IDL 講習会 (初級編)

2021年9月**29日(水)**, 30日(木), 10月6日(水), 7日(木)
Zoom オンライン開催
主催 天文データセンター
講師 巻内 慎一郎 (国立天文台 天文データセンター)

一日目

- IDL について
  - IDL とは？
  - 言語の特徴
  - IDL の配列について
  - 開発元の歴史
  - IDL の入手
- 使用準備
  - 天文台データ解析システムにおけるIDL環境
  - IDL の利用とライセンス
  - 操作方法とエディタ
  - 環境設定(1) パス(IDL_PATH)
    - 環境変数 IDL_PATH の設定方法
    - 環境変数 IDL_PATH の確認方法
  - 環境設定(2) Startup file (起動ファイル)
  - 環境設定の注意点
- 起動と終了
  - コマンドライン環境の起動と終了
  - IDL ワークベンチ(IDLDE)の起動と終了
  - IDL Workbench - IDLDE -
  - IDLDE を使った環境設定
  - 作業ディレクトリ(カレントディレクトリ)
  - 実行中のプログラムの中断方法
  - IDL の使い方と実行モード
- グラフィックス ～ データプロット(plot) etc.
  - Direct Graphics vs. Object Graphics
  - ウィンドウの操作
    - ウィンドウを開く
    - 開いたウィンドウを操作する
  - データを表示する
  - PLOT (procedure)
    - 基本的な使い方
    - よく使うオプション
    - 外見を整えるオプション
  - 複数データを重ねてプロット(OPLOT)
  - 点や線のオーバーレイ(PLOTS)
  - 文字列のオーバーレイ(XYOUTS)
  - IDLグラフィックスの座標
  - 軸の作成(AXIS)
  - 画面を分割する(!p.multi)
  - Position キーワード
  - Postscript ファイルに出力する
  - エラーバーのプロット
  - CONTOUR (procedure)
  - SURFACE (procedure)
    - SHADE_SURF
    - XSURFACE
  - CGPLOT
  - Coyote Graphics によるファイル出力
  - ヒストグラム作成
    - CGHISTOPLOT
  - 関数型のグラフィックスルーチン

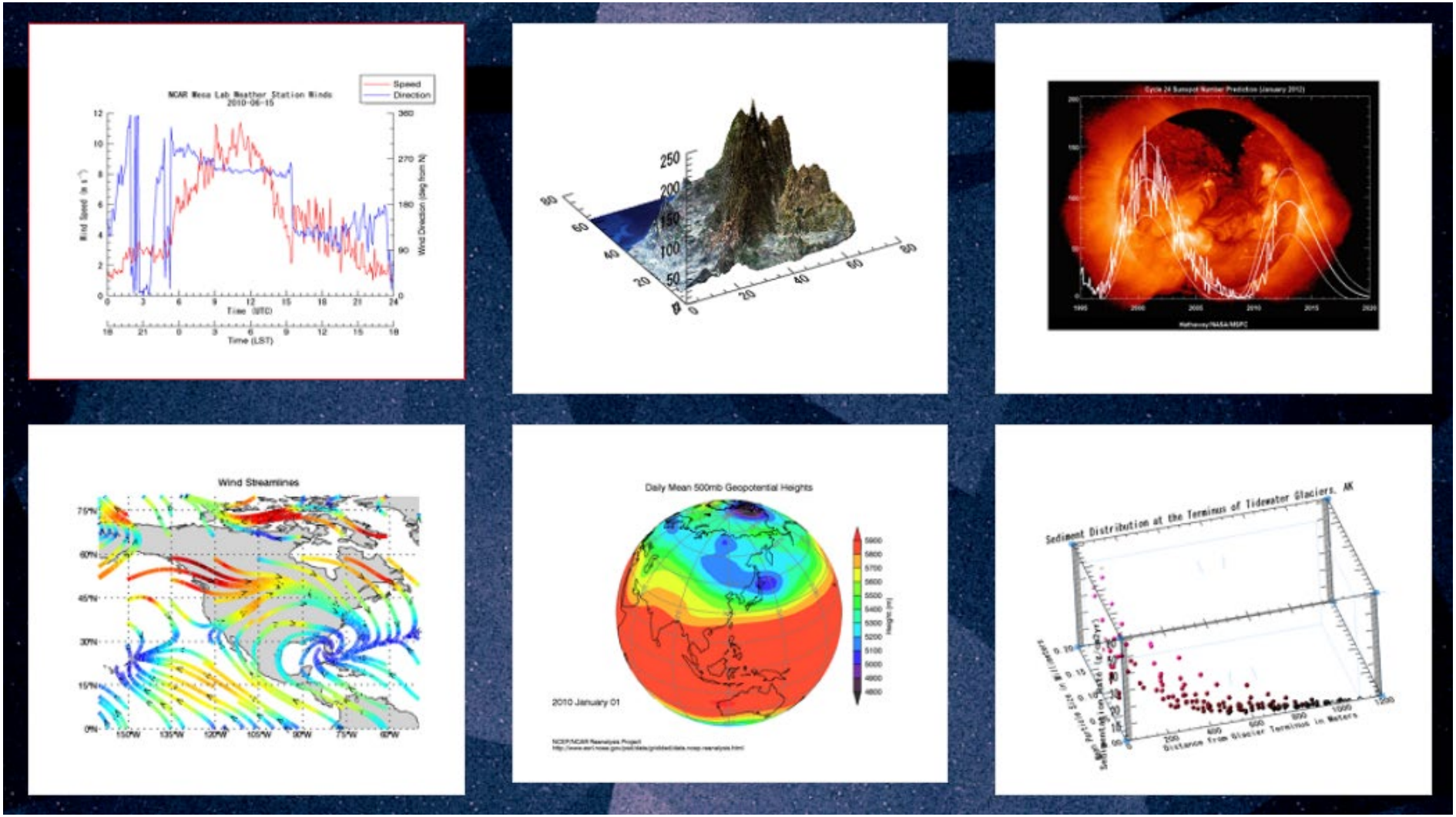
IDL について

IDL とは？

- Interactive Data Language
- データの解析と可視化に優れた機能を持つ配列指向型のプログラミング言語
- データの処理、その科学的解析から視覚化まで IDL だけで行うことが可能
- 対話的に使用する事が出来て、手軽に扱える。すぐに結果が見られる

etc.

- 豊富なグラフィックスルーチンが用意されている、画像処理に長けたプログラミングソフトウェア
- ポストスクリプト(PS)出力の他、JPEG や PNG などの形式も出力可能
- ライブラリルーチンを使用することで FITS ファイルの読み書きも出来る
- 天文・宇宙の分野ではスタンダードに使用されている
  - ほかにも、高層大気や気象、海洋物理、医療画像処理などの分野で広く使用されている
- 画像処理用のほか、数学的、統計的な処理機能などデータ解析ルーチンも豊富
- むしろどんなルーチンがすでに準備されているのか把握するのが難しいほど



IDL ベンダーサイト(https://www.harrisgeospatial.co.jp/)より: データ可視化機能の例

言語の特徴

- 配列計算が得意。ベクトル化された処理により、大量の要素、次元を持った配列も、見通しよく扱うことができる
- 直感的に書けて、理解がしやすい。
  比較的少ない行数で読みやすいコードになる
- 変数の事前宣言が不要。いつでも新しい変数を作成できる。変数のデータ型は(基本的には)IDL が自動的に判断して動的に決定できる

→ 習得が容易

- 構文は歴史的には FORTRAN 風の傾向が強かったが、最近は C 言語的な部分も多い。
  オブジェクト指向の機能等も追加されている
- IDL 自体は C 言語で書かれている
- コンパイラ型とインタプリタ型の中間的存在。
  通常、自動コンパイルで使用される。インタープリタ的な使い勝手だが、コンパイラ型に見劣りしない速度も出る
- Linux のほか、Windows や Mac OS でも動作する
  (クロスプラットフォーム)
  - どの OS で開発した IDL プログラムでも、同じように使える
- ただし、ライセンスは高価

IDL の配列について

- 最大の特徴である IDL の配列操作(ベクトル化された処理)は C 言語などで書かれた処理と比べても十分に(同等レベルに)高速
- しかし、配列の個々の要素に対してループ処理を行うなどすると、途端に遅くなる
- 配列操作は IDL 的に取り扱うべし
  - 配列同士の演算に不用意なループは用いない
  - IDL に用意されているプログラムも多くは配列をそのまま入力できる
- ただし、ループ処理は IDL でも多用される
  (配列処理に不必要に使わないことが重要)

開発元の歴史

- 大本は大学の研究所(コロラド大学ボルダー校の大気宇宙物理学研究所 LASP)による開発。
- その後、LASP から独立して設立された会社、Research Systems Inc. (RSI) が長年の開発元に。
- 2004年に ITT の子会社となり、2006年に RSI は ITT Visual Information Solutions に改称。
- 2011年には、分社化して Exelis VIS (Visual Information Solutions) となった。
- 2015年2月、米国 Harris 社が米国 Exelis 社を買収。
- 2018年1月より社名変更「Exelis VIS」から「Harris Geospatial」に
- (余談)インストールディレクトリ名にデフォルトでは社名が入る(rsi, itt, exelis, harris)ため、インストールパスが頻繁に変わり、(システムに複数バージョンをインストールしたい場合などに)一時混乱の元だった(今も?)。

IDL の入手

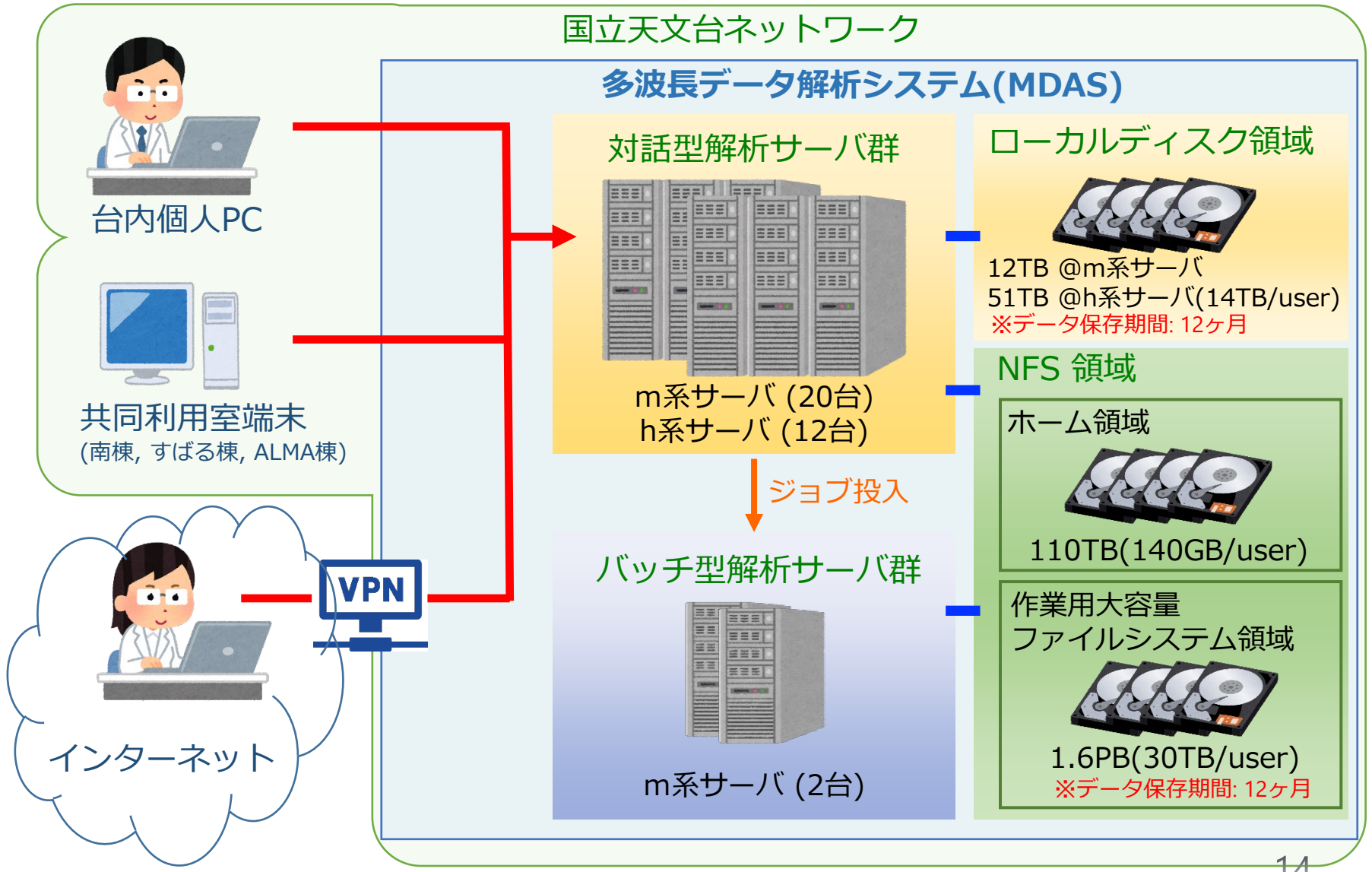
- IDL のインストールファイルはメーカーサイトでユーザ登録して申請することでダウンロード可能
- VM (Virtual Machine)が含まれる
- VM は無料で使用できる
  - IDL プログラムを VM アプリケーションとして配布することが可能。(ただし、動作上のいくつかの制限もある)
- ヘルプファイルも含まれる
- ライセンスされていない IDL 本体はデモモードで起動する(試用可能)

使用準備

天文台データ解析システムにおける IDL 環境

- 国立天文台 天文データセンターが運用している多波長データ解析システム(MDAS)には、様々な天文データ解析に使用されるソフトウェア類がインストールされており、**IDL** も(ほぼ)最新版が使用できる
- 多波長データ解析システムを使用できる人
  - 国立天文台の職員および国内外の天文学研究者(大学院生を含む)
    - 学部生は事情に応じて個別の許可が必要
- 使用できる計算機
  - 解析サーバの OS は Linux (Red Hat Enterprise Linux)
  - 解析サーバには天文台内の自分のマシンからリモートログインするか、共同利用室に設置された端末からログインする
  - 天文台外からの使用には VPN を通した接続が必要
  - グラフィックス表示には X Window か VNC を使用する

多波長データ解析システムの構成と接続



IDL の利用とライセンス

- IDL の使用にはライセンス認証が必要
  - ライセンス認証がされない場合は7分間のデモモードで起動する
- 多波長データ解析システム全体では、通常時最大80人が同時使用できるライセンスを備えている
- 同じホストにログインして IDL を複数起動しても使用されるライセンスはひとつ分
  - 別ホストで使用すると、ライセンスを余計に消費する
- idlstat コマンドで現在のライセンス使用状況を確認することが出来る
  - idlstat は多波長解析システムで用意された独自ツール (IDL のベンダー提供コマンド の wrapper プログラム)

(参考情報) ライセンス管理の仕組みが ver. 8.6 以降で変わった

操作方法とエディタ

- IDL を対話型で使用する際は、シェルターミナルから起動してそのままコマンドを入力していく
- IDL のプログラムを作成する場合は、任意のテキストエディタを起動してソースコードを記述する
- テキストエディタは、IDL 用に用意された統合開発環境 IDLDE (後述)を用いても良いし、その他のエディタを使用しても構わない
- 多波長解析システムでは Emacs, Vim, gedit (GNOMEの標準テキストエディタ) などが使用できる。使い慣れたもの、使いやすいものを使えば良い

多波長データ解析システムで利用可能なソフトウェア

GNOME 端末

gnome-terminal

テキストエディタ

gedit, emacs, vim

FITS ビューア

fv, ds9

画像ビューア

eog, display (ImageMagick)

PostScript ビューア

gv (Ghostview)

PDF ビューア

evince

GNOME エディタ(gedit) の IDL 用シンタックスハイライトの設定方法
右上のハンバーガーメニュー(三)から「表示」→「ハイライトモード」で "IDL-Exelis" を選択

環境設定(1) パス(IDL_PATH)

- IDL パス (IDL_PATH) の役割
  - IDL のプログラムは、手動コンパイルして実行することももちろん可能だが、自動コンパイルして実行する使い方が普通
  - IDL はプログラム(ファイル名が “*.pro” のプロシージャや関数)を IDL パスの中から自動的に検索して、コンパイル、実行できる
  - IDL のパスは通常、環境変数 IDL_PATH に設定する
  - ベンダーが提供するプログラム(デフォルトライブラリ)のほか、ユーザが追加したライブラリルーチンや自作プログラムが置かれたディレクトリのパスをあらかじめ設定しておけば、毎回同じ環境で使用できる

- プログラムは、パスに書かれたディレクトリから指定した順番で検索されて、最初に見つかったプログラムがコンパイルされる
- このため、同じ名前のプログラムが異なるディレクトリに存在すると、混乱の元になるので注意。
  たとえば、
  - 意識せずに同じ名前で作成されたプログラム
  - 同じプログラムだがバージョンが異なる

  の存在などがトラブルの元になりやすい。

自作プログラムの名前は、既存プログラムと重複しないように意識することが必要

環境変数 IDL_PATH の設定方法

bash の場合の例

```
$ IDL_PATH=~/work:+~/idl/mylib:"<IDL_DEFAULT>":${IDL_PATH}
$ export IDL_PATH
```

(先頭の $ はプロンプトです)

- ✓ : (コロン)でディレクトリのパスをつなぐ
- ✓ 先頭に + を付けると、そのサブディレクトリもすべて追加される
- ✓ "<IDL_DEFAULT>" は開発元が用意したライブラリの全パスを意味する
- ✓ この設定前にすでに IDL_PATH 環境変数が存在しており、それを残したまま追加する場合は、${IDL_PATH} を含める

csh/tcsh の場合の例

```
$ setenv IDL_PATH ~/work:+~/idl/mylib:"<IDL_DEFAULT>":${IDL_PATH}
```

- ◘ 毎回使う設定は、シェルの設定ファイル .bashrc や .cshrc に書いておく
- ◘ 多波長解析システムではデフォルトで標準的な IDL_PATH が共通設定されている

環境変数 IDL_PATH の確認方法

- 設定した(設定されている) IDL_PATH 環境変数を確認するには、

```
$ printenv IDL_PATH
```

✓ printenv は環境変数を表示する Linux のコマンド

あるいは、

```
$ echo $IDL_PATH
```

環境設定(2) startup file (起動ファイル)

- startup ファイルは IDL の起動時に自動実行される IDL で書かれたバッチ(スクリプト)ファイル
- IDL コマンドを使った環境設定や、作業の事前操作など、毎回決まった手順を実行したい場合に、設定しておく
  - たとえば、作業やプロジェクトごとに独自に必要なパスを設定する、コンパイル時やエラー発生時の挙動を設定する、保存してある毎回使う変数データを展開する、など
- ファイル名は何でも構わないが、*.pro としておくのが無難。(IDL のファイルとして識別するため。ただし、中身はプロシージャではなくスクリプトである)
- 環境変数 IDL_STARTUP にファイル名を指定する

環境設定の注意点

- IDL_PATH や startup file の設定や確認にはワークベンチ (IDLDE) の設定機能を使用すると分かりやすい
  (詳細は次章)
  - 「ウィンドウ」->「設定」-> 「IDL」-> ....

ただし、注意が必要

- 環境変数による設定とワークベンチによる設定の両方が存在する場合は、環境変数が優先される
- 多波長データ解析システムでは、デフォルトのユーザ環境として環境変数 IDL_PATH が設定済み(この状態でワークベンチ側の設定を行っても設定は保持されない)

起動と終了

コマンドライン環境の起動と終了

■ 起動

- Linux では端末(ターミナル)でコマンド名 "idl" を入力
  - (参考) Windows の場合は、スタートメニューなどから「IDL 8.x Command Line」を実行。(パスが通っていればコマンドプロンプトから "idl" でも可。)

```
$ idl
```

■ 終了

- IDL のプロンプトに対して "exit" を入力

```
IDL> exit
```

❗ 起動時にライセンス認証が実行される。ライセンスが無い(あるいは認証が出来ない)場合は、7分間だけ使用できるデモモードになる。(7分が経過すると自動終了する)

起動時に表示されるメッセージ (ver.8.6.1の例)

```
$ idl
IDL 8.6.1 (linux x86_64 m64).
(c) 2017, Exelis Visual Information Solutions, Inc., a subsidiary of Harris Corporation.

Licensed for use by: Fujitsu Limited - Aoyama via 133.40.130.160:7070
License: MNT-5509912:****-****-****-E890
A new version is available: IDL 8.7
https://harrisgeospatial.flexnetoperations.com

IDL>
```

← 新バージョンがある場合のお知らせ表示

ライセンスエラーの場合に表示されるメッセージ (ver.8.5.1の例)
7分間だけ使用可能なデモモードで起動する

```
$ idl
IDL Version 8.5.1 (linux x86_64 m64). (c) 2015, Exelis Visual Information Solutions, Inc., a subsidiary of Harris
Corporation.
LICENSE MANAGER: Cannot find license file.
 The license files (or license server system network addresses) attempted are
listed below.  Use LM_LICENSE_FILE to use a different license file,
 or contact your software provider for a license file.
Feature:       idl
(略)
Entering timed demo mode. Each session is limited to 7 minutes of operation. Printing and file saving are disabled.

To learn more about our license options for this product, please contact your account manager or Exelis Visual
Information Solutions, Inc., a subsidiary of Harris Corporation. at info@exelisvis.com.

IDL>
```

IDL ワークベンチ(IDLDE)の起動と終了

■ 起動

- Linux では端末(ターミナル)でコマンド名 "idlde" を入力

```
$ idlde &
```

  - ✓ コマンドの後ろに & (バックグラウンドで実行する)を付けないと、起動したターミナルを占拠してしまうので注意
  - ✓ (参考) Windows の場合は、スタートメニューなどから「IDL 8.x」を実行

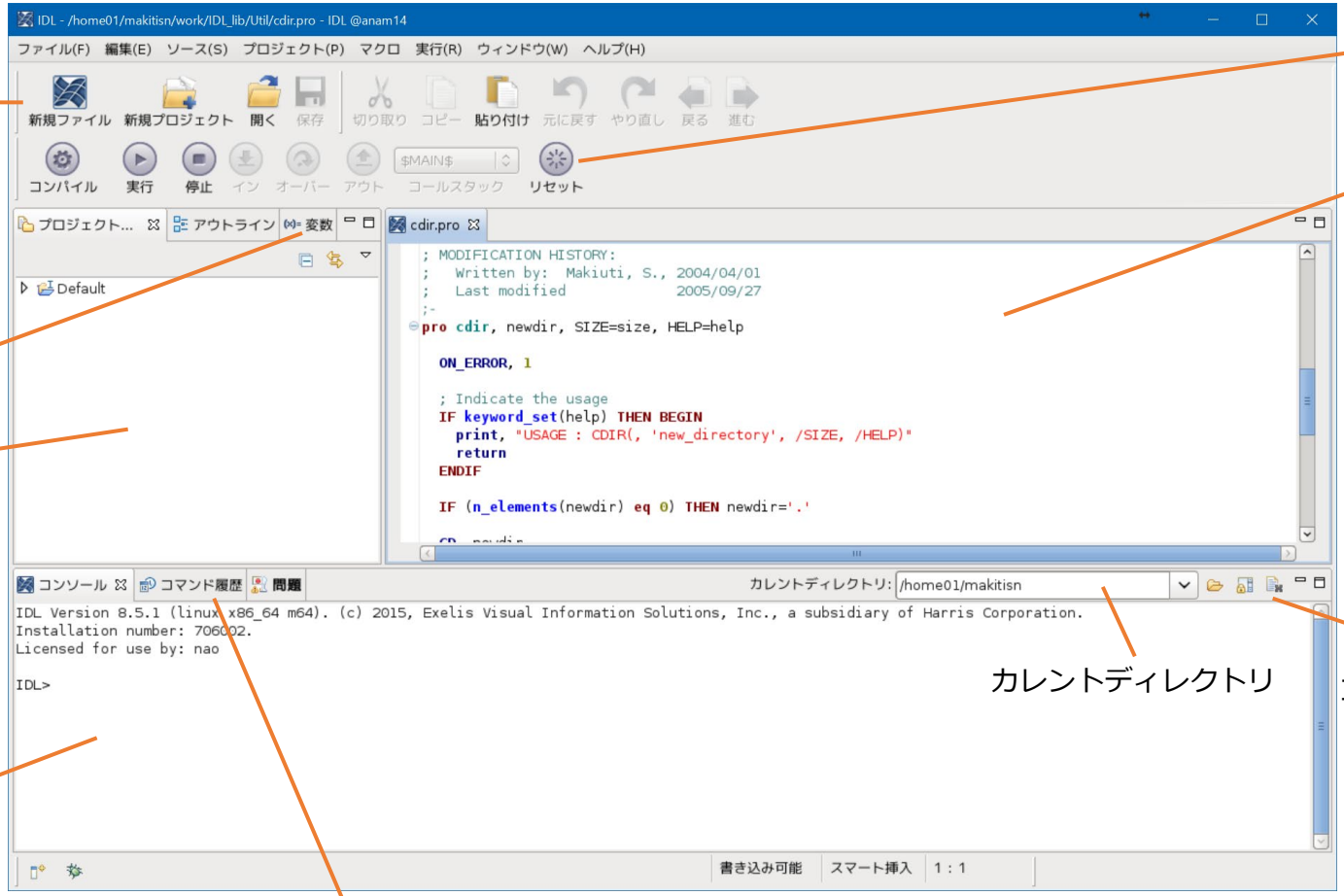
■ 終了 (次のいずれかの方法)

- メニューバーから「ファイル」→「終了」を選択
- 「コンソール」領域の IDL プロンプトに対して "exit" を入力
- IDLDE ウィンドウのクローズボタンで閉じる (この場合は通常、確認ダイアログが表示される)

IDL Workbench (プログラム統合開発環境) - IDLDE -

- オープンソースの Eclipse フレームワークをベースにして用意された、IDL 用のグラフィカルな開発環境
- コマンドラインの操作も可能。(コンソール領域がある)
  - ただし、純粋なコマンドライン環境の方が「より軽い」「コンソールの表示領域が大きく取れる」などの利点がある
- 複数のソースを連携したプロジェクトの管理や、デバッグツールなどを使いこなせれば、多機能な IDLDE は便利に使える
- IDLDE のレイアウトや各項目の表示・非表示などは、かなり自由にカスタマイズが可能
  - 「ウィンドウ」→「ビューのリセット」でデフォルトに戻せる
- IDL_PATH など各種設定を GUI で確認、操作できる
- エディタ画面では IDL の各構文が分かりやすく色分けして表示される

IDLDE (IDL Development Envionment)



IDLDE を使った環境設定

■ よく使う項目

「ウィンドウ」→「設定」

→「IDL」"起動ファイル", "作業ディレクトリ"

→「パス」(IDLパス)

- IDL パスは、環境変数 IDL_PATH が別に設定されている場合、そちらが優先される。(IDLDE で行ったパス設定は、起動時に IDL_PATH に置き換えられる)
  IDL_PATH が設定されていない場合は、IDLDE の環境設定で設定した IDL パスはコマンドライン環境にも反映される
- IDL パス以外の設定項目も基本的に同様。IDL をコマンドライン環境で使用する場合も、設定やその確認は IDLDE から行うと GUI で分かりやすい

作業ディレクトリ(カレントディレクトリ)

- IDL のプロセスは「作業ディレクトリ」"current working directory" 内で実行される
- コマンドラインモードでは IDL を起動したディレクトリ
- ワークベンチ(IDLDE)には「デフォルトの作業ディレクトリ」の設定がある。(環境変数 IDL_START_DIR でも設定可能)
- プログラム(*.pro) の最初の検索パスやファイル入出力の対象ディレクトリは、作業ディレクトリ
  - もちろん、ファイルを扱う際に、ファイル名に上位パスを付けたフルパスで指定することは可能
- この認識が曖昧だと、書き出したファイルがどこに保存されたのかなど、分からなくなることもあるので注意
- IDL 内部で作業ディレクトリを変更するには、下記の(Linux コマンドに準じた) IDL コマンド群が使える
  - CD, PUSHD, POPD

実行中のプログラムの中断方法

- 実行中のプログラムを手動で止めたい場合、キーボードから Ctrl+c (Cntl キーを押しながら c キー) をタイプすることで可能
  - ワークベンチ使用時は、Ctrl+c でプログラムを止めるとき、コマンドラインエリアにフォーカスしておく。エディタにフォーカスされている場合は Ctrl+c はコピー操作になる
- 間違って無限ループを作ってしまったときや、長時間かかるプログラムを途中で止めたくなったときなどに使用する
- デバッグなどのために、プログラムの中にあらかじめ中断を仕込む用途には STOP プロシージャを使う。中断されたプログラムは .CONTINUE コマンドで再開できる

IDL の使い方と実行モード

- IDL の使い方には、対話式に一行ずつコマンドを実行していくインタラクティブモードと、プログラムを書いて実行するプログラムモードがある
- 大量のデータを処理する場合や、同じ処理を繰り返す必要がある場合にはプログラムを書く必要があるが、基本的なデータ処理と可視化などのデータ解析の場面など多くの場合はインタラクティブモードの使用で事足りる
- インタラクティブモードでの処理は、一連のコマンド群をファイルに保存しておいて、バッチ処理(スクリプトの実行)という形で実行することもできる
- また、手順を記録(メモ)したテキストファイルから、まとめてコピー＆ペーストでコマンドラインに流し込むような使い方も可能で、便利で効率的に使える

グラフィックス

データプロット(PLOT) etc.

Direct Graphics vs. Object Graphics

- Object Graphics はオブジェクト指向プログラミングによるグラフィックシステムとして IDL version 5.0 から実装された
- IDL 8.0 以降、従来の Direct Graphics を使ったプロシージャ型のグラフィックスルーチンの他に、 Object Graphics 機能を使った関数型のルーチンが加わった
- Object Graphics では、オブジェクトの生成や初期化等、描画するまでにいくつかの手順が必要なため、手軽さ・軽快さには劣る
- Direct Graphics の方が動作は軽い → 一般的なインタラクティブな使用には Direct Graphics の方が適している
- Object Graphics では、描画情報を保持して後から変更することが可能

※ 本講習会では Direct Graphics を中心に解説する。

ウィンドウの操作

- 通常は必要なグラフィックスウィンドウは自動で開かれる
  たとえば、plot コマンドを使うと、自動的に開いたウィンドウにプロットされる
- ウィンドウの大きさや(初期の)位置を指定したり、複数のウィンドウを開いて使い分けるなどするために、ウィンドウ操作コマンドを使って自分で操作することも出来る

ウィンドウ操作コマンド

WINDOW	WDELETE	WSET	ERASE
開く	閉じる	出力先 (current window) を切り替える	中身を消す

ウィンドウを開く

- [使い方]

```
WINDOW[, window_index, /free] [, {x|y}size=value, {x|y}pos=value, title=string]
```

  - ✓ window_index : ウィンドウ番号(0-31, 指定が無ければ0から)
  - ✓ /free : 32番以上の空いている番号を使う
  - ✓ {x|y}size : ウィンドウの縦/横幅をピクセル単位で指定する
  - ✓ {x|y}pos : ウィンドウを開く位置を指定する
  - ✓ title : デフォルトでは 'IDL *n*' (*n* はウィンドウ番号)と表示されるウィンドウタイトルを指定する

- すでに開いている番号に開くと、前のウィンドウは消される
- 新しいウィンドウを開くと、それが current window になる

```
IDL> window, 1, xsize=800, ysize=600
```

開いたウィンドウを操作する

WDELETE[, window_index, ...]

- ✓指定した番号(window_index)のウィンドウを閉じる
- ✓指定が無ければ current window を閉じる
- ✓指定する番号は複数並べることが可能

WSET[, window_index]

- ✓ current window を指定した番号に切り替える

ERASE

- ✓開いている current window の中身を消す
  (背景色で塗りつぶす)

データを表示する

■ データ可視化のための主なコマンド (procedures)

- plot　　: 散布図、ライングラフ
- oplot　: 既存のグラフ上に別のグラフを重ね描き
- plots　: 既存のグラフの上に点または線を描く
- axis　　: 軸を定義して表示する
- xyouts　: グラフィック上に文字列を表示する
- contour: 3次元データをコントア表示する
- surface : 3次元データをメッシュで表現する

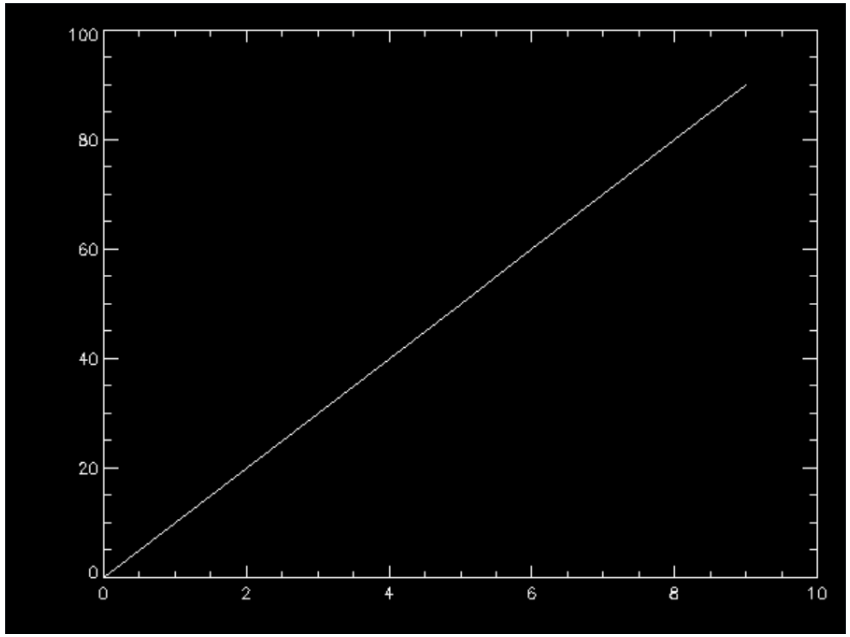
PLOT (procedure)

- (おそらく)もっとも頻繁に使う procedure
  ただし、より高機能な拡張されたライブラリツールもある(後述)
- もっとも基本的な使い方は、

plot, [x,]y　　(x, y はプロットするデータアレイ)

例)
IDL> plot, indgen(10)*10

※ indgen() はインデックス配列の生成コマンド



- ✓ 引数がひとつ(y)のみの場合、x は自動的にインデックス値(0,1,2,..)となる
- ✓ 描画範囲やレイアウトなどは IDL によって適当に調整される
  (手早くデータを確認したいときに便利)
- ✓ オプションを使うことで、様々な箇所をカスタマイズできる

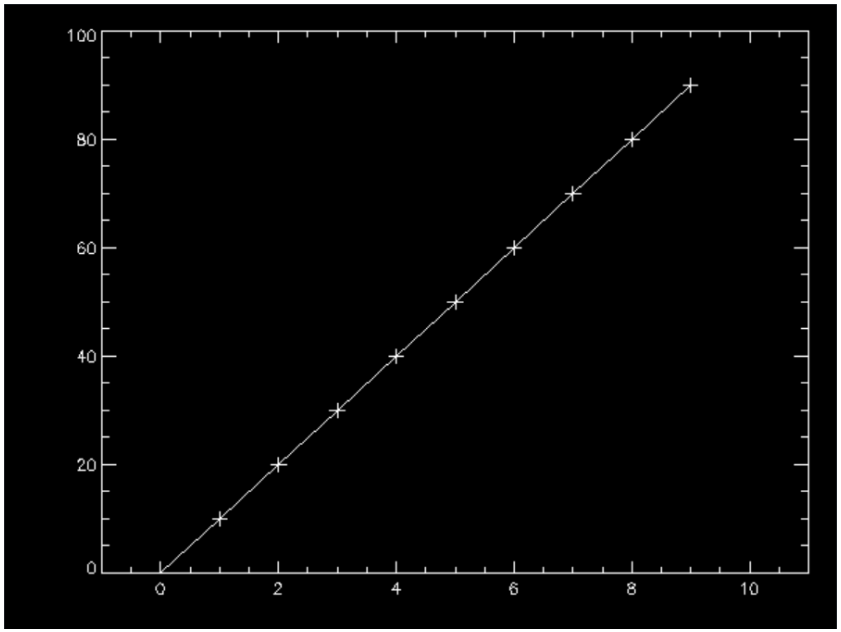
よく使うオプション (1)	
psym=number	データ点のシンボル 1: Plus sign (+) 2: Asterisk (*) 3: Period (・) 4: Diamond (◇) 5: Trianle (△) 6: Square (□) 7: X 8: User-defined 数字にマイナス符号(-)を付けると、シンボルと線の両方を表示する
symsize=value	シンボルのサイズ(default: 1.0)
{x|y}range=[min, max]	(x,y)軸の範囲(レンジ)の指定 指定しない場合、データ全体を含む様に自動調整される
/ynozero	yデータがすべて正の場合に、Y軸がゼロから描かれるのを禁止する

よく使うオプション (2)	
{x|y}style =value	(X,Y)軸の表示方法 下記数字の合計値で指定する 0: (default) IDLが判断した適切なレンジ 1: 指定値あるいはデータの最大・最小値の正確な範囲 2: 適当なマージンを設ける 4: 軸を描画しない 8: ボックススタイルの軸を表示しない(左側、下側の軸線のみ) 16: /ynozero と同じ(Y軸のみで有効)
/{x|y}log	対数(log)スケールでプロットする
/isotropic	X軸とY軸のスケールを合わせる
/nodata	軸のみを描いて、データはプロットしない
/noerase	既存のグラフィックスを消さずに、上に重ねて描く

オプションを使用した例

```
IDL> plot, indgen(10)*10, psym=-1, symsize=1.5, xrange=[-1,11], xstyle=1
```

- シンボルマーク“+”で、実線付き
- シンボルのサイズは標準の1.5倍
- X軸のレンジは -1 から 11 までの正確な範囲でプロット



さらに見栄えを整える

- オプションを使ってプロットの外見を整える
- これらのオプションの多くは、PLOT procedure だけではなく、CONTOUR procedure など他の多くのグラフィックスコマンドに共通

プロットの外見を整えるオプション類 (1)	
position=[x0, y0, x1, y1]	ウィンドウ内のプロット位置を指定。左下と右上の相対座標(0.0 - 1.0)で指定する。
{x|y}margin=[v0, v1]	プロット領域外側の余白サイズを指定する。 単位は charsize(文字サイズ)。 デフォルトはx:[10,3], y:[4,2] ただし、position が優先される。
title=string	プロットのタイトル
subtitle=string	プロットのサブタイトル
{x|y}title=string	(x|y)軸のタイトル
charsize=value	文字サイズ(default 1.0)
{x|y}charsize=value	(x|y)軸ごとの文字サイズ(default 1.0)
charthick=value	文字の太さ (default 1)
color=value	色の指定(default: highest color table index) 詳しくは後述。

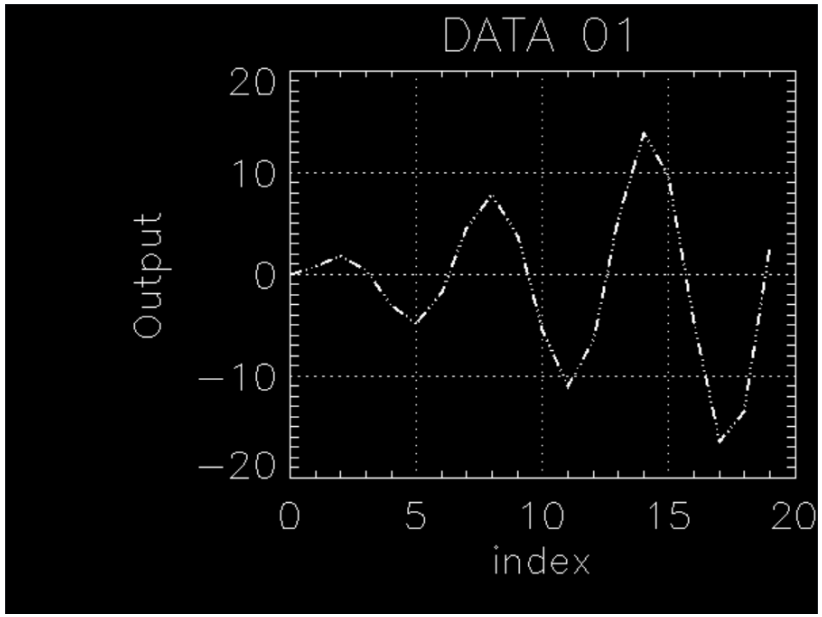
プロットの外見を整えるオプション類 (2)	
linestyle={0 to 5}	線の種類 0 Solid (default) 1 Dotted 2 Dashed 3 Dash Dot 4 Dash Dot Dot 5 Long Dashes
thick=value	線の太さ (default 1.0)
{x|y}thick=value	軸と目盛線の太さ (default 1.0)
{x|y}ticks=value	目盛の数
{x|y}minor=value	目盛間の補助目盛の数
ticklen=value	目盛り線の長さ (default 0.02; プロット範囲全体の長さに対する比)
{x|y}ticklen=value	(x,y)軸ごとの目盛り線の長さ (default 0.02)
{x|y}gridstyle={0 to 5}	グリッド線の種類(種類は linestyle と同じ) グリッド線を描くには ticklen=1.0 を設定する。

オプションを使用した例 (2)

```
IDL> plot, indgen(20)*sin(indgen(20)), xmargin=[12,1], $
IDL>    title='DATA 01', xtitle='index', ytitle='Output', charsize=2.5, $
IDL>    linestyle=4, thick=2.0, ticklen=1.0, xgridstyle=1, ygridstyle=1
```

※ 行末の '$' 記号はコマンド途中の改行。上の例で '$' を付けずに、一行ですべてを書いても良い。

各オプションで何を設定してあるか確認してみて下さい。



複数のデータを重ねてプロットする; OPLOT

- plot で最初のデータをプロットした後、oplot で2番目以降のデータを重ねてプロットする
- plot と同様に、thick, linestyle, psym, symsize 等のオプションが使える
- 表示される範囲(レンジ)は最初の plot で決まる
- 区別しやすいように色を変えるなどすると良い

color オプションによる色の指定 (decomposed colors)

Black	: '000000'XL
Red	: '0000FF'XL
Green	: '00FF00'XL
Blue	: 'FF0000'XL
Yellow	: '00FFFF'XL
Gray	: '7F7F7F'XL

(Blue, Green, Red の順にそれぞれ 00 から FF の間の 256 階調で指定する)

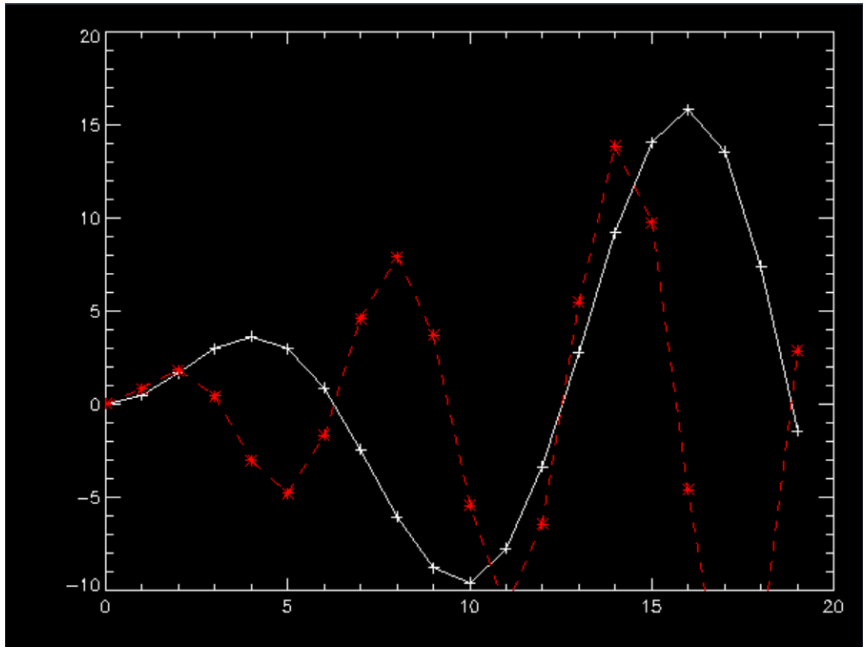
(注) 色の指定がうまく働かない場合は、IDL で使用しているカラーモデルが"decomposed color model" (通常はこれが起動時のデフォルト)になっていない。その場合は下記コマンドを実行する。

```
IDL>device, decomposed=1
```

OPLOT を使った例

最初のグラフの上に、別のグラフを赤色の破線で重ねる

```
IDL> plot, indgen(20)*sin(indgen(20)/2.0), psym=-1
IDL> oplot, indgen(20)*sin(indgen(20)), psym=-2, $
IDL>              linestyle=2, color='0000FF'XL
```



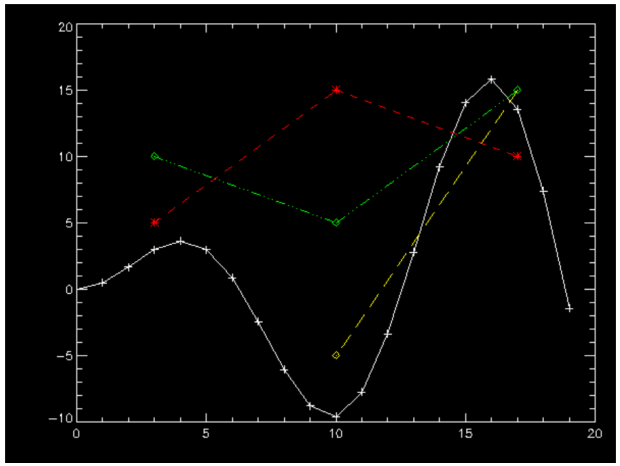
既存グラフィック(plotなど)の上に点や線を描く; PLOTS

- [使い方]

plots, x[, y], [, /continue]

✓ x, y はスカラーまたはベクトル(1次元アレイ)
✓ /continue オプションは最後のプロット点から続けて線を引く

例)
```
IDL> plot, indgen(20)*sin(indgen(20)/2.0), psym=-1
IDL> plots, [3,10,17], [5,15,10], psym=-2, linestyle=2, color='0000FF'XL
IDL> plots, [3,10,17], [10,5,15], psym=-4, linestyle=4, color='00FF00'XL
IDL> plots, 10, -5, psym=-4, linestyle=5, color='00FFFF'XL, /continue
```



既存のグラフィックの上の座標 x, y に文字列(string)を表示する; XYOUTS

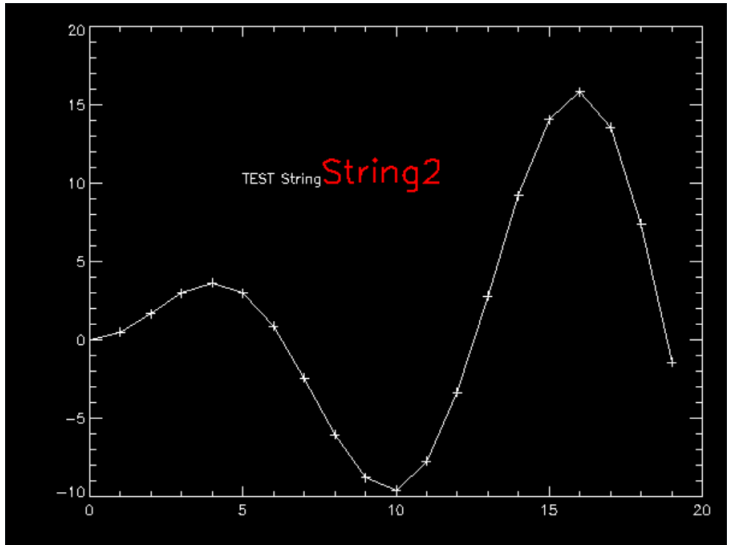
- [使い方]

```
xyouts[, x, y], string
```

✓ 座標 x, y を省略した場合は直前の最終位置が起点になる
✓ オプション charsize, charthick などが使える

例)

```
IDL>plot, indgen(20)*sin(indgen(20)/2.0), psym=-1
IDL>xyouts, 5, 10, 'TEST String'
IDL>xyouts, 'String2', color='0000FF'XL, $
IDL>            charsize=1.8, charthick=2
```



✓ デフォルトでは指定座標(x,y)が文字の左下
✓ ALIGNMENT オプションを使って右揃えや中央揃えにも出来る
✓ 座標の与え方については次ページ

IDL グラフィックスの座標

- プロットグラフィックス上で指定できる座標系は次の3種類が存在する
  - DATA ; plot 等を実行した後から有効になる座標系。プロットしたデータの値で定義される(デフォルト)
  - NORMAL; グラフィックウィンドウの左下を[0.0,0.0]、右上を [1.0,1.0] と定義した座標系
  - DEVICE ; グラフィックウィンドウの左下を原点[0, 0]として、ピクセル単位で定義される座標系

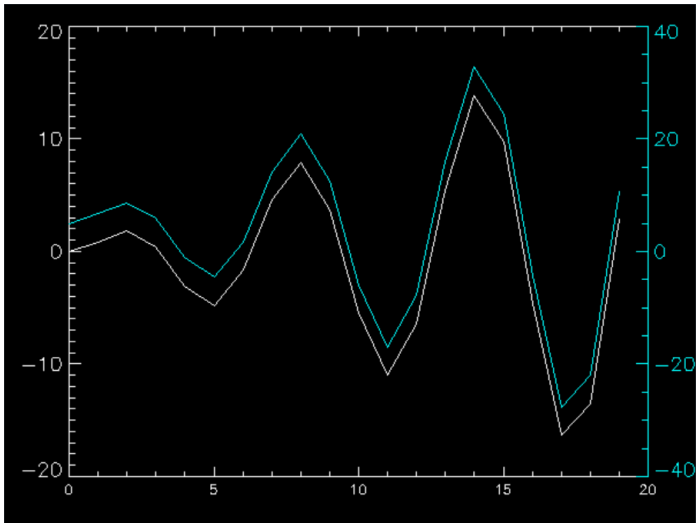
例) グラフィックウィンドウ中央を起点として文字を書く場合

```
IDL> xyouts, 0.5, 0.5, 'TEST String', /Normal
```

軸の作成; AXIS

- たとえば、異なるスケールを持つデータに対して、それぞれ独立のスケールのY軸を左右に作成する(第1軸と第2軸を描く)

```
  ; データ作成(2種類)
IDL> y1 = indgen(20)*sin(indgen(20))
IDL> y2 = indgen(20)*sin(indgen(20))*2+5
  ; データはプロットしないでX軸だけ描く
IDL> plot, y1, ystyle=4, xmargin=[8,6], /nodata
  ; 左側のY軸を描く
IDL> axis, yaxis=0, charsize=1.5
  ; 最初のデータをプロット
IDL> oplot, y1
  ; 右側のY軸を描く。
  ; /save オプションで新しいY軸のスケールを保存
IDL> axis, yaxis=1, yrange=[min(y2), max(y2)], $
IDL>          color='FFFF00'XL, charsize=1.5, /save
  ; 次のデータをプロット
IDL> oplot, y2, color='FFFF00'XL
```



画面を分割する; !p.multi

- グラフィックウィンドウを分割して、複数のプロットを描く
- !p はプロット関係の設定値を保存しているシステム変数構造体。(システム変数は先頭に ! が付いている)

(参考) システム変数の構造体 !p の中身を表示する

```
IDL>help, !p
```

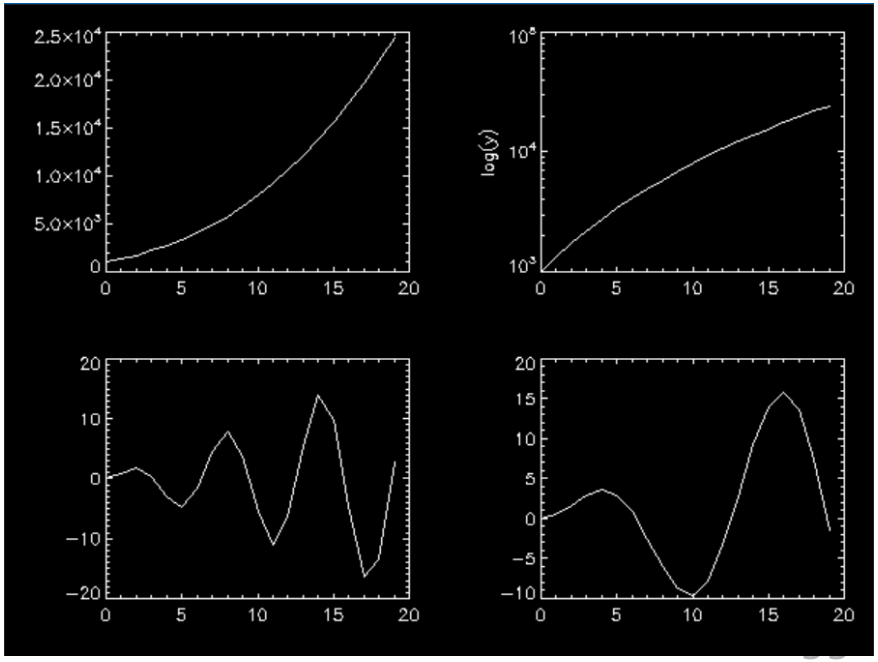
- !p で設定できる値のほとんどは plot や contour などのコマンドのオプションとして個別に指定できる
- しかし、画面分割を指定する !p.multi はオプションでは指定できない
- !p.multi は5つの要素を持つ配列。2番目の要素(!p.multi[1])と3番目の要素(!p.multi[2])がそれぞれ、画面を横と縦にいくつに分割するかを指定する値

(参考) デフォルト(起動時)の !p.multi

```
IDL> print, !p.multi
           0           0           0           0           0
```

例) 画面を4つ (2x2) に分割してプロットする場合

```
IDL> !p.multi=[0,2,2]   ; 配列の3番目まで書けばOK
  ;; これ以降、4つの領域に順番にプロットされる
IDL> plot, (findgen(20)+10)^3
IDL> plot, (findgen(20)+10)^3, /ylog, ytitle='log(y)' ; Y軸logスケール
IDL> plot, findgen(20)*sin(findgen(20))
IDL> plot, findgen(20)*sin(findgen(20)/2.0)
IDL> !p.multi=0          ; 分割設定を元に戻す
```



❗分割した画面の数を超えてプロットすると、また最初の位置から順番にプロットされる(前のプロットは消える)

position キーワード

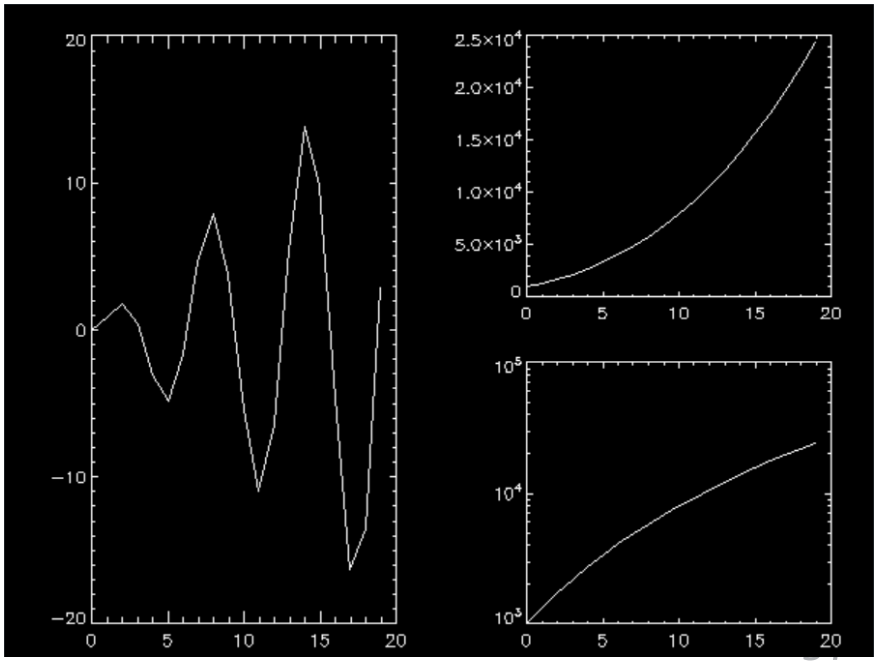
- !p.multi による画面分割は手軽だが、出力される位置や順番は固定されており、細かい設定は出来ない
- 代わりに、ひとつずつ位置 (position) を指定していくことで同じような複数プロットが可能
- [キーワードの書式]

```
position=[x_min, y_min, x_max, y_max]
```

例) 3つのプロットをレイアウトする

```
  ;; データを準備
IDL> dt1=(findgen(20)+10)^3
IDL> dt2=findgen(20)*sin(findgen(20))
  ;; 位置を指定しながらプロット
IDL> plot, dt1, position=[0.60, 0.55, 0.95, 0.95]
IDL> plot, dt1, position=[0.60, 0.05, 0.95, 0.45], /noerase, /ylog
IDL> plot, dt2, position=[0.10, 0.05, 0.45, 0.95], /noerase
```

(注) 2つめ以降は前のプロットを消さないように /NoErase オプションを付ける



Postscript ファイルに出力する

- グラフィックス出力先のデバイスを PS に切り替えてプロットする
  - 最後にファイルを閉じる処理を忘れずに (忘れやすい!)
  - 切り替えた出力デバイスも元に戻す

PS デバイス用のオプション (DEVICE procedure)	
filename	出力ファイル名 (default: idl.ps)
/closefile	ファイルを閉じる
/portrait	縦置き (default)
/landscape	横置き
/color	カラー出力 (default: grayscale)
{x|y}size=value	描画範囲のサイズ
{x|y}offset=value	描画範囲の原点位置
/inches	size と offset の単位を inch にする (default: cm)
font_size=value	フォントサイズ(単位:ポイント, default: 12 points)

例) ポストスクリプト出力手順例

```
;; 前準備
IDL> init_dev = !d.name                ; 現在の出力デバイスを保存しておく
                                       ; !d はデバイスに関するシステム変数の構造体
IDL> set_plot, 'PS'                    ; 出力先として PS デバイスを指定する
IDL> device, filename='output.ps'      ; 出力するファイル名(省略すると 'idl.ps')

;; プロットする
IDL> plot, indgen(20)*sin(indgen(20))      ; 画面には何も出力されない

;; 後処理
IDL> device, /close_file           ; 出力ファイルを閉じる
                                   ; ## これを忘れると白紙のままになったりする ##
IDL> set_plot, init_dev            ; 出力先のデバイスを元に戻す
```

IDL から Linux シェルに抜けて、出力した Postscript ファイルを確認

```
IDL>$
$ gv output.ps
```

- ✓ 通常時の出力デバイスは 'X' (Linux の X Window System の場合)
- ✓ Microsoft Windows の画面の場合 'WIN', プリンタ出力の場合 'PRINTER'
- ✓ PS にカラー出力する場合は decomposed=0 をセット(Indexed color を使用)

エラーバーのプロット

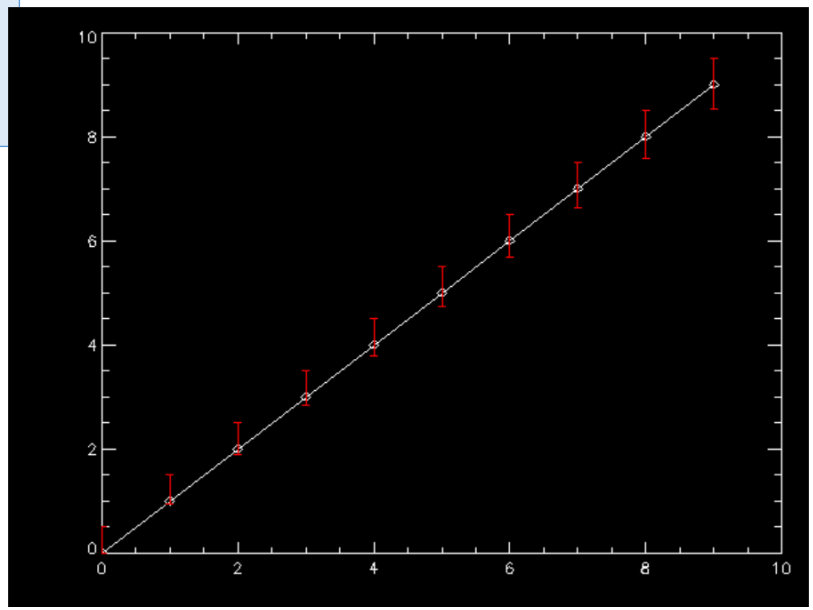
- ライブラリツールを含めていろいろある
- 使用方法や使えるオプションに差があるなどするので、用途と必要に応じて使い分けると良い

エラーバー表示	
errplot	既存プロットの上にエラーバー(正負個別指定)
ploterr	エラーバー付きプロット
oploterr	既存プロットの上にエラーバー(正負同じ)
ploterror	(astrolib) エラーバー付きプロット(x, y 方向) 正負個別プロット可
oploterror	(astrolib) 既存プロットの上にエラーバー(x, y 方向)
errorplot()	function (IDL 8.0 以降)

ERRPLOT を使った例

```
  ;; データ
IDL> dt = indgen(10)
  ;; 下側エラー
IDL> err1 = indgen(10)*0.05
  ;; 上側エラー
IDL> err2 = make_array(10, /float, value=0.5)

  ;; プロット
IDL> plot, dt, psym=-4
IDL> errplot, dt, dt-err1, dt+err2, color='0000ff'xl
```



CONTOUR (procedure)

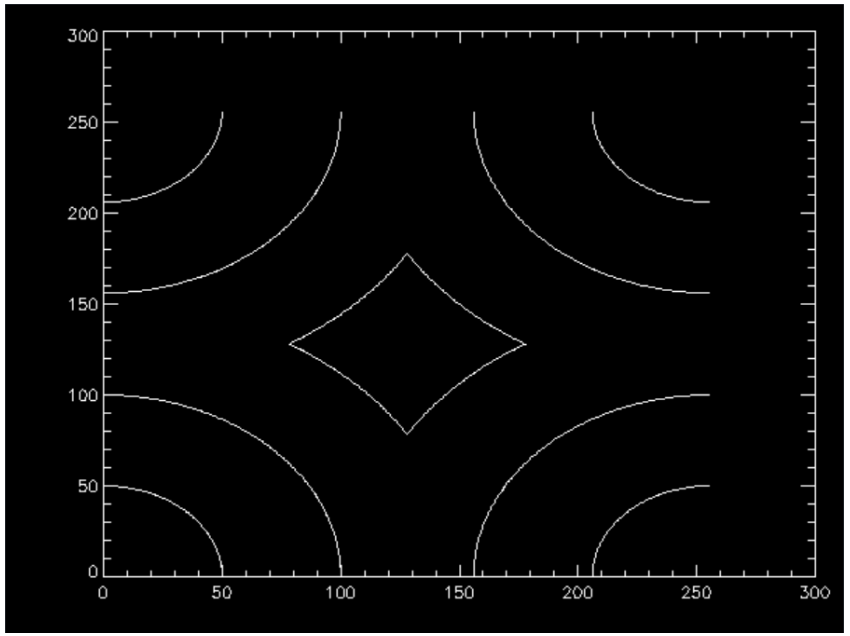
- 2次元イメージ(画像)をコントアで表示する
- [使い方]

contour, z[, x, y]　(z は 1 or 2次元データアレイ)

例)
IDL>contour, dist(256)

※ サンプルデータとして dist(256) で 256x256 の2次元データを生成して、それをコントア表示している

✓x, y 座標値を与えない場合はピクセル番号でマップされる

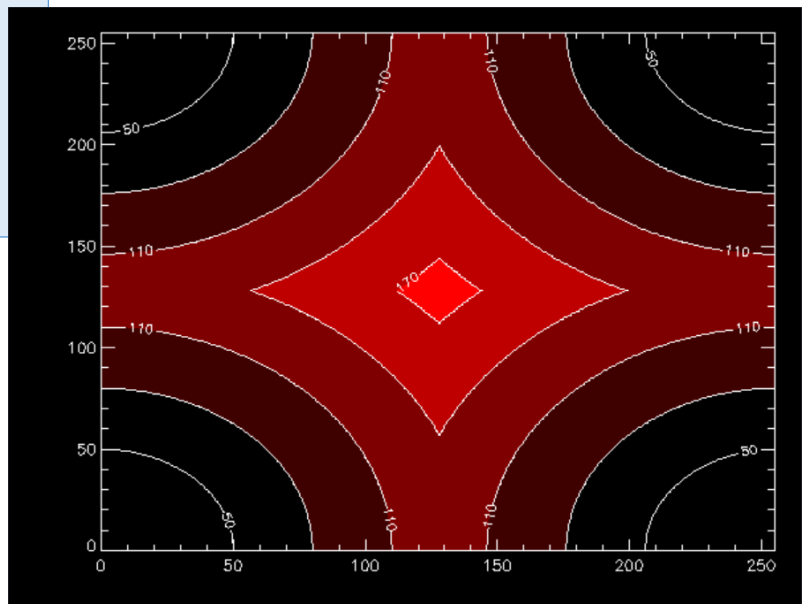


CONTOUR でよく使うオプション	
levels	コントアレベル(任意値)
nlevels	等高線の数(整数 1-60)
/follow	等高線の値を表示(自動調整)
c_labels	ラベル(等高線の値)の任意表示切替 (0/1, default 0)
c_charsize	コントアラベルの文字サイズ
c_color	コントア線の色
c_linestye	コントア線の種類 (0-5, default 0)
/overplot	既存グラフィックスの上にプロット
/fill	レベルごとに色を付ける
/cell_fill	レベルごとに色を付ける
max_value	プロットされる最大値
min_value	プロットされる最小値
/irregular	不規則に配置されたデータでプロット (x, y が必須)

コントアプロット例

- 等高線5本で、ラベルはひとつおきに表示、色を付けてコントア表示する

```
 ;; 等高線の設定値
IDL> lev=[50,80,110,140,170]
IDL> c_lab=[1,0,1,0,1]
 ;; 色を付けてコントアプロット
IDL> contour, dist(256), levels=lev, /fill, /xsty, /ysty
 ;; ラベルを付ける(重ねる)
IDL> contour, dist(256), levels=lev, c_labels=c_lab, /over
```



✓ 色づけとラベル表示は同時に行えないため、2度に分けて描画している

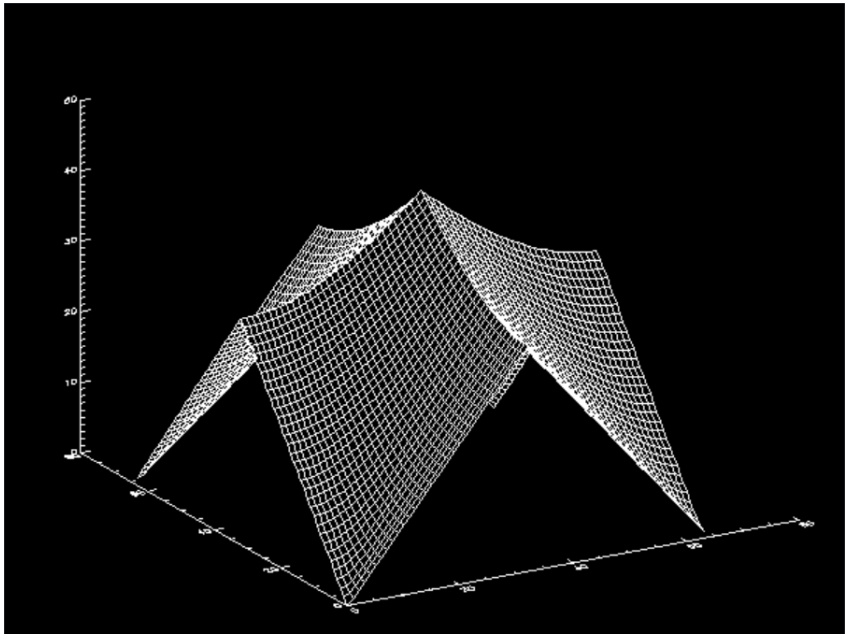
SURFACE (procedure)

- 2次元イメージ(画像)をメッシュ表示する
- [使い方]

surface, z[, x, y]　(z は 1 or 2次元データアレイ)

例)
IDL>surface, dist(64)

✓x, y 座標値を与えない場合は
ピクセル番号でマップされる



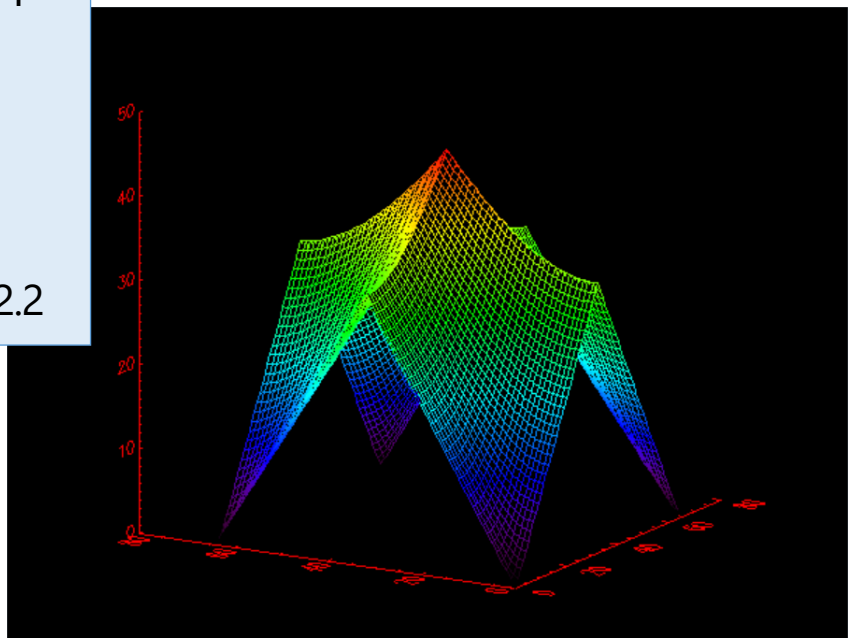
SURFACE でよく使うオプション	
az	Z 軸周りの回転角(度) (右回り、default 30度)
ax	X 軸周りの回転角(度) (右回り、default 30度)
shades	z レベルに対するカラーインデックス (default テーブル最大値)
skirt	裾を表示する場合の z レベル (指定が無ければ裾無し)
/lower_only	下面のみを表示
/upper_only	上面のみを表示
min_value	表示する最小値
max_value	表示する最大値
/{x|y|z}log	(x,y,z)軸を対数スケールにする

surface プロット例

- Indexed color モードにして、
  レインボーカラーテーブルを使って表示する

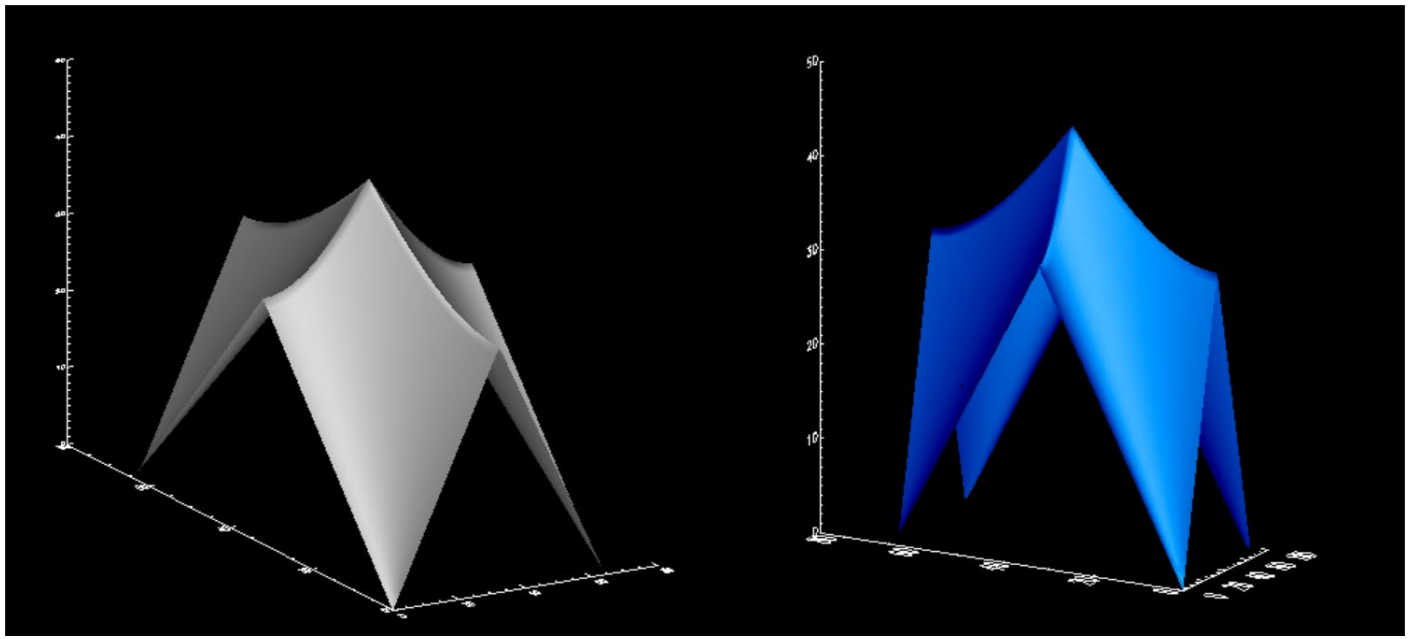
```
 ;; カラーモードをセットしてカラーテーブルを使用
IDL> device, decomposed=0
IDL> loadct, 13 ; レインボーカラーテーブルのロード
 ;; z データ作成
IDL> z=dist(64)
 ;; カラーインデックスをスケーリング
IDL> shades=bytscl(z)
 ;; 描画
IDL> surface, z, az=60, ax=20, shades=shades, chars=2.2
```

※ カラーモードやカラーテーブルの説明は後述



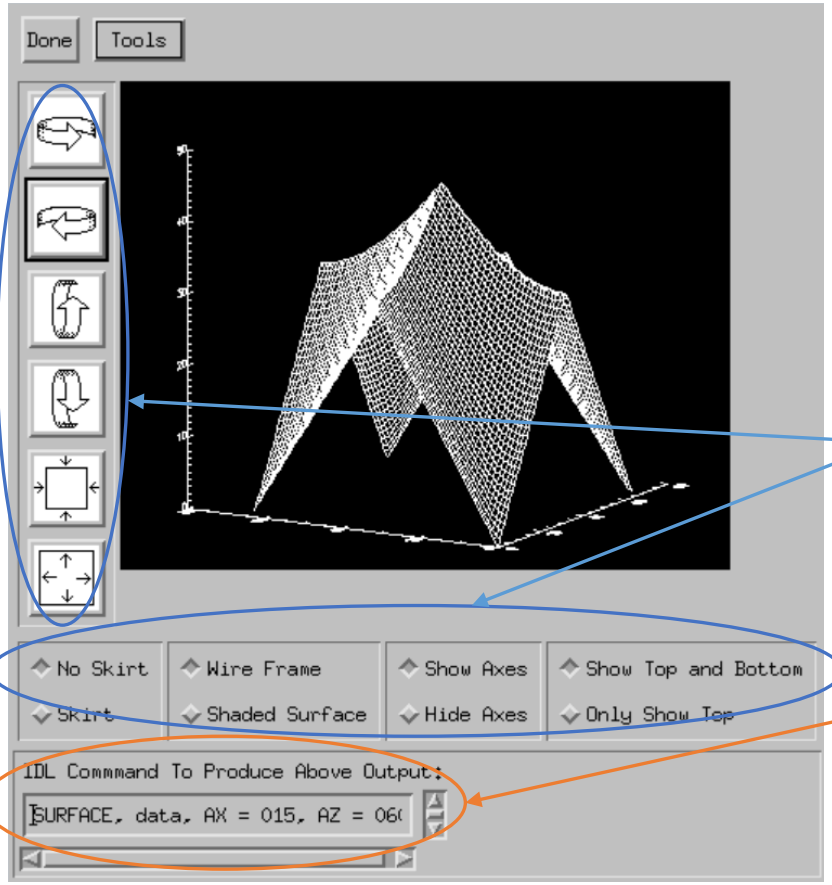
SHADE_SURF (procedure)

- 2次元イメージ(画像)を連続的な影付きの面として表示する



```
; カラーテーブル1番を使用
IDL> device, decomposed=0
IDL> loadct, 1
```

XSURFACE



- surface, shade_surf の結果をGUIで手軽に確認できるユーティリティーツール
- 表示角度や、いくつかのオプションが簡単に試せる
- それぞれの結果を得るためにコマンドラインに入力するべきコマンドも表示される

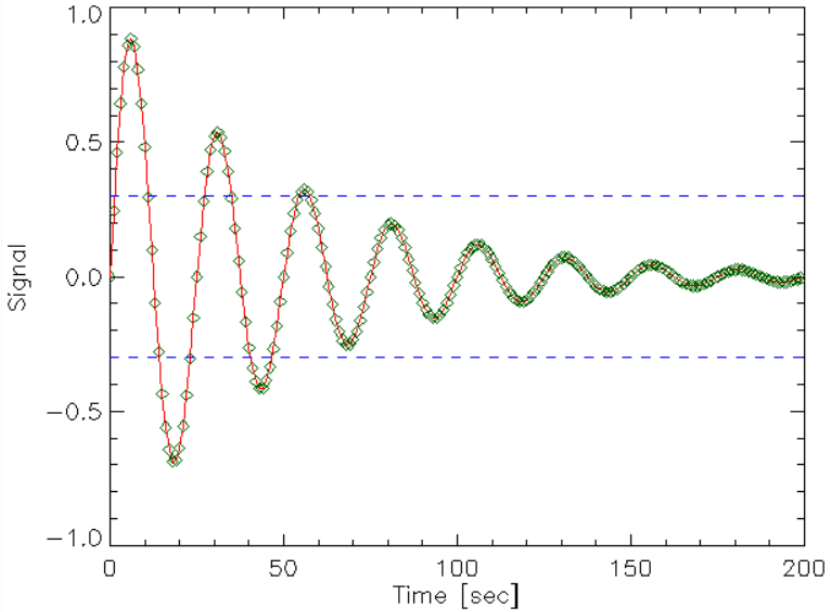
CGPLOT

- Coyote IDL Program Libraries に含まれるプロット作成プロシージャ
- Astronomy User's Library にも導入されている (一緒に配布されている)
- IDL の本家 PLOT プロシージャよりも多機能で、見た目的にも優れたプロットが容易に作成できる
- 基本的に本家 PLOT の拡張になっているので、オプションなどほぼ同様の使い方が可能で、PLOT の代わりとして(置き換えて)使える

❗ Plot のほかにも、Contour, Surface, TV などの代替となる各種ルーチンが Coyote Graphics System (CGS) として公開されている

```
IDL> data = sin(2.0*findgen(200)*!PI/25.0)*exp(-0.02*findgen(200))
IDL> cgPlot, data, psym=-4, color='red', $
IDL>              symcolor='darkgreen', symsize=1.2, $
IDL>              xtitle='Time [sec]', ytitle='Signal'
IDL> cgPlot, [0,200], [0.3,0.3], linestyle=2, color='blue', /Overplot
IDL> cgPlot, [0,200], [-0.3,-0.3], linestyle=2, color='blue', /Overplot
```

- ✓ デフォルトの背景色は白色
- ✓ 色の指定に名前 (red, blue, cyan, etc.) が使える
- ✓ cgPlot は Plot とともに OPlot の拡張 (wrapper) プログラムになっている。/Overplot オプションを使用すると、前の描画の上に重ねて表示される



Coyote Graphics によるグラフィックスのファイル出力

- cgPlot 等は Output オプションを使用してグラフィックスを直接ファイル出力する機能を持つ
  - ただし、ImageMagick および Ghostscript がインストールされている環境が必要

```
cgPlot, x[, y][, Output={'PS'|'EPS'|'PDF'|'BMP'|'GIF'|'JPEG'|'PNG'|'TIFF'} ]
```

あるいは、ファイル名を直接指定する。形式は拡張子で判断される
(例 Output='out.png')

- ポストスクリプト出力デバイス切り替えルーチン
  cgPS_Open, cgPS_Close (各種オプション使用可)

(例)

```
IDL> cgPS_Open, file='test.ps'
IDL> Plot, indgen(10), psym=-4
IDL> cgPS_Close
```

ヒストグラム作成

- IDL には入力データの密度分布を求めるための HISTOGRAM 関数が用意されている
- HISTOGRAM() で得た密度分布をプロットすることでヒストグラムプロットが得られる
- しかし、この IDL 標準の手順はいささか面倒なので、ここでは一気にヒストグラムプロットを行う Coyote Libraries の CgHistoPlot プロシージャを紹介する
- Astronomy Library には含まれて(一緒に配布されて)いない

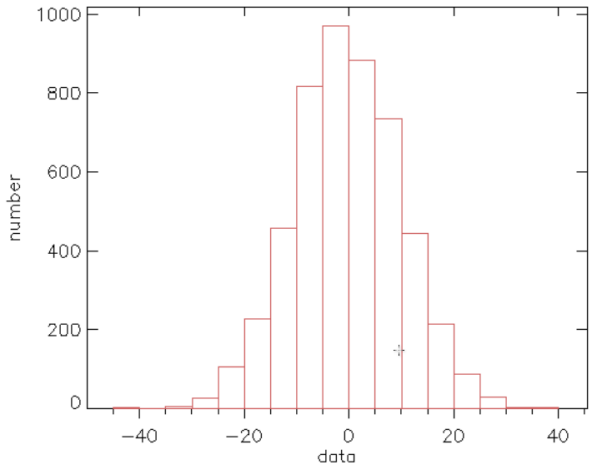
CGHISTOPLOT

cgHistoPlot, data[, binsize=value, nbins=value, datacolorname=string, /fillpolygon, polycolor=string , histdata=variable, locations=varibable, ...]

- ✓ 区切り(bin) は binsize または nbins で調整
- ✓ ヒストグラムの色を指定したり、塗りつぶしたりもできる
- ✓ histdata, locations は出力変数。ヒストグラムデータを再利用したい場合にこの変数に保存する
- ✓ 他にも多数のオプションが存在する (cghistoplot.pro のヘッダの説明参照)

```
IDL> data = RANDOMN(seed, 5000)*10
IDL> cghistoplot, data, bins=5, histd=hd, loc=loc, $
IDL>                     xtitle='data', ytitle='number'
```

(※ randomn() はガウス分布の乱数生成コマンド)



関数型のグラフィックスルーチン

- IDL 8.0 以降、**オブジェクトグラフィックス**による新しいグラフィックスルーチンが導入された
- PLOT ほか基本的なグラフィックス機能が一通り用意されている
- 従来のダイレクトグラフィックスによるルーチンがプロシージャ型だったのに対して、関数型になっている
- IDL6.0 から iTools というオブジェクトグラフィックスルーチンが導入されていたが、動作が重いなどあまり評判は良くなかった(と思う)
- 新しい関数型ルーチンは iTools からは大幅に改良されいてる
- とは言え、ダイレクトグラフィックスに比べると軽快さには劣る
- オブジェクトの属性を後から変更できる、保存や印刷機能が備わっている、などの利点もあるので、試してみて、用途に応じて使い分けると良いだろう

```
IDL> y = sin(2.0*findgen(200)*!PI/25.0)*exp(-0.02*findgen(200))
IDL> ; Plot
IDL> p1 = plot(y, SYMBOL=4, COLOR='blue')
IDL> p2 = plot(y+0.1, COLOR='magenta', linestyle=3, /overplot)
IDL> ; オブジェクトのプロパティを変更
IDL> p1.SYM_INCREMENT = 5
IDL> p1.SYM_COLOR = "orange"
IDL> p1.SYM_FILLED = 1
```

ツールバー:
プロパティの変更、グラフィックスファイル出力、印刷、線や文字などの追加などが GUI からも可能

軸などを選択した状態でマウスホイールを使った拡大縮小も可能