Rubyによる地球流体データの解析可視化ライブラリ

堀之内武（北大地球環境）

電脳davisプロジェクト

- data analysis and visualization
- 地球流体電脳倶楽部のソフト開発集積活動の一。
- この言葉を使いだしたのは90年代末頃だったような。
  - それ以前からDCL: Fortran用グラフィックライブラリ(GKS) by 塩谷雅人他

99年：JST 計算科学技術活用型特定研究開発推進事業

「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」（林祥介代表）

- 現在のdavis活動の基礎構築に貢献
- 新しいGtoolに向けたデータ構造の設計，試作 → Gtool4 NetCDF規約（メタデータ）
- オブジェクト指向スクリプト言語(Ruby/Python)の検討と試作 → RubyDCL

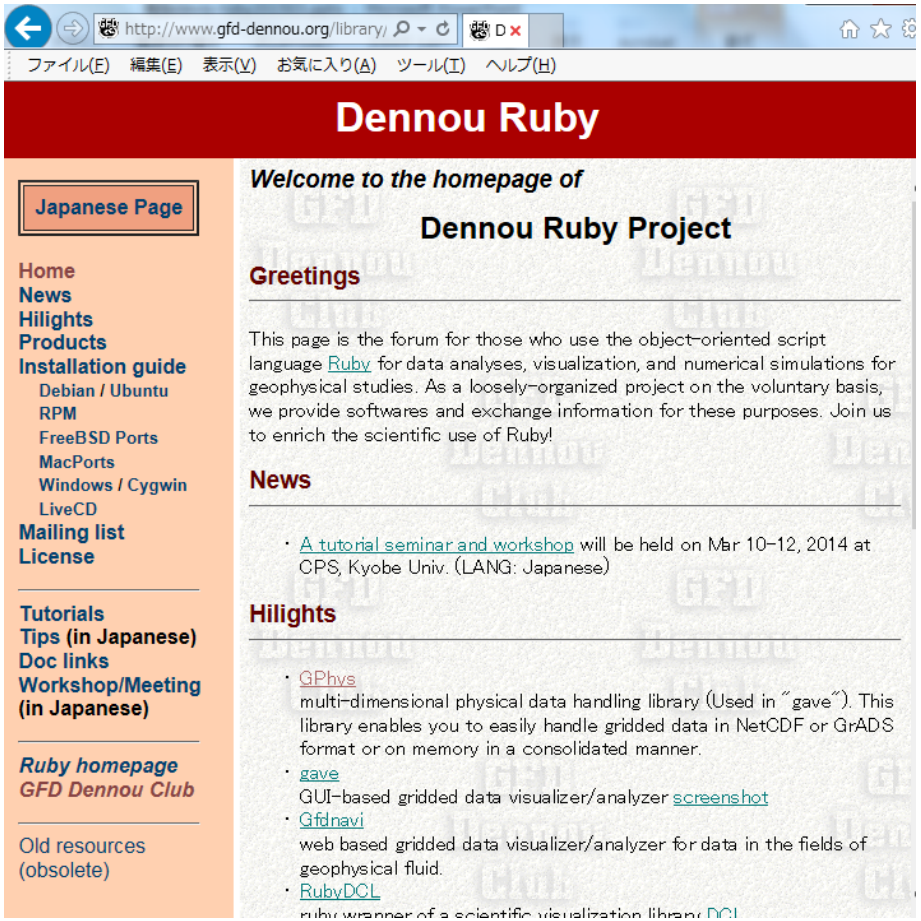
その後の展開：電脳Rubyソフト群（本日の話題）

なぜ Ruby？

- **基本思想：データ解析ではCPU時間より人間時間が貴重**
- 型なし、スクリプト言語（インタープリター）
  ⇒　素早くプログラムが開発できる
- 洗練され使いやすいオブジェクト指向言語　⇒　開発・保守効率がよく、汎用なソフトを作り易い　⇒　コミュニティーでツールを共有
- 対話的に利用可能　⇒　試行錯誤に良い
- 拡張性が高い ⇒ C（やFortran）のライブラリーの有効利用
- 増え続けるライブラリー（ネットワーク関連 / GUI / データベース等々）　⇒　高度なサービスを実現しやすい。
- 文字処理が容易（データ解析中に文字処理が必要になることは多い）
- ゴミ集め、例外処理等の近代的支援機能あり

電脳Ruby「プロジェクト」

- 地球流体研究のためのRuby用ライブラリを開発
- 成果はオープンソースで公開(BSD 2 clauseライセンス)
  - 電脳サーバーで（電脳davisサーバーに「小物置き場も」）



我々が扱うデータ→…を基にIOを考える

- 離散的(自由度有限:計算機上の必然)
- 多次元(時間, 空間, 波数空間…) (incl. 0次元)
  - 座標の存在(一次元以上で)
- 単位や名前があるのが普通

- データファイルの形式はいろいろ:NetCDF, GRIB, GrADS, テキスト, etc. etc.
  - 構造いろいろ, 読み方いろいろ。
  - 伝統的アプローチだと, 初めてのデータは読めるようになるだけで一苦労。

→データの論理構造の共通性に基づいてクラスライブラリを作成し, ここのデータ形式対応は下層にカプセル化

NetCDF:「自己記述型」データ形式

- 気象(地球流体)業界で広く使われる形式の一。バイナリ。(Ver.3は独自, Ver.4:HDFベース)
- バイナリ構造は隠蔽されていて，APIを通じて名前ベースでアクセス。(Ver.4 APIは後方互換)
- ユーザーズガイドによる規約が広く守られている。
  - 単位や名前を表す属性名
  - 座標を表す変数への辿り方

  ⇒ 物理量を表す変数名から芋づる式に辿れる:「自己記述」
- ユーザーズガイド規約の上に，より詳細なメタデータ規約も:CF, Gtool4,...(多くは互いの親和性大)

NetCDFファイルの構成例
（テキストダンプツールによる）

NCEP再解析（客観解析）の気温データを収めるNetCDFファイルの「ヘッダ」（メタデータ）の内容。

ダンプツール ncdump で表示後、一部省略。

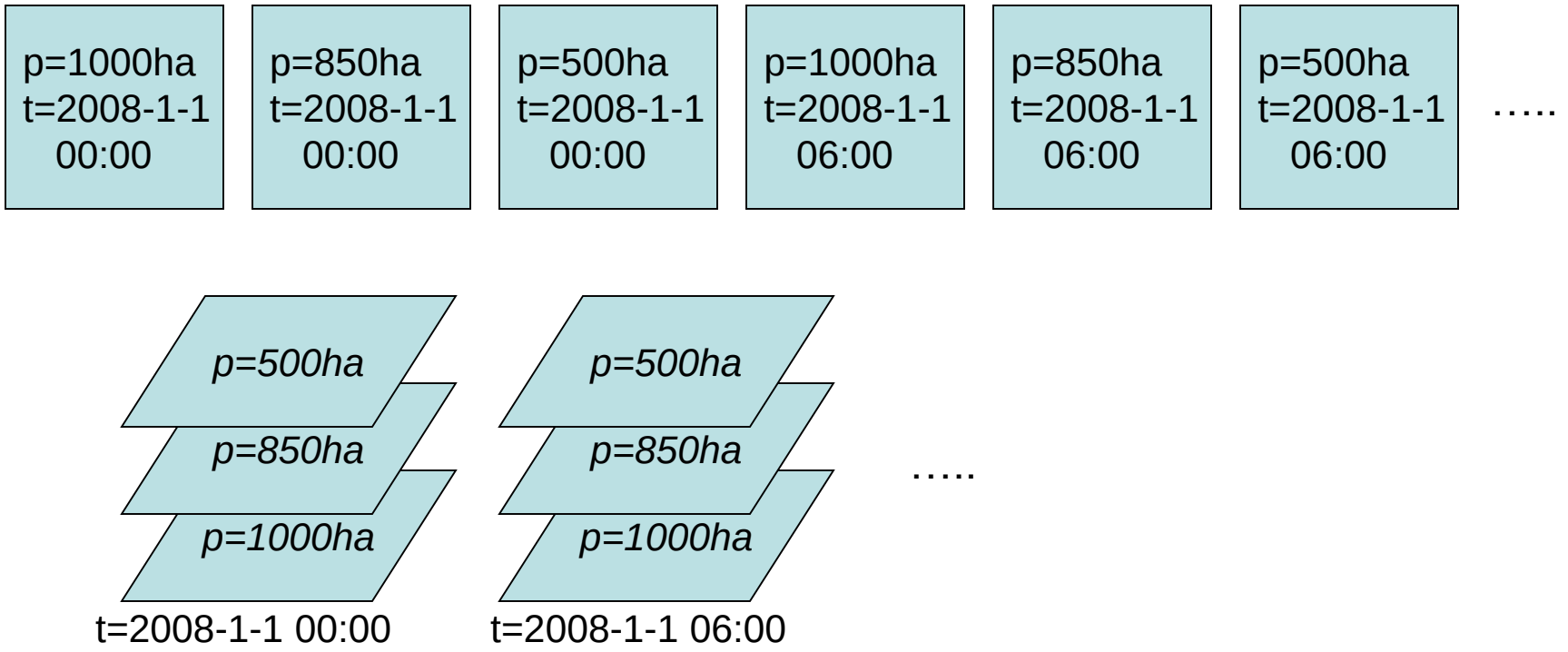
airが気温の4次元データ: lon, lat, level, timeの関数。

バイナリ構造は隠蔽され、名前ベースでアクセス。

```
$ ncdump -h air.2007.nc
netcdf air.2007 {
dimensions:
        lon = 144 ;
        lat = 73 ;
        level = 17 ;
        time = UNLIMITED ; // (365 currently)
variables:
        float level(level) ;
                level:units = "millibar" ;
                level:long_name = "Level" ;
        float lat(lat) ;
                lat:units = "degrees_north" ;
                lat:long_name = "Latitude" ;
        float lon(lon) ;
                lon:units = "degrees_east" ;
                lon:long_name = "Longitude" ;
        double time(time) ;
                time:units = "hours since 1-1-1 00:00:0.0" ;
                time:long_name = "Time" ;
        short air(time, level, lat, lon) ;
                air:long_name = "mean Daily Air temperature" ;
                air:valid_range = 150.f, 350.f ;
                air:units = "degK" ;
                air:add_offset = 477.66f ;
                air:scale_factor = 0.01f ;
                air:missing_value = 32766s ;
                air:precision = 2s ;
// global attributes:
                :Conventions = "COARDS" ;
                :title = "mean daily NMC reanalysis (2007)" ;
                :history = "created 2007/01/03 by Hoop (netCDF2.3)" ;
                :description = "Data is from NMC initialized reanalysis¥n",
                        "(4x/day).  It consists of most variables interpolated to¥n",
                        "pressure surfaces from model (sigma) surfaces." ;
```

参考：GRIB形式

- 気象予報機関の世界標準
- 水平2次元スライスに関する独立したバイナリデータの集合体。
- ヘッダはバイト（＆ビット）単位で各種符号が規定されている。
  - 物理量の種類や時刻や高度はヘッダーに書かれている
  - 時系列や高度方向の次元の認識は解釈系に任されている
- NuSDAS、「国内二進」などの気象庁の形式も同様な構成



Rubyによるデータ解析可視化
基盤ライブラリ：GPhys

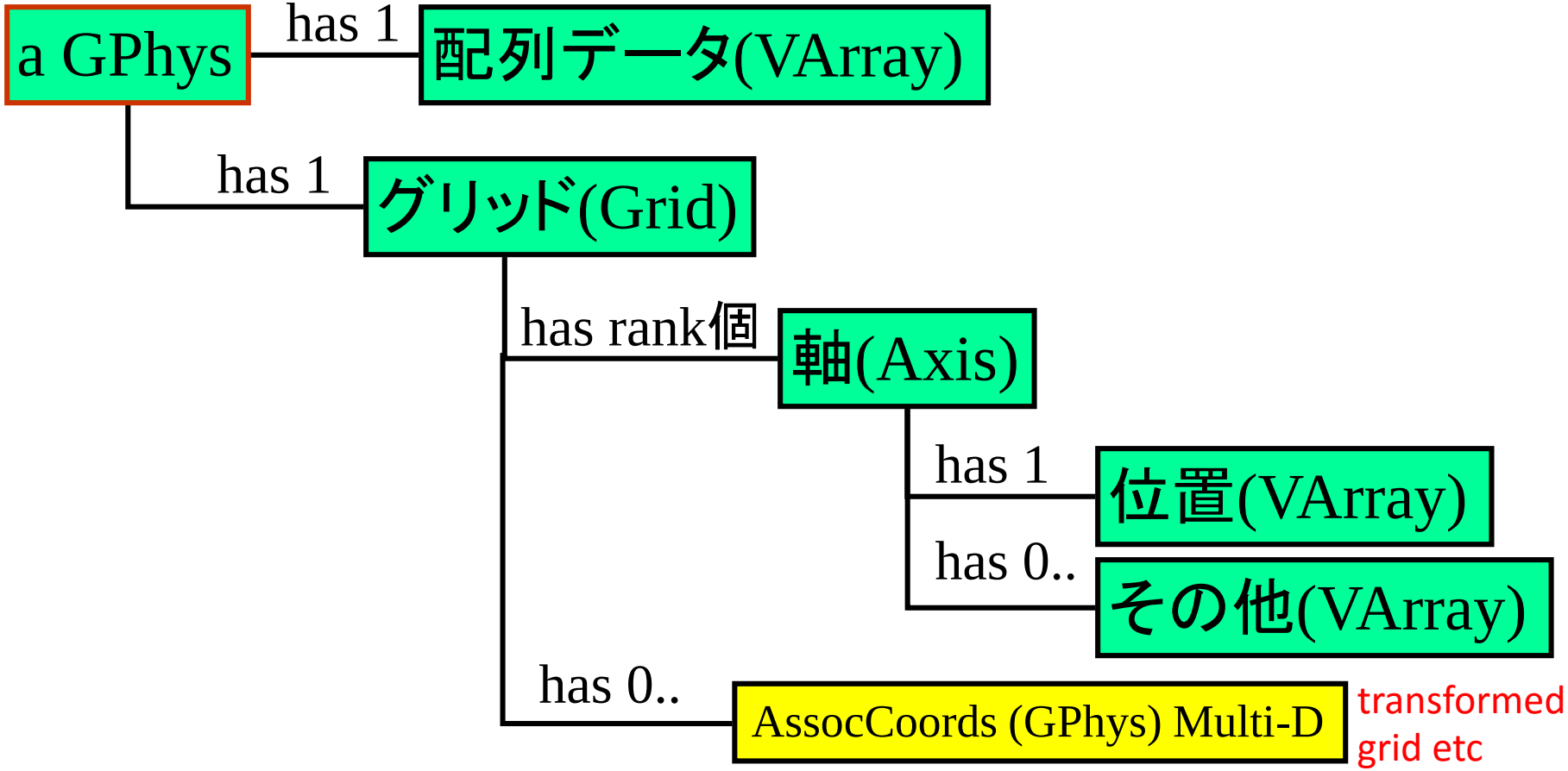
- GPhys = Gridded Physical quantity
- 任意次元（0,1,…）の座標系における物理量をあらわす「クラス」（型）であり、また、GPhysクラスを頂点とするライブラリ。
- データの物理的実体を隠蔽して（下位の諸ライブラリにそれぞれよろしくやってもらって）、ファイルの形式や次元性によらず、統一的なAPIで操作できる。

応用分野

- 流体等、連続空間における物理量に関するデータ解析（incl.可視化）と数値シミュレーション

GPhysオブジェクトの構成

- グリッド（座標データ）と配列データからなる。
- 数学・算術演算や、積分等の座標に関する演算が行なえる。

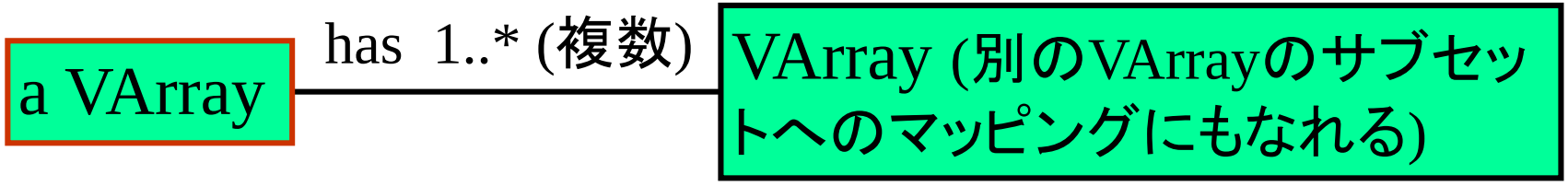


GPhysの重要な構成要素：VArray

- Virtual Arrayの略
- 配列のように振舞うが、データ実体は、Ruby用多次元配列(NArray)やファイル中の多次元データなど多様な場合を統一的にサポート。（サポート形式：NetCDF, GRIB, GrADS, NuSDAS, HDF5-EOS）
- 他のVArrayのサブセットだったり、複数のVArray合成の場合も。
- NetCDF同様「属性」を持てる。

パターン1

パターン2



足回り: NArray (多次元配列)

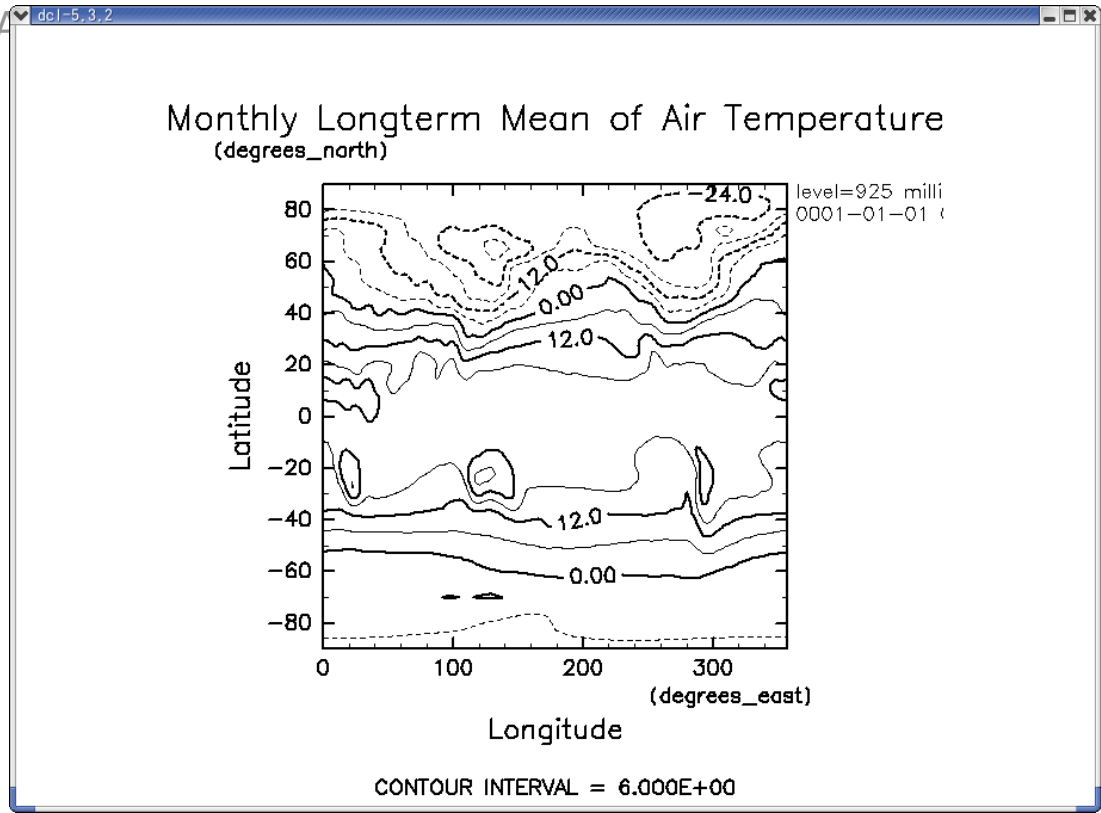
- 作者:田中昌宏
- Ruby非標準(標準添付ライブラリに含まれてない)。Rubyの多次元配列のDe facto標準…(最近はそうでもない？)
- Cによる拡張ライブラリ。数値型(主に)。
- Rubyの柔軟性を活かし，任意次元の配列を扱うアプリケーションが書きやすい。(yorickの「ラバー次元」サポートも効果的)
- 0はじまり。負値で後ろから – Ruby の標準の配列(1D)と同じ。
- 標準は2GBの壁(size_tでなくintを使ってた)。
  - メジャーリビジョンが計画されている
  - 2GB越え＆OpenMP部分サポートのフォークも(by西澤誠也)

利用例（irbによる対話セッション）

スタートアップ用おまじないファイル

コマンドライン入力

```
% irb -r ggraph_startup.rb
 *** MESSAGE (SWDOPN) ***  GRPH1 : STARTED / IWS =  1.
irb(main):001:0> temp = gpopen('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> contour  temp.cut('level'=>925)
 *** WARNING (STSWTR) ***  WORKSTA
=> nil
irb(main):003:0>
```



演算例：

teddy = temp – temp.mean(“lon”)

（経度平均を引く。次元の対応は自動判断）

tfc = temp.fft(nil, 0,2)

（1次元目(0)および3次元目(2)に関するフーリエ変換. nil→forward。座標軸は波数軸に。単位も変換。）

さらに演算例

スタートアップファイルを用いた別表記だと
`u = gpopen("u.nc/U")`

```
u = GPhys::IO.open("u.nc","U")
v = GPhys::IO.open("v.ctl","V")
uv = u * v
outfl = NetCDF.create("out.nc")
GPhys::IO.write(outfl, uv)
```

← in NetCDF [m/s]
← in GrADS [m/s]
← result on memory [m^2s^{-2}]
← 出力ファイル
← 出力（座標も一緒に）

経度,緯度，高度，時刻の4次元データより経度,時間に関するスペクトルを求める．（前処理後処理いろいろ．メソッドチェーンで）

```
sp = u.detrend(3).cos_taper(3).
     fft(false,0,3).abs**2
pw = sp.rawspect2powerspect(0,3).
     spect_zero_centering(0).
     spect_one_sided(3)
```

GPhysのその他の特徴

- 遅延評価
  - 必要になるまでデータは読まない／コピーしない．例：サブセット切り出しは対応写像だけをバーチャルに
- 大きな実データを無理なく扱う仕組み
  - 処理を自動分割するイテレータのサポートなど
- 演算時に単位を自動更新
- 付属ライブラリ
  - すばやく可視化できる描画ライブラリGGraph
    - クイックルックから論文用の凝った図まで（手数は凝り方に応じて。凝り出すと急に難しくなるギャップなしに）
  - データ解析ライブラリGAnalysis（気象学用などいろいろ）
  - 簡単なデータ解析や描画用の実行コマンド群

課題

- 並列処理
  - 足回り(NArray)の一層の対応（下から）
    - いろいろ考えるべきことがある。「Cで拡張」を良くやるので，そこをどうするかも。
  - イテレーターを活用したGPhysの対応（上から）
- サポートするデータ処理，数値計算の充実
- ドキュメンテーションやサンプルの充実（積年の課題 ^^; ）

GPhysレベルの並列化について（案）

- 「イテレーター」の利用。
  - 任意の次元でまわすループが現在すでにある → とりあえずそれを別プロセスに投げる形で実装。（なお，配列の分割の指定法はいろいろあり得る。）
- GPhysがもつタイルにわかれたデータのバーチャル統合機能（シミュレーションの分割出力を一体として扱いたいときなどに便利） → それぞれを（あるいはいくつかまとめて）別プロセスに割り当てる。
  - 並列化を隠蔽できるが，柔軟性が低くできることが限られるかも

- いずれも未設計。（やってみたい方募集します…）

まとめ

- オブジェクト指向言語を使うことでファイル形式に依らない統一的な入出力と，データハンドリングを実現
- Rubyを使うことでオープンソースの様々な枠組みが利用可能
  - Webベースのデータサーバー（DB, 解析可視化，知見文書蓄積）構築ツール Gfdnavi（今回は話してない）
  - ドキュメンテーション(RDoc)，テストフレームワーク，etc.（今回は話してない）
  - Cでポータブルな拡張→伝統的資源の拡張