2012年度第1回IRAF講習会
「CL Scriptの基礎とパッケージ化」

広島大学・宇宙科学センター 秋田谷　洋

2012/7/26, 7/30 改訂

2012/7/19(-20)
国立天文台・三鷹・天文データセンター

Contents

1. Introduction

今回の参加者のみなさんのIRAF経験

	設問	人数
1	ほとんど使ったことがない	3
2	コマンドラインから基本的なタスクの実行ができる	10
3	数十行程度のCL scriptを書ける	0
4	より複雑なCL scriptを書ける	0

本講習の目標

- コマンドラインからのIRAF操作の基本を覚える・復習する。
- 簡単な画像処理のCL scriptが書けるようになる。

本講習の内容

1. IRAFの基本設定

   (経験者にとっては復習・兼、本講習の環境準備)

2. コマンドプロンプトからの基本操作：撮像画像一次処理を通じて

   (再び経験者にとっては復習ですが、他の人の流儀が参考になることもあるでしょう。)

3. 簡単なCLスクリプトの作成と実行

4. 複数のCL スクリプトのお手軽パッケージ化

テキストの表記・環境に関する但し書き

- コマンド入力
  - Unix シェルコンソールから

$ command ⏎

  - IRAF cl プロンプトから

ecl> help language ⏎

clpackage.language:

Language package.

...

  - ここで、"$", "ecl>" は、入力待ち状態でのコマンドプロンプト。赤字が入力文字列。"⏎"はEnter(Return)キー。青字はコマンドからの出力。
- Unixシェルは、csh系(csh, tcsh) を想定。

※sh, bashだと環境変数の扱いなどが変わってきます。IRAF使うときは、csh系の方がなにかと無難です。

2. IRAFの基本情報と初期設定

環境準備

- 補助ソフトウェア立ち上げ
  - テキストエディタ
  - 多機能FITS画像ビュワー SAOImage ds9
- IRAFの初期設定
  - ターミナルソフトxgterm立ち上げ
  - IRAF用ディレクトリ作成
  - 設定ファイルlogin.cl作成

テキストエディタ

- 設定ファイル、ファイルリスト、CL scriptの編集に必要。
- 使いやすいものを使ってください。最低限、ファイルを開く・編集する・保存する、ができればOKです。

- gedit
  - X Window上で動く。直感的。
  - コンソールから "gedit &" または、Gnome メニューの "Applications"→"Accessories"→"gedit Text Editor"
- emacs
  - コンソール上 or X Window上ともに動く。多機能。多少キーバインド覚える必要あり。
- vi
  - コンソール上。手軽。特有のキーバインドを覚える必要あり。

※ 講師(秋田谷)は、小作業はvi、CL script編集はemacsが好みです。

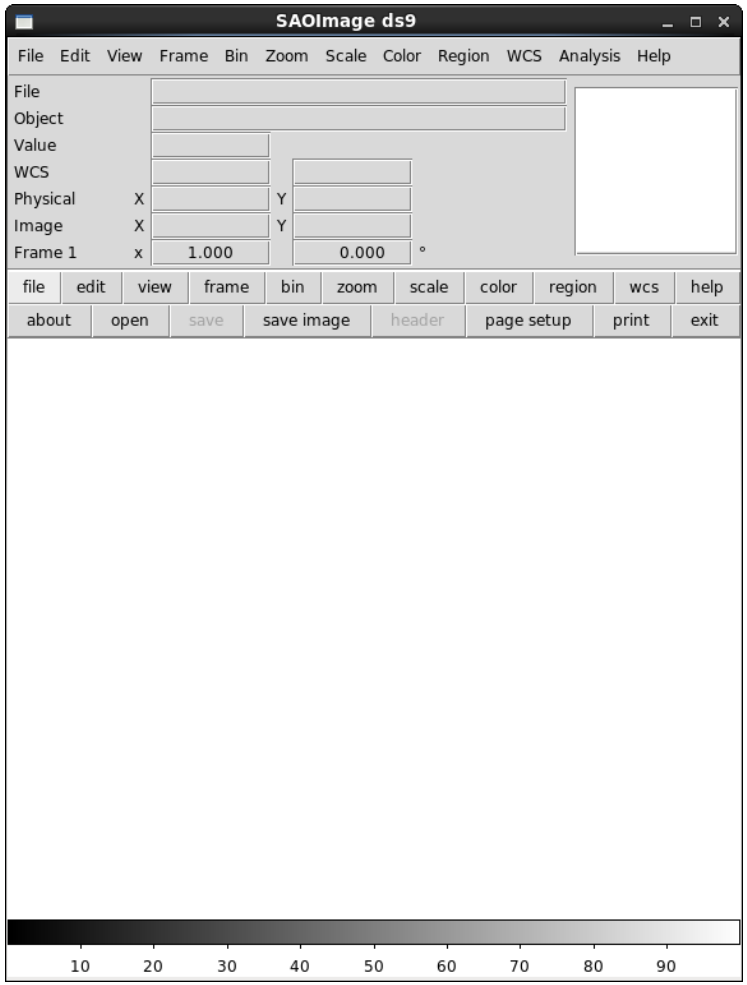
多機能画像ビュワー SAOImage ds9

- 光赤外天文業界で標準的な画像表示ツールの一つ。
- 多彩な画像表示・処理機能搭載。
- IRAFとの親和性非常に高い。

- 立ち上げ方法
  - 適当なシェルコンソールで

$ ds9 & ⏎

"&" を付けるのは、コマンドを
バックグラウンドで
実行するための常套手段。



IRAFの初期設定(1)

1. xgterm (ターミナルソフトウェア)を立ち上げる。

$ xgterm –sb ⏎ (xgtermの立ち上げ。-sbはスクロールバーを表示するオプション)

- Xgtermが立ち上がる。カーソルをxgtermに移動して、以後xgterm上で作業。

※ xtermなども使えますが、iraf使う場合はxgtermがおススメです。

2. 適当なIRAF用のディレクトリを作る

$ cd /adc/data/⏎ (作業用大元ディレクトリに移動)

$ mkdir guest?? ⏎ (作業用ディレクトリの準備"??"は各自の数値を。)

$ cd guest?? ⏎

$ mkdir iraf ⏎ (~/iraf/ というディレクトリを作成。名前は任意。)

$ cd iraf ⏎ (今作ったディレクトリに移動)

$ pwd ⏎ (今いるディレクトリを確認)

/adc/data/guest??/iraf　　#ここが以後のIRAF作業の起点。

- ※ 人によっては、ホームディレクトリで作業する場合もあるようです。(好み・流儀の問題です)

講習後講師注) 講習時は、各自の作業ディレクトリを /adc/data/guest??/.としていました。後にこの資料を参照される方は、ホームディレクトリ /home/*username*/に置き換えて読まれると良いでしょう。()

IRAFの初期設定(2)

3. IRAFの初期設定 (初めてirafを使うときのみ)

$ mkiraf ⏎ (irafの初期設定ファイルを作成する)

-- creating a new uparm directory

Terminal types: xgterm,xterm,gterm,vt640,vt100,etc.

Enter terminal type: xgterm ⏎ (← 普段使うターミナルを
xgtermに設定する)

A new LOGIN.CL file has been created in the current directory.

You may wish to review and edit this file to change the defaults.

(→ login.cl という初期設定ファイルが作られました。)

$ ls –F ⏎ (何が作られたか確認してみましょう)

login.cl → IRAFの初期設定ファイル(テキストファイル)

uparm/ → IRAFのタスク群のパラメター保存ディレクトリ
(最初は空っぽです。)

初期設定ファイル：login.cl の中身 (1)

- login.cl の中身を確認してみましょう。

  (例) $ less login.cl ⏎ (上下キーでスクロール、"q"で終了)

```
# LOGIN.CL -- User login file for the IRAF command language.

# Identify login.cl version (checked in images.cl).
if (defpar ("logver"))
    logver = "IRAF V2.16 March 2012"

set     home            = "/home/lecture/iraf/"
set     imdir           = "HDR$/"
set     cache           = "/iraf/cache/lecture/"
set     uparm           = "home$uparm/"
set     userid          = "lecture"

# Set the terminal type.  We assume the user has defined this correctly
# when issuing the MKIRAF and no longer key off the unix TERM to set a
# default.
#if (access (".hushiraf") == no)
#    print "setting terminal type to xgterm..."
#stty xgterm


# Uncomment and edit to change the defaults.
#set    editor          = vi
#set    printer         = lp
#set    pspage          = "letter"
#set    stdimage        = imt800
#set    stdimcur        = stdimage
#set    stdplot         = lw
#set    clobber         = no
#set    imclobber       = no
#set    filewait        = yes
#set    cmbuflen        = 512000
#set    min_lenuserarea = 64000
set     imtype          = "fits"
#set    imextn          = "oif:imh fxf:fits,fit fxb:fxb plf:pl qpf:qp stf:hhh,??h"
```

(1)基本パラメーターの設定

```
# Default USER package; extend or modify as you wish.  Note that this can
# be used to call FORTRAN programs from IRAF.

package user

task    $adb $bc $cal $cat $comm $cp $csh $date $dbx $df $diff  = "$foreign"
task    $du $find $finger $ftp $grep $lpq $lprm $ls $mail $make = "$foreign"
task    $man $mon $mv $nm $od $ps $rcp $rlogin $rsh $ruptime    = "$foreign"
task    $rwho $sh $spell $sps $strings $su $telnet $tip $top    = "$foreign"
task    $awk $vi $emacs $w $wc $less $rusers $sync $pwd $gdb    = "$foreign"
task    $xc $mkpkg $generic $rtar $wtar $buglog                 = "$foreign"
#task   $fc = "$xc -h $* -limfort -lsys -lvops -los"
task    $fc = ("$" // envget("iraf") // "/unix/hlib/fc.csh" //
            " -h $* -limfort -lsys -lvops -los")
task    $nbugs = ("$(setenv EDITOR 'buglog -e';" //
            "less -Cqm +G " // envget ("iraf") // "local/bugs.*)")
task    $cls = "$clear;ls"
task    $clw = "$clear;w"
task    $pg = ("$(less -Cqm $*)")

if (access ("home$loginuser.cl"))
    cl < "home$loginuser.cl"
;
Keep
```

(2) Iraf外コマンドの登録

```
# Load the default CL package.  Doing so here allows us to override package
# paths and load personalized packages from our loginuser.cl.
clpackage
```

```
# List any packages you want loaded at login time, ONE PER LINE.
images          # general image operators
plot            # graphics tasks
dataio          # data conversions, import export
lists           # list processing
```

(3)自動読み込みパッケージの指定

```
# The if(deftask...) is needed for V2.9 compatibility.
if (deftask ("proto"))
    proto       # prototype or ad hoc tasks

tv              # image display
utilities       # miscellaneous utilities
noao            # optical astronomy packages
vo              # Virtual Observatory tools
```

(3)

```
prcache directory
cache   directory page type help
# Print the message of the day.
if (access (".hushiraf"))
    menus = no
else {
    clear; type hlib$motd
}


# Uncomment to initialize the SAMP interface on startup.
if (deftask ("samp") == yes) {
  printf ("Initializing SAMP .... ")
  if (sampHubAccess() == yes) {
      # Enable SAMP messaaging.  Set default handlers that don't require
      # VO capabilities.
      samp quiet
      samp ("on",                                       >& "dev$null")
#     samp ("handler", "table.load.votable", "tinfo $url",      >& "dev$null")
#     samp ("handler", "image.load.fits", "imstat $url",        >& "dev$null")
      samp noquiet
      print ("on")
  } else
      print ("No Hub Available¥n")
}


# Delete any old MTIO lock (magtape position) files.
if (deftask ("mtclean"))
    mtclean
else
    delete uparm$mt?.lok,uparm$*.wcs verify-

keep
```

login.cl チェックしておきたい項目

- set home = "/home/lecture/iraf/"
  - iraf の基本ディレクトリ。環境変数 homeで参照できる。
- set imdir = "HDR$/“
  - iraf形式の画像ファイルをどこに作成するかを指定する。”HDR$/” であれば、作業が行われたディレクトリとなる。irafシステムの初期設定によっては、特定のディレクトリがデフォルトで指定されている場合があるので注意。
- #set printer = lp
  - implotなどのプロットツールからの印刷先を指定する。
- set imtype = "fits“
  - 画像ファイルの拡張子が省略されたとき、どの画像形式とみなすかを指定。最近はほとんどfitsを扱うので”fits”としておけば問題ない。古いIRAFでは、ここのデフォルトが”imh”(IRAF画像形式)となっている場合があるので注意。また昔開発された解析パッケージでは、あえてここを”imh”にしなければ動かないものもあるかもしれない。

IRAFの立ち上げ

- xgterm上で、"login.cl" ファイルのあるところに移動し、"cl" コマンドを実行する。

$ cd /adc/data/guest??/iraf/ ⏎

$ pwd　　#今どこにいるか確認。

/adc/data/guest??/iraf

$ cl ⏎ (irafの立ち上げ)

```
NOAO/IRAF PC-IRAF Revision 2.16 EXPORT Thu May 24 15:41:17 MST 2012
  This is the EXPORT version of IRAF V2.16 supporting PC systems.

Welcome to IRAF.  To list the available commands, type ? or ??.  To get
detailed information about a command, type `help <command>'.  To run  a
command  or  load  a  package,  type  its name.   Type  `bye' to exit a
package, or `logout' to get out  of the CL.    Type `news' to find  out
what is new in the version of the system you are using.

Visit http://iraf.net if you have questions or to report problems.

The following commands or packages are currently defined:

    apropos    images.    noao.      proto.     system.    vo.
    dataio.    language.  obsolete.  softools.  tables.
    dbms.      lists.     plot.      stsdas.    utilities.

ecl>
```

ecl>　(→ 初期メッセージ後、iraf command language の入力プロンプトが表示される)

ecl> から何ができるのか?

- CL (=IRAF Command Language) ; ecl = enhanced CL
- IRAF環境のコマンドインタプリター (逐次解釈プログラム)
  - コマンド・task の実行
  - パラメターの設定
  - エラー処理

  などを行う。
  ※ 詳細は、ecl> help intro ⏎ で。

- ecl> プロンプトから実行できるもの
  - IRAF builtin(組み込み)コマンド・関数、制御構文
    - IRAFが持っている基本的な機能を持つコマンド・関数群("cd", "printf"など)、制御構文("for", "while", …など)
  - IRAF task
    - IRAF特有のプログラム言語で書かれた複雑な解析プログラム
    - 既存のIRAF taskとbuiltin command、制御構文の組み合わせで書かれた"CL-script"を一つの解析プログラムとして登録したもの
  - UNIXコマンド
- CLから抜けるには、“logout”コマンド。

  ecl> logout ⏎

どんなコマンド・関数・制御構文が使えるのか?

- 基本的にはUNIXシェルコマンド・C言語によく似ている。しかし、ちょくちょく微妙な違いがあるので注意！
- コマンド・制御構文を確認してみよう。

ecl> help language ⏎

制御構文

```
Language components:

    break * Break out of a loop
     case * One setting of a switch
 commands - A discussion of the syntax of IRAF commands
  cursors - Graphics and image display cursors
                                                    …
```

Builtinコマンド・関数

```
Builtin Commands and Functions:

   access - Test if a file exists
     back - Return to the previous directory (after a chdir)
     beep - Send a beep to the terminal
      bye - Exit a task or package
    cache - Cache parameter files, or print the current cache list
       cd - Change directory
    chdir - Change directory
       cl - Execute commands from the standard input
    clbye - A cl followed by a bye (used to save file descriptors)
    clear - Clear the terminal screen
                                                    …
```

さらに詳しく知りたいときは、

ecl> help printf ⏎ (例: printf関数の詳細を知りたい)

※ "help" 自体が、builtinコマンドですね。

help languageで “*”印のついている制御構文(break, caseなど)のhelpは、

ecl> help language.if ⏎ のように、"language."の後に続ける。

UNIXシェルコマンド・C言語関数とIRAFコマンドの対応例

UNIX shell, C	IRAF
cat	type, concatenate
continue	next
grep	match
rm	delete
mv	rename, movefiles, imrename
ls -1	files
ls	dir
exit	bye
more, less	page

など

IRAFの変数型

- IRAFでは以下の変数型がある。
  - int : 整数型
  - real : 実数型
  - string : 文字列
  - bool: 論理(boolean)型 (yes or no)
  - file : ファイル (※単にファイル名の文字列には stringを使う)
  - elc.
- 使う前に、宣言が必要

```
ecl> real value1 ⏎                  #実数変数 value1 の宣言
ecl> value1 = 5.6 ⏎                 # 値を代入
ecl> = value1 ⏎                     # 値を表示 ("= 変数名")
5.6
ecl> string filename1 ⏎             #文字列変数 filename1の宣言
ecl> filename1 = "image1.fits" ⏎        #文字列代入と表示
ecl> = filename1 ⏎
image1.fits
ecl> value2 = 5 ⏎     # 宣言していない変数を使おうとすると・・・
ERROR: parameter 'value2' not found          #怒られる
```

事前宣言済み変数

- 以下の変数は、事前に宣言されている。宣言文なしで使える(= 勝手に他の用途に使うことができない！)。
  - 整数(int)型: i, j, k
  - 実数(real)型: x, y, z
  - 文字列(string)型: s1, s2, s3
  - 論理(bool)型: b1, b2, v3
  - ファイル型 : list
- 変数の一覧を見たいとき

  ecl> lparam cl ⏎

予約語は変数名に使えない

- 基本コマンドや宣言文構文など、「予約語」は変数名に使えない!

ecl> string file ⏎

** Syntax error

**: string file

^

>>>

“file” は、file型変数を宣言するコマンドなので、

ファイル名の文字列変数などに使えない。

ecl> string if ⏎ = × (“if” は制御構文で用いる命令なので不可)

ecl> string string1 ⏎ = ○ (“string”ではなく”string1”という名前なのでOK)

変数処理を試してみよう: ついでに表示の関数も

- 簡単な計算も可能

```
ecl> real x1, x2, x3 ⏎
ecl> x1=5.6; x2=6.7 ⏎
ecl> x3=x1*x2 ⏎
ecl> print( x3 ) ⏎
37.52
```

- 少し凝った結果表示も試してみよう

```
ecl> string info ⏎
ecl> info = "x1*x2= " ⏎
ecl> printf("# test : %s %6.2f¥n", info, x3 ) ⏎
# test : x1*x2=   37.52
         ^^^^^^^   ^^^^^^
```

infoの内容　x3の内容

printf関数 : C言語とほぼ同じ (詳細は help printf)⏎
%s : 引数に与える文字列に置き換える
%6.2f : 引数に与える実数(6桁・小数点以下2桁)に
置き換える
¥n : 改行文字

ファイル名処理(1)：ワイルドカード

ecl> cd (サンプルファイル置場)/sample1/ ⏎
ecl> ls HP005414*fits ⏎
　　# HP005414で始まりfitsで終わるファイル名
HP0054140_0.fits HP0054140_1.fits HP0054141_0.fits HP0054141_1.fits
ecl> ls HP0054177_?.fits ⏎
HP0054177_0.fits HP0054177_1.fits
　　# HP0054117_で始まり、一文字分任意で、
　　# その後_1.fitsが続くファイル名
ecl> ls HP005366[358]_0.fits ⏎
HP0053663_0.fits HP0053665_0.fits HP0053668_0.fits
　　# HP05366で始まり、次の1文字が3、5、8で、
　　# その後_0.fitsが続くファイル名
　　# (注: []表記は、lsのようにUNIXコマンドをtaskに
　　#登録したものに対しては有効だが、imcopy の
　　#ように IRAF独自のコマンド・taskでは使えない。)

ファイル名処理(2)：リストファイル

- ファイル名を羅列したテキストファイル(一行当たり一個)に記載されたファイルすべてに、まとめて処理を行うことができる。
- "@"マーク+リストファイルのファイル名 という表記。

```
ecl> files HP00*_1.fits > file.lst ⏎
ecl> cat file.lst ⏎                    #確認
        ...
HP0054139_1.fits
HP0054140_1.fits
HP0054141_1.fits
        ...
ecl> imstat @file.lst ⏎
#               IMAGE     NPIX      MEAN    STDDEV     MIN      MAX
        ...
    HP0054139_1.fits   2350416     746.2     341.5     474.   52226.
    HP0054140_1.fits   2350416     743.3     324.3     459.   38427.
    HP0054141_1.fits   2350416     742.2     322.2     468.   38780.
        ...
```

UNIXコマンドの使い方

- UNIXコマンドはそのまま打っても使えない。頭に"!"をつける。

  ecl> uname –a ⏎

  ERROR: task `uname' not found　　#(そんなtaskは無い!)

  ecl> !uname –a ⏎

  Linux new-r19 2.6.32-220.el6.x86_64 #1 SMP Tue Dec 6 19:48:22 GMT 2011 x86_64 x86_64 x86_64 GNU/Linux

- login.cl に以下のように追記しておくと、"!"なしで実行できるようになる。(IRAFは立ち上げなおす必要あり)

  (= UNIXコマンドをIRAFのtaskとして登録することになる)

```
package user

task    $adb $bc $cal $cat $comm $cp $csh $date $dbx $df $diff  = "$foreign"
task    $du $find $finger $ftp $grep $lpq $lprm $ls $mail $make = "$foreign"
task    $man $mon $mv $nm $od $ps $rcp $rlogin $rsh $ruptime    = "$foreign"
task    $rwho $sh $spell $sps $strings $su $telnet $tip $top    = "$foreign"
task    $awk $vi $emacs $w $wc $less $rusers $sync $pwd $gdb = "$foreign"
task    $xc $mkpkg $generic $rtar $wtar $buglog     $uname          = "$foreign"
                                                …
```

IRAFの基本心構え(1) 面倒がらずhelpを読もう

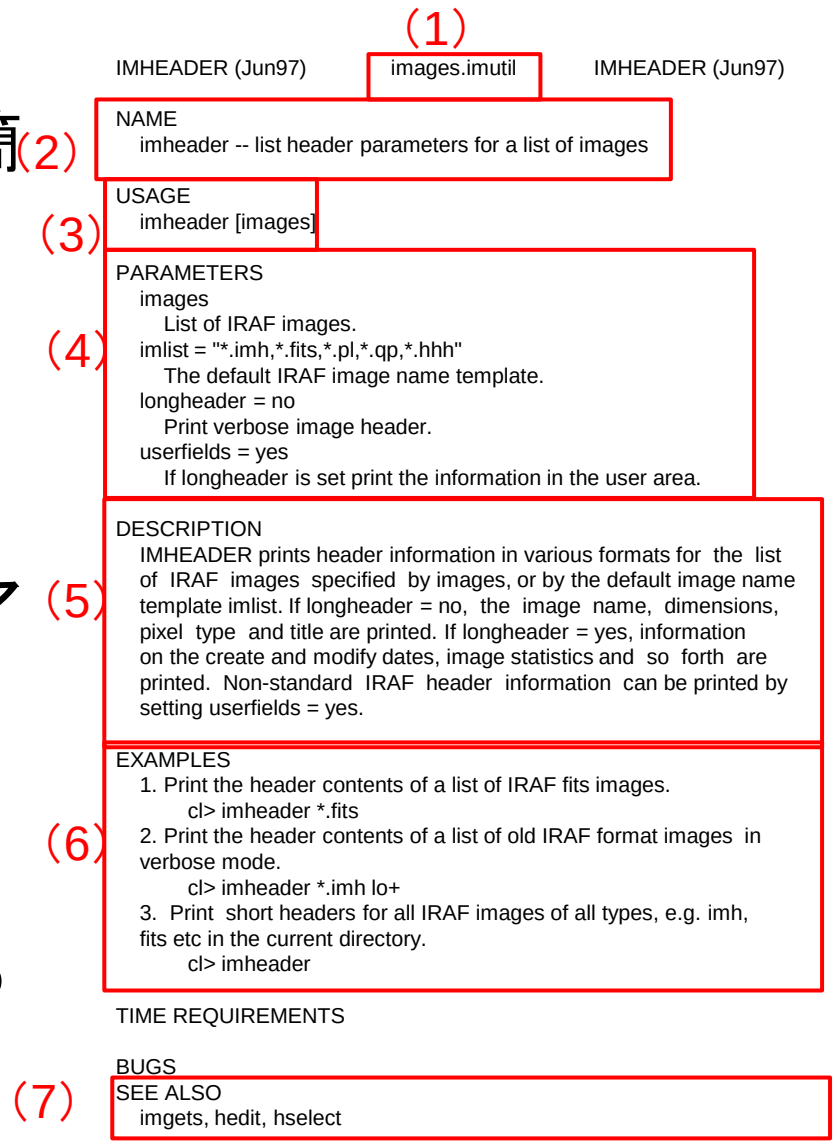
- taskの詳細は、help コマンドでオンラインマニュアルが読める。
  $ help imhead ⏎ (一方向読み) 又は
  $ phelp imhead ⏎ (双方向読み)

```
IMHEADER (Jun97)          images.imutil          IMHEADER (Jun97)

NAME
   imheader -- list header parameters for a list of images

USAGE
   imheader [images]

PARAMETERS

   images
      List of IRAF images.
                                                              …
```

- 英語で時に長いです。しかし、
  - どんなパラメターを指定できるかが分かる(予想外の機能があることを発見できる)
  - 実行例が載っている (せっかちに使いたいとき便利)
  - 機能に関連するtaskの一覧がある (新しいtaskの発見へ)
  - 中身が本当に何をやっているのか知る必要があるときには読むしかない。

helpファイルの注目点

1. taskが所属するパッケージ名
2. NAME：taskの正式名とごく簡単な機能説明
3. USAGE：実行方法
4. PARAMETERS: 指定できるパラメターの種類と指定方法
5. DESCRIPTION: 処理内容・アルゴリズム等の詳細な説明
6. EXAMPLES：具体的な実行例
7. SEE ALSO：機能が関連する他のtask一覧



IRAFの基本心構え(2) 欲しい機能のtaskを探す

- (例) 画像を90°回転させたいんだけど、そんなtaskは無いのかな?

ecl> references rotate ⏎ ("rotate"というキーワードでtaskを検索)

searching the help database...

im3dtran - 3d image transpose (used for rotates as well) [vol]

rotate - Rotate and shift a list of 2-D images [imgeom]

→ まさに"rotate"という名前の taskがあった!

ecl> help rotate ⏎

…

SEE ALSO

imtranspose, imshift, magnify, lintran, geotran, geomap

→ rotateの使い方が分かった。さらに、似たような機能のtaskも芋づる式に見つかった。helpで調べてみよう・・・。

IRAFの基本心構え(3) 作業ログをとろう

- テキストエディタを開いておき、実行したコマンドラインは、メモとともに逐一テキストファイルにペーストしていくと後々便利。
  - 解析の間違いがあっても、後から見直して気づくことができる。
  - 解析の手順を整理・分析・再検討する資料になる。
  - 後で全く同じ解析を再現できる。
    - 途中で解析間違いがあっても、そこまで戻ってやり直すのが容易。
    - 似たパターンの解析をあとで繰り返すとき、そのまま or 微調整するだけで済む。(場合によっては、コンソール画面にコピー&ペーストでOK)
    - CL scriptへの移行が容易。(コマンドラインでも後述の"program mode記法"を使っておくと、さらにscriptへの発展が容易になる)

以後の講習方針

- 広島大 1.5mかなた望遠鏡で撮影した星の画像について、基本的な一次処理、
  - 必要領域
  - バイアス引き
  - フラット補正

を行います。

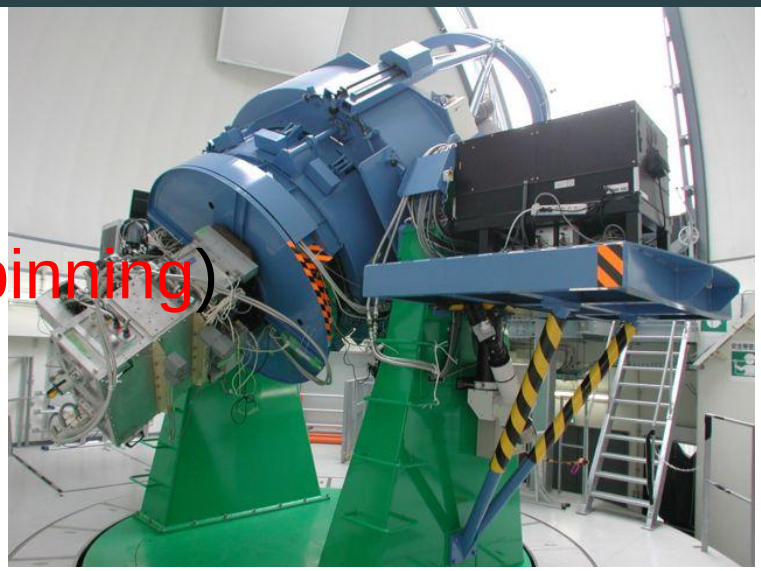
- この作業を、
  - コマンドラインから
  - CL scriptから

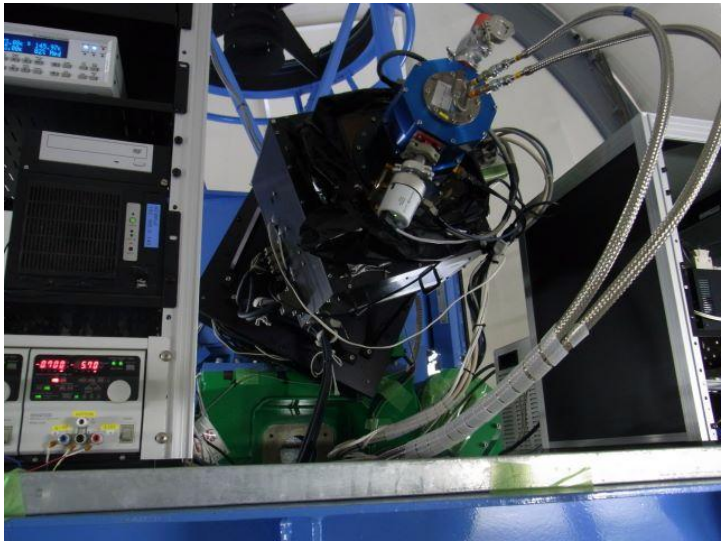
それぞれ行うことで、IRAFとCL scriptの基本を学んでいきましょう。

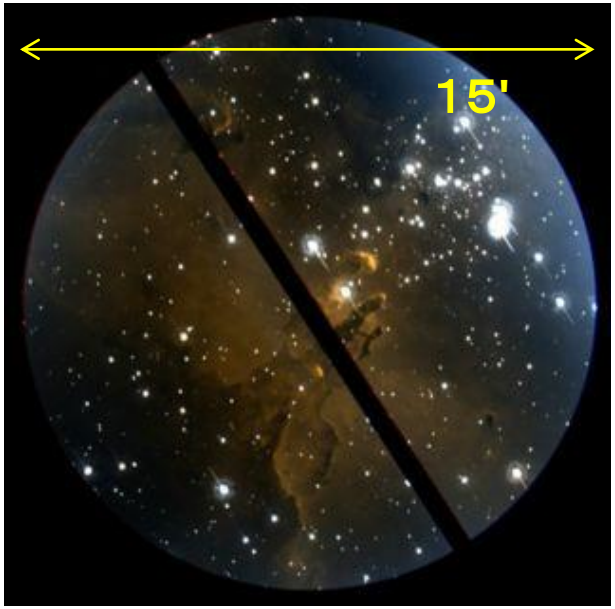
3. CLプロンプトからのコマンド・タスクの実行

広島大かなた望遠鏡 HOWPol 撮像画像

- 波長域：0.45-1.1 μm
  (filters: B,V,R,I,z',Hα)
- 撮像: φ15' (0.6"/pix; 2x2 pixels binning)
- 偏光撮像、分光も可能

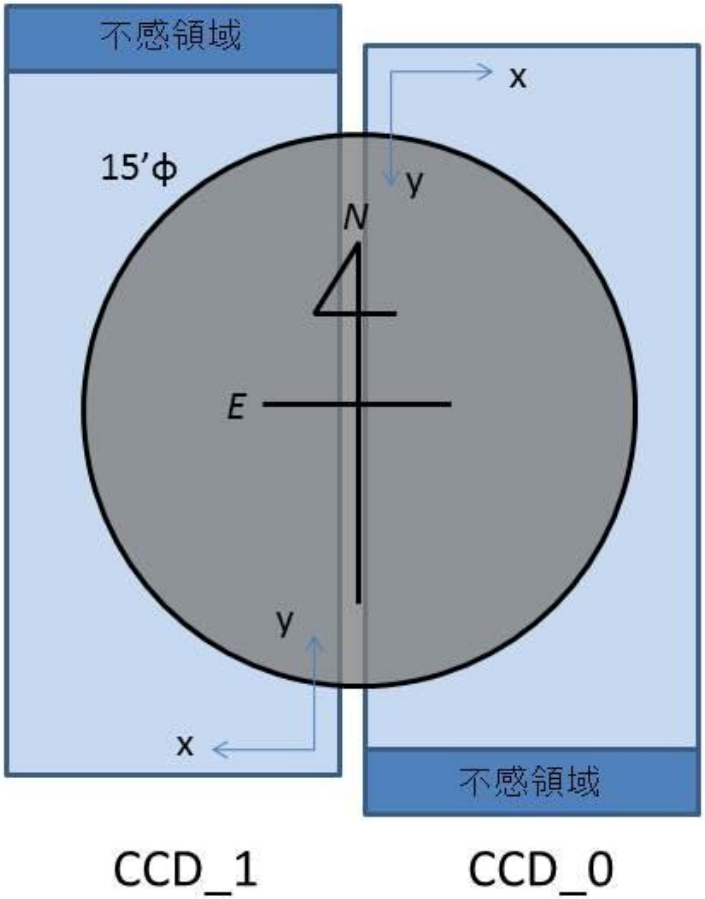






HOWPolの視野とCCD 2枚の関係

視野とCCDとの並び方



・Nsローテータ方位角ΔPA=0° の場合（PAの原点は観測期により異なる）

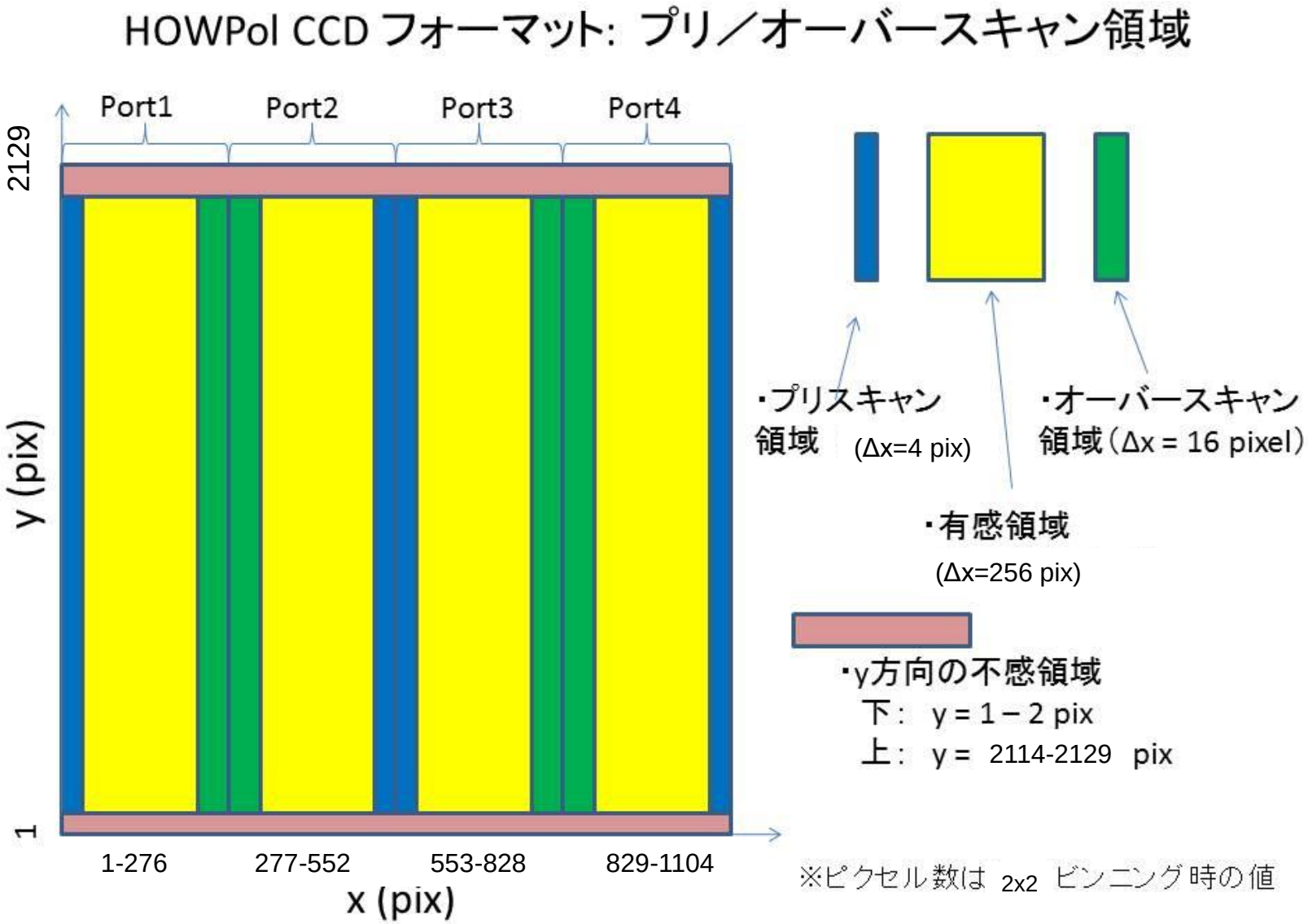
・CCD間のギャップ間隔は約27”

・ds9で表示する際は、tileで左側にCCD_1をX反転で、右側にCCD_0をy反転で表示すると、視野をちょうど良い向きにカバーする。

・分光時は、スリットは東西方向で、北が短波長側、南が長波長側

CCD一枚当たり 1104 x 2129 pixels

オーバースキャン・プリスキャン領域



サンプル画像

- (画像置場)/sample1/
  - obslog.txt : 画像取得時の詳細情報テキストログ
  - HP00xxxxx_p.fits : 画像ファイル
    - xxxxx : 通し番号
    - p : CCDのチップ番号 (0 又は1)
  - 同梱画像
    - 54139～54141 : 標準星 PG1530+057 画像(V-band; 30 sec x3)
    - 53661～53670 : フラット画像 (V-band; 5 sec x 10)
    - 54168～54177 : バイアス画像 (0 sec x 10)
    - flat_V_1_tr.fits : チップ1・port2部分を切り出したflat画像。
    - flat_V_[01].fits : 切り出し前のflat画像

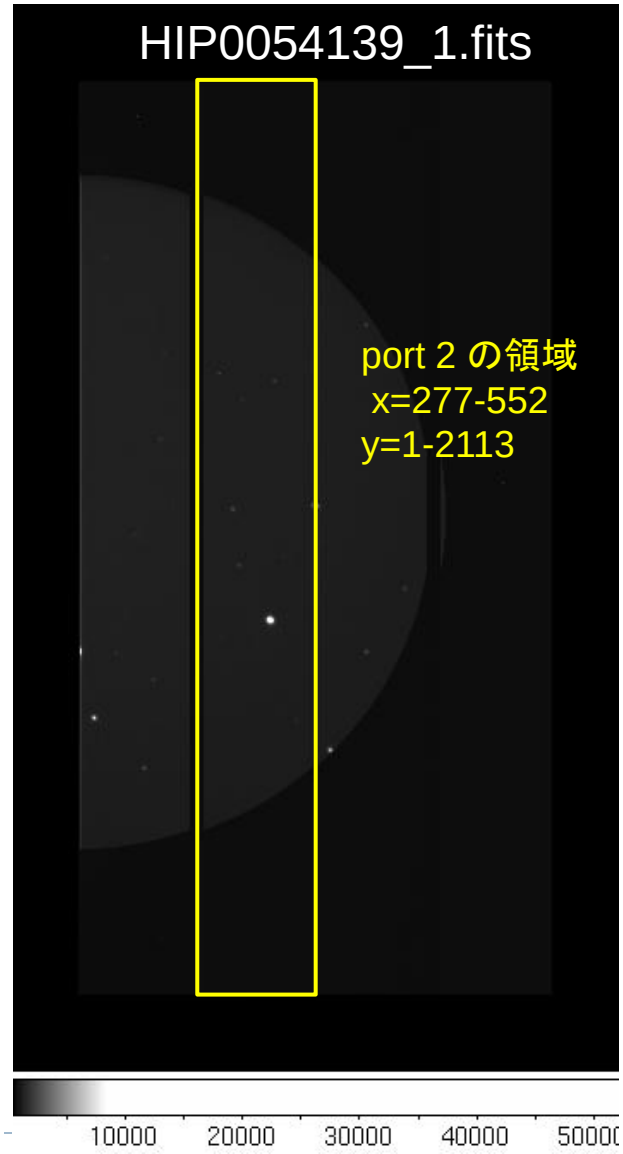
今日の実習の練習問題

- 標準星PG1530+057のV-band画像中、読み出しport 2の部分について、
  1. 画像領域の切り取り
  2. バイアスレベルの引き算
  3. フラット補正

  (いずれも天体画像処理の基本的な一次処理)

を行います。

これを通じて、IRAF基本操作、スクリプト化を覚えていきましょう。



一次処理練習：0. 画像準備

- 作業ディレクトリの準備

  ecl> cd /adc/data/guest??/iraf/ ⏎

  ecl> mkdir work1 ⏎

  ecl> cd work1 ⏎

- 画像の準備

  ecl> imcopy (画像置場)/HP0054139_1.fits . ⏎

  # 標準星画像の「1枚」をコピー

  #(最後の"ピリオド(=現在のディレクトリを指示)"を忘れずに

  ecl> imcopy (画像置場)/flat_V_1_tr.fits . ⏎

  # flat画像をコピー

- Let's try
  - 画像のヘッダーの内容を調べてみよう。

  ecl> imhead HP0054139_1.fits lo+ ⏎

(役立ちタスク: imheader)

画像取得時の情報は(よく整備された装置であれば)FITSヘッダーに網羅されている。かならず参照するべき情報。

- 基本情報の表示

  ecl> imhead HP0054139_1.fits ⏎

  HP0054139_1.fits[1104,2129][ushort]: pg1530+057

- 詳細表示 (オプション long+)

  ecl> imhead HP0054139_1.fits lo+ ⏎

```
HP0054139_1.fits[1104,2129][ushort]: pg1530+057
No bad pixels, min=0., max=0. (old)
Line storage mode, physdim [1104,2129], length of user area 7047 s.u.
Created Mon 10:28:53 16-Jul-2012, Last modified Mon 10:28:53 16-Jul-2012
Pixel file "HP0054139_1.fits" [ok]
EXTEND  =                    T / File may contain extensions
BSCALE  =           1.000000E0 / REAL = TAPE*BSCALE + BZERO
BZERO   =           3.276800E4 /
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
DATE    = '2012-07-16T10:28:53' / Date FITS file was generated
IRAF-TLM= '2012-07-16T10:28:53' / Time of last modification
OBJECT  = 'pg1530+057'         / Name of the object observed
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
DETID   = '1       '           / Detector ID
SPV     = 'read_2x2'           / spv_com pattern name
OBSERVER= 'Yamanaka,Itoh'      / Name(s) of observer(s)
INSTRUME= 'HOWPol  '           / Name of intrument
EXP-ID  = 'HP0054139'          / Exposure sequential number
FRAMEID = 'HP0054139_1'        / Frame identification number
DATA-TYP= 'OBJECT  '           / OBJECT, BIAS, FLAT, DARK, COMPARISON, etc
DATE-OBS= '2012-01-16'         / Observation start date in UT (yyyy-mm-dd)
UT      = '19:52:26'           / Typical UTC at exposure (hh:mm:ss)
UT-STR  = '19:52:26'           / UTC at exposure start (hh:mm:ss)
UT-END  = '19:52:57'           / UTC at exposure end (hh:mm:ss)
JST     = '04:52:26'           / Typical JST at exposure (hh:mm:ss)
MJD     =         55942.828258 / Modified Julian Day at exposure center
MJD-STR =         55942.828079 / Modified Julian Day at exposure start
MJD-END =         55942.828438 / Modified Julian Day at exposure end
RA      = '15:33:10.00'        / RA of the target base position
DEC     = '+05:33:45.0'        / DEC of target base position
EQUINOX =               2000. / Equinox of RA and DEC
EXPTIME =              30.014 / Exposure time [s] including shutter offset
EXPTIME0=                30. / Exposure time [s] set by observer
OBSERVAT= 'Higashi-Hiroshima'  / Name of observatory
LONGITUD=    1.3277670000E+02 / Longitude of observatory
LATTITUD=    3.4404200000E+01 / Lattitude of observatory
TELESCOP= 'Kanata_1.5-m'       / Name of telescope
TELD-STR= '1326743546 2012-01-17 04:52:26' / Time of Tel Log at exp. start
TELD-END= '1326743577 2012-01-17 04:52:57' / Time of Tel Log at exp. end
JSTT-STR= '04:52:26.90'        / JST nearly at exposure start (Tel Log)
JSTT-END= '04:52:58.00'        / JST nearly at exposure end (Tel Log)
MJDT-STR=              55943. / MJD nearly at exposure start (Tel Log)
MJDT-END=              55943. / MJD nearly at exposure end (Tel Log)
LST     = '12:26:12.00'        / Typical local sidereal time (Tel Log)
LSTT-STR= '12:26:12.00'        / LST nearly at exposure start (Tel Log)
LSTT-END= '12:26:43.20'        / LST at exposure end (Tel Log)
HA      = '-03:06:44.60'       / Typical hour angle
HA-STR  = '-03:06:44.60'       / Hour angle nearly at expsure start
HA-END  = '-03:06:13.40'       / Hour angle nearly at expsure end
HA-DEG  =       -3.1080555556 / Typical hour angle in degree
```

さらに続く……

一次処理練習：1. port 2 画像領域切り取り

- port 2 画像領域(x=277-552; y=1-2113)切り取り

  ecl> imcopy HP0054139_1.fits[277:552,1:2113] HP0054139_1_tr.fits ⏎

  → "xxxx.fits[x1:x2,y1:y2]" ("~.fits" と "["の間はくっつける)とすることで、画像xxxx.fits中の一部分x=x1～x2, y=y1～y2を領域指定できる。これを、HP0054139_1_tr.fits ("tr"は"trimming"の意を込めた)というファイルにコピーする。

- Let's try!
  - ds9 で表示して切り取った画像を確認してみよう。
  - 画像サイズをtask: imhead で確認してみよう。

    ecl> imhead HP0054139_1.fits,HP0054139_1_tr.fits ⏎

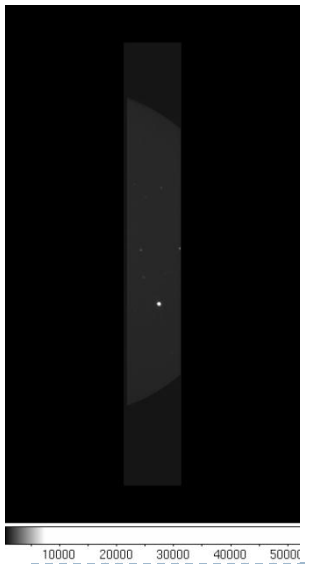
    #注意: ","の前後にスペースは入れないこと。

    HP0054139_1.fits[1104,2129][ushort]: pg1530+057

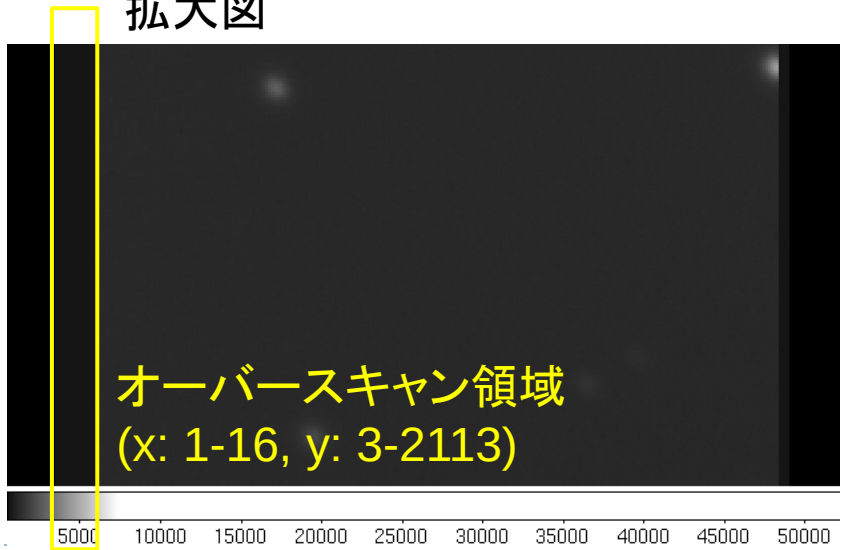
    HP0054139_1_tr.fits[276,2113][ushort]: pg1530+057

一次処理練習：2. バイアス差し引き (1)

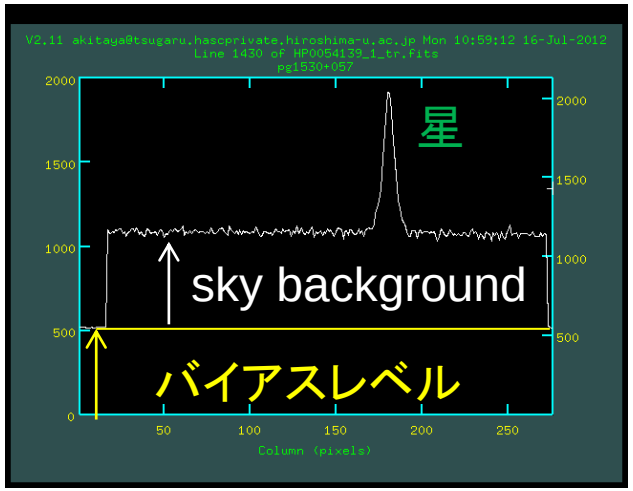
- 画像全体には、CCDの電圧に起因する人工的な信号値(バイアス値)が加算されている。これは、天体の画像領域から差し引く必要がある。
- この値は、オーバースキャン領域(光を当てずに読み込んだ画素の信号値から知ることができる。
- よって、以下が必要。
  1. オーバースキャン領域のカウント値を求める。
  2. その値を、画像全体から差し引く。



拡大図



断面図



(役立ちタスク: implot)

ecl> implot HP0054139_1_tr.fits ⏎

- 断面グラフが表示される。グラフ中にカーソルを合わせて、

:l 1100 1200 ⏎　　# y=1100-1200の平均断面表示

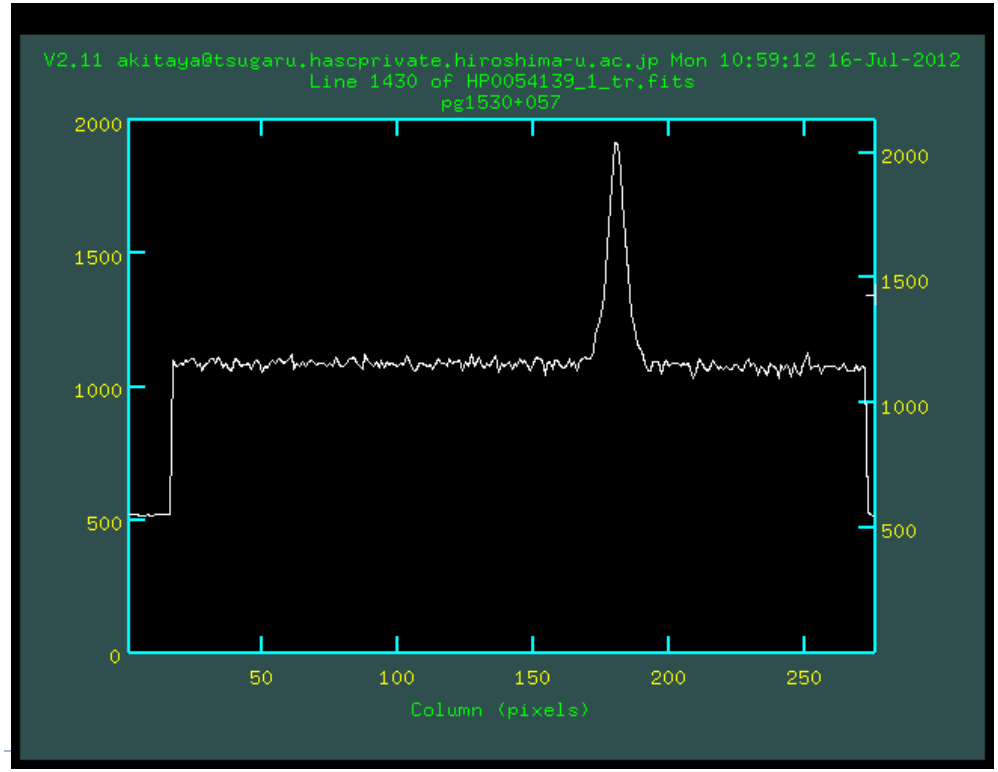
:c 300 310 ⏎　　# x=300-310 の平均断面表示

:x 100 200 ⏎　　# 横軸表示範囲を100-200に設定

:y 0 2000 ⏎　　# 縦軸表示範囲を0-2000に設定

?　　# help 表示

q　　# 終了



一次処理練習：2. バイアス差し引き (2)

1. バイアス値を、オーバースキャン領域[1:16,3:2113]の信号値の「中間値(median)」をもとに調べる(後述 imstatistics解説も参照)。

ecl> imstat HP0054139_1_tr.fits[1:16,3:2113] format- field="midpt" ⏎
521.0192

2. バイアス値を画像全体から差し引く (後述 imarith 解説も参照)

ecl> imarith HP0054139_1_tr.fits - 521.0192 HP0054139_1_bs.fits ⏎
バイアス値を引いた画像作成。"bs"~bias subtracted。

- Let's try!
  - 画像をds9で表示して、カウントをスキャンしてバイアスが引かれたことを確認してみよう。
  - implotで断面を見てみよう。

(役立ちタスク: imstatistics)

- 画像信号値の統計量表示

ecl> imstat HP0054139_1_tr.fits ⏎

```
#                IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
 HP0054139_1_tr.fits    587604     849.1     469.2      497.    52226.
```

- 覚えておきたいオプション

1. field="....."

ecl> imstat HP0054139_1_tr.fits field="image, mean, midpt" ⏎

```
#                IMAGE      MEAN     MIDPT
 HP0054139_1_tr.fits     849.1     1007.
```

→ "画像名(image)、平均値(mean)、中間値(midpt)"のみ表示。
選択できる値の種類は、ecl> help imstat で。

2. format-

ecl> imstat HP0054139_1_tr.fits field="mean" format- ⏎

849.1331

→ 凡例表示 ("# IMAGE ...")を抑制。高精度表示。値を変数に読み込むとき便利。

(役立ちタスク: imarith)

- 画像どうし、画像と数値の演算処理 (画像名は適当)

  ecl> imarith image1.fits + image2.fits image3.fits ⏎

  → image1.fitsとimage2.fitsを足し算して image3.fitsを出力。

  ecl> imarith image3.fits * 2.5 image4.fits ⏎

  → image3.fitsの全体に数値2.5を乗じた画像image4.fitsを出力。
- 使える演算子
  - +, -, *, /, min, max
- オプション "ver+" をつけると、詳しい演算内容もテキスト出力される。
- 関連するtask として、"imfunction" (log10, sqrt, ...が可能)もある。(詳細は help imfunction で)。

一次処理練習：3. フラット補正

- 天体画像を、一様光を照射して別途撮像した画像(フラット画像)で割り、装置・検出器の画像内の照射効率むらを補正する。

  ecl> imarith HP0054139_1_bs.fits / flat_V_1_tr.fits HP0054139_1_fl.fits ⏎

- Let's try!
  - 画像をds9で表示して、バイアス引き画像との相違を比較してみよう。
    - ds9で、「Frame」→「New Frame」、「Frame」→「Tile Frame」として、複数のFrameを開くと、同時に比較できる。「Frame」→「Blink Frames」だと画像を交互に表示。複数画像の位置合わせは「Frame」→「Match」→「Frame」→「Image」

一時処理まとめ(ファイル名直書き)

1. 切り取り

   imcopy HP0054139_1.fits[277:552,1:2113] HP0054139_1_tr.fits
2. バイアス値調査

   imstat HP0054139_1_tr.fits[1:16,3:2113] format- field="midpt"
   521.0192
3. バイアス値引き

   imarith HP0054139_1_tr.fits - 521.0192 HP0054139_1_bs.fits
4. フラット補正

   imarith HP0054139_1_bs.fits / flat_V_1_tr.fits
   HP0054139_1_fl.fits

- たった4行。天体画像は1枚だけ。ファイル名直打ちでも大した苦労はない。
- しかし、大量の天体画像がある場合は?

リストファイルを使った複数画像の連続処理

- 先ほどは、天体画像「1枚」への処理。今度は「複数(3枚)」への効率の良い処理を行ってみましょう。リストファイルを使います。

- 目指すのは以下のようなもの

  imarith image1.fits / flat.fits image2.fits

  (単一ファイル) → (単一ファイル)

  ↓

  imarith @images1.lst / flat.fits @images2.lst

  (複数ファイルのリスト) (複数ファイルのリスト)

リストファイルを使った処理: 環境・ファイル準備

- 別の作業ディレクトリを用意する。
  (前の作業の残骸があるといろいろ面倒なので)

ecl> mkdir home$work2 ⏎
  # IRAFの"homeディレクトリの下にwork2ディレクトリを作る)
ecl> cd home$work2 ⏎

- 必要ファイルのコピー

ecl> cp (ファイル置場)/sample1/HP00541[34]?_1.fits . ⏎
  # 標準星画像「3枚」をコピー (ファイル番号の十の
  # 位が3か4で、かつ、一の位が任意のchip 1の画像
  #ファイルを指示)
  # "[34]"の表現はimcopyでは使えないので、
  # ここではUNIXのコピーコマンド cp を使っている。
ecl> imcopy (ファイル置場)/sample1/flat_V_1_tr.fits . ⏎
  # flat画像をコピー

リストファイルを使った処理: リストの作成(1)

- 基本となるリストファイルの作成

  ecl> files HP00541*_1.fits > obj.lst ⏎

- 確認

  ecl> type obj.lst ⏎

  HP0054139_1.fits

  HP0054140_1.fits

  HP0054141_1.fits

- 領域切取済画像名のリスト、バイアス引画像名のリスト、フラット処理済画像名のリストをそれぞれ作成

  ecl> !sed 's/.fits/_tr.fits/' obj.lst > obj_tr.lst ⏎

  ecl> !sed 's/.fits/_bs.fits/' obj.lst > obj_bs.lst ⏎

  ecl> !sed 's/.fits/_fl.fits/' obj.lst > obj_fl.lst ⏎

リストファイルを使った処理: リストの作成(2)

- 確認
  - リストファイルを確認してみよう。

ecl> !paste obj.lst obj_tr.lst ⏎

#"paste" テキストファイルを並べて表示するUNIXコマンド

HP0054139_1.fits　　HP0054139_1_tr.fits

HP0054140_1.fits　　HP0054140_1_tr.fits

HP0054141_1.fits　　HP0054141_1_tr.fits

(役立ちUNIXコマンド sed)

ecl> !sed 's/文字列1/文字列2/' textfile.txt ⏎

textfile.txt中の文字列1を文字列2に変換して出力する。

※ 本講習会講師はsedばかり使いますが、awkの方がより高機能(らしい)です。お好きな文字処理コマンドをお使いください。

リストファイルを使った処理: リストの作成(3)

- 領域指定も含めたリストが必要

```
ecl> !sed 's/.fits/.fits¥[277:552,1:2113]/' obj.lst > obj_region.lst ⏎
ecl> !sed 's/.fits/.fits¥[1:16,3:2113]/' obj_tr.lst > obj_tr_region.lst ⏎
                    # "["にくっついている¥は、"["を特殊文字とみなさない
                    #ためのエスケープ処理
```

- 確認

```
ecl> cat obj_region.lst ⏎
HP0054139_1.fits[277:552,1:2113]
HP0054140_1.fits[277:552,1:2113]
HP0054141_1.fits[277:552,1:2113]
```

- これでだいたい準備は整いました。

リストファイルを使った処理: 実行(1)

- 「一時処理まとめ」の頁と比較しつつ実行してみましょう。

1. 切り取り

ecl> imcopy @obj_region.lst @obj_tr.lst ⏎
HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits
HP0054140_1.fits[277:552,1:2113] -> HP0054140_1_tr.fits
HP0054141_1.fits[277:552,1:2113] -> HP0054141_1_tr.fits
　　# 3枚分、一気に処理が実行される。

2. バイアス値調査

- ここでは、「数値のリストファイル」を活用します。

ecl> imstat @obj_tr_region.lst field="midpt" format- > ovscan.xy ⏎
　　# imstatの出力結果を、リダイレクトを使って"ovscan.xy"に書き出し。

- 確認すると、

ecl> cat ovscan.xy ⏎
521.0192
520.494
520.777

リストファイルを使った処理: 実行(2)

3. バイアス値引き

ecl> imarith @obj_tr.lst - @ovscan.xy @obj_bs.lst ⏎

4. フラット補正

ecl> imarith @obj_bs.lst / flat_V_1_tr.fits @obj_fl.lst ⏎

- これで、3枚の天体生画像について、フラット処理までの一時処理が完了。

ecl> ls HP*fl*fits ⏎

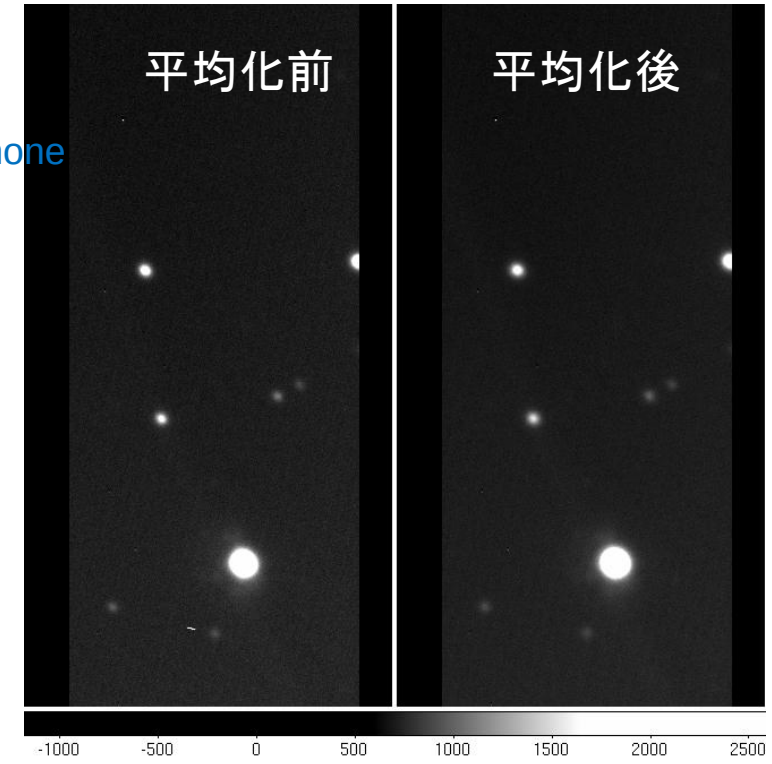
HP0054139_1_fl.fits HP0054140_1_fl.fits HP0054141_1_fl.fits

複数画像の平均化

- 折角なので、3枚の天体画像を平均化してみましょう。平均化アルゴリズムには中間値を用いてみましょう。

ecl> imcomb @obj_fl.lst object_fl_ave.fits combine="median" ⏎

```
Jul 16 13:40: IMCOMBINE
  combine = median, scale = none, zero = none, weight = none
  blank = 0.
                Images
    HP0054139_1_fl.fits
    HP0054140_1_fl.fits
    HP0054141_1_fl.fits

  Output image = object_fl_ave.fits, ncombine = 3
```



- Let's try!
  - 平均化前と平均化後の画像をds9で比較してみよう。

(役立ちタスク: imcombine)

- 画像の平均化(又は類似処理)を行う

ecl> imcombine HP*_bs.fits average.fits ⏎

- 覚えておきたいオプション
  1. combine="..."
     - 演算処理の種類を指定する。平均(average), 中間値(median)、加算(sum)があり。
  2. reject="..."
     - 複数画像間で、外れ値を除去するアルゴリズムを指定する。宇宙線除去などで活躍。
     - なし(none)、σ-clip(sigclip)、CCD gain情報を用いた除去(ccdclip)など。方法によっては、他のパラメターの適切な設定も必要。
  - rejmask="mask.fits"
    - rejectアルゴリズムで外れ値除去が行われたピクセルを記録できる。
  1. sigma="stddev.fits"
     - 各pixelの標準偏差画像を出力する。例えばbias画像から検出器読み出しノイズの分布を調べるときなどに重宝。

一時処理まとめ(リストファイル使用)

- リストファイル群を用意した上で、

1. 切り取り

```
imcopy @obj_region.lst @obj_tr.lst
```

2. バイアス値調査

```
imstat @obj_tr_region.lst field="midpt" format- > ovscan.xy
```

3. バイアス値引き

```
imarith @obj_tr.lst - @ovscan.xy @obj_bs.lst
```

4. フラット補正

```
imarith @obj_bs.lst / flat_V_1_tr.fits @obj_fl.lst
```

- 4行で、複数の画像をまとめて処理できた。
  (p.47 の4行と比較してみましょう)

4. CL scriptの作成と実行

CL script

- IRAF コマンド、制御構文、タスク、外部プログラム(含むUNIXコマンド)の一群を記述したテキストファイル。
- IRAF タスクとして実行できる。

- CL scriptの使い方
  1. スクリプトファイルを作成する 。拡張子=cl(xxxxxxx.cl)。
  2. task として登録する。
  3. 実行する

CL script: とにかく実行してみよう

- サンプルclスクリプトコピー

  ecl> cd home ⏎ #IRAFホームディレクトリに移動

  ecl> cp (データ置き場) /sample2/sample1.cl . ⏎

- taskの登録

  ecl> task sample1 = sample1.cl ⏎

- 実行

  ecl> sample1 NAOJ ⏎

  #適当な文字列引数(ここでは "NAOJ")を一つ与える。

  Text : NAOJ

  5.500 x 2.700 = 14.850

  1 2 3 4 5 6 7 8 9 10

CL scriptの基本構文：sample1.cl

```
#
# CL script sample1
#    (with arguments)
#      2012/7/17 H. Akitaya
#
```

(1) コメント行

```
procedure sample1( text1 )
```

(2) スクリプトの宣言 (必須)

```
string text1 { prompt = "Input :" }
```

(3) 引数(ある場合)の変数宣言

```
# main routine
begin  (必須)

real a, b
int i

# main
a = 5.5
b = 2.7
```

(4) メインルーチン

(5) メインルーチン中の変数宣言

```
printf("Text : %10s ¥n", text1 )
printf("%7.3f x %7.3f = %7.3f ¥n", a, b, a*b  )

for( i=1; i<=10; i+=1 ) {
    printf( "%d ", i )
}
printf("¥n")
  (必須)
bye
  (必須)
end

# end
```

構文ルールいくつか：基本構成(1)

- "#"で始まる行はコメント行 (1)
- スクリプトの宣言(正式には"CL procedure"の宣言)文を先頭に記述 (2)

  procedure sample1(text1)
  - "sample1"というタスク。引数は "text1"という変数

  ※引数が無いscriptの場合は、

  procedure sample1

  のように記述する。
- 引数 (ある場合は)の変数型を宣言する(procedure直下で!)。(3)

  string text1 { prompt = "Input :" }
  - 変数“text1”は文字列型。起動時に省略されたら “Input:” という文字列を表示して、対話的に入力を促す。(補足ページ参照)

  ※ 引数のないscriptでは不要

構文ルールいくつか：基本構成(2)

- メインルーチンは、beginで始め、bye、endで終わる(4)。メインルーチン内で使う変数は、『全て』begin直下にまとめて宣言しておく (5)

```
begin
        real a, b
        int i
        # main routine
bye
end
```

注) 変数宣言は、『全て』begin直下にまとめること(beginと変数宣言の間には、いかなるコマンド・制御構文・タスク等の実行行も含まれていてはならない！)。
スクリプトの途中で変数が必要になったからといって、その箇所に宣言文を紛れ込ませると、スクリプト自体が動きません。

- 最終行に、コメント行を何か入れておく。

```
# end
```

  - 必須ではない。しかし、CL scriptでは、最後の実行可能行(sampleでは、end)のあとに「改行コードが」入っていないと、スクリプト自体が動かない場合があるため、こうしておくとそのミスを防げる。

構文ルールいくつか：基本構文(1)

- 文字列出力 :printf(" ... ", arg1, arg2, ...)
  - C言語のprintfとほぼ同じ。
    - ""内に表示文字列と制御文字列(%～)、後に引数を列挙。

printf("Text : %10s ¥n", text1)

"%10s": 後で与える変数を、10桁(="10")の文字列(="s")で表示する。

"¥n": 改行コード

printf("%7.3f x %7.3f = %7.3f ¥n", a, b, a*b)

"%7.3" : 7桁(うち小数点以下3桁)の実数表示。

構文ルールいくつか：基本構文(2)

- 制御構文(1) for ループ

```
for( i=1; i<=10; i+=1 ) {
    printf( "%d ", i )
}
```

  - i=1で始まり、一回のループごとに iを+1加算。iが10以下の間、{ }内を実行。
  - 制御構文の対象は、{ }でくくる。
  - 詳細は、help language.forで。

構文ルールいくつか：基本構文(3)

- sample2.clをtask登録・実行してみてください。
  - sample2の引数は、実数2つです。
  - 実行例

```
ecl> task sample2 = sample2.cl
ecl> sample2 5.6 6.7
                #複数の引数は、スペースで区切って入力。
```

- 制御構文(2) 条件分岐

```
if( a == b ) {
        print ("a=b¥n")
}else{
        print ("a != b¥n" )
}
```

if (条件文)で条件文を判断し、真ならば直下の{}内を実行。
偽ならば、else以下の{}内を実行。

詳細は help language.if で。

構文ルールいくつか：基本構文(4)

- 記述が複数行にまたがる場合は、行末に"¥"を記して、続きを次行に書く。

```
if( a > b && ¥
        b < 0 ){
        print("Hey !!¥n")
}
```

  - a > b かつ b< 0ならば、文字列出力。

taskの登録: 引数を持つ場合・持たない場合(1)

- taskの登録方法は、引数を持つ場合と持たない場合で異なる。
- 引数を持つ場合 : sample3_w_args.cl

```
procedure sample3_w_args( input_fn , output_fn , value1 )
file input_fn     { prompt = "Input Image Name" }
file output_fn      { prompt = "Output Image Name" }
real value1       { prompt = "Input Value 1" }
begin
...

ecl> task sample3_w_args = sample3_w_args.cl ⏎
        # "task"のあとのタスク名宣言に"$"をつけない。
ecl> sample3_w_args in.fits out.fits 5.4 ⏎
# Test script with arguments.
# Input: in.fits
# Output: out.fits
# Value1:    5.400
```

taskの登録: 引数を持つ場合・持たない場合(2)

- 引数を持たない場合：sample3_wo_args.cl

  procedure sample3_wo_args

  begin

  ...

  ecl> task **$sample3_wo_args** = sample3_wo_args.cl ⏎

  　　# "task"のあとのタスク名宣言に"$"をつける。

  ecl> sample3_wo_args ⏎

  # Test script without arguments.

login.clへのtask登録記述 (1)

- taskの宣言 (task ...=....cl)は、CLの再起動ごとに必要。
- 頻繁に使うtaskは、宣言をlogin.clに記述しておけば便利。
- sample1.clを例に。

1. CL scriptが login.clと同じ場所にある場合。
   - login.clの、"package user"のあと、その後の"keep"の直前あたりに、以下task宣言文を記述。

   task sample1 = sample1.cl
2. CL scriptが他のディレクトリにある場合。ディレクトリパス付でclファイルを指定。

   task sample1 = /home/xxxx/iraf/myclscript/sample1.cl
   - IRAF ホームディレクトリの下に、myscript/ というディレクトリを作って、sample1.clを置いた場合。環境変数 home (login.clの最初に定義)を使って、次のように書いてもよい。

   task sample1 = home$myscript/sample1.cl

   →"home$"部分がIRAF ホームディレクトリの文字列に置き換わる。

login.clへのtask登録記述 (2) (講習後補足)

- IRAFのバージョン、taskの名前によっては、「login.clの、“package user”のあと、その後の“keep”の直前あたり」頻にtask 宣言文を記述すると、cl 立ち上げ時のlogin.cl 読み込み時にエラーが出てしまう場合があるようです。その場合は、login.cl の末尾に

  task sample1 = sample1.cl

  task sample2 = sample2.cl

  ...

  keep

  のように、task宣言文の羅列 + keep 文 (“keep”の後に改行を入れるのを忘れずに!)として記述してみてください。

  - “keep”文は、task宣言の内容を、login.clを読み込み終わったあとにも維持しておくために必須です。
- task宣言などの個人設定は、“loginuser.cl”という別ファイルに記述しておく流儀もあるようです(デフォルトのlogin.clでは、このファイルが存在すれば、そこから内容を読み込むように設定されています)。

画像解析CL script(1)：命令文の単純埋め込み

- コマンドラインの練習で行った、単一ファイル向け一時処理 4行

```
imcopy HP0054139_1.fits[277:552,1:2113]  ¥
        HP0054139_1_tr.fits
imstat HP0054139_1_tr.fits[1:16,3:2113] format- ¥
        field="midpt"
imarith HP0054139_1_tr.fits - 521.0192 ¥
        HP0054139_1_bs.fits
imarith HP0054139_1_bs.fits / flat_V_1_tr.fits ¥
        HP0054139_1_fl.fits
```

を、引数なしのCL スクリプトにしてみよう。

(バイアス値を最初から知っていることになっており、インチキが入ってますが、とりあえず気にしない。)

画像解析CL script(1)：命令文の単純埋め込み

- 引数を持たないCL scriptの定型パターン

```
procedure task名
begin
        # main routine
bye
end
#end
```

の、#main routine部分に、実行したコマンド群を書き込めばOK。

- 但し、若干記法を変更する必要あり。

command mode記法とprogram mode記法

- command mode記法 (コマンドラインで使える・procedure宣言したCL script 内では使えない)

  imcopy file1.fits file2.fits

  imarith file3.fits * file4.fits file5.fits

  imstat file6.fits format-

  - コマンド・タスクの後に、引数をスペースで区切って表記。
- program mode 記法 (procedureで宣言したCL script中で使える・コマンドラインで使っても良い)

  imcopy("file1.fits", "file2.fits")

  imarith ("file3.fits", "*". "file4.fits", "file5.fits")

  imstat("file6.fits", format-)

  - コマンド・タスクの後をカッコ"()"でくくり、引数をカンマ","で区切る。
  - 固有のファイル名は""でくくる。オプション・変数・数値はくくらない。
  - program modeでないと、変数と固有文字列の区別がつかない(難しい)

Program mode 記法での変数の効果

- 以下2つは、同じ結果となる。

1. case1 (変数不使用)

```
imarith( "file1.fits", "*", 123.45, "file2.fits")
        #具体的なファイル名は""でくくる。数値はくくらない
```

2. case2 (変数使用)

```
string filename1, filename2
real value
filename1 = "file1.fits"
filename2 = "file2.fits"
value1 = 123.45
imarith( filename1, "*", value, filename2)
        # 変数名は""でくくらない。変数がそれぞれ値に
        # 置き換わってimarithに与えられる。
        # もし "filename1" とくくってしまうと、"filename1"という
        # 名前の画像ファイルを指定することになってしまう。
```

(重要) CL script ではProgram mode記法を使う

- “procedude”宣言で始めるCL scriptでは、”program mode記法”しか使えない。

(正)

```
procedure test1
begin
        imcopy( “file1.fits”,  “file2.fits”)
...
```

(誤)

```
procedure test1
begin
        imcopy file1.fits file2.fits
...
```

- command mode記法による簡便的なCL scriptの記述方法・利用法もありますが、本講習では割愛します。

画像解析CL script(1)：命令文の単純埋め込み

- Let's try!: 前項の4行の命令を実行する、適当な名前のCL scriptファイル(xxxx.cl)を作成してみましょう。
- 命令記述は、全て”program mode 記法”にする必要があります。
- 作例: redsample1.cl

```
#
# reduction sample 1
#    very simple case
#
#

procedure redsample1

begin

imcopy( "HP0054139_1.fits[277:552,1:2113]", "HP0054139_1_tr.fits" )
imstat( "HP0054139_1_tr.fits[1:16,3:2113]",  format-, field="midpt" )
imarith( "HP0054139_1_tr.fits", "-", 521.0192, "HP0054139_1_bs.fits" )
imarith( "HP0054139_1_bs.fits", "/", "flat_V_1_tr.fits", "HP0054139_1_fl.fits" )

bye

end

#end
```

実行テスト

- 準備
  - 適当な新しいディレクトリ(例えば/adc/data/guest??/iraf/work3/)を作成し、HP0054139_1.fits, flat_1_tr.fits, 作成したCL script(redsample1.cl) をコピー
  - ecl> 上からそのディレクトリに移動
- タスクの宣言

  ecl> task $redsample1=redsample1.cl ⏎　# "$"付なのに注意！
- 作成したCL scriptを、redsample1 というタスクとして実行。

  ecl> redsample1 ⏎

  HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits

  521.0192

  ecl> ls HP0054139_1*.fits ⏎

  HP0054139_1.fits HP0054139_1_bs.fits HP0054139_1_fl.fits
  HP0054139_1_tr.fits

- コマンドラインと同じ処理が実現できた！(エレガントではないけれど)

画像解析CL script(2) :任意画像の処理

- 次の機能を搭載して、汎用性を持たせてみましょう。
  - 引数として、文字列で5桁の数値"xxxxx" を与えると、画像"HP00xxxxx_1.fits"について一次処理する
  - flat画像を引数で指定する。
  - オーバースキャン領域を、ちゃんと画像ごとに測定して、その後の差引に反映させる。(インチキなし!)
  - 不要な途中ファイルを削除する。
  - 出力ファイルの有無を確認する。
- 引数付きのCL script 定型パターンを使えばOK。
- 新しく覚えることは、
  1. 文字列と変数の連結
  2. コマンド・タスクの出力を変数に格納する
  3. ファイルの有無の確認

文字列と変数の連結(1)

- 次のように文字列変数と""で括った文字列を"//"で接続すれば、両者を連結した新しい文字列を作成できる。

```
ecl> string number ⏎
ecl> string extension ⏎
ecl> number="12345" ⏎
ecl> extension="_1.fits" ⏎
ecl> print( "HP00"//number//extension ) ⏎
HP0012345_1.fits

ecl> string newstr⏎
ecl> newstr = "HP00"//number//extension ⏎
ecl> print( newstr ) ⏎
HP0012345_1.fits
```

文字列と変数の連結(2)

- 数値変数と文字列の接続もOK

  ecl> int x1, x2, y1, y2 ⏎

  ecl> x1=1; x2=100; y1=1; y2=100 ⏎

  ecl> imstat("flat_V_1_tr.fits["//x1//":"//x2//","//y1//":"//y2//"]") ⏎

  # IMAGE NPIX MEAN STDDEV MIN MAX

  flat_V_1_tr.fits[1:100,1:100] 10000 0.00232 3.087E-4 0.001525 0.00352

  → imcopy, imstatでの任意の領域設定に応用可能。

- 文字列処理は、他にも多彩な関数あり。

  help language.stringで。

コマンド・タスクの出力を変数に格納(1)

- コマンド・タスクの出力を、パイプ"|" で別の関数に流す。これを、scanf関数で変数に代入する。

```
ecl> imstat ("HP0054139_1.fits", format-, field="midpt" ) ⏎
536.3184
        # ただ実行すると画面に表示されるだけ。
ecl> real bias ⏎
ecl> imstat ("HP0054139_1.fits", format-, field="midpt" ) ¥ ⏎
                                        | scanf("%f", bias ) ⏎
ecl> print( bias ) ⏎
536.3184
        # パイプを使って、scanfで読むと、値がreal型
        # 変数 biasに格納された。
        # scanfの形式は、printfと類似。
        #「実数型"%f"の変数を一つ読み、それを、変数bias
        #  に格納する」
```

コマンド・タスクの出力を変数に格納(2)

- scanfは複数の出力も扱える。

```
ecl> imstat ("HP0054139_1.fits", format-, ¥ ⏎
                    field="midpt, mean" ) ⏎
536.3184  746.186
ecl> real median, mean ⏎
ecl> imstat ("HP0054139_1.fits", format-, ¥ ⏎
        field="midpt, mean" ) | scanf("%f %f", median, mean ) ⏎
ecl> print( median, mean ) ⏎
536.3184  746.186
```

出力ファイルの有無を確認

- CL scriptの中で以下のような構文が使える。

```
string fn_out
fn_out = "output.fits"
if( ! access( fn_out ) ){
        imcopy( "test.fits", fn_out )
}else{
        printf("# Error: output file %s exists.¥n", fn_out)
}
```

- 詳細は、help accessで。

画像解析CL script(2) :任意画像の処理

- Let's try! : スクリプトを作成して実行してみましょう。
- 作例 : redsample2.cl

```
#
# reduction sample 2
#    processing arbitrary number image
#
#

procedure redsample2( img_number, fn_flat )

string img_number { prompt= "image number (xxxxx) :" }
string fn_flat { prompt= "flat filename : " }

begin

string fn_org, fn_trimed, fn_biassub, fn_flattened
real bias

fn_org="HP00"//img_number//"_1.fits"
fn_trimed="HP00"//img_number//"_1_tr.fits"
fn_biassub="HP00"//img_number//"_1_bs.fits"
fn_flattened="HP00"//img_number//"_1_fl.fits"

if( access( fn_trimed ) ){
    imdelete( fn_trimed )
}
imcopy( fn_org//"[277:552,1:2113]", fn_trimed )

imstat( fn_trimed//"[1:16,3:2113]",  format-, field="midpt") | scanf(¥
 "%f", bias )
print( bias )

if( access( fn_biassub ) ){
    imdelete( fn_biassub )
}
imarith( fn_trimed, "-", bias, fn_biassub )

if( !access( fn_flattened ) ){
    imarith( fn_biassub, "/", fn_flat, fn_flattened )
}else{
    printf("# Error: %s exists !¥n", fn_flattened )
}

# cleaning
imdelete( fn_trimed )
imdelete( fn_biassub )

bye

end

#end
```

画像解析CL script(2) :任意画像の処理

- 実行例 (ファイル準備は省略; redsample1.clの実行時と同様に新しい作業場所を作ると良い)

```
ecl> task redsample2=redsample2.cl ⏎
ecl> redsample2 54139 flat_V_1_tr.fits ⏎
HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits
521.0192
ecl> ls HP0054139_1*.fits ⏎
HP0054139_1.fits  HP0054139_1_fl.fits
```

→ 途中ファイルは残っていない

```
ecl> redsample2 54139 flat_V_1_tr.fits ⏎
HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits
521.0192
# Error: HP0054139_1_fl.fits exists !
```

→ 2回目は、ファイルが存在しているので警告が出た。

画像解析CL script(3) : 複数のファイルを受け付ける

- redsample2.clでは、単一の被処理ファイルのみを指定できた。
- 複数のファイルを、リストファイルやワイルドカード表現で指定できるようにする。

  → これができれば、最強。
- 次項に作例と実行例のみ示します。

画像解析CL script(3) : 複数のファイルを受け付ける

- 作例: redsample3.cl

```
#
# reduction sample 3
#    processing arbitrary number image
#      accept list file, wild card expressions
#

procedure redsample3( in_images, fn_flat )

string in_images { prompt= "images (xxxxx) :" }
string fn_flat { prompt= "flat filename : " }

struct *imglist

begin

string fn_org, fn_trimed, fn_biassub, fn_flattened
string fn_header
string imgfiles
real bias

imgfiles = mktemp( "_redsample3_tmp" )

sections( in_images, option="fullname", > imgfiles )

imglist = imgfiles
while( fscan( imglist, fn_org ) != EOF ){
  fn_header = substr( fn_org, 1, strlen( fn_org )-strlen(".fits") )

  fn_trimed=fn_header//"_tr.fits"
  fn_biassub=fn_header//"_bs.fits"
  fn_flattened=fn_header//"_fl.fits"

  if( access( fn_trimed ) ){
     imdelete( fn_trimed )
  }
  imcopy( fn_org//"[277:552,1:2113]", fn_trimed )

  imstat( fn_trimed//"[1:16,3:2113]",  format-, field="midpt") ¥
                                       | scanf( "%f", bias )
  print( bias )

  if( access( fn_biassub ) ){
     imdelete( fn_biassub )
  }
  imarith( fn_trimed, "-", bias, fn_biassub )

  if( !access( fn_flattened ) ){
     imarith( fn_biassub, "/", fn_flat, fn_flattened )
  }else{
     printf("# Error: %s exists !¥n", fn_flattened )
  }

  # cleaning
  imdelete( fn_trimed )
  imdelete( fn_biassub )
}
delete( imgfiles )

bye
end
#end
```

画像解析CL script(3) : 複数のファイルを受け付ける

- 実行例

  ecl> ls HP*fits ⏎

  HP0054139_1.fits HP0054140_1.fits HP0054141_1.fits

  　　#標準星の画像3枚が存在

  のとき、

  ecl> task redsample3=redsample3.cl ⏎

1. ワイルドカード表記で

  ecl> redsample3 HP00541*_1.fits flat_V_1_tr.fits ⏎

  HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits

  521.0192

  HP0054140_1.fits[277:552,1:2113] -> HP0054140_1_tr.fits

  520.494

  HP0054141_1.fits[277:552,1:2113] -> HP0054141_1_tr.fits

  520.777

画像解析CL script(3) : 複数のファイルを受け付ける

2. リストファイルで

```
ecl>  files HP00541*_1.fits > filelist1.lst ⏎
ecl> redsample3 @filelist1.lst flat_V_1_tr.fits ⏎
HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits
521.0192
HP0054140_1.fits[277:552,1:2113] -> HP0054140_1_tr.fits
520.494
HP0054141_1.fits[277:552,1:2113] -> HP0054141_1_tr.fits
520.777
```

- ポイント
  - ファイル名表記(ワイルドカード・リスト形式含む) を文字列変数で受け取る。
  - sections を使って、ファイル名の羅列をテキストファイルに書きだす。
  - ファイル名羅列のテキストファイル名を、リスト形式のstruct型変数imglistに渡し、whileループでファイル名を一つ一つ取り出して処理を行う。
  - 詳しくは、文末資料紹介にある"An Introductory User's Guide to IRAF Scripts" を参照。

(役立ちタスク: 一時ファイル作成・掃除：mktemp)

- 画像処理の途中経過を一時的に、適当な名前のファイルに保存しておきたいことがよくある。
- いちいち名前付けるのは面倒。
- 一時ファイルを作成しよう。

```
ecl> string tempfile1 ⏎
                        #一時ファイル名を保存する文字変数定義
ecl> tempfile1 = mktemp( "_testscript_tmp" ) ⏎
ecl> = tempfile1 ⏎
_testscript_tmp6610d
                        # ヘッダー文字+ランダム文字のファイル名文字列が生成された
ecl> imcopy ("bias.fits[101:200,101:200]",  tempfile1 ) ⏎ #ファイルコピーの例(画像名は適当)
bias.fits[101:200,101:200] -> _testscript_tmp6610d
ecl> ls ⏎
bias.fits  login.cl  _testscript_tmp6610d.fits  uparm
                        # "一時ファイル名 + .fits" のファイルが作られた。
          …
ecl>  imdelete( tempfile1 )  ⏎   # 使い終わったら消しておこう
```

- 消し忘れがあっても "_testscript_tmp"で始まるファイルは一時ファイルであるとすぐわかる。必要なファイルと区別して後で安全に消すことができる。

5. 複数のCL scriptの お手軽パッケージ化

自作CL scriptのパッケージ化(1)

- 自作CL scriptが溜まってきたら、login.clへの宣言も冗長になる。パッケージにまとめてみよう。
- 手順
  1. 置き場所を決めてディレクトリを作成する。ここでは、

     home$myscript

     にします。(実態は、home=/home/guest/iraf/のとき、/home/guest/iraf/myscript)

     ecl> mkdir home$myscript ⏎
  2. 作成したCL scriptを全て置き場所にコピーして移動。

     ecl> copy redsample?.cl home$myscript ⏎

     # cp ではなくcopy命令を使ったのは、irafの環境変数homeを

     # 使うため。

     ecl> cd home$myscript ⏎

自作CL scriptのパッケージ化(2)

3. パッケージ名.cl というclファイルを作成。
   ここでは、"redsample"というパッケージ
   名にする。内容は以下のようにする。

```
# Package "redsample"
# load necessary packages

set scriptdir= " home$myclscript/"

package redsample

task $redsample1 = scriptdir$redsample1.cl
task redsample2 = scriptdir$redsample2.cl
task redsample3 = scriptdir$redsample3.cl

clbye()
#
```

(1) 必要であれば、パッケージを読み出す
(パッケージ名の羅列でOK)

(2) scriptファイルの置き
場を環境変数に設定

(3) packageの宣言

(4) task 宣言を羅列

自作CL scriptのパッケージ化(3)

4. login.clにredsample.clのみを、引数なしタスクとして登録

```
...
if (access ("home$loginuser.cl"))
    cl < "home$loginuser.cl"
;                   # と書かれている下あたりに、

task $redsample= home$myscript/redsample.cl
```

- パッケージ読み込み例

```
$ cd ~/iraf/ ⏎
$ ecl ⏎
$ ecl> redsample ⏎
 redsample1  redsample2  redsample3

redsample>
```

(4) task 宣言を羅列

redsampleに登録されているタスク群が表示され、以後、そのまま使える。
プロンプトもredsample>に!

6. 総括

講習内容の確認

1. IRAFの基本設定
   - 環境設定、login.cl

1. コマンドプロンプトからの基本操作：撮像画像一次処理を通じて
   - 画像直接指定からリストファイルの活用へ

2. 簡単なCLスクリプトの作成と実行
   - 引数を用いた汎用スクリプトの作成

3. 複数のCL スクリプトのお手軽パッケージ化
   - 多数のスクリプトを一括登録・読み出し

触れられなかった話題・今後について

- 言語・CL script関連
  - ファイル入出力
  - 配列
- IRAF全体
  - 特有の観測モードの話題(測光、分光、、、)
  - taskとpackage構造
  - task parameterの管理

などなど。

- 今日の講習は、講師(秋田谷)のかなり偏ったIRAF知識に依っています。
  - この方がエレガントなのに、などは多数あるはず。
- 先人の残した資料(日本語・英語)が多数あります。ぜひ目を通してください。そして、他の人の良い技をどんどん盗みましょう。
  - そして、いい技があったら私にも教えてください・・・。

終

補足・資料・自由課題

CL script引数の対話的入力(1)

- 引数を持つCL script: redsample2.cl

```
procedure redsample2( img_number, fn_flat )
string img_number { prompt= "image number (xxxxx) :" }
string fn_flat { prompt= "flat filename : " }
begin
...
```

において、task登録後、

```
ecl> redsample2 ⏎
```

のように引数なしで実行した場合、

```
 image number (xxxxx) ::
flat filename ::
```

のようなpromptが出て対話的に順次変数内容の入力が行えるはずなのですが、実際は期待のように動作しません。

CL script引数の対話的入力(2)

- 実際の実行例：

```
ecl> redsample2 ⏎
image number (xxxxx) : 54139 ⏎
image number (xxxxx) : (54139): 54139 ⏎ # image number の
image number (xxxxx) : (54139): 54139 ⏎ # 値を、4回も
image number (xxxxx) : (54139): 54139 ⏎ # 問い合わせられる。
HP0054139_1.fits[277:552,1:2113] -> HP0054139_1_tr.fits
521.0192
flat filename : : flat_V_1_tr.fits ⏎
                        # ここでようやくflat画像名を聞かれる
ecl>                    # 一応正常には終了。
```

- どうやら、CL script中で変数が使われる度に、(すでに値を入力したかどうかにかかわらず)変数の内容の入力が求められてしまうようです。

CL script引数の対話的入力(3)

- これを回避するには、以下のようにすると良いようです。

```
procedure redsample2( _img_number, _fn_flat )
string _img_number { prompt= "image number (xxxxx) :" }
string _fn_flat { prompt= "flat filename : " }
                                # 引数受け取り専用の使い捨て変数
                                # (ここでは “_xxxxx...”)を宣言。
begin
string img_number
string fn_flat
                                # メインルーチンで使う変数を別途宣言
...                     #(他のメインルーチン用の変数宣言)

img_number = _img_number
fn_flat = _fn_flat
                        #  引数受け取り専用変数を、登場順にメインルーチン
                        # 変数に代入。(要は、ここで引数受け取り専用変数を「使い」、
                        # 対話的入力を促す。)
                        # 以後、引数受け取り専用変数は一切使わない。

...                     # メインルーチン
```

IRAF関連参考資料(1)

- 総合解説
  - 「IRAFクックブック第2版」(天文情報処理研究会編)
    - 基本から応用まで。現在は紙面でしか入手できませんが全IRAFユーザー必読。
  - IRAF 公式ページ: Recommended Documentation
    - http://iraf.nao.ac.jp/iraf/web/docs/recommend.html
    - 英語だが、基本が良くまとめられている。
- 入門者用情報
  - いらっしゃいませ IRAF へ ようこそ
    - http://hamalabo.sakura.ne.jp/Soft/iraf_beginners/
    - 入門者必読。
- 初中級者用チュートリアル
  - IRAF講習会資料 (国立天文台天文データセンター)
    - http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html
  - IRAF for beginners
    - http://www.astr.tohoku.ac.jp/~mikito/IRAF/
    - 測光・分光まで幅広く。Q&Aもあり。

IRAF関連参考資料(2)

- FAQs
  - IRAFノウハウ集(FAQ) (天文情報処理研究会)
    - http://jaipa.nao.ac.jp/iraf_knowhow.html
    - 古い情報もありますが有益です。
- CL script
  - IRAF 公式ページ: Recommended Documentation 内、"An Introductory User's Guide to IRAF Scripts" (compressed postscript)
    - http://iraf.nao.ac.jp/iraf/ftp/iraf/docs/script.ps.Z
    - cl script書く前に必読の一書。もし、あなたがCL scriptをそれなりに書く必要に迫られたならば、(英語に苦労してでも)遠回りして一読して損はないはずです。(本講習の内容もここに拠る部分多し)
- ちょっとしたテクニック
  - IRAF TIPS
    - http://home.hiroshima-u.ac.jp/akitaya/research/memo/iraftips.html
    - お役に立てば・・・

CL script関連で見ておきたいhelp

- コマンド入力の基本構文

  ecl> help command ⏎
- taskの定義

  ecl> help task ⏎
- task (CL procedure)宣言

  ecl> help language.procedure ⏎
- 文字列処理関数あれこれ

  ecl> help language.string⏎
- 文字列読み込み関数あれこれ

  ecl> help language.scan⏎
- 文字列表示関数あれこれ

  ecl> help language.print⏎

自由課題

- 過去のIRAF講習会資料の内容を実践してみよ。

  http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html

  - 本講習と特に関連が深いのは、
    - 2011年度 第1回「CL Script の基礎と実演 (川端 弘治 : 広島大学宇宙科学センター)」
    - 2010年度 第3回 「CL Script 入門 (古荘 玲子: 国立天文台 天文データセンター)」
    - 2009年度 第2回「CLスクリプトの作成 (吉田 道利 : 国立天文台　岡山天体物理観測所)」

    (敬称略。所属は当時)

自由課題

- LIPS用エシェル偏光分光データパイプライン解析パッケージ (配布場所/sample3/lipsred/lips_clscript/の中のclスクリプト群を眺めて、使ったことのないコマンドや命令、構文について調べて見よ。
  - lips.cl : パッケージ定義CL script
  - lipsinit.cl : 解析の初期設定用CL script
  - lipsred.cl : パイプライン順次実行用 CL script

    → lipsred.clに順次記述されているCL script群が、パイプラインの全体。

- LIPS
  - http://1601-031.a.hiroshima-u.ac.jp/lips/

自由課題

- バイアス画像：(#54168～54177)の10枚について
  1. 適当な画像1枚を選び、読み出しノイズ(ADU単位)を調べよ。それが、読み出しポートの番号によってどのように異なるかを示せ。(例えばimstatを使う)

適当なCCDチップ・読み出し領域を選び、次の特性を調べよ。

- 10枚にわたる、バイアス信号値の時間変化。画像番号と信号値の数値データファイルを作成せよ。
- その統計量(平均値と標準偏差)。 (IRAF コマンド"average"を使うと楽)

自由課題

- チップ1･port1 についても同様に解析してみよ。
  - フラット画像が必要。フラット画像の作成手順は以下の通り。

1. 画像番号53661～53670を準備する。
2. 全画像についてport1領域を切り取る。
3. 全画像についてbias成分を引く。
4. 全画像を平均化する。

(可能であれば、imcombineのrejection機能を用いて見よ。)

画像中、カウント一番大きな領域のおよその値を読み取り、画像全体をその値で割る(normalize)

以上

memo

- deleteは","指定で複数指定できない?
- file は予約語
- promptを正しく表示させるには、ダミー変数で受け取りすぐに正式変数に代入で受け渡す。
- task sample1=sample1.cl が、login.clの途中では通らない。