version 1.0

November 2010

PyRAF プログラミング入門

Y. Nakajima
Computer and Data Management Division
Subaru Telescope
NAOJ

Chapter 1

PyRAF入門

1.1 PyRAFのすすめ

PyRAF とは IRAF のタスクを Python から関数として利用できるようにするためのソフトウェアである。STScI が 1998 年から開発をしている。それには次のような動機があった。

(1) IRAF-CL ではデバッグやエラーハンドルが難しい。詳細なエラーレポートが出ないので、どこにバグがあるのかが非常にわかりにくい。
(2) CL-スクリプトなんていうのは光赤外天文屋しか使わない。もっと広く使われている言語を使えないか?
(3) IRAF 以外のソフトウエアと統合しやすくしたい

こう考えたときに、Python を wrapper として使うことが最善の選択であった。無料のオープンソースであること、ユーザーおよび開発者のコミュニティが深くて広くこれからも成長しそう、そして比較的容易な言語であることが理由である。

もし、これから自分でプログラムを組んでデータ処理/解析をゼロから始めようと思っている人がいて、IRAF を基本天文処理ソフトとして選ぶのであれば、

CL-スクリプトではなく PyRAF スクリプトを習得することを強くお勧めする。

「PyRAF スクリプトを習得する」とは書いたが、正確には「Python スクリプ

トを習得して、Python から IRAF タスクを呼ぶほんの少しのお約束を覚える」ということである。要は Python スクリプトにある。CL スクリプトとちがって、Python はより広く使われており、教科書もたくさんある。ネットで検索するとすぐに疑問が解決することもある。現在、世界の天文情報処理のコミュニティでは Python を使うのが大きな流れとなっている。覚えてて損はない。

1.2 PyRAF をコマンドラインで使う

1.2.1 基本

PyRAF は IRAF の CL のようにコマンドラインでインタラクティブに使うことができる。ターミナルで、`pyraf` と入力するといつもの IRAF のログインバナーとともに PyRAF が立ち上がる。なお、ホームディレクトリの`~/iraf` というディレクトリに login.cl があれば、PyRAF の起動はどのディレクトリにいても可能である。ホームディレクトリに `iraf` というディレクトリを作成して、`mkiraf` を実行する。IRAF-CL とは違い、プロンプトは `-->` である。それ以外は普通に IRAF-CL で使うように使えばよい。

```
--> imhead dev$pix long+
dev$pix[512,512][short]: m51  B  600s
No bad pixels, min=-1., max=19936.
Line storage mode, physdim [512,512], length of user area 1621 s.u.
Created Mon 23:54:13 31-Mar-1997, Last modified Sun 16:37:53 12-Mar
Pixel file "HDR$pix.pix" [ok]
'KPNO-IRAF'           /
'31-03-97'            /
IRAF-MAX=          1.993600E4  /  DATA MAX
IRAF-MIN=         -1.000000E0  /  DATA MIN
... 以下省略 ...
```

このようにサンプルデータのヘッダを見たり、imstat したりできる。

```
--> imstat dev$pix
#          IMAGE      NPIX      MEAN     STDDEV        MIN        MAX
         dev$pix    262144     108.3      131.3        -1.     19936.
```

ds9 をバックグラウンドで立ち上げて disp するのも普段通り。

```
--> ! ds9 &
--> disp dev$pix
frame to be written into (1:16) (1):
z1=35. z2=346.0218
```

最近の IRAF の ecl のように、tab 補完や矢印上下キーでのヒストリー機能が利用できる。

PyRAF を終えるには

```
--> .exit
```

PyRAF での入力を logfile を残すには

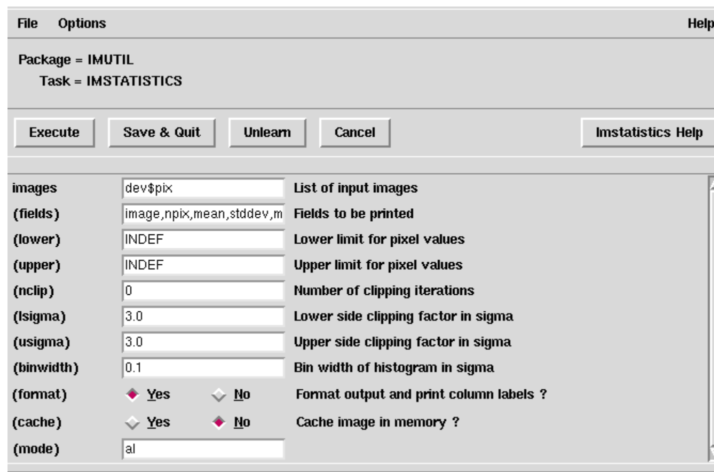
```
--> .logfile[filename[append|overwrite]]
```

とする。デフォルトは append である。また、logfile に残すのをやめるには、もう一度.logfile とだけうてばよい。

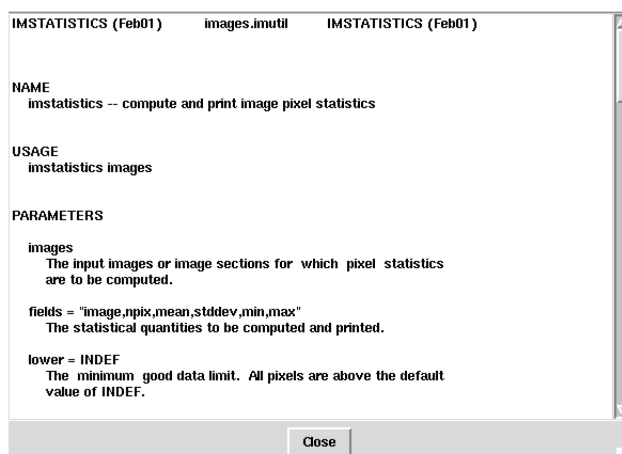
日々のコマンドライン使いにおいて、IRAF-CL から PyRAF へ移行するのに障壁はないであろう。

1.2.2 GUI の epar

「コマンドラインで使う」とは言ったが、PyRAF では epar で GUI の画面が立ち上がる。



ここでパラメータを設定して実行したり、パラメータの初期化 (Unlearn) したりできる。さらには、Help 画面が別ウィンドウで現れる。



GUI でやるのが嫌だったら、

```
--> imstat dev$pix fields=midpt,stddev
```

とか

```
--> iraf.imstat('dev$pix', fields='midpt,stddev')
```

とか

```
--> iraf.imstat.fields='image,midpt,mode'
--> iraf.imstat('dev$pix')
```

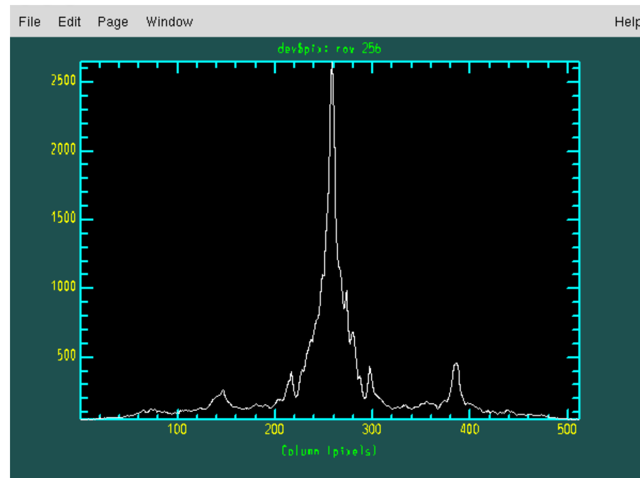
というふうに変数設定して実行することもできる。

1.2.3 グラフィック

PyRAF では xgterm を使わなくてもよい。CL では xterm で imexam の radial profile 表示をさせたりすると xterm にわけのわからない文字の羅列が出る。そのため CL では xgterm を使う必要がある。PyRAF ではそういう必要がない。OpenGL と Tkinter を用いた独自のグラフィックカーネルを持っているからである。ためしに、

```
--> prow dev$pix
```

を実行してみると、下のようなウィンドウが現れる。xgterm のものと違い、上部にメニューバーがあり、画面の print や save などができる。メニュー [Window] の [New] を選択すると、新しいグラフィクウィンドウが現れる。これを使うと複数のウィンドウに複数の結果を表示させることができる。



1.2.4 その他

- 面倒くさがり屋のために

```
--> i = iraf
--> i.imstat.fields='image,midpt,mode'
--> i.imstat('dev$pix')
```

という機能もある。 iraf を何度もタイプしなくてもよい。

• `Stdout=1` を使い、タスクの結果を文字列として保存することができる。

```
--> s = iraf.imhead("dev$pix", long=yes, Stdout=1)
--> print s[0]
dev$pix[512,512][short]: m51 B 600s
--> iraf.head(nl=3, Stdin=s)
dev$pix[512,512][short]: m51 B 600s
No bad pixels, min=-1., max=19936.
Line storage mode, physdim [512,512], length of user area 1621 s.u.
```

上記のように `Stdin` も使える。`Stderr` もリダイレクトとして使用が可能である。

• 起動時にログインバナーが出るのが嫌なら、`pyraf -s` とする。s は silent の意味。`pyraf -h` でその他の起動時オプションの説明が現れる。

Chapter 2

PyRAFプログラミング

2.1 簡単な例

まずはシンプルなスクリプトから。画像ファイル (dev$pix) を読み込んで imstat するスクリプト imstat1.py 。

```
#! /usr/bin/env python

from pyraf import iraf

iraf.images()
iraf.imstat("dev$pix")
```

シェルのコマンドラインで chmod +x をしてから imstat1.py を実行すると、結果が表示される。PyRAF を立ち上げる必要はない。

• 1 行目はこれが python スクリプトであることを計算機に教えている。この行がない場合には、python imstat1.py というようにアタマに python が必要。
• 3 行目は PyRAF を使うためのおまじない。
• 5 行目で images パッケージを開く。imstat を使用するためである。
• 6 行目で画像の読み込みと処理。

[注意事項]
python ではインデントが意味を持つ。ためしに 6 行目を 1 文字右にずらしてみよう。エラーが出る。

上記のスクリプトは PyRAF のコマンドラインでは動かない。次のような imstat2.py が必要である。

```
#! /usr/bin/env python

from pyraf import iraf

def run_imstat():
    iraf.images()
    iraf.imstat("dev$pix")
```

PyRAF を立ち上げて[1]、import imstat2 と入力する。これでスクリプト内の run_imstat() という関数が使えるようになる。imstat2.run_imstat() で動く。関数名を定義する行の最後には:を付け、その下のブロックはタブでインデントする。PyRAF で import するときにはスクリプト名および def に続く関数名に制限がつくので注意。ハイフンが使えない。imstat-2.py は使えないが、imstat_2.py ならよい。

実はこの imstat2.py はシェルのコマンドラインでは動かない。シェルからも PyRAF からも使えるようにするために以下のように imstat3.py を書く。

```
#! /usr/bin/env python

from pyraf import iraf

def run_imstat():
    iraf.images()
    iraf.imstat("dev$pix")

if __name__== "__main__":
    run_imstat()
```

chmod +x をすれば、シェルからは直接 imstat3.py で実行が可能で、PyRAF からは import imstat3 して imstat3.run_imstat() が実行できる。PyRAF からしか使わないのであれば chmod +x をする必要はない。

実は、mkiraf で作成されたデフォルトの login.cl を使っている限り、login.cl の中で images パッケージが立ち上げられているので、上記の例で iraf.images() がなくても iraf.imstat() は実行が可能である。
別の例を見てみよう。(mkimage.py)

[1] login.cl で定義している環境が関係しない場合なら python のコマンドラインからでもよい。

```
#! /usr/bin/env python

from pyraf import iraf

def run_mkimage():
    iraf.imred(_doprint=0)
    iraf.ccdred(_doprint=0)
    iraf.ccdtest()
    iraf.mkimage("test.fits","make","100","2","256 256")

if __name__== "__main__":
    run_mkimage()
```

_doprint=0 がないと最初のパケージ呼び出しのときにパッケージ立ち上げ時の表示が出る。デフォルトは_doprint=1 である。

2.2 実践で使えそうな例

もうちょっと実践で使い回しのできそうな例 imstat_file.py を挙げる。

```
#! /usr/bin/env python

import os,sys                  os.access と sys.argv をつかうため
from pyraf import iraf

def imstat_file(infile):   sys.argv[1] で受けた引数は infile へ

    if os.access(infile, os.R_OK):        ファイルは読めるか？
        f=open(infile)          ファイルオープン
        for line in f:
            fname, xx, yy = line[:-1].split(' ') 改行コード削除して
            if not line.startswith("#"):                    から分離
                fitsname=fname+'.fits'
                if os.access(fitsname, os.R_OK):
                    iraf.imstat(fname)
                else:
                    print "Error: can't read", fname
        f.close()
    else:
        print "Error: can't open", infile

if __name__== "__main__":
    imstat_file(sys.argv[1])      引数を一つ受付けて imstat_file に渡す
```

```
%more hoge.list
ff0010 10 10
ff0011 8 8
ff0012 11 8
ff0013 11 8
%imstat_file.py hoge.list
```

ここではファイル名 (.fits を省略したもの) と何か数字のコラムが二つ記述されているファイルを読み込む。そして、順番に画像ファイルを読み込み imstat していく。各行の先頭に#がついていればその行は無視される。

次に、上記のスクリプトに手を加えてみる。imstat のパラメータを変更して、かつ、その出力をファイルに書き出す。(imstat_file2.py)

```
#! /usr/bin/env python

import os,sys
from pyraf import iraf

def imstat_file(infile):

    if os.access(infile, os.R_OK):
        fout=open('imstat_out.txt', 'w')
        f=open(infile)
        for line in f:
            fname, xx, yy = line[:-1].split(' ')
            if not line.startswith("#"):
                fitsname=fname+'.fits'
                if os.access(fitsname, os.R_OK):
                    iraf.imstat.fields='image,mean,midpt,stddev'
                    out_imstat=iraf.imstat(fname,Stdout=1)
                    print >> fout, out_imstat[1]
                else:
                    print "Error: can't read", fname
        f.close()
        fout.close()
    else:
        print "Error: can't open", infile

if __name__== "__main__":
    imstat_file(sys.argv[1])
```

imstat からの返り値の out_imstat は**リスト**になっている。各行が配列になっている。一行目は '# IMAGE MEAN MIDPT STDDEV' として out_imstat[0] に入っている。ここでは二行目に入っている値を取り出す。

出力全体をファイルに書き出す場合には、

```
iraf.imstat(fname,Stdout="out.txt",Stderr="err.txt")
```

のようにしてもよい。Stdout="STDOUT"としてやると、標準出力にリダイレクトされる。Stderr="STDERR"も同様。

上記スクリプトでは、import os,sys で os および sys という**モジュール**を読み込む。python では多くのモジュールが用意されており、それらを利用することで楽にプログラムを組むことができる。

python で準備されているモジュールの中でも、math，shutil，glob などのモジュールがよく使われるだろう。例を挙げておく。

```
#! /usr/bin/env python

import os,sys,math,shutil,glob

def run_myscript():

    for file in glob.glob('*.fits'):
        os.remove(file)

    x=100
    y=math.log10(x)

    prefix=['jband','hband','kband']

    for band in prefix:
        image=band+'0001.fits'
        shutil.copy(image, 'temp.fits')
        os.system('myprog temp.fits')
        os.rename('temp.fits', band+'temp.fits')
```

copy や remove は iraf の imcopy や delete を使ってももちろんいいのだが、こういうやり方を覚えていても損はないだろう。

os.sysyem を使うと、UNIX コマンドや自作のプログラムなどが使えて便利だ。ソース検出だけを SExtractor にさせるのもよいだろう。

[**演習**]

(1) 上記の imstat_file.py を改造して、画像の背景のメジアン値の平均を出力させよ。

(2) 画像 (ひとつでよい) を読み込み、背景のメジアン値 (skylevel) と標準偏差 (stddev) を計り、その値を digiphot.apphot.daofind のパラメータに代入して、星の検出を行え。
daofind で重要なパラメータ設定は以下のように行うとよい。他はデフォルト値。

```
datapars.fwhmpsf = 2.2
datapars.sigma = stddev
datapars.datamin = skylevel-6*stddev
datapars.datamax = 15000
datapars.readnoise = 30
datapars.epadu = 5
findpars.threshold = 10
```

(3) 上記の daofind に引き続き apphot.phot を行うスクリプトを作成せよ。
上記の daofind 時の datapars の設定に加え、phot で重要なパラメータ設定は以下のように行うとよい。他はデフォルト値。

```
centerpars.cbox = 5
centerpars.maxshift = 2
fitskypars.annulus = 10
fitskypars.dannulu = 5
photpars.aperture = 9
```

(4) 上記の phot の結果の .mag.1 のファイルから、xc，yc，mag，merr，cier，sier，pier を txdump タスクを用いてファイルに書き出せ。あるいは、そのように上記スクリプトを改良せよ。

2.3 エラーの処理

python にはエラーを捕まえて処理する try:except: という制御文がある。PyRAF では IRAF の吐き出すエラーも、この try:except: で捕まえることができる。(trytest.py)

```
#! /usr/bin/env python

from pyraf import iraf

def test_try():

    try:
        iraf.columns("garbage",37)
    except iraf.IrafError, e:
        print "error was caught"
        print e

if __name__== "__main__":
    test_try()
```

このように、エラーが出る可能性のある場所を try:except: で囲んでやる。エラーがあったときには、IRAF の吐いたエラーをそのまま出力することができる。

2.4 タスクの登録

python で作成したスクリプトを PyRAF タスクとして使う[2]方法を説明する。epar を使ってのパラメータ設定もできるようになる。

ここでは mytask というタスクを作る。パラメーターファイルの mytask.par を作成する。書式などは login.cl と同じディレクトリにある uparm の中のパラメーターファイルを参考にする。

```
inword,s,a,"",,,"string to print"
xx,r,a,5.,,,"real value"
iy,i,a,1,,,"test integer"
mode,s,h,"al"
```

inword という文字列、xx という浮動小数点、iy という整数をパラメータとして設定した。xx の 5. や iy の 1 はデフォルト値である。設定しなくてもよい。

つぎに mytask.py を作成する。

[2]IRAF-CL の場合だったら、login.cl の package user の下あたりに、
task myscript ="/home/nakajima/script/myscript.cl"
のように書き加えれば使える。

```
from pyraf import iraf

def mytask(inword,xx,iy):
    print inword
    yy=2.0*xx
    ii=10*iy
    print "yy=%.1f ii=%d" % (yy,ii)        printf に相当する文

parfile = iraf.osfn("home$script/mytask.par")
t = iraf.IrafTaskFactory(taskname="mytask", value=parfile,
                         function=mytask)
```

iraf.osfn() の中には、上記で作成した par ファイルのパスを書く。

このmytaskをPyRAFの中で定義するには、

```
--> pyexecute("home$script/mytask.py")
```

のようにPyRAFのコマンドラインで打つ。毎回これを打つのが面倒な場合には、login.clの最後のほうのkeepの上あたりにpyexecute("home$script/mytask.py")を書き込んでおくとよい。

これで、epar mytaskでGUIが立ち上がる。