NAIST-IS-DT0061008

博士論文

統計解析システムにおけるプログラミング言語の研究

小林 郁典

2002年 2月 5日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

小林 郁典

審査委員：　渡邉 勝正 教授
杉本 謙二 教授
関　浩之 教授
木村 晋二 助教授

統計解析システムにおけるプログラミング言語の研究*

小林 郁典

内容梗概

統計解析システムは，実験，観測，調査で得られたデータを様々な視点で眺め，いろいろな種類の要約統計量を計算し，図表を描き，測定した現象の背景にある本質を探るための環境である．これらを具体的に行うための手法は，統計学の成果として数多く報告され，統計解析システムの上で実現される．しかし，統計解析は，データに対して定型的にこれらの手法を適用すれば目的を達成できるという単純なものではなく，データによって手法を使い分けたり，評価のための判断基準を変更したりしなければならないことがある．このような作業を行うために，統計解析システムには専用のプログラミング言語が要求される．統計解析システムのプログラミング言語は，統計解析システムの可能性を広げるための重要な道具と位置づけられる．

この論文は，３つの統計解析システム（重回帰分析支援システム RASS，その新版としての RASS2，Java 言語ベースの汎用統計解析システム Jasp）のプログラミング言語をどのように設計・実装したのか，これらを使用することでどのような効果があるのかを述べるとともに，これらを実際に実現するために必要であった関連技術に関するいくつかの研究成果について述べたものである．これらの言語は，利用者が簡単にしかも安定的にプログラミングができるように工夫されている．

本論文は６章で構成される．第１章の緒論では，本論文の位置づけと目的を明確にし，本論文の構成を紹介する．

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DT0061008, 2002 年 2月 5日.

２章では，知的データ解析支援システムに求められるプログラミング言語の条件について考察するとともに，その具体例として実装した知識ベースの重回帰分析支援システム RASS (Regression Analysis Supporting System) のプログラミング言語 (RASS 言語) について説明する．RASS 言語の設計においては，RASS が持つ支援機構の枠組みを崩さないこと，知識処理のための論理型プログラミングと手法を記述するための手続き型プログラミングが混在できること，アルゴリズムを容易に記述できることなどに注意した．

３章では，RASS を拡張した RASS2 とそのプログラミング言語について紹介する．RASS2 は，重回帰分析を知的に支援する環境であるだけでなく，さらに，その構成や内容を柔軟に変更できるように設計された．これを実現するために，オブジェクト指向のオブジェクトを局所的な重みを持つルールを持つように拡張し，そのルールに基づいて自発的に利用者の解析を支援することができるモジュールを基本構成要素とした．RASS2 には，このモジュールを記述するためのプログラミング言語が備わっている．

４章では，対話型プログラミング環境でオブジェクトに機能を簡単に追加する方法を提案し，その実装方法について紹介する．この成果は，統計解析システムのプログラミング言語にも応用することができる．これを実現するために，条件部と実行部から構成される特殊なメソッドであるアクティブルールをオブジェクトの枠組みの中に組み込み，アクティブルールに基づいてそのオブジェクトが自発的に振る舞うようにしている．この機能を利用すれば，対話型プログラミング上でオブジェクトを操作する利用者は，従来の方法と比べて簡単にオブジェクトを拡張させることができる．

５章では，汎用統計システム Jasp (JAva based Statistical Processor) のプログラミング言語の機能と文法について説明する．Jasp は，Java 言語ベースで開発されており，先進的なコンピュータ技術を統計解析において簡単に利用できることを目指している．Jasp 言語は，関数の記述を基本とした手続き型のスクリプト言語であり，そして，できるだけ簡便にオブジェクト指向プログラミングができる枠組みと柔軟性，拡張性を兼ね備えるように設計されている．Jasp 言語の実装には，Java 言語のスクリプト言語である Pnuts を利用した．

最後の６章で，本研究で得られた成果を要約し，統計解析システム用のプログラミング言語の開発における指針を述べる．

キーワード

統計解析システム, プログラミング言語, 言語設計, RASS, Jasp

Studies on Programming Languages of Statistical Analysis Systems*

Ikunori Kobayashi

Abstract

Statistical analysis systems are environments for viewing and browsing data, calculating statistics, drawing statistical charts, and exploring essences of the phenomenon. Now, plenty of methods to analyze data are proposed and used on such systems, but these methods are often changed or expanded properly according to data. Special programming languages are indispensable to the systems for such tasks. The programming languages of statistical analysis systems are tools for developing their abilities.

This thesis describes how to have designed and implemented the programming languages for RASS (Regression Analysis Supporting System), RASS2 and Jasp (Java based statistical processor), and some results of related researches. RASS is a kind of expert systems for assisting user's regression analysis, and Jasp is one of general purpose statistical analysis systems. The languages for these systems have been designed to be able to program easily and stably for end users.

This thesis consists of 6 sections. The aims and backgrounds of this research are clarified in Chapter 1.

Chapter 2 describes requirements for a programming language of knowledge base supporting systems for regression analysis, and explains how to have designed and implemented the programming language for RASS as the concrete

*Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0061008, February 5, 2002.

example. RASS language has been designed not to break the framework of the supporting mechanism, to combine a logic language and a procedural language and to represent statistical algorithms easily and simply.

Chapter 3 describes RASS2 and its programming language. RASS2 is an improvement of RASS to provide consulting mechanisms with high flexibility and extendibility to users. To achieve this, extended objects which define rules internally are used as the basic components of RASS2. Rules in extended objects have local priorities and are evaluated spontaneously. The RASS2 language is an environment to define extended objects for users.

In Chapter 4, a simple extending method of objects on the interactive programming and prototyping environment in object-oriented script languages is proposed. This results can be applied also to the language of statistical analysis systems. To implement this facility, active rules which are specialized methods based on "condition-acton" form are included into the frame of objects, and the active rules are made to behave spontaneously. Users can enhance objects in an easier way compared with general methods.

Chapter 5 introduces the features and the syntax of the language of the general purpose statistical system Jasp. The Jasp language is a procedural function-based script language and has object-oriented mechanism which provides much of ease, flexibility and extendibility. To develop this language, Pnuts, a script language for Java, is used for the base.

The results achieved in the above researches are summarized in Chapter 6, and design principles for developing programming languages of statistical analysis systems are proposed.

Keywords:

Statistical analysis system, Programming language, Language design, RASS, Jasp

目 次

図 目 次

表 目 次

1. 緒論

本章では，研究の対象としてのデータ解析用ソフトウェアとそのプログラミング言語について説明し，その後，研究の目的と本論文の構成について述べる．

1.1 データ解析のためのソフトウェア

データ解析とは，実験，観測，調査で得られたデータを様々な視点で眺め，いろいろな種類の要約統計量を計算し，図表を描き，測定した現象の背景にある本質を探る手法の総称である．簡単には“ データの特徴を把握する方法 ”とも言えるであろう．このデータ解析をコンピュータ上で行うためのソフトウェアが，データ解析用ソフトウェアである．

データ解析用ソフトウェアは，コンピュータが発明された頃から研究・開発されてきた．現在も数多くのデータ解析用ソフトウェアが，コンピュータ技術の発展に伴いながら使いやすい環境でより高度な解析ができるように改良されている．データ解析用ソフトウェアは，用途や規模によってさまざまな呼び方が存在する．実際には明確に区別して使われていないこともあるが，おおまかにデータ解析用ソフトウェアを分類すると図 1.1のように区分される．

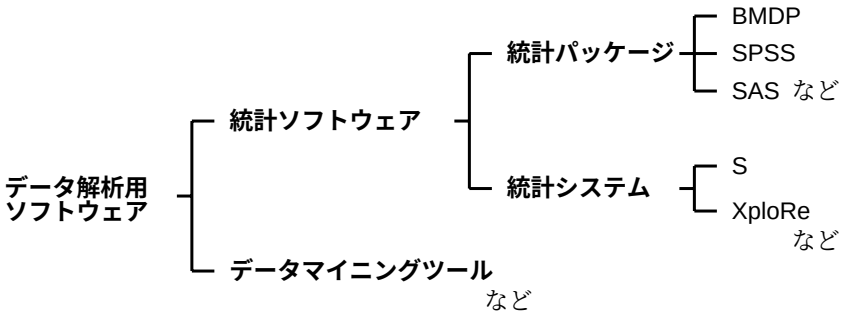

図 1.1 データ解析用ソフトウェアの分類

まず，データ解析用ソフトウェアは，統計手法を主として用いるかどうかで大きく分けられる．統計的な手法を用い，データから集団としての特徴を明らかに

するための情報を計算することに利用されるのが，統計ソフトウェアである．データマイニングツールと言われるものは，データに隠れている知識や，パターン，新しい規則を発見するためのソフトウェアである．主として統計的な手法を利用しないので，統計ソフトウェアとは区別される．

さらに，統計ソフトウェアは，統計パッケージと統計システム (“ 統計解析システム ”とも言われる) に分かれる．

統計パッケージは，統計解析のためのサブルーチンや関数をライブラリとしてまとめたもののことを指し，その多くは，Fortran や BASIC のような高水準言語で作成されている．1961 年に統計計算のための汎用統計パッケージの先駆けとして BMD (biomedical computer program) がカリフォルニア大学で開発された．その後，新しいプログラムの追加や改良が行われ，現在では BMDP [17] として商品化されている．また，シカゴ大学で開発された SPSS (Statistical Package for the Social Science)[28] は，社会科学の分野での利用を目的とした統計パッケージである．ノースカロライナ大学で開発された SAS (Statistical Analysis System)[26] は，データ処理に優れ，モデリングなどの機能を含む統合的なシステムを目指したものである．SPSS と SAS も BMDP のように商品化され，これら 3 つを世界の 3 大統計パッケージと呼ぶことがある．日本では，岡山大学で開発された統計解析ハンドブック [54, 46] が，有名な統計パッケージのひとつである．

統計パッケージを利用するためには，統計パッケージの使い方と統計パッケージを利用できる汎用プログラミング言語を習得し，さらに，ファイルから解析データを読みとり，統計パッケージを呼出し，結果を整理するプログラムを利用者が注意深く作成する必要がある．

このような煩わしさを解消し，利用者が簡単に統計解析をすることができるように統計パッケージを発展させたものが統計システムである．統計システムは，データの準備から結果の出力までをひとつの環境の中で行え，組み込まれていない手法を実行できるように専用の対話型プログラミング機能を持つ．S は，データ解析とグラフィックスのための対話型環境を目指して 1980 年代初頭にベル研究所で開発されたものである [6]．その後，データ解析に特化した汎用言語（S 言語）として拡張され，最も普及した統計システムとなった [7]．2000 年にドイツ

のフンボルト大学で開発された XploRe[10] は，設計段階から Window 環境を強く意識した高度な操作性と比較的新しい統計手法を持つ統計システムである．これ以外にも数多くの統計システムが公開・販売されている [44]．さらに，最近では，OMEGA プロジェクト [22] のように新しいタイプの統計システムの開発も進められている．

統計パッケージ及び統計システムは，それぞれ，図 1.1で紹介した汎用的なものと，特殊用途向けのものが存在する．例えば，東京大学で開発された SALS[49] は，最小二乗法を中心とするデータ解析のための特殊な統計パッケージである．また，1970 年代後半からブリティッシュ・コロンビア大学で開発された SHAZAM[35, 36] は，計量経済学のために作られた統計パッケージである．知識工学の全盛期の頃には，知識ベースの支援機構を導入した統計解析ソフトウェアの研究も行われていた [8]．

また，最近では，図 1.1に示したような枠組みでデータ解析用ソフトウェアを区分することができなくなってきている．例えば，データマイニングの機能を追加した統計システムが開発されているし，統計パッケージとして紹介した３大パッケージも統計システムと区別できないほど使いやすく改良されている．図中では紹介しなかったが，表計算ソフトも，データ解析が可能な性能を持つように発達しているし，統計解析を可能にする組込み型マクロの開発も行われている．このように，データ解析用ソフトウェアは，今後もこれまでの技術的財産や経験を活かしながら，新しいコンピュータ技術を吸収し，さらに発展していくであろう．

1.2　統計解析システムの役割

一般的に，汎用的な統計解析システムが利用者（データ解析者）に代わって担う主な作業は，(1) 解析データの管理，(2) 各種統計手法の実現，(3) 計算結果及び統計図の表示・印刷，に分類される．統計解析システムは，これらの作業をいかに簡単かつ正確に実行できるかどうかで，その能力が評価される．

具体的には，解析データの管理では，他のファイル形式からのインポートや各種分布系に従った乱数の生成などを行える機能（データ生成機能）やデータのソート，マスキング，削除，抽出，変換，型指定などを行える機能（データ編集機能）

が求められる．

２つめの各種統計手法の実現とは，用意した解析データに各種統計手法を適用することである．代表的な統計手法としては，１変数に着目すると，平均や分散，中央値，尖度などの基本統計量や正規性の検定手法が，２変量に着目すると，共分散行列や相関行列の作成，カイ２乗検定，単回帰分析などが，多変量に着目すると，主成分分析や重回帰分析，クラスター分析などが挙げられる．一般的には，利用される統計解析手法のほとんどは，統計解析システムにあらかじめ組み込まれており，それらを何らかの方法で呼び出すことで適切なモデルや必要な統計量を得ることができる．ただし，統計解析システムに解析で使われる全ての手法をあらかじめ組み込ませておくことは困難なので，利用者が既存の手法を編集したり，新しく手法を作成することができる仕組みが統計解析システムには要求される．

３つめの計算結果及び統計図の表示・印刷では，解析途中で計算された数多くの統計量から必要なものを選択し，見た目よくディスプレイやプリンタに表示させるような機能と，視覚的にデータの特性を表現するヒストグラムや箱髭図，幹葉図，散布図などの統計図の作成や調整のための機能が求められる．

1.3　統計解析システムのプログラミング言語

実際の統計解析の作業は，大きく二通りに分かれる．ひとつめは，解析対象のデータの特性が分かっていて，データに対して定型的な手順を用いることで解析ができる場合である．集計表やグラフを作成したり，平均や標準偏差などの統計量を計算するような仕事である．このような目的のためには，統計パッケージや，統計解析の機能を持つ表計算ソフトを利用すれば効率よく解析が行える．もうひとつは，よく分からない対象から得たデータの解析を試みる場合や，実験結果から何らかの意味のある事実を導き出したいという場合である．これは，いわゆる問題解決を目的とした仕事に対応する．このような解析は，簡単ではない．なぜなら，生データから必要な情報を抽出・加工したり，いろいろなモデルをデータに当てはめたりすることを探索的に行う必要があるからである．このような非定型的な作業を行うための機能をあらかじめ統計ソフトウェアに組み込ませておくこ

とは困難なので，解析者は，統計ソフトウェアの持つ機能をうまく活用し，制御しながら，このような作業を遂行しなければならない．従って，統計ソフトウェアには，利用者がやりたいことを表現できるような機能，つまりプログラミング言語が必要になる．

統計解析システムのプログラミング言語は，1.2節で紹介した統計解析システムの仕事を基本的にはすべて表現できなければならない．具体的には，解析データをシステム上で保持するためのデータ構造とその操作命令，各種統計量を求めるための数値演算記述能力（行列演算を含む），統計手法の呼出し，図表の制御命令，乱数や確率分布の管理などである．統計解析システムのプログラミング言語は，“ なんでも表現できる ”汎用プログラミング言語とは異なり，統計解析のために特化（限定）された応用指向型のプログラミング言語として位置づけられる．

統計解析システムの利用者が作成しようとするプログラムは，再利用される頻度はあまり高くない．また，利用者は，統計解析をすることが主であり，プログラムを作成するためにはあまり時間を費やしたくないと思うのが一般的である．従って，統計解析システム上で実行されるプログラム，つまりプログラミング言語は，その実行効率よりも，そのプログラムの作成を効率良く行えることが重要視される．このような要求には，人間の思考形態により近い形でアルゴリズムを表現できる高水準のプログラミング言語がふさわしい．また，これは，利用者がプログラムを作成するときだけの要求ではなく，組み込まれている手法を確認するときの要求でもある．いくら統計解析システムが優れていても，誤って利用すれば，統計的に正しい結果は得られない．利用者は，必要に応じて利用する手法がどういうアルゴリズムで実装されているのかを確認したいことがある．このように，統計解析システムのプログラミング言語は，作り易さと読みやすさをできるだけ利用者に提供するように設計されなければならない．

1.4 本論文の目的

この論文は，著者がこれまでに開発したいくつかの統計解析システムとそのプログラミング言語について，どのように設計・実装したのか，どのような効果があったのかを説明するとともに，これらを実現するために必要であった関連技術

に関するいくつかの研究成果について述べたものである．

特に，統計解析システムにとってプログラミング言語はどのように機能すべきなのか，システム上で利用者が実施したい作業（要求）をできるだけ少ない労力でうまく表現するためにはどのようにプログラミング言語を設計し，どのような技術を用いて実装すればよいのかを明らかにすることが本論文の目的である．

この目的を達成するために，いくつかのプログラミング言語を開発している．開発したプログラミング言語のひとつは，統計解析でよく利用される重回帰分析を知的に支援する機能を持つ統計システムのためのものである．支援機構を持つ特殊な統計システムの枠組みを壊さずに，システムの持つ最大限の機能を引き出せるようにプログラミング機能を設計し，実装している．もうひとつは，汎用統計システムのプログラミング言語である．このプログラミング言語は，統計計算の記述だけでなく，システムそのものを拡張できるように設計されたものである．

1.5 本論文の構成

本論文は6章で構成される．本章に続く2章では，先述した知的データ解析支援システムに求められるプログラミング言語の条件について考察するとともに，その具体例として知識ベースの重回帰分析支援システム RASS (Regression Analysis Supporting System) のプログラミング言語をどのように設計したのかについて説明する．RASS のプログラミング言語の設計においては，RASS の支援機構の枠組みを崩さないこと，論理型プログラミングと手続き型プログラミングが混在できること，アルゴリズムを容易に記述できることなどに注意している．

3章では，RASS を拡張した RASS2 とそのプログラミング言語について紹介する．RASS2 は，重回帰分析を知的に支援する環境であるだけでなく，その構成や内容を柔軟に変更できる環境を提供することもその目的として設計された．これを実現するために，オブジェクト指向のオブジェクトを，局所的な重みを持つルールを持つように拡張し，そのルールに基づいて自発的に振舞うモジュールを基本構成要素としている．RASS2 には，このモジュールを記述するためのプログラミング言語が備わっている．

4章では，前章で紹介した RASS2 言語の拡張したオブジェクト内で表現した

ような自発的振舞いのためのルールを活用すれば，対話型プログラミング環境で簡便にオブジェクトを定義したり，拡張したりすることができることを紹介する．前述した通り，対話型プログラミング環境は，多くの統計システムで実装されている利用者とのインタフェースである．この研究成果は，このような統計システムで有効に利用できる．

５章では，汎用統計システム Jasp (JAva based Statistical Processor) のプログラミング言語について説明する．Jasp は，Java 言語ベースで開発されており，先進的なコンピュータ技術を統計解析において簡単に利用できることを目指している．Jasp では，統合化されたグラフィカルなインタフェースとキャラクタベースのインタフェース，大規模なデータ処理のための分散処理，他言語とのインタフェースを含む高い拡張性，などを実現している．Jasp のプログラミング言語は，小規模なプログラム開発を対象としているだけでなく，1.3節で述べたような統計システムの機能を利用者が平易な文法で制御することができるように設計上の工夫が施されている．６章で，本論文をまとめる．

2. 知識ベースの支援機構を持つ統計解析システムRASSのプログラミング言語

2.1 はじめに

計算機科学における知識工学やエキスパートシステムの研究の進展にともない，データ解析の分野でも，知識処理機構を持ち，自動解析や手法選択の支援などを行える知的システムの研究が進められてきた．中野らの開発した重回帰分析支援システム RASS (Regression Analysis Supporting System)[50, 19] もそのひとつである．RASS は，グラフィックス表示を含む重回帰分析の基本的な手法を容易に実行でき，またそれらの明らかな誤用をシステム自身ができるだけ防止することを目標として作成された．すなわち，重回帰分析の基本的統計計算機能の他にそれらの利用に関する統計的知識も組み込まれており，利用者が統計手法の選択に困難を感じるような場合には，それまでの解析結果に応じてある程度の助言を行うことができる．また，RASS の知識だけを用いた自動解析も可能である．

ところで，一般にシステム利用者の要求は多様であり，どのように多くの機能をシステムに持たせようと，それですべての利用者を満足させることはできない．従って，多くのシステムはマクロ機能，または拡張用言語を持ち，利用者が自分自身のための新しい命令をプログラミングできるようにしている．データ解析においても，ひとつの仮説を検証するためにも複数の統計量が考えられ，それらのすべてをはじめから組み込んでおくことは不可能であるから，拡張用言語は不可欠なものである．

しかしながら，これまで発表された知識処理機構を持ったデータ解析システムにおいて，機能拡張用言語を備えたものはあまり見受けられない [9]．その理由のひとつとして考えられることは，このようなシステムにおいては新しい計算手法を作成する場合，内部構造に合致するようにプログラムする必要があるということである．すなわち，単に新しい計算手法を定義できるだけでは知的システムの拡張用言語としては不十分であり，その言語でプログラムを書けば自然にそれまでのシステムと整合性を保てるようになっていなければならない．ところが，知的システムにおいては統計計算手法とその利用のための知識が複雑に関係付けら

れているため，そのような言語を実現することが容易ではなかったと思われる．

RASS は，重回帰分析の標準的な順序という知識を表現し，またそれに関する統計手法を解析の段階に応じて適用できるように，オブジェクト指向の考え方を基本として設計されている．そのため，手法と知識は最初から自然に結びつけられている．これらの機能を実現する際，システムの作成には論理型言語である Prolog を用い，数値計算の部分は C 言語でプログラムしてそれを Prolog の新しい述語とした．従って，新しい統計計算機能を組み込むことは可能ではあるが，そのためにはこれらのプログラミング言語の理解とシステム全体に対する詳細な知識が要求される．これでは一般ユーザがそれを実行することは難しい．すなわち，拡張用言語はないと言わざるをえない．そこで，利用者ができるだけ簡単に新しい統計計算機能を追加できるようにする目的で RASS 言語を開発した．その設計にあたっては，すでに実現されている部分と自然に整合性を保つこと，重回帰分析において多用される行列計算が容易に記述できること，論理型プログラミングと手続き型プログラミングが混在できること，などに注意した．本章では，知的データ解析支援システムに求められるプログラミング言語の条件について考察するとともに，その具体例として RASS 言語を説明する．

2.2 知的統計解析システムにおけるプログラミング言語

統計解析は，与えられたデータを加工していくつかの統計量を計算し，それらを見てデータの生成過程に関するなんらかの判断を下すという操作を，満足のいく結果が得られるまで繰り返し実行することである．知的統計解析システムは，このうちの統計計算作業のほとんど全てを実行するとともに，利用者の判断を助け，また，システム独自の判断をも提供する．

統計解析においては，なによりもそのデータに固有の知識が重要である．データ生成過程の固有技術的性質から，データには多くの制約が課されているのが普通であり，それらを考慮しない統計解析には意味がない．このようなデータに関する知識は個々のデータごとに特有のものであり，汎用システムの中に一般的知識として保存しておき，後で再利用することは難しい．また，これを統計解析手法と結びつけ，意味のある解析を行うことは，統計学およびその固有技術分野の

広範な知識を持つ “専門家” でなければできない．それと同じ作業を現在の知識工学の手法によって計算機上で実現することはほとんど不可能であろう．このため，知的統計解析システムに可能な判断は，統計学の手法に関する限定的なものとならざるを得ない．したがって，このようなシステムに現在要求されていることは，まったく新しい種類のデータを解析して専門家並みのよい解析結果を得ることではなく，むしろ定型的な統計処理に対して大きな誤用がないようにすることであると考えられる．

ただし，このように“ 保守的な”システムにおいても，新しい計算手法を組み込む手段としての言語は必要である．新しい有効な統計量が開発されたときにはそれをシステムに付加したいし，システムを利用して解析するデータが同じようなものばかりである場合には，固有技術から示唆される特殊な変数変換を組み込みたいこともあるだろう．

これまで多くのデータ解析システムにおいては，拡張用言語はその記述の自由度をできるだけ大きくする方向に進化してきた．例えば S[3] はもはや統計システムと言うより，統計関数とグラフィックス機能が豊富な汎用言語である．特にデータに対して種々の統計計算を試行錯誤的に容易に行える環境となることを重点において設計されている．そのため，プログラムとしては少々あいまいなものでもシステムが既定値として自動的に補うことによりできるだけ動作させようとする．

知的統計解析システムの拡張用言語としては，それほどの柔軟性は不必要である．思いついた手法をすぐ実行できることより，その手法とシステムにすでに貯蔵されている知識や推論機構との関係に注意しながら，汎用的に使える形で記述することのほうが重要である．それまでのシステムと整合的に新しい手法を組み込むことが比較的簡単に行えるという条件の下でのみ，自由度を高めなければならない．

もちろん，このことは言語の記述能力が低くてもよいということを意味するわけではない．利用者の要求を前もって予測することはできないので，言語は基本的には “なんでもできる”能力がなければならない．具体的にはシステム自身をその言語で再構成できるくらいの記述能力がなければ不便であろう．

さて，データを加工する際には一般に複雑な数値計算が必要である．その手順はアルゴリズムとして厳密に定義される．それをプログラムとして記述するには，Fortran や C 言語のような手続き型言語が便利である．また，行列計算は多用されるので，組み込み言語にはそれに対する配慮があることが望ましい．

一方，知的システムとして，計算された統計量からある判断を下すには，統計量の値の判断に関する知識データベースおよび推論のための機構が必要である．一回の判断に関する知識は簡単なアルゴリズムとして定義できるくらいのものであるが，解析が進んでいく過程においては統計量の値によって非常に多くの場合わけを経ることになる．従って，全体を手続き型のアルゴリズムとして記述することは困難であり，いわゆるエキスパートシステム技術を利用した知識データ表現と推論機構で実現することになる．そして，これらの機能はリスト処理とパターンマッチングによってプログラミングするのが便利であることが知られており，それらの操作が容易であるように設計された Lisp や Prolog 言語などを用いて実現することが多い．従って，知的データ解析システムの拡張用言語としては，統計計算のための手続き型の部分と知識処理のための部分が自然なかたちで結び付いているように設計するべきであろう．

2.3 RASS の特徴

本節では，言語機能を備える以前の RASS の仕組みについて簡単に紹介する．

RASS はオブジェクト指向の考え方で設計されている．それはオブジェクト指向プログラミングが知識表現として使いやすいということが主たる理由であった．重回帰分析で用いられるデータや統計量を解析の段階に応じて分類し，それらをクラスオブジェクト (以後，クラスという) の属性としてまとめると，解析の順序に関する知識は自然に表現されることになる．また，適用できる統計計算手法はクラスに付随するメソッドとして定義しており，それは解析のどの段階でどの手法を用いるべきかという知識をあらわすことにもなる．解析を進めるには，クラスの実現であるインスタンスオブジェクト (以後，インスタンスという) にメッセージを送るという操作を繰り返す．

RASS は，UNIX 環境上で稼動する．オブジェクト指向の枠組みと推論のため

```
:- load_foreign(['lsdiag.o', init_lsdiag).

class(lSDiag, lSRes).

slot(lSDiag, pred,     null, ' 予測値').
slot(lSDiag, res,      null, ' 残差').
slot(lSDiag, stdRes,   null, ' 標準化残差').
slot(lSDiag, stuRes,   null, ' スチューデント化残差').
slot(lSDiag, leverage, null, ' レベレッジ').
slot(lSDiag, dffits,   null, 'DEFFITS').
slot(lSDiag, cook,     null, 'COOK-D').
slot(lSDiag, dw,       null, 'Durbin-Watson').

method(lSDiag, initialize,[
  sm(self, name, Self),        % Getting 4 slot values in
  sm(Self, depVar, Y),         % the upper classes.
  sm(Self, indepVar, X),
  sm(Self, coef, Coef),
  c_lsdiag(X,Y,Coef,Pred,Res,StdRes,StuRes,Dw,Dffits,Cook,Lev),
  sm(Self, putValue, pred,     Pred), % Setting 8 slot values.
  sm(Self, putValue, res,      Res),
  sm(Self, putValue, stdRes,   StdRes),
  sm(Self, putValue, stuRes,   StuRes),
  sm(Self, putValue, leverage, Lev),
  sm(Self, putValue, dffits,   Dffits),
  sm(Self, putValue, cook,     Cook),
  sm(Self, putValue, dw,       Dw),
  sm(regAdv, created, Self)]).
```

図 2.1 以前の RASS のクラス lsDiag の定義（一部）

の処理を Prolog （SICStus-Prolog[30] と SWI-Prolog[37] 上で動作確認済み）で，数値計算の処理を C 言語 (GNU C Compiler) で記述している．さらに，ヒストグラムなどの図は Gnuplot を，表は tcl/tk を利用して表示させている．

ここで一例として，回帰診断のための統計量 DFFITS の計算手法（プログラム）がこれまでの RASS ではどのように実装されていたのかを紹介する．まず，回帰診断という概念は lsDiag というクラスで表現される．従って，DFFITS は，この lsDiag の属性として図 2.1 のように管理される．図 2.1は，これまでの RASS におけるクラス lsDiag の定義のためのプログラムの一部である．図中の 1 行目 `load_foreign()` は，SWI-Prolog で C 言語のモジュールを呼び出すことができるようにするための宣言である．述語 `class()`，`slot()`，`method()` によりそれぞれクラスの宣言，属性の宣言，メソッドの宣言を Prolog の事実として記述している．プログラム中では，クラス lsDiag のための 8 つのスロットとひとつのメソッドの定義を紹介した．このメソッド initialize は，クラス lsDiag のインスタンスが生成された後に呼び出されるスロットの値を充足するためのものである．Prolog を習得している利用者には，このプログラムを直感的に理解できるかもしれないが，属性への値の代入には，`putValue` という RASS 独自の命令が使われているし，Prolog 独自の変数やそのためのパターンマッチングが使われているので，このプログラムは一般にはわかりにくい．また，メソッド定義の中の `c_lsdiag()` で，処理を C 言語に移しているが，これに対応する C 言語プログラムも Prolog 側で利用されることを意識して作成されなければならない．図 2.2に，以前の RASS における DFFITS を計算するための C 言語プログラムの一部を示す．なお，図中の . . . は省略を表す.

Prolog から C 言語を呼び出す場合，基本的には独自の仕様に基づいて，C 言語の関数を定義し，その関数を専用の関数で登録する必要がある．また，Prolog には実数型の配列のようなデータ構造は無く，リスト型でそれらを管理しなければならない．従って，Prolog と C 言語の間では，基本データ構造の違いからデータをトランスレートするための処理をどちらかのプログラム上で記述する必要がある．RASS では，処理速度をはやくできる C 言語側でこの処理を実装している．図 2.2の詳細な説明は省くが，C 言語をよく理解している人でなければ，このプ

```
#include <stdio.h>
#include <math.h>
#include "system.h"
#include "nrutil.h"
#include "matrix.h"

FOREIGN_TYPE c_lsdiag(PROLOG_TERM pl_x,
          PROLOG_TERM pl_y,      PROLOG_TERM pl_coef,
          PROLOG_TERM pl_pred,   PROLOG_TERM pl_res,
          PROLOG_TERM pl_stdres, PROLOG_TERM pl_stures,
          PROLOG_TERM pl_dw,     PROLOG_TERM pl_dffits,
          PROLOG_TERM pl_cook,   PROLOG_TERM pl_leverage) {
  int status = TRUE;
  long i, j, row, col;
  double **hat, *e, *si2;
  double *dffits; /* for DFFITS */

  list2_size(pl_x, &row, &col); /* set row and col */
  hat = dmatrix(1, row, 1, row);
  dffits = dvector (1, row);
  ...

  for (i = 1; i <= row; i++) { /* DFFITS Statistic value. */
    dffits[i] = sqrt (hat[i][i] / (1.0 - hat[i][i]))
          * e[i] / sqrt (si2[i] * (1.0 - hat[i][i]));
  }
  ...
}

void init_lsdiag(void){
  PL_register_foreign ("c_lsdiag", 11, c_lsdiag, 0);
}
```

図 2.2 以前の RASS における DFFITS のための C 言語プログラム（一部）

ログラムを理解することは困難であろう．

このように，これまでの RASS の開発は，利用者の解析を支援するという主たる目的を実現することが優先され，実際の解析で必要となる新しい手法を実装したり，処理の流れを変更したりできるような仕組みを利用者に対して提供することが疎かになっていた．

2.4 RASS 言語の文法と特徴

RASS 言語は，前節の RASS の基本設計に基づいて統計手法をプログラミングできるように設計された．つまり，新しい統計手法はどれかのクラスのメソッドとして定義される．そのために必要な統計量は属性として保存する．ただし前節で考察したように，統計計算 (特に行列演算) とパターンマッチングに関してはそれらが容易に書けるように考慮した．

2.4.1 構文規則

RASS 言語の構文は，図 2.3の形式に従う．図中の'<' と'>' で囲まれた箇所が，利用者によって定義される部分である．プログラムの最初で，追加しようとする統計量あるいは手続きの属するクラス名を<CLASS>で指定する．このクラス名はすでに定義されているもの (主として RASS で定義されているクラス) でも，新たに定義しようとするクラスでもよい．<SUPER-CLASS>は対象となるクラスの親クラス (上位クラス) であり，親クラスのすべての属性やメソッドが子クラス (下位クラス) でも有効であることはオブジェクト指向言語の特徴である．属性はオブジェクトの内部データであり，追加したい場合には，slot() の中で定義する．ここでは 2 つの属性を記述しているが，このように複数の属性をコンマ (,) で区切って定義できる．<SLOT-NAME>が属性の識別子であり，属性名とも言う．

```
type_ <TYPE>
```

で，属性のデータ型 (すなわちクラス名) を指定する．指定できるデータ型については後で述べる．属性のデータ型は省略してもよく，この場合，属性に最初に代入されるデータの型がその属性のデータ型となる．

```
<CLASS> is_subclass_of <SUPER-CLASS> : (
  slot(
    <SLOT-NAME> type_ <TYPE> slot_comment_ <COMMENT>,
    <SLOT-NAME> type_ <TYPE> slot_comment_ <COMMENT>
  ),
  method(
    <METHOD-NAME> is_defined_by (
      require( <事前条件> ),
      do( <実行部> ),
      explanation( <説明> )
    ),
    <METHOD-NAME> is_defined_by (
      require( <事前条件> ),
      do( <実行部> ),
      explanation( <説明> )
    )
  )
).
```

図 2.3 RASS 言語の構文形式

```
slot_comment_ <COMMENT>
```

にはコメントを記述する．<COMMENT>は，説明文字を’と’で囲んだものである．手続きであるメソッドの記述は，method() 内で行う．図中では 2 つのメソッドを記述しているが，属性と同様，1 つ以上であれば同時にいくつ記述しても構わない．<METHOD-NAME>がメソッドの識別子である．もし，メソッドに引数を持たせたい場合には，メソッド名の後に括弧を用いて記述する．仮引数としては大文字で始まる名前を用いる．例えば，

```
method(
  example(X, Y) is_defined_by (
   ...
  ),
  example(X) is_defined_by (
   ...
  )
)
```

のようにする．メソッド名が同じでも引数の数が異なれば，別のメソッドとして区別される．メソッド内部の記述は，require()，do()，explanation() の 3 つのブロックに分けて行う．require() ブロックは，そのメソッドを実行する際に，守られていなければならない条件を記述する部分である．ここには属性の値を変更するような実質的な処理を書くことはできず，次の do() ブロックの処理が行える状態にあることを確認するための文のみを書く．この部分が満足されなければ，いかなるオブジェクトの状態も変化しないでメソッドは終了する．このように処理に入るための条件を明確にしておくことは，表明を行うことであり，正しいソフトウエアを書くために有効であることが指摘されている [18]．do() ブロックは，処理を行うためのメソッドの本体を記述するところである．require() ブロックとは異なり，正常終了しない場合にもオブジェクトの状態が変化することがある．すなわち，いくつかの属性の値が変更されるかもしれない．explanation() は，メソッドの説明をする箇所である．このようにコメントをプログラムの一部

として持っておくと，後で説明などに利用することができる．なお，require()，explanation() ブロックは省略されることもある．

2.4.2 実行文

do() ブロックにおける実行文の主たるものは，メッセージ送信と代入であり，これらはコンマで区切れば複数指定できる．

メッセージ送信は

sm(<インスタンス名>，<メッセージ名>，<引数 1>，<引数 2>,...)

の形をとり，メッセージ名はメソッド名であることが多い．これは RASS の起動中に表示されるコマンドプロンプト上で，利用者が RASS に命令を与えるときに使用する基本形でもある．

代入の記号としては:=を使い，左辺には変数名，右辺に数式を書く．変数については次節で述べる．

そのメソッド内だけで利用するローカル変数が必要なら do() ブロックの最初で local_variable() によって宣言する．変数の型を指定することもできる．また，Prolog のプログラムを書けばそのまま実行されるので，それによって言語の機能を拡張することもできる．

2.4.3 変数

RASS 言語における変数には 2 種類ある．ひとつはオブジェクト変数であり，これは属性やローカル変数のことである．もうひとつは Prolog 変数であり，メソッドの仮引数や，sm() から返された値を利用するために用いる．

オブジェクト変数は，システムに既存のクラスや利用者が作成したクラスの実体であるインスタンスをさす．統計解析，特に行列計算のプログラムが容易になるようにいくつかのクラスが定義されており，それらを属性またはローカル変数の型として使うことができる．その例を表 2.1に示す．

大文字，または下線で始まる文字の並びで識別される Prolog 変数は型をもたない．この変数は値が“ 設定されている ”状態か“ 設定されていない ”状態かの

表 2.1 RASS 言語における変数のためのクラス

クラス名	意味
number	数値 (実数, 整数)
string	文字列
vector	ベクトル (成分は数値)
svector	ベクトル (成分は文字列)
lVector	各成分にラベルを持った vector
lsVector	各成分にラベルを持った svector
matrix	行列 (成分は数値)
smatrix	行列 (成分は文字列)
lMatrix	行と列にラベルを持った matrix
lsMatrix	行と列にラベルを持った smatrix

2 つしかなく，1 度ある値に“ 設定される ”とそのルーチン内での設定のやり直しはできない．実際にはメソッド間，またはメッセージ送信の結果を授受するために用いる．また Prolog のプログラムを埋め込んだときにそれとの入出力のためにも利用する．従って，この変数は RASS 言語と (それを実現している)Prolog 言語の言語間インターフェースと考えてもよい．

2.4.4 行列演算

重回帰分析，または多変量解析においては，行列を頻繁に利用する．RASS 言語ではよく利用する行列の操作を関数としてあらかじめ組み込んである．例えば，転置行列を求める `tr()`，逆行列を求める `inv()`，対角成分を取得する `diag()`，行列式を求める `det()`，各成分の平方根をとる `sqrt()` 等がある．

さらに，行列の成分を指定するための便利な機能を備えている．ある行列の一つの成分を指定する場合は，一般的なプログラミング言語と同様に，

変数名 (行番号，列番号)

で指定する．また，リストを使用して一度に複数の成分を指定することもできる．ここで言うリストとは，'[' と']' で囲んだ数字や識別子の並びである．例えば，2 行目の 2 列と 4 列を指定するには，a(2，[2，4]) と記述する．さらに，*や~を使用した指定方法もある．*は対応する行や列の全ての成分を指定する．例えば，a(*，2) は，2 列目の成分全てを指定する．~は，行 (列) 番号やそのリストとともに用いて取り除く成分を指定する．a([1，2]，~2) は 1 行目と 2 行目の全ての成分の中で，2 列目を除いた成分を指定することになる．また，ラベルを持った行列やベクトルでは，成分番号のかわりにラベルで指定することもできる．

2.4.5 制御構造

RASS の制御構造としては if 文と while 文がある．条件判断のための if 文の形式は

if(関係式，文の並び)

であり，括弧内の関係式が成立した時に，文の並びを実行する．文の並びは文をコンマで区切ったもので，文の数に制限はない．関係式は，比較演算子を用いて 2 つのスカラの大小関係を記述したものか，式を記述したものである．式が与えられた場合は，その評価値がスカラで正の値を持てば文の並びを実行する．

繰り返しには while 文を用いるが，その形式は

while(関係式，文の並び)

であり，関係式が満足された時に，文の並びを順に実行する．そして，実行が終れば再び関係式の評価が行われる．関係式の評価は，if 文の場合と同じである．

2.4.6 演算子

条件判断や繰り返しでよく利用する比較演算子には，<，<=，>，>=，==，!= がある．算術演算子としては，四則演算用の+，-，*，/，ベキ乗の^，アダマール積を求める#がある．四則演算用の 4 つの演算子は行列演算にもそのまま利用することができる．ただし，除算の場合は除数がスカラでなければならない．

表 2.2 RASS のシステムコマンド一覧 (一部)

コマンド名	説 明
create_instance(*Instance*, *Class*)	クラス *Class* からインスタンス *Instance* を生成する
program(*FileName*)	プログラム *FileName* を組み込む
asNum(*Object*)	*Object* の要素が数字かどうか
clear_gnuplot	グラフウィンドウの内容の消去
exist_slots(*Var* {, *Var*})	*Var* をメソッド内で利用可能かどうか
local_variable(*Var* {, *Var*})	メソッド内で有効な局所変数の宣言

2.4.7 RASS の組み込みコマンド

RASS の組み込みコマンドの一部を表 2.2に示す．これらの文は，全て結果として論理値を返す．これらは，RASS の支援機構を実装した Prolog の述語と同等な位置に存在する．表中の最後の２つのコマンド exist_slots() と local_variable() は，クラスのメソッド定義内だけで有効なものである．

また，解析上必要なコマンドはオブジェクトのメソッドとして実装しているものがある．例えば，解析データのファイルを読み込むためには，クラス lsMatrix の readFile というメソッドを用いればよいし，文字列の表示には，クラス user_output に表示させたい文字列を引数として write というメソッドを用いればよい．従って，RASS 言語のプログラマは，既存のクラスのメソッドをある程度知っておく必要はある．

RASS 言語で記述したプログラムを RASS 上に組み込ませるためには，それをファイルに保存しておかなければならない．このプログラムを RASS に導入するためには，表 2.2中の program() を使用する．例えば，作成したプログラムを mylsd.lng というファイル名で保存していれば，コマンドライン上か RASS の Make ファイル中で

```
program('mylsd.lng').
```

と入力する．もし，このプログラムに文法的な問題があれば，このコマンドが呼

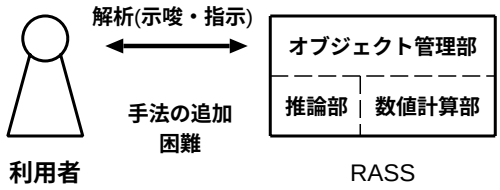

図 2.4 RASS 言語を備える前の RASS の構成図

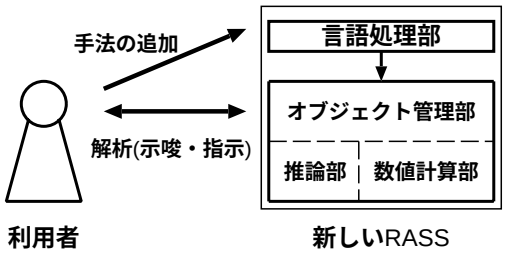

図 2.5 RASS 言語を備えた後の RASS の構成図

び出される段階でエラーメッセージが表示される．

2.5 RASS 言語の位置づけと実装

RASS 言語を備える前の RASS の構成図を図 2.4に，RASS 言語を備えた後の RASS の構成図を図 2.5に示す．構成から見た両者の違いは，言語処理部がシステムとして加わっているかどうかである．RASS 言語で記述されたプログラムは，言語処理部で RASS 本来の持つオブジェクト管理部で解釈できるプログラムに翻訳される．そして，そのプログラムがオブジェクト管理部に登録される．

RASS 言語の実装には，RASS の推論部やオブジェクト管理部の実装で使用した Prolog を用いている．Prolog は文字列処理に長けており再帰呼出しができるので，字句解析や構文解析，意味解析のための処理を簡便に記述することができる．利用者が記述したプログラムに文法的なエラーがあった場合には，言語処理部から利用者に対してそのメッセージが表示される．

RASS 言語を備える前の RASS では，図 2.2の例のように，統計解析の手法ごとに C 言語の関数を割り当てて実装されていた．この方法は，Prolog と C 言語におけるデータの受け渡し回数を減らせるので高速に処理できる利点があるが，拡張性や柔軟性が低くなる．これに対して RASS 言語では，利用者が自由に記述した数値計算式を実際に評価できるようにするために，数値計算部を一般化している．具体的には，数値演算の基本（例えば，行列同士の四則演算）ごとに C 言語の関数を割り当てた．これにより，数値計算のための処理時間は全体的に若干遅くなるが，利用者側の記述における柔軟性を高めることができる．現状では，ハードウェアの発達により，この遅延が通常の解析に悪影響を与えるようなことはない．

2.6 プログラム例

本節では，以下で RASS 言語によるプログラムの例を 2 つ示す．ひとつめは，従来の RASS では実装されていなかった手法を RASS 言語により記述した例である．もうひとつは，RASS 言語の自己記述性を確認するために，RASS の既存のクラスを RASS 言語で再定義したものである．

2.6.1 回帰診断統計量 DFBETAS

DFBETAS は Belsley ら [4] によって提案された回帰診断のための統計量で，各観測値が回帰係数にどれほどの影響を及ぼしているかを計るために利用される．

従属変数ベクトルを y，定数項を含めた独立変数行列を X とする．回帰モデルとして $y = X\beta + \varepsilon$ を仮定する．ここで β は回帰係数ベクトル，ε は誤差ベクトルである．

このとき，DFBETAS の値は

$$DFBETAS_{\alpha j} = (\hat{\beta}_j - \hat{\beta}_{j(\alpha)})/s_{(\alpha)}\sqrt{[(X'X)^{-1}]_{jj}}$$

で定義される．ただし，$\hat{\beta}$ は回帰係数の最小二乗推定値，$\hat{\beta}_{j(\alpha)}$ と $s_{(\alpha)}$ は，それぞれ，α 番目の観測値を除いた時の回帰係数の最小二乗推定値および標準偏差である．

この計算を RASS 言語により記述したものが図 2.6である．RASS に組み込まれているクラスの中で DFBETAS が属するべきものは，明らかに，回帰診断のためのクラス lSDiag (least square diagnostics) である．このクラスの親クラスは lSRes (least square result) である．

計算結果を保持するために dfbetas という属性を作成する．メソッド calDFBETAS は DFBETAS を求める先述の式をそのまま RASS 言語の構文規則に従って記述した手続きである．

`require()` ブロックの `exist_slot()` は親クラスの属性を参照するという宣言である．

`do()` ブロックの中では，while 文を利用して観測値の数だけ繰り返しを行い，その中で独立変数のラベル付き行列 `indepVar` と従属変数のラベル付き行列 `depVar` のそれぞれの一行を削除するために~を使用した．

計算が終った後で `sm(dfbetas, dsp)` により，その結果を表示する．メソッド `dsp` はすべてのオブジェクトに定義されていて，それぞれのオブジェクトにふさわしい形式で表示するためのものである．

2.6.2 基本統計量を保持するクラス

RASS 言語の自己記述性を調べるために，RASS に組み込まれている基本統計量のクラス bStat(basic statistics) の一部の機能を実現してみる (図 2.7)．

RASS では基本統計量はクラス bStat で計算されるが，それは回帰分析データをあらわすクラス regDat (regression data) の子クラスである．ここでも regData と同じものを RASS 言語で実現したクラス myRegDat を親クラスとして子クラス myBStat を定義する．

myBStat では基本統計量の計算が行われるので，その結果を保持するための属性を定義する．その属性の値を計算するのは，このクラスのインスタンスが生成されるときである．RASS 言語ではインスタンスの生成は initialize というメソッドで行われる．このメソッドは myRegDat のメソッド，例えば mkMyBStat(make myBStat) などから呼ばれる．`do()` ブロックでは，まず，従属変数と定数項を除いた独立変数をまとめた行列 `m` を作り，メッセージ送信により行列に

```
lSDiag is_subclass_of lSRes : (
  slot(
    dfbetas type_lMatrix slot_comment_ 'DFBETAS'
  ),
  method(
    calDFBETAS is_defined_by(
      require(
        exist_slots(indepVar, depVar, nObs)
      ),
      do(
        local_variable(beta, s, b, xxih, n, x type_ number),
        xxih := sqrt(diag(inv(tr(indepVar)*indepVar))),
        beta := inv(tr(indepVar)*indepVar)*tr(indepVar)*depVar,
        n := nObs,
        dfbetas := indepVar,
        while(n >= 1,
          x := nObs - n + 1,
          b := inv(tr(indepVar(~x,*))*indepVar(~x,*))
               *tr(indepVar(~x,*)*depVar(~x,*),
          s := (tr(depVar(~x,*)-indepVar(~x,*)*b)
               *(depVar(~x,*)-indepVar(~x,*)*b)
               /(nObs-indepVar))^(0.5),
          defbeta(x,*) := tr((beta-b)/(xxih*s)),
          n := n-1
        ),
        sm(dfbetas, dsp)
      ),
      explanation('DFBETAS show how each obserbation',
              'influences regression coefficients.')
    )
  )
).
```

図 2.6 RASS 言語による DFBETAS を求めるプログラム

```
myBStat is_subclass_of myRegDat : (
  slot(
    mean        type_ lMatrix slot_comment_ ' 平均値',
    minimum     type_ lMatrix slot_comment_ ' 最小値',
    maximum     type_ lMatrix slot_comment_ ' 最大値',
    skewness    type_ lMatrix slot_comment_ ' 歪度',
    kurtosis    type_ lMatrix slot_comment_ ' 尖度',
    covariance  type_ lMatrix slot_comment_ ' 共分散',
    correlation type_ lMatrix slot_comment_ ' 相関係数'
  ),
  method(
    initialize is_defined_by (
      require(
        exist_slots(indepVar depVar)
      ),
      do(
        local_variable(m type_ lMatrix),
        m := colBind(depVar, indepVar(*, ~1)),
        sm(m, mean, Mean),  mean := Mean,
        sm(m, min, Min),    minimum := Min,
        sm(m, max, Max),    maximum := Max,
        sm(m, skew, Skew),  skewness := Skew,
        sm(m, kur, Kur),    kurtosis := Kur,
        sm(m, cov, Cov),    covariance := Cov,
        sm(m, cor, Cor),    correlation := Cor
      )
    ),
    cov(Var1, Var2, Cor) is_defined_by (
      require(
        exist_slots(correlation)
      ),
      do(
        local_variable(c type_ lMatrix),
        c := correlation,
        sm(c, getValue, Var1, Var2, Cor)
      )
    ),
```

図 2.7 基本統計量のクラスの再定義（一部）

定義されているメソッドを利用して実際の計算を行い，その結果を属性に格納する．explanation() ブロックは略されている．メソッド cor は二つの変数名を与えてそれらの間の標本相関係数を返すものである．インスタンスが作成されているならば計算はすでに行われているので，必要な値を入手するだけのものである．

2.7 本章のまとめ

本章では，重回帰分析支援システム RASS に実装した言語処理機構（RASS 言語）について説明した．RASS 言語は，知的統計解析システム用に実装された拡張用言語である．それは，オブジェクト指向プログラミングのパラダイムに基づいた手続き型の高水準プログラミング言語である．利用者は，RASS の開発に使われたシステム記述言語を使わなくても専用の RASS 言語を利用することで，重回帰分析やそれに関連する計算アルゴリズムを記述できるようになった．これまでのところ，RASS 言語のような拡張用言語を実現した知的統計解析システムは見受けられない．

RASS だけに限らず，多くの知的統計解析システムは，図 2.4 に示したように，内部に知識処理機構と計算処理機構を持ち，これらを統括する管理機構によりシステム本体が実装されている．また，内部のこれらの機構は，いくつかのシステム記述言語で実装されていることがほとんどである．このような状況で言語処理機構を実装するには，知識処理機構と計算処理機構の直接的な関連づけをやめ，計算処理機構の一般化を行い，管理機構に言語処理能力を構築することがおそらく最も汎用的な方法であろう．RASS 言語は，この方法を確認するひとつの試みでもある．

最新の RASS では，管理機構を RASS 言語で書き直している．これは，RASS 言語が重回帰分析に必要な計算アルゴリズムをすべて表現できることを意味している．また，利用者は，ソースプログラムから RASS の構造を比較的簡単に理解することができる．

3. 自発的なオブジェクトによる重回帰分析支援システム RASS2 とそのプログラミング言語

3.1 はじめに

データから有用な情報を入手し，最適なモデルを見つけ出すために，データ解析用システムがある．現在，S[6] や XploRe[10] などのような統計解析システムが広く普及し，利用されている．しかしながら，これらのシステムを使ってデータ解析をする場合，システム固有の詳細な操作方法を習得する必要がある，初心者が誤った解析をしても形式的に答えが求ってしまう，という問題がある．

このような問題を解決するために知識処理技術を持つ統計解析システムの研究が行われてきた [8, 50, 53]．これらの研究により知識処理機構を持つ統計解析システムの有効性はある程度認められたが，一般に普及しているとはいえない．その理由としては，統計解析に関する知識やその重要度は，統計学の専門家によって，あるいはデータの特性によって異なることがあるのに，多くの支援システムが利用者レベルで知識を柔軟に変更できる機構を持っていないことが挙げられる．前章で紹介した重回帰分析支援システム RASS[50, 19, 40] においても，その開発過程でこのような課題があることを認識した．

そこで，RASS に組み込まれている知識をより柔軟に変更したり，拡張することができるようにするために，オブジェクトがルールを保持し，それに基づいて自発的に振る舞うモジュールをデータ解析支援システムの構成要素として採用することにした．そして，それを使用して実際に，重回帰分析のための支援システム RASS2[13, 41] を開発した．

本章では，次節で，RASS2 の基本構成要素である拡張したオブジェクトの概要について述べ，3 節で，そのオブジェクトを用いてどのように RASS2 を構成したのかを紹介する．4 節では，RASS2 上でどのように手続きやルールを記述するのかを 2 つの例を用いて説明する．5 節では，RASS2 の実行方法と RASS との比較について簡単に触れる．

3.2 オブジェクトの拡張について

3.2.1 オブジェクトを拡張する必要性

統計解析は，多くの統計手法とその計算結果（統計量）に基づいて行われる．これらはそれぞれオブジェクト指向のメソッドと属性に対応するため，統計解析のシステムにオブジェクト指向の考え方は馴染みやすい．実際，このような考え方を導入したシステムの開発が行われている [31].

一方，統計解析に関する主要な専門的知識は，(1) データに対する制約条件，(2) 統計量の計算法，(3) それらの手法を利用するタイミング，(4) 統計量の解釈の判断基準，などである．これらの知識を具体的に表現するには，常に適用する対象（データ）が何であるのかを明確にする必要があるので，これらの知識は，データを表現するオブジェクトと密接に関係付けることが自然である．

これまでにオブジェクト指向の枠組みでエキスパートシステムを構築した事例がいくつか報告されているが，通常，この枠組みの中で知識ベースシステムを実装することは容易ではない [29]．これは，標準的なオブジェクト指向の考え方では，属性やメソッドと同様に知識をオブジェクトの内部構造の一部として実現していない（カプセル化していない）ことに難しさの原因の一端があると考えられる．この場合，システムの中に知識処理のためのクラスを実現したり，クラスの中で知識ベースを構築しなければならないが，他のオブジェクトとの関係が複雑になるので柔軟性や汎用性は低くなる．なぜなら，必要に応じて知識を変更したい場合には，知識処理機構を十分に把握し，知識処理以外の機能との関係を熟知することが要求されるからである．

また，統計解析システムの多くは対話的に命令を与えてその結果を見ながら解析を進める．このような場合には，支援システムが，利用者からの命令により変化した状況に対応して，瞬時に利用者への提案 (示唆) を更新することも必要である．メッセージを受け取ることで処理を実行に移すオブジェクト指向では，このような自発的な振舞いの実現には，特別な枠組みが要求される．

そこで，データ解析のための知識をオブジェクト内で出来るだけ簡単に記述できるようにし，さらに，その知識を適用できる状況になれば，そのことを自発的にシステムに対して提案できるようにオブジェクトを拡張することにした．この拡

張されたオブジェクトは“ 自発的なオブジェクト ”と考えることもできる．RASS2は，この自発的なオブジェクトを基本構成要素として構築されている．

一般的なオブジェクトを基本構成要素とするシステムに知識処理機構を組み込ませることは，これまでにいくつかの研究で行われてきた (例えば，[5])．また，これらの研究の延長として，いわゆるエージェントに関する研究が多く進められてきた [39]．エージェントの定義は，研究者によってさまざまであるが，自律的な行動をとれるように実装されたオブジェクトと考えることができる．これらの研究は，システムが利用者の代わりに思考したり，何らかの動作を行うことを目的としているので，RASS2 のオブジェクトも，このようなエージェントの一種であると考えることができる．RASS2 のオブジェクトの特徴は，ルールを属性やメソッドと同等にカプセル化しており，そのルールに従って自発的に振舞えるようにしていることと，ソースコードの可読性を重視していることである．

3.2.2 自発的なオブジェクトの動き

一般的に，オブジェクト指向のオブジェクトは，物理的あるいは概念的にまとまりをなすソフトウェア上の“ もの ”である．オブジェクトに要求される必要最小限の機能としては，(1) データ（属性）と手続き（メソッド）を合併体として定義できる枠組み，(2) メッセージの授受に基づく行動，(3) 継承機構，である．また，オブジェクトの雛形のことをクラスと呼び，クラスから作られる実体をインスタンスと呼ぶ [47]．

RASS2 のオブジェクトの概念図を図 3.1に示す．このオブジェクトは，オブジェクトの特徴を踏襲し，その中で有効なプロダクションルール形式の単純かつ断片的に表現されるルール（起動条件）を内部構造として保持する．また，それぞれのルールは，そのオブジェクト内で起こり得る適用可能なルールの競合を解消するために，システム設計者や利用者の考え方を反映した局所的な重み（局所優先度）が設定される．具体的な記述方法は 3.2.5節で述べる．

RASS2 のオブジェクトは，普通のオブジェクトのように，メッセージを受けとることで自らの持つメソッドを実行できるだけでなく，自らの持つルールに基づいて何らかの処理を実行に移すように提案することができる．具体的には，各オ

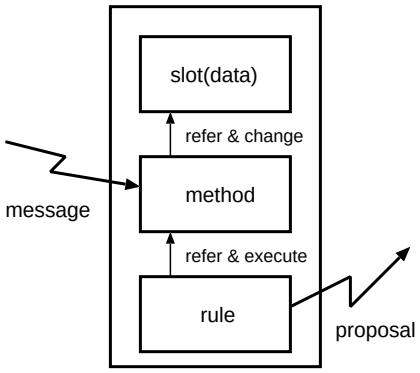

図 3.1 RASS2 のオブジェクトの概念図

ブジェクトが持つ全てのルールに対して，その条件部を評価し，それが真になったルールを実行に移すように RASS2 の統括部に提案する．そして，それを受け取った RASS2 の統括部が，その内容を利用者に伝える．条件部では，各オブジェクトが持つ内部状態を変更しないようなメソッドを利用して，システムや自らの状態を監視する．利用者やシステムによってその提案が受理されると，ルールの実行部で記述されている手続きが順に実行される．

ルールの具体例としては，(1) 重回帰分析のためのデータが用意できたら，線形最小二乗法をデータに適用する，(2) 分散拡大要因（VIF）が 10 以上のとき，変数選択をやり直す，などが挙げられる．RASS2 のオブジェクトは，条件が整えば利用者からの問い合わせ（メッセージの受信）が無くても，ここで挙げた線形最小二乗法の適用や，変数選択のステージに移動するように，自発的にシステム（間接的には利用者）へ提案する．

3.2.3 RASS2 の特別なクラスオブジェクト

オブジェクト指向におけるクラスは，現実世界の“ もの ”の雛型であり，これから実際のオブジェクト（インスタンス）が生成される．重回帰分析のためのクラスを用意し，解析の流れに応じてそれらのインスタンスを生成することで目的の処理を実行しようとした場合，(1) 特定のクラスから複数個のインスタンスを

生成する必要があるので，インスタンスを何らかの方法で管理しておきたい，(2) 一般的にインスタンスの生成は他のインスタンスに委ねなければならないが，こうしなくても自発的にインスタンスを生成する方法が欲しい，という要求がある．そこで，これらに対処するために，RASS2 では，システム上でクラスを定義すると同時にクラス名と同名の 1 つのインスタンスをシステム上に存在するようにした．本稿では，これをクラスオブジェクトと呼び，雛形であるクラスと区別している．また，その属性やメソッドも，そのクラスのインスタンスのものとは区別される．このような考え方は，オブジェクト指向言語である Smalltalk のメタクラス [24] や Java 言語のクラス変数やクラスメソッドの概念に近い [39]．

RASS2 においてインスタンスを生成するということは，新たな情報を入手すること，つまり，解析の状況を一段階進めることである．また，データ解析では解析途中で評価基準や解析の流れを変更したいことがよくある．クラスオブジェクトを導入したことで，あるクラスのインスタンスが複数個生成されている場合，クラスオブジェクトに問い合わせることで必要なものを見つけ出すことができる，インスタンスの生成を状況に応じて適切に示唆してくれる，解析に必要なクラスを他のクラスに影響を与えることなく動的に組み替えることができる，ということを利用者に提供することができた．

3.2.4 RASS2 言語の文法規則の概略

RASS2 のオブジェクトのクラス定義は，図 3.2と図 3.3の形式で記述される．図中の { } は，その中の項目の零回以上の反復を示している．定義内は，クラス属性（class_slot），インスタンス属性（slot），クラスメソッド（class_method），インスタンスメソッド（method），クラスルール（class_rule），インスタンスルール（rule）の 6 つに分かれる．このうち，クラス属性とクラスメソッドとクラスルールは，3.2.3節で述べたクラスオブジェクトのためのものである．例えば，クラス属性は，インスタンスに共通する情報などを表現する．生成したインスタンス名を記録しておく場所などがこれに該当する．

フォーマット中の *CLASS* にはクラス名を，*SUPER* には親クラス名を 1 つだけ指定する．メソッドとルールの具体的な手続き *STATEMENTS* の記述は，手

```
<CLASS> ::=
  CLASS is_subclass_of SUPER :(
    class_slot( <SLOT> {,<SLOT>} ),
    slot( <SLOT> {,<SLOT>} ),
    class_method( <METHOD> {,<METHOD>} ),
    method( <METHOD> {,<METHOD>} ),
    class_rule( <RULE> {,<RULE>} ),
    rule( <RULE> {,<RULE>} )
  ).
<SLOT> ::=
  NAME type_ <TYPE> |
  NAME type_ <TYPE> slot_comment_ STRINGS
<METHOD> ::=
  NAME is_defined_by (
    require( <CONDITIONS> ),
    do( <STATEMENTS> ),
    explanation( STRINGS )
  ).
<RULE> ::=
  NAME$LOCAL_WEIGHT is_defined_by (
    condition( <CONDITIONS> ),
    propose( <STATEMENTS> ),
    explanation( STRINGS )
  ).
```

図 3.2 RASS2 言語におけるクラス定義フォーマット（前半）

<*TYPE*> ::= `number` | `vector` | `list` | `...` | `string` | *CLASS* | <*REF*>
<*STATEMENTS*> ::= <*STM_LIST*> | <*LOCAL*>, <*STM_LIST*>
<*STM_LIST*> ::= <*STATEMENT*> {,<*STATEMENT*>}
<*STATEMENT*> ::= <*ASSIGN*> | <*CNTRL*> | <*CND*> | *COMMAND*
<*LOCAL*> ::= `local_variable(`<*SLOT*> {,<*SLOT*>}`)`
<*ASSIGN*> ::= *OBJECT* `:=` *STAT.EXPR*
<*CNTRL*> ::= <*IF*> | <*WHILE*> | <*FOREACH*>
<*IF*> ::= `if(`<*CND*>`, [`<*STM_LIST*>`])`
| `if(`<*CND*>`, [`<*STM_LIST*>`], [`<*ELSE_LIST*>`])`
<*WHILE*> ::= `while(`<*CND*>`, [`<*STM_LIST*>`])`
<*FOREACH*> ::= `foreach(`*VAR*, *VALUES*, `[`<*STM_LIST*>`])`
<*CND*> ::= *COMP.EXPR* | <*OR*> | <*AND*> | <*NOT*>
<*OR*> ::= `or(` <*CND*>, <*CND*> {,<*CND*>} `)`
<*AND*> ::= `and(` <*CND*>, <*CND*> {,<*CND*>} `)`
<*NOT*> ::= `not(` <*CND*> `)`
<*REF*> ::= `@`*CLASS*

図 3.3 RASS2 言語におけるクラス定義フォーマット（後半）

続き型言語を強く意識した文法に従って記述する．この文法規則の設計においては，前章の RASS 言語 [40] を参考にした．各 *STATEMENTS* は結果として論理値を持つ．*STRINGS* には，コメントを’と’で囲んで記述する．また，属性宣言での *TYPE* では，number，vector などの基本データ型やクラス名，クラス参照（*REF*）を指定することができる．

メソッドに引数を持たせる場合には，メソッド名 *NAME* の後に括弧を用いて記述する．引数としては定数，オブジェクト名，テンポラリ変数を用いることができる．メソッドは戻り値を持たないので，メソッドを実行することによって求まる値は，テンポラリ変数の引数を利用することで値の授受が行われる．テンポラリ変数は，大文字，または下線で始まる文字の並びで識別され，値の授受や一時的な利用を目的とした変数である．

メソッド定義の内部は，`require()`，`do()`，`explanation()` の 3 つのブロックに分かれる．`require()` ブロックでは，メソッドの実行を保証するために必要な条件を記述する．引数のチェックなどもここで行う．ここでは，属性の値を変更するような実質的な処理を書くことはできず，次の `do()` ブロックの処理が行える状態にあることを確認するための参照に基づく処理を記述する．この部分が満足されなければ，いかなる状態も変化しないでメソッドは終了する．図 3.2の `require()` ブロック内の *CONDITIONS* の記述は，*STATEMENTS* と基本的に同じであるが，ローカル変数の宣言と代入文は使用できない．`do()` ブロックでは，実質的な処理を記述する．なお，*COMMAND* については 3.3.5節で説明する．

3.2.5 RASS2 言語におけるルールの定義

ルールは，拡張されたオブジェクトの自発的動作を表現する．図 3.2の *RULE* 定義内での *NAME* がルールの識別子である．もし，ルールに引数を持たせたい場合には，メソッドと同様に，ルール名の後に括弧を用いて記述する．どのオブジェクトもルールを直接実行に移すことはできない．

一般的に，データ解析において，対象や問題を限定すればどういう知識を先に適用するのか判断できるし，同時に適用できる知識があれば，どちらを選択すべきなのかを知っていることが多い．つまり，対象となる問題をあるオブジェクト

に限定すれば各知識の重要度を理解することができるし，知識同士が競合を起さないように重要度を設定することは不可能ではない．

そこで，ルール定義の *LOCAL_WEIGHT* に，そのオブジェクト内で有効なルールの局所的な重み（局所優先度と呼ぶ）を数値として指定する．これは，システム開発者や利用者の考え方を反映するものである．局所優先度は，0 以上の数値で指定される．いくつかのルールに同じ局所優先度を設定してもよいが，省略することはできない．

ルール定義の内部は，`condition()`（条件部），`propose()`（実行部），`explanation()`（説明部）の3つのブロックに分かれる．`condition()` ブロックでは，次の `propose()` ブロックで記述される処理を自発的に起動させる条件を記述する．このブロック内で記述する *CONDITIONS* ブロックは，メソッドの `require()` ブロックと同じであり，属性の値を変更するような実質的な処理を書いてはいけない．

`propose()` ブロックでは，自発的に実行する処理を記述する．文法は，メソッドの `do()` ブロックと同じ *STATEMENTS* に従うが，通常，属するオブジェクトのメソッドを呼び出す文を書く．

3.3 RASS2 の構成

3.3.1 RASS2 の基本サイクル

RASS2 の実行環境では，(1) オブジェクトのルールの評価（提案の生成），(2) その中で適切な提案を 1 つ選ぶ（提案の選択），(3) 選ばれた提案を実行に移す（提案の実行），というサイクルを繰り返す．提案の生成は，RASS2 が，システム上に生成されたオブジェクトの順にそのルールの条件部をチェックする．提案の選択は，ルールの局所優先度と 3.3.4節で述べる manager クラスによって行われる．提案の実行では，(2) で選択された提案の実行部に記述されている手続きが実行される．

表 3.1 RASS2 の組込みクラス一覧（一部)

クラス名	説 明	親クラス
object	全てのオブジェクトの最上位クラス	
agent	自発的オブジェクトの最上位クラス（自発的動作を実現するための機能を持つ)	object
string	文字列を表すクラス	object
number	数値を表すクラス	object
list	リストを表すクラス	object
user	利用者とのインターフェースを表すクラス	object
statGraph	統計グラフを表すクラス（histogram, stemLeaf, boxPlot, scatter 等の派生クラスがある)	object
history	解析履歴を記録するクラス	list
dist	確率分布を表すクラス（normal, chi2, t, f の派生クラスがある)	object
manager	提案の調整のためのクラス（3.3.4節参照)	agent
matrix	行列（要素が数値のみ）を表すクラス（ラベル情報を付加した lMatrix が派生クラスとして存在する)	agent
smatrix	行列（要素が文字列のみ）を表すクラス（ラベル情報を付加した lsMatrix が派生クラスとして存在する)	agent

3.3.2 組込みクラス

RASS2 を実装するために組み込んだ主要なクラス (一部) を表 3.1に示す．利用者は，これらのクラスにメッセージを送ることで簡単に利用することができる．

object クラスを直接継承したクラスは，一般的なオブジェクト指向のオブジェクトのように受動的に振舞う．この場合，たとえクラスに知識が定義されていても自発的に振舞うことはできない．agent クラスから派生したクラスは，自発的な動作が可能なオブジェクトとなる．

RASS2 において，重回帰分析に関係するクラスは次の 5 つである．

orgData (original data) 利用者が用意した生データを表すクラスである．継承した親クラスは，行と列のラベルを保持し，要素が文字列の行列を表すクラス lsMatrix である．各要素は文字列として取り扱われる．欠測値の存在等，データとして不完全な状態であればそのことを示唆する．

numData (numerical data) 解析用の数値データを表すクラスである．継承した親クラスは，行と列のラベルを保持し，要素が数値のみの行列を表すクラス lMatrix である．解析データとして問題があれば，それを知らせ，対策を示唆する．データを変換・削除するメソッドなどを持つ．

regData (regression data) 回帰分析のためのデータを管理するクラスである．属性として従属変数の情報と各変数ごとの基本統計量を持つ．従属変数のデータに問題があれば，それを知らせ，対策を示唆する．親クラスは，agent である．

lsModel (least square model) 回帰モデルを管理するクラスである．属性には独立変数や，最小二乗法を適用した結果を持つ．メソッドには独立変数 (ラグ項等を含む) の削除や登録，各種統計量の算出法などがある．ルールには決定係数や多重共線性，F 値等を順に評価し，問題があればその問題の程度に応じてどう対処するのかを表現したものがある．親クラスは，agent である．

lsDiag (least square diagnostics) 回帰診断の結果を管理するクラスである．属性として診断用の統計量を保有する．lsModel と同様に，てこ比や Cook の

距離，DFBETAS などの統計量に関するルールが組み込まれている．また，これらの算出に必要なメソッドがある．親クラスは，agent である．

図 3.4は，上記 5 つのクラスを重回帰分析の流れに沿って配置したもの（左側）と，それらのインスタンスの関係を表したもの（右側）である．クラス間の下向き矢印は，継承を表しているのではなく，矢印の下側のクラス（下位クラス）のインスタンスが生成されるときに，矢印の上側のクラス（上位クラス）のインスタンスの情報を参照して生成されることを意味している．例えば，lsModel のインスタンスは，regData のインスタンスの情報に基づいて生成される．図中の矢印の横の **1** と **m** は，生成されるインスタンスの数を表す．例えば，**1** と **m** を付記された矢印で結ばれている 2 つのクラスは，**1** と記された側のクラスの 1 個のインスタンスに対して，**m** と記された側のクラスのインスタンスが 1 個以上生成されることを表す．同様に，**1** と **1** を付記された矢印で結ばれている 2 つのクラスは，それぞれのインスタンスが 1 対 1 で対応することを表す．従って，解析が進むと，これらのクラスのインスタンスが解析の流れに沿って図 3.4の右側の例のような階層を形成する．ただし，あるインスタンスをさらに深く解析しても意味のないことが分かった場合には，下位クラスのインスタンスは作成されない．

3.3.3 組み込みルールの特徴

RASS2 のオブジェクトに組み込まれているルールは，重回帰分析の解説書 (例えば，[25, 45]) で示されている知識などを参考に実装されている．これらをシステムの視点で大別すれば，解析を進めるためのものと，問題解決を計るためのものになる．

解析を進めるための知識としては，図 3.4に示したようなインスタンスを解析の流れに応じて生成するためのものがある．このとき，インスタンスの生成を特定のインスタンス（例えば，上位クラスのインスタンス）が行うとそれらのクラス間の従属性が強くなり，オブジェクトの独立性が失われる．そこで，生成しようとするインスタンスのクラスルールを活用してインスタンスを生成する．

また，各インスタンスが持つ問題を解決するためのルールとしては，インスタンスルールが利用される．このインスタンスルールでは，データやモデルの変更

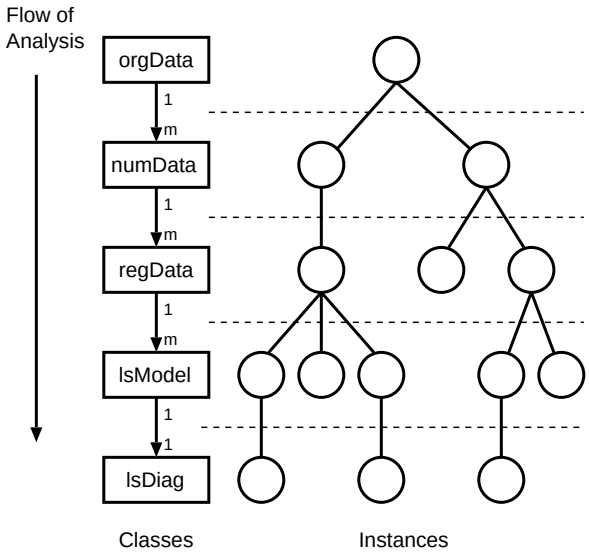

図 3.4　重回帰分析の流れを表現したクラスとそれらから生成されたインスタンスの例

を伴うために新しいインスタンスが生成されることがある．そして，もし，新しく生成されたインスタンスから問題解決のための提案がなければ，そのインスタンスは統計的に妥当なものと判断される．最終的に，利用者は図 3.4のように生成されたインスタンスの階層から，統計的に問題の少ないインスタンスを見つけ出す.

3.3.4　提案の調整

RASS2 上のオブジェクトから同時に複数の提案があれば，提案に付帯する局所優先度に基づいて実行に移す提案を選択する．ただし，提案に付帯する局所優先度は，あるオブジェクト内だけで取り決めたものであるので，そのままシステム全体の大域的な優先度と見なすとシステムの整合性が保てなくなるおそれがある．また，各オブジェクトは，システムの目的や状況に応じて適切に提案の強さを修正することが望ましい．そこで，これを RASS2 上で実現するために，表 3.1 に記した manager クラスのインスタンスを利用することにした.

manager クラスの役割は，提案の収集，提案の重要度の調整，メッセージ送信の代行，の 3 つである．まず，提案の収集では，システム上に存在する全てのオブジェクトからの提案を自らのインスタンス属性 proposals （`list` 型）に保存する．そして，提案の重要度の調整では，自らの持つインスタンスメソッド adjustRules により proposals の中の提案の重要度を変更する．そして，その結果を利用者に表示する．manager クラスは，この一連の動きを自動的に実行する．メッセージ送信の代行とは，利用者が RASS2 に命令を与える時に，命令を与えるオブジェクトとメッセージを全て入力しなくても，manager のインスタンスに単純な命令（go）を送信するだけで，提案を実行できる枠組みを意味する．これについては，3.5節で具体的に述べる．

なお，manager クラスのインスタンスにより提案の重みの修正が行われても競合が解消しない場合は，最初に適合した提案が実行される．

重回帰分析には，データやモデルの統計的に不適切な問題を探し，それを解消することでより適切なモデルを探すという大まかな流れがある．そこで，RASS2 では，各オブジェクトのルールの局所優先度を (1) 1 以上 2 未満, (2) 2 以上 3 未満，(3)3 以上，の 3 段階に設定した．局所優先度が増すと，そのルールの重要度は増す．(3) は，ルールの条件が真となれば直ちにそのルールの実行部を実行に移さなければならないようなルールに設定される．例えば，これ以上あるオブジェクトの解析を進めても仕方がない時に，解析の対象を他のオブジェクトに移すようなルールが該当する．(1) は，ルールの条件が真となってもすぐに処理をしなくてはいけないほどの重大な問題でないと考えられるルールに設定される．例えば，VIF が 6 から 10 未満の値をとるような時には，多重共線性に疑いがあることを利用者に伝えるようなルールが該当する．(2) は，(1) の処理を行う前に解析を進めたり，統計量の確認を促すようなルールに設定される．このように局所優先度を取り決めたので，解析の途中で (3) に該当するルールがなければ，(2) のルールにより解析が進む．もし，(3) のような明らかに不適切な問題が発見されると解析を進めずにその問題を解決しようとする．(1) に該当するような明らかではないが考慮すべき問題に対しては，とりあえず解析が進められ，(2) と (3) に適合するルールがなくなった段階でそれらの処置が行われる．

表 3.2 RASS2 のシステムコマンド一覧 (一部)

コマンド名	説 明
sm(*Object*,*Message* {,*Arg*})	オブジェクト *Object* へメッセージを送信する
load(*FileName*)	クラス定義ファイル *FileName* を読み込む
activate_object(*Object*)	オブジェクト *Object* のルールの適用を開始する
inactivate_object(*Object*)	オブジェクト *Object* のルールの適用を停止する
create_instance(*Class*,*Instance*)	クラス *Class* からインスタンス *Instance* を生成する
remove_instance(*Instance*)	インスタンス *Instance* を削除する

RASS2 では，オブジェクトからの提案の調整をするために，manager クラスのインスタンスとして，adviser と名付けたものを組み込んでいる．重回帰分析の専門家は，あるモデルに着目すればそれについて深く調べていくという縦型探索に近い思考形態をとるのが一般的である．そこで，adviser に，1 つ前に選択され，実行された提案をしたオブジェクトから提案されているルールの局所優先度を 1 だけ増やすインスタンスメソッド adjustRules を持たせ，あるインスタンスを重点的に解析する傾向を示すようにしている．

3.3.5 RASS2 の組み込みコマンド

RASS2 の組み込みコマンドの一部を表 3.2に示す．オブジェクトへメッセージを送るためには sm() 文を利用する．文中の *Arg* は引数であり，不要であれば指定しなくてもよいし，複数指定することもできる．メッセージ名が属性名であれば，送り先オブジェクトの属性を参照し，その結果（属性値）がテンポラリ変数 *Arg* に単一化される．単一化とは，Prolog 言語で見受けられる代入操作のことであり，２つの変数の状態を同じ内容に設定する動作を表す．この際，値を持つ変数（定数でもよい）から値を設定されていない変数へその値が写される．なお，sm は，send message の省略形である．

RASS2 では，必要なクラスをファイル単位で保存しておき，それらを load() コマンドを用いてシステム内に導入する．これにより，図 3.2と図 3.3に従って記

述したクラスが RASS2 上に展開される．この段階で実体としてのクラスオブジェクトも RASS2 上に実現される．クラスの導入は，一般的にはシステムの起動前に行われるが，このコマンドを使ってシステムの実行中でも導入することができる．

クラス及びインスタンスのルールの適用を開始（活性化）するには

```
activate_object()
```

を用いる．あるいは，このコマンドの代わりとして，agent クラスのメソッド activate（引数は不要）を用いることもできる．これに対比して，ルールの適用を止めるには

```
inactivate_object()
```

を用いる．この代わりとして agent クラスのメソッド inactivate（引数は不要）を用いることもできる．

インスタンスの生成や削除に関するプリミティブなコマンドとして，クラスからインスタンスを生成するための create_instance() と，インスタンスを削除するための remove_instance() がある．通常は，インスタンスの生成や削除はこれらの文を直接利用しないで，それぞれ object クラスのクラスメソッド create_instance() とインスタンスメソッド kill（引数不要）を継承して利用する．

RASS2 では，C 言語等の main 関数に該当するものとして初期設定ファイル initialize がある．これは固定のファイル名である．初期設定ファイルには，システムの初期段階で必要となるクラスの導入，インスタンスの生成，オブジェクトの活性化をコンマで区切って順に記述する．システムが起動したら，初期設定ファイルの内容が順番に実行される．この間，各オブジェクトは自発的に振舞うことはできない．初期設定ファイルを必ず用意しておく必要はない．この場合，利用者は，コマンドラインから load() や sm() を用いて目的の動作を実現させることができる．

3.3.6 開発環境

RASS2 は，Linux 上で開発され，稼動する．また，拡張されたオブジェクトに関する部分は，Prolog (SWI-Prolog) によって実装され，行列演算などの数値計算の箇所は，C 言語が利用されている．

3.4 RASS2 のプログラム例

3.4.1 例 1:インスタンスの生成

最初の例として，クラスがクラスルールに基づいてインスタンスを作成する動きを示すプログラムを紹介する．これは，RASS2 の起動直後に，利用者に対して解析対象のデータファイル名の入力を促すためのクラスルールとそれに関するクラス属性とクラスメソッドである．これらは，クラス orgData で実現されている．

図 3.5に示したプログラムは，クラス定義の一部である．created_instance というクラス属性は，システム上に実在する orgData のインスタンス名を記録するためのものである．クラスメソッド count_instance(Num) は，システム上に実在する自分のインスタンス数を数えるものである．その答えは引数 Num に単一化される．その require() ブロック内の exist_slot() は引数に記した属性が存在するかどうかを確認する組み込みコマンドである．

クラスメソッド create_instance0 は，利用者から解析するデータファイル名を聞き，クラスメソッド create_instance1(Name) を呼び出すものである．この require() ブロックでは，利用者を表すクラス user に対してファイル名を入力するように促し，入力してもらったデータファイル名 DataFile が存在するかどうかを組み込みコマンド exist_file() で確認している．利用者にファイル名を問い合わせた命令の中の[と]で囲まれた部分は，実際に出力装置に表示される情報を記述する部分である．ただし，システムがそれを表示する前にこの中を評価するので，この中に表示させたい変数の内容や文をカンマで区切って記述することができる．do() ブロックでは，引数を持つ create_instance1(Name) を呼び出している．この場合，入力されたファイル名 DataFile が create_instance1 の引数 Name へ写される．

```
orgData is_subclass_of lsMatrix : (
 class_slot( created_instance type_ list ),
 class_method(
   count_instance(Num) is_defined_by (
     require(exist_slot(created_instance)),
     do(sm(created_instance, size, Num)),
     explanation('Count number of instances')
   ),
   create_instance0 is_defined_by (
     require(
       sm(user, output, ['Input data file name:']),
       sm(user, inputWord, DataFile),
       exist_file(DataFile) ),
     do(
       sm(self, create_instance1, DataFile)
     ),
     explanation('Create instance') ),
   create_instance1(Name) is_defined_by (
     require(
       exist_file(Name),
       sm(created_instance, nomember, Name)
     ),
     do(
       create_instance(orgData, Name),
       sm(Name, readFile, Name),
       sm(created_instance, push, Name),
       sm(Name, activate) ),
     explanation('Create instance') ) ),
 class_rule(
   create_my_instance$2 is_defined_by (
     condition(
       sm(self, count_instance, Size),
       Size == 0 ),
     propose(sm(self, create_instance0)),
     explanation(
      'First, we need to read data' )))
).
```

図 3.5 orgData のインスタンス生成用プログラム

create_instance1(Name) は，インスタンスを生成するクラスメソッドである．この do() ブロックでは，表 3.2の create_instance() を使用して Name という名前のインスタンスを生成する．そして，そのインスタンス Name に対してデータファイル名 Name を引数に readFile というメッセージを送り，データファイルから解析データを読みこませている．readFile は orgData のインスタンスメソッドである．次に，created_instance に生成したインスタンス名 Name を push により登録し，最後に，生成したインスタンスを activate により活性化させている．

クラスルール create_my_instance は,“ orgData のインスタンスがまだ 1 つも生成されていなければ，それを作る ”という知識を表現したものである．condition() ブロックでは，orgData クラスを意味する self に対して，生成したインスタンス数を尋ね，その値が 0 かどうかを比較している．そして，これが成り立てば，create_instance0 を実行するように提案する．RASS2 の起動直後には，orgData のインスタンスは存在しないので，利用者はシステムから自動的にこの助言を得ることになる．RASS2 のこのような “お節介な” 動きは，操作方法をよく知らない初心者にとって非常に効果的に働く．

3.4.2 例 2:ルールの適用

次の例は，明らかに不適切な問題があった場合のルールの適用に関するものである．具体的には，クラス numData で定義されている外れ値に関するものである．解析データに外れ値が存在することは，通常，好ましくない．探索的データ解析では，外れ値を極外値と外側値に区別して取り扱うことがあるので，RASS2 でも外れ値をこの 2 つに区別している．図 3.6には，外れ値（極外値）を求めるためのインスタンス属性とインスタンスメソッドの定義を記す．また，図 3.7に外れ値（極外値）があった場合の対策をルールとして表現したものを記す．

図 3.6の fFence は係数 (閾値) を，fValue は極外値の位置を保持するための属性である．係数の値はデータの特性に応じて変化されるものであるが，探索的データ解析で利用される極外値の閾値 [55] を参考に，デフォルトとして 3 がその初期設定メソッド内で設定されるようにしている．初期設定メソッドとは，インスタンスが生成された時に自動的に呼び出されるメソッドのことで，RASS2 で

```
slot(
 fFence type_ number slot_comment_ 'Fence for far out values',
 fValue type_ points slot_comment_ 'Position of for out values'
),
method(
 calculate_farout is_defined_by (
  require(
   exist_slot(fFence, data, fValue),
   sm(self, row_ordered_label, RowLabel),
   sm(self, col_ordered_label, ColLabel)  ),
  do(
   local_variable( up     type_ number,
                   dw     type_ number,
                   uHinge type_ lVector,
                   lHinge type_ lVector,
                   hinge  type_ lVector ),
   sm(self, hinge, LH, UH, H),
   uHinge:=UH, lHinge:=LH, hinge:=H,
   foreach(Y, ColLabel, [
     up:=uHinge(Y)+fFence*hinge(Y),
     dw:=lHinge(Y)-fFence*hinge(Y),
   foreach(X, RowLabel, [
     if(or(data(X,Y)>=up, data(X,Y)=<dw),
       [sm(fValue, pushPoint, X, Y)])
     ]) ]) ),
  explanation('Search for far out values'))
),
```

図 3.6 外れ値（極外値）のための属性とメソッド

はコンストラクタと呼んでいる．インスタンスメソッド calculate_farout で実際に極外値を計算する．require() ブロック内の data は，クラス lMatrix のインスタンス属性であり，行列型の解析データを表している．numData が行と列のラベルを持つ行列を表すクラス lMatrix を継承しているので，このように上位クラスの属性を利用することができる．そして，自分自身に row_ordered_label と col_ordered_label というメッセージを送って，要素が 1 から行数（あるいは列数）までのリスト型データをそれぞれの引数に単一化する．

do() ブロックでは，まず，メソッドの実行に必要なローカル変数を 5 個宣言している．ここで lVector は，ラベルを持つベクトルを表す組み込みクラスである．次に，hinge というメッセージを使用して，上ヒンジ，下ヒンジ，ヒンジ散布度を求めるインスタンスメソッドを呼び出している．ここでヒンジとは，データをその中央値で区切り，さらにそれらの中央値を求めたものである．上側のヒンジを上ヒンジ，下側のヒンジを下ヒンジ，上ヒンジと下ヒンジの差をヒンジ散布度という [55]．foreach 文による 2 重ループを使用して，各列要素（変数）ごとの上側外境界点と下側外境界点を求め，その行要素（観測値）に極外値が無いかを if 文を利用して判断している．

データに外れ値が含まれていれば，そのまま解析を進めることはよくない．対応する行（観測値）を外れ値として削除するか，その列（変数）を対数変換することが簡便な方法である．図 3.7の delete_observation は，極外値を最も多く含む変数を探し，その数が 1 個であれば，それに対応する観測値を削除するように提案するインスタンスルールである．condition() ブロックでは，メソッド get_farout_colMax を呼び出すことで，極外値を最も多く含む変数を Col として，その変数の極外値の数を Size として取得する．Size が 1 と等しければ，次に，削除しようとする観測値名を得るために，Col を引数としてメソッド get_faroutValue を呼び出し，対応する観測値名を Obs に単一化させている．ルールの引数は，ルールの処理に必要なものではなく，提案先 (オブジェクトの外部) に渡す情報を記述するものである．従って，このルールが利用者に提案されるときには，Obs は具体的な観測値名と置き換わっている．このルールの第二引数 farout は，そのままの状態で提案先に情報として渡される．propose() ブロック内では，このルー

```
rule(
  delete_observation(Obs,farout)$3 is_defined_by(
    condition(
      sm(self, get_farout_colMax, Col, Size),
      Size == 1,
      sm(self, get_faroutValue, Col, Obs)
    ),
    propose(
      sm(self, show_BoxPlot, Col),
      sm(self, delete_observation, Obs),
      sm(self, inactivate) ),
    explanation('Delete a far out value')
  )
),
```

図 3.7 外れ値（極外値）のためのルールの一部

ルが選択された時に実行する処理を記述している．具体的には，condition() ブロックで求めた変数 Col の箱髭図をメソッド show_BoxPlot を使って表示する．次に，メソッド delete_observation により，対応する観測値を削除した新しい numData のインスタンスを生成する．外れ値があるので，このデータ（インスタンス）の解析を続けることは好ましくない．そこで，クラス agent のインスタンスメソッド inactivate を使って自発的動作を止めさせている．なお，この例の delete_observation のように，ルール名とメソッド名が同じであっても，ルール名がメッセージ名にはならないので問題はない．

外れ値を求める時に利用する係数の値を変更したいときは，属性の値だけを変更すればよい．関係するメソッドやルールを変更する必要はない．また，この例に関連して，例えば外れ値が複数あった場合の対処法を実現するには，numData を継承した新しいクラスを作成し，追加分の機能を定義すればよい．

```
proposal >>> test_lsm proposes "check_t_value" : (3.3)[1]
proposal >>> test_lsm proposes "check_f_value" : (3.3)[2]
proposal >>> test_lsm proposes "check_linearity" : (3.2)[3]

adviser >>> test_lsm proposes "delete_indepVar(x1, multicolinearity, severe)" : (4.1)

|:
```

図 3.8 RASS2 からの示唆の提示例

3.5 RASS2 の実行と RASS との比較

RASS2 を実行すると，図 3.8のように解析の現状から適用できるルールのリストが実行画面に表示される．利用者から命令が与えられ，システム上で何らかの変化が起きれば，オブジェクトの自発性により，その変化に応じてこのリストの内容が更新される．このようにして，RASS2 は利用者に助言を与える．

このリストの中では，adviser からの提案が最も優先度が高い．利用者がそれを実行に移したい時には，次のように adviser に go というメッセージを送る．

```
|: sm(adviser, go).
```

proposal からの提案は，adviser のものよりも優先度は低いが，適用可能なルールを優先度の高い順に提示している．括弧の中の数字が大域的な優先度，カギ括弧の中の整数が順番を表している．利用者が，この中のルールを実行したい場合には，順番を引数に持たせて adviser に go を送る．

```
|: sm(adviser, go, 3).
```

あるいは，sm 文を利用してリストに無い命令を直接与えることもできる．RASS2 の実行画面のスナップショットを図 3.9に示す．

RASS2 には，以前のバージョンの RASS[50] とほぼ同じ内容の知識がルールとして実装されている．従って，RASS2 の提案に従って自動解析を行なうと，RASS とほぼ同じ解析結果を得ることになる．ただし，このように自動的に得られた結果は，RASS と同様に，専門家レベルの解析結果とは言えず，初歩的なミスの無い程度のものである．

RASS2 に限らず，統計解析システムにデータ解析に関するあらゆる知識をはじめから入れておくことは困難である．しかしながら，現実にはデータ固有の知

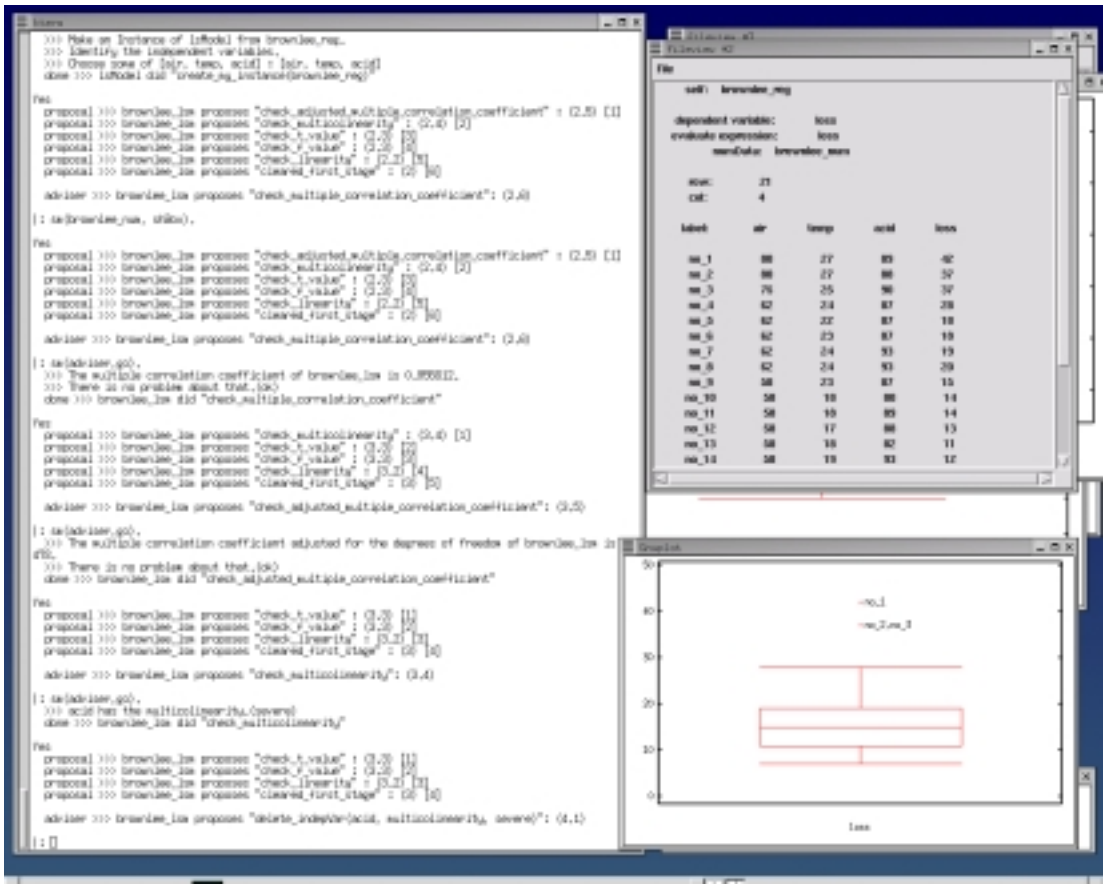

図 3.9 RASS2 の実行画面

識などをシステムに実装して解析を行いたいという要望は強い．このような場合には，利用者は，必要な手続きやルールを RASS2 に組み込まれているクラスを拡張することで RASS2 内に実装する．例えば，ある特定領域用の回帰モデルを lsModel を継承した myModel と名付けたクラスで実装し，それを RASS2 の lsModel と置き換えることで特化した解析を実現できる．RASS は，必要なルールを関連する手続きと同じクラス内で管理していなかったので，このような変更や組みかえが容易ではなかった．

3.6 本章のまとめ

重回帰分析支援システム RASS2 は，支援機構を実現するために，ルールを内包するように拡張したオブジェクトを利用している．このオブジェクトにより，支援に必要なルールを関連する手続きやデータから分離させることなく実装することができた．これにより，一般的なオブジェクト指向の特徴であるモジュールの独立性の高さを RASS2 のオブジェクトにも持たせることができた．モジュールの独立性が高ければ，RASS2 の拡張性や柔軟性は高くなり，利用者のさまざまな要求を満たすことが容易になる．例えば，利用者が RASS2 に組み込まれていない独自の統計手法やその評価方法を新しく導入したり，既存の判断基準を変更して特定のデータに特化させた解析支援をさせたりすることが可能になる．

RASS2 は自発的にルールを評価する．そして，利用者に評価結果としての適用可能なルールのリストを提示する．ルールにはオブジェクト内で有効な優先度が与えられており，それに基づいて競合が解消される．ルールの記述方法は，なるべく分かりやすく，かつ，メソッドと同じような形式となるように設計されている．さらに，RASS2 では，基本的な重回帰分析の流れに沿ってクラス群を用意し，重回帰分析に必要なルールを関連するそれらの中に組み込ませている．このようなことから，利用者は，支援機構の枠組みを強く意識することなく，RASS2 のルールの変更や拡張を容易に実現することができる．

RASS2 は，利用者にとって解析を支援するための環境だけでなく，汎用的なルールを発見するための環境としても利用することができる．このような作業のためには，ルールを変更したり組み替えたりするための簡単な機構がシステムに要求されるが，RASS2 では独自のオブジェクトを利用することでこのような要求にも対応することができるであろう．

4. 対話型プログラミング環境でのアクティブルールの効果

4.1 はじめに

プログラミングの形態は，対象となるソフトウェアシステムの要求に応じてさまざまなものが存在する．その中で，対話型プログラミングは，利用者がシステムに命令を少しずつ与え，その結果を見ながら目的の処理を実現していく方法である．このため，このような環境は，古くから主にプロトタイプ開発や問題解決型システムに利用されてきた．また，データ解析システムにおける利用者とシステムのインタフェースとしても利用されてきた．

近年，対話型プログラミング環境で利用されるスクリプト言語でオブジェクトを記述できる能力を持ったものがいくつか公開された．例えば，Perl[33] や Ruby[52] は，オブジェクト指向プログラミングを可能とするスクリプト言語である．また，Java 用のスクリプト言語である Pnuts[48] も，その言語仕様の拡張性の高さから独自のオブジェクト指向プログラミングの枠組みを提供する．

オブジェクト指向スクリプト言語は，汎用的なオブジェクト指向プログラミング言語よりも簡単な構文を持ち，自由度の高い運用を利用者に提供する．しかしながら，オブジェクト指向プログラミングは，インタラクティブなシステム開発のために考案された方法論ではないので，これらの処理系を用いて開発を行うことは決して楽な作業ではない．

そこで，対話的環境でのオブジェクトの運用性を高めるために，オブジェクトに特殊機能としてのアクティブルールを持たせることを試みた．アクティブルールは，手続きの能動的起動条件を持つもので，メソッドの実行終了時に評価され，その中で条件の正しいものが逐次実行される．アクティブルールを追加したことで，データ解析システムで見られるような対話型のプログラミング時に有効な利点を見つけることができたので，本章ではこれについて述べる [42].

次節では，アクティブルールの定義を行い，3 節でその適用事例を紹介する．4 節でアクティブルールの評価を行い，5 節で Pnuts 言語におけるアクティブルールの実装法について簡単に紹介する．

4.2 アクティブルール

本研究でオブジェクトに追加させたアクティブルールは，Condition-Action 型の構造を持つ．これは，条件部 (Condition) と実行部 (Action) から構成され，それぞれメソッドの特殊形である．クラスは複数のアクティブルールを構造的に保持することができ，それらはメソッドと同様に継承される．

アクティブルールは属するクラス単位で次のように評価される．

1. あるインスタンスのメソッドが実行される

2. そのインスタンスに属する全てのアクティブルールの条件部が評価される

3. その中で真と評価されたルールの実行部が順不同で逐次実行される

ルールの実行部でメソッドが呼出されたとしても，連鎖的に (1) に処理が移ることなく，一連の処理 (1,2,3) は一巡で終了する．

本章では，この動きを Pnuts 上で実装した．Pnuts では，オブジェクト記述を DynamicClass という Java クラスを利用して実装している．このクラスを拡張して上記の動きを実装することにした．アクティブルールは，その条件部と実行部をそれぞれ Pnuts 関数として表現し，ひとつの組としたものを DynamicClass のフィールドとして登録する．さらに，対話型環境下でメソッドの実行が終了して，処理をその環境へ戻す直前にアクティブルールを評価・実行する流れを追加した．Pnuts への実装の方法については，4.5節で述べる．

アクティブルールは次のように定義される．

```
rule(<NAME>, <CONDITION>, <ACTION>)
```

ここで，<NAME>はアクティブルールの識別名，<CONDITION>は条件部，<ACTION>は実行部である．制約条件として，条件部は論理値を返す関数とする．また，条件部と実行部で記述する手続きには引数を与えない．登録したアクティブルールを削除する場合は，removeRule(*NAME*) コマンドを使用する．

```
use("pnuts.oop")
DataVector = defclass("DataVector")
DataVector.var("data")
DataVector.var("sum")
DataVector.var("size")
DataVector.method("new",  // constructor
    function (x) { this.data = x
                   this.size = x.length
                   this.sum  = 0
                   this.calcSum(x) })
DataVector.method("calcSum",
    function (x) {for(i=0;i<x.length;i++){
                   this.sum += x[i] }})
DataVector.method("getSum",
    function () {return this.sum})
%
obj = DataVector([1,2,3,4,5]) //create an object
```

図 4.1 Pnuts によるクラスの定義例

```
DataVector.var("mean")
DataVector.method("calcAverage",
    function () {this.mean = this.sum / this.size})
DataVector.method("getAverage",
    function () {return this.mean})
%
obj.calcAverage()
obj.getAverage() // ==> 3
```

図 4.2 オブジェクトの直接的な拡張法

4.3 適用事例

4.3.1 問題の設定

我々が適用しようとする問題の基本的な条件は以下の通りである．

1. 利用者が対話的にプログラムを入力する
2. 宣言済みのクラスにはいくつかのフィールド（データ）と手続き（メソッド）があり，すでにその実体（オブジェクト）がシステム上に存在する
3. そのオブジェクトにメソッドとこのメソッドの実行結果を保持するためのフィールドを追加する
4. 計算手法を追加されたクラスを他のユーザと共に継続的に利用する（以後のオブジェクトの生成時には，この拡張が反映されることを前提とする）

ここでのオブジェクトは，Pnuts や Ruby と同様にフィールドとメソッドだけから構成され，単一継承機構を持ち，アクセス制限や型指定などの制約のないものとする．

ここで具体的な例として，ベクトル型の数値データを表現したクラスを取り上げる．このクラスを Pnuts (1.0rc1) で記述したものを図 4.1 に示す．文法の詳細は

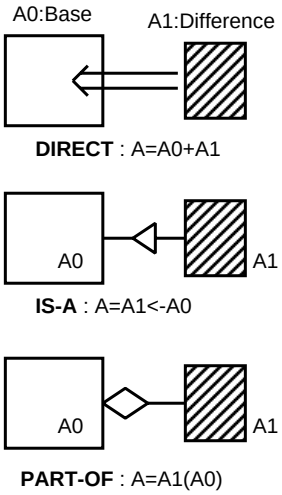

図 4.3 オブジェクトに対する 3 つの拡張方法

省略するが，defclass() がクラス名の定義，this がインスタンスを表し，var() がフィールドの追加，method() がメソッドの追加を行なうためのコマンドである．また，new というメソッドはコンストラクタとして扱われる．

我々が想定しているのは，このプログラムの実行後に，宣言しているクラスに平均値を求めるメソッドを追加し，その結果をフィールドに保持させることである．このような問題に対しては，次の 3 つの方法が考えられる（図 4.3参照）．

1. 対象となるクラスを直接拡張する方法
2. 継承（IS-A 関係）による方法
3. 継承（PART-OF 関係）による方法

次節では，これらの解決方法について検討する．

4.3.2 一般的な拡張方法

対象となるクラスを直接拡張する方法とは，利用者が図 4.1の入力に引続き，DataVector に対して図 4.2のように必要なフィールドとメソッドを直接追加する

ことである．図 4.2のプログラムでは，追加したメソッド calcAverage を明示的に呼び出せば，フィールド mean の値が充足される．このような方法は，一般的に，クラスの動作を不安定にする可能性が高いと言われている．

もうひとつの継承による方法は，オブジェクトを拡張するための最も一般的な選択である．継承には，IS-A 関係による継承と PART-OF 関係による継承があるが，この問題に対しては，IS-A 継承が一般的である．これを実装したものが図 4.4である．図中の super は親クラスを表す識別子である．この方法は，前述のクラスを直接拡張する方法と比べれば，既存のクラスの安定性を損なうことなく，より確実に実装することができる．しかし，この方法は，クラスの機能を追加するたびに新しいクラスを定義しなければならなくなるので，利用者にとってクラスの管理を煩わしくしてしまう．従って，対話型プログラミング環境では，先の直接拡張する方法が一般的に好まれる傾向にある．

PART-OF 関係を使用して機能の拡張を実装した例が図 4.5である．この例では，PART-OF 関係で継承することは概念的に好ましくないが，比較のために紹介する．IS-A 型継承と比較すると，データの重複がなくなるが，新しいクラスの宣言や関係するオブジェクトを設定するための複数のコマンドが要求される．

追加した機能を評価し，その有効性が認められた場合，利用者が次に要求することは，以後新たなオブジェクトが生成される度に，この追加分が自動的に反映されるように設定することであろう．これに対して，直接拡張させた方法では，図 4.1の中のコンストラクタに目的のメソッド（calcAverage()）を追加したものを再定義する必要がある．IS-A 型継承でも，直接拡張させた方法と同様に，派生クラスのコンストラクタを再定義することになる．前者の場合はコンストラクタ用プログラムが作業中に分からない場合が考えられる．また，もとのコンストラクタを入手することができたとしても，そのプログラム量に比例して利用者の負担が増加し，現実的な作業でなくなる可能性も考えられる．

4.3.3 アクティブルールを使用する拡張方法

これに対して，アクティブルールを使用すれば，コンストラクタを改めて定義する必要なく，要求を満たすことができる．図 4.6は，図 4.2に対応するプログラ

```
NewDVector = defclass("NewDVector", DataVector)
NewDVector.var("mean")
NewDVector.method("calcAverage",
    function () {this.mean = this.sum / this.size})
NewDVector.method("getAverage",
    function () {return this.mean})
NewDVector.method("new",  // constructor
    function (x) { super.new(x) })
%
new_obj = NewDVector([1,2,3,4,5])
new_obj.calcAverage()
new_obj.getAverage() // ==> 3
```

図 4.4 IS-A 継承の例

ムである．図中のアクティブルールの条件部では,“ フィールド mean の値が空であれば ”という条件を表現している．このアクティブルールが組み込まれると，新たにこのクラス DataVector のオブジェクトが作られるたびにこのアクティブルールは真と評価され，自動的にメソッド calcAverage() が実行に移される．図 4.4に対応させるアクティブルールも図 4.6 と同様である．ただし，この場合，定義するアクティブルールを DataVector ではなく，NewDVector に追加する．

4.4 アクティブルールの評価

前章では，アクティブルールを使用することで，一般的な方法に比べ，オブジェクトの拡張時での作業を簡素化でき，その記述も直感的に行うことができることを示した．これは，アクティブルールがメソッドの実行後に評価されるので，メソッドにより引き起こされる何らかの状態を見て，必要であれば能動的に別のメソッドを起動することに起因する．利用者としては，アクティブルールがオブジェクトを常に設定した状態に維持させてくれていると考えることもできる．紹介し

```
NewDVector = defclass("NewDVector")
NewDVector.var("pt", DataVector)
NewDVector.method("setPart",
    function (arg) {this.pt=arg})
NewDVector.var("mean")
NewDVector.method("calcAverage",
    function () {
      this.mean=this.pt.sum/this.pt.size})
NewDVector.method("getAverage",
    function () {return this.mean})
%
new_obj = NewDVector()
new_obj.setPart(obj)
new_obj.calcAverage()
new_obj.getAverage() // ==> 3
```

図 4.5 PART-OF 継承の例

```
DataVector.var("mean")
DataVector.method("calcAverage",
    function () {this.mean = this.sum / this.size})
DataVector.method("getAverage",
    function () {return this.mean})
DataVector.rule("average",
    function () {this.mean == null},
    function () {this.calcAverage()})
```

図 4.6 アクティブルールの使用例

た例でも,“ あるフィールドを常に充足しておく ”, という状態をアクティブルールが能動的にやってくれていると考えることができる．アクティブルールは，そのクラスの普遍的な状態（性質）を表現するために利用される．

あるクラスのメソッドの実行に伴うアクティブルールの評価は，実行したメソッドと同じクラスのアクティブルールだけが評価される．従って，もしそのメソッドの実行が他のオブジェクトに何らかの変化を引き起こしたとしても，そのオブジェクトのルールが評価されることはない．変化を引き起こされたオブジェクトのルールを評価するためには，利用者が明示的にルール評価のためのメッセージ (sync()) をそのオブジェクトに送信しなければならない．このようなアクティブルールの局所的な振舞いは，利用者がオブジェクトの動きを追従しやすく，また，メソッド呼出しの無限ループに陥ることを回避する利点を持つ．

アクティブルールを活用すれば，クラスにおけるメソッド間の独立性を高くすることができる．例えば，２つのメソッド m1 と m2 があり，m2 の起動が m1 に依存している場合には，m1 側で m2 の起動文を記述することになる．アクティブルールを利用すれば，両者を独立させたまま，２つのメソッドを起動させることもできる．具体的には，m1 の実行をキャッチするようにアクティブルールの条件部を書き，その実行部で m2 を呼び出す．

ここで具体的な例を紹介する．図 4.7には，先ほどのプログラム例と同じクラス DataVector の中の２つのメソッドを示す．メソッド addData は，インスタンスの要素を追加するためのものである．メソッド checkData は，要素の値が 10000 以上であればそれを知らせるためのものである．このような要素の値の確認は，要素が変更されたときに呼び出されるべきものである．従って，addData で要素が追加された後で，addData が checkData を呼び出すようにプログラムされている．ここで，引数を２つ与え，その２つの引数を要素として追加するメソッド addData を追加することを考えてみよう．おそらく，要素を追加するという作業は容易に実現できるかもしれないが，checkData をプログラムの最後に入力することを忘れてしまう可能性が考えられる．また，何らかの設定条件の変更により，checkData が削除されたり，変更されたりすると，addData の動作が不安定なったり，処理に無駄が生じる怖れも考えられる．これに対して，アクティブルールを

```
DataVector.method("addData",
    function () {
        this.data = this.data + [x]
        this.checkData() })
DataVector.method("checkData",
    function () {
        d = this.data
        for(i=0; i<d.length; i++){
          if(d[i] > 10000 )
              println("over 10000!")
        }
    })
```

図 4.7 あるメソッドが他のメソッドに依存しているプログラム例

使用してこの例を実装したものを図 4.8に示す．メソッド addData と checkData は独立させていることがわかる．両者の関係は，アクティブルール checkData により保たれていることになる．このアクティブルールの条件を“ データが変更されていれば ”とした．これを実現するためには，このルール用に属性をひとつ宣言する必要や，何らかのメソッドが実行されるたびにルールの条件部の確認を行う必要があるが，メソッドの追加・削除を繰り返すようなプログラミング時には有効である．

アクティブルールの条件部の評価は，そのクラスのメソッドが起動されるたびに行われる．従って，保有するアクティブルールの数に比例してそのための処理時間は増加する．これに対して，処理効率を高めたい場合には，アクティブルールをメソッドに移行すべきである．この際，アクティブルールの記述がメソッドのそれと同じであるので，容易に移行することができる．また，複数のアクティブルールが真と評価され，その記述が干渉しあうような場合には，一連の評価作業が終了しても，ルールの記述が正しく反映されないこともある．利用者は，こ

```
DataVector.var("preData")
DataVector.method("addData",
    function (x) {
        this.data = this.data + [x] })
DataVector.method("checkData",
    function () {
        d = this.data
        for(i=0; i<d.length; i++){
          if(d[i] > 10000 )
              println("over 10000!")
        } })
DataVector.rule("checkData",
    function () {
        this.preData != this.data },
    function () {
        this.checkData()
        this.preData = this.data })
```

図 4.8 アクティブルールを使用して 2 つのメソッドの関係を疎にした例

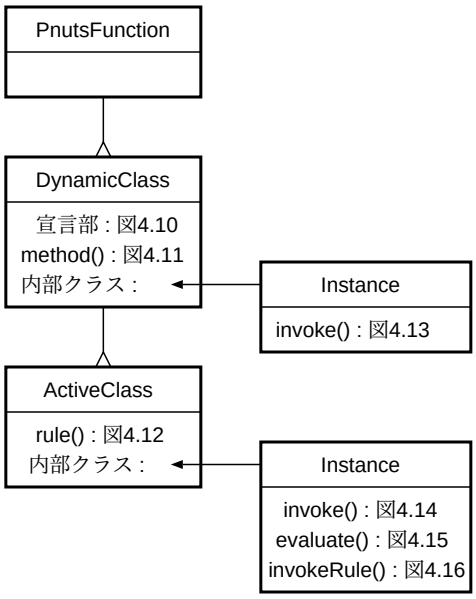

図 4.9 アクティブルールを実装するためのクラスの関係

れらの問題を避けるようにプログラムを作成する必要がある．

4.5 Pnuts におけるアクティブルールの実装

先述の通り，本章で取り上げたアクティブルールは，Pnuts のクラス定義環境で実装された．ここでは，この実装方法について簡単に述べる．まず本節で取り上げる図（プログラム）の関係を図 4.9に示す．Pnuts のクラス定義は，DynamicClass というクラスで実装される．Pnuts 上では，このクラスを利用して対話型のオブジェクト指向プログラミング環境を実現している．

DynamicClass の宣言の一部を図 4.10に示す．図中の . . . は省略を表す．Pnuts は Java 言語だけで開発された Java 言語のためのスクリプト言語である．プログラムの最初には，DynamicClass を実装するために必要な Pnuts 用の Java クラスと一般の Java のクラス（図中では省略されている）をインポートしている．クラス DynamicClass は，Pnuts の関数を表すクラス PnutsFunction を継承している．

Pnuts 上で新しいクラスを定義することは，DynamicClass のインスタンスを

```
import pnuts.lang.Pnuts;
import pnuts.lang.Package;
import pnuts.lang.PnutsFunction;
import pnuts.lang.Context;
import pnuts.lang.AbstractData;
...

public class DynamicClass extends PnutsFunction
                          implements Serializable {
  String name;
  private static Context ctx = new Context();
  Hashtable attributes;
  Hashtable methods;
  ...
```

図 4.10 DynamicClass の宣言（一部）

作成することである．図 4.1の例では，DataVector という DynamicClass のインスタンスを生成しているのである．プログラム中の属性 name は，そのインスタンス名を記憶しておくためのものである．ctx は，Pnuts の実行環境の内部状態を表すコンテクストである．var() や method() によって DynamicClass のインスタンスに登録された属性やメソッドは，それぞれプログラム中のハッシュ型属性 attributes と methods で記憶される．

DynamicClass にアクティブルールを追加する場合には，このクラスのソースコードを直接拡張するか，このクラスを継承した別名のクラスを定義すればよい．別名のクラス（ここでは，”ActiveClass”とする）を定義する場合には，

```
public static ActiveClass extends DynamicClass {
  Hashtable rules_condtion;
  Hashtable rules_action;
```

と宣言する．ここで，rules_condition と rules_action は，アクティブルールの条件部と実行部をそれぞれ記憶させるための属性である．

対話型環境でクラスにメソッドを追加するために利用したコマンド method() は，DynamicClass 内では図 4.11のように定義されている．例えば，図 4.1内で利

```
public void method(String name, PnutsFunction func){
  int narg = minArgs(func);
  String expr = packageStatement(func.getPackage()) +
         importStatements(func.getImportEnv(narg)) +
         "function (this, super) " + func.unparse(narg);

  PnutsFunction f = (PnutsFunction)Pnuts.eval(expr, ctx);
  Vector vec = (Vector)methods.get(name);
  if (vec == null){
    vec = new Vector();
    methods.put(name, vec);
  }
  if (vec.size() < narg + 1){
    vec.setSize(narg + 1);
  }
  vec.setElementAt(f, narg);
}
```

図 4.11 DynamicClass のメソッド method()

用した

```
DataVector.method("getSum", function (){return this.sum})
```

は，このメソッドの引数 name に"getSum"を，func にコマンドラインで与えた無名関数を使って生成された PnutsFunction のインスタンスが設定され，実行に移される．変数 narg には，func の引数の数が代入され，変数 expr には，func を DynamicClass 環境下で改めて宣言（登録）するためにメソッド定義をソースコードの状態に戻し (unparse())，情報を付加した文字列が代入される．ここで，メソッド定義のソースコードの前に付加された文字列 function(this, super) は，前述した識別子 this と super をメソッド内で利用できるようにするためのものである．これにより，第 2 引数として与えたメソッドは，次の行 Pnuts.eval(expr, ctx) において引数を 2 つ持つ無名関数として Pnuts により評価される．評価された結果（関数のポインタ）は変数 f に代入される．このように，Java の中で Pnuts の命令を文字列で作成し，それをクラス Pnuts に与えることで Java から

Pnuts を実行させることができることも Pnuts の特徴である．そして，変数 f を先に紹介した属性 methods の中に登録している．このメソッドを実行しようとした時点で，変数 f が呼び出され，それにより第２引数として与えたメソッドが Pnuts 上に展開されることになる．これは，Pnuts が関数の内部で関数の定義をすることができる機能と Java から Pnuts を呼び出せる仕組みをうまく利用している．

アクティブルールを登録するために利用するコマンド rule() の実体は，図 4.12 のようになる．アクティブルールをメソッドの特殊形としているので，アクティブルールの定義は，メソッドの定義と同じことを条件部用と実行部用に２回行っているだけである．

DynamicClass 内には，インスタンスを表すための内部クラス Instance が宣言されている．

```
public class Instance implements AbstractData, Serializable {
```

例えば，図 4.1の最後の行

```
obj = DataVector([1,2,3,4,5])
```

では，変数 obj に DataVector のインスタンス，つまり内部クラス Instance のインスタンスが代入されている．

本章でのアクティブルールは，あるインスタンスのメソッドが実行された後に評価され，条件部が正しければ実行に移される．この動きを実装するために，内部クラス Instance のメソッド invoke() を利用する．Pnuts では，あるインスタンスにメッセージが送られると内部でその受け取り側のクラスのメソッド invoke() が呼び出されるようになっている．例えば，

```
obj.work(1,2,3)
```

と Pnuts に与えると，内部では

```
obj.invoke("work",[1,2,3])
```

と解釈される．

```
public void rule(String name, PnutsFunction c, PnutsFunction b){
      int narg1 = minArgs(c);  // for cond
      int narg2 = minArgs(b);  // for body
      String expr1 = packageStatement(c.getPackage()) +
                   importStatements(c.getImportEnv(narg1)) +
                   "function (this, super) " + c.unparse(narg1);
      String expr2 = packageStatement(b.getPackage()) +
                   importStatements(b.getImportEnv(narg2)) +
                   "function (this, super) " + b.unparse(narg2);

      PnutsFunction f1 = (PnutsFunction)Pnuts.eval(expr1, ctx);
      PnutsFunction f2 = (PnutsFunction)Pnuts.eval(expr2, ctx);
      Vector vec1 = (Vector)rules_condition.get(name);
      if (vec1 == null){
        vec1 = new Vector();
        rules_condition.put(name, vec1);
      }
      if (vec1.size() < narg1 + 1){
        vec1.setSize(narg1 + 1);
      }
      vec1.setElementAt(f1, narg1);

      Vector vec2 = (Vector)rules_action.get(name);
      if (vec2 == null){
        vec2 = new Vector();
        rules_action.put(name, vec2);
      }
      if (vec2.size() < narg2 + 1){
        vec2.setSize(narg2 + 1);
      }
      vec2.setElementAt(f2, narg2);
    }
```

図 4.12 アクティブルールを定義するためのメソッド rule()

```
public Object invoke(String name, Object[] args, Context co){
  ...
  int narg = args.length;
  PnutsFunction func =
           DynamicClass.this.getMethod(name,nargs,this);
  if (func != null){
    return func.call(args, co);
  }
  throw new RuntimeException("no such method:" + name);
}
```

図 4.13 内部クラス Instance のメソッド invoke()

DynamicClass のインスタンス（例えば，DataVector）の内部クラスのインスタンス（例えば，obj）にメッセージが送られたときに呼び出される invoke() の一部を図 4.13 に示す．DynamicClass のメソッド getMethod() を使ってハッシュ型の属性 methods から該当するメソッドを PnutsFunction 型で取り出し，それに，call() を送ることでメソッドを実行させている．

このような仕組みを利用し，アクティブルールの自発性を実現させるようにメソッド invoke() を拡張した．これを図 4.14に示す．プログラム中の invokeFirst() は，先の図 4.13 と同じ内容のものである．まず最初に，この関数を利用して従来の方法でメソッドを実行に移す．変数 flg は，内部クラスの属性である．これは，invoke() の利用によるメソッドの呼出しが行われているかどうかを判断するためのフラグである．アクティブルールが，ルールからメソッドを呼び出しても再帰的にルールを評価しないようにするために利用されている．プログラム中の関数 evaluate() は，クラス内で定義されているルールを評価するための内部クラス Instance のメソッドである．このプログラムを図 4.15に示す．

関数 evaluate() は，条件部が真であるルールの名前を ArrayList 型で返す．その情報に基づいて，真であるルールの数だけ繰り返しながら，真であるルールの実行部を invokeRule() を使用して順番に実行する．

図 4.15に示した内部クラス Instance のメソッド evaluate() は，登録されているルール（つまり，ActiveClass のインスタンスに登録されているルール）をす

```
public Object invoke(String name, Object[] args, Context co){
  Object ans = new Object();
  ans = invokeFirst(name, args, co);

  if(flg == 0){
    Object dumy = new Object();
    ArrayList vec = evaluate(co);
    flg = 1;
    for(int i=0; i<vec.size(); i++){
      String fn = (String)vec.get(i);
      dumy = invokeRule(fn, co);
    }
    flg = 0;
  }
  return ans;
}
```

図 4.14 アクティブルールのためのメソッド invoke()

```
public ArrayList evaluate(Context context){
  Enumeration enum = getRules();
  ArrayList vec = new ArrayList();

  while(enum.hasMoreElements()){
    String name = (String)enum.nextElement();
    PnutsFunction func =
       ActiveClass.this.getCondition(name, 0, this);
    if (func != null){
      Object ans = func.call(new Object[]{}, context);
      if(ans.toString().compareTo("true") == 0) {
        vec.add(name);
      }
    }
  }
  return vec;
}
```

図 4.15 アクティブルールのためのメソッド evaluate()

```
public Object invokeRule(String name, Context context){
  if ("getType".equals(name)){
    return getType();
  }
  if ("getClass".equals(name)){
    return getClass();
  }
  PnutsFunction func =
       ActiveClass.this.getRuleBody(name, 0, this);
  if (func != null){
    return func.call(new Object[]{}, context);
  }
  throw new RuntimeException("no such rule:" + name);
}
```

図 4.16 アクティブルールのためのメソッド invokeRule()

べて集め，それらの条件部が実行環境の現状（この状況は，コンテクストが保持している）のもとで真か偽かを判断している．

アクティブルールの実行部を実行するのが，図4.16 に示したメソッド `invokeRule()` である．アクティブルールの実行部を単独で見た場合には，引数の無いメソッドと見なすことができるので，アクティブルールの実行には特別な処理をさせていない．以上のようにして，Pnuts にアクティブルールを追加した．Pnuts の設計の良さのために，アクティブルールの実装は比較的容易に行うことができた．また，今後さまざまな機能を追加することも容易に行うことができるであろう．

4.6 アクティブルールの関連研究

アクティブルールという言葉は，データベースの分野で既に使用されている [21]．一般的に，データベースでのアクティブルールは，保持するデータの状況の変化に応じてルールを能動的に起動させ，データのハンドリングや整合性を保つことを主目的としている．我々のアクティブルールもデータベースのそれと基本的には同じ考えである．例えば，普遍的な状態の維持という機能的な働きにつ

いては両者の違いはない．ただし，本研究では，インタラクティブなプログラミング作業時での効果について議論しており，対象が異なる．

オブジェクトにルールを追加する試みは，これまでにも数多く行われている．大別すると，オブジェクトに推論をさせる目的で組み込まれたものと，自律的な振舞いをさせる目的で組み込まれたものがある．前者のルールはいわゆるプロダクションルールに該当し，そのオブジェクトも推論のための特殊な機能を保持するので構造的に本研究とは異なるものである [5]．後者のケースは，エージェントに関する研究 [51] でよく見受けられるものであり，本研究と同じものであると考えられる．ただし，本論文では，対話型プログラミング上での効果を中心に議論したものであり，エージェント研究の目指す運用上の自律性を目指しているものではない．ただし，本システムを拡張すれば，エージェントシステムの開発プラットフォームとして利用できると考えている．

4.7 本章のまとめ

一般的に，対話型プログラミング環境では，できるだけ単純な文法規則で迅速に目的の作業を記述できることが要求される．本章は，オブジェクトにアクティブルールを記述できる枠組みを与え，それによりオブジェクトの動的な機能追加を簡便かつ直観的に行なえることを示した．

統計システムのプログラミング環境は，そのほとんどが対話型である．統計システムが大規模になり，オブジェクト指向の環境を取り入れなければ，うまく制御できなくなってきていることを鑑みれば，このような機構が，次世代の統計システムのプログラミング環境の一部として利用されることが期待される．

本章で示した例は，アクティブルールの能動性を活かし，本来利用者が条件を満たした時に与えるべき命令をアクティブルールを持つオブジェクトがそのルールに従って勝手に（自発的に）実行させているものである．与える命令量を減らすことが好まれる対話型環境では，このような動きは有効であった．また，本章の成果は，次章で述べる汎用型統計システム Jasp 上で利用することを計画している．具体的には，アクティブルールを用いて，入力操作の軽減を計ったり，解析の助言を行えるようにすることである．

5. 汎用統計システム Jasp のプログラミング言語

5.1 はじめに

近年のコンピュータ技術の発展は，データ解析に新しい可能性を提供した．例えば，ハードウェアの発達によって一度に処理できるデータ量が増し，より複雑な解析ができるようになった．また，ネットワークの発達，とりわけインターネットの普及により，必要なデータを手軽に検索することもできるようになった．このようなコンピュータを取り巻く環境の変化の中で，統計解析を行うためのシステムもこれに応じた変化が求められるようになっている．

コンピュータの黎明期より，データ解析のためのソフトウェアは開発され続けている．そして，その時代のコンピュータの能力に比例して，それらのソフトウェアも発展してきた．これらは，長年培われた枠組みに新しい機能を拡張するやり方で拡張が進められているものと，最新のコンピュータ技術をうまく導入できるように基本から設計されて開発が進められているものがある．我々も，近年のネットワークを中心としたコンピュータ技術の発展をひとつのターニングポイントとして捉え，従来の統計システムにはない操作性と機能，拡張性を持った統計システム Jasp[20, 14, 38, 15] を設計・開発している．

統計システム，あるいは統計パッケージと言われるデータ解析のためのアプリケーションソフトウェアは，統計解析の作業をいかに直感的かつ正確に行えるのか，さらに複雑なこと（様々なこと）ができるのかという視点で評価されることが多い．前者については，システムにおけるグラフィカルユーザインターフェース (GUI) の設計とシステムが持つ潜在的な機能に強く関係する．後者については，システムが利用者に対して提供するプログラミング機能の記述能力に強く関係する．

統計システムにとってプログラミング環境は非常に重要である．一般に統計システムの利用者のシステムに対する要求は多様であり，システムの開発者がどのように多くの機能をシステムに追加しようと，それですべての利用者を満足させることは難しい．従って，利用者が統計システムに対して実行させたい命令を正しく表現することができるプログラミング環境とその専用言語が要求される．こ

れは特に利用分野を特定しない汎用向けの統計システムに課せられる要求である．また，そのプログラミング言語は，統計システムが備えている機能を十分に活用・制御できる記述能力を備えていなければならない．

近年のインターネットの普及によるコンピュータのグローバル化により，統計システムに対する新しい可能性と要望が誕生した．例えば，インターネット上で公開されている多量なデータをそのまま利用したり，インターネット上で公開されている統計解析上有益なソフトウェアをシステムに組み込んで利用してみたいということが挙げられる．また，解析の分散処理とギガのオーダに向上したハードウェアの高性能化により統計システムが解析できるデータ量も利用者が把握できないほどまでに増加した．データを記述するフォーマットも XML など様々なものが考案されている．このような状況を踏まえれば，今後さらにデータ解析を取り巻く環境は大きく変わっていくことが予想される．ただし，これまでに培われてきた統計解析のための数多くのプログラム資産を無視し，活用しないことは得策ではない．有益な過去の資産を活かしながら，統計解析の新しい一面を統計システム上で実現しなければならない．

現在，普及している統計システムや統計パッケージも近年のコンピュータの技術革新の影響を強く受け，利用者にそのメリットが活かされるように拡張や変更が行われている．しかしながら，ほとんどの統計システムでは，この作業がそのシステムの開発者に委ねられていて，一般の利用者がシステムを拡張することは容易ではない．それらに備えられているプログラミング環境を用いても，多くのシステムでは，実装することが不可能であることが多い．この理由として，多くの統計システムのプログラミング機能は，一時的で非定型な処理（つまり，解析手法の追加）を記述することを主な目的と位置づけられており，利用者レベルでのシステムの拡張性を考慮してはいないということと，さらに，拡張性よりも正確な計算結果を算出することに重点を置かれてきたからであろう．正確な統計結果を算出することは，今なお重要な要求事項であることに変わりはないが，統計システムを取り巻く状況が多様化していく中で，統計システムの利用者レベルでシステムを拡張できる枠組みが要求される．

そこで本章では，従来の統計システムのプログラミング言語には見あたらない

統計システムを拡張できる枠組みを持った統計システム Jasp のプログラミング言語 (Jasp 言語) について述べる．

本章では，次節で統計システム Jasp とそのプログラミング言語の基本設計方針を紹介し，3 節で Jasp 言語の特徴について，4 節でそのプログラミング環境について，5 節で Jasp 言語を用いた Jasp の拡張例について説明する．また，3 節では，Jasp 言語におけるデータの宣言方法，プログラムの基本記述形態である Jasp 関数，オブジェクト指向プログラミングを可能にする Jasp クラス，Jasp 言語における Java クラスの利用方法，GUI への対応について触れる．

5.2 基本設計方針

我々が Jasp 及び Jasp 言語を設計するときに，特に気を付けた点は，次の通りである．

1. これまでの統計解析用のソフトウェアや統計システムの共通点や，それらに関する広く評価されている特徴をできるだけ取り入れる

2. 有益なプログラム資産を活用する

3. システムとして拡張性を高める

4. 直感的に操作できるインターフェースを備える

このような条件をできるだけ達成させるために，我々はシステム開発に Java 言語を採用した．

Java 言語は，汎用型のオブジェクト指向プログラミング言語である．これは，1991 年にサン・マイクロシステムズによって発表された．Java のプログラムは，バイトコードと呼ばれる形態にコンパイルされ，Java Virtual Machine (Java VM) 上で稼動する．従って，Java VM が稼動するコンピュータ上であれば，OS に依存することなく Java のプログラムを実行させることができる．Java 言語は，ネットワーク機能やそれに付随するセキュリティ機能，グラフィックス，リモート操作，データベースアクセスのための豊富なライブラリ群を標準で持っている．近

年，Java で開発されたプログラムがインターネット上で数多く公開されるようになってきた．

Java 言語は，高性能な汎用プログラミング言語である．しかし，これをそのまま統計システムのプログラミング言語に採用すると統計システムの利用者が十分に扱えない可能性がある．そこで，Jasp の言語機能として Java 言語のスクリプト言語である Pnuts[48] を利用することにした．ただし，Pnuts も汎用の言語であり，特に統計計算向きに設計されているわけではないので，我々は，Jasp と Pnuts の間にプリプロセッサを置き，Pnuts の長所を活かしながら Jasp 言語を実装した．プリプロセッサを利用すると処理速度の低下を招くが，実装のしやすさ，Pnuts 及び Jasp 言語の仕様変更に伴う作業量の軽減を計ることができる．

Pnuts は手続き型のプログラム言語である．その手続きは，Pascal や C 言語などで使われる関数と同様な文法に沿った関数で記述される．また，データ型の宣言を省略することができるので，これまでの多くの統計システムの言語機能と同様に，比較的簡便な文法規則に沿ってプログラムを作成することができる．Jasp 言語による基本的な手続きの記述形態を，この Pnuts の関数を統計解析用に改良した専用の関数（“ Jasp 関数 ”と呼ぶ）とした．Jasp の利用者は，一時的かつ小規模なプログラムを Jasp 関数を使用して実装する．

利用者が作成するプログラムは，断片的で小規模なものだけではない．作成したいくつかのプログラムをまとめて実行したり，あるプログラムから別のプログラムを呼び出すこともある．Jasp 関数は，記述上それぞれが独立しているので，複数の関数をひとつにまとめてうまく利用することが難しい．これにうまく対応するのがオブジェクト指向プログラミングである．関係する関数をオブジェクト指向の枠組みの中でうまく統合し利用者に提供できれば，利用者は，目的の作業を表現しているプログラムを実行させやすいし，プログラムを拡張する場合にもオブジェクト指向の継承機構を利用して安定的に行える†．そこで，Jasp 言語にオブジェクト指向プログラミングができる枠組みを実装させた．この枠組みは，Jasp 上で独自のクラス（“ Jasp クラス ”と呼ぶ）を取り扱うことができるものである．

†統計システムにオブジェクト指向プログラミングを導入する効果については，本論文の 2 章と 3 章で触れている．

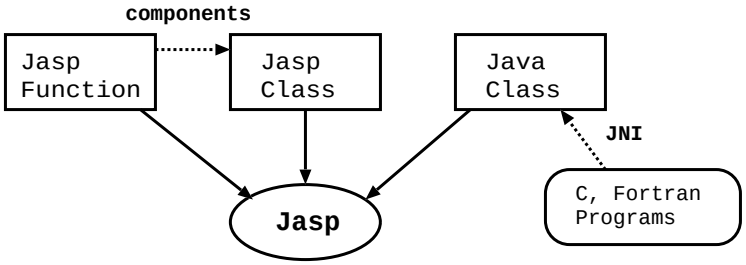

図 5.1 Jasp プログラミングの概念図

利用者が実際にプログラムを作成する場面を観察すると，その多くは，断片的なプログラムを作成し，その動きが正しいと確認できた後に，そのプログラムを拡張していき，これを繰り返すことで目的の作業を実行させる，という段階を踏むことが多い．従って，Jasp の利用者は，プログラミングの初期段階では，Jasp 関数を利用し，その動きを確認したのちに Jasp クラスを使用して概念的に関係のある Jasp 関数をひとつのクラスとしてまとめることで Jasp にプログラムを実装する．

Jasp 関数と Jasp クラスが利用者にとって役立つものであるとしても，Java, Fortran，C や C++で記述された統計解析用のプログラムをそのまま統計システム上で利用したいことがある．例えば，そのプログラムを過去に使用した経験があり，Jasp 上でも使ってみたいとか，Jasp 関数と Jasp クラスでは，表現することが困難な機能を実装したいという時である．Jasp 言語は，このような要求にも応えられる能力を持つ．まず，Fortran，C や C++で記述されているプログラムは，Java の JNI (Java Naitive Interface) 機能 [16] を用いれば，Java のプログラム中で他言語のプログラムを呼び出すことができる．そして，Pnuts の Java のクラスを直接参照できる機能を利用して，Jasp 言語は，これらのプログラムを間接的に呼び出すことができる．

このように Jasp 言語は，3 種類のプログラミングスタイルを持つ．Jasp 関数によるプログラミング，Jasp クラスによるプログラミング，Java 言語によるプログラミングである．これらの関係を図 5.1に，それぞれの目的を表 5.1に示した．

表 5.1 Jasp プログラミングの目的

構成要素	目的
Jasp 関数	計算手順の記述
Jasp クラス	統計解析の概念によって関連する Jasp 関数をまとめる枠組み
Java クラス	Jasp システムの拡張

近年開発されている統計システムのユーザインタフェースは，ウィンドウ環境のもとで直感的に命令を与えられるようになっているものがほとんどである．つまり，どのように優れたプログラミング環境を備えていても，統計システムに対して高い操作性を利用者は期待しているのである．しかしながら，グラフィカルな操作だけではプログラミングを行うことはできない．いくら優れたグラフィカル操作を実現したとしても，それだけでプログラミングの代わりを担うことはかなり困難であろう．従って，統計システムにおいては，グラフィカルなユーザインタフェースとキャラクタベースのプログラミング環境の統合が必要になる．

Jasp は，この点も考慮して設計されている．もし，プログラミング環境で新しい Jasp 関数を定義すると，それを表現したアイコンがグラフィカルな操作画面上に現れ，マウス操作だけでその関数を実行に移すことができる．

マウス操作で Jasp 関数を利用する場合，利用者が参照できる情報はその関数名と引数の数だけである．グラフィカルな操作画面上で，関数の本体を見せることは，ほとんどの利用者にとって煩わしいだけである．その関数を作成した利用者だけが利用するのであれば，特に問題はないが，もし他の利用者にも利用して欲しい場合には，このままではその関数に関する情報が乏しい状態となり，利用頻度が下がったり，誤用を招く怖れがある．そこで，グラフィカルな操作画面で Jasp 関数の情報を参照することができるように，Jasp 関数の定義の際にその関数の利用のための情報を追加できるような仕組みを実装した．

5.3 Jasp 言語

Jasp 言語は，これまでによく利用されている S や XploRe などの統計システムのプログラミング言語をできるだけまねて設計されている．これは，これらの統計システムを利用している人が抵抗なく Jasp の環境に馴染めるようにするためである．本節の前半では，このような部分の説明を行い，具体的に Jasp 言語がどのようにプログラムされるのかを紹介する．本節の後半では，Jasp 言語のオブジェクト指向プログラミングや Java 言語の参照方法，グラフィカルな操作画面のための情報記述方法について説明する．

5.3.1 データの操作

Jasp 言語は，データの型指定を省略することができる言語として設計されている．変数は，はじめに設定される値の型によってそのデータ型が決定される．

Jasp 言語で扱えるデータ型は，次の 3 種類に大別される．

1. 基本データ型
2. Jasp クラス
3. Java クラス

基本データ型は，一般的なプログラミング言語と同じように，整数，実数，文字列などを表すものである．Jasp クラスは，Jasp 言語専用のクラス定義機構を利用して宣言されたクラスのことである（5.3.3節参照）．Java クラスは，Java 言語を用いて作成されたクラスのことである（5.3.4節参照）．利用者は，これらのデータ型を同等に扱うことができる．

データ解析では，行列演算が多用される．従って，Jasp 言語では行列を表現するためのデータ型 (JaspMatrix) が用意されている．JaspMatrix は，正確には Java クラスであるが，利用者が扱いやすいように特殊な表記法が用意されている．例えば，次の式は，`mat` と名付けられた 2 行 3 列の行列を生成する．

```
mat = JaspMatrix(2,3)
```

要素を指定するには [ ] 演算子を利用して，例えば，

```
mat[1,2]
```

のように記述する．また，第 1 行の要素を削除し，第 1 列から第 3 列を選択するためには，

```
mat[!1,1:3]
```

と入力する．さらに，JaspMatrix は，行と列のラベルを持つことができ，それらを使用して次のように要素の指定ができる．

```
mat["no2", "x3"]
```

行列の演算子には，次のようなものがある．

```
x + y        ; 要素同士の和
x - y        ; 要素同士の差
x * y        ; 積
x.inv        ; x の逆行列
x.trans      ; x の転置行列
x.det        ; x の行列式
x.eigen      ; x の固有値ベクトル
```

ここで，利用例として，Durbin-Watson 統計量 [25] を求めるプログラムを示す．この統計量は次の式で計算される．

$$D = \sum_{i=2}^{n}(e_i - e_{i-1})^2 / \sum_{i=1}^{n} e_i^2$$

ここで，e_i は i 番目の残差である．この統計量は，誤差項に系列相関があるかどうかを検討するために利用される．

```
dw = 0.0
for(i=2; i<=res.nr; i++){
  dw = dw + pow(res[i,1]-res[i-1,1], 2.0)
}
dw = dw / (res.trans * res)
```

ここで，res は残差を表す JaspMatrix で，res.nr はその行数を表す．

5.3.2 Jasp 関数

先述の通り，Jasp 言語は，これまで多くのデータアナリストが利用してきた統計システムの言語と同じように手続き型の言語とした．手続き型言語は，最も利用されているコンピュータ言語の形態であり，目的の処理の流れに基づいて，そのままプログラムを記述できる特徴を持つ．Jasp 言語では，これに対応するものが Jasp 関数である．Jasp 関数は，入力としていくつかの引数を与え，出力としてひとつの計算結果を戻す．ただし，プログラムの流れを理解しやすくするために，関数の内部で外部のデータ（変数）の値を変更することができないようにしている．

Jasp 関数のフォーマットは次の通りである．

```
function NAME ( ARGUMENTS ) BODY
```

ここで，NAME は関数の名前，ARGUMENTS は引数のリスト，BODY は関数の本体である．引数のリストは空でもよいし，’,’ で区切られたいくつかの引数でもよい．もし同名の関数が存在しても引数の数でそれらは区別される．関数の本体は，いくつかの文 (statement) で構成される．もし複数の文が必要である場合には，それらを’{’ と’}’ で囲む．また，その際，各文は’;’ か改行で区切られる．他の手続き型言語のように，Jasp 言語は選択のための if 文や switch 文，繰り返しのための for 文や while 文などを利用することができる．関数の呼出し元への値の返却には，return 文が使われる．

Jasp 関数の内部で宣言された変数は一時的なものである．それらの有効範囲は，その関数の内部に限定され，他の関数によって変更されるようなことはない．呼出しに利用された引数の値が Jasp 関数内で変更されても，その変化は呼出し側に影響することもない．Jasp 関数は再帰呼出しが可能である．また，Jasp 関数の中で Jasp 関数を定義することができる．

先ほどの Durbin-Watson 統計量を求める Jasp 関数は，図 5.2 のようになる．

```
function durbin_watson( res ) {
  dw = 0.0
  for(i=2; i<=res.nr; i++){
    dw = dw + pow(res[i,1]-res[i-1,1], 2.0)
  }
  dw = dw / (res.trans * res)
  return dw
}
```

図 5.2 Durbin-Waston 統計量を計算する Jasp 関数

5.3.3 Jasp クラス

オブジェクト指向プログラミングは，互いに関係するデータ（属性）と手続き（メソッド）をまとめたオブジェクトによりシステムを構築することである．これを用いるメリットは，従来の手続き主体のプログラムよりもオブジェクトを用いた方が我々の思考形態に近い形でプログラムを捉えやすいこと，その継承機構による高い拡張性，などが長所として挙げられる．逆に，

1. オブジェクトを作成したり利用したりするには，オブジェクトの枠組みを熟知しておかなければならない

2. 常に対象となるオブジェクトと送りつけるメッセージの両方を指定しなければならない

3. オブジェクトの階層が大きくなりすぎると，目的のオブジェクトを見つけにくくなる

という運用上の問題点が指摘されている．

オブジェクト指向プログラミングを可能にした統計システムとしては LISP-STAT[31] が有名である．これは，リスト処理を得意とする純粋な関数型言語である LISP 言語を開発プラットフォームとして採用している．LISP 言語は，対話型操作が可能なインタプリタであり，標準ではオブジェクト指向プログラミングの枠組みは提供していないが，その言語仕様の柔軟性から比較的容易にオブジェク

ト指向プログラミングを実現できる．実際，CLOS[12] のようなシステムが発表されている．LISP-STAT の基本要素は関数である．データ解析の多くは実装されている関数を利用することで解析を進め，必要に応じてオブジェクトを利用するというハイブリッド型を採用している．すべてをオブジェクトにすると，利用者はどのオブジェクトの何と言うメソッドを使えばよいのかをいつも気にしなくてはならなくなるので，システムの初心者には不適であると言われている．従って，LISP-STAT のようなハイブリッド型は，汎用統計システムの設計方針としては適切であろう．

統計システムのプログラミング環境でオブジェクトを取り扱うことができれば，各種統計手法の枠組みに応じて解析データからオブジェクトを用意し，それにメッセージを送る方法で解析を進めることができる．従って，利用者は，手続き主体で解析を進めるのではなく，データを主体とした概念的な操作で解析を進められる．また，近年の統計システムは，GUI が標準である．そのウィンドウを多用する GUI の構築には，オブジェクト指向プログラミングが効果的なのは明らかである．我々の Jasp プログラミング環境は，このようなハイブリッド型でオブジェクト指向プログラミングを利用者に対して提供することができるように設計された．ただし，LISP-STAT や S 言語のようなプロトタイプベースのオブジェクトではなく，クラスベースのオブジェクトを利用者が定義できるようにしている．プロトタイプベースのオブジェクトは，対話型プログラミング環境では記述しやすいというメリットが考えられるが，プログラムの再利用性が低いなどの問題もある．Jasp は専用のエディタを持つので，クラスベースのオブジェクトを利用することにした．

汎用型の統計システムでは，あらかじめ組み込まれている手法は多くなる．システムの開発者，あるいはシステムを拡張しようとする利用者は，それらを使いやすいようにうまくシステム上で分類し，実装することが要求される．例えば，関数のような手法を基準としたケースでは，それらの名前が衝突しないように名前付け規則などに従って開発し，そして，そのリストをウィンドウなどを利用して利用者に見せる方法が一般的な方法であろう．ある程度の数であれば，このような問題は起こりにくいが，利用者が覚えきれないくらいの関数がシステム上に

存在するようになれば，このような問題が深刻になってくる．これに対して，オブジェクトでは，概念的に分類して手続きやデータを管理させることができる．

Jasp クラスの定義のためのフォーマットは次の通りである．

```
jaspclass NAME(SUPER) BODY
```

ここで，NAME はクラスの名前，SUPER は親クラスの名前である．親クラスの指定が不要な場合には (SUPER) の部分を省くことができる．BODY はコンストラクタ，メソッド，プライベート関数から構成される．プライベート関数は Jasp 関数のことであり，メソッドはプライベート関数と外部とのインタフェースとして利用される手続きである．クラス名と同名のメソッド名は，コンストラクタとして扱われる．Jasp クラスの中では，内部データ（属性）の型宣言をする必要はない．また，Jasp 関数をまったく修正することなく Jasp クラスの中にプライベート関数として取り込むことができるという特徴を持つ．

図 5.3に Jasp クラスの簡単な宣言例を示す．図中のクラスは線形回帰を表すクラス定義である．この図では，2 つのメソッドと 1 つのプライベート関数を紹介している．図中の... は省略を表す．beta と estimated は計算結果を保持するための属性である．メソッドの中の先頭に this を付けられた変数は，属性として扱われる．また，this で始まる手続きの呼出しは，同じクラス内のメソッドが呼ばれる．this が付かない手続き呼出しは，そのクラス内のプライベート関数が呼び出される．図中の c_bind() は，その第 1 引数の行列要素と第 2 引数の行列要素を列方向で結合させたものを返す組込み関数である．

このクラスから実体であるインスタンスを作成するには，次のコマンドを利用する．

```
reg1 = LinearRegression("datafile.dat")
```

そして，そのインスタンスの属性を参照するには，

```
reg1.estimated
```

と入力する．この場合，従属変数の予測値が返される．

図 5.4は，図 5.3を継承したプログラム例である．この例では，線形回帰モデルの診断を行うためのクラスを取り上げる．

```
jaspclass LinearRegression {
  method LinearRegression( file ) {
    data = read( file )
    y = data[,1]
    t = Vector(data.nr, 1.0)
    x = c_bind(t, data[,!1], "RIGHT")
    this.LinearRegression(y,x)
  }
  method LinearRegression(y, x) {
    this.beta = ols(y, x)
    this.estimated = x * beta
    ...
  }
  function ols(y, x) {
    coeff = (x.trans*x).inv*x.trans*y
    return coeff
  }
  ...
}
```

図 5.3 Jasp クラスによる線形回帰のためのプログラム

```
jaspclass Diagnostics(LinearRegression) {
  method Diagnostics( file ) {
    super.LinearRegression( file )
    y = this.depVar
    x = this.indepVar
    this.Diagnostics(y, x)
  }
  method Diagnostics(y, x) {
    this.hat = hat(x) // projection matrix
    this.leverage = diag( this.hat )
    this.dw = durbin_watson( this.res )
    ...
  }
  function durbin_watson( res ) {
    dw = 0.0
    for(i=2; i<=res.nr; i++){
      dw = dw + pow(res[i,1]-res[i-1,1], 2.0)
    }
    dw = dw / (res.trans * res)
    return dw
  }
  ...
}
```

図 5.4 Jasp クラスによる回帰診断のためのプログラム

1 行目で親クラスとして LinearRegression を指定している．メソッド中の super は親クラスのことを表す．引数が file のコンストラクタの中で，親クラスのコンストラクタを呼出している．プライベート関数 durbin_watson の定義は，図 5.2と全く同じである．このように Jasp クラスでは，Jasp 関数をそのままプライベート関数として利用することができる．

5.3.4 Java クラス

Jasp 言語の基本である Pnuts は，Java のクラスを変数として直接利用することができる．この機能を活かして，Jasp 言語でも Java のクラスを変数として利

用することができる．Java 言語で記述されるクラスは，高機能な能力を提供することができるので，Jasp 上で Java クラスを活用することができれば，必然的に Jasp の能力や拡張性を高めることになる．

Java クラスを Jasp 言語で利用するためには，識別子 class か関数 import を使用する．例えば，次は識別子 class を利用して Java のクラス BigDecimal を bd という変数に代入させている．

```
bd = class java.math.BigDecimal
```

これ以降，bd は Java のクラス BigDecimal を表し，そのインスタンスを生成するには，例えば，

```
pi = bd(3.14159265358979323846264338328)
```

と入力する．ここで，pi はクラス BigDecimal のインスタンスオブジェクトとなる．同じ作業を関数 import を利用した場合には，次のように記述できる．

```
import("java.math.BigDecimal")
pi = BigDecimal(3.14159265358979323846264338328)
```

関数 import では，複数のクラスを指定するために，次のようにワイルドカードを使うことができる．

```
import("cern.jet.random.*")
```

Jasp プログラムでは，インスタンス・フィールドとインスタンス・メソッドのアクセスのために，ドット演算子が使われる．また，クラス・フィールドとクラス・メソッドのアクセスのために，ダブルコロン演算子が使われる．

図 5.5に，Java クラスを導入した Jasp プログラムの例を紹介する．このプログラムは，パラメータが 4 つまでのモデルの最尤推定値を計算する関数である．非線形最小二乗アルゴリズムの部分（クラス Opti_F9）は，インターネット上で（無料で）公開されているライブラリ [32] を利用している．図中の model は，func，data，para によって計算された尤度関数を表している．size はパラメータ数であり，para はパラメータの初期値である．また，関数 mle の中では，関数を作

成するための関数 makeModel が定義されている．これが実行されると，パラメータの数に応じた無名関数が作成される．従って，変数 model にはこの無名関数が代入される．

次に，この関数 mle を利用したプログラム例を紹介する．ここでは，乱数を 50 個生成し，それを使って正規分布のパラメータの最尤推定値を求めるためのプログラムである．図 5.6に示す．このプログラム中における p[0] と p[1] は，それぞれ正規分布の平均値と標準偏差を表す.

5.4 Jasp のプログラミング環境

5.4.1 プログラミングのためのウィンドウ

Jasp を起動すると図 5.7と図 5.8に示したような 2 つのウィンドウが現れる．それぞれ CUI (Character User Interface) ウィンドウと GUI (Graphical User Interface) ウィンドウと呼ばれている．CUI ウィンドウは，プログラミングのためのウィンドウである．Jasp のプログラミング環境とは直接的にこのウィンドウのことを指す．GUI ウィンドウは，結果の表や図を出力したり，グラフィカルな操作などのためのアイコンが表示される．

CUI ウィンドウにはエディタエリアと入出力エリアの 2 つの領域がある．エディタエリアでは他のエディタを別に起動することなくプログラムの入力を行うことができる．ここでは，利用者がプログラムの一部をマウスで選択して実行に移すこともできる．この機能は，プログラムの開発時のデバッグなどで有効である．入出力エリアは，1 命令ずつの実行を与え，それに応じて Jasp からの結果が表示されるところである．ここでの Jasp からの結果はテキストベースの部分だけに限定される．

Jasp では，GUI ウィンドウ上での操作は入出力エリアで与える 1 つの命令文と 1 対 1 に対応するように実現している．従って，GUI ウィンドウでの操作はすべて入出力エリアに解析ログとして記録される．CUI ウィンドウ上のプルダウンメニューの中に解析ログをファイルに書き出す命令があるので，Jasp 上での解析手順の再現を可能にする．

```
function mle(func, data, para){
  size = para.length
  if(size == 1){
    function makeModel(data, model){
      function (x1){
        p = [x1]
        model(func, data, p)
      }
    }
  } else if(size == 2){
    function makeModel(data, model){
      function (x1, x2){
        p = [x1, x2]
        model(func, data, p)
      }
    }
  } else if(size == 3){
    function makeModel(data, model){
      function (x1, x2, x3){
        p = [x1, x2, x3]
        model(func, data, p)
      }
    }
  } else if(size == 4){
    function makeModel(data, model){
      function (x1, x2, x3, x4){
        p = [x1, x2, x3, x4]
        model(func, data, p)
      }
    }
  }
  model = makeModel(data, likelifood)

  opti = Opti_F9::define(model, size) // set a target model
  opti.initialize(para)   // set initial values
  ans = opti.optimize()
  return ans
}
```

図 5.5 最尤推定値を求める Jasp 関数

```
function nor(x, p){// normal distribution
  pi = 3.141593
  exp(-1*(x-p[0])*(x-p[0]) / (2*p[1]*p[1]))/(sqrt(2*pi)*p[1])
}
   // make random data for simulation
data = normal(50)
   // initial values of parameters
para = [0.1, 1.1]
ans = mle(nor, data, para)
```

図 5.6 関数 mle の使用例

5.4.2 GUI のためのコメント

Jasp 言語では，変数や引数のデータ型を指定する必要はない．これは，利用者，特にシステムの初心者が，プログラムを簡単に作成できるようにするためである．多くの統計システム言語でこのような仕組みは見受けられる．しかしながら，データ型を宣言することのメリットも存在する．例えば，すでに存在するプログラムの内容についてよく知らない利用者にとってプログラムの構造を理解しやすくしたり，そのプログラムを GUI 上で利用する際に，そのプログラムの呼出しに関する事前確認を行わせることが可能になる．また，データ型をたくさん定義できるシステムにおいては，何らかの方法で変数や引数のデータ型を利用者がわかるようにすることは，そのプログラムをより読みやすくする．そこで我々は，関数の引数のデータ型を定義することができるようにコメントを利用することにした．

コメント部の中で，@summary や@param，@return で始まる文があれば，それは GUI にとって特別な意味を持つ．例えば，図 5.9において，”/**”と”*/”で囲まれた箇所が Java 言語と同等にコメントとして扱われるが，先ほどの 3 つの識別子を含む行はその情報が GUI のために利用される．

@summary で始まる文は，コメントの下で定義している関数の説明をするところである．@return で始まる文は，その関数の戻り値の説明をするところで，さらに，@param で始まる文は，引数の説明をするところである．@param 行は次の

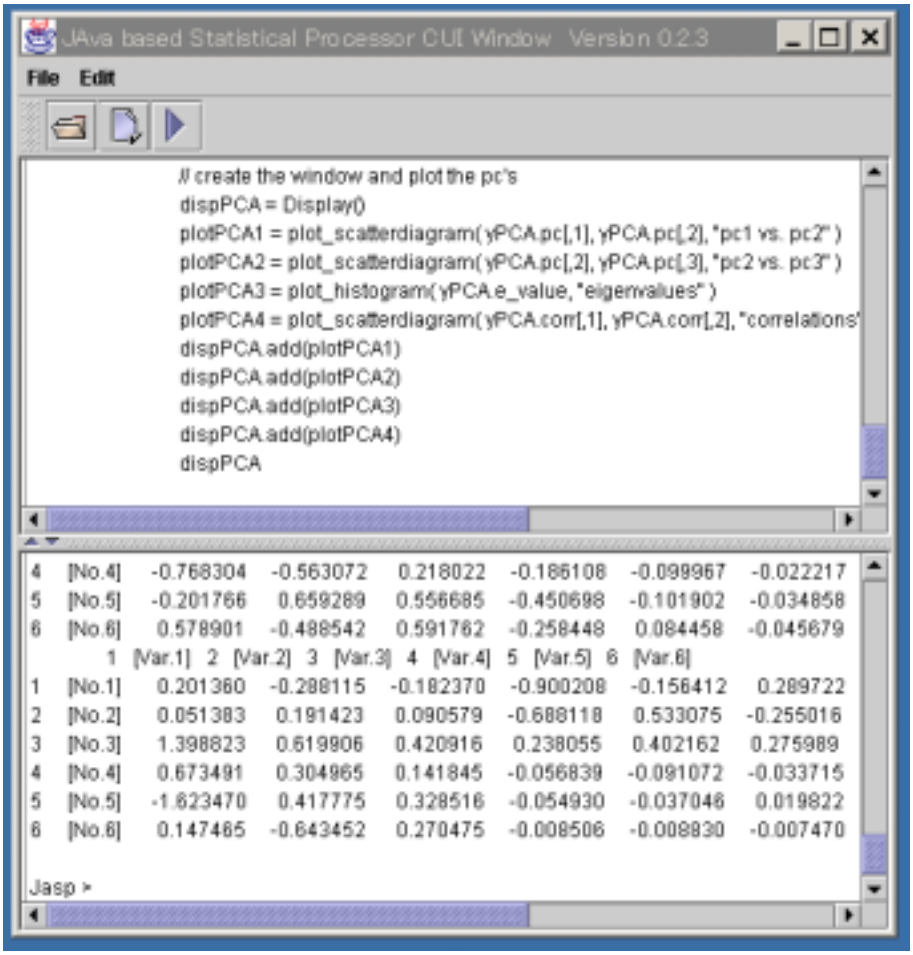

図 5.7 Jasp の CUI ウィンドウ

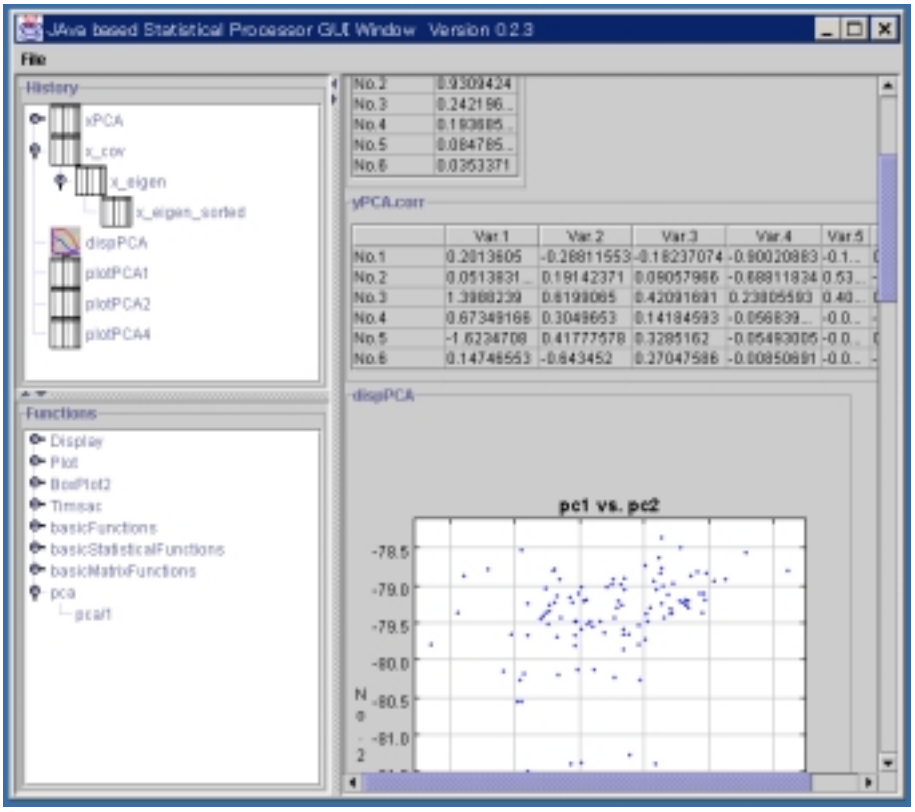

図 5.8 Jasp の GUI ウィンドウ

```
/**
 * @summary "estimate coefficients"
 * @param JaspMatrix y "dependent variable"
 * @param JaspMatrix x "independent variables"
 * @return "coefficients"
 */
function estimate_coefficients(y, x){
  beta = (x.trans * x).inv * x.trans * y
  return beta
}
```

図 5.9 GUI のためのコメント

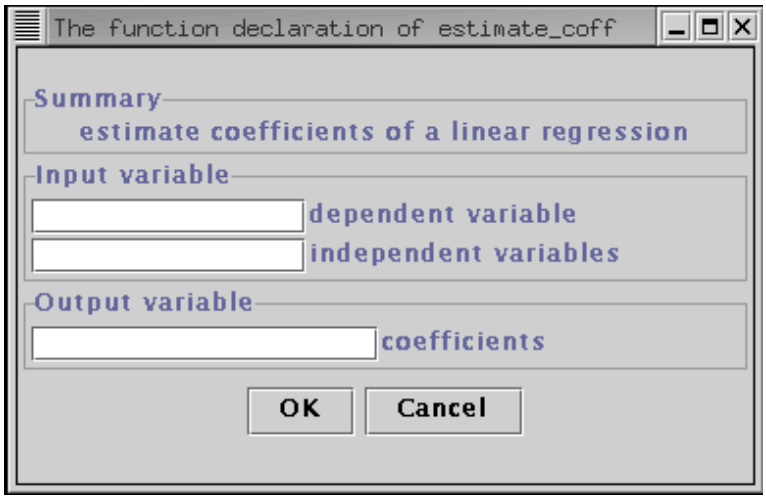

図 5.10 コメント情報を利用した Jasp 関数の呼出し

フォーマットに従う．

```
@param TYPE NAME "STRINGS"
```

ここで，TYPE は，NAME と名付けられた引数のデータ型を意味している．STRINGS は，説明である．

これらの行は，Jasp サーバ上では，一般的なコメントとして無視されるが，Jasp クライアント上の言語処理系では，この情報が活用される．利用者が Jasp を使用するとき，Jasp クライアントは 2 つのウィンドウ（Jasp の CUI ウィンドウと GUI ウィンドウ）を表示する．さらに，GUI ウィンドウは 3 つのサブウィンドウを持つ．その中の左下のウィンドウが，利用可能なコンストラクタと関数を表示するが，この時，コメント情報を記述されていないものは表示されないようにした．コメント情報のないコンストラクタや関数は，利用者の個人的な使用目的のために定義されたものであるとみなし，利用者が安易に実行に移せないことにした．GUI ウィンドウの左上のサブウィンドウでは，アイコンとそれらの木構造表現により解析の履歴が分かるようにしている．アイコンは，データや適用したモデル，解析結果を表している．利用者がアイコンをマウスで指定すると，利用可能な手続きがポップアップメニューとして表示される．このメニューの中で，コ

メント情報（その手続きの説明）を見ることができる．さらに，その一覧から目的の手続きを選択すると，ポップアップウィンドウが開き，その中で引数の情報が表示される．この仕組みは，利用者の間違った操作を未然に防ぐ役割を担う．図 5.9の例を図 5.10に示す.

5.5 Jasp の拡張例

本節では，Jasp 言語を用いた Jasp の拡張例について紹介する．ここでは，Jasp 上で XML 文書を扱うことができるようにした拡張例を中心に述べる．

5.5.1 XML 文書の取扱い

XML (eXtensible Markup Language) は，W3C (World Wide Web Consortium) が標準化を行っている次世代の構造化文書規約である．Web site 作成のための機能も兼ね備えているので，HTML (Hyper Text Markup Language) の次の記述言語と位置づけられている．マークアップ言語は，タグと呼ばれる構造により値に属性を付加する形でデータを記述する．マークアップ言語はテキストなので，利用者（人間）とコンピュータの両方にとって読みやすく，分かりやすい.

XML は今後ますますデータ記述フォーマットとして活用されていくと予想される．特に，XML を取り扱えるアプリケーションソフトがいくつか発表されてきているので，XML がデータ解析システムの中間フォーマットとして活用されていくことが明らかになってきている．Jasp で XML 文書を扱うことができれば，解析データや解析結果の共有を容易にすることができ，システムとしての汎用性を高めることになる.

XML でどのようなタグが使えるのか，あるいは，どのような属性を与えられるかは DTD (Document Type Definition) と呼ばれる定義ファイルで規定される．ただし，XML 文書には必ず DTD が必要ということではなく，構造のチェックの必要がなければ DTD を省略することができる．DTD の無い XML 文書を well-formed XML 文書と呼ぶ.

ただし，XML の自由度の高さとデータの種類の多さから，データ記述の構造を規定することが難しく，DTD に関してさまざまな提案や勧告が行われている.

例えば，解析データの記述に関与しているものでは，DDI[27] や DandD[43] などが挙げられる．Jasp では，XML に関する Jasp 言語の API を利用することができるので，比較的容易に各種の DTD に従った XML 文書を Jasp に読み込ませることができる．

具体的には，Jasp 上で XML 文書を扱うために，DOM (Document Object Model) と呼ばれるプログラミングインタフェースを使用する．特に，DOM 対応の Java 言語用 API (Application Programming Interface) が Web 上で無料で公開されている [2] ので，Jasp ではこれを利用する．

Jasp 言語は，Java クラスを変数としてそのまま扱うことができるので，この API を使用することで XML 文書を表す XML インスタンスを Jasp プログラミング環境で作成することができる．ただし，API が提供する機能は多種多様であり，そのままの形で Jasp の利用者に提供しても使いこなすことが困難である．そこで，Java 言語のレベルでこれらの API のデータ解析上必要な機能だけを抽出し，使いやすくするための Wrapper クラスを用意した．これを Jasp 言語上で利用することで，利用者が XML を容易に扱うことを可能とする．

次に，Jasp 上で XML 文書を取り込む例を紹介する．DTD として DandD を採用している．例えば，DTD に従って記述されたデータ `acidj.xml` を Jasp 上に読み込ませるためには，次のコマンドを入力する．

```
acid = JaspSimpleXML("acidj.xml")
acid_tree = acid.printTree()
acid_tree
acid_matrix = importDataVector(acid, 3)
acid_matrix
```

1 行目では，XML 文書から Jasp 言語の変数を作成している．Jasp 関数 `JaspSimpleXML()` の第 1 引数には，XML 文書ファイル名を渡す．`acid` は DOM オブジェクトを表す変数である．2 行目は，XML 文書の構造を文字列で返すメソッド `printTree()` を呼び出している．`acid_tree` は String クラスのインスタンスである．3 行目は，その文字列を GUI ウィンドウ上に表示させるためのコマンドである．Jasp は，変数だけの入力で，その内容を GUI ウィンドウに表示する．4 行目の `importDataVector()`

```
function importDataVector(dom, colNum){
  dvec = dom.getDataVector(1)
  len = dvec.length
  ans = JaspMatrix(len, colNum)
  for(i=1; i<=len; i++){
      ans[i,1]=dvec[i-1] }
  label = dom.getDataVectorAttr(1, "Id")
  ans.setColLabel(label, 1)
  for(j=2; j<=colNum; j++){
      dvec = dom.getDataVector(j)
    for(i=1; i<=len; i++){
      ans[i,j]=dvec[i-1] }
    label = dom.getDataVectorAttr(j-1,"Id")
    ans.setColLabel(label, j)
  }
  return ans
}
```

図 5.11 関数 importDataVector

は Jasp 関数である．この関数の内容は図 5.11である．この関数は，第 1 引数には XML インスタンス名を，第 2 引数にはインスタンス内にあるタグ `DataVector` の内容を何個取ってくるのかを指示している．この戻り値は，JaspMatrix のインスタンスとなる．5 行目は，4 行目で求めた行列の内容を GUI 上に表示させている．

関数 `importDataVector()` は，Jasp 言語利用者が記述できるレベルのものである．図 5.11の 2 行目と 9 行目の `getDataVector()` は Java レベルで実装したものである．具体的には，`JaspDandD` クラスのメソッドである．6 行目と 11 行目の `getDataVectorAttr()` は，`DataVector` タグの属性を参照するためのメソッドである．この場合，属性 ID の内容を行列の列ラベルに設定している．

JaspMatrix のインスタンスにメッセージ `writeXML()` を送れば well-formed な XML 文書を作成できる．`writeXML()` には引数として書き出すファイル名を指定する．例えば，次のような要素を持つ JaspMatrix があるとする．

```
<JaspMatrix>
    <observation>
            <x>1</x>
            <y>2</y>
            <z>3</z>
    </observation>
    <observation>
            <x>4</x>
            <y>5</y>
            <z>6</z>
    </observation>
</JaspMatrix>
```

図 5.12 exMatrix.xml の内容

```
    x  y  z
No1 1  2  3
No2 4  5  6
```

インスタンス名が test であれば,

```
test.writeXML("exMatrix.xml")
```

と入力すれば XML 文書"exMatrix.xml"が作成される．図 5.12に exMatrix.xml の中身を記す.

5.5.2 その他の拡張例

文部科学省統計数理研究所で開発された TIMSAC (TIMe Series And Control)[1] は，時系列データの解析，予測，制御のためのプログラムパッケージである．一変量 AR モデル，一変量 ARMA モデル，多変量 AR モデルの当てはめ，一 (多) 変量パワースペクトル，周波数応答，ノイズ寄与率などを計算できる Fortran で記述されたライブラリのことである．現在，その大部分が C 言語に移植され，Microsoft の Windows 上で稼動するようになっている．Jasp では，この TIMSAC のプログラムを利用できるようにしている．具体的には，JNI 機構を使って TIMSAC の

C プログラムを Java クラスから呼び出せるようにして，その Java クラスを Jasp 言語を使って直接呼び出している．Java の JNI の部分をプログラミングしなければならないが，TIMSAC をすべて Jasp 言語に移植することと比較すればかなり少ない作業量で実装することができた．TIMSAC を利用して時系列データの標本自己相関関数を計算する Jasp クラス `Autocorrelation` の一部を図 5.13 に示す．

現在，パーソナルコンピュータでデータを整理するために最も多く利用されているソフトウェアは，Microsoft の Excel であろう．Excel は，Microsoft Windows(95/Me/2000) と Macintosh 上で稼動する表計算ソフトである．その組込み関数を利用すれば，基本的な統計処理や図の作成もできる．従って，現在の多くの小規模なデータが Excel のデータ形式で保存され，やりとりされることがよく見受けられる．このような現状から，商用の統計システムの中には，標準で Excel のデータ形式をサポートしているものがある．

Jasp でも Excel のデータのインポート，及びエクスポートができる機能を実装している．この実装のために，Jasp では JCom[34] を採用した．JCom は，Java から Windows の COM (Component Object Model) を利用することができるブリッジである．おおまかには，COM にアクセスするための Java プログラムと考えることもできる．COM は，Microsoft が規定した Windows 上におけるオブジェクトの呼出し方法の規約であり，現在のほとんどの Windows アプリケーション (Microsoft Office など) は，この COM に基づいて開発されている．JCom は COM にアクセスするための Java のクラスをいくつか提供しているので，それを直接使用して，Jasp 上で Excel のデータ形式のデータを Jasp 上にインポート及びエクスポートできるようにした．

実装の部分の具体的なプログラムの紹介は省略するが，例えば，Excel のデータファイル `sample.xls` を Jasp に読み込む場合には，

```
x = readExcel("sample.xls", 1, 114, 1, 1)
```

と入力する．これにより，Excel のデータが Jasp 上で行列データ `x` として管理することができる．ここで，Jasp 関数 `readExcel` の第１引数にはファイル名，第２と第３引数には指定する行のはじめと終わり，第４と第５引数には指定する列

```
jaspclass Autocorrelation(Timsac) {
  /**
   * @summary "Calculation sample autocorrilation from data."
   * @param JaspMatrix data "Source data of JaspMatrix class"
   * @return cov "Instance object of Autocorrelation class"
   */
   method Autocorrelation(data) {
     this.data = data
     lag = (int) (2.0 * sqrt(data.nr) - 0.5)
     autcor = TimsacAutcor(data[,1], data.nr, lag + 1)
     this.timsacAutcorResult = autcor

     this.autcor = JaspMatrix(autcor.cn).trans
     this.autcor.setColLabel(["autcor"])

     AutocorrelationResult = this.autcor
     this.autcov = JaspMatrix(autcor.cxx).trans
     this.autcov.setColLabel(["autcov"])
     AutocorrelationGraph = Display(2, 1)
     AutocorrelationPlot1 = plot_timeseries(this.data, "data")
     AutocorrelationPlot2 =
              plot_correlation(this.autcor, "autocorrelation")
     AutocorrelationGraph.add(AutocorrelationPlot1)
     AutocorrelationGraph.add(AutocorrelationPlot2)
     AutocorrelationGraph
   }
   ...
```

図 5.13 TIMSAC を利用する Jasp クラス（一部）

のはじめと終わりを与える．

逆に，JaspMatrix 型の変数 x を Excel のシートにエクスポートするには，

```
saveExcel(x, "sample.xls", 1, 3, true)
```

と命令する．ここで，第１引数は JaspMatrix 型の変数，第２引数は，保存するファイル名，第３と第４引数はデータをエクスポートするシートの開始セル（つまり，JaspMatrix 型の変数の第１行第１列の場所）の行番地と列番地，第５引数はラベルを貼り付けるかどうかのフラグを意味する．

JCom を使用すれば，Excel のデータ形式がどのようになっているのかを知らなくてもデータのやりとりのための機能を Jasp に実装させることができた．実装に必要な知識は，基本的に JCom の仕様の理解だけである．このように，Jasp 関数が JCom のような Java のプログラムを直接参照できるので，実装のための作業量を減らし，Jasp の拡張性を高めている．

この他にも，Jasp 上でいくつかの拡張作業が行われている．ひとつは，統計システム XploRe の関数（”Quantlets”と呼ばれている）を Jasp 関数に翻訳するためのトランスレータの開発 [14] である．これにより，XploRe に備わっている数多くの統計解析用の関数を Jasp 上で実行できるようになる．また，韓国の成均館大学の Huh 教授が開発したデータビジュアライジングツール Davis[11] を Jasp 上で利用できるようにするプロジェクトを Huh 教授の研究グループと Jasp の研究グループが共同で進めている．

5.6 Jasp 言語の評価

本節では，Jasp 言語の評価を代表的な汎用統計解析システムのプログラミング言語と比較することで行う．ここで取り上げる代表的な汎用統計システムは，S, LISP-STAT，XploRe の３つである．まず前半で，これらと Jasp のシステムレベルでの機能の違いを明確にし，後半で，プログラミング能力の比較を行う．Jasp を含めた主要な汎用統計解析システムの機能の一覧が表 5.2である．

表中の項目 OOP とは，オブジェクト指向プログラミングのことである．S と LISP-STAT の OOP と Jasp の OOP は異なる部分がある．前者ふたつのシステ

表 5.2 主要な汎用統計解析システムの機能一覧

項目	S	LISP-STAT	XploRe	Jasp
開発言語	C++	LISP	C++	Java
実行環境	UNIX,Win	UNIX,Win,Mac	Win	JRE[*1]
専用言語の形態	手続き型	関数型	手続き型	手続き型
OOP	○	○	×	○
ライブラリ機能	○	×	○	○
他言語インタフェース	○	○	○	○
組み込み関数の種類	多	少	多	少 (開発中)
ネットワーク対応	×	×	○	○
ソースコードの公開	×	○	×	○

[*1]: Java Runtime Environment

ムでは，プロトタイプベースのオブジェクトを取り扱うのに対し，Jasp の方は，クラスベースのオブジェクトを取り扱っている．プロトタイプベースのオブジェクトは，クラスベースのものと比べると，対話型のプログラミング環境では取り扱いが簡単であるということと，クラスという型を強く意識せずにプログラミングができるというメリットがある．しかしながら，オブジェクト定義のためのプログラムが断片的になってしまうために，オブジェクトの全体像の認識が難しくなったり，そのプログラムの再利用性が低くなったりする．また，C++などの汎用プログラミング言語ではクラスベースの方が広く普及しているという現実がある．そこで，Jasp では専用のプログラミング環境（CUI ウィンドウ上のエディタ）をうまく活用することで，対話型プログラミング環境におけるクラスベースのオブジェクトのデメリットを解消させ，再現性と再利用性の高い OOP を提供している．XploRe のように OOP ができないシステムでは，すべての処理を関数で実装しなければならない．基本的な処理の数が少なければ全てを関数で実装する方が使いやすいと思われるが，システムに多くの機能が組み込まれるようになるとオブジェクトで実装した方がわかりやすい場合がある．また，システムの拡張性を高めるのであれば，OOP は必要な機能のひとつであると思われる．

表5.2中のライブラリ機能とは，各プログラミング言語の基本単位である関数をまとめるための仕組みのことである．システムの起動時にすべての組み込み関数

をメモリ上に展開しておくと，処理速度の低下や実行に必要なメモリの増大をまねく．これを防止するために，開発者や利用者が，ライブラリ機能を使っていくつかの関数をまとめ，必要な時にそれらをシステム上に呼び出すのである．Jaspでは，S や XploRe と同様に関数をライブラリとしてまとめる機能を持つが，この他に Jasp クラスという関数をまとめる枠組みを持っている．S や LISP-STATも，オブジェクトを定義することはできるが，5.3.3節で述べたように，Jasp では，関数（Jasp 関数）の定義プログラムをまったく変更することなくクラス内に移行させることが可能である．従って，Jasp は，他のシステムには無い一種のライブラリ機能をもうひとつ別に持っていると考えることができる．

S や LISP-STAT に限らず，ほとんどの統計解析システムは他言語インタフェースを持っている．これは，各実行環境でコンパイルされた C（C++）と Fortranのオブジェクトモジュール，あるいは実行モジュールを呼び出せる機能のことである．これらは，引数として情報を与え，その結果を統計解析システム側に返すことができる．このとき，呼び出される C や Fortran のプログラムは，独自の規則に従って作成され，コンパイルされていなければならない．また，このためのコンパイラは，どのコンパイラでも良いということでは無いし，決して簡単に入手できるとは限らない．これに対し，Jasp の他言語インタフェースは，5.3.4節で述べたように，Java のクラスをそのまま Jasp の変数として取り込むことができる．Java に標準の JNI 機能を用いれば C や Fortran のプログラムを Java の環境に取り込むことも可能である．これを実現するために利用者が面倒な手順を踏む必要もないし，Java のコンパイラ（JDK）もインターネットから無料で入手することができる．取り込んだ Java のクラスは Jasp のコンポーネントとして働く．Jasp の利用者は，これらのコンポーネントを Jasp 言語によって制御することができるので，システムに依存する低レベルな拡張や制御までも比較的簡単に実現することができる．この具体例は，5.5節でいくつか紹介されている．このような仕組みは，スクリプティング [23] に該当する．スクリプティングとは，システム記述言語で作成されたモジュール（コンポーネント）をうまく結合させることで（参考文献 [23] では“ gluing ”という言葉で説明されている）システムを開発する方法論のことである．S や LISP-STAT の他言語インタフェースでは，このよう

に他言語モジュールをシステム内で取り扱うことはできない．従って，Jasp は，スクリプティングを導入した最初の統計解析システムである．Jasp 言語は，この gluing をするための統計解析に特化したプログラミング言語であると位置づけられる．

さらに，ここでは，表 5.2で取り上げた汎用統計解析システムをそれぞれ比較しながら Jasp の位置づけを明確にしてみる．長い歴史を持つ S は，一般に馴染みやすい手続き型言語と多くの組み込み関数によって沢山の利用者を獲得している．オブジェクト指向プログラミングや他言語インタフェースがあるので，利用者は，比較的高度なプログラミングが可能である．ただし，開発言語とプログラミング言語の言語形態が異なり，ソースコードが公開されていないのでシステムレベルの拡張や変更がとても困難である．LISP-STAT も長い歴史を持つが，これは，データ解析のために使用されるというよりも，統計解析システムに求められる新たな機能を追加するという実験環境として利用されていることの方が多い．データ解析として多用されない理由は，多くの利用者が，その専用言語である LISP を使って統計解析の処理を記述することに馴染めないからである．また，組み込み関数が比較的少ないということも影響している．しかしながら，LISP-STAT が LISP をそのプログラミング言語と開発言語として選択しているので，他のシステムよりもシステム内部の透視性が高く，比較的簡単にシステムそのものを拡張することができる．従って，データ解析者よりも計算機統計学の研究者に広く受け入れられている．XploRe は，比較的新しい汎用統計解析システムである．プログラミング能力を高めて LISP-STAT のような利用者レベルの高い拡張性を目指すというよりも，あらかじめシステムに多くの関数を用意してそれに基づいて解析環境を提供し，必要であれば利用者に対してプログラミング環境を使用してもらう，という位置づけをとっている．近年に設計され，洗練されたユーザインタフェースを持つ XploRe は，統計解析に専念した次世代の統計解析システムと言うことができるであろう．従って，XploRe のプログラミング言語は，システムを制御できる自由度は低く抑えられているものの，統計解析の数値計算アルゴリズムを簡単に記述することができるように設計されている．Jasp は，システム全体の能力から見れば，XploRe と同じ新世代の統計解析システムに属するもの

である．ただし，XploRe とは異なり，システムの低レベルの拡張を可能にしているので，LISP-STAT のような実験的な利用にも対応しうるものであると考えられる．これらのことから判断すると，Jasp は，S のようなプログラミング能力を持ち，LISP-STAT のようなシステムレベルの拡張を可能にした新世代の統計解析システムと位置づけられる．

一般的に，プログラミング言語の評価は難しい．評価基準がたくさんあり，これらが判断する人の主観によるところが強いからである．言語の評価でよく用いられる基準としては，作りやすさや記述能力などが挙げられる．あるいは，たくさんの人が使っているからその言語は優れているという評価が下されることもある．

プログラミング言語の作りやすさについては，判断する人の経験に基づくところが多く，また，プログラミングの対象によって大きく異なるので客観的な評価を下すことは難しい．一般的に，高水準言語と言われている言語スタイルが我々にとって馴染みやすく作りやすい言語であると言われている．Jasp 言語だけに限らず，多くの統計解析システムのプログラミング言語は，この手続き型の高水準言語を採用している．文法的な違いはいくつか存在するが，それぞれのプログラミング言語は統計解析の中心的役割である数値計算を表現することは可能である．本論文では，個々の言語の文法的な比較をすることはしないが，統計解析システムの言語として最低限必要な文法的能力としては，多次元データに対応すること，データ型を意識することなく計算アルゴリズムを表現できること，対話型のユーザインタフェースに対応していることが挙げられるであろう．Jasp 言語は，これらの能力を兼ね備えている．

統計解析システムのプログラミング言語に関する記述能力は，数値計算の信頼性や限界に関すること，統計解析システムが持つ機能をどれほど制御できるのかということなどで評価される．

数値計算の信頼性や限界の具体例としては，データの有効桁数や同時に扱えるデータ数（多次元データの最大次数）などが挙げられる．Jasp 言語では，データの状態に応じて適切に Java の基本データ型を割り当てる機能がある．特に，5.3.4 節で紹介した BigDecimal を使えば，保存容量として最大 32 バイトまでの数値を扱うことも可能である．同時に扱えるデータ容量は，S の場合は，データを内部

でファイルとして扱っているのでデータ容量の制約は外部記憶装置の容量に比例する．Jasp の場合は，JRE の能力に直結している．S よりも処理速度の面で高速であるが，容量は S よりも低く抑えられる．

もうひとつの統計解析システムが持つ機能をどれほど制御できるのかという点であるが，先述の通り，Jasp は完全にシステムを制御できる能力を持っている．Jasp 言語のスクリプティング能力により Java で開発されている Jasp システムの主要コンポーネントを制御することができ，さらに，プログラムの規模に応じて関数からクラスにシームレスにプログラムを移行することができるプログラミングスタイルを持っている．このような記述スタイルを持った統計解析システムのプログラミング言語はこれまでのところ他には存在していない．

5.7 本章のまとめ

本章では，統計システム Jasp の専用言語の設計と特徴について述べた．さらに，Jasp 言語が Jasp の拡張性を高めている事例をいくつか簡単に紹介した．情報技術の躍進と伴に，システムとして要求される能力は大きくなっている．統計システムについても例外ではなく，より効果的に統計解析を支援する役割を担い続け，今後も発展していくであろう．このためには，これまでに蓄積されてきた既存の有益なプログラム資産を活かしながら統計システムをより良く拡張していていくことが求められる．従って，既存の技術と新しい技術をひとつのシステム上で整合性や一貫性を保ちながらうまくインテグレートすることが必要になる．Jasp 言語は，その枠組みを具体的に提供するひとつの実例である．

6. 結論

本論文は，統計解析システムに必要なプログラミング言語をどのように設計すればよいのか（設計方法），またどのように実装すればよいのか（実装方法），さらに開発した統計システムにおけるプログラミング言語の効果について言及したものである．特に，

【1】知的支援機構を持つ統計解析システム上でプログラミング言語を開発すること

【2】汎用型統計解析システムの拡張性を高めるためのプログラミング言語を開発すること

の目標を掲げ，これらを達成するために，RASS 言語，RASS2 言語，Jasp 言語という３種類の統計解析システム用のプログラミング言語を開発した．目標【１】に対応するものが RASS 言語と RASS2 言語，目標【２】に対応するものが Jasp 言語である．開発したこれらの言語の主たる新規性と課題を表 6.1に示す．さらに，統計解析システムの柔軟性や解析支援機構の実装に関連したソフトウェアモジュールの自発的振舞いについて，とりわけ，その対話型プログラミング環境における効果を紹介した．

２章で述べた RASS 言語は，重回帰分析に特化した“ 統計解析エキスパートシステム ”のひとつである RASS のプログラム記述能力を高めるために開発されたものである．RASS 言語は，一般に馴染みのある手続き型言語で，簡単に行列演算を表現することができ，システムに悪影響を及ぼすことなく実現したい手続きを実装できるという特徴を持つ．多くの統計解析エキスパートシステムは，知的支援を実装するために記号処理を得意とする言語（例えば，Prolog や Lisp）が利用されている．このような言語は，一般には広く利用されていないし，統計計算のための手法を分かりやすく表現することが得意ではない．従って，このようなシステムの利用者がシステムに組み込まれていない手法を追加することは事実上不可能であった．RASS 言語を実装する前の RASS にもこのような状況が見受けられた．これらのシステムの必要性は十分に認められているが，せめて RASS 言語で実装したようなプログラミング能力を持たせなければ，一般に広く利用され

表 6.1 開発した３つの言語の目的と課題

開発言語	新規性と課題	
RASS 言語	**新規性**	統計解析エキスパートシステムの高水準言語の開発
	課　題	支援機構のためのルールの記述ができない
RASS2 言語	**新規性**	知識（ルール）の記述を可能にした拡張 RASS 言語
	課　題	利用者レベルでのシステムの拡張性が低い
Jasp 言語	**新規性**	汎用統計システムに利用者レベルの高い拡張性を提供する言語
	課　題	解析支援のための知識の記述

ることは難しい．RASS 言語のようなプログラミング環境は，知的統計解析システムに求められる最低限の機能のひとつと考えることができる．ただし，RASS 言語は，RASS の知的支援のためのルールを記述する能力を持たない．RASS に知的支援のための知識を追加・修正するためには，RASS の開発方法を十分に理解しなければならないという課題が残されている．

３章で開発した RASS2 言語は，RASS の利用者が解析支援のための知識を記述しにくいという問題を解決するために開発されたものである．RASS2 言語は，RASS 言語の成果を取り入れて設計されているが，RASS 言語が RASS システムと利用者の中間に位置するように実装されているのに対し，RASS2 言語は RASS2 システムのプラットフォームとして実装されている．

利用者が知識を簡単に RASS2 上で表現できるようにするために，システムの基本構成要素をルールを内包した特殊なオブジェクトとした．ルールをオブジェクト内で表現したことで，ルール間の干渉を低く抑えることや，ルールの拡張や追加をシステムの安定性を維持させたまま容易に実現できるようになった．前者の理由は，ルールの有効範囲がオブジェクト内に限定されるからであり，後者の理由は，オブジェクト指向の継承機構をルールにも適用できるからである．RASS と RASS2 により，知的支援機構を持つ統計解析システムにおいてオブジェクト指向の枠組みがうまく機能することが明らかにできた．ただし，RASS2 言語のようにオブジェクトを基本構成要素にすると一時的な使用のためのプログラムで

さえもオブジェクトの枠組みの中に記述しなければならないし，その呼出しも単純ではない．また，独自のオブジェクト指向プログラミングの枠組みに強くとらわれるためにRASS2の利用者レベルでの拡張は簡単ではない．

５章で開発したJasp言語は，汎用統計解析システムJaspの持つデータ解析能力を制御でき，さらに，データ解析を取り巻く新しい技術を利用者がJaspに追加できるようにするために開発された．Jasp言語の文法の設計には，RASS言語の経験と広く使われているいくつかの統計解析システムのプログラミング言語を参考にした．Jasp言語の特徴は，利用者がJasp上で実装したいプログラムを段階的に実装できるというプログラミングスタイルにある．利用者は，実行確認ができる小規模な手続きを表現した関数（Jasp関数）を自然にオブジェクト指向プログラミングの枠組み（Jaspクラス）に拡張することができる．利用者は，Jasp関数だけでもプログラミングができるし，最初からJaspクラスを設計することからはじめることもできる．作成するプログラムの目的や作成者の能力に応じてプログラミングができる環境をJasp言語は提供することができた．また，Jasp言語は，Javaクラスをそのまま変数として利用することができるので，Jaspの拡張性を高くすることができる．実際に，これまでの多くの統計解析システムでは実現が困難であった機能を比較的容易にJaspで実現させることができた．

たとえ優れた機能を持つ統計解析システムができたとしても，その機能を利用者が十分に活かせなければ意味がない．利用者には統計解析の知識だけでなく，システムの操作方法の習得が求められているのが現状である．解析結果を得ることが第一目的の利用者に対してこのような負担は決して歓迎されることではない．これは，近年の汎用統計解析システムの能力の向上に伴ってますます深刻な問題となってきている．このような問題に対処するために，RASSのように解析状況や解析の履歴に応じて利用すべき手法を提示したり，利用者の誤操作を防ぐことができるような機能を汎用統計解析システムにも何らかの方法で実装するべきであろう．Jaspには，まだこのような支援機構は実装されていない．将来的には，４章の成果とRASSの経験に基づいてJaspにこのような機能を実装するつもりである．

４章で紹介したオブジェクトの自発性に関する試みは，その成果を次世代の統計解析システムのプラットフォームとして利用することを念頭に置いて行われたものである．RASS2 言語においても知的支援のためにモジュールの自発的振舞いが効果的に働くことが分かったが，これはより汎用的にソフトウェアの自発性の効果を探求したものである．RASS2 言語のルールは，適用可能な手続きを利用者に示すところに自発性を利用しているが，アクティブルールは，手続きを実行に移すところにルールが使われている点が異なる．アクティブルールの方が利用者からより独立した状態で自発性を表現することができる．

ソフトウェアがルールのような何らかの条件に従って自発的に振舞うことができると，その振舞いを記述する開発者やコンピュータの負担は増すが，ソフトウェアが利用者の代わりを担うことができるので，利用者の負担は軽減される．プログラミング環境においても，気づかないことを知らせてくれたり，プログラマの代理として何らかの作業を勝手に処理させることができる．４章では，まだこのような実験環境を整備した程度であるが，先述の通り，Jasp にこの枠組みを移行し，自発性の効果を確かめたい．

ソフトウェア開発では，(1) 目的や要求の分析，(2) 仕様の設計，(3) コーディング（実装），(4) テストや保守，などの段階を踏むことが一般的である．統計解析システムのプログラミング言語を開発する際にもこの段階を踏むことが大切なポイントである．

(1) については，“ コンピュータ上でデータを統計的に解析する ”という目的は明らかであるが，それが特殊用途向けのものなのか汎用的なものなのか，あるいは，教育向けのものなのか実務指向のものなのか，などをまず明確にすることが望ましい．(2) の設計は，開発環境（OS とか既存のシステムのこと）と言語との関係を明らかにする内部設計と，言語と利用者との関係を明らかにする外部設計に分けることができる．内部設計では，開発環境の限界や特徴の評価をしたり，解析データの管理方法を検討したりすることが要求される．外部設計は，文法の規定やプログラミング環境を設計することであるが，利用者にとって使いやすく，分かりやすいことが求められるので，設計者の過去の経験や主観的な判断に基づいてしまうところである．一般的には，プログラムの可読性とその記述能力，入

力の効率と実行の効率は相反する事項であると言われているので，これらを常に意識して設計することが重要である．実装については，既存のプログラミング言語をうまく活用することが大切であろう．プラットフォームを最初から開発することは大変な労力が要求される．Jasp 言語の開発が，RASS2 言語のそれと比べて予想以上に短時間で済んだことは決して無視できない本研究からの経験である．

謝辞

本論文をまとめるにあたり，奈良先端科学技術大学院大学情報科学研究科の渡邉勝正教授には，終始懇切なご指導とご鞭撻を賜りました．ここに深甚なる感謝の意を表します．

また，本研究で開発した言語処理系の開発につきまして，文部科学省統計数理研究所統計計算開発センターの中野純司教授には，数多くの有益なご指導を賜りました．深甚なる感謝の意を表します．

さらに，奈良先端科学技術大学院大学情報科学研究科の杉本謙二教授，関浩之教授，木村晋二助教授には，ご多忙にも関わらず審査委員をお引き受け下さり，数々のご指導とご教示をいただきました．深く感謝致します．

徳島文理大学工学部助手の山本由和先生，総合研究大学院大学数物科学研究科の藤原丈史氏，徳島県工業技術センターの岡田雅史氏には，本論文に関するシステム開発で大変お世話になりました．厚く御礼申し上げます．

徳島文理大学名誉学長の添田喬先生，徳島大学名誉教授の田上重美先生，徳島文理大学工学部教授の小山健先生，徳島文理大学工学部教授の田渕敏明先生，徳島文理大学家政学部教授の古本奈々代先生，呉工業高等専門学校教授の岡中正三先生，徳島文理大学助手の森本滋郎先生には平素より私の研究活動に対して適切な助言をしていただきました．心から感謝致します．

本論文をまとめるための環境を提供していただいた奈良先端科学技術大学院大学情報科学研究科情報システム学専攻言語設計学講座の皆様，徳島文理大学工学部の関係者の皆様に感謝を申し上げます．

この他にも多くの方の支援により，この論文をまとめることができました．この場をお借りして心より感謝申し上げます．

参考文献

[1] Akaike, H., Nakagawa, T.: *Statistical Analysis and Control of Dynamic Systems*, Kluwer Academic Publishers (1988).

[2] Apache XML Project: Xerces, `http://xml.apache.org/` (2001).

[3] Becker, R. A., Chambers, J. M., Wilks, A. R.: *The New S Language: A Programming Environment for Data Analysis and Graphics*, Wadsworth (1988) [渋谷政昭, 柴田里程 訳: S 言語–データ解析とグラフィックスのためのプログラミング環境 I, II, 共立出版 (1991)].

[4] Belsley, D., Kuh, E., Welsch, R.: *Regression Diagnostics*, Wiley (1980).

[5] Bigus, P. J., Bigus, J.: *Constructing Intelligent Agents with Java*, Wiley (1998).

[6] Chambers, J. M., Hastie, T. J.: *Statistical Models in S*, Wadsworth (1992) [柴田里程 訳: S と統計モデル, 共立出版 (1994)].

[7] Chambers, J. M.: *Programming with data : a guide to the S language*, Springer (1998).

[8] Gale, W. A. (ed.): *Artificial Intelligence & Statistics*, Addison-Wesley (1986).

[9] Hand, D. J. (ed.): *Artificial Intelligence Frontiers in Statistics : AI and Statistics III*, Chapman & Hall (1993).

[10] Härdle, W., Klinke, S., Müller, M.: *XploRe Learning Guide*, Springer (2000) [垂水共之 監訳: 統計解析環境 XploRe ラーニングガイド, 共立出版 (2001)].

[11] Huh, M. Y., Song, K. R., Yamamoto, Y., Nakano, J., Fujiwara, T., Kobayashi, I.: Davis-Jasp : A Data Mining Slution by Combininig Two

Separate Java-Based Systems, *Bulletin of the International Statistical Institute 53rd session Contributed Papers Book 3*, pp.197–198 (2001).

[12] Keene, E. S.: *Object-Oriented Programming in COMMON LISP*, Symbolics (1989).

[13] Kobayashi, I., Nakano, J.: A Regression Analysis Supporting System based on Agents, *Proceedings of The Ninth Korea and Japan Joint Conference of Statistics*, pp.51–55 (1997).

[14] Kobayashi, I., Fujiwara, T., Yamamoto, Y., Nakano, J.: The Language and the Extendibility of the Statistical System Jasp, *ISM SYMPOSIUM - Statistical software in the Internet age* -, pp.65–73 (2001).

[15] Kobayashi, I., Fujiwara, T., Yamamoto, Y., Nakano, J.: Extendibilities of a Java based statistical system, *Proceedings in International Conference on New Trends in Computational Statistics with Biomedical Applications*, pp.109–115 (2001).

[16] Liang, S.: *The Java Native Interface: Programmer's Guide and Specification*, Addison Wesley (1999).

[17] Median Marketing Group Inc.: BMDP, `http://www.ppgsoft.com/BMDP/` (2001).

[18] Meyer, B.: *Object-oriented Software Construction*, Prentice Hall (1989) [二木厚吉 監訳, 酒匂寛, 酒匂順子 訳: オブジェクト指向入門, アスキー (1990)].

[19] Nakano, J., Yamamoto, Y., Kobayashi, I.: Object Oriented Knowledge Representation for Multiple Regression Analysis, *Short Communications in Computational Statistics*, pp.156–157 (1994).

[20] Nakano, J., Fujiwara, T., Yamamoto, Y., Kobayashi, I.: A Statistical package based on Pnuts, *COMPSTAT2000*, pp.361–366, Physica-Verlag (2000).

[21] Norman, W. P., Diaz, O.: Active Database Systems, *ACM Computing Surveys*, Vol.31, No.1, pp.63–103 (1999).

[22] Omega Project: `http://www.omegahat.org` (2001).

[23] Ousterhout, J. K.: Scripting: Higher Level Programming for the 21st Century, *IEEE Computer*, Vol.31, No.3, pp.23–30 (1998).

[24] Pinson, L., Wiener, R.: *An Introduction to Object-Oriented Programming and Smalltalk*, Addison-Wesley (1988) [羽生田栄一 監訳: Smalltalk：オブジェクト指向プログラミング, トッパン (1990)].

[25] Ryan, T.: *Modern Regression Methods*, Wiley (1997).

[26] SAS Institute Inc.: Statistical Analysis System, `http://www.sas.com/` (2001).

[27] Society Science Community: Data Documentation Initiative, `http://www.icpsr.umich.edu/DDI/` (2001).

[28] SPSS Inc.: *SPSS Base 10.0 Applications Guide*, Prentice Hall (1999). (`http://www.spss.com/`)

[29] Sully, P.: *Modeling the World with Objects*, Prentice Hall (1995) [本位田真一 監訳: オブジェクト指向モデリング, 日経 BP (1995)].

[30] Swedish Institute of Computer Science : *SICStus Prolog*, `http://www.sics.se/sicstus/`, (1990).

[31] Tierney, L.: *LISP-STAT*, John Wiley & Sons (1990) [垂水共之, 鎌倉稔成, 林篤裕, 奥村晴彦, 水田正弘 訳: LISP-STAT, 共立出版 (1996)].

[32] Verrill, S.: *Package-optimization*, Forest Products Laboratory, `http://www1.fpl.fs.fed.us/` (1998).

[33] Wall, L., Christiansen, T., Schwartz, R. L.: *Programming Perl 2nd Edition*, O'Reilly (1996).

[34] Watanabe, Y.: *JCom (Java-COM bridge)*, `http://www11.u-page.so-net.ne.jp/ga2/no-ji/jcom/` (2001).

[35] White, K. J.: *SHAZAM User's Reference Manual Version 7.0*, McGraw-Hill (1993).

[36] White, K. J.: A General Computer Program for Econometric Methods - SHAZAM, *Econometrica*, pp.239–240 (1978).

[37] Wielemaker, J.: *SWI-Prolog Reference Manual*, `ftp://swi.psy.uva.nl/pub/SWI-Prolog` (1990).

[38] Yamamoto, Y., Nakano, J., Fujiwara, T., Kobayashi, I.: A Mixed User Interface for a Statistical System, *ISM SYMPOSIUM - Statistical software in the Internet age* -, pp.33–42 (2001).

[39] 金淵培: エージェント技術の現状と実用化, 人工知能学会誌, **12**, pp. 850–860 (1997).

[40] 小林郁典, 中野純司: 重回帰分析支援システム RASS におけるプログラミング言語, 統計数理, **45**, pp.1–11 (1996).

[41] 小林郁典, 中野純司, 渡邉勝正: 拡張したオブジェクトによる重回帰分析支援システム, 計算機統計学, 印刷中 (2001).

[42] 小林郁典, 渡邉勝正: 対話型プログラミング環境でのオブジェクトの一拡張法, オブジェクト指向最前線 *2001*, pp.65–68, 近代科学社 (2001).

[43] 柴田里程: データリテラシー, 共立出版 (2001).

[44] 新村秀一: 日本における統計ソフトの過去・現在・未来, 計算機統計学, **10**, pp.37–61 (1997).

[45] 田中豊: 回帰診断, 品質, **17**, pp.80–90 (1987).

[46] 田中豊, 垂水共之, 脇本和昌: *パソコン統計解析ハンドブック II 多変量解析編*, 共立出版 (1984).

[47] 所真理雄, 松岡聡, 垂水浩幸: オブジェクト指向コンピューティング, 岩波書店 (1993).

[48] 戸松豊和: Java 用スクリプト言語 Pnuts, `http://javacenter.sun.co.jp/pnuts/` (2001).

[49] 中川徹, 小柳義夫: 最小二乗法による実験データ解析-プログラム *SALS*, 東京大学出版会 (1982).

[50] 中野純司, 山本由和, 岡田雅史: 知識ベース重回帰分析支援システム, 応用統計学, **20**, pp.11–23 (1991).

[51] 本位田真一, 飯島正, 大須賀昭彦: エージェント技術, 共立出版 (1999).

[52] まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語 *Ruby*, アスキー (1999).

[53] 南弘征, 水田正弘, 佐藤義治: データ解析支援システムにおける知識ベースの枠組とその実現, 計算機統計学, **6**, pp.37–48 (1993).

[54] 脇本和昌, 垂水共之, 田中豊: *パソコン統計解析ハンドブック I 基礎統計編*, 共立出版 (1984).

[55] 渡部洋, 鈴木規夫, 山田文康, 大塚雄作: 探索的データ解析入門, 朝倉書店 (1985).

著者研究業績

本論文に関係する研究業績

学術刊行物（第一著者分）

(1) 小林郁典, 中野純司：重回帰分析支援システム RASS におけるプログラミング言語, 統計数理, **45**, pp.1–11 (1996).（本論文の２章に関する内容）

(2) 小林郁典, 中野純司, 渡邉勝正：拡張したオブジェクトによる重回帰分析支援システム, 計算機統計学, **14**, 2, (2002). 印刷中.（本論文の３章に関する内容）

(3) 小林郁典, 渡邉勝正：対話型プログラミング環境でのオブジェクトの一拡張法, オブジェクト指向最前線 2001, pp.65–68, 近代科学社 (2001).（本論文の４章に関する内容）

(4) Kobayashi, I., Fujiwara, T., Nakano, J., Yamamoto, Y.：A Procedural and Object-Oriented Statistical Scripting Language, Computational Statistics, **17**, (2002)(to appear).（本論文の５章に関する内容）

学術刊行物（第一著者分以外）

(1) Nakano, J., Fujiwara, T., Yamamoto, Y., Kobayashi, I.：A Statistical package based on Pnuts, COMPSTAT2000, pp.361–366, Physica-Verlag (2000).（本論文の５章に関する内容）

(2) 藤原丈史, 中野純司, 山本由和, 小林郁典：Java 言語による統計解析システム Jasp, 統計数理, **49**, 2, (2002). 印刷中.（本論文の５章に関する内容）

(3) Fujiwara, T., Nakano, J., Yamamoto, Y., Kobayashi, I.：An Implementation of a Statistical Language based on Pnuts, Journal of the Japan Society of Computational Statistics, **15**, (2002)(to appear).（本論文の５章に関する内容）

国際会議での発表

(1) Nakano, J., Yamamoto, Y., Kobayashi, I. : Object Oriented Knowledge Representation for Multiple Regression Analysis, Short Communications in Computational Statistics, pp.156-157 (1994). （本論文の２章に関する内容）

(2) Kobayashi, I., Nakano, J. : A Regression Analysis Supporting System based on Agents, Proceedings to The Ninth Korea and Japan Joint Conference of Statistics, pp.51–55 (1997). （本論文の３章に関する内容）

(3) Kobayashi, I., Fujiwara, T., Yamamoto, Y., Nakano, J. : The Language and the Extendibility of the Statistical System Jasp, ISM Symposium - Statistical software in the Internet age -, pp.65–73 (2001). （本論文の５章に関する内容）

(4) Kobayashi, I., Fujiwara, T., Yamamoto, Y., Nakano, J. : Extendibilities of a Java based Statistical System, Proceedings in International Conference on New Trends in Computational Statistics with Biomedical Applications, pp.109–115 (2001). （本論文の５章に関する内容）

(5) Yamamoto, Y., Nakano, J., Fujiwara, T., Kobayashi, I. : A Mixed User Interface for a Statistical System, ISM Symposium - Statistical software in the Internet age -, pp.33–42 (2001). （本論文の５章に関する内容）

(6) Yamamoto, Y., Fujiwara, T., Nakano, J., Kobayashi, I. : A Statistical system Jasp and its interface to TIMSAC programs, The International Symposium on Frontiers of Time Series Modeling, pp.330–331 (2000). （本論文の５章に関する内容）

(7) Huh, M. Y., Song, K. R., Yamamoto, Y., Nakano, J., Fujiwara, T., Kobayashi, I. : Davis-Jasp : A Data Mining Solution by Combininig Two Separate Java-based Systems, Bulletin of the International Statistical Institute 53rd session Contributed Papers Book 3, pp.197–198 (2001). （本論文の５章に関する内容）

(8) Yamamoto, Y., Nakano, J., Fujiwara, T., Kobayashi, I. : Implementing distributed computing abilities of a statistical system, Bulletin of the International Statistical Institute 53rd session Contributed Papers Book 3, pp.203–204 (2001). （本論文の５章に関する内容）

その他の研究業績

学術刊行物

(1) 小林郁典, 古本奈々代, 原田寛子 : 飲酒後の呼気中アルコール濃度のピーク値の経時変化に関するモデル化とシミュレーション, アルコール研究と薬物依存, **26**, 5, pp.401–418 (1991).

(2) 森本滋郎, 山本由和, 小林郁典, 古本奈々代, 田渕敏明 : 時変パラメータ追跡のための一適応アルゴリズム, 計測自動制御学会論文集, **33**, 11, pp.1108-1110 (1997).

(3) 森本滋郎, 山本由和, 小林郁典, 古本奈々代, 田渕敏明 : NLMS アルゴリズムにおけるステップサイズパラメータの一適応的設定法, システム制御情報学会論文誌, **10**, 11, pp.594-606 (1997).

(4) 古本奈々代, 田渕敏明, 清澄良策, 小林郁典 : 改良型デマーテル手法による医師診断基準の数量化分析, 医療情報学, **18**, 3, pp.205–210 (1998).

(5) 森本滋郎, 山本由和, 小林郁典, 古本奈々代, 田渕敏明 : NLMS, RLS, KF 型適応アルゴリズムの適応性能比較, 計測自動制御学会論文集, **35**, 6, pp.791-799 (1999).

(6) 田渕敏明, 古本奈々代, 森本滋郎, 山本由和, 小林郁典 : 漸近的に最適な RLS 適応同定アルゴリズム, システム制御情報学会論文誌, **13**, 1, pp.1–13 (2000).

(7) 田渕敏明, 古本奈々代, 清澄良策, 小林郁典, 森本滋郎 : 味覚感度の一推定法とその高齢者と若年成人との比較への応用, 計測自動制御学会論文集, **36**, 8, pp.717–719 (2000).

(8) 古本奈々代, 田渕敏明, 清澄良策, 小林郁典, 森本滋郎 : 顔型グラフによる介護福祉事業の充実度評価, 医療情報学, **20**, 5, pp.413–418 (2001).

国際会議での発表

(1) Kobayashi, I., Morimoto, J., Tabuchi, T., Furumoto, N. : Identification Methods of Alcohol Kinetics Model when the Measured Time is unknown, Proceedings of The Second China-Japan Joint Symposium on Medical Informatics, pp.131–133 (1997).