IDLによる解析と可視化

松本洋介（千葉大学）

宇宙磁気流体・プラズマシミュレーションサマースクール
2013年8月6日　千葉大学統合情報センター

内容

- イントロダクション
- 環境設定
- データ入出力
- 1次元プロット
- 2次元可視化
- 3次元可視化
- プロシージャの描き方
- 図の保存・アニメーション作成
- まとめ

イントロダクション

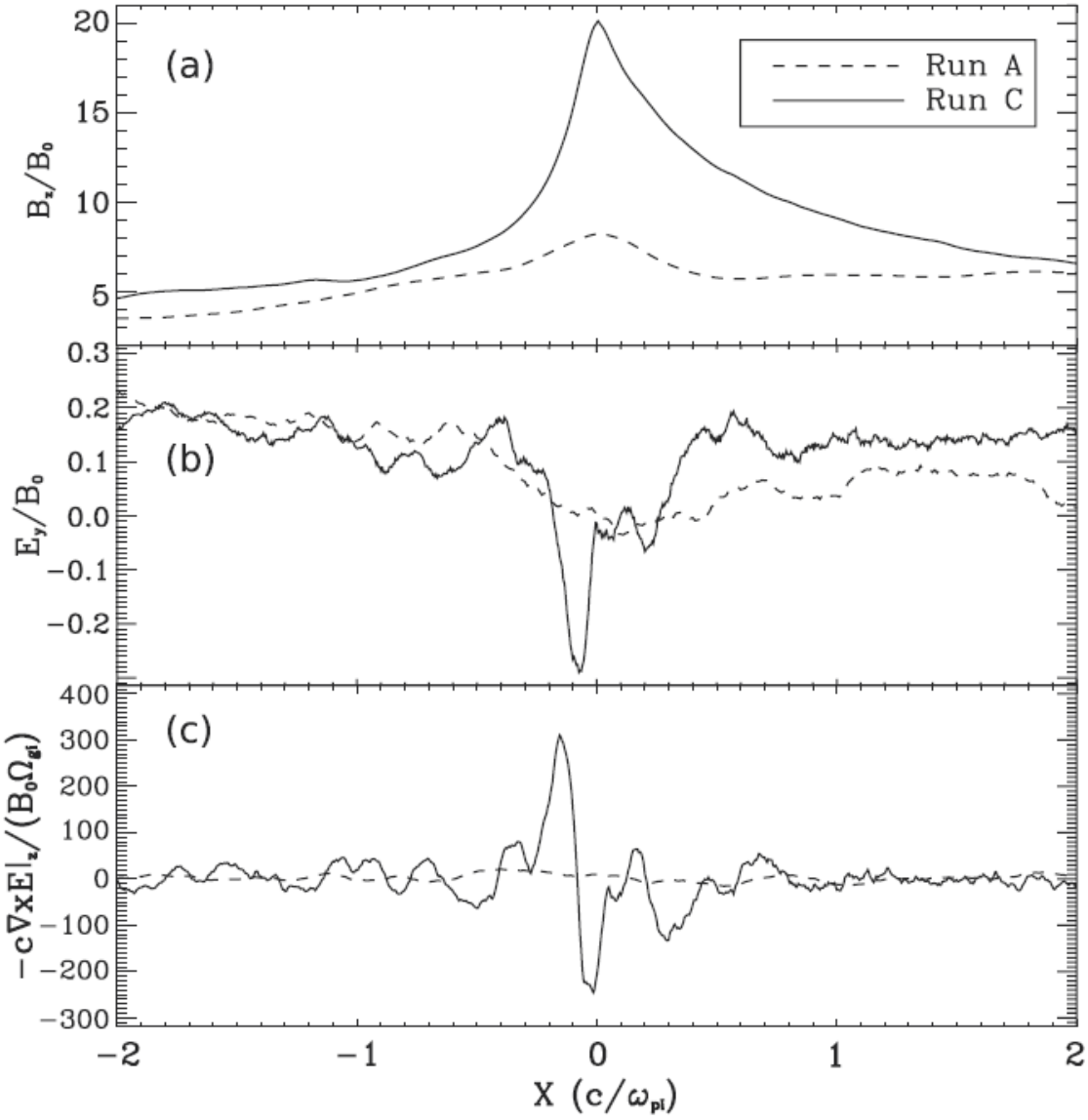
IDL：Interactive Data Language

- Exelis Visual Information Solutionsより販売
  - 1フローティングライセンス（Linux版）：25万円
- 太陽系探査衛星の解析環境として採用され続けている（例：SolarSoft）
- シミュレーション結果の解析・可視化にも
- Fortranライクなインタプリタ言語
- プロシージャ、関数
- 結構自動並列処理してくれる（FFT、sort、ベクトル演算、、）
- Linux, Windows, Mac-OS
- 32bit, 64bit OS対応

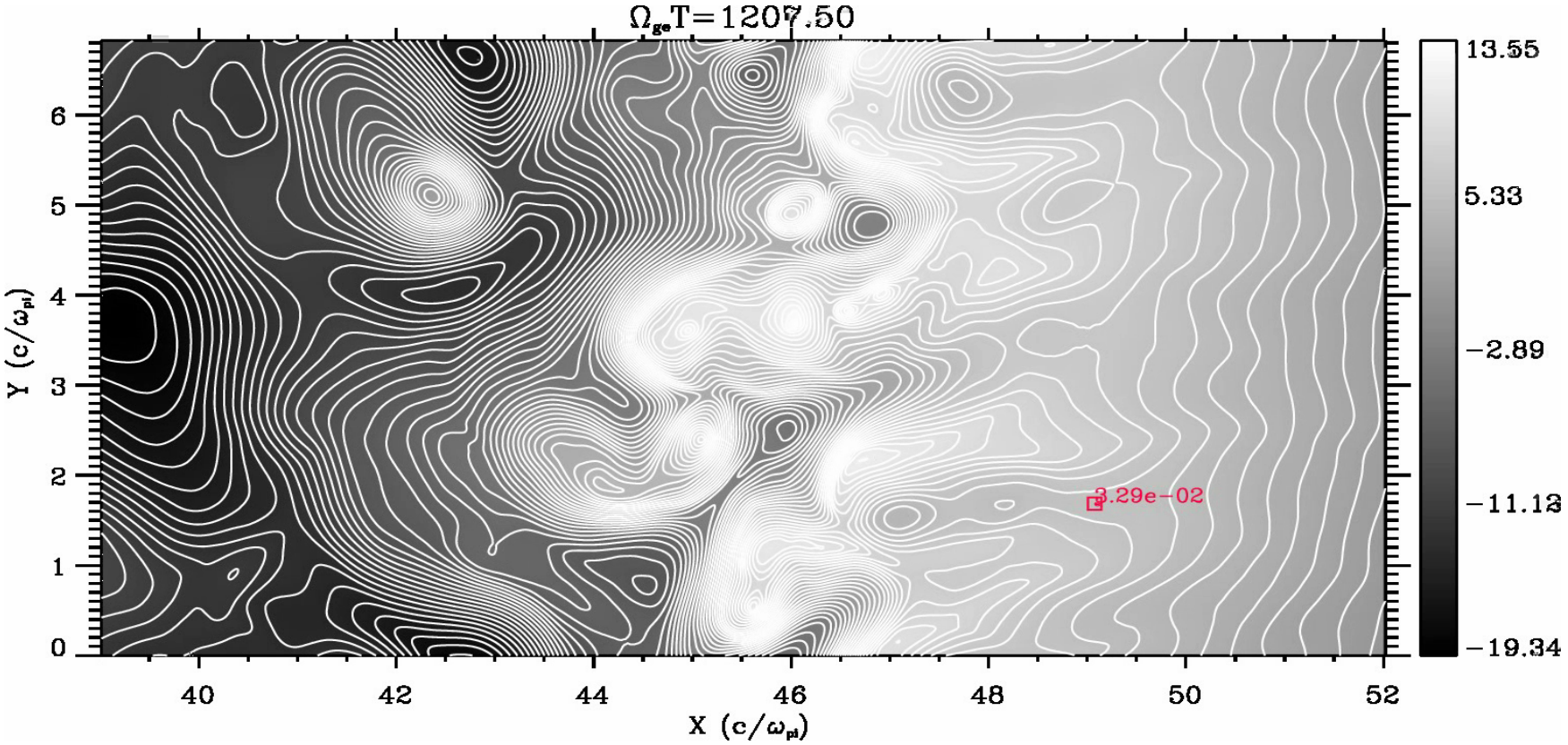
大規模計算とポスト処理

- スパコンで大規模計算して、数TBの結果が出た
- さて、どうしよう。。。
- IDLなら、データのポスト処理・解析・可視化まで可能
  （私は実際にそうしています）
  1. 並列出力データの結合
  2. データの読み込み、ポアソン方程式を解いてポテンシャル計算（一部並列処理）
  3. 可視化
- AVSではきれいな図が描けますが、AVS上でポアソン方程式を解けます？
- 実際の使用例を見てみよう！

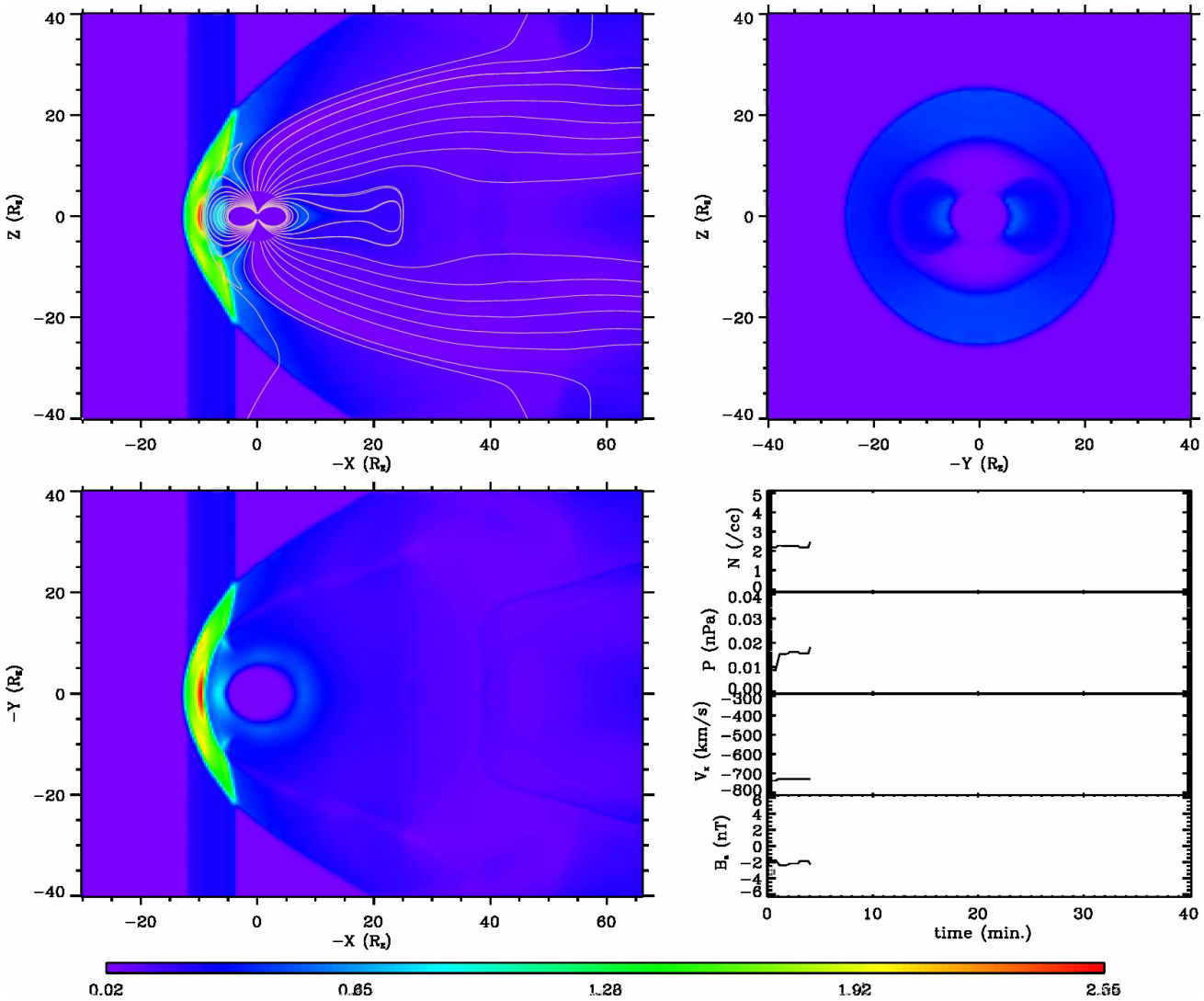
IDLでこんなことまでできます1



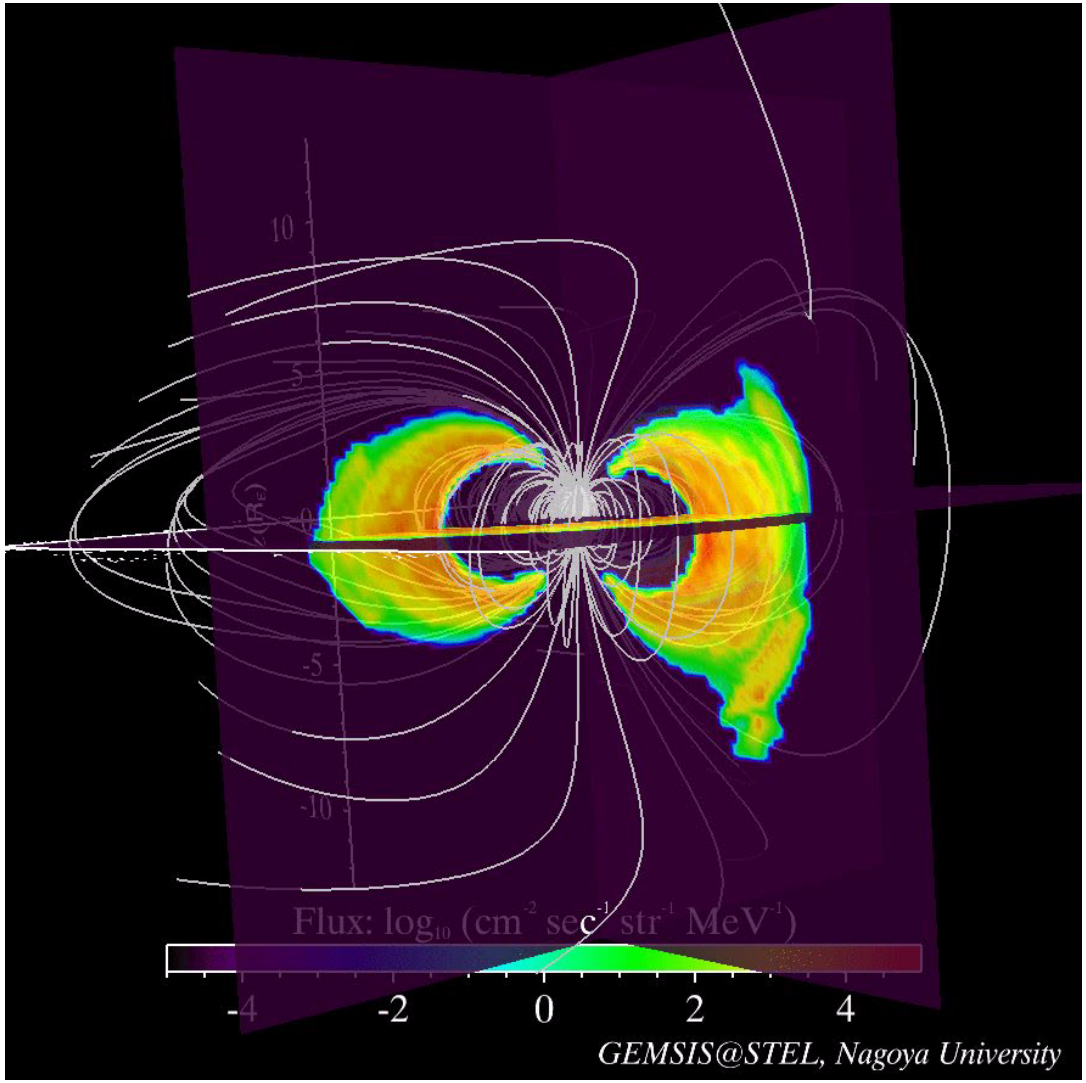
IDLでこんなことまでできます2



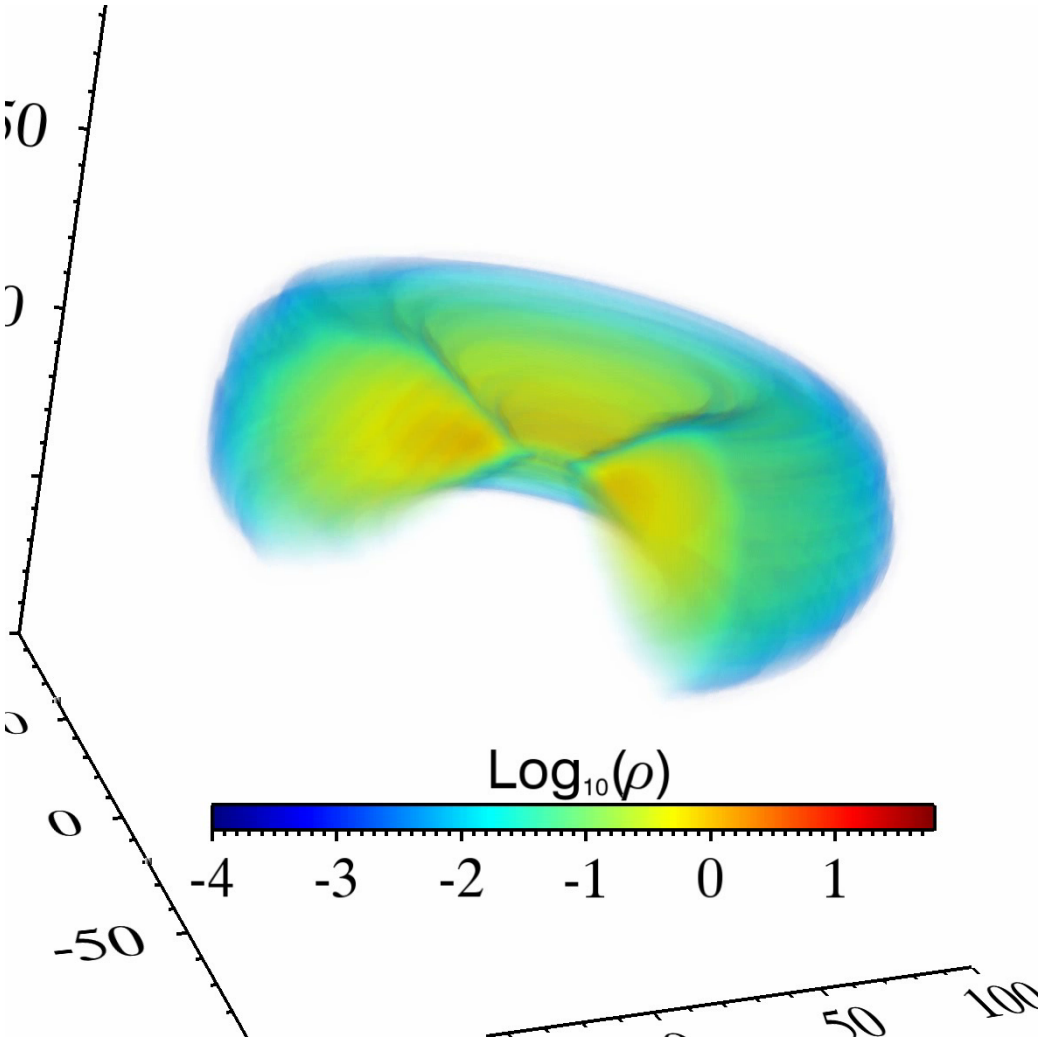
IDLでこんなことまでできます1+2



IDLでこんなことまでできます3.1

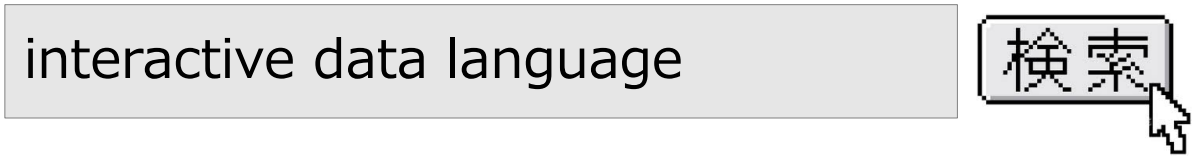


IDLでこんなことまでできます3.2



本講義のねらい

- IDLは何でもできるが、それらを全て使いこなすには時間がかかる
- 私がこれまで開発した、基本的なプロシージャ（ライブラリ）を使って、IDLによる解析を体験
- 内容は http://www.astro.phys.s.chiba-u.ac.jp/~ymatumot/idl/ がベース。ソースは公開しているので、それらを元に自由に発展させてください



環境設定

はじめに

- ~/idl/の中身を確認（setup.shで作成される）
- 実行は”idl”
- 環境変数 $IDL_STARTUP
  - idlを起動したときに実行するプログラムを指定したもの
  - tcshの場合、”setenv IDL_STARTUP ~/idl/init.pro”
  - 起動時に~/idl/init.proの内容が実行されるようになる
- サブルーチンはプロシージャと呼ばれる。拡張子は”.pro”
- IDLのPATH（次スライド）に含まれるディレクトリにあるプロシージャは、呼ばれると自動的にコンパイルされる。その際、ファイル名＝プロシージャ名である必要がある

初期設定プロシージャ (~/idl/init.pro)

```
;; set path environment
!path = expand_path('+~/idl/')+':'+!path

;; set plot style
set_plot, 'x'

;; set background color white
!p.background = 255
!p.color = 0

;; set color map for 24-bit display
device, decomposed=0, retain=2, true_color=24
loadct,12,/si
```

IDLのパスにユーザのディレクトリを追加

グラフィック環境をXウィンドウに

ウィンドウの背景を白
文字等の色を黒に

デバイスの設定
Xウィンドウでは必要

経験からカラーテーブル12
をデフォルトにしています。
線プロットのときに、使いやすいです。

データ入出力

file_read.pro

- IDLではデータの入力がやや煩雑である。そのための手続きは（User's manual 17章）、
  - openr,1,'filename' ;;ファイル名を指定して、装置番号１に割り当てる。
  - Input=fltarr(size) ;;データサイズの変数を作る
  - readf,1,input ;;変数inputにデータを読み込む

  といった命令が必要である。ここで問題なのは、データのサイズを毎回調べて指定する必要がある、命令が3回必要であるといったように、多数のデータを読み込む際には大変不便である。
- file_readは、複数のテキストデータを読み込んで配列に格納するプロシージャである

file_read.pro 使い方

```
file_read
    ディスク上にある数値データを読み込み、値を返す関数

Syntax
    result= file_read([filename][,format=format][,/string]
                      [,/silent][,/compress])

Arguments
    filename
        読み込むデータのファイル名。ワイルドカードが使え、
        パターンにマッチしたファイルを一度に読み込む。ただ
        し、読みこむ全ての数値データ配列数は等しい必要がある。

    format
        読み込むデータのデータフォーマットを指定する。

    string
        読み込むデータを文字列として読み込む。

    silent
        読み込むデータ情報の出力をしない。

    compress
        圧縮されたデータを読み込む

Example1
    IDL> data= file_read('010000_bx.dat')
    column: 256
    line: 400
    IDL> help, data
    DATA DOUBLE=Array[256,400]
```

1次元プロット

plot, oplot

- 線プロットの基本コマンド
- たくさんオプションがあるので、困ったら IDL>?

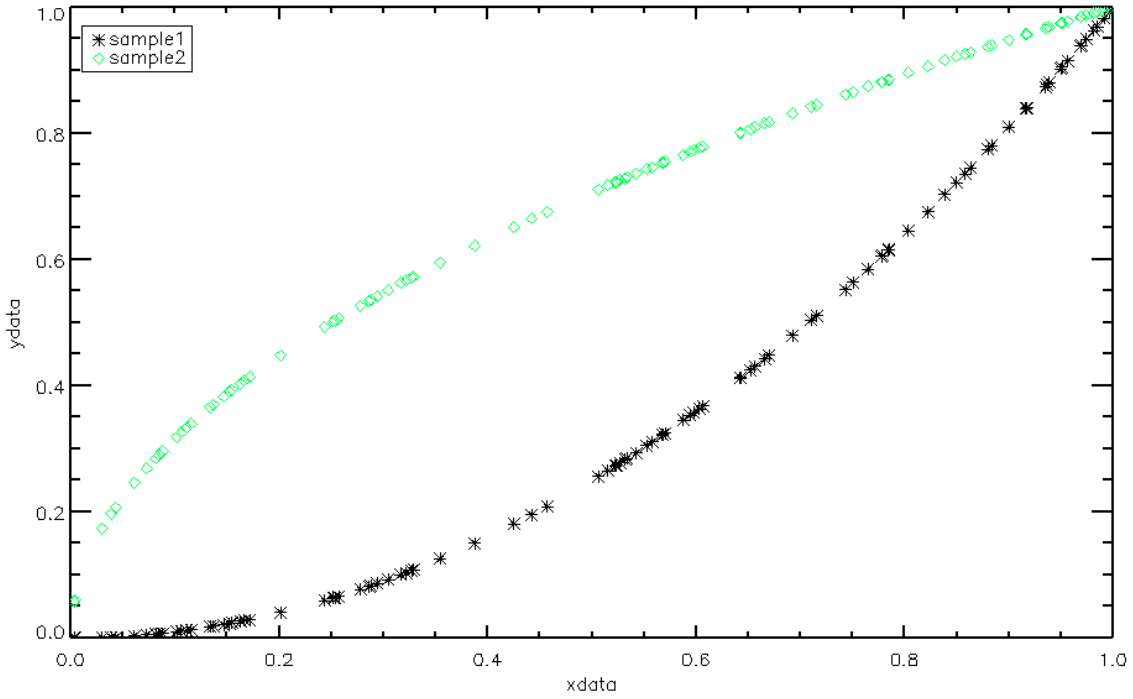
Syntax
plot, x, y, [,/xlog] [,/ylog] [,xtitle='xtitle'] [,ytitle='ytitle']
[,title='title'][,xrange=[xmin,xmax]], [,yrange=[ymin,ymax]]
[,line=line] [,psym=psym] [,charsize=charsize] ...many more!

Arguments
x, y
横軸、縦軸に対応する1次元配列のデータ

Keywords
x(y)log
横（縦）軸を対数にする
x(y)title
x（y）軸のタイトル
title
図全体へのタイトル
x(y)range
プロットする図のx（y）軸の範囲
line
プロットする線の種類
psym
点プロットする際のシンボルの指定
charsize
軸の文字のサイズ（デフォルト1.0）

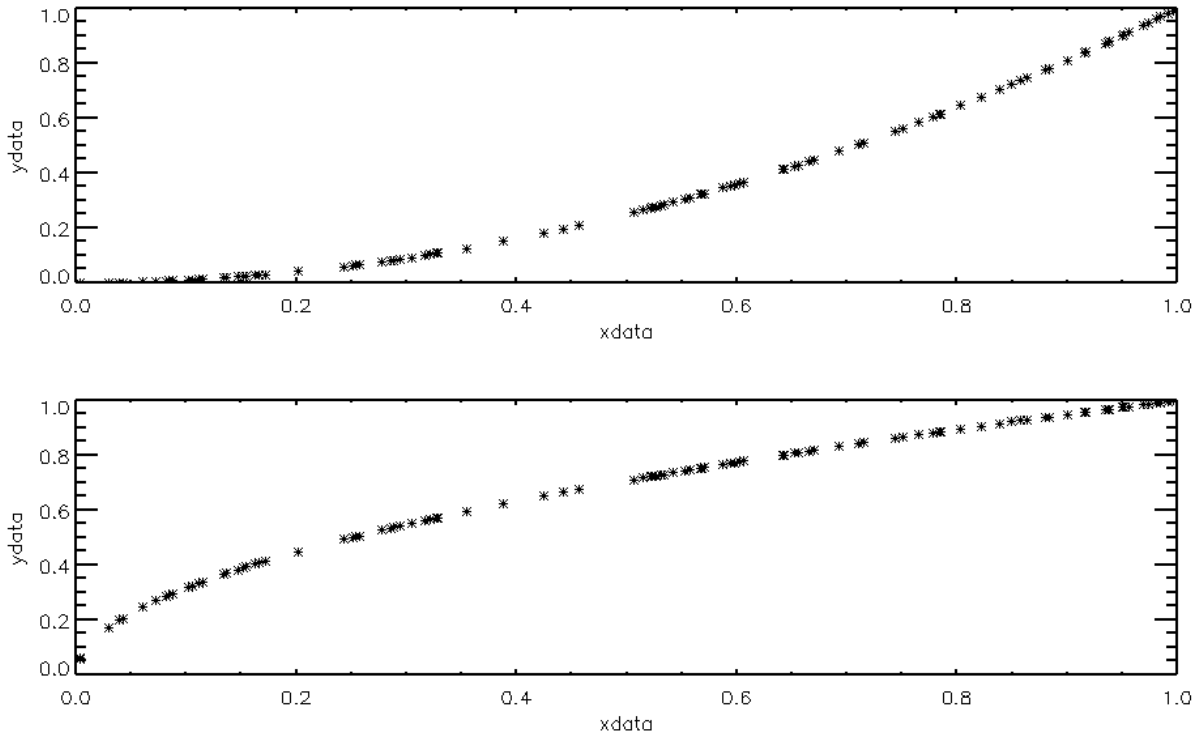
使い方例1（ほんの一例です）

```
IDL> xdata = randomu(seed,100)
IDL> plot, xdata, xdata^2 ,xrange=[0,1],yrange=[0,1],$
IDL> xtitle='xdata',ytitle='ydata',psym=2
;;違うデータを重ね描きする場合はoplotを使う
IDL> oplot, xdata, xdata^0.5 ,col=50,psym=4
IDL> legend, ['sample1','sample2'],psym=[2,4],col=[0,50]
   ;;~/idl/legend.proを使用
```



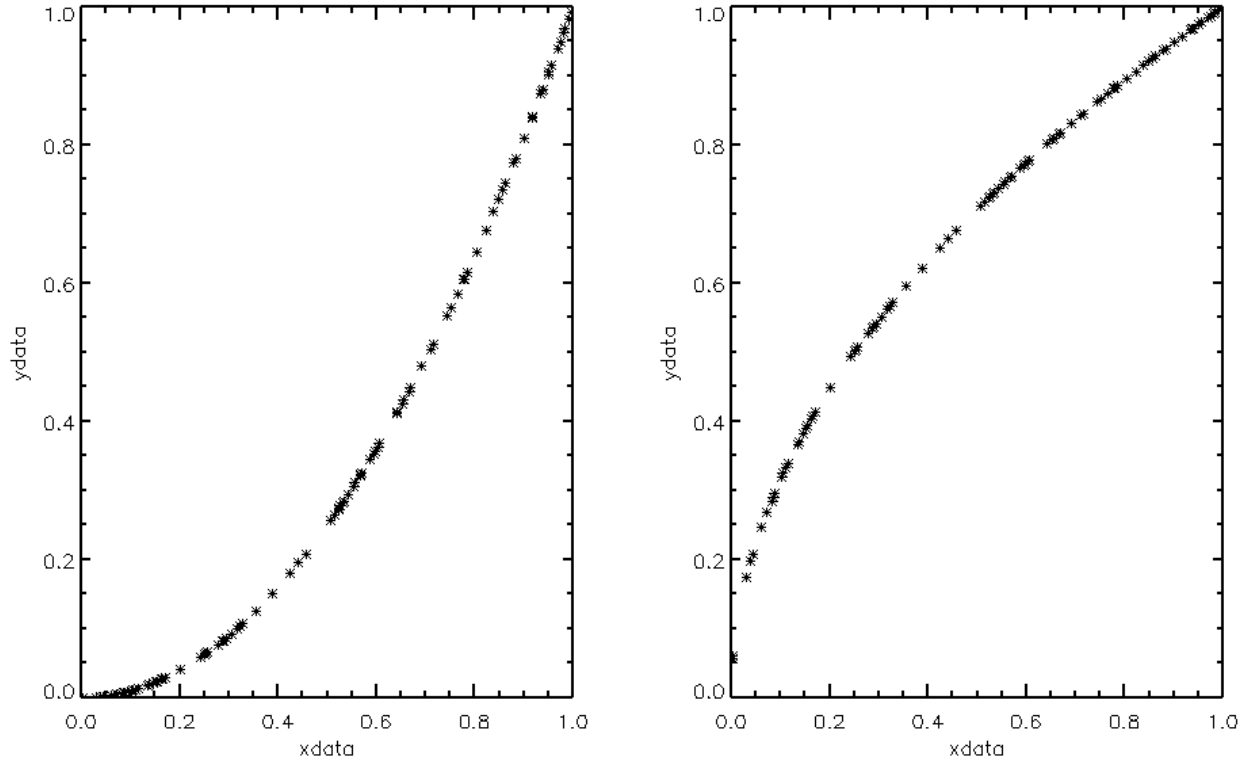
使い方例2（ほんの一例です）

```
IDL> !p.multi=[0,1,2] ;;1x2のプロット図を指定
IDL> plot,xdata, xdata^2 ,xrange=[0,1],yrange=[0,1],$
IDL> xtitle='xdata',ytitle='ydata',psym=2
IDL> plot,xdata, xdata^0.5 ,xrange=[0,1],yrange=[0,1],$
IDL> xtitle='xdata',ytitle='ydata',psym=2
IDL> !p.multi=0
```



使い方例3（ほんの一例です）

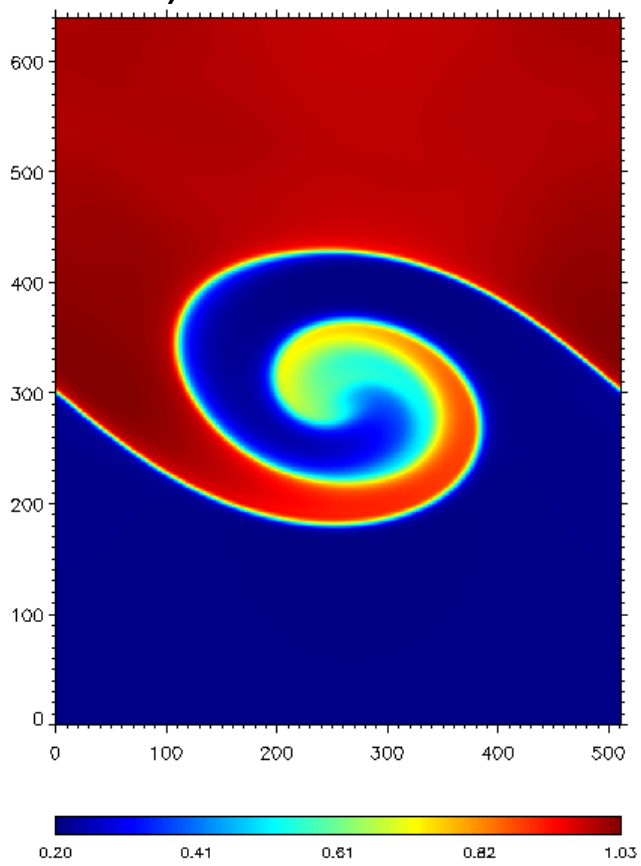
```
IDL> !p.multi=[0,2,1] ;;2x1のプロット図を指定
IDL> plot,xdata, xdata^2 ,xrange=[0,1],yrange=[0,1],$
IDL> xtitle='xdata',ytitle='ydata',psym=2
IDL> plot,xdata, xdata^0.5 ,xrange=[0,1],yrange=[0,1],$
IDL> xtitle='xdata',ytitle='ydata',psym=2
IDL> !p.multi=0
```



2次元可視化

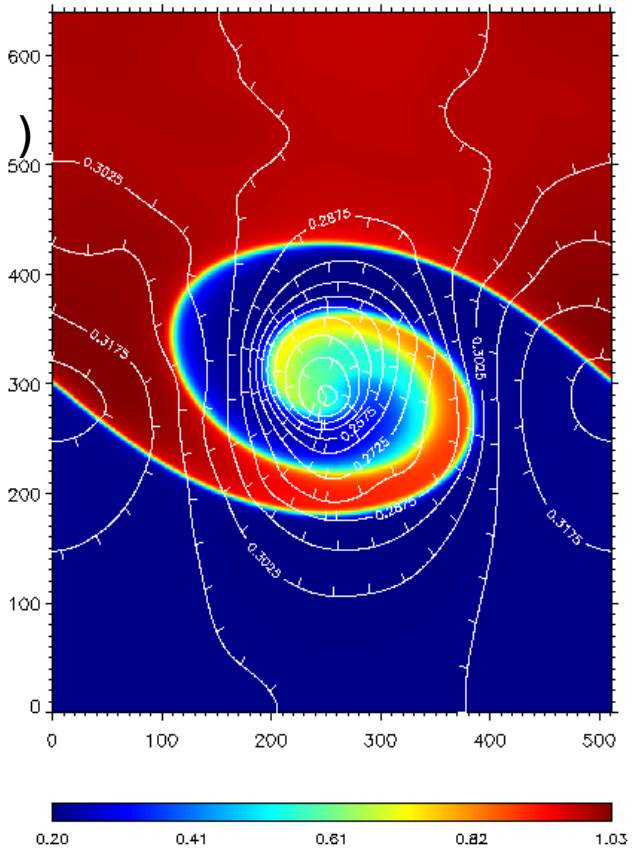
plot_clcnt.pro

- 2次元カラーコンター図
- 同ディレクトリ内の、color_bar.pro, set_window.proが必要
- 使用例：
  - `IDL> deni = file_read('024000_den_i.dat')`
  - `    column: 512`
  - `    line: 641`
  - `IDL> plot_clcnt,deni,ct=33`
- 詳細な使い方はHP



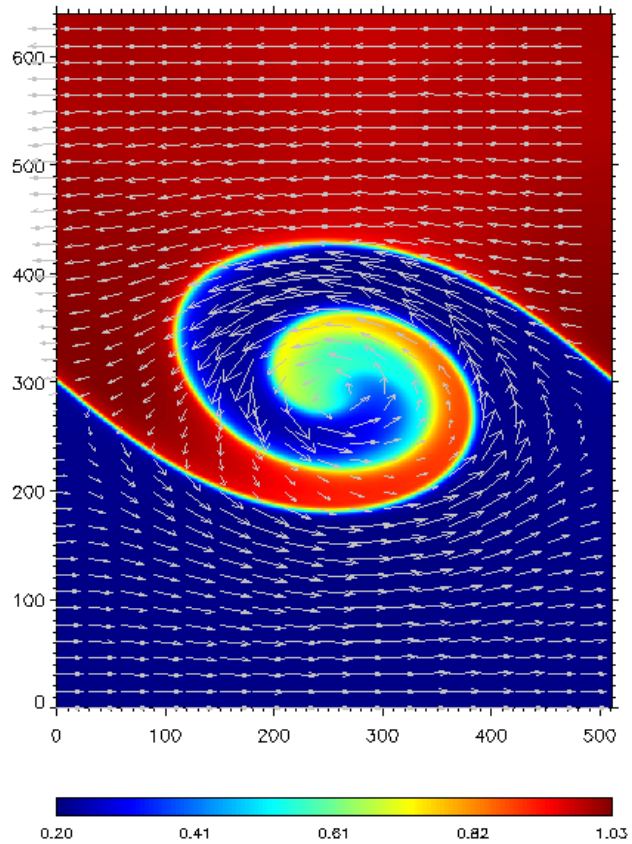
plot_cnt.pro

- ２次元配列のデータから、等高線を描く。
- 同ディレクトリ内の、set_window.proが必要
- 使用例：
  - IDL> deni = file_read('024000_den_i.dat')
  - column: 512
  - line: 641
  - IDL> pi = file_read('024000_pi.dat')
  - column: 512
  - line: 641
  - IDL> plot_clcnt,deni,ct=33
  - IDL> plot_cnt,pi
- 詳細な使い方はHP



plot_vec.pro

- 2次元配列のデータを矢印でベクトル表示する
- 同ディレクトリ内の、set_window.proが必要
- 使用例：
  - `IDL> deni = file_read('024000_den_i.dat')`
  - `    column: 512`
  - `    line: 641`
  - `IDL> vxi = file_read('024000_vxi.dat')`
  - `    column: 512`
  - `    line: 641`
  - `IDL> vyi = file_read('024000_vyi.dat')`
  - `    column: 512`
  - `    line: 641`
  - `IDL> plot_clcnt,deni,ct=33`
  - `IDL> plot_vec,vxi,vyi,30,15,color=192`
- 詳細な使い方はHP

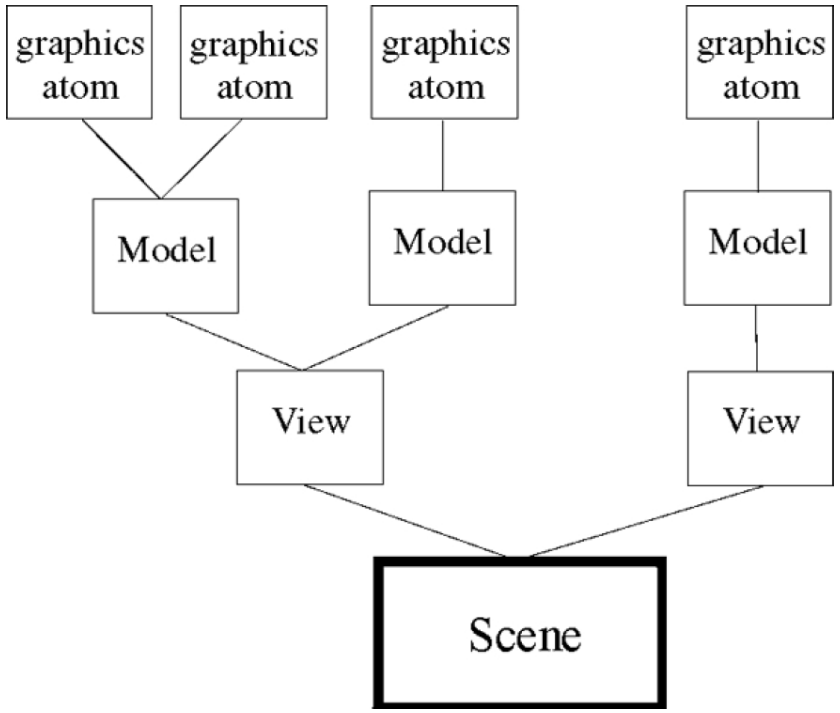


3次元可視化
（アドバンスドコース）

Direct / Object グラフィックス

- ２次元描画で使われていたこれまでの方法をDirect Graphicsと呼ぶ
- 対して、３次元可視化ではObject Graphicsの手法を用いる。
- オブジェクト指向プログラミング
- version8以降、線プロットもObject Graphicsで描画が可能になっているが、これまで見たように、２次元描画まではこれまでのDirect Graphicsで充分である

Objectグラフィックスの階層構造



- **graphics atom**：最下層に位置するもの。2次元プロット、等高線、画像、3次元ボリュームなど。
- **Model**：graphic atomsが集まり。graphic atomsに対する座標変換はこのModel単位に対して適用される。
- **View**：Modelを組み合わせたひとつの描画領域
- **Scene**：View自体で最上層になりうるが、いくつかのViewを用意した場合はSceneが最上層に位置する。
- View／Sceneは仮想的なキャンパスなので、実際に表示する先としてウィンドウ（IDLgrWindow）、クリップボード（IDLgrClipboard）などがある。

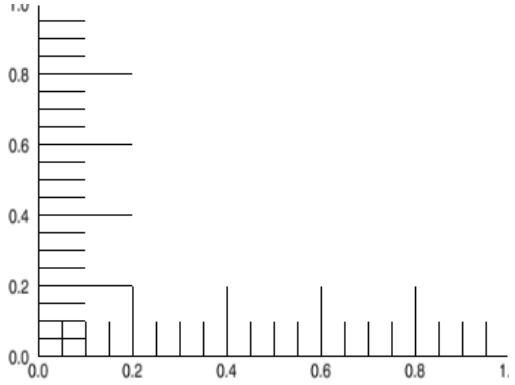
インスタンス生成

- 各インスタンス（オブジェクト）を生成する
  - `IDL> myWindow = OBJ_NEW('IDLgrWindow')`
  - `IDL> myView = OBJ_NEW('IDLgrView')`
  - `IDL> myModel = OJB_NEW('IDLgrModel')`

- graphics atom（軸）のインスタンスを生成
  - `IDL> myXaxis = OBJ_NEW('IDLgrAxis',0)` ;X軸の作成
  - `IDL> myYaxis = OBJ_NEW('IDLgrAxis',1)` ;Y軸の作成

軸を描いてみよう

- graphics atomをmodelに追加して、描画する。
  - `IDL> myModel->add, myXaxis` ;ModelにX軸を追加
  - `IDL> myModel->add, myYaxis` ;ModelにY軸を追加
  - `IDL> myView->add, myModel` ;ViewにModelを追加
  - `IDL> myWindow->draw, myView` ;Viewをウィンドウ上に描画
- ->は ″-”+”>”

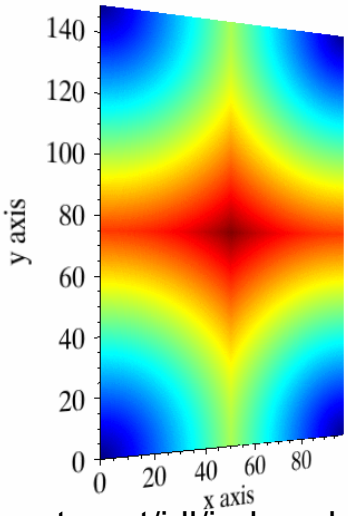


- myModelのメソッドであるaddを使って、graphics atomをmyModelに追加していることを意味する。
  また、myWindowのdrawメソッドを使って、ウィンドウ上にmyViewの内容を書き出している。

ここから先はHPを見ながら

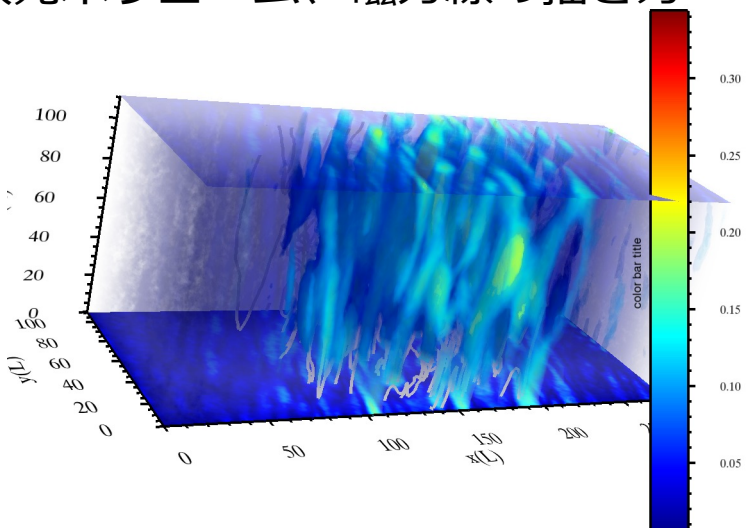
- http://www.astro.phys.s.chiba-u.ac.jp/~ymatumot/idl/index.php?3次元可視化%2FUsing IDL Objects

  2次元コンター図の3次元的描画方法



- http://www.astro.phys.s.chiba-u.ac.jp/~ymatumot/idl/index.php?3次元可視化%2F3-D Visualization

  3次元スライス図、3次元ボリューム、磁力線の描き方



図の保存・アニメーション作成

図の保存

- 描画ウィンドウを画像ファイル（png）として保存
  - リモートでIDLのウィンドウを飛ばしているときはお勧めしない
  - IDL> write_png, 'result.png', tvrd(true=1)

- ~/idl/set_ps.proを利用して、eps形式で保存
  - IDL> set_ps, 'result.eps' ;; 描画前に設定
  - IDL> plot_clcnt, data, ct=33
  - IDL> device,/close

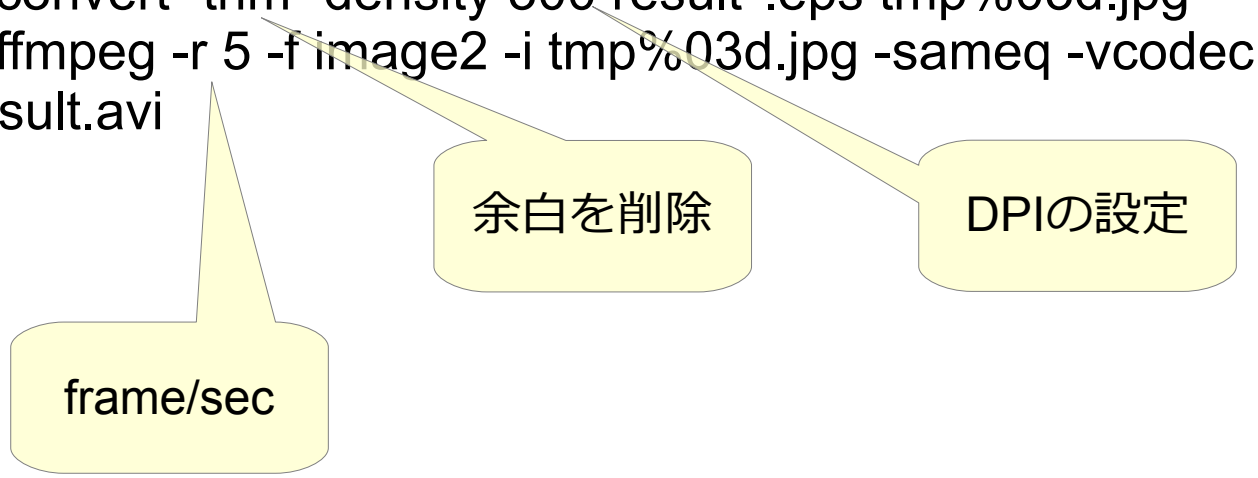
- eps形式で図を保存したほうが、論文にも転用できるので、私は基本eps形式で保存しています。

アニメーション作成

- まずはIDLのプロシージャ等で、繰り返し図を作成

- linuxコマンドconvertを使う場合（演習室端末）
  - $ convert -trim -density 300 result*.eps result.gif

- ffmpegを使う場合（研究室ワークステーション）
  - $ convert -trim -density 300 result*.eps tmp%03d.jpg
  - $ ffmpeg -r 5 -f image2 -i tmp%03d.jpg -sameq -vcodec mjpeg result.avi



プロシージャ・関数の書き方

プロシージャ：procedure_name.pro

```
pro procedure_name, a, variable1, variable2, …

a = variable1 * variable2^2

end
```

関数： function_name.pro

```
function function_name, variable1, variable2, …

a = variable1 * variable2^2
return,a

end
```

最初の呼び出しと同時にコンパイルされる。再コンパイルするには、IDL> .compile

```
IDL> procedure_name, a, 4, 3
IDL> a = function_name(4,3)
```

バッチファイルの書き方

例：make_figures.pro（連番の名前で図を保存するバッチ処理）：

```
data = file_read('00?0000_bz.dat')
info = size(data)
nx = info[1]
ny = info[2]
nt = info[3]

for it=0,nt-1 do begin
   set_ps, 'result'+strcompress(string(it,format='(i3.3)'),/remove)$
           +'.eps'
   plot_clcnt,data[*,*,it],ct=33
   device, /close
endfor

end
```

```
IDL> .run make_figures
```

で、make_figures.proの中の手続きが実行される

まとめ

- IDLができることは膨大すぎて、まとまりません。
- http://www.astro.phys.s.chiba-u.ac.jp/~ymatumot/idlを見ながら、実際に使ってみてください。
- 演習時間に私（＋簑島、高橋）に気軽に質問してください。