データ解析･可視化ライブラリーGPhys

～柔軟性の高いプログラミングを目指して

京都大学宙空電波科学研究センター　　川那辺　直樹

GPhysとは

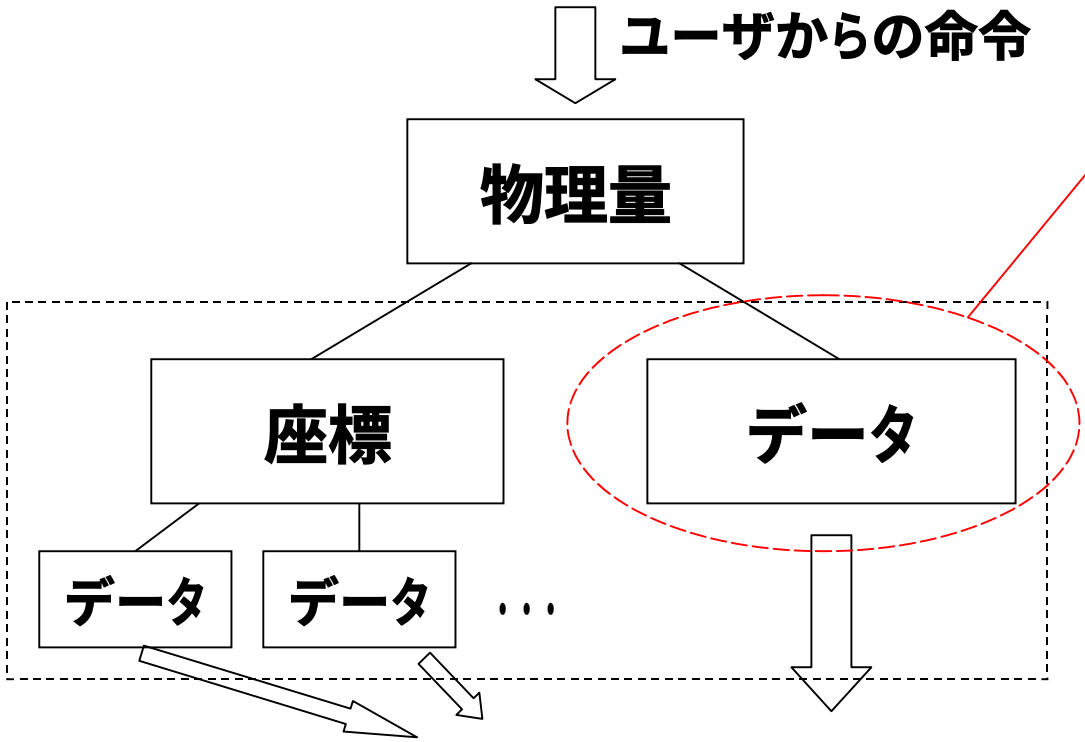
データ変数 (数値配列、属性) と座標変数 (数値配列、属性) ひとつのオブジェクトとして取り扱うクラスライブラリ

ファイル形式に依存しないプログラミングを可能にする。

これまでのプログラミング

- データ形式毎に異なるプログラムを書かなければならない。
- データと座標には対応関係があるがそれが生かされていない。

GPhysの構成



データ形式によって異なる。

各データクラスに、データ形式に依存しないインターフェースを持つメソッドを用意する。

↓

そのメソッド内でデータ形式に応じた呼び出しが行われる。

- **物理量** (GPhys)　**ユーザがアクセスする部分**
- **データ** (GData)　**数値配列と単位などデータに関する情報をもつ。データ形式毎にサブクラスを作る。**(GDataNetCDF)
- **座標** (Coord)　**各次元毎の座標変数をまとめ、扱うためのクラス**(CoordReg)

データの統一的な取り扱い

ファイル形式毎に異なるクラスを用意する。(GDataNetCDFなど)

それぞれのクラスのメソッドを統一する。(メソッド名、引数)

例)

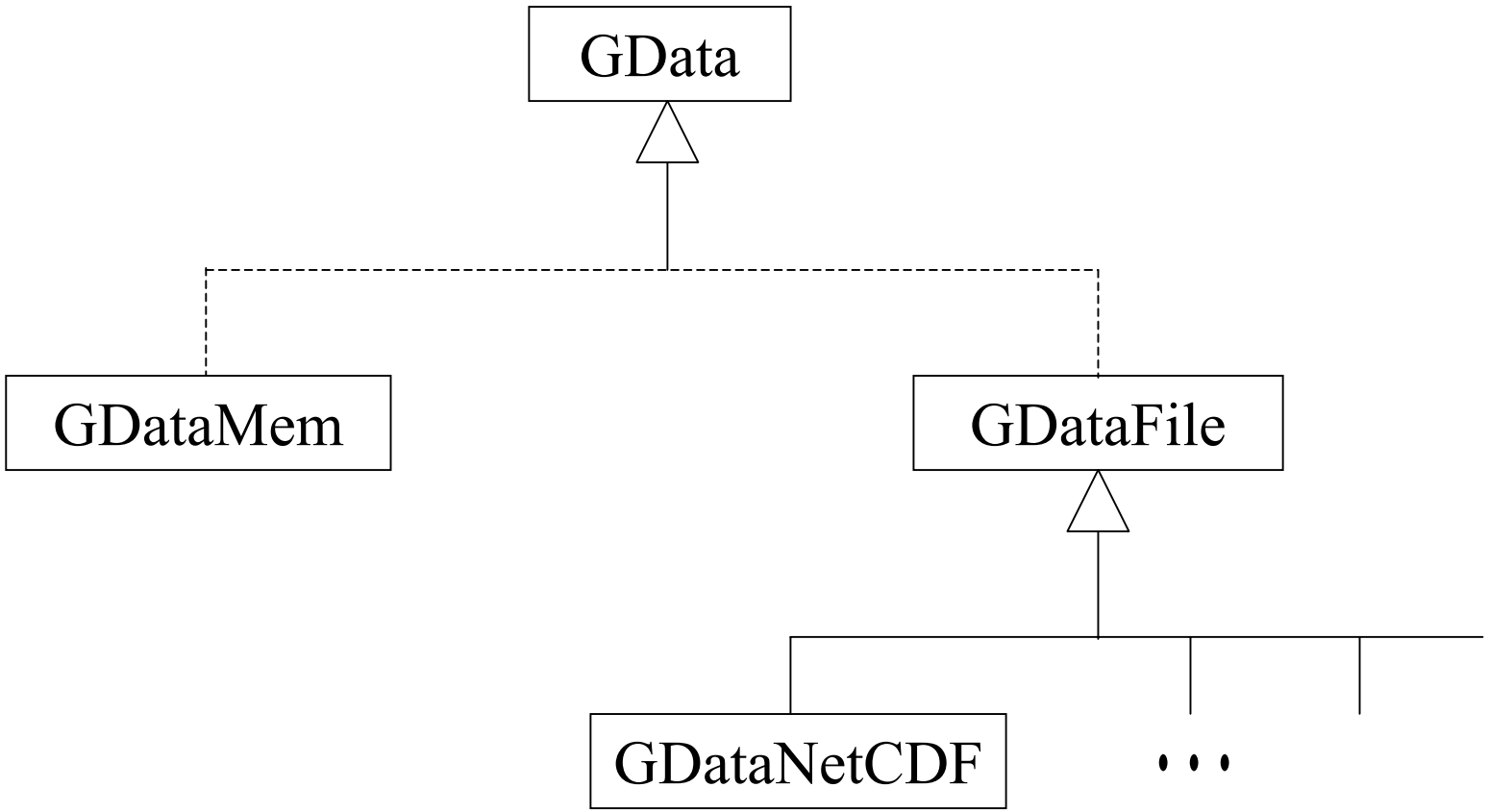
dataオブジェクトのクラスはファイル形式毎に異なる。

↓

そのクラスのgetメソッドが呼び出される。

各データクラスにはgetという名でメソッドを定義する。

データクラスの統一



共通部分をスーパークラスに定義する。

実装、管理の効率化を図る。

データクラスを統一するには

各データクラスにおいてメソッド名及びその中身が共通であるものをスーパークラスのメソッドとして定義できる。

↓

I/O**ライブラリレベルで統一を図る必要がある。**

class GDataNetCDF

@netcdfvar (NetCDFVar)

class GDataMem

@ary (Narray)

@attr (Attribute)

↘

Hash**のサブクラスとして新たに定義**

全てのI/O**ライブラリはファイルクラス**(NetCDF)**、変数クラス**(NetCDFVar)**、属性クラス**(NetCDFAtt)**からなるよう設計する必要がある。**

Attributeクラス

Hashのサブクラスとして定義

Hash

{“a”=>1, :a=>2}

h[“a”] → 1

h[:a] → 2

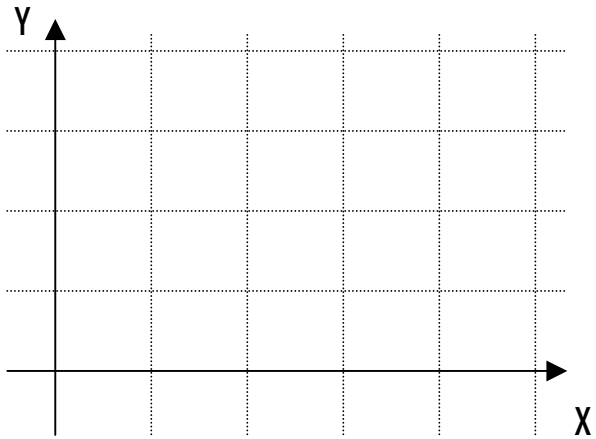
Attribute

{:a=>1}

h[“a”] → 1

座標クラス

CoordReg



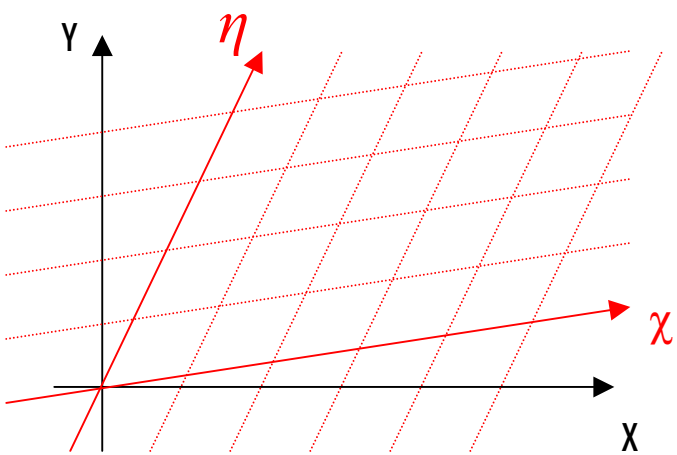
通常の格子点座標を扱うクラス

Class CoordReg

@coord : [[x_axis, x_assist], [y_axis, y_assist], ・・・]

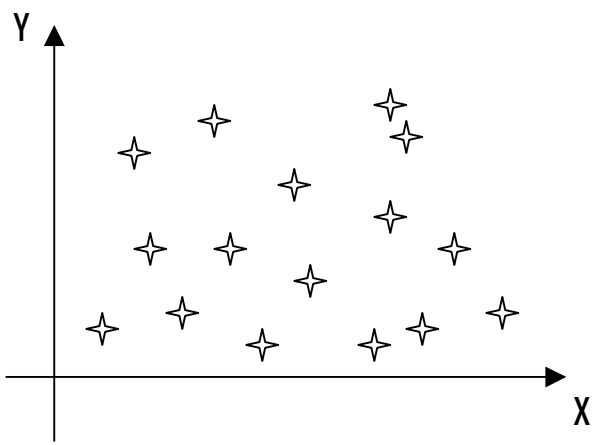
1次元目のGData オブジェクト

1次元目の補助変数 {:weight=> w}



CoordTrans

(X,Y) → (χ, η) **で見るとき**



CoordScat

格子点座標でない座標

例： 衛星観測データ

データと座標の一体化

Gphysクラスの利用により一体として扱うことが可能

```
class  GPhys
    @data       (GData)
    @coord      (Coord)
```

データと座標との関連を判断する必要がある。

NetCDFでは各次元の長さ、名前からデータと座標の対応を判断

↓

GPhysクラスでこの関連判断を行うべきではない。

各フォーマットごとに関連を判断するメソッドを用意する。

→ モジュール化し隠蔽

主な機能

- 単項・二項演算

(例)　T(x, y)：x とy の2次元データ　Tbar：Tの1次元目に関する平均(1次元データ)

のとき、b=T-Tbar　を求める。

物理量ライブラリ使用

```
T=GPhysRect.open(GDataNetCDF, filename)

Tbar = T.avg(0)

b = (T-Tbar).copy(GDataNetCDF, newfilename)
```

2次元 NetCDF　2次元 NetCDF　1次元 メモリ → 2次元に拡張

- 異なる次元数のデータにも対応
- データ形式に関係なく計算可能
- 出力先を指定すると自動的にファイルの作成を行う。

物理量ライブラリ不使用

```
file = NetCDF.open(filename)
T = file.var("t")
Tval = T.get
Tbarval = Tval.avg(0)
bval = Narray.float(*Tval.shape[1..-1])
for  i  in  0..(Tval.shape[0]-1)
    bval[ 0..-1,  j ] =  Tval[ 0..-1 ,  j ] – Tbarval[ j ]
end
```

ループを回して異なる次元数のデータに対処している。

```
newfile = NetCDF.create(newfilename)
          :
newvar = newfile.def_var("b", "float", newdims)
          :
newvar.put(bval)
newfile.close
```

出力先に応じて新しいファイルを作成しデータを出力させなければならない。

•可視化機能

AdvancedDCLを用いた可視化モジュールを作成、利用することで実現している。

用いるNetCDF**データ (ファイル名**air.mon.ltm.nc)

データ
float air (lon, lat, level, time)
long_name : “Monthly Longterm Mean of Air Temperature”
units : “degC”

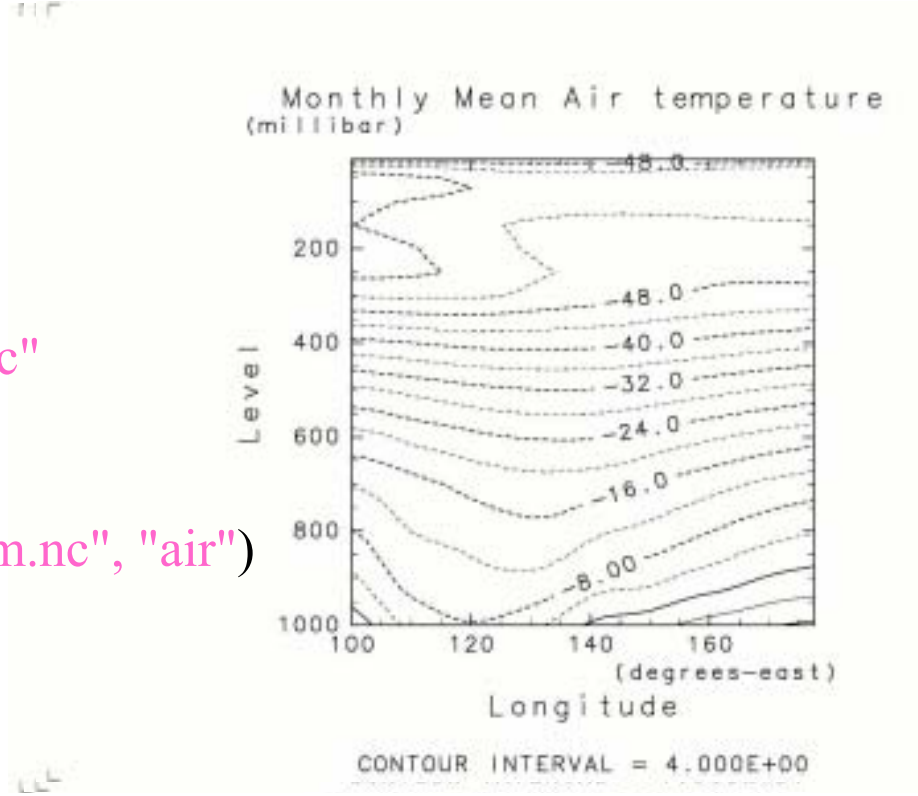
座標
float lon(144)
long_name : “Longitude”
units : “degrees_east”

float lat(73)
long_name : “Latitude”
units : “degrees_north”

float level(17)
long_name : “Level”
units : “millibar”

float time(12)
long_name : “Time”

```
01:  require "numru/gphysrect"
02:  require "numru/gphys/gphys_graphic"
03:  include NumRu
04:  gphys=GPhysRect.open("air.mon.ltm.nc", "air")
       #デフォルト
05:  gphys.contour
       #座標値を指定して切り出し、コンターを描く。
06:  gphys.to2d(:dv1=>120).contour
       # 座標値の範囲を指定して切り出し、コンターを描く。
07:  gphys.to2d(:dv1=>[100,175], :dv2=>40).contour({"nlev"=>18})
```



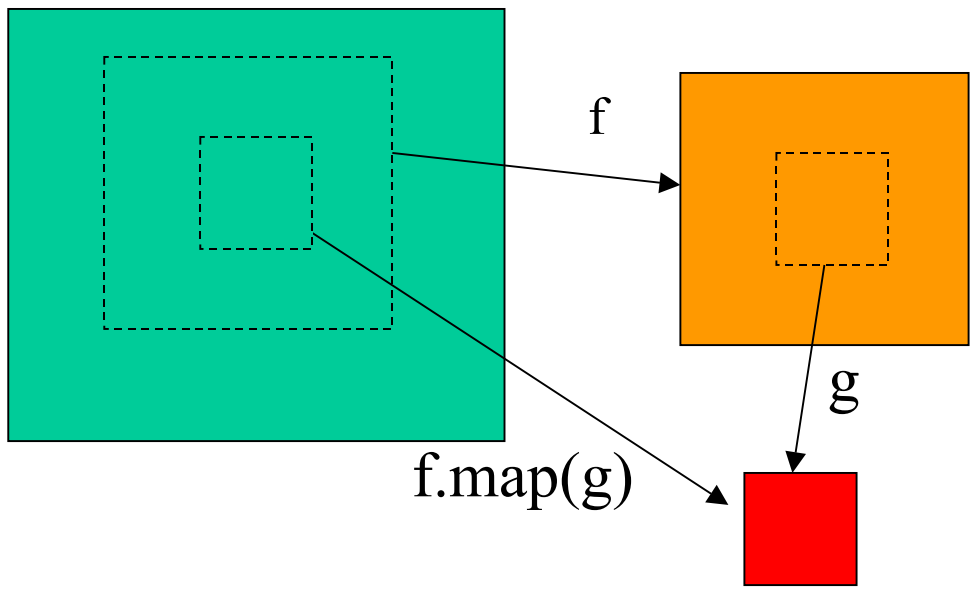
```
require "numru/advanceddcl"
include NumRu::AdvancedDCL
require "numru/netcdf"
include NumRu


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150行のプログラム)

Mapperクラスを用いた参照のみのデータ切り出し

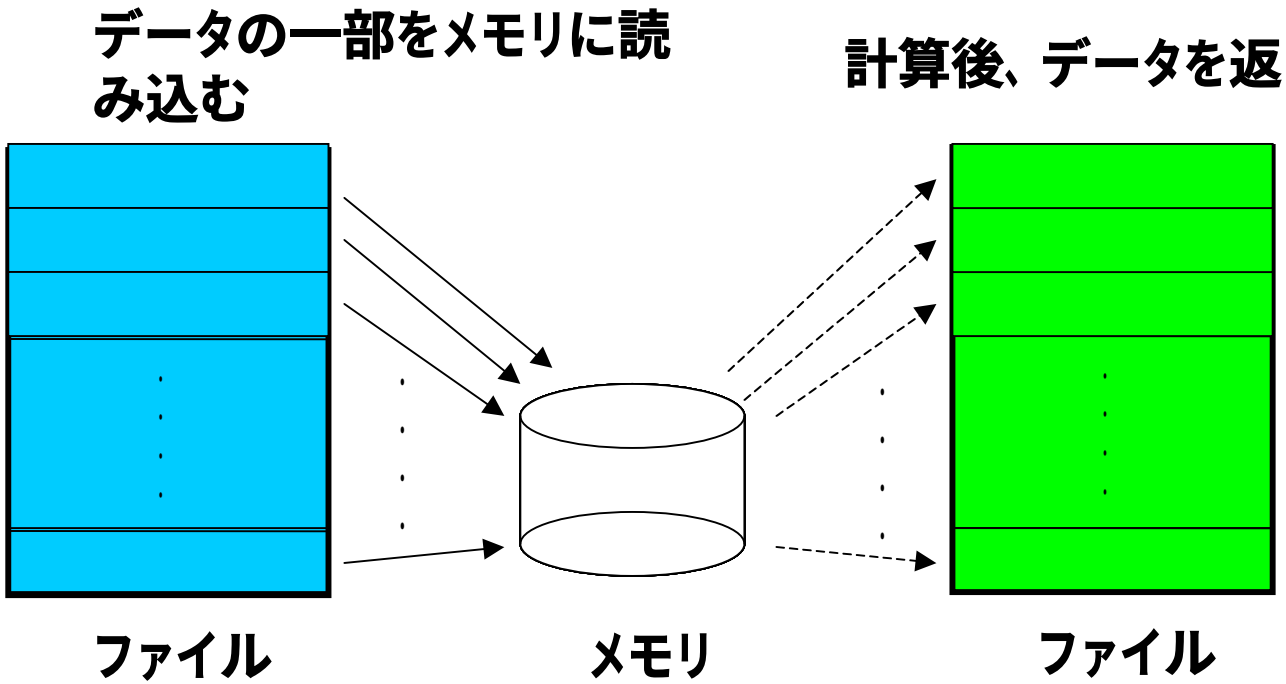
Mapperクラス

データへのマッピング情報を保持するクラス



別ファイルを作ることなくデータの切り出しを行うことが出来る。

巨大なデータファイルの取り扱い



データファイルのサイズが大きい場合、物理量ライブラリはデータの分割を行う。

- 分割をどのように行うのかは物理量ライブラリ内で自動的に決定される。
- 分割を行わない次元を指定することが出来る。
- 分割するサイズの上限を設定することが出来る。

データ分割メソッド

•each_subset

```
bigdata.each_subset(1){|sub|
        p  sub.sum(1)
        a－1
}
```

大容量データbigdata**は自動的に**sub**というサブセットに分割される。**

sub**に関して**{ }**内の処理を繰り返し行う。**

•each_slicer

```
bigdata.each_slicer{|i|
        bigdata3[i]  =  bigdata1[i]  +  bigdata2[i]
}
```

i **は** Mapper **オブジェクト**

•each_subset_set （ **未実装** ）

```
bigdata.each_slicer(outputfile){|i|
        a  =  sin(i)  *  cos(i)
        a - 1
}
```

将来的には、単項･二項演算等分割が必要となるメソッドににこれらの分割メソッドを組み込み、データの分割を自動で行い、計算させるようにする。

まとめ

本研究ではデータ形式に依存しないプログラミングを可能にするため、新しいデータの取り扱いを提案し、それを実現するため物理量ライブラリの開発を行った。物理量ライブラリでは、

- データと座標の一体化によるプログラミング量の減少
- データ形式、次元数に依存しない単項・二項演算
- データの可視化
- メモリにのりきらない大規模データを自動的に分割する機能

が一部を残して実現した。

これからの課題

- 未実装機能を実装する。
- 他形式のデータに対応する。

  NetCDF以外のデータ形式にも対応する必要がある。