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

(京都大学宙空電波科学研究センター)

はじめに

- 地球・惑星の流体の研究（気象,海洋,etc）
- プログラミングによるデータ解析
- ほとんどの研究者のやり方は非効率
  ⇒　なんとか改善したい　⇒　本研究開発

発表の構成

- 地球･惑星科学におけるデータ
- 従来のデータ解析手法とその問題
- データ解析のための言語の選択
- インフラ開発
- 物理量ライブラリーの提案と概要
- 実行例と従来法との比較
- まとめ
- 将来展望

地球・惑星科学におけるデータ

- 例　グローバルな気温のデータ

  T(longitude, latitude, height, time)

  3次元の格子点上の温度データの時系列 ⇒ 4次元

- データに共通する特徴
  - 主変数（従属変数）に対する座標値（独立変数）の存在　⇒　**離散化された、場の物理量**
  - 各々名前や単位といった属性を持つ（“温度”, K ; “経度”, 度;…）

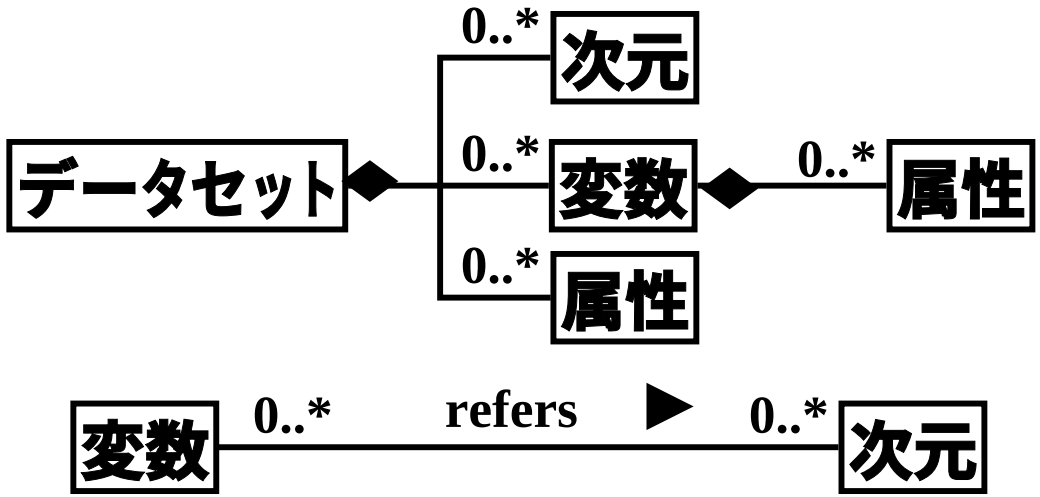
データを格納するファイル

- 「自己記述的」形式（近年）
  - データを読むのに必要な情報がファイル中に存在
  - NetCDF, HDF：専用IOライブラリー. バイナリー構造隠蔽
  - その他： GrADS, grib
- 研究者・グループ毎の独自形式 （旧来）

NetCDFファイル形式

- 「配列指向」
- 複数の変数＝多次元配列（スカラーを含む）を納める
- 変数、ファイルは名前と値の組による「属性」を持てる

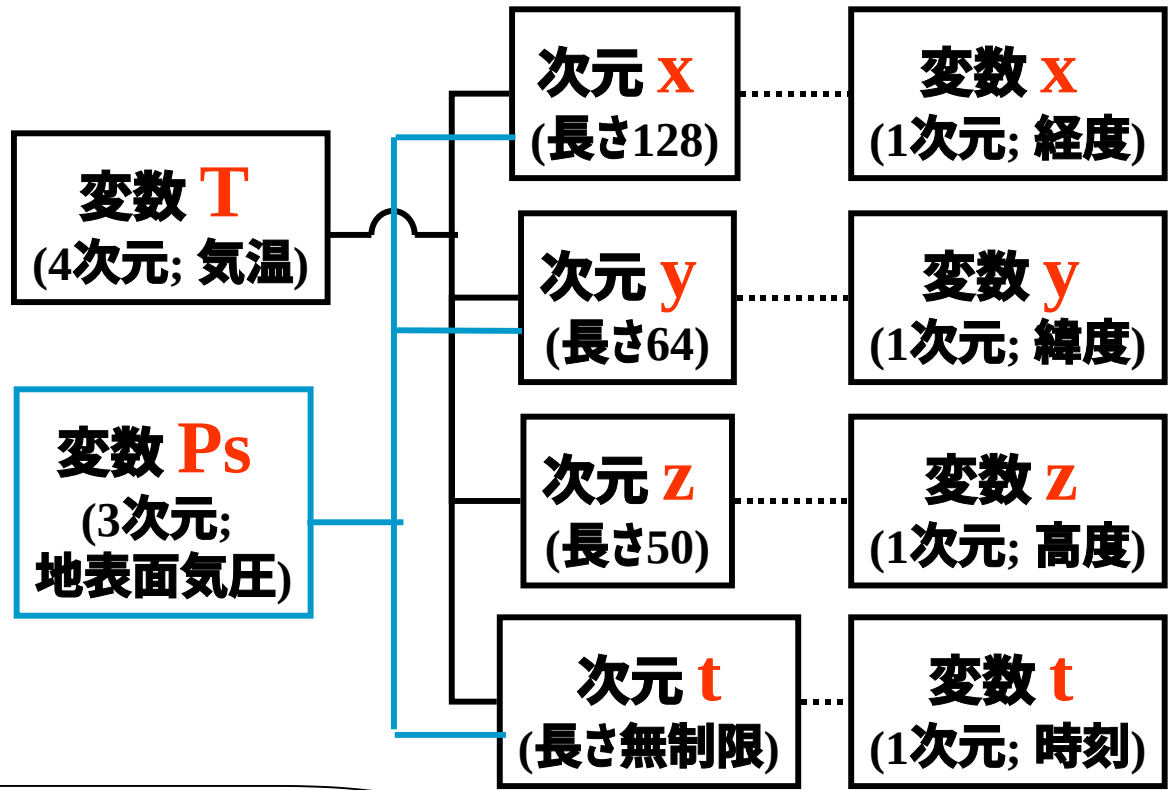
NetCDFファイルの構成

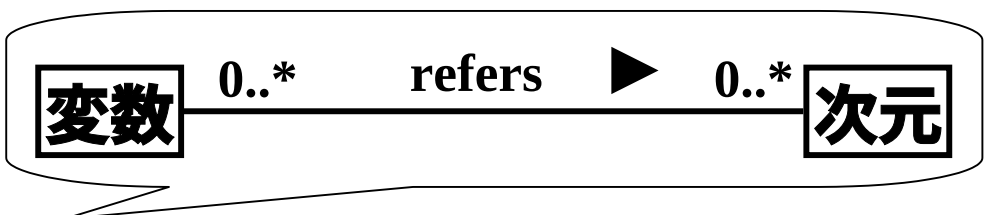


NetCDF形式における変数相互の関連づけ

- 次元名と一致する名前を持つ変数に座標値を収める

例





従来のデータ解析手法とその問題

- スタイル:
  1. プログラミングによる
  2. GUIによる
- データの自己記述性:
  a. 利用しない
  b. 利用する

- **問題点**

(1-a) **プログラムがデータ構造依存になりやすい。不必要に長くなる。（言語：**Fortran, C, IDL, Matlab, yorick,…）

(1-b) **現在、使いやすいライブラリーが存在しない**

(2) **予め用意されたメニューに縛られる**

データ解析のための言語の選択

- データの格納形態に依存しない ⇒ オブジェクト指向
- 日常的にプログラミング； 試行錯誤とプログラミングの間のシームレスな移行 ⇒ 型無し、対話的に利用可能

⇒ Ruby を選んだ

- Rubyを使う場合の、その他のメリット
  - 移植性。ソース公開
  - 拡張性： 既存資源のラッピングによる活用
  - 文字処理
  - ネットワーク関連等、豊富かつ増え続けているクラスライブラリー

インフラ

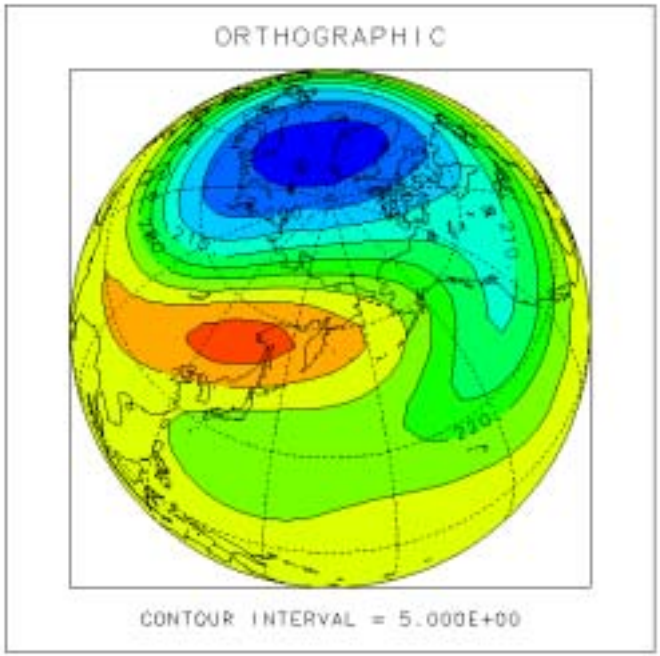
- **多次元配列**
  - NArray **を利用**　(C**のべた並びポインターを利用した多次元数値配列クラス**)

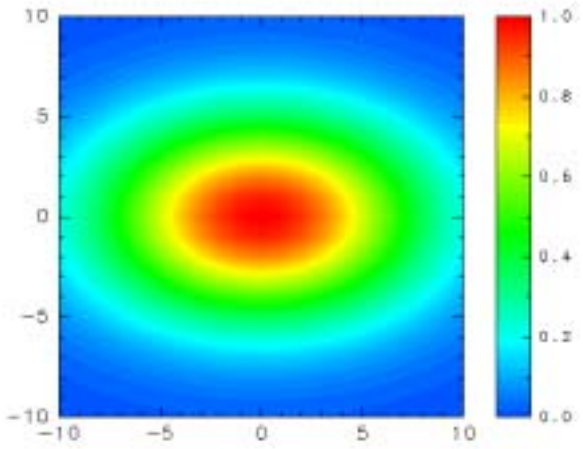
独自開発済:

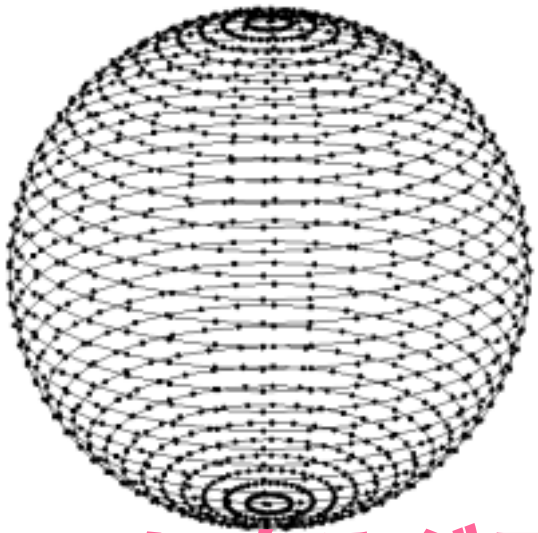
- NetCDF　IO**ライブラリーのラッパー ⇒ 公認**
- **可視化ライブラリー: 地球流体電脳ライブラリー** (DCL)**のラッパー**

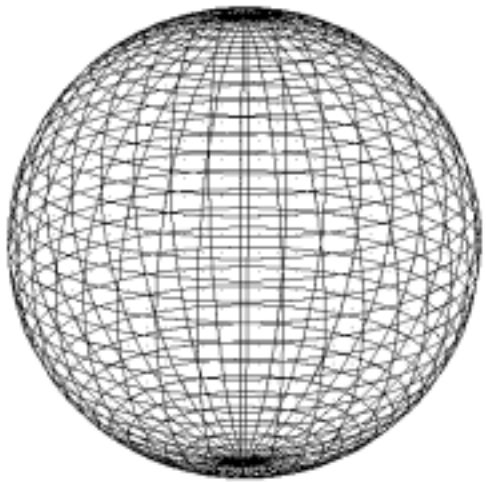
地球流体電脳ライブラリ (DCL)

- 多くのUNIXと、Windowsで稼動
- 様々な描画機能
- 表示法やレイアウトの多彩できめ細かな調整可能
- 地球・惑星流体科学での応用を念頭においた、欠損値処理や地図投影

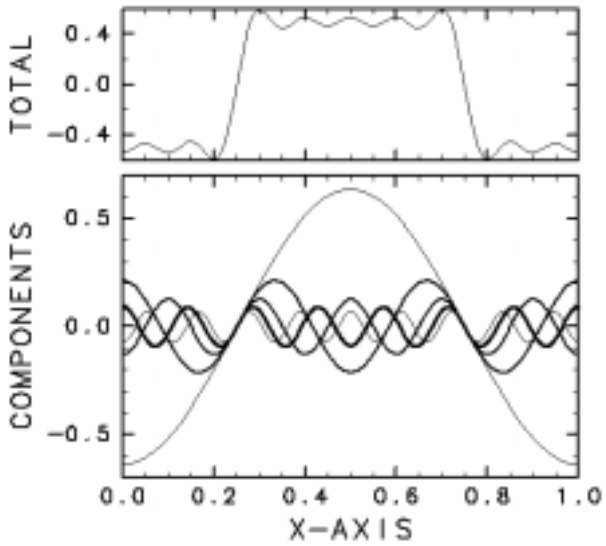


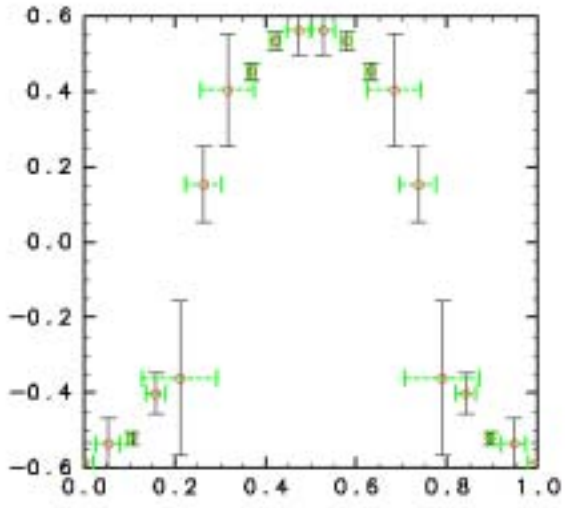




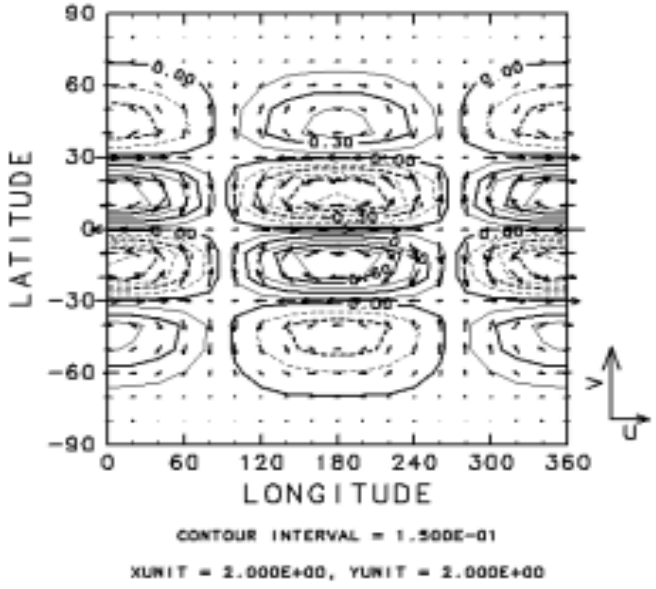


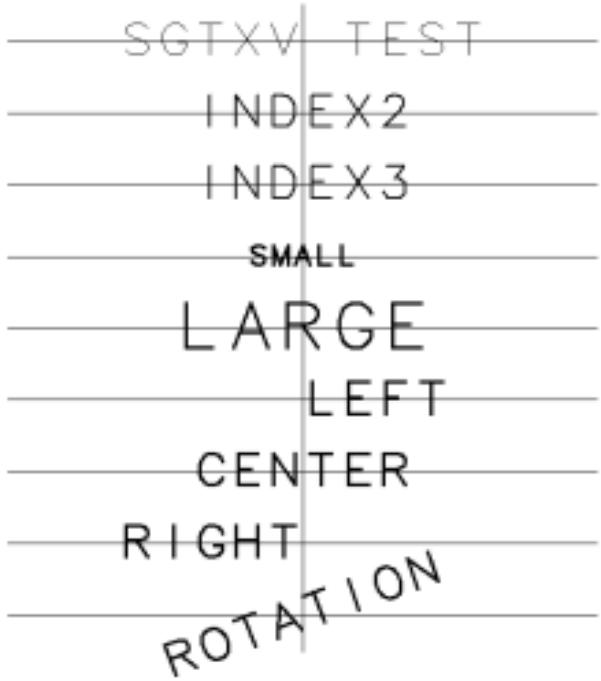
DCLによるグラフィックスのサンプル





DCLによるグラフィックスのサンプル

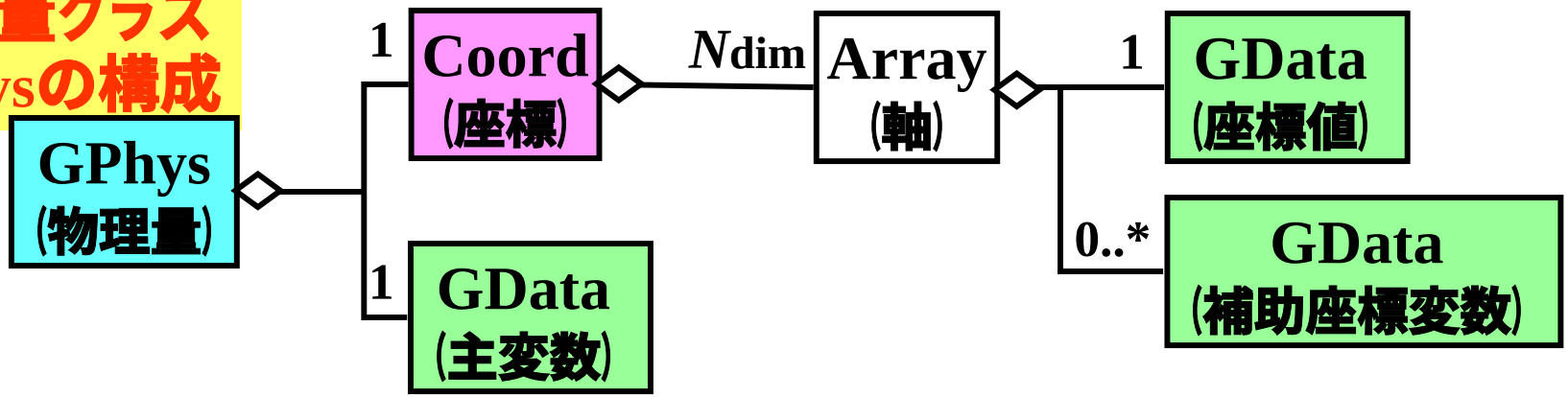




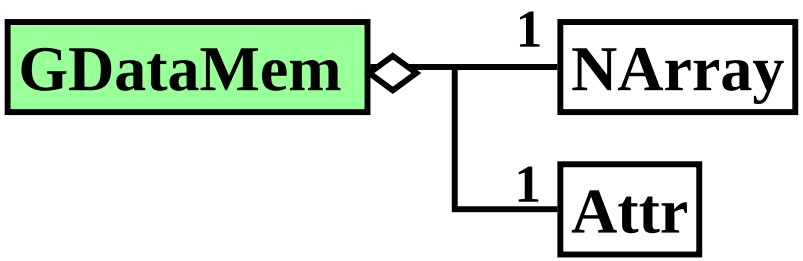
物理量ライブラリーの提案と概要

- 「離散化された、場の物理量のモデル」を考え、データの格納形式によらずそれに当てはめる。

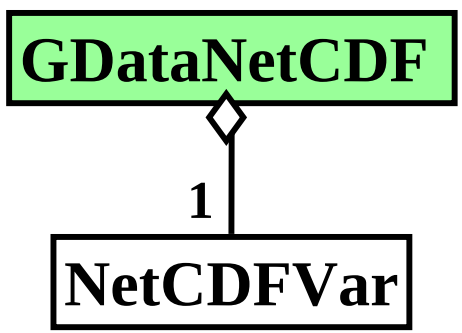
物理量クラス
GPhysの構成



メモリー上のGData



NetCDFファイル上のGData



各クラスの主なメソッド

- GData （多次元データ）
  - 単項演算, 2項演算
  - 値・属性の設定、取りだし（値はNArrayで）
  - 型変換
  - サブセット切り出し（参照のみ／本当の切り出し）
  - 数学・統計ライブラリー（ほとんど未実装）
  - 大規模データ分割の自動分割イテレーター
- Coord （座標）
  - サブセットの切り出し
  - 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　(物理量)
  - 座標に関する情報を必要としないメソッドは GData単体と同じ.
  - サブセット切り出し (byインデックス／座標値)
  - 座標変換、内外挿 (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 (モジュール化)

ファイル中のデータとGPhysの対応の自動化

- **変数と属性の対応** (GData)
  - NetCDF**の場合そのまま。一般に、ファイル中の変数のオブジェクトに対して属性を設定できるようにする。属性のない形式の場合メモリ上の属性クラスの利用により一応解決（未実装）**
- **変数間の関連** (GPhys)
  - **関連判断のためのメソッドを収めるモジュールに局所化（未実装）。**NetCDF**の場合、座標変数は名前で探し、座標補助変数は属性から探す。**

大規模データの扱い

- メモリー上に一度に読み込むには大きすぎるデータの処理用に、詳細を隠蔽した形で自動分割、繰り返し処理を行うイテレーターを作成。(参照によるサブセット切り出しを利用)
- 使用例:

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

実行例と従来法との比較

気温の4次元データ (NCEP気候値)

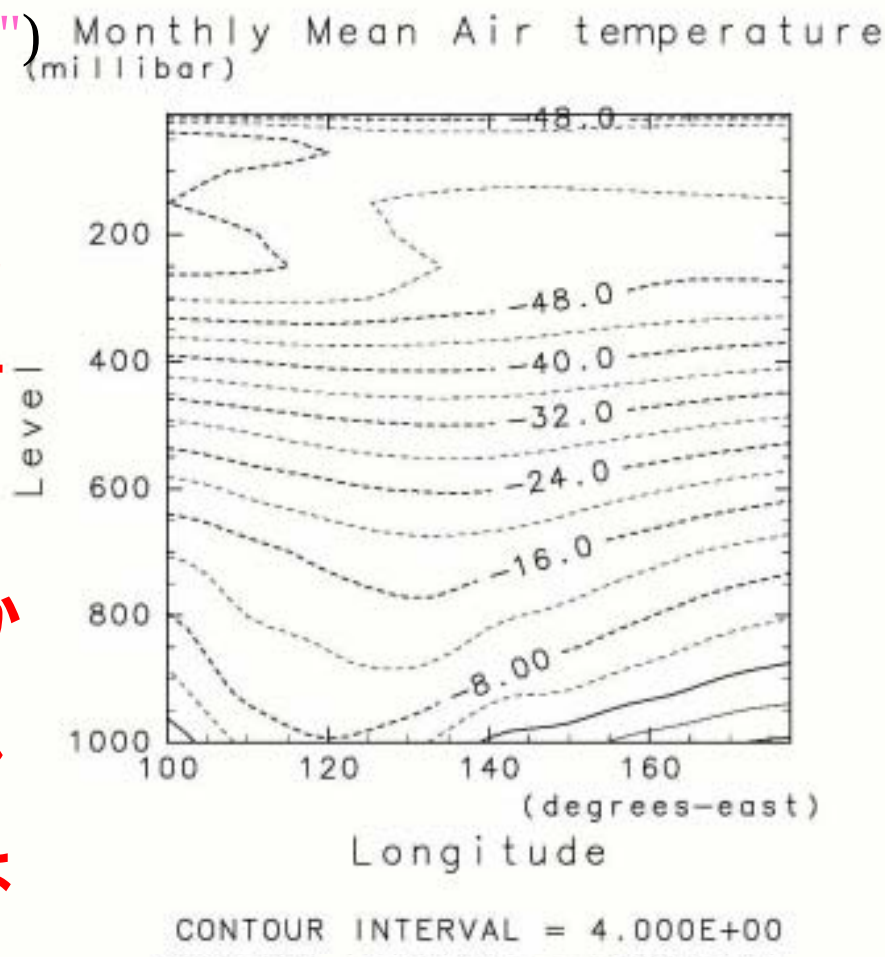
```
require "numru/gphysrect"
require "numru/gphys/gphys_graphic"
include NumRu
gphys =
    GPhys.open("air.mon.ltm.nc","air")
# デフォルト
gphys.contour
```

1次元目の座標値が120(E120°)となる部分を切り出しコンターを描く。
to2dは結果を2次元の物理量にする。

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

1次元目の座標値が100(E100°)から175(E175°)まで、2次元目の座標値が40(N40°)となる部分を切り出しコンターを描く。"nlev"=>18によりコンターの本数を18本に指定 (標準は12本)

```
gphys.to2d(:dv1=>[100,175],
          :dv2=>40).contour(:nlev=>18)
```



```
require "numru/advanceddcl"
include NumRu::AdvancedDCL
require "numru/netcdf"
include NumRu

file = NetCDF.open("air.mon.ltm.nc")
air = file.var("air")
lonname = air.dim_names[0]
latname = air.dim_names[1]
levelname = air.dim_names[2]
timename = air.dim_names[3]
lon = file.var(lonname)
lat = file.var(latname)
level = file.var(levelname)
time = file.var(timename)

airval = air.get({"start"=>[48,0,0,0], "end"=>[48,-1,-
1,0]})
```

従来の方法でプログラムすると左のようになる（約100行）

（つづく）

物理量同士の計算

(例)　p(x, y)：x とy の2次元データ

GPhys使用

```
pbar = p.avg(0)

b = (p – pbar) / pbar

    # ↑ 次元数が異なる

b.copy(NetCDF.create(filename))
```

GPhys不使用

```
pval = p.get
pbarval = pval.avg(0)
bval = Narray.float(*pval.shape[1..-1])
for  i  in  0..(pval.shape[1]-1)
      bval =  (pval[i, true] – pbarval)/ pbarval
end

newfile = NetCDF.create(newfilename)
xd = newfile.def_dim(“x”, p.shape[0])
yd = newfile.def_dim(“y”, p.shape[1])
newfile.def_var(“x”, “float”, [xd])
```

ループが必要

(まだまだつづく)

自己記述性的ファイル
を手作業で構成すると
相当な行数になる

結論

- 地球・惑星流体科学におけるデータ解析をRubyで行うための基礎的なライブラリーを開発した。
- このため、離散化された場の物理量データのモデルを考え、解析するデータを「物理量オブジェクト」に当てはめて扱うことを提案し、クラスライブラリーを実装した（実装はまだ初期段階）。
- 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桁減少する。また、ユーザーが書くプログラムの汎用性が増し、研究コミュニティー内のプログラムの共有と蓄積に役立つ。

将来展望

- 短期的には、未実装部分の実装と解析メソッドの増強、対応ファイル形式拡充
- 可視化機能のGUI
- 数値シミュレーションへの応用
  - 素早い開発が重要
  - ベクトル化、並列化
- リモートデータへのアクセスの実現と、ネットワークを通じた物理量データ配信の高度化
  - より使いやすいデータセンターの実現

離散化された場の物理量の概念モデル