第 8 章

IDL を使った可視化

名古屋大学　越智 康浩

本章では、数値計算の結果を出力した 2 次元のサンプルデータを読み込み、可視化する手法の一例を紹介する。今回は Fortran プログラムで出力した 2 次元のサンプルデータを利用し、このデータを可視化する方法について述べる。IDL は 1 次元、3 次元データも扱うことができるが、ここではデータ解析の際に最も頻繁に利用する 2 次元画像の作成方法に焦点を絞る (なお AVS を使った 3 次元データの可視化については次章を参照のこと)。本章で使用するサンプルデータの作成の仕方については §8.7 を参照のこと。本章では、まず必要なメインプログラムやプロシージャについて §8.1 で確認する。次に §8.2でサンプルデータを読み込む方法について説明し、そして §8.3 で読み込んだデータをディスプレイやファイルに出力する方法について説明する。ファイルに出力する際のカラーテーブルについては §8.5 で紹介する。

8.1　準備

本節では、§8.2 以降で説明する可視化に必要なメインプログラムや設定についてまとめる。§8.1.1 では、§8.2 以降の操作に必要なメインプログラムやプロシージャそしてサンプルデータなどを一覧を示す。§8.1.2 ではこれらのメインプログラムを利用するための設定について述べる。

8.1.1　使用するファイル

本節では §8.2 以降の操作に必要なメインプログラムやプロシージャの一覧を示す。表 8.1は、§8.2, 8.3 で説明するメインプログラムの一覧である。ただしこれらのメインプログラムは全て 2 次元データ専用である。

表 8.2は、表 8.1のメインプログラムを実行する場合に必要なプロシージャである[1]。

表 8.3は §8.2, 8.3 で使用するサンプルデータの一覧である。なおこれらのサンプルデータの詳細については §8.7 を参照のこと。

これらのメインプログラムとプロシージャ、そしてサンプルデータは付属 CD-ROM の *Chap8* というディレクトリの中に収録されている。ディレクトリ *Chap8* の中はメインプログラムとプロシージャを納めた *idl* ディレクトリと、サンプルデータを納めた *data* ディレクトリに分かれている。

[1] これらのプロシージャは 横山 央明 氏 (東京大学), 松本 倫明 氏 (法政大学), 杉本 香菜子 氏 (名古屋大学), そして著者が作成または編集した。

メインプログラム	用途
read2dss.pro	データを IDL に読み込ませる
tvcn2dss.pro	カラースケール、コントア、矢印を重ねてプロットさせる
tvcn2dclosess.pro	*tvcn2dss.pro* と同じだが任意の領域だけプロットさせる
tvcn2dSLss.pro	*tvcn2dss.pro* と同じだが流線もプロットさせる
tvcn2d4pss.pro	*tvcn2dss.pro* と同じだが 4 つの画像をプロットさせる
tvcn2d6pss.pro	*tvcn2dss.pro* と同じだが 6 つの画像をプロットさせる
tvcn2dss_png.pro	*tvcn2dss.pro* と同じだが PNG ファイルを出力する
tvcn2dss_eps.pro	*tvcn2dss.pro* と同じだが EPS ファイルを出力する
write2dss_png.pro	*tvcn2dss_png.pro* と同じだが自動的に出力する

表 8.1: 本章で使用、あるいは説明するメインプログラムの一覧。ただしこれらのメインプログラムは全て 2 次元データ専用である。これらのメインプログラムは全て付属 CD-ROM の *Chap8* ディレクトリ中の *idl* ディレクトリの中に収録されている。

プロシージャ	用途
beginp.pro	画面上で画を描く領域を指定する
endp.pro	*beginp.pro* で変更されたシステム変数を元に戻す
tvcn.pro	カラースケールとコントアを描く
tvcn2ve.pro	カラースケール、コントア、矢印を描く
tvcn2ve2.pro	*tvcn2ve.pro* と同じだが、矢印の数が 2 倍多い。
vfield.pro	ベクトル場を描く
c_scale.pro	カラースケールを描く
ps1.pro	device を “X” から “PS” に切替え、 PS ファイルに画を出力する。
ps1_col.pro	“X” からカラーテーブルを取り込み、 “PS” に内挿する。
ps2.pro	*ps1.pro* で変更されたシステム変数を元に戻す。
ps_ajust_offset.pro	画が紙面の中心になるように位置を調整する。
ps_filename.pro	command line から out put file 名を読む。
ps_readfile.pro	指定したファイルから変数のラベルや文字列を読む。
ps_size.pro	command line やファイルからから画の大きさや位置を読む。。
cont_level.pro	コントアのレベルを適切に調節する。
bw.pro	白黒の EPS ファイル作成のためのカラーテーブル
rainbowf.pro	カラーの EPS ファイル作成のためのカラーテーブル
idl_startup.pro	IDL の初期化時に実行される環境設定のプログラム

表 8.2: 表 8.1のメインプログラムを実行するために必要なプロシージャの一覧。これらのプロシージャは全て付属 CD-ROM の *Chap8* ディレクトリ中の *idl* ディレクトリの中に収録されている。

サンプルデータ	用途
tsample.D	§8.2, 8.3 で主に使用するバイナリデータ
tsample.d	*tsample.D* のヘッダファイル
sample0.D	§8.3.7, 8.4.3 で使用するバイナリデータ
sample0.d	*sample0.D* のヘッダファイル
sample1.D	§8.3.7, 8.4.3 で使用するバイナリデータ
sample1.d	*sample1.D* のヘッダファイル
sample2.D	§8.3.7, 8.4.3 で使用するバイナリデータ
sample2.d	*sample2.D* のヘッダファイル
sample3.D	§8.3.7, 8.4.3 で使用するバイナリデータ
sample3.d	*sample3.D* のヘッダファイル
sample1fps.mov	§8.4.4 の方法で作成したサンプルムービー

表 8.3: 本章で使用使用するサンプルデータ。 ただしこれらのデータは全て 2 次元である。これらのサンプルデータは全て付属 CD-ROM の *Chap8* ディレクトリ中の *data* ディレクトリの中に収録されている。

8.1.2 設定

前節では使用するメインプログラムやそのために必要なプロシージャの一覧を見た。本節ではこれらのメインプログラムを使用するための設定について述べる。以降は UNIX または Linux マシンで IDL を利用する場合を想定して説明を進める。なお Windows で IDL を利用する場合の説明は割愛した。

作業は以下の 3 つである。

1. ディレクトリの作成とメインプログラム、プロシージャの保存
2. パスの設定
3. デバイスの設定

メインプログラム、プロシージャの保存

まずメインプログラム用のディレクトリと、データ用のディレクトリ 2 つを作成する。メインプログラム用のディレクトリとは 表 8.1,8.2 のメインプログラム、プロシージャを保存するためのものである。すでに IDL を利用したことがあり、IDL のメインプログラムを保存したディレクトリがある場合には特に作成する必要はなく、そのディレクトリに表 8.1, 8.2 のメインプログラム、プロシージャを保存すればよい。ディレクトリがない場合はホームディレクトリに *idl* という名前のディレクトリを作成する。同様にサンプルデータを保存するためのディレクトリ (ここでは *data* とする。) も作成する。例えば国立天文台共同利用 WS 群で使用する場合、WS にログインした後、下記のように作成する。

(ディレクトリの作成)

```
r01{ochiys}1: cd ~
/home2/ochiys
r01{ochiys}2: mkdir idl
r01{ochiys}3: mkdir data
```

ディレクトリが作成できれば、その中に表 8.1,8.2 のメインプログラム、プロシージャを *idl* に、サンプルデータを *data* に保存する。これらのメインプログラムとプロシージャ、そしてサンプルデータは付属 CD-ROM の *Chap8* というディレクトリに収録されている。ディレクトリ *Chap8* の中はメインプログラムとプロシージャを納めた *idl* ディレクトリと、サンプルデータを納めた *data* ディレクトリに分かれている。表 8.1,8.2,8.3 のメインプログラム、プロシージャ、データの合計容量は約 660KB である。念のために全てのメインプログラムとプロシージャが保存できたかどうか確認する (プロシージャが足りないと、メインプログラム実行の際エラーが生じる可能性がある)。全て保存できれば以下のように表示される。

(保存できたかどうか確認)

```
r01{ochiys}3: cd ~/idl
/home2/ochiys/idl
r01{ochiys}4: ls
README          ps1.pro             rainbowf.pro      tvcn2dss.pro
beginp.pro      ps1_col.pro         read2dss.pro      tvcn2dss_eps.pro
bw.pro          ps2.pro             tvcn.pro          tvcn2dss_png.pro
c_scale.pro     ps_ajust_offset.pro tvcn2d4pss.pro    tvcn2ve.pro
cont_level.pro  ps_filename.pro     tvcn2d6pss.pro    tvcn2ve2.pro
endp.pro        ps_readfile.pro     tvcn2dSLss.pro    vfield.pro
idl_startup.pro ps_size_offset.pro  tvcn2dclosess.pro write2dss_png.pro
r01{ochiys}5: cd ~/data
/home2/ochiys/data
r01{ochiys}6: ls
README    sample1.D      sample2.D  sample3.D tsample.d
sample0.D sample1.d      sample2.d  sample3.d
sample0.d sample1fps.mov sample2d.f tsample.D
r01{ochiys}7:
```

全て保存できていれば、次にパスの設定を行う。

パスの設定

ホームディレクトリ下の、*.cshrc* に下記のような *IDL_PATH* を設定する。環境変数 *IDL_PATH* は、ユーザが使用するメインプログラムやプロシージャ、関数のファイル (拡張子 *.pro* がつくファイル) を検索するためのディレクトリのリストである。詳細は Introduction to IDL version 5.1 (アダムネット株式会社 1998) p 31 を参照のこと。

(IDL_PATH の設定)

```
r01{ochiys}1: cat ~/.cshrc
setenv IDL_PATH \+~/idl:\+$IDL_DIR/lib:\+$IDL_DIR/examples
```

環境変数 *IDL_DIR* は、IDL メインディレクトリの場所である。ここには文字フォント、システムカラーテーブルファイルなどの初期化ファイルが保存されている。上記の設定により IDL を利用する場合、どのディレクトリで作業していても

~/idl, *$IDL_DIR/lib*, *$IDL_DIR/example*

の場所までファイルを探しに行く。これら以外の場所にプロシージャや関数のファイルがある場合にはその場所も追加する。

デバイスの設定

ここではデバイス (X ウィンド) の設定について述べる。X ウィンドの設定が不適切な場合、カラーの画像が表示されないという深刻なトラブルが生じる可能性があるので、X ウィンドの設定は重要である。

まず、IDL の環境設定のためのプログラム *idl_startup.pro* が IDL の初期化時に実行されるように設定する。この *idl_startup.pro* は §8.1.2 でメインプログラムやプロシージャを保存した際に、一緒に *~/idl* の中に保存されている。まず、§8.1.2 で編集したホームディレクトリ下の、*~/.cshrc* に下記のような環境変数 *IDL_STARTUP* の設定を追加する。

(**IDL_STARTUP の設定**)

```
r01{ochiys}2: cat ~/.cshrc
setenv IDL_PATH \+~/idl:\+$IDL_DIR/lib:\+$IDL_DIR/examples
setenv IDL_STARTUP ~/idl/idl_startup.pro
```

追加した後も下記のように設定を適用しなければならないので注意。

(**設定の適用**)

```
r01{ochiys}3: source ~/.cshrc
```

これにより、IDL を起動した際、*idl_startup.pro* に記述した設定が自動的に反映される。プログラムファイル *idl_startup.pro* の中は以下のようになっている。

(**環境設定プログラム** *idl_startup.pro*)

```
r01{ochiys}4: cat ~/idl/idl_startup.pro
DEVICE,DECOMPOSED=0,TRUE_COLOR=24,RETAIN=2
```

上記プログラムの記述について簡単に説明する。プロシージャ 「DEVICE」 は IDL に代わってユーザーがデバイスを制御するものである。プロシージャ「DEVICE」の詳細についてはオンラインガイド (起動方法は IDL を起動した状態で “?” を入力) を参照のこと。キーワード「DECOMPOSED=0」はカラーインデックスを指定する際、Red, Green, Blue のカラーベクタごとに個別にインデックスを割り当てることを意味する。最近、広く普及している 24 bit ディスプレイでは、このような方法で Red, Green, Blue のカラーベクタにそれぞれ 8 bit ずつインデックスを割り当て

ているので 16,700,000 種類の色を同時に表示させることができる。このような色の表示のさせ方を True Color と呼ぶ。True Color の詳細については Introduction to IDL version 5.1 (アダムネット株式会社 1998) p 37 を参照のこと。キーワード「TRUE_COLOR=24」は True Color ビジュアルを選択する際に指定し、値は使用しているディスプレイの 1 ピクセル当たりの bit 数 (色数: color depth) を選ぶ。ここでは国立天文台共同利用 WS 群、同 “可視化システム” 高速画像処理機器群で IDL を利用することを想定し、それらのディスプレイの色数は 24 であったので、上記のように設定した[2]。これら 2 つのキーワードが設定されていないと、画像がカラーで表示されない可能性もあるので忘れずに設定して欲しい。最後のキーワード「RETAIN=2」は Backing store の設定である。値を 「2」 と指定することで、IDL のウィンドが他のウィンドと重なって消えた部分を再描画するためのデータを IDL に保存させる。このように設定しておかないと、ウィンドが例えばターミナルと重なる毎に再描画させなければならないので大変効率が悪い。

8.2 データの読み込み

本節ではサンプルデータを IDL に読み込ませる方法について述べる。ここで想定しているのは UNIX マシン (ex. 国立天文台共同利用 WS 群, VPP5000) で出力したサンプルデータを UNIX マシン (ex. 同 WS 群, “可視化システム” 高速画像処理機器群) で可視化する場合である。Linux マシンまたは UNIX マシンで出力したデータを Linux マシンで可視化する場合もほとんど同じである [3]。

本節ではサンプルデータは与えられたものとして、それを IDL に読み込ませる所から見ていく。なお本節で用いるサンプルデータを作成するためのソースプログラムや生成されるファイルについては §8.7.1を参照のこと。

まずサンプルデータがあるディレクトリ (*data*) に移動し、IDL を起動する。そして *.r read2dss* と入力する。

(**IDL の起動と** *read2dss.pro* **の実行**)

```
r01{ochiys}1: cd ~/data
r01{ochiys}2: idl
IDL Version 5.5, Solaris (sunos sparc). (c) 2001, Research Systems, Inc.
Installation number: 70630.
Licensed for use by: NAO

IDL> .r read2dss
% Compiled module: $MAIN$.
Input file name to read =
:
```

すると、ファイル名を入力するように聞かれるので「tsample」と入力する「(「tsample.d」や「tsample.D」のように拡張子をつけないことに注意)。もし *.r read2dss* と入力した時に「No such file or directory」 というメッセージが表示された場合にはパスが通っていない

[2] リモートログインして使用する場合についても、最近では、多くのパソコンのディスプレイも色数が 24 以上に対応しているので上記の色数の設定でも恐らく問題ないと考えた

[3] ただし UNIX マシンで出力したバイナリデータを Linux マシンで可視化する場合、バイナリデータを読み込む際にオプションが必要である。詳細は §8.6.2 を参照

可能性があるのでパスを通す必要がある。詳細は §8.1.2、または §8.6.1 を参照。

(ファイル名の入力)

```
IDL> .r read2dss
% Compiled module: $MAIN$.
Input file name to read =
:tsample [RET]
t=      0.00000
IDL>
```

これで IDL にデータが読み込まれた。*read2dss.pro* の中は下記のようになっている。

(*read2dss.pro*)

```
; read2d.pro
;    to read a data file generated by 2D code
;-
;
fn='file name'
print, 'Input file name to read = '
read, fn                                    ; ファイル名の読み込み
;
ctitle='title'
precision='FLOAT'
timei=0.
;
OPENR, Unit1, fn+'.d', /GET_LUN                      ;tsample.d を開く
READF, Unit1, ctitle, FORMAT='(A11)'                 ; ファイルタイトル、
READF, Unit1, ixs, ixe, iys, iye, FORMAT='(4I5)'     ; 配列の範囲 (x 方向、
READF, Unit1, timei, FORMAT='(/,/,/,/,7X,F15.7)'     ;y 方向)、時刻
READF, Unit1, precision, FORMAT='(A6)'               ; データ型の読み込み
FREE_LUN, Unit1
;
case precision of
'FLOAT ': begin
;print,'data type: ',precision
x = FLTARR(ixe-ixs+1)                       ; 配列の定義 (float 型の場合)
y = FLTARR(iye-iys+1)
rho = FLTARR(ixe-ixs+1, iye-iys+1)
vx = FLTARR(ixe-ixs+1, iye-iys+1)
vy = FLTARR(ixe-ixs+1, iye-iys+1)
bx = FLTARR(ixe-ixs+1, iye-iys+1)
```

```
by = FLTARR(ixe-ixs+1, iye-iys+1)
end
else: begin
;print,'data type: ',precision              ; 配列の定義 (float 型以外の場合)
x = DBLARR(ixe-ixs+1)
y = DBLARR(iye-iys+1)
rho = DBLARR(ixe-ixs+1, iye-iys+1)
vx = DBLARR(ixe-ixs+1, iye-iys+1)
vy = DBLARR(ixe-ixs+1, iye-iys+1)
bx = DBLARR(ixe-ixs+1, iye-iys+1)
by = DBLARR(ixe-ixs+1, iye-iys+1)
end
endcase
;                                                    ;tsample.D から
OPENR, Unit2, fn+'.D', /F77_UNFORMATTED, /GET_LUN    ; 配列の読み込み
READU, Unit2, x
READU, Unit2, y
READU, Unit2, rho
READU, Unit2, vx
READU, Unit2, vy
READU, Unit2, bx
READU, Unit2, by
FREE_LUN, Unit2
;
print,'t=',timei
END
```

8.3 tvcn による図の作成

前節では、IDL に 2 次元のサンプルデータを読み込ませた。本節では、このデータを画像として出力する方法について述べる。ここではカラースケール、コントア、矢印を重ねた図 8.1を作成する方法を見ていく。使用するメインプログラムは *tvcn2dss.pro* である。

8.3.1 表示したい物理量の選択

まず、前節のようにサンプルデータを読み込ませた後、 *tvcn2dss.pro* を起動する。カラーで表示させるためには、 *tvcn2dss.pro* を起動する前にカラーテーブルをロードしておく。

(*tvcn2dss* の起動)

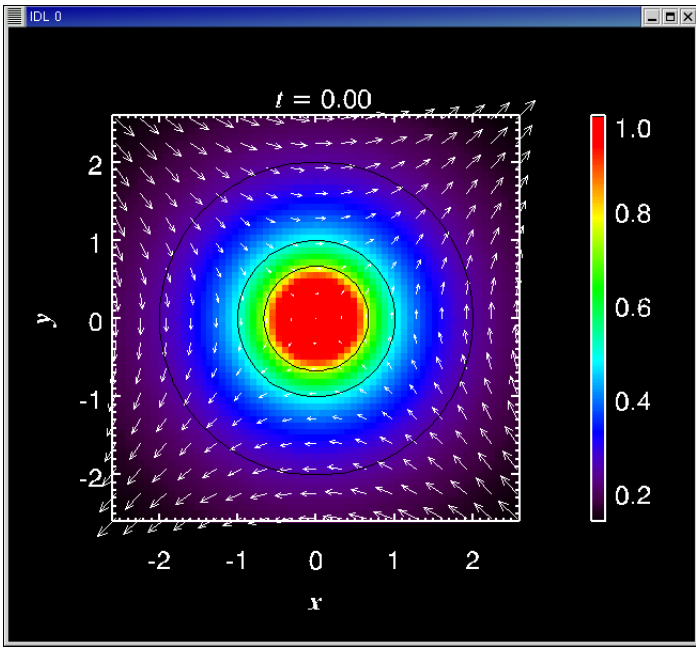

図 8.1: カラースケール、コントア、矢印を重ねた図

```
IDL> .r read2dss
% Compiled module: $MAIN$.
Input file name to read =
: tsample
t=      0.00000
IDL> loadct,39
% Compiled module: LOADCT.
% Compiled module: FILEPATH.
% Compiled module: PATH_SEP.
% LOADCT: Loading table Rainbow + white
IDL> .r tvcn2dss
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:
```

カラーテーブルは、もともと IDL に 41 個 (0 - 40 番) 添付されている。ここでは 39 番 (“Rainbow + white”) を使用する。メインプログラム *tvcn2dss.pro* を実行すると、まず最初にカラースケールに用いる物理量を選択するように聞かれるので、カラースケールに用いたい物理量を *rho, vx, vy, bx, by* などの中から選択する。選択できる物理量は *tvcn2dss.pro* の 30 行目付近に定義されている。

(*tvcn2dss* **30 行目付近**)

```
phys=rho
print,'Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"'
read,' for color scale,:',parameter
case parameter of
    'rho':phys=rho                          ; 選択可能な物理量
    'vx':phys=vx                            ; とその定義
    'vy':phys=vy                            ;
    'bx':phys=bx                            ;
    'by':phys=by                            ;
    'lrho':phys=alog10(rho)                 ;
    'vt':phys=sqrt(vx^2+vy^2)               ;
    'bt':phys=sqrt(bx^2+by^2)               ;
    'em':phys=(bx^2+by^2)/(8*!pi)           ;
    'va':phys=sqrt(bx^2+by^2)/(4*!pi*rho)   ;
    '':phys=physoldt                        ;
else:begin
    print,'"lrho" will be used!'
    phys=alog10(rho)
    end
endcase
physoldt=phys
```

これを見ると分かる通り、*rho* を選択すると、*phys* という配列に *rho* が格納される。もし *rho, vx, vy, bx, by, lrho, vt, bt, em, va* または空白以外を入力すると、*rho* の常用対数が *phys* に格納される。もし表示させたい物理量が上記以外のものであれば、各自で *tvcn2dss.pro* の中に追加してもらいたい。

カラースケールに用いる物理量を入力すると、次にコントアに用いる。物理量を選択するように聞かれるのでカラースケールの場合と同様に選択する。コントアに使用できる物理量もカラースケールに使用できるものと同じである。ここではカラースケール、コントアともに *rho* を選択する。

(**カラースケール、コントアの選択**)

```
IDL> .r tvcn2dss
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho          ; カラースケールの選択
 for contour.:rho              ; コントアの選択
Input "v" or "b"
 for vector.:
```

次に vector (矢印) を選択するように聞かれるので,*v* または *b* を選択する。*v* は x 成分に *vx*, y 成分に *vy* を用いる。 *b* は x 成分に *bx*, y 成分に *by* を用いる。 ここでは *v* を選択する。

(**矢印の選択**)

```
IDL> .r tvcn2dss
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho
 for contour.:rho
Input "v" or "b"
 for vector.:v                  ; 矢印の選択
x-y slice
velocity scale=      56.2501
IDL>
```

するとウィンドが開き、図 8.1 のような画像が表示されるはずである。

8.3.2 カラースケール範囲の指定

§8.3.1 では、カラースケールで表示させた物理量 (*rho*) はその最小値から最大値まですべて表した。しかし、実際には特定のスケールだけ詳しく見たい (例えば高密度の部分だけ細かく見る) 場合も考えられる。その場合にはカラースケールの範囲を指定することで、特定のスケールを細かく表示させることが可能である。 カラースケールの範囲は *tvcn2dss.pro* の 76 行目付近の「t_min」と「t_max」で指定されている。

(*tvcn2dss.pro* **76 行目付近**)

```
; parameters for tv
t_max=max(phys)
t_min=min(phys)
```

上記の設定では、カラースケールの最大値、最小値は配列 *phys* (ここでは *rho*) の最大値、最小値に一致している。特定のスケールレンジを細かく表示させるには、例えば下記のように書き換える。

(カラースケールの最大値、最小値の指定)

```
; parameters for tv
t_max=1.0
t_min=0.5
```

このように指定すると図 8.2のように特定のスケールレンジ ($0.5 \leq rho \leq 1.0$) を細かく表示できる。

8.3.3 コントアや矢印の指定

§8.3.2 では、カラーで表示する物理量のスケールレンジを指定した。ここではコントアや矢印を指定する方法を述べる。まず初めに、コントアレベルを指定する方法から見ていく。コントアレベルは *tvcn2dss.pro* の 82 行目付近で指定されている。

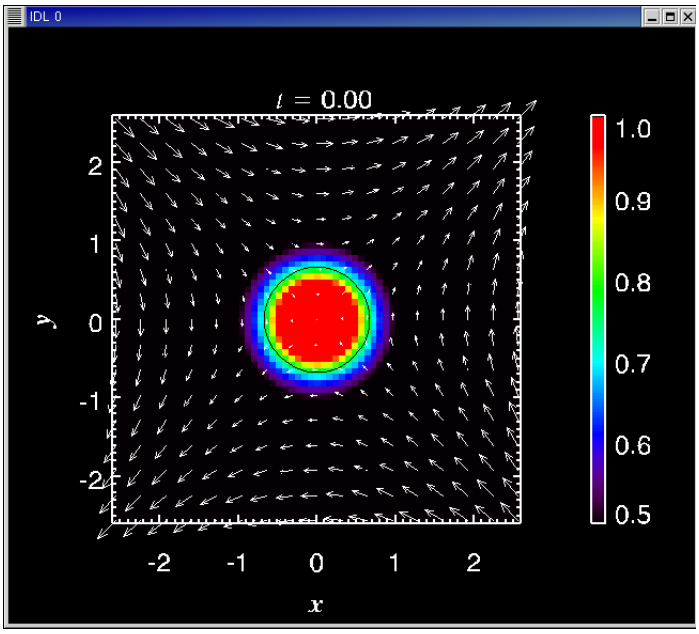

図 8.2: 図 8.1と同じであるが特定のスケールレンジ ($0.5 \leq rho \leq 1.0$) を細かく表示させた。

(*tvcn2dss.pro* **82 行目付近**)

```
; parameters for contour
c_lev=findgen(4)/4
c2_lev=findgen(16)/4-4
```

上の設定は図 8.1を作成したときの設定である。 コントアレベルの指定には 2 つの変数 (c_lev と c2_lev) が用いられている。従って、例えば一方は間隔を細かくし、もう一方は間隔を広くするという使い方もできる。上の設定では一方のコントアレベル (c_lev) は 0 (ゼロ) から 0.75 までの範囲を 0.25 きざみで区切り、もう一方のコントアレベル (c2_lev) は 4 から 0 までの範囲を 0.25 きざみで区切っている。 例えば以下のように書き換えると、図 8.3のようになる。

(**コントアレベルの指定**)

```
; parameters for contour
c_lev=findgen(10)*0.05+0.5
c2_lev=findgen(10)*0.05
```

ちなみに、コントアの色も 「c_lev」と「c2_lev」 で別々に設定することができる。コントアレベル 「c_lev」と「c2_lev」 の色の指定は *tvcn2dss.pro* の 127 行目付近にある 「c_col」と「c2_col」である。下のように指定すると「c_lev」で指定したコントアは黒、「c2_lev」で指定したコントアは白になる。

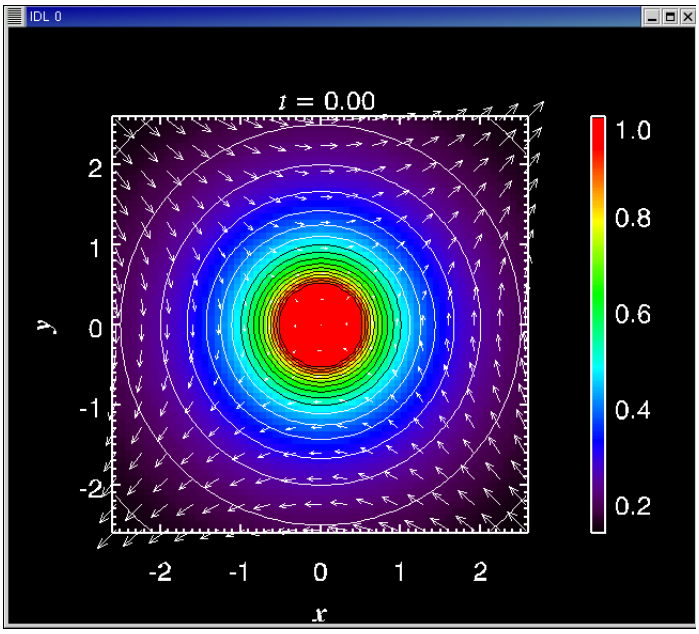

図 8.3: 図 8.1と同じであるがコントアの間隔をより狭くした。

(コントアの色)

```
levels=c_lev,c_col=1,$
l2evels=c2_lev,c2_col=0
```

ちなみに上記の 1 行目の最後にある「$」記号はプログラム行を継続させる、という意味である。次に矢印について見ていく。矢印の間隔の指定は *tvcn2dss.pro* の 87 行目付近で指定されている。

(*tvcn2dss.pro* **87 行目付近**)

```
; parameters for vector
i_skip=ixe/16
j_skip=iye/16
```

変数 「i_skip」、「j_skip」 はそれぞれ x, y 方向で、矢印の始点の間隔 (スキップ) を表す。変数「ixe」、「iye」はそれぞれ x, y 方向のメッシュの数 (今読み込んでいるデータ の場合 64) であるからこの場合、i_skip = j_skip = 4 (メッシュ) となる。従って、矢印の始点は 4 メッシュごとに並んでいる。 言い替えると、x, y 方向それぞれに 17 個の矢印が表示される。 下記のように書き換えると、矢印は x, y 方向それぞれに 26 個ずつ配置され、それは図 8.4のようになる。

(矢印間隔の指定)

```
; parameters for vector
i_skip=ixe/25
j_skip=iye/25
```

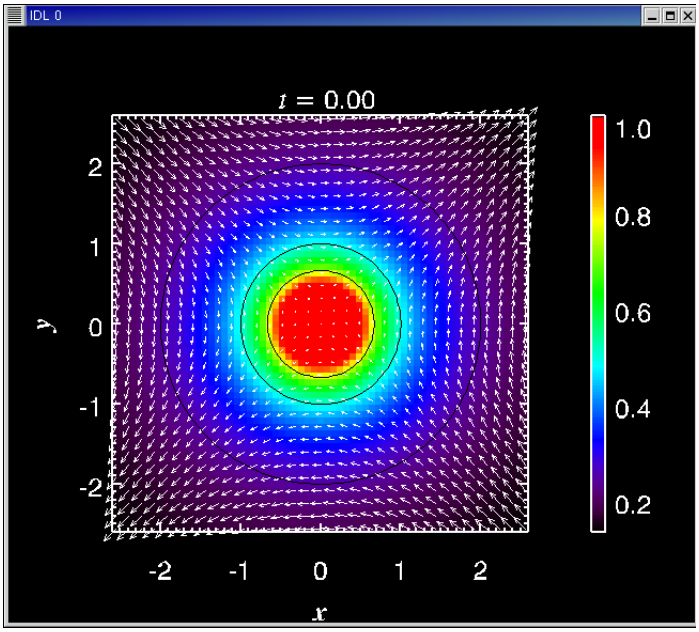

図 8.4: 図 8.1と同じであるが、矢印の数を増やした。

次に、矢印の長さについて述べる。矢印の長さは *tvcn2dss.pro* 113 行目付近に指定されている。

(*tvcn2dss.pro* **113 行目付近**)

```
v_scale=long(max(sqrt(sv1^2+sv2^2)))/((x(1)-x(0))*i_skip)
if (v_scale le 0.) then $
  v_scale=max(sqrt(sv1^2+sv2^2))/((x(1)-x(0))*i_skip)
```

上の設定では $v = \sqrt{vx^2 + vy^2}$ の最大値を基にスケールされている (今の場合 $sv1 = vx$, $sv2 = vy$ である)。現在の「v_scale」の値は「*tvcn2dss.pro*」を実行した際、コマンドライン上に、「velocity scale= ****」という形式で表示される。

(**v_scale の値**)

```
IDL> .r tvcn2dss
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho
 for contour.:rho
Input "v" or "b"
 for vector.:v
x-y slice
velocity scale=      56.2501    ; v_scale の値
IDL>
```

もし任意の速度スケールを固定したい場合は、例えば下記のように変数 「v_scale」 に定数を代入し、他の 「v_scale」についての記述はコメントアウトする。代入する値は表示される 「v_scale」の値を参考にすると良い。ただし、矢印の長さは代入した「v_scale」 の値に反比例することに注意。

(矢印の長さの指定)

```
v_scale=30.0
```

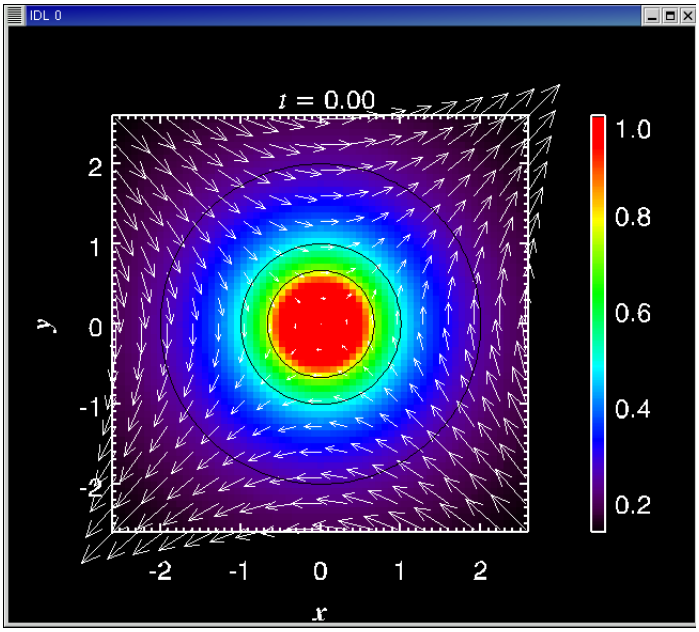

図 8.5: 図 8.1と同じであるが、矢印を長くした。

矢印の色も指定することができる。例えば、矢印の色とカラースケールの色が近く、区別がつきにくい場合には適宜変更してもらいたい。矢印の色の指定は *tvcn2dss.pro* の 124 行目にある 「v_col」に 0 から 255 の値を代入することでできる。もし 「v_col=−1」 とすると、矢印の色は白色になる。下記は図 8.1を作成した時の設定である。

(*twcn2dss.pro* **124 行目: 矢印の色**)

```
iskip=i_skip,jskip=j_skip,v_col=-1,scale=v_scale,$
```

8.3.4 タイトルの指定

前節までに表示させる物理量を選択したり、カラースケール、コントア、矢印を調節する方法について見てきた。本節では画面に表示させるタイトルの指定方法について見ていく。

図 8.1では、時間をタイトルとした。タイトルは *twcn2dss.pro* の 129 行目付近で指定している。

(*twcn2dss.pro* **129 行目付近**)

```
; parameters for title
stime=STRING(FORMAT='(F5.2)',timei)
xyouts,-max(x)*0.2,max(y)*1.05,/data,'!8t!3 ='+stime
```

「stime」には時刻 「timei」の値を文字列に置き換えたものが入る。関数「STRING」は引数を文字列に変換する。プロシージャ「xyouts」は文字列を任意の位置に表示させる。ここでは「/data」によりデータ座標系で位置を指定している (データ座標の詳細は Introduction to IDL version 5.1 (アダムネット株式会社 1998) p94 を参照)。インデックス「!8」 や 「!3」 は文字の種類を指定する。インデックス「!8」は「Complex Italian」、「!3」は「Simplex Roman」を表す。その他の種類については Introduction to IDL version 5.1 (アダムネット株式会社 1998) の p109 表 7 を参照。

文字列以外にも、任意の長さの線を表示させることも可能である。例えば、基準の速度の大きさを矢印で表し、それとデータ領域内の矢印と比較することができる。

(**タイトルに矢印を追加する方法**)

```
v0=5.0
arrow,max(x)*0.6,max(y)*1.10,max(x)*0.6+(v0/v_scale),max(y)*1.10,/data,$
hsize=30.*v0/v_scale
xyouts,max(x)*0.65+(v0/v_scale),max(y)*1.09,'!8v=1.0 v!D0!N!3',/data
```

図 8.1を作成する時、矢印に v (速度) を選択した。ここでは基準速度を想定し、それを 「v_0」 と書くことにする。そして、この 「v_0」の大きさを持つ矢印をタイトルの位置に表示させることを考える。図 8.6はこのプログラムを使って矢印をタイトルの位置に表示させた例である。基準速度 「$v0$」の大きさは利用時に適宜変更してもらいたい。ここでは仮に「5.0」 と置くことにする。プロシージャ「arrow」は頭のついた矢印を表示するものである。ここでは長さが 「$v0$」 の矢印を表示させるように指示している (値 「$v0$」 を「v_scale」 でスケールしている理由は、データ領域内の矢印も同じ 「v_scale」でスケールしているから)。プロシージャ「arrow」の詳細については IDL Reference Guide A-M version 5.3 (Research Systems, Inc. 1999) p 51 を参照のこと。 上記プログラムの最後の行は 「$v = 1.0\,v_0$」 という文字列を、上の行で出力した矢印の隣に配置するように指示している。最後の行の中で「!D」はそれに続く「0」を下付添字に指定している。インデックス「!N」は下

付 (または上付) 添字から元に戻すための指示である。ちなみに上付添字は「!U」で指定できる。

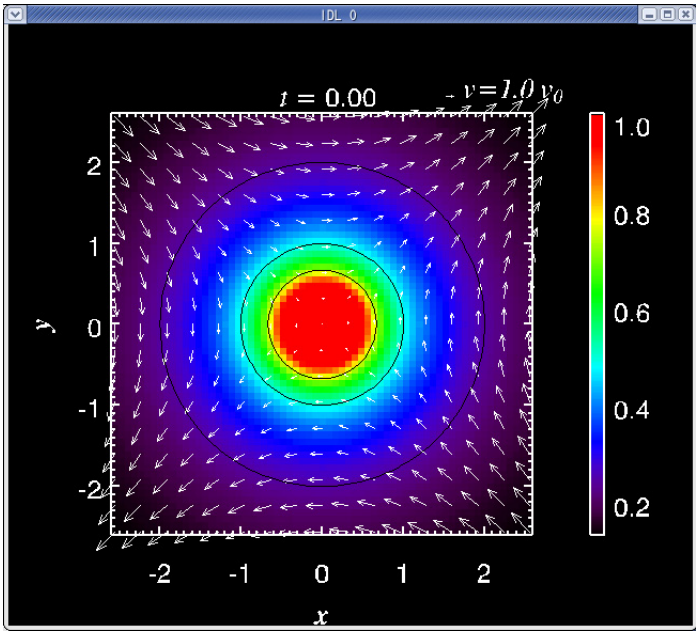

図 8.6: 図 8.1と同じであるが、タイトルの位置に基準速度 「$v0$」 と同じ大きさをもつ矢印 (図右上) を追加した。

8.3.5 文字の指定

図中の文字の大きさや線の太さは *twcn2dss.pro* の 17 -22 行目に指定されている。

(*twcn2dss.pro* **17 - 22 行目**)

```
!p.font=1               ; graphics text font system を指定
!p.charsize=3           ; 文字の大きさの指定
!x.thick=2.5            ; x 方向に沿った線の太さ
!y.thick=2.5            ; y 方向に沿った線の太さ
!z.thick=2.5            ; z 方向に沿った線の太さ
!p.thick=1.5            ; コントアや矢印の太さ
```

最初の *!p.font* は graphics text font を指定するもので、上記のように整数「1」を代入すると、True Type font systemが使用される。 True Type font system では、文字は多角形の集合として表される (ほとんど画像として扱われる)。もし *!p.font* に値「0」を代入すると文字は device font (hardware font) として扱われる。 device font では文字がディスプレイなどのデバイスから直接与えられる (デバイスには依存するが、文字は文字のまま扱われる)。どちらのタイプの font を選択するかについては、もし画像をポストスクリプトファイルとして出力して後からイラストレータなどで文字を編集することが想定される場合には、 device font を選択することを勧める。なお graphics

text font の詳細については IDL Reference Guide (Objects & Appendixes) version 5.3 (Research Systems, Inc. 1999) p 2196 以降を参照のこと。

最後の *!p.thick* はコントアや矢印などの線の太さを指定する。記号 *!p* に続く変数はプロットシステム変数と呼ばれ、全てのプログラムユニットが使用できる定義済の変数である。上記のように定義することにで、このプログラムの中では線の太さは *thick=1.5* に指定される。システム変数の詳細については Introduction to IDL version 5.1 (アダムネット株式会社 1998) p 35 -37, IDL Reference Guide (Objects & Appendixes) version 5.3 (Research Systems, Inc. 1999) p 2157 以降を参照のこと。

8.3.6 一部の領域だけ拡大表示させる方法

ここまでは IDL に読み込ませた配列の全領域を表示させてきた。しかし、場合によっては一部の領域だけ可視化すれば良いこともある。そこで本節では読み込んだ配列の内、一部の領域だけ拡大し、可視化する方法について見ていく。

一部の領域だけ拡大するために、まず *read2dss.pro* を使って IDL に配列全てを読み込ませ、次に各配列の必要な部分だけを格納した新たな配列を作成し、その新たな配列を可視化するという方法をとる。新たな配列を作成し、それを基に可視化するメインプログラムが *tvcn2dclosess.pro* である。この *tvcn2dclosess.pro* を使って、図 8.6の中心部分だけ拡大したのが、図 8.7である。

(*tvcn2dclosess.pro* の実行)

```
IDL> .r read2dss                    ; データの読み込み
% Compiled module: $MAIN$.
Input file name to read =
: tsample
t=      0.00000
IDL> .r tvcn2dclosess               ;tvcn2dclosess の実行
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho               ; カラースケールの選択
 for contour.:rho                   ; コントアの選択
Input "v" or "b"
 for vector.:v                      ; 矢印の選択
x-y slice
velocity scale=      36.9231
IDL>
```

全領域を可視化する *tvcn2dss.pro* の大きな違いは、可視化する部分の情報だけを持った配列を新たに作成することである。下記のメインプログラムは *tvcn2dclosess.pro* の中で、可視化する部分の情報だけを持った配列を新たに作成している部分である。

(*tvcn2dclosess.pro* **24 行目以降**)

```
; array subscript          ; 元々の配列の内、どの部分を切り抜くかを指定。
ixss=(ixe-ixs+1)/4         ; ここでは中心領域（全領域の 1/4 の面積）を指定。
ixes=(ixe-ixs+1)*3/4
```

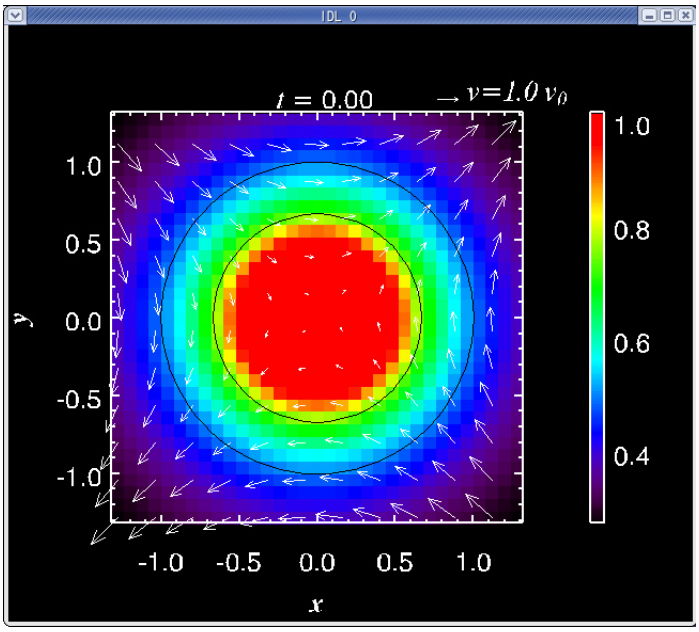

図 8.7: 図 8.6と同じであるが、中心部分だけ拡大した。右端のカラーバーの範囲が図 8.6よりも狭くなった理由は、中心領域だけ見た場合の rho の最小値が、全領域で見た場合の最小値と比べて大きいからである。今の場合、カラーバーの下限値は rho の下限値に合わせている。

```
iyss=(iye-iys+1)/4
iyes=(iye-iys+1)*3/4
;
; array definition
case precision of
'FLOAT ': begin
;print,'data type: ',precision
xs = FLTARR(ixes-ixss+1)                    ; 中心領域のデータを格納する
ys = FLTARR(iyes-iyss+1)                    ; 配列の定義（データの精度が
rhos = FLTARR(ixes-ixss+1, iyes-iyss+1)     ;float 型の場合）
vxs = FLTARR(ixes-ixss+1, iyes-iyss+1)
vys = FLTARR(ixes-ixss+1, iyes-iyss+1)
bxs = FLTARR(ixes-ixss+1, iyes-iyss+1)
bys = FLTARR(ixes-ixss+1, iyes-iyss+1)
;
end
else: begin
;print,'data type: ',precision
xs = DBLARR(ixes-ixss+1)                    ; 中心領域のデータを格納する
ys = DBLARR(iyes-iyss+1)                    ; 配列の定義（データの精度が
```

```
rhos = DBLARR(ixes-ixss+1, iyes-iyss+1)   ;float 型以外の場合)
vxs = DBLARR(ixes-ixss+1, iyes-iyss+1)
vys = DBLARR(ixes-ixss+1, iyes-iyss+1)
bxs = DBLARR(ixes-ixss+1, iyes-iyss+1)
bys = DBLARR(ixes-ixss+1, iyes-iyss+1)
end
endcase
;
;substitution                          ; データを新しい配列 (xs) に格納
for i=ixss,ixes do begin
xs(i-ixss)= x(i)
endfor
;
for j=iyss,iyes do begin
ys(j-iyss)= y(j)              ; データを新しい配列 (ys) に格納
endfor
;
for j=iyss,iyes do begin
   for i=ixss,ixes do begin
rhos(i-ixss,j-iyss)=rho(i,j); データを新しい配列 (rhos) に格納
vxs(i-ixss,j-iyss)=vx(i,j)  ; データを新しい配列 (vxs) に格納
vys(i-ixss,j-iyss)=vy(i,j)  ; データを新しい配列 (vys) に格納
bxs(i-ixss,j-iyss)=bx(i,j)  ; データを新しい配列 (bxs) に格納
bys(i-ixss,j-iyss)=by(i,j)  ; データを新しい配列 (bys) に格納
   endfor
endfor
```

上記プログラム以外の部分は *tvcn2dss.pro* と *tvcn2dclosess.pro* はほとんど変わらない (変数 x が xs に置き換わる程度)。

8.3.7 複数の図を表示する方法

前節までは一つの画面に一つのデータを表示させる方法を見てきた。実際には一つの画面に異なるデータを並べて表示させて、データを比較する場面もある。本節では複数のデータを読み込んで、一つの画面に表示させる方法について見ていく。

図は時刻 *tstep* の異なる 4 つのデータ (*sample0, sample1, sample2,* そして *sample3*) を一つの画面に並べて表示させたものである。

一つの画面に並べて表示させるためのメインプログラムは *tvcn2d4pss.pro* である。メインプログラム *tvcn2d4pss.pro* だけではデータファイルを読み込み、可視化の両方を行うので、 *read2dss.pro* を実行する必要はない。

(*tvcn2d4pss.pro* の実行)

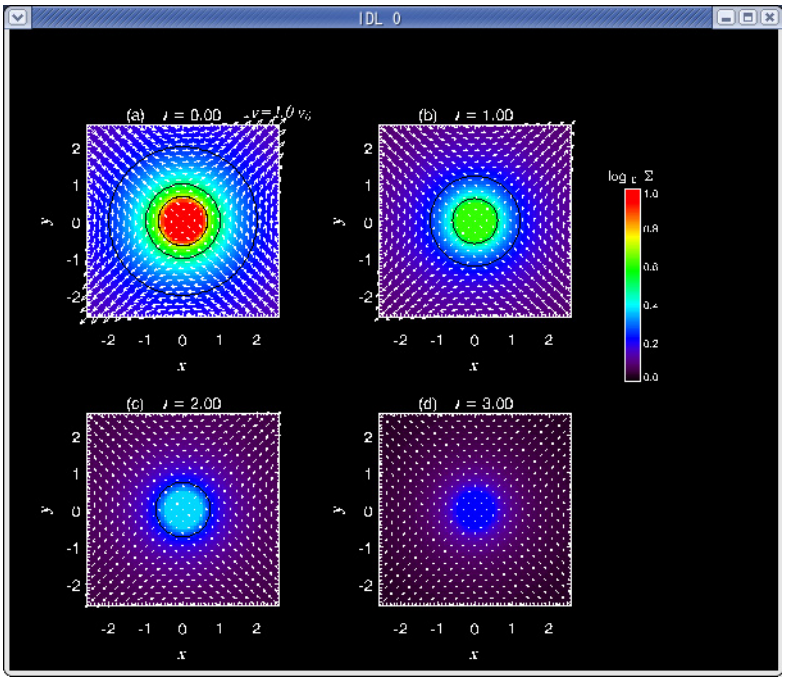

図 8.8: 異なる 4 つの時刻 (*tstep*) のデータを一つの画面に並べた。

```
IDL> .r tvcn2d4pss
% Compiled module: $MAIN$.
t=      0.00000
velocity scale=      56.2501
velocity scale=      56.2501
t=      1.00000
velocity scale=      56.2501
velocity scale=      56.2501
t=      2.00000
velocity scale=      56.2501
velocity scale=      56.2501
t=      3.00000
velocity scale=      56.2501
velocity scale=      56.2501
IDL>
```

データファイル名や画面上の位置、そして図を識別する記号は *tvcn2d4pss.pro* の中の 17 - 22 行目に指定している。下記の例ではデータファイルに (*sample0, sample1, sample2,* そして *sample3*) を指定している。

(*tvcn2d4pss.pro* **17 -22 行目; ファイル、位置、記号の指定**)

```
fn=['sample0','sample1','sample2','sample3']     ; ファイル名の指定
px1=[0.10,0.48,0.10,0.48]      ; 各図の左下の x 座標
py1=[0.55,0.55,0.10,0.10]      ; 各図の左下の y 座標
```

```
px2=[0.40,0.78,0.40,0.78]      ; 各図の右上の x 座標
py2=[0.85,0.85,0.40,0.40]      ; 各図の右上の y 座標
panel_n=['(a)','(b)','(c)','(d)']  ; 各図につける記号の指定
```

一つの画面に 6 つの図を並べて表示したい場合は、上記のやり方でファイル名、各図の座標や記号を 6 つ指定すればよい。ただし、本節では 4 つの図を並べて表示させる *tvcn2d4pss.pro* について説明する (6 つの図を表示させるメインプログラム *tvcn2d6pss.pro* についても基本的なやり方は同じである)。

表示させるカラースケール、コントア、そして矢印の指定は *tvcn2d4pss.pro* の中の 153 行目から 158 行目の間に直接書き込む。下の例ではカラースケールに rho, コントアに rho, 矢印に速度 v (sv1, sv2 はそれぞれ矢印の x, y 成分である) を選択している。

(**カラースケール、コントア、そして矢印の指定**)

```
; color & countour & vector ;;;;;;;;;;;;;;;;;;
;color scale
phys=rho
;contour
physc=rho
;vector
sv1=vx
sv2=vy
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

もし、カラースケール、コントアに rho の常用対数、矢印に磁場 b を指定する場合は以下のようにすればよい。

(**カラースケール、コントア、そして矢印の変更**)

```
; color & countour & vector ;;;;;;;;;;;;;;;;;;
;color scale
phys=alog10(rho)
;contour
physc=alog10(rho)
;vector
sv1=bx
sv2=by
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

また、矢印の間隔、長さ、カラースケールの最大値、最小値、そしてコントアの間隔を指定する場合は *tvcn2d4pss.pro* の 74 行目から 87 行目の部分を §8.3.2,§8.3.3 と同じやり方で編集すればよい。ここでの指定は 4 つの図に共通して適用される。

(*tvcn2d4pss.pro* **74 - 87 行目**)

```
; parameters for vector
i_skip=ixe/16
j_skip=iye/16
v_scale=long(max(sqrt(vx^2+vy^2)))/((x(1)-x(0))*i_skip)
if (v_scale le 0.) then $
  v_scale=max(sqrt(vx^2+vy^2))/((x(1)-x(0))*i_skip)
;
; parameters for tv
t_max=1.0
t_min=0.
; contour levels
;
c_top=!d.table_size
c_lev=findgen(4)/4
c2_lev=findgen(16)/4-4
```

カラーバーの位置と向きも自由に変えることができる。位置と向きは *tvcn2d4pss.pro* の 191 行目以降で指定している。下記は図 8.8を作成したときの設定である。

(*tvcn2d4pss.pro* **191 行目付近: カラーバーの位置と向き**)

```
; portrait
beginp,[0.80,0.45,0.82,0.75]
 c_scale,phys,top=c_top-3+1,max=t_max,min=t_min,bottom=1,$
/portrait,yaxis=1, title='log!D10!N !9 S !3'
; landscape
;beginp,[0.400,0.960,0.670,0.975]
;c_scale,phys,top=c_top-3+1,max=t_max,min=t_min,bottom=1,xaxis=0,$
;title='log!D10!N !9 S !3'
```

プロシージャ *beginp* は画面上で画を描く領域を指定するものであり、この宣言の後にカラーバーを表示したい座標を指定する。座標は 「始点の x 座標、始点の y 座標、終点の x 座標、終点の y 座標」の順番である。キーワード「title」はカラーバーのタイトルである。もし、カラーバーを横向きに表示させたい場合は、上記の 「; portrait」から「; landscape」の間をコメントアウトして、「; landscape」以下の部分をコメントインすればよい。

8.3.8 流線を表示させる方法

本節では流線を表示させる方法について見ていく。流線は、矢印だけでは分かりにくいガスの流れを理解するための有効な手段である。流線を表示させるメインプログラムは *tvcn2dSLss.pro* である。図 8.9は図 8.6の上に流線を重ねた結果である。作成方法は下記の通りである。

(*tvcn2dSLss.pro* **の実行**)

```
IDL> .r read2dss                  ; データの読み込み
% Compiled module: $MAIN$.
Input file name to read =
: tsample
t=      0.00000
IDL> .r tvcn2dSLss                ; tvcn2dSLss の実行
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho             ; カラースケールの選択
 for contour.:rho                 ; コントアの選択
Input "v" or "b"
 for vector.:v                    ; 矢印の選択
x-y slice
% Compiled module: TVCN2VE.
velocity scale=      56.2501
IDL>
```

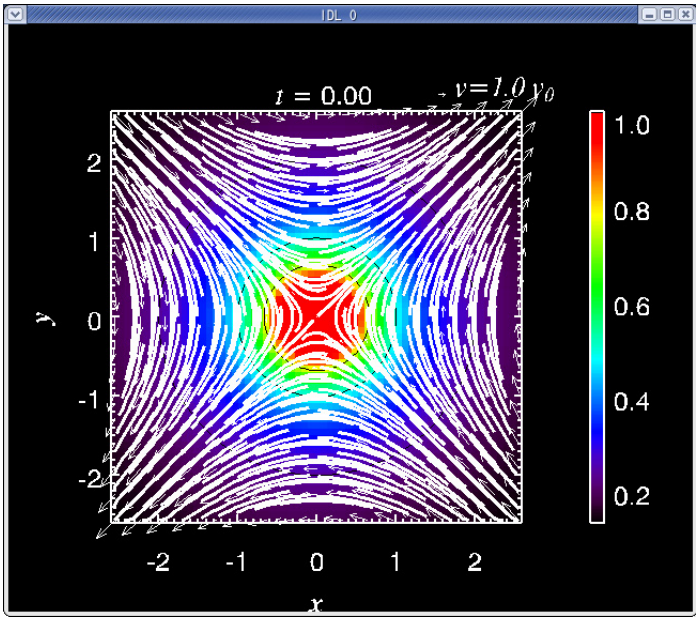

図 8.9: 図 8.6に流線を重ねた図

このメインプログラムの中での流線の求め方は、流線の式

$$\frac{dx}{ds} = U_x \tag{8.1}$$

$$\frac{dy}{ds} = U_y \tag{8.2}$$

をデータの中心から外側に向かって積分 (一次精度) することである。ここで ds は流線に沿った微小長さ、U_x, U_y はそれぞれ

$$U_x = v_x/\sqrt{v_x^2 + v_y^2 + \epsilon} \tag{8.3}$$

$$U_y = v_y/\sqrt{v_x^2 + v_y^2 + \epsilon} \tag{8.4}$$

で定義される (ϵ は微小量)。この積分は、データの全領域を幾つかの領域に分割して、それぞれの領域ごとに個別に行っている。分割した領域が大き過ぎると積分のとき誤差が大きくなり、正しい流線が得られない。逆に分割した領域が小さ過ぎると、隣り合う流線の間隔が狭くなり、見づらくなる。従って適切な大きさに分割しなければならない。分割の大きさを指定している変数は *iskip2*(x 方向) と *jskip2*(y 方向) である。

(*tvcn2dSLss.pro* **75 - 76 行目**)

```
iskip2=(nmax-1)/20
jskip2=(nmax-1)/20
```

上記の例では x, y 方向ともにデータの全領域を 20 個に分割している (*nmax* は x または y 方向のメッシュの数である)。

8.4 ファイルの出力方法

前節までは、作成した画像をディスプレイに表示させてきた。実際には、作成した画像をプレゼンテーションや LaTeX の文章に挿入したりするためにファイルの形で残したい場合がある。本節では作成した画像をファイルの形で出力する方法について見ていく。

8.4.1 PNGファイルの出力

まず最初に、PNG 形式の画像を作成するメインプログラム *tvcn2dss_png.pro* について見ていく。PNG 形式の画像はパワーポイントなどのプレゼンテーションツールに使用することができる。メインプログラム *tvcn2dss_png.pro* の使用方法は §8.3.1 から §8.3.4 で説明した *tvcn2dss.pro* の使用方法とほとんど同じである。

(*tvcn2dss_png.pro* **の使用方法**)

```
IDL> .r read2dss                ; データの読み込み
% Compiled module: $MAIN$.
Input file name to read =
: tsample
t=      0.00000
IDL> .r tvcn2dss_png            ; tvcn2dss_png の実行
% Compiled module: $MAIN$.
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho           ; カラースケールの選択
 for contour.:rho               ; コントアの選択
Input "v" or "b"
 for vector.:v                  ; 矢印の選択
x-y slice
velocity scale=      56.2501
IDL>
```

メインプログラム *tvcn2dss.pro* と異なり、ディスプレイに画像は表示されない。
図 8.10は *tvcn2dss_png.pro* を使って作成した PNG ファイル (LaTeX に挿入するために PNG 形式から EPS 形式に変換したが...) である。

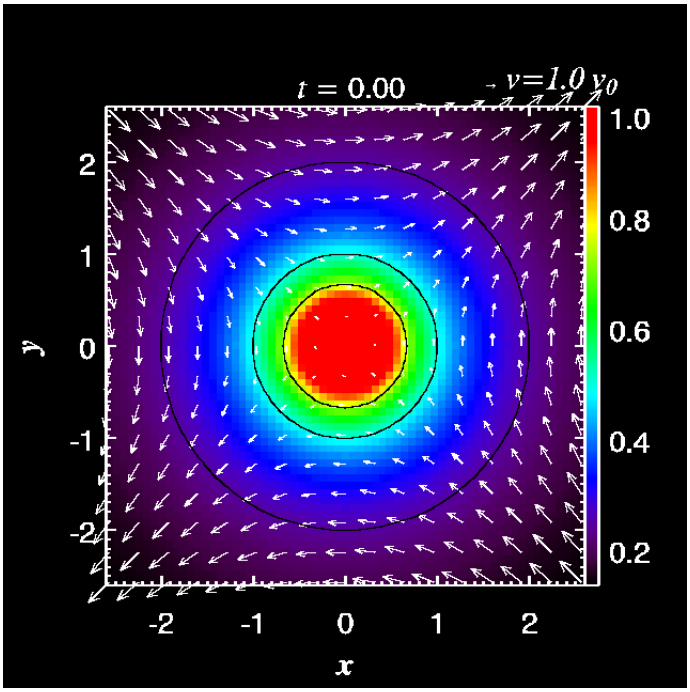

図 8.10: 図 8.6と同じであるが、PNG 形式のファイルに書き出した (LaTeX に挿入する都合上 PNG 形式から EPS 形式に変換した)。

出力されるファイル名は *tvcn2dss_png.pro* の 143 行目、プロシージャ *write_png* の行に指定されている。下記の例では *ochif10.png* である。

(**ファイル名の指定**; *tvcn2dss_png.pro* **143 行目**)

```
write_png,'ochif10.png', TVRD(),r,g,b
```

画像の大きさは *tvcn2dss_png.pro* の 18 行目 で指定する。下記の例では縦、横ともに 640 ピクセルである。

(**画像の大きさ指定**; *tvcn2dss_png.pro* **18 行目**)

```
xsize=640 & ysize=640
```

8.4.2 EPS ファイルの出力

次に EPS ファイルを出力する方法について見ていく。使用するメインプログラムは *tvcn2dss_eps.pro* であり、実行の仕方は *tvcn2dss.pro* とほぼ同じである。

(*tvcn2dss_eps.pro* **の使用方法**)

```
IDL> loadct,40            ; カラーテーブルのロード
% LOADCT: Loading table Rainbow + black
IDL> .r read2dss          ; データの読み込み
% Compiled module: $MAIN$.
Input file name to read =
: tsample
t=      0.00000
IDL> .r tvcn2dss_eps      ; tvcn2dss_eps の実行
% Compiled module: $MAIN$.
  xsize = 12.7cm,   ysize = 12.7cm
xoffset =  3.7cm, yoffset =  7.8cm
Input "rho,vx,vy,bx,by,lrho,vt,bt,em,va" or "return"
 for color scale,:rho     ; カラースケールの選択
 for contour.:rho         ; コントアの選択
Input "v" or "b"
 for vector.:v            ; 矢印の選択
x-y slice
velocity scale=      56.2501
Postscript device is closed
IDL>
```

メインプログラム *tvcn2dss_png.pro* を実行したときと同様、画像はディスプレイに表示されず、ファイルに出力される。図 8.11 はその出力された EPS ファイルである。

出力されるファイル名と大きさは *tvcn2dss_eps.pro* の 17 行目で指定している。下記の例では、ファイル名が *ochif11.eps*、縦、横の長さが 12.7 cm となる。

(**ファイル名と大きさの指定**; *tvcn2dss_eps.pro* **17 行目**)

```
ps1,/col,/in,psfile='ochif11.eps',xsize=12.7,ysize=12.7
```

上記プログラム最初の *ps1* は、device を “X”から “PS”に切替え、Postscript ファイルに画を出力するプロシージャ (*ps1.pro*) を呼び出す。キーワード */col* はカラーの作画を描く際にセットする。

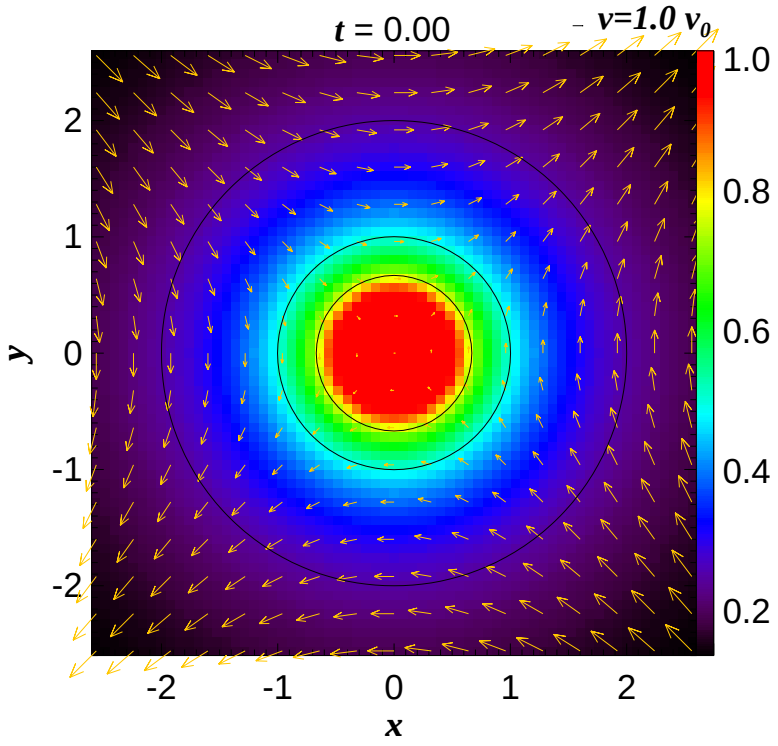

図 8.11: 図 8.6と同じであるが、 EPS 形式のファイルに書き出した。ただし、矢印は黄色にしている。矢印の色の指定については §8.3.3 を参照のこと。

キーワード */in* は “X”のカラーインデックスを “Postscript”のカラーインデックスに interpolation することを指示する。

8.4.3 連続スナップショットの作成方法

前節までは、基本的には一つのファイルを読み込ませて、一つの画像を出力するという方法について見てきた。しかし時間発展 (ex. 原始連星にガスが降着していく過程など) を追跡したい場合など、多数のバイナリデータを同じような形式で可視化する場面もある。その場合、一つ一つのデータを手動で「読み込み」、「画像出力」を行うのは非常に面倒である。そこで本節では自動でデータの「読み込み」、「画像出力」を行う方法について見ていく。

使用するメインプログラムは *write2dpng.pro* である。このメインプログラムはムービー (「パラパラ漫画」) を作成するためのプログラムであり、たくさんの指定したバイナリデータを自動で読み込み、PNG 形式で画像を出力する。ただしここで作成できるのは各時間ステップごとの静止画だけであり、動画にするためには動画作成ソフト (例えば Quick Time Pro) が必要である。なお、ここで作成した静止画をムービーにする方法については §8.4.4で述べる。図 8.12 は *write2dpng.pro* を実行して作成された図である。データは §8.3.7 で使用したものと同じである
(*sample0, sample1, sample2,* そして *sample3*)。

(*write2dpng.pro* **の実行**)

```
IDL> .r write2dss_png
% Compiled module: $MAIN$.
Input "v" or "b"
 for vector.:v                ; 矢印の選択。ここでは ‘‘v’’ を選択。
t=      0.00000
x-y slice
velocity scale=      56.0000
% Loaded DLM: PNG.
t=      1.00000
x-y slice
velocity scale=      56.0000
t=      2.00000
x-y slice
velocity scale=      56.0000
t=      3.00000
x-y slice
velocity scale=      56.0000
IDL>
```

カラースケールやコントアに用いる物理量の選択、カラースケールの最大値、最小値の指定、コントアや矢印の間隔の指定は *write2dpng.pro* の 91 - 108 行目で行う。指定の仕方は §8.3.7 で述べたやり方と同じである。

(*write2dpng.pro* **91 - 108 行目**; カラースケールやコントアなどの指定)

```
phys=rho                     ; カラースケールの選択
physc=rho                    ; コントアの選択
;
```

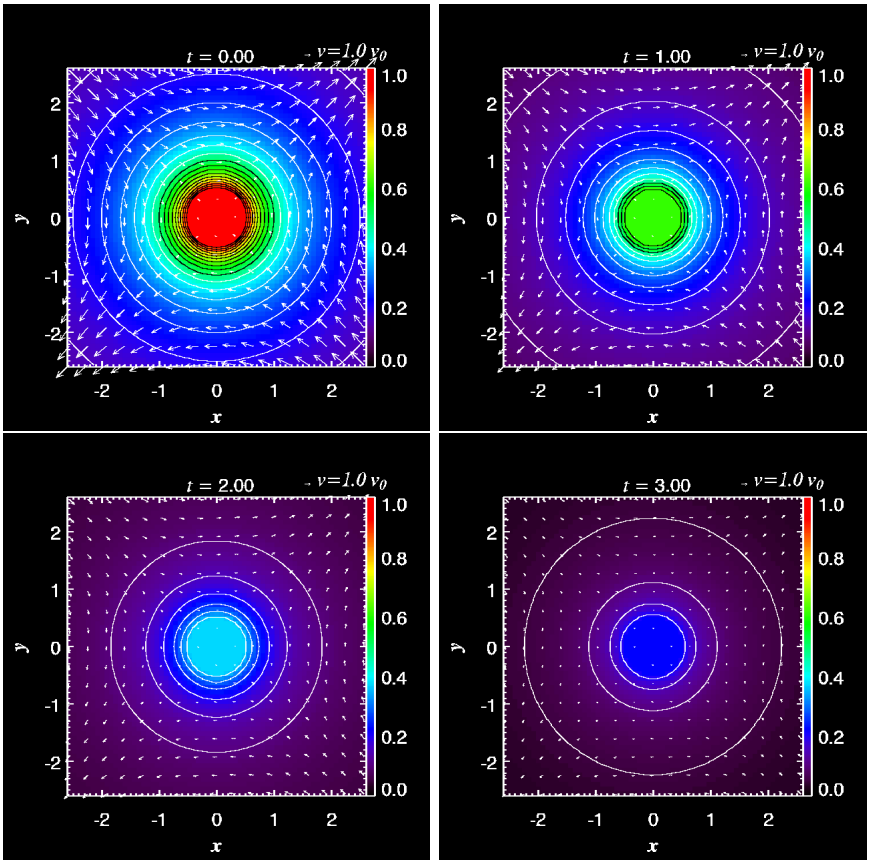

図 8.12: 異なる 4 つの時刻 (*tstep*) のデータ (*sample0, sample1, sample2,* そして *sample3*) を自動で読み込み、PNG 形式でファイルに出力させた (LaTeX に挿入する都合上、後から EPS 形式に変換している)。

```
; parameters for tv
t_max=1.0                ; カラースケールの最大値の指定
t_min=0.                 ; カラースケールの最小値の指定
;
;
;
; parameters for contour
. . .
c_lev=findgen(10)*0.05+0.5  ; コントア間隔の指定
c2_lev=findgen(10)*0.05     ; コントア間隔の指定
;
; parameters for vector
i_skip=ixe/16               ; 矢印間隔の指定
j_skip=iye/16               ; 矢印間隔の指定
```

読み込みむファイル名と出力ファイル名の指定

メインプログラム *write2dss_png.pro* では、下記のやり方で読み込むデータを指定し、PNG ファイルを出力している。

(*write2dss_png.pro* **35 - 41 行目**)

```
fn1=findfile('sample*.d')     ; 読み込むファイルを探す
fn=strmid(fn1,0,7)            ; 部分文字列の抽出
;
its=0                         ; 繰り返しの始値
ite=size(fn,/n_elements)-1    ; 繰り返しの終値
;
for l=its,ite do begin        ; 繰り返しの始まり
```

上記のメインプログラムの主な点について説明する。関数 findfile はカレントディレクトリにある *sample*.d* ($*$ はワイルドカード) という名前のファイルを探し、その名前を文字列型の配列 *fn1* に代入する。文字列の中に数字がある場合は、小さい数字を含む名前から先に格納される。関数 strmid は文字列から文字を抽出する。 上記の指定では文字列の左端 (0 番目) から数えて 7 個の文字を抽出する。 関数 size は上記の場合、文字列型配列 *fn* の要素数がいくつであるかを返し、それから 1 を引いて変数 *ite* に代入する。例えばカレントディレクトリに (*sample0.d, sample1.d, sample2.d,* そして *sample3.d*) というファイルがある場合、それぞれの関数が返す結果は以下のようになる。

(**各関数の返す値**)

```
IDL> $ls sample*.d
sample0.d  sample1.d  sample2.d  sample3.d
IDL> fn1=findfile('sample*.d')
IDL> print,fn1
sample0.d sample1.d sample2.d sample3.d
IDL> fn=strmid(fn1,0,7)
IDL> print,fn
sample0 sample1 sample2 sample3
IDL> ite=size(fn,/n_elements)-1
IDL> print,ite
           3
IDL>
```

(*write2dss_png.pro* **158 -160 行目**)

```
tvlct,r,g,b,/get      ; RGB color system から colar table を読み込む
write_png,'sample'+strmid(fn(l),6,1)+'.png', TVRD(),r,g,b ;PNG file へ出力
endfor                ; 繰り返しの終り
```

上記は PNG ファイルの画像を出力している部分である。PNG ファイル名は、*sample?.png* であり、? には読み込んだデータファイル名と同じ数字が入る (ex. 読み込んだデータファイル名が *sapmle2.d* であれば、出力される PNG ファイル名は *sample2.png* である)。プロシージャ tvlct は指定した変数からカラーテーブルを読み込む。ここではキーワード get により現在のカラーテーブルのベクトルを変数、r,g,b に取り込ませている。プロシージャ write_png は画像を PNG ファイルに書き込む。関数 TVRD は device の指定された部分のコンテンツ (各ピクセルの情報; 0 または 1 で表される) を返す。これらの関数やプロシージャの詳細については IDL Reference Guide A-M version 5.3, IDL Reference Guide N-Z version 5.3 (Research Systems, Inc. 1999) を参照のこと。

8.4.4 ムービーの作成

前節では自動的に静止画を作成する方法について見てきた。本節ではこれらの静止画を使ってムービー (「パラパラ漫画」) を作成する方法の一つを紹介する。ここでは Windows にインストールされた Quick Time Pro を使って作成する方法について説明する (ただし Quick Time Pro はシェアウェアである)。

まず、前節で作成した、時間ステップ毎にファイル名のついた静止画 (イメージシーケンス) を Windows 上のフォルダに保存する。次に Quick Time Pro を起動し、
「ファイル」 →「イメージシーケンスを開く (Q)」
をクリックし、イメージシーケンスのあるフォルダを開く。ここで注意すべき事は

1. イメージシーケンスの順番が正しいかどうか (時間ステップ通り並んでいるかどうか)
2. ムービーにしたいイメージファイル以外のファイルが混ざっていないかどうか (混ざっているとそれもムービーの 1 コマになってしまう可能性があるから)

である。そしてムービーにしたいイメージシーケンスの先頭のイメージを選択し、
「開く (O)」ボタンをクリックする。すると「イメージシーケンスの設定」というウィンドが現れるのでフレームレートを選択する。「OK」ボタンを押すとムービーが作成され、画面に表示されるの

で再生を行い、問題が無いければ「ファイル」 → 「書き出し」をクリックし、名前を付けて保存する。なお図 8.12の 4 枚の PNG ファイルを使って作成したムービー (*sample1fps.mov*; 1 フレーム / 秒) を付属 CD-ROM の *Chap8* ディレクトリの中の *data* ディレクトリの中に収録したのでムービーを作成する際の参考にして欲しい。

8.5 カラーの選択

前節まででデータを可視化するための基本的な手順について見てきた。本節では可視化する際に使用できるオリジナルのカラーテーブルの幾つかを紹介する。 IDL にはもともと 41 個のカラーテーブル (0 - 40 番) が添付されている。しかしこれらでは十分対応できない場合がある。その場合には、下記に紹介するオリジナルのカラーテーブルを使用すればよい。

8.5.1 白黒の EPS ファイルを作成する場合

(使用するプロシージャ) :: *bw.pro*

白黒のカラーテーブルは IDL に添付されている (“B-W LINEAR”)。しかし、§8.4 で説明したメインプログラム *tvcn2dss_eps.pro* を使用したとき、カラースケールに選択した物理量の値が小さければ小さい程、その場所は黒くなり見にくい。カラースケールに選択した物理量の値が小さければ小さい程、その場所が白くなる方が見やすい。そのようなカラーテーブルが *bw.pro* である。図 8.13は同じデータを、(a) IDL に添付されたカラーテーブル (“B-W LINEAR”), (b) *bw.pro* をロードして作成した画像である。

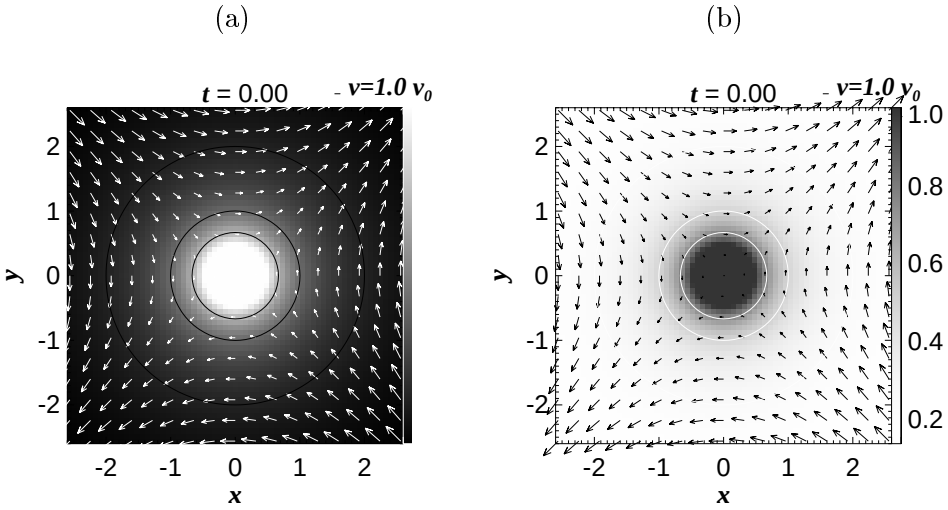

図 8.13: (a) IDL に添付されたカラーテーブル (“B-W LINEAR”) をロードし、メインプログラム *tvcn2dss_eps.pro* を使用して作成した画像。(b) カラーテーブル *bw.pro* をロードして同様に作成した画像。

カラーテーブル *bw.pro* のロードは下記の通りである。

(*bw.pro* のロード)

```
IDL> bw
% Compiled module: BW.
IDL>
```

8.5.2 カラーの EPS ファイルを作成する場合

(使用するプロシージャ) :: *rainbowf.pro*

IDL に添付されているカラーテーブルは §8.4 で説明した *tvcn2dss_eps.pro* を使用すると、カラースケールに選択した物理量の値が小さければ小さい程、その場所が濃い色になるものが多い。しかし、作成した EPS ファイルを LaTeX に挿入して白い紙に印刷すると、その濃い色の部分がかえって目立つこともある。 そこで §8.4.1 で述べたようにカラースケールに選択した物理量の値が小さければ小さい程、白色に近づく方が見やすい。そのようなカラーテーブルが *rainbowf.pro* である。図 8.14は同じデータを、(a) IDL に添付されたカラーテーブル (“Rainbow + black”) ,(b) *rainbowf.pro* をロードして作成した画像である。

(a) (b)

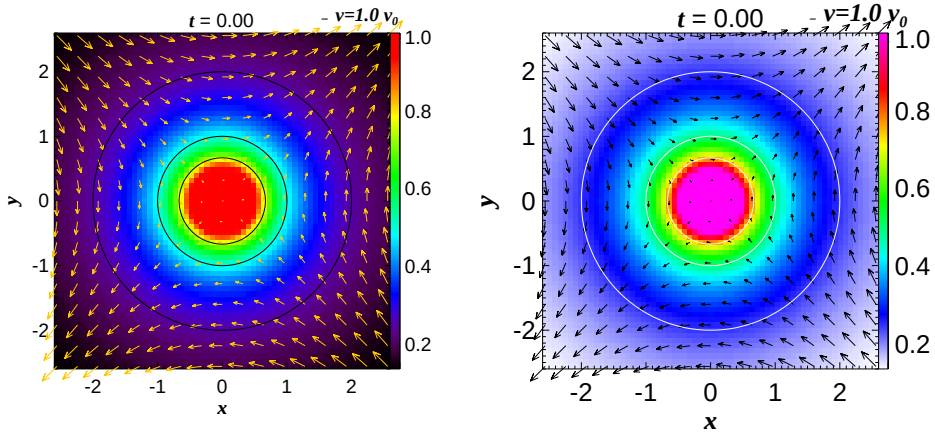

図 8.14: (a) IDL に添付されたカラーテーブル (“Rainbow + black”) をロードし、メインプログラム *tvcn2dss_eps.pro* を使用て作成した画像。(b) カラーテーブル *bw.pro* をロードして同様に作成した画像。

カラーテーブル *rainbowf.pro* のロードは下記の通りである。

(*bw.pro* のロード)

```
IDL> rainbowf
% Compiled module: RAINBOWF.
IDL>
```

8.6 トラブルシューティング

本節では §8.2、 8.3、そして 8.4 の操作を実行する際に発生するかも知れないトラブルとその対処方法について述べる。

8.6.1 “No such file or directory”

IDL のメインプログラム (ex. *read2dss.pro*) を実行した際に下記のようなエラーメッセージが表示されることがある。

(エラーメッセージ)

```
IDL> .r read2dss
% Error opening file. File: read2dss
  No such file or directory
IDL>
```

これはコマンドのパスが通っていない場合に表示されるので、§8.1.2で述べた方法にしたがってホームディレクトリ下の、*.cshrc* に下記のような *IDL_PATH* を設定する。

(IDL_PATH の設定)

```
r01{ochiys}1: cat ~/.cshrc
setenv IDL_PATH \+~/idl:\+$IDL_DIR/lib:\+$IDL_DIR/examples
```

これにより *~/idl*, *$IDL_DIR/lib*, *$IDL_DIR/example* の場所を検索する。これら以外の場所にコマンドがある場合にはその場所も指定しなければならない。追加した後も下記のように設定を適用しなければならないので注意。

(設定の適用)

```
r01{ochiys}2: source ~/.cshrc
```

8.6.2 “Corrupted f77 unformatted file detected.”

UNIX マシンで出力したバイナリデータを Linux マシンで読み込もうとすると、下記のようなエラーメッセージが表示されることがある。

(エラーメッセージ)

```
IDL> .r read2dss
% Compiled module: $MAIN$.
Input file name to read =
: tsample
% READU: Corrupted f77 unformatted file detected. Unit: 100, File: tsample.D
% Execution halted at: $MAIN$      45 /home/aplab/yasuhiro/idl/read2dss.pro
IDL>
```

この原因は UNIX と Linux ではデータの形式が異なるからである。解決策はメインプログラム *read2dss.pro* の 45 行目 に「/SWAP_IF_LITTLE_ENDIAN」を追加する。

(*read2dss.pro* **45 行目**)

```
OPENR, Unit2, fn+'.D', /F77_UNFORMATTED, /GET_LUN, /SWAP_IF_LITTLE_ENDIAN
```

8.7 ソースファイル

8.7.1 サンプルデータ用のソースファイル

本節では §8.2, 8.3, 8.5で使用したバイナリデータを作成したソースプログラムについて補足する。ソースプログラム (*sample2d.f*) は *rho, vx, vy, bx, by* という 2 次元配列を定義して、値を代入し、それをヘッダーファイル (*sapmle.d*; 配列の大きさなどの情報) とバイナリデータ (*sample.D*; 配列の値の情報) を出力するための簡単なプログラムである。

(*sample2d.f*)

```
      IMPLICIT REAL*8(A-H,O-Z)
      integer, parameter :: nmin=0, nmax=2**6
      real rho(nmin:nmax,nmin:nmax),
     $     vx(nmin:nmax,nmin:nmax),
     $     vy(nmin:nmax,nmin:nmax),
     $     bx(nmin:nmax,nmin:nmax),
     $     by(nmin:nmax,nmin:nmax),
      REAL x(nmin:nmax),y(nmin:nmax)
      real :: tstep
      integer i,j,ixs,ixe,iys,iye
      tstep=0.
      ixs=nmin
      ixe=nmax
      iys=nmin
      iye=nmax
C write header file "sample.d"
```

```
      OPEN(2,FILE="sample.d")
      WRITE(2,100) "sample data"
  100 FORMAT(A11)
      WRITE(2,101) ixs,ixe,iys,iye
  101 FORMAT(0P,4I5)
      WRITE(2,105)
  105 FORMAT('************************************************')
      WRITE(2,104)
  104 FORMAT(2X,'ixs',2X,'ixe',2X,'iys',2X,'iye')
      WRITE(2,105)
      WRITE(2,102) "sample.D"
      WRITE(2,106) TSTEP
      WRITE(2,107)
 102  FORMAT('FILENAME=',A10)
 106  FORMAT('TIME=',F15.7)
 107  FORMAT('FLOAT ')
      CLOSE(UNIT=2)
C initial condition
      do i=ixs,ixe
         x(i)=0.08*(i-32)
      end do
      do j=iys,iye
         y(j)=0.08*(j-32)
      end do
      DO j=iys,iye
         DO i=ixs,ixe
            RI=MAX(sqrt(x(i)**2+y(j)**2),1.D-8)
            rho(i,j)=0.5/RI
            vx(i,j)=5.*y(j)
            vy(i,j)=5*x(i)
            bx(i,j)=x(i)/sqrt(x(i)**2+y(j)**2)
            by(i,j)=y(j)/sqrt(x(i)**2+y(j)**2)
         enddo
      enddo
C  write binary "sample.D" in single precision
      OPEN(12,FILE="sample.D",FORM='UNFORMATTED')
      write(12) x
      write(12) y
      write(12) rho
      write(12) vx
      write(12) vy
```

```
      write(12) bx
      write(12) by
      CLOSE(UNIT=12)
      END
```

この *sample2d.f* では、あらかじめ配列の大きさ (e.g. ixs, ixe, iys, iye) や物理量 *rho, vx, vy, bx, by* を与えているが、実際には利用者が使用する配列の大きさや、シミュレーションを行って得られた物理量のデータを使用すればよい。

上記のプログラム *sample2d.f* をコンパイル、実行するとヘッダファイル (ファイル名 *sample.d*) とバイナリデータ (ファイル名 *sample.D*) が生成される。

(コンパイル、実行)

```
k15{ochiys}1: frt sample2d.f
k15{ochiys}2: ./a.out
```

ファイル *sample.d* は配列の大きさやバイナリデータが書き込まれたファイルの名前、時間ステップ、データ型などの情報を持ったヘッダーファイル (テキスト形式) である。 *sample.D* は 配列の値の情報を持つバイナリデータである。 *sample.d* の中は、下記のようになっている。

(*sample.d*)

```
k15{ochiys}3: cat sample.d
sample data
    0   64    0   64
************************************************
  ixs  ixe  iys  iye
************************************************
FILENAME=  sample.D
TIME=      0.0000000
FLOAT
k15{ochiys}4:
```

データの書き出し方について、上記プログラム *sample2d.f* のようにヘッダファイル (*sample.d*) とバイナリデータ (*sample.D*) の両方を書き出しておくと、§8.2 で扱った *read2dss.pro* をそのまま使って IDL にデータを読み込ませることができる。また *read2dss.pro* に配列の大きさやバイナリデータが書き込まれたファイルの名前、時間ステップ、データ型などをあらかじめ書き込んでおけば、ヘッダーファイルを作成する必要はなくなり、バイナリデータだけを読み込めばよい。どの方法

を用いるかは利用者の目的に応じて選んでもらいたい。

参考文献

[1] Research Systems 1999, IDL Reference Guide A-M (version 5.3: Research Systems, Inc.)

[2] Research Systems 1999, IDL Reference Guide N-Z (version 5.3: Research Systems, Inc.)

[3] Research Systems 1999, IDL Reference Guide (Objects & Appendixes) (version 5.3: Research Systems, Inc.)

[4] Research Systems 2003, IDL Online Guide (version 6.0: Research Systems, Inc.)

[5] アダムネット株式会社 1998, Introduction to IDL (version 5.1: アダムネット株式会社)