version 1.0

July 2012

pythonによる光赤外天文データ処理プログラミング入門

Yasushi Nakajima
Computer and Data Management Division
Subaru Telescope
NAOJ

Contents

Chapter 1

はじめに

1.1 python のすすめ

- python は習得が容易な言語である
- 広く使われている言語で、ウェブに多くの情報があり書物も多く出版されている
- 科学系のライブラリが豊富である
- IRAF のタスクが関数として利用できる
- フリーのソフトウエアである

上で並べた項目が、天文等のデータ処理プログラミング言語として python をすすめる大きな理由です。

python は google や YouTube さらには facebook などでも利用されています。広く使われているため、ウェブ上に多くの情報があるので何か分からないことがあれば検索すればたいていのことは解決します。系統的に学習するための書物も多く出版されています。このあたりが、IRAF の CL スクリプトとは大きく異なる点です。

python では数値演算、統計処理、可視化などの科学系ライブラリが非常に豊富なので、開発効率が高いです。天文系でいうと FITS や DS9 を扱うライブラリもそろっています。さらには PyRAF をインストールすれば IRAF のタスクが関数として利用できるのです。同様に PyMIDAS によって MIDAS(日本ではユーザーが少ないですが) が利用できます。

PyRAF によって IRAF のおいしいところだけ利用できるのはありがたいことです。IRAF のタスクは長い時間にわたって多くのユーザーに利用されてきたので、いわゆる「枯れたシステム」となっています。IRAF を信頼性の高いライブラリとして利用できるのです。これまで、あるいは一昔前までは天文データ処理には IRAF を使い、IRAF の CL スクリプトを使うことが一般的でした。IRAF の「信頼性の高い」タスクを使ったデータ処理スクリプトを作成するには、IRAF の CL スクリプトを使わなければなりませんでした[1]。しかし CL スクリプトは決して優秀なスクリプト言語ではなく、クセがあったり、超常現象を起こしたりします。教科書も少なく、web での情報も少ないので習得が困難です。今では、PyRAF をインストールすることで、CL スクリプトとは決別することができます。CL スクリプトと他の何かを組み合わせてやっていたことが python スクリプトだけでできてしまうのです。

python はフリーなので、広いユーザーコミュニティで利用してもらうソフトウエアを作るのに適しています。オープンソースなので、ユーザーによるバグ改善や新機能の貢献も期待できます。

他には、オブジェクト指向プログラミングが可能なので大規模なソフトウエアを開発するときに効率のよい開発が行えることや、C や C++との統合も容易に行えるので、python では処理速度が遅い箇所は C/C++で補うこともできるといったことも長所にあげられると思います。

[1]厳密にいえば、shell スクリプトから IRAF のタスクを利用することもできます。しかし、ライブラリとして使うというほど簡単な操作ではありません。perl から IRAF タスクを呼ぶモジュールもありますが、動作が不安定ですし、2002 年以降更新が止まっています。

1.2 このテキストを読みすすめるにあたって

このテキストを読む上で、必ずしも python を知っている必要はありません。python は非常に習得が容易な言語です。このテキストを読みつつ、python の勉強をしてもよいでしょう。他のプログラム言語 (C 言語や perl など) の経験があれば、プログラムの各行が何を意味しているのかだいたいは想像がつくと思います。(ちょっとクセのある部分は本文中で初出のときに注釈をいれてあります。) あるいは必要に応じてウェブで検索して調べてみてください。python は広く使用されているプログラム言語です。ウェブ上に非常に多くの情報が見つかります。

python のバージョンは 2.5 以降を想定しています。しかし python3 系には対応していません。Linux をインストールするとたいてい python がついてきて、バージョンはだいたい 2.6 か 2.7 だと思います (2012 年現在)。なのでたいていは問題ないでしょう。

OS は広い意味での UNIX(Linux, MacOSX, BSD など) を想定しています。シェルコマンドラインでの UNIX の基本操作を知っていることが前提です。

python に最初からは入っていないモジュールを多く使います。このテキストでは `pyfits`, `numpy`, `PyRAF`, `matplotlib`, `pyds9`, `aplpy` を利用します。あらかじめ自分のマシンにインストールしておいてください。

PyRAF を利用するので、IRAF の基本的なタスク (display, imstat, imarith, imcombine) の知識があると読み進めやすいです。IRAF のコマンドラインでの操作や CL スクリプトによるプログラミングの経験はなくてもよいです。

1.3 インタラクティブに使ってみる

このテキストの目的はプログラミングですが、まずは対話型コマンドラインで python をさわってみましょう。python でどんなことができるのか「さわり」を見てみたいと思います。

シェルのコマンドラインで `python` と入力すると python の対話型コマンドラインが立ち上がります。適当な FITS ファイルを二つ手元に用意して、以下のコマンドを入力してみましょう。

```
>>> from ds9 import ds9
>>> d=ds9()
>>> d.set('file sample1.fits')
>>> d.set('scale zscale')
>>> d.set('zoom to fit')
>>> import pyfits
>>> hdu1=pyfits.open('sample1.fits')
>>> hdu2=pyfits.open('sample2.fits')
>>> im1 = hdu1[0].data
>>> im2 = hdu2[0].data
>>> print  im1[100, 200]
>>> im3 = im2 - im1
>>> print im3
>>> hdr1=hdu1[0].header
>>> print hdr1['date']
>>> import stsci.numdisplay as nd
>>> nd.display(im3, zscale=True)
>>> import matplotlib.pyplot as plt
>>> plt.plot(im3[100,])
>>> plt.show()
>>> exit()
```

こんな感じです。exit() のかわりに Ctrl+D でも終了できます。

ipython を使うと、対話型コマンドラインが少し便利になります。コマンド入力時にタブ補完が効いたり、システムのシェルコマンドが使えたりします。シェルのコマンドラインで ipython と入力して立ち上げます。おすすめです。

Chapter 2

python プログラミング

2.1 データの読み書き

2.1.1 テキストファイル

空白区切りの 2 列のテキストデータの入ったファイルを読み込み、別のテキストファイルに書き出します。

test.txt

```
orion 1200
ophiuchus 2000
taurus 3000
```

sample211.py

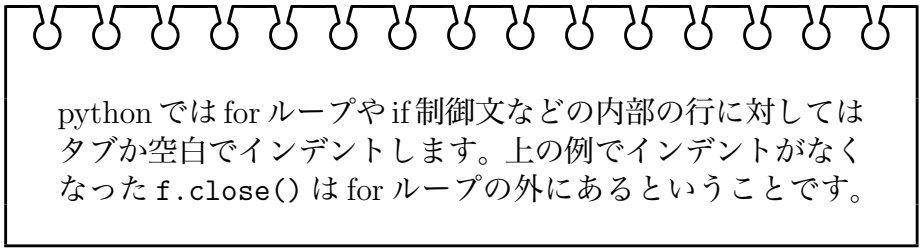
```
#! /usr/bin/env python

fo=open('testout.txt', 'w')    #    書き出しには'w' が必要
f=open('test.txt')
for line in f:
    varr=line[:-1].split()    # 改行コードを削除してから空白文字で
                              # 分離してからリストに入れる
    num=2*int(varr[1])        # 文字から整数に変換後に演算
    print >> fo, '%s %d' %  (varr[0], num)   # フォーマット文
f.close()
fo.close()
```

シェルのコマンドラインで chmod +x をしてから sample211.py を実行すると、結果が表示されます。ここでは適当に名前をつけた varr というリスト (配列) に文字列を代入しています。

• python では C 言語と同じく配列のインデックスは 0 からはじまります。

• テキストから読み込んだリストは文字列です。数値演算をする場合には整数 (int) や浮動小数 (float) に変換する必要があります。



python では for ループや if 制御文などの内部の行に対してはタブか空白でインデントします。上の例でインデントがなくなった f.close() は for ループの外にあるということです。

2.1.2 FITS ファイル

次に FITS ファイルを読み込みましょう。

sample212.py

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

import pyfits                        #  pyfits をロード

hdulist=pyfits.open('sample1.fits') # 適当な名前をつけたリスト
                                     # (配列) に FITS を読み込み
scidata=hdulist[0].data      #   画像データ (2 次元配列)
hdr=hdulist[0].header        #   ヘッダ

print '(x, y) = (106, 102)'
print scidata[101,105]       # python の配列は (y, x)

print '(x, y) = [102:105, 102:105]'
print scidata[101:105,101:105]  #  インデックス 101 は含むが
                                #  105 は含まれない

exptime=hdr['expos']       #    ヘッダは連想配列に
scidata *= 1/exptime    #   scidata = scidata * 1/exptime と同じ
print exptime
print scidata[101:105,101:105]

hdulist.close()          #  ファイルをクローズ
```

ここでは pyfits を利用して FITS ファイルの読み込みをしています。`import pyfits` を宣言することで、pyfits の関数を利用することができます。
pyfits は拡張FITS に対応しています。FITS ファイルはヘッダ部とデータ部のセット=HDU(Header Data Unit) から成ります。シンプルな一つのヘッダ+データ (HDU) のファイルの場合、Primary HDU しかないので hdulist[0] のみが存在します。拡張部がある場合には hdulist[n] (n=1,2,3, ...) に格納されます。

- python では**配列は (y, x) の順** で表します。
- 次のように exptime の値を読み込むこともできます。

```
exptime = hdulist[0].header['expos']
```

- 「11 番目のキーワードの値」として読み込むことも可能 です。

```
val =  hdulist[0].header[10]
```

- ヘッダキーワードをリストとして取り出すことも可能です。

```
>>> keylist = hdr.ascard.keys()
>>> print keylist
['SIMPLE', 'BITPIX', 'NAXIS', 'NAXIS1', 'NAXIS2', ... ]
```

- `x+=1` や `x*=2` は `x=x+1` や `x=x*2` と同じ演算を表します。これらを拡張演算ステートメントと呼びます。拡張演算ステートメントを使うと処理速度が速くなります。

ヘッダのキーワード、値、コメントをテキストファイルに書き出しましょう。

sample213.py

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

import pyfits

fimg=pyfits.open('sample1.fits')

card = fimg[0].header.ascard

for kw in card.keys():
    print '%s \t %s \t %s' % (kw, fimg[0].header[kw],
    card[kw].comment)

fimg.close()
```

文字列をタブで区切るというフォーマットをつけてみました。
この例では、card というディクショナリと `fimg[0].header[]`, `card[].comment`

という連想配列を扱っています。

• pyfits は gzip や fpack で圧縮された FITS ファイルも直接読めます。

```
hdu=pyfits.open('sample1.fits.gz')
hdu=pyfits.open('sample1.fits.fz')
```

次に FITS ファイルの書き出し方を覚えましょう。まずは既存の FITS ファイルを書き換える場合です。

sample214.py

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

import pyfits

fimg=pyfits.open('test.fits', mode='update')

prihdr=fimg[0].header
prihdr['object']='NGC9999'   #  既存のカードの値の書き換え
prihdr.update('observer','me')    #  新しいカードを加える

scidata=fimg[0].data
scidata[500,500]=-1000

fimg.close()
```

ファイルを開くときに、mode='update' をつけておくと、close() のときに変更分が書き込まれます。

• FITS のヘッダを見ると、キーワードは全て大文字です。上記の例では'object' と小文字になっていますがちゃんと認識されます。また、'observer' は大文字に自動的に変換されます。'me' は小文字のままです。

次に新規ファイルの作成です。— sample2 - sample1 の引き算イメージだけをデータとして持つ FITS ファイル (new.fits) と、sample2 - sample1 をプライマリの HDU に持ち、sample1 - sample2 を拡張 HDU に持つ FITS ファイル (new2.fits) を作成します。

sample215.py

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

import pyfits

img1=pyfits.open('sample1.fits')
img2=pyfits.open('sample2.fits')

data1=img1[0].data
data2=img2[0].data

img1.close()
img2.close()

data3=data2-data1

hdu=pyfits.PrimaryHDU(data3)
img3=pyfits.HDUList([hdu])    #  最低限必要なヘッダが自動生成される

img3.writeto('new.fits')
img3.close()

data4=data1-data2

hdu2=pyfits.ImageHDU(data4)   #  拡張部の HDU を作成
img4=pyfits.HDUList([hdu,hdu2])

img4.writeto('new2.fits')
img4.close()
```

• FITS の最初に入る HDU は `pyfits.PrimaryHDU()` で、それ以降に入る分については `pyfits.ImageHDU()` で作成するのがポイントです。

new2.fits については

`ds9 new2.fits[0] new2.fits[1]` (bash の場合)

`ds9 'new2.fits[0]' 'new2.fits[1]'` (tcsh の場合)

として表示してみましょう。

2.1.3 PyFITS convenience functions

pyfits には convenience functions というものが存在します。単純な操作だけを行うお手軽関数です。

```
scidata = pyfits.getdata('test.fits')
hdr = pyfits.getheader('test.fits')
```

これらは主にコマンドラインでの対話型操作のときに便利なものです。sample212.py での scidata と hdr と同じものが得られるうえに、これらを使うと pyfits.open と close が要りません。この関数の中で open と close を毎回行っているのです。ですから、もし同じファイルに対して何度も操作を繰り返すような場合には、これらを使わない方がよいです。ファイルの open と close のオーバーヘッドが大きくなるからです。pyfits.open でファイルにアクセスしてください。

2.1.4 演習問題 1

添付資料の exercise1 ディレクトリの中には、天体の生フレームが一枚 (object.fits)、生ダークフレームが 20 枚 (es0001.fits - es0020.fits) あります。生ダークフレームのうち es0011.fits - es0020.fits が天体フレームと同じ積分時間のものです。
次の処理を行う python スクリプトを作成してください。

1. 「天体と同じ積分時間の生ダークフレームから平均のダークフレームを作成して、天体フレームから引き算をする。」
通常は平均のダークフレームを作成する場合には、シグマクリップを施すとかメジアンで計算するなどしますが、ここでは単純な算術平均で構いません。ここでは、FITS インターフェースとしては、pyfits のみを使用してください。

2. 「スクリプトにより自動的にディレクトリ内の FITS ファイルのヘッダを読み込み、天体と同じ積分時間のものだけ選んだうえで、上記 1 の処理を行う。」
ファイル名はテキストリストで与えて構いません。

2.2 FITS画像の表示

2.2.1 pyds9

pyds9を使って、ds9にFITSを表示してみましょう。

sample221.py
```
#! /usr/bin/env python

from ds9 import ds9

d=ds9()      #    ds9が立ち上がる
d.set('file sample1.fits')  # FITSの表示
d.set('height 600')
d.set('width 600')
d.set('zoom to fit')
d.set('scale zscale')
```

XPA Access Points (http://hea-www.harvard.edu/RD/ds9/ref/xpa.html) にはxpasetというコマンドを使ってds9に命令を出す例がたくさん書かれてあります。ds9でボタンを押したりプルダウンメニューから選んだりして操作することは全てxpasetで命令することができます。命令の例文の`xpaset -p ds9`に続く部分をpyds9では`d.set('')`の中に書けばよいです。

sample222.py
```
#! /usr/bin/env python

from ds9 import ds9

d=ds9()
d.set('file sample1.fits')
hdu=d.get_pyfits()
data=hdu[0].data
print data[100,200]
```

ds9に表示したFITSのデータには`get_pyfits()`を使ってアクセスすることができます。わざわざ`import pyfits`する必要はありません。

2.2.2 numdisplay

`stsci.numdisplay`を使う方法もあります。

sample223.py

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

import numpy
import pyfits
from ds9 import ds9
import stsci.numdisplay as nd   #   nd は適当につけた名前

img1=pyfits.open('sample1.fits')
img2=pyfits.open('sample2.fits')

data1=img1[0].data
data2=img2[0].data

img1.close()
img2.close()

data3=data2-data1

d=ds9()

nd.display(data3,z1=-300, z2=300)   #  z1, z2 は表示レンジ
```

上の例では pyds9 で ds9 をプログラムの中で立ち上げていますが、端末からあらかじめ ds9 を立ち上げておくのでもかまいません。

• 最初に `import stsci` として `stsci.numdisplay.display(data3)` のように関数を使うのでもかまいません。`stsci.numdisplay.` がプログラムの中に何回も出てくる場合には、短縮した名前を上記のようにつけておくと楽でしょう。

2.2.3 pyraf — display

IRAF に慣れている場合には pyraf を利用して表示するのもよいでしょう。

sample224.py

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

from pyraf import iraf
from iraf import images   # display は images パッケージの中の
from iraf import tv       # tv パッケージの中にあるので
                          # これらの import が必要
iraf.display('sample1.fits', frame=1)
```

この例では、あらかじめ端末から ds9 を立ち上げておく必要があります。先の例のようにプログラム中で ds9 を立ち上げるのでもかまいません。

• IRAF で使うように `iraf.display('sample1.fits', frame=1, fill='yes')` というように並べて書けば、パラメータの指定ができます。(パラメータの指定方法には他の方法もあります。後の節で紹介します。)
• この script を走らせると、
`Created directory ./pyraf for cache`
`Warning: no login.cl found`
のようなメッセージが出ます。無視しても構いません。
うっとおしければ、ユーザー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おくと、その中に `pyraf` というディレクトリが最初に作成され、その後には一番目のメッセージは出なくなります。また、`~/iraf` の中で mkiraf を実行して `~/iraf/login.cl` を作成しておくと二番目のメッセージは出ません。pyraf を使う script を動かすときに`~/iraf/login.cl` を読み込もうとするのです。
login.cl を作っておくと、images, tv, noao などの基本的なパッケージは login.cl の中で宣言されているので、`from iraf import images` などをしなくても `iraf.display()` が動くというメリットがあります。

2.2.4 演習問題 2

シェルのコマンドラインで、ds9 を立ち上げておいてから、
`xpaset -p ds9 2mass name m31`
とすると 2MASS の M31 の画像が ds9 に現れます。
`http://hea-www.harvard.edu/RD/ds9/ref/xpa.html` のページを参考にしながら、次の処理を行うひとつの python スクリプトを作成してください。

• ds9 を立ち上げ、2MASS の M31 の画像を表示する。
• 表示スケールを log にする。
• 表示カラーを heat にする。
• wcs システムのグリッドを表示する。
• 2MASS カタログの星をプロットする。

2.3 データ解析

2.3.1 numpy

pyfits を用いると、FITS の二次元配列のデータは numpy オブジェクトとして読み込まれます。numpy モジュールは多次元配列を操作するためのさまざまな数学関数をもちます。numpy の機能を利用してみましょう。

> オブジェクト指向に馴染みの無い方には「オブジェクト」と言われてもピンとこないでしょう。ここでは、「データを numpy オブジェクトにしておくと、そのデータに対して numpy の関数を利用することができる」というくらいに考えておいてください。

sample231.py

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

import numpy as np
import pyfits

img1=pyfits.open('sample1.fits')
data1=img1[0].data
img1.close()

a=np.min(data1)
b=np.max(data1)
c=np.median(data1)
print 'min %.1f  max %.1f  median %.1f' % (a,b,c)

harr=np.histogram(data1,bins=[480,500,520,540,560,580])
print zip(harr[0],harr[1])   #  リストを組み合わせて連想配列

bigvals=np.where(data1>10)
data1[bigvals]=np.log10(data1[bigvals]-10)
```

上の例では、画像の最小値、最大値、メジアンなどの統計量の計算とピクセル値のヒストグラムの作成、numpy の where を使っての画像の対数の計算をしています。

• histogram の bins の数値は各ビンの左端の値です。
• where は () の中の条件を満たす配列を返します。data1[bigvals] とすることで、その条件を満たすピクセルについてのみ演算が行われます。バッドピクセルの処理のときなどに便利です。

python はインタープリター言語であり、数値計算は C 言語などに比べると遅くなることが多いです。しかし、numpy を使うと配列を扱う場合には C のコードに匹敵するほどの速度で動作します。

```
sum=0
for j in range(1024):
    for i in range(1024):
        sum+=im[j, i]
```

といった計算は、numpy.sum() を使って

```
sum=numpy.sum(im)
```

とするほうが 30 倍くらい速く計算できます。

numpy の関数は各種統計計算や高速フーリエ変換など他にもたくさん (400 くらい) あります。下のページをご覧ください。

http://scipy.org/Numpy_Example_List_With_Doc

2.3.2 pyraf

pyraf モジュールを使うと、IRAF のタスクを python スクリプトから関数として呼び出すことができます。まずは基本の使い方から。

sample232.py

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

from pyraf import iraf
from iraf import images

iraf.imstat('sample1.fits')
```

iraf.imstat は画像の統計量を計算するタスクです。images というパッケージの中にあります。from iraf import images としてそのパッケージを開いてやる必要があります。

パラメータ設定

IRAF のタスクでは、パラメータを設定することで、何を測定するか (median や mode など)、どのように測定するか (sigma-clip をするか？など) を設定できます。パラメータの設定の方法は二通りです。
(1) imstat() の中に書き込む。

sample233.py
```
#! /usr/bin/env python

from pyraf import iraf
from iraf import images

iraf.imstat('sample1.fits', fields='midpt,stddev', nclip=3,
lsigma=3, usigma=3)
```

(2) iraf.imstat.fields=... のようにタスクを走らせる前に設定しておく。

sample234.py
```
#! /usr/bin/env python

from pyraf import iraf
from iraf import images

iraf.imstat.fields='midpt,stddev'
iraf.imstat.nclip=3
iraf.imstat.lsigma=3
iraf.imstat.usigma=3
iraf.imstat('sample1.fits')
```

IRAF のタスクの使い方さえ知っていれば使うのは簡単ですね。

元々、IRAF にはないパラメータの設定もあります。タスクの出力を変数に渡すか、ファイルに書き出すか、エラー出力をどのようにするか、を設定できます。

```
out_imstat=iraf.imstat('sample1.fits', Stdout=1)
print out_imstat[1]
```

のようにすると、imstat の結果を out_imstat に入れることができます。imstat の結果は複数行にわたるので、各行がリストとして out_imstat に入ります。

FITS ファイルから星を検出して測光する例を見ましょう。

sample235.py

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

from pyraf import iraf
from iraf import noao
from iraf import digiphot
from iraf import apphot
from iraf import images

myimage='sample1.fits'

iraf.unlearn('apphot')  #  パラメータの初期化

iraf.apphot.datapars.datamax=20000
iraf.apphot.datapars.readnoise=30
iraf.apphot.datapars.epadu=8
iraf.apphot.findpars.threshold=7
iraf.apphot.findpars.sharphi=0.8
iraf.apphot.daofind.interac='no'
iraf.apphot.daofind.verify='no'
iraf.apphot.datapars.fwhmpsf=3
iraf.apphot.centerpars.cbox=5
iraf.apphot.fitskypars.annulus=8
iraf.apphot.fitskypars.dannulus=5
iraf.apphot.photpars.apertures=5
iraf.apphot.phot.interactive='no'
iraf.apphot.phot.verify='no'
iraf.apphot.phot.verbose='no'
iraf.apphot.photpars.zmag=21

iraf.imstat.fields='midpt,stddev'
iraf.imstat.lower='-1000'
iraf.imstat.upper='20000'
iraf.imstat.nclip=10
iraf.imstat.lsigma=3
iraf.imstat.usigma=3
iraf.imstat.binwidth=0.1
iraf.imstat.format='yes'
iraf.imstat.cache='yes'
```

続き

```
out_imstat=iraf.imstat(myimage,Stdout=1)
med,stdev=out_imstat[1].split()

iraf.apphot.datapars.sigma=float(stdev)
iraf.apphot.datapars.datamin=float(med)-6*float(stdev)

iraf.daofind(myimage, output='my.coo')
iraf.phot(myimage, coords='my.coo',output='my.mag')
iraf.txdump('my.mag',fields='xc,yc,mag,merr',expr='cier==0 &&
sier==0 && pier==0', Stdout='my.xymag')
```

- パラメータの初期化をすると、このコードがいつでも同じ結果を出します。
- 項が二つしかないと分かっていれば、 `med,stdev=out_imstat[1].split()` のようにして変数を渡すこともできます。
- `Stdout='my.xymag'` のように標準出力をファイルに書き出すこともできます。
- `Stderr='err.txt'` としてエラーをファイルに書き出します。

この他の出力の操作として、StdoutGがあります。iraf.noao.obsutilのpsfmeasureというタスクは、星のPSFを測定するのですが、その結果をグラフィックウインドウに出力します。パイプラインスクリプトなどでは、わざわざそういうのを出してほしくない時もあります。そういう場合には

```
iraf.psfmeasure('sample1.fits', StdoutG='/dev/null')
```

としてやると出なくなります。

2.3.3 その他

math モジュール

python に標準で入っているモジュールです。三角関数や対数、指数など。配列でない数値の演算に使えます。math.pi(円周率) と math.e(自然対数の底) も定義されています。対話型コマンドラインで

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt','tan',
'tanh', 'trunc']
```

とすることで math で使える関数を見ることができます。
__doc__を使うと、各関数の簡単な説明をみることができます。これらは他のモジュールでも同様です。

```
>>> print math.fabs.__doc__
fabs(x)

Return the absolute value of the float x.
```

簡単なスクリプト例を挙げておきます。

sample236.py

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

import math

print math.pi

mydeg=30
myrad=math.radians(mydeg)
mysin=math.sin(myrad)
print mysin
```

os.system

他のソフトウエア、あるいは C や fortran のプログラムをコンパイルして作った実行形式を使うときには、os.system を使います。

sample237.py

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

import os

fname='sample1.fits'

os.system('sex %s' % fname)  # 引数はフォーマット文を使います
```

ここでは SExtractor を用いて星の検出を行います。SExtractor の default.param や default.sex を用意しておきましょう。

2.3.4 演習問題 3

exercise1 の中の object.fits からバッドピクセルをマスクする FITS ファイルを作成してください。ここでは pyfits と numpy のみを使用してください。

バックグラウンドレベルを全 pixel 値の median で計算し、バックグラウンドの標準偏差を求めて、median−3 stddev よりも値が小さなピクセルをバッドピクセルとしてください。バッドピクセルをマスクする FITS ファイルでは、正常ピクセルは値が 0 で、バッドピクセルでは値が 1 になるようにしてください。

2.3.5 演習問題 4

exercise4 の中には天体フレームが 9 枚あります (jf0658.fits - jf0666.fits)。ダークやフラットなどの処理がすでにされているものです。shift.txt の中にはこれらのフレームの、最初のフレーム jf0658.fits に対する相対シフト量が記載されています。iraf.imshift を使って画像をシフトして、iraf.imcombine で average で重ねてください[1]。

重ねてみるとわかりますが、画像の外側で星が円弧を描いています。本当は imshift ではなく geotran で回転まで考慮すべきデータです。ここでは簡単のためシフトだけを適用しました。

[1] dithering した画像を重ね合わせて S/N を上げた画像を作るときには average でやるほうが S/N が $\sqrt{\pi/2}$ だけ median よりも高くなります。

2.4 図やグラフの描画

2.4.1 matplotlib

matplotlib は python のためのグラフ描画ライブラリです。
http://matplotlib.sourceforge.net/ のページに行くと、各関数の説明とともにものすごい数のスクリプト例があります。screenshot や gallery のページでは描画したグラフがソースコードへのリンクとともに並んでいます。目で見て、こういうのが書きたいと思えば、そのソースコードをまねればよいわけです。「こういう時にはこう書く」で身につけることができます。

ここでは撮像観測のデータ解析で使いそうな例をいくつかあげます。まずは測光データの等級 vs. 等級エラーおよび等級のヒストグラムの図のプロットです。

sample241.py

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

import matplotlib.pyplot as plt

mag=[]
merr=[]
f=open('my.xymag')
for line in f:
    val=line[:-1].split()
    mag.append(float(val[2]))
    merr.append(float(val[3]))
f.close()

fig=plt.figure(facecolor='white', figsize=(10,10))
                            #  図のウインドウを立ち上げる
ax1=fig.add_subplot(211)  #  グラフを書く座標系を作る
ax1.scatter(mag,merr)  #  データの散布
ax1.set_xlim(9,16)   #  x 軸の範囲を設定
ax1.set_ylim(0,0.2)  # y 軸の範囲を設定
ax1.set_xlabel('magnitude', fontsize=15)
ax1.set_ylabel('magerr', fontsize=15)
ax1.set_title('photometry')

ax2=fig.add_subplot(212)
ax2.hist(mag, range=(9,16), rwidth=1, color='red')  # ヒストグラム
ax2.set_xlim(9,16)
ax2.set_ylabel('N', fontsize=15)
```

続き

```
plt.show()  #  描画
fig.savefig('mymag.pdf')  #  ファイルへ保存
```

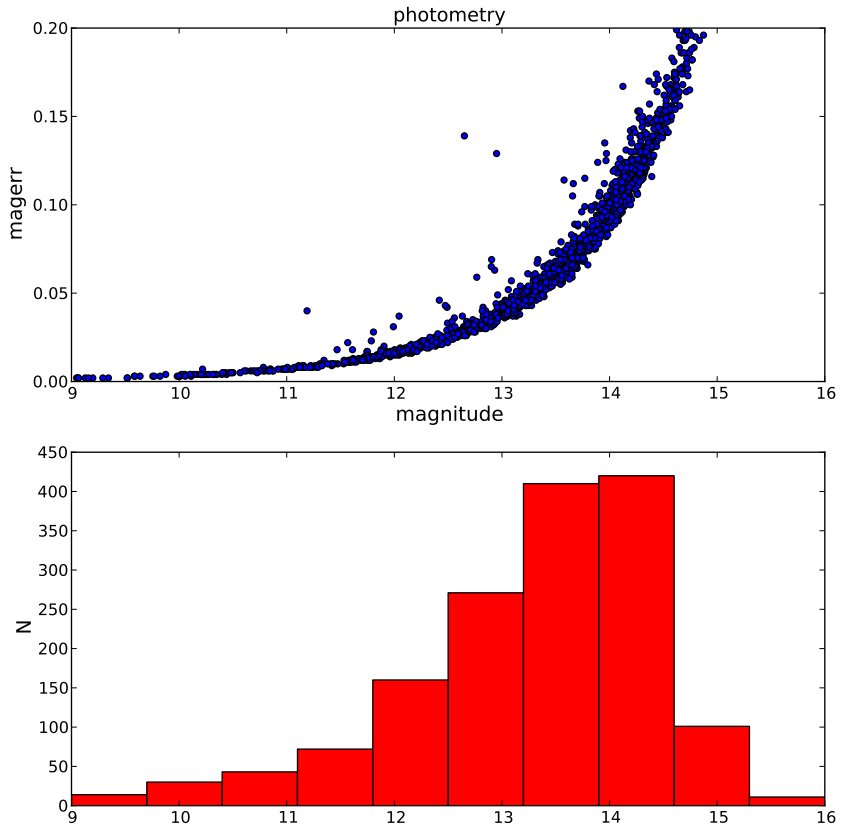


Figure 2.1: sample241.py の出力例

- `subplot(211)` は図を 2 行 x 1 列に分割したのの 1 番目の図という意味です。順番は左上から行に沿っていく順です。図が一つだけのときは (111) です。
- `scatter` や `hist` のパラメータを設定することで、色やプロットの形などを変えることができます。
- `savefig()` では png, pdf, ps, eps, svg のファイルへの書き出しが可能です。

次に FITS 画像の上にマークをつけて表示させましょう。

sample242.py

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

import pyfits
import matplotlib.pyplot as plt
import matplotlib.cm as cm

xcoo=[]
ycoo=[]
f=open('my.xymag')
for line in f:
    val=line[:-1].split()
    xcoo.append(float(val[0]))
    ycoo.append(float(val[1]))
f.close()

fimg=pyfits.open('sample1.fits')
pixdata=fimg[0].data
fimg.close()

fig=plt.figure(facecolor='white', figsize=(10,10))
ax1=fig.add_subplot(111)
ax1.imshow(pixdata,cmap=cm.gray,vmin=520,vmax=600,
interpolation='nearest',origin='lower')
ax1.scatter(xcoo,ycoo,s=100,edgecolors='yellow',facecolors='none')

plt.show()
```

• scatter の s というパラメータがプロットのサイズを決めます。半径の二乗に比例する量です。

2.4.2 演習問題 5

sample242.py で使った my.xymag を読み込んで、白地の背景に星を黒丸でプロットしてください。ここで、星の等級が 1 等明るいものは丸の面積が 2.5 倍になるようにしてください。

http://matplotlib.sourceforge.net/gallery.html を眺めて、大きさの異なる丸をプロットしている例がないか探してみましょう。

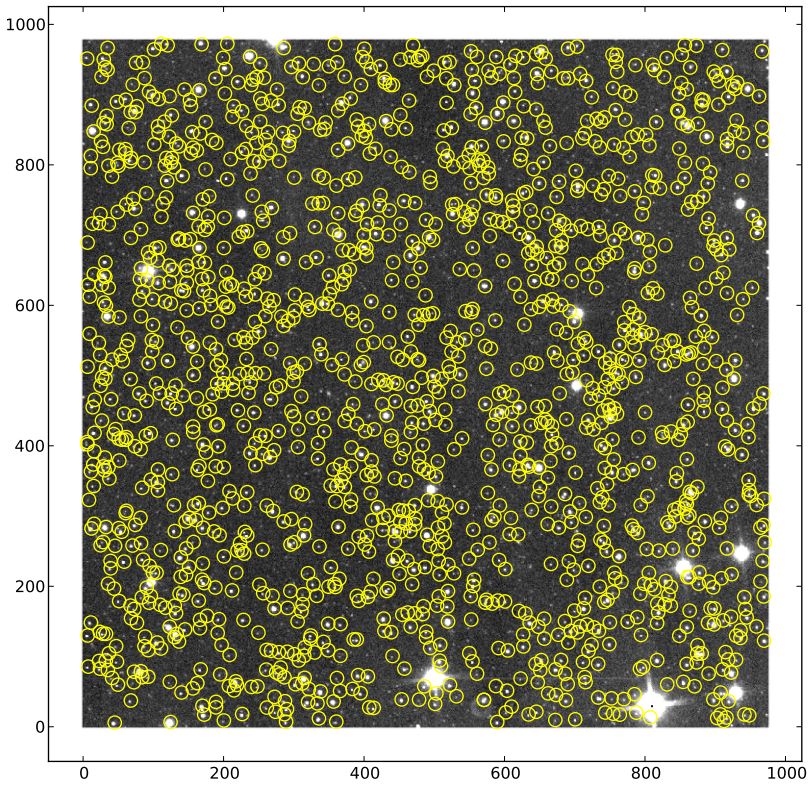


Figure 2.2: sample242.py の出力例

2.4.3 ApLPy

FITS イメージから論文用の図を作成するためのモジュールです。
`http://aplpy.github.com/`

上の例では、FITS のアレイ上の (x, y) 座標を使って星の位置をプロットしましたが、論文などではちゃんと赤経赤緯で表示したいですね。FITS にちゃんと wcs が入っていれば ApLPy を使うと非常に楽です。

また、3 色合成図を作成するのにも便利な関数が入っています。`aplpy.make_rgb_cube()` と `aplpy.make_rgb_image()` です。ただし、これらを使うには montage というソフトが入っている必要があります。

ApLpy の web ページの quickstart guide に載っている例 (少し変えています) を見ていきましょう。`http://aplpy.github.com/downloads/tutorial.tar.gz` をダウンロードして解凍したディレクトリ `tutorial` の中で実行してください。

sample243.py

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

import aplpy
import numpy

gc = aplpy.FITSFigure('fits/2MASS_k.fits', figsize=(10, 10))
gc.show_grayscale()

gc.tick_labels.set_font(size='small')

gc.show_contour('fits/mips_24micron.fits', colors='white')

data = numpy.loadtxt('data/yso_wcs_only.txt')
ra, dec = data[:, 0], data[:, 1]

gc.show_markers(ra, dec, layer='marker_set_1', edgecolor='red',
                facecolor='none', marker='o', s=10, alpha=0.5)

gc.save('myfirstplot.png')
```

- 'fits/2MASS_k.fits' の画像をグレイスケールで描きます。FITS ヘッダに wcs が入っているので、枠には赤経、赤緯の座標軸が現れます。
- その上に'fits/mips_24micron.fits' のコントアを上書きします。
- 'data/yso_wcs_only.txt' から YSO(若い星) の座標を読み取ります。ここでは numpy.loadtxt を利用しています。
- show_markers で読み取った YSO を図の上にプロットします。

このスクリプトを動かすと、DeprecationWarning:が出るかもしれませんが、無視してください。将来的に python からなくなりそうな関数を使っているので、ご丁寧に「そうじゃなくて、こうしたほうがいいですよ」と教えてくれているのです。でも、aplpy の中で使われているので仕方ありません。
うっとおしければ、
warnings.simplefilter('ignore', DeprecationWarning)
を import numpy の次あたりに挿入してください。

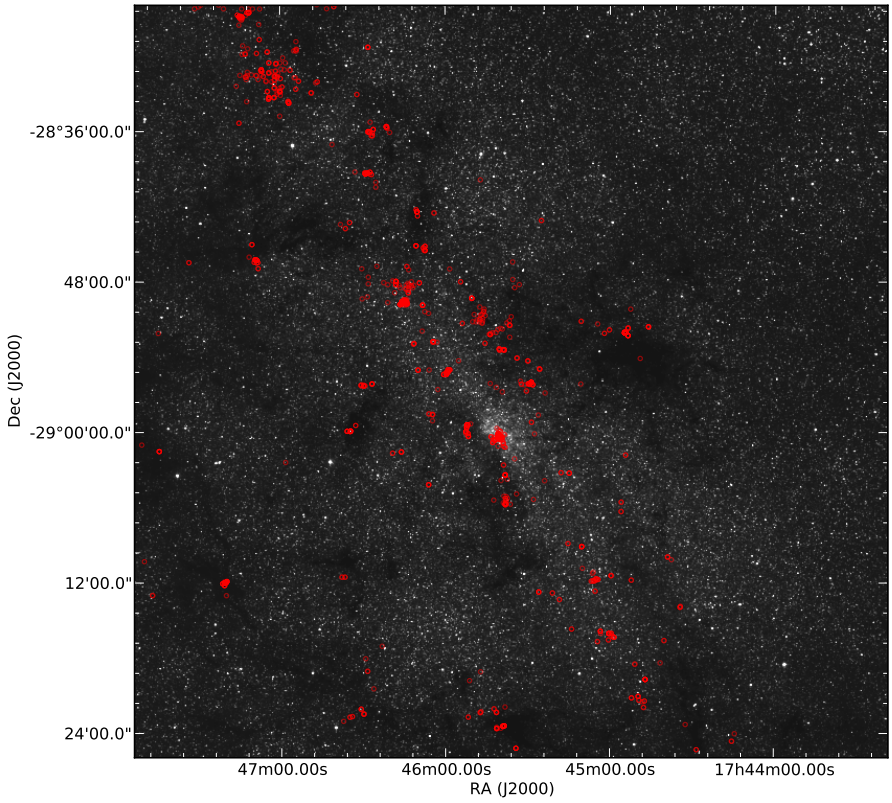


Figure 2.3: sample243.py の出力

tutorial の data 中には 2MASS の同じ視野の J,H,Ks のデータがそろっています。せっかくなので 3 色合成図を作ってみましょう。

sample244.py

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

import aplpy

aplpy.make_rgb_cube(['fits/2MASS_k.fits','fits/2MASS_h.fits',
                     'fits/2MASS_j.fits'],'2MASS_cube.fits')
aplpy.make_rgb_image('2MASS_cube.fits','2MASS_cube.tif')
```

簡単ですね。元の FITS ファイルには wcs が入っている必要はありますが、位置合わせされている必要はありません。`make_rgb_cube()` が 3 枚の画像を合わせてくれます。必要であれば `make_rgb_image()` の引数として、`vmin_r=0, vmax_r=100`

というふうに表示レンジを r,g,b それぞれに指定してやってカラーバランスを調整することもできます。tiff ファイルだけでなく、PDF にも書き出せます。

Figure 2.4: sample244.py の出力

2.5 関数

ここまでのスクリプト例は、python で利用できる「用意されている関数」の使い方に注目したものでした。実際のスクリプト作成では、メインルーチンに「自作の関数」— サブルーチンを組み合わせて作ります。そうすることで、スクリプトの見通しが良くなったり、別のスクリプトで自作の関数を使い回せて開発効率があがります。

sample251.py

```
#!/usr/bin/env python
import os,sys
from pyraf import iraf
from iraf import images

def imstat_file(infile):   #  自作の関数

    if os.access(infile, os.R_OK):    #  ファイルが存在する？
        f=open(infile)
        for line in f:
            if not line.startswith("#"):  #  #で始まる行は無視
                fname, xx, yy = line[:-1].split()
                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__":     #   メイン関数
    argvs=sys.argv
    argc=len(argvs)   #   引数の数
                      # コマンドそのものも引数の数に入ります
    if argc<2:
        print 'Usage: # sample251.py [name of file]'
        sys.exit()

    imstat_file(sys.argv[1])    #   関数に引数を渡している
```

- `def ...():` で関数をはじめます。
- `if __name__== "__main__":` でメイン部をはじめます。

• 関数の中はインデントします。

myfile.txt
```
sample1 1 1
sample2 2 2
#sample3 3 3
```

上記のような myfile.txt を用意して、./sample251.py myfile.txt を実行してみましょう。

このようにスクリプトを作っておくと、他のスクリプトからこのスクリプトをモジュールとして使うことができ、def imstat_file() をモジュールの関数として使うことができます。

sample252.py
```
#!/usr/bin/env python
import os,sys
from pyraf import iraf
from iraf import images
import sample251   #   .py はつけない

sample251.imstat_file('myfile.txt')
```

このとき注意することがあります。python スクリプトのファイル名にハイフンを使うと import できません。sample25-1.py というスクリプトを実行することはできるのですが、import sample25-1 はできません。同様に import sample25.1 もできません。import sample25_1 のようなアンダースコアは使えます。(しかし python 的には推奨しないようです。)

2.6 便利なツール

2.6.1 pyflakes

pyflakes は Python スクリプトを分析し、様々なエラーを検出するプログラムです。バグ出しに非常に有効です。これまでに試したスクリプトのコピーを作成して、わざとエラーを含むスクリプトを作成して `pyflakes test.py` のように試してみましょう。どの箇所にエラーがあるかを示してくれます。

Chapter 3

スクリプト集

3.1 空のFITSファイル

全てのピクセルでの値が0(32ビット浮動小数点数)である画素数1024×1024ピクセルのFITSファイルを作成します。

sample31.py

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

import numpy as np
import pyfits

data=np.zeros((1024,1024), dtype='float32')
hdu=pyfits.PrimaryHDU(data)
img=pyfits.HDUList([hdu])

img.writeto('zero.fits')
img.close()
```

ここでnp.zerosをnp.onesに変えると、ピクセル値が1になります。

3.2 ヘッダのカードの削除

FITS ファイルのヘッダの 12 行目以降のカードをまとめて削除します。flist.txt には FITS ファイル名が一行にひとつづつ書かれてあります。

sample32.py

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

import pyfits

f = open('flist.txt')
for line in f:

    img = pyfits.open(line[:-1], mode='update')
    prihdr = img[0].header
    cards = prihdr.ascard
    for i, kw in enumerate(cards.keys()):
        if i > 11:
            del cards[kw]
    img.close()

f.close()
```

3.3 位置シフトを手動で測定

ディザリングして撮った画像を重ね合わせたいので、ここでは平行移動のズレだけを測定します。テキストファイルに、一行にひとつづつ FITS ファイル名を書き込みます。そのファイルを引数として与えます。(例：`sample33.py fitsname.txt`) すると ds9 が立ち上がります。`frame to be written into (1:16) (1):` のような表示がコマンドラインに現れますのでリターンします。すると最初のフレームが表示されます。全てのフレームに写っているそこそこ明るく、サチっていない星ひとつを見極めます。`iraf.imexm` が起動しているので、カーソルをその星に合わせて `a` のキーを押します。次に `q` のキーを押します。すると次のフレームに移ります。また `frame to be written into (1:16) (1):` が出ますのでリターンします。これを繰り返します。最後に、最初のフレームを基準とした位置シフト量 (x, y) がフレーム名とともにコマンドラインに表示されます。これを元に iraf.imshift をして iraf.imcombine するとよいです。

sample33.py

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

import os,sys
from pyraf import iraf
from ds9 import ds9

def getshift(infile):

    xx=[]
    yy=[]
    imlist=[]

    if os.access(infile, os.R_OK):
        f=open(infile)
        for line in f:
            fitsname=line[:-1]
            if not line.startswith("#"):
                if os.access(fitsname, os.R_OK):
                    imlist.append(fitsname)
                    iraf.display(fitsname,fill='yes')
                    result=iraf.imexam(Stdout=1)
                    values=result[2].split()
                    xx.append(float(values[0]))
                    yy.append(float(values[1]))
                else:
                    print "Error: can't read", fitsname
        f.close()
    else:
        print "Error: can't open", infile


    print imlist[0],'0.00','0.00'
    for i in range(1,len(xx)):
        print imlist[i],xx[0]-xx[i],yy[0]-yy[i]


if __name__ == "__main__":

    d=ds9()
    getshift(sys.argv[1])
```