PythonとC/C++プログラムを連携させた解析プログラム

安田　直樹
東京大学・IPMU
2010年11月24日・IRAF講習

今日の講習内容

- Pythonについて
  - 基本的な機能の練習
- Swigについて
  - 簡単な例
  - いくつかの応用
  - Numpy array との連携
- 画像処理のサンプル
  - Image クラスの定義
  - フラット画像作成
  - フラット処理、スムージング、ピーク検出

実行環境

- Python 2.6.6
- swig 1.3.40
- numpy 1.5.0
- pyfits 2.3
- pyds9 1.1
- svn co http://svn.scipy.org/svn/numpy/tags/1.5.0/doc/swig でダウンロードした swig/numpy.i を /usr/local/share/swig/1.3.40/python など swig をインストールしたディレクトリの 対応する場所にコピーする。

Pythonとは

- フリーなオブジェクト指向スクリプト言語
- 「シンプル」で「習得が容易」
- 高度なプログラミングや大規模開発も可能
- 機能を拡張する豊富なライブラリが世界中で開発・公開されている
- 天文関係でも広く使われ始めている

Hello Python

python/hello.py

- Pythonの実行・終了

```
% python
Python 2.6.6 (r266:84292, Nov 16 2010, 16:27:48)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
>>> print “Hello Python!”⏎
Hello Python!
>>> print 5 + 3⏎
8
>>> ^D (Crtl+D) 終了
%
```

- ファイルにスクリプトを書いて実行

```
% python hello.py⏎
Hello Python!
8
%
```

リストとループ

python/loop.py

- リスト　：　配列みたいなもの

```
a = [“abc”, “def”, “ghi”, “jkl”]
for b in a:      ←ブロックの最初の行の末尾にコロン
____print b
   ↑
```

ブロックの区切りは字下げ（インデント）で表現

- forループ

```
for i in range(5):
    print i

range(5) ➔ [0,1,2,3,4]
```

ファイルI/O

python/io.py

- ファイルを開く

```
file = open(‘test.txt’)     読み込み
file = open(‘out.txt’, ‘w’)   書き込み
```

- 読み書き

```
line = file.readline()
file.write(line)
```

関数・モジュール

python/func.py

- 関数

```
def times(x, y):
    return x * y
```

  - 引数にはデフォルト値を指定することもできる

- モジュール

```
import func
from func import *
```

  - 呼び出しは前者の場合

```
x = func.times(3.14, 4)
```

pyfits

pyfits/sample.py

- FITSファイルを読み書きするライブラリ

  http://www.stsci.edu/resources/software_hardware/pyfits

```
hdulist = pyfits.open('test.fits')
hdulist[0].header['OBJECT']
hdulist[0].data[y, x]
```

- 画像データは numpy array として扱われる
  - 0-indexed なので、ds9 とは1ピクセルずれている
  - 添字の順番が [y, x] の順

pyds9

pyds9/sample.py

- ds9に画像の表示を始めとするコマンドを送るライブラリ
  http://hea-www.harvard.edu/saord/ds9/pyds9
- X Public Access (XPA) という仕組みを使っている
  XPA Access Points のリファレンスは以下
  http://hea-www.harvard.edu/saord/ds9/ref/xpa.html
- 使い方

```
d = ds9.ds9()                ← ds9が起動する
d.set('file test.fits') ← ファイルから読み込む

arrayを送ることもできる
d.set('array [xdim=2080,ydim=4100,bitpix=-32]', data)
```

画像の統計量を計算する

pyfits/stats.py

- Pythonだけで計算
- Numpyの機能を使って計算
  http://www.scipy.org/Numpy_Example_List
- かかる時間を比較してみる

- Pythonだけでは非常に時間がかかる
  Numpyでできることであればよいが、
  そうでない場合は. . .

PythonとC/C++

- Pythonはスクリプト言語
  - 可読性が高い
  - コンパイルの必要なし
  - 実行速度は速くない
- C/C++はコンパイラ言語
  - 修正のたびに再コンパイルが必要
  - 実行速度は速い

- プログラム全体の流れはPythonで記述し、
  実行速度の必要な部分をC/C++で記述すれば、
  効率的に解析システムを構築できる

PythonとC/C++をつなぐツール

- boost.python
- swig
- ctypes
- 直接インターフェイスを書く

- 使い勝手などを比較した訳ではないが、
  ここでは、swigを使ってみることにする

Swigの簡単な使い方

swig/example.*

- Cプログラムを用意する

サンプルCプログラム

swig/example.c

```
/* File : example.c */

#include <time.h>

double My_variable = 3.0;

int fact(int n) {
     if (n <= 1) return 1;
     else return n*fact(n-1);
}

int my_mod(int x, int y) {
     return (x%y);
}

char *get_time() {
     time_t ltime;
     time(&ltime);
     return ctime(&ltime);
}
```

Swigの簡単な使い方

swig/example.*

- Cプログラムを用意する
- Swig用インターフェイスファイルを用意する

Swigインターフェイスファイル

swig/example.i

```
/* example.i */
%module example      ← Pythonから呼ぶときのモジュール名
%{
/* Put header files here or function declarations like
below */
extern double My_variable;        ← コンパイルに必要な型宣言
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
%}

extern double My_variable;         ← これらがラップされる
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
```

Swigの簡単な使い方

swig/example.*

- Cプログラムを用意する
- Swig用インターフェイスファイルを用意する
- swigコマンドでラッパーファイルを作成する

```
% swig -python example.i⏎
example.py, example_wrap.c が作成される
```

- コンパイルする

```
% gcc -fPIC -c example.c example_wrap.c -I/adc/local/include/
python2.6/⏎
example.o, example_wrap.o が作成される
```

- shared objectを作成する

```
% gcc -shared example.o example_wrap.o -o _example.so⏎
_example.so が作成される
```

- コンパイルに必要なコマンドは今後 compile.sh にまとめてあります。

サンプルの実行

swig/sample.py

```
import example

print example.cvar.My_variable
print example.fact(4)
print example.my_mod(10,3)
print example.get_time()
```

- グローバル変数は
  <モジュール名>.cvar.<変数名>
- その他の関数はPythonの関数を呼び出すのと同じ形式

Swigの応用例(1)

output/example.*

- 出力の返し方
  - Cの関数でポインタの引数が結果の場合
    ```
    void add(double a, double b, double *result)
    ```
    インターフェイスファイルに
    ```
    %include “typemaps.i”
    %apply double *OUTPUT { double *result };
    ```
    typemaps.i を使って引数の使い方を指定できる
  - 結果の値を2つ返したい場合
    ```
    void mod(int a, int b, int *quo, int *rem)
    ```
    同じくインターフェイスファイルに
    ```
    %include “typemaps.i” # 実際は1回だけでいい
    %apply int *OUTPUT { int*quo, int *rem };
    ```

Swigの応用例(1)

output/sample.py

- 実行

```
import example

a = example.add_0(3,4)
print a

b = example.add(3,4)          ← 通常の関数と同じように
print b                         結果を受け取れる

quo, rem = example.mod(7,3)   ← リストで結果を受け取れる
print quo, rem
```

Swigの応用例（２）

vector/example.*

- C++の vector クラスを使いたい

```
double average(std::vector<int> v)
std::vector<double> half(const std::vector<double>& v)
```

- インターフェイスファイルに

```
%include “std_vector.i”
%template(IntVector) std::vector<int>;
%template(DoubleVector) std::vector<double>;
```

Swigの応用例(2)

vector/sample.py

- 実行

```
from example import *

iv = IntVector(4)            ← vector<int>を作成
for i in range(0,4):
    iv[i] = i
print average(iv)

print average([0,1,2,3])   ← リストでも呼べる

print half([1,2,3])
```

- Swigのマニュアルは

  http://www.swig.org/Doc1.3/index.html

Numpy arrayをCへ

numpy/rms.*

- Numpy arrayのrmsをCで計算したい
- Cの関数形は
  `double rms(double *seq, int n)`
- Pythonからは
  `v = rms(array)`
  のように呼び出したい
- numpy.iという用意されたtypemapsを利用

Numpy arrayをCへ

numpy/rms.i

- ```
  %module rms
  %{
  #define SWIG_FILE_WITH_INIT     ← おまじない
  #include “rms.h”
  %}

  %include “numpy.i”              ← おまじない

  %init %{
  import_array();                 ← おまじない
  %}

  %apply (double *IN_ARRAY1, int DIM1) {(double *seq, int n)};
  %include “rms.h”      ↑
                    1次元の入力arrayの意味
  ```

- 詳しくは
  http://projects.scipy.org/numpy/export/3714/trunk/numpy/doc/swig/numpy_swig.pdf
  ```

CからNumpy array

ezrange/ezrange.i

- Cで作成した numpy array を Python に返す
- `%apply (double *IN_ARRAY1, int DIM1) {(double *seq, int n)};`
  のかわりに
  `%apply (int *ARGOUT_ARRAY1, int DIM1) {(int *rangevec, int n)};`
  を使う　　↑
  引数として返す1次元array
- 要素数はPythonから渡してやる必要がある

２次元arrayへの拡張

numpy2/stats.*

- 2次元のarray(=画像)を受け取って、平均・分散を計算する
- `%apply (double *IN_ARRAY1, int DIM1) …` は
  `%apply (double *IN_ARRAY2, int DIM1, DIM2) …` へ
- Cの関数の側ではarrayは1次元配列
  `double mean(double *array, int n, int m)`
- Numpyで計算したのと、実行時間を比較してみる

画像処理のサンプル

- PythonとC++を組み合わせて簡単な画像解析のプログラムを作成する
- 手順
  1. 複数の画像からフラット画像を作成する
  2. ある画像をフラット画像で割り算する
  3. フラット済みの画像をスムージングする
  4. スムージングされた画像でピークを検出

Imageクラス

image0/Image.*

- Numpy array から以下のメンバを持ったImageクラスを作成できるようにする
  - Numpy arrayのvectorの処理の仕方が分からなかった
- ```
  class Image {
      int npx, npy, npix;
      double *data;
  public:
      Image();
      Image(double *array, int n, int m);
      ~Image();
      int getNpx() const;
      int getNpy() const;
      int getNpix() const;
      void getNaxis(int *npx, int *npy);
      double getValue(int x, int y) const;
  };
  ```
  ```

Imageクラス

image0/Image.*

- コンストラクタ
  データアレイの領域を確保して、データをコピーする

```
Image::Image(double *array, int n, int m) {
   data = new double[n*m];              // 領域の確保
   for (int j = 0; j < n; j++) {
      for (int i = 0; i < m; i++) {
         data[i+j*m] = array[i+j*m];  // 値のコピー
      }
   }
   npx = m; npy = n; npix = n*m;
}
```

Imageクラス

image0/sample.py

- 画像をpyfitsで読み込んで、Imageクラスのオブジェクトを作成する
  `img = Image.Image(hdulist[0].data)`
- ピクセルの値が正しく得られるか確認する
  - ds9で読み込んで、その表示と一致するか？

- クラスのメンバにポインタが入っているのでコピーコンストラクタ、代入演算子を正しく定義しておく → image01/Image.*
- エラーチェックなどは考えていないので注意！

フラット画像の作成

image1/Image.*

- 画像のリストをファイルから読み込んで、
  それぞれの画像のmedianを測定し、
  その値で割り算して、
  画像間で同じピクセルのmedianを取る
- フラット作成には通常15-20枚の画像を使うが、
  今回は6枚だけ
- medianの計算、割り算、画像間のmedianを
  取るメソッドなどを追加する

フラット画像の作成

image1/Image.*

- 以下のメソッドを追加する

```
Image::Image(int npx, int npy)
```
画像の大きさだけを指定して、データ領域を確保するコンストラクタ。

```
void Image::setValue(int x, int y double v)
```
ピクセルの値をセットする

```
void Image::getData(float *oarray, int n)
```
データ領域を numpy array として取り出す

```
Image& Image::operator/=(double a)
```
すべてのピクセルの値をaで割る

```
double Image::median()
```
画像のmedianを計算する

```
static Image Image::getMedian(std::vector<Image>& v)
```
vに含まれる画像間でピクセルごとにmedianを取った画像を作成する

フラット処理、ピーク検出

image2/Image.*

- 作成したフラット画像でフラット補正を行い、
  簡単なスムージングをして、
  ピークを検出する
- 各過程の画像をds9に表示し、
  検出したピークの場所をマークする
- スムージングは ｎｘｎピクセル内の平均値を
  計算する単純なもの

フラット処理、ピーク検出

image2/Image.*

- 以下のメソッドを追加する

```
Image& Image::operator/=(const Image& a)
```
ピクセルごとに画像の割り算する

```
Image Image::boxavg(int width)
```
各ピクセルに対してその周り width x width の領域で平均を取った画像を作成する

```
std::vector<Point> findPeaks(double thres)
```
ピークの値がthres以上のピークを検出して、その位置をvectorで返す
ピークの検出は縦、横のピクセルの値と比較して大きいかどうか

- ピークの位置を返すためにPointクラスも作成

```
class Point {
    int x, y;
public:
    Point(int x, int y);
    int getX();
    int getY();
}
```

応用演習

- Imageクラスに加減乗除演算をすべて実装してみる
- medianだけでなく、平均、分散などの統計量の計算も実装してみる
- スムージングをGaussianでできるようにしてみる
- 検出したピークのカウント値に応じてマークの大きさを変えるようにしてみる

まとめ

- Swig を使って、C/C++のプログラムをPython から使えるようにする方法を学習した
- Swig を使ったPythonとC++を併用した解析プログラムを作成した

- より高度なことをやりたい方はマニュアルを参照して、挑戦して下さい。