scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)

scipy - 1

scipy is an extension of numpy which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)

scipy - 1

scipy is an extension of numpy which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)

scipy - 1

scipy is an extension of numpy which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)

scipy is an extension of numpy which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)
- Optimization(scipy.optimize)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)
- Optimization(scipy.optimize)
- Signal processing (scipy.signal)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)
- Optimization(scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrix operations (scipy.sparse)

scipy is an extension of numpy which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)
- Optimization(scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrix operations (scipy.sparse)
- Special functions (scipy.special)

scipy - 1

scipy is an extension of `numpy` which adds more and more complex algorithms for scientific and technical applications.

- Clustering module (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms(scipy.fftpack)
- Differential equations and integration (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Multi dimensional image processing (scipy.ndimage)
- Optimization(scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrix operations (scipy.sparse)
- Special functions (scipy.special)

Simulating a telemetry system

We want to evaluate the performances of an algorithm to measure the distance of an object using two images with parallax.

As a starting point we use the cross-correlation matrix to find the amount of shift between two synthetic images

Simulating a telemetry system

We want to evaluate the performances of an algorithm to measure the distance of an object using two images with parallax.

As a starting point we use the cross-correlation matrix to find the amount of shift between two synthetic images

file: twoimages.py

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

# Generate two 1024x1024 images shifted by given amount
# by interpolating in the input image

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] # Note: swapped axes
    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] # Note: swapped axes
    i2 = map_coordinates(img, xi, order=1, mode='nearest')

    return i1, i2
```

Simulating a telemetry system

We want to evaluate the performances of an algorithm to measure the distance of an object using two images with parallax.

As a starting point we use the cross-correlation matrix to find the amount of shift between two synthetic images

file: twoimages.py

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

# Generate two 1024x1024 images shifted by given amount
# by interpolating in the input image

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_
    i1 = map_coordinates(img, xi, or

    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] # Note: swapped axes
    i2 = map_coordinates(img, xi, order=1, mode='nearest')

    return i1, i2
```

See:
→ numpy.mgrid
→ scipy.map_coordinates

Simulating a telemetry system

We want to evaluate the performances of an algorithm to measure the distance of an object using two images with parallax.

As a starting point we use the cross-correlation matrix to find the amount of shift between two synthetic images

file: twoimages.py

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

# Generate two 1024x1024 images shifted by given amount
# by interpolating in the input image

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_
    i1 = map_coordinates(img, xi, or

    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] # Note: swapped axes
    i2 = map_coordinates(img, xi, order=1, mode='nearest')

    return i1, i2
```

See:
→ numpy.mgrid
→ scipy.map_coordinates

→

Simulating a telemetry system

file: crosscorr.py

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

def crosscorr(l, r): # Compute power spectrum
                     # from correlation matrix
                     # using FFT/iFFT.
    lf = fft2(l)
    rf = fft2(r)
    lf = np.conj(lf)
    iff = np.absolute(ifft2(lf*rf))
    return fftshift(iff) # fftshift realigns the output
```

Simulating a telemetry system

file: crosscorr.py

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

def crosscorr(l, r): # Compute power
                     # from correlati
                     # using FFT/iFFT
    lf = fft2(l)
    rf = fft2(r)
    lf = np.conj(lf)
    iff = np.absolute(ifft2(lf*rf))
    return fftshift(iff) # fftshift realigns the output
```

See:
- → numpy.fft.fft2
- → numpy.conj
- → numpy.fft.ifft2
- → numpy.absolute
- → numpy.fft.fftshift

Simulating a telemetry system

file: crosscorr.py

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

def crosscorr(l, r): # Compute power
                     # from correlati
                     # using FFT/iFFT
    lf = fft2(l)
    rf = fft2(r)
    lf = np.conj(lf)
    iff = np.absolute(ifft2(lf*rf))
    return fftshift(iff) # fftshift realigns the output
```

See:
- → numpy.fft.fft2
- → numpy.conj
- → numpy.fft.ifft2
- → numpy.absolute
- → numpy.fft.fftshift

1 - generate two shifted images

```
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")
```

Simulating a telemetry system

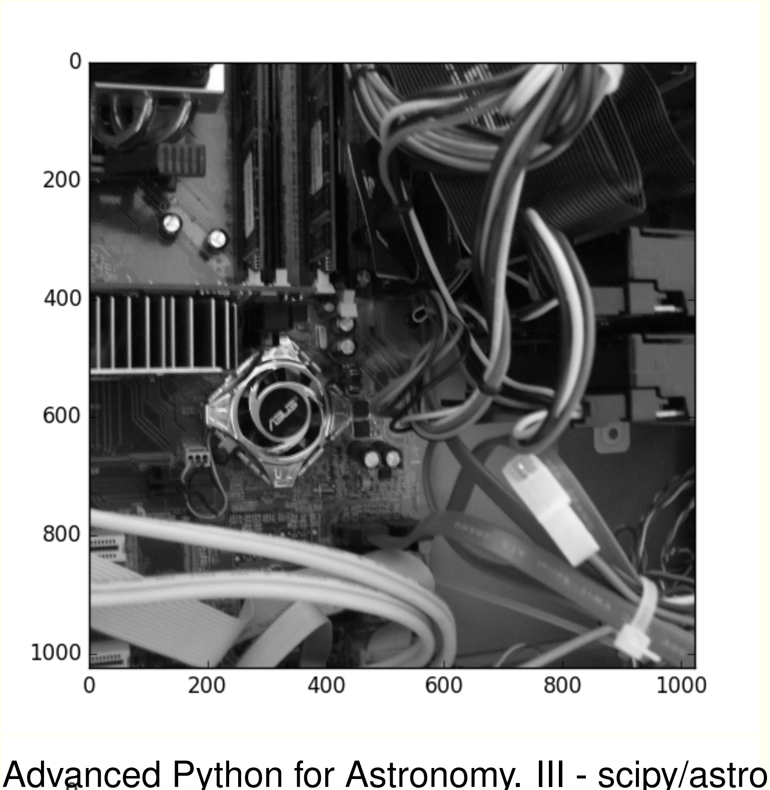
file: crosscorr.py

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

def crosscorr(l, r): # Compute power
                     # from correlati
                     # using FFT/iFFT
    lf = fft2(l)
    rf = fft2(r)
    lf = np.conj(lf)
    iff = np.absolute(ifft2(lf*rf))
    return fftshift(iff) # fftshift realigns the output
```

See:
- → numpy.fft.fft2
- → numpy.conj
- → numpy.fft.ifft2
- → numpy.absolute
- → numpy.fft.fftshift

1 - generate two shifted images

```
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")
```



Simulating a telemetry system

2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i2)

In [8]: plt.imshow(c0)

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

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

Simulating a telemetry system

2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i2)

In [8]: plt.imshow(c0)

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

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

⇐ Unshifted power spectrum

Simulating a telemetry system

2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)      ⇐ Unshifted power spectrum

In [7]: c1 = crosscorr(i1, i2)      ⇐ Shifted power spectrum

In [8]: plt.imshow(c0)

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

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

Simulating a telemetry system

2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i

In [8]: plt.imshow(c0)

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

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

Unshifted power spectrum

Shifted power spectrum

Show the cross correlation matrix image

Simulating a telemetry system

2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i

In [8]: plt.imshow(c0)

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

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

⇐ Unshifted power spectrum

⇐ Shifted power spectrum

⇐ Show the cross correlation matrix image

⇐ Show two line profiles

Simulating a telemetry system

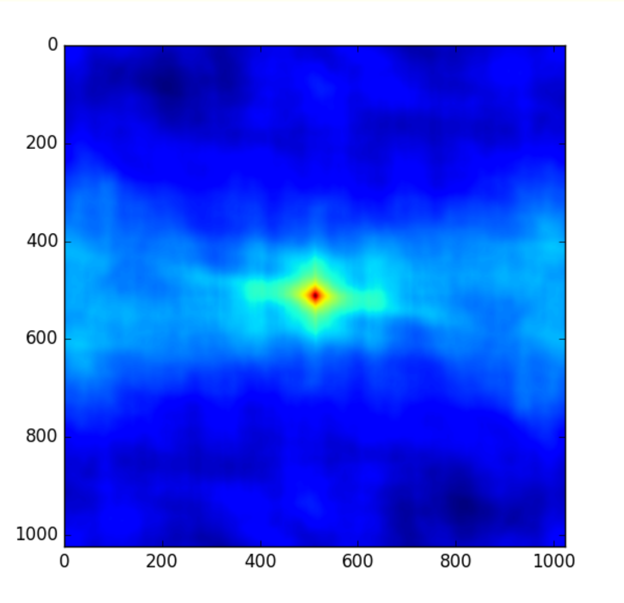
2 - compute the power spectrum of cross corr.

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

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i

In [8]: plt.imshow(c0)

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

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

Unshifted power spectrum

Show the cross correlation matrix image

Show two line profiles



Simulating a telemetry system

2 - compute the power spectrum of cross corr.

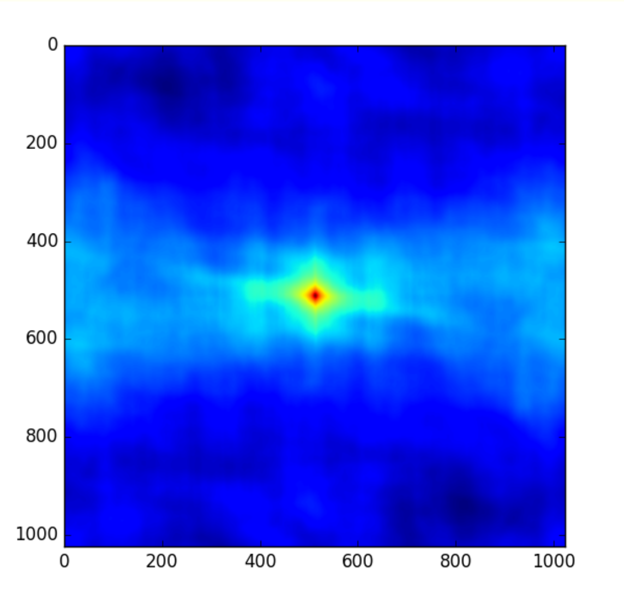
```
In [5]: from crosscorr import crosscorr

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i

In [8]: plt.imshow(c0)

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

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

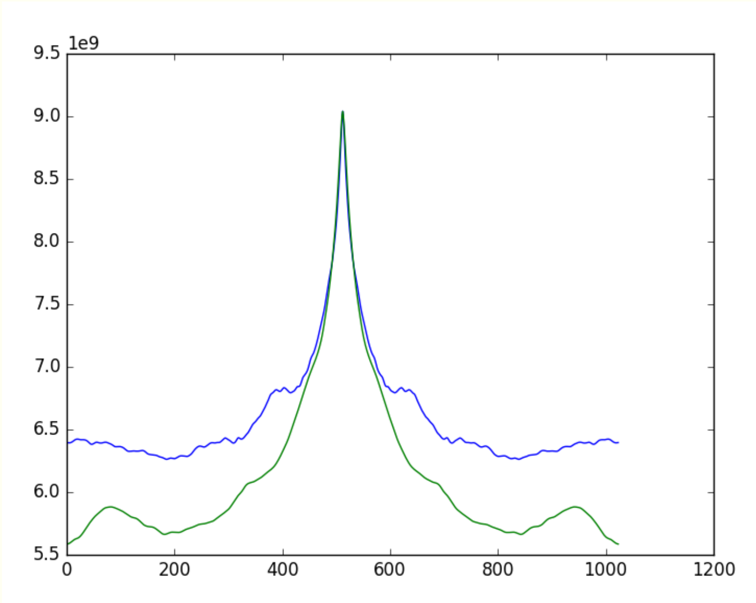
Unshifted power spectrum

Shifted power spectrum

Show the cross correlation matrix image

Show two line profiles





Simulating a telemetry system

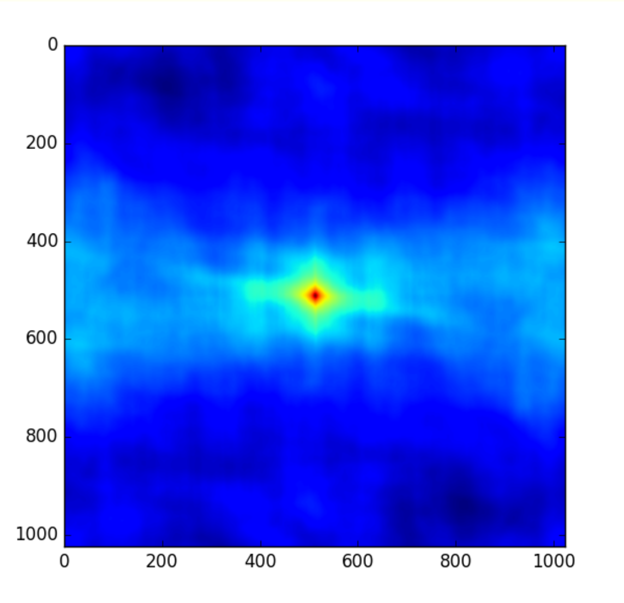
2 - compute the power spectrum of cross corr.

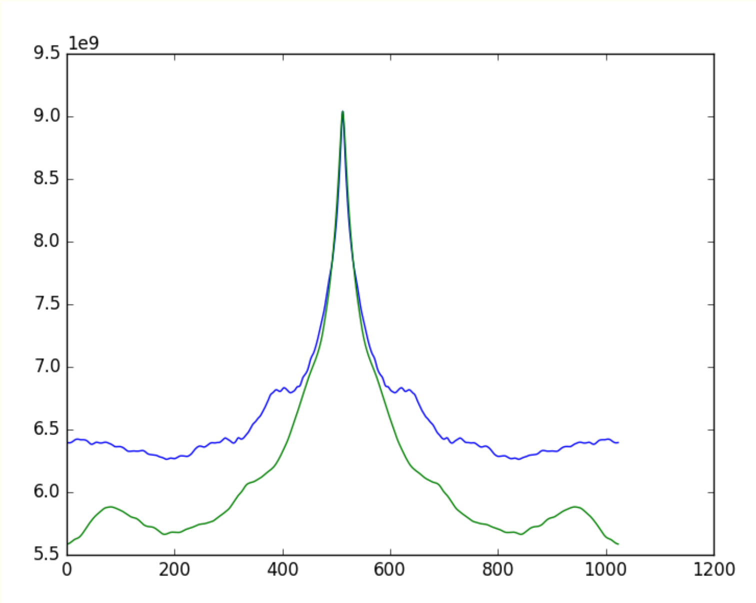
```
In [5]: from crosscorr import crosscorr

In [6]: c0 = crosscorr(i1, i1)

In [7]: c1 = crosscorr(i1, i

In [8]: plt.imshow(c0)

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

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

⇐ Unshifted power spectrum

⇐ Show the cross correlation matrix image

⇐ Show two line profiles





→

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
```

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
```

See: numpy.argmax

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
```

See: numpy.argmax

x, y: coordinates of the maximum

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
```

See: numpy.argmax

x, y: coordinates of the maximum

3 - find maximum

```
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])
```

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
```

See: numpy.argmax

x, y: coordinates of the maximum

3 - find maximum

```
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])
```

Is it possible to find the shift value with sub-pixel accuracy?

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
```

See: numpy.argmax

x, y: coordinates of the maximum

3 - find maximum

```
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])
```

Is it possible to find the shift value with sub-pixel accuracy?

Method: find the maximum of a best-fit 2D gaussian

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)) - da
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]])
```

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)) - da
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]])
```

Note: this function returns a function

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)) - da
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]])
```

Note: this function returns a function

See: np.indices

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
#   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)) - da
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]])
```

Note: this function returns a function

See: np.indices

See: np.ravel

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
#   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)) - da
    p, success = scipy.optimize.leastsq(errorfunction, params)
    return np.array([p[0], p[2], p[1], p[4], p[3]])
```

Note: this function returns a function

See: np.indices

See: np.ravel

See: scipy.optimize.leastsq

Note: we have reordered returned values to have: height, x_center, y_center, x_width, y_width.

Let's try to apply the fitting procedure:

```
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])
```

Let's try to apply the fitting procedure:

```
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])
```

`c0s` and `c1s` are portions of the two correlation matrices around the maximum

Let's try to apply the fitting procedure:

```
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])
```

`c0s` and `c1s` are portions of the two correlation matrices around the maximum

The result is not very encouraging. Let's try to understand why.

Let's try to apply the fitting procedure:

```
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])
```

`c0s` and `c1s` are portions of the two correlation matrices around the maximum

The result is not very encouraging. Let's try to understand why.

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))
```

Let's try to apply the fitting procedure:

```
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])
```

`c0s` and `c1s` are portions of the two correlation matrices around the maximum

The result is not very encouraging. Let's try to understand why.

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))
```

Function `gauss2d()` generates an image from a 2D gaussian function with given parameters

Let’s show the best-fit 2D gaussian, too:

```
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)
```

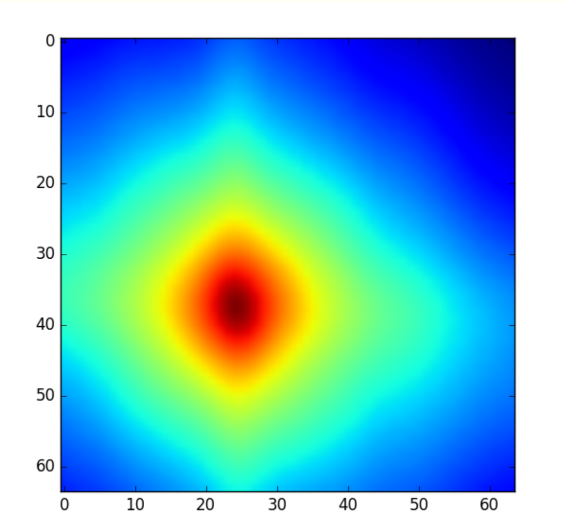
Let's show the best-fit 2D gaussian, too:

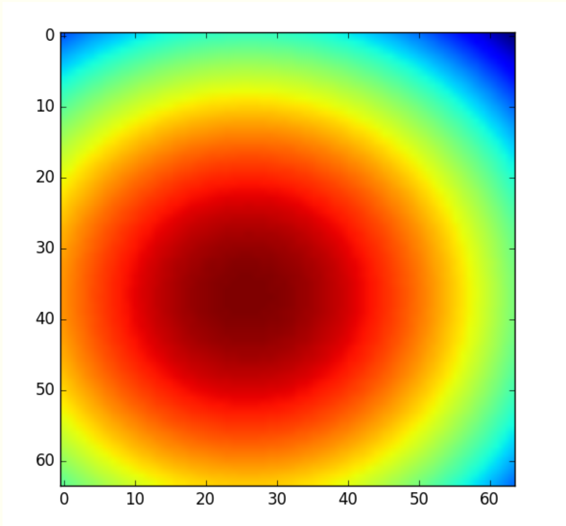
```
In [25]: from gauss2d import gauss2d

In [26]: gbest = gauss2d(fit1[0], 64, 64, *fit1[1:])
                      ⇑                   ⇑
In [27]: plt.imshow(c1s)        Note!

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

Let’s show the best-fit 2D gaussian, too:

```
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)
```

Note!

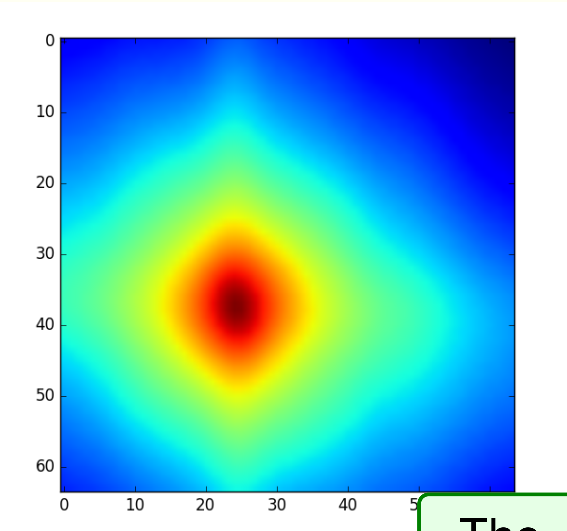


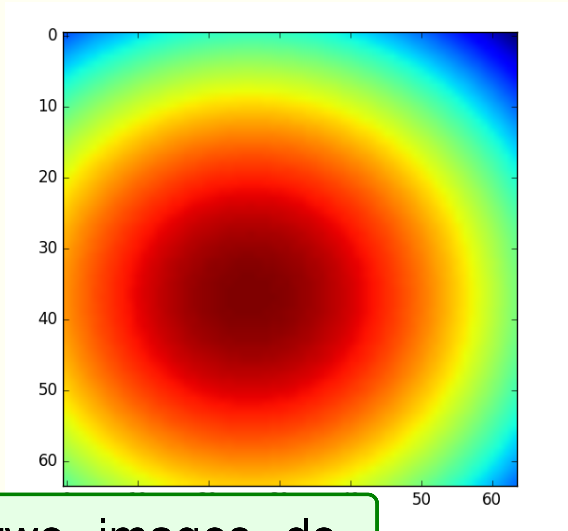


Let's show the best-fit 2D gaussian, too:

```
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)
```

⇑ **Note!** ⇑



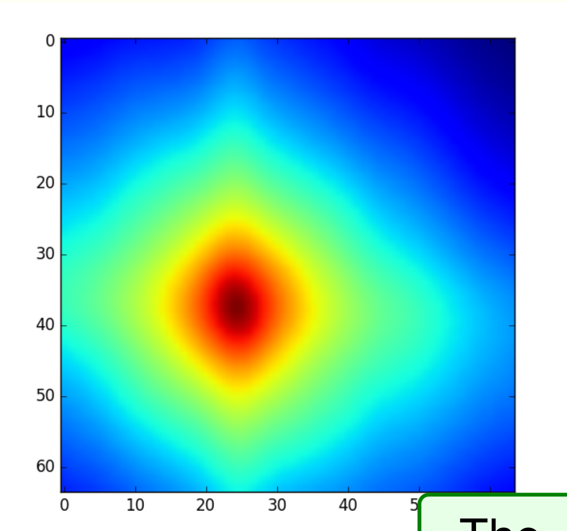
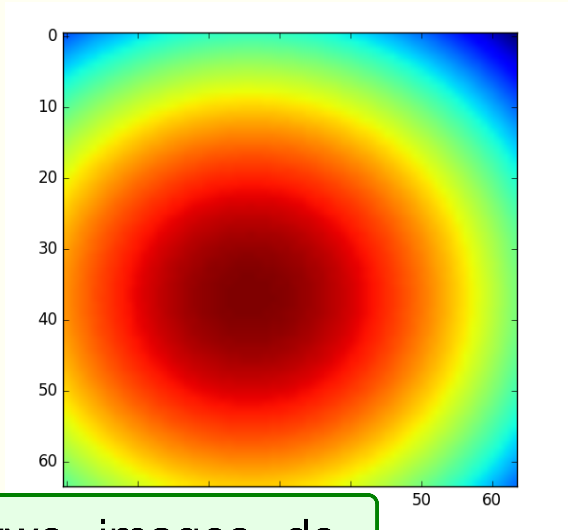


The two images do not look very similar!

Let's show the best-fit 2D gaussian, too:

```
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)
```

Note!

The two images do not look very similar!

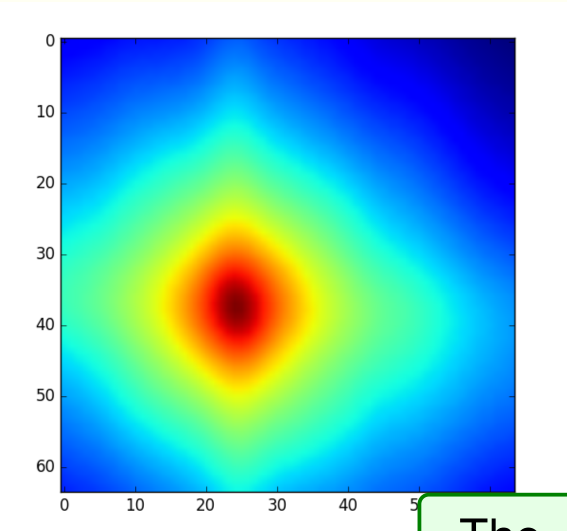
Let's also see two profiles across the maximum:

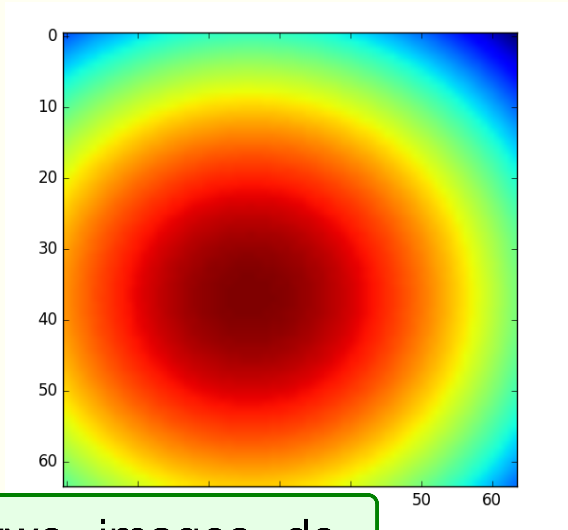
```
In [29]: plt.plot(c1s[:,37])

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

Let’s show the best-fit 2D gaussian, too:

```
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)
```

Note!

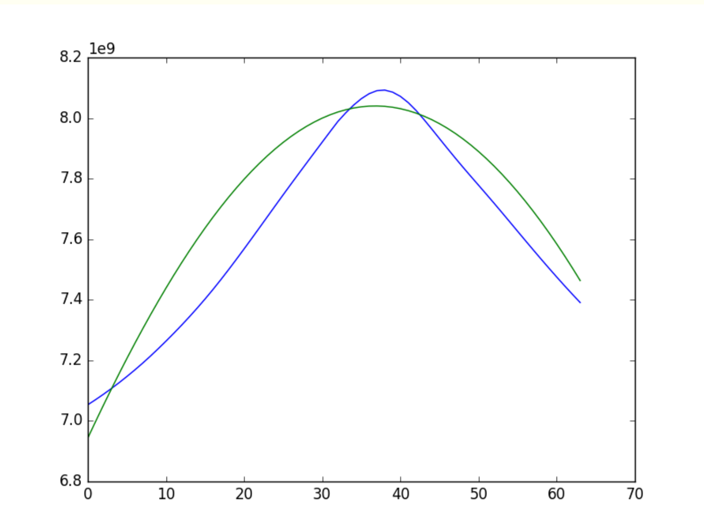




The two images do not look very similar!

Let’s also see two profiles across the maximum:

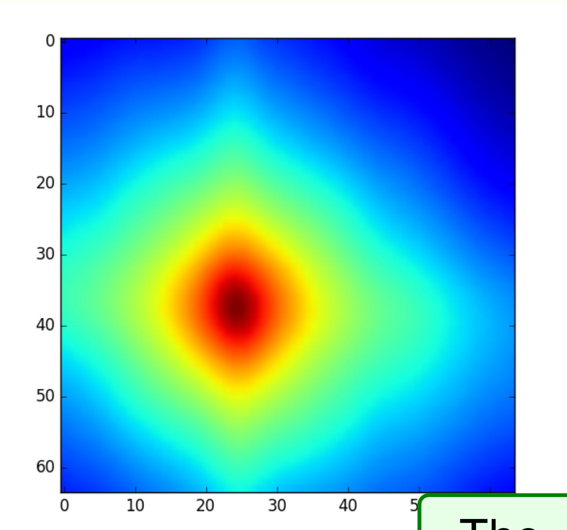
```
In [29]: plt.plot(c1s[:,37])

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

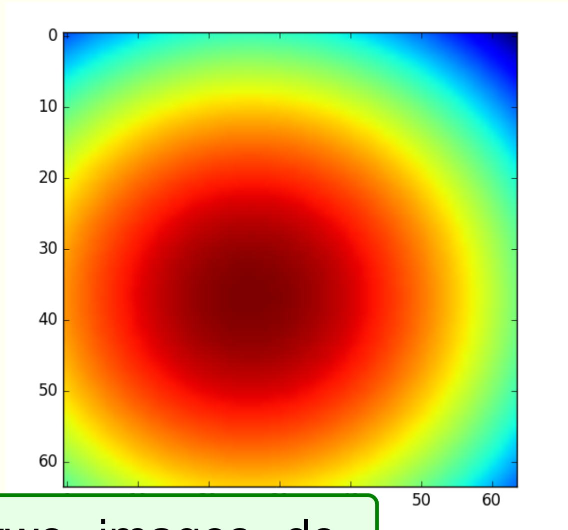


Let's show the best-fit 2D gaussian, too:

```
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)
```

Note! ⇑ ⇑

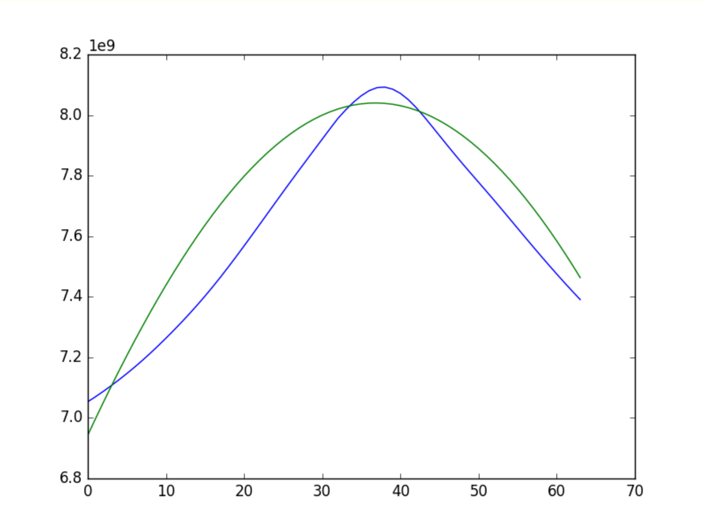




The two images do not look very similar!

Let's also see two profiles across the maximum:

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

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



We can tell at least two reasons why the evaluation is not good:

- The gaussian shape is not particularly fitted for the purpose.
- The asymmetry of the portions of matrices chosen

What else could we do?

- Choose a symmetric portion of the image around the maximum

What else could we do?

- Choose a symmetric portion of the image around the maximum
- Use curves other than a gaussian (e.g.: even degree polynomials)

What else could we do?

- Choose a symmetric portion of the image around the maximum
- Use curves other than a gaussian (e.g.: even degree polynomials)
- Fit 2D curves (either gaussian or polynomial) along X and Y directions separately

What else could we do?

- Choose a symmetric portion of the image around the maximum
- Use curves other than a gaussian (e.g.: even degree polynomials)
- Fit 2D curves (either gaussian or polynomial) along X and Y directions separately
- ....

What else could we do?

- Choose a symmetric portion of the image around the maximum
- Use curves other than a gaussian (e.g.: even degree polynomials)
- Fit 2D curves (either gaussian or polynomial) along X and Y directions separately
- ....

The `astropy` package

- The `astropy` was born as a project started around year 2000 at STScI to develop a “modern” data analysis framework to substitute the `iraf` command language.

The `astropy` package

- The `astropy` was born as a project started around year 2000 at STScI to develop a “modern” data analysis framework to substitute the `iraf` command language.
- One of the early results of the project was the `Numarray` package which was later on merged with a similar project (`Numeric`) to generate `NumPy`.

The astropy package

- The astropy was born as a project started around year 2000 at STScI to develop a “modern” data analysis framework to substitute the iraf command language.
- One of the early results of the project was the Numarray package which was later on merged with a similar project (Numeric) to generate NumPy.
- Later on, in an integration and standardization effort , more sub-packages have been added to astropy.

The `astropy` package

- The `astropy` was born as a project started around year 2000 at STScI to develop a “modern” data analysis framework to substitute the `iraf` command language.
- One of the early results of the project was the `Numarray` package which was later on merged with a similar project (`Numeric`) to generate `NumPy`.
- Later on, in an integration and standardization effort , more sub-packages have been added to `astropy`.
- The development is still going on as you may gather from the fact that `astropy` sub-packages are at different levels of completion.

The astropy package

- The `astropy` was born as a project started around year 2000 at STScI to develop a “modern” data analysis framework to substitute the `iraf` command language.
- One of the early results of the project was the `Numarray` package which was later on merged with a similar project (`Numeric`) to generate `NumPy`.
- Later on, in an integration and standardization effort , more sub-packages have been added to `astropy`.
- The development is still going on as you may gather from the fact that `astropy` sub-packages are at different levels of completion.

astropy content

- Data structures and core functions
  - Constants (astropy.constants)
  - Units and physical quantities (astropy.units)
  - N-dimensional datasets (astropy.nddata)
  - Tabular data (astropy.tables)
  - Astronomical coordinates (astropy.coordinates)
  - World Coordinate System (astropy.wcs)
  - Models and interpolation (astropy.modeling)
  - Analytic functions (astropy.analytic_functions)

astropy content

- Data structures and core functions
  - Constants (astropy.constants)
  - Units and physical quantities (astropy.units)
  - N-dimensional datasets (astropy.nddata)
  - Tabular data (astropy.tables)
  - Astronomical coordinates (astropy.coordinates)
  - World Coordinate System (astropy.wcs)
  - Models and interpolation (astropy.modeling)
  - Analytic functions (astropy.analytic_functions)
- Input/Output
  - FITS file I/O (astropy.io.fits)
  - ASCII tables I/O (astropy.io.ascii)
  - VO tables I/O (astropy.io.votable)
  - VO data access (astropy.io.vo)
  - Miscellaneous (astropy.io.misc) HDF5, YAML and more

astropy content

- Data structures and core functions
  - Constants (astropy.constants)
  - Units and physical quantities (astropy.units)
  - N-dimensional datasets (astropy.nddata)
  - Tabular data (astropy.tables)
  - Astronomical coordinates (astropy.coordinates)
  - World Coordinate System (astropy.wcs)
  - Models and interpolation (astropy.modeling)
  - Analytic functions (astropy.analytic_functions)
- Input/Output
  - FITS file I/O (astropy.io.fits)
  - ASCII tables I/O (astropy.io.ascii)
  - VO tables I/O (astropy.io.votable)
  - VO data access (astropy.io.vo)
  - Miscellaneous (astropy.io.misc) HDF5, YAML and more
- Astronomical algorithms and utilities
  - Cosmology (astropy.cosmology)
  - Convolution and filtering (astropy.convolution)
  - Data visualization (astropy.visualization)
  - Statistics (astropy.stats)

astropy content

- Data structures and core functions
  - Constants (astropy.constants)
  - Units and physical quantities (astropy.units)
  - N-dimensional datasets (astropy.nddata)
  - Tabular data (astropy.tables)
  - Astronomical coordinates (astropy.coordinates)
  - World Coordinate System (astropy.wcs)
  - Models and interpolation (astropy.modeling)
  - Analytic functions (astropy.analytic_functions)
- Input/Output
  - FITS file I/O (astropy.io.fits)
  - ASCII tables I/O (astropy.io.ascii)
  - VO tables I/O (astropy.io.votable)
  - VO data access (astropy.io.vo)
  - Miscellaneous (astropy.io.misc) HDF5, YAML and more
- Astronomical algorithms and utilities
  - Cosmology (astropy.cosmology)
  - Convolution and filtering (astropy.convolution)
  - Data visualization (astropy.visualization)
  - Statistics (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.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.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' '2010-0
```

astropy - examples/1

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' '2010-0
```

astropy - examples/2

Problem: how hard was at Hypparcos time to find the day of solstice?

Let's evaluate the length of the shadow of a 1 m long pole at noon in three days around the solstice

Problem: how hard was at Hypparcos time to find the day of solstice?

Let's evaluate the length of the shadow of a 1 m long pole at noon in three days around the solstice

file: solstice.py *(finds the time (UTC) of summer solstice in 2018 (1 sec approx.))*

```
import time
from astropy import coordinates
from astropy.time import Time

start = time.mktime((2018,6,20,0,0,0,0,0,-1))
increment = 86400

while increment >= 1:
    tim_2 = 0
    tim_1 = 0
    dec_2 = -100.0
    dec_1 = -100.0
    utime0 = start
    print("Step: %6d"%increment, end=" - ")
    while True:
        tim_0 = Time(utime0, format="unix")
        sun_dec = coordinates.get_sun(tim_0).dec.value
        if sun_dec <= dec_1:
            break
        utime0 += increment
        dec_2 = dec_1
        dec_1 = sun_dec
        tim_2 = tim_1
        tim_1 = tim_0
    increment = int(increment/2.)
    start = tim_2.value
    tim_1.format="iso"
    print(tim_1, dec_1)
```

Problem: how hard was at Hypparcos time to find the day of solstice?

Let's evaluate the length of the shadow of a 1 m long pole at noon in three days around the solstice

file: solstice.py *(finds the time (UTC) of summer solstice in 2018 (1 sec approx.))*

```
import time
from astropy import coordinates
from astropy.time import Time

start = time.mktime((2018,6,20,0,0,0,0,0,-1))
increment = 86400

while increment >= 1:
    tim_2 = 0
    tim_1 = 0
    dec_2 = -100.0
    dec_1 = -100.0
    utime0 = start
    print("Step: %6d"%increment, end=" - ")
    while True:
        tim_0 = Time(utime0, format="unix")
        sun_dec = coordinates.get_sun(tim_0).dec.value
        if sun_dec <= dec_1:
            break
        utime0 += increment
        dec_2 = dec_1
        dec_1 = sun_dec
        tim_2 = tim_1
        tim_1 = tim_0
    increment = int(increment/2.)
    start = tim_2.value
    tim_1.format="iso"
    print(tim_1, dec_1)
```

→

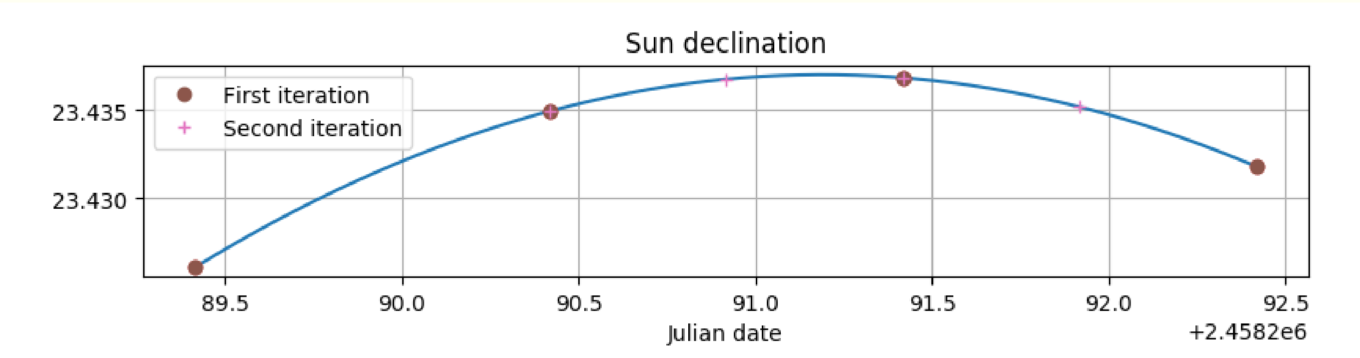
Let's run `solstice.py` from the ipython prompt:

```
$ ipython --pylab

In [1]: %run solstice.py
Step: 86400 - 2018-06-21 22:00:00.000 23.436812973859176
Step: 43200 - 2018-06-21 22:00:00.000 23.436812973859173
Step: 21600 - 2018-06-21 16:00:00.000 23.436986791903372
Step: 10800 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  5400 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  2700 - 2018-06-21 16:45:00.000 23.43698863303954
Step:  1350 - 2018-06-21 16:45:00.000 23.43698863303954
Step:   675 - 2018-06-21 16:33:45.000 23.436988804079192
Step:   337 - 2018-06-21 16:33:44.000 23.436988804020356
Step:   168 - 2018-06-21 16:33:43.000 23.436988803960602
Step:    84 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    42 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    21 - 2018-06-21 16:34:46.000 23.43698880592122
Step:    10 - 2018-06-21 16:34:45.000 23.436988805918734
Step:     5 - 2018-06-21 16:34:50.000 23.43698880592192
Step:     2 - 2018-06-21 16:34:49.000 23.436988805923132
Step:     1 - 2018-06-21 16:34:48.000 23.43698880592342
```

Let's run `solstice.py` from the ipython prompt:

```
$ ipython --pylab

In [1]: %run solstice.py
Step: 86400 - 2018-06-21 22:00:00.000 23.436812973859176
Step: 43200 - 2018-06-21 22:00:00.000 23.436812973859173
Step: 21600 - 2018-06-21 16:00:00.000 23.436986791903372
Step: 10800 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  5400 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  2700 - 2018-06-21 16:45:00.000 23.43698863303954
Step:  1350 - 2018-06-21 16:45:00.000 23.43698863303954
Step:   675 - 2018-06-21 16:33:45.000 23.436988804079192
Step:   337 - 2018-06-21 16:33:44.000 23.436988804020356
Step:   168 - 2018-06-21 16:33:43.000 23.436988803960602
Step:    84 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    42 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    21 - 2018-06-21 16:34:46.000 23.43698880592122
Step:    10 - 2018-06-21 16:34:45.000 23.436988805918734
Step:     5 - 2018-06-21 16:34:50.000 23.43698880592192
Step:     2 - 2018-06-21 16:34:49.000 23.436988805923132
Step:     1 - 2018-06-21 16:34:48.000 23.43698880592342
```

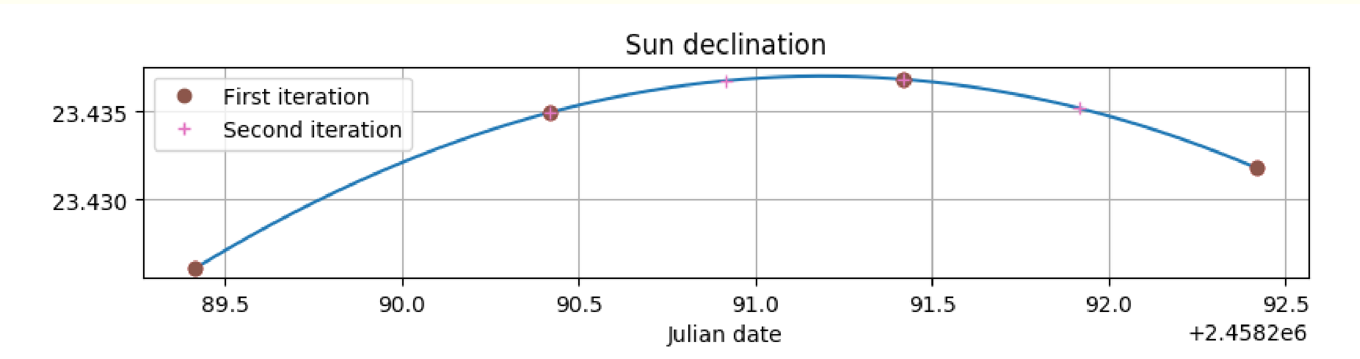
Here's how it works:



Let's run `solstice.py` from the ipython prompt:

```
$ ipython --pylab

In [1]: %run solstice.py
Step: 86400 - 2018-06-21 22:00:00.000 23.436812973859176
Step: 43200 - 2018-06-21 22:00:00.000 23.436812973859173
Step: 21600 - 2018-06-21 16:00:00.000 23.436986791903372
Step: 10800 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  5400 - 2018-06-21 16:00:00.000 23.436986791903372
Step:  2700 - 2018-06-21 16:45:00.000 23.43698863303954
Step:  1350 - 2018-06-21 16:45:00.000 23.43698863303954
Step:   675 - 2018-06-21 16:33:45.000 23.436988804079192
Step:   337 - 2018-06-21 16:33:44.000 23.436988804020356
Step:   168 - 2018-06-21 16:33:43.000 23.436988803960602
Step:    84 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    42 - 2018-06-21 16:35:07.000 23.436988805760024
Step:    21 - 2018-06-21 16:34:46.000 23.43698880592122
Step:    10 - 2018-06-21 16:34:45.000 23.436988805918734
Step:     5 - 2018-06-21 16:34:50.000 23.43698880592192
Step:     2 - 2018-06-21 16:34:49.000 23.436988805923132
Step:     1 - 2018-06-21 16:34:48.000 23.43698880592342
```

Here's how it works:



→

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

⇐ Just to use the imports

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

Just to use the imports

Make a sequence of `Time` objects

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

Just to use the imports

Make a sequence of `Time` objects

Compute the corresponding sequence of sun declinations

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

⇐ Just to use the imports (In [1])

Make a sequence of `Time` objects (In [4])

⇐ ! (In [6])

Compute the corresponding sequence of sun declinations (In [5])

How the plot is generated

(Ipython output suppressed)

```
$ ipython --pylab

In [1]: %run solstice.py
 ...
In [2]: start = time.mktime((2018,6,20,0,0,0,0,0,-1))
In [3]: end = time.mktime((2018,6,23,0,0,0,0,0,-1))
In [4]: times = Time(np.arange(start,end,60), format="unix")
In [5]: sundec = coordinates.get_sun(times).dec
In [6]: times.format="jd"
In [7]: plot(times.value, sundec.value)
In [8]: grid()
In [9]: days0=Time(np.linspace(times[0].value,times[-1].value,4), format="jd")
In [10]: dec0=coordinates.get_sun(days0).dec
In [11]: plot(days0.value,dec0.value,"o",label="First iteration")
In [12]: days1=Time(np.linspace(days0[1].value,days0[3].value,4), format="jd")
In [13]: dec1=coordinates.get_sun(days1).dec
In [14]: plot(days1.value,dec1.value,"+",label="Second iteration")
In [15]: plt.legend()
In [16]: ax=plt.gca()
In [17]: ax.set_aspect(50)
In [18]: ax.set_aspect(60)
```

Just to use the imports (points to In [1])

Make a sequence of `Time` objects (points to In [4])

Compute the corresponding sequence of sun declinations (points to In [5])

! (points to In [6])

→

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_20.alt.value)*0.017453292519943295)
In [21]: shadow_21 = np.tan((90-sun_pos_21.alt.value)*0.017453292519943295)
In [22]: shadow_22 = np.tan((90-sun_pos_22.alt.value)*0.017453292519943295)

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.38570068931067275, 0.3856520089291466, 0.3857434821516582)
```

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

See: `TimeDelta` and `...get_site_names()`

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_20.alt.value)*0.017453292519943295)
In [21]: shadow_21 = np.tan((90-sun_pos_21.alt.value)*0.017453292519943295)
In [22]: shadow_22 = np.tan((90-sun_pos_22.alt.value)*0.017453292519943295)

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.3857006893106727 5, 0.3856520089291466, 0.3857434821516582)
```

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

See: `TimeDelta` and `...get_site_names()`

Noon at Medicina

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_20.alt.value)*0.017453292519943295)
In [21]: shadow_21 = np.tan((90-sun_pos_21.alt.value)*0.017453292519943295)
In [22]: shadow_22 = np.tan((90-sun_pos_22.alt.value)*0.017453292519943295)

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.38570068931067275, 0.3856520089291466, 0.3857434821516582)
```

astropy - examples/2 - contd.

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_20.alt.value)*0.017453292519943295)
In [21]: shadow_21 = np.tan((90-sun_pos_21.alt.value)*0.017453292519943295)
In [22]: shadow_22 = np.tan((90-sun_pos_22.alt.value)*0.017453292519943295)

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.3857006893106727 5, 0.3856520089291466, 0.3857434821516582)
```

See: `TimeDelta` and `...get_site_names()`

Noon at Medicina

Sun coordinates at noon

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta
```

See: `TimeDelta` and `...get_site_names()`

```
In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec
```

Noon at Medicina

```
In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)
```

Sun coordinates at noon

```
In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)
```

Sun AltAz coordinates

```
In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_20.alt.value)*0.017453292519943295)
In [21]: shadow_21 = np.tan((90-sun_pos_21.alt.value)*0.017453292519943295)
In [22]: shadow_22 = np.tan((90-sun_pos_22.alt.value)*0.017453292519943295)

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.3857006893106727 5, 0.3856520089291466, 0.3857434821516582)
```

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_
In [21]: shadow_21 = np.tan((90-sun_pos_
In [22]: shadow_22 = np.tan((90-sun_pos_

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.3857006893106727 5, 0.3856520089291466, 0.3857434821516582)
```

See: `TimeDelta` and `...get_site_names()`

Noon at Medicina

Sun coordinates at noon

Sun AltAz coordinates

The difference in length is about 0.5 mm for a pole of 10 meters!

Maybe the winter solstice is easier?

An ancient astronomer whould measure the length of a pole's shadow every day at solar noon and note the minimum

```
$ ipython --pylab

In [1]: from astropy import coordinates as coord
In [2]: from astropy.time import Time, TimeDelta

In [3]: medicina=coord.EarthLocation.of_site("medicina")
In [4]: lon_sec = TimeDelta(medicina.lon.value*240., format="sec")

In [5]: noon_20 = Time("2018-06-20 12:00:00")-lon_sec
In [6]: noon_21 = Time("2018-06-21 12:00:00")-lon_sec
In [7]: noon_22 = Time("2018-06-22 12:00:00")-lon_sec

In [8]: sun_20 = coord.get_sun(noon_20)
In [9]: sun_21 = coord.get_sun(noon_21)
In [10]: sun_22 = coord.get_sun(noon_22)

In [11]: altaz_20 = coord.AltAz(location=medicina, obstime=noon_20)
In [12]: altaz_21 = coord.AltAz(location=medicina, obstime=noon_21)
In [13]: altaz_22 = coord.AltAz(location=medicina, obstime=noon_22)

In [14]: sun_pos_20 = sun_20.transform_to(altaz_20)
In [15]: sun_pos_21 = sun_21.transform_to(altaz_21)
In [16]: sun_pos_22 = sun_22.transform_to(altaz_22)

In [17]: sun_pos_20.alt
Out[17]: <Latitude 68.90833863 deg>
In [18]: sun_pos_21.alt
Out[18]: <Latitude 68.91076665 deg>
In [19]: sun_pos_22.alt
Out[19]: <Latitude 68.90620432 deg>

In [20]: shadow_20 = np.tan((90-sun_pos_
In [21]: shadow_21 = np.tan((90-sun_pos_
In [22]: shadow_22 = np.tan((90-sun_pos_

In [23]: shadow_20, shadow_21, shadow_22
Out[23]: (0.3857006893106727 5, 0.3856520089291466, 0.3857434821516582)
```

See: `TimeDelta` and `...get_site_names()`

Noon at Medicina

Sun coordinates at noon

Sun AltAz coordinates

The difference in length is about 0.5 mm for a pole of 10 meters!

Maybe the winter solstice is easier?

file: winter_sol.py

```
import time
from astropy import coordinates
from astropy.time import Time

start = time.mktime((2018,12,19,0,0,0,0,0,-1))
increment = 86400

while increment >= 1:
    tim_2 = 0
    tim_1 = 0
    dec_2 = 100.0
    dec_1 = 100.0
    utime0 = start
    print("Step: %6d"%increment, end=" - ")
    while True:
        tim_0 = Time(utime0, format="unix")
        sun_dec = coordinates.get_sun(tim_0).dec.value
        if sun_dec >= dec_1:
            break
        utime0 += increment
        dec_2 = dec_1
        dec_1 = sun_dec
        tim_2 = tim_1
        tim_1 = tim_0
    increment = int(increment/2.)
    start = tim_2.value
    tim_1.format="iso"
    print(tim_1, dec_1)
```

file: winter_sol.py

```
import time
from astropy import coordinates
from astropy.time import Time

start = time.mktime((2018,12,19,0,0,0,0,0,-1))
increment = 86400

while increment >= 1:
    tim_2 = 0
    tim_1 = 0
    dec_2 = 100.0
    dec_1 = 100.0
    utime0 = start
    print("Step: %6d"%increment, end=" - ")
    while True:
        tim_0 = Time(utime0, format="unix")
        sun_dec = coordinates.get_sun(tim_0).dec.value
        if sun_dec >= dec_1:
            break
        utime0 += increment
        dec_2 = dec_1
        dec_1 = sun_dec
        tim_2 = tim_1
        tim_1 = tim_0
    increment = int(increment/2.)
    start = tim_2.value
    tim_1.format="iso"
    print(tim_1, dec_1)
```

The red arrows show the differences with respect to `solstice.py`

file: winter_sol.py

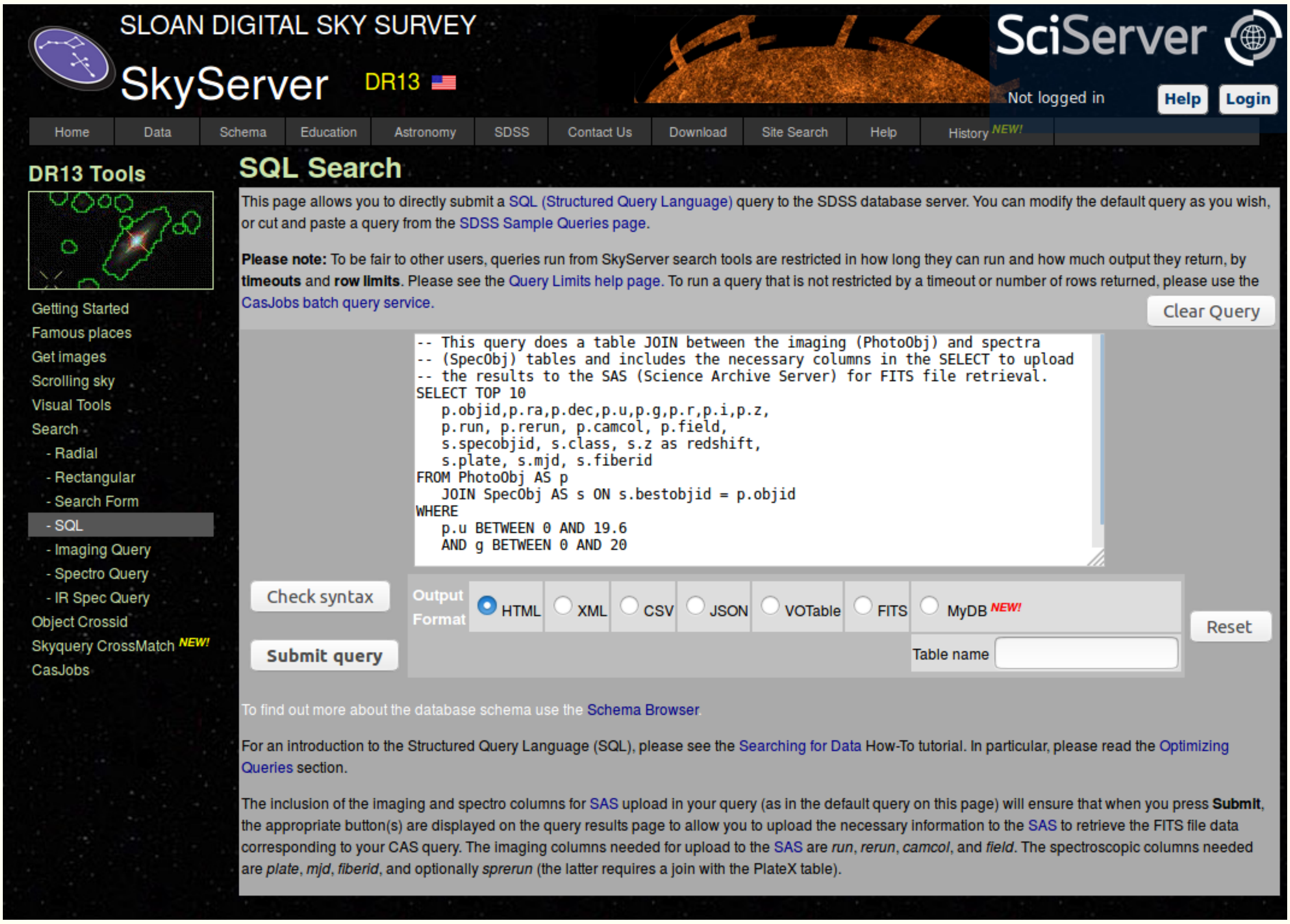
```
import time
from astropy import coordinates
from astropy.time import Time

start = time.mktime((2018,12,19,0,0,0,0,0,-1))
increment = 86400

while increment >= 1:
    tim_2 = 0
    tim_1 = 0
    dec_2 = 100.0
    dec_1 = 100.0
    utime0 = start
    print("Step: %6d"%increment, end=" - ")
    while True:
        tim_0 = Time(utime0, format="unix")
        sun_dec = coordinates.get_sun(tim_0).dec.value
        if sun_dec >= dec_1:
            break
        utime0 += increment
        dec_2 = dec_1
        dec_1 = sun_dec
        tim_2 = tim_1
        tim_1 = tim_0
    increment = int(increment/2.)
    start = tim_2.value
    tim_1.format="iso"
    print(tim_1, dec_1)
```

The red arrows show the differences with respect to `solstice.py`

→

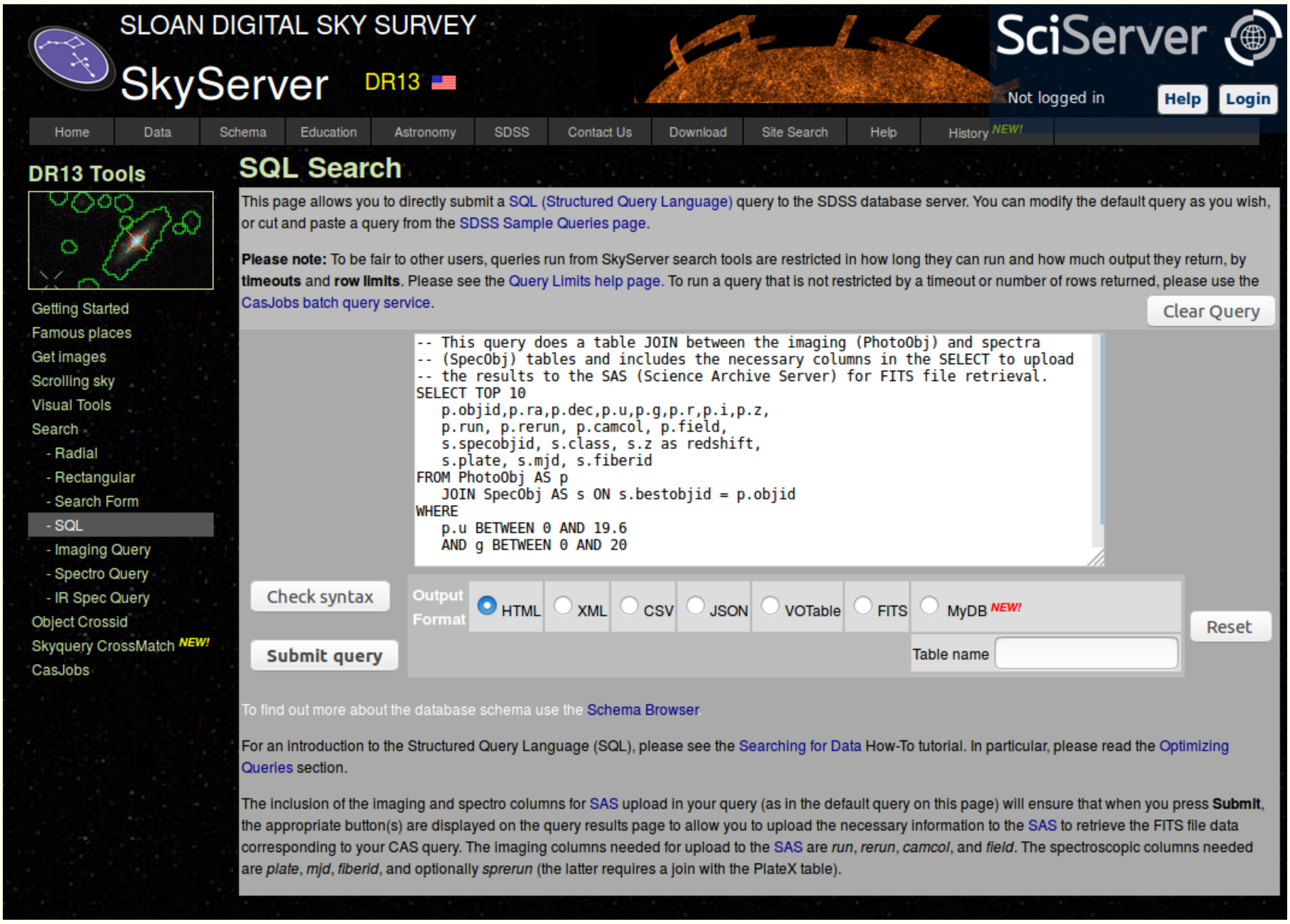
The Sloan Digital Sky Service

- SQL Interface
  Allows access to info and data via SQL queries.



The Sloan Digital Sky Service

- SQL Interface
  Allows access to info and data via SQL queries.



→

Data retrieval automation

file: sdss.py

```
import sys, os, requests, json

URL="http://cas.sdss.org/public/en/tools/search/x_results.aspx"
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": "json", "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)
```

Data retrieval

The `requests` standard module provides simple functions to manage HTTP queries

file: sdss.py

```
import sys, os, requests, json

URL="http://cas.sdss.org/public/en/tools/search/x_results.aspx"
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": "json", "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)
```

Data retrieval

The `requests` standard module provides s… age HTTP

Note that the access to this URL seems to be allowed only from specific network addresses

file: sdss.py

```
import sys, os, requests, json

URL="http://cas.sdss.org/public/en/tools/search/x_results.aspx"
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": "json", "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)
```

Data retrieval

file: sdss.py

> The `requests` standard module provides s... age HTTP

> Note that the access to this URL seems to be allowed only from specific network addresses

> Complex SQL query to retrieve top 10 objects in a square region around given coordinates

```python
import sys, os, requests, json

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

QRY = """SELECT TOP 10
   p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.
   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": "json", "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)
```

Data retrieval

file: sdss.py

> The `requests` standard module provides s... age HTTP

> Note that the access to this URL seems to be allowed only from specific network addresses

```
import sys, os, requests, json

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

QRY = """SELECT TOP 10
   p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.
   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 =
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": "json", "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)
```

> Complex SQL query to retrieve top 10 objects in a square region around given coordinates

> “%f” is to be substituted with provided coordinate values

astropy - examples/4

Data retrieval

file: sdss.py

> The `requests` standard module provides s
> age HTTP

> Note that the access to this URL seems to be allowed only from specific network addresses

```python
import sys, os, requests, json

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

QRY = """SELECT TOP 10
   p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.
   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 =
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": "json", "search
    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)
```

> Complex SQL query to retrieve top 10 objects in a square region around given coordinates

> “%f” is to be substituted with provided coordinate values

> Sending the HTTP request

Data retrieval automation

```
$ ipython
...
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',
.....
```

Data retrieval automation

```
$ ipython
...
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',
.....
```

Launch `sdss.py` with proper arguments

Data retrieval automation

```
$ ipython
...
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',
.....
```

Launch `sdss.py` with proper arguments (← In [1])

The result is a list of dictionaries (converted from the json reply) (← Out[2])

Data retrieval automation

What if we want to get data files, too?

Data retrieval automation

What if we want to get data files, too?

file: getfits.py

```
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("Created file:", localname)
```

Data retrieval automation

What if we want to get data files, too?

file: getfits.py

Note the new version of python string interpolation

```
import os
from astropy.utils.data impo

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("Created file:", localname)
```

Data retrieval automation

What if we want to get data files, too?

file: getfits.py

Note the new version of python string interpolation

```
import os
from astropy.utils.data impo

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("Created file:", localname)
```

download_file(): "all-in-one" function to retrieve a file via HTTP

Data retrieval automation

What if we want to get data files, too?

file: getfits.py

```
import os
from astropy.utils.data impo

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("Created file:", localname)
```

Note the new version of python string interpolation

download_file(): "all-in-one" function to retrieve a file via HTTP

Downloading data files:

```
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
```

The FITS format

- The FITS (**F**lexible **I**mage **T**ransport **S**ystem) format is a data archiving standard largely used in astronomy.

The FITS format

- The FITS (**F**lexible **I**mage **T**ransport **S**ystem) format is a data archiving standard largely used in astronomy.
- FITS I/O support in python was traditionally provided by module: `pyfits`.

The FITS format

- The FITS (**F**lexible **I**mage **T**ransport **S**ystem) format is a data archiving standard largely used in astronomy.
- FITS I/O support in python was traditionally provided by module: `pyfits`.
- Now the module is part of `astropy` (sub-module: `astropy.io.fits`)

The FITS format

- The FITS (**F**lexible **I**mage **T**ransport **S**ystem) format is a data archiving standard largely used in astronomy.
- FITS I/O support in python was traditionally provided by module: `pyfits`.
- Now the module is part of `astropy` (sub-module: `astropy.io.fits`)

```
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, D, D, D, E, E]

In [12]: ffile[0].header
Out[12]:
SIMPLE =                    T /
BITPIX =                  -32 / 32 bit floating point
NAXIS  =                    2
.........
```

The FITS format

- The FITS (**F**lexible **I**mage **T**ransport **S**ystem) format is a data archiving standard largely used in astronomy.
- FITS I/O support in python was traditionally provided by module: `pyfits`.
- Now the module is part of `astropy` (sub-module: `astropy.io.fits`)

```
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, D, D, D, E, E]

In [12]: ffile[0].header
Out[12]:
SIMPLE =                T /
BITPIX =              -32 / 32 bit floating point
NAXIS  =                2
.........
```

Let's go back to moon phase ...

file: mphase.py

```
import numpy as np
from astropy.coordinates import get_moon, get_sun
from astropy.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.now()
    print("Moon phase right now:", mphase(now).to("deg"))
```

Let’s go back to moon phase ...

file: mphase.py

```
import numpy as np
from astropy.coordinates import get_moon, get_sun
from astropy.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.now()
    print("Moon phase right now:", mphase(now).to("deg"))
```

Let’s use it from ipython:

```
$ ipython
.....
In [1]: %run mphase.py
Moon phase right now: 132.2569254531053 deg
```

Let's go back to moon phase ...

file: mphase.py

```
import numpy as np
from astropy.coordinates import get_moon, get_sun
from astropy.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.now()
    print("Moon phase right now:", mphase(now).to("deg"))
```

Let's use it from ipython:

```
$ ipython
.....
In [1]: %run mphase.py
Moon phase right now: 132.2569254531053 deg
```

Let's play around with coordinate objects:

Let's go back to moon phase ...

file: mphase.py

```
import numpy as np
from astropy.coordinates import get_moon, get_sun
from astropy.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.now()
    print("Moon phase right now:", mphase(now).to("deg"))
```

Let's use it from ipython:

```
$ ipython
.....
In [1]: %run mphase.py
Moon phase right now: 132.2569254531053 deg
```

Let's play around with coordinate objects:

```
In [3]: help(get_sun)

In [4]: sun=get_sun(now)

In [5]: ?sun

In [6]: sun.distance
Out[6]: <Distance 0.983694400624037б AU>

In [7]: moon=get_moon(now)

In [8]: sun.separation(moon)
Out[8]: <Angle 47.627267333388936 deg>
```

How do we find the starting point for moon phase used in the example about births?

How do we find the starting point for moon phase used in the example about births?

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
```

How do we find the starting point for moon phase used in the example about births?

file: newmoon.py

This new version of `mphase()` gets the time input as `time.time()` values instead of `astropy.Time()`

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

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

How do we find the starting point for moon phase used in the example about births?

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
```

This new version of `mphase()` gets the time input as `time.time()` values instead of `astropy.Time()`

Let's go on from the ipython prompt:

```
$ ipython --pylab

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()
```

How do we find the starting point for moon phase used in the example about births?

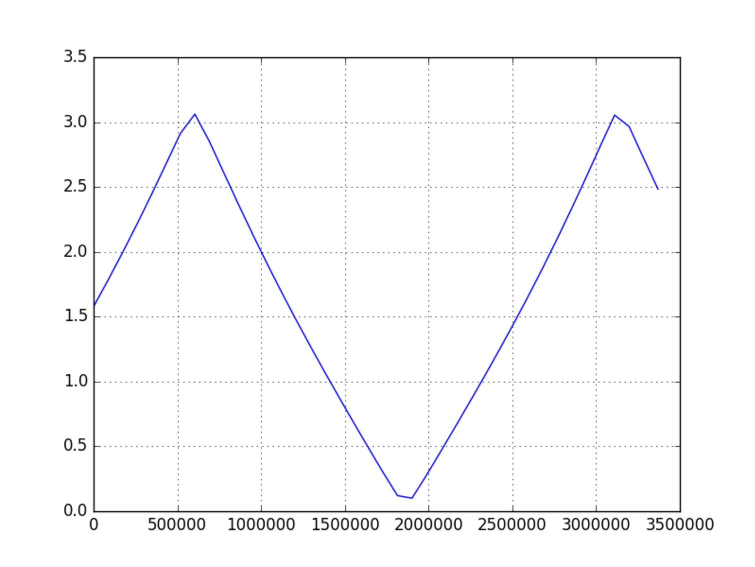
file: newmoon.py

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

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

This new version of `mphase()` gets the time input as `time.time()` values instead of `astropy.Time()`

Let's go on from the ipython prompt:

```
$ ipython --pylab

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()
```



The plot shows the moon phase starting from 1/1/1970 for 40 days.

Now we look for a minimum of the `mphase()` function using an optimization function from `scipy`.

Now we look for a minimum of the `mphase()` function using an optimization function from `scipy`.

```
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: 0.057890689131074945
 message: 'Solution found.'
    nfev: 21
  status: 0
 success: True
       x: 1862913.8958219313
```

Now we look for a minimum of the `mphase()` function using an optimization function from `scipy`.

```
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: 0.057890689131074945
 message: 'Solution found.'
    nfev: 21
  status: 0
 success: True
       x: 1862913.8958219313
```

Detour: In last year's seminar I used the very same example with python 2.7 and got the following result:

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

Now we look for a minimum of the `mphase()` function using an optimization function from `scipy`.

```
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: 0.057890689131074945
 message: 'Solution found.'
    nfev: 21
  status: 0
 success: True
       x: 1862913.8958219313
```

Detour: In last year's seminar I used the very same example with python 2.7 and got the following result:

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

Which differs for about 0.03 seconds and finds a slightly lower value !!!

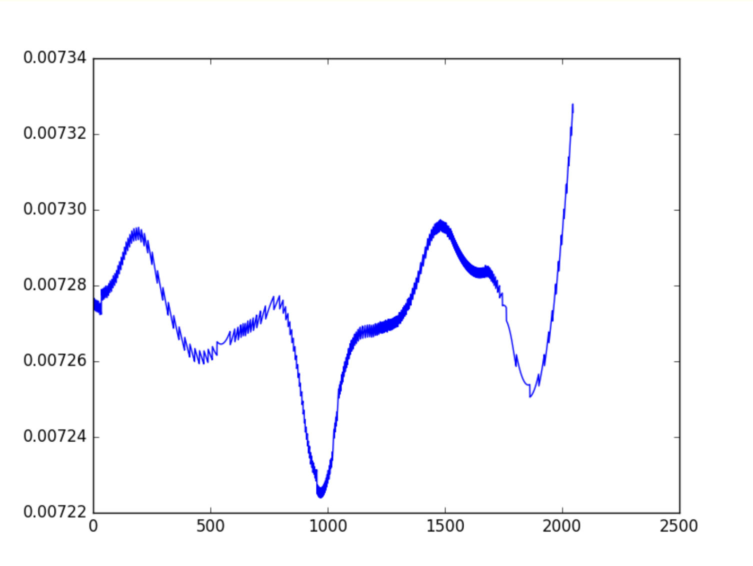
How to read FITS files

How to read FITS files

```
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)
```

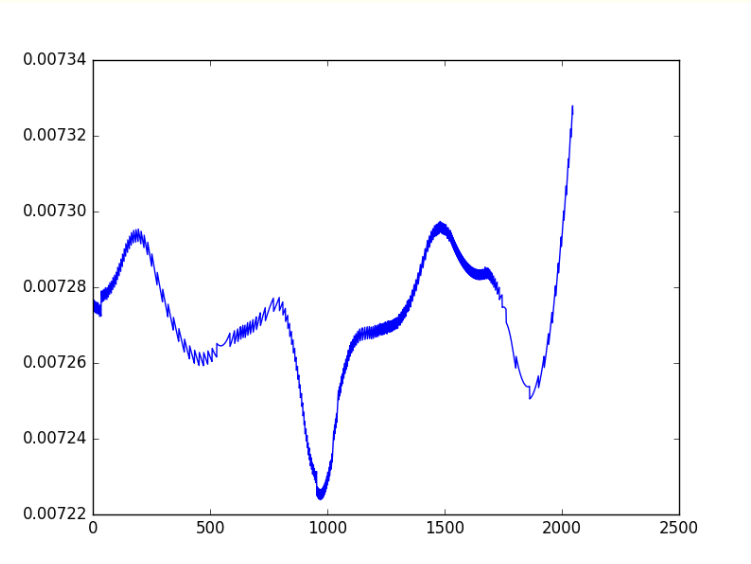
How to read FITS files

```
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)
```



How to read FITS files

```
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)
```



How to read table data from FITS files

```
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','FIEL

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.FRAMERATE = '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
```

More about FITS - 1

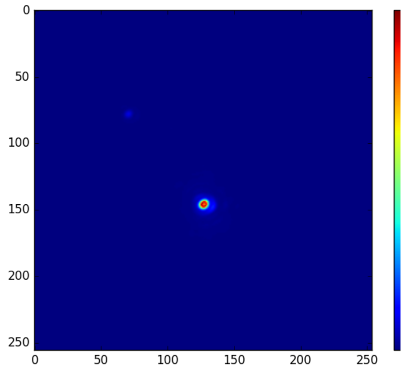
Looking into a FITS file

```
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.FRAMERATE = '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
```

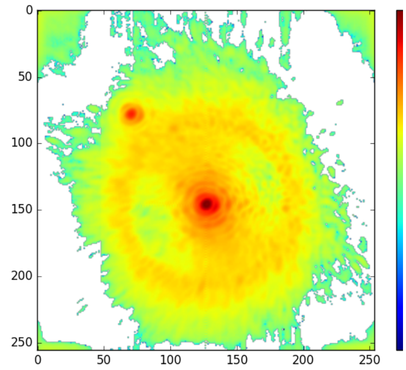
Let's look into the file content

```
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")
```

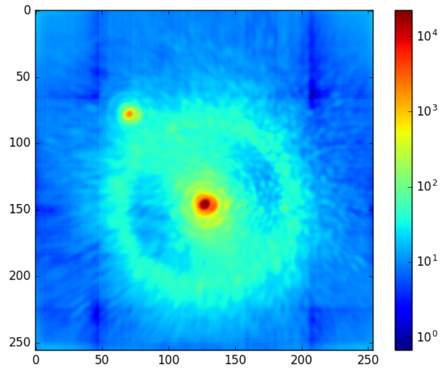
And here are the corresponding images:



In [5-6]



In [8-9]



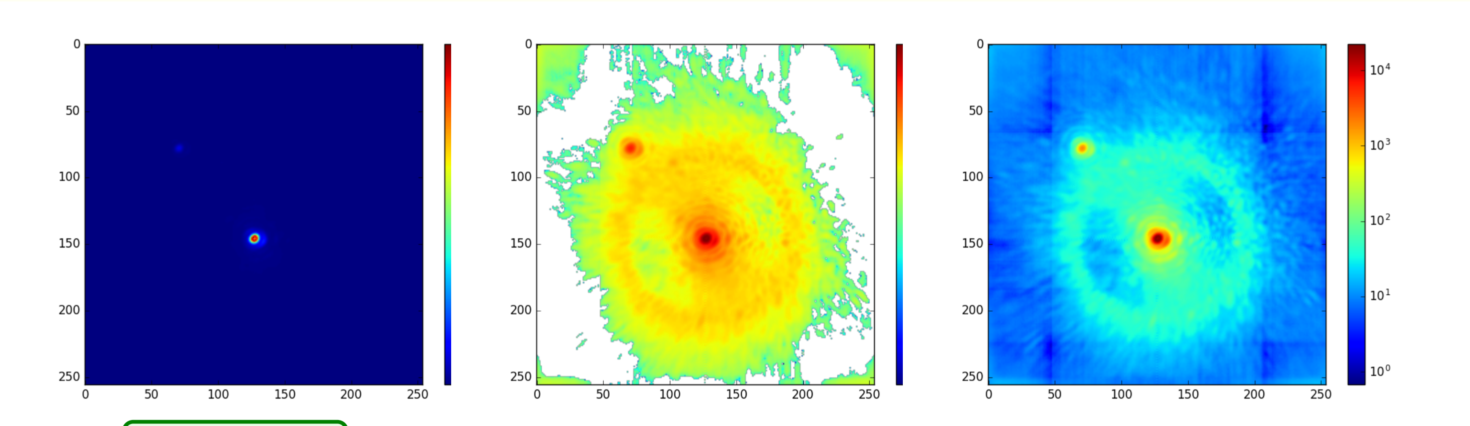
In [12-13]

Let's look into the file content

```
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")
```

Default linear colormap

And here are the corresponding images:



Linear

In [5-6] In [8-9] In [12-13]

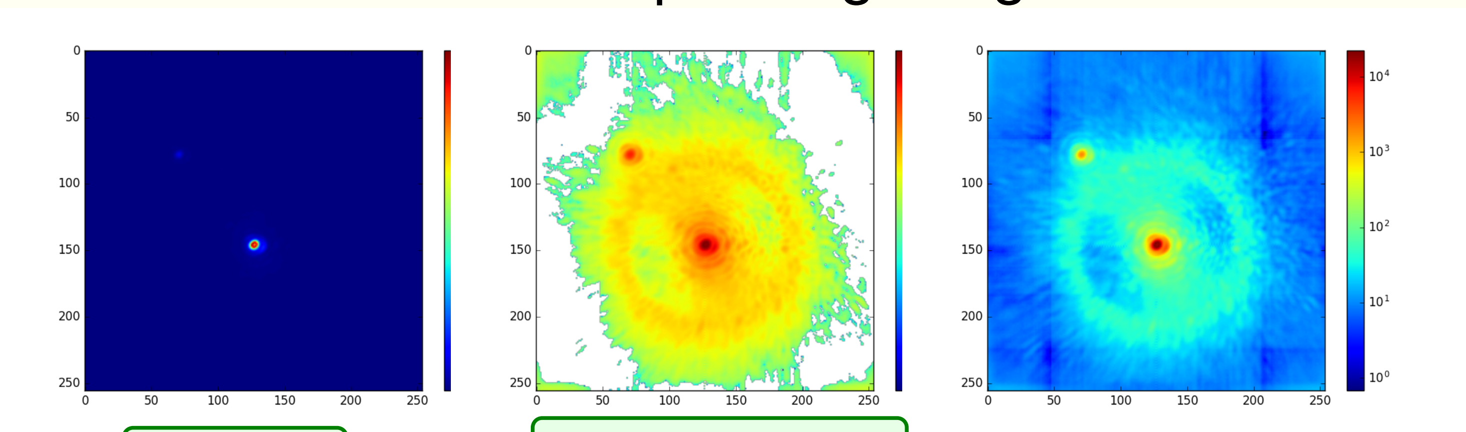
Let's look into the file content

```
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")
```

Default linear colormap (In [5])

Logarithmic colormap (In [8])

And here are the corresponding images:



Linear — In [5-6]

Logarithmic 1 — In [8-9]

In [12-13]

Let’s look into the file content

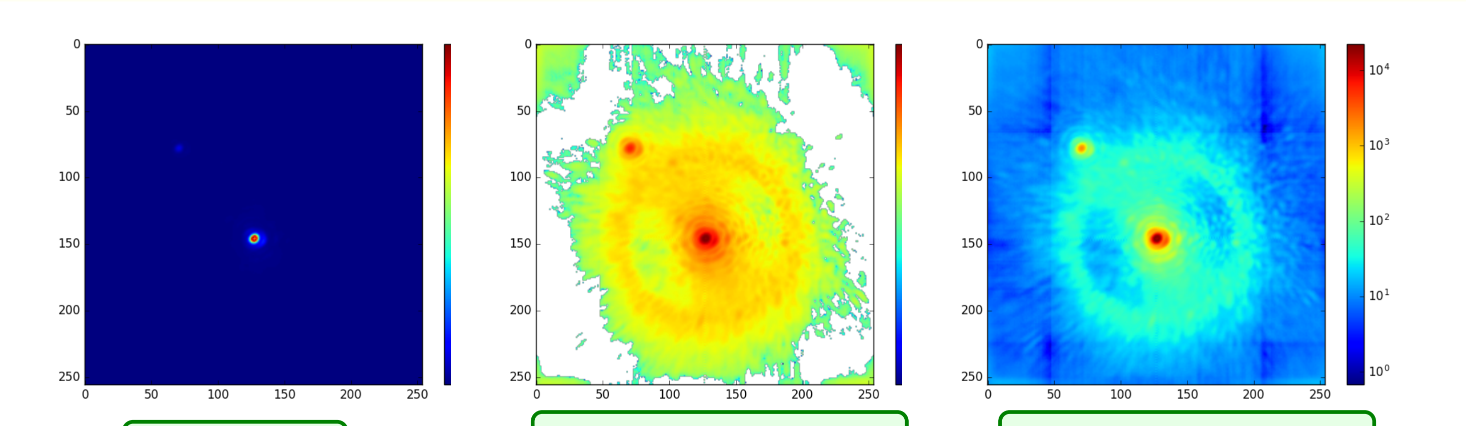
```
In [4]: scidata = hdus[0].data

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")
```

Default linear colormap

Logarithmic colormap

Shifted up image

And here are the corresponding images:



Linear

In [5-6]

Logarithmic 1

In [8-9]

Logarithmic 2

In [12-13]

More about FITS - 2

Let’s look into the file content

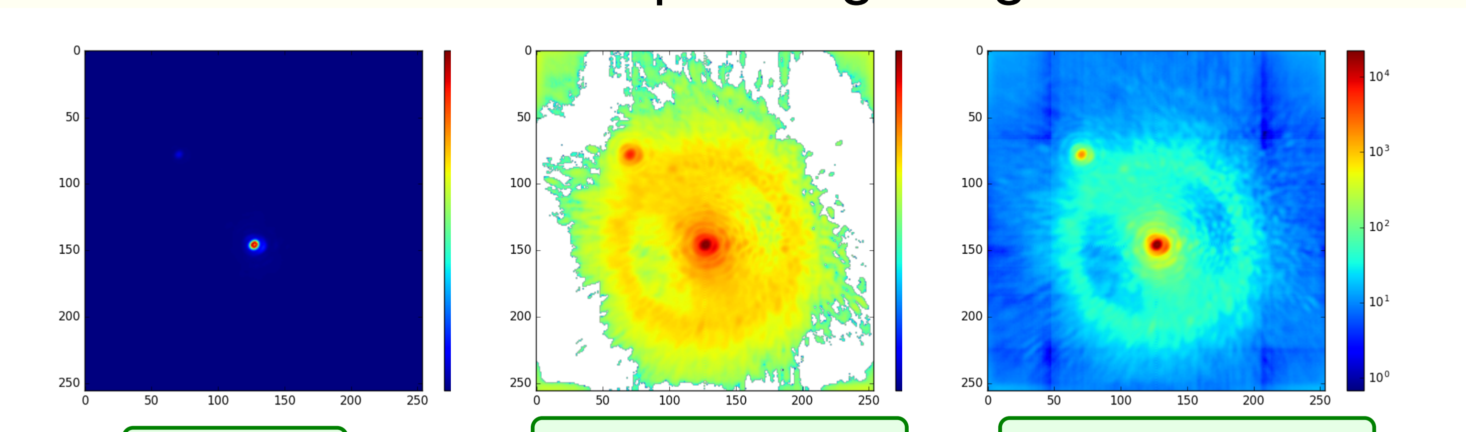
```
In [4]: scidata = hdus[0].data

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")
```

Default linear colormap (In [5])

Logarithmic colormap (In [8])

Shifted up image (In [12])

And here are the corresponding images:



Linear — In [5-6]

Logarithmic 1 — In [8-9]

Logarithmic 2 — In [12-13]

astropy affiliated packages

“Affiliated” packages are astronomical Python packages independently developed whose creators have asked to be included into the AstroPy community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

astropy affiliated packages

“Affiliated” packages are astronomical Python packages independently developed whose creators have asked to be included into the AstroPy community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

Some examples:

`astropy` affiliated packages

“Affiliated” packages are astronomical Python packages independently developed whose creators have asked to be included into the `AstroPy` community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

Some examples:

- `Montage-wrapper`: Python interface to the “Montage Astronomical Image Mosaic Engine”

`astropy` affiliated packages

"Affiliated" packages are astronomical Python packages independently developed whose creators have asked to be included into the `AstroPy` community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

Some examples:

- `Montage-wrapper`: Python interface to the "Montage Astronomical Image Mosaic Engine"
- `Ginga`: generalized FITS file viewer.

`astropy` affiliated packages

“Affiliated” packages are astronomical Python packages independently developed whose creators have asked to be included into the `AstroPy` community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

Some examples:

- `Montage-wrapper`: Python interface to the “Montage Astronomical Image Mosaic Engine”
- `Ginga`: generalized FITS file viewer.
- `APLpy`: the Astronomical Plotting Library in Python (based on `matplotlib`).

`astropy` affiliated packages

“Affiliated” packages are astronomical Python packages independently developed whose creators have asked to be included into the `AstroPy` community, in order to promote the use of the package, enforce interoperability, and conformance to consolidated standards.

Some examples:

- `Montage-wrapper`: Python interface to the “Montage Astronomical Image Mosaic Engine”
- `Ginga`: generalized FITS file viewer.
- `APLpy`: the Astronomical Plotting Library in Python (based on `matplotlib`).
- `Astroquery`: tools for accessing on-line astronomical data.

```
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 0.8600000

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:>}' descript
0.86000001
```