NAOJ/ADC IDL 講習会資料 (2017 Oct)

IDL 初～中級者のための天文データ解析用 IDL講座

大山陽一

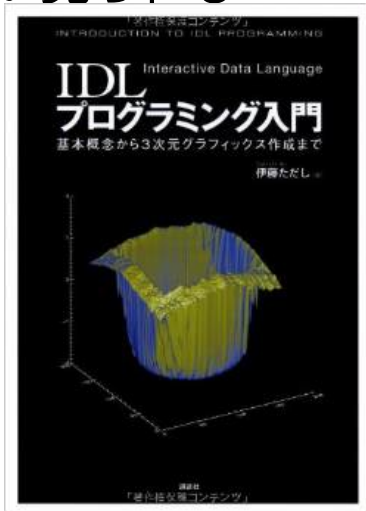
ASIAA（台湾）

重要なお断り

- IDLは、RSI(->ITT->Exelis Visual Information Solutions -> Harris Geospatial Solutions)(とその販売代理店)の商品です。大山は彼らとはいっさい関係なく、金銭その他の利便の提供も受けていません。
- IDLに義理は無いので、IDLの宣伝はしません。
  - 大山は1ユーザーとしてIDLの良さを認識して、皆さんにお勧めしますが、買ってくれなくても結構です。
- IDL のライセンス料が高いという苦情は、私は受け付けませんし、どうにもできません。
  - 最初は ADC のユーザーとして IDL を使用したら良いでしょう。
- 今後の IDL 使用環境については、ADC のスタッフと相談してください。
  - 大山は ADC の経営企画に関与していません。

なにはなくとも参考書

- Practical IDL programming
  - By L. E. Gumley (MORGAN Kaufmann 発行)
  - 超おすすめ。1教室に1冊常備。¥12133 at Amazon
  - ユーティリティーも気が利いていて、使える。
- On-line IDL references
  - http://www.harrisgeospatial.com/docs/using_idl_home.html
    - IDL Routines or IDL Language だけで良い。
  - Linux のターミナル(シェル)から 'idlhelp' でも、同じものが見られる
  - Google で検索すれば、類似品が山ほど出てくる。
    - 古いバージョンの物も多いが、Harris のページなら最新。
- Coyote's Guide to IDL programming
  - By David Fanning
  - http://www.idlcoyote.com/
  - Web 上の tips 集も面白い。
    - 中級以上？
- IDLプログラミング入門—基本概念から3次元グラフィックス作成まで— (KS理工学専門書) (伊藤 ただし)
  - まさかこんな本が日本語で発売されるとは。。。すばらしい。



講習内容の大枠

- 初級者コースの講習 -> 復活！(2017 Spring by 巻内さん)
- 資料(大作。必読！) は ADC ホームページ(トップページからすぐ分かるところ)にあります。
  - https://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html#idl_beg

- 今回の講習会:より実践的な講習
  - 天文 FITS 2次元画像解析アプリへの応用を念頭に。。。
  - 終了後すぐに、自らの課題に活かせるように。
- IDL 基礎・特徴のおさらい
- IDL の特徴を活かしたプログラミング
- デバッグの基礎
- サブルーチン化
- 実習

IDL 言語の特徴

- IDL の基礎の基礎の復習です。
- IDL の特徴が活かせるアプリケーションとは？
- キーワード
  - 「超」高級言語
  - Interactive Data Language
  - Data Visualization Tools

IDL 言語の特徴1

- インタプリタ
  - 柔軟なプログラム開発環境。
  - ただし速度は遅い。
- アレイ言語
  - IDL に読み込んでしまえば、
    - FITS 画像も jpeg もアレイ。
    - FITS table も ascii table もアレイ。
  - アレイを使うと、
    - プログラムが分かりやすい、見やすい。
    - やりたい処理をそのまま書ける。
  - アレイのサブスクリプト（インデックス）を処理する、という概念を理解すべし。
- 強力なデータ処理言語＋データ表示
  - C -> file output on disk -> gnuplot では非効率。
- Python+NumPy+Matplotlib に近いか？

IDL 言語の特徴2

- Procedure/Function は ascii file で配布。
  - コンパイル済みプログラム or 実行形式 (.exe) は、基本的に存在しない。
  - 他人のプログラムも、全て中身が読める。
    - コード内容は、全公開。ブラックボックス処理は、ない。
    - 他人のプログラムを改良・応用して活かす。
  - 欲しいプログラムは、まずネットで探す。
- すべてはオン・メモリ処理。
  - メモリ量（究極的にはメモリ速度も）がネックに。
    - メモリは可能な限り増設しましょう。
  - C.f. IRAF では file I/O 速度が (disk 容量も) ネックに。

IDL vs. IRAF

- IDL はプログラム言語
- IRAF は解析環境＋貧弱なシェル
- IDL はメモリベース
- IRAF はファイルベース
  - 結果は一旦ファイルに書き込む。
  - 次のルーチンは、ファイルを読む所から始まる。
- IRAF は良くも悪くもブラックボックス
- IDL は低レベルのルーチンまでハッキング可能
  - 新しい装置の新しい解析手法も作れる。
- IRAF は無償 (NOAO: 天文台)
- IDL は有料 (RSI->ITT->Exelis…: 営利企業)
  - ただし無償のツール・コードが出回っている。

IDL vs. C (科学技術計算の場合)

- コードの読みやすさ: IDL = 100 X C
  - C はそもそもシステム開発言語。
  - C の”ポインタ”、”*”、“&” などは、忘れましょう。
    - 忘れました。
    - Printf で悩んだ時代が懐かしい。
- デバック速度: IDL = 100 X C
  - インタープリタなので、デバッグはとても楽。
  - 試行錯誤によるコードのアップデート作業も、楽。
- 計算速度: IDL = 1/10 X C
  - IDL に速度を求めてはいけません。
  - 最終手段: 高速化するルーチンだけ C で書いて、それを IDL から呼び出す、という技はある。
    - 自分で使った事は無いが、たまに見る。
      - SDSS 分光データ処理パイプライン、とか。

これだけ知っていれば読める！ IDL 文法の特徴

- 初心者講習会に出た方は、スキップしてください。
- 巻内資料に、とても良くまとまっています。

大文字と小文字

- 全く関係無し。
  - もちろん String 以外。
  - 個人の見やすい書き方で。
    - 私は小文字派。

0 から数えよ

- 画像の左下のコーナーは、
  - IDL では [0,0]
  - IRAF では [1,1]
  - DS9 や SKYCAT は [1,1]
- For loop を使うときは、
  - 0 から n_elements(x)-1 まで。

, と ; と : と $ と &

- ‘,’ : すべての argument の区切り。
  - 一番よく使う。
  - または、array index の区切り。
- ‘;’ : コメント行。
  - 一般に、行の頭につける。
  - しかし、行の最後につけても良い。
- ‘:’ : array の index の範囲指定
  - Image[1:10,10:20] --- 大きな image array のうちの x=1~10, y=10~20 のサブ・アレイ
- ‘$’ : 複数の行を1行コマンドとして使う時のおまじない。
  - 基本は、1行1コマンド
- ‘&’ : 一行に複数のコマンドを書く時のおまじない。
  - 基本は、1行1コマンド

[ ] と ( )

- [ ]: array の座標（インデックス）
  - x[0]: x array の最初の内容
  - y[0,10]: y array の x=0, y=10 の座標の内容
  - image[*,*,1]: ２次元画像のスタック・キューブ（３次元）から、１番目の画像を抜きだす。
  - などなど。
- ( ): サブルーチン (function) の argument。
    - または、数学の計算順序を示すカッコ。
  - Y＝f(x)など。
- 非常に古いバージョンの IDL では、( ) を array の index として使っていたため、古いコードでは x(0,10) などと書いてある場合がある。
  - これでも動くが、お勧めできない。
    - 混乱の元。

&& と AND, || と OR, ~ と NOT

- 集合の論理積などを表すのが、&&, ||, ~
  - 「ベン図」を思い出そう。
  - IF 文で多用。
    - もし A かつ B であれば、これを実行、などなど
- ビット単位の論理積など表すのが、AND, OR, NOT
  - 2ビットの計算例
    - 000 AND 111 = 000
    - 000 OR 111 = 111
  - Where 文で多用。
    - A かつ B の条件を満たすインデックスを求める、などなど
- 経験的に、where 文で誤って && など使ってしまい、気づかずにデバッグで悩むことが多い。
  - 取り違えても偶然正しい挙動を示す場合もあり、気づきにくい。
  - where は特別、と割り切って覚えても、多分問題は無い。

EQ と =, GT と <

- EQ, GT などは、比較演算子。
  - IF 文などで多用。
    - IF A GT B then…
- = は代入文。
  - A=B など。(A に B の内容をコピーする)
- <, > は特殊な演算子。
  - A = (B < 5) などと使う。
    - もし B が 3 なら A=B=3
    - もし B が 10 なら A=5
  - Index の範囲指定などで使うと便利。
    - A_cut=A[b > 0:c < (x_max-1)] など。
      - 仮に b や c が A array の index の範囲外を指定したとしても、エラーを回避できる。

“ “ と ‘ ‘

- どちらも “string” の範囲指定をするもの。
  - ただし、”” は特殊な用途があるので、’ ’ がおすすめ。
    - 例えば、string が数字で始まる場合、問題が発生する。
      - IDL> print,'0a'
      0a
      - IDL> print,"0a"
      print,"0a"
            ^
      % Syntax error.
      - IDL> print,'aa'
      aa
      - IDL> print,"aa"
      aa
    - 昔これで悩んだことがある。

variable の振るまい1

- あらかじめ定義する必要はない。
  - C の様に、最初に define する必要はない。
  - インタプリタだから。
- 代入される段階で、初めて定義される。
  - Variable のタイプは、代入相手によって決まる。
  - A=1; A は integer
  - A=1.0; A は float
  - A=fltarr(10); A は float の array (size=10)
  - A= ‘1.0’ ; A は string
- Variable のタイプは、再設定できる。
  - A=1.0 の後に A= ‘1.0’ 値と違うタイプに再設定できる。
    - 結果は A は string に変わる。

variable の振るまい2

- A が「定義されているかどうか」をプログラム上で調べることができる。
  - Undefined を使った IF 条件分岐、など。
  - Procedure/function のオプションの有無の判定など。
- IDL のパラメーター割当はルーズなので、便利な反面、混乱の元。
  - 統一的な命名法や、サブルーチンを使った variable の整理、などがおすすめ。

数字（アレイ）の計算

- 基本は、まったく普通の感覚で。
  - +-*/^
  - alog10(), sqrt(), sin(radian)
  - A^2.,10^-3
  - 3*(1+2)=9
- 数字 -> 「数字のアレイ」として使っても、同じに動く
  - 上記で、A はスカラーでも良いし、ベクターでも良いし、イメージ（2次元アレイ）でも良い。
    - 例：sqrt(A) は、A の内容がそれぞれすべて平方根になる。
    - A+B は、A と B の内容がそれぞれすべて足し合わされる。
  - ただし、アレイのディメンションは合せておく事。
    - 良くあるエラーの元。

特殊な計算

- 何でもできる。
  - でも気をつけて。
  - IDL は落ちずにプログラムは続いて行く。。。
    - Print,1/0
    - 1
    - % Program caused arithmetic error: Integer divide by 0
    - Print,1*!values.f_nan,1*!values.f_infinity
    - NAN INF
    - % Program caused arithmetic error: Floating illegal operand
  - 文句は言われるが、これらはエラーストップではない。
    - Warning の扱い。
    - うまく活用しましょう。
  - 逆に、arithmetic error は頻出するので、あまり気にしすぎる必要はない。

よく使う、数学関数

- Total
- Median,mean,stddev,variance
  - Moment 関数で一発。
- Sin/cos
- Max,Min
- Finite
  - Argument が有効な数字かを調べる関数。
    - 無効な数字：無限大、無限小、Not A Number (NAN)
  - 知っていると意外と便利
    - Mask 処理に応用できる。
      - 1 or 0 mask ではなく、1 or NaN mask とする。

たまに使う、文字列関数

- Strcmp, strtrim, strlen
  - IDL の科学計算ではあまり必要ないですが、たまに file I/O などで必要な場合があります。
  - あまり強力ではないですが、一通りのことは出来ます。

Array を使いこなそう

- １次元：A=[1,2,3] --- 3 要素ベクトル
- ２次元：B=[[1,2,3],[1,2,3]] --- 3X2 array
- 範囲指定：a[1:2], b[0:1,1:2], c[*,2:5], d[2:*]
- アレイインデックスがアレイ
  - E=[1,2,3,4,5] & f=[0,1,2] & print,E[f]
  - E の 0 番目、1番目、2番目の内容が表示される。
- Help で結果を確かめよう。
  - IDL> help,a,a[1:2]
  - A　　INT　　= Array[3]
  - <Expression>　INT　　= Array[2]
- (A+B)[0:2] という技もあり。
  - C=A+B かつ C[0:2] という意味。
    - A と B のディメンションがあっていないと、そもそも足せないので注意。

Where を使いこなそう

- 「Index 使い」になろう
  - For loop で array 内容をスキャンしながら IF 文を掛けるのは、古いです。

```
For x=0,99 do begin
    For y=0,99 do begin
        If image[x,y] LT -100 then image[x,y]=!values.f_nan
    Endfor
Endfor
```

  - IDL 流は、上記と同じことを次の2行で。

```
bad_index=where(image LT -100)
Image[bad_index]=!values.f_nan
```

    - 「ベン図」をイメージしながら、where の中に条件を書き込むだけ。
    - Where の良さが分かると、止められません。

最低限知っておきたい IDL 内部コマンド、関数

- アレーを作る
  - FItarr, intarr, …
  - Findgen, indgen, …
- アレーを調べる
  - Where
  - Size, n_elements
  - help
- 結果を表示する
  - print
  - Plot (oplot)
  - (Atv <- これだけ外部)
- Procedure/function のキーワードを調べる
  - Keyword_set
- デバッグ
  - Message/stop
  - .reset

知っておきたい外部function/procedure

- IDLASTRO (http://idlastro.gsfc.nasa.gov)
  - 天文向け IDL 外部ルーチン総本家。検索やリンクもある。
  - 各種そろっているが、以下はマスト。
    - FITS read/write; ASCII table read/write
- Others
  - Atv: 2次元画像ビュワー or ds9 for IDL.
    - http://www.physics.uci.edu/~barth/atv/ by Aaron Barth
    - ちょっと機能が足りないが、ないと大変困る。
    - 各方面で改造されて、応用されている。
  - Coyote (graphics) library
    - デフォルトのグラフィックライブラリに満足できない方は、ぜひ。
    - 前に紹介した Coyote IDL page (http://www.idlcoyote.com/) から。
- ネットの向こうにいる開発者に、感謝。

最低限知っておきたい IDL 言語の制御構造

- 条件分岐
  - IF 文
  - Case 文
- ループ
  - For 文
  - Break, continue
- ジャンプ
  - Goto 文

- どれも簡単なので、参考書を見てください。

FORループ, IF条件分岐について、もう少し。

```
For x=0,xmax-1 do begin
    y=func(x)
Endfor
```

- or

```
For x=0,xmax-1 do y=func(x)
```

```
If A GT B then begin
    a=gunc(b)
Endif
```

- or

```
If A GT B then a=gunc(b)
```

- 一行ですむ場合は、if/for と同じ行に続けて書きます。
- 実行内容が複数行にまたがる場合は、begin… end(for/if) でまとめます。

- If then begin … endif else begin … endelse もよく使います。

IDL プログラミングを始める前に知ってほしい事柄

- これから本格的に IDL をいじりたい方が、その環境を準備するときに役に立つと思われる、ちょっとした情報。
- 自分の研究室に戻って設定するときに、役立ててください。

開発環境

- Idlde
  - IDL 開発会社が作った開発環境 (developing environment)
  - 良くできていて、愛用者もいるが、大山は好きでない。
    - 私の学生さんが以前使ってました。ただし、やめたらしい。
    - 開発者支援機能がいろいろついていて便利、らしい。
- Idl コマンドライン on ターミナル + 汎用エディタ
  - 簡単・簡便・必要十分
  - エディタは Emacs + IDLWAVE 環境がおすすめ。
    - IDLWAVE: https://www.gnu.org/software/emacs/manual/html_mono/idlwave.html
      - IDLWAVE is an add-on mode for GNU Emacs and XEmacs which enables feature-rich development and interaction with IDL…
    - 構文の色付け、1行ヘルプ、などなど機能多数。
  - Vi でも gedit (Linux 端末 @ ADC のデフォルト) でも、もちろん OK.
  - お好きな組み合わせで、どうぞ。

IDL 起動前の設定と、起動後の確認

- IDL のインストール: プロに任せる。
- .cshrc ((t)cshell の場合) にて、IDL_PATH の設定
  - 自分の IDL ライブラリ path を追加せよ。
  - Source .cshrc を忘れずに。
- Terminal から‘Idl’or‘idlde’で起動。
  - 例:idl の場合は、コマンドラインで idl と実行すると、IDL プロンプトが出る。
    - castor:/asiaa/home/ohyama% idl
    - IDL Version 8.3 (linux x86_64 m64).
    - …
    - IDL>
  - これで、linux 環境から IDL 環境に入った。
- “Window”して、graphic 画面が出れば OK.
  - エラーが出たら、X11 の設定に問題がある。
    - プロに相談せよ。
- bash など他のシェルの方は、適当に翻訳してください。

マニュアルはこれだけ
オンライン・ヘルプを使うべし

- 内部 procedure/function は、Unix shell から ‘idlhelp’ で、または本家のオンライン版を、どうぞ。
  - http://www.harrisgeospatial.com/docs/using_idl_home.html
  - とりあえず Using IDL -> IDL routines だけ眺めればよい。
  - まずは index か search で探す。
  - Example が便利。
  - See also… (ページの最下部) を、たどってみよう。
- IDL 初心者講習会資料も、かなり使えます。

- 外部 procedure/function については、そのソースコード自身にヘルプが書いてあるのが慣例。
  - ダウンロードしたら、まずはコードの先頭部（だけ）を読むベシ。

外部 function のヘルプ例

内部関数の場合も、
似たり寄ったりです。

```
;+ ここがファイルの先頭
; NAME:
;     MRDFITS;
; PURPOSE:
;     Read all standard FITS data types into arrays or structures.
; CALLING SEQUENCE:
;      Result = MRDFITS( Filename/FileUnit,[Exten_no/Exten_name, Header], /FPACK, STATUS=status)
; INPUTS:
;      Filename = String containing the name of the file to be read or…
; OUTPUTS:
;      Result = FITS data array or structure constructed from the designated extension…
; OPTIONAL OUTPUT:
;       Header = String array containing the header from the FITS extension.
; OPTIONAL INPUT KEYWORDS:
;       /FSCALE - If present and non-zero then scale data to float numbers for arrays…
; OPTIONAL OUTPUT KEYWORDS:
;       STATUS - A integer status indicating success or failure of the request.
; EXAMPLES:
;       (1) Read a FITS primary array:
;               a = mrdfits('TEST.FITS')    or
;               a = mrdfits('TEST.FITS', 0, header)
; 以下、本物の IDL プロがラムが続きます。
```

Calling sequence では、ヘルプ用の特殊な表記（IDL 文法的には誤り！）が一部使われています。

- カギ括弧 ([]: カッコ内はオプション、の意味)、割り算マーク (A/B: A または B、の意味)

注目 : 必須 vs. オプション。インプット vs インプット・キーワード。<- 後で解説します。

IDL本体（基礎）に集中すべし。おまけは当面忘れる。

- No GUI
- No iTOOL
- No Object programming
- Etc…

- GUI だけやむなく使ったことがあるが、他は全く使用経験ゼロ。
  - たまに外部 GUI ライブラリを使うことはある。

IDL の始め方

- idl または idlde で起動。
  - プロンプト “IDL>” が出る。
- プロンプトにコマンドをタイプする or コピー＆ペーストとで「メモファイル」からコピーする。
  - IDL は1行1コマンドが原則。
  - インタプリタなので、1行実行ごとに結果がメモリに残る。それを利用して、さらに処理を進める。
- 「メモファイル」がたまってきたら、何行分もまとめてコピー＆ペーストする。
- もっとたまってきたら、メモファイルをそのまま読み込むと便利な場合もある。
  - IDL> @mymemo.txt; @ を使って、mymemo.txt の内容を一行づつ読み込んで、実行する。
- IDL プログラム形式に改良する。
  - mymemo.txt そのままで、既にプログラムの形式にほとんどなっている。
    - ただし、おまじないとして、拡張子を .pro に変えて、最後の行に end を加えて下さい。
  - .run mymemo で実行。
    - .run は、特殊な IDL 本体に対する命令（ドット・コマンド、と呼ばれる）である。

つづき

- プログラムと「メモファイル」の違い
  - プログラムは、エラーが出たらそこで止まる。メモファイルは、エラーが出ても次にいってしまう。
    - あなたがコピペをするのと同じことを自動的にやっているだけだから。
  - プログラムでは、言語の制御構造が使いやすい。
    - If, for などが典型例。
      - これらがあると、一行1命令の原則が崩れるから。
      - 無理にこれらを含む「メモファイル」をそのまま読み込むと、syntax error で止まってしまう。
    - もちろんメモファイルでも if, for は使える。

```
IDL > .run
[プロンプトが変わることに注意。]
- for i=0,10 do begin
- print,i
- endfor
- end
[ここで初めて IDL がここまでの4行を認識して実行し、結果が出力される。]
IDL > [いつものプロンプトに戻る。]
```

    - 結局、ミニプログラムをつくって実行しているだけ。
  - どちらの場合も、結果がメモリに残った状態で終了する。
    - 引き続きお楽しみください。

デバッグの達人への道

- IDL ならデバッグは簡単。その特徴を活かす際の基礎知識集です。
  - 一度 IDL デバッグの醍醐味を覚えると、他の言語で開発できなくなります。。。

IDL プログラム実行時の、コンパイルとエラーの種類

- コンパイルエラー
  - インタプリタでも、コンパイルする。
    - コンパイル：最低限の構文チェックなどを指す。
  - サブルーチンは、呼び出されると自動コンパイルされる。
    - したがって、メインプログラムの途中でコンパイルエラーが起き得る。
    - 前もってコンパイルしておくこともできる。
- 実行時エラー
  - コンパイルが通っても、実行時エラーは発生する。
    - 例：Array の割り当て等は、実行時に決まるので。
  - 実行時エラーが起きると、そのままのメモリ状態で IDL シェルに落ちる。
    - インタラクティブモードになる。

エラーメッセージを読むべし

- 知るべき事
  - エラーの種類
  - エラーが発生したサブルーチン
    - 今自分はどこ(どのルーチン)にいるのかを知る。
    - 大きなプログラムでは、自分の位置を見失いがち。
  - エラーが発生したプログラムファイル
    - 1つの xxx.pro に複数の procedure/function がある場合もあるので、注意。
  - エラーが発生したプログラム行
    - IDL のエラー行表示は、信頼できる。
  - エラーが発生した具体的な箇所
    - ‘^’で問題の行のどこで止まったかが表示される。
    - ただし、あまり参考にならないこともある。

Error の例 1

- % Syntax error.
- At: /home/ohyama/xxx.pro, Line 10
- % Compiled module: xxx.
- % Attempt to call undefined procedure/function: ’xxx'.
- % Execution halted at: yyy 61 /home/ohyama/yyy.pro
- % $MAIN$
- IDL> help,/trace
- % At yyy 61 /home/ohyama/yyy.pro

- YYY (親ルーチン) 61 行目で、サブルーチン XXX が呼ばれ、XXX.pro がコンパイルされたが、その 10 行目に syntax error が発生したため、コンパイルが失敗した。
- そのため、XXX は実行されず、その直前の YYY の 61 行目で停止。

Error の例 2

- % Attempt to subscript A with <LONG (-1)> is out of range.
- % Execution halted at: xxx 185 /home/ohyama/xxx.pro
- % yyy 18 /home/ohyama/yyy.pro
- % zzz 61 /home/ohyama/zzz.pro
- % $MAIN$
- IDL> help,/trace
- % At xxx 185 /home/ohyama/xxx.pro
- % yyy 18 /home/ohyama/yyy.pro
- % zzz 61 /home/ohyama/zzz.pro
- % $MAIN$

- ZZZ（親）-> YYY（子）-> XXX (孫ルーチン）へと進んできたが、XXX の 185 行目で実行時エラーが発生し、その場で停止。

エラー停止時の IDL の状態

- インタラクティブ状態になっている。
- メモリの内容はそのまま保存されている。
  - ただし、今いるサブルーチン内の情報に限る。
  - メモリの内容を
    - 確認できる。
    - 修正できる。
    - 修正後、その場から再スタートできる。
- 意図的にエラーを発生させることができる。
  - デバッグ時に有効。
- エラー停止後、再度最初からやり直し実行する時は、メモリを一旦クリアすること！
  - さもないと、途中のサブルーチンの情報を元に、メインプログラムがイニシャルされる恐れ有り。混乱の元。

これだけは知っておきたい デバッグ用コマンド

- Print
  - デバッグメッセージの埋め込み
- Message
  - 強制エラー発行による、実行停止。デバッグモードへ移行。
- Stop
  - 好きな場所で一時停止。再開可能 (.cont)。
- Help
  - Variable の内容確認
  - Help,/trace で、今いるサブルーチンの確認
- .comp
  - コンパイル（し直し）
- .cont
  - 継続実行
  - エラーからの復帰
- Return or return,0
  - 強制的親ルーチンへの復帰
- .reset
  - メモリ内容をすべてクリアして、IDL 起動時の最初の状態に戻す。
    - ただし、IDL 環境変数はリセとされない、などの例外はある。
  - はじめからやり直す場合は、まず .reset すること。

エラーストップする90%の理由1

- IDL 設定に関するエラー。
  - プロット用 Window がでない。
    - よくある X11 の設定の問題。管理者に問い合わせ。
  - Color が出ない？画面が再描画されない？おかしい？
    - おまじないコマンド：device,decomposed=0,retain=2
- Procedure/Function undefined
    - % Attempt to call undefined procedure/function: ’xxx'.
  - 存在しないサブルーチンにアクセスしようとしている。
    - IDL path に該当プログラムファイルが存在しない。
      - Print,!path をしてみよ。きっと path が通っていない。
      - unix shell で IDL_PATH を追加・変更した後は、IDL を再起動させる必要あり。
    - 該当プログラムが、エラーでコンパイルできない。
      - .comp xxx.pro をしてみよ。きっとエラーがある。
    - 多くの場合、単なる typo (xxx.pro のつもりが xxy.pro を呼び出した)
- Variable undefined
    - Variable is undefined: xxx.
  - 定義していない array にアクセスしようとしている。
    - Array の定義が、遅すぎる or 忘れている or typo。
      - 該当 variable を help してみよ。

エラーストップする９０％の理由２

- Subscript out of range
  - あり得ない array の座標にアクセスしている。
    - % Attempt to subscript xxx with <LONG (-1)> is out of range.
    - % Subscript range values of the form low:high must be >= 0, < size, with low <= high:
    - あり得ない for loop カウンタ
    - Where 関数がマッチしていない。該当無しを表す -1 が帰ってきている。
      - Subscript を help してみよ。
      - Where 関数の Match count 返り値を使って、事前にチェックするのが鉄則。
- Function/procedure へのパラメーター渡しがおかしい。
  - % xxx: Incorrect number of arguments.
  - 例：y=func(x,a) には 2 つの argument が必要なのに、func(x,a,b) と3つ与えた。
  - たいがいが typo か勘違い。

エラーはでないが、処理内容が挙動不審な場合の90%の理由1

- スカラーのハズが、size=1 の array になっている。
  - Id がスカラーなら、Array[id] はスカラー
  - Id が array なら、array[id] はアレイ
  - アレイA＊スカラーBは、アレイC
    - アレイCのサイズはアレイAのサイズ
  - アレイA＊アレイBは、アレイC
    - アレイCのサイズはアレイA, B の小さい方のサイズ！
  - 悩む前に、help で内容を確認せよ。
- IF,whereなどの条件分岐が、期待どうりでない。
  - AND と &&, OR と || は大丈夫？
  - 悩む前に、IF 条件内容を help せよ。
- String に意図しないスペースが入っている。
  - ‘a’ と ‘_a’ と ‘a_‘ が混乱している。(‘_’ は空白）
    - Print だと気づかない。Help せよ。

エラーはでないが、処理内容が挙動不審な場合の90%の理由2

- Singed/unsigned Integer/long integer の違い。

```
IDL> help,32767S
<Expression>    INT  =  32767 (<= 2^15-1)
IDL> help,32767S+1S
<Expression>    INT  = -32768 (<= 2^15-1) bit が回ってマイナスになる！
```

- 処理結果が意図せず integer になった！
  - *2 の場合も *2.0 と明記しましょう。
- Function/procedure へのパラメーター渡しがおかしい。
  - Argument 数が少なくても動く！足りない部分は undefined 扱い。
    - 多すぎる場合は、エラーとなる。
  - Argument をすべて help してみよ。
- まったく同名の、異なる procedure/function がある！
    - そんな馬鹿な、と思っても、たまに起きる。そして、大いに悩む。
  - たとえば、バックコンパチでない複数バージョンのプログラムの全てにパスが通っている、など。
  - IDL_PATH の設定を確認せよ。
  - findpro 関数 (IDLASTRO) も便利。(unix の where みたいなもの)

IDL プログラム高速化

- 早いコンピューターを買う。
  - いたずらにプログラム上で速度を追求しない。読みやすさ優先。
    - IDL では、コードの高速化改良余地は少ないが。
  - メモリスワップしているようなら、メモリの増設が有効。
- FOR ループ、IF 文などは、なるべく使わない。
  - 細切れにせず、IDL built-in のルーチンをつかう。
    - そもそも早い。うまくいけばマルチスレッドが働く
  - Array 処理や Where を活用せよ。
- 多次元 Array アクセスの順番。
  - For loop をどちらを先に回すか？ -> コラムメイジャー
  - A[0,0] のお隣は A[1,0]; A[0,1] は遠い。。。
- メモリ管理（スワップ回避）
  - メモリを消費する大型 array は、
    - 多数の大フォーマット array はなるべく同時に使わない。サブルーチンを活用。
    - 不要になったら、消す。
      - A=0 を代入（こそくな手段）
      - 後でデバッグできなくなって泣く事も。

データの QL

- IDL は Interactive に Data をいじくる言語です。
  データの Quick Look の技を身につけましょう。

- Help
- Print
- Print,moment
- Plot (oplot)
- ATV

QL の達人となるべし1

- 解析処理のデバッグのため、データを様々な形で眺める技を身につける。
  - エラーは出ないが、処理結果がおかしいときのデバッグこそ、IDL の強いところ。

- HELP: まずは help で中身を確かめる。
  - Variable の dimension, 内容の種別（文字か数字か undefined か）を知る。
- Print: とりあえずプリントして中身を見る。
  - […] (サブアレー指定) を使って array の一部だけを見ることも、有効。
- Print,moment(): 対象が大きい場合は、だいたいの統計量をつかむ。
  - 平均値、分散など。

QL の達人となるべし2

- Plot,Oplot: トレンドを2次元グラフ上でつかむ。
  - Plot,xarray,yarray,xrange=[x1,x2],yrange=[y1,y2],/xlog,/yog,psym=3,color=1
  - Oplot,xarray,yarray2,psym=4,color=2
  - １次元 plot も可：Plot,yarray
  - ２次元以上のデータは、[…] を使って次元を落とす。
    - Plot,image[*,0] ->画像の x 軸に沿ったプロファイルの表示。
- ATV: ２次元画像を見る。
  - Atv,image2d,/block
    - あとは GUI で好きにいじる。
    - /block はおまじない。普通は要らないのだが、たまにつけないとうまく動かないことがある。理由は理解してません。
  - ３次元以上のデータは、[…] を使って次元を落とす。
    - image[*,*,0] -> z=0 でカットした面の画像。

サブルーチン

- 慣れてしまえば、サブルーチン化はとても簡単・便利。積極的に使いましょう。
- ２つの方法
  - Function
    - Output=function(argument,/keyword)
  - Procedure
    - Procedure,input_arg,output_arg,/keyword
- まったく同じ機能（コード）は、どちらの方法でも実現可能。
  - 呼び出し方が違うだけ。
  - 書き方が微妙に違うだけ。
  - お好きな方法で、どうぞ。

サブルーチン分割のススメ

- プログラム実行内容の整理
- Variable の整理
  - テンポラリの variable は、サブルーチンから脱出するときに、消える。
  - 逆に、過去の情報を参照したいときは、外部の変数に記憶させる。
- プログラムの汎用化
  - 何度も呼び出せる。
  - 微妙に挙動の異なる繰り返し部は、keyword などで対応。
  - 例
    - Stacked_image=mystack(image,/median)
    - Stacked_image=mystack(image,/clip_sigma,sigma=3.)
    - Mystack.pro の中に IF 文を書き、keyword 毎に処理を分岐させる。

Argument と keyword

- Argument, keyword は、入出力の両方に使える。
  - 同じ argument を、サブルーチンへの入出力の両方に使える。
  - しかも、C の様に、type を指定する必要もない。
  - なので、なんでもあり。たまにここで混乱する。
    - 例：サブルーチンにファイル名を表す string 変数を与え、内部でファイルを読み出した結果の float array を同じ変数（今度は float array）を使って取り出すことができる。
    - そんな馬鹿なサブルーチンは作らないと思うが、バグでそうなってしまっても IDL は文句を言わないので、そのまま次が実行されてしまう。。。
- Keyword は 1 or 0（true or false）のスイッチ
  - /keyword と keyword=1 は等価
  - Keyword がセットされたかどうかを知るには、Keyword_set 関数
    - または、キーワード変数が undefined なら、その keyword は設定されていない。
      - Undefined は size 関数で検知。

サブルーチンの様式

- 最初に、I/O (argument) を定義する。
- Pro または function で始まり、end で終わる。
  - 全ての処理は、この二つの間に。
- Function の場合は、返り値を return で指定する。

```
Pro myprocedure,input,output
        Output=myfunction(input)
End
Function myfunction,input
        Output=somefunction(input)
        Return,output
End
```

- 通常、サブルーチンは別ファイル (*.pro) に格納。
  - 1ファイル、1サブルーチン。
  - サブルーチン名とファイル名は、同じにすること。
    - ただし、ファイル名は .pro を必ずつける。

サブルーチンを使う。

- IDL_Path を通す。（既出）
  - 通常、IDL を起動したディレクトリにファイルをおいておけば、自動的に path は通っている。
- .compile でコンパイルしてみる。
  - .compile しなくても、呼ばれれば自動でコンパイルされますが。。。（既出）
- 呼び出すときは、他の IDL intrinsic 関数と同じ。

サブルーチン構造と、メモリアクセス

- データはすべて local 扱い
  - ただし、例外的に global を作ることができる。
- IDL プロンプトからアクセスできるのは、
  - 各サブルーチン内では、そのサブルーチンのメモリ内容だけ。
  - 別のサブルーチンに移動すると、親ルーチンの内容はアクセスできなくなる。
    - IDL が、その状態でアクティブなメモリ内容を自動で切り替えている。
- 同じ変数名を親ルーチンとサブルーチンの両者で使用していても、サブルーチン毎に変数の内容は入れ替わる。
  - Local なので、当然。

サブルーチン化とデバッグ手法1

サブルーチン内でエラーで止まったらどうするか？

Variable の内容の問題の場合

例：subscript の計算ミスで、array subscript out-of-range が起きた。

- 今どこにいるかを知る。
  - Error message を読む or help,/trace
- バグの修正法を考える。
  - QL 手法を駆使して、variable の中身を確認。
  - 問題のある変数を探し出し、修正を考える。
- 変数を外から書き替える。
  - インタラクティブモードなので、自由にコマンドラインから変数をいじれる。
- .cont を実行する。(continue)
  - 運が良ければ、あたかもエラーがなかったかのように、処理が続く。
    - もちろん姑息な手段がうまくいかない場合も多いので、その場合は次ページ。
- うまく切り抜けたら、サブルーチンそのものを修正する。
- 最初から実行し直す。

サブルーチン化とデバッグ手法2

致命的なバグで、実行継続が不可能な場合

例：巨大な for loop の中で subject out-of-range が起きた。

- 今どこにいるかを知る。
  - Error message を読む or help,/trace
- バグの修正法を考える。
  - QL 手法を駆使して、variable の中身を確認。
- プログラムを修正する。Edit and save.
- 一つ上のルーチンに戻る。
  - Return (procedure の場合) or return,0 (function の場合)
- 修正プログラムを再コンパイルする。
  - .compile ‘program’
- 修正サブルーチンを試す。
  - コマンドラインから、単体で（そのサブルーチンだけ）実行する。
- うまくいったら、最初から実行し直す。
- エラーストップしても、無駄にしないで復活させるベシ。

2次元画像処理以外のよくあるIDL の使い道

- IDL はプログラム言語ですので、何でもできます。
- 2次元 FITS 画像解析以外で、天文関連でIDL が便利で活躍する場面を、少しだけ紹介します。

なんとか統計、なんとかフィット

- 例：観測データの単純な平均・分散を求めたり、リニアフィットを行うのではなく、明らかな「外れ・問題データ」を取り除いたり、やや特殊な「なんとか統計」を行う。
- 例 (idlastro より)
  - Meanclip.pro
  - Biweight_mean.pro
  - Resistant_mean.pro
  - Robust_sigma.pro
  - Robust_poly_fit.pro
  - …
- すべて、IDL 言語で実装されたプログラム（procedure/function) です。
  - 中身を読むと、何シグマでデータを仕分けする、なんていうことが、そのまま丁寧にプログラムされています。
  - IDL だからいろいろなマニアックな統計ができるのではなくて、世界中の人がよい procedure/function を作って公開してくれているだけ。
  - IDLASTRO や google で検索して、適切な物を探してください。

複雑なフィット関数を用いたデータのフィット

- 例:
  - 楕円銀河の ¼ 乗則、または銀河の表面輝度マップをバルジ＋ディスク構造でモデルフィットする。
  - 複数の輝線をもつスペクトルを、天文学的に正しい条件でフィットする。例えば、Hα と [NII] ダブレットがあり、[NII] ダブレットの強度比はつねに 1:3 だが、[NII]/Hα は可変。すべての輝線はおなじ後退速度にある。この場合の輝線強度と後退速度は？ただし輝線プロファルはガウス形で近似できる。
- 一般的な（任意関数に対する）最小二乗フィット・ライブラリーが公開されています。
  - IDL built-in 版もありますが、MPFIT library が有名です。
    - http://www.physics.wisc.edu/~craigm/idl/fitting.html
    - これも、IDL で地道にコードされたプログラムです。
  - 最小二乗の数値的な求め方は、専門書を調べてください。初期値から始まって、ベストフィット解を逐次的に求めます。
- 状況に従った「モデル関数」を IDL の function (例えば、y=ax+b) として作って、観測データ (x_obs[*], y_obs[*]) と適当な初期値 (a0, b0) を MPFIT library に渡せば、ベストフィットパラメーター（例えば a, b, delta_a, delta_b）が得られます。
  - 詳しくは、MPFIT のウエブを見てください。簡単です。

天体のカタログ操作

- 大規模な天体カタログは、FITS テーブルで与えられることが多いです。
- FITS カタログは、FITS 画像と同じ読み込みルーチン（後述）で読めます。
  - どちらもおなじ FITS 規約に従っているから。
- IDL で読み込んでしまえば、画像もカタログもどちらも「アレイ」です。
  - 例えば、天体数が100のカタログを読めば、要素数100のアレイができます。
  - これまでの知識を活用して、自由に処理してください。
    - 例：r, b バンド等級から、r-b カラーを求める（単純なアレー同士の引き算）
    - r-b カラーから、「赤い銀河」だけを選ぶ（if 文、where 関数など）
- 一般に、一つの天体に対して多くの情報があり、テーブルになっています。
    - 例：それぞれの天体には name, RA, DEC, b_mag, r_mag の情報があります。
  - この場合、「構造体」と呼ばれる特殊な IDL 変数が使われることが多いです。
    - 例：mycat[0].name='NGC0000' & mycat[0].ra=123.456 … mycat[n-1].r_mag=0.0
    - ここで、mycat が構造体で、N の要素を持つアレーです。
  - それぞれの要素 n に対して、name, RA, DEC などの具体的な数値、文字列が付随しています。個別の情報の変数が、「.」（ピリオド）の後に付きます。
    - 例えば、b_r_color=mycat.b_mag – mycat.r_mag & print,b_r_color[0] などとします。
- 構造体は初心者にはハードルがやや高いですが、なれれば簡単ですし、カタログ操作（というより、管理）を劇的に楽にしてくれるので、便利です。

（講義編の）最後に

- ざっくばらんなコメントです。

- もっと勉強したい方へ。
  - 他人のプログラムを読もう。
    - そして改造。。。
  - 最初に紹介した参考書の斜め読みが良いと思います。
  - IDL の友達を作りましょう。（いま周りにいます）
- もっと複雑な処理がしたい方へ。
  - 複雑な処理も、結局は単純な処理の組み合わせ・繰り返しなので、少しづつくみ上げてください。
  - サブルーチン化とデバッグのコツが分かれば、いくらでも複雑・大きなプログラムができます。
  - シンプルかつ美しいプログラムを書くことを、お勧めします。
    - 高速化は2の次。
    - Array 名に意味を持たせたり、要所要所にコメントを残す、など。

つづき

- IDL は万能ではありません。
  - 私は今でも定期的に IRAF を使っています。Splot とか。
  - うまく組み合わせてください。
    - 基本は、低レベルのルーチンは IDL が得意です。
- IDL は、他の天文研究でも使えます。たとえば
  - 複雑かつ奇麗な plot
  - データの interpolation/fitting などの、数学処理。
  - 特別な統計処理。”なんとか統計・なんとかフィット”とか。
  - 大きな天文 Catalog 処理
    - where 関数が威力を発揮！
- もっとよい講習を受けたい方へ。
  - ADC のアンケートに答えてください。
  - この講習資料への具体的なフィードバック（文句）も大歓迎。
  - その他なんでも、ADC スタッフか私まで意見をください。