Vol. 43 No. 1 情報処理学会論文誌 Jan. 2002

# Rubyによる地球・惑星流体科学のためのプログラミング環境の開発

堀之内 武† 川那辺 直樹†

地球・惑星流体科学においては、離散化された多次元の場の物理量を解析するため、プログラミングが頻繁に行われる。しかし、現在多くの研究者が取る手法は開発効率が悪い。本研究では、扱うデータを隠蔽してプログラミングの効率を大幅に向上させるための基盤の開発を行った。データ解析に用いる言語にはRubyが相応しいと判断し、Rubyによるクラスライブラリーを提案、開発した。このライブラリーを用いることで、プログラムの行数が従来の手法を用いるより1〜2桁減少した。

# Development of a programming basis for earth and planetary fluid sciences with Ruby

Takeshi Horinouchi† and Naoki Kawanabe†

## 1. はじめに

気象学、海洋物理学、地球惑星電磁気学、地球惑星流体力学などの科学分野では、地球や惑星における流体を研究対象としている。即ち、研究対象は連続体という「場の物理量」である。ただし、ここで言う「場」は、必ずしも実空間を意味しない。例えば、物理量のスペクトルは頻繁に扱われるが、これは振動数または波数の関数である。これらの学問分野における研究活動は計算機とは切り離せない。天気予報や気候シミュレーションは、ここ数10年常に時々の最新の大型計算機を利用して行われてきた。観測データも計算機で処理される。例えば、連続的に大気をモニターする人工衛星や大気レーダーは年にギガからテラバイトの単位の膨大なデータを産み出す。計算機においては文字通りの連続体は扱うことは出来ないので（それは無限のデータ量を意味する）、扱われるのは何らかの形で離散化された、そして多くの場合多次元の、物理量である。

当該分野においては、研究活動のかなりの部分は、この多次元データから「科学的」な情報を引き出すことである。そのために様々なデータの処理や可視化（以後データ解析と呼ぶ）が行われる。しかしながら、研究の歴史の長さにも関わらず、人間の労力の投入という観点から見ると、データ解析は必ずしも効率的に行われていない。当該分野の研究者の多くは、データの構造に大きく依存したプログラミングによりデータを解析している。従って、物理的には「同じこと」を行う場合も、次元性や格子配置等といった物理的な構造やファイル形式が異るデータを扱う場合に、大きな労力を要する。そして、年々の研究の高度化に伴い、一人の研究者が扱うデータの種類は増大しており、研究者の負担は増大している。データ量についても、計算機の進歩に連れて、あるいはそれを上回るペースで増大しており、その意味でも楽になってはいない。

上記の問題を解決するには、データ解析のためのプログラミング基盤の抜本的な整備が必要である。このため、我々は、オブジェクト指向技術を用いて、データ構造になるべく依存しない形で操作可能な、物理量のためのクラスライブラリーを開発し、データ解析の基盤とすることを目指している。本研究では、本格的なオブジェクト指向言語であり、かつ一般に使いやすいと考えられるRubyを採用する。なお、本研究での開発目的は、データを解析するための基盤を作ることに留まらず、数値シミュレーションへの応用も目指している。さらに、将来展望としては、ネットワーク上に分散したデータの利用、データサーバーへの応用等を考えている。本プロジェクトのホームページへは文献1)よりアクセス出来る。

以下、第2節では、当該分野におけるデータについて説明し、第3節で従来のデータ解析法とその問題点を述べる。次いで第5節で概念的な構成を述べ、第6節で実装面について述べる。そして、第7節で実行例と従

† 京都大学
Kyoto University

来法との比較を示し、第8節で結論と将来展望を述べる。なお、当ライブラリーを開発するためにあたって、Ruby には含まれない可視化等のライブラリーの開発を行う必要があった。付録でそれらに簡単に触れる。

## 2. 地球・惑星科学におけるデータ

本節では地球・惑星科学におけるデータがどのようなものかを説明する。まず、例を用い、データに内在的な特徴を述べる。ついで、それを格納するファイルの形式について論じ、代表として NetCDF ファイル形式を紹介する。

### 2.1 データの特徴

例として地球を取り巻く大気のデータを考える。日々の天気の予報に用いられる数値モデルでは、通常、経度・緯度・高度に関して格子を設定し、格子点上における気温等の物理量を予報変数とする。従ってその出力は、経度、緯度、高さ、そして時間についての４次元配列状になる。但し、このデータ物理量について何か調べたり可視化するためには、この配列に加えて、各格子点の位置やサンプルされた時刻、つまりデータの座標値が必要である。さらに、数値データだけではなく、当物理量は気温で単位は K である、第1次元目は経度で単位は度であると言った情報を必要とすることも多い。

上の例に見られるように、当該分野で扱うデータの多くは座標と一般に多次元のデータ配列からなる。座標、データ配列はそれぞれ名前や単位といった属性を持つ。但し、全ての格子点上でデータが存在するとは限らず、観測データでは観測不良などに伴うデータ欠損があり得る。観測的な欠損がなくても、上の例のように高度を軸に取れば、高地では地面の下になる格子点で気温は定義されない。なお、長さの揃った長方形的な多次元配列では表されない不規則格子が用いられる場合、さらに非格子的にサンプルされたデータが扱われる場合もあるが、空間における位置とそこでのデータの組であるという点は共通である。なお、第1節で述べたように、ここで言う「空間」は必ずしも実空間ではない。

### 2.2 データを格納するファイル

一般に、一つのデータセットを形成するファイルが一つであるとは限らない。本稿でファイルと言う場合は、OS から見れば複数のファイルにまたがる場合も含むことにする。

歴史的には、地球・惑星科学におけるデータは、作成者が思い思いの形式で作成していた。従ってそれは研究者・グループ毎にばらばらであり、読むためには、ドキュメンテーションからデータの並び方等を判断するか、データ作成者が供給するサブルーチン (多くは FORTRAN である) を利用することが必要である。

近年では、データを解釈するのに必要な情報を原理的には全て埋め込むことが出来る、「自己記述型」の幾つかの形式が利用されることが多くなった。特に良く利用される自己記述型データ形式としては NetCDF[2)3)], HDF[4)5)] がある。共に、移植性の高い専用ライブラリーを介して入出力を行わせることでバイナリー構造を隠蔽しており、両者の概念的な構成も類似する。次節では NetCDF ファイルについて詳しく述べる。これら以外に、上記の例で挙げたような大気・海洋の４次元データに特化した自由度の小さい GrADS[6)7)] 形式や、grib 形式も良く用いられる。前者はもともとは同名の可視化ソフトウェア用の形式であるが、バイナリー構造の簡便さのために利用が広まった。後者は複雑で利用も難しいが、世界気象機関の規格であるため、各国の気象機関のデータ配布に使われることが多い。GrADS, grib とも、データの次元構造に制限があるだけでなく、どのような属性を定義出来るかという点も予め決まっている点でも、NetCDF, HDF より自由度が劣る。

### 2.3 NetCDF ファイル形式

NetCDF[2)3)] は「配列指向」のデータセットであり、１つのファイルに複数の多次元配列変数を納めることが出来る。配列要素の型は変数毎に一つに決まっており、整数、浮動小数点数のいずれかである（細かく言えば unsigned char, char, 2及び4バイト符合付整数, 4及び8バイト浮動小数点数である）。

NetCDF では、ソースコードが公開された多くのプラットフォームで稼働する専用のライブラリーを介して入出力を行わせることで、データのバイナリー構造は隠蔽される。XDR を用いて浮動小数点は常に IEEE big endian で入出力されるなど、プラットフォームに依存しない入出力が行われる。なお、NetCDF ファイルの内容は、浮動小数点のテキスト化による誤差を無視すれば Common Data Language (CDL) と呼ばれる独自の形式でのテキスト表現で表すことが出来る。このため、NetCDF ファイルはより抽象的に NetCDF データセットと呼ばれる。

図1に、NetCDF データセットの構成を示す。NetCDF データセットは任意個の次元と変数を持つ。データセット及び変数は任意個の属性を持つことが出来る。次元、変数、属性は全て英数字からなる名前で識別される。変数は0次元のスカラーを含む多次元配列である。一方、NetCDF における「次元」は、変数の各次

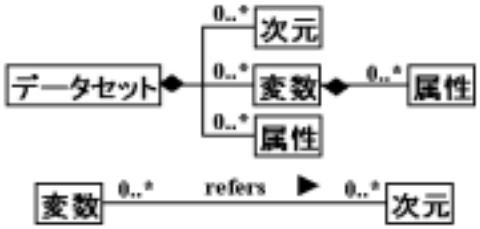



**図1** NetCDF形式におけるファイル構成の概念図。菱形を端に持つ線はコンポジション集約を表し、“0..*” は任意個であることを示す。

Fig. 1

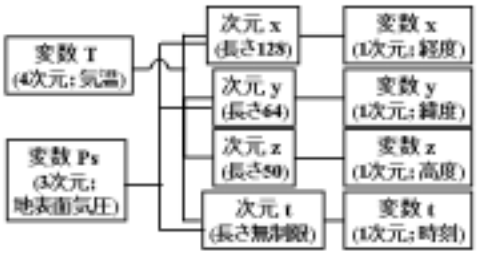



**図2** NetCDF形式における変数相互の関連づけの例。この例では同じファイル中の変数 T 及び Ps が座標変数を共有している。

Fig. 2

元の長さを指定するための整数オブジェクトである。変数の次元の長さを、次元オブジェクトを用いず直接指定することは出来ない。よって、変数はその次元数だけの次元オブジェクトと関連づけられている (図 1 下側のダイアグラム)。また、一つの次元は一般に複数の変数に関連づけられる。なお、データセット中に1つは長さ不定の「無制限次元」を持つことが出来る。

属性の値は文字列、または数値のスカラー乃至1次元配列である。データセットの属性はグローバル属性と呼ばれる。属性名については、ドキュメンテーションに強く推奨されている標準的な属性の「慣例」があり、実際に作られるほとんどのデータはそれに従っている。具体的には、変数の内容的な名前（変数名 x に対して「longitude」など)、単位、データ欠損閾値等が対象となっている。しかし、標準慣例のみでは必ずしも充分でないということで、拡張慣例が幾つか提案されている[8)9)10)11)]。

NetCDF では次元名と同名の変数を座標変数とみなすというルールにより、従属変数と独立変数 (座標) が緩やかに結び付けられている。これを図2に例示する。このデータセット中には、次元名 x, y, z, t にぞれぞれ対応する1次元変数 x, y, z, t が存在し、座標値を保持している。そして、T という4次元変数と Ps という3次元変数が、独立変数を共有している。なお、解釈のルール上変数 x の座標変数は x 自身となる。

## 3. 従来のデータ解析手法とその問題

地球・惑星科学における従来のデータ解析方法は以下のように2つの観点から分類することが出来るであろう。

データの自己記述性:

(a) データの自己記述性を利用しない。または非自己記述データを利用する

(b) データの自己記述性を利用する

スタイル:

(1) プログラミングによるデータ解析

(2) GUI またはコマンドライン入力によるデータ解析

2つの分類には相関がある。即ち、(a) の場合 (2) は困難であるため (1) となる。一方、(b) の場合 (1),(2) とも可能であるが、現状では (2) が中心である。なお、(2) のコマンドライン入力と (1) プログラミングの境は曖昧であるが、(2) のほうは、関数が書けないなど、まともな言語とは言難いものを指しているつもりである。GUI とまとめてあるのは説明の便宜のためである。以下、それぞれの組み合わせについて論じる。

(a-1) データの自己記述性を利用しないプログラミングによるデータ解析

現在主に使用されている言語は、FORTRAN77, Fortran90, C, yorick, IDL, Matlab などの手続き型言語である。yorick 以下の3つはインタープリター型で型付けがなく、対話的にも利用できる。yorick はフリー、IDL と Matlab は商用である。

このケースには以下のような問題点がある。先ず、プログラムがデータ構造依存になり、使い回しが難しくなりがちである。また、データの自己記述性を利用しないため、利用した場合に比べて遥かに長い行数のプログラムが必要である場合が多いという問題もある。つまり、プログラムを一つ開発するにも開発コストが本来必要な以上にかかる上に、プログラムの適用範囲が限られるのである。

使用する言語の違いによる開発効率の違いも無視できない。Fortran や C を用いる場合、型付がなく対話的にも利用できる言語を用いる場合に比べて開発効率が悪い。また、Fortran90 以下は配列機能が充実しているが、FORTAN77 と C はそれすらなく、データ解析に用いるのは論外と言って過言でない。しかし、実際は現在もまだこれらを使っているグループは多く、地球・惑星科学の進展を遅らせる原因の一つになっていると言える。

(b-2) データの自己記述性を利用した、GUIやコマンドラインアプリケーション

NetCDF または HDF に対応した GUI アプリケーションとしては、ncview, ferret, IVE などが存在するが、いずれも機能は限定的である。コマンドラインアプリケーションとしては GrADS が、GrADS 形式、及び GrADS 互換な特定の拡張慣例に従う NetCDF, HDF を扱える。他に gtool3, gtool4[14] が存在する。

このケースにも問題が多い。GrADS の場合、当該学問分野で扱うデータの一部にしか対応しないので、それだけで研究できる研究者は少ない。GUI アプリケーションの場合、データの自由度のと引き替えに、データ解析の自由度が限られるのが問題である。GUI を使えば、与えられたメニューにあることは容易に出来る一方で、メニューにないことを行うのは不可能であるか、出来るとしても数多くのステップを要するであろう。従って、常に新しい手法を個々のユーザーが開発、使用していくことが要求される科学研究には使いにくい。さらに、より根本的に、プログラムという形での解析手法の一般化と蓄積が出来ないという問題がある。このことは、コマンドラインアプリケーションについても言える。

GUI は補助的に用いると大きく能率が上がる場合がある。ただし、現状では NetCDF または HDF を扱う GUI アプリケーションは、いずれもごく限られたメニューしか持たない。

(b-1) データの自己記述性を利用したプログラミングによるデータ解析

自己記述性を活かすためには、そのための構造体またはクラスのライブラリーが必要である。しかし、広く用いられているそのようなライブラリーは存在しないようだ。但し、手続き型言語を使って一つのファイル形式に特化した構造体のライブラリーは幾つか存在し、それぞれ多少のユーザーに使われている。

## 4. 言語の選択

本研究で開発するのは、第3節の分類における (b-1) の基礎となるライブラリーである。データ解析をする立場からは、データが保持されている物理的な実体（外部ファイルにあるか、メモリー上にあるか。ファイルならどの形式かなど）に可能な限り依存しないプログラミングが行えることが望まれるので、オブジェクト指向言語を採用することが望ましい。ファイル形式を隠蔽したオブジェクトを立ててユーザーにアクセスさせることで、複数の自己記述的データ形式が、統一されたインターフェースを通じて操作されるようになり得るからである。さらに、非自己記述的データについても、情報を補うテンプレートを与えることで自己記述的データと同様に扱うことは可能である。つまり、本開発の成果を利用して (a-1) を (b-1) の土俵に乗せることも視野に入れている。

言語の選択に当っては、エンドユーザー、即ち研究者・学生が、日常的にプログラミングを行なう必要があるということを考慮する必要がある。データ解析においては、時間をかけて作りこんだプログラムを繰り返し使うのでなく、常にプログラミングをしながら研究するというスタイルが普通であるため、開発効率が高く、対話的にも利用できるものが望ましい。対話的に利用できる言語は、コマンドライン入力とプログラミングの間をシームレスに移行できるため、試行錯誤の結果有望であると判断された解析手法をプログラミングとして蓄積しやすいという利点がある。以上を考慮して Ruby[15] を選択した。C++ や Java は、より使用人口が多いものの適さない。なお、Ruby でなく Python を使うことも可能であるが、Ruby のほうが使いやすく相応しいと判断した。

地球惑星科学研究の立場から Ruby にを用いる利点として、その他に以下を挙げることが出来る。

- 移植性： 稼動プラットフォームが多い (多くの UNIX と Windows, Mac)。C で書かれており、ソースコードで公開されているので、移植さえ出来ればどこでも利用できる。非商用であるため、教育用の大量導入もしやすい。
- 拡張性: C 言語を用いて容易に拡張ライブラリーを作ることができ、多くのプラットフォームで Ruby 本体の再コンパイルなしに利用できる。よって、C や Fortran で書かれた既存の関数、サブルーチンを活用することが出来る。
- 文字処理がし易い： 可視化やファイル名・変数名の処理等、データ解析の様々な場面において文字処理は必要である。
- ネットワーク関連等豊富で増大しつつあるクラスライブラリー：
  将来の発展を考える上で重要である (第8節参照)。
- その他：後から取って付けたのでないオブジェクト指向のため自然。ゴミ集め等、基本的な現代的機能を装備。

一方、デメリットとしては、 Ruby の標準配布パッケージには、多次元配列クラスや可視化ライブラリーが存在しないなど、基礎となるべきライブラリーが幾つか組み込まれていないということがある。多次元

配列については、最近デファクト標準に近づいているといえる拡張ライブラリーを用いることにし、その他のライブラリーについては、C言語で書かれたものを「ラップ」して Ruby で利用できるようにした。これについては付録を参照のこと。

## 5. 離散化された場の物理量のデータモデル

扱うデータ構造の自由度の高くすることと、データの扱いの高度化（即ち一つの命令により多くのことをさせてプログラムの行数を減らすこと）の間にはトレードオフがある。従って当該分野の大部分のデータをカバーし、且つその範囲で出来るだけ自由度を絞ることが必要である。このため、扱うデータに共通する内在的な構造を適切に抽象化し、ファイル形式に関わらずそれを操作の基本的な単位とすることが有効であると考えた。

基本単位としては以下に述べる抽象的な「物理量」データモデルを考え、外部ファイルの入出力は物理量のメソッドを通じて行なう。さらに、すべてのデータが実行時のメモリー上に存在するデータに関しても同じ基本単位で扱う。よって、この基本単位を用いることで、データ内容が記録されている様態に依存しないプログラミングが可能となるはずである。しかし、このようなことを、現実に用いられている多くのファイル形式に合わせてどう実現すればよいかはトリビアルな問題でない。第6節では実際の開発に即してこれについて論じる。

第2.1節に述べた、地球・惑星流体科学データの特徴を踏まえて、図3のように、離散化された場の物理量のデータモデルを構成した。本研究では、第6節で述べるように、このツリー構造の全体をデータ処理の基本的な単位とするクラスライブラリーを開発する。

本データモデルは、図1,図2 に示したNetCDF のそれを発展させたものである。個々の変数が多次元配列と属性からなるという点は同じである。しかし、変数間の関連付けに関しては、以下のような違いがある。

- NetCDF の変数の組み合わせは主変数とその次元数分の座標位置変数からなる。しかし、実際には、軸に関する情報は格子点の座標値に留まらないことが多い。NetCDF ではそれを属性により補うことは可能であり、幾つか手法が提案されているが、現状では統一されていない[11]。
- NetCDF では名前を介した関連で繋がっている変数群が本モデルに対応すると見なせるというだけで、「物理量オブジェクト」は陽には存在しない。また、「座標」オブジェクトも存在しない。本モデルでは、物理量オブジェクトを作ることで関連が陽的になり、座標オブジェクトを設定することで柔軟性が増している。例えば、座標変換など各座標軸に対して独立に適用されるとは限らない操作や、不規則な格子への対応が行い易い。
- 本モデルではファイルは前面に出ない。中身がファイルに保持されるデータであれば、ファイルは物理量オブジェクトの初期化の時に指定され、ファイルオブジェクトは内部的に保持される。

## 6. 実　　装

本節では作成したクラスライブラリーについて解説する。ただし、まだ開発は初期段階であり、未実装な機能が多い。以下では未実装部分も含めて記述、実装されてないものにはその旨を記す。

### 6.1 物理量クラス GPhys の構成

第5節で示したデータモデル(図3)に従って、図4のように離散化された場の物理量クラス GPhys を構成した。物理量 GPhys は、座標クラス Coord と多次元変数クラス GData のオブジェクトを１つづつ持つ。Coordオブジェクトは格子点値等、データの座標に関する情報を持ち、GDataオブジェクトは主物理量の値や属性を保持する。GDataは実体のない抽象クラスであり、データが全て実行時のメモリー上に存在する場合はGDataMem、NetCDF ファイルに格納されている場合は GDataNetCDF クラスと呼ばれる。現在のところ他の形式のファイルに対応する GData クラスは未実装である。

本クラスライブラリーのユーザーは基本的に GPhys オブジェクトをプログラミングの対象とする。但し、GPhys 既存のメソッドでは対応できない場合は、必要に応じて GData, Coord 等を用い、GPhys の新しいメソッドを作ることになる。この場合でも、ファイル形式依存性は GData の内部に隠蔽されているので、汎用性が保たれる。

GDataMem においては、物理量の値は多次元配列 NArray (付録参照) に蓄えられ、属性はハッシュテーブルを利用した Attr クラスのオブジェクトに蓄えられる。このオブジェクトは一つで任意個の属性を管理するので、GDataMem 中に一つだけ存在する。ファイル上にデータが存在する場合は、GData 本体はファイルハンドラーを持つだけである。GDataNetCDF の場合、それに相当するのはNetCDF 変数のクラスである NetCDFVar のオブジェクトである。GDataNetCDF に対する値等の取り出しや設定メソッドは、配下の NetCDFVar オブジェクトに移譲される。なお、書き


**図3** 離散化された場の物理量のデータのモデル。菱形は集約を表す。
Fig. 3

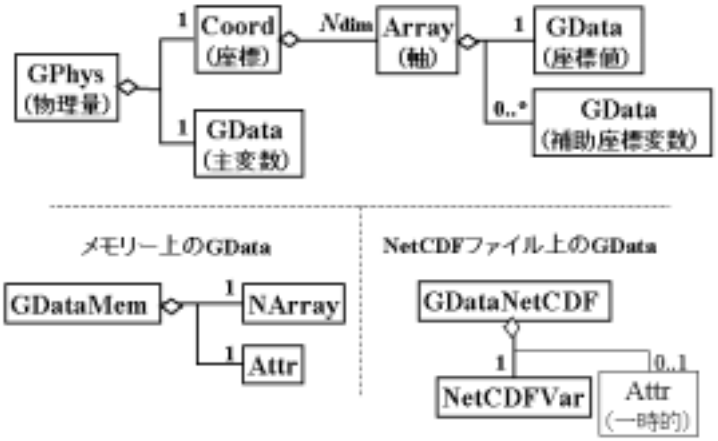



**図4** 離散化された場の物理量クラス GPhys の構成。長方形内の英字はクラス名を示す。
Fig. 4

込みの出来ないファイル中の変数に一時的に属性を設定出来るよう、Attr オブジェクトを1つ持つことが出来る。

座標クラス Coord は、離散化された座標に関するデータを保持する。GPhys 中では主変数と座標は次元数と各次元の長さが一致するように保たれる。Coord オブジェクトは、その次元数だけの「軸」オブジェクトを持つ、これは座標値及び任意の補助座標変数の組である。座標値は矩形格子ではそれぞれ1次元だが、変形格子では多次元となる (未実装)。また、非格子データでは1軸当りの座標変数が複数個になることがある (未実装)。これらは GData オブジェクトに保持されるため、データが外部ファイル中に存在する場合と実行時メモリー上に存在する場合がある。

### 6.2 各クラスの機能

ライブラリーの中心は GPhys であるが、その機能は可能な限り、GData, Coord に移譲されるので、まずこの2つについて述べて、最後に GPhys について、両者の組合わせから産れる GPhys 独自の機能を中心に述べる。

#### GData

主なメソッド

- 値、属性の設定、取り出し
- 単項演算 (算術演算、要素毎の数学演算、平均・分散等)
  計算結果はデフォルトでは GDataMem のオブジェクトになるが、出力先は他のクラス (即ちファイル上) に切り替えられる。データ欠損の扱いは未実装。以下同様。
- 2項演算 (算術演算)
  異る NetCDF のサブクラス間で演算も可能。
- GDataサブクラス間の型変換
  結果的にファイルからデータを読み出してメモリーに格納したり、ファイル形式変換などが行われる。
- サブセット切り出し
  参照のみの場合 (第6.4節の Mapper 利用) と実際に切り出す場合がある。
- 大規模データ分割の自動分割イテレーター

(第6.4節参照)

- 数学・統計ライブラリー
  現在はほとんど未実装。GNU Scientific Library (GSL)[16] などの Ruby インターフェースの組み込みが予定されている。

**Coord**

主なメソッド

- サブセットの切り出し
  GData と同様に、参照のみの場合と本当の切り出しが行える。また物理的な座標値を指定する切取も行えるようになっている。
- 座標変数、補助変数の設定、取り出し
- 内外挿 (未実装)
- 座標変換 (未実装)

**GPhys**

主なメソッド

- 座標に関する情報を必要としないメソッドは GData 単体のメソッドと同じである (実装は移譲による)。サブセット切り出しに関しては、Coord について述べたのと同じ機能がある。座標と主変数の対応は正しく保たれる。座標変換は Coord に移譲し、内外挿は主変数と内外挿先の座標を現在の座標のメソッドに与えることで行う (未実装)。
- ２項演算は次元数の違うデータ間でも長さ及び名前により次元の対応が取れれば実行される。
- 離散フーリエ変換等、計算結果は座標値に依存しないが、演算の結果座標変数に変化があるような演算では Coord オブジェクトが正しく更新される (未実装)。
- 数値微分等、座標に関する情報を必要とする演算 (ほとんど未実装)。
- 可視化機能
  AdvancedDCL (付録参照) を利用。モジュール化してある。

以上、現在のところ実装済みの数学・統計演算は非常に限られているが、今後ユーザーの貢献により増えていくことが期待される。

### 6.3 ファイル中のデータと GPhys の対応の自動化

前述のように、NetCDF に限らず外部ファイル中のデータは GPhys の構造に当てはめた上で読み書きされる。ユーザーは物理量データのメソッドを通じてデータにアクセスするので、どのフォーマットのデータを読み書きする場合も基本的に同じ命令を呼ぶことになる。物理量データの構成要素をどのようにファイルに格納するかは形式依存であるが、それは隠蔽される。

この当てはめは自動的に行うことが必要である。自動化出来ない場合、GPhys オブジェクトを構成する段階で、ファイル形式やデータ格納の仕方に応じたプログラミングが必要となり、前述のファイル構造の隠蔽が破れてしまうからである。

当然のことながら、自動化するためには、ファイルから何らかの形で GPhys に対応する構造を読み取ることが可能でなければならない。つまり、第2.2節で述べた自己記述的データフォーマットである必要がある。しかしながら、自己記述型データの場合でも GPhys に対応する内容を機械的に読むことが完全には出来ない場合もある。また、自己記述型でないデータを扱わねばならないケースもあり得る。そのような場合でも、足りない情報を補うためのテンプレートを与えることで、それ以後の処理を自動化することが可能である。以下に対応づけの方法を述べる。

- 変数と属性の対応
  NetCDF 変数は任意の属性が定義可能であるため、GData と NetCDF 変数は１対１に対応付けられる。よって、GData における変数値や属性値を求めるメソッドは単に NetCDFVar に処理を委譲すれば良い。一方、属性が定義できない、あるいは定義できる属性が限られているファイル形式においては (未対応)、その形式における多次元変数を代表するオブジェクト (NetCDF の場合の NetCDFVar に対応するもの) が、GDataMem で使われる Attr (属性) オブジェクトを持てるようにする。
- 変数間の関連
  NetCDF では座標位置変数は名前を介して判断される。また補助変数については属性で指定されることがあるが、その方式は完全には統一されてない。一般に、ファイル形式によって変数間の関連の付け方は異なる。そこで、主変数を元に関連を判断するメソッドを納めるファイル形式毎のモジュールを用意し、判断の方法はモジュール内に隠蔽する (未実装)。NetCDF における補助変数のように、判断の仕様が統一されてないケースに対しては、外部から判断のためのテンプレートを指定・変更出来るようにする。NetCDF の場合、それは補助変数の種類名とそれを指定する属性名の組のハッシュテーブルである。

### 6.4 サブセットマッピングと大規模データの扱い

第6.2節で述べたように、GData, Coord, GPhys に

おいて、サブセットの切り出しは参照のみでも行えるようになっている。これを利用して、メモリー上に一度に読み込むには大きすぎる外部ファイル中のデータの処理を、複数回にわけて行う仕組みを実装した。

参照による切り出しは、元のデータとサブセットの対応関係を保持する Mapper クラスのオブジェクトを GData オブジェクトに持たせることで実現する。Mapperクラスは、マッピングの合成を行うことが出来、サブセットのサブセットも正しく取ることが出来る。従って、巨大なデータの一部だけをデータ解析対象とすることが効率よく出来る。一方で、Mapper オブジェクトの配列を用いて、時系列等で分割されたデータセットを繋げて一つとして扱うオブジェクトを定義することも可能である (未実装)。

大規模データ全体を処理する場合は、この Mapper を利用して処理を分割する。自動的に分割を行うためのアルゴリズムを考え、分割方法の詳細を隠蔽した形で繰り返し処理を行うイテレーターを作った。分割は、予め設定されたサイズに照らして、出来るだけ「ゆっくり変る」次元から行われる。その際、行う処理の都合で分割してはいけない次元が指定できる (例えば第3次元目についての平均を取る場合は第3次元目を指定する)。サイズが閾値以下になる分割が行えない場合、デフォルトでは分割不十分なまま処理を続行する。イテレータには、分割されたサブセット GData オブジェクトをループ変数にするものと、Mapper のオブジェクトをループ変数にするものがある。前者についてはブロックの返り値を元のオブジェクトに代入するものとしないものの2つがある。以下は、代入を行う場合のコードの例である。

```
bigdata.each_subset_set{|sub|
   x = - sub * (sub - 1.0 )
   x[ x.lt 0 ] = 0
   x
}
```

この例では bigdata を自動的に分割したサブセットへの参照をsub とし、{ } で囲まれたブロックの最後に評価したx で置き換えられる。なお、bigdata が GDataNetCDF の場合、sub も GDataNetCDF となるが、演算はメモリー上で行われ、x は GDataMem となる。

なお、未実装であるが、GData標準の演算にはこれらのイテレーターを組み込み、イテレーターそのものを隠蔽出来るようにする。但しデフォルトではこの自動分割機能はオフにする。

## 7. 実行例と従来法との比較

### 可視化

米国海洋大気庁 (NOAA) が NetCDF で提供している、月別の気候値データ[17] を使用することにする。気温のファイルは、以下のように構成されている。

```
dimensions:
    lon = 144 ;
    lat = 73 ;
    level = 17 ;
    time = UNLIMITED ; // (12 currently)
variables:
    float level(level) ;
    float lat(lat) ;
    float lon(lon) ;
    double time(time) ;
    float air(time, level, lat, lon) ;
```

ここでは、気温を含むファイルを NetCDF のテキスト表現 CDL に変換したものより、変数の定義に関係する部分だけ抜き出している。各変数は名前や単位等の属性を持っており、以下のデモではそれらが自動的に解釈されて表示されているが、上では省略した。気温を表す変数はair である。その定義における次元の並び方は、C 式に最もゆっくり変るものを最初に書いてあるが、以下の Ruby コードではその逆に、最も速くかわるものを 0 として、ゆっくり変る次元に向けて数えていく。

GPhys を使う場合の可視化は、もっとも簡単には以下のプログラムで行える。各行の左の番号は行番号を表すために付した。よって実際のプログラムには含まれない。

```
1:  require "numru/gphysrect"
2:  require "numru/gphys/gphys_graphic"
3:  include NumRu
4:  temperature = GPhys.open("air.mon.ltm.nc","air")
5:  temperature.contour
```

最初の2行はライブラリーのロードであり、最初に1度だけ行えば良い。名前空間の重なりを防ぐよう、ライブラリー全体が NumRu というモジュールに入っている。3行目はそれをインクルードして、現在スコープで以後は NumRu というモジュール名をタイプしないで済むようにしている。4行目では、ファイル air.mon.ltm.nc 中の air を主変数とする GPhys を構成する。第6.3節に述べた自動化により、1行で済んでいる。5行目では、等高線 (contour) 図を作成している。その結果

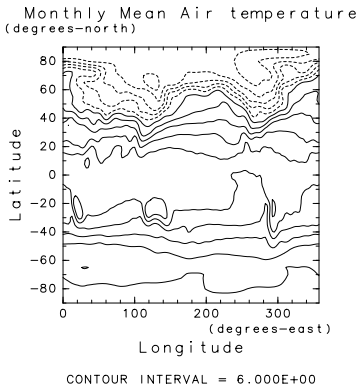



**図5** GPhysを用いた可視化のデモ1
Fig. 5

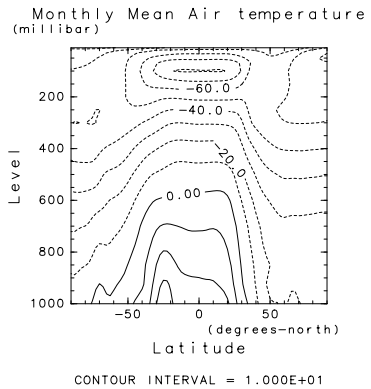



**図6** GPhysを用いた可視化のデモ2
Fig. 6

を図5 に示す。等高線は2次元量に関する図であるため、気温を構成する4次元から何らかの2次元の断面を選ぶ必要があるが、デフォルトでは最初の2次元に関する図を、それ以外の次元の最初の要素から切り出して描画する。よって、本例では最下層、1月のデータが描かれている。なお、可視化ライブラリーに基礎になっているDCL (付録参照) は様々な描画方法と強力なレイアウト機能を持つ。これらはすべて Ruby から利用できるようになっており、論文出版に堪えるクオリティーの描画が出来る。

以下は、上記の描画に続いて、別の断面を選んで描画する場合の例である。

```
6:  gphys.to2d(:dv1=>120).contour
```

ここでは、第一次元 (経度) の座標値が 120 に最も近い断面を取っている。結果を図6に示す。他にも、範囲を指定したり、物理的な座標でなくインデックスの番号を用いた指定が可能である。

以上の例においては、GPhys オブジェクト内の座標の情報を元に、図の縦軸や横軸が自動的に正しく書かれ、物理的な座標値で指定された断面の切取りが行われている。同じことを、従来多くの研究者が行ってきたように、データの自己記述性を用いず、配列や文字列を別々に扱ってプログラムを書くと、約100行のプログラムとなった。すなわち、GPhys を用いることでプログラムの長さが1〜2桁減ったことになる。

### 数学演算

例として平均操作を取り上げる。以下では、NetCDF ファイル上にデータがある3次元の GPhys オブジェクト `p` から、第一次元目の平均値を引いている (次元は0から数える)。

```
1:  pbar = p.avg(0)
2:  b = p - pbar
```

第2行目は、第1次元が落ちて2次元になった `pbar` と、3次元の `p` との間の演算である。次元数が異なるデータ間では差分演算子は次元の対応を調べて演算するようになっているので、この場合でも演算は期待通りに行われる。対応付けは、転置はないものとして次元の長さを比較し、それで一意に決まらない場合は座標値変数の名前の一致から決められる。計算結果の出力先は、デフォルトではメモリ上であるので、`b` は GDataMem のオブジェクトとなる。これを NetCDF ファイルに出力するには、以下を実行すれば良い。

```
3:  b.output(GDataNetCDF,filename)
```

上記の演算を GPhys を用いないで行う場合、次元数の異る演算のところではループを用いる必要がある。また、上記の `b` に相当する計算結果は単なる多次元配列 (または GData) であるから、その結果を自己記述的なファイルとして出力するためには、計算結果の第1,2次元が元のデータの第2,3次元であることを正しく認識して構成しなければならない。従って、プログラムの長さは格段に長くなり、バグ混入の可能性も増える。特に、従来の多くの研究者のように純粋な配列と文字列のみでプログラミングする場合、結果を正しくファイルに書出すためのプログラムの長さは、GPhys を用いた場合に比べて、やはり1〜2桁増えるであろう。

## 8. むすび

本研究では、地球・惑星流体科学への応用を念頭に、データ解析プログラミングのための基礎ライブラリーの開発を行った。当該分野で扱われる、離散化された場の物理量のモデルを考え、解析対象とするデータをそれに当てはめることを提案し、まだ初期段階ではあ

るが、それに従って実装を行った。

本ライブラリーの利用においては、利用者、即ち研究者・学生がプログラミングの基礎単位とするのは、データの物理的な格納形態を隠蔽した多次元データのオブジェクトである。それは、座標 Coord、多次元データ GPhys 及びこれらを統合した、場の物理量 GPhys である。第7節に例示したように、利用者は GPhys を基本的な解析の単位とすることが推奨されているが、新たな解析手法をプログラムする際には GData や Coord もプログラミングの対象となる。扱うデータが一度にメモリーに読み込むには大きすぎる場合にも対応するため、本ライブラリーは、詳細を隠蔽してデータを分割し、繰り返し処理を行わせるイテレーターを備える。

本ライブラリーを用いることで、データ解析のプログラムの行数が、従来法と比較して、1〜2桁も減少する。また、利用者が書くプログラムは、用いたデータのファイル形式や、さらには場合により次元数・座標配置等のデータ自体の構造にも依存しにくいため、研究コミュニティー内でデータ解析のためのプログラム資源を共有、蓄積しやすい。

最後に、地球・惑星流体科学における Ruby の利用に関する将来展望を述べて結びとする。

- **数値シミュレーションへの応用**:
  計算機の性能がどれだけ上がっても、常にそれを最大限に利用した効率のよい計算の需要はあり続けるだろう。しかし、例えば、最も複雑で大規模な気候モデルを用いても、あるいはそのような数値モデルを用いるがゆえに、その中で発生する大気現象の「理解」はもたらされないことも多い。理解するためには、複雑で包括的な数値モデルだけでなく、様々なレベルで単純化されたモデルを併用することが重要である。その場合、最高の計算効率を追求する必要はない。むしろ重要なのは開発効率である。本研究で開発しているライブラリーは C のべた並びポインターを利用した NArray に基づいているため、そこそこの効率は出るであろう。そして、開発効率という点では、従来数値計算に用いられてきた Fortran より遥かに良い。現在、Ruby のベクトル計算機への移植が進行中である。さらに、将来的には MPI 等を利用した並列化を、通信を出来るだけ隠蔽した形でプログラミングできるようなライブラリーが登場することが望まれる。
- **リモートデータへのアクセスと、ネットワークを通じたデータ配信**:
  現在インターネットを通じたデータ配信が盛んに行われている。その量や、研究者がそれに依存する度合いは今後も益々増大するであろう。そのようなデータを解析する場合、データをダウンロードしてから行うことが一般に行われているが、巨大なデータセットを扱う場合、そのための苦労は無視できない。データをダウンロードすることなしにアクセスし、転送が必要な解析結果だけを転送することが出来れば、そのほうが望ましい。現在でも幾つかのデータセンターは CGI による作画のサービスを提供している。しかし、第3節で述べたのと同様に、CGI ではデータ供給者が用意した機能しか利用できない。そしてそれはかなり限られていて、研究にまでは結び付かないことが多い。利用者が望む解析を実現するためには、解析メソッドは利用者に供給させる必要があるであろう。そのためには例えば、Ruby による分散オブジェクトを利用するといった手段が考えられる。原理的には、データがローカルホストにあるのとさほど変らない形でアクセスできるようにすることは可能である。しかし、データサーバーの安全性を確保した上で、解析を利用者に任せる仕組みを実現するにはどうすればいいかということは未検討である。

**謝辞** 本開発に協力頂いた後藤謙太郎、黒井啓子、榊間俊洋、高橋千賀子、神代剛、高橋憲義の各氏、議論し助言を頂いた塩谷雅人、林祥介の各氏を初めとする電脳Rubyプロジェクトの皆さんに感謝します。

## 参 考 文 献

1) Horinouchi, T.: Dennou Ruby Project, http://www.gfd-dennou.org/arch/ruby/ (2001)
2) Rew, R., Davis, G.: NetCDF - An Interface for Scientific-Data Access *IEEE Computer Graphics and Applications*, 10, pp. 76-82 (1990).
3) Rew, R., et al.: NetCDF, http://www.unidata.ucar.edu/packages/netcdf/.
4) Fortner, B.: HDF: The hierarchical data format *Dr Dobbs Journal*, 23, pp.42 (1998).
5) The National Center for Supercomputing Applications (NCSA): Hierarchical Data Format, http://hdf.ncsa.uiuc.edu/hdf4.html.
6) Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L., Wolski R: The GrADS project: Software support for high-level grid application development, *Inter. J. of High Per-*

*formance Computing Applications*, 15, pp. 327-344 (2001).
7) Institute of Global Environment and Society: Grid Analysis and Display System (GrADS), http://grads.iges.org/grads/head.html.
8) Cooperative Ocean/Atmosphere Research Data Service: COARDS Convention, http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html (1995)
9) Brian Eaton: NCAR-CSM NetCDF Conventions, http://www.cgd.ucar.edu/cms/eaton/netcdf/NCAR-CSM.html, (1997).
10) Jonathan Gregory, Bob Drach, and Simon Tett: GDT netCDF conventions for climate data, version 1.3, http://www-pcmdi.llnl.gov/drach/GDT_convention.html (1999)
11) 豊田英司, 石渡正樹, 竹広真一, 堀之内武, 林祥介: gtool4 の開発 — 地球流体データを念頭においた自己記述的かつ可搬的なデータ形式と処理系, (投稿準備中).
12) Rasch, P.: yorick, http://www.cgd.ucar.edu/cms/pjr/yorick/ (1997).
13) Research Systems: Interactive Data Language, http://www.rsinc.com/idl/.
14) Toyoda, E.: gtool4 Fortran90 tools/library http://www.gfd-dennou.org/arch/gtool4/ (2001).
15) 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 (2000).
16) GSL – The GNU Scientific Library, http://sources.redhat.com/gsl/ (2001).
17) Climate Diagnostics Center, NOAA: The NCEP/NCAR Reanalysis Project, http://www.cdc.noaa.gov/cdc/reanalysis/reanalysis.shtml.
18) Tanaka, M.: Numerical Ruby, http://www.ir.isas.ac.jp/m̃asa/ruby/ (2001)
19) Tsunesada, Y.: Ruby/GSL, http://www.ruby-lang.org/en/raa-list.rhtml?name=Ruby%2FGSL
20) 塩谷雅人他: 地球流体電脳ライブラリ, http://www.gfd-dennou.org/arch/dcl/

## 付　　録

### A.1 物理量ライブラリーの依存ライブラリー開発

本節では、当ライブラリーを開発するためにあたって必要となった、Ruby の標準ライブラリーには含まれないライブラリーの開発について述べる。詳細はホームページ[1)] を参照されたい。

全てのライブラリーは NArray[18)] という多次元数値配列のクラスを用いている。Ruby に組み込みの配列は、要素として任意のオブジェクトを取れるが、その分計算が遅くメモリー効率が悪い。また、1次元配列である。一方、NArray はCのポインターを利用して、単一の数値型の多次元配列となっており、効率が良い。NArray は非標準であるが、利用が増えており、Ruby で多次元数値配列を扱う際の事実上の標準となっている。サポートする型は整数 (1バイト符合なし, short, int)、実数 (float, double)、複素数 (float, double) である。

これまでに開発したものは、可視化ライブラリー及び NetCDF ライブラリーであるが、いずれも既存 C 言語によるライブラリーのラッパーとして実現している。さらに、現在、数値計算のための GNU Scientific Library の Ruby への移植が進んでいる[19)]。

#### A.1.1 NetCDF インターフェース

C で書かれた NetCDF の入出力ライブラリー[3)] のラッパーとして実現した。オリジナルの C のライブラリーでは、第2.3節に述べたデータセットや変数といった NetCDF 全てのオブジェクトは、整数の ID が割り当てられる。このため、変数を指定するには、データセットの ID と変数の ID の両方が必要になるという欠点がある。我々が開発した Ruby インターフェースではそのようなことがないようにし、NetCDF の構造をそのまま反映して以下の４クラスで構成した。

- NetCDF：データセット (ファイル)
- NetCDFDim：次元
- NetCDFVar：変数
- NetCDFAtt：属性

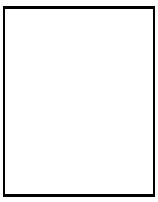
現在、本ライブラリーは、NetCDF の公式な Ruby インターフェースと位置づけられている。[3)]

#### A.1.2 可視化ライブラリー

可視化ライブラリーは地球流体電脳ライブラリ (DCL)[20)] のラッパーとして実現した。DCL は Fortran77 による描画ライブラリーで、UNIX の多くのプラットフォームと Windows で稼働する。様々な1次元、2次元の数値データ描画機能及び幾種類かの3次元描画機能を持ち、表示やレイアウトの多彩できめ細かな調整が可能である。地球・惑星流体科学での応用を念頭に、欠損値処理や地図投影などをサポートしている。

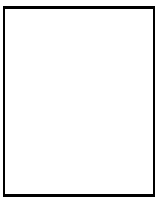
Ruby ではこれを C に変換したものを用いている。まず、DCL のほとんど全ての関数に対する1対1のラッパーを作り (モジュール RubyDCL)、その上に Ruby での利用により良く馴染む上位ライブラリー AdvancedDCL を構築した。

(平成？年？月？日受付)
(平成？年？月？日採録)



**堀之内　武**（正会員）

昭和43年生．平成9年京都大学大学院理学研究科地球惑星科学専攻博士課程修了．同年ワシントン大学大気科学科客員研究員．平成11年京都大学宙空電波科学研究センター助手．大気力学の研究に従事する．理学博士．平成10年日本気象学会山本・正野論文賞受賞．日本気象学会，アメリカ地球科学連合各会員．



**川那辺直樹**

昭和52年生．現在京都大学大学院情報学研究科通信情報システム専攻修士2年．