J. Plasma Fusion Res. Vol.84, No.4 (2008) 217-227

講座 オープンソースソフトウェアを使った実践データ解析

5．地球流体科学における Ruby の利用

堀之内　武
京都大学生存圏研究所

（原稿受付：2008年2月18日）

5.1　はじめに

今回，「オープンソースソフトウェアを使った実践データ解析」の一環として，地球流体電脳 Ruby プロジェクトの活動を紹介してほしいという依頼をいただきました．当プロジェクトは，大気や海洋などの「地球流体」の研究に，Ruby を用いるためのソフトウェアの共有や情報交換を行う，ボランティアベースのプロジェクトです．参加はメーリングリストに登録するだけなので，プロジェクトというよりはフォーラムのような緩い括りのものとなっています（ML には生物学専門で参加されている方もいます）．私自身は大気の研究をしています．大規模な数値シミュレーションには Fortran のモデルを使いますが，それ以外の数値解析，データ処理，可視化はほぼすべて Ruby で行っています．

本稿では，まず5.2節で，当プロジェクトの前史，歴史を，筆者個人の視点で語ります．実用的なヒントを早く得るためには読み飛ばしていただいて構いません．次に，5.3節で，地球流体研究における Ruby の利用の実際について述べます．その際，Ruby で流体の数値データを扱うための，我々の中核クラスライブラリ GPhys を用いたサンプルをまず紹介して，Ruby を使うことでどれくらいプログラムが短く簡潔にできうるかを示してから，それを支える多次元配列，可視化，数値計算ライブラリについて述べます．我々が扱うのは流体ですので，プラズマ・核融合の分野と共通性は高く，ほぼ同じことをする，あるいはさらに，GPhys を直接拡張して利用することも可能なケースは多いのではないかと想像しています．5.4節では，「Ruby でひろがる世界」と称して，ドキュメンテーション用のツールを紹介し，さらに，充実した Ruby のオープンソースライブラリを用いて開発された，グラフィカルユーザインタフェースプログラムや，Web ベースでデータの公開や共有にも使える解析・可視化プログラムを紹介します．

5.2　地球流体電脳 Ruby プロジェクト

地球流体電脳倶楽部と Ruby への道のり

私が大学院を過ごした1990年代，気象学分野では，大型計算機を用いた数値計算はもちろん，PC 等を用いたデータ解析・可視化においても FORTRAN が標準でした．今でも実行速度を求める数値シミュレーションコードはほとんど Fortran です．一方，データ解析・可視化には様々な言語が用いられるようになりました．

当時，可視化においては，米国では NCARG（NCAR は米国の大気科学の中核的なセンター），日本では DCL（Dennou Club Library）という FORTRAN ライブラリが広く使われていました．DCL は，京都大学の塩谷雅人氏が NCARG に満足できず作ったもので，当初は SGKS（small-GKS）という名前でした．それが，気象学に限らず地球流体分野の計算機資源をアーカイブ・公開する「地球流体電脳倶楽部」の中核ライブラリとして発展し，DCL として現在に至っています．地球流体電脳倶楽部は，MIT 留学中に UNIX とインターネットと黎明期に遭遇した東京大学（当時）の林祥介氏，京都大学の酒井敏氏らが中心になって設立されました．www.gfd-dennou.org を中心に複数のミラーサーバを運営し，計算機ソフトや様々な文書を公開しています．その経緯は，林氏が日本気象学会和文誌に寄せた[1]に詳しく述べられています．その冒頭を引用します．

> 地球流体電脳倶楽部は，大学での教育と研究のための計算情報資源の開発と蓄積を目指して協力する，気象学・海洋学を中心とする地球流体力学関係有志の集まりである．その活動はボランティア精神を基盤にしている．各々がその必要・野望のもとに出せる力を出して協力できるところを協力し，利用できるものを利用し合う，というものである．

さて，研究室に入った私は，周囲に教わって，当然のごとく FORTRAN と DCL を使うようになりました．特に疑問を持つことなく，計算にも描画にも，毎回 FORTRAN プログラムを作成，コンパイル，実行しました．そこからの最初の転機は，M2も半ばを過ぎてから，沼口敦氏の手になる大気大循環モデル AGCM5（電脳倶楽部版）を使い出したことです．AGCM5 の出力ファイルは，2次元または3次元の格子点値にヘッダーをつけたレコードが，時間に沿って並ぶもので，同氏による Gtool3 という FORTRAN ライブラリーで読み書きできるようになっていました．Gtool3 には「サンプルプログラム」として，コマンドラインで実行するプログラム群が付属しており，簡単な統計処

Practical Data Analysis Using Open Source Software 5. Ruby for Geophysical Fluid Sciences

HORINOUCHI Takeshi　　*corresponding author's e-mail: horinout@rish.kyoto-u.ac.jp*

理や描画はコマンドラインで行えるようになっていました．レイアウトを気にしなければ，空間3次元×時間の4次元データの任意の断面がコマンド一発で描けるので大変重宝しました．一方で，AGCM5の出力以外には旧来のFORTRANプログラミングで対処しましたので，「どのデータもGtool3形式だったら良いのに」と思ったものでした．しかし，Gtool3形式はそこまで対応できるほど汎用ではありませんでした．

博士課程終了後，米国で研究員をしました．データ解析・可視化には，周りの人が使っていた（日本でも聞いたことはあって興味があった）IDLという商用のプログラミングツールを使うようになりました．IDLは，型宣言が不要なインタプリタ言語で，コマンドラインでも使え，それなりに手軽でした．同時にNetCDFという，Gtool3よりも汎用なデータ形式（次節参照）を使うようになり，IDLでNetCDFデータの描画が行えるGtool3のようなライブラリを自作して使うようになりました．しかし，汎用性という面では，手続き型言語の壁にあたり，「本来ならここまで汎用にできていいのに」と思うレベルに届かせるのは簡単ではありません．また，IDL付属の描画ライブラリは，DCLよりも細部の出来が悪く，論文レベルの描画のため細かな制御を行おうと思うと，かなり経験的な試行錯誤をしなければならないことも思い知らされました．

その頃，Meyerの名著[2]を読み，かつて日本で何冊か薄い本を読んでも理解できず挫折した，「オブジェクト指向」とは何であるかとその利点が，はじめて理解できるようになりました．オブジェクト指向は，汎用で再利用性の高いソフトを開発するための技術であるということが納得でき，上記の汎用性の問題は，オブジェクト指向プログラミングで解決できるはずだということがわかりました．そして，IDLで描画ライブラリを作る傍ら，Fortran90のモジュールや総称名定義を活用して，オブジェクト指向風にNetCDFファイルを扱うライブラリ作成も行いました．

しかし，Fortran90で「オブジェクト指向する」のはそう楽ではありません．そこで，電脳倶楽部のメーリングリストで，「IDLのようにコマンドラインでも使えるオブジェクト指向言語はないかな」とぼやいたところ，「PythonとRubyというのを聞いたことがある」と塩谷氏からコメントを貰いました．当時はPythonの本しか手に入らなかったので，とりあえずPythonを触ってみることからはじめました．これはとても良さそうだと思いましたが，本格的に使う前に帰国となり，当時北海道大学にいた塩谷・林氏が，Ruby使いの数学科の大学院生（当時）の後藤謙太郎（通称「ごとけん」）氏に出会ったなどという話題もあり，今度は試しにRubyを使ってみました．1999年のことで，日本でもまだ一冊もRubyの本がなく，Web上のリファレンスマニュアルだけが頼りでした．しかし，Rubyを使い始めた途端に，良く練られていて使いやすい言語仕様に魅了されました．地球流体データの扱いに使うという面では，PythonでもRubyでも良かったと，今でも思います．私自身はRubyを愛用するに至りましたが，基本的には好みの問題だろうと思います．Pythonについては，当時既に3章で紹介されたNumRyがあり，ライブラリ面ではより有利でした．一つ，機能面でRubyの後押しとなったのは，C等で書かれたライブラリのラッパを作成して取り込むことが容易で，インタプリタの再コンパイルも不要である（Pythonでは必要）ということでした．本格的に使うには，NetCDF，DCLなど多くのライブラリをRubyに取り込まないとなりませんので．

地球流体電脳 Ruby プロジェクト

同年，林氏が科学技術振興事業団（JST）の短期集中型予算を「地球惑星流体現象を念頭においた多次元数値データの構造化」という課題で受けました．その一環として，私の乗り気やごとけん氏の存在もあって，Rubyで地球流体のデータを解析を行うための開発をしてみようということになり，地球流体電脳Rubyプロジェクトが発足しました（http://ruby.gfd-dennou.org/）．同時にRubyに限らず，地球流体のデータ解析可視化に関する研究開発活動を地球流体電脳davis（data analysis and visualization）プロジェクトと呼ぶようにもなりました．Ruby開発者のまつもとゆきひろ氏に札幌まで来ていただき，講演していただいたこともありました．同研究の成果としては，富士通エフアイピー㈱との共同開発により，DCLのRuby版を開発したということが挙げられます．JST予算は1年で終了しましたが，プロジェクトはその後も続いています．その間に，現在はRubyでの地球流体データ解析プログラミングで最も活躍している西澤誠也氏が私の出身研究室の大学院生となり，Rubyを使いはじめています．

IDLを使っているときは，毎日のように「IDLはなんてタコなんだ」と思っていました（そのせいで米国でのオフィスメートは「タコ」という日本語を覚えてしまいました）．ところが，Rubyを使ってみると，嬉しい驚きの連続で，プログラミングが大変楽しくなりました．それでも，私がIDLを卒業し，Rubyに完全に乗り換えるのは，後述するGPhysというライブラリを開発した2003年初頭にまでずれこみました．電脳Rubyプロジェクトは，それまではインフラとなる拡張ライブラリを少しづつ用意する助走期間となりました．よって，当プロジェクトの本格稼動は比較的最近のことと言えるでしょう．

圧倒的に短い行数でデータ処理，可視化プログラムが書けるGPhysを開発したことにより，地球流体分野でのRubyの利用者は少しずつ増えてきています．ただし，地球流体研究のデファクトスタンダードというところまでは程遠いです．普及活動という面では，2003年より，毎年3月にワークショップと初心者向けチュートリアルをつなげて開催しています．ただ，大学の研究室では教員や先輩が教えるというのが大きいですので，それを飛び越えて使ってみようという学生は，やはりちらほらとしか現れません．しかし，根本的な問題としては，まだ資源集積とドキュメンテーションが足りないということがあります．オブジェクト指向の利点は，再利用性・汎用性の高いソフトを作りやすいということですので，研究コミュニティのソフトウェア資産形成に大きく役立ちます．地球流体科学の将来のためにも，継続的に努力していきたいと思っています．

コミュニティのソフトウェア資産を集積するには，手軽な資源置き場があると役に立つでしょう．電脳 Ruby プロジェクトでは，ようやく，この原稿を書いている2008年冬になって念願がかなって，Wiki を用いた「小物置き場」を開設する見通しとなりました．Wiki の実装には Ruby ベースの Hiki を用いることで，Ruby 使いの管理者がカスタマイズしやすく，また投稿者にとっては，Ruby の標準的なドキュメンテーションのスタイルと親和的なサイトにすることができます．今後は，このサイトに利用者のプログラムが寄せられるようになることを期待しています．

さて，GPhys 開発後緩やかに発展していた電脳 Ruby プロジェクトにおいて，2006年より新たな展開がありました．それは，5.4節に述べる，Gfdnavi の開発です．各研究者が手持ちのデータの解析に使えるだけでなく，Web ベースのデータ提供，共有にも使えるアプリケーションです．使用する言語は，Ruby を中核に，SQL，Javascript，html に広がりました．これまで我々が行ってきたのは，地球流体の研究者による自分たちのための道具作りでしたが，本開発は情報系の研究者の協力をいただいて，一部は彼／彼女らの研究ネタになる形で共同開発を進めています．

5.3 Ruby を用いたデータ解析・可視化

本節では，地球流体のデータ解析可視化のための Ruby ライブラリを紹介します．Ruby の利点を実感していただくために，我々の中核ライブラリ GPhys のサンプルをはじめに紹介します．ついで，より基礎的な数値配列やファイル入出力について述べ，GPhys の実装に戻ります．最後に，拡張ライブラリ，数値計算ライブラリについて簡単に述べます．

GPhys というのはライブラリ名であり，その中心的なクラスの名前でもあります（クラスについては第 4 章を参照してください．クラスとは手続き型言語の型に相当しますが，自由に定義できます．その型に基づいたデータ実体をオブジェクトといいます．）．GPhys が扱うのは，離散化された，連続空間の物理量（広い意味での格子点データ）です．GPhys オブジェクトは，内部データとして，各物理量の格子点値に加えて，それぞれの座標や名前，単位などの情報を持ちます．GPhys で扱えるファイル形式は複数あり，拡張することも可能ですが，共通点としてファイル中のデータ変数に名前でアクセスできることが必要です．各格子の座標値等も，データ変数から機械的に辿れる必要があります．ただし，ここで「何々できる」とは，それができるライブラリが用意されているということですので，様々な形式に柔軟に対応できます（必要な名前はライブラリ内でつけても良いなど）．

入手・インストール

地球流体電脳 Ruby プロジェクトのライブラリは http://ruby.gfd-dennou.org から入手できます．Debian，Fedora 等の Linux ディストリビューションでは，apt，yum などのコマンドで簡単にパッケージをインストールでき，Windows 用には，Ruby 本体に加え当プロジェクトのライブラリ他様々な関連ライブラリ（GNU Scietific Library や Gnome，3 次元可視化の Vtk など多岐）が，一度の"Setup"でインストールできるパッケージが用意されています．これは，Windows用の他のRubyバイナリと共存できます．その他，Mac OS用パッケージなども用意されています．なお，以下で使用するサンプルプログラムは本講座の特設ページにも置く予定です．

GPhys の利用例

ここで用いるデータは，ftp://ftp.cdc.noaa.gov/Datasets/ncep.reanalysis.derived/pressure/air.mon.ltm.nc より入手できる，NetCDF 形式のファイルです．内容は，グローバルな気温の 3 次元分布の平均的な季節進行です．NetCDF 形式は次々小節で紹介しますので，ここではさしあたりサンプルソースの理解に必要なことに絞って書きます．このファイルでは，気温は，緯度，経度，高度（気圧面），時間（月）に関する 4 次元配列となっており，`air` と名づけられています．各座標軸の格子点値を収める一次元配列には，`lon`，`lat`，`level`，`time` という名前がついています．GPhys ライブラリがインストールされていると，次のプログラムにより描画を行うことができます．

リスト 1：global_temp.rb

```
1  require "numru/ggraph"
2  include NumRu
3  iws = ARGV[0] || 1
4  DCL.gropn(iws)
5  DCL.sgpset('lcntl', false)
6  gphys = GPhys::IO.open('air.mon.ltm.nc', 'air')
7  GGraph.contour( gphys.cut('level'=>925) )
8  DCL.grcls
```

```
% ruby global_temp.rb
 *** MESSAGE (SWDOPN) *** GRPH1 : STARTED / IWS = 1.
 *** WARNING (STSWTR) *** WORKSTATION VIEWPORT WAS MODIFIED.
 *** MESSAGE (SWPCLS) *** GRPH1 : PAGE =  1 COMPLETED.
```

結果は**図 1** のようになります．ソースの最初の 5 行は定型のおまじないです（後述）．6 行目で，`air.mon.ltm.nc` 中の変数 `air` をもとに GPhys オブジェクトを初期化し，7 行目で 925 hPa の気圧面を指定して等高線を描画し，

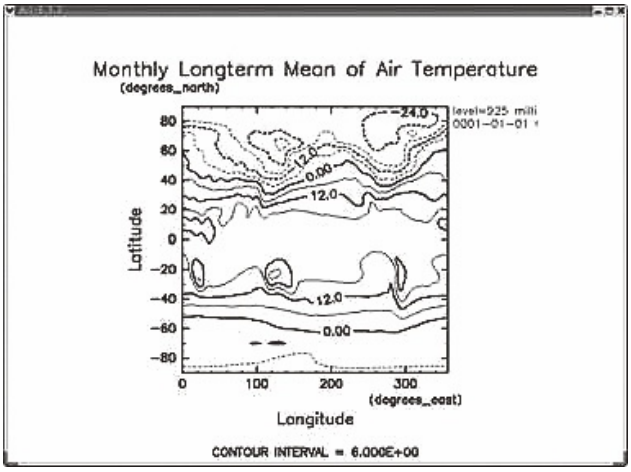

図1　global_temp.rb の実行結果（スクリーンショット）

最後に終了のおまじないを呼ぶだけのソースとなっています．しかし，図には座標軸が緯度，経度であることが示されています．これは，GPhys オブジェクトが座標情報（の読み出し方）を持っているからです．7 行目で使われている cut は，座標値ベースの切り出しや範囲絞込みを行うメソッドです（メソッドは手続き型言語の関数に相当します）．例のように次元を名前で指定できるほか，順番並び引数で名前によらず指定もできます．この例では cut(true,true,925,true) となります（true は全範囲の選択）．このように引数の仕様を柔軟にできるのは Ruby のメリットの一つです．

最初のおまじない部分を説明します．1 行目は GPhys の描画ライブラリ NumRu::GGraph を読み込むことを指定しています．我々のライブラリは，このように NumRu というモジュールにくるむことで名前空間を保護していますが，いちいち NumRu:: をつけて呼ぶのは面倒なので，2 行目の include で，外して呼べるようにします．3 行目はコマンドライン第 1 引数の値を変数 iws に代入します．or をあらわす||により，省略値を 1 にしています．4 行目はそれを使って，出力デバイスを初期化します．省略値の 1 であれば画面に，2 であれば PostScript に出力などとなります．5 行目は，ほんとのおまじないです．DCL では，アンダースコアが下付添え字を表すと解釈されますが，今回用いるファイル内の文字情報には，区切りとしてのアンダースコアが含まれるので，特殊文字解釈を抑止するというのが，その内容です．7 行目冒頭の GGraph というのが GPhys の描画モジュールです．内部では DCL を使いつつも完全には隠蔽せず，DCL の直接の呼び出しを協調的に行うようになっているのが，若干敷居が高いところです．しかし，DCL が持つ多彩な機能を余すこと無く使えるため，論文掲載に耐える図が作れます．

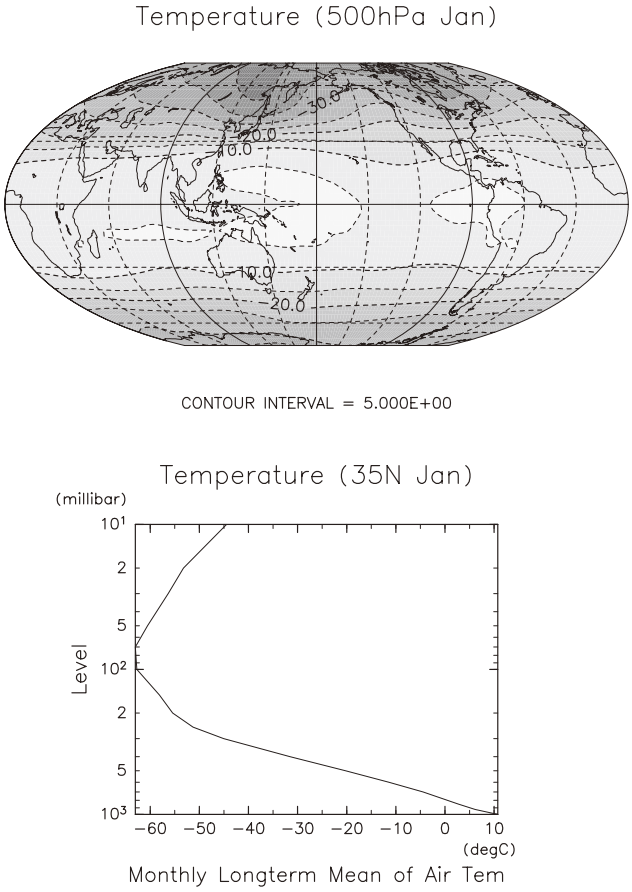

図2　global_temp2.rb の実行結果（PostScript）．

さて，試行錯誤したい場合は，Ruby の対話的インタプリタ irb を使うのが便利です．以下では，最初の 5 行のおまじないを ggraph_startup.rb というファイルに収録して省略できるようにした場合の例を示します．下線を引いたところがユーザの入力です．

```
% irb -r ggraph_startup.rb
*** MESSAGE (SWDOPN) *** GRPH1 : STARTED / IWS = 1.
irb(main):001:0> gphys = GPhys::IO.open('air.mon.ltm.nc', 'air')
=> <GPhys grid=<4D grid <axis pos=<'lon' in 'air.mon.ltm.nc' sfloat[144]>>
    <axis pos=<'lat' in 'air.mon.ltm.nc' sfloat[73]>>
    <axis pos=<'level' in 'air.mon.ltm.nc' sfloat[17]>>
    <axis pos=<'time' in 'air.mon.ltm.nc' float[12]>>>
 data=<'air' in 'air.mon.ltm.nc' sfloat[144, 73, 17, 12]>>
irb(main):002:0> GGraph.contour( gphys.cut('level'=>925) )
*** WARNING (STSWTR) *** WORKSTATION VIEWPORT WAS MODIFIED.
=> nil
irb(main):003:0>   ←次の入力のプロンプト（セッションは続く）
```

上で=>に続くのは，irb が各行の戻り値を「お任せ」で文字列化したものです(ただし表示法は再定義できます)．1 行目に対する出力より，開いたのが lon，lat，level，time の 4 次元データであることなどがわかります．

同じデータを使った描画サンプルをもう一つ載せます．次のプログラムを実行すると**図 2** が得られます．今回は引数で PostScript を指定した結果を示しました．上図は水平断面を地図投影で，下図は北緯35°で経度方向に平均した鉛直プロファイルを示します．縦軸の気圧は，対数をとるとほぼ高度に線形に対応しますので，対数スケーリングで表示しています．本ソースでは指定パラメタが増えていますが，GGraph は，段階的にパラメタを増やしていくことで凝った描画ができるよう設計してあります（凝ろうとすると急に複雑にならないよう)．

リスト 2 ：global_temp2.rb

```
require "numru/ggraph"
include NumRu
iws = ARGV[0] || 1
DCL.swpset('iwidth',550)        # 画面サイズ（横）
DCL.swpset('iheight',700)       # 画面サイズ（縦）
DCL.sgscmn(5)                   # 白黒カラーマップ
DCL.gropn(iws)
DCL.sldiv('y',1,2)              # 画面を 1 × 2 に分割
DCL.sgpset('isub', 96)          # __ が下付添字にならぬよう制御文字を､に変更
DCL.sgpset('lfull', true)       # viewport を 1 × 1 の正方形とせず画面に合わす
DCL.uzfact(0.8)                 # 文字サイズを 0.8 倍に
gphys = GPhys::IO.open('air.mon.ltm.nc', 'air')
# <上図>
GGraph.set_fig('itr'=>12, 'viewport'=>[0.07,0.93,0.1,0.55])
GGraph.set_map('coast_world'=>true, 'grid'=>true)
GGraph.tone( gphys.cut('level'=>500), true,
             'interval'=>5, 'clr_min'=>50,  # 色範囲を変えて濃い色を避ける
             'title'=>'Temperature (Jan, 500hPa)', 'annotate'=>false )
GGraph.contour( gphys.cut('level'=>500), false, 'interval'=>5 )
#GGraph.color_bar   # カラーバー（今回は非表示）
# <下図>
GGraph.set_fig('itr'=>2, 'viewport'=>[0.25,0.75,0.12,0.52])
GGraph.line( gphys.cut('lat'=>35).mean('lon'), true,
             'exchange'=>true, 'title'=>'Temperature (35N Jan)',
             'annotate'=>false )
DCL.grcls
```

今度は，数値計算プログラムを示します．4 次元以上のデータ中の第 1，4 次元についての 2 次元のパワースペクトルを計算し，NetCDF ファイルに出力します．Ruby の配列添字が 0 からはじまるのに合わせて，GPhys では次元を 0 から数えることにしていますので，それぞれ 0，3 と表されます．

リスト 3 ： pw2D.rb

```
require "numru/gphys"
include NumRu
def usage
 "\n USAGE: % ruby #{$0} filename variablename¥n"
end
filename = ARGV[0] or raise(usage)     # 第 1 引数：入力ファイル名
varname = ARGV[1] or raise(usage)      # 第 2 引数：FFT を掛ける変数名
GPhys::fft_ignore_missing(true, 0.0)
      # データ欠損は 0.0 で埋めることに（実際には欠損がない場合用）
gphys = GPhys::IO.open(filename, varname)
pre = gphys.detrend(3).cos_taper(3)
      # 4 次元目について，線形トレンド除去&cosine テーパリング
sp = pre.fft(false,0,3).abs ** 2
```

```
14        # false は正変換（true は逆変換）．0,3==>第１，４次元で２次元 fft
15  power = sp.rawspect2powerspect(0,3).spect_one_sided(3).spect_zero_centering(0)
16        # パワースペクトルに単位と値を変換．時間について 1-sided に etc
17  outfile = NetCDF.create("pw2D_#{varname}.nc")    # 出力ファイル名（変数名入り）
18  GPhys::IO.write(outfile, power)                  # 座標変数を含む全体を書出し
19  outfile.close
```

本プログラムは，４行目から読み取れるように，ファイル名と変数名をコマンドライン引数で与えます．ともに必須ですので，６，７行目では，引数がない場合は「例外」(エラー)を発生して，使用法を表示して終了するようにしています（演算子 or は||と同じだが，=より優先度が低い)．11行目では，第４次元目についてトレンド除去などの前処理を行います．上記のような全球４次元データであれば，第１次元目は経度でサイクリックなため，このような前処理は不要なので省略しています．そのような知識に依存せず，より汎用なプログラムにするには，FFT を掛ける次元や，それぞれについての前処理の有無を，引数で指定するようにすれば良いでしょう．さて，13行目では，FFT を掛け，絶対値を取って２乗します．メソッド fft は結果を GPhys で返しますが，座標軸も正しく波数軸にして返すようになっています．単位も（ついていれば）正しく演算されて，もとが m なら m^{-1}などとなります(基本単位に分解して演算できるライブラリも当プロジェクトで作っています)．その後15行目で生スペクトルからパワースペクトルに変換（定数倍）できるのも，このように座標情報があるからです．15行目ではさらに，４次元目について one-sided にするなど，細かな後処理も行っています．最後にファイルに出力します．データの書き出し命令は18行目だけですが，power という変数が表す GPhys オブジェクトには座標軸が含まれますので，自動的に出力されるようになっています．この例では，第１，４次元目は波数軸，それ以外は元の軸となります．なお，FFT 計算には FFTW3（http://www.fftw.org/）を用いており，任意長で計算できます．

スペクトル解析を実践する上で面倒なのは，FFT そのものよりも，上で示したような前処理，後処理や，波数軸も正しく出力するといったことでしょう．GPhys のように座標軸も一まとめにすることで，これらの処理を自動化し，隠蔽できることがおわかりと思います．また，出力は，元データ同様に座標軸を含む NetCDF ファイルですので，**リスト１**とまったく同様に簡単に描画できます．第４章では，クラスを使う利点とメソッドチェーンの便利さが述べられましたが，ここでも発揮されていることがおわかりでしょう．

GPhys に最初から組み込まれている数値計算メソッドは限られています．しかし，Ruby では既存のクラスにメソッドを自由に追加できます．次の例では，moment.rb というファイルに，平均の回りの n 次のモーメント計算する moment というメソッドを定義します．最初に numru/gphys を require していることで，追加定義になります．10行目以降は，ruby moment.rb という形で呼んだときのみ実行される，テスト用のプログラムです．このファイルをライブラリとして他のファイルで利用するには，require "moment"と記述します（.rb は省略する)．

リスト４：moment.rb（メソッド追加例）

```
1   require "numru/gphys"
2   module NumRu
3    class GPhys
4     # 平均の回りの n 次モーメント
5     def moment(n, *dim)
6       ( ( self - self.mean(*dim) )**n ).mean(*dim)
7     end
8    end
9   end
10  ###### test program #####
11  if $0 == __FILE__
12   include NumRu
13   raise('need 3 or more args') if ARGV.length<3
14   fnm, vnm, n, = ARGV
15   dims = ARGV[3..-1].collect{|s| s.to_i}
16   gphys = GPhys::IO.open(fnm,vnm)
17   mmt = gphys.moment(n.to_i, *dims)
18   p mmt
19  end
```

5行目では，2番目の引数dimsの先頭に*がついています．これにより，メソッド moment の引数は一個以上の任意個となります：最初の引数が n に渡され，2番目以降の引数は配列にまとめられて dims に与えられます（一つしか引数がない場合，長さゼロの配列になる）．一方，メソッド呼び出しにおいては，6行目のように，配列に*を冠して与えると，Rubyインタプリタによって展開され，全要素が引数の並びとなってメソッドに渡されます．この仕組みにより，可変個の引数を簡単に引き継ぐことができます．

メソッド moment における「平均」とは，2番目以降の引数で次元を指定していれば，それらに沿ったものとなります．例えば gphys.moment(2, 0) と呼べば，第1次元（次元番号0）に沿った平均を引いた2次モーメント，つまり第1次元に沿った分散になります．6行目の self - self.mean(*dim) の部分は，次元数の異なるデータ同士の引き算となりますが，GPhys では次元に名前がついているため，対応を一意に解決して期待されるとおりの演算を行います．なお，引数を一つだけ（つまり n のみ）与えた場合は，全データ点に関する単純平均が使われます．

10行目以降のテスト部分も，簡潔ながら一定の汎用性を確保しており，

```
% ruby moment.rb filename varname n [dims..]
```

という形で呼び出せます．ただし，出力はファイルでなくおまかせで標準出力に出すだけです（18行目）．

数値配列

ここからは，数値データの取り扱いにかかる要素ライブラリを見ていくことにします．拙稿[3]もご参照ください．

数値計算に多次元配列はつきものです．Ruby 標準の配列 Array クラスは，第4章で紹介されたように高機能ですが，数値計算には必ずしも便利でありません．その観点での欠点として，1次元なので多次元にはネストなどの工夫が必要，すべての要素を Ruby オブジェクトにするので要素数が膨大になるときは遅い，演算子+が加算でなく配列の連結を意味するなど配列単位の数値演算が想定されていないということが挙げられます．これらを解決するのが，非標準ながら Ruby における多次元数値配列のデファクトスタンダードである NArray（http://narray.rubyforge.org/）です．NArray は，浮動小数点（単精度，倍精度），整数（1，2，4バイト）などに揃った型を持つ要素を納める，多次元配列のクラスです．添字の順番は Fortran と同じコラムメジャーです．データは C のポインターを使ってメモリ上に連続した領域に納められますので，高速です．機能的は Fortran90 の配列とほぼ同等です．Ruby では，角括弧 [] をメソッドして自由に定義できますので，NArray ではこれを要素へのアクセスやサブセット取り出しに使います．標準の Array 同様，添字は0からはじまり，負の要素で後ろからの位置も指定できます（－1，－2はそれぞれ最後，最後から2番目の要素を指します）．以下に NArray の利用例を載せます（そのまま irb に入力すれば確認できます．）

NArray の利用例

```
require "narray"
include NMath                          # NArray 対応 Math ライブラリ
na = NArray.float(4,3,2).random!       # 4 × 3 × 2 の配列を作り乱数代入
nb = na[true,1..-1,0]                  # 各次元につき［全選択，2番目～最後，1番目］の切出
nc = sin(nb)**2 * cos(nb)              # 要素毎の演算
nd = na.mean(0,1)                      # 0，1次元目に沿った平均
na[0,false] = na[-1,false]             # 最初の次元の最後の要素を最初の要素に代入
                                       # （false は任意個の次元の全選択）

m = NMatrix[ [1.0,2.0], [3.0, 4.0] ] # 2 × 2 の行列 M
y = NVector[ 1.0, 1.0 ]                # 長さ2のベクトル y
x = m.lu.solve(y)                      # LU 分解して Mx = y を解く
```

データファイルの形式と扱い

地球流体分野で扱うデータは主に数値データです．しかし，多くの分野でそうでしょうが，単純な生のスカラ値で済むことは稀で，複数の物理量や計測値が，名前や単位，座標（時間，空間 etc）に関する情報と合わせて提示されることではじめて完結することが普通です．もしも，データの作成者が各々自己流でデータ構造やファイル形式を定義すると，流儀には際限がありませんので，他人から貰ったデータを満足に扱うためには，まず，データを計算機に読込ませるだけで大仕事となりがちです．これでは，データ交換に支障がありますので，我々の分野では，データ形式の標準化が進められています．ただし，一つの形式に収斂はしておらず，主に次のような形式が用いられています．

- NetCDF：米国の UCAR という機関で開発されている形式です．内部構造は隠蔽され，多言語で用意された専用 API を通じて読み書きします．ファイル中に存在するのは，多次元（ゼロ次元スカラも含む）の「変数」，変数の各次元の長さを表す「次元」，　変数やファイルのメタデータを格納する，名前と値の組からなる「属性」です．座標軸をどう表現するか，「属性」で物理量の単位やデータ欠損をどう表現するかの規約が存在するため，ファイルが機械的に解釈できる「自己記述」的なデータ形式となっています．大学・研究機関で広く

用いられています．

・GRIB：世界気象機関が定める，気象予報機関のデータ交換やデータ公開のための形式です．各レコードが水平 2 次元断面を納め，時系列や高度を含む 3，4 次元データはその集合体として表現されます．座標系や座標軸，物理量の種類，圧縮等に関するフラグが詳細に定められており，気象予報に関わるデータを広くカバーします．

・HDF/HDF5：NetCDF と似ていますが，より複雑なデータ構造をカバーします．反面，統一的な「自己記述」性には劣ります（NetCDF 的な標準もあるがあまり使われていません．利用者がそれぞれの仕方で自己記述的にはできますが）．人工衛星データの提供で使われることが多いです．最近の衛星データ提供では，自己記述性を高める標準化を行った規約 HDF-EOS が用いられることが増えてきました．

・GrADS：バイナリデータに簡単なヘッダーを付したものです．経度 - 緯度 - 高度 - 時間のデータに限られます（高度軸，時間軸がないのは可）．

多次元データは一般に量が大きいので，以上いずれもバイナリです．一方，テキストで表される比較的小さな計測データについては標準がなく，未開の荒野になっています．

上記のバイナリ形式は，互いに大きく異なり互換性がありませんが，データを構成する論理的な要素には共通性があります．それは，何らかの多次元空間をサンプリングした物理・化学量に関する数値データであり，名前や単位，座標を持つということです．そして，形式は違えど，それらの解釈を自動化できるということも共通です．ただし，HDF に関しては，HDF を使うというだけでは機械的解釈性が保証されないので，汎用には扱えませんが．なお，宇宙プラズマのシミュレーションにおいては，HDFは広く用いられているようです．

さて，オブジェクト指向言語を用いれば，上で述べたような論理的，あるいは概念的な共通性を活かして，データ形式・構造の違いを乗り越えて，「同じことをするプログラムは同じに書ける」ようにできます（同じように書けるではなく，同じに書けるです）．GPhys は，それを体現したものとなっています．

GPhys の実装

GPhys オブジェクトは**図 3** のような構成を取っています．主となるデータは一般に多次元で，その格子の各次元の「軸」は一般に一次元です．標準的には，各軸は何らかの座標に対応しますが，そうでない場合も扱えるようになっています．主データも座標データも，VArray というクラスのオブジェクトで表されます．VArray は Virtual Array の略であり，「事実上」数値の配列と見なせるものを包括的に扱います．ただし，それぞれに名前（上の例なら air など）がつき，NetCDF 同様にキーワード属性を持てるというところが異なります．VArray オブジェクトにおけるデータ実体は，NArray，名前の文字列，キーワード属性

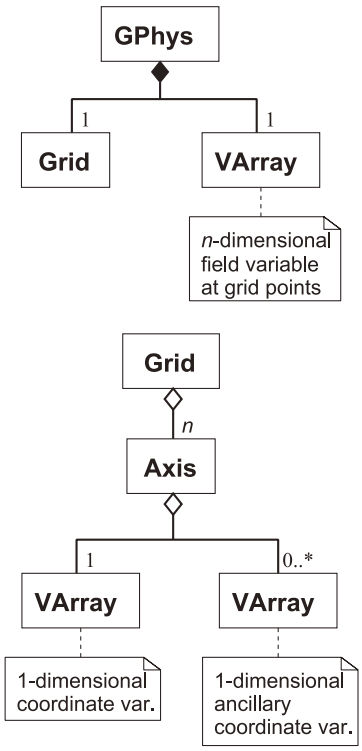

図 3　GPhys オブジェクトの構成概略．

（実装は連想配列ベース）で構成できます．一方，NetCDF などのファイル中の多次元配列を直接指し示すものでもあり得ます（その場合数値データはメモリー上には持ちません）．このため，ファイル中の変数も，それを読み込んで演算した結果（メモリ上のみに存在する）も，同じ枠組みで扱われます．さらに，VArray は，他の VArray のサブセットへのマッピングでもあり得ます．**リスト 1** で使用したデータ切り出し命令 cut は，サブセットマッピングを作るだけです．よって，その時点でデータ読込みは発生しません．データ読み込みを，実際に必要になったとき（可視化する，数値演算するなど）まで遅延するというのは，GPhys の大きな設計方針です．例は示しませんが，処理を自動的に分割するイテレータなども備えており，大規模なデータの処理に耐えるように作られています．

GPhys には座標の概念があるものの，基本的には多次元配列です．このため，四則や数学演算，サブセット切り出し（[] メソッド）などが定義されています．API の仕様は NArray と統一してありますので，NArray の演算に使うソースコードは，多くがそのまま GPhys でも使えます．

GPhys は複数のファイル形式に対応します．例えば，リスト 1 の 6 行目の `GPhys::IO.open` という GPhys のコンストラクタが行うことは，次のようになります．

・ファイル形式を判別し，それぞれのファイル形式用の入出力モジュールの open メソッドに処理を委譲する．

・各モジュールの open メソッドは，それぞれの形式の流儀に則って，ファイルの内容から**図 3** の構造を構成する．ただし，数値データは読み込まず，ファイル中

の変数へのポインターだけを確保するのが標準の動作である（読み込むことも可）.

このように構成されているため，データの概念的な構成がGPhysのデータモデルに合致する限り，ファイル形式に関わらず扱えます．例えば，第3章の例で用いられた時系列のテキスト形式のデータを当てはめることも可能でしょう．

拡張ライブラリについて

ここで，CやFortranのライブラリを，Rubyから利用するための「ラッパ」(wrapper) を作成して，「拡張ライブラリ」として利用することについて述べます．あわせて，数値計算に関する代表的な拡張ライブラリを紹介します．

RubyはCで実装されています（Javaでの実装もありますがここでは扱いません）．ArrayやHashなどのRubyの組み込みクラスはCで書かれており，標準添付ライブラリの多くもCで書かれています．Rubyの拡張ライブラリの作者は，Cで書かれた組み込みクラスがRubyで利用できるようになっている仕組みを，そのまま利用できます．このため，Ruby本体と同じ移植性を持つ拡張ライブラリを容易に作成することができるのです．拡張ライブラリを導入する際，インタプリタ本体の再コンパイル必要はありません．Rubyの実行速度は，Cに比べればはるかに遅いですが，処理のボトルネックは拡張ライブラリにすることで改善できます．上述のNArrayはCで書かれた拡張ライブラリです．

既存のライブラリをRubyから利用するラッパをつくる場合，比較的小さなライブラリであれば，直接コーディングしやすいですが，大規模なライブラリの場合は，SWIGというラッパの自動生成ツールが役立ちます．SWIGではC++ライブラリのラッパも作成できます．

科学計算に有用な拡張ライブラリとして，GNU Scientific Library（GSL）という巨大な数値計算ライブラリをRubyから使えるようにしたRuby/GSLが挙げられます．GSLは，データ保持に独自仕様の構造体を多用します．Ruby/GSLでは，それぞれに対応するクラスが定義されているのに加え，多くがNArrayに変換できるため，他のライブラリと容易に組み合わせて使えます．GSL以外にもCで書かれた数値計算ライブラリのラッパが多く公開されています．

Rubyでは，Fortranで書かれたライブラリのラッパを作ることもできます．ラッパ自体はCで書く必要があるため，Cとの互換性が明白な，FORTRAN77でも使えるデータ構造が，直接のやり取りの対象になります（ライブラリ内部でFortran90の構造体等を使うことは可能です）．ただし，Fortranで書いた拡張ライブラリの移植性の確保は，次の理由により，簡単ではありません．Fortranコンパイラでは，実行ファイル生成時に標準ライブラリへのリンクを行います．ところが，Rubyラッパ生成には，リンカとしてCコンパイラを用いますので，Fortranコンパイラがリンクする標準ライブラリを陽に教える必要があります．これを一般的に行うことは難しいので，コンパイラを限定した形で作ることになるでしょう．ただし，ソースレベルでCに変換すれば，移植性は確保できます．DCLのRuby版は，FORTRANソースをCに変換したものを使うことで移植性を確保しています．

5.4 Rubyでひろがる世界

世界中で，Rubyのための様々なライブラリが開発され，オープンソースで公開されいて自由に利用できます．このため，グラフィカルユーザインタフェース (GUI) を持つプログラムや，Webベースの高度なサービスなどを実現できます．また，ソースのドキュメンテーションにおいても，役に立つツールがあります．

RDocによるドキュメント生成

Rubyには，RDocというドキュメンテーション生成ツールが標準添付されています．RDocは，Rubyのソースコードを解釈し，プログラム要素間にリンクが張りめぐらされたドキュメントを生成します．ドキュメントの形式は，htmlのほか，riというコマンドラインツールで表示できる形式が選べます．RDocは，ソース中の各要素（クラス，メソッドなど）の直前に書かれたコメントを，それぞれのドキュメンテーションと解釈します．RDocのコメント文の書き方は直感的で分かりやすく，簡単に見出し文や，箇条書き，コード引用などが行えます．例として，本稿で紹介した4つのソースコードに，次のようにRDocを掛けた結果を，**図4**に示します．

```
% rdoc --charset sjis --inline-source global_
temp.rb global_temp2.rb moment.rb pw2D.rb
```

GUIツール，3次元描画ライブラリ

Rubyでは，Tk，Gnome (Gtk)，Foxなど，様々なGUIツールが利用可能です．これらをDCLなどの描画ライブラ

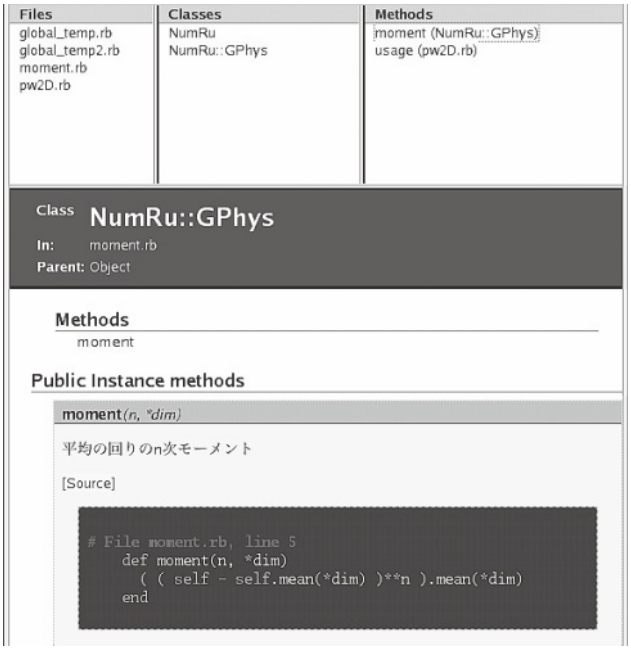

図4　本稿で紹介した4つのソースコードを元にRDocで生成したhtmlをブラウズした例.

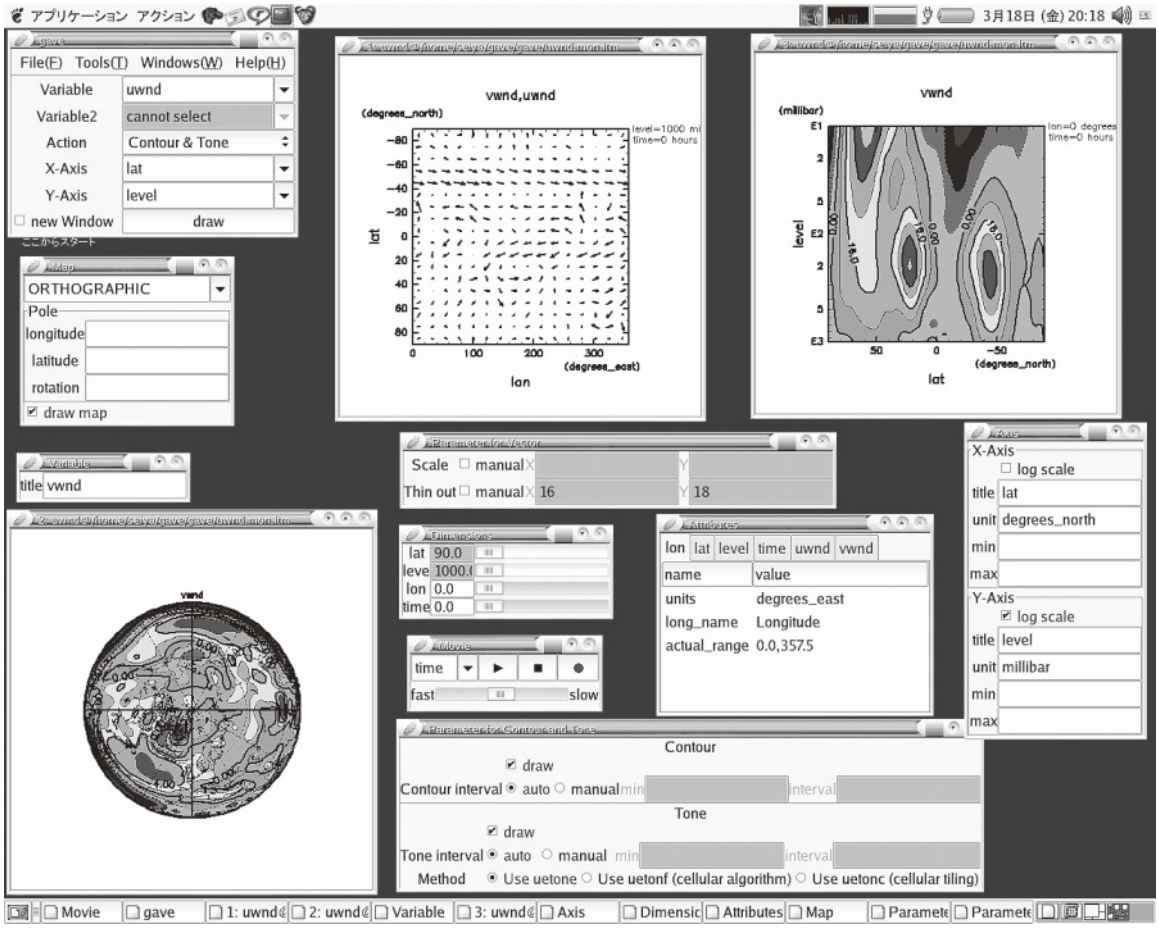

図5　GUI ツール gave のスクリーンショット．
http://ruby.gfd-dennou.org/products/gave/gave.png を白黒に変換．

リと組み合わせて使うことで，マウス操作で描画できるツールが作れます．**図5**は，京都大学の西澤誠也氏が作ったGtkベースのGUIツールGaveのスクリーンショットです．Linuxはもちろん，Windowsでも動作します．データアクセスにはGPhysを用いていますので，複数のファイル形式に対応します．Gaveは，多次元のデータを，様々な断面表示やアニメーションにより，初期的に「把握する」のに役に立ちます．また，GUIで行った描画を再現するRubyプログラムを生成することもできます．

西澤氏は，三次元描画ライブラリVtkのRubyラッパも開発しています．三次元可視化の場合は，視点の移動などを自由に行うためにGUIツールと組み合わせることが欠かせませんが，TkやGtkなどで実装できます．

Webベースのデータベース・解析・可視化ツールGfdnavi

2006年より，筆者，西澤氏および，お茶の水大の渡辺知恵美氏(専門は情報学，データベース)を中心とするグループで，新しいツールGfdnaviを開発しています（http://www.gfd-dennou.org/arch/davis/gfdnavi/）[4]．これは，上述のGPhysと，Ruby on Rails(http://www.rubyonrails.org；以下Rails）という開発フレームワークを用いて作成した，Webベースのアプリケーションです．Gfdnaviは，地球流体データのデータベースを作り，Webブラウザで検索，解析，可視化できるようにします．

Railsは，Rubyの柔軟性を最大限に活かし，短いコードで素早くWebアプリケーションが開発できる驚異的なツールで，今や，世界でRuby利用者が増える原動力となっています．Ruby on Railsでは，データを関係データベース（RDB）に保存します．Mysql，PostgreSQL，DB 2など，商用，非商用を問わず，多くのRDBマネジメントシステム(RDBMS)が使えます．また，Railsを使うと，アプリケーションに専用Webサーバが導入されますので，Webサーバを運用しない個人のPCでも，手軽にWebベースのアプリケーションが動かせます．もちろん，Apacheなどの標準的なWebサーバでも動作します．そのおかげで，Gfdnaviは，個人が自分のデータの管理・解析・可視化に用いるのに適する一方で，常時運用されるWebサーバ上でデータ公開を行うのにも適したものとなっています(人工衛星による降水観測データ公開などに使われ始めています)．Railsには，Google Mapで使われて一躍有名になった，非同期http通信Ajaxを，Rubyから容易に使えるようにしたライブラリも含んでいます．Gfdnaviはこれを多用しており，応答待ちで操作が中断されないようになっています．

図6（上）にGfdnaviの構成概略を示します．利用者は，Webブラウザにより Gfdnaviサーバにアクセスします．データ本体はディスク上のNetCDFファイル等ですが，その内容に関する情報（メタデータ）はRDBに登録されていますので，検索や内容ブラウジングへの対応はRDBベース

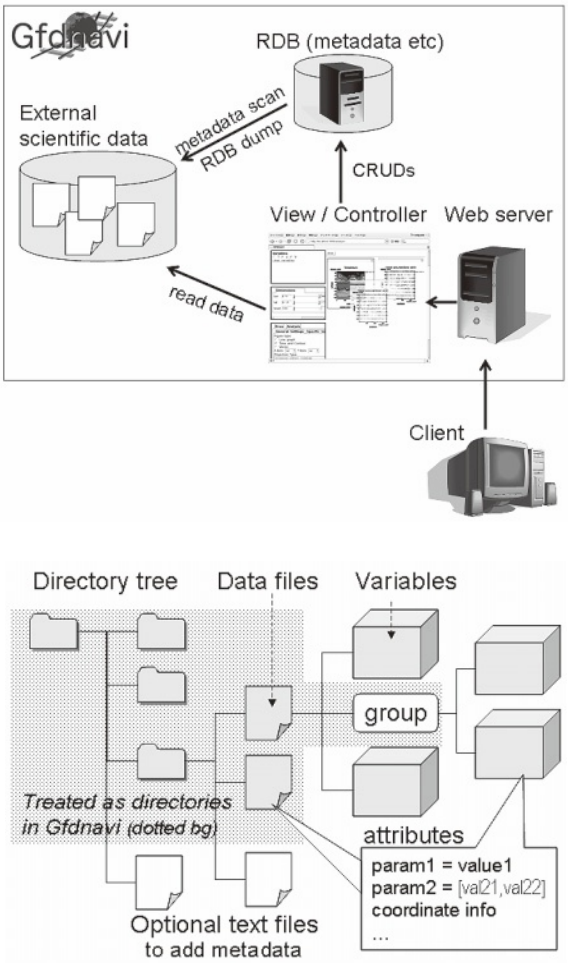

図 6　上：Gfdnavi の構成概略とアクセスの流れ．下：メタデータデータベースのもとになるディレクトリ階層の模式図．

で行い，データ選択後の解析・可視化操作への応答は，GPhysを使って直接データファイルを対象に行うことになります．

Gfdnavi のメタデータ登録は，自動登録スクリプトによって行われます．同スクリプトは，指定されたトップディレクトリ以下のすべてのデータファイル中のデータを，ディレクトリ構造も含めてデータベース化します（図 6 下）．これも GPhys を使いますので，複数のデータ形式を統一的にサポートします．ディレクトリ構造がデータベース化されているため，Gfdnavi のデータ選択インタフェースでは，MS Explorer 風のディレクトリツリーがサポートされています．そのほか，テキストや時空間の範囲などからの検索も行えるようになっています．

データ選択後は，簡単な数学統計処理や可視化を行うことができます．それを元に，手元で描画などを改良できるよう，ブラウザで行った処理を再現するスクリプトと，そのために必要なデータを最小限に切り出したものがダウンロードできるようになっています．また，行った可視化をサーバ上で再現するパラメタ付の URL の取得もできます．こちらは，研究者間での情報交換に役立ちます．Gfdnavi には，ユーザーの概念もあり，登録ユーザーはデータ処理のための Ruby メソッドを登録することができます．また，処理結果をデータベースに保存できます．他にも様々な機能があります．

現在，データから見いだした知見を文書化し，データ解析手法とともにデータベース化できるようにする開発も行っています．また，複数の Gfdnavi サーバを横断的に検索・利用できるようにするための研究・開発も進行中です．

以上のようなアプリケーションを，ソフト開発を本務としない者が少人数で開発するのは，一般には難しいでしょう．Ruby という生産性の高い言語と，Rails という優れたツールがあってこそのことと思います．

5.5　おわりに

本稿では，地球流体科学における Ruby の利用ということで，地球流体電脳倶楽部というフォーラムを舞台に開発・公開されている，データ解析・可視化用のライブラリやアプリケーションを紹介しました．何らかの参考になりましたら，そしてあわよくば，さらなる情報交換や開発協力等に発展させられたら望外の喜びです．

参 考 文 献

［1］林　祥介：天気 **42**, 548 (1995).
［2］B. Meyer, *Object-oriented software construction* 2nd ed. (Prentice Hall, 1997).
［3］堀之内　武：数値計算と可視化，Ruby Library Report (2005), http://jp.rubyist.net/magazine/?0006-RLR.
［4］堀之内　武，西澤誠也，渡辺知恵美，森川靖大，神代剛，石渡正樹，林　祥介，塩谷雅人：電子情報通信学会第18回データ工学ワークショップ論文集，D2-8 (2007).