オブジェクト指向言語Rubyによるデータ解析・可視化のためのクラスライブラリー開発

堀之内 武・川那辺 直樹・塩谷 雅人
(京大宙空電波研究センター),
後藤 謙太郎,
神代 剛 (富士通FIP),
電脳Rubyワーキンググループ

目的

- Rubyを地球･惑星流体データの解析、可視化、さらにシミュレーションに用いるための基礎的なライブラリー群を開発する

問題意識

- 現在多くの研究者が行なっている可視化やデータ解析の方法はあまり時間効率が良くない
- データの大規模化･多様化
- データのネットワーク分散

Rubyの特徴

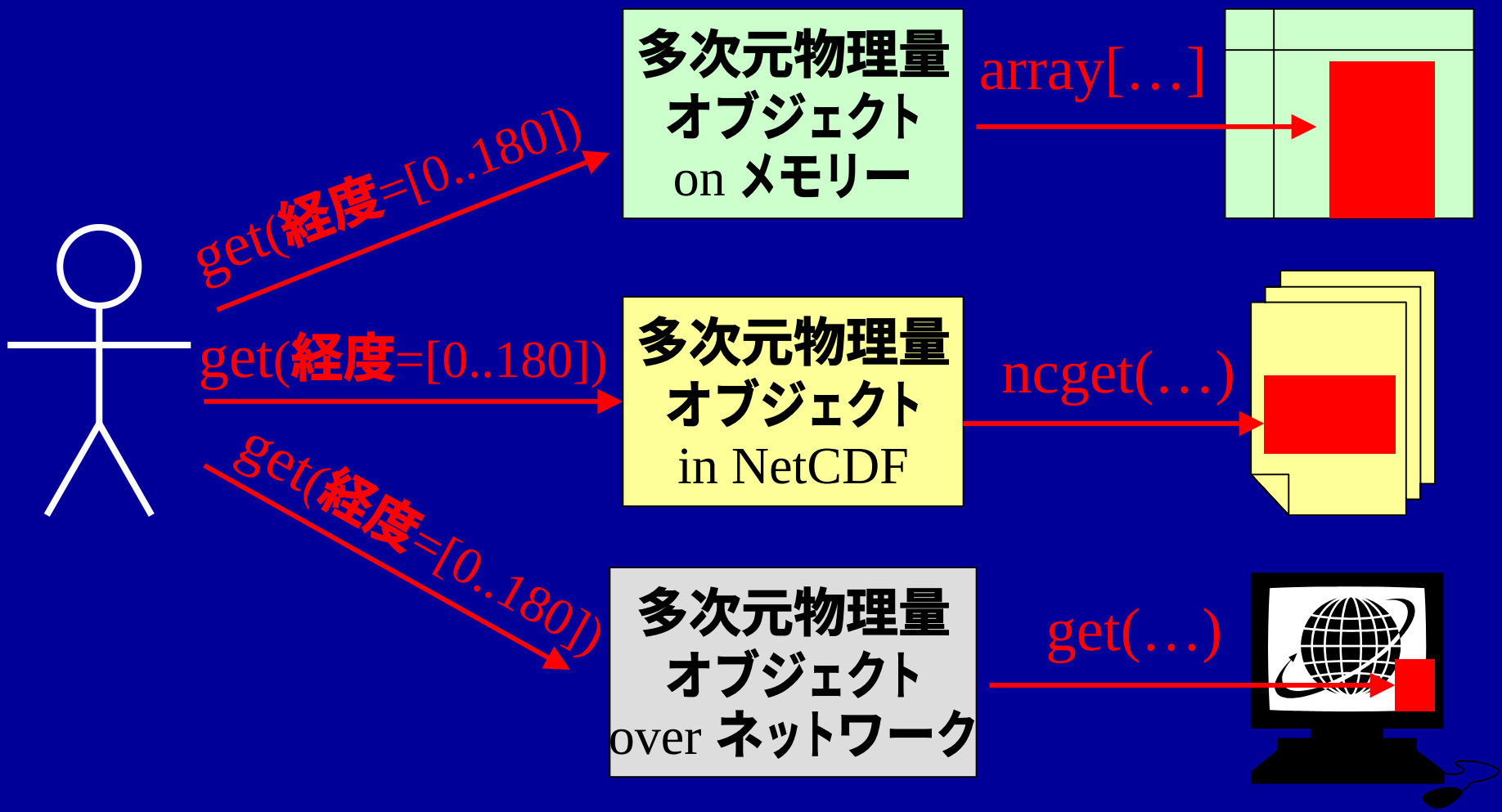
- 型なし、スクリプト言語（インタープリター）
  ⇒ 素早くプログラムが開発できる
- 対話的に利用可能
- 後付けでないオブジェクト指向言語
- 移植性・拡張性が高い
- 増え続けるライブラリー（ネットワーク関連 / GUI / データベース等々）
- 文字処理が容易
- GC、例外処理等の支援機能あり

他言語との比較

	インタプリター	コンパイル型
オブジェクト指向	Ruby	Java
		C++
手続き型	yorick IDL Matlab (商用品)	Fortran C

何故オブジェクト指向言語か？

- 一例：データ構造・媒体を隠蔽した統一的な扱い、抽象度の高い命令 ⇒ プログラミングが楽、汎用性



実行速度の問題

- 流体の場合、計算量を増やすのは主にデータサイズ (グリッド数)

⇒

- Cのべた並びポインターを構造体にくるんだ**多次元数値配列クラス**NArrayを利用
- C等で書かれた (既存) 数値計算ライブラリーをラッピング

プロジェクト概容紹介
(基礎ライブラリー)

効率的多次元配列と数学ライブラリー

GNU Scientific Libraryラッパー(by常定)

他：今のところわずか

描画ライブラリー

DCL (電脳Club Library) ラッパー (完成)

ファイルIOライブラリー

NetCDF IOライブラリー (完成)

他形式 (HDF, grib等)　未作業

多次元物理量クラス

GPhys

(実験段階)

現在NetCDFに対応

プロジェクト概容紹介
(現時点での応用プログラム)

- Gtk, DCLによるNetCDFビューワー (by西澤、実験段階)
- 数値シミュレーション用MVCフレームワーク (by高橋 (前講演)、実験段階)

まとめ

- オブジェクト指向スクリプト言語Rubyを地球･惑星流体データの解析、可視化、シミュレーションに用いるための基礎的なライブラリー群を開発している
- これまでの開発･利用例を通じてその有用性が示されてきたが、今後さらに精力的な開発を行なう必要がある

今後の展望

- 現在未完成・未サポートライブラリーの作成
- 利用者獲得 ⇒ 改良・機能増強
- 分散データベースへの対応 (Pandra, DODS)
- 気象学等の細分野用アプリケーション作成
  ⇒　コミュニティーでの資源共有

電脳Rubyホームページ：ruby.gfd-dennou.org

協力者・利用者募集中　⇒　堀之内まで

horinout@kurasc.kyoto-u.ac.jp

fin

多次元物理量ライブラリーGPhys
(実験段階)

- 自己記述的データオブジェクトのクラス：NetCDF型データモデル
- (今のところ) NetCDFファイル中のデータ、実行時にメモリー上に存在するデータに対応
- メモリーに乗り切らない大規模ファイルの処理を行なうための、自動分割イテレーターをサポート

実行例

```
require "numru/gphys"
require "numru/gphys/graphic"
include NumRu

data =
    GPhys.open("air.mon.ltm.nc",”air")
```

グラフィックス、デフォルト：

```
data.contour
```

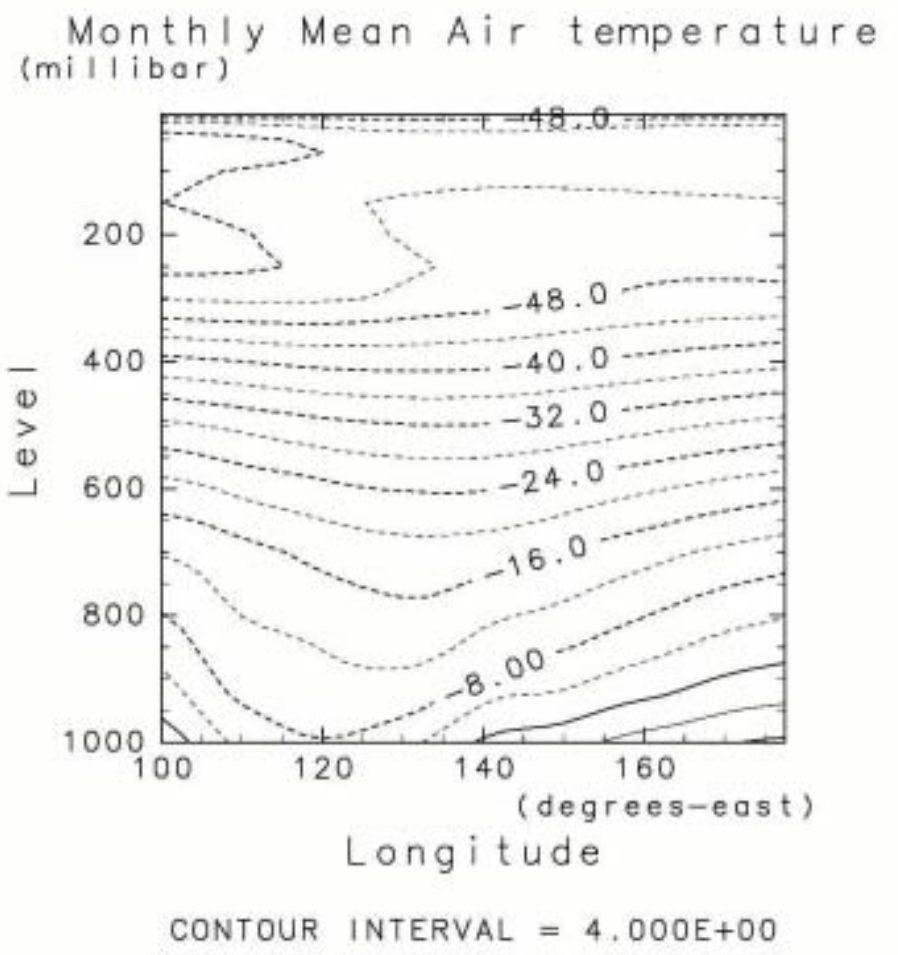
1次元目の座標値が120(E120°)となる部分を切り出しコンターを描く：

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

別の断面や、描画法の指定：

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

気温の4次元データ（NCEP気候値）

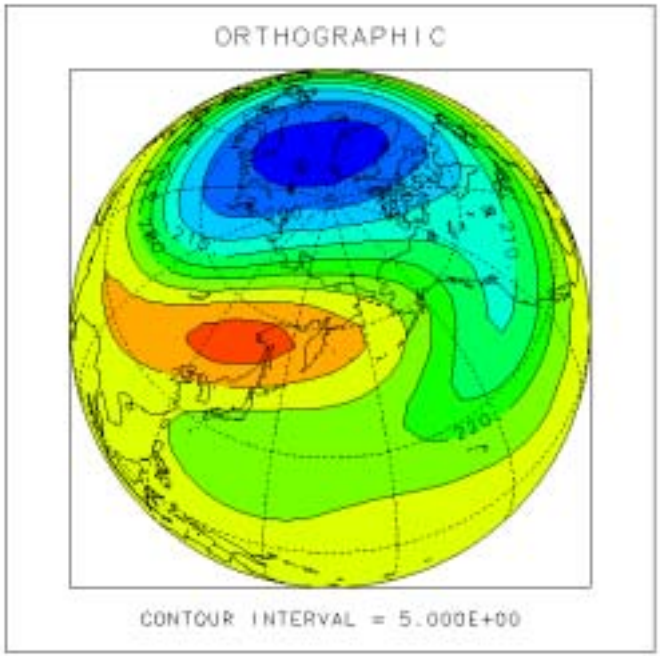


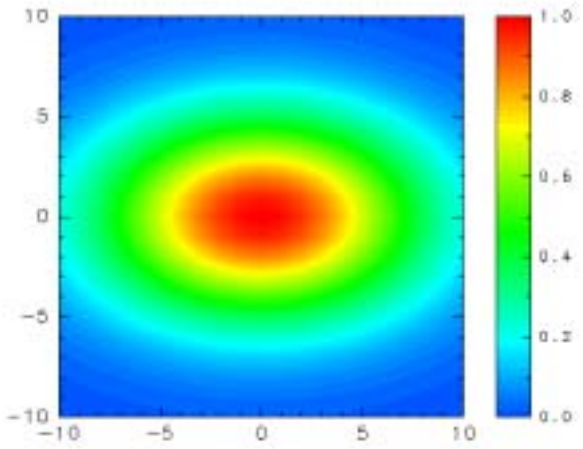
例：二項演算

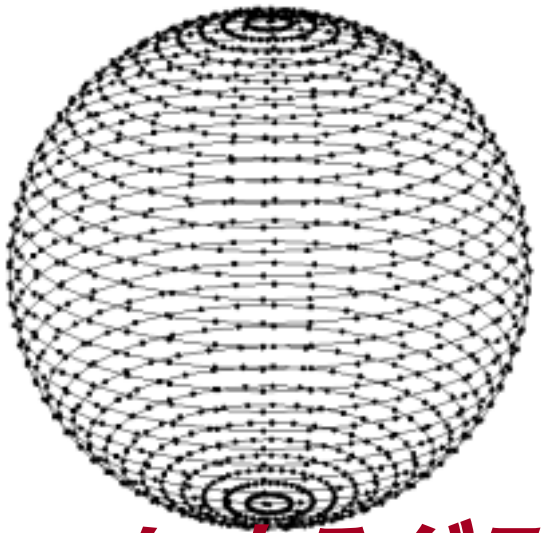
p： x, y の2次元データとする

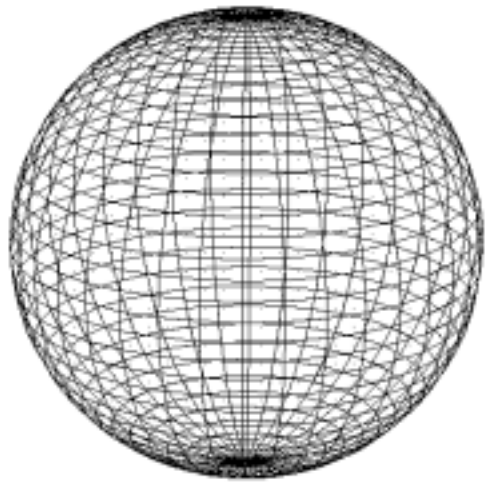
```
pbar = p.avg(0)
b = (p – pbar) / pbar
    # ↑   ↑  次元数が異なる
b.copy(NetCDF.create(“hoge.nc”))
    # 自己記述性を保ったまま
      ファイルへ出力
```

戻る

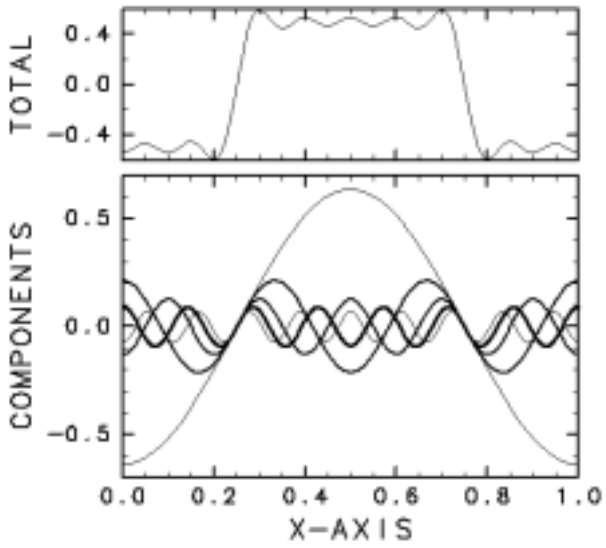


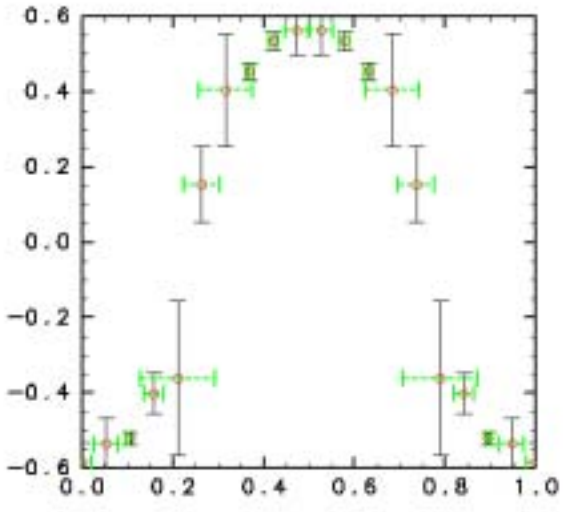




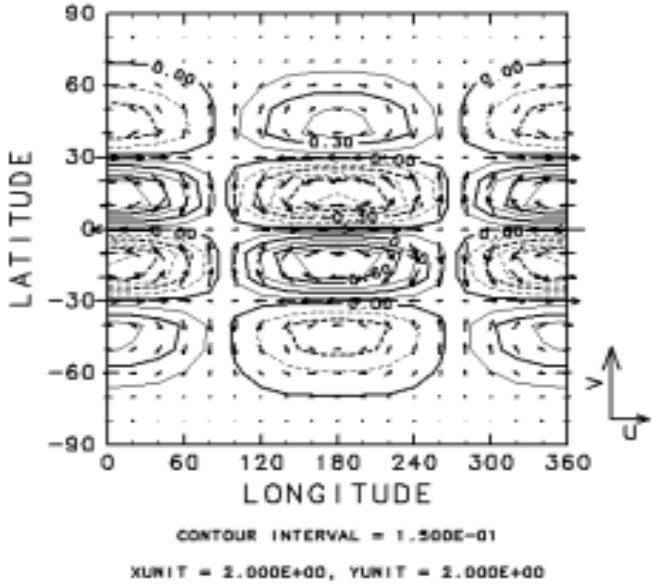


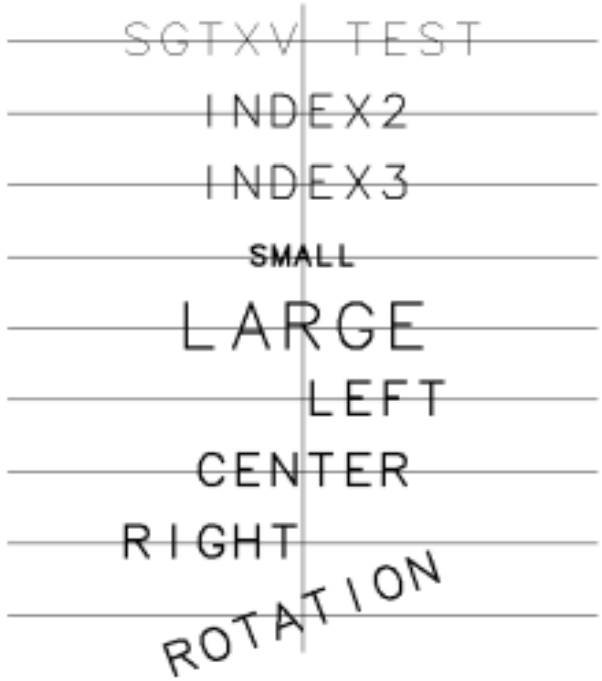
DCLによるグラフィックスのサンプル





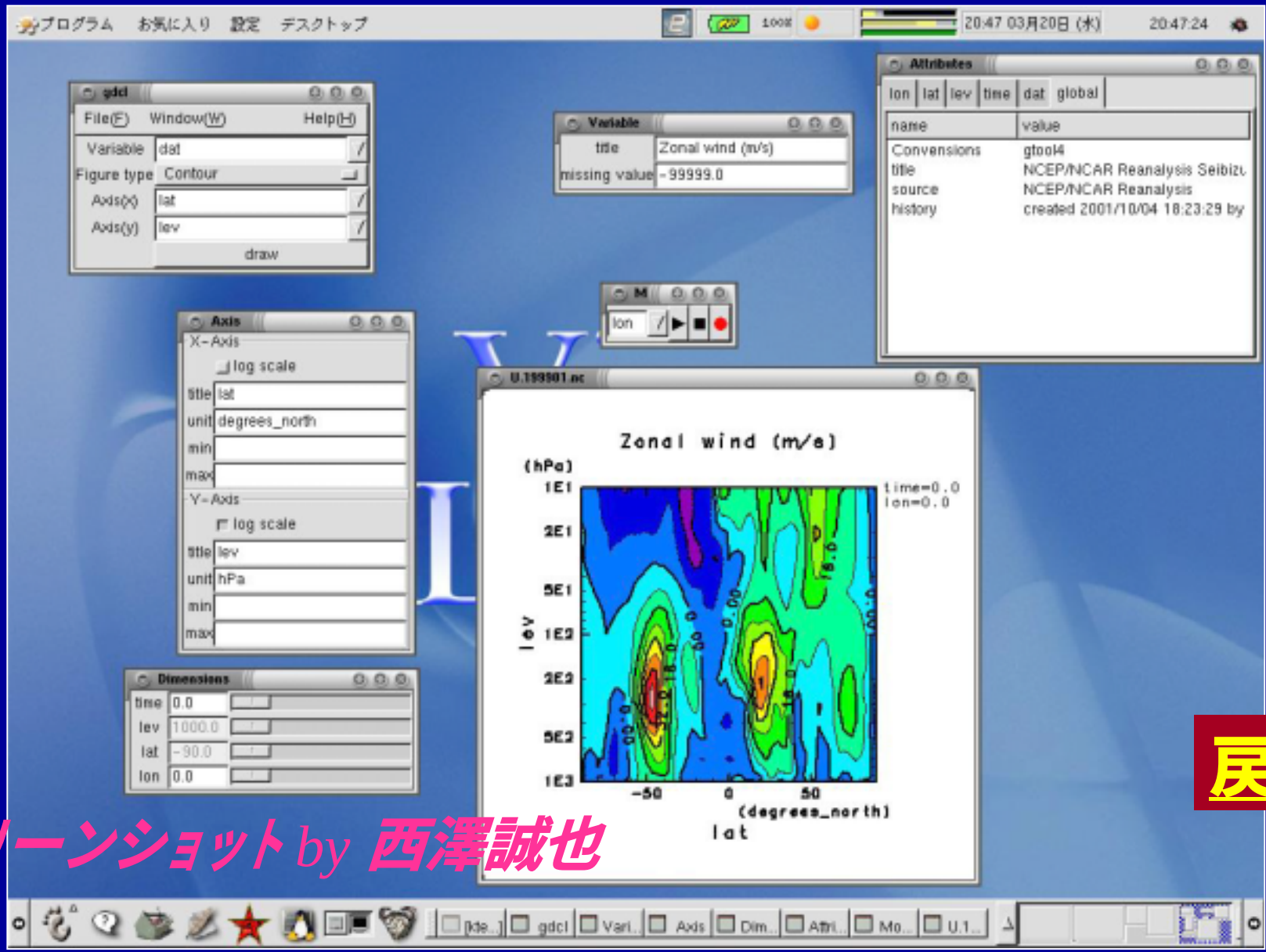
DCLによるグラフィックスのサンプル(2)





戻る

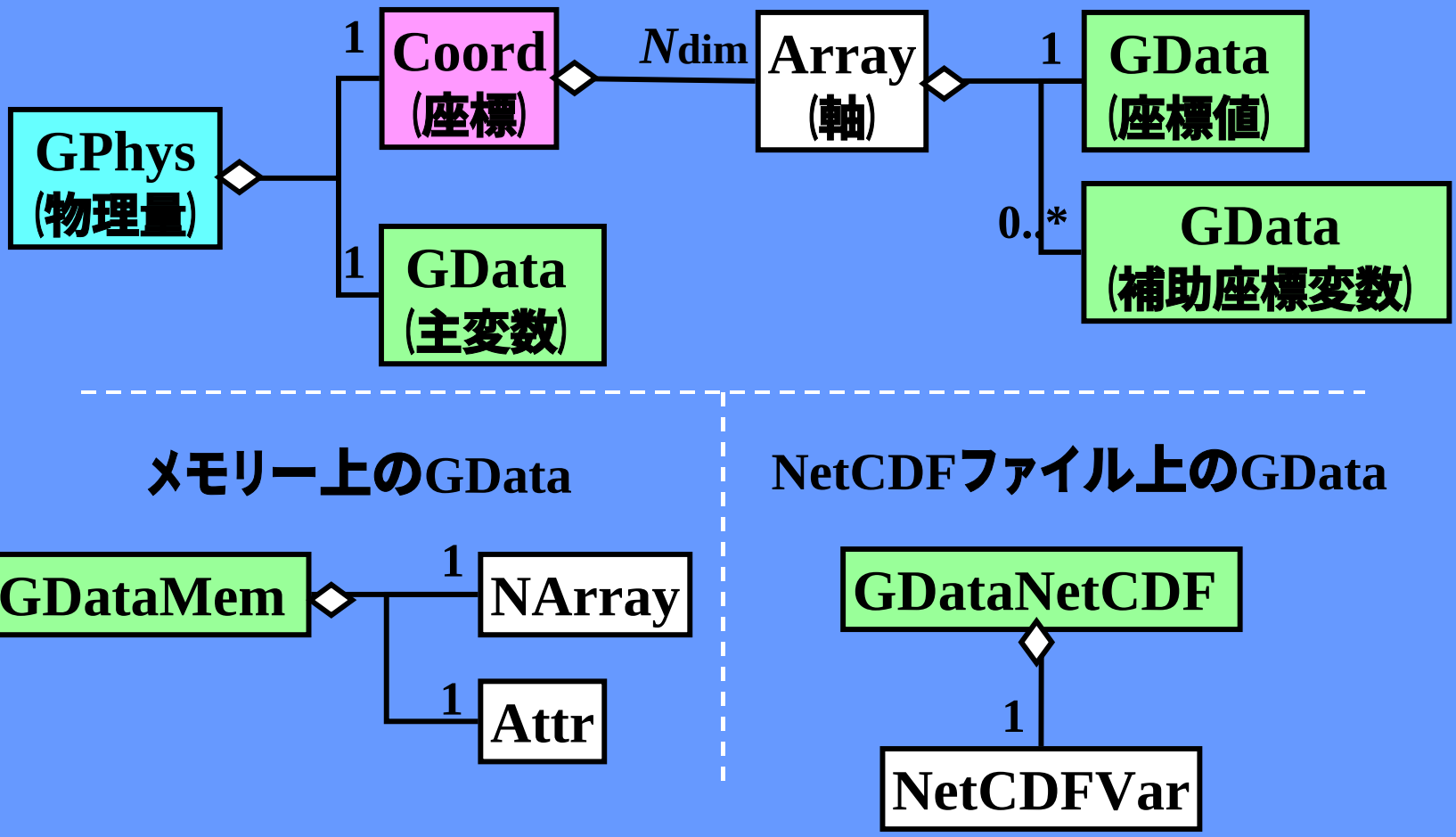
Gtk, DCLによるNetCDFビューワー



戻る

スクリーンショット by 西澤誠也

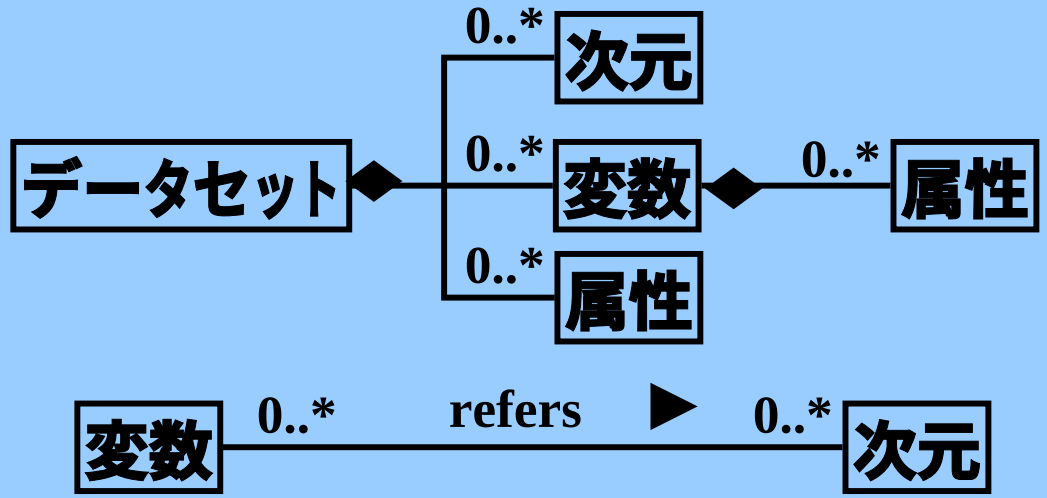
GPhysクラスの内部構成



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

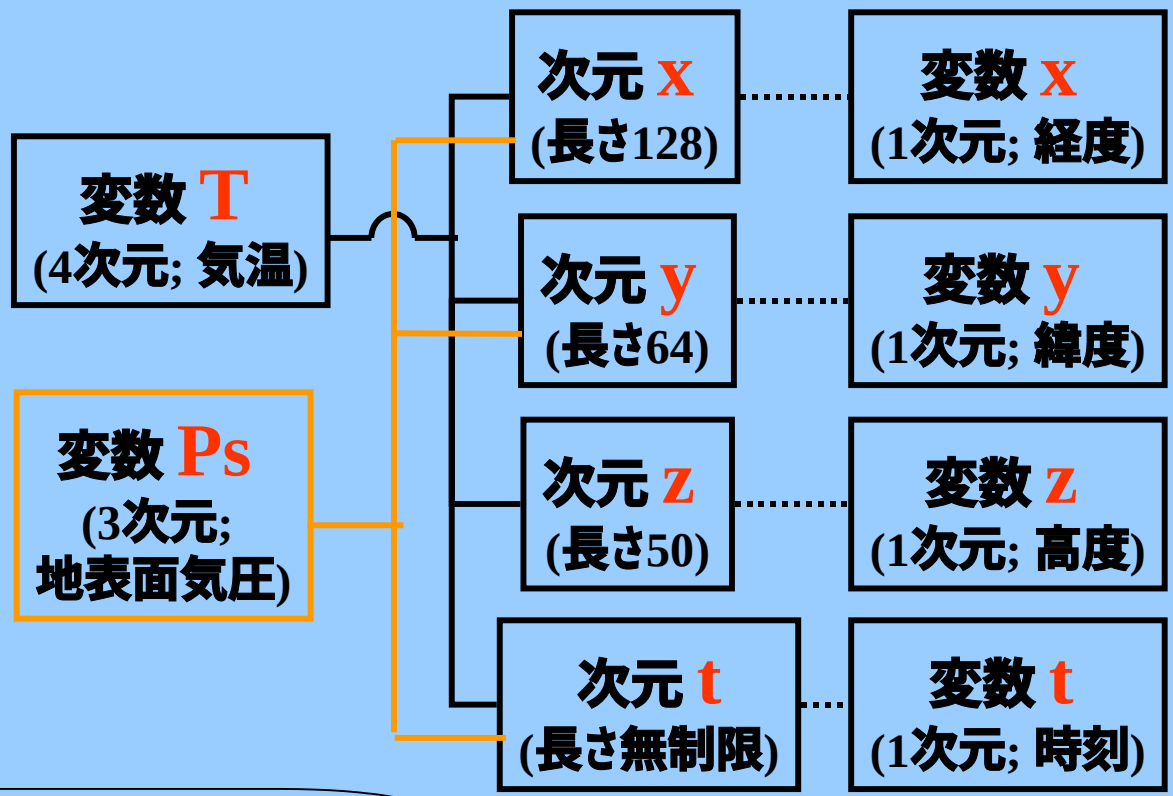
NetCDFファイルの構成

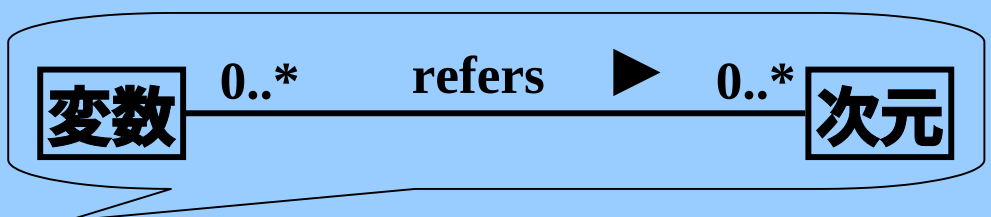


NetCDF形式における変数相互の関連づけ

- 次元名と一致する名前を持つ変数に座標値を収める

例





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
}
```

```
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）

例：二項演算

p： x, y の2次元データとする

GPhys使用

```
pbar = p.avg(0)

b = (p – pbar) / pbar

    # ↑ 次元数が異なる

b.copy(NetCDF.create(filename))

    # 自己記述性を保ったまま
      ファイルへ出力
```

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
```

ループが必要

```
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])
```

（まだまだつづく）

自己記述的ファイルを手作業で構成すると相当な行数になる

戻る