IDL9.0 入門

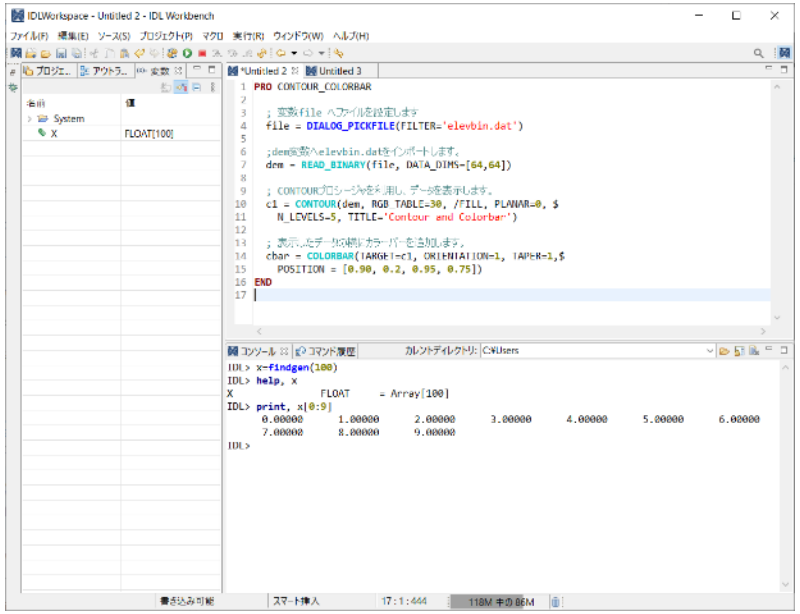
NV5 Geospatial株式会社

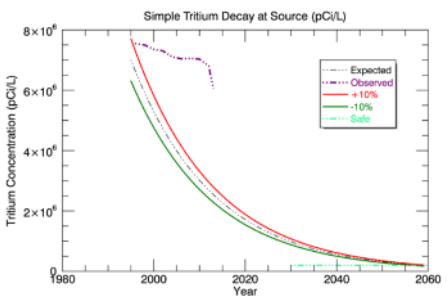
IDLについて

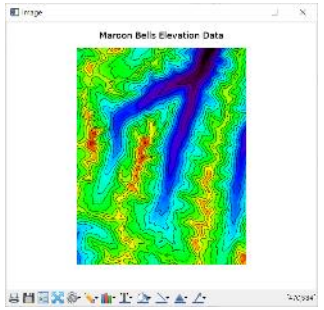
IDLの概要

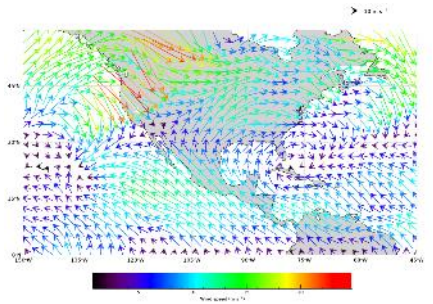
IDL: Interactive Data Language

- データ解析や可視化など多種多様なルーチンを搭載
- クロスプラットフォーム対応
- 配列指向型言語で大容量データのハンドリング効率化
- 統合開発環境の提供
- GUIアプリケーションの開発機能と無償実行環境の提供









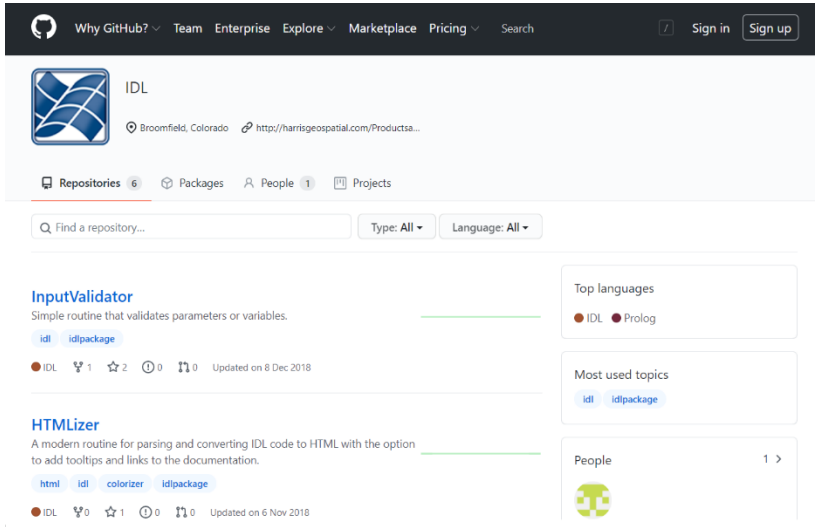
IDLのリソース

IDL Forums

- IDL Discussion は当社ホームページ上に設けられており、ユーザ同士が問題の討議や意見の交換が行えます。
- https://www.nv5geospatialsoftware.com/Support/Forums/aff/217

GitHub

- 米国本社によって管理・運営されているGitHub のページでは、IDLに関する様々なコードが公開されています。
- https://github.com/interactive-data-language

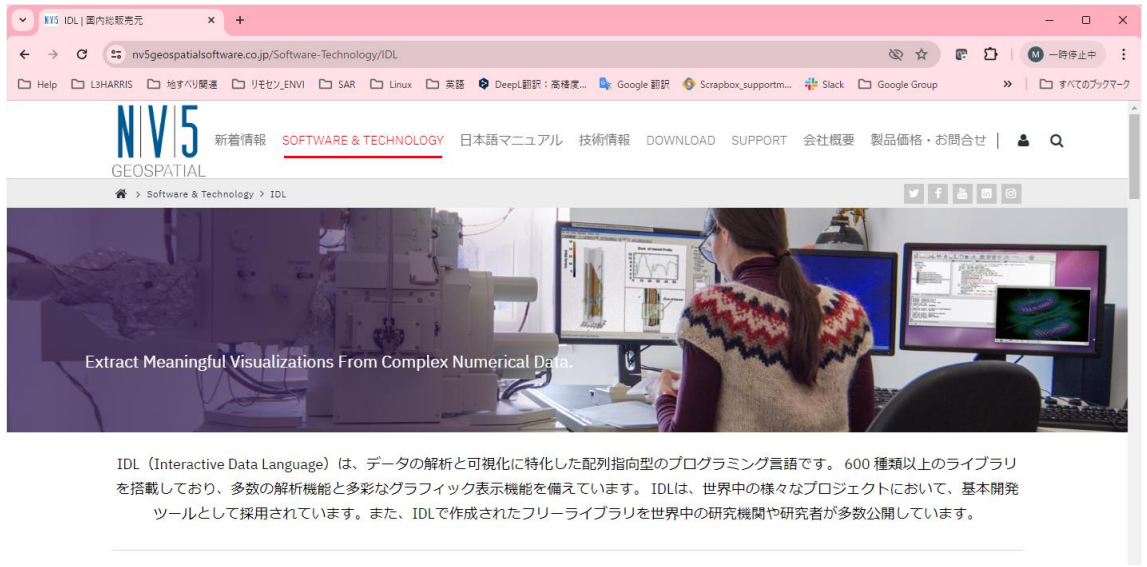


問い合わせ先

NV5 Geospatial株式会社では、IDL 及びその他の製品に関する技術サポート、お客様の使用事例などの情報を提供しています。

NV5 Geospatial 製品・サポート連絡先
E-mail：
support_jp@NV5.com

Home Page：
https://www.nv5geospatialsoftware.co.jp/



演習に入る前に

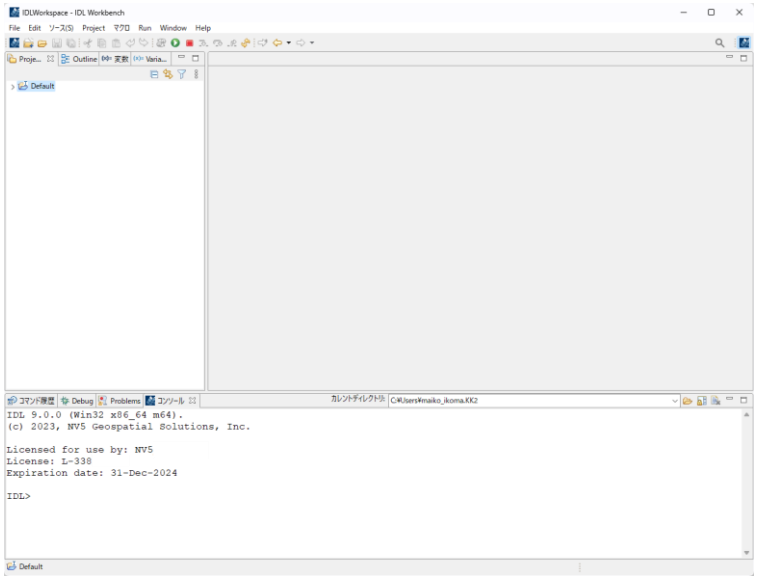
IDLの起動方法やUIについて説明します

IDLワークベンチの起動方法

Windows
[Start Menu] > IDL9.0 > IDL9.0

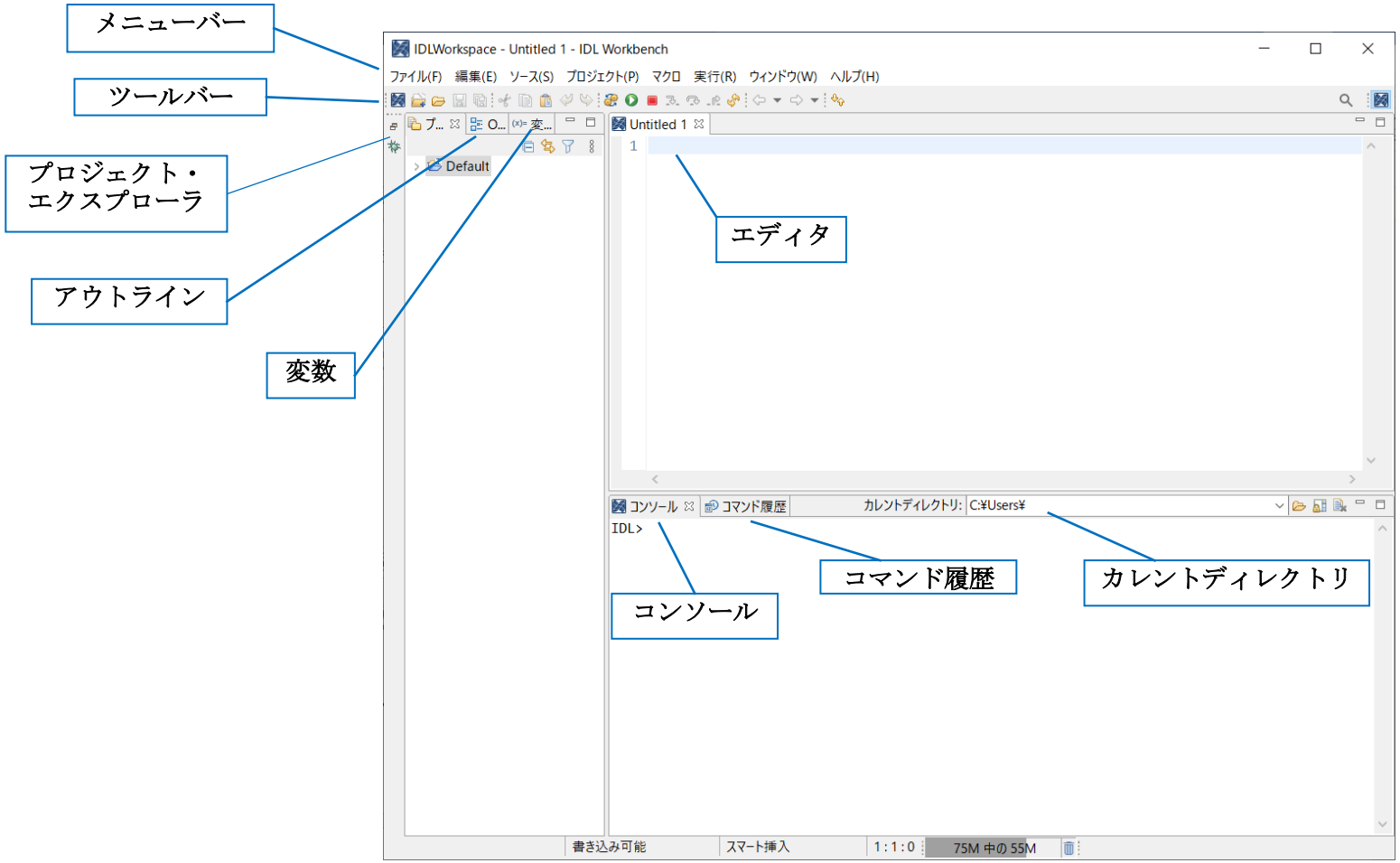
Mac
/Applications/NV5/ -> IDL9.0のアイコンを選択

Linux
シェルプロンプトでidlde と入力してください。



IDL Workbench

IDLワークベンチ インターフェースについて



Note:ワークベンチにはEclipse フレームワークを使用しており、Eclipse はすべてのプラットフォームでネイティブアプリケーションとして表示される拡張クロスプラットフォーム環境です。そういったことから、ワークベンチは全てのプラットフォームで同じ外観、及び操作性を提供しています。

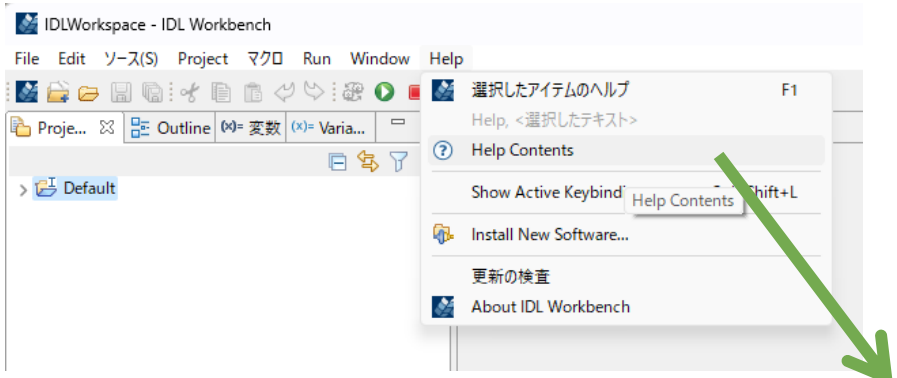
IDLワークベンチの機能紹介 1

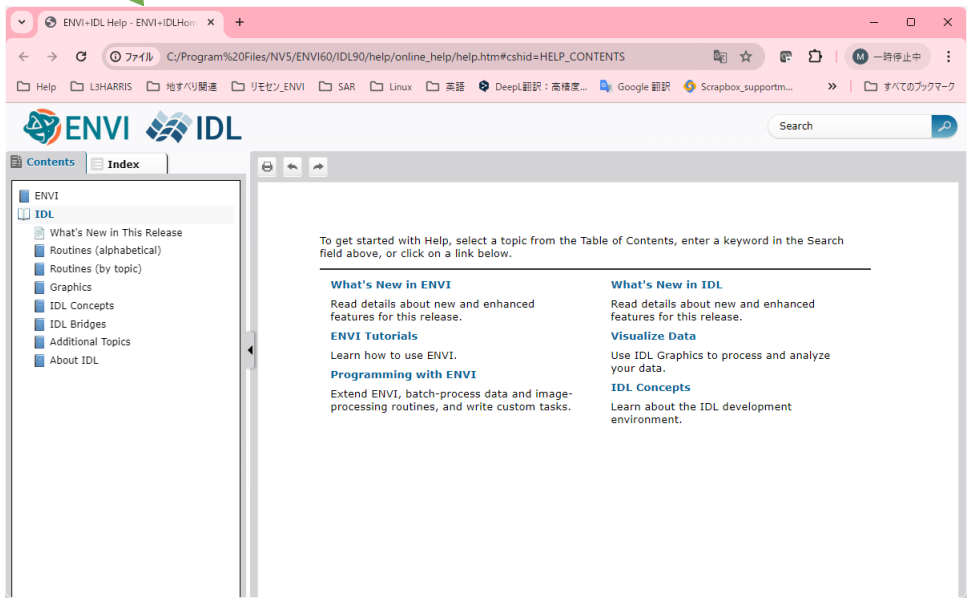
ワークベンチビュー	機能
メニューバー	ファイルを開く、編集、コンパイル、IDLプログラムを実行など、その他ワークベンチの機能を操作します。
ツールバー	メニューをグラフィカルなアイコンで提供します。
プロジェクト・エクスプローラ	ファイルシステム上のIDLのプログラム、データ、サポートファイルディレクトリを表示しています。ファイル名をダブルクリックすることでファイルを開きます。
アウトライン	ファイル内のプログラムのリストを表示します。プログラムをダブルクリックし、エディタにソースを表示し、アウトラインを確認してください。
エディタ	IDLプログラムを編集し、デバックするウィンドウです。各構文が色分けされています。
変数	コンソールで入力された、またはIDLのデバッグ時にストップされた際に使用されている変数値をリスト表示しています。

IDLワークベンチの機能紹介 2

ワークベンチビュー	機能
コンソール	コマンドプロンプト機能： 「IDL>」の行にステートメントを入力することで、インタラクティブに変数や動作の確認を行うことができます。 また、IDLアプリケーションのコンパイル、実行などのコマンドもここから実行することができます。 コマンドプロンプト上で、カーソルキーの上向き矢印を押下することで、入力したコマンドの履歴を表示し、選択した履歴をそのまま実行することが可能です。 IDLではコマンドの入力時に、大文字・小文字は区別しておらず、IDL内では全て大文字として認識し、実行されます。 コンソール機能： コマンドプロンプトで実行した内容のログが出力されます。コンパイル情報やアプリケーションの実行時のエラー確認を行うことができます。
コマンド履歴	過去に入力したコマンドを確認することができます。また、このリストからコンソールへドラッグ＆ドロップまたはダブルクリックすることでIDLステートメントをそのまま実行することが可能です。
カレントディレクトリ	作業ディレクトリの指定を行います。隣にあるフォルダマークをクリックすることで作業ディレクトリを変更することが可能です。

IDL HELP



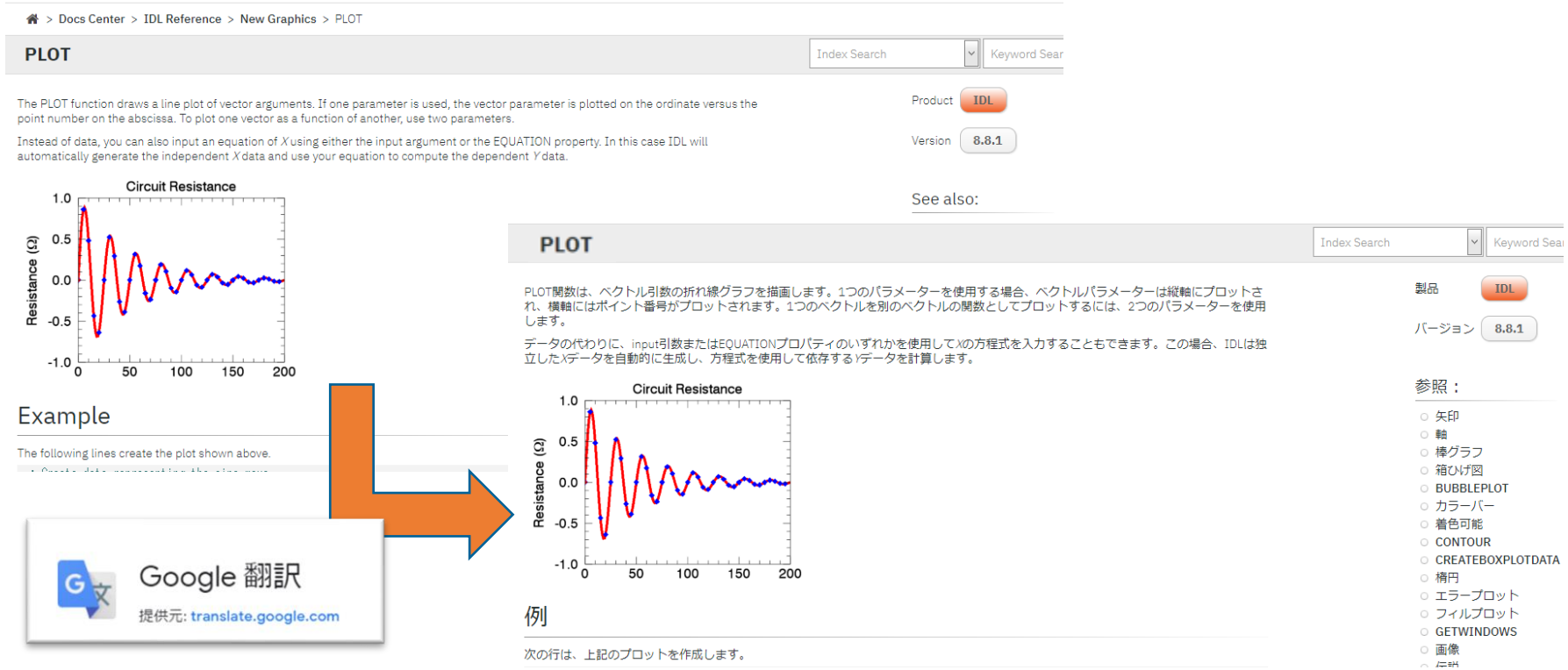


指定のブラウザを使用して、
IDLのHELP機能が起動します。

Note: IDLについてほとんどすべての内容を理解することができます。本入門トレーニング内でも度々参照しますので、ヘルプシステムは開いておくと便利です。

DOCUMENTATION CENTER

- IDLのHELP機能と同じ内容のものが米国本社のホームページ上のDocumentation Centerで提供されています。
- ブラウザによっては翻訳アドオンを適用することが可能なので、機械翻訳によって大まかな意味を捉えるのであればこの方法が便利です。
  - ✓ https://www.nv5geospatialsoftware.com/docs/home.html



作業ディレクトリの設定

■ファイルを開く、保存するときに IDL が使用するデフォルトの場所です。

■作業ディレクトリを設定

本書で使用するデータが格納されているディレクトリを作業ディレクトリに設定します。

以下のコマンドを、IDLワークベンチのコンソール画面から入力し作業ディレクトリを変更してください。

```
IDL> CD, 'C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data'
```

※XXはバージョンを表しています。

IDLの基本知識

変数とは

■データを格納する名前付きの容器。以下が関連付けられます。
- ●データタイプ
  - – バイト型、整数型、浮動小数型など
- ●構造
  - – スカラ、配列、構造体、リスト、ハッシュなど

■変数の宣言不要
- ●ダイナミックデータタイプ(作成時に自動決定)

例)

```
IDL> a = 1
IDL> HELP, a
A               INT       =        1

IDL> b = 0.1
IDL> HELP, b
B               FLOAT     =      0.100000
```

配列とは

同じデータタイプのデータを並べたデータ構造の一種です。

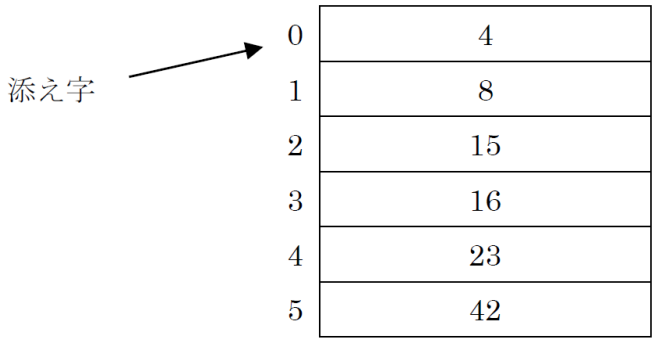
■IDLは配列指向型言語

●演算子とライブラリルーチンでは、配列とスカラのどちらも使用可能

●データは配列型で保持

例)

```
IDL> x = [4, 8, 15, 16, 23, 42]
IDL> HELP, x
X INT = array[6]
```



■配列作成ルーチン

●上記のように配列を作成することもできますが、IDLは配列を作成するためのルーチンが多数ある

– BYTARR関数はバイト型の配列を作成する関数です。
引数の10は配列の要素数を示しています。

例)

```
IDL> barr = BYTARR(10)
IDL> HELP, barr
BARR            BYTE      = Array[10]
IDL> PRINT, barr
   0   0   0   0   0   0   0   0   0   0
```

Note: 配列の作成ルーチンについてはIDLヘルプシステムの以下の項目をご参照ください。
IDL > Routines (by topic) > Array Creation

IDLルーチンの呼び出し

ファイルの読み込み、書き込みなどの一般的なルーチンから、データのノイズ除去やプロットなどのデータ解析・可視化に特化したルーチンまで、非常に多くのルーチンを持っており、以下の2つのタイプがあります。

■関数

●戻り値あり

左辺に戻り値、右辺には関数名を記述し括弧内にカンマ区切りで引数やキーワードを設定します。

例）

```
IDL> ret = sampleFunction(arg1,arg2)
```

■プロシージャ

●戻り値なし

プロシージャ名の後にカンマ区切りで引数やキーワードを設定します。

例）

```
IDL> sampleProcedure, arg1, arg2
```

引数とキーワード

IDLルーチンには様々な入出力パラメータが用意されています。

■引数
- ●ルーチンを呼び出す際に必須のパラメータ
- ●引数は位置パラメータ。位置によって指定（第一引数、第二引数.....）

■キーワード
- ●省略可能なパラメータ
- ●引数以降に指定し、位置、順序は関係なし
- ●キーワード名に値や変数を代入する
- ●省略した場合、ルーチン内部ではデフォルト値で処理

例）

```
IDL> x = BINDGEN(100)
IDL> p = PLOT(x, COLOR='red', THICK=2)
```

引数: 1行目の100、2行目のx
キーワード: 2行目のCOLORとTHICK

Note: それぞれのルーチンの書式についてはIDLヘルプシステムの各ルーチンのページの「Syntax」を参照してください。また、「Syntax」の見方については以下の項目をご参照ください。
IDL > IDL Concepts > Getting Started > IDL Documentation Syntax

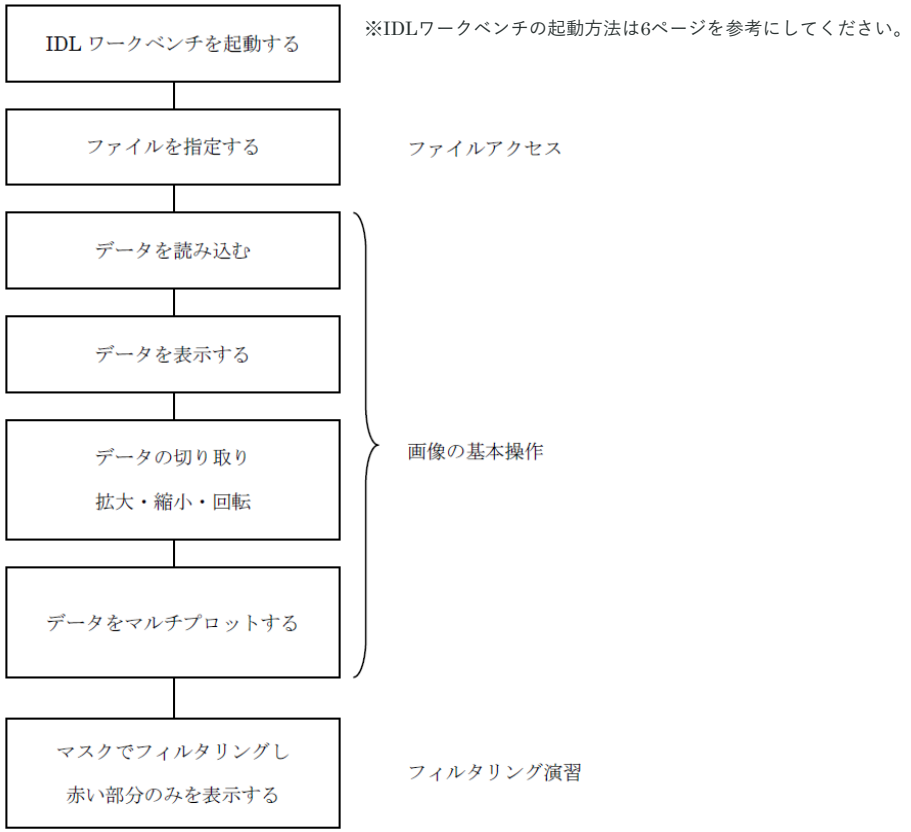
演習1
ファイルアクセスと画像の操作

ファイルアクセスと画像の操作について学習します

はじめに

この章では、バラのJPEG画像を使用し、ファイルアクセスと画像操作について以下の項目を学習します。IDL でのファイルの指定方法と、IDL へのデータの読み込み方法を学習し、続いて画像データの基本操作（表示・切り取り・拡大 縮小・回転）を学習します。最後に、画像処理の演習として、フィルタリング処理を行い画像内の赤い部分を抜き出します。

■演習内処理フロー



ファイルを指定する

データの読込みを行うには、まずどのファイルからデータを読み込むか指定する必要があります。IDLではどのようにファイルを指定するのでしょうか。ここではIDLのファイルアクセスについて説明します。

■直接指定する

データファイルのパスとファイル名をクォーテーションで囲み、ファイルを指定します。とても単純な指定の方法です。

```
IDL> file = $
>  'C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥Day.jpg'
IDL> PRINT, file
C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥Day.jpg
```

■ファイルアクセスルーチンを使用する

IDLにはファイルを指定するためのいくつかのルーチンが用意されています。

以下の代表的なファイルアクセスルーチンを次項から説明します。

- FILE_SEARCH関数
- FILE_WHICH関数
- DIALOG_PICKFILE関数

ファイルアクセスルーチン

NV5 GEOSPATIAL

■FILE_SEARCH関数

引数で指定したディレクトリ内のファイルを検索します。見つかったすべてのファイルパスを文字列型で返します。

```
IDL> file = FILE_SEARCH('*.tif')
IDL> PRINT, file
boulder.tif
examples.tif
image.tif
```

■FILE_WHICH関数

第一引数で指定したディレクトリ内から第二引数で指定したファイルを検索します。最初に見つかったファイルパスを文字列型で返します。ディレクトリを指定しない場合は、システム変数「!PATH」のディレクトリ内を検索します。

```
IDL> file = FILE_WHICH('people.dat')
IDL> PRINT, file
C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥people.dat
```

Note:「!」記号から始まる変数はIDLのシステム変数です。システム変数とは、すべてのプログラムで使用できるIDLで定義済みの変数です。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > IDL Concepts > IDL Programming > Variables > System Variables > Constant System Variables

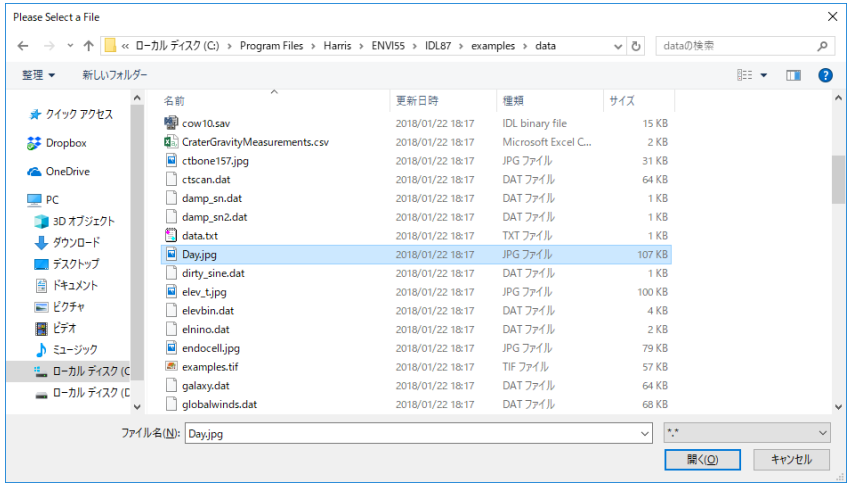
Note: !PATHシステム変数のデフォルトは、<IDL_DEFAULT>というIDLのインストールディレクトリのlibやexamplesを含む特殊なトークンです。!PATHにはユーザがパスを追加することもできます。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > IDL Concepts > IDL Programming > Variables > Environment Variables

ファイルアクセスルーチン

■DIALOG_PICKFILE関数
ファイルを選択するためのダイアログを表示し、対話的にファイルを指定するための関数です。

以下のようなダイアログが表示されます。ダイアログを使用し、以下のファイルを選択します。選択したファイルパスが文字列型でfile変数に代入されます。

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥Day.jpg




```
IDL> PRINT, file
C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥Day.jpg
```

Note: 関数の詳細や、その他のファイルアクセスルーチンについてはIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Input/Output

JPEGファイルを読み込む

READ_JPEGプロシージャを使用しJPEGファイルを読み込みます。また、データがどのように読み込まれたかHELPプロシージャで確認します。

【使用するデータ：バラの画像】

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥rose.jpg

1. 読み込むファイルを指定します。

```
IDL> file = 'rose.jpg'
```

2. READ_JPEGプロシージャを使用し、rose変数へrose.jpgを読み込みます。

```
IDL> READ_JPEG, file, rose
```

3. 読み込んだデータを調査します。

```
IDL> HELP, rose
ROSE            BYTE      = Array[3, 227, 149]
```

Note: IDLは一般的なデータフォーマットの読み込み / 書き込みを行うための便利なルーチンを多数持っています。IDLのファイルの読み込み / 書き込みについての詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Input/Output

データを表示する

IDLでは、読み込んだデータを表示するための多数のルーチンが用意されています。ここでは画像データを表示するため、IMAGE関数を使用します。

1. 読み込んだデータをIMAGE関数で表示します。IMAGE関数は画像データを表示するためのルーチンです。画像データのための引数には2次元配列、もしくは3次元配列を指定します。データが2次元なら白黒で、3次元のいずれかの次元が3であったら、IDLはそれをRGBデータとして表示します。

```
IDL> i = IMAGE(rose)
```

Note: IDLには他にも多くの表示ルーチンが用意されています。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Graphics

便利機能：コンテンツアシスト
IDLワークベンチにはコンテンツアシストという便利な機能が搭載されています。コンソールやエディタ上にマウスカーソルを置いて、Ctrl+Spaceキーを押すとポップアップされます。コンテンツアシストには選択した文字列から始まる、変数やルーチンの一覧がリスト表示されます。Macintoshでは、Ctrl+SpaceキーがOSにより別のキーボードショートカットとして割り当てられているため有効ではありません。ご使用の際はOSによるショートカット割り当てを無効にするか変更してください。LinuxはShift+Ctrl+Spaceキーを使用してください。

データを切り取る

IDLは画像を配列型で持つため、配列の一部のデータを抜き出すだけで、画像の一部を切り取ることができます。

1. ここでは横40-190、縦10-120を切り取ります。

```
IDL> clip = rose[*, 40:190, 10:120]
IDL> i = IMAGE(clip)
```

Note: 「*」は「すべて」を、「:」は「範囲指定」を表わすワイルドカードです。ワイルドカードの詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: F > FILE_SEARCH > Supported Wildcards and Expansions

便利機能：コンテキスト・センシティブ・ヘルプ
IDL ワークベンチのエディタ、あるいはコンソール内などに不明なファンクションやプロシージャなどがあった場合は、その部分のヘルプを直接呼び出すことができます。分からない箇所をクリックし、IDLワークベンチの「ヘルプ」→「選択したアイテムのヘルプ」を選択するか、右クリック(サブ)メニュー→「選択したアイテムのヘルプ」を選択してください。F1 (Windows)、Shift+F1 (Linux とSolaris)が該当のショートカットキーになります。Macintoshでは、ショートカットが有効ではありません。

データを拡大・縮小する

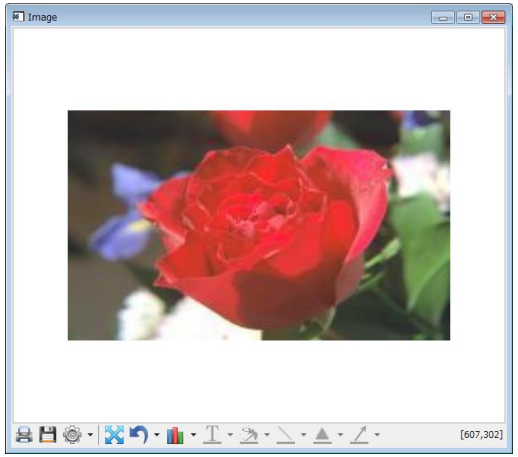
IDLには画像のサイズを変更するためルーチンが用意されています。CONGRID関数は任意のサイズに画像を変更し、REBIN関数は元の画像サイズの整数倍のサイズに変更する関数です。

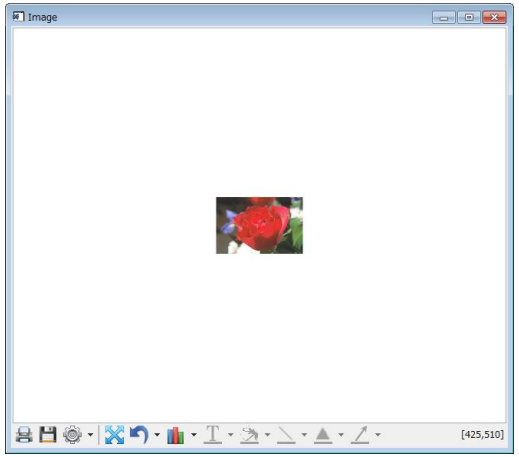
1. CONGRID関数を使用し画像サイズを500x300に拡大します。

```
IDL> zoomin = CONGRID(rose, 3, 500, 300)

IDL> zi = IMAGE(zoomin)
```

2. CONGRID関数を使用し画像サイズを約半分に縮小します。

```
IDL> zoomout = CONGRID(rose, 3, 113, 74)

IDL> zo = IMAGE(zoomout)
```





データを回転する

ROTATE関数を使用することでデータを回転したり、転置することができます。ROTATE関数は2次元配列データのみ使用可能ですので、ここでは画像データのR（赤）のデータのみを使用します。

1. red変数にR（赤）のデータを抜き出します。

```
IDL> red = rose[0, *, *]
IDL> HELP, red
RED             BYTE      = Array[1, 227, 149]
```

2. REFORM関数を使用しred変数を2次元配列に変換します。

```
IDL> red = REFORM(red)
IDL> HELP, red
RED             BYTE      = Array[227, 149]
```

Note: REFORM関数は、配列の全体のサイズは変えずに次元を変更するための関数です。この例題コードのように、サイズが1の配列の次元を削除するために使用することができます。
IDL > Routines (alphabetical) > Routines: R > REFORM

3. ROTATE関数でデータを上下反転し表示します。

```
IDL> rt = ROTATE(red, 7)
IDL> i = IMAGE(rt)
```

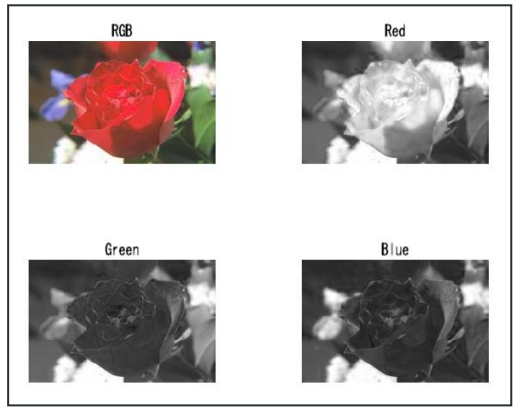
Note: ROTATE関数はデータそのものを並び替えるルーチンです。第二引数の設定値7は、上下反転を指定します。設定値は0~7まであります。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: R > ROTATE

マルチプロットする

LAYOUTキーワードを使用すると、1つのウィンドウに複数のプロットをおこなうことができます。

1. ここでは、rose.jpgのカラー画像と、R、G、Bそれぞれのデータのグレースケール画像を1つのウィンドウに表示します。

```
IDL> i = IMAGE(rose, LAYOUT=[2, 2, 1], TITLE='RGB')
IDL> i = IMAGE(REFORM(rose[0, *, *]), LAYOUT=[2, 2, 2],
> TITLE='Red', /CURRENT)
IDL> i = IMAGE(REFORM(rose[1, *, *]), LAYOUT=[2, 2, 3], $
> TITLE='Green', /CURRENT)
IDL> i = IMAGE(REFORM(rose[2, *, *]), LAYOUT=[2, 2, 4], $
> TITLE='Blue', /CURRENT)
```



Note: LAYOUTキーワードは3つの要素を持つ1次元配列を指定します。LAYOUT=[列数, 行数, インデックス]のように指定します。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: I > IMAGE

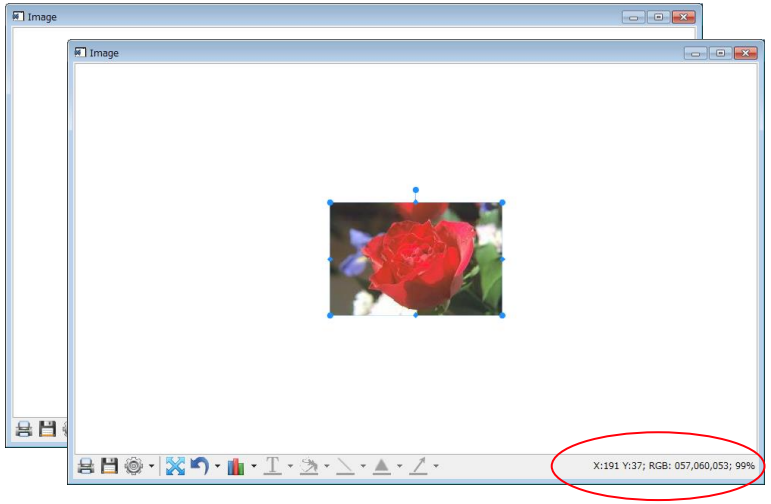
IDLワークベ便利機能：コマンド履歴
ンチのコンソールウィンドウにカーソルを置き、上下キーを入力すると、今まで入力したコマンドが表示されます。上キーでさかのぼり、下キーで戻ります。
似たコマンドを入力する場合にとても便利です。

マスクをかけて表示する

ここでは画像処理の演習として、マスクを作成し、画像をフィルタリングして表示します。バラの画像をフィルタリングし、赤い部分を抜き出し表示してみましょう。

1. まずは、画像のデータ値を見ることにより、赤く表示されている部分にどのような特徴があるのか調査します。
   ウィンドウ内の画像の部分をマウスでクリックしカーソルをあてると、ウィンドウの右下にX座標、Y座標、RGBのデータ値、縮尺が表示されます。赤の部分にはどういった特徴があるか確認します。

```
IDL> i = IMAGE(rose)
```



マスクをかけて表示する

2. この演習ではRのデータ値とGのデータ値の差分に着目してマスクを作成します。関係演算子のGE（Greater Equal）を使用し、Rのデータ値とGのデータ値の差分が40以上のピクセルを1、その他が0となるようなマスク画像を作成します。GEは条件が真なら1を偽なら0を返す演算子です。

```
IDL> red = rose[0, *, *]

IDL> green = rose[1, *, *]

IDL> msk = (FIX(red) - FIX(green)) GE 40
```

Note: red - greenの計算では、あらかじめFIX関数により変数を整数型にキャストしておきましょう。 redとgreenのデータ型はBYTE型です。BYTE型は範囲が0-255ですので負の値を格納できません。red - greenの演算を行なうと結果が負の値になる場合がありますが、BYTE型のまま計算を行うと誤った演算結果を算出してしまいます。変数を扱う上で、データ型には注意が必要です。FIX関数の詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: F > FIX

```
IDL> m = IMAGE(BYTSCL(REFORM(msk)))
```

Note: BYTSCL関数はデータをストレッチングするためのルーチンです。上の例題コードでは、0と1の値を持つmsk変数を0-255の範囲にストレッチしているため、白黒の画像が表示されます。BYTSCL関数の詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: B > BYTSCL

マスクをかけて表示する

3. バラの画像のRGBそれぞれのデータにマスクを適用し、フィルタリングします。

```
IDL> img = BYTARR(3, 227, 149)
IDL> img[0, *, *]= rose[0, *, *] * msk
IDL> img[1, *, *]= rose[1, *, *] * msk
IDL> img[2, *, *]= rose[2, *, *] * msk
IDL> i = IMAGE(img)
```

演習2
標高データの3D表示と等高線表示

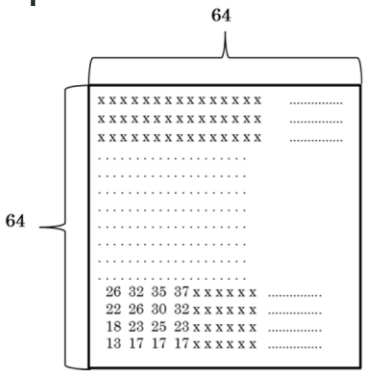
ラスタ形式の標高データをいくつかの方法で表示します

はじめに

この章では、ラスタ形式の標高データを3次元で表現したり、等高線を表示する演習を行ないます。また、等高線にカラーを付与し、カラーバー表示することで、より見やすい表示を行います。

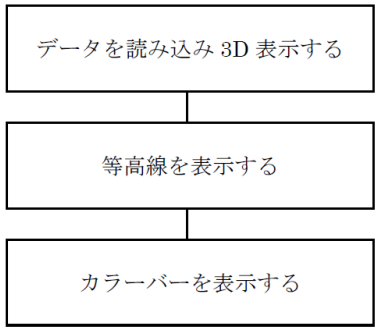
【使用するデータ：標高データ（サイズ64 x 64）】

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥elevbin.dat



elevbin.datのイメージ図

■演習内処理フロー



データを読み込み3D表示する

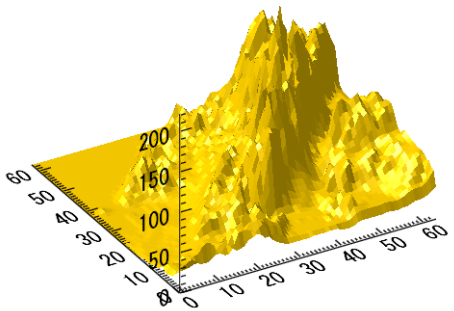
この演習では3D表示および操作する方法を説明します。

1. READ_BINARY関数を使用しデータを読み込みます。バイナリファイルはファイル自体にデータタイプや縦横のサイズなどの読み込みのための必須の情報を持っていません。ユーザ自身がこれらの情報をあらかじめ知っておく必要があります。ここで読み込むelevbin.datはバイト型の64 x 64のデータです。READ_BINARY関数はDATA_TYPEキーワードを指定しないとデフォルトでバイト型で読み込みます。

```
IDL> elev = READ_BINARY('elevbin.dat', DATA_DIMS=[64,64])
```

2. elev変数をSURFACE関数を使用し、表示します。

```
IDL> s = SURFACE(elev)
```



3. マウス操作で画像を回転してみましょう。表示されたオブジェクトをクリックすると、マウスカーソルが のように円を描いたものになりますので、ドラッグしサーフェスを回転します。サーフェスは360度回転します。

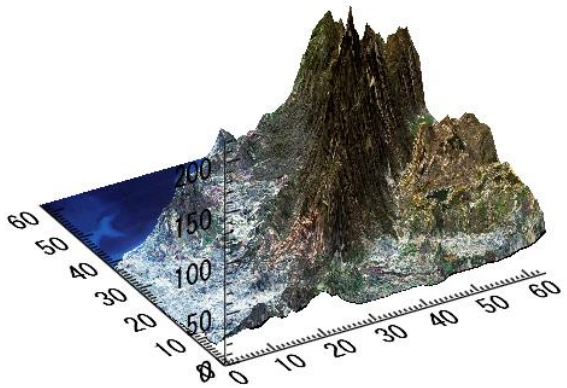
データを読み込み3D表示する

4. サーフェスの3次元データの上へ地表の画像を重ねて表示します。画像データをimage変数へ読み込み、TEXTURE_IMAGEキーワードで読み込んだデータを設定します。

【重ねるデータ：地表イメージ】

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥elev_t.jpg

```
IDL> READ_JPEG, 'elev_t.jpg', image
IDL> s = SURFACE(elev, TEXTURE_IMAGE=image)
```

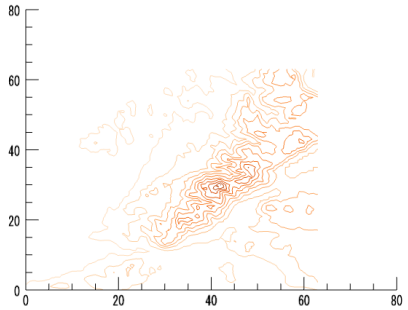


等高線を表示する

この演習では等高線を表示します。

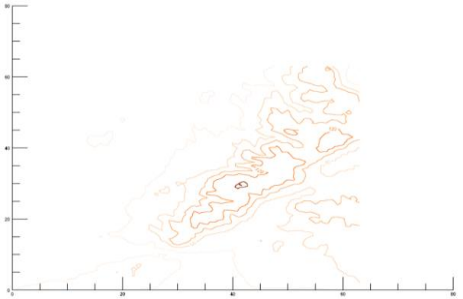
1. CONTOUR関数でコンタを表示します。C_LABEL_SHOWキーワードは等高線のラベルの表示/非表示を設定します。

```
IDL> c = CONTOUR(elev, C_LABEL_SHOW=0)
```



2. 複数の等高線のカスタムコンタプロットを作成します。C_VALUEキーワードは指定した値の等高線を表示します。

```
IDL> c = CONTOUR(elev, C_VALUE=[30,60,90,120,200], $
> C_LABEL_SHOW=[0,1], FONT_SIZE=6)
```

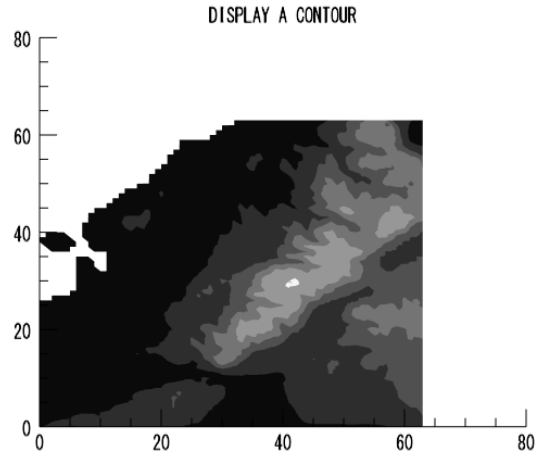


Note: C_LABEL_SHOWキーワードはどの等高線のラベルを表示するキーワードです。スカラ型で1を設定するとすべての等高線のラベルを表示します。等高線の数と同数の要素を持ち、値に0か1を設定できる配列を指定することで、どの等高線にラベルを表示するかを指定出来ます。（0なら非表示、1なら表示）
等高線の数より配列が小さい場合は、配列の設定を繰り返します。詳細はIDLヘルプの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: C > CONTOUR

等高線を表示する

3. RGB_TABLEキーワードで等高線にカラーを付与することができます。また、FILL キーワードを使用して色を塗りつぶします。この2つのキーワードで、各レベルを別々の色（もしくはグレーの陰影）で塗り潰したコンタプロットを作成することができます。POSITIONキーワードではウィンドウ内の表示位置を指定でき、TITLEキーワードではタイトルを表示できます。

```
IDL> c = CONTOUR(elev, RGB_TABLE=0, C_VALUE=[0,30,60,90,120,200],$
> /FILL, POSITION=[0.1, 0.1, 0.8, 0.8], TITLE='DISPLAY A CONTOUR')
```

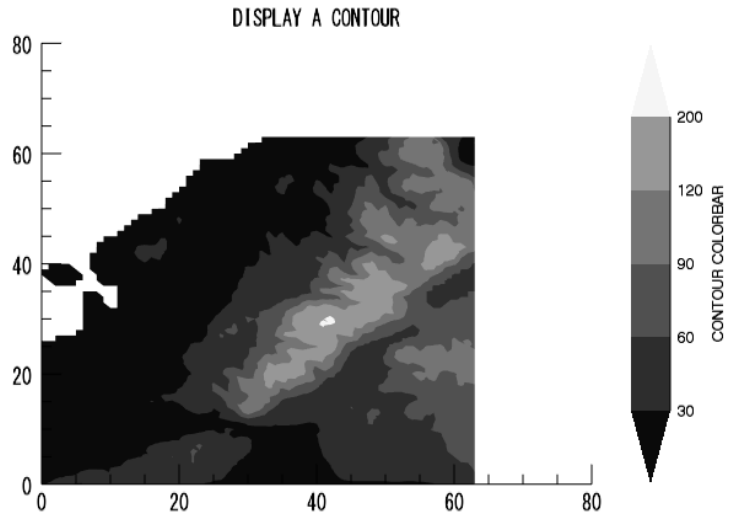


Note: 等高線の間隔を明確に指定する場合は、C_VALUEキーワードを使用してください。
N_LEVELSキーワードを使用し、任意の数だけ等高線を描画することも可能です。N_LEVELSキーワードでの等高線の間隔は自動で割り振られます。
POSITIONキーワードは4つの要素を持つ1次元配列を指定します。デフォルトでは値は0から1の範囲をとります。[左下のX座標、左下のY座標、右上のX座標、右上のY座標]を表します。詳細はIDLヘルプの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: C > CONTOUR

等高線を表示する

4. カラーバーを表示します。

```
IDL> cb = COLORBAR(TARGET=c, ORIENTATION=1, TAPER=1, $
> POSITION=[0.85, 0.1, 0.9, 0.8], TEXTPOS=1, TITLE='CONTOUR COLORBAR')
```



Note: IDLはグレースケール以外にも多数のカラーテーブルを搭載しています。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Graphics > Graphics Gallery > Modifying Visualizations > Fonts and Colors > Loading Default Color Tables

演習3
信号処理

IDL のデジタル信号処理について学習します

はじめに

この章ではIDL でのデジタル信号のノイズ除去、フィルタリングについて説明します。

演習内でノイズの追加されたデータセットを作成し、高周波ノイズを含む減衰正弦波データから成る既存のデータセットを使用し、ウェーブレットフィルタリングを行います。

使用するほとんどのプロシージャと関数はたいてい2次元以上で使用されますが、今回の例ではわかりやすく1 次元信号のみを使用して説明します。複数のIDL コマンドを使用するだけで、複雑で強力な信号処理を実行することができます。

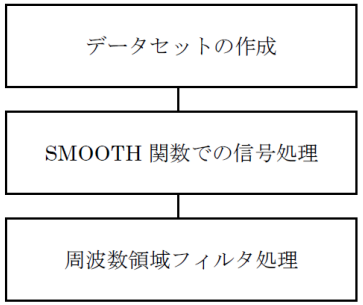
■IDLでのノイズ除去

物理過程から得られるすべての信号には不必要な周波数成分(ノイズ) も含まれています。ノイズによっては、スムージングや周波数領域内でノイズをマスクすることで簡単に除去することができます。IDLには、SMOOTH関数などいくつかの基本的なデジタルフィルタルーチンや、ウェーブレットフィルタリングを行うためのWV_DENOISE関数のような複雑なデジタルフィルタルーチンなど様々なルーチンが用意されています。

また、時間的に離散した、一連の実数値を調べてデジタル信号に含まれる情報の意味を理解するため、IDLでは信号解析変換を行うためのいくつかの変換ルーチンも用意しています。この演習ではローパス、ハイパスフィルタリングでフーリエ変換のためのFFT関数を使用します。

データセットの作成

■演習内処理フロー



■データセットの作成

演習で使用するデータセットを作成します。ノイズを追加して、現実のデータにより近づけるようにします。作成したデータセットとノイズを追加したデータセットをプロットした後、2つのデータを重ねてプロットし、違いを表示します。

1. 以下のコマンドを入力して時間と共に増加する周波数をもった正弦波関数を作成し、DATAという変数に格納します。

```
IDL> data = SIN((FINDGEN(200)/35)^2.5)
```

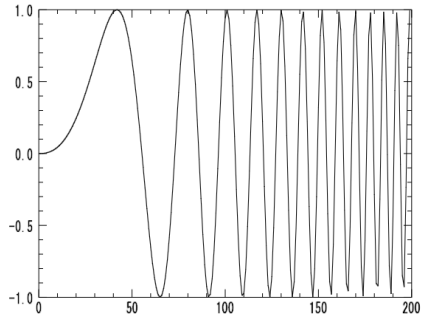
FINDGEN 関数は、それぞれの添え字の値を各要素に持つ浮動小数点型配列を返します。 それらの添え字は正弦波のベースとなる「時間」値を増加させます。 35 で割って2.5 乗した各「時間」値の正弦波関数は、DATA 変数の要素に格納されます。

Note: SIN関数は引数（ラジアン単位）の正弦値を返します。IDLには他にも多数の数学のルーチンが用意されています。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Mathematics

データセットの作成

2. このデータセットをプロットするには、以下を入力します。

```
IDL> p = PLOT(data)
```



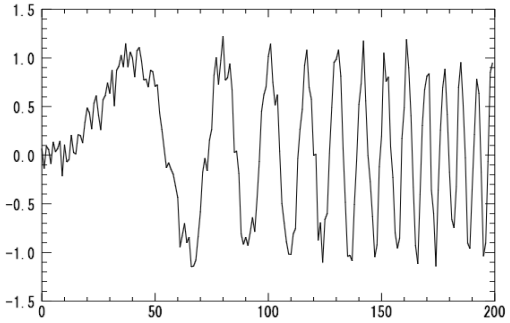
3. 一様に分布されたランダムなノイズをこのデータセットに追加し新しい変数に格納します。

```
IDL> noisy = data + ((RANDOMU(SEED,200)-0.5)/ 2)
```

RANDOMU 関数は一様に分布されたランダムな値の配列を作成します。ノイズが付加されているオリジナルのデータセットはnoisyという新しい変数に格納されます。このデータセットをプロットすると、実際のテストデータのように見えます。

4. 配列をプロットします。

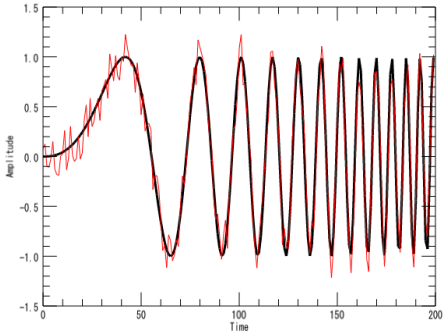
```
IDL> p = PLOT(noisy)
```



データセットの作成

5. 以下のコマンドを入力し、オリジナルのデータセットとノイズが含まれているものを、OVERPLOTキーワードを使用し、重ねて表示します。また、見やすくするためにノイズデータを赤く表示します。

```
IDL> p = PLOT(data, XTITLE='Time', YTITLE='Amplitude', THICK=3)
IDL> p = PLOT(noisy, COLOR='red', /OVERPLOT)
```



XTITLE とYTITLE キーワードを使用して、X とY 軸のタイトルを作成することができます。COLORキーワードでラインの色を、THICK キーワードで、太さを変更することができます。これによりデータの間の区別が行えます。

Note: PLOT関数はデータを可視化するための基本的なルーチンです。様々なキーワードやプロパティを持っており、多彩な描画が可能です。また、他の可視化ルーチンと同様にTITLEキーワードでプロットのタイトルや軸のタイトルも表示できます。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: P > PLOT

SMOOTH 関数での信号処理

IDL のSMOOTH 関数を使用すると、noisyデータセットのノイズを簡単に除去することができます。

指定された幅の移動(boxcar) 平均値によって平滑化された配列を返します。SMOOTH関数の詳細はIDLヘルプシステムのSMOOTHを参照して下さい。

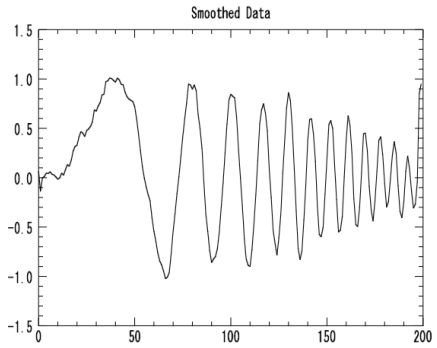
1. 以下のコマンドを入力し、ノイズ除去されたデータセットを格納する新しい変数を作成します。

```
IDL> smoothed = SMOOTH(noisy, 5)
```

2. 新しいデータセットをプロットします。

```
IDL> s = PLOT(smoothed, TITLE='Smoothed Data')
```

TITLE キーワードはプロット上の中心にタイトルテキストを入力します。SMOOTH がノイズを除去すると、周波数が増加する毎に振幅が減少していくことを確認してください。



周波数領域フィルタ処理

周波数領域フィルタ処理は、ノイズを除去するもう１つの方法です。ノイズはサンプルデータ内の不要な高周波要素です。ノイズを含むデータにローパス・フィルタ（低域通過フィルタ）を適用することによって、円滑化もしくは軽減された高周波が残ります。 以下のコマンドを順次入力しフィルタ機能を構築します。

1. 各要素をその添え字の値に設定するFINDGEN関数を使用して浮動小数点配列を作成し、それを変数Y に格納します。

```
IDL> y = [FINDGEN(100),FINDGEN(100)-100]
```

2. 次に、最初の99 要素の反転画像をY の最後の99 要素として作成します。

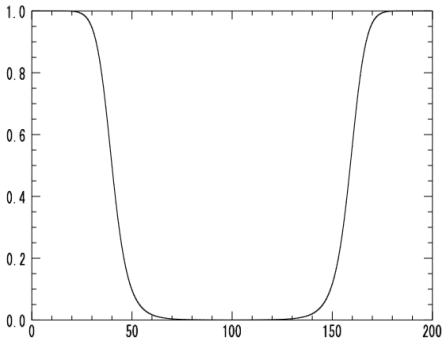
```
IDL> y[101:199] = REVERSE(y[0:98])
```

3. Y に基づいたフィルタ関数を格納するfilter 変数を作成します。

```
IDL> filter = 1.0/(1+(y/40)^10)
```

4. 最後にプロットします。

```
IDL> f = PLOT(filter)
```



周波数領域フィルタ処理

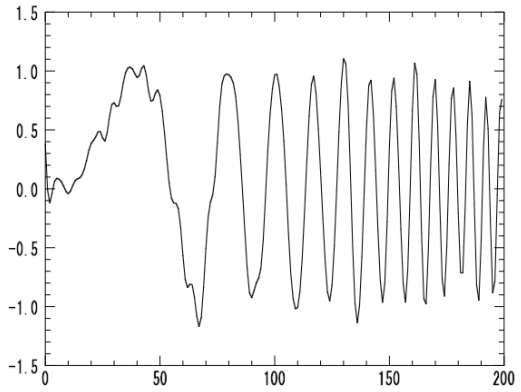
次のステップでは、フィルタをnoisyデータに適用します。周波数領域のデータにフィルタをかけるために、フィルタの周波数特性にデータの高速フーリエ変換(FFT) 成分を掛け算します。その後逆フーリエ変換を適用し空間領域にデータを返します。フーリエ変換にはFFT関数を使用します。

5. ここで、noisyデータセットにローパスフィルタを使用することができます。 以下を入力し、変数lowpass にフィルタをかけたデータを格納します。

```
IDL> lowpass = FFT(FFT(noisy,1)*filter,-1)
```

6. フィルタ処理したデータをプロットします。乱数発生により表示されるプロットは若干異なります。

```
IDL> i = PLOT(lowpass)
```



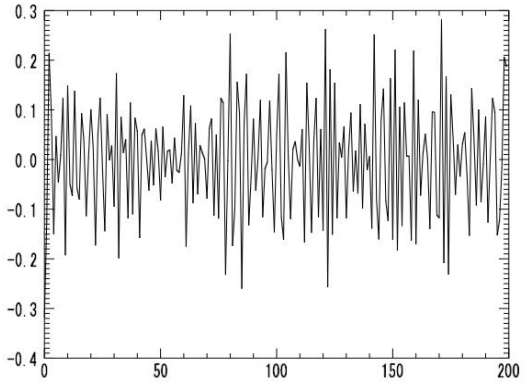
周波数領域フィルタ処理

7. 同じフィルタ機能をハイパスフィルタ（高周波もしくはノイズ要素のみ通過可能）として使用することもできます。以下を入力してください。

```
IDL> highpass = FFT(FFT(noisy,1)*(1.0-filter),-1)
```

8. Y軸の範囲を設定し、結果をプロットします。

```
IDL> h = PLOT(highpass, YRANGE=[-0.4,0.3])
```



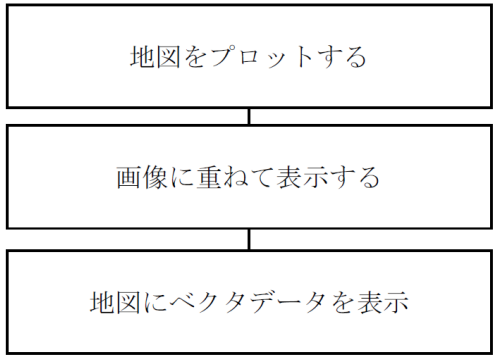
演習4
マッピング

IDLの持つマップ情報を利用し、画像データやベクタデータを
地図上に重ねて表示する方法などについて学習します

はじめに

IDL のマッピング機能により、様々な投影図法でデータをプロットすることができます。 この章では、地図の表示方法や、地図画像や大陸、ベクタデータを重ねて表示する演習を行います。

■演習内処理フロー

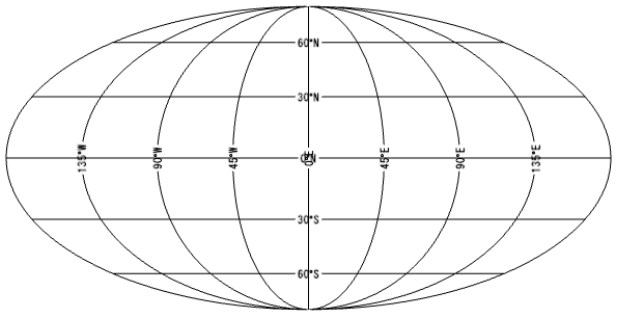


地図をプロットする

世界地図や大陸を表示します。

1. MAP関数を使用し地図を表示します。第一引数には投影法の情報を設定します。

```
IDL> m = MAP('Mollweide')
```



Note: MAP関数で設定できる投影法に関しては、IDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: M > MAP
また、マッピングのルーチンに関しては、IDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Mapping

地図に画像を重ねて表示する

IMAGE関数を使用すると地図に画像を重ねて表示できます。

【使用するデータ：世界の大陸のイメージ画像】

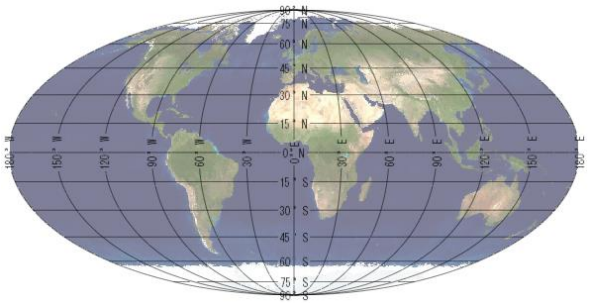
C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥Day.jpg

1. READ_JPEGプロシージャでJPEG画像を読み込み、表示します。

```
IDL> READ_JPEG, 'Day.jpg', day
IDL> dayimg = IMAGE(day)
```

2. IMAGE関数の地図投影法のMAP_PROJECTIONキーワードを使用し画像を地図に重ねて表示します。GRID_UNITSキーワードでグリッドの単位を指定します。ここでは緯度経度単位を使用します。IMAGE_LOCATIONキーワードで画像の左下に対応する地図座標を指定し、IMAGE_DIMENSIONSキーワードで画像を描画する範囲を指定します。

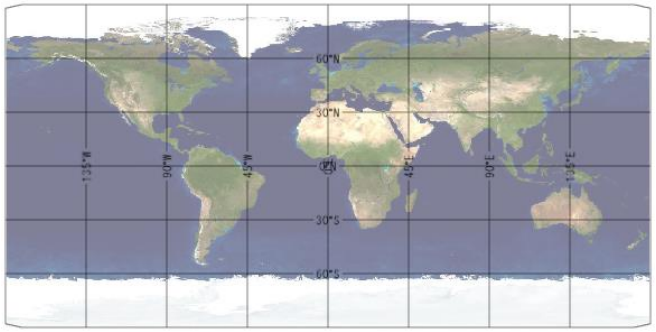
```
IDL> map = IMAGE(day, LIMIT=[-90,-180,90,180], GRID_UNITS=2,$
> IMAGE_LOCATION=[-180,-90], IMAGE_DIMENSIONS=[360,180],$
> TRANSPARENCY=50, MAP_PROJECTION='Mollweide')
```



地図に画像を重ねて表示する

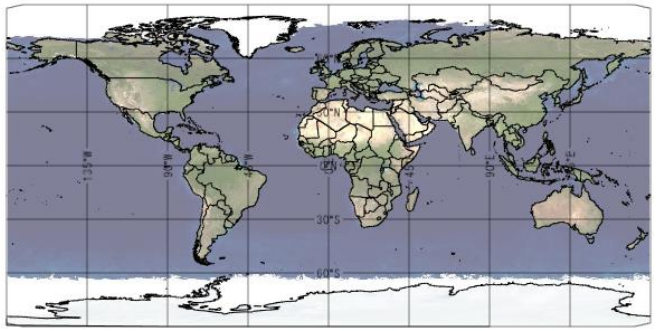
3. MAP_PROJECTIONプロパティを変更することによって簡単に違う投影法で表示することができます。ここでは手順2のコマンドの戻り値mapのMAP_PROJECTIONプロパティをEquirectangularに変更します。

```
IDL> map.MAP_PROJECTION= 'Equirectangular’
```



4. MAPCONTINENTS関数を使用し、大陸を重ねて表示します。

```
IDL> cont = MAPCONTINENTS(/COUNTRIES)
```



地図にベクタデータを重ねて表示する

前項で表示した地図上にベクタデータを重ねて表示します。

【使用データ：風速・風向のベクタデータ

（ただし、事前にSAVEプロシージャでデータをパッケージ化したもの）】

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥globalwinds.dat

1. 表示するベクタデータを読み込みます。globalwinds.datは事前にSAVEプロシージャでデータがパッケージ化されているデータであるため、RESTOREプロシージャを使用することで、読み込むことができます。

```
IDL> RESTORE, 'globalwinds.dat'
```

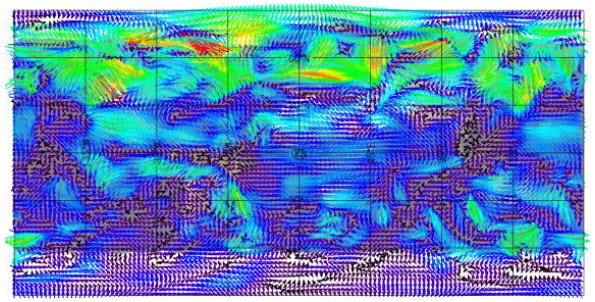
Note: SAVEプロシージャでのデータのパッケージ化に関しての詳細は、IDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: S > SAVE

地図にベクタデータを重ねて表示する

2. VECTOR関数を使用し、風のベクタデータを前項で表示したEquirectangular図法の画像に重ねます。

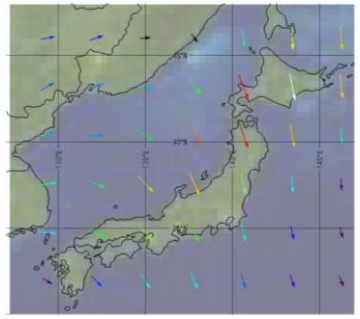
```
IDL> vect = VECTOR(u, v, x, y, /OVERPLOT, RGB_TABLE=39,$
> AUTO_COLOR=1, HEAD_ANGLE=15, HEAD_SIZE=0.8)
```

Note: VECTOR関数の引数には（u）ベクトルの水平成分、（v）垂直成分、（x）経度、（y）緯度を設定します。AUTO_COLORを1に設定することで、ベクトルの大きさを元にカラーを付与しています。VECTOR関数の詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: V > VECTOR



3. このままだと見難いため、日本付近のみを表示します。LIMITプロパティを変更することで表示範囲を変更できます。

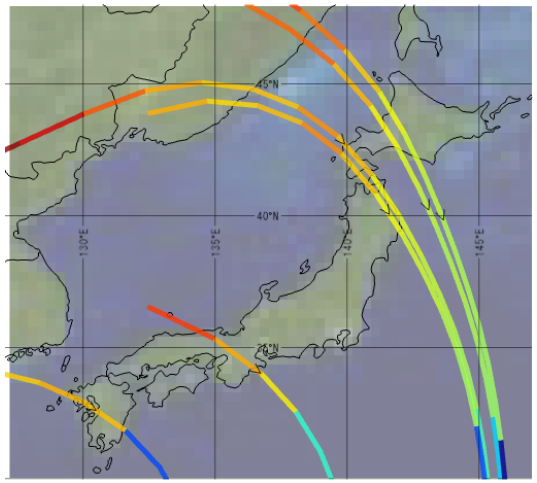
```
IDL> map.limit=[30, 127, 48, 147]
```



地図にベクタデータを重ねて表示する

4. 最後にSTREAMLINE関数を使用しベクタデータを流線形に重ねます。重ねる前に、DELETEメソッドでベクタ表示を削除します。

```
IDL> vect.delete

IDL> stream = STREAMLINE(u, v, x, y, /OVERPLOT, $

> TRANSPARENCY=20, RGB_TABLE=33, AUTO_COLOR=1, THICK=5)
```



演習5
IDLでのプログラミング

IDLでのプログラミングについて学習します

IDLでのプログラミングについて

IDL アプリケーションの範囲は非常に簡単なもの(例えばIDL コマンドラインに入力する短いプログラム) から非常に複雑なもの(GUI を持つ大きなプログラム) まで様々です。 1 つのデータセットを解析する小さいプログラムを作成していても、商用の大規模なアプリケーションを作成していても、IDL 言語で使用されているプログラミング概念を理解する必要があります。

IDL のプログラミングは、C、C++、またはFORTRAN などの言語に慣れている開発者にとっては親しみやすく感じられると思います。 IDL の言語は、これらの言語と同様の構文と演算方法を使用する高度なプログラム言語です。

プログラミング環境はほとんど同じで容易に移行できますが、IDL の構造とツールはプログラミングをより迅速かつ効果的に行なうことができます。 IDL が他のプログラム言語よりも優れている点の概要は、以下の通りです。

■**配列演算** – 配列を使用することで、各データ要素の演算を繰り返し実行する必要がなくなるので効率的で適切なプログラムを作成できます。

■**ダイナミックデータタイプ** – IDL ではコードの前後関係からデータタイプが決定されるので、変数を明確に入力する必要がありません。 変数の作成や変更は、同一プログラム内でも随時行なうことができます。

IDLでのプログラミングについて

■**IDL ワークベンチ開発環境** – プログラムを色つきで表示する機能、コーディングツール、自動コンパイル、およびビジュアルデバッグツールなど開発をスピードアップする双方向性を提供します。 プログラマーはテスト用にプログラムを迅速にコンパイルして実行し、エラーが発生している場所を直ちに調べることができます。

■**対話型プログラミングモード** – 対話型モードにより、コマンドラインからコマンドを実行してコード行を即座にテストすることができます。

■**グラフィカルユーザーインターフェース(GUI) ツール** – IDL にはGUI アプリケーション開発用に以下のツールが用意されています。

■**ウィジットプログラミング** – IDL のウィジットツールライブラリを使用して、ボタンやスライダなど簡単なコントロールを作成します。 ウィジットプログラミングでは、ユーザーインターフェースのデザインや機能性を完全に制御することができます。

■**組み込みのルーチン** – IDL にはグラフィカルユーザーインターフェース(GUI) のプログラミング、数値分析、およびデータ可視化用に膨大なルーチンライブラリが用意されています。

■**統合型開発** – IDL では 他の開発言語で作成された外部プログラムを呼び出したり、外部プログラムからIDL を呼び出すことができます。

IDLでのプログラミングについて

■**配布** – IDL では、ソースコードとしてまたはSAVE ファイルと呼ばれるコンパイルされたバイナリ形式のいずれかの方法でアプリケーションを配布することができます。 IDL の開発ライセンスを持っている人は誰でもIDL のソースコードを実行することができます。 同僚や顧客がIDL の開発ライセンスを所有していない場合でも、自由に使用できるIDL バーチャルマシンでコンパイルされたIDL アプリケーションの大半を実行することができます。 IDL ライセンスでのみ利用可能な機能をアプリケーションで使用している場合には、ランタイムライセンスを購入して配布するか、コンパイルされたアプリケーションコードにライセンスディレクトリを組み込むことができます。

■**他言語とのリンク** – IDL では、C言語やJAVA、Pythonなどの他言語からのIDLルーチンの呼び出しや、他言語のルーチンをIDLのルーチンから呼び出すことができます。

Note:他言語とのリンクの詳細に関してはIDLヘルプシステムの以下の項目を参照してください。
IDL > IDL Bridges

IDLのステートメント

プログラム言語でのステートメントとは、命令語や命令文などを示します。変数の宣言や、ルーチンの呼び出し、分岐やループなどの制御文、演算子を使用した演算などもすべてステートメントと呼びます。ここではIDLの分岐文やループ文、演算子について説明します。

■分岐文とループ文

IDLには、CやJAVAなどの一般的なプログラミング言語と同様にループ文や分岐文の構文があります。

●IF...THEN...ELSEステートメント

ここでは一般的な分岐文のIF...THEN...ELSEステートメントを説明します。IFに続く条件ステートメントの結果が真ならTHENに、偽ならELSEに続くステートメントを実行します。

```
IDL> a = 4
IDL> IF a EQ 4 THEN PRINT, 'a is 4' ELSE PRINT, 'a is not 4'
a is 4
IDL> a = 5
IDL> IF a EQ 4 THEN PRINT, 'a is 4' ELSE PRINT, 'a is not 4'
a is not 4
```

IDLのステートメント

■分岐文とループ文

●FORステートメント

ここでは一般的なループ文のFORステートメントを説明します。FORに続くステートメントにループカウンタを設定します。初期値と最大値とインクリメント値を設定できます。ループ回数回、DOに続くステートメントを実行します。

```
IDL> FOR count = 1, 5 DO PRINT, 'count =', count
count =       1
count =       2
count =       3
count =       4
count =       5
```

●複合ステートメント

分岐文やループ文の実行ステートメントが複数のステートメント（複数行）からなる場合、BEGIN...END（ENDIF、ENDELSE、ENDFORなど）の複合ステートメントが必要になります。ただし、コマンドラインからの入力はできず、プログラミング内でのみの使用となります。

Note:その他の分岐文とループ文に関してはIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (by topic) > Statements

IDLのステートメント

■条件文で使用する演算子

条件文で使用する比較演算子を以下の表にまとめます。

演算子	説明	例
EQ	～に等しい	`IDL> print, 5 eq 2, 2.0 eq 2` `0 1`
NE	～に等しくない	
LE	～以下	
LT	～未満	
GE	～以上	
GT	～より大きい	
AND	ビット積	`IDL> print, 5 and 6` `4`
OR	ビット和	
&&	論理積	
||	論理和	

Note: 演算子のより詳細な説明はIDLヘルプシステムの以下の項目を参照してください。
IDL > IDL Concepts > IDL Programming > Operators

IDLのプログラミングタイプ

IDLにはメインプログラム、プロシージャ、関数の、3つのプログラミングタイプがあります。

■メインプログラム

いくつかのコマンドを別のファイルを作成しないで実行する場合は、通常メインプログラムをIDLコマンドラインで作成します。メインプログラムは、プロシージャ(PRO) や関数(FUNCTION) のように、先頭部分で明確に指定されません。 メインプログラムでは、プロシージャや関数と同様、END ステートメントが必要になります。

ファイルを保存して実行する場合は、コンパイルをかけてから実行（もしくは.GOコマンドを実行）してください。メインプログラムは、他のルーチンから呼び出すことも引数を受け取ることもできません。 メインプログラムは.RUN コマンドを使用してコマンドラインから実行するか、ファイルに保存して実行することができます。

例：メインプログラム（ファイルに保存する方法（sin_plot.pro））

```
x = FINDGEN(100)
y = SIN(x)/10
p1 = PLOT(x, y)
END
```

コマンドラインから保存したプログラムを実行する場合は、以下のようなコマンドとなります。

```
IDL> .RUN sin_plot
```

例：メインプログラム（.RUNを使用してコマンドラインから実行する方法）

```
IDL> .RUN
- x = FINDGEN(100)
- y = SIN(x)/10
- p1 = PLOT(x, y)
- END
```

IDLのプログラミングタイプ

■ 名前付きプログラム：プロシージャと関数

プロシージャと関数は共にモジュラープログラムで、個別に実行したり、他のプログラムから呼び出すことができます。 プログラムには複数のプロシージャや関数を含めることができ、他のプログラムも必要なだけ呼び出すことができます。 開発者は多くのプロシージャや関数を個別に保存するか、これらを同一のファイルにまとめるかを選択することができます。

●プロシージャ

プロシージャは明確に定義されたタスクを実行する自己完結的な一連のIDL ステートメントです。 プロシージャは、プロシージャ定義ステートメント（PRO < プロシージャ名>）で定義されます。ここでのプロシージャ名は作成するIDL ステートメントの名前となります。 パラメータは、プロシージャで使用されている名前付き変数です。

例：プロシージャ（TEST.PRO）

```
PRO TEST
   x = FINDGEN(100)
   y = SIN(x)/10
   p1 = PLOT(x, y)
END
```

実際に使用する場合は、以下のようなコマンドとなります。

```
IDL> TEST
```

Note: IDL で作成されたプログラム例の参考として、IDL インストールディレクトリ配下の/examples/demo/demosrc にあるIDL付属のデモプログラムが利用可能です。これらのプログラムをIDLワークベンチへ表示し、プロシージャや関数を使用したプログラムを参照していただけます。ただし、これらのプログラムに変更や保存はしないでください。

IDLのプログラミングタイプ

NV5 GEOSPATIAL

●関数

関数は明確に定義されたタスクを実行する自己完結的な一連のIDL のステートメントで、実行されると呼び出しプログラムユニットに値を返します。 全ての関数は、関数を終了するのに使用されるRETURN ステートメントのパラメータとして与えられている関数の値を返します。 関数は、関数定義ステートメント(FUNCTION <関数名>) で定義されます。関数名は作成するIDL ステートメントの名前となります。デフォルトでは、関数は新しい変数を作成するので、リターン値へのアクセスを簡単にする必要がある場合には関数を使用してください。

例:関数(TEST.PRO)

```
FUNCTION TEST
   x = FINDGEN(100)
   y = SIN(x)/10
   data = y#y
   RETURN, data
END
```

実際に使用する場合は、以下のようなコマンドとなります。

```
IDL> data = TEST()
```

Note: 関数はIDLワークベンチの実行アイコンから直接実行することができません。関数を呼び出すには上記のように名前を指定して呼び出す必要があります。

引数とキーワード

プロシージャと関数には引数とキーワードを定義することができます。

キーワードは省略可能なパラメータですので、プログラム内部で制御する必要があります。

●プロシージャ

プロシージャで引数やキーワードを定義するには、プロシージャ名の後にカンマ区切りでパラメータを並べてください。

例：プロシージャ（TEST1.PRO）

```
PRO TEST1, prm1, prm2, Key=prm3
    PRINT, 'prm1 = ', prm1
    PRINT, 'prm2 = ', prm2
    IF KEYWORD_SET(prm3) THEN PRINT, 'prm3 = ', prm3
END
```

実際に使用する場合は、以下のようなコマンドとなります。

```
IDL> TEST1, 1, 2, Key=3
prm1 =        1
prm2 =        2
prm3 =        3
```

Note: KEYWORD_SET関数はキーワードがセットされているかどうかをチェックするための関数です。セットされていると1、セットされていないと0を返します。詳細はIDLヘルプシステムの以下の項目を参照してください。
IDL > Routines (alphabetical) > Routines: K > KEYWORD_SET

引数とキーワード

●関数

プロシージャで引数やキーワードを定義するには、プロシージャ名の後にカンマ区切りでパラメータを並べてください。

例：プロシージャ（TEST2.PRO）

```
FUNCTION TEST2, prm1, prm2, Key=prm3
   PRINT, 'prm1 = ', prm1
   IF KEYWORD_SET(prm3) THEN PRINT, 'prm3 = ', prm3
   RETURN, prm2 + 1
END
```

実際に使用する場合は、以下のようなコマンドとなります。

```
IDL> ret = TEST2(1, 2, Key=3)
prm1 =         1
prm3 =         3
IDL> PRINT, ret
       3
```

IDLのプログラムとコンパイル

NV5 GEOSPATIAL

■IDLのプログラムファイル

IDLのプログラムファイルは拡張子「.pro」のファイルです。ファイル名はファイル内に記述したプロシージャまたは関数名となります。1つのファイル内に複数のプロシージャや関数を記述することができます。ただし、ファイル名となるメインのルーチンはファイル内の一番最後に記述してください。

■自動コンパイル

IDL では、ルーチンが既にIDL のメモリにある場合（IDLのコアライブラリ、あるいはルーチンがすでにコンパイル済みの場合）には、IDL はそれを実行します。 ルーチンがメモリにない場合には、IDL は検索パス内でルーチン(ファイル名.pro) を検索して、それを自動的にコンパイルします。 ファイル名となるメインのルーチンを最後に記述する理由は、ファイルの先頭のルーチンから順に、ファイル名と同名のルーチンが見つかるまでコンパイルを行うためです。以降のルーチンは自動コンパイルされませんのでご注意ください。

Note: IDL の検索パスはユーザがパスを追加したり編集することが可能です。詳細に関してはIDLヘルプシステムの以下の項目を参照してください。
IDL > IDL Concepts > Setting Preferences > Directory and Search Path Preferences
IDL_PATH

Note: IDL のルーチンにはすべて固有の名前があり、ユーザ作成のルーチンが同じ名前の場合には競合する場合があります。 IDL でこのような競合が発生すると、自動コンパイルのメカニズムはユーザ作成のルーチンを無視し、IDL固有のルーチンを使用します。

IDLのプログラムとコンパイル

IDL のプログラム機能を示すため、サーフェスで使用した標高データを使用しコンタカラーの標高データを表示し、カラーバーも合わせて表示します。

1. IDL ワークベンチから、ファイル→ 新規→ IDL ソースファイルを選択して新しいIDL エディタを開きます。
2. 以下のPROからのコードを新しいエディタに入力し、プログラムを作成します。以下のデータを使用します。

C:¥Program Files¥NV5¥ENVIxx¥IDLxx¥examples¥data¥elevbin.dat

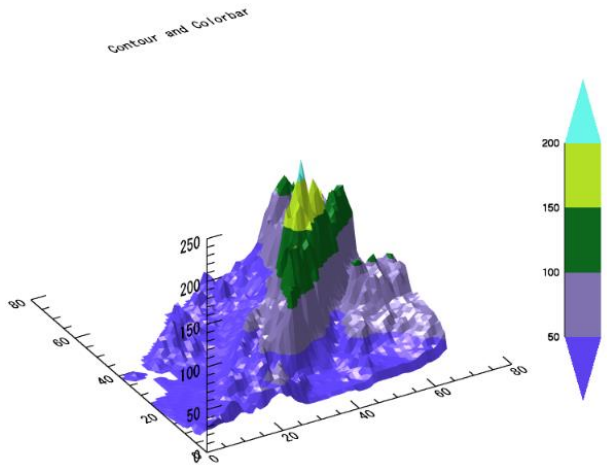
```
PRO CONTOUR_COLORBAR
        ; 変数file へファイルを設定します
        file = DIALOG_PICKFILE(FILTER='elevbin.dat')
        ;dem変数へelevbin.datをインポートします。
        dem = READ_BINARY(file, DATA_DIMS=[64,64])
        ; CONTOURプロシージャを利用し、データを表示します。
        c1 = CONTOUR(dem, RGB_TABLE=30, /FILL, PLANAR=0, $
                N_LEVELS=5, TITLE='Contour and Colorbar')
        ; 表示したデータの横にカラーバーを追加します。
        cbar = COLORBAR(TARGET=c1, ORIENTATION=1, TAPER=1,$
                POSITION = [0.90, 0.2, 0.95, 0.75])
END
```

Note: IDL コードでセミコロン(;) はコメント行が始まることを表します。 コメント行に説明を入れることによってコードがどのような処理を行うのかが分かり、コードを理解するのに役立ちます（IDL 自身はこの行を実行しません）。

コンパイルと実行

作業中のプログラムを表示するには、以下の手順に従います。

1. ファイル→ 別名保管を選択し、「contour_colorbar.pro」と入力して、ファイルをcontour_colorbar.pro として保存します。
2. ツールバーのコンパイルボタンをクリックまたは、実行→コンパイルcontour_colorbar.proを選択してコンパイルを行います(IDL パスにある場合は、自動的にコンパイルされます)。
3. ツールバーのコンパイルボタンをクリックまたは、メニューの実行→実行を選択し、プログラムを実行します。ウィンドウに結果として作成された画像が以下のように表示されます。



Note: 作成したプログラムの実行中にエラーが発生した場合、コードに誤字が無いかどうか再度確認してください。

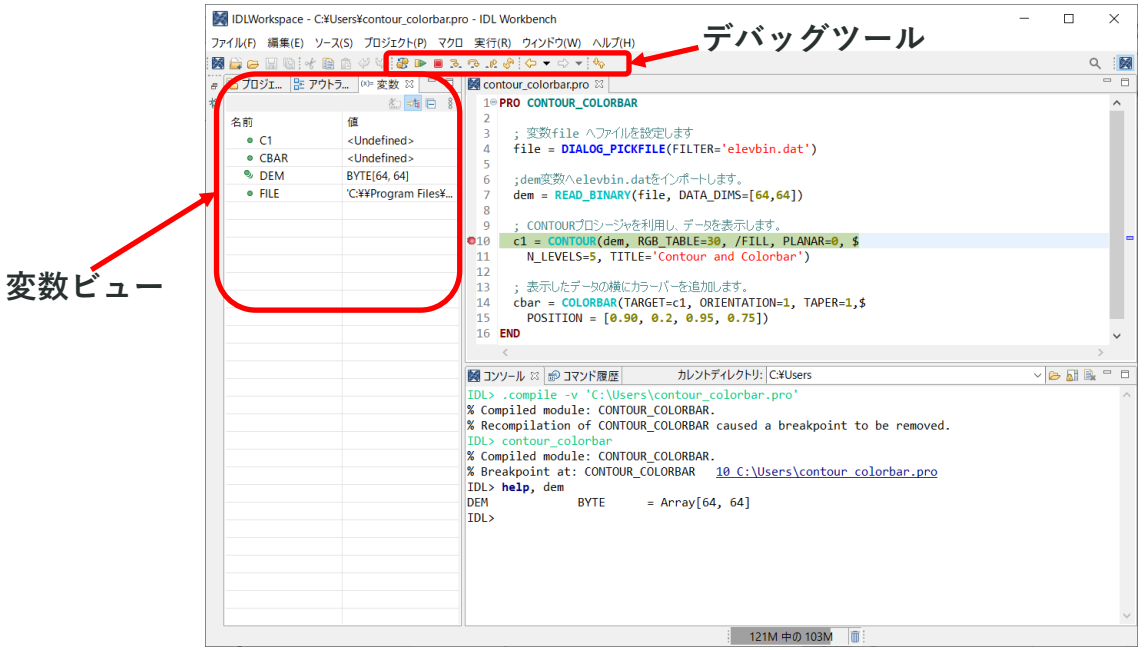
Note:プログラムを実行する度に必ずコンパイルする必要はありません。 実行ボタンをクリックすると、ファイルがコンパイルされていない場合は自動的にコンパイルされます。 コンパイルした後でソースコードを変更した場合は、プログラムをもう一度コンパイルする必要があります。

デバッグとブレークポイント

IDL ワークベンチには、コードからエラーを検索して修正するための強力なツールが装備されています。

■デバッグ

デバッグは、コードのエラーや間違った動作を検出して修正する処理です。 IDLワークベンチには、プログラムの実行の監視、実行の停止と再スタート、プログラムのステップ実行、変数値の検査と変更などのツールが用意されています。これらのツールを用いてプログラムが想定通りに動作しているか確認し、動作していない場合は問題の原因を検出することができます。



デバッグとブレークポイント

■IDLワークベンチのデバッグツール

以下にIDLワークベンチにおける基本的なデバッグツールを紹介します。

デバッグツール	機能
	テキストエディタで選択されているプログラムコードをコンパイルします。
	テキストエディタで選択されているプログラムコードを実行します。
	ブレークポイント、あるいはエラー発生で一時停止したプログラムを再開します。
	実行されているプログラムを停止します。
	プログラムをステップ実行します。ソースコードの関数やプロシージャ上でクリックするとその処理内にステップインします。
	プログラムをステップ実行します。ソースコードの関数やプロシージャ上でクリックするとその処理を実行し次の行に移動します。（関数やプロシージャを1行とみなし、その処理内には入りません）
	実行している関数やプロシージャを抜けます。（リターンします）
	IDLのセッションをリセットします。

デバッグとブレークポイント

NV5 GEOSPATIAL

■ブレークポイント

ブレークポイントとは、デバッグ作業でプログラム実行中に意図的に一時停止させるポイントです。ブレークポイントを設定することにより、デバッグ作業者は一時停止する直前の変数の値などを確認することができます。IDLワークベンチでブレークポイントを設定するには、テキストエディタのコードの行の横の左側余白をダブルクリックしてブレークポイントのオン/ オフを切り替えることができます。また、ブレークポイントを配置する行にカーソルを移動して、Ctrl-Shift-B を押すか、Run→ Toggle Breakpointを選択します。 エディタの左側の余白部に赤の点が表示されます。

```
9   ; CONTOURプロシージャを利用し、データを表示します。
10  c1 = CONTOUR(dem, RGB_TABLE=30, /FILL, PLANAR=0, $
11    N_LEVELS=5, TITLE='Contour and Colorbar')
```

■変数の値を表示する

ブレークポイントが設定されているプログラムを実行すると、設定されたブレークポイントの箇所でプログラムを一時停止します。プログラムが停止したら、変数ビューを使用するか、コマンドラインにて対象の変数のHelpやPRINTを実行、またはエディタで変数上にマウスポインタを置いて、現在の実行範囲での変数の値を調べることができます。

■一時停止中に変数を可視化する

IDLでは実行中のプログラムをブレークポイントで一時停止したときに、コマンドラインからPLOT関数やIMAGE関数などのコマンドを実行することで、変数の可視化を行うことができます。これにより、さらにデバッグの効率を上げることができます。

NV5

GEOSPATIAL