをピリオドにしても代入文とは見なされず、コンパイルエラーメッセージを出力することができる。

プログラミング言語の文法は、このように人間のささいなミスからプログラムの意味が別のものになってしまうような形式であってはならない。また、同一の記述が、異なる複数の意味に解釈できるような曖昧なものであってもならない。さらに、プログラミング言語の文法の個々の規則は、直交していなければならない。すなわち、まったく異なる 2 つの文法規則を組み合わせると、元の意味とは異なった解釈ができてしまってはならないということである。しかし、この直交性を初めから保ってプログラミング言語を設計することはなかなか難しいことである。

[†1] 実際には、コンパイラオプションを指定することにより、FORTRAN66 の形式で DO 文をコンパイルできるようになっていたコンパイラが多かったので、実務的に直ちに困るということはなかった。

[†2] 実際には、ここで示した文法変更を行わなくても、この種のエラーは現在のコンパイラによって発見可能である。現在のコンパイラでは、この場合には、「DO10K という変数に値 1.10 を代入しているが、その後使用されていない」という警告メッセージを出力する。

1.2.3 ライブラリの発展

FORTRAN や C のような常用言語には、直接的に行列を扱う機能がないが、この点を補完するものとして、多くのサブルーチンライブラリが開発されている。基本的な線形計算を行うライブラリとして BLAS[33] が有名である。行列計算の代表的な例として、行列の三角分解や連立一次方程式を解く LINPACK[8] や行列の固有値問題を解く EISPACK[42] がある。最近では、この 2 つを統合し、さらにスーパーコンピュータやパラレルコンピュータ用にチューニングされた LAPACK[2] がある。日本国内では、二宮による NUMPAC[31] が広く使われている。これらのライブラリは、さまざまなコンピュータの上で汎用的に動作するように作成されている。一方、特定のコンピュータに対して固有に開発されたライブラリもある。代表的なものとして、日立製のコンピュータで動作する MATRIX/HAP[13]、NEC 製のコンピュータで動作する ASL[44] などがある。これらは、特に自社のスーパーコンピュータ上で念入りにチューニングされ、効果を発揮している。また、これらの自社開発ライブラリは、外部には公開されていない自社製品のノウハウを組み込むことができるので、そのメーカのコンピュータに限定して用いるのであれば、かなりの効率を期待することができる。

しかし、ライブラリの利用にはいくつかの欠点もある。利用者が、ライブラリの各サブルーチンに与えるパラメタの意味や記述位置を正確に指定しなければならないので、プログラムの作成に時間がかかり、かつ作成されたプログラムの読解性が悪くなる。メーカ固有のライブラリを利用した場合には他社のコンピュータに対して移植性がなくなる。しかしながら、このような不便があっても、利用者側に立って考えると、ライブラリの利用では、記述性や読解性が多少損なわれても、そのライブラリの数学ソフトウェアとしての信頼性が最も重要なのであり、実際には記述性や読解性は軽視される傾向があった。しかし、ライブラリの記述性や読解性の低さは、サブルーチン呼び出しという形態でライブラリを利用している以上、やむを得ない側面もある。

実際に、ライブラリの信頼性を確認する作業は困難を極める。メーカ作成のライブラリであっても、机上のテストではバグを完全に取り去ることは難しい。一般にライブラリを一般公開しても最初のうちは現実問題に初めて直面することによって発見されるバグが多く、なかなか安心して使えない。一般利用者による長期の使い込みがライブラリの信頼度を高める。この点では、LINPACK や EISPACK は、実に多くのチェックサイトが存在しているので、信頼性が非常に高い。

1.2.4 行列演算用言語と LAMAX

しかし、その一方で、行列を扱うことが可能な専門家向け言語も、いままでに数多く作成され利用されてきた。本項では、このような行列演算用言語と LAMAX-S の大きな違いについて述べる。行列を扱うことのできる代表的なプログラミング言語として、APL[28] や SPEAKEASY などがある。また、統計演算のアプリケーションとして SAS の IML[40] や APTECH システムズ社の GAUSS などがある。これらのシステムでは、行列演算をそのままの形で記述できる。たとえば、$B = (X'X)^{-1}X'Y$ という式は SAS の IML では、

```
B = inv(X'*X)*X'*Y
```

と記述する。また、ANSI 規格 (含 JIS 規格) の BASIC にも簡単な行列演算を記述する機能がある。これらの言語と LAMAX-S が大きく異なる点は、次の 3 点に要約できる。

1. 行列の構造 (たとえば、バンド行列、対角行列、上三角行列、下三角行列など) および対称性やスパース性を全面的にプログラミング言語の文法として導入している。

2. 行列に対してそれがもつ数学的な性質 (たとえば、正定値など) を指定する。これらは、LAMAX-S が行列の構造や特性を利用して FORTRAN プログラムを生成する際のヒントにすることができる。

3. 任意のライブラリを利用可能にする。この観点から、LAMAX-S は、「行列の計算を行う式を記述すれば、それに基づいてさまざまなライブラリを呼び出す FORTRAN プログラムを生成する」処理系という点で、各種ライブラリに対する数式のインタフェースであると考えることもできる。

行列の構造を利用して線形計算を効率よく実行するライブラリは、すでにいくつか存在している。たとえば、前述した LINPACK をはじめとする三角分解や連立一次方程式の解法のための行列演算パッケージでは、バンド行列や 3 重対角行列など基本的な構造が導入されている。LAMAX-S では、これらのライブラリを積極的に行列演算時の実行時ルーチンとして取り入れることができるようにしている。ライブラリ中の多くのサブルーチンの中から適切なものを選択するために、可能な限り多くの数学的情報が必要となる。たとえば、$Ax = b$ という連立一次方程式を解くという記述に対して A が正定値対称行列であることがわかっていれば、「係数が正定値対称行列である連立一次方程式を解くサブルーチン」を呼ぶことができる。LAMAX-S では、そのための記述を言語文法として導入している。

LAMAX-S は、数学的な記述をベースにしてプログラムを作るという考えを用いているが、これに類似した言語に DEQSOL(Differential Equation Solver Language)[14] がある。DEQSOL は問題対象を偏微分方程式の数値シミュレーションに限定して特にスーパーコンピュータの能力をフルに引き出すことをその狙いとしている。これに対し、LAMAX-S は、数値計算における汎用的なプログラミング言語を目指し、動作環境も基本的にはスーパーコンピュータをその主要環境とするがパソコン上での利用もその対象としている。

また、MATLAB は、行列演算を行うライブラリとして LINPACK や EISPACK を呼び出す代表的な行列演算用言語である。MATLAB では行列データの性質や式の形を利用して、LAMAX-S と同じように「最も良い」と思われる LINPACK や EISPACK のルーチンを呼ぶ。たとえば、MATLAB では、$x = A^{-1}b$ を `x=A\b` と記述する[†3]。`A` の寸法が $n \times n$ で、`b` が n 要素のベクトルの場合、ガウス消去法による $Ax = b$ の解が求められ、`A` の寸法が $m \times n (m > n)$ で、`b` が m 要素のベクトルの場合、QR 分解によって解が求められる。このような考え方は LAMAX-S に近いが、LAMAX-S と異なる点として、LAMAX-S では、このような考え方をさらに押し進め、第 4 章で述べるように計算量低減化処理として扱い、より積極的にプログラム中の式を変形し実際の計算を行うということが挙げられる。また、その際には、数学的知識を処理系内部に組み込むのではなく、データベース化し、処理系の外側に置く。さらに、LAMAX-S では、ライブラリを LINPACK などに固定せず、任意のライブラリを呼び出せるようにするという点が MATLAB のような言語とは異なる。

数学的な概念を取り入れたシステムとして代表的なものに、数式処理システムがある。数値計算では、数値データを対象としているのに対し、数式処理システムでは、直接数式を扱う。しかし、よく知られているように、一般に 5 次以上の方程式の場合、その代数解を直接得ることができないので、行列計算そのものすべてを数式処理システムで計算しようとすると無理が生じる。また、数式処理では、主に純粋に数学的な知識を用いて問題解決を計るが、筆者らの目指すシステムでは、数値計算上のさまざまな工夫 (主に計算量の低減化) を導入して問題解決を計ることを目標としている。つまり、数式処理システムでは、代数解 (場合によっては数値解) を得ることを目標としている。一方、LAMAX-S 処理系は、与えられた問題を数値的に解くための極力計算量の少ないアルゴリズムを得て、それを実行するためのサブルーチン呼び出しを含む FORTRAN プログラムの生成を目指している。

1.2.5 数値計算用言語と最適化コンパイラ

数値計算プログラムは、与えられた計算を可能な限り速く行うことが要求されている。そのため、数値計算用言語のコンパイラの持つ最適化処理技法は、さかんに研究され、体系化されている。本節では、現在の最適化処理技法について概観し、LAMAX-S で主張している最適化処理技術が既存の方法とどこが異なるのかについて述べる。

最適化処理は、図 1.2 に示すように、コンパイラの一部に組み込まれているのが普通である。

[†3] ちなみに LAMAX-S では、`x=A^*b` と記述する。

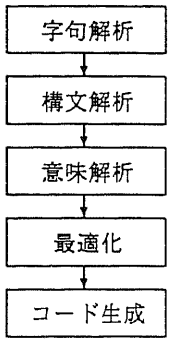


図 1.2: コンパイラの流れ

最初の最適化コンパイラはFORTRANに始まり、その後、COBOL、PL/Iなどにも取り入れられ、現在では、多くのプログラミング言語のコンパイラが最適化機能を導入している。

最適化には多くの手法があるが、その代表的なものを表 1.1 に示す。

表 1.1: 最適化の種類

種　類	説　明
フロー解析	プログラムの流れを調べて、変数に対するデータの代入や参照の状況を調べて、プログラムの最適化を計る。場合によっては、プログラム上の誤りを検出する。
畳込み	プログラム上にある計算式で値が分かるものをコンパイル時に計算しておく。
共通式の削除	プログラム中に同じ式が2度以上現れた場合、最初に計算した値を利用する。
式の評価順序の変更	演算式の順序を変えて、計算回数を減らす。
ループの最適化	ループはプログラムの中でも実行時間のかかる場所である。ここを重点的に最適化すれば、計算時間の短縮に効果的である。ループ内で演算結果が変わらない式をループするたびに計算して場合があれば、その演算式をループの直前に出す。
レジスタ割付け	CPU内のレジスタを効率よく使うような機械語を生成する。
ヒープホール最適化	生成された機械語命令を部分的に最適化する。
代数的性質の利用	プログラム中に現れた計算式をより計算時間が短くなるように変形する。たとえば、x^2を$x*x$に替えたり、2^n倍の計算をシフト演算で置き換える方法などがある。また、$x/2$を$x*0.5$にすることもある。

前述のように、FORTRANやCのほとんどのコンパイラには最適化オプションが備わっている。参考のために、SUNワークステーションでの最適化FORTRANコンパイラの効果を示す。LINPACKを呼び出す4つのFORTRANプログラム(重回帰分析：逆行列を用いる方法、LU分解法を用いる方法、コレスキ分解を用いる方法、QR分解を用いる方法)を作成し、そのメインルーチンとLINPACKサブルーチンを一つのファイルに格納し、最適化オプションを付けてコンパイルした場合の実行時間と付けないでコンパイルした場合の実行時間を表 1.2 に示す。最適化を行うことにより、ほぼ、1/3程度から1/2程度の実行時間に短縮していることが分かる。

また、スーパーコンピュータでは、生成する機械語命令をベクトル化用にチューニングする機能もコンパイラに組み込まれている。ただし、対象とするプログラムによっては完全にベクトル化を自動化すること

表 1.2: 最適化 FORTRAN コンパイラの効果 (実行時間：単位ミリ秒)

手 法	最適化		効 果
	な し	あ り	
逆行列	105.2	34.6	32.9 %
LU 分解	86.4	25.2	29.2 %
コレスキ分解	78.6	24.1	30.7 %
QR 分解	33.7	16.4	48.7 %

ができないため、ベクタイザなどの支援ツールを用いることもある。

パラレルコンピュータに対しても同様のコンパイラが存在する。つまり、FORTRAN プログラム内の並列可能部分を自動的に認識し、そのための機械語命令を生成するコンパイラである。このようなコンパイラの基本的な動作 [24] は、(a) ソースプログラム中のデータ依存・制御依存解析による並列性検出、(b) プロセッサへのタスク割当と実行順序を決定するためのスケジューリング、から構成される。実際のシステムとして、Rice 大学の Fortran D[11]、Purdue 大学の Kali[30]、Vienna 大学の Vienna Fortran[7] などが研究されている。

LAMAX-S 処理系は、プリプロセッサとして実現することを前提にしている。LAMAX-S では、図 1.3 に示すように、数学的性質の記述に基づく最適化処理およびそれに関連するいくつかの処理を行う。それ以外の最適化は、一般の最適化 FORTRAN コンパイラに任せる。LAMAX-S 処理の最適化処理の特徴は、(1) プログラム中の宣言や計算手順に記述された数学的な記述を手がかりにして、プログラムの計算量を低減化するという点、(2) 処理内容をルールベース化して、処理系内部に組み込まない点である。ルールベース化することによって、処理系そのものを修正することなく最適化機能を改善することが可能になるという利点が得られる。

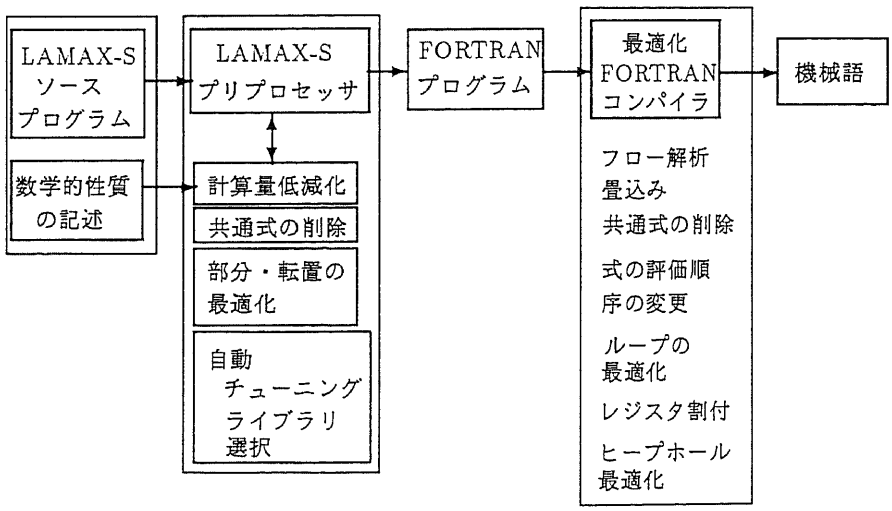


図 1.3: フルセット LAMAX-S 処理系のイメージ

1.2.6 Fortran90 と LAMAX-S

1950 年代に開発された FORTRAN は、1991 年に ISO(国際標準化機構) の国際規格として制定された Fortran90 へと発展した。本項では、Fortran90 と LAMAX-S の関係について述べる。

まず、Fortran90 の登場によって、LAMAX-S の意義が失われるかも知れないという危惧がある。しかし、Fortran90 と LAMAX-S はまったく異なるプログラミング言語である。LAMAX-S は、さまざまな構造の行列 (たとえば、対称行列やバンド行列など) を扱い、さらにさまざまな数学的な最適化機能を前提とし、その処理を容易にするための文法規則になっている。Fortran90 では、よりスーパーコンピュータのための機械語を生成しやすい言語構造にはなっているが、行列演算に関しては、乗算などの機能が組み込まれているにすぎない。

しかし、Fortran90 と LAMAX-S は、対立するものではない。LAMAX-S は FORTRAN を生成するプリプロセッサであるから、当然、Fortran90 も LAMAX-S の出力するプログラミング言語の対象となりえる。Fortran90 には、構造体を定義する機能、C++で用いられている演算子のオーバーローディング機能、スーパーコンピュータを意識した組込み関数などがあり、これらの機能は、むしろ、LAMAX-S の処理系の負担を軽くするものであると考えられる。

現在は、LAMAX-S の生成するプログラムは FORTRAN77 を仮定しているが、将来は、Fortran90 も視野に入れるべきであろう。

第 2 章 行列演算用言語 LAMAX とその開発経緯

本章では、行列演算用言語 LAMAX の開発経緯を述べながら、プログラミング言語文法の設計とその基準について論じる。

LAMAX の実質的な最初の開発開始年は 1979 年である。最初にミニ LAMAX[51] と呼ばれるシステムが、青山学院大学情報科学研究センターの研究の一環として、当時青山学院大学情報科学研究センターの本郷茂助手 (現青山学院大学経済学部助教授) の指導のもとに当時青山学院大学理工学部学生であった筆者によって作成された。ミニ LAMAX は、ミニ LAMAX の文法で書かれたプログラムを FORTRAN プログラムへ変換する、いわゆる、プリコンパイラとして作成された。その後、ミニ LAMAX をベースにして、その拡張版である LAMAX-E システム [52] がやはり情報科学研究センターの研究の一環として作成された。ミニ LAMAX および LAMAX-E は、その開発をすでに終了している。

本章では、LAMAX の一連の発展経緯について説明する。第 3 章で、ここで得られた経験によって設計された LAMAX-S の文法について説明する。

2.1 LAMAX-S 以前の 2 つの LAMAX: ミニ LAMAX と LAMAX-E

2.1.1 LAMAX 開発の動機 (~1978 年)

1970 年代後半、当時の青山学院大学情報科学研究センターの本郷には、計量経済のモデル分析のプログラム作成の協力依頼が山のように殺到していた。このプログラムは、ほとんどが行列の演算処理からなるものであった。このプログラミング作業を軽減するために、本郷は、1978 年、行列演算パッケージを作成した [16]。このパッケージが通常の行列演算パッケージと大きく異なる点は、行列のデータ域をヒープ領域 (動的メモリエリア) として自動的に管理する点である。つまり、行列計算では、作業用の行列の寸法の管理などが面倒な処理であるが、このような管理をこのパッケージが行ってくれるので、プログラミングの効率がかなり向上した。しかし、論文やアルゴリズムに記述された行列の式をそのままの形でコーディングすることはもちろんできない。複雑な数式を単純な二項演算の形に直してサブルーチンコールとして展開しなければならないという面倒な作業が必要となる。

2.1.2 ミニ LAMAX(1979 年~1981 年)

そこで、このライブラリをベースにして、当時大学生であった筆者が行列演算用言語を設計しその処理系を作成することになった。

本郷から出された言語仕様の基本構想は次の 2 点に集約できる。

- 利用者が憶えることが少ない。
- FORTRAN 利用者が違和感なく使える。

この方針のもと、筆者が文法仕様のたたき台を作成し、それにさまざまな文法仕様を付け加えていった。

しかし、最初の 2 年は、なかなか文法仕様が完全にはまとまらず、また、筆者が大学を卒業して直接メンテナンスできなくなっても管理できるように、プリコンパイラそのものをジェネレータで作成しようとした [50] のであるが、そのためのひな型作りが思うように進まなかった。

研究期限切れぎりぎりの 1981 年 2 月頃になると、情報センター内部でも、開発を疑問視する傾向が出始めたので、筆者は最低限の機能を持つ処理系を作成することにした。

そこで、大学の実習で筆者が作成した BASIC インタプリタをベースとして、その上に当初予定していたバージョンの 1/4 程度のシステムを FORTRAN を用いて約 1ヶ月間で作成し、1981 年 3 月に完成した。これがミニ **LAMAX** である。これは、1981 年 4 月から、青山学院大学情報科学研究センターで一般公開された。筆者は、当初、機能の低さなどから、それほど利用されることはないであろうと思っていたが、本郷が積極的に学内に広めたこともあって、数人の利用者が誕生した。この処理系は、1981 年の秋の情報処理学会 [51] で発表した。ミニ LAMAX の概要を示すために、以下にミニ LAMAX による重回帰分析のプログラム例を図 2.1 に示す。

```
 1  *      INPUT X,Y
 2  *      B=1/(X'*X)*X'*Y
 3  *      YE=X*B
 4  *      U=Y-YE
 5  *      FN GET_DIM (@N,@K) <-----: (X)
 6  *      SIGMA2=(U'*U)/(@N-@K)
 7  *      PRINT X,Y,B,YE,U,SIGMA2
 8  *      DW=((U<1:@N-1,1>-U<2:@N,1>)'*(U<1:@N-1,1>-U<2:@N,1>))/(U'*U)
 9  *      PRINT DW
10  *      ALLOCATE E<@N,1> ALL 1
11  *      AVY=Y'*E/@N
12  *      @AVY <= AVY<1,1>
13  *      I<1,1> <= 1
14  *      R2=I-U'*U/((Y-@AVY*E)'*(Y-@AVY*E))
15  *      PRINT R2
16          STOP
17          END
```

図 2.1: ミニ LAMAX プログラム例：重回帰分析

これは、

$$\text{モデル} \qquad y = X\beta + u$$

$$\begin{array}{ll} \mathrm{y} & : \ (\mathrm{n} \times 1) \\ \mathrm{X} & : \ (\mathrm{n} \times \mathrm{k}) \\ \beta & : \ (\mathrm{k} \times 1) \\ \mathrm{u} & : \ (\mathrm{n} \times 1) \end{array}$$

に対して

$$\hat{\beta} = (X'X)^{-1}Xy$$

$$\hat{y} = X\hat{\beta}$$

$$\hat{u} = y - \hat{y}$$

$$\sigma^2 = \frac{\hat{u}'\hat{u}}{n-k}$$

$$d.w. = \frac{\sum_{i=2}^{n}(\hat{u}_i - \hat{u}_{i-1})^2}{\hat{u}'\hat{u}}$$

$$R^2 = 1 - \frac{\hat{u}'\hat{u}}{\sum_{i=1}^{n}(y_i - \hat{y})}$$

を求めるプログラムである[†1]。ミニ LAMAX では、LAMAX の記述 (すなわち、行列演算) をしたい行には、その 1 カラム目に*(アスタリスク) を記述しなければならない。この行を LAMAX 行と呼び、LAMAX 行に記述された FORTRAN の変数、つまり、行列ではないスカラ変数には、すべて@を付けて行列変数と区別する。このようにすることで、ソースプログラム上の変数が、行列かそれ以外のものかの区別が容易になる (はずであった。この点については後述)。図 2.1 の 1 行目は、行列 X および Y のデータ入力を意味している。行列の行数、列数は入力されるデータの最初に指定されている。つまり、このプログラムは、いったんコンパイルしてしまうと、数学的に意味があれば任意の寸法の行列データに対して実行させることができる。これは、前述のように行列演算パッケージが動的に行列データ域の管理を行っているので可能なわけである。逆に述べれば、行列の寸法は、プログラムが実行され、データが入力されないと分からない。そこで、行列の寸法を知る機能が必要である。ミニ LAMAX では、行列に関するさまざまな処理を次の構文で記述する。

```
FN 機能名 (出力パラメタ) <-----: (入力パラメタ)
```

<-----の-はいくつ記述してもよいという規則になっていた。行列 A の行数と列数を取得して、それぞれ、N, I にセットするには、

```
FN GET_DIM (@N,@I) <-----: (A)
```

と記述する。N, I はスカラなので@が先頭に付いている。なお、この構文を筆者は、FN command function と名付けた。数式は、算術式どうりに記述する。'(単一引用符) は転置行列を意味する。A<2:3,4:5>という記述は行列 A の 2 行目から 3 行目、4 列目から 5 列目を取り出した部分行列を意味する。2:2 のときは 2 と省略できる。A の i 行 j 列めの要素は、A<i,j>と記述する。<=は、型変換代入と呼ばれ、スカラと行列の間のデータの変換を伴う代入文に対して用いられる代入演算子である。STOP, END は FORTRAN 固有の命令であるので 1 カラム目に*を付けない。

処理系としてはかなり小さい (記述言語は FORTRAN で約 3,000 ステップ。行列の実際の計算は、参考文献 [16] のライブラリを用いた) が、一通り行列演算プログラムを組むための基本的な事項はすべて用意されている。たとえば、固有値や固有ベクトルなどを計算する FN command function もある[†2]。

2.1.3 ミニ LAMAX から LAMAX-E へ (1981 年)

さて、ミニ LAMAX の数カ月の使用期間を通して、次のような事柄があきらかになった。

1. 「利用者のために」と思って設計した項目が、実は逆に利用者に負担を掛けていた。

2. 処理系の負担を軽減させるための文法規則が、利用者に相当負担を掛けていた。

[†1] $d.w.$はダービン=ワトソン比、Rは重相関係数である。
[†2] 実際の計算は、EISPACK のサブプログラムを用いている。

3. 利用者は、処理系作成が考えもしないような使い方をする。

4. 利用者は、最初は便利であると思ってくれていた機能も、慣れてくると当然であると考えるようになってより高度なことを要求して来る。

1. は、たとえば、次の点である。当初、行列変数と通常の FORTRAN の変数を明示的にソースプログラム上で分かるようにすべきであると筆者は考えた。このようにする事によって利用者もコンパイラも処理負担が軽減される。そこで、ミニ LAMAX の文法記述行では、行列データではない FORTRAN の変数名には @ 記号を付けることにした。たとえば、

```
A = @S * ( B + C)
```

という記述は、行列演算 B + C の結果をスカラ (S) 倍して、A にセットするという意味である。確かにプログラム上で直ちに行列とスカラの違いが把握できて良いのであるが、実際には、利用者は @ を付け忘れることが多かった。ミニ LAMAX のプログラムは、どんなに優れたユーザであっても、一回目のコンパイルで動いたことはめったになかったが、その最大の理由は「@ の付け忘れ」であった。このことから、LAMAX-E では、FORTRAN の変数 (スカラ変数) に @ を付けることをやめることにした。そのかわり、ドキュメンテイタを整備し、たとえば、LAMAX-E では、専用のドキュメンテイタをプリプロセッサに組み込み、印刷されたプログラムリスト上で行列の変数を重ね打ちする事によって目立たせたり、行列変数の下にアンダラインを引くようにした。

2. は、LAMAX 行に「*」を付けるようにした点であった。これも利用者が付け忘れることがわりと多かった。ミニ LAMAX のコーディングに慣れてきたユーザにとって、* を付けるというのは面倒な作業であるだけではなく、ミニ LAMAX の世界と FORTRAN の世界を区別しなくてはならないという精神的負担を強いていた。* の目的は、ミニ LAMAX の処理対象を * の付いた行 (LAMAX 行) に限定し、* のない行 (FORTRAN 行) には、自由に FORTRAN の文を書けるようにするためであった。もちろん、処理系がこれを判断することもできるが、ミニ LAMAX では、処理系単純化のために、人間に書いてもらうようにしていた。LAMAX-E では、この区別は処理系が行うことにした。

3. は、表示用に用意した TITLEPRINT 文に関してであった。これは、次の構文をしていた。

```
TITLEPRINT comment
```

この文の実行により、*comment* の部分をそれが出力された日時と共に表示 (印字) される。筆者は、行列データを出力する際の見出しにこの TITLEPRINT 文を使うことを想定していたが、実際にユーザの間で流行した方法は次のように、プログラムの先頭部分にコメントとして置くことであった。

```
TITLEPRINT ---------------------------------------------
TITLEPRINT     PROGRAM NO. 123     BY C. MORITAKA
TITLEPRINT ---------------------------------------------
```

このようにしておくと、プログラムの実行に先立ち、この部分が日時と共に印字されるので、リストと実行結果の相性がとても良くなる。この使い方は、文法設計者にはまったく思いつかない方法であった。

4. は、たとえば、サブルーチンや関数の利用があげられる。文法設計者 (筆者) は、ミニ LAMAX のプログラムステップはさほど大きくならないであろうと考えていた。なぜなら、ミニ LAMAX の 1 行が FORTRAN の十数行に相当すると考えられたからである。ほとんどのプログラムは、メインプログラムだけで、サブルーチンや関数を用いる必要性はないであろうと考えた。しかし、実際に公開してみると、最初は単純な問題を試していたユーザもだんだん本格的な問題を扱うようになってきて、サブルーチンのもつモジュール性が必要になってきた。つまり、ある特定の手法 (アルゴリズム) を部品化したり、プログラムの見通しをよくするためにモジュール化したりすることが要求された。これも最初の見通しとはかなり異なってしまった。

2.1.4 LAMAX-E の設計と開発 (1981 年秋～1982 年秋)

2.1.4.1 ミニ LAMAX の経験と LAMAX-E の設計方針

ミニ LAMAX の運用経験から導かれた LAMAX-E の設計基本方針は次の通りである。

1) ミニ LAMAX との互換性は取らない。

ミニ LAMAX は、そもそも「動かす」ことだけを前提としたシステムであるので、これに執着する必要はない。

2)FORTRAN の部分と LAMAX の部分の違いを利用者に意識させない。

LAMAX 行か FORTRAN 行かは処理系が認識すればよい。また、@ 記号をスカラ変数の頭に付けるなどというユーザにとって面倒なことはやめる。そのかわり、行列データ型を導入し、宣言文で行列型の変数であることを宣言する。しかし、この宣言は 1 度行えば良いので、利用者にとっては大した負担にはならない。また、行列と他のスカラ変数とのソースプログラム上での区別は、処理系が支援する。つまり、特別な清書システムを作る。また、このようなプリプロセッサ形式のシステムにはよくあることであるが、FORTRAN 部分に関するエラーメッセージは生成された FORTRAN プログラムに対して表示されるので分かりにくい。また、利用者は、LAMAX-E のエラーメッセージと FORTRAN のエラーメッセージを 2 回に分けて調べなければならない。そこで、LAMAX-E では、FORTRAN の簡単な構文チェックも同時に行うことにした。

また、LAMAX-E の文法では、LAMAX-E 記述と FORTRAN 記述との違和感を取り除くことが重要なので、行列型変数の配列や COMMON 変数、行列型を返す関数なども記述できるようにしなければならない。

3) 利用者のニーズの見直しとそれに応えるシステム設計をやり直す。

LAMAX-E の利用者は、おもに統計計算・計量経済モデル分析のためのプログラミングを行う研究者・学生などである。このような分野のプログラムの特徴は次の通りである。

1. 行列計算が主体である。
2. 複素数などの型はあまり用いない。
3. 大次元の密行列を用いるので、実行時間の縮小化が必要となる。
4. プログラムに不慣れな人も利用する。
5. 行列演算上の数学的な常識を取り込む。

この各項目に対して、LAMAX-E は次のような機能を持つように設計された。

1. 豊富な行列演算子を用意する。単に加算や乗算などの演算だけでなく、行列の行和などを求めるのに便利な定数行列を作る演算子なども組み込む。
2. 処理系実現の簡易さから、倍精度実数型の要素を持つ行列のみを扱うようにした。
3. 行列処理に対する最適化機能を導入する。LAMAX-E が備えている最適化機能は、次の通りである。
   (a) 基本ブロック内の共通式の削除
   (b) 転置行列・部分行列の最適化
4. 清書機能やトレース機能など利用者への支援機能を強化する。
5. たとえば、1 行 1 列の行列は、スカラとして扱う。

1 行 1 列の行列をスカラとして扱うことには、実は非常に難しい問題が含まれている。LAMAX-E では、行列の寸法は、実行に決定される。つまり、ある行列が 1 行 1 列かどうかは実行時にならないと決まらない。たとえば、次のような記述があるとする。ここで、A,B,C は行列型の変数である。

```
A = B * C
```

ある場合には、B の寸法が 5 × 4 で、C の寸法が 4 × 6 かも知れない。この場合には、計算可能で、A の寸法は 5 × 6 になる。また、ある場合には、B の寸法が 1 × 1 で、C の寸法が 4 × 6 かも知れない。この場合には、行列の演算としては意味を持たないが、B がスカラと見なされるので、C を B 倍した行列が A にセットされる。このように同じプログラムでも場合によって、実行時に変数の型が行列型になったり、(行列の要素の型である) 倍精度実数型になったりする。つまり、実行時に動的に変数の型が変わるという、いわゆる、レイトバインディングの機構が必要になる。Lisp のように型に関して柔軟な言語システムで処理するのであれば、この実現方法はさほど難しくはないが、FORTRAN 上でこれを実現しなければならない。実際の処理系を作成する立場の筆者は、最初この点に関しては、処理系が複雑になることを恐れて頑強に反対した。しかし、利用者側からみればしごく当たり前の処理である。結局、実行時ルーチン上でこの処理を吸収することにして、導入することにした。

2.1.4.2 LAMAX-E の文法の概要

これらの設計方針から導かれた LAMAX-E の文法の概要は、次の通りである。

MATRIX 文 行列型の変数はすべて型宣言文で宣言する。その際、行列の行と列の寸法は指定しない。行列の行と列の寸法は実行時に決まる。行列型の配列、COMMON 変数、行列型を返す関数も定義できる。

拡張算術式 行列演算を含んだ算術式である。行列の演算は数式通りに記述できる。1 行 1 列の行列はスカラと見なされて実行される。

拡張算術演算子 表 2.1 に一覧表を示す。

行列の要素と部分 この部分はミニ LAMAX と同じであるが、部分行列を表現するための記号 < > の他に [ ] も可能にした。

結合行列 いくつかの行列を結合してより大きな行列を作成する。その際、スカラも 1 行 1 列の行列として扱うので値の定まっている行列を作り出すこともできる。

同値要素行列 行ベクトルの和を求めたりするときに要素の値がすべて 1 の列ベクトルをかけて表現することがある。同値要素行列は、そのような要素の値がすべて同一の行列を簡潔に表現する。

FN Command Function 行列の行列式の値や固有値、固有ベクトル、行列の寸法などを簡単に求める構文である (表 2.2)。

図 2.2 に、LAMAX-E の簡単なサンプルプログラムを示す。ミニ LAMAX と比較すると、FORTRAN プログラムとかなり良く融合していることが分かる。

2.1.4.3 LAMAX-E の設計と開発

LAMAX-E の設計は、おもに 1981 年秋に行われた。LAMAX-E では、処理系の作成に着手する前に完全な文法書を作成し、それを公表 [52] した (1981 年 12 月)。そして、処理系作成中に入門書 [53](1982 年 6 月)、処理系作成後に例題集 [18](1982 年 12 月) を公表した。実際の処理系作成は、1982 年 2 月ごろから行われた。LAMAX-E 処理系は、ミニ LAMAX に比べてプログラムの総ステップ数で 10 倍程度の規模のシステムになることが予想された。当時、汎用大型コンピュータ上での移植性を考えると、問題なく使えるシステム記述言語は、FORTRAN か PL/I ぐらいしかなく、処理系の軽さとコンピュータ間の移植

表 2.1: LAMAX-E の行列算術演算子

優先順位	行列演算子	意　味	形　式	評価の型
1	+	正符号	+S, +M	S, M
1	−	負符号	−S, −M	S, M
2	'	転置行列	M'	M
3	**	べき乗	S**S, M**S	S, M
4	*	乗算	S*S, S*M, M*S, M*M	S, M, M, M
4	&	行列要素間の乗算	M&M	M
4	/	除算	S/S, M/S	S, M
4	/	逆行列	S/M, M/M	M, M
4	%	行列要素間の除算	M%M	M
5	+	加算	S+S, M+M	S, M
5	−	減算	S−S, M−M	S, M

本表において、S はスカラ、M は行列を意味する。

表 2.2: LAMAX-E の FN command function 一覧

FN GET_DIM(*row*, *col*) <---: (matrix)
行列 matrix の行数を *row*、列数を *col* へ代入する。

FN RANK(*rank*) <---: (matrix)
行列 matrix の rank(階数) を *rank* へ代入する。

FN DET(*det*) <---: (matrix)
行列 matrix の行列式の値を *det* に代入する。

FN TRACE(*trace*) <---: (matrix)
行列 matrix のトレース (行列の対角要素の和) を *trace* へ代入する。

FN G_INVERS(ginvers, *rank*) <---: (matrix, *mode*)
行列 matrix の一般逆行列 (ginvers) と rank(*rank*) を求める。
ただし、*mode*=1 のとき、full rank の一般逆行列
mode=2,3 のとき、一般逆行列
mode=4 のとき、Moore-Penrose 逆行列
mode=5 のとき、反復法による Moore-Penrose 逆行列

FN EIGEN (evalue, evector) <---: (matrix)
対称行列 matrix の固有値を evalue へ、固有ベクトルを evector へ代入する (ヤコビ法)。

FN EISP_SYM (evalue, evector) <---: (matrix)
対称行列 matrix の固有値を evalue へ、固有ベクトルを evector へ代入する (EISPACK)。

FN EISP_QR (evalue, evector) <---: (matrix)
対称行列 matrix の固有値を evalue へ、固有ベクトルを evector へ代入する (QR 法)。

FN EISP_AB (evalue, evector) <---: (A, B)
下式の行列 A、B の固有値を evalue へ、固有ベクトルを evector へ代入する (EISPACK)。
$Ax = \lambda Bx. |A - \lambda B| = 0, |B^{-1}A - \lambda I| = 0$

本表において、ゴシックは行列、イタリックはスカラを意味する。

```
      MATRIX X,Y
      MATRIX B
      MATRIX YE,U
C
      MINPUT X,Y
      B = 1 / (X'*X)*X'*Y
      YE = X*B
      U = Y - YE
      FN GET_DIM (N,K) <----------: (X)
      SIGMA2 = (U'*U)/(N-K)
      MPRINT X,Y,B,YE,U
      WRITE(*,*) 'SIGMA2=', SIGMA2
      DW = ( (U[1:N-1,1]-U[2:N,1])'*(U[1:N-1,1]-U[2:N,1]) )/( U'*U )
      MPRINT DW
      ALLOCATE E[N,1] ALL 1
      AVY = Y'*E/N
      WRITE(*,*) 'AVY=',AVY
      STOP
      END
```

図 2.2: LAMAX-E プログラム例：重回帰分析

性を考慮して FORTRAN を選択した。すでに構造化プログラミングを支援する FORTRAN 77 が一般的に利用できるようになっていた。しかし、LAMAX-E プリプロセッサは最適化機能などを含むため、内部アルゴリズムが複雑になることが予想され、見通しの良い言語環境が必要であると思われた。そこで、当時の青山学院大学理工学部経営工学科間野研究室で開発されていたトップダウンコーディングを支援する MDL-FORTRAN 言語 [39] を採用することにした。MDL-FORTRAN も、また、FORTRAN へのプリプロセッサであり、MDL-FORTRAN 処理系自体もブートストラッピングにより MDL-FORTRAN で書かれていた。また、MDL-FORTRAN には、抽象データ型 [32] に近い概念を支援するバンドルサブルーチンという概念があった [27]。これを用いることにより、複雑になりがちなプログラム構造をかなり整理した形で作成することができた。LAMAX-E は、1982 年 9 月に最初のバージョンが完成した。その後、1986 年 9 月の最終リリースまで、十数回のバージョンアップがあった。LAMAX-E 処理系は、FORTRAN 換算で約 27,000 ステップ (コメントを除く) であるが、この十数回のバージョンアップに耐え得ることが可能であったのは、MDL-FORTRAN を記述言語としたことが大きいと考えている。一方、実行時ルーチンとなる行列演算パッケージも、LAMAX-E への拡張とともに全面的な見直しが行われ、まったく新しく作成した [17]。これは本郷が担当し、約 25,500 ステップの FORTRAN プログラムとなった。処理系開発に際しては、当時ようやく実用的になってきた、シンボリックデバッガを大いに活用した。開発は、一貫して、青山学院大学情報科学研究センターに設置されていた NEC 製メインフレーム ACOS 350 上で行われた。

LAMAX-E 処理系は 1982 年 9 月に完成した。その翌月から青学内で一般公開したが、最初はバグが多発し、利用者にかなり迷惑をかけた。約 1 年間学内で試用し信頼性を十分高めた後、他大学大型計算機センター (東京大学・北海道大学) への移植を開始した。各大学大型計算機センターへの移植に際しては、各大学のライブラリ開発の援助を受けた。表 2.3 にミニ LAMAX および LAMAX-E の年表をまとめた。

表 2.3: LAMAX の開発の流れ：ミニ LAMAX から LAMAX-E まで

1978	行列演算パッケージの開発 (本郷)
1979.4	LAMAX プロジェクトの開始
1981.3	ミニ LAMAX 完成
1981.4	ミニ LAMAX 青山学院大学で一般公開
1981.9〜10	LAMAX-E の文法の設計
1981.10	ミニ LAMAX 学会発表 (情報処理学会)
1981.12	LAMAX-E の文法書　発行
1982.2	LAMAX-E 開発開始
1982.9	LAMAX-E 完成 青山学院大学内一般公開
1983.11	東京大学大型計算機センター　一般公開
1984.1	北海道大学大型計算機センター　一般公開
1986.9	LAMAX-E 最終リリース
1987.7	専修大学　一般公開
1987.9	LAMAX-S 構想発表
1988.1	パーソナルコンピュータ (MS-DOS) 上に移植　PC-LAMAX と命名

2.1.4.4 LAMAX-E のもつ最適化機能

2.1.4.1 で示したように、LAMAX-E は、次に示す最適化機能を有している。

1. 基本ブロック内の共通部分式の削除

2. 転置行列・部分行列の最適化

「基本ブロック内の共通部分式の削除」とは、プログラム中の連続する複数の行の計算式において、共通に現れる計算式 (これを共通部分式という) を一度だけ計算するようにプログラムの計算手順を変更する

処理である。あえて、この処理を行う理由は、次の 2 つの事実に基づいている。(1) 一般に行列計算に要する時間は、他のスカラ計算に比べて長い。(2) ミニ LAMAX の経験から、ユーザは行列計算式をモデルそのままの形で記述することが多く、必然的に共通式が多くなる。

「転置行列・部分行列の最適化」とは、転置行列あるいは部分行列を含む計算式において、次の 2 つの事実 (1)、(2) を用いてこれらの処理時間を実質的にゼロにする処理のことである。(1) 加減乗算の行列演算において、転置行列演算子が指定されているときは、計算方向を変えれば良い。たとえば、

```
A = B + C
```

という行列演算では次の計算を行えばよい[†3]が、

```
do 10 j=1, n
   do 10 i=1, m
     A(i,j) = B(i,j) + C(i,j)
10 continue
```

次の例のように、転置行列演算子 (') が指定されている場合には、

```
A = B' + C
```

次のプログラムのように、行列 B の添字をかえれば良い。

```
do 10 j=1, n
   do 10 i=1, m
     A(i,j) = B(j,i) + C(i,j)
10 continue
```

(2) 加減乗算の行列演算において、部分行列が指定されている場合、演算範囲を狭めるだけでよい。たとえば、次のプログラムでは、

```
A = B[5:9,4:7] + C
```

という演算では次の計算を行えばよい。

```
      jj = 4
      do 20 j = 1, n
         ii = 5
         do 10 i = 1, m
            A(i,j) = B(ii,jj) + C(i,j)
            ii = ii + 1
10       continue
         jj = jj + 1
20    continue
```

LAMAX-E では、行列の 3 則演算 (加減乗算) すべてに転置行列の有無、部分行列の指定を伴った仮想的な命令群を作成し、その実行時ライブラリを作成した。

このことによって、LAMAX-E では、転置行列処理、部分行列処理の実行時間は実質的にゼロとなった。

[†3] この do ループで、添字 i が内側のループにあるのは、よく知られているように、行列 A, B, C のメモリ領域を連続にアクセスするためである。

2.1.4.5 LAMAX-E の利用状況

LAMAX-E は、青山学院大学をはじめ、東大大型計算機センターなどを介して一般公開されたので、ユーザ層の完全な把握は困難になった。筆者らが知る限りでは、本郷を中心とする計量経済の研究のためのプログラム作成、武蔵工業大学での卒業研究、などで使われた。東大大型計算機センターでの 1990 年頃の LAMAX-E の利用件数は年間 1,000 件程度である。

2.1.5 LAMAX-E の開発から派生した ADT-RASM86

LAMAX-E プリプロセッサの開発の際に、データ抽象化の概念に近いバンドルサブルーチンという MDL-FORTRAN の機能を利用したことを前述した。この経験をもとに、データ抽象化の研究を進め、アセンブリ言語にデータ抽象化を組み込んだ処理系 **ADT-RASM86** というシステムを作成した [59][57][58]。

LAMAX-E の開発が一段落した頃、筆者は、青山学院大学大学院博士後期課程の 1 年であった。このころ、マイクロコンピュータを搭載したパーソナルコンピュータが普及し始めた。当時のパーソナルコンピュータのプログラミング環境は、実行効率をあまり気にしないのであれば、C が利用できるようになっていた。しかし、当時の低レベルのパーソナルコンピュータの機能をフルに利用するには、アセンブリ言語を用いる必要があった。もちろん、現在でも、コンピュータの機能を十分に利用するには、アセンブリ言語を用いる方が効果的であるが、当時は、メモリ容量や実行速度の点から、アセンブリ言語利用の比重が高かった。

しかし、よく知られているように、アセンブリ言語は、記憶効率や実行効率は良いが、プログラムの生産性や保守性が極度に悪い。そこで、筆者は、LAMAX-E の開発で得た経験から、アセンブリ言語にデータ抽象化を導入することを提案し、それを支援する処理系を作成し、ADT-RASM86 と名付けた。

ADT-RASM86 は、1985 年 1 月にプロトタイプ版の開発が始まり、約 2 人月で同年 3 月上旬に完成した。その後、プロトタイプ版を用いて 3 月上旬から約 1.8 人月をかけて、マルチウィンドウ型スクリーンエディタを試作し、その使用経験を基に再設計し、1985 年 5 月から約 1 人月かけて ADT-RASM86 処理系が完成した。

ADT-RASM86 処理系は、ADT-RASM86 ソースプログラムを RASM86 プログラムに変換する ADT-RASM86 データ抽象化プリプロセッサ、ADT-RASM86 ソースプログラムから仕様書を生成する SPEC、ADT-RASM86 ソースプログラムの清書出力を得る ADT-RASM86 ドキュメンテイタ、から構成される。

ADT-RASM86 の処理系としての性能について述べる。アセンブリ言語と比較すると、実行速度はほぼ同じであり、オブジェクトのサイズはアセンブリ言語のほぼ 3 倍程度の大きさとなる。ADT-RASM86 のプログラム開発期間の圧縮率は、約 1/10〜1/5 程度であるので、この性能は十分であると考えられる。

2.2 LAMAX-S の文法とその決定方針

本節では、LAMAX-S の文法とその設計過程について説明する。また、実際の LAMAX-S の処理系についても説明する。1989 年 6 月に LAMAX-S の実現可能性を探るため、実際のスーパーコンピュータをターゲットとして、LAMAX-S のパイロットバージョンの処理系を作成した。その経験をベースに、LAMAX-S の具体的な文法が決定され、プロトタイプバージョンが作成された。最初の実用的なプロトタイプ処理系 (V.1.1) は、1990 年 10 月に完成し、1994 年夏まで 4 回のバージョンアップを行った。プロトタイプバージョンは、実際にパソコンやワークステーションで動作した。さらに、このプロトタイプバージョンを改良し、LAMAX-I という商品が発売された。本節では、パイロットバージョン開発までの経緯について詳しく述べる。実際の LAMAX-S 文法については、第 3 章で説明する。

2.2.1 LAMAX-E から LAMAX-S へ

前節で述べた事柄を背景として、ミニ LAMAX および LAMAX-E は作成されたので、計量経済向きのプログラム作成には非常に適したシステムになった。しかし、その後の検討および利用者、識者からのアドバイスなどで、次のような点が問題点となってきた。

機能性の側面

1. 行列の要素のデータ型として倍精度実数型しか使えない。整数型や複素数型を要素に持つ行列も必要である。
2. 連立一次方程式を解く演算子が欠如している。
3. 密行列しか扱えない。スパース行列やバンド行列が明示的に扱えない。
4. 対称行列が扱えない。
5. 行列の行や列の操作、たとえば行や列の交換、挿入、抜取りがない。

処理形態の側面

1. バッチ指向であり、会話処理に向いていない。
2. 処理系のサイズが大きすぎる。
3. スーパーコンピュータに向いたコードを生成しない。
4. 行列要素へのアクセスが低速である
5. 行列データ域をヒープ領域として扱っているためのオーバヘッドが大きい。
6. 単価の高いメインフレームだけでなく、個人ベースで使えるワークステーション上でも動作するようにすべきである。

ここで提起された問題点の多くは、LAMAX の利用者を統計・計量経済などの社会科学系から自然科学系へとその層を広げたことによるところが多い。これらの分析から、LAMAX-E と LAMAX-S を対比して見ると、表 2.4 のようになる。

2.2.2 LAMAX-S のパイロットバージョンの作成と評価

まず筆者は、処理系作成上のネックとなる点を予備調査するためと、その年の数値解析シンポジウムで発表するために、前述のように 1989 年 5 月から 6 月にかけて LAMAX-S のパイロットバージョンを作成した [62]。これは、LAMAX-S の簡易プリプロセッサと実際にスーパーコンピュータ上で動作する実行時ルーチンから構成される。この簡易プリプロセッサは、青山学院大学理工学部経営工学科の μ-VAX II の UNIX 4.3BSD 上で動作し、生成された FORTRAN プログラムは、青山学院大学情報科学研究センターの NEC 製 スーパーコンピュータ SX-1EA(約 330MFLOPS) 上で動作する。プリプロセッサは、yacc および lex で作成したので、処理系作成の手間はかなり軽減された。

パイロットバージョンの作成を通して得た経験の中で最も重要なものは、実行時ルーチン用のライブラリについてである。LAMAX-S では、バンド行列や三角行列など多種多様な構造の行列を扱う。その各々すべてに対して、各社のスーパーコンピュータ用にチューニングされたライブラリを LAMAX-S 側で用意することは、実質的には不可能に近い。たとえば、連立一次方程式を解くためにも、LAMAX-S では、表 2.5 に示す係数行列に対する実行時ルーチンが必要になる。

表 2.4: LAMAX-E と LAMAX-S の対比表

項　目	LAMAX-E	LAMAX-S
適用分野	統計・計量経済 一部の OR 社会科学系	統計・計量経済、OR 構造解析、有限要素法 社会科学系、自然科学系
必要な演算機能	行列演算、逆行列 固有値・固有ベクトル	行列計算、逆行列 固有値・固有ベクトル 連立一次方程式 三角分解 (LU 分解、コレスキ分解)
行列に対する操作	三則演算 (加減乗) 部分、転置、要素 結合	三則演算 (加減乗) 部分、転置、要素、対角要素 結合 行・列の交換などの操作
行列データ域	動的割り付け	動的割り付け 静的割り付け
要素の型	倍精度実数型のみ	整数型、単精度整数型 実数型、倍精度実数型、4 倍精度 複素数型、倍精度複素数型、など
行列定数	結合行列で代用	結合行列、単位行列、その他
行列の構造	密行列のみ	密行列、帯行列、バンド行列など
標準組込み関数・サブルーチン	なし。FN command function で代用	豊富な組込みの関数やサブルーチンを用意
適用コンピュータ	大型汎用コンピュータ後にパソコン、ワークステーション	スーパーコンピュータ 大型汎用コンピュータ ワークステーション パソコン

表 2.5: LAMAX-S で必要になる係数行列の例 (一部)

行　列	正方行列	バンド行列	三角行列	三重対角行列
密行列	密正方行列	密バンド行列	密三角行列	密三重対角行列
疎行列	疎正方行列	疎バンド行列	疎三角行列	疎三重対角行列
密対称行列	密対称正方行列	密対称バンド行列	—	密対称三重対角行列
疎対称行列	疎対称正方行列	疎対称バンド行列	—	疎対称三重対角行列

この表 2.5 の各々に対して、各社のスーパーコンピュータ向けのチューニングを施さなければならない。そこで、各メーカが提供しているライブラリを積極的に利用することにした。各社提供のライブラリは、各社の優秀な人材が自社のスーパーコンピュータの特性を良く調べて、最高の性能を引き出せるように工夫してコーディングしているに違いないからである。

パイロットバージョンから得られた考察を基に浮かび上がった LAMAX-S 処理系と LAMAX-E 処理系との比較を表 2.6 にまとめる。

表 2.6: LAMAX-E 処理系と LAMAX-S 処理系の比較

	LAMAX-E	LAMAX-S
プリコンパイラの記述言語	MDL-FORTRAN	C,(yacc)
OS	汎用大型、MS-DOS	汎用大型、UNIX,MS-DOS,OS2
必要なパス回数	1 パス	少なくとも数パス必要
実行時ルーチンの記述言語	FORTRAN	FORTRAN
実行時ルーチン用ライブラリ	自作 LAMAX-E 専用 EISPACK	各社のライブラリ または LINPACK などの フリーソフトウェアライブラリ
単純な行列の演算	実行時ルーチンの呼び出し	DO 文と配列の操作
作業領域	動的割り付け	動的割り付け 静的割り付け
行列データの扱い		行列データをオブジェクトと見なして管理する
混合演算の実現法		行列クラス階層に基づく動的な構造変換

このパイロットバージョンは LAMAX-S の効果と処理系作成上のネックを分析するために作成したものであるが、設計した文法がかなり効果的であることを周りの研究者に知らしめるには十分であった。その際の経験と上述の設計方針に基づいて、次の LAMAX-S 文法を設計した。機能面では、かなりの部分が LAMAX-E に基づいているが、構文上の LAMAX-E との互換性はあまり考慮されていない。しかし、LAMAX-E のプログラムを LAMAX-S に移植するには宣言部と数式の一部を変更すればよく、さほど困難はないと思われる。LAMAX-S の文法上の大きな特徴は、次の点である。

1. LAMAX-S では、行列要素の型は、倍精度実数型だけでなくさまざまな型の要素を持つ行列を扱えるようにした。さらに密行列だけでなく、対称行列やバンド行列、スパース行列のようなさまざまな構造の行列を扱えるようにした。

2. LAMAX-E ではプログラム中に行列変数の記述できない部分 (if 文の中など) があったが、LAMAX-S ではプログラム中意味のある場所であればどこにでも行列変数を記述できるようにした。

3. 基本的には、LAMAX-E にあったすべての機能は、そのまま記述できるようにするが FN command function 文は廃止する。この機能は、行列の組込み関数、組込みサブルーチンで対処する。

4. 行列データの寸法を明示できるようにする。LAMAX-E では、コンパイル時に行列の寸法が完全に分かっていても、行列データ域は実行時に動的に割り付けられていたので、配列で表現されたデータに比べて行列データの扱いはかなりオーバーヘッドを有していた。LAMAX-S では、コンパイル時に行列の寸法が分かっている場合には、静的に割り付ける。もちろん、LAMAX-E では一般的であった動的な寸法を持つ行列も可能にした。

2.3 プロトタイプ LAMAX-S の開発

本節では、LAMAX-S のプロトタイプの開発過程についてその最初の段階から詳しく述べる。なお、LAMAX-S 処理系の開発は、筆者らと (株) システム計画研究所の共同研究の形式をとることにした。筆者ら大学側の研究グループは、主に言語文法の設計や処理系の概要設計を担当し、実際のインプリメンテーションは企業側が行う。そのかわりに、作成したシステムは、商品として同社より発売することになった。

2.3.1 概念形成:1987 年～1989 年 3 月

LAMAX-S の構想を初めて公表したのは、1987 年 9 月の情報処理学会全国大会 [56] であった。LAMAX-S の構想は、その前身である LAMAX-E の当時 5 年以上に渡るリリース経験と多くの研究者の意見を取り入れて考案された。この構想に対するフィージビィティスタディが行われ、内部の基本メカニズムや各社のスーパーコンピュータとの適合性、行列を応用したアプリケーションアルゴリズムとの適合性、などの検討がなされ、その結果は各種学会 (数値解析シンポジウム・東大大型計算機センター研究会・情報処理学会全国大会など) で報告された。1988 年秋には、行列オブジェクトを動的に扱う実行時システムのプロトタイプを TURBO Pascal (Version 4.0) 上に作成し、多様な構造を効率よく処理するメカニズムが確立された [61]。この考え方は、第 6 章で説明する行列オブジェクトへと発展した。

2.3.2 パイロットバージョンの開発:1989 年 4 月～1989 年 7 月

1989 年 4 月から、その年の数値解析シンポジウムの発表に向けて、前述のパイロットバージョンの LAMAX-S を開発することになった。この開発は、前述のように、青山学院大学に設置されていたスーパーコンピュータ NEC SX-1EA 上に、1989 年 5 月～6 月にかけて行われた。行列演算のライブラリは NEC 提供の ASL[44] を用いた。このパイロットバージョンは、かなり有効に動作し、実際の問題をスーパーコンピュータ上で解くことが可能となった。

2.3.3 LAMAX-S の基礎調査:1989 年 8 月～1989 年 10 月

システム計画研究所で LAMAX-S を商品として開発しようという計画が動きだしたのは、1989 年 8 月のことである。具体的に述べると、最初の版は、汎用機のスーパーコンピュータではなくて、(株)NKK が当時開発を進めていたパーソナルスーパーコンピュータ PIAX(ぴあっくす) の上に載せる版として開発することとなった。PIAX は、CPU にインテル 386SX(16MHz)、演算プロセッサにインテル 860(32MHz) を装備し、カタログスペックでは LINPACK を 10MFLOPS 以上で実行することができるパーソナルスーパーコンピュータである。LAMAX-S の概要は、1989 年 10 月のデータショウの NKK のブースで発表され、一般に初めて公開された。

商品化に向けて、LAMAX-S の文法の最終設計が進められていった。プロトタイプシステムの標準の行列ライブラリとして LINPACK を用いることが決められたのもこの時期である。

2.3.4 LAMAX-S の基礎開発:1989 年 11 月～1989 年 12 月

文法事項もほぼまとまり、実際の処理系の核となる部分の開発が開始された。開発言語は、処理系に関しては ANSI 規格の C 言語、実行時処理系は FORTRAN に決定した。開発言語として、C 言語を採用することには、2 つの問題がある。第 1 の問題としては、FORTRAN ほどコンピュータ間の移植性が保証されないことであり、第 2 の問題としては、スーパーコンピュータ上で確実に動作する C 言語コンパイラがないこと (当時)、があげられる。このことは、スーパーコンピュータ上での LAMAX-S のセルフコンパイリングができないことを意味している。最初の問題に対しては、ANSI 規格を遵守し複雑なコーディングを避けることにより対応できよう。C 言語コンパイラに関しては、次の見通しから十分対応できるだろうと判断した。

1. 将来的には、各スーパーコンピュータの OS は基本的には UNIX に移ると思われる。

2. LAN の普及により、コンパイリングはその処理により適したワークステーションで行い、実行はスーパーコンピュータで行うという振り分けが容易になる。

処理系作成は、まず、字句解析部 (内田)・構文解析部 (井上) の開発から開始され、テスト用のサンプルプログラムが多数作成された。構文解析は、バックトラックをなるべくしないような、トップダウンパーサとして作成された。そのため、FORTRAN と LAMAX-S の文法の構文図が作成された。初期の開発は、NEC PC-9801 が用いられたが、1989 年 12 月に SUN が導入されたので、SUN 上でも開発が行われた。

2.3.5 システム計画研究所への開発の移行:1990 年 1 月～1990 年 3 月

1990 年になってからは、開発の中心がシステム計画研究所内に移った。

開発が進展するにつれて文法の不備な点がいくつか発見された。その多くは、同一の表現に対して複数の解釈がなされるものであった。

簡単な例を示そう。たとえば、使用文字の問題があった。当初、連立一次方程式の解を求める演算子として$文字を使用していたが、FORTRAN 処理系によっては$文字は変数名として扱う。そこで、$をやめて? 文字を採用した[†4]。このような文法上のあいまいさは、1980 年 3 月までにはほぼ取れた。1990 年 3 月には LAMAX-S の文法書、入門書が完成した。これらのドキュメントはすべて LaTeX で作成された。処理系開発を完全にシステム計画研究所内に移すために、LAMAX-S の処理系の概要と実行時処理の概要を約 1 週間程度徹底的に討論した。1990 年 3 月には、情報処理学会全国大会 [26][63] で発表し、さらに池袋のサンシャインで行われたスーパーコンピュータショウの NKK のブースでも発表した。1990 年 3 月末までには、構文解析部がかなり動作するようになり、いわゆるシンタックスチェッカとしての動作が可能となった。

2.3.6 プロトタイプ処理系の本格的な開発:1990 年 4 月～1990 年 8 月

本格的な開発作業は、この時期に集中して行われた。実機となる PIAX も導入され、すぐにテストができるようになった。まず、最初の版では、LAMAX-S の仕様から、スパース行列の扱いを除くことにした。さて、LAMAX-S の処理系設計のポイントは、次のような点であった。LAMAX-S では、多種多様な行列のそれぞれの構造に対して複雑な操作が多数用意されているので、そのクロスした結果の操作が膨大なものになってしまう。たとえば、構造だけでも、矩形・正方・バンド・対角・三重対角・上三角・下三角・行ベクトル・列ベクトルがあり、これらはすべてメモリ上でも要素の展開位置が異なっている。さらに、これらの一部に対して、対称性が指定できるし、また、交代行列やエルミート行列の指定もできる。これらの指定の有無によってメモリ上での要素の位置はまったく変わってくる。たとえば、これらの行列構造に対してその部分行列を取ることを考えてみよう。行列の部分として指定された要素の位置はその構造によっても異なるし、また、要素として抜き取られた部分の構造がどの構造になるかも大切な要因である。プロトタイプ版では、部分の抜き取りは、密行列のみに限定した。

この作業を通じて、当初作成された文法書に記載されていない事項、あるいは記載されていても不明確であったりあいまいであった事項がかなり指摘された。また、LINPACK を主要ライブラリとして採用したことで、行列のデータ構造を LINPACK に合わせる必要があった。そのため、たとえば、交代行列などは、データとしては (本来持つ必要はないが、LINPACK にそれに対応する構造がないので) 対称部分も実際にはメモリ領域を確保し、アクセスされた時点で対称部分の処理をするという方式を採用した。

[†4] その後の検討で、この演算子そのものが廃止された。LAMAX-S (Ver. 1.5) では、solve 文で処理する。

2.3.7 公開に向けて:1990 年 9 月～12 月

正式なリリースに先立つプレリリース版が 1990 年 9 月 20 日にリリースされた。最初のリリース版 (Version 1.0) が、10 月の終わりに出された。最初の開発を始めてから、約 1 年後であった。これは、NKK の PIAX 上で動作する LAMAX-S である。PIAX は、OS として MS-DOS を採用しており、LAMAX-S は MS-FORTRAN 用の FORTRAN ソースプログラムを生成する。実際のベクトル計算は、PIAX に内蔵された i860 によって行われ、LAMAX-S もこれを用いるようなコードを生成する。この版は、LAMAX-S の仕様からスパース行列を除いてある。MS-FORTRAN 版の欠点は、セグメントの制限のために大きな行列を使えないという点にあった。12 月 14 日には、Version 1.01 がリリースされ、かなりバグも少なくなった。しかし、安定した動作をするようになったのは、1991 年 1 月 22 日にリリースした Version 1.1 であろう。これは、かなり信頼性の高い処理系であると同時に、この版は NDP-FORTRAN に対応している。これによってメモリが実装されている限り、その大きさで配列を確保できるので、大きな行列を扱うことが出来るようになった。Version 1.1 の処理系は、プリプロセッサ部 (C 言語) で 9 万行 (ソース約 2.4M バイト)、実行時処理部 (FORTRAN) で約 15 万行である。

2.3.8 プロトタイプ開発後の流れ：1991 年以降

本項では、その後の LAMAX-S プロトタイプシステムの発展について説明する。

2.3.8.1 OR 学会賞の授賞

1991 年秋、PIAX 版の処理系を NEC のパーソナルコンピュータ PC-9801 上に移植した。この版を、プロトタイプ PC98 版 LAMAX-S 処理系と命名し、日本オペレーションズ・リサーチ学会の事例研究奨励賞 (ソフトウェア部門) に応募したところ、1992 年 4 月この賞を頂くことができた。この賞は、OR の教育・実務において広く OR 学会会員に役立つもので、教育用ソフトウェア (大学・企業・その他教育機関における OR 教育研修用のソフトウェア)、または OR 技術の先端的ソフトウェアに対して授与される賞である。

また、1992 年 9 月に日本オペレーションズ・リサーチ秋季研究大会でこの賞の授賞による招待講演を行った [64]。

2.3.8.2 文法事項の再検討

プロトタイプ PC98 版 LAMAX-S 処理系の完成により、LAMAX-S サンプルプログラムを手元のコンピュータで動作させることができるようになった。このため、実際に、LAMAX-S プログラムの記述実験が行えるようになった。実際に使用したところ、文法事項の改善すべき点が数カ所見つかり、システム計画研究所の井上美明と筆者で修正を加えた。このため、新しい文法でも処理できるように、新文法を旧文法に変換するコンバータを筆者が作成した (図 2.3 参照)。このコンバータを用いることで、膨大なステップ数の LAMAX-S 処理系に手を加えることなく、見かけ上、新文法でプログラムを記述することができるようになった。

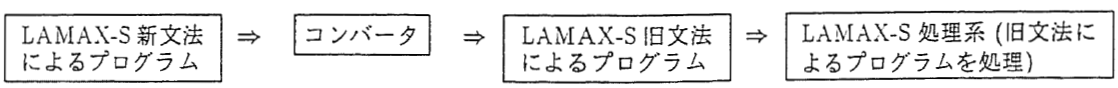


図 2.3: コンバータの役割

ちなみに、プロトタイプ LAMAX-S 処理系 Ver.1.5(商品名：LAMAX-I) からは、この新しい文法に完全に対応している。文法事項の改善点などの詳細に関しては、第 3.8 節で詳しく述べる。

2.3.8.3 EWS への移植

プロトタイプ PC98 版 LAMAX-S は、あくまでも MS-DOS を用いたパソコン上で動作する版であった。筆者の属する研究室では、プログラム開発はすべて EWS で行う方針になっていたので、このプロトタイプ処理系では卒業研究が行えない。そこで、1993 年 3 月から 4 月にかけて、このプロトタイプ版を EWS (SUN SPARCstation) 上に移植することになった。移植作業でネックとなったのは、C で組まれたアドレス計算の部分であった。このプロトタイプ処理系は、メインメモリのアドレス境界を気にしなくてもよいパソコンで作成されていたため、アドレス計算の結果がメモリ中の語境界をまたぐことがあり、このアドレス計算に関する手直しをいくつか行った。また、SUN の提供する C コンパイラにバグがあり、gcc を用いてコンパイルを行った。現在 (1995 年 5 月) では、LAMAX-S (Ver. 1.5) は、SUN および HP の EWS 上、MS-FORTRAN (MS-DOS, Windows NT) 上で動作している。

2.3.8.4 信頼性の向上

1993 年 5 月頃から、EWS 上の LAMAX-S が利用可能となり、筆者の所属する神奈川大学工学部経営工学科の卒業研究に本格的に用いることになった[†5]。しかし、処理系の詳細を知らない学生が作成したプログラムは、LAMAX-S のバグをチェックするには、大いに役に立った。なぜならば、LAMAX-S の処理系の詳細を知っている我々がプログラムを作成しても、無意識のうちに処理系のネックとなりそうな点をエスケープしてしまうのである。数十件のバグレポートをシステム計画研究所に送り、約 1 年後の 1994 年夏ごろにはほぼ問題なく利用できるようになった。

2.3.8.5 商品版 LAMAX-I の発売

(株) システム計画研究所より、1995 年 4 月にプロトタイプ版が LAMAX-I として発売された。このプロトタイプ版の構成は次の通りである。これは、パソコンの MS-DOS 上あるいは EWS 上で動作し、次のような制限がある。

- サポートしているライブラリは LINPACK のみである。
- スパース行列はまだサポートしていない。
- 最適化処理は行っていない。

LAMAX-S プリプロセッサのプロトタイプ処理系の概要を図 2.4 に示す。LAMAX-S プリコンパイラは、行列演算式をそれに対応する FORTRAN プログラムに変換する。共通部分式の削除や式を変形して逆行列を連立一次方程式に変換するなどの最適化処理は行っていないが、行列データの構造に関する情報や数学的特性の指定を利用して、その特性にあわせた LINPACK のライブラリを呼ぶようにして、より効率のよいプログラムを生成できるようにしている。

[†5] 実は、1992 年度から LAMAX-S を用いた卒業研究は行っていた。しかし、パソコン上で動作する LAMAX-S であったので開発効率が悪く、さらにかなりバグもあり、本格的には利用できなかった。

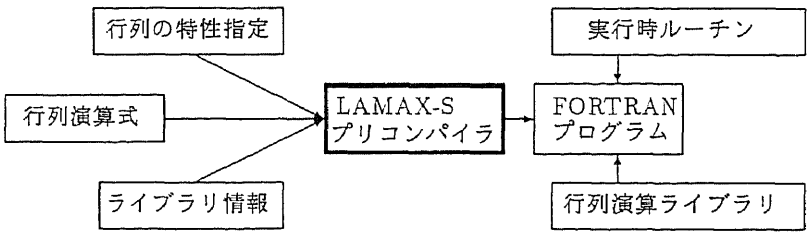


図 2.4: LAMAX-S 処理系の概要図

第 3 章 LAMAX-S の文法とその設計

3.1 はじめに

LAMAX-S は基本的には FORTRAN に行列演算機能を付加した言語である。このため、行列演算以外の機能は、すべて FORTRAN を利用すればよく、LAMAX-S の設計は行列演算機能に限定することができる。LAMAX-S を設計する際に筆者が掲げた目標は、3 つある。

まず、第 1 の目標は、FORTRAN との親和性を計ることである。LAMAX-S の文法は、FORTRAN の自然な拡張であるように配慮した。この理由は、ほとんどの人が知っている共通プログラミング言語である FORTRAN の利用者であれば、それほど苦労せずに LAMAX-S を使いこなせるようにするためである。そこで、LAMAX-S の文法では、数式の書き方、行列変数の宣言の仕方、サブルーチンや関数内での行列の扱いなど、極力 FORTRAN との融合性を持たせた。

第 2 の目標は、数学的な専門知識を持つ人が効率のよいプログラムを書けるようにすることである。つまり、解くべき問題の数学的特性をプログラム上に記述すれば、その特性を LAMAX-S コンパイラが利用してより効率のよいプログラムを生成できるようにする。プログラムを書く人は、行列変数の構造やその数学的特性を指定し、行列の算術式を記述するだけでよい。この数学的特性とは、行列の構造・対称性・スパース性、その行列の数学的性質 (正定値性、エルミート対称性など) などであり、これらを利用するライブラリを選択する際のヒントにする。行列の算術式のパターンからは、その演算結果の特性が類推できる。たとえば、X'を行列 X の転置行列であるとすると、$X'X$の演算結果は対称行列 (さらに Xがフルランクであれば正定値対称行列) であることがわかる。また、$x = A^{-1}b$ は連立一次方程式 $Ax = b$ の求解であることがわかる。このように与えられた数学的特性と行列の算術式から「よいプログラム」を生成するのはあくまでも LAMAX-S の仕事である。そのために、LAMAX-S ではプリコンパイラに極力多くの数学的な特性の情報を与えられるような文法構造にしなければならない。

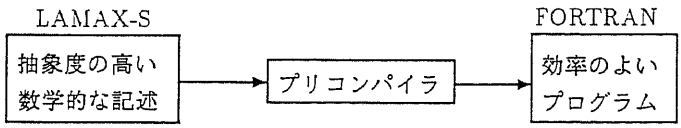


図 3.1: LAMAX-S 翻訳課程

LAMAX-S が効率のよいプログラムを生成できるようにするためには、その数学的特性から (1) ライブラリのどの手法のサブルーチンを用いるか、(2) 行列演算に潜むベクトル処理を抜き出しスーパーコンピュータなどの高速コンピュータに対応すること、などの処理を正しく行うことができなければならない。

第 3 の目標は、特定のコンピュータの上だけでなく、パソコンからワークステーション、さらには各社のスーパーコンピュータの上で同じように動作するようにすることである。このためには、LAMAX-S はさまざまな種類のライブラリを同じように利用できるようにする必要がある。各社のスーパーコンピュータ上には、そのスーパーコンピュータを効率よく用いることができるライブラリが用意されている。これらのライブラリを LAMAX-S では積極的に利用する。

これらの目標を達成するために、本章では LAMAX-S の文法の設計がどのようになっているかについて述べる。

3.2 数式の記述性

LAMAX-S では、数式の記述性について特に重視し、関数による方法を極力さけ、理解しやすい記号を取り入れることにした。一般の数式では転置行列演算子は X^T あるいは X' であるのでそれに準じ X' とする。参考のために、非線形最適化問題に対する準ニュートン法にあらわれる代表的行列表現である BFGS 公式 [4]

$$H_{k+1} = H_k + (1 + \frac{y_k^T H_k y_k}{s_k^T y_k}) \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{s_k^T y_k}$$

を LAMAX-S で表現した例を次に示す。正方行列 H_{k+1}, H_k は H、ベクトル y_k は y、ベクトル s_k は s で表現する。ただし、式の意味及び各記号の意味は、ここでは説明を省略する。

```
H = H + ( 1 + (y'*H*y)/(s'*y)) * ((s*s')/(s'*y)) - (H*y*s'+s*y'*H)/(s'*y)
```

連立一次方程式は solve 文で表現する。連立一次方程式 $Gx = a$ をコレスキ分解を用いて解くという処理は

```
solve  G * x = a @using Cholesky
```

とする[†1]。@using Cholesky は、特にコレスキ分解を明示的に示したい意図で用いており、これがなければ LAMAX-S 処理系が変数 G の宣言から適当な三角分解を選び出す。逆行列は逆行列演算子 (^) で表現するので、$x = G^{-1}a$ は x=G^*a とも書けるが、G の逆行列を計算するよりも連立一次方程式として解いたほうが計算量が少ない。しかし、行列の演算がそのままの形でプログラム上に記述されているので、この式のパターン (x=G^*a) を見て、逆行列の記述を連立一次方程式に変えて解くことが可能となる。参考のために LAMAX-S の演算子一覧を表 3.1 に示す[†2]。

さらに、次のような行列の合成や行列の数値表現がある。

$$U = \begin{pmatrix} U & h_1 \\ h_2 & U^T \end{pmatrix} \quad A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

LAMAX-S では、これを簡潔に表現するために、「|」で行列の結合の開始と終了、「,」で横方向の連結、「!!」で次の行への改行を意味する。上記の式は、

```
U = | U, h1 !! h2, U' |          A = | 1, 2, 3 !! 4, 5, 6 !! 7, 8, 9 |
```

となるが、FORTRAN の継続行を用いれば、行列表現に近づきもっとわかりやすくなる。

```
      U = | U, h1  !                    A = | 1, 2, 3 !
     -    ! h2, U' |                   -    ! 4, 5, 6 !
                                       -    ! 7, 8, 9 |
```

この左の例では、行列 U の寸法が変化してより大きな行列になる。FORTRAN では配列の寸法を実行時に変更することはできないが、行列を扱えるほとんどすべての言語では行列の寸法を実行時に動的に変えることができる。LAMAX-S でも、行列の寸法は、宣言時に固定する方法と実行時に動的に決定する方法の

[†1] この solve 文は、LAMAX-S の目標の一つである FORTRAN 文法との親和性に欠けているという指摘がある。この点について筆者も同意見であるが、読解性の観点からこのような文章表現にしている。本来なら、solve(x, G*x=a) とでもすべきところであろう。また、この表現では、未知数 x のとき、方程式 $xG = a$ が解けないとの指摘もある。この点もまた事実であり、現在の LAMAX-S では、次に説明する逆行列演算子 (^) を用いて、x=a*G^ としている。処理系がこの式から方程式の解として解くことを期待しているわけである。これらについての改善法については、第 3.9 節を参照されたい。

[†2] ここに示した演算子以外にもたとえば、クロネッカーのテンソル積などの演算子の導入を希望する声も多く聞かれた。しかし、すでに演算子を表現するための十分な文字コードがなくなってしまったのである。これらについての改善法については、第 3.9 節を参照されたい。

表 3.1: LAMAX-S における演算子

優先順位	演算子	意 味	形 式	結果の型
1	'	転置行列	M'	M
2	**	べき乗	S**S,M**S	S,M
2	^	逆行列	M^	M
3	*	行列の乗算	S*S,S*M,M*S,M*M	S,M,M,M
3	&	要素ごとの乗算	M&M	M
3	/	除算	S/S, M/S	S,M
3	%	要素ごとの除算	M%M	M
3	//	文字列の連結	C//C	C
4	+	正符号	+S,+M	S,M
4	-	負符号	-S,-M	S,M
4	+	加算	S+S,M+M	S,M
4	-	減算	S-S,M-M	S,M
5	=	代入	S=S,M=M	S,M
5	<=	挿入	M{S,S}<=M	
5	=>	抜き出し	M{S,S}=>M	
5	<=>	交換	M<=>M	
5	<<	分割	M<<M	

S:スカラ、M:行列・ベクトル、C:文字列

2つを用意している。さて、このときに問題となるのが、実行時に 1 行 1 列になった行列の扱いである。一般の計算では 1 行 1 列の行列はスカラとして扱われる。プログラミング言語の処理上、この処理の扱いは面倒になるが、行列演算用言語としては必要であり、LAMAX-S でもそのように処理している。

固有値や固有ベクトルを求めたり、行列要素の最大値やその位置を求めるには、行列用の組込み関数を用いる (第 3.6 節参照)。行列の行や列の入れ換えのための操作も用意されている。

連立一次方程式を解いたり、固有値や固有ベクトルを求めるためには、その行列の構造や数学的特性がわかっていると都合がよい。LAMAX-S では、行列変数の宣言時に、これらを指定するようになっている。

3.3 行列データの宣言

FORTRAN では、型 (type)・類 (class) という分類がある。型には、整数型、実数型、倍精度実数型、複素数型などがあり、類には、変数、配列、関数、サブルーチン、文関数などがある。LAMAX-S では、この他にさらに「数学的特性」を追加する。数学的特性とは、「構造」、「構造特性」、およびそれらが持つ「数学的性質」のことをいう。したがって、整数型のバンド行列の変数、実数型の三重対角行列を返す関数などが定義できる。数学的特性は表 3.2 に示す 9 個の構造の他に、要素の密度や対称性[†3]などの構造特性を指定でき、さらに表 3.3 に示す数学的性質を示すことが可能となる。

また、行列データには、寸法が固定である静的行列と、寸法が実行時に動的に変化する動的行列の 2 種類がある。行列データを格納する行列変数の宣言は、FORTRAN の型宣言文を拡張した次の形式とする。

```
要素の型:matrix[寸法] 変数名,...
要素の型:matrix[寸法  {: 構造名   {, 付加情報}} ] 変数名,...
要素の型:vector[寸法  {: 付加情報} ] 変数名,...
要素の型:rvector[寸法  {: 付加情報} ] 変数名,...
```

この表現において、{} はその内部が省略可、,... はその前の要素をコンマで区切って並べることを意味する。

[†3]もちろん、対称性は、矩形行列やベクトル、上三角行列、下三角行列など数学的に意味をなさない構造には指定できない。

表 3.2: LAMAX-S で指定できる構造と数学的性質

構　造	宣　言　法	構造特性	
		要素の密度	対称性
矩形行列 (m × n)	matrix[行数, 列数]		
正方行列	matrix[行数, 列数]		
列ベクトル	vector[要素数]	dense	symmetric
行ベクトル	rvector[要素数]		
バンド行列	matrix[行数, 列数:band(上バンド幅, 下バンド幅)]	sparse(要素数)	asymmetric
対角行列	matrix[行数, 列数:diagonal]		
3 重対角行列	matrix[行数, 列数:tri_diag]		
上三角行列	matrix[行数, 列数:upper_tri(対角変位)]		
下三角行列	matrix[行数, 列数:lower_tri(対角変位)]		

本表において下線は既定値を意味する。

表 3.3: LAMAX-S における数学的性質

名　前	宣　言　法	意　味
正定値	pos_def	正定値
正	positive	要素がすべて正
零一	zeroone	要素がすべて 0 か 1
零一負	zerooneminus	要素がすべて 0 か 1 か-1
交代行列	skew_sym	交代行列
エルミート	hermitian	エルミート行列
直交行列	orthogonal	直行行列
ユニタリ行列	unitary	ユニタリ行列

さて、最初の例は、矩形行列あるいは正方行列を意味する。たとえば、

```
double precision:matrix[10,20] X
integer:matrix[100,100]        Y
```

では、X は 10 行 20 列の矩形行列、Y は 100 行 100 列の正方行列、を意味する。X の要素の型は倍精度実数型、Y の要素の型は整数型である。次に、構造などを指定する場合の構文の例を示す。寸法の後に、行列の構造や数学的性質を指定する。たとえば、

```
integer:matrix[100,100:band(3,4)] Z
```

は、Z が 100 行 100 列のバンド行列であり、上バンド幅が 3、下バンド幅が 4 であることを意味している。この要素の型は整数型である。さらに、

```
double precision:matrix[100, 100:symmetric,sparse[1000]] Q
```

という複雑な宣言は、Q が 100 行 100 列の対称スパース行列であり、実際の非ゼロの要素数は 1000 個であることを意味している。また、

```
complex:matrix[100,100:hermitian] R
```

という宣言は、R がエルミート行列であることを宣言している。また、

```
real:matrix[100,100:lower_tri(0)] LD
real:matrix[100,100:lower_tri(1)] L
```

という宣言は、LDが対角部分を含む下三角行列であることを、Lが対角部分を含まない下三角行列(狭義下三角行列)であることを、宣言している。

次に、ベクトルを表現する宣言について述べる。たとえば、

```
real:vector[100] V
real:rvector[100] R
```

という宣言は、Vが列ベクトルであることを、Rが行ベクトルであることを、宣言している。また、

```
real:vector[100000:sparse(1000)] VS
```

という宣言は、VSが10万個の要素を持つ列ベクトルであるが、そのうち1000個の要素が非ゼロであるスパース列ベクトルであることを宣言している。

矩形行列とベクトル以外の構造をもつ行列はすべて正方行列である。

行列の寸法を固定しないで、実行時に動的に可変にするには、寸法に「*」を指定する。たとえば、

```
real:matrix [*,*:band(*,*),symmetric,pos_def] B
real:vector [*]                               V
```

という宣言では、正定値対称バンド行列Bと列ベクトルVを定義している。BやVの寸法は、次の例のように、代入時や実行時に行列データ域を割り当てる「allocate文」によって決定される。

```
B = X * Y  ← (X*Y)の演算結果をBに代入
allocate(B[100,100:band(3,4)]) ← Bを100行100列、上バンド幅3、下バンド幅4にして割当てる
```

また次のように数学上矛盾した指定をするとエラーになる。

```
integer :matrix [100,100:band(3,4),symmetric] R  ←バンド幅が異なるので対称行列にはならない
real    :matrix [100,100:hermitian] H    ←エルミート行列の要素は複素数
```

3.4 行列の構造とその扱い

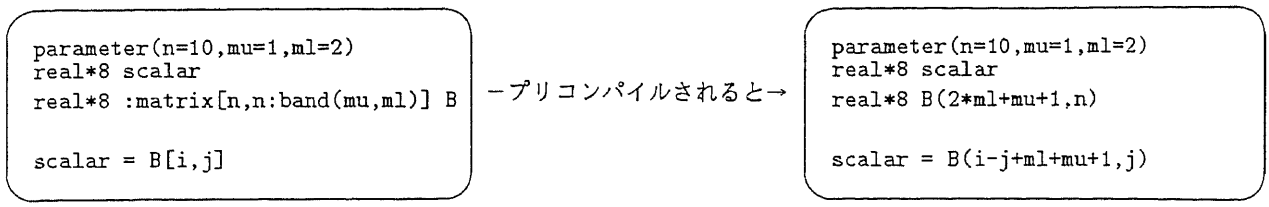
行列の構造の詳細について利用者はいっさい意識する必要がない。たとえば、A, B, C, Dを行列変数とするときに、次の式は数学的に意味をもてば、各変数の構造は何であってもよい。

```
real*8 matrix:[10,10]          A
real*8 matrix[10,10:band(3,3)] B
real*8 vector[10]              C
real*8 matrix[10,10]           D

A = B * C + D
```

```
real*8 matrix:[10,10]              A
real*8 matrix[10,10:upper_tri(0)] B
real*8 matrix[10,10:tri_diag]      C
real*8 matrix[10,10:diagonal]      D

A = B * C + D
```

各構造間の演算結果の構造は、数学的妥当性に沿って決定される。各構造の実際のデータ表現および演算はコンパイラが管理し、プログラムの作成者は単に数式のみを記述すればよい。ライブラリが異なれば、行列の構造のデータ構造や個々のサブルーチンの引数の順序や意味も異なる。たとえば、LINPACKでは非対称バンド行列 B_{ij} 要素は、実際には、$b_{k,j}$ に格納される。ここで、$k = i - j + m$、ただし、$m = m_l + m_u + 1$ である[†4]。しかし、他のライブラリたとえば、NEC SX上のASLやHITAC S上のMATRIX/HAPでは、$b_{t,i}$(ただし $t = j - i + m_l + 1$)に格納される。LINPACKがライブラリとして仮定されている場合には、プリコンパイルされると次のようになる。

[†4]ただし、m_uを上バンド幅、m_lを下バンド幅とする。



LAMAX-S ソースプログラム　　FORTRAN プログラム (LINPACK の場合)

もちろん、これは LINPACK の場合であり、他のライブラリの場合には異なる FORTRAN プログラムになる。これを実現するために LAMAX-S では、各ライブラリのデータ表現やサブルーチンの指定法をデータベースとして供えるようにしている。ここで大切なことは、LAMAX-S プログラムはあくまでも抽象的な数学表現だけで数式を表現することを目指しているので、具体的なバンド行列などへのデータ構造の変換は LAMAX-S 処理系にまかせるという点である。ライブラリによっては LAMAX-S が必要とする各種構造に対する連立一次方程式の解法などの機能を十分に供えていないものがある。その場合には、その足りない部分だけを LINPACK などの他のライブラリで補うことはできない。なぜならば、各構造のデータ表現が異なるからである。そこで、LAMAX-S によって記述された線形計算用の「標準ライブラリ」を用意してそれを利用する。LAMAX-S によって記述されているので、そのライブラリに変換する LAMAX-S 処理系によって、その標準ライブラリをコンパイルすれば、翻訳された FORTRAN プログラム上に反映される構造はそのライブラリとまったく同じ構造になる。

代入によって、行列変数に他の行列データの値がコピーされるが、この場合、構造の違う任意の他の構造に代入したい場合がある。LAMAX-S の代入演算子 (=) は、この処理を自動的に行う。代入が行われた場合、代入側の非ゼロ要素には常に見かけ上ゼロがコピーされる (見かけ上というのは、たとえば、バンド行列の非ゼロ要素の部分のメモリは割り付けられていないことが多く、実体がないため、このように呼んだ)。たとえば、B をバンド行列、D を対角要素とすると

```
D = B
```

では B の対角部分だけが D にセットされる。D の対角要素以外はゼロと仮定される。なお、このように構造の違う行列変数にデータを代入することを LAMAX-S では、強制代入と呼んでいる。

行列変数に構造を与えるのは、メモリ効率と実行速度を向上させるためであるので、とくにそれを意識しない場合には、すべて矩形行列か正方行列で宣言してもかまわない。

3.5 部分行列と行列要素

LAMAX-S では、部分行列および行列要素の表現方法は、LAMAX-E とまったく同じである。その例を、図 3.2 に示す。ただし、行列の列ベクトル・行ベクトルを表現する記法が導入された。

3.6 LAMAX-S の組込み関数と組込みサブルーチン

LAMAX-S では、行列内の最大値や最小値、和や寸法を取得するための組込み関数が用意されている。その一覧表を表 3.4 に示す[†5]。

組込みサブルーチンとして、minput(行列変数) と mprint(行列変数) が用意されている。minput が標準入力から行列データを入力し、mprint が標準出力へ行列データをその構造に従った印刷形式で出力する。

[†5]ただし、行列操作に関する部分 (たとえば、固有値など) は、プロトタイプ処理系には組み込まれていない。外部サブルーチンとして提供されている。

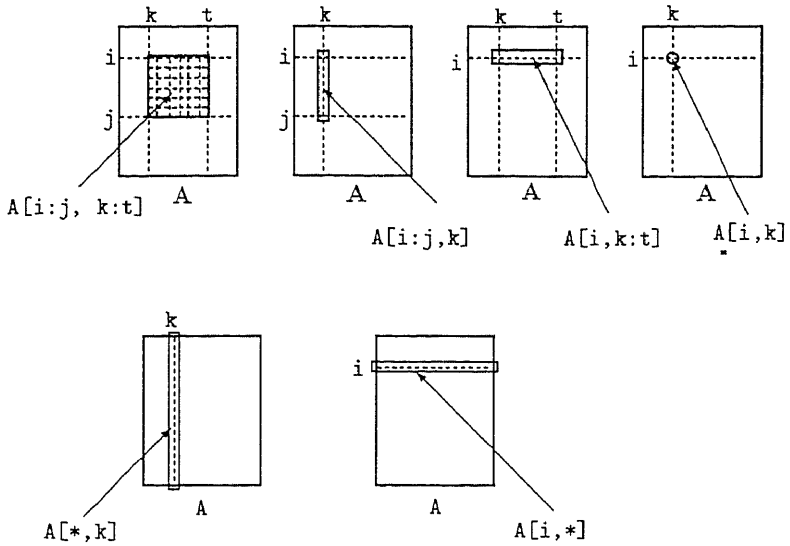


図 3.2: LAMAX-S における部分行列と行列要素の表現

表 3.4: LAMAX-S における組込み関数

分 類	関数名	結果の構造	結果の型	意 味
最大値	allmax(M)	スカラ	I4	最大の値を持つ行列要素の値
	almaxr(M)	スカラ	I4	最大の行列要素の値を持つ行の位置
	almaxc(M)	スカラ	I4	最大の行列要素の値を持つ列の位置
	absmax(M)	スカラ	I4	絶対値最大の値を持つ行列要素の値
	asmaxr(M)	スカラ	I4	絶対値最大の行列要素の値を持つ行の位置
	asmaxc(M)	スカラ	I4	絶対値最大の行列要素の値を持つ列の位置
最小値	allmin(M)	スカラ	I4	最小の値を持つ行列要素の値
	alminr(M)	スカラ	I4	最小の行列要素の値を持つ行の位置
	alminc(M)	スカラ	I4	最小の行列要素の値を持つ列の位置
	absmin(M)	スカラ	I4	絶対値最小の値を持つ行列要素の値
	asminr(M)	スカラ	I4	絶対値最小の行列要素の値を持つ行の位置
	asminc(M)	スカラ	I4	絶対値最小の行列要素の値を持つ列の位置
和	allsum(M)	スカラ	−	行列要素の総和
	rowsum(M)	行ベクトル	−	行列の列和
	colsum(M)	ベクトル	−	行列の行和
寸法	irow(M)	スカラ	I4	行列の行数
	icol(M)	スカラ	I4	行列の列数
	upband(B)	スカラ	I4	バンド行列 B の上バンド幅
	lwband(B)	スカラ	I4	バンド行列 B の下バンド幅
	trsize(T)	スカラ	I4	(上・下) 三角行列の対角変位
行列操作	rank(M)	スカラ	I4	行列のランク
	det(M)	スカラ	M の要素の型	行列式の値
	ginvrs(M)	行列	−	一般化逆行列
	eigenv(M)	列ベクトル	−	固有値
	eigen(M)	行列	−	固有ベクトル
	normp(M,p)	スカラ	M の要素の型	p 乗ノルム
	norm1(M)	スカラ	M の要素の型	絶対和ノルム
	norm2(M)	スカラ	M の要素の型	ユークリッド・ノルム
	norm8(M)	スカラ	M の要素の型	最大値ノルム

3.7 プログラムの記述法とコメント

プログラムの記述法 (継続の仕方・変数名の長さなど) などは、FORTRAN77 に準拠している。ただし、インラインコメントの記述法を Fortran90 の文法から導入している。つまり、行内に ！を記述するとそれよりもその行の右側の部分はインラインコメントになる。

3.8 LAMAX-S における文法改善

2.3.8.2 で述べたように、LAMAX-S の文法を 1992 年に改善した。主な改善点は、行列・ベクトルの宣言方法と数式の記述法である。前節までの説明は、改善後の LAMAX-S の文法である。

3.8.1 行列・ベクトルの宣言方法の改善

改善前は、行列・ベクトルは、たとえば、次のように宣言することになっていた。

```
real A:rectagular[10,20]
real B:square[10]
real C:band[10,3,3]
real D:band[10,3,3:symmetric]
real E:band[1000,30,30:symmetric,sparse[1000]]
real F:square[100:symmetric,pos_def]
real G:vector[100]
real H:rvector[100]
```

これは、A は 10 行 20 列の矩形行列であり、B は 10 行 10 列の正方行列であり、C は 10 行 10 列のバンド行列 (上バンド幅、下バンド幅がともに 3) であり、D は 10 行 10 列の対称バンド行列 (上バンド幅、下バンド幅がともに 3) であり、E は 1000 行 1000 列の対称スパースバンド行列 (上バンド幅、下バンド幅がともに 30) であり、F は 100 行 100 列の正定値対称行列であり、G は 100 個の要素を持つ列ベクトルであり、H は 100 個の要素を持つ行ベクトルであることを宣言している。

この方法では、まず、構造のあとに宣言されたパラメータの意味が解釈しにくい。たとえば、次の 2 つの例を用いて説明する。

```
real : matrix[100,100:band(5,6)] A      ← 新文法による宣言
real   A:band[100,5,6]                  ← 旧文法による宣言
```

旧文法では、上下バンド幅の指定が「band」から離れてしまっているので、プログラムを読んだときに、その解釈に一瞬戸惑う。これに対し、新文法では、その部分は、band(5,6) にまとめられているので、分かりやすい。

また、宣言の仕方を変更したもう一つの理由は、将来、行列やベクトル以外の概念を扱うことができるようにするためである。現在の LAMAX-S には集合がない。このため、行列の添字集合などを表現できない。そこで、将来的には、set などの概念を次のように簡単な拡張で取り込めるようにしたい。

```
integer:set[100] Index
```

3.8.2 数式の記述法の改善

3.8.2.1 逆行列の表現

旧文法では、ミニ LAMAX、LAMAX-E の流れを受けて、逆行列をわり算の記号を用いてもよいようになっていた。たとえば、$B \leftarrow A^{-1}$ という処理を次のように書いてもよかった。

```
B = 1 / A
```

しかし、これがまずいことがユーザから指摘された。それは、意味上の問題もあるが、これがプログラムエラーを見過ごす原因になっているということであった。たとえば、次のプログラムを考える。

```
real*8 s
real*8:vector[10]  v, p
real*8:rvector[10] q, t
real*8:matrix[10,10] A

s = ( ( p' * v ) / ( q * t' ) ) * A
```

このプログラムでは、p'*v も q*t' もベクトルの内積計算でともにスカラになる。このとき、当然、スカラ同士のわり算になり、結果もスカラである。このときは行列 A がスカラ倍されて s に格納される。ところが記述ミスで v, p と q, t の宣言が逆になってしまったとしよう。すると、p'*v と q*t' が行列になってしまう。すると、/で逆行列を表現するのであれば、q*t' の結果の逆行列が計算され[†6]、p'*v とその結果の乗算が行われ、さらに行列 A との乗算計算が行われてしまう。実際に、この種のミスでトラブルを起こしたユーザが存在した。そこで、新文法では、次のように記述するようにした。

```
s = ( ( p' * v ) * ( q * t' )^ ) * A
```

つまり、^ で逆行列を表現する。このようにすれば、(式) ^ の式は行列であることが要求され、この種のミスは起きない。

3.8.2.2 代 入

LAMAX-S (Ver. 1.4) までは、代入 (同一構造間の代入) と強制代入 (異なる構造間の代入) を厳密に区別し、強制代入演算子:=を用いていた。これは、不便であるとの声が聞かれ、すべて=に統合した。

3.9 本章の結論と今後の課題

本章では、筆者の設計した LAMAX-S の文法を示してきた。プロトタイプシステムを用いた記述実験では、数多くのサンプルプログラムを作成することができ、記述性の良いことが示されている。これについては、7.2.1.2 で示す。

しかし、さまざまなサンプルプログラムや処理系の最適化技法の研究を進めるうちに追加が必要な文法事項も明らかになってきた。たとえば、行列の三角分解を行う decomp 文の導入、行列の交換 (行単位・列単位)、行列要素の indexing などである。decomp 文については、第 4 章の中で述べる。行列の交換は、初期の LAMAX-S 文法およびプロトタイプ処理系には組み込まれていたのであるが、さまざまな構造 (対称行列やバンド行列など) に対する処理に対応しきれず、文法事項から現在はずされている。しかし、たとえば、シンプレックス法などのように行列の部分ベクトルなどを頻繁に交換するアルゴリズムも多いので、この機能の実現化が望まれる。行列の indexing とは、行列内の行ベクトル、列ベクトル、場合によっては要素などの位置情報を、index として管理する方法である。この方法を採用すれば、たとえば、同一行列内の列ベクトルの交換が index の交換だけで済む。さまざまな構造の行列に対する効率の良い indexing と文法の表現が開発されれば、これを利用して行列の交換などを実現することができると考えている。

さて、今後の課題について多少別の観点から筆者の意見を述べる。LAMAX-S の文法を設計していて感じたことは、より記述性を高める言語を設計するには、現在主流のテキストベースのプログラミング言語の文法では、もはや表現が困難であるということである。たとえば、現在の多くのプログラミング言語は分数の表現さえも、1 行の中に平面的に記述しなければならない。一方、最近の多くのワードプロセッサは、

[†6] 実際には、この行列 (q*t') は正則ではないのでここでエラーが発生する。

数式エディタを備えており、WYSIWYG (What You See Is What You Get) の思想を実践している。筆者は、これからのプログラミング言語は、GUI (Graphical User Interface) を積極的に導入し、場合によっては、専用エディタを含む CASE ツールのような総合システムとして構築すべきではないかと考えている。現在の文法上の問題として、未知数 x のとき、連立一次方程式 $xG = a$ を solve 文で表現できないという問題がある。解決策の一つとして、solve 文に「`@unknown 未知変数`」という構文を導入し、`solve x * G = a @unknown x` などとする方法も考えられるが、GUI を導入すればもう少し分かりやすい表現になると思われる。GUI を用いれば、実に多様な記号を演算子として採用できるし、また、文の表現の仕方にも選択の幅が広がる。LAMAX-S の次の言語は、このような統合システムとして実現されると考えている。

第 4 章 LAMAX-S と行列演算の最適化

4.1 はじめに

LAMAX-S は、プリプロセッサとして実現されることを前提として言語仕様が設計されており、このプロトタイプ処理系も、LAMAX-S のプログラムを FORTRAN 77 のプログラムに変換するようになっている。

LAMAX-S の文法が、FORTRAN や C などの他の常用のプログラミング言語と大きく異なる点は、LAMAX-S では、問題対象のモデルの行列に関する数学的特徴をプログラム上に記述することを可能にしたという点である。たとえば、行列の変数に対しては、その行列の構造 (バンド行列、三角行列、対角行列、対称性など) や行列の性質 (正定値など) を指定する。行列の変数に付加されたこれらの情報と、実際にプログラム上に記述された行列の算術式の形をヒントにして、LAMAX-S プリコンパイラは、効率のより良い FORTRAN コードを生成することが可能となっている。筆者らが作成したプロトタイプ処理系は、行列の変数に付加された構造を識別して、それに対応した LINPACK サブルーチンを呼び出すプログラムを生成するようになっていた。しかし、プログラムに記述された行列の算術式の形や特性を利用して効率のより良いプログラムを生成するという最適化機能は、LAMAX-S 処理系の当初の開発目標の一つであるが、このプロトタイプ処理系には未実装であった。常用のプログラミング言語の最適化機能は、どちらかというとプログラムの実行手順に関する分析 (フロー解析・ループの最適化・レジスタ割付けなど) が中心であり [3]、LAMAX-S で実現しようとしている最適化方式には適用が困難である。

本論文の目的は、LAMAX-S プリプロセッサが、LAMAX-S プログラム上に記述された行列の数学的な記述を全面的に利用して、効率のより良い FORTRAN プログラムを生成する最適化技術についての提案 [68] とその具体的な処理内容について述べることである。具体的には、LAMAX-S プリプロセッサが生成する FORTRAN プログラムに対して行列の演算ルール (変形則) を適用することによって、プログラムの計算方法を変更し、より少ない計算量で実行できるようにプログラムを変換するための処理方式 (本論文では、この処理を**計算量低減化**と呼ぶ) について述べる。この方式の基本的な考え方は、演算ルール群に基づいて、プログラム上の式を可能ないくつかの形に変形し、その中で計算量の最も少ないものを選択するというものである。

また、本論文で提案された技術が実現可能であることを示すために、筆者らが作成した LAMAX-S のための最適化プリプロセッサのプロトタイプシステム (以下、**LOPS**(LAMAX-S Optimization Prototype System) と略す) とそれを用いたいくつかの事例研究および実験結果について報告する。LOPS は、簡略化された LAMAX-S のソースプログラムを入力し、本論文で提案する処理方式に従って、より少ない計算量で計算するための手順を出力する。LOPS はプロトタイプシステムであるので、FORTRAN プログラムそのものは出力しない。いくつかの行列の算術式に対して、LOPS を適用したところ、おおむね妥当な計算手順が出力され、本方式の妥当性が確認された。

ところで、筆者らが、LAMAX-S を通して行っているこれら一連の研究は、数値計算におけるプログラミング環境を改善しようとする試みである。このような環境改善を目標にした研究は他にも多くある。古くは、APL 言語 [28] に代表されるように、数学の標記法の研究成果が、プログラミング言語に導入された。近年では、知識ベースを利用して、数学表現を知的に理解するシステムを構築し、コンピュータと人間の間の数式を介したユーザインタフェースを改善しようとする Yanjie Zhao らの研究 [73] がある。また、膨大な数の数学ソフトウェアの案内システムとして GAMS (Guide to Available Mathematical Software) が有名である [37][6]。これは、GUI を備えた非常に高度なソフトウェア検索システムであり、数値計算を行うプログラムを作成する研究者の労力を減少している。また、Mathematica を代表とする数式処理システム

も、単に数式処理を行うだけでなく、式をビジュアルにグラフ化する機能などを導入し、統合環境的な発展に至っている。

このような研究の流れの中で、筆者らの研究では、まず、最初のステップとして、行列計算に限定しているが、非常に強力な数学的記述力を持ったプログラミング言語とその最適化処理系を提供しようとするものである。

本章の構成について記す。第 4.2 節では、重回帰分析の事例を用いて、筆者が提案する計算量低減化の概要について説明する。第 4.3 節では、計算量低減化の仕組みについてその概要を述べる。第 4.4 節では、行列の代数的な演算規則をルールベース化して計算量低減化を実現するための手法について、第 4.5 節では、式の展開や因数分解の変形に基づく計算量低減化について、筆者の提案を示す。第 4.6 節では、第 4.2 節から第 4.5 節までの主張を確認するために、筆者らが作成した LOPS について言及する。第 4.7 節で、より効果的に計算量低減化を行うために、LAMAX-S に組み込むべき文法の提案と今後の課題について述べる。

4.2 LAMAX-S における計算量低減化の概要

FORTRAN などのプログラミング言語の最適化手法は古くから研究され整理されてきた。しかし、その手法の多くは、数学的な性質を利用するというよりは、ループ内における不変式のループ外への移動や生成された機械語の列の前後関係から処理効率のより良い機械語列を生成する、といった手順的な変更によるものが中心であった[†1]。

本論文では、プログラム上に記述された行列の数学的な特徴の記述からプログラムの計算量を低減化するという最適化を行うことを提案する。この最適化の具体的な例を説明するために、LAMAX-S によって記述された重回帰分析プログラムを取り上げる。

```
          parameter(m=100,n=50)
          real*8:matrix[m,n] X
          real*8:vector[m]   y
          real*8:vector[n]   B
c
          call minput(X)
          call minput(y)
c
          B = (X'*X)^*X'*y
c
          call mprint(B)
c
          end
```

Program 1. LAMAX-S による重回帰分析のプログラム

このプログラムでは、m 行 n 列の行列 X、要素が m 個のベクトル y、要素が n 個のベクトル B を宣言している。これらの行列・ベクトルの要素は倍精度 (real*8) である。サブルーチン minput は引数に記述された行列あるいはベクトルを標準入力からデータとして入力する。サブルーチン mprint は引数に記述された行列あるいはベクトルを標準出力へ見やすい書式で出力する。演算子 ' は転置行列を意味し、X' は X^T を意味する。演算子 ^ は逆行列を意味し、X^ は X^{-1} を意味する[†2]。

さて、同一の問題 (この例では重回帰分析) に対して、さまざまな解法が考えられる。解法を選択する目安の一つに計算量がある。筆者の提案する計算量低減化では、与えられた問題を解くために、行列の演算ルールなどを適用して可能な限り多くの解法を生成し、その各解法それぞれに対して、計算量を算出し、そ

[†1] もちろん、一般的な FORTRAN コンパイラでも、式を変形して最適化を行うこともある。たとえば、a*b+a*c を a*(b+c) に変更して乗算の計算回数を低減するなどの最適化を行っている。

[†2] 文献 [65] で LAMAX-S を発表してからも文法の改訂作業が続けられている。LAMAX-S (Ver. 1.5) では、逆行列を ^ で表現する。文献 [65](Ver. 1.1) では、X**(-1) で表現していた。

の中で最も計算量の少ない解法を選択する。解法の選択の基準に関しては、計算量の他に、計算精度や作業領域を尺度とする方式も考えられる。本論文では、計算量を評価尺度としているが、与えられたそれぞれの解法に対して、計算誤差や作業領域を求める方法を得ることができれば、本論文での議論は、これらの評価尺度に対してもほとんどそのまま適用できる。

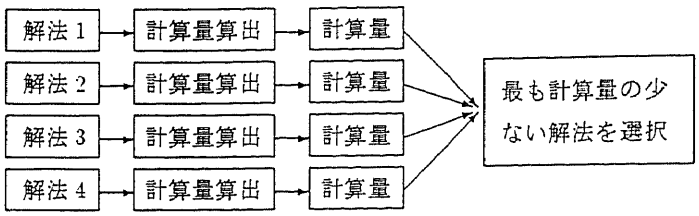


図 4.1: 最も計算量の少ない解法を選択する

重回帰分析の場合には、次の 4 通りの解法が考えられる。

Program 1. を素直に FORTRAN プログラムに変換するのであれば、$(X^TX)^{-1}$を計算して、$B = (X^TX)^{-1}X^Ty$に代入するプログラムが考えられる。しかし、よく知られているように、逆行列を求めることは計算量が多くなり、また誤差も大きいので望ましいことではない [36]。そこで、プログラム上の式を連立一次方程式の形に変形し、$(X^TX)B = X^Ty$を LU 分解法で B について解くことが考えられる。さらに、(X^TX)が正定値対称行列になることに着目すれば、$(X^TX)B = X^Ty$をコレスキー分解法で B について解くことも考えられ、これにより計算量をさらに減らすことができる。また、$B = (X^TX)^{-1}X^Ty$を変形して正規方程式 $(X^TX)B = X^Ty$と考え、$X = QR$(Q : 直交行列、R : 上三角行列) と QR 分解して $RB = Q^Ty$ を B について解くことも考えられ、これによりさらに計算量を減らすことができる。

表 4.1: 重回帰分析のためのさまざまな解法の計算量

解　法	計算手順	計算量	作業容量	LINPACK
逆行列	W1 = X'*X	mn^2	$2n^2 + n$	DGECO
	W2 = X'*y	mn		DGEDI
	W1 = W1^{-1}	$n^3 + 3n^2$		
	B = W1*W2	mn		
	合計	$n^3 + (3+m)n^2 + 2mn$		
LU 分解	W1 = X'*X	mn^2	$n^2 + n$	DGECO
	W2 = X'*y	mn		DGESL
	solve W1*B=W2	$\frac{1}{3}n^3 + 4n^2$		
	合計	$\frac{1}{3}n^3 + (m+4)n^2 + mn$		
コレスキー分解	W1 = X'*X	mn^2	$n^2 + n$	DPPCO
	W2 = X'*y	mn		DPPSL
	solve W1*B=W2	$\frac{1}{6}n^3 + 4n^2$		
	合計	$\frac{1}{6}n^3 + (m+4)n^2 + mn$		
QR 分解	decomp X = Q*R	$mn^2 - \frac{1}{3}n^3 (m \geq n)$	$2n + m$	DQRDC
		$m^2n - \frac{1}{3}m^3 (m < n)$		DQRSL
	solve R*B = Q'*y	$3(2m-k)k + \frac{1}{2}k^2, (k = min(m,n))$		
	合計 ($m \geq n$ のとき)	$-\frac{1}{3}n^3 + (m - \frac{5}{2})n^2 + 6mn$		

Program 1. を LINPACK サブルーチンを用いて作成した場合の各解法に要する計算量の概算を表 4.1 に示す[†3]。なお、この計算量は、乗算および加算の計算回数である。作業容量は、行列データあるいはベクト

[†3] コレスキー分解では、X^TXの半分のみを求めればよいが、表 4.1 では、計算の単純化のためにすべての要素を求めるものとして計算量を算出した。

ルデータを一時的に蓄えるためのデータ領域の大きさである。

m=10、n=5 の組み合わせから、m=200、n=100 までの組み合わせの場合の理論的な計算量を、表 4.1 から算出したグラフを図 4.2 に示す。

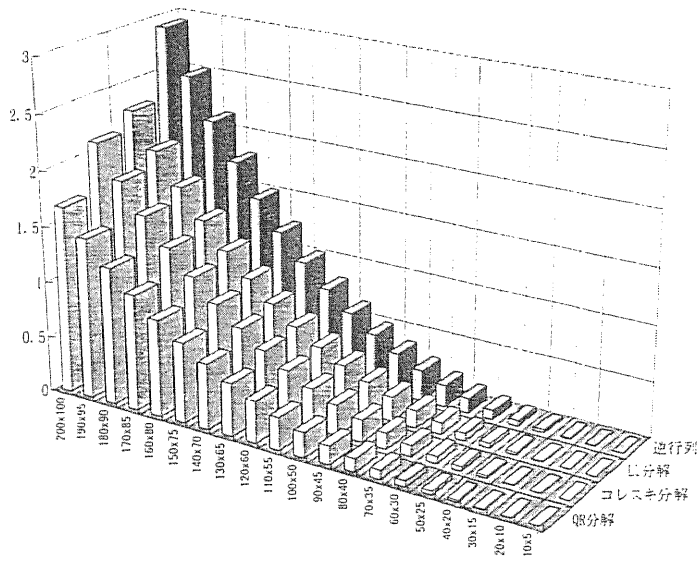


図 4.2: 重回帰分析のための計算量 (理論値)

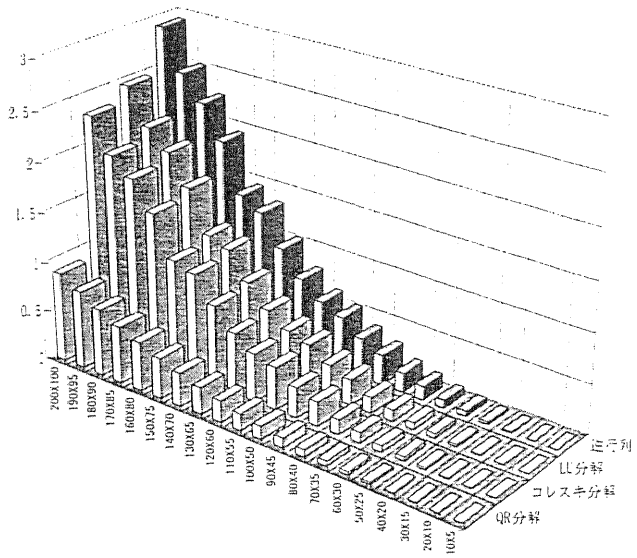


図 4.3: 重回帰分析のための計算量 (実測値)

また、図 4.2 に示した理論的な計算量と実測値のずれを調べるために、LINPACK を呼び出す FORTRAN プログラムを作成して、実際に Sun4 上で計算させた実験結果を図 4.3 に示す。

この 2 つのグラフからわかるように、実際に計算させてみると、LU 分解・コレスキー分解の時間が、理論的に算出された計算量 (図 4.2) よりも多少多く、また QR 分解による計算時間が理論値よりも少なくなっている。計算量と実測値が異なるのは、表 4.1 の計算量の算出が、単に乗算や加算の計算回数だけに注目

しているからである。実際には、配列のアドレス計算やデータの移動のための時間が無視できない量になっていることによるものと考えられる[†4]。しかし、ここで大切なことは、個々の計算の絶対的な計算時間の算出よりも、他の解法との相対的な比較であって、逆行列が最も時間がかかり、次に LU 分解、そしてコレスキー分解との順で処理時間が短くなり、QR 分解が処理時間が最も短く解を得ることができるということがわかれば十分である、ということである。したがって、この問題に対して、LAMAX-S プリプロセッサは、Program 1. を計算量の最も少ない QR 分解を用いた FORTRAN プログラムに変換すればよい。

このように、プログラム上に記述された行列の変数や算術式の数学的特性を利用して、計算量を減らすために LAMAX-S プリプロセッサが行う最適化処理を、本論文では計算量低減化と呼ぶ。このような最適化処理は、現在、数値計算の世界で常用的に使われているプログラミング言語である FORTRAN や C では、ソースプログラム上に数学的な記述がほとんどないので不可能である。一方、LAMAX-S では、数学的な記述がソースプログラム上に示されているので、計算量低減化が可能となる。

計算量低減化が有効となる場面は数値計算プログラムにおいて数多く見受けられる。以下にもうひとつの例を示す。

固有値を求めるハウスホルダー法では、3 重対角化を行うためにハウスホルダー変換を行う。その式は、LAMAX-S では、Program 2. のように記述される。

```
A = ( I - 2 * u * u' ) * A * ( I - 2 * u * u' )
```

Program 2. ハウスホルダー変換：効率の悪い手順

ここで、A は 3 重対角化の対象となる n 行 n 列の行列、I は n 行 n 列の単位行列、u は n 個の要素を持つベクトルである。この式は、このまま計算すると約 $2n^3$ の乗算が必要になるが、Program 3. のように計算すれば約 $2n^2$ の計算時間ですむ [49]。

```
t = u' * A * u
w = 2 * ( A * u - t * u )
A = A - u * w' - w * u'
```

Program 3. ハウスホルダー変換：効率のよい手順

計算量低減化を行うことにより、ソースプログラム上には Program 2. のように記述されていても、実際の計算手順は、Program 3. のように行って処理効率を上げることが可能となる。

4.3 本研究で提案する計算量低減化の実現方法

4.3.1 計算量低減化の処理方式の概要

第 4.2 節では、筆者が提案する計算量低減化の概要について重回帰分析という事例を中心に述べた。本節では、計算量低減化の具体的な処理内容とその実現方法について述べる。基本的な考え方は、前節で述べたように、与えられたプログラムから可能な限り多くの計算処理手順を生成し、その各々に対して計算量を算出し、その中から最も計算量の少ないものを選ぶというものである。処理手順の生成は、図 4.4 に示すように、LAMAX-S ソースプログラムの形で与えられた式を機械的に変形するための部分と、数学的知識から構成される行列の演算ルール群に基づいて式を変形するための部分の 2 つの部分から構成される。

機械的な式の変形とは、3 つ以上の連続する乗算を計算量が最も少なくなる計算手順に変換したり、式を展開したり因数分解したりして計算量を減らしたりする処理である。これらの処理は、計算量低減化処理として LAMAX-S プリコンパイラにあらかじめ組み込んでおく[†5]。

ルールによる変形とは、行列の数学的知識とそれに関する式の変形規則を演算ルールとして扱うことによって、数学的な知識に基づいた式の変形を実現するためのものである。たとえば、第 4.2 節で示したハウスホルダー変換の場合には、図 4.5 のルールが考えられる。

[†4] 特に RISC マシンの場合、この傾向が顕著であることが報告されている [41]。また、プログラムに対するチューニングの有無も実際の計算時間に影響を与える [67]。

[†5] 今回、筆者らが作成した LOPS でも、この部分はプログラムとして組み込んである。

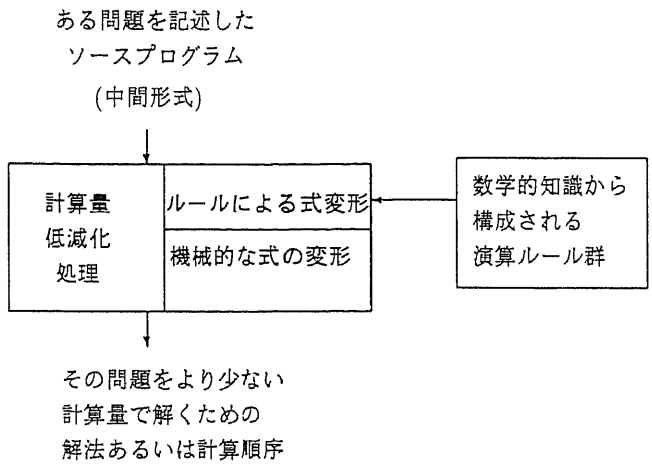


図 4.4: 計算量低減化の過程の概要

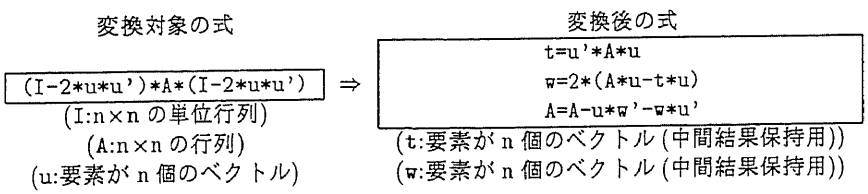


図 4.5: ハウスホルダー変換に対する変形規則

LAMAX-S ソースプログラムの式の中に、この演算式のパターンに一致するものがあれば、それを右の計算順序に変換する。

第 4.2 節でも示したように、数値計算をより効率よく行うための解法は一般的に知られているものでも数多くあり、今後も、さまざまな研究者によって新しい解法が次々に開発されていくものと考えられる。そこで、このような解法を最適化プリコンパイラにプログラムとして組み込んでしまうのではなく、追加や変更・修正が容易なルールの形で管理する方が望ましい。

4.3.2 計算量低減化の具体的な処理例

本節では、計算量低減化の具体的な処理内容を説明するために、第 4.2 節で示した重回帰分析を再び例として取り上げ、これに対して計算量低減化処理を施す。この例では、表 4.2 に示す機械的な式の変形処理 (処理 1、処理 2) および表 4.3 に示すルール群 (ルール 1〜ルール 4) を用いることが考えられる。

表 4.2: 重回帰分析の計算量低減化のための機械的な式の変形

分 類	内 容
処理 1	3 つ以上の連続した乗算を、計算量の最も少ない組み合わせで計算する。
処理 2	括弧でくくられた部分を、作業変数として抜き出す。

表 4.3: 重回帰分析の計算量低減化のための数学的変形則

分 類	内 容	変形規則
ルール 1	A が任意の行列 (ただし、フルランク) のとき、A^TA の結果は正定値対称行列になる。	`A'*A` ⇒ (正定値対称行列)
ルール 2	A が任意の行列、x、b がベクトルのとき、$x = A^{-1}b$ は、連立一次方程式 $Ax = b$ を x について LU 分解法で解けば良い。	`x=A^*b` ⇒ `solve A * x = b`
ルール 3	A が正定値対称行列、x、b がベクトルのとき、$x = A^{-1}b$ は、連立一次方程式 $Ax = b$ を x についてコレスキー分解法で解けば良い。	`x=A^*b` ⇒ `solve A * x = b`
ルール 4	$(X^TX)^{-1}X^Ty$ という式は、X を QR(Q : 直交行列、R : 上三角行列) と QR 分解して、$RB = Q^Ty$ について解けば良い。	`B=(X'*X)^*X'*y` ⇒ `decomp X = Q*R` `solve R * B = Q'*y`

これらの処理 (機械的な式変形) や (数学的知識を利用した) ルールをソースプログラムに順次適用しそのプログラムの計算手順を改良して、その中で計算量の最も少ないものを選ぶというのが基本的な考え方である。

以下に表 4.2 の処理、表 4.3 のルールを適用して、表 4.1 に示す 4 つの解法を導出する方法について述べる。

本論文で提案する計算量低減化処理では、その処理対象は、一つのブロック単位ごとになる[†6]。ブロックとは、goto 文や if 文などでプログラムの流れが分岐しない一連の計算式の列をいう。また、ブロックの内部にはその外部から goto 文などで入ることはできない。

重回帰分析の最初の計算式は次のものである。この場合、1 つの式で 1 つのブロックを形成する。

ブロック 1

```
B = (X'*X)^*X'*y
```

[†6]ブロック単位ごとに最適化するという処理は、従来の FORTRAN などの最適化コンパイラも行っている。

ブロック 1 に処理 2 を適用するとブロック 1 は次のブロック 2 に展開できる[†7]。

ブロック 2

```
W1 = X'*X
B  = W1^*X'*y
```

このブロック 2 に対して、3 つの乗算を含む 2 番目の式 (B = W1^*X'*y) に処理 1 を適用して、X'*y を先に計算するブロック 3 に展開する。

ブロック 3

```
W1 = X'*X
W2 = X'*y
B  = W1^*W2
```

ブロック 3 を、その通りに解釈すれば、W1 の逆行列を求めて、それに W2 を乗じて、B を求めるという次の計算手順が導出できる。

ブロック 4

```
W1 = X'*X
W2 = X'*y
W1 = W1^
B = W1*W2
```

また、ブロック 3 にルール 2 を適用した場合、LU 分解を用いた連立一次方程式へと変換できる。

ブロック 5

```
W1 = X'*X
W2 = X'*y
solve W1 * B = W2 (LU分解)
```

一方、ブロック 3 にルール 1 を適用した場合、W1 が正定値対称行列であることがわかるので、ルール 3 を適用してコレスキー分解の連立一次方程式へ変換できる。

ブロック 6

```
W1 = X'*X
W2 = X'*y
solve W1 * B = W2 (コレスキー分解)
```

また、最初の式 (ブロック 1) にルール 4 を適用すれば、次の QR 分解のプログラムが得られる。

ブロック 7

```
decomp X = Q * R
solve R * B = Q' * y
```

この例のように、計算量低減化では、式の変形を各ブロックに何段階にも適用してゆく。

一つのブロックに対して適用可能な複数の異なるルールあるいは (第 4.5 節で説明する) 式の変形処理を適用すると、図 4.6 に示すように異なる複数のブロックが生成される。これを、本論文ではブロックの展開と呼ぶことにする。

生成されたこれらのブロックの各々に対して、さらに式変形の処理やルールを適用してゆく。最終的には、各段階で適用可能なすべての処理およびルールがそれぞれのブロックに適用され、図 4.7 のようなブロックのトリー構造が作成される (1 つの箱が 1 つのブロックを表す)。各ブロックにこれ以上ルールや式の変形処理が適用できないというところまで、ブロックを展開する。

[†7] 括弧の中の X'*X が抜き取られ、作業変数 W1 に代入される。

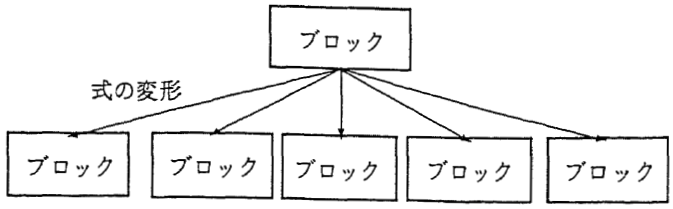


図 4.6: 変形則を用いたブロック展開

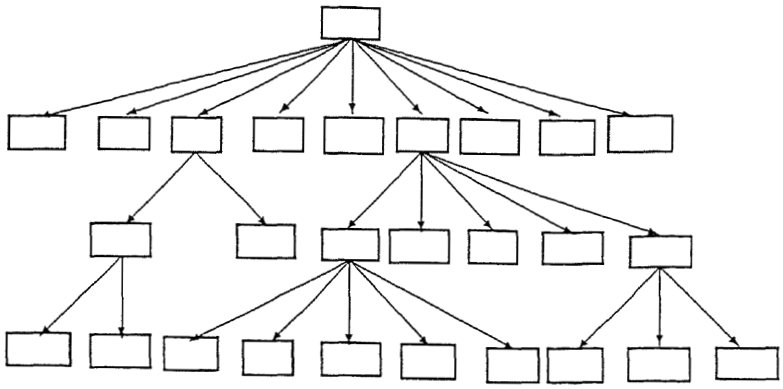


図 4.7: ブロックトリー構造

一つの出発ブロック (最初のブロック) に対して生成されたブロックの総数を展開ブロック数と呼ぶことにする。また、ルールをブロックに適用して得られたブロックの計算手順と同じブロックがすでに他のブロックに重複して存在する場合には、そのブロックは生成しない[†8] 。重複してそれ以上生成されなかったブロックの総数を**重複ブロック数**と呼ぶことにする。また、出発ブロックに対し、最も多く式変形の適用されたブロックの式変形の適用回数を**段数**と呼ぶことにする。たとえば、図 4.7 の例では、出発ブロックは、最も上にあるブロックで、このブロックに対して、展開ブロック数は 26 個、段数は 3 段である。

LOPS で、重回帰分析のプログラムを処理してみたところ、展開ブロック数は 40 個、重複ブロック数は 12 個、段数は 5 段であった[†9]。計算量の最も少ないものは、当然のことながら QR 分解を行う手順であった。

しかし、考えられる計算手順をすべて列挙するこの方式では、式が複雑になると生成されるブロック数が多くなりすぎて実用的でなくなる。その場合には、生成する展開ブロック数の個数の上限値を決めておき、その上限を越えた場合にはそれ以上の展開を中止し、その中から最も計算量の少ないものを選ぶという方法を採用した。LOPS を用いた実験結果では、この方法であっても良好な結果が得られている。

4.4 ルールベースに基づく計算量低減化

本節では、計算量低減化の仕組みのうち、数学的知識をルールベース化して扱う処理方式について説明する。

[†8] この場合、同じ計算手順を持つということが大切である。たとえば、A=(B+C*D)*(E+F*G) を展開した次の 2 つのブロックは、作業変数名が異なっていても、同じ計算手順とみなす。ルールの適用の順序の違いによって、このような重複したブロックが生成される。

```
W1 = B+C*D          W2 = B+C*D
W2 = E+F*G          W1 = E+F*G
A = W1*W2           A = W2*W1
```

[†9] これらの数値は、用意されたルールの個数によっても多少異なる。

4.4.1 数学的な知識ルールを利用した式の変形

LAMAX-S では、変数に対する行列の数学的性質がプログラム上に記述されているので、その数学的な性質を利用して、プログラム上に記述された計算式を計算量がより少なくなるように変形することができる。いくつかの例を示す。

Q が直交行列であることを LAMAX-S では次のように宣言する。

```
real*8:matrix[n,n:orthogonal] Q
```

この場合、プログラムの式の中に、あるいは式を変形した途中結果の中に、Q'*Q あるいは Q*Q' という計算パターンが現れた場合、これを単位行列に置き換えることができる。また、Q^(Q の逆行列) は、Q'(Q の転置行列) に置き換えることができる。

LAMAX-S では、上三角行列 U、下三角行列 L を次のように宣言する。

```
real*8:matrix[n,n:upper_tri(0)] U
real*8:matrix[n,n:lower_tri(0)] L
```

LAMAX-S では、組込み関数 EIGENV によって固有値を求めることができる。U あるいは L に対して EIGENV が適用されている場合、結果は次のようになる。

$$\begin{aligned} \texttt{EIGENV(U)} &\Rightarrow (u_{11}, u_{22}, ..., u_{nn})^T \\ \texttt{EIGENV(L)} &\Rightarrow (l_{11}, l_{22}, ..., l_{nn})^T \end{aligned}$$

また、すでに固有値がわかっている行列に対して、行列式を求める場合、その固有値から行列式を求めることができる。この処理は、一般の最適化コンパイラで行われているデータフロー解析[†10]を拡張して行うことができる[†11]。

4.4.2 ルールの分類

本節では、これまでに述べてきたルールの内容と処理について詳しく説明する。

LOPS では、ルールを表 4.4 [†12]に示す 3 種類に分類した。

表 4.4: ルールの種類

種 類	内 容	処 理
変形ルール	式を変形するためのルールである。例として、逆行列を連立一次方程式に変形するルールがある。	(1) 抜き取り、(2) 展開
性質ルール	式の数学的性質を定義するためのルールである。例として、X がフルランクであれば X'*X が正定値対称行列であるというルールがある。	抜き取り
コード生成ルール	FORTRAN コードに変換できるレベルの記述であることを識別し、計算量を与える。	テンプレートに従った FORTRAN コードの生成 (LOPS では、計算量の算出のみ)

以下に、これらの変形ルールおよび性質ルールの一部を示す[†13]。

[†10]データフロー解析とは、データの定義 (代入) の流れを分析して、プログラムを最適化する手法である [3]。同一ブロックの中で、固有値が求められている場合のみ、この処理が実現できる。あるいは、行列データに固有値をすでに計算したかどうかのフラグを付けて、フラグが立っている (固有値がすでに計算されている) 場合には、その値を用いることなどが考えられる。

[†11]現段階での LOPS は、処理単位がブロックであるので、データフロー解析は行っていない。

[†12]この表の処理の説明において、「抜き取り」とは、対象の演算をもとの式から抜き出して、その部分を計算して作業変数に代入する代入文をもとの式の直前に置くことを意味する。また、展開とは a*(b+c) のような式を a*b+a*c という形に変形することを意味する。

[†13]与式は LAMAX-S のプログラム形式で書かれているが、逆行列の表現はわかりやすさのため、A^{-1} の形で示している。また、A^* は、A の共役転置行列を意味する。その他の表現も場合によって数学的な表現にしてある。表 4.6 、表 4.7 についても同様である。

表 4.5: 変形則ルール群 (一部)

与式	⇒	書き換え則	条件	結果の性質/備考
A''	⇒	A	A:任意の行列	元の性質が保存
A-A	⇒	ゼロ行列	A:任意の行列	
(A+B)'	⇒	A'+B'	A,B:任意の行列	
(A*B)'	⇒	B'*A'	A,B:任意の行列	
A'	⇒	−A	A:交代行列	
A^{-1}	⇒	A'	A:直交行列	
A'*A	⇒	I	A:直交行列	Iは単位行列
A*A'	⇒	I	A:直交行列	Iは単位行列
A^{-1}	⇒	A^*	A:ユニタリ行列	
A^**A	⇒	I	A:ユニタリ行列	Iは単位行列
A^*+B^*	⇒	$(A+B)^*$	A,B:任意の行列	
A^**B^*	⇒	$(B*A)^*$	A,B:任意の行列	
A^{-1}*B	⇒	*solve* A * @ = B (LU 分解)	A,B:任意	@は与式の演算結果
A^{-1}*B	⇒	*solve* A * @ = B (コレスキー分解)	A:正定値対称,B:任意	@は与式の演算結果
B=(X'*X)$^{-1}$*X'*y	⇒	decomp X=Q*R; solve R*B=Q'*y	X:(m×n 行列), B:n 個のベクトル,y:m 個のベクトル	
|A|	⇒	$\prod_{i=1}^{n} a_{ii}$	A:(n×n 上下三角行列)	スカラ

表 4.6: 固有値問題に対する変形則群 (一部)

与式	⇒	書き換え則	条件	結果の性質/備考
eigen(A)	⇒	$(a_{11}a_{22}\cdots a_{nn})'$	A:上下三角行列	
eigen(A)	⇒	$(a_{11}a_{22}\cdots a_{nn})'$	A:対角行列	
|A|	⇒	$\prod_{i=1}^{n} \lambda_{ii}$	A:任意の行列、λ_i:固有値	固有値が予め既知の場合
eigen(A')	⇒	eigen(A)	A:正方行列	
eigen(P^{-1}*A*P)	⇒	eigen(A)	A:正方行列、P:正則行列	

表 4.7: 性質ルール群 (一部)

行列表現	条件	特性
A'	A:対称行列	対称行列
A+B	A,B:対称行列	対称行列
A*B+B*A	A,B:対称行列	対称行列
P'*A*P	A:対称行列、P:(m,n) 行列	対称行列
A^{-1}	A:対称行列、正則	対称行列
c*A	A:対称行列、c:スカラ	対称行列
A^*	A:エルミート行列	エルミート行列
A+B	A,B:エルミート行列	エルミート行列
A*B+B*A	A,B:エルミート行列	エルミート行列
P^**A*P	A:エルミート行列、P:(m,n) 行列	エルミート行列
A^{-1}	A:エルミート行列、正則	エルミート行列
c*A	A:エルミート行列、c:実数	エルミート行列
A+B	A,B:対称行列	対称行列
A-B	A,B:対称行列	対称行列
A'*A	A:任意の行列	準正定値対称行列
A*A'	A:任意の行列	準正定値対称行列
A*A^*	A:任意の行列	エルミート行列
A'*A	A:フルランク (行数 > 列数)	正定値対称行列
A*A'	A:フルランク (行数 < 列数)	正定値対称行列
A^{-1}	A:正定値実対称行列	正定値

計算量低減化では、これらのルールを用いて次の処理を行う。

変形ルールに関しては、(1) そのルールに対する式を抜き取って作業変数にセットする、(2) 式を演算ルールに従って変形する、という 2 つの処理を行う。たとえば、$AG^{-1}c$ という計算を行う LAMAX-S のプログラムがあるとする。

```
Z = A*G^*c
```

この場合、Table 3. のルール 2 がG^*c に適用され、その部分が作業変数 w1 に抜き出されると次のようになる。

```
solve G*w1=c
Z = A*w1
```

性質ルールも同様に、適用された部分を作業変数に抜き取るという処理を施す。

しかし、変形ルールおよび性質ルールの具体的な適用に関しては、LOPS の実験結果から、若干の注意が必要になる。この点については第 4.6 節で説明する。

コード生成ルールは、LOPS では、単に計算量の測定のために用いており、実際の FORTRAN コードは生成しない。実際のコード生成を伴う本格的な最適化プリプロセッサを作成する際には、FORTRAN プログラムのテンプレートを伴い、このルールによってコード生成を行う (第 5 章参照)。

4.5 機械的な式変形による計算量低減化

計算量低減化を実現するためには、ルールベースの他に、機械的に式を変形する処理が必要になる。本節では、それらの手法について説明する。

4.5.1 共通部分式の削除

同一ブロック内にある同じ形の算術式で、その計算結果が常に同じ値になる計算式を、共通部分式と呼ぶ。共通部分式を削除するという最適化処理は、これまでの最適化コンパイラでも一般的に行われてきたが、計算量低減化の処理では、それとは異なる配慮が必要になる。

まず、共通部分式を削除することが常に良い結果をもたらすとは限らないことである。第 4.2 節で示したように、ハウスホルダー変換は、その式の中に、I-2*u*u' という共通部分式を持つ。しかし、これを共通部分式として抜き出し、つぎのようにしてしまうと、もとの計算式のパターンが崩されてしまい、ハウスホルダー変換を最適化するルール (4.3.1 項参照) が適用されなくなってしまう。

```
W = I - 2 * u * u'
A = W * A * W
```

そこで、このような場合、両方の処理[†14]を行い、その結果生成された 2 つブロックの計算量を推定し、小さい方を選択する。しかし、すべてのルールを適用してブロック展開するという規則を守れば、この両方のブロックはその中に必ず存在する。

また、行列演算の場合には、共通部分式があらかじめ、明示的にプログラム上に現れていない場合もある。たとえば、次の例では、一見しただけでは、共通部分式はない。

```
Q = S' * L' * L * S
```

しかし、A'*B'⇒ (B*A)' という変形ルールを用いて、与式を変形すると、

```
Q = ( L * S )' * L * S
```

となり、共通部分式が現れる。

この例のように、行列演算の場合には、直接的に共通部分式がなくても、式を変形することによって、共通部分式が現れることがある。

[†14] (1) 共通部分式を削除するという処理、(2) ハウスホルダー変換の最適化ルールを適用するという処理。

4.5.2 その他の式の変形

次に示す式の変形により、計算量低減化の処理を行うことができる。

1. 3 つ以上連続した乗算の計算
2. 式の展開
3. 因数分解

3 つ以上連続した乗算は、計算量が最も少なくなる手順で計算する。プロトタイプシステム LOPS では、3 つ以上連続した乗算の場合、そのすべての計算可能な組み合わせに対する計算量を求めて、その中で最小の計算量を与える計算手順を選ぶ方式を採用している。この方式では、乗算の個数が多くなると、少ない計算量を見つけるための組み合わせが増大して処理不可能になる。LOPS では、乗算の個数が 10 個を越えると、最適手順をあきらめる[†15]ようになっているが、実際の問題ではこのようなケースは少ない。

展開は、主に共通部分式の削除のために用いられる。以下の例では、各行列の変数の寸法は n 行 n 列であるとする。

次の例はすでに計算された値を利用するために式を展開している。その結果、C を求めるのに行列の乗算ではなく行列の加算ですむようになる。

```
A = X * Y
B = X * Z     ⟹     C = X*Y + X*Z     ⟹   C = A + B
C = X*(Y+Z)          ( X*(Y+Z) を展開 )
```

また、計算式を因数分解や共通項をくくり出すことによっても計算量を減らすことができる。次に 2 つ例を示す。

$X=A^2+AB-BA-B^2$	$\Longrightarrow$	X=(A+B)*(A-B)
(乗算:4, 加減算:3)		(乗算:1, 加減算:2)

A=X*Y+Z*Y	$\Longrightarrow$	A=(X+Z)*Y
(乗算:2、加算:1)		(乗算:1、加算:1)

因数分解のための主要な公式は、そのための変形ルール群を用意することで対応する。

4.6 プロトタイプシステム : LOPS

4.6.1 LOPS の概要

第 4.2 節から第 4.5 節にかけての提案が実現可能であることを示すために、筆者らは LAMAX-S プログラムをターゲットとして計算量低減化を行うためのプロトタイプシステム LOPS を作成した。

図 4.8 に示すように、LOPS は、簡略化された LAMAX-S のソースプログラム (行列変数の宣言とブロック単位の算術代入文) を入力し、それを出発ブロックとして、3 種類のルール群 (変形・性質・コード生成) を適用し、ブロックの展開を行い、必要な統計量 (展開ブロック数、重複ブロック数、段数、適用されたルール) などを計測し、その中で最も計算量が少ないと見積もられた手順を表示する。ただし、実際のプログラムは生成しない。

[†15] プログラム上は、15 個までなら現実的な計算時間内で処理できる (SPARCstation 10:約 135MIPS の場合)。

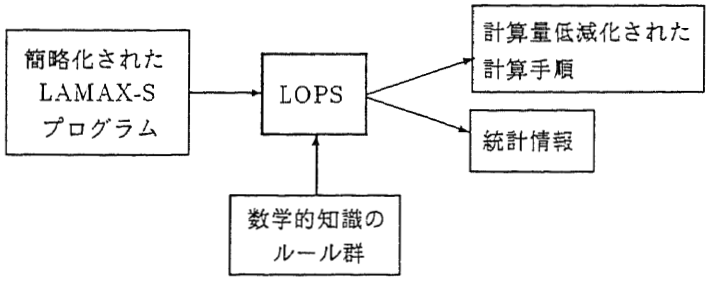


図 4.8: LOPS の概要

4.6.2 LOPS の実験結果から明らかになった処理上の注意点

LOPS の実験結果から、ルールの適用に対して注意を払わないと、ブロックの展開処理が無限ループに陥るケースがいくつかあった。それについて説明する。

4.6.2.1 性質ルールに関する処理上の注意点

X'*X⇒ 正定値対称行列 という性質ルールを用いて、次の重回帰分析のプログラムを変形する。

```
B = (X'*X)^*X'*y   ⟹   W1(sym) = X'*X
                       B = W1(sym)^*X'*y
```

この変形された式に対して、コレスキー分解などの最適化を施すことができることは前述した。

しかし、ここで注意しなければいけないことは、式全体に性質ルールを適用してそれを抜き出すと、ブロックの展開処理が無限ループに陥ることである。たとえば、

```
Z  = X' * X
```

という式に X'*X⇒ 正定値対称行列 という性質ルールを適用して、それを作業変数に代入すると、

```
W1 = X' * X
Z  = W1
```

となり、さらに W1 の式に対して同じ性質ルールが適用され、結局はこの処理過程が無限に続いてしまう。

```
W2 = X' * X
W1 = W2
Z  = W1
```

変形ルールの場合には、抜き取られた式が変形されるので、このようなことは起こらないが、逆向きの変形ルールの適用により元のブロックに戻されることがある。この場合、重複ブロックとなるので、そのルールの適用は行わないようにする必要がある。

4.6.2.2 変形ルールに関する処理上の注意点

多くの処理では、変形ルールの場合、それを抜き出して処理すればうまくいく (4.4.2 項参照)。しかし、うまくいかないケースもある。

```
Z = S' * L' * L * S
```

これに対しては、`A'*B'⇒ (B*A)'`という変形ルールを適用する。

この場合、ルールが適用された部分を作業変数に抜き取ると次のようになる。

```
W1 = (L * S)'
Z  = W1 * L * S
```

これに対して、共通部分式の削除を行うと、次のようになって効率が多少悪くなってしまう。

```
W2 = L * S
W1 = W2'
Z  = W1 * W2
```

そのため、作業変数を用いずに、次のように直接式を変形しなければならない。

```
Z = (L * S)' * L * S
```

この式に対して、共通部分式の削除を行うと、次のように理想的な形になる。

```
W1 = L * S
Z  = W1' * W1
```

4.6.3 実例による評価実験

LOPS を用いていくつか複雑な式を実際に計算量低減化の処理を行った実験結果について報告する。

まず最初に表 4.8 に 3 つの事例を示す[†16]。

表 4.8: LOPS の実験結果

もとの式	条 件	展開	重複	段数
$d=(J^TJ+A)^{-1}J^Tr$	d,r:ベクトル、J:矩形行列、A:正方行列	53	21	6
$A=(I-2uu^T)A(I-2uu^T)$	I:単位行列、u:ベクトル、A:正方行列	121	60	9
$s=p^T(Z^TGZ)p+p^TZ^T(Gx+c)$	s:スカラ、p,x,c:ベクトル、Z,G:行列	421	234	9

実際に出力された計算手順について述べると、たとえば、LOPS が出力した表 4.8 の最後の式 ($s=p^T(Z^TGZ)p+p^TZ^T(Gx+c)$) の計算手順は、「$W_0 \leftarrow p^TZ^T; W_2 \leftarrow W_0G; W_1 \leftarrow W_2x+W_0c; W_3 \leftarrow W_2Z; s \leftarrow W_3p+W_1$」となり、この結果は満足できるものであるといえる。

展開されるブロック数が多くなる場合には、その上限値を設ける。LOPS では、その上限値はオプションとして与えられる。今回の実験では、重複しない展開ブロック数が 1000 個を越えた場合、ブロックの展開処理を中止する。ただし、この場合、選ばれた計算手順には共通部分式の削除がその時点ではまだ十分でないために重複した計算や無駄な作業変数が含まれている場合があることが実験の結果わかっている。そのため、これらの削除を行う処理を、選ばれたブロックに対して行う必要がある。

以下の式[†17]は、この上限値を越える例である。これを、LOPS で最適な処理手順を求めたところ、重複しない展開ブロック数は 1000 個を越えた。重複ブロック数が 2355 個であるので、表 4.8 での展開ブロック数 (表中の「展開」) は 3355 個となった。段数は 20 段であった。

$$Z=\frac{1}{2}(\mu-A^T\lambda)^TG^{-1}(\mu-A^T\lambda)+(AG^{-1}c+b)^T\lambda-c^TG^{-1}\mu$$

[†16] 表の見出しの中で、展開は展開ブロック数、重複は重複ブロック数、を意味する。

[†17] Z,A:矩形行列、μ,λ ,c,b:ベクトル、G:対称行列である。なお、QP 問題 $f(x)=\frac{1}{2}x^TGx+c^Tx$ を線形不等号制約条件 $Ax \leq b$、非負条件 $x \geq 0$ のもとで、Wolf に従って双対問題を作るとこの式が得られる [4]。

この結果得られた計算手順は、「Gをコレスキー分解する; $GW_0 = c$を解く; $GW_1 = \mu$を解く; $W_8 \leftarrow A^T\lambda$; $W_4 \leftarrow AW_0 + b$; $W_5 \leftarrow \mu - W_8$; $GW_6 = W_5$を解く; $Z \leftarrow \frac{1}{2}W_5^T W_6 + W_4\lambda - c^T W_1$」と妥当な結果が得られている[†18]。

これ以外の筆者らの実験でも、与えられた計算式に対して、ほぼ満足できる結果が得られている。

4.7 LAMAX-S への適用とそれに関する課題

本節では、本研究のまとめとして、計算量低減化をより完全なものにするために、LAMAX-S に導入する新しい文法事項と今後の課題について述べる。

4.7.1 LAMAX-S に導入する新しい文法事項

実際に、計算量低減化処理を施すプロトタイプ処理系を作成して、さまざまな問題に適用させたところ、LAMAX-S の言語仕様に対して、追加すべき点が明らかになった。

最も重要なものは、行列を三角分解する decomp 文の導入である。この文は、初期 (1989 年) の LAMAX-S には導入されていたが、結局、次の形の LU 分解の値を常に保証できないので、文法事項からはずしてしまった。

```
real*8:matrix[n,n]              A
real*8:matrix[n,n:lower_tri(0)] L
real*8:matrix[n,n:upper_tri(0)] U

decomp A = L * U
```

しかし、この文に関しては、ピボッティングのための置換行列 P を導入し、次のように表現して、再び LAMAX-S の文法に導入する必要がある。

```
real*8:matrix[n,n:pivot] P

decomp P * A = L * U
```

このようにすれば、たとえば、LU 分解を用いた連立一次方程式 A*x=b のプログラムも次のように書ける。

```
real*8:vector[n] x,b

decomp P * A = L * U
solve P * L * c = P * b
solve U * x = c
```

もちろん、この例の場合には、solve A * x = b とすればよいのであまり利点はない。しかし、計算量低減化の観点から、decomp 文が、有効である例を示す。本論文で何回も例として用いた重回帰分析 (B=(X'*X)^*X'*y) を用いる。これを変形ルールの適用によって QR 分解を用いた計算手順に変換すると、次のようになる。

```
real*8:matrix[m,n]              X
real*8:vector[m]                y
real*8:vector[n]                B
real*8:matrix[m,n:orthogonal]   Q
real*8:matrix[n,n:upper_tri(0)] R

decomp X = Q * R
solve  R * B = Q' * y
```

[†18] 作業変数に、W_2, W_3, W_7がないのは、これらの変数が不要であったため、削除されたからである。

この場合、正規方程式を直接解くサブルーチンがライブラリ中になくても、QR 分解を行うサブルーチンと上三角行列を係数とする連立一次方程式を解くサブルーチン (あるいはその計算手順を示したプログラムフレーム) があれば、結果的に効率の良い手順でこの問題を解くプログラムを生成することができる。

また、たとえば、Program 1. において、一般に $m > n$ でなければ意味がない。LAMAX-S のソースプログラム上にこのような行列の寸法の制限を条件の形で与えられると、よりいっそう効果的な最適化が可能になると考えられる。今後は、LAMAX-S の実際の運用経験の中から、より数学的な概念をプログラム言語の文法としてどのように取り込んでゆくかについて研究を進める必要がある。

4.7.2 数学的特性のルールベース上での定義

LAMAX-S プロトタイプ処理系 [65] では、行列の構造、行列の特性は、すべて処理系に組み込まれていて、新しい構造や特性を追加することは、処理系のプログラム本体を修正しない限り不可能であった。

しかし、行列の特性などには、実にさまざまなものがあり、すべて最初からそれらを処理系の中に導入することはひじょうに難しい。

一方、LOPS では、処理系の中に行列の構造や特性をいっさい持たずに、すべてルールの中で定義可能な事項とした。密行列とベクトルの演算の一部と一般的な式の変形のみが、LOPS 内部に組み込まれている。

筆者らが今後開発する LAMAX-S のフルセット処理系では、この方針を取り入れて、行列の構造や特性をすべてルールベース内で定義できるようにすることが望まれる。

4.8 本章の結論と今後の課題

4.8.1 本章の結論

前節で示した通り、LOPS の出力結果はさまざまな問題に対して、妥当なものであった。数式が複雑になると、展開数を制限する必要があるので、与えられた条件 (ルール) のもとで出力された手順の最適性が失われるが、それでもある程度の効果を上げていることが分かった。また、展開数をかなり限定 (100 個以下) しても、多くの式では与えられた条件下での最適手順を生成することも実験の結果分かっている。しかし、この点に関しては、より多くの例題で実験を積む必要がある。

今後のプログラミング言語の流れの一つとして、LAMAX-S のように記述の抽象度を上げて、より積極的に数学的な概念をプログラミング言語の文法の中に導入するという考えが必要であると筆者は考えている。

本論文で提案した最適化の方法は、そのような言語処理系には必ず組み込む必要があると思われる。ここで提案した内容がそのような言語処理系の設計に参考になれば幸いである。

また、LOPS プログラムの開発は、神奈川大学大学院工学研究科経営工学専攻の相浦宜徳氏によるところが大きい。ここに感謝の意を表したいと思います。

4.8.2 今後の課題

本研究で残された課題は 6 つある。最後にこれらの課題を示して本章を締めくくることにする。

4.8.2.1 評価尺度の問題

LOPS では、与えられた解法の計算量を (倍精度) 実数型の乗算および加算の回数で表現している。しかし、この評価尺度がうまくいかない場合もある。たとえば、スーパーコンピュータなどでは、ベクトル化処理によってプログラムの効率がまったく異なる。そのため、筆者らは、基本的なベクトル演算に関しては、実際に自動計測するシステムを作成した [67]。今回の LOPS には、この研究成果は取り入れていないが、スーパーコンピュータに対応させる場合には、計算量の算出だけでなく、ベクトル演算を取り入れた評価尺度を導入することが必要になるであろう。

また、現在の LOPS では、具体的な寸法の数値が与えられないと処理ができない。寸法がパラメータで表現されていたとしても、簡単な数式処理などを行うことにより、対応できるようにするべきであろう。

4.8.2.2 グローバルなデータフロー解析

LOPS では、計算量低減化処理は、ブロック単位ごとに行っていた。しかし、一般の最適化 FORTRAN コンパイラは、プログラム全体に渡ってデータフローなどの解析を行っている。主要な行列データの流れに関しては、データフロー解析を行う必要があると筆者は考えている。たとえば、ブロック外で LU 分解値や固有値が求められている場合、他のブロックの中でその結果を利用できるようにすることなどが考えられる。

4.8.2.3 検証システム

LOPS では、実際に FORTRAN プログラムを生成しないので、計算量低減化の処理の結果の検証は、人手で行っていた。今回のシステムでは、使用したルール数はそれほど多くはないが、本格的な最適化 LAMAX-S プリプロセッサシステムを開発する際には、これらのルールの正当性の検証と、それによって生成された FORTRAN プログラムの正当性の検証とを適切に行うことが要求される。現在、ルール間の整合をチェックする簡単なシステムを試作中である。

4.8.2.4 処理速度

LOPS では、共通部分式の削除を行わなければ、処理速度に関しては問題なく計算量低減化処理を行う。しかし、共通部分式の削除を行うと、Sun(SPARCstation 10: 約 135MIPS) で 4.6.3 項の例 (QP 問題の双対問題) を展開ブロック数の上限個数を 2000 個にして処理すると、約 15 秒ほどかかってしまう[†19]。これは、共通部分式を発見するアルゴリズムに問題があるためである。現在の方法では、乗算演算子の優先順位に関係なく共通部分式を探すため、検索対象の数式を、数式としてではなく文字列として重複して一致するものを探すという処理を行っている。これをさらに高速なアルゴリズムに変換する必要がある。

また、毎回コンパイルと実行を繰り返すようなプログラムにおいて、最適化処理に要する時間が、最適化によって減少される行列計算の時間よりも長くなってしまうケースも考えられる。このようなケースでは、結果的により大きなオーバヘッドが生じてしまうので、最適化の効果が減少するとしても展開ブロックの個数を制限する必要がある。しかし、多くの実験では、展開ブロックの最大個数を、100 個とかなり少なく制限しても、ある程度満足のいく結果が得られている。また、ライブラリのように、対象プログラムのコンパイルは一回だけでその後、多くのユーザによって利用されるようなプログラムでは、コンパイルにかなりの時間を要しても、最適化の度合いを深めるべきである。このような観点から、最適化に関するさまざまなオプションをユーザのニーズに合わせて用意する必要がある。

4.8.2.5 計算誤差

むやみに計算順序を変えると、桁落ちなどを引き起こす場合がある。式の機械的な変形を行う場合、計算誤差に対する十分な配慮を行う必要がある。

4.8.2.6 より多くのルールの収集

扱う行列がバンド行列やスパース行列の場合、本論文で提案している計算量低減化の効果は大きいと考えられる。現在開発中のフルスペックの LAMAX-S では、スパース行列をサポートする予定であるので、この方面のざまざまなルールの収集を行う必要がある。

[†19] 展開ブロック数が 1000 個程度の上限を越えない場合には処理時間はほとんど問題にならない。

第 5 章 LAMAX-S と自動チューニング

5.1 はじめに

第 3 章で述べたように、LAMAX-S 処理系開発の一つの目標は、行列演算用言語 LAMAX-S で記述されたプログラムを、さまざまなタイプのコンピュータ上で効率よく動作できるように最適化することである。FORTRAN や COBOL などのいわゆる規格化されたプログラミング言語の利点の一つに、移植性の確実さがある。たとえば、FORTRAN のプログラムは、規格の通りにプログラムを記述しておけば、ほとんどの場合、プログラムを手直しすることなしに他のアーキテクチャのコンピュータに移植して、ただちに実行することができる。しかし、プログラムの処理速度を考慮に入れた場合、そのコンピュータのハードウェア特性に合わせて、プログラムをチューニングするという作業を行なう必要がある (図 5.1)。

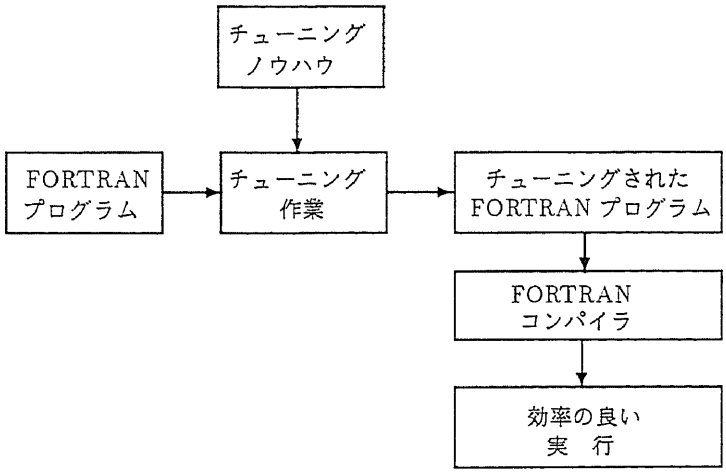


図 5.1: チューニング作業

このチューニングの方法は、そのプログラムが動作するコンピュータのハードウェア特性によって異なる。つまり、あるコンピュータに対してチューニングされ、そのコンピュータ上では効率よく動作するプログラムであっても、別のハードウェア特性を持つコンピュータ上で必ずしもそのプログラムが効率よく動作するとは限らない。たとえば、ベクトル型スーパーコンピュータに対してチューニングされたプログラムは、パソコン上で必ずしも効率よく動作すると限らないのである。

LAMAX-S で記述されたプログラムは、行列計算の形で計算手順が記述されている。第 4 章では、この計算手順の計算量を低減化する方法について述べたが、本章では、LAMAX-S プリプロセッサが FORTRAN プログラムを生成する最終段階で、動作するコンピュータのハードウェア特性に合わせてチューニングされたプログラムを、LAMAX-S プリプロセッサが生成する方法について筆者の提案を述べる。なお、本論文ではこの機構を自動チューニング機能と呼ぶ。

さて、自動チューニング機能は、コンピュータのハードウェア特性を示すハードウェア特性表を個々のコンピュータごとに作成することによって実現する。このハードウェア特性表は、簡単なデータベースと見ることもでき、本論文では、これをチューニングデータベースと呼ぶ。このチューニングデータベースは、LAMAX-S プリプロセッサが生成する FORTRAN プログラムを自動チューニングする際に利用される。自動チューニング機能の概要を図 5.2 に示す。

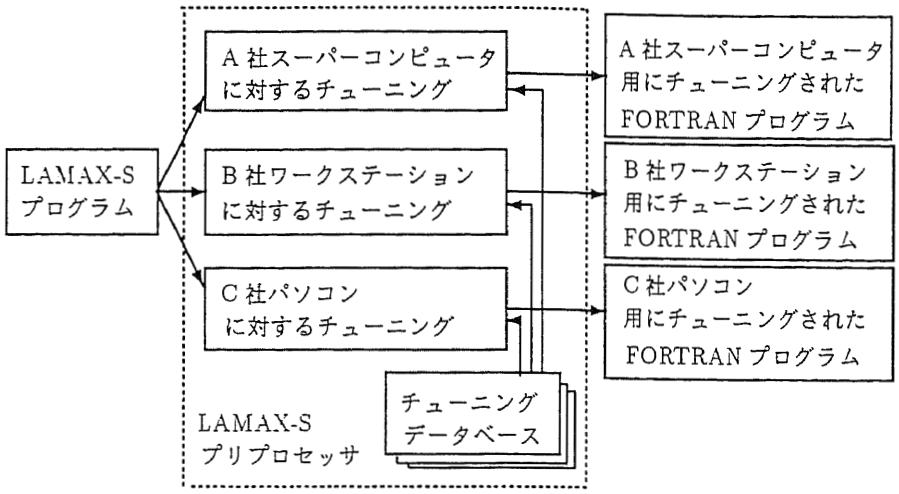


図 5.2: LAMAX-S における自動チューニング機能の概要

ところで、LAMAX-S は、バンド行列や対称行列などさまざまな構造の行列を扱う。これらの多くは密行列とは異なるデータ構造を持つため、それに合わせたチューニングが必要になる。

本章では、フルセット LAMAX-S プリプロセッサを作成するために筆者が提案するチューニング処理を伴うコード生成の方法について説明する。

次の第 5.2 節では、コード生成の方法について、自動チューニングとの関係を示しながら、その概要を説明する。第 5.3 節では、現在の代表的なコンピュータアーキテクチャとそれらに対するチューニング手法について簡単に概観し、本章の議論の対象範囲を明確にする。第 5.4 節では、実際に自動チューニングを行うのに必要なチューニングデータベースの情報を得るために、開発した **ATTPG**(Automatic Tuning Test Program Generator) とそれを用いたいくつかの結果について述べる。

5.2 LAMAX-S における FORTRAN プログラム生成の概要

5.2.1 LAMAX-S プロセッサ全体の流れ

本節では、LAMAX-S プリプロセッサが最終的に FORTRAN を生成する方式について、その概要をチューニング処理とからめながら説明する。そのプロセスの概要を図 5.3 に示す。

LAMAX-S ソースプログラムは、大きく分けて、「行列変数定義」と「行列演算式」に分かれる。これに対して、第 4 章で述べた計算量低減化などの処理が施され、いったん中間形式に変換される。この中間形式が、数段階の処理過程を経て FORTRAN プログラムに展開される。中間形式を FORTRAN プログラムに変換するために、プログラムフレームと呼ばれるプログラムの鋳型を用意する。LAMAX-S プロセッサは、このプログラムフレームをベースにして中間形式を FORTRAN プログラムに展開する。

LAMAX-S プリプロセッサが FORTRAN プログラムへの展開対象とする行列演算は、「構造を持たない行列どうしの演算」・「構造を持つ (あるいは持たない) 行列どうしの演算」・「ライブラリを呼び出す演算」・「基本的なベクトル演算」に分類される。本節では、これらの展開の方法について述べる。

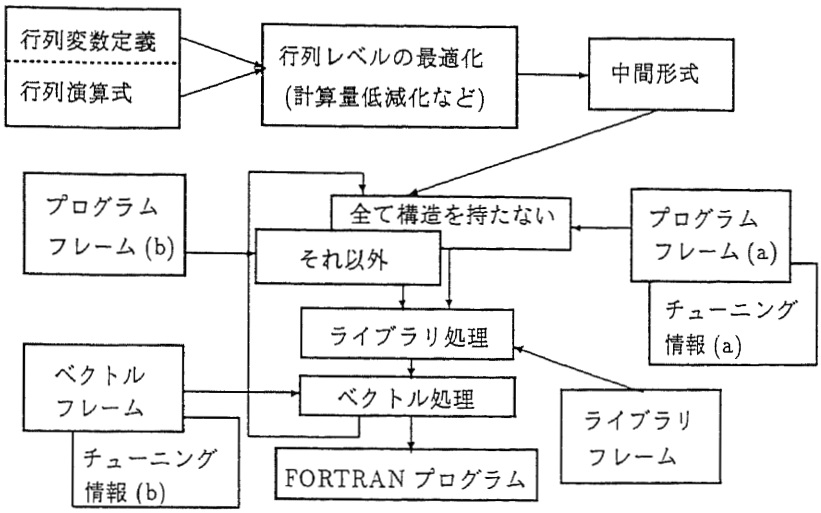


図 5.3: LAMAX-S の最終的な FORTRAN 生成の過程の概要

5.2.2 構造を持たない行列・ベクトルの演算の処理

10 行 10 列の非対称矩形行列 A, B, C [†1] において、A=B*C という行列の乗算に対しては、次のような (正方行列を含む) 矩形行列の乗算のプログラムフレームが用意される[†2]。

```
##fill ( #A, #zero )
##do #k=1, #b, #s
   ##do #j=1, #n
     ##do #i=1, #m
      #A(#i,#j) = #A(#i,#j) + #B(#i,#k) * #C(#k,#j) \ ! 以下はアンローリング記述
                        { + #B(#i,k+#t) * #C(#k+#t,#j) } < #t=1,#s-1 >
     ##end-do
   ##end-do
##end-do
```

1 行目の#fill(#A, #zero) では、行列#A の全要素に値ゼロを格納することを意味している。2 行目から 9 行目は、実際の乗算のプログラムフレームを示している。##do は、適当な文番号を発生し DO 文を生成することを指示し、##end-do はそれに対する continue 文を生成することを意味する。5 行目の逆スラッシュ記号 (\) は、次の行への継続を意味している。また、5 行目の!は、それ以降の記述がコメントであることを意味している[†3]。6 行目はアンローリング (70 ページ参照) と呼ばれる手法の指定を意味している。変数#A, #B, #C には、(LAMAX-S ソースプログラム上にある) 与えられたもとの演算の変数名に置き換え、DO 文の制御変数#i, #j, #k には適当な整数型の変数名を生成し、それに置き換えることを意味している。

ところで、1 行目の#fill を FORTRAN の DO 文の形式[†4] で記述しないことには意味がある。この部

[†1] すなわち、LAMAX-S プログラムでは、次のように宣言された行列の変数である。

```
real:matrix[10,10] A, B, C
```

[†2] 実際には、これにアンローリング (70 ページ参照) の準備や後処理のためなどの指示が加わるので、この例のように単純な形式ではない。ここでは説明のためにそのエッセンスのみを示している。

[†3] この記法は Fortran90 のコメントの記法に準拠している。なお、LAMAX-S でも ! をインラインコメントの記法としている。

[†4] たとえば、次のようなものが考えられる。しかし、これではチューニングができない。

```
##do #j=1, #n
  ##do #i=1, #n
    #A(#i, #j) = 0.0
  ##end-do
##end-do
```

分は、配列にゼロを高速に入れるためのプログラムフレームがベクトルフレームの中に用意されているので、それを用いる。また、この乗算では、前述のようにアンローリングを用いているが、そのための情報がチューニングデータベースに保存されていて、それを用いて FORTRAN プログラムに展開する。チューニングデータベースには、機種ごとに、ベクトル長・要素の型などの要因に対する最適な情報が格納されている。これらのデータは、第 5.4 節で説明する ATTPG によって得ることができる。

さて、プログラムフレームは、その演算の中に、構造をまったく持たない行列[†5]に対するもの (図 5.3 では、プログラムフレーム (a)) と、構造を持つ行列を一つでも持つもの (プログラムフレーム (b)) に対するものに分かれる。プログラムフレーム (a) では、チューニングを含んだ形式で密行列やベクトルに対するプログラムの鋳型が用意される。そのため、チューニング情報 (a) が付加される。このチューニング情報は、5.4.2 項で説明するコンピュータの特性解析システムで得られた情報がベースとなる。プログラムフレーム (a) で扱われる代表的な演算パターンを表 5.1 に示す。

表 5.1: プログラムフレーム (a) で扱われる代表的な演算パターン

A = A	V = R'	R = V'	M = M'	M = M□
A = A + A	A = A - A	A = A & A	A = A % A	A = A' * A
M = M * M	M = M * V	M = M * R	s = R * V	M = V * R

構造を持たない M:matrix、V:vector、R:rvector、s:スカラ
A:M,V,R のいずれか

5.2.3 構造を持つ行列の演算

構造を持つ行列の演算[†6]に対してもプログラムフレーム (b) が用意される。これは、各々のライブラリのデータ構造に合わせてチューニングを施してコーディングされたプログラムがベースになっている。たとえば、LINPACK の場合、非対称バンド行列の対角部分のベクトルが実際のメモリ配置では等間隔になることを利用して、ベクトル長の長いプログラムフレームを作成することができる。ベクトル長が長いということは、スーパーコンピュータにとっては実行速度向上のキーとなる (このような点については、第 5.3 節で述べる)。

また、構造を持つ行列の演算において、全ての演算に対するプログラムフレームを網羅して用意することは、その組み合わせの数を考えれば明らかなように、ほとんど実現不可能である[†7]。そこで、プログラムフレーム (b) では、ベクトル表現で、プログラムフレームを記述することによって全体の記述量を少なくしている。この点については後述する。プログラムフレーム (b) で扱われる代表的な演算パターンを表 5.2 に示す。

構造を持つ行列の演算の具体例を対称行列どうしの乗算を用いて示す。ここでの説明では、ライブラリ及びデータ構造として、LINPACK を前提とする。まず、最初に対称行列のデータ構造について述べる。LINPACK では、次のように対称行列のデータを配置する。

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ & a_{22} & a_{23} & \cdots & a_{2n} \\ & & a_{33} & \cdots & a_{3n} \\ & & & \ddots & \vdots \\ & & & & a_{nn} \end{pmatrix}$$

[†5] すなわち、密行列でかつ矩形・ベクトル (vector)・行ベクトル (rvector) であり、対称行列のようにデータが圧縮されていない行列のこと。

[†6] 構造を持たない行列と持つ行列の混合演算もここに分類される。

[†7] たとえば、15 種類の構造がある場合、15 × 15 = 255 通りの組み合わせが考えられる。これをすべての演算に用意することは大変な作業となる。

表 5.2: プログラムフレーム (b) で扱われる代表的な演算パターン

M = S	M = B	S = SB	B = T	S = S	M = B'
B = B + B	S = S - S	M = L + U	T = T % T	V = V % V	R = R % R
M = B * V	S = M * M'	S = B * B'	T = B	B = T + B	L = M

構造を持たない M:matrix、V:vector、R:rvector、s:スカラ、A:M,V,R
S:symmetric、B:band、SB:symmetric band、T:tridiagonal
L:lower tridiagonal, U:upper tridiagonal

この行列が実際には次のように 1 次元に圧縮されている。

$$a_{11}a_{12}a_{22}a_{13}a_{23}a_{33}\cdots a_{nn}$$

つまり、対称行列のデータの実際の配置の順序の様子は図 5.4 のようになる。対称行列であるので各行および各列のちょうど対角部分までデータが連続して配置され、対角の次の要素から飛び飛びにデータが配置されている。

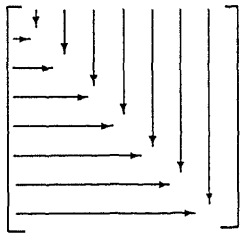


図 5.4: 対称行列の圧縮形におけるベクトル方向

ここで、対称行列どうしの乗算をそれらのベクトルの内積で考えてみると図 5.5 のようになる。各行ベクトルを見ると、前述のように連続部分と非連続の部分に分かれる。

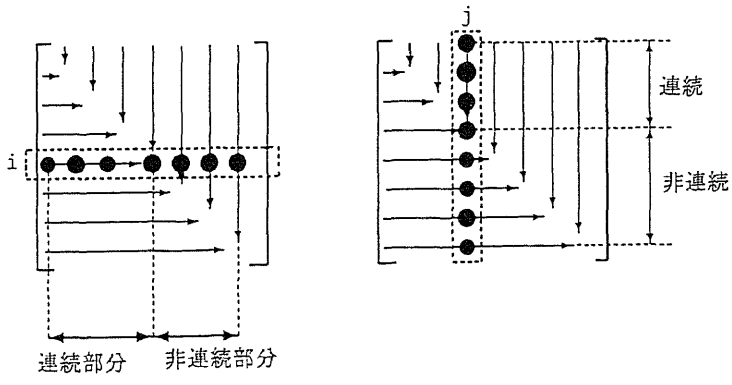


図 5.5: 各行のベクトルの状況

図 5.5 において、i 行目の行ベクトルの先頭の要素の位置は、$\frac{(i-1)i}{2}+1$ で計算され、次の位置は $\frac{(i-1)i}{2}+2$

となり、対角部分 (すなわち、A[i,i]) は、$\frac{(i-1)i}{2}+i$ となる。これ以降 (すなわち、A[i,k], k=i+1, ..., n) は、$\frac{(k-1)k}{2}+i$ となって要素の位置は連続しなくなる。

さて、この内積のプログラムは、『「連続部分＋非連続部分」から構成されるベクトルの内積』という形式に一般化することができる。いま、これを LAMAX-S のプログラムの形で示すと次のようになる。

```
      parameter(n=100)
      real:matrix[n,n]           A
      real:matrix[n,n:symmetric] B,C

      do 10 j=1, n
        do 10 i=1, n
           A[i,j] = B[i,*] * C[*,j]
   10 continue
```

ここで、ベクトル表現 B[i,*] や C[*,j] のデータの配置は、ちょうど「連続部分＋非連続部分」であることに注目されたい。繰り返しになるが、この演算は、対称行列の乗算を計算するための内積計算から、「連続部分＋非連続部分」というベクトルと「連続部分＋非連続部分」というベクトルの内積という演算に一般化できる。詳細は、5.2.4 項で述べるが、LAMAX-S プリプロセッサでは、連続[†8]ベクトルを「s」、非連続ベクトルを「r」、スカラを「t」という記号で表現する。「連続部分＋非連続部分」というベクトルはこの 2 つを組み合わせて、「sr」と表現する。すると、これは「t = sr (内積) sr」という演算パターンで表現できる。つまり、対称行列どうしの乗算というプログラムフレームでは、要素の演算についてまでの詳細なプログラムパターンを指定する必要はなく、このベクトル表現が指定されていればよいことになる。もちろん、「t = sr (内積) sr」という演算パターンに対するプログラムフレームは、別途作成する必要がある (これは、5.2.4 項で述べる)。このことを明確にするために、もう一つ例を示す。次の例は、対称行列と非対称行列の乗算である。

```
      parameter(n=100)
      real:matrix[n,n]           A,B
      real:matrix[n,n:symmetric] C

      do 10 j=1, n
        do 10 i=1, n
           A[i,j] = B[i,*] * C[*,j]
   10 continue
```

この場合、「sc = s (内積) sr」という演算パターンが採用される。つまり、A[i,j] = B[i,*] * C[*,j] のベクトル表現さえ導出できれば、任意の構造のプログラムフレームが、次のパターンだけで表現できることがわかる。

```
      do 10 j=1, n
        do 10 i=1, n
           A[i,j] = B[i,*] * C[*,j]
   10 continue
```

しかし、バンド行列などのようにベクトルの参照位置によって表現が異なる構造を持つ行列もあるので、すべての組み合わせをこのように単純に表現することはできない。しかし、この記法によって、プログラムフレームの記述量をかなり低減することが可能となる。

[†8] 等間隔で配置されているベクトルも含む。

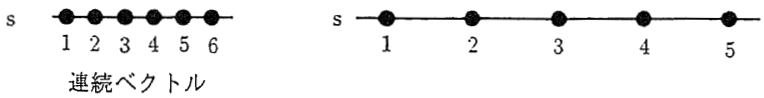
5.2.4 ベクトル表現とその演算

5.2.3 項で示したように、LAMAX-S では、次の形式のベクトルを扱う。ベクトルは、基本ベクトルと基本ベクトルを組み合わせた応用ベクトルがある。

基本ベクトルには、等間隔ベクトル、不等間隔ベクトル、ゼロベクトルの 3 つがある。等間隔ベクトルは、ベクトル中の要素が等間隔でメモリ中に存在するベクトルをいう。特に、要素の間隔が 1 のものを連続ベクトルと呼ぶことにする。不等間隔ベクトルは、ベクトル中の要素の並び方がランダムであり、その位置を計算式によって与えるタイプのベクトルである (図 5.6 参照)。ゼロベクトルは、その要素の値がすべて (仮想的に) ゼロであるベクトルであり、実際のメモリ領域はとられない。

表 5.3: 基本ベクトル

記号	説　明
s	等間隔ベクトル (連続ベクトルを含む)
r	不等間隔ベクトル (間隔が式で与えられる)
z	ゼロベクトル



連続ベクトル

図 5.6: 等間隔ベクトル

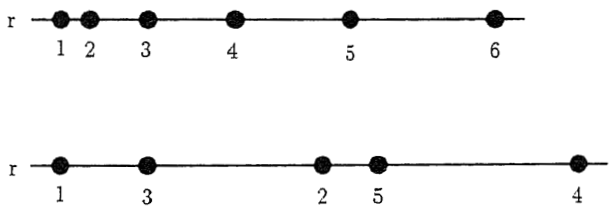


図 5.7: 不等間隔ベクトル

応用ベクトルはこの組み合わせから構成され、表 5.4 のものが用意されている。

しかし、実際に頻繁に使われるのは、s, r, sr, rs, zs, zr, sz, rz, zsz, zrz 程度であろう。

これらのベクトル表現に対して、演算のフレームが用意される。演算は表 5.6 に示すように代入も含めて定義する。最初の 4 つ (加算・減算・乗算・除算) のプログラム構造はまったく同一になる。

ATTPG を用いてこれらのプログラムフレームに対する最適なチューニングのための情報を得て、それに基づいてチューニングデータベースを作成する。1 つだけ簡単な例を示す。2 次元配列にデータをセットする場合、コンパイラによっては 2 次元配列を 1 次元配列に直してセットする場合が多い。これはスーパーコンピュータではかなり有効な手法であるが、ワークステーションやパーソナルコンピュータでもある程度有効である。たとえば、非対称密行列 A の全要素にゼロを代入することを考えてみよう。A は実際には 2 次元配列として宣言される。チューニング手法の一つとして、この 2 次元配列 A を 1 次元配列として FORTRAN の equivalence 文で宣言し、それを利用する方法がある。その具体例を以下に示す。

表 5.4: 応用ベクトル

記号	説　明
s	全ての領域が等間隔ベクトル(連続ベクトルを含む)
r	全ての領域が不等間隔ベクトル
z	全ての領域がゼロ
sr	最初の部分がs(等間隔)、残りの部分がr(不等間隔)
rs	最初の部分がr(不等間隔)、残りの部分がs(等間隔)
zs	最初z(ゼロ部)があって、残りはs
zr	最初z(ゼロ部)があって、残りはr
sz	最初sで、残りがz
rz	最初rで、残りがz
zsz	最初z(ゼロ部)があって、次にs、残りがz
zrz	最初z(ゼロ部)があって、次にr、残りがz
zsrz	最初z(ゼロ部)があって、次にsr、残りがz
zrsz	最初z(ゼロ部)があって、次にrs、残りがz

表 5.5: スカラの表現

記号	説　明
t	スカラ

表 5.6: ベクトル演算の種類

内　容	説　明
加算 (+)	要素同士の加算
減算 (−)	要素同士の減算
乗算 (&)	要素同士の乗算
除算 (%)	要素同士の除算
内積 (*)	要素同士の乗算の和
和 (Σ)	要素の和 ($v_1 + v_2 + \cdots + v_n$)
積 (Π)	要素の積 ($v_1 \times v_2 \times \cdots \times v_n$)
最大値 (max)	要素の最大値
最小値 (min)	要素の最小値
絶対値最大値 ($absmax$)	要素の絶対値の最大値
絶対値最小値 ($absmin$)	要素の絶対値の最小値
関数 ($function$)	各要素に関数を適用して新しいベクトルを生成
代入 (=)	要素の代入

```
      parameter (n=1000)
      real*8 A(n,n),AE(n*n)
      equivalence( A,  AE )
```

このとき、次の2つ[†9]のプログラムフレームがあるとしよう。

```
c   2次元配列版
      do 10 j=1, n
        do 10 i=1, n
          A(i,j) = 0.0d0
   10 continue

c   1次元配列版
      do 10 k=1, n*n
          AE(k) = 0.0d0
   10 continue
```

後述するATTPGは、このようなテストプログラムを作成し、その効果を計測するシステムである。これを実際にSUNワークステーションで行ってみると表5.7のようになる[†10]。

表5.7: 定数セット：速度の比較

動作コンピュータ	次数(n)	2次元配列版(sec)	1次元配列版(sec)	比　率	処理コマンド
SUN (SPARCstation 10)	1000	1.96	0.48	24.5%	f77 zeroset.f
SUN (SPARCstation 10)	1000	1.70	0.18	10.6%	f77 -O4 zeroset.f

このことから、定数を非対称密行列にセットするFORTRANプログラムは、SUNワークステーションの場合、1次元配列版を採用すれば良いことがわかる[†11]。ベクトルフレーム用のチューニングデータベースには、このようにどのコーディング法が最もよいかの情報がコンピュータの機種別に用意されている。

なお、ベクトル表現は、ベクトル処理で最終的にFORTRANプログラムに変換される。

5.2.5 ライブラリを利用する演算の処理

FORTRANで記述しきれない機能もある。たとえば、逆行列の値を求めるプログラムでは、ライブラリを利用する必要がある。この場合、そのライブラリを呼び出すためのコード規約が含まれる。たとえば、A=B^という処理に対しては、次のようなプログラムフレームが用意される。

```
#A = #B ! 行列Bを Aに複写
##idecomp(#P, #A) ← (#A) ! 行列Aを LU分解して、その結果を Aと置換行列 Pにセット
##inverse( #A ) ← (#P, #A) ! その結果から逆行列を計算
```

これは、ライブラリの中から非対称密行列の行列のLU分解を行うサブルーチンをまず呼び出して、次に実際にその分解結果を利用して逆行列を求めるサブルーチンを呼び出すことを意味している。

このように、##idecompや##inverseのような形に分けたのは、多くのライブラリが三角分解を行ってその結果を利用するようになっているからである[†12]。

実際の処理は、図5.3のライブラリ処理で行われる。

[†9]実際には、この2つだけではなく、ゼロクリアするためのもっとたくさんのプログラムフレームが用意される。

[†10]表5.7の「処理コマンド」の-O4オプションは、FORTRANコンパイラの最適化の度合いを最も高くするという意味である。

[†11]後述するが、スーパーコンピュータ用のFORTRANコンパイラは、ここで述べた2次元配列の1次元配列化という最適化を行ってくれるものがほとんどである。しかし、ワークステーション用FORTRANコンパイラやパソコン用FORTRANコンパイラの中にはこのような最適化を行わないものもあるので、ここで示した解析は意味がある。

[†12]idecompのiは、この三角分解が内部的な処理であることを示すために付けられたもので、internalの略である。

5.3 現在のコンピュータアーキテクチャとそのチューニング手法

LAMAX-S が動作するコンピュータは、スーパーコンピュータ、メインフレーム、ワークステーション、パーソナルコンピュータなどさまざまなアーキテクチャを持っている。本節では、LAMAX-S プリプロセッサがチューニングを行う際に必要となる基本的な技術についてコンピュータの種類別に整理する。特に、LAMAX-S はベクトル型スーパーコンピュータを意識して設計されているので、スーパーコンピュータについては詳しく述べることにする。なお、本項の内容は、既存のコンピュータに対する調査という観点から、文献・資料からの引用や抜粋が中心となっている。

5.3.1 スーパーコンピュータのアーキテクチャとそのチューニング

現在、実用化されているスーパーコンピュータの多くのタイプがベクトルコンピュータと呼ばれるものである。本項では、このベクトルコンピュータに関して、チューニングという観点から整理する。

5.3.1.1 パイプラインとベクトル処理

ベクトルコンピュータは、メモリ中に配置されたベクトルデータを、ベクトル処理機構により連続的にベクトルレジスタに転送し、パイプライン処理[†13]で高速にベクトル演算を行なうコンピュータである。結果はベクトルレジスタに格納され、再び連続的にメモリに転送される。これらの演算は、ベクトル命令と呼ばれる特殊な命令群によって行われる。一般に処理するベクトルデータの長さ (ベクトル長) が長いことは、プログラムの高速化において有利である。また、ベクトルデータが長すぎて、ベクトルレジスタに格納しきれない場合には、何回かに分けて繰り返される。これをストリップマイニングと呼んでいる。この機構は、機種によってソフトウェアで行われることもあれば、ハードウェアで行われることもある。

参考のために、日立製のスーパーコンピュータ S810 のベクトル処理の例を表 5.8 [†14]に示す。

表 5.8: S810 のベクトル処理の例

演算タイプ	演　算
四則演算	$A_i = B_i * C_i + D_i$
総和/内積	$S = S + A_i * B_i$
1 次巡回演算	$A_i = A_{i-1} * B_i + C_i$
整理/論理演算	$N_i = L_i * M_i$
変換	$A_i = FLOAT(N_i)$
リストベクトル	$A_{ji} = B_{ji} * C_{ki}$
条件つき演算	$IF \quad A_i < B_i \quad THEN \quad C_i = D_i \quad ELSE \quad C_i = E_i$
集約/分散	$WHERE(A_i > B_i) \quad PACK(C_i = B_i)$
検索	$S = MAX(A_i)$
関数	$A_i = SIN(B_i)$

5.3.1.2 自動ベクトル化 FORTRAN コンパイラ

さて、このベクトル命令をユーザが直接プログラム化することは、生産性の低さや移植性の観点などから、問題がある。これに対処するため、日立、日本電気、富士通などの国産スーパーコンピュータメーカは、FORTRAN コンパイラに自動ベクトル化機能を積極的に組み入れてきた。その結果、一般の FORTRAN

[†13]パイプライン処理とは、一つの処理を複数のステージ (たとえば、浮動小数点データの加算の場合、(a) 指数部の比較、(b) 仮数部のシフト、(c) 仮数部の整数加算、(d) 規格化に分かれる [47]) に分け、その各ステージを並列に動作させることにより、命令の実行間隔を短縮して、実質的な処理時間を短縮させる処理の方式である。スーパーコンピュータだけでなく、現在ではさまざまなアーキテクチャのコンピュータで採用されている。

[†14]文献 [43]165 ページより引用。

プログラムの主にDO文の部分をベクトル化し、ある程度チューニングを行うことが可能となっている。しかし、自動ベクトル化FORTRANコンパイラがベクトル化できるための条件がある。その条件の代表的なものとして、**回帰演算**でないことが挙げられる[47]。次のFORTRANプログラムを考える。

```
      do 10 i=2, n
          A(i) = A(i-1)*B(i)
   10 continue
```

このDOループでは、A(i)の値を決めるのに、その直前のループで求められたA(i-1)の値を利用している。このような演算を回帰演算といい、一般に、m回前のループで決定された値を利用する回帰演算をm次回帰演算という。1次回帰演算は、スーパーコンピュータの種類によってはベクトル命令として実行することが可能であるが、それでも通常のベクトル命令よりは遅くなる。2次以上の回帰演算はそのままでは、ベクトル命令としては実行されないので、そのための工夫が必要となる[47]。

また、ベクトル化できるDOループのコーディング上の規制もある。たとえば、DOループの内部からその外側への飛び出し点を含む場合、ベクトル化できないことがある。さらに、一般的にいえば、ベクトル化されるDOループは最内側のDOループである。

DO文の内部の書き方もチェイニングという機能のためにさまざまな問題を含む。チェイニングとは、複数個のパイプライン演算器をベクトルレジスタを介して結合して、あたかも一つのパイプライン演算器のように動作させる機能である。チェイニングは、ベクトルレジスタの個数、並列動作可能なパイプライン演算器の種類などによって、その効果が異なる。

自動ベクトル化FORTRANコンパイラを利用するには、その特性を十分に理解する必要がある。参考のために、日立製のスーパーコンピュータ用FORTRANベクトルコンパイラであるFORT77/HAPを例にとり、説明する。

FORT77/HAPは、S-810, S-820など日立製のスーパーコンピュータ共通で使えるFORTRANコンパイラであり、コンパイラオプションによってさまざまなモデルに対するオブジェクトを生成する。たとえば、表5.9に示すモデルに対するオブジェクトを生成することができる。

表5.9: FORT77/HAPのオプション：モデルの選択

オプション	説　明
MODEL0	S-810/5(モデル5)用のベクトル命令を生成する。
MODEL1	S-810/10(モデル10)用のベクトル命令を生成する。
MODEL2	S-820/20(モデル20)用のベクトル命令を生成する。
MODEL20	S-820/20(モデル20)またはS-820/15(モデル15)用のベクトル命令を生成する。
MODEL40	S-820/40(モデル40)用のベクトル命令を生成する。
MODEL60	S-820/60(モデル60)用のベクトル命令を生成する。
MODEL80	S-820/80(モデル80)用のベクトル命令を生成する。

この他に、表5.10[†15]に示すコンパイラオプションがある。多くのオプションはコンパイラがベクトル命令を生成する際のヒントとなるべき情報をコンパイラに与えるようになっている。たとえば、SUMオプションは、総和を求める場合、添字の順序通りに演算するかどうかを決めるものである。これは、データの性質に大きく依存する。つまり、精度に敏感なプログラムの場合、データが絶対値の小さい順になっていると積み残しの計算誤差が少なくなる。このようなデータの性質はコンパイラには分からない。しかし、このような判断はプログラムの作成者ならば可能である。

ここで説明したように、コンパイラオプションは、プログラムを書いた人は知っていてもコンパイラが知りえない情報をコンパイラに伝える役割をもっている。コンパイラオプションはプログラム全体に有効で

[†15]文献[15]298ページより引用。

あるが、局所的にコンパイラに指示したい場合がある。その場合、多くのベクトル化コンパイラは、コンパイラ指示行を用いて、局所的にコンパイラに指示を与える。FORT77/HAP の場合、*VOPTION を用いる。

LAMAX-S では、行列演算の計算式からある程度これらの情報について判断を行うことができるので、コンパイラオプションのうちのいくつかをあらかじめプログラムフレームに記述しておく。

表 5.10: FORT77/HAP の主なオプション (下線は既定値)

オプション	説 明
ANALYZE, NOANALYZE	対話型ベクトルチューニング支援 FORT/VF、または、VECTIZER 用に、静的解析情報、静的予測情報、および動的解析情報を出力する。 NOANALYZE の場合、情報を出力しない。
VEXPAND, NOVEXPAND	ループ内の演算回数が少ない場合には、ループ展開により実行速度を向上させることができる。この指定では、コンパイラがループ展開可能でかつ速度向上の効果があると判断したときにベクトル化可能ループを展開する。NOVEXPAND を指定するとループ展開を抑止する。
FVAL, NOFVAL	ベクトル化されたループに単純変数への代入がある場合に、この変数のループ終了時の値を保証する必要があるかどうかを保証する。FVAL を指定したときは保証され、NOFVAL を指定したときは保証されない。
PSETUP, NOPSETUP	多重ループのセットアップ並列化が、可能かどうかを指定する。PSETUP は可能であることを指定する。NOPSETUP は並列化をしない。
SUM, NOSUM	SUM は、総和を求めるベクトル命令の演算順序を、添字の変化の順序と関係なく行う。NOSUM は、添字の変化の順序通り行う。
LOOP(n)	n を最大ループ長と考えてベクトル化処理を行う。指定がないときは、 コンパイラが独自に定める。
VIST, NOVIST	VIST は、S-820 の高速リストベクトル命令のうち、ストア命令を使用したオブジェクトを生成する。 NOVIST は生成しない。(同一インデックスがあると結果が不正になることがある。)
SVOBJ, NOSVOBJ	SVOBJ はベクトル化される最内側ループに対して、プログラム実行時のループ繰り返し回数によってベクトルループとスカラループに振り分けるオブジェクトを生成する。NOSVOBJ は生成しない。プログラム実行時に短ループ (ループ繰り返し回数が 3 未満) になりやすいループに有効である。
I2VEC, NOI2VEC	ループ内の文に対して、整数型 2 バイトおよび論理型 1 バイトの変数を含む文をベクトル化の対象とするかどうかを指定する。ただし、ベクトル化すると逆効果になるとコンパイラが判断した場合にはベクトル化しない。I2VEC はベクトル化の対象とし、NOI2VEC はベクトル化の対象としない。
PARTIAL, NOPARTIAL	PARTIAL は、ループが部分的にベクトル化できる場合、ベクトル化する。NOPARTIAL は、ループ全体がベクトル化できる場合だけベクトル化する。
SVDATA, NOSVDATA	SVDATA は、配列間のデータ依存関係を実行時に判断してベクトル実行とスカラ実行に振り分けるオブジェクトを生成する。NOSVDATA は、そのようなオブジェクトを生成しない。

5.3.1.3 メモリインタリーブ

スーパーコンピュータに限らず、コンピュータの性能を向上させるには、主記憶装置のデータアクセスに要する時間を短くすることが非常に重要である。メインフレーム以上のコンピュータでは、主記憶装置は、2 つ以上の独立したバンクと呼ばれるメモリモジュールから構成され、それぞれ同時にアクセスできるようになっている。たとえば、日立製 S-810 の場合、図 5.8 に示すように、主記憶は 0 から 127 までのバンク (計 128 バンク) で構成されている。この各バンクに実際のメモリが割り付けられているが、その順番はちょうどバンク番号の増加方向となっている。つまり、i 番目のアドレスがバンク 0 に割り付けられていれば、$i+1$ 番目のアドレスはバンク 1 に割り付けられる。また、バンク 127 の次はバンク 0 に戻る。各バンクは、図 5.8 で示すように、0, 8, 16, ..., 120 というように 8 個おきにまとめられ、そこに 1 本のポートが接続され、それぞれ同時にデータを転送することができるようになっている。そのため、連続したアドレスの 8 個のアクセス要求を同時に処理することができる。しかし、あるアクセス要求の次に 8 バイト分だけアドレスの異なるアクセス要求が発生すると、同じポートに対する競合アクセスとなり、同時に処理することがで

きない。これをポートコンフリクトと呼ぶ。また、128バイト分だけアドレスの異なるアクセス要求は、同一バンクのアクセスとなり、これをバンクコンフリクトと呼ぶ。ポートコンフリクトもバンクコンフリクトも、アクセス速度を低下させる要因であるが、特にバンクコンフリクトの場合は、その度合いが著しい。

プログラムとの関係で述べれば、2次元配列の宣言方法によって、バンクコンフリクトが起こることがあり、その場合には、メモリを多少犠牲にして宣言時の寸法をかえる必要がある。

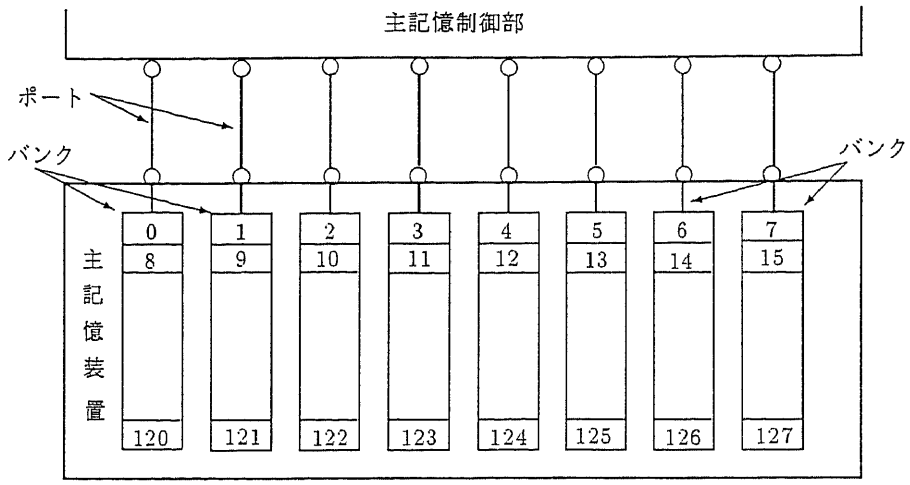


図 5.8: S-810の主記憶装置：文献70ページより一部引用

5.3.1.4 ループアンローリングとパイプラインの並列化

古くから、プログラムの実行速度を向上させるために、ループアンローリングというコーディング手法が用いられてきた。たとえば、LINPACKのソースプログラムなどはその手法が用いられている。この古くから存在する手法をスーパーコンピュータに適用すると、ベクトルプロセッサの演算パイプラインを並列に動作させることができ、スーパーコンピュータにおけるプログラムの実行速度向上に寄与することが知られている。

まず、ループアンローリングについて簡単に説明する。次のプログラムは、100×100の正方行列A, B, Cがあり、A, Bが与えられていて、かつ、Cの内容がすべて0に初期化されている場合の$A \times B$の一般的な(ループアンローリングを使わない)プログラムである。

```
    do 100 k=1, 100
      do 100 j=1, 100
        do 100 i=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

このプログラムをループアンローリングを使うと次のようになる。

```
    do 100 k=1, 119, 2
      do 100 j=1, 100
        do 100 i=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j) + A(i,k+1)*B(k+1,j)
100 continue
```

これは、ベクトルプロセッサの場合、A(i,k)*B(k,j)とA(i,k+1)*B(k+1,j)の2つのベクトル演算が、ベクトル化される最内側のループに現れ、結果的にベクトル演算密度が2倍になったと考えることができる。

さて、上記のプログラムは、さらに次のようにより複雑なループアンローリングを施すこともできる。

```
      do 100 k=1, 97, 3
        do 100 j=1, 100
          do 100 i=1, 100
              C(i,j) = C(i,j) + A(i,k)*B(k,j)
     -                        + A(i,k+1)*B(k+1,j)
     -                        + A(i,k+2)*B(k+2,j)
  100 continue
c     k=100 の時の処理が残ってしまうのでその処理を行う。
      k=100
      do 200 j=1, 100
        do 200 i=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
  200 continue
```

このプログラムでは、外側のステップ刻みを 3 づつにして、最内側ループのベクトル演算密度を 3 倍にしている。ただし、この場合、k=(1,2,3), (4,5,6), ...,(97,98,99) と動くので、k=100 の時の処理が残ってしまう。この処理は別のループで行う必要がある。筆者らは、このステップ刻みのことをループアンローリングのストライドと呼び、第 5.4 節で説明するように、数多くの実験を行った。この実験から、プログラムの実行速度を与えられた条件下で最高にするストライドは、コンピュータによって大きく異なることがわかった。

また、このような DO ループにおいて、添字 i,j,k をどの順序で記述するかということも大きな問題である。このプログラムにおいては、次の 6 つの形式が考えられる。

```
    do 100 k=1, 100
      do 100 j=1, 100
        do 100 i=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

kji 方式

```
    do 100 k=1, 100
      do 100 i=1, 100
        do 100 j=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

kij 方式

```
    do 100 j=1, 100
      do 100 k=1, 100
        do 100 i=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

jki 方式

```
    do 100 j=1, 100
      do 100 i=1, 100
        do 100 k=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

jik 方式

```
    do 100 i=1, 100
      do 100 k=1, 100
        do 100 j=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

ijk 方式

```
    do 100 i=1, 100
      do 100 j=1, 100
        do 100 k=1, 100
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
100 continue
```

ikj 方式

この 6 つの方式のうち、どの記述法が良いのかはコンピュータによって異なる。

5.3.1.5 チューニング支援ツール

多くのベクトル化 FORTRAN コンパイラは、プログラムのベクトル化率を向上させるためのチューニング支援ツールを持つ。これは、コンパイラがベクトル化できなかった DO ループに対して、ベクトル化不

可能な理由やその対策を利用者に示唆するものである。近年、GUI(Graphical User Interface) を用いた会話型のものも登場している。

また、実際にプログラムを実行させ、実データを利用して、ベクトル化効率などを計るシステムもある。

5.3.2 その他のコンピュータのアーキテクチャとそのチューニング

5.3.2.1 メインフレーム

メインフレーム (いわゆる汎用大型コンピュータ) は、前節で説明したスーパーコンピュータの基礎となっているので、ベクトル処理以外の点は、かなり参考となる。たとえば、メモリのバンク構成や機械語命令のパイプライン処理などがあげられる。

しかし、筆者らの調査では、その特性が必ずしもスーパーコンピュータと一致しない点もあり、独自の解析が必要であることを示している。

さらに、メインフレームの中には、**IAP**(Internal Array Processor) と呼ばれるベクトル演算を処理する特殊な装置を付加しているものもある。これは、FORTRAN コンパイラに IAP を使用する旨をコンパイラオプションによって与えて、プログラムをコンパイルする事によってユーザが利用できる[†16]。IAP を効率よく使うにも、そのためのチューニングが必要となる。

また、メインフレーム上の FORTRAN コンパイラの最適化機能はかなり充実しており、これを十分効果的に利用するようにする必要がある。

5.3.2.2 ワークステーション

近年のワークステーションは、**RISC 技術**により、プログラムの高速化を計っており、いわゆる MIPS[†17]をベースとしたコストパフォーマンスの良さは目をみはるものがある。RISC プロセッサの場合、CPU の実行速度よりもメモリのアクセス速度が相対的に遅くなるため、極力、データ移動を伴わないようにアルゴリズムやコーディングを行う必要がある。

また、キャッシュの効果も大きいため、キャッシュをうまく利用するようなチューニングが必要である。

さらに、コンパイラの性能も重要である。RISC プロセッサの場合、その命令セットアーキテクチャを有効に活かすようなコードを生成することが大切であるが、それがコンパイラによってどこまで達成されているかが重要である。コンパイラの特性も十分に実行効率に影響を与える。

また、CONVEX のように、マルチプロセッサ構成のパラレル演算を行う機能を持っていて、プログラムのプロセスを複数のプロセッサに分散させて実行させるタイプのコンピュータもある。このようなコンピュータに対するチューニングは、一般のコンピュータに対する方法とかなり異なることが筆者らの調査で分かっている。

5.3.2.3 パーソナルコンピュータ

LAMAX-S がパーソナルコンピュータで使われる多くの場面は、教育の分野ではないかと考えられるので、その場合、大量データの高速処理が要求されることは少ないかも知れない。しかし、最近のパーソナルコンピュータは、性能が飛躍的に向上し、少し前のワークステーションの性能に遜色劣らない。パーソナルコンピュータにおいても効率よく動作させることは重要なことである。

パーソナルコンピュータの場合、MS-DOS などオペレーティングシステムによるメモリの制約があったが、これは Windows や OS/2 などの普及により解消されてゆくであろう。

[†16] 国産メーカの多くは、スーパーコンピュータやメインフレームの IAP などの機械語の詳細を公表していないので、直接アセンブリ言語などで IAP を操作するプログラムを書くことは一般ユーザには難しい。

[†17] MIPS(Mega Instruction Per Second) は、1 秒間に 10^6 回命令を実行する単位である。最近のワークステーションでは 100MIPS 以上の性能が一般的である。

スーパーコンピュータやメインフレーム (あるいはワークステーション) では、FORTRAN コンパイラの最適化機能をかなり利用することができた。残念ながらパーソナルコンピュータ用の FORTRAN コンパイラの最適化機能は、それらを凌ぐものはほとんどなく、スーパーコンピュータなどでは、コンパイラの最適化機能に頼っていた部分を、LAMAX-S プリプロセッサ側で処理しなければならない。

また、パーソナルコンピュータの場合、数値演算プロセッサの有無、内蔵あるいは外付けの高速数値演算アクセラレータ (たとえば、トランスピュータやインテル i860) の有無、キャッシュの有無、メモリの実装状況などもプログラムの実行効率に影響を与える。

5.4 実存コンピュータの特性解析システム:ATTPG

5.4.1 実存コンピュータの特性解析システムの必要性

前節で説明したように、コンピュータのアーキテクチャに合わせてプログラムのコーディングの仕方を変更する作業をチューニングと呼び、これがプログラムの実行効率に大きく影響を及ぼす。しかし、このチューニングの方法は、コンピュータのアーキテクチャ並びにそのコンピュータのハードウェア構成に大きく依存する。たとえば、ハードウェア的な面からは、CPU のアーキテクチャ(たとえば、RISC か CISC か、レジスタの個数、IAP の有無、ベクトルプロセッサかどうか、などなど)、主記憶装置の容量、メモリの構成 (バンクの個数など)、キャッシュの有無あるいは容量、などがある。ソフトウェアの面からは、オペレーティングシステムの種類 (マルチタスクかシングルタスクか)、コンパイラの最適化の度合い、ライブラリの種類、などがある。

プログラムをチューニングしようとするとこれらの点を総合的に考慮する必要がある。しかし、コンピュータのこのようなさまざまな要因を、理論的に捉えてモデル化することはある程度は可能であっても、コンパイラの最適化特性などを考慮に入れた場合、そのモデル化は非常に困難であると考えられる。しかし、LAMAX-S のコード生成を考えると、FORTRAN プログラムを生成する最終段階では、数多くの DO ループを生成しなければならず、LAMAX-S コンパイラが、ターゲットコンピュータに合わせてチューニングを施すという処理がどうしても必要となる。そのためには、ターゲットコンピュータのハードウェア・ソフトウェアの特性をどうしても知る必要がある。

そのために、本研究では、実在コンピュータに対して、その特性を解析するシステム **ATTPG**(Automatic Tuning Test Program Generator) を作成した。これは、さまざまな手法のサンプルプログラムを自動的に生成し、それを実際にターゲットコンピュータ上で流して、その動作特性を把握するというものである[†18]。これは、非常に簡単な方法ではあるが、実際の確実性が高い方法である。

5.4.2 実存コンピュータの特性解析システムの概要

ATTPG のシステム概要を図 5.9 に示す。

ATTPG は、調査項目スペックから、計測に必要な情報を得て、場合によっては、チューニングの対象となるサンプルプログラム (ベースプログラム) をベースにしてテストプログラムを生成する。これを計測対象となるターゲットコンピュータ上で実際に動作させる。この実行結果は、実行時間の数値の羅列であるので、これをもとにデータベースとして整理しやすい形式に変換する。

ATTPG で調査できる対象にはさまざまなものがあるが、スーパーコンピュータに関する代表的なものに、表 5.11 に示す項目がある。これらの項目は、マニュアルなどの文献によれば、コンパイラがある程度処理してくれるものもある。本項では、本システムを具体的に示すために、表 5.11 の各項目を 5.4.2.1 から 5.4.2.6 において説明する。

表 5.11 の項目は主にスーパーコンピュータに対するものであるが、ループアンローリングなどは、パーソナルコンピュータでも十分に効果がある。

[†18]実際には、これは一つのプログラムではなく、小規模の数多くのプログラムの集合体から構成される。

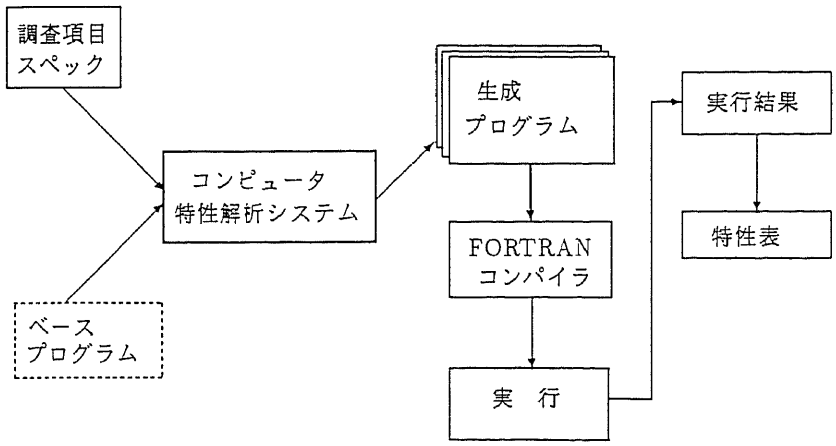


図 5.9: ATTPG の概要

表 5.11: ATTPG の主な調査項目 (スーパーコンピュータに関するもの)

項 目	説 明
ループアンローリング	DO ループに対して、さまざまなループアンローリングのテストを行う。最適ストライドの調査、kij 方式・ijk 方式の調査などが行われる。
作業用ベクトルの利用	プログラムを記述する際に、中間的な演算結果を作業ベクトルに保持した方が効果的かどうかテストする。
リストベクトルの利用	リストベクトルを利用した方が効果的かどうかテストする。
DO ループの並列性	DO ループの中に多くの独立した代入文を入れた方が効果的かどうかテストする。
アンローリングとリストベクトル	ループアンローリングとリストベクトルの組み合わせの効果を調べる。

5.4.2.1 ループアンローリング

ループアンローリングについては、5.3.1.4 項で詳しく述べた。この調査項目としては、ループアンローリングのストライド、DO ループの方式 (ijk 方式, kji 方式, など) を調べる。

5.4.2.2 作業ベクトルの利用

中間結果を作業配列に入れた方が良いかどうかのテストである。たとえば、次のプログラムを例に取る (このプログラムを「作業ベクトル未使用」とする)。

```
      do 10 k=1, 100
        do 10 j=1, 100
          do 10 i=1, 100
            A(i,j) = A(i,j) + ( C(i,k)+D(i,k) ) * E(i,j)
   10 continue
```

プログラム：作業ベクトル未使用

ここで、C(i,k)+D(i,k) の部分が、重複して実行されることに注意されたい。つまり、次のように先に計算しておいた方が全体の計算量が減ることは明確である (このプログラムを「作業ベクトル使用」とする)。

```
      do  5 k=1, 100
        do 5 i=1, 100
            W(i,k) = C(i,k) + D(i,k)
    5 continue
      do 10 k=1, 100
        do 10 j=1, 100
          do 10 i=1, 100
            A(i,j) = A(i,j) + W(i,k) * E(i,j)
   10 continue
```

プログラム：作業ベクトル使用

計算量から判断すれば、「作業ベクトル未使用」の方が効率が悪いように思えるが、スーパーコンピュータの場合、DO ループ内部のベクトル化の対象を多くした方がコンパイラがより効果的にベクトルレジスタやベクトル演算等のスケジューリングを行えるかもしれない。この調査では、そのような点を調べる。

5.4.2.3 リストベクトルの利用

リストベクトルとは、他の配列のインデックスが格納された整数配列を添字とすることにより間接的に演算を行うことである。たとえば、A(1), A(3), A(7), A(11), A(21) にデータをセットしたい場合、次のようにプログラムを記述することができる。

```
      integer INDEX(5)/1, 3, 7, 11, 21/
      real*8  A(5)
c
      do 10 i=1, 5
         A( INDEX(i) ) = 0.0d0
  10  continue
```

この間接的な添字の指定により、たとえば、さまざまなチューニングが考えられる。この配列 INDEX をリストベクトルと呼ぶ。たとえば、次のプログラムについて考えよう。

```
      do 10 k=1, 100
        do 10 j=1, 100
          do 10 i=1, 100
            A(i,j) = A(i,j) + B(i,k)*C(k,j)
10 continue
```

これは、3重のDOループとなっている。最内側DOループのベクトル長は100であるが、ベクトルプロセッサの場合は、ベクトル長が長いほどチューニングの効果がある。そこで、この観点からこれをチューニングしてみよう。最内側ループにおいて、添字j，iがどのように動くか考えてみると、異なるn^2通りの対が一つづつ現れる。そこで、次のような2つのリストベクトルを用意する。

```
      integer Li(10000), Lj(10000)
```

このリストベクトルに次の値をセットする。ただし、$N = 100$とする。

	1	2	3	$\cdots$	N	$N+1$	$N+2$	$N+3$	$\cdots$	$2N-1$	$2N$	$2N+1$	$2N+2$	$\cdots$	N^2-1	N^2
Li	1	2	3	$\cdots$	N	1	2	3	$\cdots$	$N-1$	N	1	2	$\cdots$	$N-1$	N
Lj	1	2	3	$\cdots$	N	2	3	4	$\cdots$	N	1	3	4	$\cdots$	$N-2$	$N-1$

Ljの値が少しづつずれているのは、バンクコンフリクトを避けるためである。このリストベクトルを使うと、先ほどのプログラムは次のようになり、最内側ループにおける繰り返し数が10000となる。

```
      do 10 k=1, 100
        do 10 L=1, 10000
          A(Li(L),Lj(L)) = A(Li(L),Lj(L)) + B(Li(L),k)*C(k,Lj(L))
10 continue
```

スカラコンピュータでは、上記のプログラムは余計な処理が入るので遅くなるが、スーパーコンピュータの場合には、リストベクトルをベクトルプロセッサで処理できるので、ベクトル長が長くなったことと合わせて効果がある。ただし、多くのFORTRAN最適化コンパイラでは、上記のプログラムにおいて、配列Aの定義・引用関係が正しいかどうか判断できないので、このままではベクトル化できない。そこで、次のように強制ベクトル化指示行を挿入するのが普通である。

```
      do 10 k=1, 100
       (強制ベクトル化指示行)
        do 10 L=1, 10000
          A(Li(L),Lj(L)) = A(Li(L),Lj(L)) + B(Li(L),k)*C(k,Lj(L))
10 continue
```

5.4.2.4 DOループの並列性

スーパーコンピュータ用コンパイラの多くは、一つのDOループの中に複数の代入文がある場合、そのコンピュータが持つベクトル演算器を最大限に並列に動作させて効率化を計るようなコードを生成する。

たとえば、次のプログラムは、2つのDOループが独立して記述されている。

```
      do 10 j=1, 100
        do 10 i=1, 100
          A(i,j) = B(i,j) + C(i,j)
10 continue
      do 20 j=1, 100
        do 20 i=1, 100
          X(i,j) = Y(i,j) + Z(i,j)
20 continue
```

これは、DO ループの制御変数の指定が同じ DO ループであるので、次のようにしても構わない。

```
      do 10 j=1, 100
        do 10 i=1, 100
           A(i,j) = B(i,j) + C(i,j)
           X(i,j) = Y(i,j) + Z(i,j)
   10 continue
```

このようにすることで、ベクトル型コンピュータの場合、ベクトル演算器が並列に動作し、効率がよくなる可能性がある。また、スカラコンピュータでも、DO ループの制御に時間がかかる場合、効率が良くなる可能性もある。

5.4.2.5 アンローリングとリストベクトル

ループアンローリングとリストベクトルのテクニックを同時に行う。いま、次のプログラムがあるとする。

```
      do 10 k=1, 100
        do 10 j=1, 100
          do 100 i=1, 100
             A(i,j) = A(i,j) + B(i,k)*C(k,j)
   100 continue
```

これにリストベクトルを利用した形式にすると次のようになる。

```
      do 10 k=1, 100
        do 10 L=1, 10000
          A(Li(L),Lj(L)) = A(Li(L),Lj(L)) + B(Li(L),k)*C(k,Lj(L))
   100 continue
```

さらにこれにループアンローリングを施す。

```
      do 10 k=1, 99, 2
        do 10 L=1, 10000
          A(Li(L),Lj(L)) = A(Li(L),Lj(L))
     -                     + B(Li(L),k  )*C(k  ,Lj(L))
     -                     + B(Li(L),k+1)*C(k+1,Lj(L))
   100 continue
```

このプログラムの効率を知ることによって、リストベクトルとループアンローリングの相乗効果を知ることができる。

5.4.2.6 その他の調査項目

この他に連続ベクトル・等間隔ベクトル・バンクコンフリクトの影響などを調べる。

5.4.3 実存コンピュータの特性解析システムの結果

本研究では、ATTPG を表 5.12 に示すさまざまなコンピュータに対して行った。だたし、これらの全てに対して、すべての項目をテストしたわけではなく、また、この調査はこの 7 年間の間に行ったものを含むので、現在、リプレースなどのため利用不可能なコンピュータも含まれている。本項では、参考のために、ループアンローリングに関してこの結果の一部を示す。

表 5.12: ATTPG の調査対象コンピュータ

対象コンピュータ	説 明
HITAC S-820/80	東大大型計算機センターに設置された日立製作所製のスーパーコンピュータ。
FACOM VP-30	神奈川大学に設置された富士通製のスーパーコンピュータ。
NEC SX-1EA	青山学院大学に設置された日本電気製のスーパーコンピュータ。
HITAC M880/310	東大大型計算センターに設置された日立製作所製のメインフレーム。
FACOM M-770/10	神奈川大学に設置された富士通製のメインフレーム。
CONVEX (C220 等)	某メーカおよび神奈川大学に設置された CONVEX 社製のコンピュータ。
SUN	神奈川大学に設置された SUN 製の EWS である (複数の機種)。
NEC PC-9801RA	神奈川大学に設置された日本電気製のパーソナルコンピュータ。

5.4.3.1 スーパーコンピュータの場合

まず、最初に S-820/80 について、倍精度実数型の行列の乗算に対する kji 方式と jki 方式のどちらが良いか調べてみた[†19]。この結果を、図 5.10 に示す[†20]。図 5.10 の縦軸の単位は実行時間 (秒) である[†21]。したがって、下にいくほど効率がよいことを示す。横軸はストライド[†22]である。ストライドが 1 の場合、アンローリングを行わないことを示す。

この結果からストライドが 5 のときだけ、jki 方式が有利であるが、ほとんどの場合、kji 方式が有利であることがわかった。

次に、kji 方式の場合、寸法によって効果に違いがあるかどうか調べた。図 5.11 にその結果を示す[†23]。

この結果から、アンローリングをまったく施さない場合に比べて、どの寸法 (50×50、51×51、52×52、53×53) でも、半分程度に実行時間が短縮していることがわかる。しかし、最適なストライドは、この結果からは、だいたいの傾向が同じであるが、寸法によって多少異なることが分かる。しかし、その違いはひじょうに小さく、この例ではストライドが 4 程度で十分であると思われる。

今度は、整数型の行列の乗算に対するアンローリングの効果について調べてみる。図 5.12 に、S-820/80 における整数型の行列の乗算に対するアンローリングの効果を示す[†24]。この場合も、同様に kji 方式が有利であるが、ストライドが 3 から 6 程度のときに、効率が多少悪くなるので注意が必要となる。この場合、ストライドとしては、2 あるいは 8 程度がよい。

それでは、まったく別のスーパーコンピュータを見てみることにする。次の図 5.13 は、SX-1EA に対するもの[†25]であるが、ストライドの効果はひじょうになめらかになっている。

これらの例からも、アンローリングの効果はスーパーコンピュータによってかなり異なることがわかる。

5.4.3.2 メインフレームの場合

メインフレームとして、M-880/310 について調べる。100×100 の倍精度実数型の行列の乗算について、kji 方式と jki 方式について調べてみた結果を図 5.14 に示す。この場合、kji 方式でも jki 方式でも最適なストライド数は、10 である。しかし、全体的な傾向としては、jki 方式の方が有利であることが分かった。

[†19] i を最内側ループに置かないと、ベクトル長が長くならないので、i は必ず最後に置く必要がある。
[†20] 寸法は 100×100 である。
[†21] 図 5.10 から図 5.17 の縦軸の単位はすべて秒である。
[†22] 図中で「分割数」と記されているのは、ストライドのことである。
[†23] 要素の型は、倍精度実数型である。
[†24] 寸法は、100×100 である。
[†25] 行列の要素の型は倍精度実数型、kji 方式。

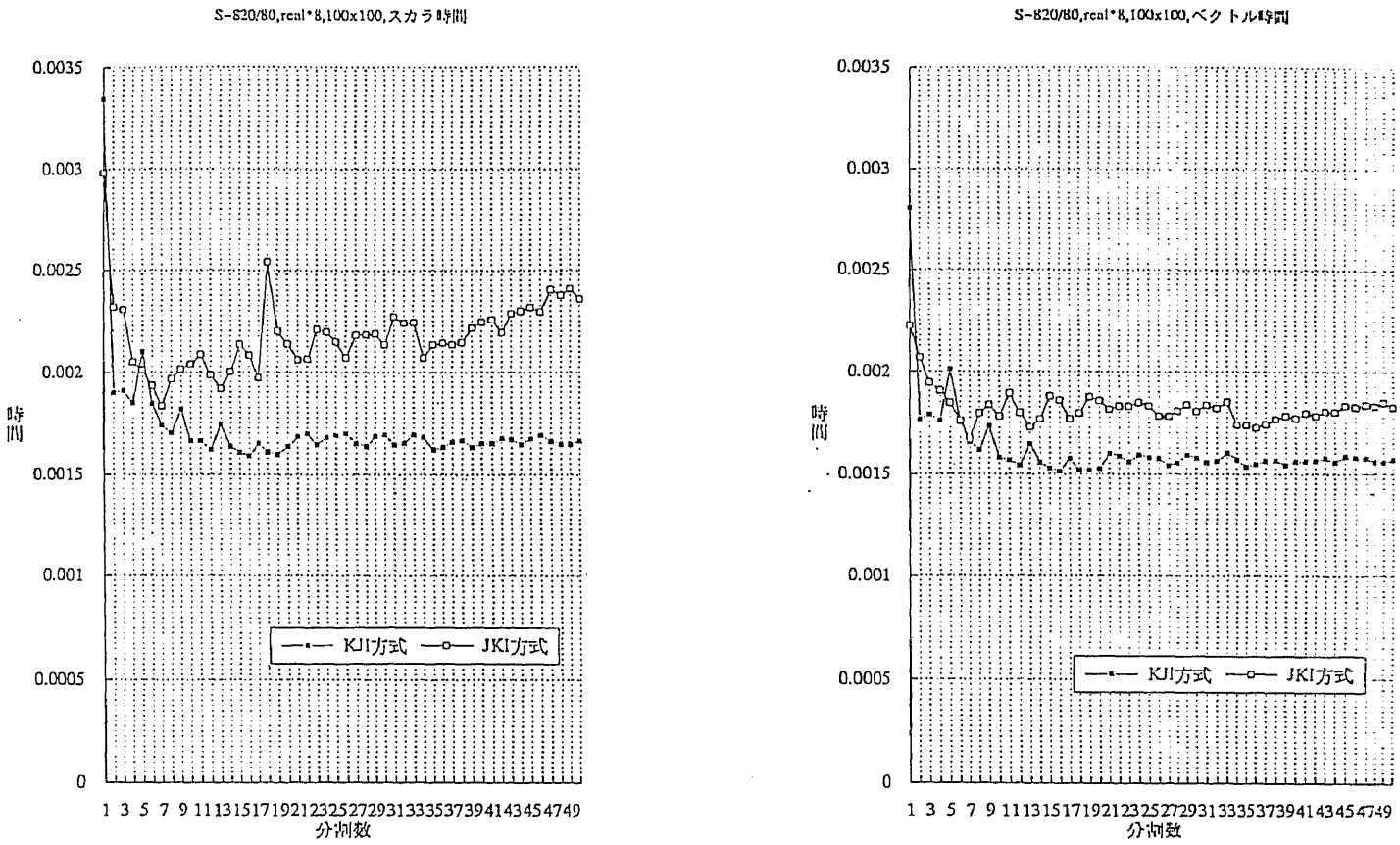


図 5.10: S-820/80 の調査：kji 方式と jki 方式 (倍精度実数型；左：スカラ時間、右：ベクトル)

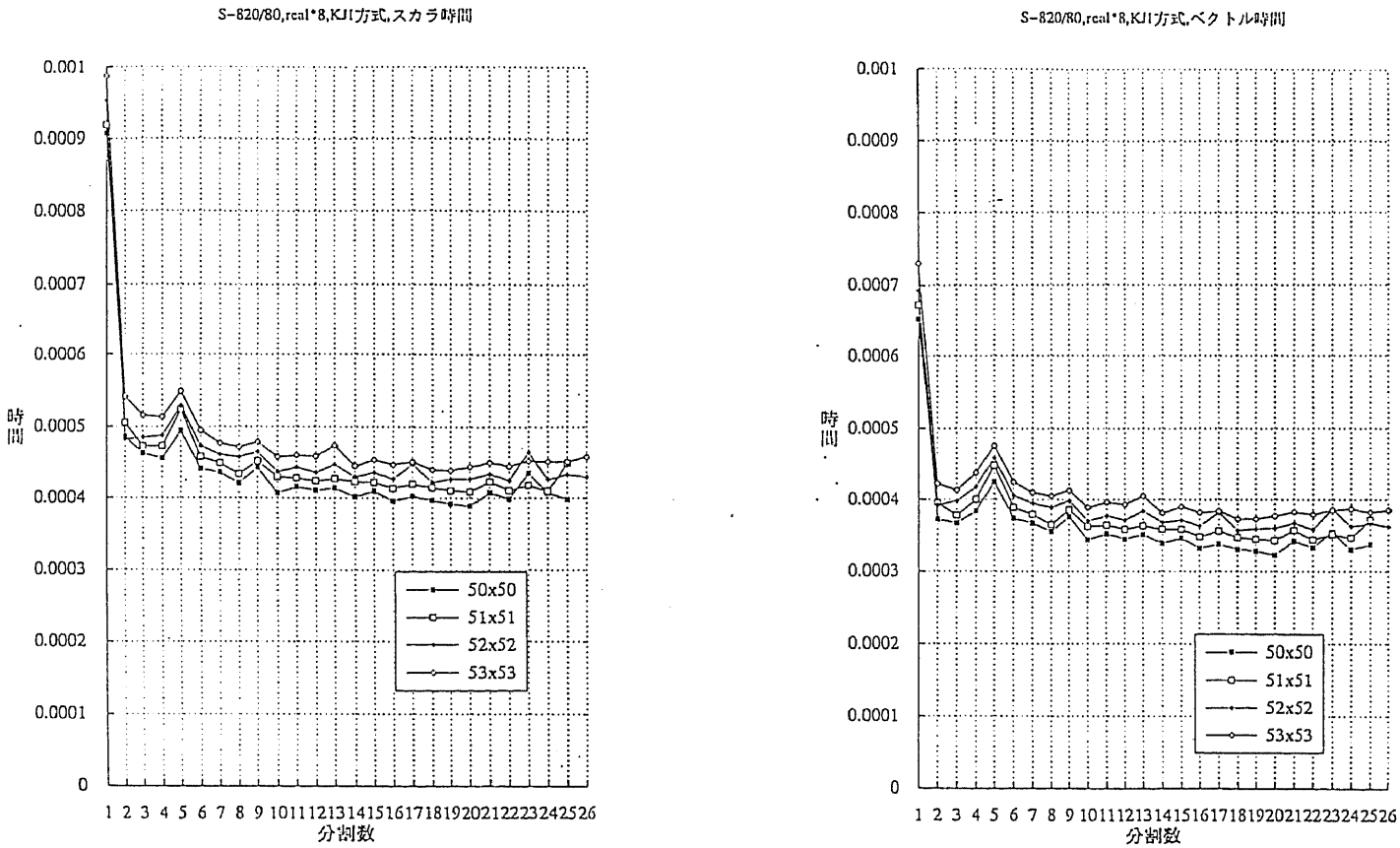


図 5.11: S-820/80 の調査：kji 方式の寸法による違い (倍精度実数型；左：スカラ時間、右：ベクトル)

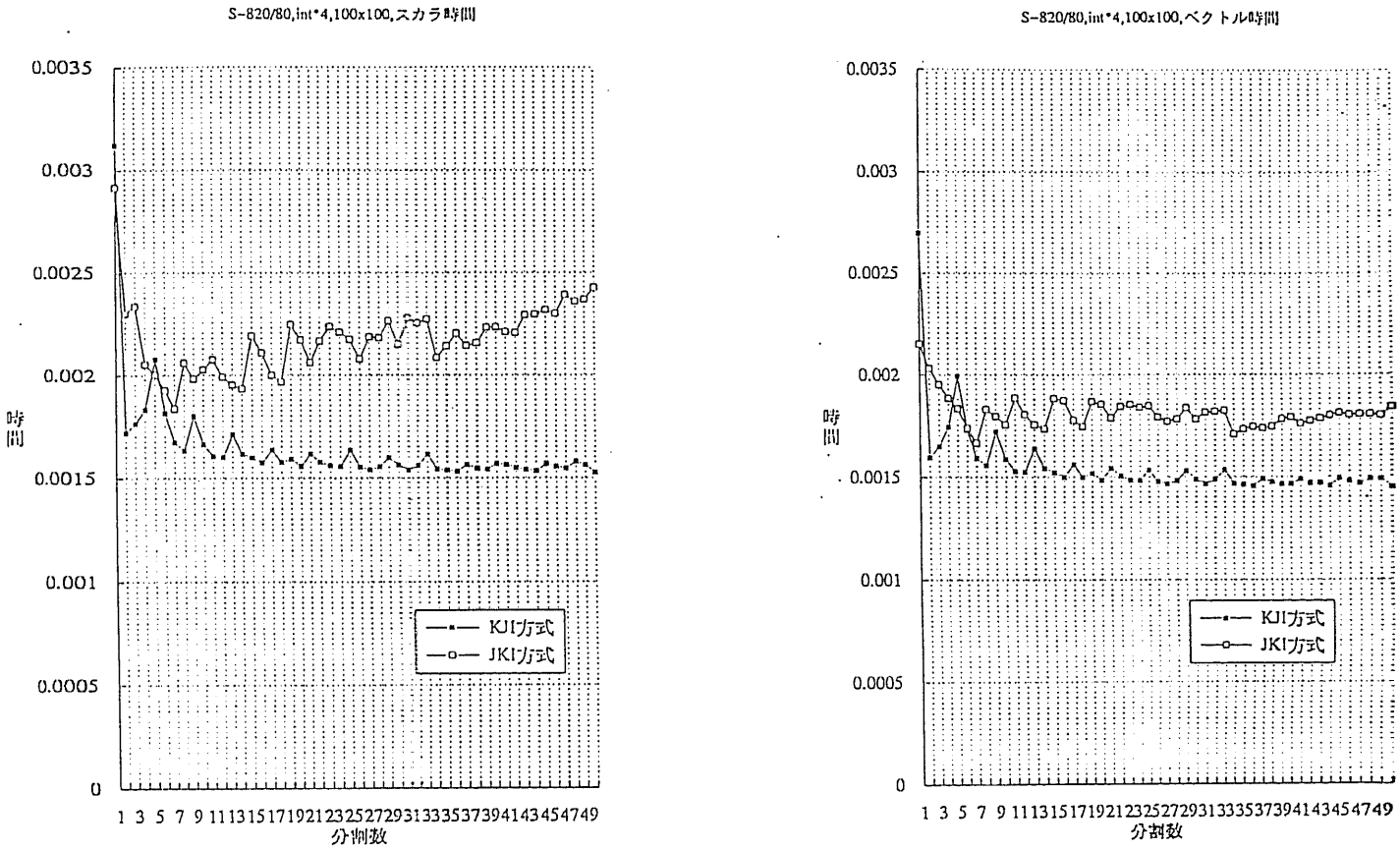


図 5.12: S-820/80 の調査：kji 方式の整数型の調査 (整数型：左：スカラ時間、右：ベクトル)

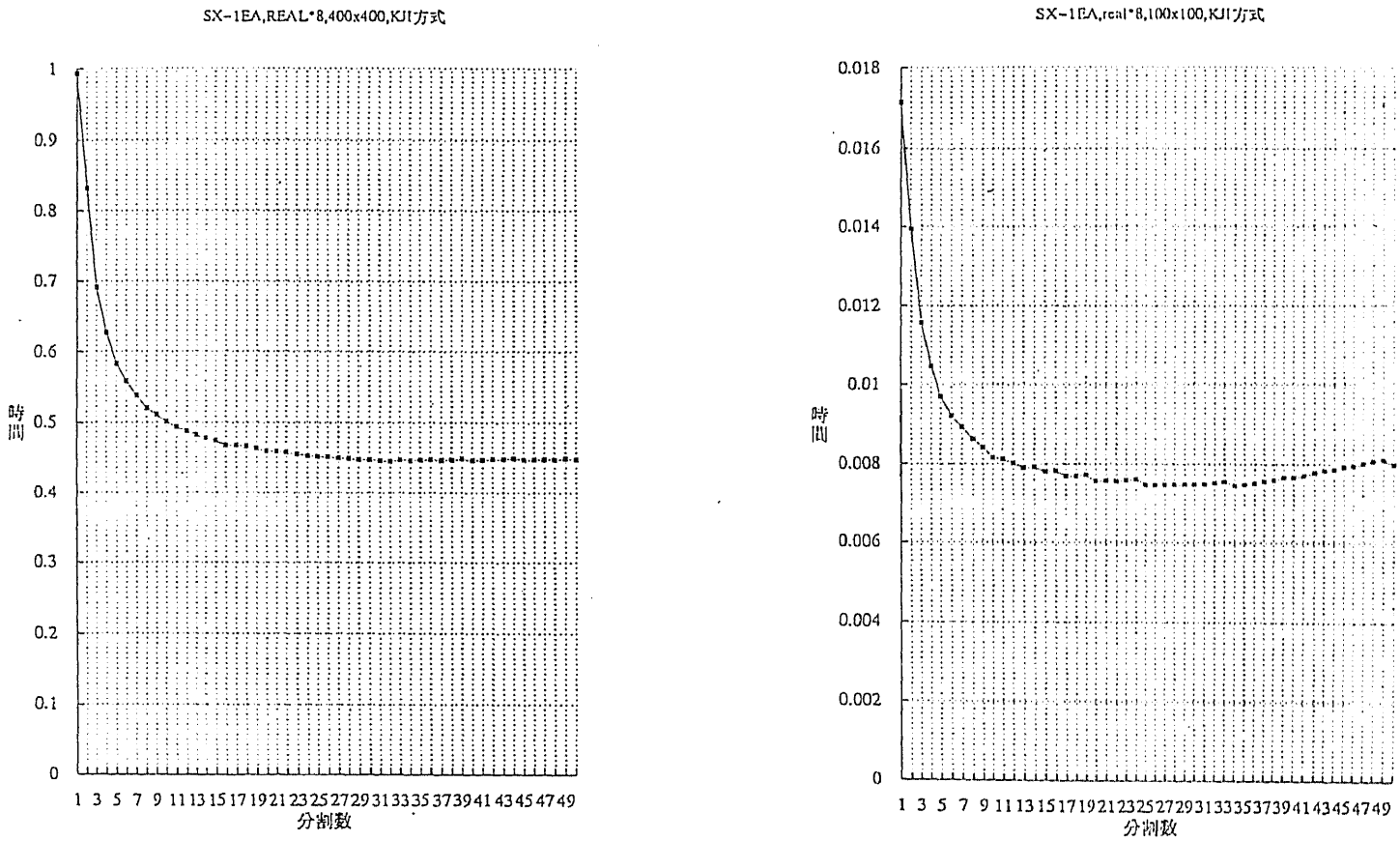


図 5.13: SX-1EA の調査：kji 方式 (倍精度実数型；寸法：左 (400 × 400)、右 (100 × 100))

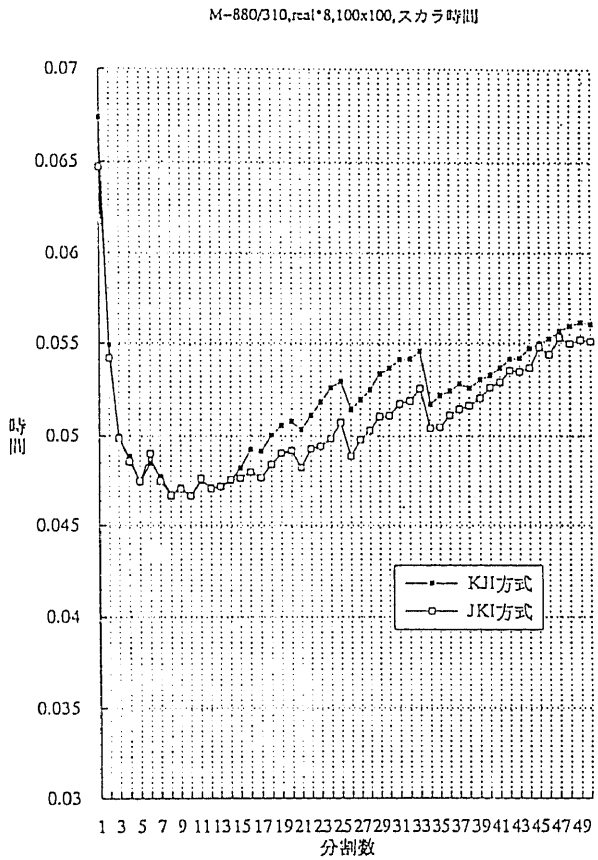


図 5.14: M-880/310 の調査：kji 方式と jki 方式；倍精度実数型

5.4.3.3 ワークステーションの場合

CONVEX(C220) 上で調べる。200 × 200 と 100 × 100 の場合について倍精度実数型の行列の乗算 (kji 方式) を調べてみた結果を図 5.15 に示す。この場合、アンローリングがまったく効果がないことが分かった。

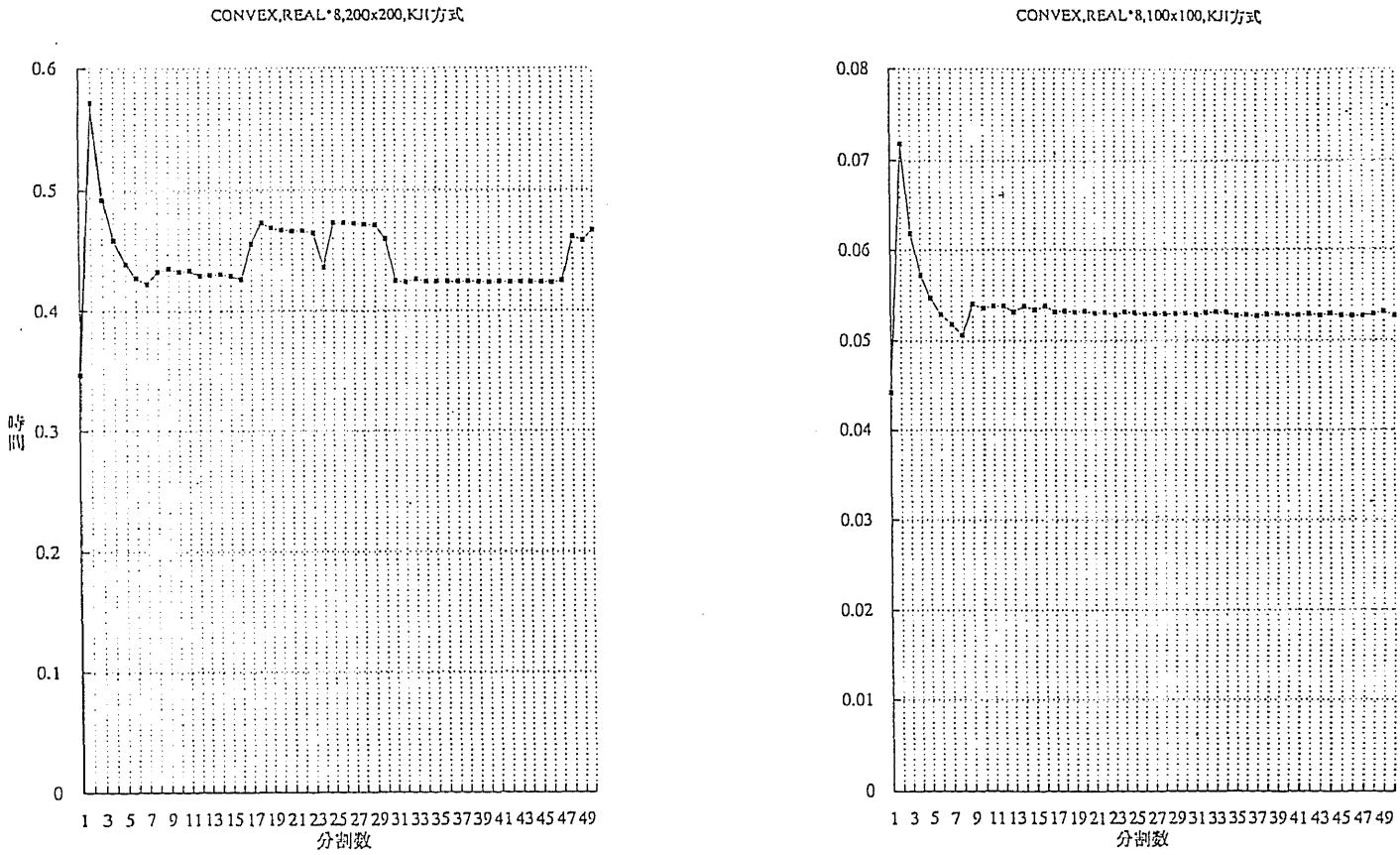


図 5.15: CONVEX の調査 : kji 方式 (倍精度実数型 ; 寸法 : 左 (200 × 200)、右 (100 × 100))

5.4.3.4 パーソナルコンピュータの場合

パーソナルコンピュータとして、PC-9801(RA) †26 について調べる。20 × 20、40 × 40 の倍精度実数型の行列の乗算について、kji 方式と jki 方式について調べてみた結果を図 5.16 に示す。この場合、両者にはほとんど差がないことが分かった。またストライドとしては、20 × 20 のときは 2 が、40 × 40 のときは 3 が、効果的であることが分かった。

パーソナルコンピュータでアンローリングが効果があるのは、レジスタとメモリとのアクセス回数が減少するなどの効果があるからである。そこで、40 × 40 の整数型 (integer*2) の場合についても調べてみた (図 5.17 参照)。

この場合、ストライドは 3 が最も効果的であった。

†26 インテル 80386、数値演算コプロセッサなし。

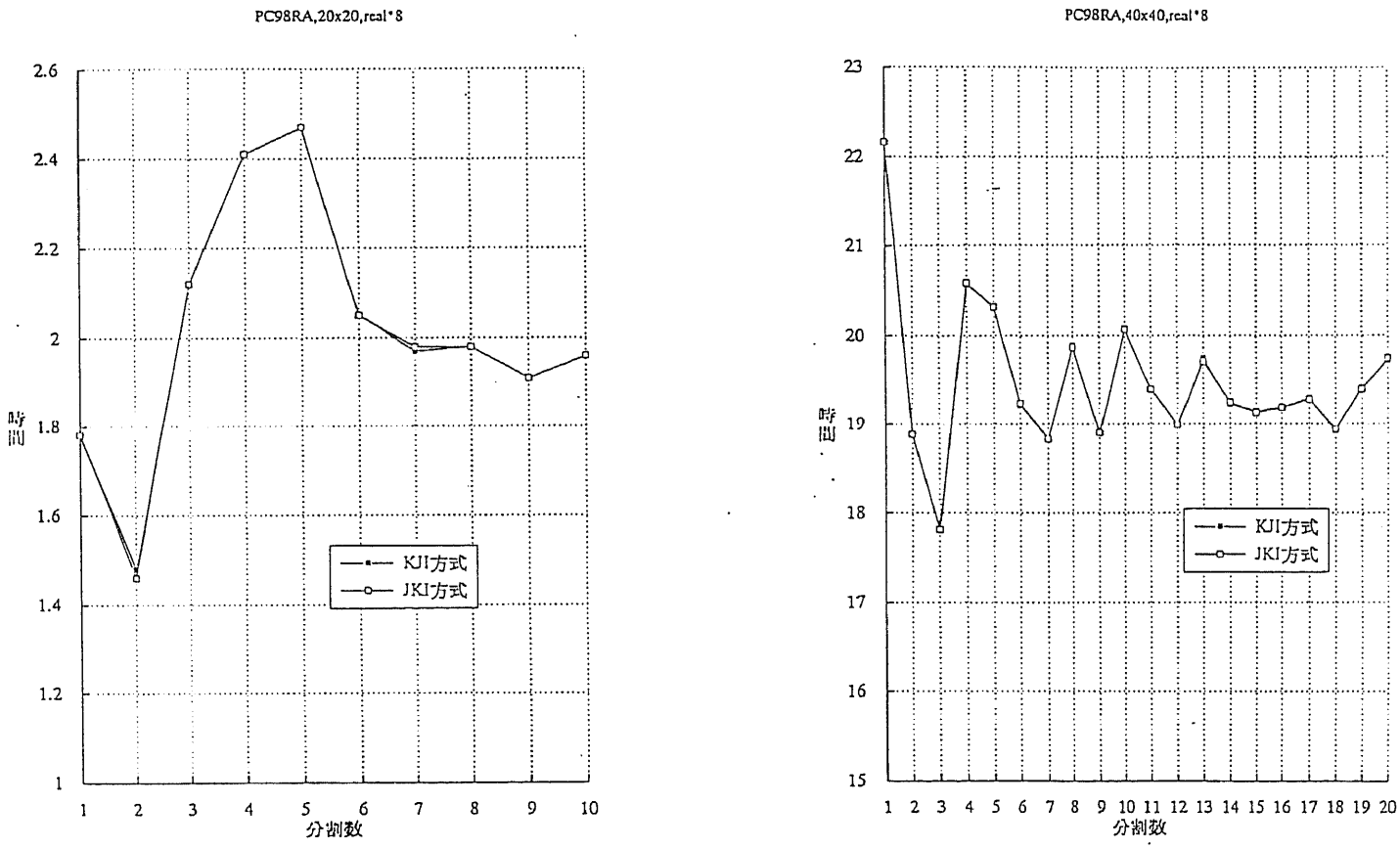


図 5.16: PC9801(RA) の調査：kji 方式と jki 方式 (倍精度実数型；寸法：左 (20 × 20)、右 (40 × 40))

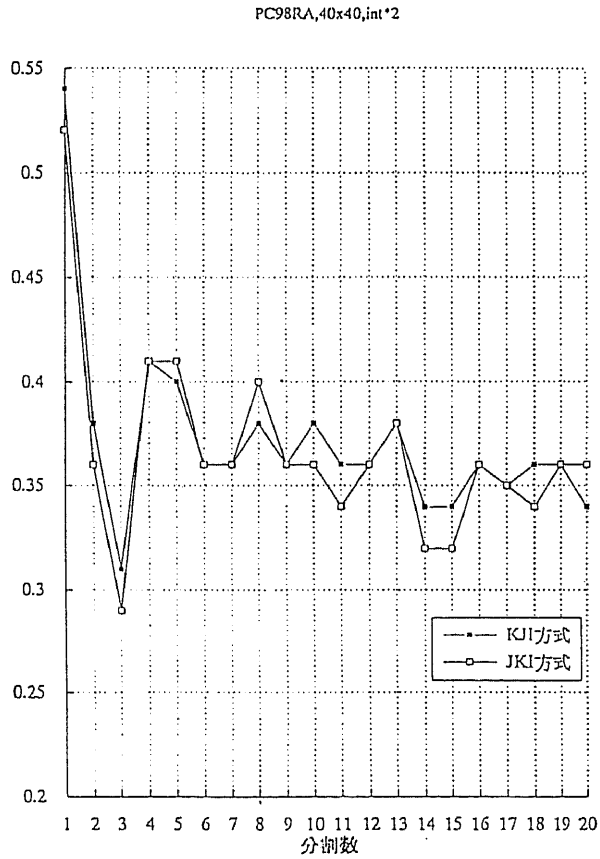


図 5.17: PC9801(RA) の調査：kji 方式と jki 方式 (整数型 (integer*2)；寸法：40 × 40)

5.4.3.5 特性解析システムの結果のまとめ

以上、ATTPG を用いてさまざまなコンピュータに対してアンローリングの効果を調べてきた。その結果、予想されたことではあるが、コンピュータによってまったく効果が異なることがわかった。機種ごとに、このような調査を行うことが大切であるばかりでなく、たとえば、同一モデルであっても、設置されている場所によって、主記憶装置の容量やキャッシュの構成などが異なるであろうと考えられるので、設置場所ごとにこのような調査が必要であろうと思われる。

5.5 本章の結論と今後の課題

本章では、LAMAX-S が生成する FORTRAN プログラムをチューニングするための筆者の提案について、その基本的な考え方を述べた。要約すれば、LAMAX-S が生成するであろうプログラムの雛形がプログラムフレームとして存在しているので、それをベースに人間が考えられるだけのさまざまな形式のチューニングを施したテストプログラムを自動生成し、実際のターゲットコンピュータ上でさまざまな条件下 (行列の要素の型・寸法) で動作させ、最適なチューニング手法を選択し、さらに場合によってはチューニングパラメタ (たとえば、ループアンローリング時のストライドなど) を取得し、簡単なデータベースを作成する。LAMAX-S プリプロセッサは、このデータベースと前述のプログラムフレームをベースにしてチューニングされた FORTRAN プログラムを生成する。

また、多種多様な構造を持つ行列の演算のプログラムフレームの記述量を減らすために、ベクトルフレームを用意し、さらにベクトルレベルに対するチューニングを施す方法を考案した。

さらに、最近では、たとえば、LU 分解にしても、さまざまなアーキテクチャのコンピュータに対して、さまざまな手法 (たとえば、ベクトルアルゴリズムやブロック化アルゴリズムなど) が考案されており、これらを自然な形式で導入することが望まれる。

ここで述べた方法に関する研究はすべて終了し、現在 (1995 年 5 月) は、(株) システム計画研究所と筆者らによって LAMAX-S へのこの研究成果の組込みが行われている段階であり、現時点では、その有効性が確実に検証されたわけでない。しかし、ATTPG によってこの 7 年間に渡って計測されたチューニング情報は、多種多様なものを含んでおり、この情報だけでもかなり価値があると思われる。また、本章で述べた処理は、コンパイラの処理技術としては単純なものであり、LAMAX-S の組込みに関して、技術的には比較的易しいものである。

今後の課題を次に 2 点ほど示す。

実機による測定には、多大の計算時間が必要となる。このためにいくつかの弊害がある。ひとつはコストの問題である。スーパーコンピュータの測定で東大大型計算機センターなど有料の計算センターを用いたが、1 回の測定で数千円以上の予算が必要であった。この測定は、神奈川大学工学部工業経営工学科 (当時、現在は経営工学科) の卒業研究の学生に卒業研究として行ってもらい、その結果はすべて京都大学数値解析研究所講究録に記載されていて、純粋に学術的なものであった。しかし、これを、たとえば商用ベースで行う場合、当然コストが安いとしても国立大学大型コンピュータセンターのスーパーコンピュータを使うわけにはいかない。企業が商用ベースでチューニングデータベースを作成するには、かなりのコストがかかってしまうと考えられる。

もう一つは、この種の測定では CPU をかなり占有するので、他のユーザの迷惑になることがあることである。神奈川大学で CONVEX の測定を行った際には、昼間測定すると他のユーザに迷惑がかかるとセンター側から指摘され、深夜 (午前 1:00 頃から午前 4:00 まで) に行わざるを得なかった。

いずれにしても、この方法は、ターゲットコンピュータを占有して使える状況を仮定している。この状況が作り出せないと困難な方式であるという欠点を持つ。これらの点の克服が、今後の課題である。

最後になりましたが、ATTPG の作成と実際の計測は、筆者の近傍にいた青山学院大学理工学部経営工学科および神奈川大学工学部経営工学科の卒業研究の学生に頼るところが大であり、ここに感謝の意を表したいと思います。

第 6 章 LAMAX-S と行列オブジェクト

6.1 はじめに

ミニ LAMAX および LAMAX-E では、固有値問題以外の行列計算[†1]は、自作のライブラリを用いていた。一方、LAMAX-S においては、行列計算は、数値計算の世界で代表的なライブラリを任意に選んで利用できることを目標としている。このようなライブラリには、LINPACK, EISPACK, LAPACK, ISML, MATRIX/HAP, ASL などがある。このように任意のライブラリを呼び出せるようにすることは、近年、さかんに主張されているオープン化という観点からも重要なことである。

LAMAX-S では、ここで述べたように実存するライブラリを利用することを仮定している。しかし、本章では、LAMAX-S の処理系の最適化の研究の一貫として、このライブラリに対しても研究のメスを入れる。ただし、本研究では、数学上あるいは数値解析的な手法の研究ではなく、行列計算ライブラリの行列データの取扱い (データ構造の表現、データの格納法、計算のタイミングなど) についてのみ考察する。多くのライブラリでは、数学上あるいは数値解析上の問題を正しく解決することを主目的としているので、行列データの取扱いについてはあまり考察されていない。もちろん、LAPACK のようにスーパーコンピュータやパラレルコンピュータに対してチューニングを施しているライブラリも登場しているが、行列データの扱いに重点を置いているわけではない。。

本研究では、オブジェクト指向を導入することにより、柔軟性の高い「行列データの管理システム」を構築することを目的とする。基本的な考え方を述べる。現在の多くのライブラリでは、たとえば連立一次方程式を解く際の三角分解値やピボッテイングの情報は、利用者自ら管理することになっている。さらに、その時系列的な処理手順も利用者が管理する必要がある。たとえば、前述の連立一次方程式の求解では、「(1) 三角分解をして、(2) その結果を利用して解を求める」という手順になっているが、これは正しくこの順番で利用者が記述しなければならない。本研究では、オブジェクト指向のメカニズムを利用することによって、このような手順を表現することを試みる。さらに、行列データをオブジェクトとして管理することで多くの利点も生まれる。たとえば、すでに固有値の求められている行列オブジェクトに、その行列式の値を求める必要が生じた場合、すでに求められている固有値を利用して行列式の値を求めるといった処理が可能になる。この処理は、利用者が指示するのではなく、ライブラリの中の管理機構によって自動的に行う。このような処理も、オブジェクト指向では比較的簡単に実現できる。この考え方を筆者は、行列フォルダモデルとしてまとめた。

また、LAMAX-S では、対称行列や対称バンド行列さらにはスパース行列など多様な行列を扱う。筆者は、このような多様な行列の構造をオブジェクト指向のクラス階層で表現し、各行列間の構造の変換 (たとえば、3 重対角行列をバンド行列に変換したり、対称行列として圧縮表現されているデータを非対称密行列の形式に変換したりする操作) をすっきりした形式で表現する方法を提案する。この考え方を筆者は、行列構造階層モデルとしてまとめた。

本章では、上述の基本的な考え方を実際に行列計算のライブラリに適用し、柔軟な構成の行列演算ライブラリの構築が可能であることを示す。また、この主張が実際のコンピュータ上で実現可能であることを示すために、筆者らは、C 言語にオブジェクト指向を導入した C++言語を用いて、本章で提案する行列フォルダモデルと行列構造階層モデルを実現したプロトタイプシステム FML(Flexible Matrix Library) を作成した。

[†1] 固有値問題は、EISPACK を用いていた。

LAMAX-S は、生成するプログラムとして FORTRAN を前提としているため、C++を利用している FML はただちに LAMAX-S に取り入れることはできない[†2]。本章で扱う内容は、現在文法が決定されている LAMAX-S の次のバージョンに対する効率的な実行時ルーチンのための基礎研究であると捉えることもできる。

第 3 章から第 5 章までの記述は、LAMAX-S のプリコンパイル時に最適化を行う方法を述べた。本章では、実行時に、行列データへなされた操作をベースにして最適化を計る方法を提案する。

6.2 オブジェクト指向と数値計算

本章では、オブジェクト指向と数値計算との関係について簡単に整理する。

オブジェクト指向はもともとは、グラフィックスや GUI(Graphical User Interface) などのプログラム記述、オペレーティングシステムの一部の機構の記述などのために利用されていた。これらの分野では、複雑なデータ構造を持つさまざまな形式のデータ (いわゆるオブジェクト) を、多様な処理方式で処理するタイプのプログラミングが必要であり、いわゆるオブジェクト指向との相性が良いと言われている。実際に、近年の GUI を伴うオペレーティングシステムやグラフィックスシステムの開発は、オブジェクト指向の考え方なしにはできないと言われている。

これに対し、数値計算は、単調なデータの繰り返し処理が多く、また処理の高速性が要求されているため、オブジェクト指向計算には不向きであると一般には考えられていた。しかし、LINPACK などでも分かるように、対称行列やバンド行列などのように構造を持つ行列は、データが圧縮されているので、特殊なデータ構造で表現されている。また、その種類も多種多様に富んでおり、これらに対する操作 (演算) もさまざまなものが要求される。

このような観点から、筆者は数値計算の分野であっても行列計算についてはオブジェクト指向の考え方が適用できると考えた。しかし、問題点もある。最も大きな問題は、実行効率の問題である。実行効率の良さは数値計算プログラムにとって非常に重要なことである。オブジェクト指向は、オブジェクト指向のメカニズムを実現するために、多少のオーバーヘッドを伴う。このオーバーヘッドが大きければ、行列計算にオブジェクト指向を導入する価値がなくなる。

6.3 本研究で提案する行列計算モデル

前述のように、FML では、行列ホルダモデルと行列構造階層モデルの 2 つの機能を実現している。本節では、この 2 つのモデルについて説明する。

6.3.1 行列ホルダモデル

行列フォルダモデルとは、行列のさまざまな特性を内部情報として保持することにより、行列計算の効率化と利用者の負担を軽減することを目的として考案されたモデルであり、どのような情報をどのタイミングで利用するかという機能を持つ。

まず、行列フォルダを定義する。行列フォルダは、図 6.1 に示すように、行列の構造や寸法などの情報の他に、実際の行列データへのリンクや三角分解値へのリンクを含む。この項目は、行列の構造によってさまざまなものが考えられる。

実際の具体例を用いて説明しよう。次の図 6.2 は、寸法 10×10 の対称行列の例である。実際の行列データは別の場所に割り当てられ、行列フォルダには、そのデータへのアドレス、いわゆるポインタが格納される。

この行列に対して、三角分解、固有値・固有ベクトルの処理が施されると、図 6.3 のようになる。

[†2] もちろん、オペレーティングシステムの中には、FORTRAN と C や C++などとのリンクを可能にするものもある。そのような場合、若干のインタフェースを作成するだけで利用可能になると思われる。

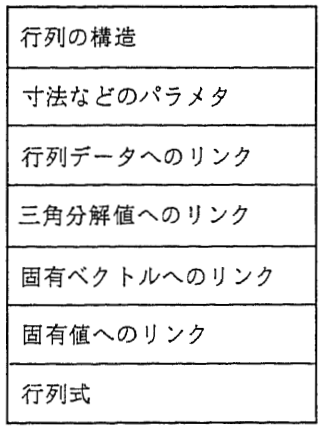


図 6.1: 行列フォルダモデル

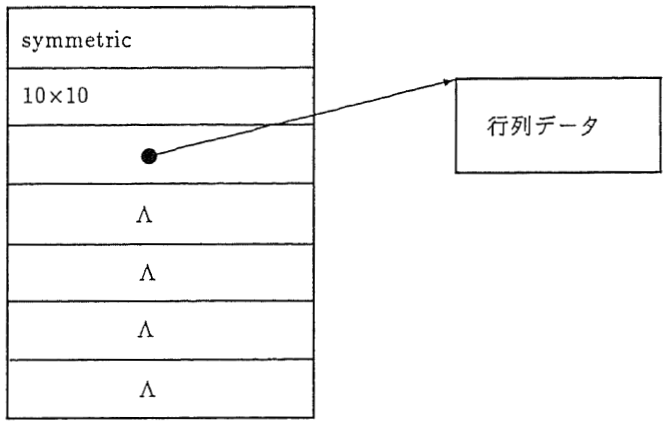


図 6.2: 行列フォルダモデル：対称行列の例

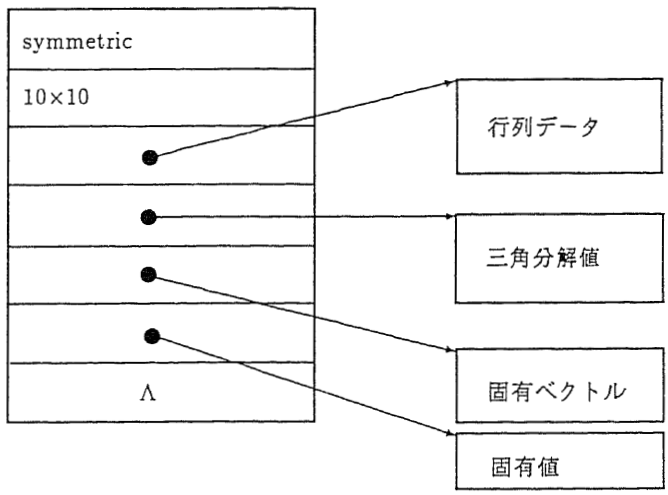


図 6.3: 行列フォルダモデル：各種処理が施された例

このように、たとえば、「三角分解された」という情報をオブジェクト(行列データ)側にもたせておくことにより、利用者がその管理をする必要がなくなる。たとえば、次のLAMAX-Sプログラムを例にこの利点を説明する。

```
if(ある条件) then
     solve A * x = b  ! 連立一次方程式 (a)
end if
solve A * y = c  ! 連立一次方程式 (b)
```

このプログラムにおいて、「ある条件」が成立していれば行列Aの三角分解がすでに一度行われているので、連立一次方程式(b)を解く際に、再び三角分解しなくてもよい。しかし、「ある条件」が成立しなければ、連立一次方程式(b)を解く際に、三角分解が必要になる。この解析は、プログラムを分岐させる条件の組み合わせが単純であれば、データフロー解析などにより、コンパイル時に解決できる。しかし、条件が複雑に入り組んでしまうとコンパイル時に解決することが難しくなる。しかし、この行列フォルダモデルを適用すれば、この条件の組み合わせがどんなに複雑になっても、行列変数が「自分自身が三角分解されたかどうか」実行時レベルで知っているので、無駄な三角分解処理をしなくてもよくなる。

また、現在のLAMAX-Sでは、行列変数が実行時に構造を変えることを許していない[†3]が、三重対角化などアルゴリズムの種類によっては、行列データの構造が実行時に動的に変化するものがある。行列フォルダモデルでは、行列の構造の情報がデータ(すなわち行列オブジェクト側)にあるので、動的な行列構造の変更が可能になるなどの利点がある。

6.3.2 行列構造階層モデル

LAMAX-Sでは、多種多様な構造の行列を扱う。この行列計算を管理する方法として筆者は行列構造階層モデルを提案する。基本的な考え方は、行列の構造をオブジェクト指向のクラス階層を使って表現するというものである。行列構造階層モデルのクラス階層を図6.4に示す。

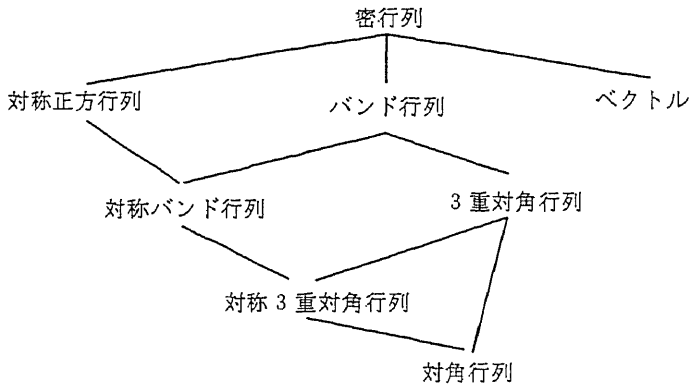


図 6.4: 行列構造階層モデル

これは、次のように考える。まず、もっとも基本的なクラスとして密行列がある。行列構造階層モデルでは、「すべての構造は、密行列へ変換可能である」と解釈する[†4]。次に、対称正方行列クラスは密行列クラスを継承している。これは、「対称正方行列クラスの構造を持つ行列は密行列クラスへ直接変換可能であ

[†3] 将来のLAMAX-Sでは、行列の構造を実行時に変更させることを可能にしたいと考えている。しかし、そのためには、その実現方法や実際のアルゴリズムとの関係をさらに調査する必要がある。

[†4] もちろん、大規模なスパース行列の場合、メモリ容量の関係で密行列に変換できない場合がある。今回のFMLでは、スパース行列に関しては、考察の対象からはずしている。しかし、スパース行列の場合、途中にスパースベクトルなどを置いて、間接的に処理することも考えられる。これについては、今後の研究課題としたい。

り、密行列クラスの関数の中で利用できるものは、それを用いる」ことを意味している[†5]。同様のことがバンド行列クラス、ベクトルクラスにもいえる。さらに、対称バンド行列は、対称正方行列クラスとバンド行列クラスの両方を多重継承している[†6]。最も、下位のクラスは対角行列である。

このクラス階層から理解できるように、一般のオブジェクト指向のクラス階層とは異なり、このモデルでは、下位の階層にいけばいくほど、データの量が少なくなる。

さて、このモデルによって、次の 2 つの利点を得られる。

最初の第 1 の利点として、行列の構造をその特徴に従って階層構造化し管理するため、同一の特徴を持つ構造の処理関数を共通化することが可能となる点が挙げられる。つまり、これはオブジェクト指向のメソッドサーチの概念を利用するもので、いわゆる差分プログラミングの効果が得られる。行列のデータ構造など共通点がある行列の構造では、同じ関数を利用することが多いのでこれによって実際の記述量を減らすことができる。たとえば、非対称行列の要素の総和を求める関数は、多くの場合、データ内のすべての要素を単純に加算すればよく、多くの行列構造で共有できる。

第 2 の利点として、異なる構造の行列間の計算であっても、実行時にシステムが図 6.4 のクラス階層の上部へ自動的に構造変換を行い計算をすることが可能となることが挙げられる。つまり、行列構造のクラス階層に従って、行列の構造を汎化することによって、計算できない構造の計算が可能になる。たとえば、例として、「バンド行列とベクトルの乗算ルーチン」がないが、「密行列とベクトルの乗算ルーチン」はあるとする。この場合、クラス階層に従って、バンド行列を密行列に変換して、「密行列とベクトルの乗算」として計算を続行させるといった処理が可能となる。

第 3 の利点として、行列の数学的な特性を考慮に入れることにより、行列の特性情報のより多い型にシフトし、より効率の良い計算ルーチンを利用することが可能となる[†7]。これは、ちょうど前述の利点の逆で、行列構造を特化する。すなわち、行列にある種の操作がなされた時点で、より構造の特化された方へ行列データを変換することにより、数学的な特性をより多く考慮する計算ルーチンを適用することができ、計算効率を向上させることが期待できる。

6.4 試作システム：FML の実現とその評価

6.4.1 FML の概要

筆者らは行列フォルダモデルと行列構造階層モデルを実際に C++上に構築したシステム FML(Flexible Matrix Library) を試作した。FML は、実験的に作成したシステムであるので、LAMAX-S で扱う全ての構造を網羅しているわけではない。FML で管理する行列構造のクラス階層を図 6.5 に示す。

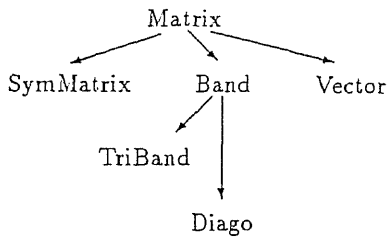


図 6.5: FML で管理する行列構造のクラス階層

[†5]実際問題としては、この場合、対称正方行列クラスのオブジェクトが密行列クラスで利用できる関数は少ない。
[†6]FML では、この多重継承は組み込んでいない。多重継承に関する技術的な話題は今後の課題である。
[†7]残念ながら、この機能はまだ FML には導入されていない。この機能の組込みは今後の課題である。

FML は、実際に行列データの管理をする部分と行列計算そのものを行う部分に分かれている。本来であれば、行列計算そのものを行う部分は、他のライブラリを利用すべきであったが、FML 構築時に C で利用できる適当な行列計算ライブラリがなかったため自作した。

また、FML では、行列構造はすべてクラスとして表現し、加減乗算などの演算は、C++の演算子のオーバーローディング機能を利用している。そのため、演算は通常の算術式通りに記述できる。このため、FML はライブラリといってもプログラミング言語に近い記述形式を持っている。

6.4.2 FML のプログラム例

FML のイメージを明確にするために、プログラム例を示す。次のプログラムは、ヤコビ法による連立一次方程式を解くサブルーチンである。

```
1  // ヤコビ法による連立一次方程式の解法
2  Matrix solve(Matrix &A,Matrix &B)
3  {
4          Matrix L,U,X,XOLD,ANS;
5          Diago D;                    //対角行列の変数
6          int i,NUM;
7          double EPS=10e-12;
8          NUM = 300;
9
10         XOLD = B;                   //大きさの割付と初期化
11         XOLD.fill(1.0);
12
13         L = l(A);                   //狭義の LDU 分解 (A = L+D+U)
14         D = d(A);
15         U = u(A);
16
17         for(i=1;i<=NUM;i++)
18         {
19                 X = 1/D*(B-((L+U)*XOLD));          //  1/D は D の逆行列
20                 if(ABSMAX(X-XOLD)<=EPS)
21                 {
22                         goto jump;
23                 }
24                 XOLD = X;
25         }
26
27         printf("%d 回繰り返しても解が求まりません (ヤコビ法)\n",NUM);
28         exit(0);
29         jump:;
30         return X;
31 }
```

2 行目と 4 行目の Matrix は、密行列のクラスである Matrix クラスのオブジェクト A, B, L, U, X, XOLD, ANS を定義している[†8]。5 行目の Diago は対角行列クラス (Diago クラス) のオブジェクト D を定義している。10 行目の代入文は引数として受け取った変数 B と同じ大きさの行列 XOLD を割り付けるために用いており、その次の行 (11 行目) で、fill という関数を用いて XOLD の全要素を 1 にしている (これは初期点である)。13 行目から 15 行目は、行列 A をそれぞれ下三角部分 l(A)、対角部分 d(A)、上三角部分 u(A) にスプリットしている。FML では、上・下三角行列がないので、U、L は密行列クラスで代用してい

[†8] この宣言は、C++ではコンストラクタと呼ばれ、オブジェクトを作る際に必要となる。

る[†9]。19 行目は、解を更新している。1/D で逆行列を意味する[†10]が、D は対角行列であることが分かっているので、D の各要素の逆数を取れば良いだけであるので計算効率は非常に良くなる。

6.4.3 FML の実現上のポイント

本項では、6.3.2 項で述べた利点を FML ではどのように具現化したかについて説明する。

第 1 の利点であったクラス階層を利用した差分プログラミングである。これは、C++に基本的に組み込まれているオブジェクト指向の機能を流用すれば実現可能であり、プログラミング技術的に難しい点はない。

第 2 の汎化プロセスについては多少の説明が必要である。汎化計算のメカニズムはオブジェクト指向の特徴である継承と強力な型チェックを利用して実現している。以下のリストは、実際に、FML で記述したプログラム (部分) である。

```
Matrix    a;
SymMatrix b;

a.print();
b.print();

a = a  +  b;

a.print();
```

このプログラムを簡単に説明する。2 行目では、対称行列のクラスである SymMatrix クラスのオブジェクト b を定義している。4 行目では、密行列オブジェクト a に print というメッセージを送っている。つまり、a の値を表示している。5 行目も同様である。7 行目では、a+b の計算をしてその結果を a に再代入している。9 行目で計算された a の値を表示している。

さて、今、「Matrix クラス+SymMatrix クラス」の計算ルーチンがないものと仮定する。単純に考えれば、上記のプログラムは実行できない。

そこで、FML では、「Matrix クラス+SymMatrix クラス」の計算ルーチンがないので SymMatrix クラスを Matrix クラスに汎化して、「Matrix クラス+Matrix クラス」として計算を行う。

具体的なこのメカニズムの手順は、この例では、以下のように処理される。

(1) 「Matrix+SymMatrix」の計算ルーチンを探す

(2) 「Matrix+SymMatrix」がないので「Matrix+Matrix」の計算ルーチンを使用する (継承で親クラスである Matrix の関数を使用)

(3) 「Matrix+Matrix」の関数内に SymMatrix 型のデータ構造まま実行を移す (太字の Matrix 型変数の部分)

(4) 「Matrix+Matrix」関数内で計算を行うが set,val 関数[†11]などが仮想関数であるので、多態性を実現し、同じ記述方法で各クラス独自の関数を使い分ける。すなわち関数の一部を下記のリストに記述すると、下線部の b.val(i,j) はこの場合 SymMatrix クラスの関数を使用している。

[†9] 実はこのことが、このプログラム (ヤコビ法) やガウスザイデル法などのプログラムの実行効率を悪くしている要因の一つになっている。

[†10] この記法は古い LAMAX-S と同じものであるが、C++のオーバーローディング機能の制約からこのようにせざるを得なかった。しかし、FML は記述法やプログラムの作成効率よりもむしろ実行効率やライブラリの構築の簡易さについての研究のためなので、この点についてはあえて無視することにした。

[†11] set 関数は、行列オブジェクトに値をセットする。val 関数は、行列オブジェクトの要素を引用する。

```
Matrix operator + (Matrix &a,Matrix &b)
{
    :
  c.set(i, j, a.val(i, j) + b.val(i, j));
    :
}
```

このような手順で Matrix+SymMatrix の計算を行うことができる。このメカニズムは他のクラスにおいても同様であり、これにより実行効率はかなり低下するが、多種多様な計算を処理することができる。

6.4.4 各構造に対する行列フォルダ

FML は、各構造 (Matrix, SymMatrix, Band, Vector, TriBand, Diago に対して、加減乗算、要素ごとの乗除算などを用意し、Matrix, SymMatrix に関しては、固有値・固有ベクトル・三角分解などの計算ができるようになっている。

6.4.4.1 Matrix クラスの行列フォルダ

Matrix クラスの行列フォルダの構造 (表 6.1) とそれに用意されている各種数学関数 (表 6.2) を示す。

表 6.1: Matrix クラスの行列フォルダの構造

変数の種類	変数名	型	変数の種類	変数名	型
行数	row	int 型	ヘッセンベルグ形	*H	Matrix 型
列数	col	int 型	オーソグナル形	*O	Matrix 型
名前 (計算過程)	*name	char 型	固有値	*RMD	Matrix 型
行列の値	*m	double 型	固有ベクトル	*EIVEC	Matrix 型
上三角行列	*U	Matrix 型	狭義上三角行列	*u	Matrix 型
下三角行列	*L	Matrix 型	狭義下三角行列	*l	Matrix 型
狭義対角行列	*d	Matrix 型			

6.4.4.2 SymMatrix クラスの行列フォルダ

SymMatrix クラスは、Matrix クラスを継承しているので、表 6.1 の内容に以下の内容 (表 6.3) を加えたものが行列フォルダの項目となる。
また、SymMatrix で新たに用意された数学関数には、固有値を求めるヤコビ法がある。

6.4.4.3 Band クラスの行列フォルダ

Band クラスの行列フォルダを表 6.4 に示す。

6.4.4.4 TriBand クラスの行列フォルダ

TriBand クラスの行列フォルダを表 6.5 に示す。
数学関数としては対称行列を 3 重対角化するギブンス法が作成されている。

6.4.4.5 Diago クラスの行列フォルダ

Diago クラスの行列フォルダを表 6.6 に示す。

表 6.2: Matrix クラスにおける数学関数

関数の名前	意味
LU 分解	
void LU(Matrix &)	LU 分解をする
Matrix L(Matrix &)	L の値を返す
Matrix U(Matrix &)	U の値を返す
LDU 分解	
void ldu(Matrix &)	狭義 LDU 分解をする
Matrix l(Matrix &)	L の値を返す
Matrix d(Matrix &)	D の値を返す
Matrix u(Matrix &)	U の値を返す
固有値, 固有ベクトル	
Matrix GIVENS(Matrix &)	ヘッセンベルグ型を求める
Matrix ORTHOGNAL(Matrix &)	オーソグナル型を求める
void GVNHSN(Matrix &)	ヘッセンベルグ型を求める
Matrix RMD(Matrix &)	固有値を求める
Matrix EIVEC(Matrix &)	固有ベクトルを求める
void QR(Matrix &)	QR 法
void INVITE(Matrix &)	INVITE 法
連立一次方程式を解く	
Matrix CROUT(Matrix &,Matrix &)	連立一次方程式 (クラウト法)
Matrix solve1(Matrix &,Matrix &)	連立一次方程式 (ヤコビ法)
Matrix solve2(Matrix &,Matrix &)	連立一次方程式 (ガウスーザイデル法)
その他の関数	
void I()	単位行列にする
void clear()	ゼロ行列にする
double ABS(double)	絶対値を求める
double SIGN(double)	サイン関数 (± 0.5 を返す)
double ABSMAX(Matrix &)	行列の絶対値最大の要素
double ALLMIN(Matrix &)	行列の要素の最小値を求める

表 6.3: SymMatrix クラスの行列フォルダ (追加分)

変数の種類	変数名	型
固有値	*eigen	Matrix 型
固有ベクトル	*vector	Matrix 型
ヘッセンベルグ形	*H	Matrix 型
オーソグナル形	*O	Matrix 型

表 6.4: Band クラスの行列フォルダ (追加分)

変数の種類	変数名	型
上バンド幅	mu	int 型
下バンド幅	ml	int 型
バンド行列用転置行列	*btrns	Band 型

表 6.5: TriBand クラスの行列フォルダ (追加分)

変数の種類	変数名	型
3 重対角行列の転置行列	*btrns	TriBand 型
ヘッセンベルグ形	*H	Matrix 型
オーソグナル形	*O	Matrix 型

表 6.6: Diago クラスの変数表

変数の種類	変数名	型
逆行列	*inv	Diago 型

6.5 FML の評価

本節では、FML の実行効率について調べてみることにする。題材としては、ガウス－ザイデル法による連立一次方程式の求解を取り上げる。比較の対象として FORTRAN のプログラム†12で同じ問題を解いた例を表 6.7 に載せる。

表 6.7: ガウス－ザイデル法による連立一次方程式の求解：密行列の測定結果 (秒)

寸 法	FORTRAN	FML	FML/FORTRAN
10×10	3.34×10^{-3}	8.66×10^{-2}	25.9
20×20	3.53×10^{-2}	4.60×10^{-1}	13.0
30×30	1.04×10^{-1}	1.20×10^{0}	11.5
40×40	4.58×10^{-1}	2.71×10^{0}	5.92
50×50	1.65×10^{0}	5.81×10^{0}	3.51

解くべき問題の寸法が少ない場合、その差は著しく異なる。これは、寸法が少ない場合には、全体の実行時間に占める C++のオーバーヘッドの割合が圧倒的に大きいことを示している。寸法が大きくなると、数値計算の占める割合が増え、比が少なくなっていることがわかる。

FML は、その適用分野として、プログラムの中に多数の場合分けを含み、さらに行列が実行時に決定されるさまざまな条件のもとにさまざまな形で加工されるようなアルゴリズムでないと効果を発揮できない。ガウス－ザイデル法のような比較的単純なアルゴリズムでは、C++およびモデルの実行時オーバーヘッドが大きくなり、実用的でないことが明らかとなった。これは、ある程度は、FML の実現方法にも関係している。FML は性能を測定するためのオーバーヘッドがかなりある。たとえば、計算がどのように進行していったかのトレースを内部的に行っている。今後は、このようなオーバーヘッドが極力少なくなるように、システムを再構築する必要がある。

6.6 本章の結論と今後の課題

筆者の提案した 2 つのモデル：行列フォルダモデルと行列構造階層モデルをプロトタイプの形ではあるが実際にシステムとして実現することができた。

しかしながら、第 6.5 節で明らかなように、本章で提案した 2 つのモデルは、単純なアルゴリズムに対しては、オーバーヘッドがかなり大きくなり有効でないことが分かった。第 6.5 節で述べたことと重複す

†12 LAMAX-S が FORTRAN をベースとしているため、あえて FORTRAN で作成したプログラムを比較対象とした。しかし、SUN ワークステーションでの比較のため、C でのプログラムと大差はないと考えられる。

るが、「FML は、その適用分野として、プログラムの中に多数の場合分けを含み、さらに行列が実行時に決定されるさまざまな条件のもとにさまざまな形で加工されるようなアルゴリズムでないと効果を発揮できない」ので、このような分野の問題を捜し、再び、FML の有効性を確認することが今後の課題である。

また、今回は、スパース行列を考察の対象にしていないが、スパース行列に対する本方式の適用は今後の研究課題として重要であると考えている。

FML の作成にあたっては、神奈川大学大学院工学研究科 (現 (株) 日本電気)　秋山隆之氏によるところが大きい。特に、各行列構造の具体的な行列フォルダの項目の設計は彼の手によるものである。ここに記して感謝の意を表します。

第 7 章 LAMAX-S の応用と評価

7.1 はじめに

第 2 章では LAMAX-S の歴史について述べ、第 3 章では LAMAX-S の文法について述べ、さらに第 4 章、第 5 章、第 6 章では、LAMAX-S(Ver.2.0)[†1] の処理系の実現方法とその最適化について述べてきた。

本章では、本研究の締めくくりとして、LAMAX-S の応用と評価について述べることにする。まず、筆者らの開発したプロトタイプシステム LAMAX-S (Ver.1.5) について評価する。このシステムは、1991 年に最初の処理系が完成し、現在まで約 4 年間の利用実績があり、利用面からの評価が可能である。一方、LAMAX-S (Ver.2.0) は、本論文で提案した技術をもとに作成中の処理系であり、まだ、利用実績がない。しかし、文法が決定されているので、さまざまな分野での利用の可能性の期待度を示すことはできる。

第 7.2 節では、LAMAX-S(Ver.1.5) について適用例とその評価について述べる。特に、LAMAX-S (Ver.1.5) は、数値計算やオペレーションズ・リサーチ、統計計算、計量経済などの分野のプログラミングを伴うアルゴリズム教育に適している思われるのでそれについて述べる。第 7.3 節では、LAMAX-S (ver.2.0) が利用される分野におけるさまざまな可能性の期待度について述べる。

7.2 LAMAX-S(Ver.1.5) の適用例とその評価

LAMAX-S (Ver.1.5) は、約 4 年間筆者の周辺で 10 数名の利用者によって利用されてきており、ある程度の利用実績を積むことができた。特に効果を上げたのは、教育の場での利用である [69]。筆者の指導のもと、7 名の神奈川大学工学部経営工学科の卒業研究生が、LAMAX-S を用いてさまざまなプログラムを作成した。7 名のうち、全員が FORTRAN や C でプログラムを組むよりも、LAMAX-S で組んだ方がアルゴリズムの理解が容易であったと述べている。そこで、本節ではまず LAMAX-S (ver.1.5) を教育に適用した事例について述べる。

次に、LAMAX-S のもともとの目的の一つは、手法開発用ツールの提供ということであったので、この点について述べる。しかし、残念ながら、現在までのところ、LAMAX-S (Ver.1.5) を利用して新しい手法を開発したという報告を筆者は受けていない。そこで、筆者らは、ASNOP[4] というツールと LAMAX-S (Ver.1.5) によるプログラムを比較検討することにより、LAMAX-S (Ver. 1.5) の手法開発用ツールとしての可能性について調べた。ASNOP とは、Application System for Nonlinear Optimization Problems の略でアスノップと呼ぶ。これは、非線形最適化問題を準ニュートン法を用いて解くシステムで、1982 年から、八巻直一、本郷茂、宮田雅智 (青山学院女子短期大学)、高橋悟 (東京理科大学経営学部)、矢部博 (東京理科大学工学部) によって開発され、最初のバージョンは青山学院大学情報科学研究センターで一般公開され [70]、その後、東京大学大型計算機センターなどでも一般公開された [46]。1988 年には、日本オペレーションズ・リサーチ学会の事例研究奨励賞 (ソフトウェア部門) を受けている [45]。ASNOP のような実用的に用いられているアプリケーションとの比較によって LAMAX-S の手法開発用ツールとしての性能について述べる。

7.2.1 教育分野

本項では、主に筆者が現在所属している神奈川大学工学部経営工学科唐澤豊研究室 (経営情報工学研究室) において行った LAMAX-S を用いた教育 (卒業研究) 上の効果と学生の手による LAMAX-S (Ver. 1.5) の

[†1] いわゆるフルセットの LAMAX-S。

プログラムの実行効率などを分析する。

また、その活動を通して、学生らの作成したプログラムの他に、本郷や八巻、筆者らの作成したプログラムをサンプルプログラムとしてまとめてあり、それについても述べることにする。

7.2.1.1 神奈川大学工学部経営工学科唐澤研究室における卒業研究

筆者の指導のもと、1992 年度の卒業研究から 1994 年度の卒業研究まで約 7 名の学生が LAMAX-S を用いて、さまざまなプログラムの開発と LAMAX-S (Ver. 1.5) の実行効率の評価実験を行った。1992 年度の卒業研究生の場合 (一人)、LAMAX-S 処理系の評価というよりは、当時の処理系の信頼性が低かったため、LAMAX-S 処理系のエラーの発見が主要な活動となってしまった。しかし、1993 年後半頃から処理系の信頼性が向上し、さらにワークステーション上で利用可能となったこともあってかなり本格的に利用できるようになった。

この一連の卒業研究で学生が作成したプログラムは、連立一次方程式の直接解法・反復解法・固有値問題・非線形最適化問題などである。学生の意見としては、FORTRAN で作成する場合に比べて約 1/3 から約 1/10 程度の開発期間でプログラムを作成でき、さらにアルゴリズムとプログラムリスト上の記述ギャップが少ないので、アルゴリズムを強力に意識することができ、アルゴリズムの習得に役だったということであった。

実際に学生の作成したプログラムにより、LAMAX-S (Ver. 1.5) の動作を評価してみる。次の表 7.1 は、ガウスーザイデル法 (Gauss-Seidel's method)、SOR 法 (Succesive Over Relaxation method : 過剰緩和法)、CG 法 (Conjugate Gradient method : 共役勾配法) [†2]、ICCG 法 (Incomplete Cholesky decomposition with Conjugate Gradient method : 前処理付き共役勾配法) のプログラムを FORTRAN と LAMAX-S で作成し、解が求められるまでの反復回数をまとめたものである。データは対称スパース行列を採用している[†3]。FORTRAN のプログラムによる反復回数と LAMAX-S のプログラムによる反復回数が同じ場合には、その数値のみを示し、異なる場合には括弧 ( ) の中に FORTRAN のプログラムによる反復回数を示した。

表 7.1: 算法別反復回数 (対称スパース行列)

寸 法	ガウスー ザイデル法	SOR 法				CG 法	ICCG 法
		solve 文あり		solve 文なし			
		簡便法	数値的	簡便法	数値的		
10 × 10	8	8	8	8	8	10	4
20 × 20	8	8	8	8	8	19	6
30 × 30	10	10	10	10	10	29	7
40 × 40	12	12	12	12	12	40	8
50 × 50	17	17(16)	16	17(16)	17	45	8

表 7.1 の SOR 法において、「solve 文あり・なし」および「簡便法・数値的」ということについて説明する。「solve 文あり」、「solve 文なし」というのは LAMAX-S の関数の solve 文を用いたか否かの違いである。もともとの SOR 法では、逐次的に計算を進めるが、LAMAX-S (Ver.1.5) では要素演算が非常に遅くなるという事情がある。そのため、素直に SOR 法を LAMAX-S で記述する (これは「solve 文なし」に相当する) と表 7.3 で分かるように、計算時間が猛烈に長くなる。そこで、SOR 法のアルゴリズムを多少変更して解の更新を連立一次方程式を解く形式に改めたプログラムが「solve 文あり」に相当する。

[†2] 表の見出し中では CG 法と記すが、本文中では共役勾配法と記す。

[†3] スパースといっても、LAMAX-S (Ver. 1.5) ではスパースのデータ構造を扱うことができない。ここでは、密行列のデータ構造にスパースのデータをセットしている。また、ここで、$n \times n$ の行列データを発生させた方法について述べておく。(1) 対角要素に乱数で発生させた数をセットし、それ以外を 0 とする。(2) 対角要素以外に n 個の乱数で発生させた数をセットし、この行列を T とする。(3)$T'T$ を計算し、これをデータとする。

また、簡便法と数値的というのはSOR法における加速係数ωの2つの推定法つまり、数値的推定法と簡便法を示す [48]。

表 7.1 から見て分かるように、反復回数においては、FORTRAN のプログラムと LAMAX-S のプログラムとの間に大きな違いはない。それでは、データが対称の密行列の場合はどうであろうか。表 7.2 にデータを対称密行列としたときの同様の結果を示す。

表 7.2: 算法別反復回数 (対称密行列)

寸 法	ガウス－ザイデル法	SOR 法				CG 法	ICCG 法
		solve 文あり		solve 文なし			
		簡便法	数値的	簡便法	数値的		
10 × 10	16	16	16	15	15	10	7
20 × 20	31	28	26	23	24	21	13
30 × 30	58	52(49)	43	46	27	33	17
40 × 40	243	158(152)	142	154	138	44	23(24)
50 × 50	258	165(159)	149	105	113	57	28(31)

この結果もほとんど同じである。この例においては、FORTRAN と LAMAX-S の動作上の違いはほとんどないことが分かる。

それでは、実行時間は FORTRAN と LAMAX-S でどの程度異なるのであろうか。表 7.3 は LAMAX-S における計算時間である。計測は、SUN ワークステーション上で最適化オプションなしにコンパイルしたプログラムの結果である。

表 7.3: LAMAX-S の算法別実行時間 (秒)

対称スパース行列							
寸 法	ガウス－ザイデル法	SOR 法				CG 法	ICCG 法
		solve 文あり		solve 文なし			
		簡便法	数値的	簡便法	数値的		
10 × 10	0.3	0.3	0.3	3.5	3.3	0.7	0.7
20 × 20	0.5	0.5	0.5	12.8	12.1	2.4	3.0
30 × 30	0.9	0.9	0.9	42.5	41.3	6.3	7.8
40 × 40	1.4	1.4	1.4	106.2	106.1	15.6	17.4
50 × 50	2.2	2.3	2.2	284.0	226.7	27.9	32.4
対称密行列							
10 × 10	0.3	0.3	0.4	10.6	10.3	0.7	1.1
20 × 20	0.7	0.7	0.7	94.7	102.9	2.7	5.8
30 × 30	1.6	1.7	1.7	510.0	294.7	7.4	18.6
40 × 40	8.4	6.2	8.3	4847.0	4134.4	18.6	54.0
50 × 50	12.5	9.0	12.0	3596.6	4070.8	41.4	116.9

これに対し、FORTRAN での計算時間を表 7.4 に示す。

表 7.3、表 7.4 から分かるように、行列の寸法が少ないときは、LAMAX-S の実行時間は数秒のオーダで、教育目的としては、十分である。しかし、FORTRAN で作成されたプログラムとの比較では、やはりかなり遅くなっていることがわかる。さらに、LAMAX-S (Ver. 1.5) では、プログラムの組み方によって実行効率に致命的な差が生じることがわかった。たとえば、前述のように SOR 法 (solve 文なし) では、ほとんど要素演算となる。LAMAX-S (Ver. 1.5) では要素演算を行うと、いちいちその要素のチェックを行うため、極端に実行速度が低下する。「solve 文あり」では、要素演算を行わずに連立一次方程式を解く方法でアルゴリズムを記述している。このようにすると、表 7.4 に示されているように、FORTRAN では遅くなるが、LAMAX-S では飛躍的に実行効率が改善される。

表 7.4: FORTRANの算法別実行時間(秒)

寸 法	ガウス－ザイデル法	SOR法				CG法	ICCG法
		solve文を使う解法		solve文なしの解法			
		簡便法	数値的	簡便法	数値的		
対称スパース行列							
10 × 10	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20 × 20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30 × 30	0.1	0.1	0.1	0.1	0.1	0.1	0.1
40 × 40	0.1	0.1	0.1	0.1	0.1	0.3	0.2
50 × 50	0.3	0.3	0.3	0.2	0.2	0.5	0.2
対称密行列							
10 × 10	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20 × 20	0.0	0.1	0.1	0.1	0.0	0.1	0.0
30 × 30	0.2	0.2	0.2	0.1	0.1	0.1	0.1
40 × 40	1.2	0.8	0.8	0.6	0.8	0.3	0.3
50 × 50	2.0	1.3	1.2	0.7	0.9	0.5	0.6

しかしながら、小規模データを用いた教育目的であれば、十分、利用できると考えて良いと判断でき、また、FORTRANにおけるプログラムとの反復回数もそれほど差がないなど、アルゴリズムの特徴をそれなりに反映していると考えられる。

一方、LAMAX-Sの改善すべき点として利用者(学生)から指摘された点として、まず、LAMAX-S (Ver.1.5)の出力するエラーメッセージの内容の不親切さが挙げられた。たとえば、連立一次方程式を solve 文を用いて解けなかった場合、単に「～行でLU分解ができませんでした」というメッセージが表示されるだけである。卒業研究に着手したばかりの学生は、三角分解という概念さえ知らないので、このメッセージによってかえって最初は混乱してしまったという報告を学生から受けた。初学者でも十分に理解できるようにメッセージを内容をより親切にする必要がある。

ところで、パソコン上でLAMAX-Sを用いる場合、LAMAX-Sプリプロセッサの処理時間はほとんど問題とならないが、MS-FORTRANなどのパソコン上の言語処理プロセッサの処理速度が若干遅いため、何回もコンパイルする場合、多少問題がある。しかし、ワークステーション上の場合には、この問題はまったくない。

また、次のような2つの懸念がある。(1) LAMAX-Sのような非常に高度化されたプログラミング言語を使うと、実際のプログラミングに必要なさまざまなノウハウが身につかないのではないか。(2) LAMAX-Sでは、行列計算の手続きの詳細が隠されているので、他の言語(FORTRANやC)でプログラムを組む際に効率の良いプログラムを組めなくなるのではないか。

この懸念を取り除くために、LAMAX-S (Ver. 2.0)では、LAMAX-Sが生成するFORTRANプログラムに関する内容の詳細を、ユーザの要求によりレポートの形式で出力することが必要である。その内容としては、選択された手法やその計算手順などが挙げられる。このレポートを学生が読むことにより、内部処理の学習になると考えられる。

7.2.1.2 現在までに用意されているサンプルプログラム

サンプルプログラムは、LAMAX-Sの学習をする上でも、さまざまなアルゴリズムの学習をする上でも重要である。特に、LAMAX-Sで書かれたプログラム上には、当然のことながら、さまざまなアルゴリズムの具体的な計算手順が書かれているため、そのプログラムをトレースするだけで学習効果は大きい。

現在、用意されているサンプルプログラムを表7.5にまとめる。

実際のサンプルプログラムの例については付録に示す。

表 7.5: LAMAX-S のサンプルプログラム

分　野	内　容
非線形方程式	逐次代入法、ニュートン法
非線形最適化問題	最急降下法、ニュートン法、準ニュートン法
非線形最小二乗法	ガウスニュートン法
固有値問題	べき乗法、ヤコビ法、ハウスホルダー法
三角分解	密行列、対称行列、バンド行列、対称バンド行列
連立一次方程式	ヤコビ法、ガウスーザイデル法、共役勾配法、SOR 法、ICCG 法
その他	重回帰分析、因子分析、改訂シンプレックス法

7.2.2 手法研究

LAMAX-S が手法研究にもある程度有効であることを示すために、実用的に利用されている ASNOP との比較を行うことを本項では試みる。LAMAX-S の実行時間は、前項で示した通り、行列の寸法が大きくなると実行時間が増大することが報告された。しかし、それほど大きなデータを必要としない研究分野では十分に実用的に利用できるのではないかと考えられる。そこで、本項では、*ACM Transactions on Mathematical software, Vol. 7, No. 1 March 1981, pp.17-41* の Testing Unconstrained Optimization Software に示された非線形最適化問題の例題を LAMAX-S (Ver. 1.5) を用いて、準ニュートン法で解き、ASNOP での結果と比較することにする。

この例題の中から無制約最小化問題を選んだ。この問題は全部で 14 問ある。ASNOP で実行した結果と LAMAX-S でこの問題をプログラムを組み実行した結果を表 7.6 に示す。

表 7.6: 無制約最小化問題における LAMAX-S と ASNOP との比較

問題番号	LAMAX-S(反復回数)	ASNOP(反復回数)	LAMAX-S(実行時間：秒)
1	33	36	2.0
2	6	11	0.2
3	194	155	76.3
4	13	13	0.4
5	14	10	0.5
6	63	11	33.9
7	31	30	3.2
8	50	12	77.9
9	14	8	3.7
10	245	174	762.4
11	overflow	overflow	—
12	overflow	overflow	—
13	28	30	1.5
14	23	32	1.2

この結果、14 問中、LAMAX-S でも ASNOP でも 12 問解けたが、LAMAX-S の方が反復回数が少ないケースが 3 問あった (表中下線で示す)。また、実行時間であるが、ASNOP では、第 10 問が 2.0 秒かかったがそれ以外はすべて 0.1 秒以下であり、やはり LAMAX-S (Ver.1.5) の効率はあまりよくないことが分かる。

しかし、第 3, 6, 8, 10 問などかなり実行時間のかかるケースもあるが、反復回数に限っていえば、ASNOP に比べてそれほど悪いともいえない。実行時間がさほど問題にならない状況では、十分手法開発用ツールとしても利用可能ではないかと考えられる。

7.3 LAMAX-S(Ver.2.0) の期待

前節では、プロトタイプシステム LAMAX-S(Ver. 1.5) は行列の寸法が大きくなると、実行効率の低下が顕著になり、さらに、アルゴリズムとして要素演算の頻度が高いものも実行効率が極度に低下することが示された。

一方で、プログラムの記述性や開発効率は、当初の目標通りの成果が得られていることが分かった。

LAMAX-S (Ver. 2.0) で期待されていることは、この実行効率の改善である。まず、LAMAX-S (Ver. 2.0) では、LAMAX-S (Ver. 1.5) とは異なり、要素の位置の計算などの処理は極力プリコンパイル時に行う。また、実行時のオーバーヘッドを減らすような形でプログラムフレーム[†4]を作成することにより、かなり実行時間の効率化が計れる。さらに、プログラムに対して、チューニングを施すので、この効果も期待できる。

また、複雑なアルゴリズムを記述しても、第 4 章で述べたように、計算量低減化の効果を期待できる。

いずれにせよ、LAMAX-S (Ver. 2.0) では、実行効率は大幅に改善されると考えられるので、大規模問題への適用についても考えてゆきたい。

LAMAX-S (Ver. 2.0) において、期待されるもう一つの効果は、スパース行列を扱えるようになるということである。ただし、現時点では、スパースのデータ構造は LAMAX-S 固有のものを採用し、実行時にそれぞれのライブラリに合わせてデータ変換を行うので、効率についてはさほどよくないと考えられる。この点については、スパース行列の扱いにある程度、運用経験を積み、解決してゆきたいと考えている。

また、当初から予定していたスーパーコンピュータ用のライブラリも利用可能にする予定であるので、スーパーコンピュータ上での利用実績も得ることができる。

現在の利用者層は、主にオペレーションズ・リサーチや統計計算、計量経済関係が中心となるが、LAMAX-S (Ver. 2.0) では、当初の予定通り、スーパーコンピュータのユーザ (構造解析や流体解析などの専門家) などにも広めてゆきたいと考えている。

そのためには、有限要素法などの応用プログラムの LAMAX-S (Ver. 2.0) による記述実験などが必要となるであろう。

7.4 本章の結論と今後の課題

本章では、現在利用している LAMAX-S (Ver. 1.5) の性能を評価するとともに、LAMAX-S (Ver. 2.0) の可能性について検討した。LAMAX-S (Ver. 1,5) は、何度も述べるが、LAMAX-I という商品名で (株) システム計画研究所から発売することが可能となった。実際に、動作するようになったのは、1991 年のことであるから、処理系が安定して動作して、商品として出荷できるまで約 4 年間もかかってしまったことになる。しかし、この処理系は、本文中でも述べたように、アルゴリズム教育や小規模データでのアルゴリズム検証や開発などに用いるには十分であることが、評価実験から明らかになった。

今後、より多くの方に LAMAX-S (Ver. 1.5 : 商品名 LAMAX-I) を利用して頂き、さらに運用経験を積むとともに、LAMAX-S (Ver. 2.0) を完成させることが今後の課題である。

なお、本章において、さまざまな LAMAX-S プログラムとその計測結果が示されたが、本章のほとんどの計測結果は、神奈川大学工学部経営工学科唐澤豊研究室の卒業研究の学生の手によるものである。ここに記して感謝の意を表します。

[†4] 59 ページ 5.2.1 項で説明した。

第 8 章 結　論

本論文では、2 つの行列演算用言語ミニ LAMAX、LAMAX-E の文法設計、処理系開発の経験およびその運用経験をベースに、LAMAX-S の文法の設計を行った。さらに、LAMAX-S の記述性を評価するために、筆者らと (株) システム計画研究所はそのプロトタイプシステムを開発し、実際に 10 数名のユーザに試用させながら、文法の事項を若干改善した。そして、最終的にその記述性が有効であることを確認した。また、このプロトタイプ処理系は、アルゴリズム教育などに伴うプログラミング実習や小規模のデータを扱う手法開発などでも有効であることを示した。

ただし、プロトタイプシステムには実行効率に多少の問題を残すため、これを改善するために、フルセット LAMAX-S の開発が要求されている。筆者は、フルセット LAMAX-S の開発を行うための基礎技術となる計算量低減化と自動チューニングを提案した。また、筆者らは計算量低減化の検証プロトタイプシステム LOPS と自動チューニングのためのチューニング効果測定システム ATTPG を作成し、それらの技術の有効性を確認した。また、筆者は、行列計算にオブジェクト指向を導入し、より柔軟な処理が可能なシステムを構築するためのモデルとして、行列フォルダモデルと行列構造階層モデルを提案した。これを検証するために、筆者らは、そのプロトタイプシステム FML を作成した。

今後の筆者の大きな目標は、システム計画研究所の方々と協力して、LAMAX-S (Ver. 2.0) を完成させることである。

また、筆者に課せられたもう一つの役割は、LAMAX-S の普及である。プログラミング言語を広めるためには、やはり企業化ということが必要であると筆者は考えている。大学では、開発に携わる学生が 1 年でいなくなるので、実質的に保守ができない。また、ユーザからの要求にも直ちに応える訳にはいかない。LAMAX-S は、大学などの研究機関だけでなく、広く企業の開発現場でも使って頂きたい。やはり、企業サイドで使用するとなると、どこかできちんと責任を持って管理してくれるところが必要となるであろう。この点で、LAMAX-S はシステム計画研究所というしっかりしたパートナーがいるので恵まれている。

プログラミング言語の商品化という点では、筆者は苦い経験を持っている。LAMAX-E の開発経験をベースに筆者が開発した ADT-RASM86(2.1.5 項) システムが企業からの誘いがあったにもかかわらず、企業化しなかった。そのため、情報処理学会の論文誌や米国経営工学会のジャーナルには載ったが、残念ながら、現在、そのシステムは誰にも使われていない。大学に勤務している者の使命は、論文を書くことであるから、その使命は達したことになるが、やはり、言語開発者としてはいちまつの寂しさを感じている。このような観点から一刻も早く LAMAX-S(Ver. 2.0) を作成し、それを世に広めて行きたいと考えている。

LAMAX は、最初の研究開始から 10 年以上の歳月が流れた。数多くのプログラミング言語が生まれ、また、消滅して行く中で、LAMAX は比較的息の長いプロジェクトである。いま、考えるに、もし、最初のミニ LAMAX を作らなかったとしたら、おそらく、LAMAX-E も LAMAX-S も日の目を見なかったのではないかと思う。ミニ LAMAX は、当初予定していたシステムが、動きそうもないのであわてて作成した超プロトタイプであった。しかし、ミニ LAMAX のおかげで、LAMAX の重要性が理解され、研究を進めることができた。どんな研究においても、最初の小さな一歩が重要であることを痛感した。

謝　辞

最後に、本論文の作成にあたって始終懇切丁寧なご指導を頂いた会津大学コンピュータソフトウェア学科教授　池辺八洲彦先生、筑波大学電子・情報工学系教授　名取亮先生に深く感謝いたします。また、筑波大学電子・情報工学系教授　海老原義彦先生、同　宮本定明先生、同　稲垣敏之先生には、査読とともに数多くの貴重なご意見を頂き、ありがとうございました。そして、現在、筆者の勤務している神奈川大学工学部経営工学科教授であり筆者の上司でもある　唐澤豊先生には、本論文を執筆する機会と環境を与えて頂き深く感謝しております。

また、筆者の大学１年次より、コンピュータの基礎から教えて頂き、LAMAX 研究のきっかけを与えて頂いた青山学院大学経済学部助教授　本郷茂先生には、公私ともに指導して頂き、感謝の言葉もありません。そして、LAMAX-S 処理系開発のきっかけを与えて下さった (株) システム計画研究所取締役主任研究員　八巻直一先生にも公私ともにご指導頂き、本当にありがとうございます。

LAMAX-S の設計の際には、多くの大学・企業・研究機関に出向きましたが、そこでの情報収集は大いに参考になりました。特に毎年開催されている数値解析シンポジウムにおける大シンポジウムでの討論は大いに参考になりました。また、LAMAX-S プロトタイプシステムの開発の中心となった (株) システム計画研究所の方々にも感謝いたします。(株) システム計画研究所代表取締役社長　嶋田駿太郎氏には、遅れがちな LAMAX-S 開発作業をいつもあたたかくときには積極的に見守って頂きました。LAMAX-S 開発の中心である (株) システム計画研究所　大津崇氏、井上美明氏にも感謝いたします。特に、LAMAX-S 文法改善においては、井上美明氏の貢献が大であります。

また、青山学院大学理工学部経営工学科　間野研究室の OB の方々、特に青山学院大学理工学部助教授井田昌之先生、キャノン株式会社　佐藤衛氏には学生時代を通してさまざまな指導をして頂きました。神奈川大学工学部経営工学科　唐澤豊研究室の方々にも感謝いたします。特に、相浦宣徳氏、秋山隆之氏には、心より感謝いたします。

また、筆者の勤務先の神奈川大学工学部経営工学科の教職員の皆様に心より感謝の気持ちを表したいと思います。

そして、筆者の卒業研究および大学院学生時代を通して、初心から一貫して指導して頂いた間野浩太郎先生に心より感謝いたします。

Appendix サンプルプログラム

例として、非線形最適化問題の中から「準ニュートン法」を記述して実行した例を示す。

BFGS 公式を用いた準ニュートン法 (quasi-Newton method)

ここでたとえば、

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^3 + 10(x_1 - x_4)^4$$

を最小化する。なお、$Df(x) = \nabla f(x)$ とする。

Step0 初期点 x_0 と、初期行列 H_0 を設定する。$k = 0$ とおく。

$$k = 0$$

$$H_k = I$$

$$x_k = (3.d00, -1.d00, 0.d00, 0.1d00)'$$

Step1 停止条件を検査する。

$\|Df(x_k)\| < eps$ ならば、終了

Step2 探索方向 d_k を求める。

$$d_k = -H_k * Df(x_k)$$

Step3 d_k 方向の刻み幅を求める（直線探索）

刻み幅をα_k　とする。

$$s_k = \alpha_k d_k$$

$$x_{k+1} = x_k + s_k$$

Step4 行列 H_k を更新する。

$$y_k = Df(x_{k+1}) - Df(x_k)$$

$$H_{k+1} = H_k + (1 + \frac{y_k^T H_k y_k}{s_k^T y_k})\frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k s_k^T + s_k y_k^T H_k}{s_k y_k}$$

Step5 $k = k + 1$ として *Step1* へ行く。

$$k = k + 1$$

Step 1 へ行く。

このプログラムを次のページから示す。

```
c 準ニュートン法
      parameter ( n = 4 )
       real*8:vector[n] x
       real*8:vector[n] d
       real*8:vector[n] s
       real*8:vector[n] y
       real*8:vector[n] g
       real*8:vector[n] w
       real*8 f , fv
       real*8 ff , sy , yHy ,norm
       real*8:vector[*]   Df
       real*8:matrix[n,n] H
       real*8:matrix[n,n:diagonal] I
       real*8 eps

c Step 0 初期点 x と，初期行列 H を設定する。k = 0 とおく。

c 初期プロセス
       eps = 1.D-6
       I = 1.0d0 ::matrix[n,n:diagonal]
       H = I
       k = 0
       call minput(x)
       g  = Df(x)

c Step 1 停止条件を検査する。

  10   continue
       fv = f(x)
       norm =  absmax( g )
       write(*,100) k,fv,norm
 100   format( '  step=',i4,'   f=',d8.3,'  norm g=',d8.3)
       if( norm .lt. eps ) go to 10

c Step 2 探索方向 d を求める。

       d = - H * g

c Step 3 d 方向の刻み幅を求める（直線探索）
       size = 1.d00
       do 200 j = 1 ,20
         ff = f(x + size*d)
         if( ff .lt. fv ) go to 300
         size = 0.5d00*size
  200  continue
c
  300  fv=ff
       s = size * d
       x = x + s

c Step 4 行列 H を更新する。

       w = g
       g = Df(x)
       if( absmax( g ) .lt. eps ) go to 20
       y = g - w
c
```

```
      yHy = y' * H * y
      sy  = s' * y
c
c BFGS
        H = H + (1+yHy/sy)*(s*s') / sy  - (H*y*s'+s*y'*H) / sy

c Step 5 k = k + 1 として Step_1 へ行く。

      k = k + 1
      go to 10

c Step 6 終了プロセス
   20 continue
      write(*,*) ' f=',fv
      write(*,*) ' value of x **********************'
      call mprint( x )
      stop
      end

c 目的関数 f(x) の定義
      real*8 function f(x)
      parameter ( n = 4 )
      real*8:vector[n] x
       xa = x[1] + 10.0d0*x[2]
       xb = x[3] - x[4]
       xc = x[2] - 2.0d0*x[3]
       xd = x[1] - x[4]
       f  = xa*xa + 5.0d0*xb*xb + xc**4 + 10.0d0*xd**4
      end

c 目的関数の 1 階偏導関数 Df(x) の定義
      real*8 :vector[*] function Df(x)
      parameter ( n = 4 )
      real*8:vector[n] x
       allocate Df:vector[n]
       xa = x[1] + 10.0d0*x[2]
       xb = x[3] - x[4]
       xc = x[2] - 2.0d0*x[3]
       xd = x[1] - x[4]
       Df[1]=  2.0d0*xa + 40.0d0*xd**3
       Df[2]= 20.0d0*xa +  4.0d0*xc**3
       Df[3]= 10.0d0*xb -  8.0d0*xc**3
       Df[4]=-10.0d0*xb - 40.0d0*xd**3
      end
```

このプログラムを実行した結果を示す。

```
 step=   0   f=.757D+03  norm g=.975D+03
 step=   1   f=.634D+02  norm g=.123D+03
 step=   2   f=.360D+02  norm g=.729D+02
 step=   3   f=.249D+02  norm g=.659D+02
 step=   4   f=.151D+02  norm g=.416D+02
 step=   5   f=.476D+01  norm g=.248D+02
 step=   6   f=.232D+01  norm g=.136D+02
 step=   7   f=.148D+01  norm g=.961D+01
                :
              (略)
                :
 step=  27   f=.164D-06  norm g=.500D-04
 step=  28   f=.547D-07  norm g=.233D-04
 step=  29   f=.184D-07  norm g=.108D-04
 step=  30   f=.626D-08  norm g=.493D-05
 step=  31   f=.216D-08  norm g=.226D-05
 step=  32   f=.750D-09  norm g=.103D-05
 f=  2.546376057432119E-010
 x=
%%MPRINT VECTOR          [    4,    1 ]
[   1,   1]   .212367198871246E-02
[   2,   1]  -.212366902391035E-03
[   3,   1]  -.122694424498447E-03
[   4,   1]  -.122694600952795E-03
Stop - Program terminated.
```

このプログラムの記述時間とデバッグ時間は約 1 時間 30 分程度であった。テキスト [4] に記述されたアルゴリズムを見て、ただちにプログラミングにとりかかることができた。また、このプログラムのアルゴリズムを変更する場合は、公式を代える程度なら、その式をそのまま変更すればよいので、これもまたただちに行うことができる。

参考文献

[1] American National Standard FORTRAN (ANS X3.9-1966), American National Standards Institute, New York, 1966

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, LAPACK User's Guide, SIAM, Philadelphia, 1992

[3] A. V. Aho, and J. D. Ullman, Principles of Compiler Design, Addison-Wesley Publishing Company, Inc. 1977, (邦訳：土居範久訳、コンパイラ、培風館、東京、1986)

[4] ASNOP 研究会編、非線形最適化プログラミング、日刊工業新聞社、東京、1991

[5] J. Backus, The History of FORTRAN I, II, and III, ACM Sigplan Notices, 13, 8, August, pp. 165-180, 1978

[6] R.F. Boisvert, S.E. Howe, and D.K. Kahaner. GAMS — a framework for the management of scientific software, ACM Trans. Math. Software, No. 11, pp. 313-355, 1985

[7] Champman, B., Mehotra, P. and Zima, H., Programming in Vienna Fortran, In Proc. 1992 3rd Workshop on Compilers for Parallel Computers, pp. 121-160, 1992

[8] J. J. Dongarra, C. B. Moler, J. R. Bunch, G. W. Stewart, LINPACK User's Guide, SIAM, Philadelphia, 1979

[9] Draft proposed ANS FORTRAN BSR x3.9 x353/76, ACM Sigplan Notices, 11, 3, March, 1976

[10] Ellis M. A. and Stroustrup B., The Annotated C++ Reference Manual, Addison-Wesley, New York, 1990

[11] Hiranandani, S., kennedy, K. and Tseng, C., Compiling Fortran D, comm of the ACM, Vol. 35, No. 8, pp. 60-80, 1992

[12] 平山弘、C++言語による無限級数の加速法、日本応用数理学会論文誌、Vol. 4, No.6, pp.327-336, 1994

[13] HITAC プログラムプロダクト VOS3 行列計算副プログラムライブラリ MATRIX/HAP、MATRIX/M、日立製作所、1987

[14] HITAC プログラムプロダクト VOS3 DEQSOL E2、バウンダリ・フィット法、差分法、有限要素法

[15] HITACHI 最適化 FORTRAN77 使用の手引き (プログラムプロダクト VOS3 最適化 FORTRAN77 OFORT77 E2、HAP FORTRAN77 使用の手引き)[手引き書 6180-3-732-10] 日立製作所、1991

[16] 本郷茂、行列演算用のサブプログラムパッケージとその使い方、青山コンピュータ・サイエンス、Vol.6、No.2、pp.63-83、1978

[17] 本郷茂、内田智史、行列演算用言語 LAMAX-E の行列演算処理機構について、第 25 回情報処理学会全国大会論文集、2C-7、pp.347-348、1982

[18] 本郷茂、内田智史、行列演算用言語 LAMAX-E 例題集、青山コンピュータサイエンス、Vol.10、No.2、pp.37-88、1982

[19] 本郷茂、内田智史、行列演算用言語 LAMAX-E の紹介 (その 1)、東京大学大型計算機センターニュース、Vol.15、No.11、pp.137-152、1983

[20] 本郷茂、内田智史、行列演算用言語 LAMAX-E 利用の手引、東京大学大型計算機センター、1984

[21] 本郷茂、内田智史、行列演算用言語 LAMAX-E、北海道大学大型計算センターライブラリ説明書、No.15、1984

[22] 本郷茂、内田智史、行列演算用言語 LAMAX-E の紹介 (その 2)、東京大学大型計算機センターニュース、Vol.16、No.5、pp.35-39、1984

[23] 本郷茂、八巻直一、内田智史、行列演算用言語 LAMAX-S (2)、オペレーションズ・リサーチ、1993 年 3 月号、pp.154-159、1993

[24] 本多弘樹、自動並列化コンパイラ、情報処理、Vol. 34、No. 9、pp. 1150 - 1157、1993

[25] Ellis Horowitz, FUNDAMENTALS OF PROGRAMMING LANGUAGE, Computer Science Press, Maryland, 1983

[26] 井上美明、田中和彦、内田智史、八巻直一、本郷茂、間野浩太郎、行列演算用言語 LAMAX-S の開発 — 言語の特徴とその応用範囲 —、情報処理学会第 40 回全国大会、pp.696-697、1990

[27] Isner, J.F., A Fortran Programming Methodogy Based on Data Abstraction, *Comm. ACM*, Vol. 25, No. 10, pp. 686-697, 1982

[28] K. Iverson, A Programming Language, Wiley, New York, 1962

[29] B.W. Kernighan, D. Ritchie, The C Programming Language. Second Edition, (邦訳　石田晴久訳、プログラミング言語 C 第 2 版、共立出版、東京、1989)

[30] Koelbel, C. and Mehotra, P., Compiling Global Name-Space Parallel Loops for Distributed Execution, IEEE Trans. on Parallel and Distributed System, pp. 440-451, 1991

[31] ライブラリー・プログラム利用の手引き (数値計算編 : NUMPAC、Vol. 1-3)、名古屋大学大型計算機センター

[32] Liskov, B., Snyder, A., Atkinson, R. and Schaffert, G., Abstraction Mechanism in CLU, *Comm, ACM*, Vol. 20, No. 8, pp. 564-576, 1977

[33] Lowson, C., et al., Basic Linear Algebra Subroutines for Fortran Usage, ACM, Trans. Math. Software, No. 5, 1979

[34] M. Metcalf, J. Reid, Fortran 90 Explained, Oxford University Press , 1990 (Japanese version translated by H. Nishimura, H. Wada, M. Takata, Kyoritsu Shuppan Co., Ltd. 1993)

[35] Naur, P. ed., Revised Report on the Algorithmic Language ALGOL 60, Comm. ACM, 6, 1, pp. 1-17, 1963

[36] 二宮市三、科学技術計算への二つの提案、応用数理、Vol.2、No.1、1992

[37] 小国力編著、村田健朗、三好俊朗、ドンガラ, J.J.、長谷川秀彦著、行列計算ソフトウェア、丸善、東京、1991

[38] Padua, D. A. and Wolfe, M. J., Advanced Compiler Optimizations for Super Computers, Comm. of the ACM, Vol. 29, 1986

[39] John L. Pfaltz 著、間野浩太郎監訳, 電子計算機データ構造論、付録の部分、マグロウヒル好学社

[40] SAS Institute Inc., SAS/IML User's Guide, 5 Edition, SAS Institute Inc, 1985

[41] 寒川光、数値計算プログラミングにおけるデータ移動制御のためのブロック化アルゴリズム、情報処理学会論文誌、Vol.33、 No.10、 1992

[42] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, Matrix Eigensystem Routines - EISPACK Guide, Springer-Verlag, NewYork, 1976

[43] シドニー・ファーンバック、長島重夫訳、スーパーコンピュータ、パーソナルメディア、東京、1988 (原書 : SUPERCOMPUTER Class VI Systems, Hardware and Software Edited by Sindey Fernbach, Elsevier Science Publishers B. V. (North-Holland), 1986

[44] SX ソフトウェア科学技術計算ライブラリ ASL/SX 利用の手引、NEC 日本電気株式会社、1992

[45] 高橋悟、宮田雅智、本郷茂、矢部博、八巻直一、非線形最適化問題解法ソフトウェア ASNOP、オペレーションズ・リサーチ、Vol. 33、 No. 11、 pp. 584-587、 1988

[46] 高橋悟、本郷茂、矢部博、宮田雅智、八巻直一、非線形最適化問題のためのアプリケーション・システム (ASNOP 利用の手引き)、東京大学大型計算機センター、 1988

[47] 津田孝夫、岩波講座 ソフトウェア科学 9 数値処理プログラミング、岩波書店、東京、1988

[48] 戸川隼人、マトリクスの数値計算、オーム社、東京、1979

[49] 戸川隼人、UNIX ワークステーションによる科学技術計算ハンドブック、サイエンス社、東京、1992

[50] 内田智史、本郷茂、プリコンパイラ生成システム、青山コンピュータサイエンス、Vol.8、 No.1、 pp.45-71、 1980

[51] 内田智史、本郷茂、行列演算用言語 LAMAX、第 23 回情報処理学会全国大会論文集 3H-2、 pp.179-180、 1981

[52] 内田智史、本郷茂、行列演算用言語 LAMAX-E、青山コンピュータサイエンス、Vol.9、 No.2、 pp.51-79、 1981

[53] 内田智史、本郷茂、 行列演算用言語 LAMAX-E 入門、青山コンピュータサイエンス、Vol.10、 No.1、 pp.1-25、 1982

[54] 内田智史、本郷茂、 行列演算用言語 LAMAX-E の文法とそのプリコンパイラプロセッサについて、第 25 回情報処理学会全国大会論文集、2C-6、 pp.345-346、 1982

[55] 内田智史、本郷茂、行列演算用言語 LAMAX-E の性能評価と IAP への適合性、第 29 回情報処理学会全国大会論文集、 2P-5、 pp.361-362、 1984

[56] 内田智史、本郷茂、間野浩太郎、スーパーコンピュータ向け行列演算用言語 LAMAX-S の設計思想、第 35 回情報処理学会全国大会講演論文集、 6R-1、 pp.879-880、 1987

[57] Uchida, S., Kimura, E., Mano, K., The Debugging System for Program Reusable Environment Based on Data Abstraction、Aoyama Computer Science, Vol.14, No.2, pp.23-44, 1987

[58] Uchida, S., Kimura, E., Mano, K., "A COMPACT PROGRAMMING ENVIRONMENT FOR MICROPROCESSOR-BASED CONTROLLERS"、computers & industrial engineering (an international journal), Vol.13, pp.257-261, (PROCEEDINGS OF THE 9th ANNUAL CONFERENCE ON COMPUTERS AND INDUSTRIAL ENGINEERING), 1987

[59] 内田智史、間野浩太郎、プリアセンブリ言語 ADT-RASM86 とそのデータ抽象化機構、情報処理学会論文誌、 Vol.28、 No.10、 pp.1040-1050、 1987

[60] 内田智史、本郷茂、間野浩太郎、スーパーコンピュータ向け行列演算用言語 LAMAX-S、数値解析シンポジウム、 1988

[61] 内田智史、本郷茂、間野浩太郎、スーパーコンピュータ向け行列演算用言語 LAMAX-S について、— オブジェクト指向に基づく行列演算パッケージのパイロットバージョンの開発 —、数値情報環境の理論と実用システム開発、数値環境研究会論文集 改訂第二版増補版 第 1 巻 第 III 部第 11 章 pp. III-11-1 -III.11.6.5、 1989

[62] 内田智史、本郷茂、八巻直一、間野浩太郎、オブジェクト指向に基づくスーパーコンピュータ用行列演算パッケージ LAMAX-S、数値解析シンポジウム、 1989

[63] 内田智史、井上美明、田中和彦、八巻直一、本郷茂、間野浩太郎、情報処理学会第 40 回全国大会、pp.698-699、行列演算用言語 LAMAX-S の開発 — 実現方法 —、1990

[64] 内田智史、本郷茂、八巻直一、行列演算用言語 LAMAX-S(PC98 版)、日本オペレーションズ・リサーチ学会 1992 年度秋季研究発表会アブストラクト集、I-4(事例研究奨励賞・ソフトウェア部門賞)、pp.34-35、1992

[65] 内田智史、八巻直一、本郷茂、井上美明、唐澤豊、行列の数学的特性を重視した言語 LAMAX-S の設計思想とその処理系について、日本応用数理学会論文誌、Vol.2、 No.3、 pp.155-168、1992

[66] 内田智史、本郷茂、八巻直一、行列演算用言語 LAMAX-S(3)、オペレーションズ・リサーチ、1993 年 4 月号、pp.204-207、1993

[67] 内田智史、八巻直一、本郷茂、唐澤豊、個々のコンピュータ特性を考慮したプリプロセッサ用の最適化コードジェネレータ、数値計算技術の基礎理論、数理解析研究所講究録 832、151-161、1993

[68] 内田智史、八巻直一、本郷茂、唐澤豊、池辺八洲彦、行列演算用言語 LAMAX-S 処理系の最適化コード生成、日本応用数理学会論文誌、 Vol. 5、 No. 1、 pp. 67-85、 1995

[69] Satoshi Uchida, Naokazu Yamaki, Shigeru Hongo, Enhanced Programming Environment for Mathematical Software, Proceedings of the Third Conference of the Association of Asian-Pacific Operational Research Societies (APORS) within IFORS, World Scientific, 1995

[70] 八巻直一、矢部博、宮田雅智、本郷茂、非線形最適化問題解決のパッケージ (I)、青山コンピュータサイエンス、Vol. 10、 No.2、 pp. 1-35、 1982

[71] 八巻直一、内田智史、本郷茂、行列演算用言語 LAMAX-S (1)、オペレーションズ・リサーチ、1993 年 2 月号、pp.80-91、1993

[72] Wolfe, M., Optimizing Supercompilers for Supercomputers, The MIT Press, 1989

[73] Yanjie Zhao, Horoshi Sugiura, Tatsuo Torii, Tetsuya Sakurai, A Knowledge-Based Method for Mathematical Notations Understanding, 情報処理学会論文誌、No. 11, pp. 2366-2381, 1994