

SciPy è un'estensione di NumPy che aggiunge altri e più complessi algoritmi per uso scientifico e tecnico.

- Modulo clustering (`scipy.cluster`)
- Constants (`scipy.constants`)
- Trasformate di Fourier discrete(`scipy.fftpack`)
- Integrazione e equazioni differenziali (`scipy.integrate`)
- Interpolazione (`scipy.interpolate`)
- Input and output (`scipy.io`)
- Algebra lineare (`scipy.linalg`)
- Analisi di immagini multi dimensionali (`scipy.ndimage`)
- Ottimizzazione(`scipy.optimize`)
- Analisi dei segnali (`scipy.signal`)
- Matrici sparse (`scipy.sparse`)
- Funzioni speciali (`scipy.special`)

file: twoimages.py

```
import numpy as np
from scipy.ndimage import map_coordinates, imread

# Genera due immagini (1024x1024) shiftate della quantita'
# specificata interpolando l'immagine del file di ingresso

def twoimages(im_file,sx,sy):
    img = imread(im_file)

    x0_1 = (img.shape[0]-1024 - sx)*0.5; x1_1 = x0_1+1023.5
    y0_1 = (img.shape[1]-1024 - sy)*0.5; y1_1 = y0_1+1023.5

    xi = np.mgrid[y0_1:y1_1:1.0, x0_1:x1_1:1.0] # Nota: Assi scambiati
    i1 = map_coordinates(img, xi, order=1, mode="nearest")

    x0_2 = x0_1+sx; x1_2 = x1_1+sx
    y0_2 = y0_1+sy; y1_2 = y1_1+sy
    xi = np.mgrid[y0_2:y1_2:1.0, x0_2:x1_2:1.0] # Nota: Assi scambiati
    i2 = map_coordinates(img, xi, order=1, mode="nearest")
    return i1, i2
```

[mgrid] [map_coordinates]

file: crosscorr.py

```
import numpy as np
from numpy.fft import fft2, ifft2, fftshift

def crosscorr(l, r): # calcola lo spettro di potenza
                    # della matrice di correlazione
                    # usando FFT/iFFT.

    lf = fft2(l)
    rf = fft2(r)
    lf = np.conj(lf)
    iff = np.absolute(ifft2(lf*rf))
    return fftshift(iff) # fftshift riaggiusta la posizione
```

[fft2] [conj] [ifft2] [absolute] [fftshift]

Simulazione di un sistema di telemetria

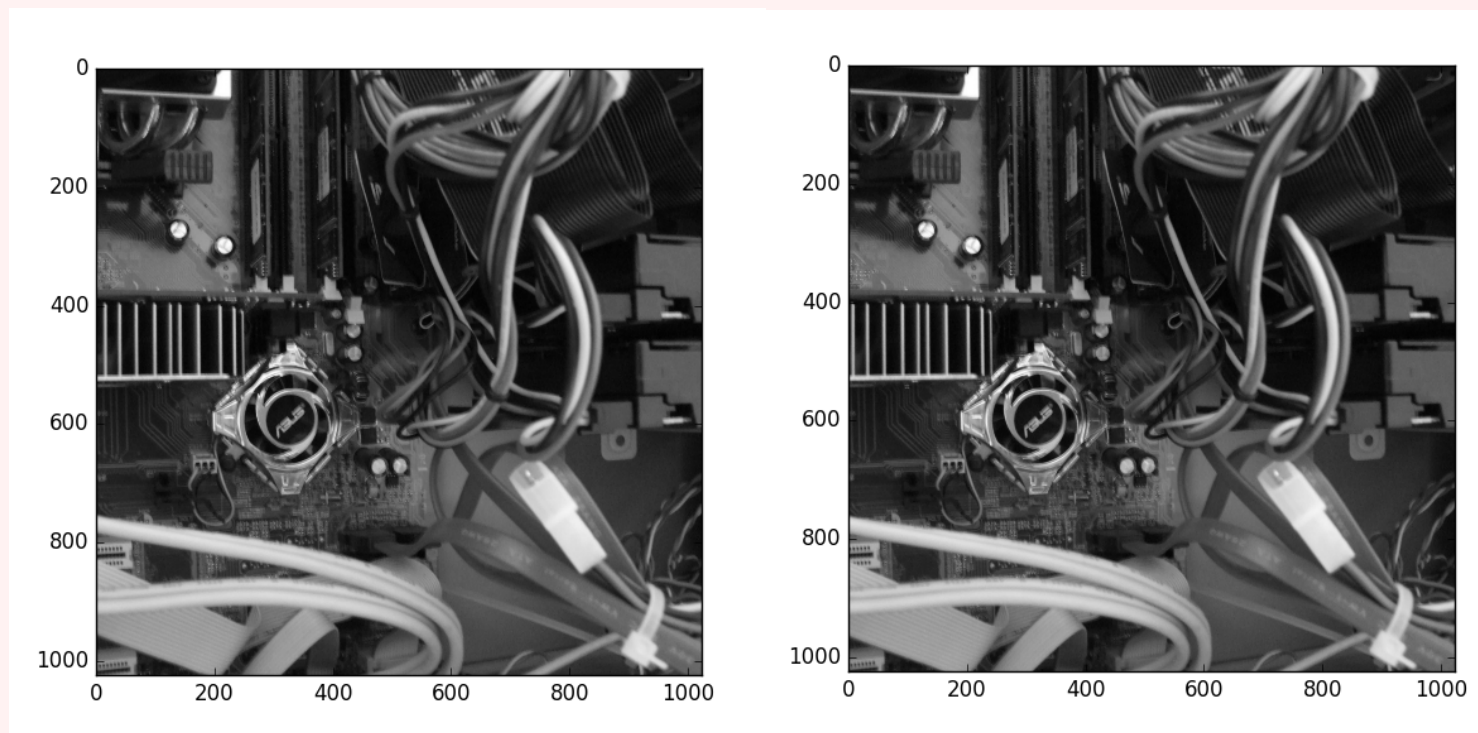
1 - genera due immagini shiftate

```
In [1]: from twoimages import twoimages
```

```
In [2]: i1, i2 = twoimages("t2.jpg", 7.77, -5.55)
```

```
In [3]: plt.imshow(i1, cmap="gray")
```

```
In [4]: plt.imshow(i2, cmap="gray")
```



Simulazione di un sistema di telemetria

2 - calcola spettro di pot. della funz. cross correl.

```
In [5]: from crosscorr import crosscorr
```

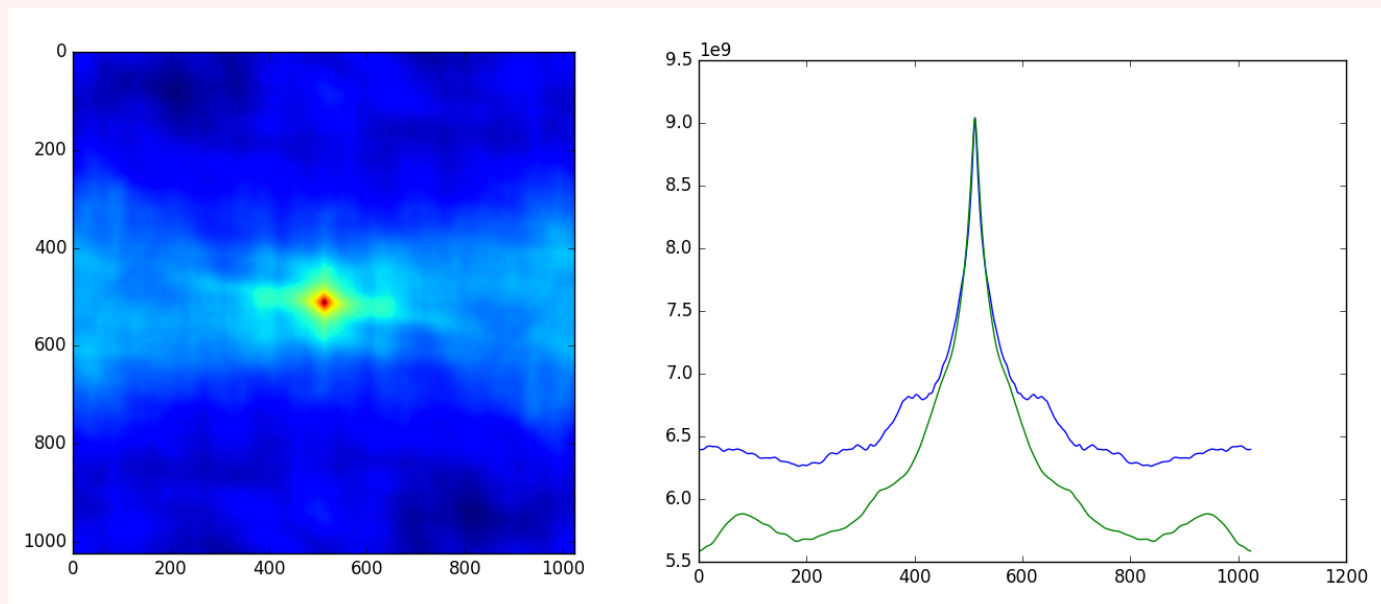
```
In [6]: c0 = crosscorr(i1, i1) # CPS di "calibrazione"
```

```
In [7]: c1 = crosscorr(i1, i2) # CPS di misura
```

```
In [8]: plt.imshow(c0)
```

```
In [9]: plt.plot(c0[512,:])
```

```
In [10]: plt.plot(c0[:,512])
```



file: maxcoords.py

```
import numpy as np

def maxcoords(arr):
    maxid = arr.argmax()
    x = maxid%arr.shape[1]
    y = maxid//arr.shape[1]
    return x, y
```

(vedi: **argmax**)

3 - trova massimi

```
In [13]: from maxcoords import maxcoords
```

```
In [14]: max0 = maxcoords(c0)
```

```
In [15]: max1 = maxcoords(c1)
```

```
In [16]: max0
```

```
Out[16]: array([512, 512])
```

```
In [17]: max1
```

```
Out[17]: array([504, 517])
```

```
In [18]: max0-max1
```

```
Out[18]: array([ 8, -5])
```

È possibile stimare il valore di shift con precisione migliore di un pixel?

Metodo: determinare i parametri di una gaussiana bidimensionale ottenuta da un best-fit

file: fitgaussian.py

```
import numpy as np
import scipy.optimize

def gaussian(height, center_x, center_y, width_x, width_y):
    # Returns a gaussian function with the given parameters.
    width_x = float(width_x)
    width_y = float(width_y)
    return lambda x,y: height*np.exp(
        -0.5*(((center_x-x)/width_x)**2+((center_y-y)/width_y)**2))

def moments(data):
    # Returns (height, x, y, width_x, width_y)
    # the gaussian parameters of a 2D distribution by calculating its moments.
    # It is used by the fitting procedure to estimate initial values
    total = data.sum()
    X, Y = np.indices(data.shape)
    x = (X*data).sum()/total
    y = (Y*data).sum()/total
    col = data[:, int(y)]
    width_x = np.sqrt(np.abs((np.arange(col.size)-y)**2*col).sum()/col.sum()))
    row = data[int(x), :]
    width_y = np.sqrt(np.abs((np.arange(row.size)-x)**2*row).sum()/row.sum()))
    height = data.max()
    return height, x, y, width_x, width_y

def fitgaussian(data):
    # Returns (height, x, y, width_x, width_y)
    # the gaussian parameters of a 2D distribution found by a fit
    params = moments(data)
    errorfunction = lambda x: np.ravel(gaussian(*x)(*np.indices(data.shape)) \
        - data)
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]]) # Aggiustamento assi
```

(vedi: indices, ravel, leastsq)

Procediamo con il fit:

```
In [19]: from fitgaussian import fitgaussian
```

```
In [20]: c0s = c0[480:-480,480:-480]
```

```
In [21]: c1s = c1[480:-480,480:-480]
```

```
In [22]: fit0 = fitgaussian(c0s)
```

```
In [23]: fit1 = fitgaussian(c1s)
```

```
In [24]: fit1[1:3]-fit0[1:3]
```

```
Out[24]: array([-6.06430762,  4.94558697])
```

Il risultato non è granché. Vediamo per quali motivi ...

file: gauss2d.py

```
import numpy as np
```

```
def gauss2d(height, sizeX, sizeY, centerX, centerY, widthX, widthY):
```

```
    x = np.arange(0, sizeX, 1, float)
```

```
    y = np.arange(0, sizeY, 1, float).reshape(-1,1)
```

```
    return height * np.exp(-0.5*(((x-centerX)/widthX)**2 + \
                                   ((y-centerY)/widthY)**2))
```

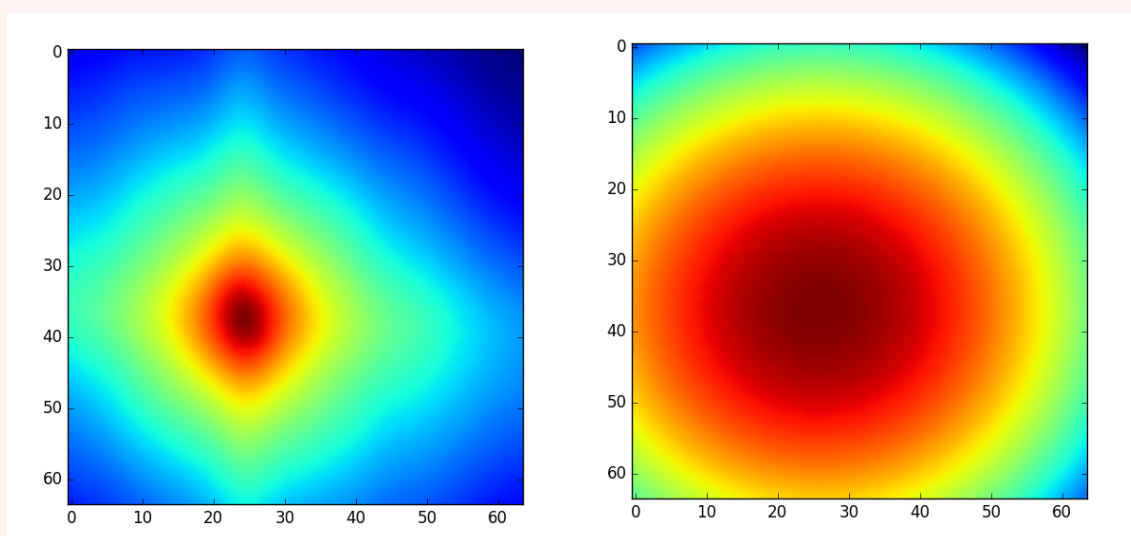
Generiamo la gaussiana di best-fit:

```
In [25]: from gauss2d import gauss2d
```

```
In [26]: gbest = gauss2d(fit1[0], 64, 64, *fit1[1:])
```

```
In [27]: plt.imshow(c1s)
```

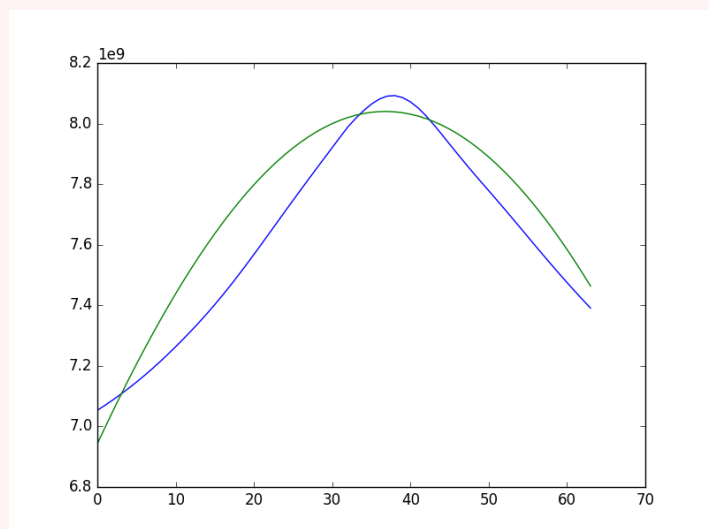
```
In [28]: plt.imshow(gbest)
```



Vediamo anche due profili in corrispondenza del massimo:

```
In [29]: plt.plot(c1s[:,37])
```

```
In [30]: plt.plot(gbest[:,37])
```



Si vedono due motivi che sembrano causare la cattiva stima:

- la gaussiana non sembra particolarmente adatta per il fitting
- la dissimmetria della finestra di campioni usata

Cos'altro si potrebbe fare:

- Fit di gaussiana 2D in intorno simmetrico del pixel massimo
- Provare con curve di forma diversa (Es.: polinomi di grado pari)
- Fit di gaussiana (o polinomio) 1D lungo le direzioni X ed Y, separatamente.
-

- Il package `astropy` deriva da un progetto iniziato intorno al 2000 presso STScI, per sviluppare un ambiente per analisi dati astronomici di concezione “moderna” destinato a sostituire il linguaggio di comandi per `iraf`.
- Uno dei primi prodotti è stato il package `Numarray` (calcolo vettoriale e matriciale) successivamente fuso con un progetto simile (`Numeric`) per dare vita a `NumPy`.
- Successivamente sono stati aggiunti gli altri sub-package con un notevole sforzo di integrazione e standardizzazione.
- Il lavoro è tuttora in corso come si deduce anche dal fatto che i vari sub-package sono in diverso stato di sviluppo.

- Strutture dati e funzionalità principali
 - Costanti (`astropy.constants`)
 - Unità di misura e quantità fisiche (`astropy.units`)
 - Dataset n-dimensionali (`astropy.nddata`)
 - Dati tabellari (`astropy.tables`)
 - Coordinate astronomiche (`astropy.coordinates`)
 - “World Coordinate System” (`astropy.wcs`)
 - Modelli e interpolazione (`astropy.modeling`)
 - Funzioni analitiche (`astropy.analytic_functions`)
- Input/Output
 - Gestione files FITS (`astropy.io.fits`)
 - Gestione tabelle ASCII (`astropy.io.ascii`)
 - Gestione tabelle VO (`astropy.io.votable`)
 - Accesso a dati VO (`astropy.io.vo`)
 - Miscellanea (`astropy.io.misc`) gestione files HDF5, YAML ed altri
- Algoritmi astronomici e programmi di utilità
 - Algoritmi per cosmologia (`astropy.cosmology`)
 - Convoluzioni e filtraggi (`astropy.convolution`)
 - Visualizzazione di dati (`astropy.visualization`)
 - Analisi statistiche (`astropy.stats`)

astropy.constants

```
>>> from astropy.constants import G
>>> print G
Name      = Gravitational constant
Value     = 6.67384e-11
Uncertainty = 8e-15
Unit      = m3 / (kg s2)
Reference = CODATA 2010
>>> print G.value
6.67384e-11
>>> print G.name
Gravitational constant
>>> print G.unit
m3 / (kg s2)
```

astropy.units

```
>>> from astropy import units as u
>>> x = 1.5 * u.parsec
>>> x
<Quantity 1.5 pc>
>>> print x
1.5 pc
>>> print x.to(u.km)
4.6285163722e+13 km
>>> x.to(u.km)

>>> y = 1.5e+13 * u.km
>>> print x+y
1.98611689342 pc
```

astropy.time

```
>>> from astropy.time import Time
>>> times = ["1999-01-01T00:00:00.123456789", "2010-01-01T00:00:00"]
>>> t = Time(times, format="isot", scale="utc")
>>> t
<Time object: scale='utc' format='isot' value=['1999-01-01T00:00:00.123' '2010-01-
```

Sloan Digital Sky Service

- Interfaccia: form

Consente l'accesso alle informazioni ed ai file di dati mediante form HTML.

The screenshot shows the SkyServer DR13 Rectangular Search interface. The header includes the Sloan Digital Sky Survey logo, the SkyServer logo, and the DR13 release information. The SciServer logo is also present in the top right corner. The navigation bar includes links for Home, Data, Schema, Education, Astronomy, SDSS, Contact Us, Download, Site Search, Help, and History. The main content area is titled "Rectangular Search" and contains a form for searching objects. The form includes fields for Type of search (Optical bands, Infrared bands), Coordinate system (Equatorial (RA / Dec), Galactic (l and b)), and search parameters (RA, Dec, Min, Max). The output format is set to HTML, and the limit number of output rows is 10. The form also includes a "Submit Request" button and a "Reset Form" button. A note at the bottom explains the search parameters and the importance of using specific attributes to constrain the search.

SLOAN DIGITAL SKY SURVEY
SkyServer DR13

SciServer
Not logged in Help Login

Home Data Schema Education Astronomy SDSS Contact Us Download Site Search Help History **NEW!**

DR13 Tools

Getting Started
Famous places
Get Images
Scrolling sky
Visual Tools
Search
- Radial
- Rectangular
- Search Form
- SQL
- Imaging Query
- Spectro Query
- IR Spec Query
Object Crossid
Skyquery CrossMatch **NEW!**
CasJobs

Rectangular Search

NOTE: To be fair to other users, queries run from SkyServer search tools are restricted in how long they can run and how much output they return, by **timeouts** and **row limits**. Please see the [Query Limits help page](#). To run a query that is not restricted by a timeout or number of rows returned, please use the [CasJobs batch query service](#).

Type of search	<input checked="" type="radio"/> Optical bands <input type="radio"/> Infrared bands	
Coordinate system	<input checked="" type="radio"/> Equatorial (RA / Dec) <input type="radio"/> Galactic (l and b)	
258.2	RA	258.3
64	Dec	64.1

	Min		Max
<input type="checkbox"/>	0	u	20
<input type="checkbox"/>	0	g	20
<input type="checkbox"/>	0	r	20
<input type="checkbox"/>	0	i	20
<input type="checkbox"/>	0	z	20

Output Format ☒ HTML ☐ XML ☐ CSV ☐ JSON ☐ VOTable ☐ FITS ☐ MyDB **NEW!**

Table name

Limit number of output rows (0 for max) to

Enter the lower and upper limits for both **ra** and **dec** either in degrees or in h:m:s, d:m:s notation. The lower limit on dec should always be less than the upper limit. The range in either coordinates should be less than 0.2 degrees, to constrain the number of objects returned, for the time being. If the lower limit on ra is less than the upper limit, it will be interpreted as a wrap around ra=0. E.g. 345 <ra<15 means to search between 23h and 1h. Check the magnitudes you would like to constrain in your query. If you prefer not to use specific attributes, leave those rows unchecked. (If you do not insert constraints and select all entries, you will receive many records!)

Sloan Digital Sky Service

- Interfaccia: SQL

Consente l'accesso alle informazioni ed ai file di dati mediante interrogazioni SQL

The screenshot shows the SkyServer SQL Search interface. At the top, there's a navigation bar with links like Home, Data, Schema, Education, Astronomy, SDSS, Contact Us, Download, Site Search, Help, and History. The main content area is titled 'SQL Search' and contains a text box for entering SQL queries. A sample query is provided, which joins the 'PhotoObj' and 'SpecObj' tables and filters for objects with redshift between 0 and 19.6 and galactic longitude between 0 and 20. Below the query box, there are buttons for 'Check syntax', 'Submit query', and 'Clear Query'. There's also a section for 'Output Format' with radio buttons for HTML (selected), XML, CSV, JSON, VOTable, FITS, and MyDB. A 'Table name' input field and a 'Reset' button are also present. The interface includes a sidebar with 'DR13 Tools' and a footer with additional information about the database schema and SQL queries.

SLOAN DIGITAL SKY SURVEY
SkyServer DR13

SciServer Not logged in Help Login

Home Data Schema Education Astronomy SDSS Contact Us Download Site Search Help History **NEW!**

DR13 Tools

- Getting Started
- Famous places
- Get Images
- Scrolling sky
- Visual Tools
- Search
 - Radial
 - Rectangular
 - Search Form
 - **SQL**
 - Imaging Query
 - Spectro Query
 - IR Spec Query
- Object Crossid
- Skyquery CrossMatch **NEW!**
- CasJobs

SQL Search

This page allows you to directly submit a [SQL \(Structured Query Language\)](#) query to the SDSS database server. You can modify the default query as you wish, or cut and paste a query from the [SDSS Sample Queries](#) page.

Please note: To be fair to other users, queries run from SkyServer search tools are restricted in how long they can run and how much output they return, by **timeouts** and **row limits**. Please see the [Query Limits help page](#). To run a query that is not restricted by a timeout or number of rows returned, please use the [CasJobs batch query service](#).

Clear Query

```
-- This query does a table JOIN between the imaging (PhotoObj) and spectra
-- (SpecObj) tables and includes the necessary columns in the SELECT to upload
-- the results to the SAS (Science Archive Server) for FITS file retrieval.
SELECT TOP 10
  p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
  p.run, p.rerun, p.camcol, p.field,
  s.specobjid, s.class, s.z as redshift,
  s.plate, s.mjd, s.fiberid
FROM PhotoObj AS p
JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
  p.u BETWEEN 0 AND 19.6
  AND g BETWEEN 0 AND 20
```

Check syntax

Output Format: ☒ HTML ☐ XML ☐ CSV ☐ JSON ☐ VOTable ☐ FITS ☐ MyDB **NEW!**

Submit query

Table name:

Reset

To find out more about the database schema use the [Schema Browser](#)

For an introduction to the Structured Query Language (SQL), please see the [Searching for Data](#) How-To tutorial. In particular, please read the [Optimizing Queries](#) section.

The inclusion of the Imaging and spectro columns for [SAS](#) upload in your query (as in the default query on this page) will ensure that when you press **Submit**, the appropriate button(s) are displayed on the query results page to allow you to upload the necessary information to the [SAS](#) to retrieve the FITS file data corresponding to your CAS query. The imaging columns needed for upload to the [SAS](#) are *run*, *rerun*, *camcol*, and *field*. The spectroscopic columns needed are *plate*, *mjd*, *fiberid*, and optionally *sprerun* (the latter requires a join with the PlateX table).

Automazione “data retrieval”

file: `sdss.py`

```
from __future__ import print_function
import sys, os, requests, json

URL="http://cas.sdss.org/public/en/tools/search/x_results.aspx"
FMT="json"
outdir="."

QRY = """SELECT TOP 10
    p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
    p.run, p.rerun, p.camcol, p.field,
    s.specobjid, s.class, s.z as redshift,
    s.plate, s.mjd, s.fiberid
FROM PhotoObj AS p
    JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
    p.u BETWEEN %f AND %f
    AND g BETWEEN %f AND %f"""

def get_list(ra_center, dec_center, size_arcmin):
    size_deg = size_arcmin/120.
    ra_min = ra_center-size_deg
    ra_max = ra_center+size_deg
    dec_min = dec_center-size_deg
    dec_max = dec_center+size_deg
    qry = QRY % (ra_min, ra_max, dec_min, dec_max)
    params = {"cmd": qry, "format": FMT, "searchtool": "SQL"}
    response = requests.get(URL, params=params)
    if not response.ok:
        raise Exception("Error retrieving data: " + response.reason)
    return json.loads(response.text)

if __name__ == "__main__":
    if len(sys.argv)!=4 :
        print("RA,DEC, size, must be given!")
        sys.exit(-1)
    ra_center = float(sys.argv[1])
    dec_center = float(sys.argv[2])
    size = float(sys.argv[3])
    found = get_list(ra_center, dec_center, size)
```

Automazione “data retrieval”

```
lfini$ ipython2
...
In [1]: %run sdss.py 10 10 1000

In [2]: found
Out[2]:
[{u'Rows': [{u'camcol': 1,
.....

In [3]: len(found[0]["Rows"])
Out[3]: 10

In [4]: found[0]["Rows"][0]
Out[4]:
{u'camcol': 1,
 u'class': u'STAR',
 u'dec': -1.0417691497987,
 u'fiberid': 208,
 u'field': 100,
 u'g': 16.17133,
 u'i': 15.3785,
 u'mjd': 52932,
 u'objid': 1237645941291614227,
 u'plate': 1515,
 u'r': 15.5894,
 u'ra': 49.6274851210218,
 u'redshift': -9.765775e-05,
 u'rerun': 301,
 u'run': 109,
 u'specobjid': 1705795582662043648,
 u'u': 17.65612,
 u'z': 15.26744}

In [5]: found[0]["Rows"][1]
Out[5]:
{u'camcol': 2,
 u'class': u'GALAXY',
.....
```

Automazione “data retrieval”

file: getfits.py

```
from __future__ import print_function
import os
from astropy.utils.data import download_file

URL0="https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/{rerun}/{run}/{camcol}/"
FNAME="frame-u-{run:06d}-{camcol}-{field:04d}.fits.bz2"

def getfits(obj_spec):
    url = URL0.format(**obj_spec)+FNAME.format(**obj_spec)
    fits_file = download_file(url, cache=True)
    localname = localname = "obj_%d.fits.bz2"%obj_spec["objid"]
    os.rename(fits_file, localname)
    print("Creato file:", localname)
```

Download dei file di dati:

```
In [6]: %run getfits.py

In [7]: for fspec in found[0]["Rows"]:
...:     getfits(fspec)
...:

Creato file: obj_1237645941291614227.fits.bz2
Creato file: obj_1237645941824356443.fits.bz2
Creato file: obj_1237645942905438473.fits.bz2
Creato file: obj_1237645942905569371.fits.bz2
Creato file: obj_1237645942905700448.fits.bz2
Creato file: obj_1237645942905831442.fits.bz2
Creato file: obj_1237645942905831562.fits.bz2
Creato file: obj_1237645942906028116.fits.bz2
Creato file: obj_1237645943973609500.fits.bz2
Creato file: obj_1237645943973675061.fits.bz2
```

- Il formato FITS¹ è uno standard per l'archiviazione di dati assai diffuso in astronomia.
- Il supporto per la lettura/scrittura di files in formato FITS era tradizionalmente fornito dal modulo pyFITS.
- Più recentemente le stesse funzioni sono state trasferite in un modulo del package astropy

```
In [8]: !bunzip2 obj_1237645941291614227.fits.bz2
```

```
In [9]: from astropy.io import fits
```

```
In [10]: ffile = fits.open("obj_1237645941291614227.fits")
```

```
In [11]: ffile.info()
```

```
Filename: obj_1237645941291614227.fits
```

No.	Name	Type	Cards	Dimensions	Format
0	PRIMARY	PrimaryHDU	85	(2048, 1489)	float32
1		ImageHDU	6	(2048,)	float32
2		BinTableHDU	27	1R x 3C	[49152E, 2048E, 1489E]
3		BinTableHDU	79	1R x 31C	[J, 3A, J, A, D, D, 2J, J, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, E, E]

```
In [12]: ffile[0].header
```

```
Out[12]:
```

```
SIMPLE =          T /  
BITPIX =        -32 / 32 bit floating point  
NAXIS  =          2  
.....
```

¹ **F**lexible **I**mage **T**ransport **S**ystem.

Ancora sulla fase della luna...

file: `mphase.py`

```
import numpy as np
from astropy.coordinates import get_moon, get_sun
from astropy.time import Time
import time

def mphase(ttime):
    sun = get_sun(ttime)
    moon = get_moon(ttime)
    elongation = sun.separation(moon)
    return np.arctan2(sun.distance*np.sin(elongation),
                      moon.distance - sun.distance*np.cos(elongation))

if __name__ == "__main__":
    now = Time(time.time(), format="unix")
    print "Fase lunare in questo momento:", mphase(now).to("deg")
```

esecuzione:

```
$ python mphase.py
Fase lunare in questo momento: 0.823475717273 rad
```

Usiamo adesso le proprietà di introspezione di python:

```
>>> from astropy.coordinates import get_moon, get_sun
>>> from astropy.time import Time
>>> import time
>>> help(get_moon)
>>> help(get_sun)
>>> now = Time(time.time(), format= unix )
>>> moon = get_moon(now)
>>> sun = get_sun(now)
>>> sun.distance
>>> moon.distance
>>> ...
```

Come trovare il punto iniziale per la fase della luna nell'esempio sulle nascite.

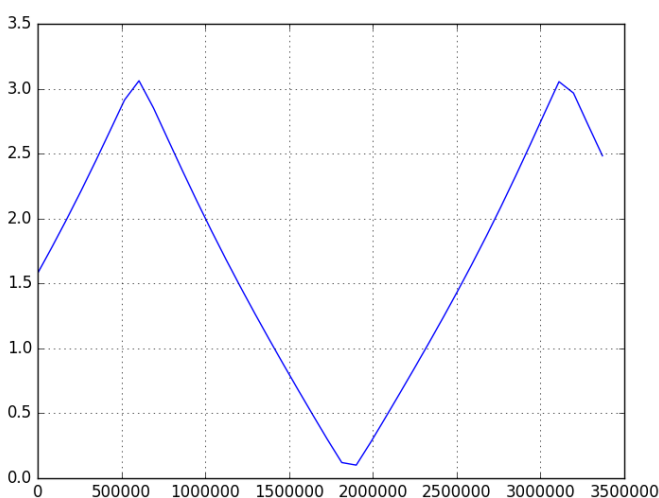
file: `newmoon.py`

```
from astropy.time import Time
import mphase as _mphase

def mphase(tt):
    ttime = Time(tt, format="unix")
    return _mphase.mphase(ttime).value
```

Come procedere

```
In [1]: from newmoon import mphase
In [2]: vmphase=np.vectorize(mphase)
In [3]: tv = np.arange(40)*86400
In [4]: vphase = vmphase(tv)
In [5]: plot(tv, vphase)
In [6]: plt.grid()
```



Il grafico mostra l'andamento della fase lunare a partire dal 1/1/1970

Troviamo adesso un minimo della funzione mphase

```
In [8]: from scipy.optimize import minimize_scalar
```

```
In [9]: res=minimize_scalar(mphase,bounds=(1500000.,2500000.),method='bounded')
```

```
In [10]: res
```

```
Out[10]:
```

```
      fun: 0.05789068913097943
 message: 'Solution found.'
    nfev: 12
   status: 0
  success: True
         x: 1862913.9234630163
```

Che è effettivamente assai vicino ad un minimo

```
In [11]: mphase(1862913.0)
```

```
Out[11]: 0.057890689166047664
```

```
In [12]: mphase(1862913.9234630163)
```

```
Out[12]: 0.05789068913097943
```

```
In [13]: mphase(1862914.0)
```

```
Out[13]: 0.05789068913126939
```

Lettura dati da file FITS

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

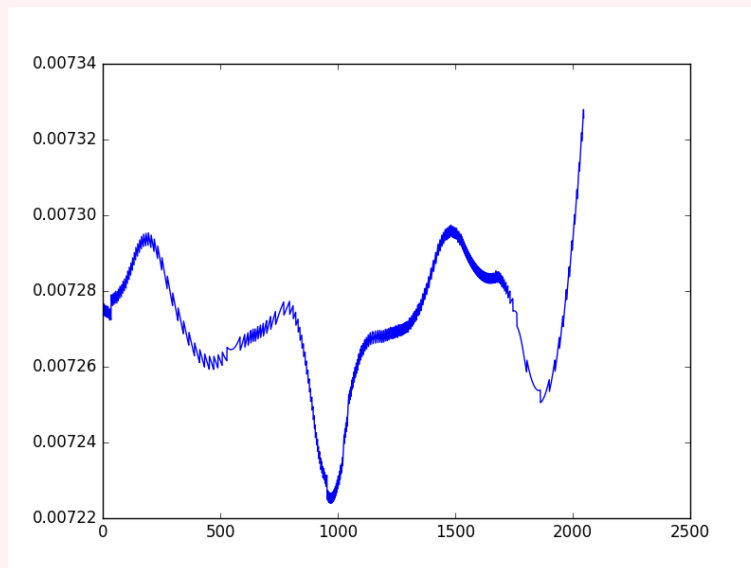
```
In [2]: fits.info("obj_1237645941291614227.fits")
```

```
Filename: obj_1237645941291614227.fits
```

No.	Name	Type	Cards	Dimensions	Format
0	PRIMARY	PrimaryHDU	85	(2048, 1489)	float32
1		ImageHDU	6	(2048,)	float32
2		BinTableHDU	27	1R x 3C	[49152E, 2048E, 1489E]
3		BinTableHDU	79	1R x 31C	[J, 3A, J, A, D, D, 2J, J, D, D, D, D,

```
In [3]: spectrum = fits.getdata("obj_1237645941291614227.fits", ext=1)
```

```
In [4]: plt.plot(spectrum)
```



Lettura dati da tabelle

```
In [6]: from astropy.table import Table
```

```
In [7]: table = Table.read("obj_1237645941291614227.fits", hdu=3)
```

```
In [8]: print table.columns
```

```
<TableColumns names=('RUN', 'RERUN', 'CAMCOL', 'FILTER', 'NODE', 'INCL', 'NAXIS', 'FIELD',
```

```
In [9]: table.more()
```

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

```
In [2]: hdus = fits.open("psf.fits")
```

```
In [3]: hdus[0].header
```

```
Out[3]:
```

```
SIMPLE =                T / Written by IDL: Fri May 26 14:23:40 2017
BITPIX =               -32 / IEEE single precision floating point
NAXIS   =                2 / number of array dimensions
NAXIS1  =               254
NAXIS2  =               256
EXTEND  =                T
DATE    = '2015-06-04'
DETECTOR= 'SHARK '
EXPTIME =                0.001
W_UNIT  = 'LBTIDX '
HIERARCH ccd39.BINNING = 1
HIERARCH ccd39.DARK_FILENAME = '20150604_030731_antidrift.fits'
HIERARCH ccd39.FRAME_RATE = '989.61 '
HIERARCH ccd39.READOUT_SPEED = 2500
HIERARCH ccd39.STATUS = 'STATE_OPERATING'
HIERARCH cuberot.POSITION = '0.000 '
HIERARCH cuberot.STATUS = 'STATE_READY'
HIERARCH cubestage.POSITION = '0.000 '
HIERARCH cubestage.STATUS = 'STATE_READY'
HIERARCH fw1.POSITION = '1.001 '
HIERARCH fw1.STATUS = 'STATE_READY'
HIERARCH fw2.POSITION = '4.000 '
HIERARCH fw2.STATUS = 'STATE_READY'
HIERARCH lamp.INTENSITY = '-9999 '
HIERARCH lamp.STATUS = 'UNKNOWN '
HIERARCH lens.POSITION_X = -35.12
HIERARCH lens.POSITION_Y = 73.33
HIERARCH lens.STATUS = 'STATE_READY'
HIERARCH pup0.CX = '20.46 '
HIERARCH pup0.CY = '58.39 '
HIERARCH pup0.DIAMETER = '31.03 '
HIERARCH pup0.DIFFX = ' 0.04 '
HIERARCH pup0.DIFFY = '-0.03 '
HIERARCH pup0.SIDE = ' 0.00 '
...
COMMENT and Astrophysics', volume 376, page 359; bibcode 2001A&A...376..359H
```

Vediamo il contenuto del file

```
In [4]: scidata = hdus[0].data

In [5]: im = plt.imshow(scidata)

In [6]: cbar = plt.colorbar(im, orientation="vertical")

In [7]: from matplotlib import colors

In [8]: im = plt.imshow(scidata, norm=colors.LogNorm())

In [9]: cbar = plt.colorbar(im, orientation="vertical")

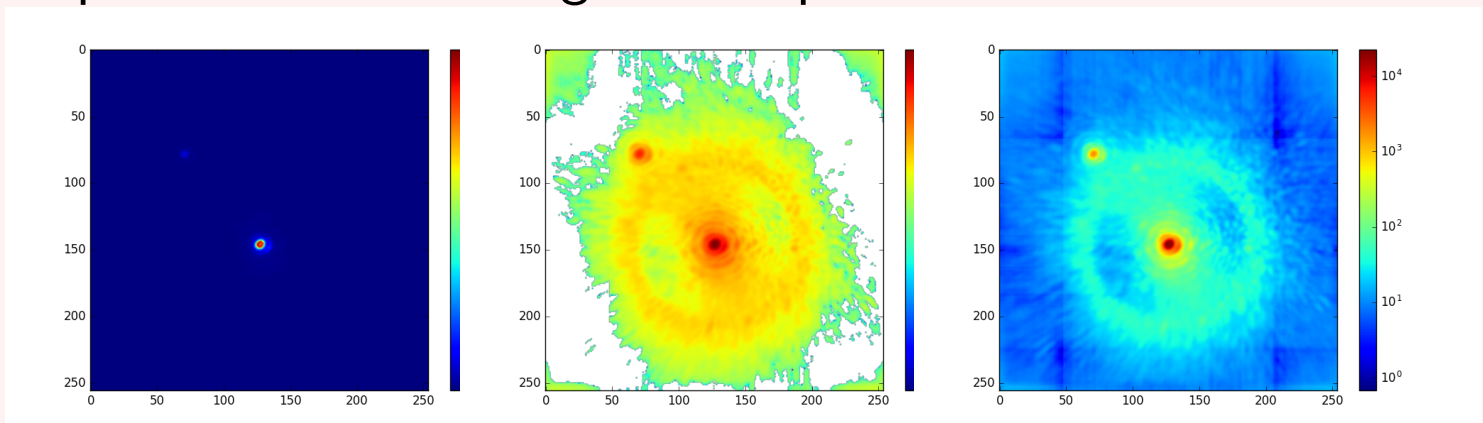
In [10]: np.min(scidata), np.max(scidata)
Out[10]: (-9.3197355, 22336.143)

In [11]: scidata += 10

In [12]: im = plt.imshow(scidata, norm=colors.LogNorm())

In [13]: cbar = plt.colorbar(im, orientation="vertical")
```

E queste sono le immagini corrispondenti:



In [6]

In [9]

In [13]

I package “affiliati” sono package Python di carattere astronomico, nati autonomamente, i cui sviluppatori hanno chiesto di entrare a far parte della comunità AstroPy allo scopo di promuovere l'uso del package e di assicurarne l'interoperabilità e l'adesione agli standard di interfacciamento.

Esempi:

- Montage-wrapper: Interfaccia Python per il “Montage Astronomical Image Mosaic Engine”
- Ginga: visualizzatore di immagini da file FITS
- APLpy: the Astronomical Plotting Library in Python (basata su matplotlib).
- Astroquery: strumenti per l'accesso a dati astronomici on-line

```
In [1]: from astroquery.simbad import Simbad
```

```
In [2]: Simbad.add_votable_fields("flux(V)","flux(U)","flux(B)")
```

```
In [3]: obj = Simbad.query_object("aldebaran")
```

```
In [4]: obj.pprint()
```

MAIN_ID	RA	DEC	RA_PREC	DEC_PREC	...	COO_BIBCODE	FLUX_V
FLUX_U	FLUX_B						
	"h:m:s"	"d:m:s"			...		mag
mag	mag						

* alf Tau 04 35 55.2390	+16 30 33.488	9	9	...	2007A&A...474..653V	0.860000	

```
In [5]: type(obj)
```

```
Out[5]: astropy.table.table.Table
```

```
In [6]: obj.keys()
```

```
Out[6]:
```

```
['MAIN_ID',  
 'RA',  
 'DEC',  
 'RA_PREC',  
 'DEC_PREC',  
 'COO_ERR_MAJA',  
 'COO_ERR_MINA',  
 'COO_ERR_ANGLE',  
 'COO_QUAL',  
 'COO_WAVELENGTH',  
 'COO_BIBCODE',  
 'FLUX_V',  
 'FLUX_U',  
 'FLUX_B']
```

```
In [7]: obj["FLUX_V"]
```

```
Out[7]:
```

```
<MaskedColumn name='FLUX_V' dtype='float32' unit='mag' format=u'!r:>'  
0.86000001
```
