Python+Jupyter notebookによる光赤外天文データ解析入門 (2017年2回目)

「IRAFなどでFITSのデータ処理を行い、Rなどで統計処理を行い、Gnuplotなどで描画する。それらの手順を手書きのノートやPCのファイルに書き留める(あるいは記憶しておくつもりでどんどん忘れていく)。インタラクティブにも行うし、スクリプトで一括処理をすることもある。」これらをPython+Jupyter notebookのエコシステムにまとめてみませんか？その一歩あるいは数歩を踏み出すための講習会です。

開催日 :

- 2017年 8月24日(木)-25日(金)

場所 : 国立天文台 三鷹キャンパス 南棟2階共同利用室

内容 : Python+Jupyter notebookを用いての光赤外データ(撮像)解析の初歩。
2017年7月に行った1回目の内容より変更しています(より初心者むけ + 演習の時間をより多く) 。
- 今回の講習を受けたのちに1回目の資料 (https://github.com/yas-nakajima/adc2017python) を自習することでステップアップできます。

概要 : Jupyter notebookの使い方、PythonからIRAFを使う方法、Numpy、astropy.io.fits、matplotlibなどデータ処理に必要なPythonモジュールの基本を講習し、PyRAFの使い方、撮像データのアパーチャ測光、そのデータ整理、可視化を実習する。対話的なコマンドの使い方からPythonプログラミングの基本までを習得する。

対象 : データ処理初心者。

- 資料(Jupyter notebook形式のファイル)をこのサイトで公開しています。この内容であれば講習を受けてみたいと思う人なら、学部生からシニアまで誰でも。
    - githubの.ipynbファイルのリンクをクリックするとブラウザで閲覧可能。
    - ここの上部の**Wiki**のリンクにjupyter notebookの使い方などの補足説明もあります。
    - 講習までに多少の修正・加筆がありますが、本質的な内容に大きく変更はありません。
    - この手のものに慣れている人はこの資料だけで自習できちゃうかもしれませんね。どうぞご利用ください。
- 必須
    - **Linux/UNIX(Mac**を含む)の**OS**での基本的な操作(ターミナルを開く、アプリを立ち上げる**...)**が可能である。
    - Linux/UNIX(Macを含む)のコマンドラインで、ls, cd, more(less), cp, rm, mkdir くらいはできる。
    - テキストエディタが使える。
- 望ましい経験など

    - IRAFを触ったことがある。
    - プログラミングをかじったことがある。(Pythonでなくてもよい)

日程　進行状況によって変更することもあります

- 1日目
    - 13:00 - 13:30 受付
    - 13:30 - 講習開始 世話人の話（計算機の説明ほか）
    - 13:40 - 17:00 (演習と休憩をはさみながら)
        - Jupyter notebookの使いかた
        - IRAF(PyRAF)基本
        - PyRAFでのアパーチャ測光
- 2日目
    - 9:30 - 11:30 (演習と休憩をはさみながら)
        - Numpy 基本
        - astropy.io.fits 基本
    - 13:00 - 14:30 (演習と休憩をはさみながら)
        - matplotlib 基本
        - scriptにまとめる等
    - 14:30 - 17:00 予備 ・相談
        - 相談とは
            - 自分の環境へのインストールに困っている
            - こういうデータ処理・解析をPythonを用いて行いたいが、どうしたら良いか？

端末 : 基本的に天文データセンターの端末(Linux)を使用する。持ち込みノートPCで実習も可。その場合にはあらかじめ必要なソフトをインストールしておくこと。Python(3.5), Jupyter notebook, IRAF, PyRAF, Numpy, astropy, matplotlib, ds9

補足：

スタータパック

Pythonには多くのライブラリが用意されています。数値計算、データ可視化をはじめ、データベース、機械学習など様々なものが利用できます。天文関係だけにしぼったとしても「さあ、Pythonで天文データ処理を始めよう」と思った場合に、どれを使えばよいのか、どこから始めればよいのか、迷ってしまいます。この講習会では光赤外データ(撮像)用の「スタータパック」を提案します。

対話的処理とスクリプトによる処理

この講習会では、ユーザインタフェースとしてJupyter notebookを使い、インタラクティブ(対話的)な処理を通じて処理方法を学びます。インタラクティブといっても、Jupyter notebookでは一行のコマンド(でもできますが)というよりも数行のコマンドからなるCellという単位で処理の入力を行います。Cellの中では、forループやif文、関数定義などを使ったプログラミングを行います。対話的に試行錯誤しながら作成したCellを後からまとめてやり、.pyという拡張子をもつテキストファイルに保存してやれば、独立したPythonプログラム(スクリプト)が出来上がります。対話的なコマンドの使い方だけではなくPythonプログラミングの基本も学ぶことになります。

書き残し共有する

Jupyter notebookはユーザインタフェース以上の役割があります。Jupyter notebookで行った対話的な処理(入力と出力)はファイルとして保存することが可能です。メモやコメントも(TeX形式で数式も)その中に書き込んで残すことも可能です。日々のデータ処理・解析の足跡をファイルに残しておくことができます。データ処理の手順や使ったパラメータなどを未来の自分のために残す。共同研究者あるいは同分野の人たちと情報を共有する。そのための強力なツールになります。

講習資料

講習0 --- Pythonとnotebook

この講習資料の使い方の説明を兼ねて、Pythonのインデントとモジュールの解説、およびnotebookの変数読み込みについて解説します。

Pythonのインデントとモジュール

C言語などの他のプログラミング言語の経験があれば、Pythonの習得は簡単です。
ただし、いくつかPythonに特徴的な事項があります。

一番顕著なのは、インデント(字下げ)が意味を持つということでしょう。
forループの構造を紹介しながら説明します。

もう一つは、モジュールのimportとモジュール内の関数の使い方です。

インデント

forループ

C言語と比べるとずいぶん勝手がちがいます。

1. インデントされたブロックが、繰り返される部分です。
2. リストの各要素を順番に繰り返し変数に代入して使います。

In [1]:
```
for i in [0,1,2,3]:
    print (i)
```

```
0
1
2
3
```

In [2]:
```
sum = 0
for i in range(6):  #   range()関数は0から引数の一つ前までの整数を順に返す
    print (i)
    sum += i
print (sum)
```

```
0
1
2
3
4
5
15
```

モジュールと関数

標準の組み込み関数(print()、range()、open()など)をのぞいて、関数を使用する際には、親となるモジュールをimportする必要があります。

mathモジュールのsqrt()関数を使う場合には次のようにします。

```
In [3]: import math
```

```
In [4]: math.sqrt(30)
Out[4]: 5.477225575051661
```

. を「の」と読んで、**math(**モジュール**) . (**の**) sqrt()** とコマンドします。

```
In [5]: math.pi
Out[5]: 3.141592653589793
```

```
In [6]: a = math.e
```

```
In [7]: print (a)
        2.718281828459045
```

関数だけではなく、定数も含まれています。

notebookの変数の読み込み

このipynbファイルを開いてすぐに、このすぐ上の **print(a)** のセルを実行([Shift]+[enter])してみてください。 エラーがでます。 「aなんて定義されていない」、と言われちゃいます。ファイルを開いただけでは、.ipynbの中の文字列が表示されただけなのです。 変数の読み込み、モジュールのimportなどはちゃんと実行してやらないといけません。

講習では、話を聞きながら、資料をなぞる感じで、一つづつCellを実行していってはいかがでしょうか？面倒な場合には、上部のメニューバーの Cell > Run All を選ぶと、ノートブック内のCellが上から順番に全て実行されます。

ただし、講習**1**では、ds9を立ち上げて、そこに画像を表示させて、、、という対話的な部分があるので、そのような場合には、Run Allは途中で止まってしまいます。

講習1 --- IRAFを使ってみる

旧来のIRAFの操作は、ターミナルからCLコマンドラインを使って対話的に行うものです。
この対話的な操作を、python + Jupyter notebookを使っても行うことができます。ここでは、IRAFの基本的なタスク、**display, imexam, imstat** を使ってみます。

pyrafのための準備

pythonからIRAFを使うにはpyrafを利用します。
pyrafを(便利に)使うためには、

- ホームディレクトリにirafというディレクトリを作成
- そのディレクトリ内で mkirafを実行し、login.cl を作成
- 必要に応じてそのlogin.clを編集
  (今回のサンプルデータの場合、26行目あたりの'#set stdimage = imt800'を
  'set stdimage = imt4096'にしておくとよいでしょう。冒頭の#を取り除き、800を4096に)

をしておきます。 こうしておくと、login.clでの設定がpyraf利用時に反映されます。また、~/iraf/uparm/に各タスク(imexam, imstatなど)のパラメータが保存されます。

補足 : ~/.iraf/login.cl がある場合には、そちらの設定が使用されます。
自分のホームディレクトリにIRAFをローカルインストールした場合、~/.iraf というディレクトリが作成されます。
~/iraf/login.clを編集したのに、それが反映されない場合は、 ~/.iraf/login.clを調べてみてください。

モジュールの読み込み

In [1]:
```
from pyraf import iraf
```

これでirafのタスクをpythonで関数として使うことができます。
(IRAFの外部パッケージstsdasがきちんとインストールされていない場合、'Warning : sscanf library not installed on ... 'がでます。とりあえず無視してください。)

サンプルデータ

サンプルデータとして、すばるSuprime Camで取得したデータを使います。 (SMOKAで公開されているデータです。) 同じディレクトリの data1 および data2 の中のFITSファイルです。

この講習では、2004-06-21の観測データの一部を利用します。フィルターはBバンドで、ドームフラットおよび二種類のターゲットの生データです。Suprime Camは10枚のCCDから成ります。ファイル名(拡張子除く)の末尾の数字がCCDの番号を示しています。 data1には5番フレーム、data2には2番フレームのデータがあります。例えば、data1/SUPA00317705.fitsとdata2/SUPA00317702.fits は同じ積分のそれぞれ、5番フレームと2番フレームのデータです。講習のなかでは主に5番フレームを用いて説明をします。演習の中で2番フレームを使うことがあります。

FITSファイル	**OBJECT**	フィルター	積分時間(秒)
data1/SUPA003175[0-6]5.fits	ドームフラット	B	10
data1/SUPA00317705.fits	target1	B	10
data1/SUPA00317885.fits	target2	B	30
data2/SUPA003175[0-6]2.fits	ドームフラット	B	10
data2/SUPA00317702.fits	target1	B	10
data2/SUPA00317882.fits	target2	B	30

<img src='./img/SUPA0031770X.png', width='500'>

Unixコマンド

Unixコマンドを使い、カレントディレクトリおよびその中身を確認することができます。

```
In [2]: pwd
```

```
Out[2]: '/Users/nakajima/git/adc2017python2'
```

```
In [3]: ls data1
```

```
SUPA00317505.fits  SUPA00317535.fits  SUPA00317565.fits
SUPA00317515.fits  SUPA00317545.fits  SUPA00317705.fits
SUPA00317525.fits  SUPA00317555.fits  SUPA00317885.fits
```

いくつかのUnixコマンドは、このように、jupyter notebookで直接使えます。使えないものもあります。その場合、! を冒頭につけて使用してやります。

```
In [6]: ! date
```

```
2017年 8月21日 月曜日 10時01分36秒 JST
```

それでは、IRAFのコマンド、display, imexam, imstatを使って、それらファイルの表示やデータの吟味、統計量の測定をしてみましょう。

DS9にFITSデータを表示してみる

別のターミナルからds9を立ち上げておきます。
irafモジュールのdisplay()関数を使い、引数としてファイルを指定します。target1の5番フレームを表示してみます。

In [7]:
```
iraf.display('data1/SUPA00317705.fits')
```

```
z1=10012.29 z2=10105.81
```

ds9に、星がいくつか写っている視野が表示されましたね。
(**FAQ** ~/iraf/login.cl で set stdimage = imt800 のままだと視野の中心付近の800x800の領域しか表示されません。)

imexam

ds9にFITS画像を表示した状態で、次のコマンドを実行し、マウスカーソルを、星のないところにあてて**m**、あるいは星にあてて**a**や**r**をタイプしてみましょう。終わるときには**q**をタイプしてください。

mは、カウスカーソルを当てた部分の5x5ピクセルの範囲のカウント値の統計を表示します。バックグラウンドの値、ばらつきなどを調べるときに使います。
aは、星(点光源)の特徴量(ピーク値、fwhmなど)を表示します。
rは、星のradial profileを表示します。 星がサチっていれば一目でわかります。

In [8]:
```
iraf.imexam()
```

```
#            SECTION     NPIX     MEAN   MEDIAN   STDDEV      MIN      MAX
[1170:1174,1963:1967]       25   10060.   10057.    14.95   10041.   10115.
#   COL    LINE     COORDINATES
#     R    MAG    FLUX     SKY    PEAK    E   PA BETA ENCLOSED   MOFFAT DIRECT
1238.67 1913.38 1238.67 1913.38
  21.61  13.08  58358.  10059.    868. 0.04   22 3.00     7.18     7.20   7.20
 21.61  13.08  58358.  10059.    868. 0.04   22 3.00     7.18     7.20   7.20
```

imexamでは、**r**とか**e**コマンドを使うとさらに別のグラフィックウィンドウが現れます。ブラウザとかの後ろに隠れているかもしれません。

imstat

FITS画像の統計量を調べるタスクimstatを使ってみましょう。

In [9]:
```
iraf.imstat.unlearn()  #  パラメータをデフォルト値に
```

In [10]:
```
iraf.imstat('data1/SUPA00317705.fits')
```

```
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
 data1/SUPA00317705.fits   8528000    10059.     108.3     9911.    43932.
```

eparでパラメータ設定

iraf.epar('タスク名**')** で、パラメータ設定の画面が別ウィンドウで開きます。GUIでパラメータ設定ができます。
(2017-07 MacOS Sierra + Python3.5.3 + Jupyter Notebook では iraf.epar('display')などでUnicodeDecodeErrorが生じる)

In [11]:
```
iraf.epar('imstat')
```

```
Task imstatistics is running...

#     MIDPT      MEAN    STDDEV
     10058.    10059.     108.3
```

このように、GUIでもパラメータ設定ができるのですが、ここでは次の方法をおすすめします。

変数としてパラメータ設定

Cellでタスクの変数にパラメータを代入してやります。この方法だと、使ったパラメータがこのノートブックに残るので、あとになって「この処理でどんなパラメータ使ったっけ？」となったときに助けになります。

In [12]:
```
iraf.imstat.fields = 'midpt, mean, stddev'
iraf.imstat.lower = 9950
iraf.imstat.upper = 11000
```

In [13]:
```
iraf.imstat('data1/SUPA00317705.fits')
```

```
#     MIDPT      MEAN    STDDEV
     10058.    10058.     15.38
```

どんな値がパラメータに入っているかを確認するには、それをprint()してやればよいです。

In [14]:
```
print (iraf.imstat.fields)
```

```
midpt, mean, stddev
```

結果の値を変数へ

上ではimstat()の結果が標準出力に表示されました。それぞれの値を変数に保存するには次のようにします。

In [15]:
```
out = iraf.imstat('data1/SUPA00317705.fits', format='no', Stdout=1)
# format='no' でヘッダ行非表示、Stdout=1で戻り値を返す

v = out[0].split()  #  戻り値はリスト
median = float(v[0])   # 文字列をfloatに変換しておく
mean = float(v[1])
stddev = float(v[2])

print (median)
print (mean)
print (stddev)
print (median + 3 * stddev) # floatに変換しておかないとここでおかしなことになる
```

```
10058.02
10057.7
15.38449
10104.17347
```

デフォルトではStdout=0なので標準出力に値が返ります。戻り値を変数として返す場合にはStdout=1とします。

helpドキュメント

このノートブック内でhelpを読むこともできます。
helpの表示が縦に長すぎる場合、左の余白部分(In[]: の下あたり)をクリックするとスクロールバーつきウインドウ表示になります。 (私の環境でブラウザChromeの場合にはデフォルトでスクロールバー表示されます。)

In [16]:
```
#iraf.help('imstat')   #  Githubでは表示が長くなってしまうのでコメントアウトしておきます
```

演習1

data1/SUPA00317885.fitsはtarget2の生データです。
新しいノートブックファイルを作成し、

1. imexamでバックグラウンドの値とばらつき、星の特徴量、を調べる。
2. imstatでカウント値のmedian, mean, standard deviationを求める。
3. imstatのnclipを2以上にしてみてください。どうなりましたか？

講習2 --- IRAFで1次処理

IRAFのタスクを使って、１次処理(生データからバイアスを引き、それをフラットで割る) を行ってみましょう。
ここでは、IRAFの基本タスク、**imarith**, **imcombine**を使います。

ドームフラットでターゲットのフレームをフラット処理する (簡易版)

講習1で扱った'./data1/SUPA00317705.fits' はtarget1を観測した生データです。これを、バイアス値を引いたあとに、フラットで割ることで、CCDの感度ムラと光学系の透過率のムラを補正します。視野全体で一様な強度で光っている(と考えている)ものを観測してフラットを作成します。 (最近の)CCDではダークを生データから引くことはしません。(地上観測の近赤外アレイでは、ダークを引いてフラットで割り、スカイバイアスを引くという処理が必要になります。)

まず、このフラットを作成する必要があります。ドームフラットの生データからバイアス値を引いたのちに規格化(メジアン値で割る)してフラットを作成します。もし、CCDの感度ムラがなく、光学系の透過率も完全に一様であれば、フラットは全てのピクセルで1.0の値をもちます。でもそんなことはまずあり得ません。通常は複数のフラットの平均から、より尤もらしいフラットを作成します。

下ではまず、ドームフラット1枚だけからフラット作成します。**imstat**と**imarith**を使用します。
次に、ドームフラット7枚からフラット作成します。**imcombine**を使用します。

ドームフラット**1**枚だけを使う

'./data1/SUPA00317505.fits' はドームフラットのBバンドの生データです。 CCDのフレームには、観測した光に加えて、X方向に一様なバイアス値が加算されています。この簡易版の処理では、そのバイアス値がY方向にも一様として処理をします。(のちにY方向の依存も考慮に入れた手法を紹介します。) そのバイアス値をオーバースキャン領域から推定します。Suprime Camの5番フレームでは2049列目あたりから右側がオーバースキャン領域です。

In [2]:
```
from pyraf import iraf
```

In [3]:
```
iraf.unlearn('imstat')
iraf.imstat.fields = 'midpt, mean, stddev'
iraf.imstat.nclip = 3
```

In [4]:
```
iraf.imstat('./data1/SUPA00317505.fits[2049:2080, *]')
```

```
#     MIDPT      MEAN    STDDEV
      9984.     9984.     5.046
```

ドームフラットの光があたっている部分のメジアンを求めておきます。

In [5]:
```
iraf.imstat('./data1/SUPA00317505.fits[1:2048, *]')
```

```
#     MIDPT      MEAN    STDDEV
     18933.    18928.     210.1
```

imarithの出番です。バイアス値を引いてから、メジアン値で規格化してフラットを作成しましょう。

In [6]:
```
iraf.imarith('./data1/SUPA00317505.fits', '-' , 9984, 'bflatn5a.fits')
iraf.imarith('bflatn5a.fits', '/' , 8949, 'bflatn5a.fits')  #  18933 - 9984 =
8949
```

このフラットでターゲットの生データを割ります。このときも、まず、生データからバイアス値を引きます。

In [7]:
```
iraf.imstat('./data1/SUPA00317705.fits[2049:2080, *]')
```

```
#     MIDPT      MEAN    STDDEV
      9989.     9989.     4.841
```

In [8]:
```
iraf.imarith('./data1/SUPA00317705.fits', '-', 9989, 'btarget1n5a.fits')
iraf.imarith('btarget1n5a.fits', '/', 'bflatn5a.fits', 'btarget1n5a.fits')
```

これでできました。 btarget1n5a.fitsをds9で表示して確かめてみましょう。

ドームフラット**7**枚を使う
iraf.imcombineの出番です。
'./data1/SUPA003175[0-6]5.fits' はBバンドのドームフラットです。

In [9]:
```
import glob # pythonの組み込みモジュール。ワイルドカードを使ったファイル処理など。
```

In [10]:
```
flist = glob.glob('./data1/SUPA003175[0-6]5.fits')
```

In [11]:
```
print(flist)
```

```
['./data1/SUPA00317505.fits', './data1/SUPA00317515.fits', './data1/SUPA0031752
5.fits', './data1/SUPA00317535.fits', './data1/SUPA00317545.fits', './data1/SUP
A00317555.fits', './data1/SUPA00317565.fits']
```

forループを使って、このリストからファイルを一つづつimstatに入力します。

In [12]:
```
for img in flist:
    iraf.imstat(img + '[2049:2080, *]')
```

```
#     MIDPT      MEAN    STDDEV
      9984.     9984.     5.046
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.039
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.036
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.032
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.038
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.056
#     MIDPT      MEAN    STDDEV
      9983.     9983.     5.071
```

上では、glob.globで抽出したリストをいったん変数に代入しましたが、以下のように直接forループに入れても大丈夫です。

In [13]:
```
iraf.imstat.fields = 'midpt'  #  どうせメジアンしか使わない

for img in glob.glob('./data1/SUPA003175[0-6]5.fits'):
    out1 = iraf.imstat(img + '[2049:2080, *]', format='no', Stdout=1)
    out2 = iraf.imstat(img + '[1:2048, *]', format='no', Stdout=1)
    print (out1, out2)
```

```
['9983.81'] ['18932.82']
['9983.021'] ['18950.5']
['9982.991'] ['18963.1']
['9983.005'] ['19029.06']
['9982.896'] ['18995.43']
['9982.938'] ['18935.77']
['9982.917'] ['18969.67']
```

out1, out2は、それぞれ、1つしか要素を持たないリストとして得られました。
下のように、リストの最初の要素を抽出することで値を得ることができます。ただし、文字列です。

In [14]:
```
out1[0]
```

Out[14]: '9982.917'

float()関数で数値(浮動小数点数)に変換してやります。

In [15]:
```
float(out1[0])
```

Out[15]: 9982.917

それでは、各ドームフラットからフラットを作成し、それらをメジアンでコンバインします。

In [16]:
```
iraf.imstat.fields = 'midpt'

num = 0
comstr = ''
for img in glob.glob('./data1/SUPA003175[0-6]5.fits'):

    out1 = iraf.imstat(img + '[2049:2080, *]', format='no', Stdout=1)  #  オーバースキャン領域
    out2 = iraf.imstat(img + '[1:2048, *]', format='no', Stdout=1)  #  光のあたってる領域
    med1 = float(out1[0])  #  文字列を数値に変換
    med2 = float(out2[0])

    nflat =  'tmp' + str(num) + '.fits'   #  それぞれのフラットを作成
    iraf.imarith(img, '-', med1, nflat)  #  バイアス値をひく
    iraf.imarith(nflat, '/', med2 - med1, nflat) #  バイアスを考慮して規格化

    num += 1
    comstr += nflat + ','  #  imcombineの引数として与えるための文字列

print (comstr)  #  なぜ下でcomstr[:-1]と、末尾の一文字を削除するか理解するためにあえて表示

iraf.imcombine(comstr[:-1], 'bflatn5.fits', combine='median')
iraf.imdelete(comstr[:-1])  #  中間ファイルを削除。お掃除お掃除。
```

```
tmp0.fits,tmp1.fits,tmp2.fits,tmp3.fits,tmp4.fits,tmp5.fits,tmp6.fits,

Aug 21 13:22: IMCOMBINE
  combine = median, scale = none, zero = none, weight = none
  blank = 0.
                  Images
               tmp0.fits
               tmp1.fits
               tmp2.fits
               tmp3.fits
               tmp4.fits
               tmp5.fits
               tmp6.fits

  Output image = bflatn5.fits, ncombine = 7
```

これでドームフラットを7枚使ったフラットができました。
生データをこれで処理してやります。

In [17]:
```
iraf.imarith('./data1/SUPA00317705.fits', '-', 9989, 'btarget1n5.fits')
iraf.imarith('btarget1n5.fits', '/', 'bflatn5.fits', 'btarget1n5.fits')
```

演習**2**

2-1.
別ターゲットtarget2を観測した、'./data1/SUPA00317885.fits' について、バイアス引き+フラット割りの処理をしましょう。これは5番フレームです。フィルターも同じBバンドなので、フラット割りには、'bflatn5.fits'が使えます。
この結果のフレームを'btarget2n5.fits'と呼ぶことにします。(後の演習で利用します)

2-2.
'./data1/SUPA00317705.fits'と同じ観測の2番フレームの生データ'./data2/SUPA00317702.fits' について、バイアス引き+フラット割りの処理をしましょう。先ほどの5番フレームとは違い、これは2番フレームなので、2番フレームのためのフラットを作成する必要があります。
(1) './data2/SUPA00317502.fits' を規格化したものをフラットとして作成する。(1枚フラット)
(2) './data2/SUPA003175[0-6]2.fits' から平均のフラットを作成する。
(3) 上のどちらか(あるいは両方)のフラットを使って、バイアス引き後のフラット割りを行う。

注意 : 2番フレームはオーバースキャン領域が5番とは異なる。

補足 バイアス値の**Y**方向依存も考慮に入れる

今回使用するフラットでは0.1 x 数パーセントの違いしかありませんが、Y方向の依存も考慮に入れた方法を紹介しておきます。ドームフラットのデータを1枚だけ使うケースを例にします。 irafのblkavgを使って、オーバースキャン領域の各lineの算術平均を求めます。(本当はメジアンがよいが。)

```
In [18]: iraf.blkavg('./data1/SUPA00317505.fits[2049:2080, *]', 'bias1.fits', 32, 1)
```

bias1.fitsはサイズが(1,4100)の1次元データです。これをX方向に2080倍のばします。

```
In [19]: iraf.blkrep('bias1.fits', 'bias2.fits', 2080)
```

このバイアスを生データから引いてやります。

```
In [20]: iraf.imarith('./data1/SUPA00317505.fits', '-' , 'bias2.fits', 'bflatn5by.fits'
         )
```

```
In [21]: iraf.imstat('bflatn5by.fits[1:2048, *]', fields='midpt')
         #     MIDPT
               8957.
```

```
In [22]: iraf.imarith('bflatn5by.fits', '/', 8957, 'bflatn5by.fits')
```

バイアスのY方向依存も考慮に入れたフラットができました。
次にターゲットのフレームをフラットで割ります。
ここでもバイアス値のY方向依存を考慮にいれます。

In [23]:
```
iraf.blkavg('./data1/SUPA00317705.fits[2049:2080, *]', 'bias1t.fits', 32, 1)
iraf.blkrep('bias1t.fits', 'bias2t.fits', 2080)
iraf.imarith('./data1/SUPA00317705.fits', '-' , 'bias2t.fits', 'btarget1n5by.f
its')
iraf.imarith('btarget1n5by.fits', '/', 'bflatn5by.fits', 'btarget1n5by.fits')
```

講習3 --- 星の測光

点光源の明るさを測定してみます。ここでは、IRAFのAPPHOTを用いて、アパーチャ測光を行います。

準備のトリミング

講習2で作成した、btarget1n5.fitsをトリミングして、オーバースキャン領域など不要な部分を除いておきます。 (左端に明るい部分が見られます。これは何らかのバイアスがのっているものと思われるのでここ(25列目まで)も除きます。)

In [1]:
```
from pyraf import iraf
```

In [2]:
```
iraf.imcopy('btarget1n5.fits[25:2048, *]', 'btarget1n5trim.fits')
```

```
btarget1n5.fits[25:2048,*] -> btarget1n5trim.fits
```

パラメータ設定

iraf.apphot関連のタスクのためのパラメータを適切に設定する必要があります。

準備

- 星のサイズ(fwhm)を求めておく
- 背景のレベルとばらつきをもとめておく

ds9を立ち上げておき、iraf.displayとiraf.imexamで星のfwhmを調べておきます。

このあとの作業では、'btarget1n5trim.fits'に対して繰り返し処理をおこなうので、

In [3]:
```
targetfits = 'btarget1n5trim.fits'
```

とファイル名を変数に代入しておきます。

In [4]:
```
iraf.display(targetfits)
```

```
z1=28.35372 z2=115.3357
```

In [5]:
```
iraf.imexam()
```

```
#   COL    LINE    COORDINATES
#     R    MAG    FLUX     SKY    PEAK    E   PA BETA ENCLOSED   MOFFAT DIRECT
1394.04  921.73 1394.04 921.73
  20.88  13.07  58984.   69.29   882.7 0.03   62 4.67     6.90     7.07   6.96
1384.47  706.26 1384.47 706.26
  21.55  12.25 125778.   69.22   1827. 0.03    9 15.8     6.97     7.37   7.18
 411.91 1136.10 411.91 1136.10
  21.58  12.33 116908.   69.13   1629. 0.06   23 4.76     7.20     7.66   7.19
```

fwhm=7.0 pixel としておきます。

次に背景のmedianとノイズの評価をします。

In [6]:
```
iraf.unlearn('imstat')
iraf.imstat.fields = 'midpt, mean, stddev'
iraf.imstat.nclip = 3
```

In [7]:
```
iraf.imstat(targetfits)
```

```
#     MIDPT      MEAN    STDDEV
      68.84      69.1     7.368
```

メジアンは69でノイズは7.4とします。

パラメータの設定

apphotモジュールをimportします。

In [8]:
```
from iraf import apphot
```

login.cl の記載内容によっては、

```
from iraf import noao
from iraf import digiphot
```

も必要かもしれません。

次に、測光に必要なパラメータを設定します。
パラメータ設定の参考にした文献は、"A Reference Guide to the IRAF/DAOPHOT Package"です。
http://iraf.noao.edu/docs/photom.html (http://iraf.noao.edu/docs/photom.html)
からダウンロード(daorefman.pdf)できます。

In [9]:
```
med = 69.  #  背景レベルとばらつき、fwhm
std = 7.4
fwhm = 7.0

iraf.apphot.unlearn()  #  デフォルト値に戻しておく

iraf.apphot.datapars.datamax = 50000  #  サチった星を数えない
iraf.apphot.datapars.readnoise = 10 #  検出器に特有な値
iraf.apphot.datapars.epadu = 2.5 #  検出器に特有な値
iraf.apphot.datapars.itime = 10  # 積分時間

iraf.apphot.findpars.threshold = 7  #  7シグマ以上のものを検出せよ
iraf.apphot.findpars.sharphi = 0.8  # 星っぽくないものを除くため

#  fwhmで決まるパラメータ
iraf.apphot.datapars.fwhmpsf = fwhm
iraf.apphot.centerpars.cbox = max(5.0, fwhm)
iraf.apphot.fitskypars.annulus = 3 * fwhm
iraf.apphot.photpars.apertures = 2 * fwhm

iraf.apphot.fitskypars.dannulus = 10.

#  背景のレベルとばらつきで決まるパラメータ
iraf.apphot.datapars.sigma = std
iraf.apphot.datapars.datamin = med - 5 * std

iraf.apphot.photpars.zmag = 27  # 等級のゼロ点

# IRAFと対話的(確認など)に行わないための設定
iraf.apphot.daofind.interac = 'no'
iraf.apphot.daofind.verify = 'no'
iraf.apphot.phot.interactive = 'no'
iraf.apphot.phot.verify = 'no'
iraf.apphot.phot.verbose = 'no'
```

まずは、iraf.daofind()で星を検出させます。

In [10]:
```
iraf.daofind(targetfits, output='out1.coo')
```

どれが星として検出されたかFITS上にプロットしてみましょう。
(線が細いので半径を3つ指定することで3つのマルを描く。)

In [11]:
```
iraf.tvmark('1', 'out1.coo', mark='circle', radii='15,16,17', color=207)
```

In [13]:
```
#iraf.help('tvmark')
```

星じゃないものも検出されちゃってますが、ここでは気にせず、daofindの出力の'out.coo'を **iraf.phot()**に読み込ませて測光します。

In [14]:
```
iraf.phot(targetfits, coords='out1.coo', output='out1.mag')
```

最初の引数は、測光する対象のFITSファイル名です。coords=で読み込ませる座標ファイルの名前、output=で結果を書き出すファイル名を指定します。

結果のファイルは、iraf.phot()固有の形式で書き出されています。

In [15]:
```
#cat 'out1.mag'   # githubでは長くなるのでコメントアウトしておきます
```

通常、必要なのは、xcenter, ycenter, mag, merr です。**iraf.txdump()**を使って、それらだけを抜き出します。

In [16]:
```
iraf.txdump('out1.mag', fields='xc,yc,mag,merr')
```

```
1384.487  706.380  16.778  0.003
1394.061  921.854  17.593  0.005
724.860  986.570  15.692  0.001
412.063  1136.186  16.854  0.003
1256.533  1254.923  19.872  0.032
1614.107  1296.000  24.509  2.293
1.975  1563.123  INDEF  INDEF
325.259  1710.853  20.233  0.047
295.760  1891.867  INDEF  INDEF
1214.740  1913.393  17.615  0.005
164.208  2090.797  20.722  0.073
1519.869  2131.413  19.975  0.035
755.494  2207.461  20.578  0.061
1609.915  2403.143  17.285  0.004
888.608  2443.525  21.548  0.153
400.753  2577.342  INDEF  INDEF
2008.147  2638.538  16.281  0.002
782.873  2801.123  18.178  0.008
1781.279  2834.660  18.671  0.012
261.980  2855.986  13.514  0.000
1841.312  2952.747  20.019  0.040
687.466  3180.978  13.860  0.001
752.111  3381.985  15.486  0.001
1741.726  4037.847  21.283  0.124
1333.792  4043.156  13.562  0.000
1791.647  4044.367  18.121  0.008
650.189  4085.125  19.898  0.038
```

INDEFなんてのもあります。これは、たまたまバッドピクセルが測光領域に含まれていた、視野の端である、あるいは、サチった星などです。測光誤差を(例えば)0.2等以下のものだけに絞ることで、INDEFのものも削除できます。 誤検出のものもここで削除できます。

In [17]:
```
iraf.txdump('out1.mag', fields='xc,yc,mag,merr', expr='merr<0.2')
```

```
1384.487  706.380  16.778  0.003
1394.061  921.854  17.593  0.005
724.860  986.570  15.692  0.001
412.063  1136.186  16.854  0.003
1256.533  1254.923  19.872  0.032
325.259  1710.853  20.233  0.047
1214.740  1913.393  17.615  0.005
164.208  2090.797  20.722  0.073
1519.869  2131.413  19.975  0.035
755.494  2207.461  20.578  0.061
1609.915  2403.143  17.285  0.004
888.608  2443.525  21.548  0.153
2008.147  2638.538  16.281  0.002
782.873  2801.123  18.178  0.008
1781.279  2834.660  18.671  0.012
261.980  2855.986  13.514  0.000
1841.312  2952.747  20.019  0.040
687.466  3180.978  13.860  0.001
752.111  3381.985  15.486  0.001
1741.726  4037.847  21.283  0.124
1333.792  4043.156  13.562  0.000
1791.647  4044.367  18.121  0.008
650.189  4085.125  19.898  0.038
```

Stdout="でファイル名を指定すると、そのファイルに書き出してくれます。

In [18]:
```
iraf.txdump('out1.mag', fields='xc,yc,mag,merr', expr='merr<0.2', Stdout='resu
lt1.txt')
```

これで、星の機械等級がもとまりました。

測光値の較正

ここで求まったものは、機械等級です。等級のゼロ点を適当に設定したものです。 どうやって本当の等級に直せば良いでしょうか。

実は、この視野には標準星が写っています。Landolt(1992)のカタログの標準星のうち以下が写っています。

<img src='./img/SA104.png', width='200'>

標準星	カタログ**B**等級(エラー)	上記結果(機械等級)	**3**列 **- 2**列	カタログ**B-V**	測定回数(カタログ)
SA104-330	15.894 (0.029)	17.593 (0.005)	1.699	0.594	15
SA104-334	13.998 (0.006)	15.692 (0.001)	1.694	0.518	24
SA104-335	12.292 (0.010)	13.860 (0.001)	1.568	0.622	4
SA104-336	15.230 (0.010)	16.854 (0.003)	1.624	0.830	14
SA104-L2	16.700 (0.033)	18.178 (0.008)	1.478	0.650	4

「3列 - 2列」の等級較正値にはばらつきがあります。 Landoltカタログで測定回数が10回以下のものは、ここでは、信頼性が低いとして採用しないことにします。 (それでもなお、SA104-336はSA104-330およびSA104-334と比べて0.07等も較正値が異なります。これはSA104-336のB-Vの値が他の2つと比べて大きいので、カラー変換の影響を考慮にいれなければいけないのかもしれません。)

結果として、較正値の平均は1.672等、標準偏差は0.03等となりました。 従って、result.txtの等級から1.672を引き、等級エラーには0.03等の誤差伝搬を加えておけばよいことになります。

In [19]:
```
import math

with open('result1c.txt', 'w') as fout: # result_c.txtを書き出し先ファイルとする
    with open('result1.txt') as fin:  #  result.txtを開く
        for line in fin:  #  1行づつ読み込み
            v = line.rstrip().split()  #  rstrip()で改行コードを削除し、split()で空白文字で行を分割
            mag = float(v[2]) - 1.672 #  等級は3列目なので、それをfloatに変換して1.672を引く
            merr = math.sqrt(float(v[3])**2 + 0.03**2)
            print (v[0], v[1], mag, merr, file=fout) #  書き出し
```

次の講習でとりあげる**numpy**を使うと、上のテキストファイル(result.txt)の読み出し、計算、書き出しのプログラムが非常に簡単になります。

演習**3**

演習2-1で処理をした'btarget2n5.fits'で測光をしてみましょう。
このときも、オーバースキャン領域などの不要な部分を削除して行いましょう。

'btarget2n5.fits'の視野の中には測光標準星は写っていません。ただし、上のtarget1と近い時間に観測したデータですので、等級ゼロ点は同じだと仮定し、上と同じ較正値(1.672)を使ってください。

講習4 --- Numpyの基本

この次に、IRAFを使わずにFITSファイルの処理をする方法を扱いますが、その準備としてNumpyの基本を知っておきましょう。

Numpyは、数値計算を効率よく処理するためのサードパーティモジュールです。 特に、多次元配列を取り扱う際に処理速度が速くなり、コードの表記も効率的になります。 pythonの標準の配列であるリスト型では処理が遅いため、科学計算ではNumpyのndarrayという多次元配列のデータ型を使います。Numpyはサードパーティモジュールですが、科学計算では標準的に使われます。このあと紹介するastropy.io.fitsおよびmatplotlibでも、このndarrayを採用しています。

以下、ndarrayの基本について説明し、numpyの基本的で使える関数について説明します。

リスト型

まずはpythonの標準のリスト型を見てみましょう。

In [1]:
```
a = [1, 2, 3, 4]
b = [10, 20, 30, 40]
```

+ 演算子はリストとリストの結合になります。* 演算子はリストの繰り返しを作成します。

In [2]:
```
a + b
```

Out[2]:
```
[1, 2, 3, 4, 10, 20, 30, 40]
```

In [3]:
```
a * 4
```

Out[3]:
```
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

ベクトル的に演算したい場合には、下のように、forループを回して各要素を取り出してから演算をする必要があります。

In [4]:
```
n = len(a)
c = [0] * n
for i in range(n):
    c[i] = a[i] + b[i]
c
```

Out[4]:
```
[11, 22, 33, 44]
```

In [5]:
```
print (c)
```

```
[11, 22, 33, 44]
```

Numpyのndarray

Numpyではもっとすっきりした処理、ベクトル処理、ができます。処理速度も速いです。

In [6]:
```
import numpy as np   #  一般的に np と省略される
```

numpy.array() 関数を使って、pythonのリストをndarray に変換します。

```
In [7]: an = np.array(a)
        bn = np.array(b)
```

```
In [8]: cn = an + bn
        cn
```

```
Out[8]: array([11, 22, 33, 44])
```

```
In [9]: print (cn)
```

```
[11 22 33 44]
```

```
In [10]: an * 4
```

```
Out[10]: array([ 4,  8, 12, 16])
```

部分のとりだし ~ スライシング **(直前追加分)**

ndarrayの一部分だけを取り出す場合には、スライシングをしてやります。

```
In [11]: myarr = np.array([[7, 3, 8], [13, 11, 16], [105, 121, 153]])
```

```
In [12]: myarr
```

```
Out[12]: array([[  7,   3,   8],
                [ 13,  11,  16],
                [105, 121, 153]])
```

1番目のインデックスは全部で、2番目のインデックスは1以上のものを取り出す場合には、

```
In [13]: myarr[:, 1:]
```

```
Out[13]: array([[  3,   8],
                [ 11,  16],
                [121, 153]])
```

2番目のインデックスも1以上のものを取り出す場合にはさらに、

```
In [14]: myarr[1:, 1:]
```

```
Out[14]: array([[ 11,  16],
                [121, 153]])
```

次の講習でFITSの一部分を取り出す場合に使います。

numpyの関数 **(1)** ~ 基本 **+** いくつか

numpyで用意されている関数は数多くあります。 cellの中で、np.とタイプしてタブキーを押すと候補が表示されます。あるいはnp.mでタブキーを押すとmで始まる関数の候補が表示されます。

```
In [15]: np.max(cn), np.min(cn), np.mean(cn)
```

```
Out[15]: (44, 11, 27.5)
```

numpy.where()

天文のデータ処理では(でなくても?)次の numpy.where( ) が便利です。 バッドマスク処理や外れ値の除外に使えます。

```
In [16]: dn = np.array([1, 1, 3, 1, 1])
```

np.where( ) 関数は、( )内の条件を満たす要素のindexを返します。 下の例では、2以上の値をとる要素に-99を代入するという意味です。

```
In [17]: dn[np.where(dn > 2)] = -99
```

```
In [18]: dn
```

```
Out[18]: array([  1,   1, -99,   1,   1])
```

次の講習で、3シグマクリップを使うときに登場します。

numpy.zeros( )

下のように、**numpy.zeros( )** を使って、全要素が0の5 x 5 の二次元配列を作成することができます。

```
In [19]: data = np.zeros((5, 5))
```

```
In [20]: data
```

```
Out[20]: array([[ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.]])
```

```
In [21]: data[0, 1] = 2   #   [0, 1]の要素に2を代入する。
```

numpyに限らず、pythonではindexは0から始まります。C言語と同じです。

```
In [22]: data
```

```
Out[22]: array([[ 0.,  2.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.]])
```

このdataの表示を見て気づいたかとも思いますが、左下を原点とした二次元画像としたときに、ndarrayでは **(y, x)**のように xとyのindexを逆にして要素が格納されます。後で、astropy.io.fitsのところでも再び触れます。

numpyの関数 **(2)** ~ 画像重ね合わせの準備

IRAFのimcombineを使わずに、astropy.io.fits + numpyで重ね合せる(つまり、複数画像の各ピクセルでのaverageやmedianを求めてcombineをする)ための準備として、numpy.stack(), numpy.average(), numpy.median() などを紹介します。

1次元

まずは1次元配列で見てみましょう。
3つの1次元配列、a, b, cがあるとき、

```
In [23]: a1 = np.array([1, 2, 3])
         b1 = np.array([2, 3, 4])
         c1 = np.array([2, 2, 5])
```

numpy.stack()関数を使うことでこれらをまとめて一つ次元の高い配列にする(スタックする)ことができます。

```
In [24]: s1 = np.stack((a1, b1, c1))
```

```
In [25]: s1
```

```
Out[25]: array([[1, 2, 3],
                [2, 3, 4],
                [2, 2, 5]])
```

そのうえで、配列を重ねてできた新しい次元の軸に沿ってmedianをとります。
上の表示例だと、縦方向にmedianをとります。
具体的には、**np.median([**スタックされた配列**], axis=0)** としてmedianの配列を得ます。

```
In [26]: meddata = np.median(s1, axis=0)
```

```
In [27]: print(meddata)
```

```
         [ 2.  2.  4.]
```

averageやsumも同様です。

```
In [28]: print (np.average(s1, axis=0))
```

```
         [ 1.66666667  2.33333333  4.        ]
```

In [29]:
```
print (np.sum(s1, axis=0))
```

```
[ 5  7 12]
```

axis=0を省略すると全ての要素について計算します。

In [30]:
```
print (np.average(s1))
```

```
2.66666666667
```

2次元

2次元の場合も同様です。

In [31]:
```
a2 = np.array([[1, 2, 1],[20, 21, 13]])
b2 = np.array([[2, 2, 2],[22, 23, 15]])
```

In [32]:
```
print (a2)
print ()
print (b2)
print ()
print (np.average(np.stack((a2, b2)), axis=0))
```

```
[[ 1  2  1]
 [20 21 13]]

[[ 2  2  2]
 [22 23 15]]

[[  1.5   2.    1.5]
 [ 21.   22.   14. ]]
```

(次に紹介する) astropy.io.fitsで読み込んだ画像の場合も同様にスタックをしてmedianやaverageをとります。

np.append()

最初からスタックする配列がそろっていれば上のように**np.stack()**を使うのが簡単ですが、通常のデータ処理の場合には、リストから次々にファイルを読みこんでスタックに追加していくことでより効率よくプログラミングができます。
次々にファイルを読み込んでスタックに追加する場合には、**np.append()**を使用します。

In [33]:
```
s2 = a1[np.newaxis, :]
s2 = np.append(s2, b1[np.newaxis, :], axis=0)
s2 = np.append(s2, c1[np.newaxis, :], axis=0)
```

In [34]:
```
s2
```

Out[34]:
```
array([[1, 2, 3],
       [2, 3, 4],
       [2, 2, 5]])
```

上で求めたs1と同じですね。

補足 **np.empty() (直前追加分)**

プログラム中で、forループで回す場合には、まず空のndarrayを作っておき、そこに追加していくほうが都合のいいことが多いです。その場合には、**np.empty()**を使います。

In [35]:
```
s3 = np.empty((0, 2, 3))
s3 = np.append(s3, a2[np.newaxis, :], axis=0)
s3 = np.append(s3, b2[np.newaxis, :], axis=0)
```

In [36]:
```
print (s3)
```

```
[[[  1.   2.   1.]
  [ 20.  21.  13.]]

 [[  2.   2.   2.]
  [ 22.  23.  15.]]]
```

In [37]:
```
print(np.average(s3, axis=0))
```

```
[[  1.5   2.    1.5]
 [ 21.   22.   14. ]]
```

numpyの関数 **(3)** ~ テキストデータの読み込み

Numpyがなくても、Pythonの標準機能でテキストファイルの読み書きはできますが、数値テーブルであることが分かっていれば、**numpy.loadtxt()**を使ってもっと簡単に読み込み、ndarrayに保存することができます。

In [38]:
```
np.set_printoptions(precision=3, suppress=True)  #  suppress=Trueで指数表示禁止。この行は必須ではない。
```

In [39]:
```
mlist = np.loadtxt('./sample/result1c.txt')
```

In [40]:
```
mlist
```

Out[40]:
```
array([[ 1384.487,   706.38 ,    15.106,     0.03 ],
       [ 1394.061,   921.854,    15.921,     0.03 ],
       [  724.86 ,   986.57 ,    14.02 ,     0.03 ],
       [  412.063,  1136.186,    15.182,     0.03 ],
       [ 1256.533,  1254.923,    18.2  ,     0.044],
       [  325.259,  1710.853,    18.561,     0.056],
       [ 1214.74 ,  1913.393,    15.943,     0.03 ],
       [  164.208,  2090.797,    19.05 ,     0.079],
       [ 1519.869,  2131.413,    18.303,     0.046],
       [  755.494,  2207.461,    18.906,     0.068],
       [ 1609.915,  2403.143,    15.613,     0.03 ],
       [  888.608,  2443.525,    19.876,     0.156],
       [ 2008.147,  2638.538,    14.609,     0.03 ],
       [  782.873,  2801.123,    16.506,     0.031],
       [ 1781.279,  2834.66 ,    16.999,     0.032],
       [  261.98 ,  2855.986,    11.842,     0.03 ],
       [ 1841.312,  2952.747,    18.347,     0.05 ],
       [  687.466,  3180.978,    12.188,     0.03 ],
       [  752.111,  3381.985,    13.814,     0.03 ],
       [ 1741.726,  4037.847,    19.611,     0.128],
       [ 1333.792,  4043.156,    11.89 ,     0.03 ],
       [ 1791.647,  4044.367,    16.449,     0.031],
       [  650.189,  4085.125,    18.226,     0.048]])
```

In [41]:
```
np.savetxt('result1calib.txt', mlist, fmt='%.3f')
```

numpy.loadtxt()のオプションにて、ヘッダ行を無視、区切り文字(カンマ区切りやタブ区切り)、どの列を読み込むか、などを指定することができます。

演習4

(直前追加分)

(1) 下の二次元配列の演算をnumpyを用いて行ってください。 <img src='./img/ex4-1.png', width='300'>

(2) 下の3つの二次元配列のメジアンを、numpy.stack()とnumpy.median()を用いて求めてください。 <img src='./img/ex4-2.png', width='350'>

補足 ブロードキャスティング

ある2次元配列があるとき、そのどちらかの軸の大きさと同じ大きさの1次元配列の足し算、引き算をすると、もう一つの軸に沿って同じ演算をしてくれます。

In [42]:
```
data1 = np.array([0, 1, 2])
data2 = np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20], [30, 30, 30]])
```

In [43]:
```
data2 + data1
```

Out[43]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

In [44]:
```
data3 = np.array([0, 100, 200])
data2 + data3
```

Out[44]:
```
array([[  0, 100, 200],
       [ 10, 110, 210],
       [ 20, 120, 220],
       [ 30, 130, 230]])
```

次のastropy.io.fitsのところで、Y方向に依存したバイアス値を引くケースで使います。

補足 配列について

Pythonの配列には、リストとタプルがあります。
リストは [ ] で囲まれ変更可能なオブジェクトです。
タプルは( ) で囲まれ変更不可能なオブジェクトです。

In [45]:
```
aa = [1, 2, 3, 4]
```

In [46]:
```
aa[2] = 10
```

In [47]:
```
aa
```

Out[47]:
```
[1, 2, 10, 4]
```

In [48]:
```
bb = (10, 20, 30, 40)
```

In [49]:
```
bb
```

Out[49]:
```
(10, 20, 30, 40)
```

In [50]:
```
bb[2] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-c69fd0eaa82a> in <module>()
----> 1 bb[2] = 100

TypeError: 'tuple' object does not support item assignment
```

タプルの要素を書き換えようとしたら怒られました。

講習5 --- astropy.io.fitsの基本

FITSデータの読み書きが可能なモジュールです。 元々、pyfitsと呼ばれていたものです。astropyに吸収されてそちらで統一的に管理されるようです。
astropy.io.fitsの使い方は、 http://docs.astropy.org/en/stable/io/fits/index.html (http://docs.astropy.org/en/stable/io/fits/index.html) が非常に参考になります。 例が多く記載されているのでわかりやすいです。

PyRAFでもFITSデータの読み書きはできるのですが、ndarrayとしてデータを読み出しておくとより自由な処理を行うことができます。例えば、IRAFには存在しない統計アルゴリズムを適用する、あるいは、matplotlibを用いた自由度の高い可視化などです。

```
In [1]: from astropy.io import fits
```

まずastropy.ioからfitsをimportしておきます。

基本のgetdataとgetheader

まずは簡単なデータ読み出し方法。 **fits.getdata( )**と **fits.getheader( )**です。 主にインタラクティブなケースで使われます。

まずはピクセル値をgetdataでndarrayとして読み出します。

```
In [2]: data = fits.getdata('./data1/SUPA00317505.fits')  #  ドームフラットの生データ
```

```
In [3]: data
```

```
Out[3]: array([[15798, 15697, 14527, ...,  9985,  9981,  9976],
               [18091, 33233, 30777, ...,  9991,  9987,  9978],
               [15287, 31092, 25091, ...,  9986,  9981,  9979],
               ...,
               [19053, 36375, 25385, ..., 10004, 10007,  9997],
               [16965, 34236, 24216, ...,  9998, 10001, 10000],
               [18178, 32768, 22645, ..., 10015, 10008, 10013]], dtype=uint16)
```

ndarrayとして読み出すので、numpyの関数が使えます。

```
In [4]: import numpy as np
```

```
In [5]: np.median(data), np.std(data), np.min(data), np.max(data)
```

```
Out[5]: (18931.0, 1212.2635026788732, 9681, 38631)
```

オーバースキャンの部分も含めて統計をとっているので、標準偏差(np.std)が大きな値になっています。
オーバースキャンの部分を除いて統計をとってみましょう。

ndarrayなので、**[y, x]**の順に範囲を指定します。
[y_start:y_end, x_start:x_end] のように範囲をしていします。値を省略すると、「最初から」 あるいは 「最後まで」となります。左右両方を省略すると「全部」です。

In [6]:
```
subdata = data[:, :2045]
np.median(subdata), np.std(subdata), np.min(subdata), np.max(subdata)
```

Out[6]:
```
(18935.0, 409.56087590112406, 9739, 38594)
```

上限値と下限値を適切に設定し、3-sigmaクリッップすると標準偏差がもう少し小さくなります。

In [7]:
```
xx = np.where((subdata > 15000) & (subdata < 30000))
med = np.median(subdata[xx])
std = np.std(subdata[xx])

xx = np.where((subdata > med - 3 * std) & (subdata < med + 3 * std))
med = np.median(subdata[xx])
std = np.std(subdata[xx])
print ('{} {:.2f}'.format(med, std))   #   format()関数使ってみた
```

```
18934.0 213.65
```

次にgetheaderを使ってFITSヘッダを読み出します。

In [8]:
```
header = fits.getheader('./data1/SUPA00317505.fits')
```

ここで、header あるいは print(header) をcellで実行すると、FITSヘッダの中身が表示されます。

In [9]:
```
# header   # これをするとgithubでは表示が長くなるのでここではコメントアウト
```

最初の10項目だけを表示させます。

In [10]:
```
header[:10]
```

Out[10]:
```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 2080 / length of data axis 1
NAXIS2  =                 4100 / length of data axis 2
EXTEND  =                    F / FITS dataset may contain extensions
BZERO   =              32768.0 / offset data range to that of unsigned short
BSCALE  =                  1.0 / default scaling factor
BUNIT   = 'ADU     '           / Unit of original pixel value
BLANK   =               -32768 / Value used for NULL pixels
```

In [11]:
```
header['OBJECT']
```

Out[11]:
```
'DOMEFLAT'
```

このようにヘッダのキーワードを指定して、その値を取り出すことができます。

[小技**]**
実は、fits.getdata( )でヘッダも読み出すことができます。

In [12]:
```
tdata, thdr = fits.getdata('./data1/SUPA00317505.fits', header = True)
```

In [13]:
```
thdr[:10]
```

Out[13]:
```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 2080 / length of data axis 1
NAXIS2  =                 4100 / length of data axis 2
EXTEND  =                    F / FITS dataset may contain extensions
BZERO   =              32768.0 / offset data range to that of unsigned short
BSCALE  =                  1.0 / default scaling factor
BUNIT   = 'ADU     '           / Unit of original pixel value
BLANK   =               -32768 / Value used for NULL pixels
```

データの加工と書き出し

このドームフラット1枚を使って、フラットを作成しましょう。 バイアス値がY方向に一様とする簡易版で行います。
ここでは、IRAFを使わずnumpyを使って行います。

In [14]:
```
omed = np.median(data[:, 2049:])  #  オーバースキャン部分のメジアン
imed = np.median(data[:, :2048])  # 光が当たる部分のメジアン
data = ( data - omed ) / (imed - omed)  #  ゲタを引き、規格化
```

整数データを割り算した結果、浮動小数点数になります。 デフォルトでは64ビットになります。

In [15]:
```
data.dtype
```

Out[15]:
```
dtype('float64')
```

無駄にファイルサイズが大きくなるので、32ビットに変更します。

In [16]:
```
data = data.astype(np.float32)
```

ヘッダのBITPIXは自動的に変更されます。

ヘッダのOBJECTの文字列を変更してみます。

In [17]:
```
header['OBJECT'] = 'B_FLAT'
```

ヘッダキーワードのBLANKはデータが整数のときのみ有効です。
それ以外のときにこの項目があると、アプリケーションによってはWarningがでます。
ここでは削除しておきましょう。

In [18]:
```
del header['BLANK']
```

このデータとヘッダを新しいFITSファイルに書き出します。**fits.writeto()**を使います。

In [19]:
```
fits.writeto('bflatn5pa.fits', data, header)   #   p for python
```

フラットなんて中間ファイルなので盛りだくさんなヘッダは不要ですよ、という時には、fits.writeto()の引数のheaderを省略すると、最低限必要なヘッダを勝手に作ってくれます。

In [20]:
```
fits.writeto('bflatn5pa_simple.fits', data)
```

元のFITSファイルに上書き更新する場合には**fits.update( )**を使います。

In [21]:
```
data = fits.getdata('bflatn5pa.fits')
data[:, 2049:] = -999999.  # オーバースキャン部の値を負の大きな値にしておく
fits.update('bflatn5pa.fits', data, header)
```

fits.getdata( ) や fits.getheader( )、fits.writeto( )などは簡単で便利 (これらはastropy.io.fitsの中で'convinience functions'と呼ばれている) なのですが、効率の悪いことをしています。その都度にファイルのオープンとクローズをしています。

その理由から、一般にプログラムコードの中でFITSファイルの読み書きをする場合には、 **fits.open()** でファイルを開き、**.header**メソッドと**.data**メソッドを使ってヘッダーとデータを読み取ります。ここでは、fits.open()の講習は省略します。興味のある人はadc2017pythonの1回目のほうの資料を参照してください。

複数枚のドームフラットからフラットを作成

IRAFを使わずに、combineします。 numpy.median(), numpy.append()を使います。

In [22]:
```
import glob

stack = np.empty((0, 4100, 2080))  #  空の配列を作成

for img in glob.glob('./data1/SUPA003175[0-6]5.fits'):

    imdata = fits.getdata(img)
    omed = np.median(imdata[:, 2049:])
    imed = np.median(imdata[:, :2048])
    imdata = ( imdata - omed ) / (imed - omed)

    stack = np.append(stack, imdata[np.newaxis, :], axis=0)

immed = np.median(stack, axis=0)
immed = immed.astype(np.float32)

fits.writeto('bflatn5p.fits', immed)
```

ターゲットの生データのバイアス引きとフラット割り

target1の5番フレームの生データを、IRAFを使わずにastropy.io.fitsとnumpyで処理してみます。

In [23]:
```
imdata = fits.getdata('./data1/SUPA00317705.fits')
flat = fits.getdata('bflatn5p.fits')
flat[np.where(flat == 0.0)] = -9999   #   0での割り算を回避
omed = np.median(imdata[:, 2049:])
imdata = ( imdata - omed ) / flat
fits.writeto('btarget1n5p.fits', imdata)
```

トリミング

オーバースキャン領域はバイアス値の引き算が終わったあとは不要ですね。
切り取ってやりましょう。

In [24]:
```
imdata = fits.getdata('bflatn5p.fits')
fits.writeto('bflatn5ptrim.fits', imdata[:, :2048])
```

人工的な画像

ピクセル値が全部ゼロの10 x 10のFITSファイルを作り、いくつかのピクセルにだけ正の値を与えてやります。

In [25]:
```
imdata = np.zeros((10, 10), dtype='float32')  # 全て0の10x10のndarrayを作成
imdata[0, 1] = 100. #  (x,y)=(1,0)に100を代入。xとyが反転していることに注意
imdata[4, 6] = 50.

fits.writeto('my10x10.fits', imdata)
```

出来上がったFITSファイルをDS9で見てみましょう。そして、xとyが反転していることを確かめてください。

演習5

5-1. 上のmy10x10.fitsを作成し、DS9で見てxとyが反転していることを確かめてください。

5-2. astropy.io.fitsとnumpyを使って以下の処理をしてください。
(1) './data2/SUPA003175[0-6]2.fits'から2番フレーム用のフラットを作成。
(2) './data2/SUPA00317882.fits' について、バイアス引き+フラット割りの処理
(3) trimmingして(2)の結果からオーバスキャン部をとりのぞく。

補足 バイアス値をY方向も考慮に入れる (numpy編)

numpyを使うと、IRAFでやるよりもすっきり書けます。
まずはフラットを作成します。

In [27]:
```
imdata = fits.getdata('./data1/SUPA00317505.fits')

bias = np.median(imdata[:, 2049:], axis=1) # X軸に沿ってmedianを計算。biasはサイズ4100の1次元配列
bias = bias.reshape(4100,1) # ブロードキャスティングできるように整形

imdata = imdata - bias   #  ブロードキャスティングで引き算

med = np.median(imdata[:, :2045])

imdata = imdata / med
imdata = imdata.astype(np.float32)

fits.writeto('bflatn5pby.fits', imdata) # ヘッダは生データのものを継承しない
```

次に、生データもバイアス値のY方向依存を考慮して引きます。

In [31]:
```
flat = fits.getdata('bflatn5pby.fits')
flat[np.where(flat==0)] = -9999
tdata = fits.getdata('./data1/SUPA00317705.fits')
bias = np.median(tdata[:, 2049:], axis=1)
bias = bias.reshape(4100,1)

tdata = tdata - bias
tdata = tdata / flat

fits.writeto('btarget1n5pby.fits', tdata)
```

講習6 matplotlibの基本

matplotlibはデータ可視化のためのパッケージです。 http://matplotlib.org (http://matplotlib.org) のページの gallery のページには膨大な数のサンプルがあり、どんなグラフを作成できるのかを見る事ができます。一度ご覧になるのをお勧めします。さらに、それぞれのサンプルコードも見る事ができます。 galleryページのサンプルは膨大なので、代表的なものだけを取り上げた次のページを見るのがまずはいいかもしれません。 http://matplotlib.org/users/screenshots.html (http://matplotlib.org/users/screenshots.html)

ここでは、光赤外撮像データ解析によく使いそうな、ヒストグラム、等級-エラープロット、FITSデータの表示の例を紹介します。

inline表示

inline表示にしてやると、notebook内にグラフを表示することができます。下のように宣言しておきます。

In [1]:
```
%matplotlib inline
```

matplotlibパッケージの中で最もよく使うモジュールはpyplotです。matplotlib.pyplot as plt の省略がよく使われます。

In [2]:
```
import matplotlib.pyplot as plt
```

測光結果データをグラフ化する

講習3で得られた測光結果、result1c.txt には、(列1) x座標、(列2)y座標、(列3) 等級、(列4)等級エラーが記されています。
numpy.loadtxt()を用いてこのデータを読み込みます。

In [3]:
```
import numpy as np
```

In [4]:
```
mlist = np.loadtxt('result1c.txt')
```

ndarray形式で読み込まれます。array([[x, y, 等級, 等級エラー], [x, y, 等級, 等級エラー], ...])
ここでまずは欲しいのが、各星の等級と等級エラーのペアです。
このあとグラフ作成時の分かりやすさのために、magとmerrを分けて別々の配列にしておきます。
下のようにいくつか例を書き出してみると分かると思いますが、2番目のインデックスが2のものが等級で、2番目のインデックスが3のものが等級エラーです。

In [5]:
```
print (mlist[0,0], mlist[0,1], mlist[0,2], mlist[0,3])
print (mlist[1,0], mlist[1,1], mlist[1,2], mlist[1,3])
print (mlist[2,0], mlist[2,1], mlist[2,2], mlist[2,3])
```

```
1384.487 706.38 15.106 0.0301496268634
1394.061 921.854 15.921 0.0304138126515
724.86 986.57 14.02 0.0300166620396
```

In [6]:
```
mag = mlist[:, 2]
merr = mlist[:, 3]
```

[:, 2]は2番目のインデックスが2のものを全て取り出すという意味です。このようにして、特定の列だけを抽出して、magおよびmerrの配列に保存しておきます。

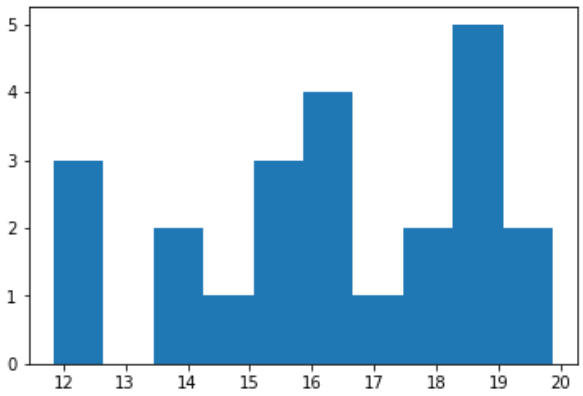
In [7]:
```
mag[:10]  # 長いので最初の10コだけ抽出
```

Out[7]:
```
array([ 15.106,  15.921,  14.02 ,  15.182,  18.2  ,  18.561,  15.943,
        19.05 ,  18.303,  18.906])
```

光度関数のヒストグラム

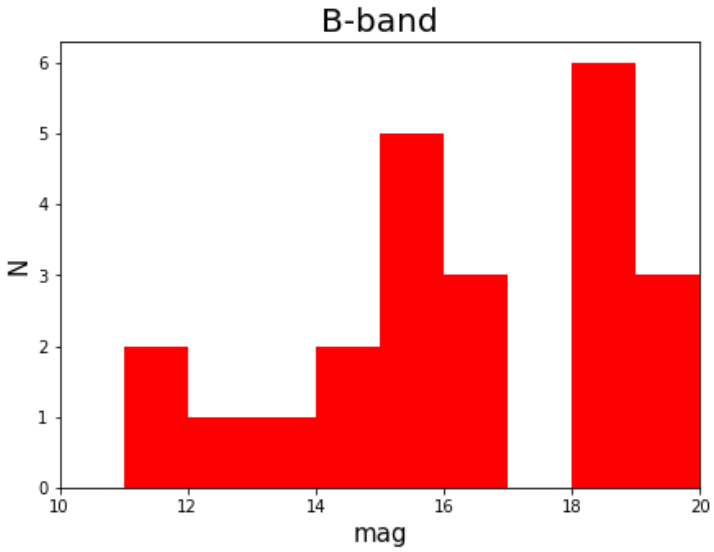
まずは何も考えずに等級のヒストグラムを描いてみます。**plt.hist()**を使います。

In [8]:
```
plt.hist(mag)
plt.show()
```



次に、オプションをいくつか加えてみます。

In [9]:
```
plt.figure(figsize=(7, 5))  #  図のサイズ
plt.hist(mag, bins=10, range=(10,20), color='red')  # ビンの数、ヒストグラムの範囲、色を指定
plt.xlim(10, 20) # グラフのX軸の範囲
plt.title('B-band', fontsize=20)
plt.xlabel('mag', fontsize=15)
plt.ylabel('N', fontsize=15)
plt.show()
```



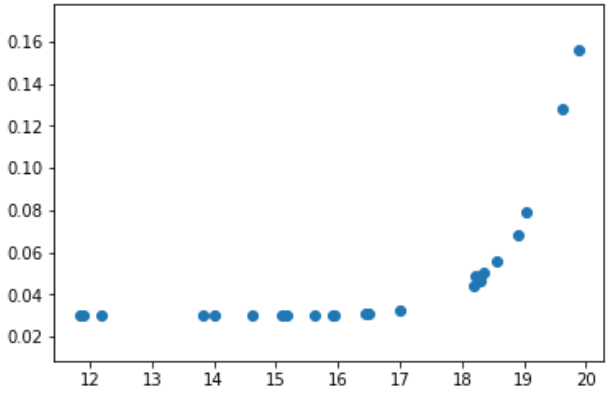
関数でどんな引数が使えるかを調べたい時には、下のように?をおしりにつけます。

In [10]:
```
plt.hist?
```

等級 **vs.** 等級エラープロット

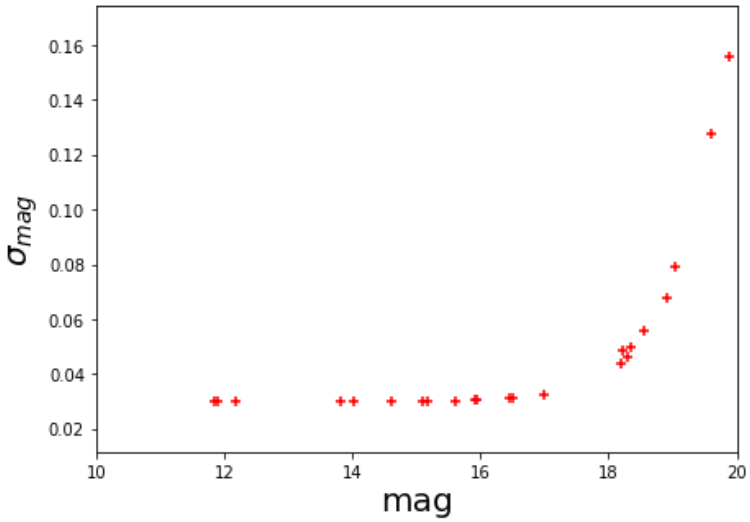
まずは何も考えずにプロット。**plt.scatter()**を使います。

In [11]:
```
plt.scatter(mag, merr)
plt.show()
```



これも、いくつかオプションを加えてみます。

In [12]:
```
plt.figure(figsize=(7, 5))
plt.scatter(mag, merr, marker='+', color='red')
plt.xlim(10, 20)
plt.xlabel('mag', fontsize=20)
plt.ylabel('$\sigma_{mag}$', fontsize=20) # TeXの表記がつかえます
plt.show()
```



FITS画像の表示

FITS画像をnotebook内に表示します。
そのためには、astropy.io.fitsでデータをndarrayとして読み込む必要があります。

In [13]:
```
from astropy.io import fits
```

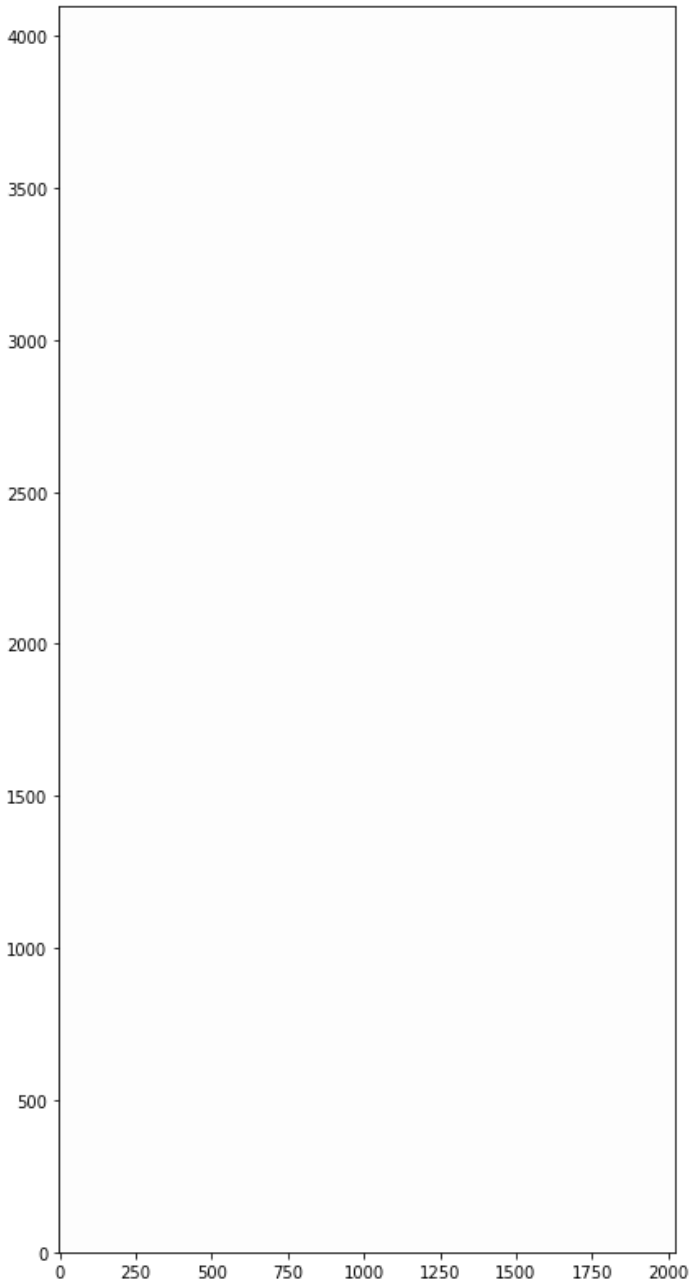
In [14]:
```
img = fits.getdata('btarget1n5trim.fits')
```

```
WARNING: VerifyWarning: Invalid 'BLANK' keyword in header.  The 'BLANK' keyword
is only applicable to integer data, and will be ignored in this HDU. [astropy.i
o.fits.hdu.image]
```

IRAFで処理したFITSヘッダにはBLANKというキーワードの行が残ってしいます。するとこのような警告が出ます。ここでは無視して大丈夫です。

まずは何も考えずに表示してみます。

In [15]:
```
plt.figure(figsize=(7, 14))
plt.imshow(img, plt.cm.gray, origin='lower', interpolation='none')
plt.show()
```



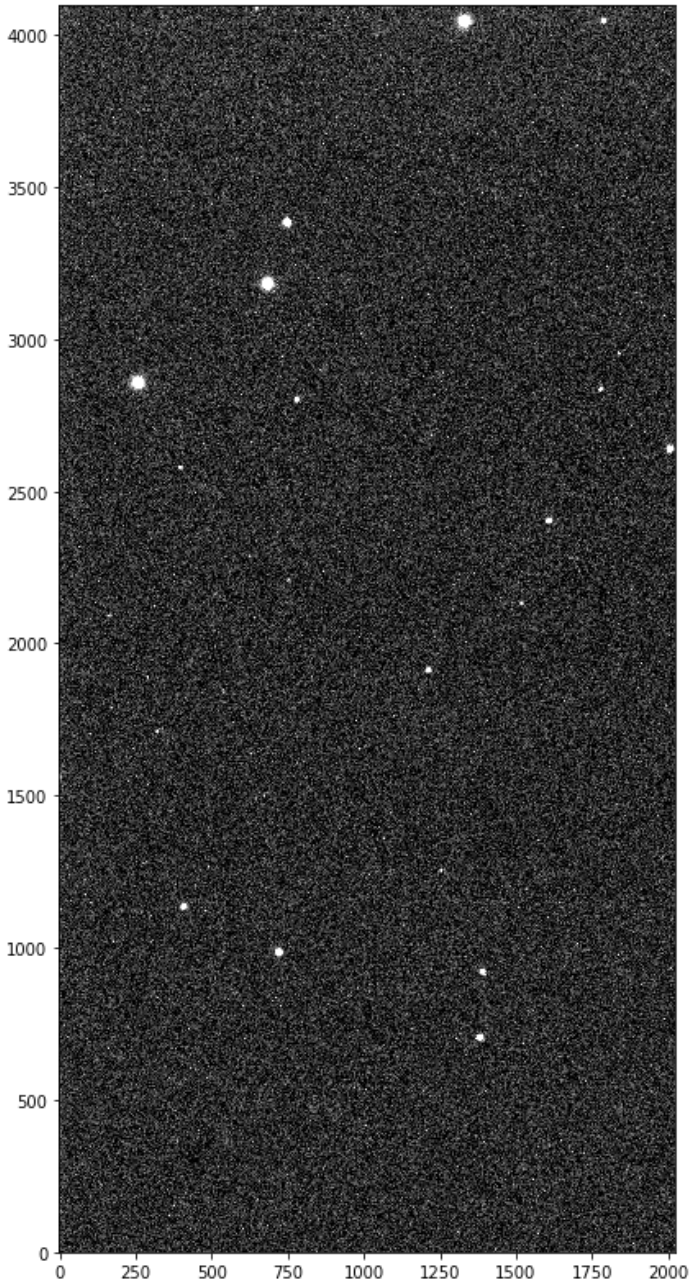
plt.cm.grayはカラーマップです。plt.cm.[カラーの名前]で指定します。
http://matplotlib.org/examples/color/colormaps_reference.html (http://matplotlib.org/examples/color/colormaps_reference.html)
上のままではよくわかりません。表示レベルをちゃんと設定してやります。

講習3で、btarget1n5trim.fitsのバックグラウンドのメジアンが69でばらつきが7.4であることを求めました。これをもとに表示レベルを設定します。

In [16]:
```
med = 69
std = 7.4
```

表示レベルの最小と最大(vminとvmax)をmed - std, および med + 5 stdに設定します。

In [17]:
```
plt.figure(figsize=(7, 14))
plt.imshow(img, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, origin='low
er', interpolation='none')
plt.show()
```



ここに印をいれてみます。 result1c.txtの中から最初の二つの星を選んでみます。

In [18]:
```
! head -2 result1c.txt
```

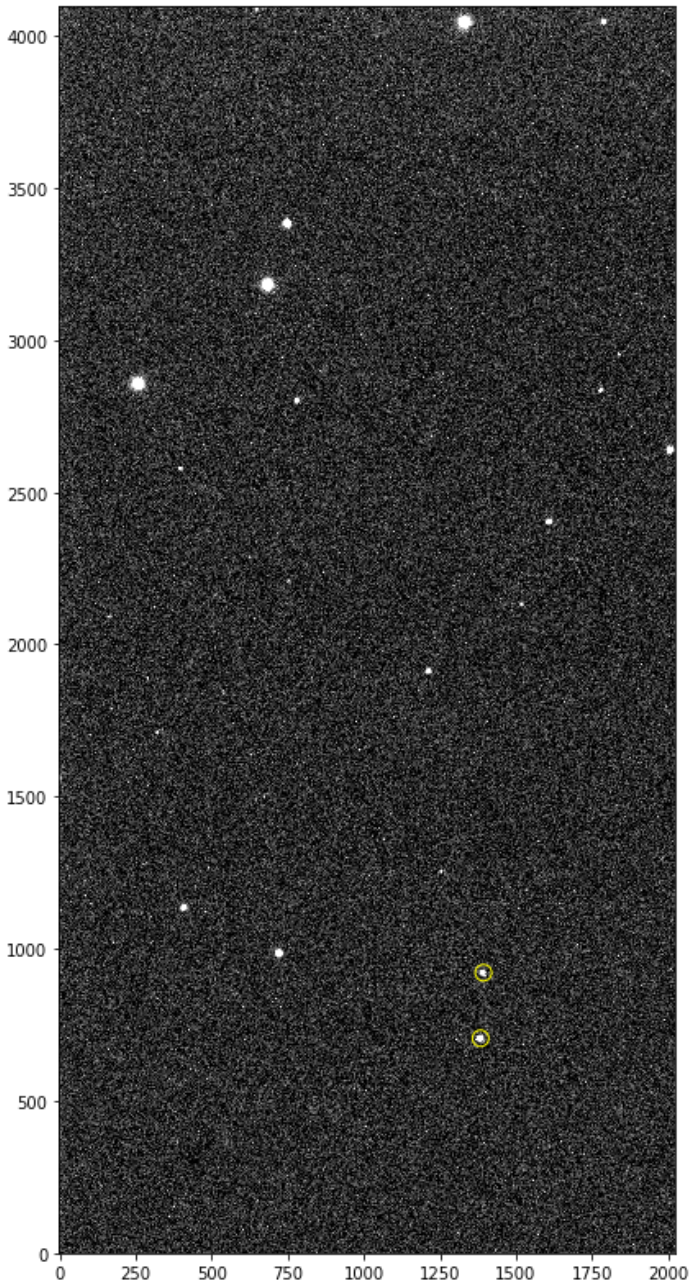
```
1384.487 706.380 15.105999999999998 0.030149626863362672
1394.061 921.854 15.921 0.0304138126514911
```

原点ピクセルのXY座標は、IRAFでは(1, 1)でPythonでは(0, 0)です。なので、下では座標値から1を引いてやります。

In [21]:
```
xcoo = [1383.5, 1393.1]
ycoo = [705.4, 920.9]
```

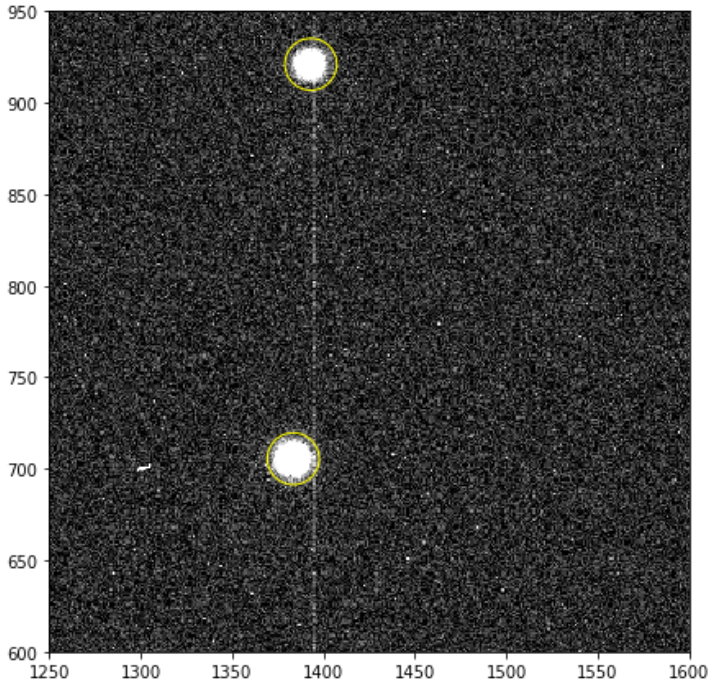
このxcoo, ycooをplt.scatter()でオーバープロットします。
xcoo, ycooのそれぞれの配列から順番にペアがプロットされます。等級vs等級エラーのときと同じです。
plt.scatter()のオプションのs=100は印の円の大きさです。面積で指定します。半径を倍にしたければ値を4倍にします。

In [22]:
```
plt.figure(figsize=(7, 14))
plt.imshow(img, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, origin='low
er', interpolation='none')
plt.scatter(xcoo, ycoo, edgecolors='yellow', facecolors='none', s=100)
plt.show()
```



標準星の周辺だけ拡大します。plt.xlim()とplt.ylim()が加わっただけです。
あとは、拡大したので印の大きさも変えました。

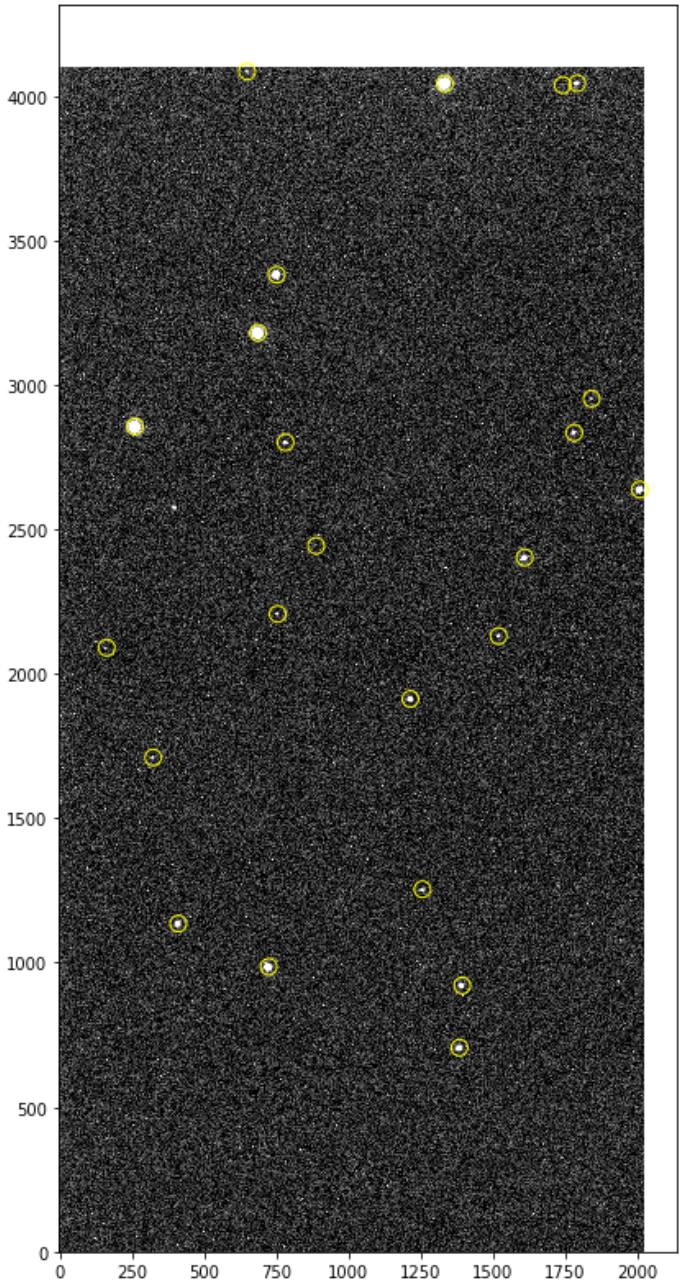
In [24]:
```
plt.figure(figsize=(7, 14))
plt.imshow(img, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, origin='low
er', interpolation='none')
plt.scatter(xcoo, ycoo, edgecolors='yellow', facecolors='none', s=1000)
plt.xlim(1250, 1600)
plt.ylim(600, 950)
plt.show()
```



せっかくなので、測光した全ての星をプロットします。

In [25]:
```
xstar = mlist[:, 0] - 1
ystar= mlist[:, 1] -1
```

In [28]:
```
plt.figure(figsize=(7, 14))
plt.imshow(img, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, origin='low
er', interpolation='none')
plt.scatter(xstar, ystar, edgecolors='yellow', facecolors='none', s=100)
plt.show()
#plt.savefig('myfig.png')   # 上のplt.show()をコメントアウトし、この行を実行するとファ
イルに保存できる。
```



演習4

演習3で行った'btarget2n5.fits'の測光結果を用いて、
(1) 「光度関数のヒストグラム」と 「等級vs等級エラーのプロット」を作成してください。
(2) FITS画像をnotebookに表示して、そこに測光した星をプロットしてください。

講習7 --- プログラムの使い回し ~ スクリプト作成等

ここまで、jupyter notebookでインタラクティブに処理を行い、いろいろなコマンドの使い方、簡単なpythonのプログラミングを見てきました。

次に、ひとかたまりのプログラムに汎用性をもたせるという観点で話をすすめます。

ここでは、まず、関数の定義の仕方から始め、pythonスクリプトの作成、コマンドライン引数の使い方、自作モジュールの使い方を説明します。

自作関数

ここまでの処理の中で、いくつかの処理の「かたまり」は、読み込むファイル名だけが変わって、繰り返し出てきました。
そのような「かたまり」は関数として定義しておくと効率的にコードを作成することができます。

<img src='./img/def_function.png', width='500'>

バックグラウンドのメジアンとノイズを、3シグマクリップして評価する処理なんていうのは、よく使うので関数にしておくと便利です。

In [1]:
```
import numpy as np
from astropy.io import fits
```

In [2]:
```
def getbackground(infits):

    data = fits.getdata(infits)
    med = np.median(data)
    std = np.std(data)

    for i in range(5):
        xx = np.where((data > med - 3 * std) & (data < med + 3 * std))
        med = np.median(data[xx])
        std = np.std(data[xx])

    return med, std
```

In [3]:
```
med, std = getbackground('sample/btarget1n5small.fits')
print(med, std)
```

```
69.3053 7.40289
```

FITSを表示する処理もよく使うでしょう。関数にしておきます。
ここでは、getbackground( )も中で使っています。

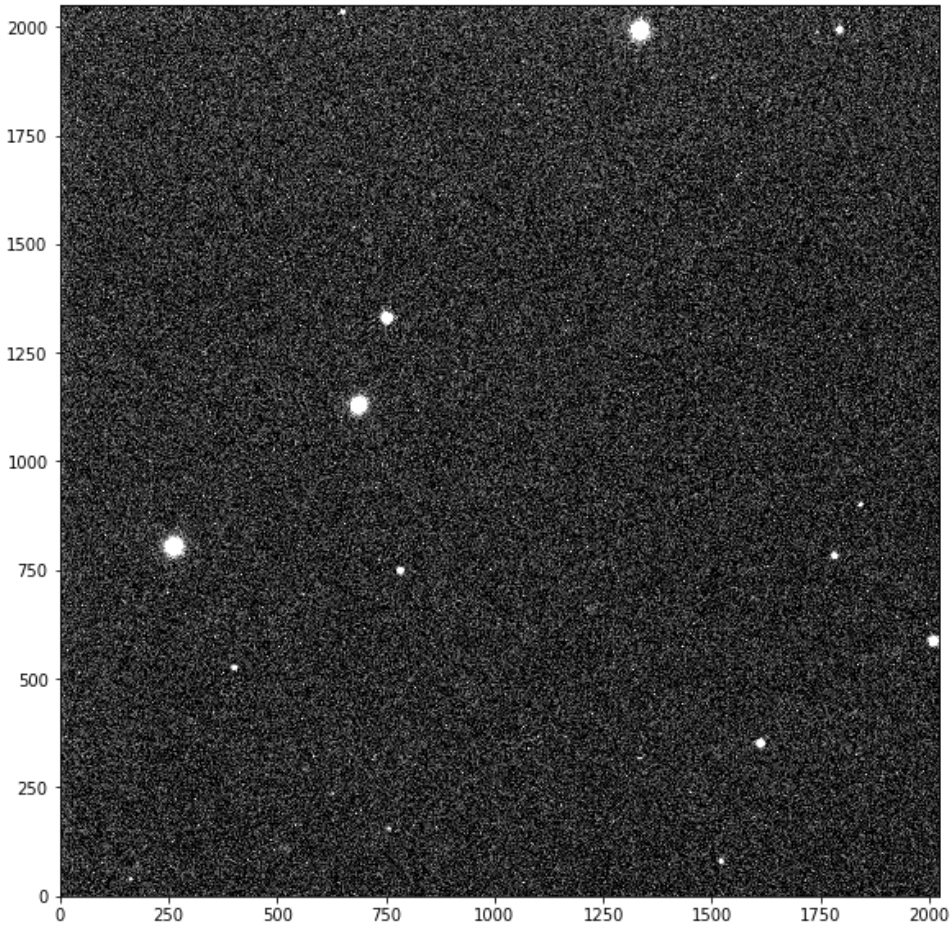
In [4]:
```
import matplotlib.pyplot as plt
```

In [5]:
```
def showfits(infits):

    med, std = getbackground(infits)
    imdata = fits.getdata(infits)

    plt.figure(figsize=(10, 10))
    plt.imshow(imdata, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, orig
in='lower', interpolation='none')
    plt.show()
```

In [6]:
```
showfits('sample/btarget1n5small.fits')
```



pythonスクリプトを作成

さて、notebookから飛び出して、pythonスクリプトを作成しましょう。
pythonスクリプトを作成するのは難しいことではありません。
テキストエディタで、セルに書き込んでいた内容を下記ならべ、拡張子が.py のファイルとして保存すればよいのです。
さらに、unix/linux系ではファイル冒頭に
#!/usr/local/bin/python3.5
のようにpythonのpathを書き込むのが一般的です。

In [7]:
```
import numpy as np

data1 = np.array([0, 1, 2])
data2 = np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20], [30, 30, 30]])

data3 = data2 + data1

print(data3)
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

これを、mycode.pyというファイルに書き込みましょう。

なんでもお好みのテキストエディタを使ってください。あるいは、jupyter notebookにはテキストエディタ機能もあります。jupyter notebook起動画面の右上のNew > Text File を選ぶと新規テキストファイルの画面になります。

そのうえで、(chmod +x してから)コマンドラインで実行します。あるいは下のようにcellからも実行できます。

In [8]:
```
%run mycode.py
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

コマンドライン引数

スクリプトに書き出したのであれば、コマンドライン引数も使いたくなります。

In [9]:
```
targetfits = 'sample/btarget1n5small.fits'
```

In [10]:
```
from astropy.io import fits
import numpy as np

data = fits.getdata(targetfits)
med = np.median(data)
std = np.std(data)

for i in range(5):
    xx = np.where((data > med - 3 * std) & (data < med + 3 * std))
    med = np.median(data[xx])
    std = np.std(data[xx])
    print ('{:.2f} {:.2f}'.format(med, std))
```

```
69.33 9.52
69.31 7.56
69.31 7.42
69.31 7.41
69.31 7.40
```

ここで、targetfitsをコマンドライン引数として読み込めれば、いろんなFITSファイルに対してこのプログラムが使えます。

コマンドライン引数を取り込むには

```
import sys
sys.argv
```

を使います。
コマンドラインに入力された文字列が空白で区切られて、リスト sys.argv に格納されます。
sys.argv[0] はプログラム名そのものです。1番目の引数はsys.argv[1]です。

In [11]:
```
cat getbackground.py
```

```
import sys
import numpy as np
from astropy.io import fits

data = fits.getdata(sys.argv[1])
med = np.median(data)
std = np.std(data)

for i in range(5):
    xx = np.where((data > med - 3 * std) & (data < med + 3 * std))
    med = np.median(data[xx])
    std = np.std(data[xx])
    print ('{:.2f} {:.2f}'.format(med, std))
```

In [12]:
```
%run getbackground.py sample/btarget1n5small.fits
```

```
69.33 9.52
69.31 7.56
69.31 7.42
69.31 7.41
69.31 7.40
```

モジュール作成

よく使う処理を、notebookの中で関数として定義して使用するのは楽チンなのですが、
notebookファイルを作成するたびに、前のnotebookからコピペして使うのはちょっと面倒ですね。

頻繁に使う自作の関数はモジュールにしておきましょう。

/home/nakajima/mypylib/mymodule.py のようなファイルを作成します。
必要なモジュールを冒頭でimportしておき、あとは自前の関数をどんどん書き込んでいけばよいです。
そして、他のプログラムから使うときには、

```
import sys
sys.path.append('/home/nakajima/mypylib/')
```

として、pathを通しておき、

```
import mymodule
```

を宣言します。 mymodule.pyの.pyはimportで呼ぶときには不要です。

ここでは、このディレクトリの中に'mypylib'というディレクトリを作成し、その中にmymodule.pyを作りました。

In [13]:
```
cat mypylib/mymodule.py
```

```
import numpy as np
from astropy.io import fits
import matplotlib.pyplot as plt

def getbackground(infits):

    data = fits.getdata(infits)
    med = np.median(data)
    std = np.std(data)

    for i in range(5):
        xx = np.where((data > med - 3 * std) & (data < med + 3 * std))
        med = np.median(data[xx])
        std = np.std(data[xx])

    return med, std


def showfits(infits):

    med, std = getbackground(infits)
    imdata = fits.getdata(infits)

    plt.figure(figsize=(10, 10))
    plt.imshow(imdata, plt.cm.gray, vmin=med - std, vmax = med + 5 * std, origi
n='lower', interpolation='none')
    plt.show()
```

次々とこの続きに、自分の関数を書き込んでいけばよいです。

さて、ここの例では次のようにしてpathを通し、

In [14]:
```
import sys
sys.path.append('./mypylib/')
```

mymoduleをimportして、mymodule.getbackground( ) を使います。

In [15]:
```
import mymodule
```

In [16]:
```
med, std = mymodule.getbackground('sample/btarget1n5small.fits')
print(med, std)
```

```
69.3053 7.40289
```

余談ですが、私は、/home/nakajima/mynotebook/ のようなディレクトリを作成し、その中に.ipynbファイルをまとめています。
データのあるディレクトリには、notebookの中で cd で change directoryして移動して処理を行います。

まとまっていると何がいいかというと、検索できるんですね。
ipynbファイルはJSONという形式のテキストファイルです。 grepで検索できます。

たくさんたまってくると、あれどこでやったかなあ。。。なんてこともよくあります。

補足説明
(wiki 資料)

1. はじめてのjupyter notebook

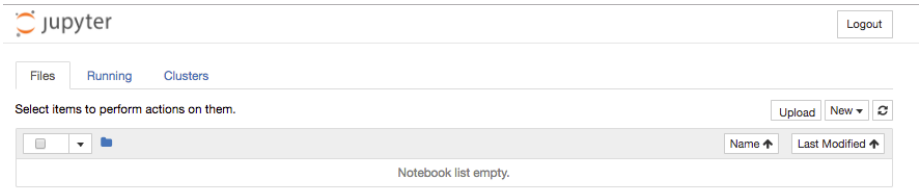
yas.nakajima edited this page 6 days ago · 1 revision

jupyter notebookは、もともとはipython notebookとよばれていたものをpython以外のプログラミング言語にも対応すべくグレードアップしたものです。julia, python, Rといった最近のデータサイエンスなんかで使われているプログラミング言語から文字をとってjupyterという名前が作られました。 https://github.com/jupyter/jupyter/wiki/Jupyter-kernels によると40以上のプログラミング言語に対応しているとのことです。もともとipython notebookだったので、jupyter notebookのファイルの拡張子は.ipynbになっています。

jupyter notebookの起動

コマンドラインで
jupyter notebook
を実行すると
デフォルトブラウザ内でjupyter notebookが起動します。

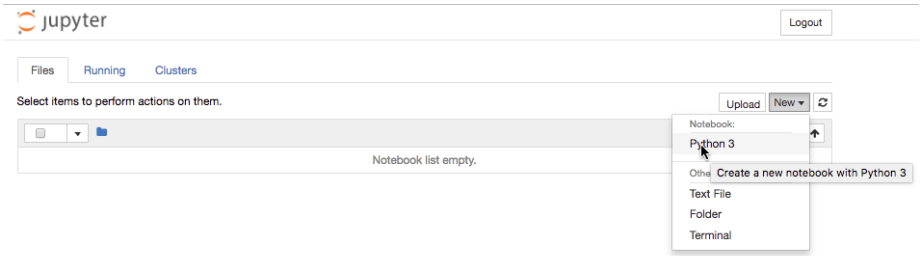
下のような画面がブラウザで表示されます。 (中身が空のディレクトリで起動すると、下のようにNotebook list emptyと表示されます。)



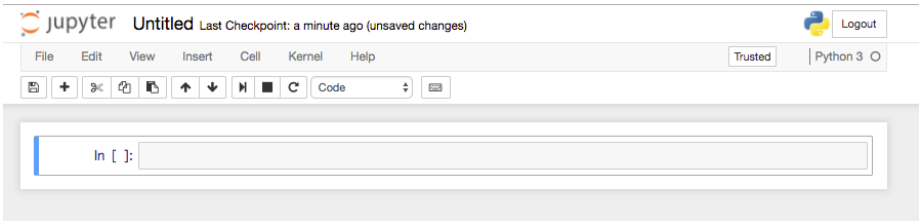
どこのディレクトリでも起動しますが、ノートブックファイル(.ipynbファイル)を保存する作業ディレクトリや解析したいFITSファイルなどのデータがあるディレクトリなどがよいでしょう。前者はファイルの整理・管理の観点から、後者はデータへのパスが簡単になるという観点からです。

新規ノートブックの作成

右のNewからプルダウンメニューでPython3を選択します。

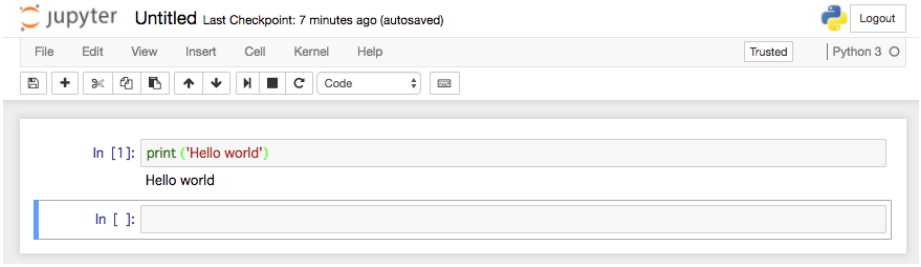


すると下のような新規ノートブック画面が出現します。



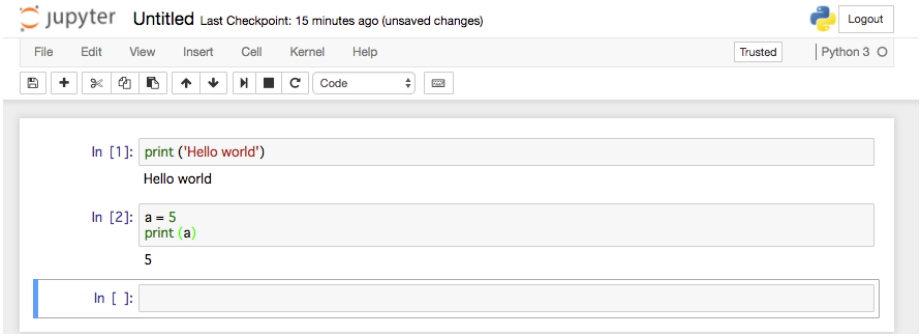
pythonと対話してみましょう

Cellの中にPythonのコマンドを入力して **[shift]+[enter]** します。



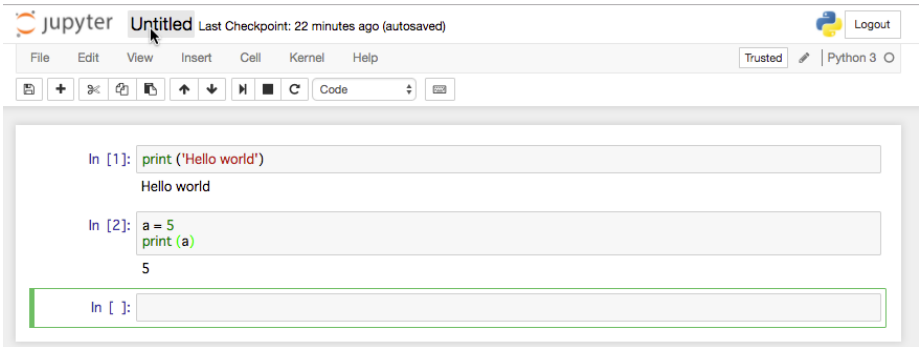
コマンドの入力に対する出力が、そのCellの下に表示されます。

コマンドが複数行にわたる場合、**[enter]** で改行します。
そのCellを実行する場合には、そのCellをマウスで選択した状態で **[shift]+[enter]** します。



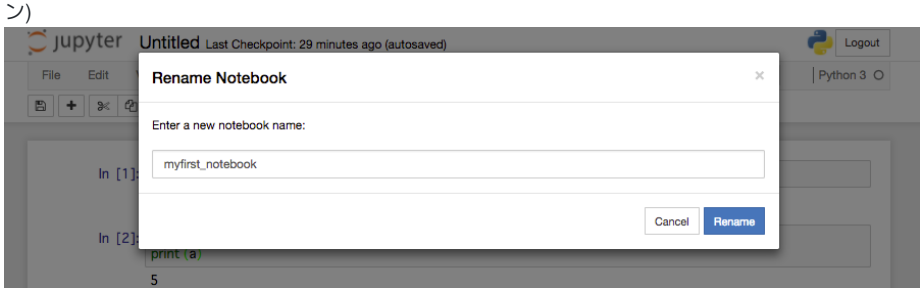
この対話を保存しましょう

上部の **Untitled** のあたりにマウスカーソルを合わせると、背景がグレイにかわるのでクリックします。

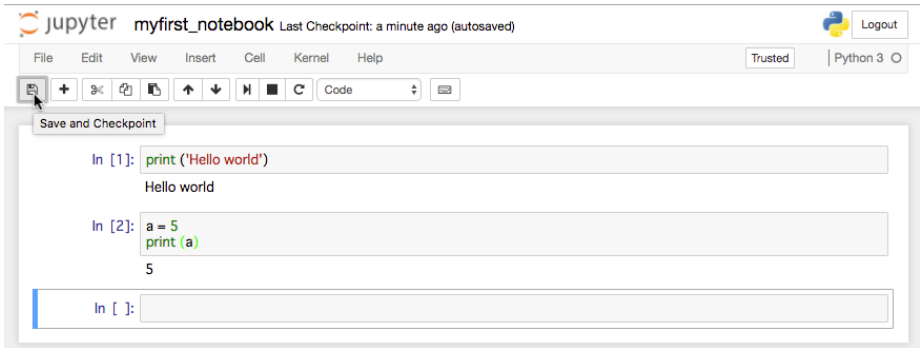


ファイル名を入力して、Renameボタンを押します。(Jupyterのバージョンによっては、OKボタ

ン)

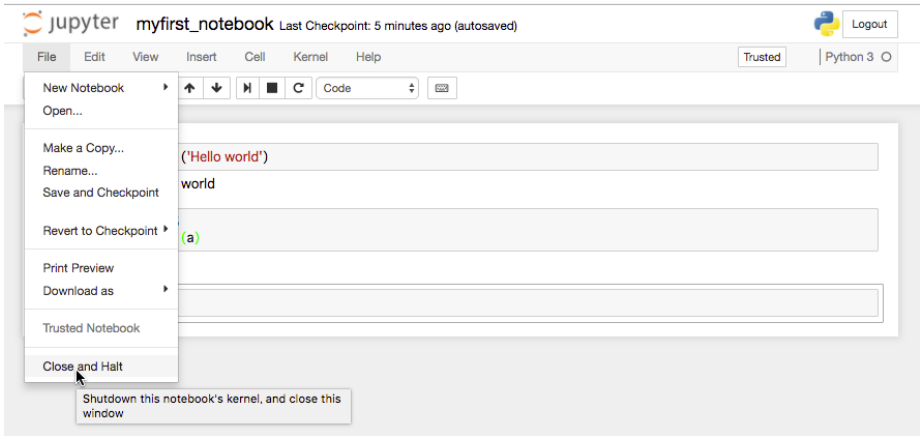


上部メニューバーの下のフロッピーのアイコンを押すと保存です。
デフォルトでは2分おきに自動保存されます。



ノートブックの終了

上部メニューバーの **File** のプルダウンメニューの一番下の **Close and Halt**を選んで終了。



jupyterを起動したときの画面に戻ります。ちゃんと名前をつけたファイルができていますね。

ここで、ファイル名のリンクをクリックすれば、そのノートブックの実行および編集が可能になります。

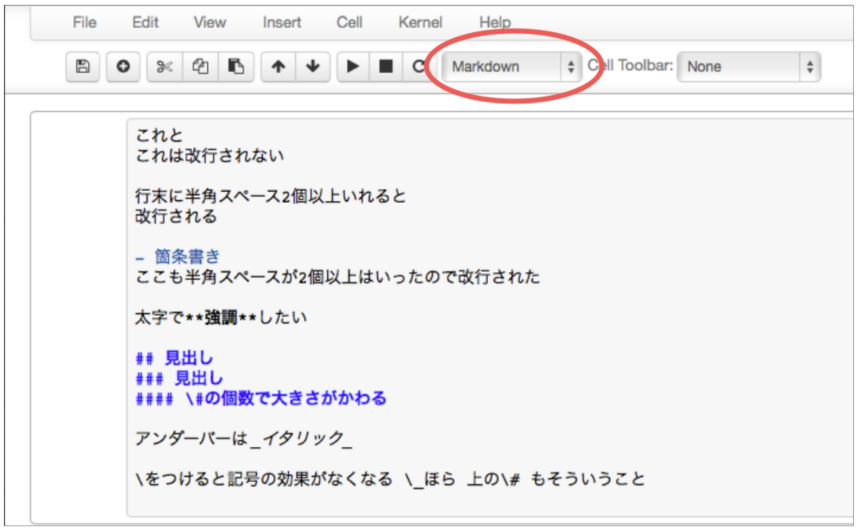
ブラウザを閉じて終了です。ターミナル**Cntrl+C**でjupyter notebookのプロセスを終了させます。
ターミナルを閉じます。

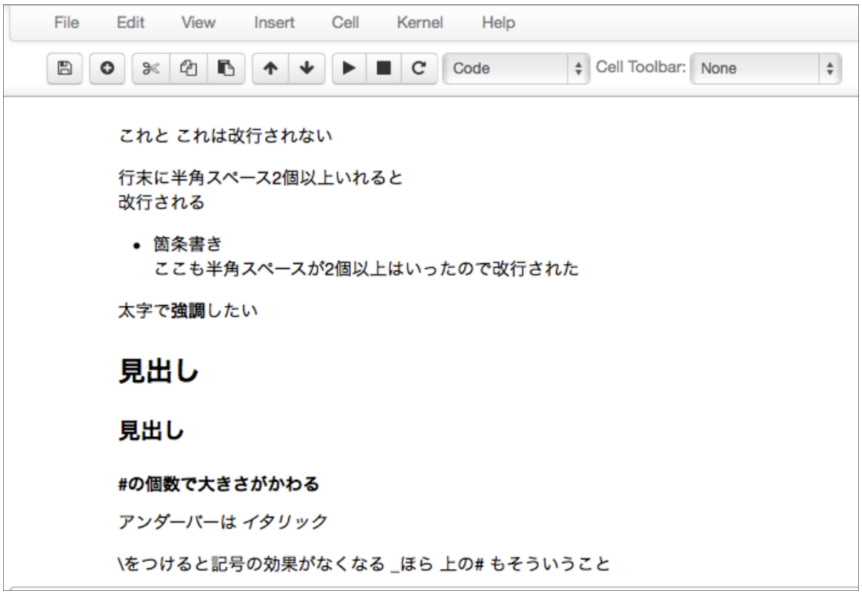
2. メモを書き込む ~ markdown

yas.nakajima edited this page 6 days ago · 1 revision

マウスのカーソルでCellを選んだうえで、jupyter notebookの上部のデフォルトでは Code になっているところをMarkdownにすると、そのセルに文章を書き込むことができます。markdown形式で文章を書き込めます。markdownでは、マークアップ言語であるhtmlよりもかなり簡単な表記ルールで、文章に構造を持たせることができます。

文章を書き込んだあとで、[shift]+[enter]するとmarkdownのルールに従ったレイアウトで表示されます。



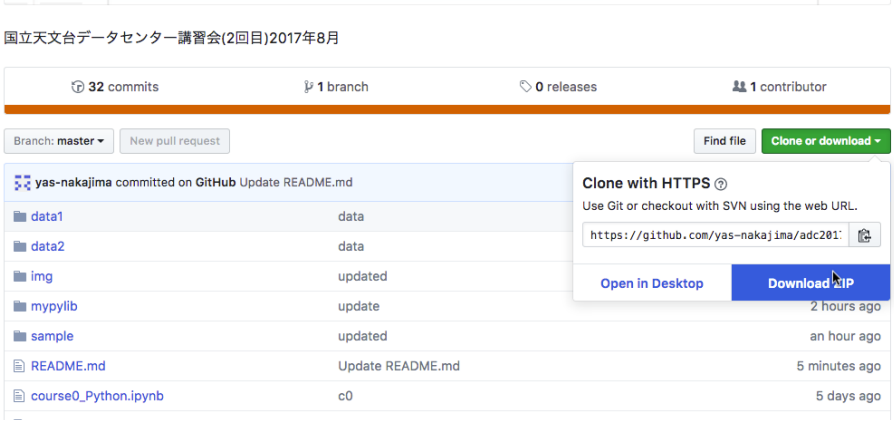


3. 資料のダウンロードとファイルのオープン

yas-nakajima edited this page 23 hours ago · 4 revisions

資料のダウンロード

adc2017python2のトップページの右の緑の[Clone or download]をクリックすると[Download ZIP]を選択できるので、そこをクリックしてください。するとあなたのPCのダウンロードフォルダにzipファイルがダウンロードされます。



そのZIPファイルを展開するとadc2017python2-masterというフォルダが現れます。(自動的に展開されてるかもしれません)
そのフォルダを、適宜、どこかのフォルダに移動してください。
そのフォルダの中に.ipynbファイルなどがあります。

圧縮データの解凍

data1などのディレクトリの中のFITSファイルはbzip2で圧縮されています。
adc2017python2-masterにはuncompress.shというシェルスクリプトを用意しています。
./uncompress.sh を実行すれば全ての圧縮ファイルは解凍されます。

ファイルのオープン

(!!GUIで.ipynbファイルのアイコンをクリックしてもオープンできません!!)
ターミナルでそのadc2017python2-masterのディレクトリに移動します。
そのディレクトリで
jupyter notebook
を実行すると、jupyter notebookが起動し、下のような画面が現れます。

Pages 7

Home

1. はじめてのjupyter notebook

2. メモを書き込む ~ markdown

3. 資料のダウンロードとファイルのオープン

4. 参考図書

appendix 1. Mac(ElCapitan以降)へのIRAFインストール

appendix 2. Jupyterってどう発音するの？

Clone this wiki locally

https://github.com/yas-na

このブラウザの.ipynbのリンクをクリックするとファイルをjupyter notebookでオープンできます。
実行や編集が可能になります。

4. 参考図書

yas.nakajima edited this page 6 days ago · 1 revision

これからPythonを始めるのであればPython3系で勉強することを勧めます。
2020年にはPython2系のサポートが切れるからです。

初心者向け

Pythonスタートブック

出版社: 技術評論社
ISBN-10: 4774142298
¥2,678

プログラミング初心者の人向け。変数とは何か、関数とか何か、といったレベルからわかりやすい説明があります。 おそらく、この講習会の参加者には物足りない内容でしょう。残念ながら、Python2系向けに書かれているが、この内容であればあまり支障はないでしょう。Python3系への改訂が望まれます。

中級者向け

実践力を身につける **Python**の教科書

出版社: マイナビ出版
ISBN-10: 4839960240
¥2,786

Pythonについて広くカバーしています。内容も整理されています。 プログラミング経験者あるいはPython初心者がもう少し勉強しようという場合にオススメ。

入門**Python3**

出版社: オライリージャパン
ISBN-10: 4873117380
¥3,996

入門とありますが、中級者+向けです。初級者+から中級者の足元固め。600ページあります。

上級者向け

科学技術計算のための**Python**入門

出版社: 技術評論社
ISBN-10: 4774183881
¥3,456

NumpyやMatplotlibなど、科学計算に使うライブラリについての記載あり。 Pythonそのものについても、掘り下げた解説が含まれています。

リファレンス

Pages 7

Home

1. はじめての**jupyter notebook**

2. メモを書き込む ~ **markdown**

3. 資料のダウンロードとファイルのオープン

4. 参考図書

appendix 1. Mac(ElCapitan以降)への IRAFインストール

appendix 2. Jupyterってどう発音するの？

Python ライブラリ厳選レシピ

出版社: 技術評論社
ISBN-10: 4774177075
¥3,110

Pythonには数多くのライブラリが用意されています。 自分で関数なんかを作ったあとに「なんだ、こんなのもうあったのか！」はありがちです。 多くのライブラリがサンプルとともに簡潔に紹介されています。

appendix 1. Mac(ElCapitan以降)へのIRAFインストール

yas.nakajima edited this page 6 days ago · 1 revision

個人で使う場合にはprivate installationで十分なので、その方法を記します。

ここでは、自分のホームディレクトリを/Users/myname/とします。適宜読み替えてください。

1. 自分のホームディレクトリにirafおよびiraf/irafというディレクトリを作成します。
   mkdir /Users/myname/iraf
   mkdir /Users/myname/iraf/iraf

2. /Users/myname/iraf/iraf で iraf.macx.x86_64.tar を展開します。(以下は~/Downloads/にダウンロードした場合の例)
   cd /Users/myname/iraf/iraf
   tar xvf ~/Downloads/iraf.macx.x86_64.tar

3. ./install を実行 (2と同じディレクトリで)
   sudo しない！　-systemの引数をつけない！
   以上の手順で、/Users/myname/.iraf にIRAFに必要なバイナリやスクリプトがインストールされます。 ~/.bashrcにも数行追記されます。

4. source ~/.bashrc を実行、あるいは今のターミナルを閉じて新しいターミナルを開いてから、
   unalias mkiraf
   mkiraf
   としてlogin.clを作成します。そのうえでclを実行すればIRAFのCL画面が起動します。
   (pyrafを使うのであれば /Users/myname/iraf で上のコマンドを実行してlogin.clを作成。これがあるので、/Users/myname/irafの下にさらにディレクトリirafを作成した。)

以前は、sudo ./install -system で /iraf/iraf/ とかにインストールする方法を紹介していました。MacのElCapitan以降ではSIP(System Integrity Protection)によって、/usr (/usr/localをのぞく)などにrootでも勝手に変更をかけることができなくなりました。sudo ./install --systemだと/usr/include/iraf.h を作成できないといって失敗します。
一台のMacを複数のユーザで使う場合には、そのような方法が必要なのですが、通常は自分一人で使うことが多いと思うので、上で紹介したprivate installationを行うことでSIPの影響を回避することができます。自分一人で使う場合には、ホームディレクトリにインストールで十分です。

補足：4のunalias mkirafって何だ？
IRAFをインストールすると、mkirafは/Users/myname/iraf/iraf/unix/hlib/mkiraf.cshへのaliasになっています。(ターミナルでaliasとすると見れます) そのmkiraf.cshの冒頭では、インストールするディレクトリに関わらず環境変数irafが set iraf = "/iraf/iraf" のように設定されています。なので、/iraf/iraf/でiraf.macx.x86_64.tarを展開した場合にしかmkiraf.cshは働きません。

いっぽう、上記方法でIRAFをインストールすると/Users/myname/.iraf/binに自動的にPATHが通ります。その中のmkiraf(は /Users/myname/iraf/iraf/unix/hlib/mkiraf.shへのシンボリックリンク)では、インストールディレクトリに応じて、環境変数がiraf="/Users/myname/iraf/iraf/"のように設定されます。こちらを使うのが正しい。

unalias mkiraf によって、mkirafコマンドで/Users/myname/.iraf/bin/mkiraf が使えるようにするのがポイントです。

2017-07 (El CapitanとSierraで確認済み)

appendix 2. Jupyterってどう発音するの？

yas-nakajima edited this page 6 days ago · 3 revisions

どうでもいい話ですが、ジュピター？ジュパイター？
私はずっとジュピターとよんできました。
木星とまぎらわしいですし、Pythonのパイ(py)なのでジュパイターがいいような気もします。

ですが、Jupyterを使用しているネイティブはジュピターとよんでいるようです。
http://youglish.com/search/Jupyter

好きなほうでよんでください。:)

Clone this wiki locally

https://github.com/yas-na