NumPy, matplotlib and SciPy

HPC Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

October 30th, 2014

NumPy

Python Objects

- High-level number objects: integers, floating point
- Containers: lists, dictionaries

NumPy

- Extension package for multi-dimensional arrays
- Closer to hardware $\rightarrow$ efficiency
- Designed for scientific computation

NumPy and Python List

Python List

```
In [1]: import numpy as np
In [2]: list = range(100000)

In [3]: %timeit [i**2 for i in list]
100 loops, best of 3: 6.43 ms per loop

In [4]: array = np.arange(100000)

In [5]: %timeit array**2
1000 loops, best of 3: 97.7 us per loop
```

Why so Slow?

- Dynamic typing requires lots of metadata around variables
- Potentially inefficient memory access
- Interpreted instead of compiled

What can you do?

- Make an object that has a single type and continuous storage
- Implement common functionality into that object to iterate in C

NumPy Features

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([ 2.,  4.,  6.])
```

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0
>>> a*b
array([ 2.,  4.,  6.])
```

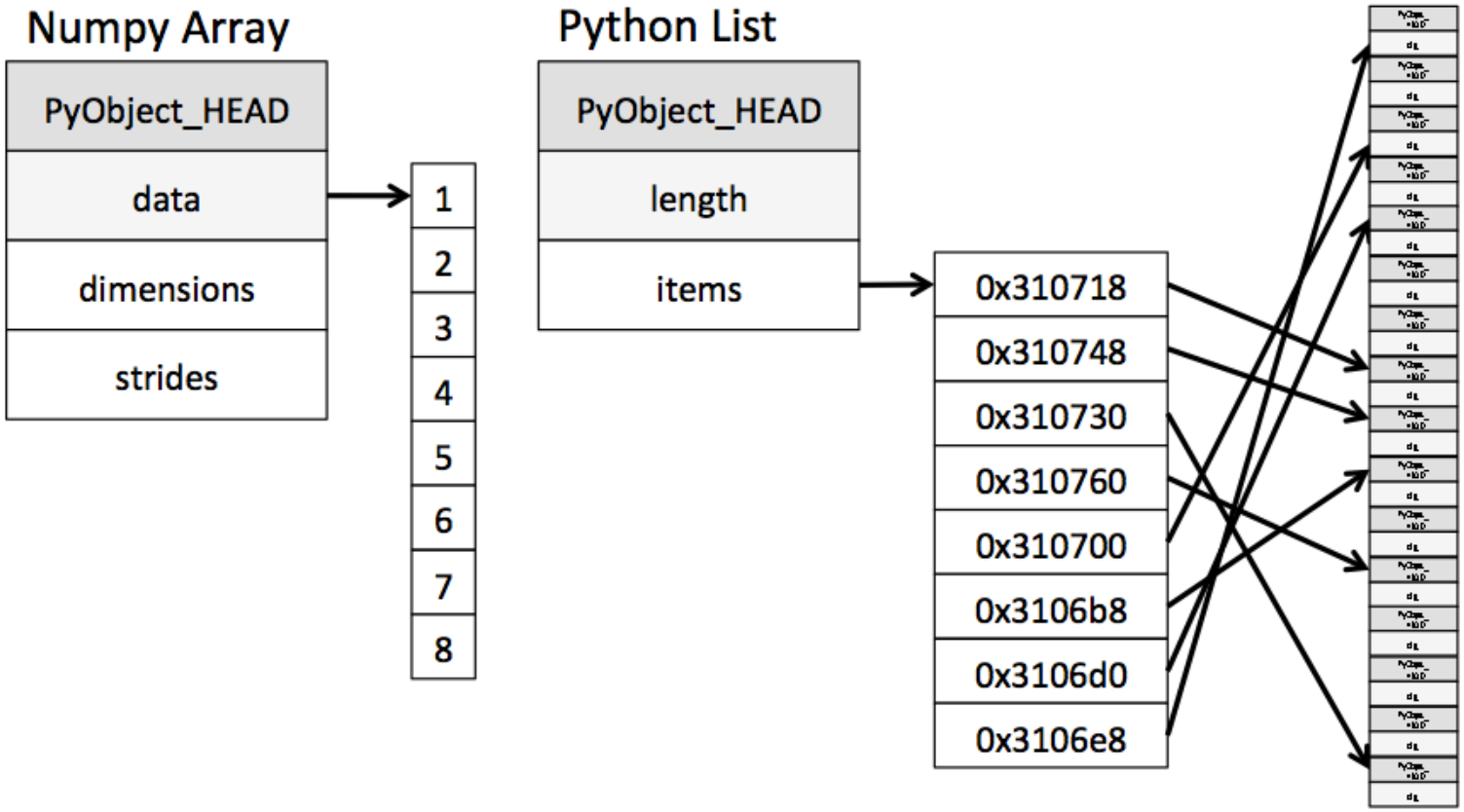
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Array Object

What makes an array so much faster?

- Data layout
  - homogenous: every item takes up the same size block of memory
  - single data-type objects
  - powerful array scalar types
- universal function (ufuncs)
  - function that operates on ndarrays in an element-by-element fashion
  - vectorized wrapper for a function
  - built-in functions are implemented in compiled C code

Data Layout



- Numpy: contiguous data buffer of values
- Python: contiguous buffer of pointers

ufuncs

- function that operates on ndarrays in an element-by-element fashion
- vectorized wrapper for a function
- built-in functions are implemented in compiled C code

Python function - ufunc

```
In [1]: import numpy as np

In [2]: import math

In [3]: arr = np.arange(100000)

In [4]: %timeit[math.sin(i) for i in arr]
10 loops, best of 3: 18.3 ms per loop

In [5]: %timeit np.sin(arr)
100 loops, best of 3: 1.77 ms per loop

In [6]: %timeit[math.sin(i)**2 for i in arr]
10 loops, best of 3: 27.3 ms per loop

In [7]: %timeit np.sin(arr)**2
100 loops, best of 3: 1.83 ms per loop
```

Mathematical functions

How to Create an Array

examples/3_numpy/array.py

```
import numpy as np
a = np.array([2, 3, 12])  # Create from list
a = np.arange(10)       # 0, 1, 2, 3, 4,..., 9
b = np.arange (0,10,2) # start, end (exclusive), step. 0, 2, 4, 6, 8
#By number of points (start, end, num. points)
a = np.linspace(0,1,5) #0, 0.25, 0.50, 0.75, 1.0
a = np.linspace(0,1,5,endpoint=False) #0, 0.2, 0.4, 0.6, 0.8
#Useful arrays
a = np.ones((4,4))
a = np.zeros((3,3))
a = np.diag(np.ones(3))
a = np.eye(3)
#with random numbers
np.random.seed(1111)    #sets the random seed
a = np.random.rand(4)   #uniform in [0,1]
b = np.random.randn(4) #Gaussian
#uninitialized
a = np.empty((3,3))
#resize
a = np.zeros(10)
a = np.resize(a, 20)
```

Data Types

`bool`	`int`	`float`	`complex`
`string`	`int8`	`float16`	`complex64`
	`int16`	`float32`	`complex128`
	`int32`	`float64`	
	`int64`		
	`uint8`		
	`uint16`		
	`uint32`		
	`uint64`		

Data Types

Basic

```
In [1]: import numpy as np

In [2]: a = np.array([1, 2, 3])

In [3]: a.dtype
Out[3]: dtype('int64')

In [4]: b = np.array([1., 2., 3.])

In [5]: b.dtype
Out[5]: dtype('float64')
```

Other

```
In [6]: c = np.array([1, 2, 3], dtype=float)

In [7]: c.dtype
Out[7]: dtype('float64')

In [8]: d = np.array([True, False, True])

In [9]: d.dtype
Out[9]: dtype('bool')

In [10]: e = np.array([1+2j, 3+4j, 5+6*1j])

In [11]: e.dtype
Out[11]: dtype('complex128')

In [12]: f = np.array(['Bonjour', 'Hello', 'Hola'])

In [13]: f.dtype
Out[13]: dtype('S7')  #Strings of max. 7 characters
```

Linear Algebra

Linear Algebra dot Function

```
In [1]: import numpy as np

In [2]: np.dot(np.arange(3), np.arange(3))
Out[2]: 5

In [3]: np.dot(np.arange(9).reshape(3,3), np.arange(3))
Out[3]: array([ 5, 14, 23])

In [4]: np.arange(9).reshape(3,3)
Out[4]:
  array([[0, 1, 2],
         [3, 4, 5],
         [6, 7, 8]])
```

Automatic Offload (AO)

- Feature of Intel Math Kernel Library (MKL)[1]
  - growing list of computationally intensive functions
  - xGEMM and variants; also LU, QR, Cholesky
  - kicks in at appropriate size thresholds (e.g. SGEMM: (M,N,K) = (2048, 2048, 256))
  - Functions with AO
- Essentially no programmer action required
  - more than offload: work division across host and MIC
  - Tips for using MKL on Phi

[1] For more information refer to https://www.tacc.utexas.edu/resources/software/ao

Automatic Offload

Set at least three environment variables before launching your code

```
export MKL_MIC_ENABLE=1
export OMP_NUM_THREADS=16
export MIC_OMP_NUM_THREADS=240
```

- Other environment variables provide additional fine-grained control over host-MIC work division
- MKL documentation
- Intel MKL Automatic Offload enabled functions

Automatic Offload

`examples/3_offload/my_dgemm.py`

Important Variables

```
OMP_NUM_THREADS (1..16)
MKL_MIC_ENABLE (0, 1)
MIC_OMP_NUM_THREADS (1..240)
OFFLOAD_REPORT (0..2)
```

Matplotlib

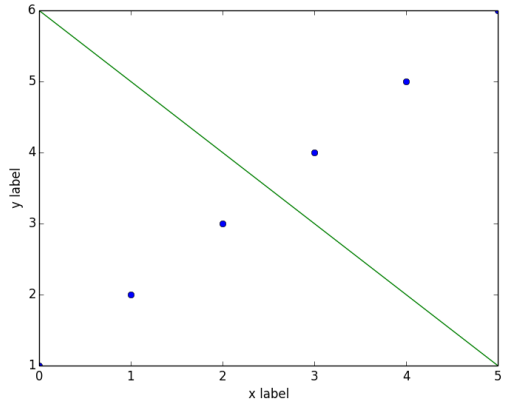
matplotlib is a python 2D/3D plotting library which:

- produces publication quality figures
- provides and interactive environment
- generates plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc
- allows a high level of customization

Matplotlib

Basic | examples/3_matplotlib/basic_1.py

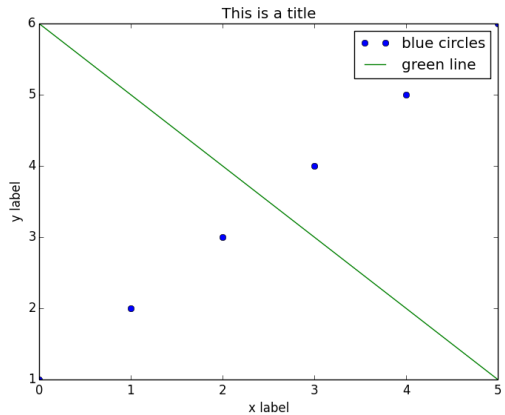
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5,6],'o')
plt.plot([6,5,4,3,2,1])
plt.ylabel('y label')
plt.xlabel('x label')
plt.show()
```



Matplotlib

Basic | examples/3_matplotlib/basic_2.py

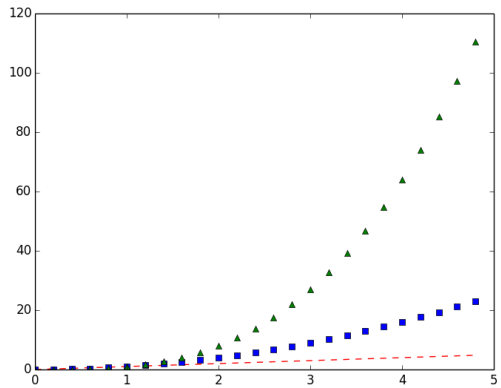
```
import matplotlib.pyplot as plt
#add a title
plt.title('This is a title')
#get the individual plots
plt1, = plt.plot([1,2,3,4,5,6],'o')
plt2, = plt.plot([6,5,4,3,2,1])
#add the legend
plt.legend ([plt1, plt2], ['blue circles', 'green line'])

plt.ylabel('y label')
plt.xlabel('x label')
plt.show()
```



Matplotlib

With NumPy | examples/3_matplotlib/numpy_plot.py

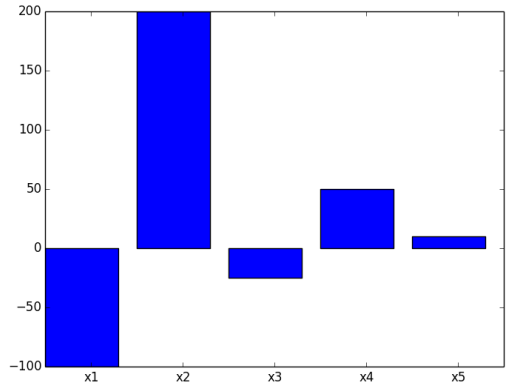
```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()

#replace `plt.show()` by
#plt.savefig('plot.png')
```



Matplotlib

Bars | examples/3_matplotlib/bars.py

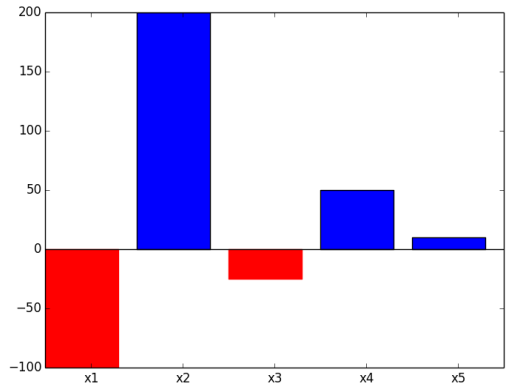
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(5)
yvals = [-100, 200, -25, 50, 10]
bars = plt.bar(x, yvals)  #specifies we are using a bar chart
plt.xticks(x+0.5, ('x1', 'x2', 'x3', 'x4', 'x5'))
plt.show()
```



Matplotlib

Bars | examples/3_matplotlib/bars_color.py

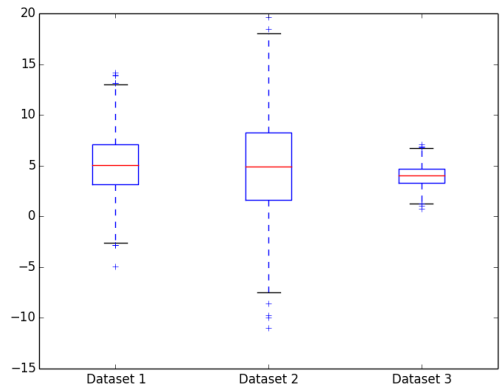
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(5)
yvals = [-100, 200, -25, 50, 10]
bars = plt.bar(x, yvals)

#change color of bars when y<0
for idx, val in enumerate(yvals):
  if (val<0):
    bars[idx].set_color('r')

plt.xticks(x+0.5, ('x1', 'x2', 'x3', 'x4', 'x5'))

#add a horizontal line at y=0
plt.axhline(0, color='black')
plt.show()
```



Matplotlib

Boxplot | examples/3_matplotlib/boxes.py

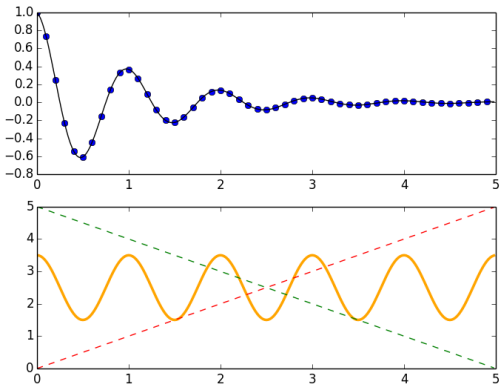
```
import numpy as np
import matplotlib.pyplot as plt
#gaussian distributions
dataset_1 = np.random.normal(5.0, 3.0, 1000)
dataset_2 = np.random.normal(5.0, 5.0, 1000)
dataset_3 = np.random.normal(4.0, 1.0, 1000)
data=[dataset_1, dataset_2, dataset_3]
plt.xticks( np.arange(3), ("Dataset 1", "Dataset 2", "Dataset
      3") )
plt.boxplot(data)
plt.show()
```



Matplotlib

Multiple Figures | examples/3_matplotlib/multiple.py

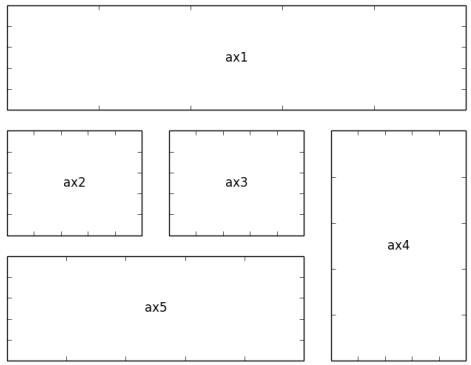
```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
  return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, 2.5+np.cos(2*np.pi*t2), color="orange",
      linewidth=2.5, linestyle="-")
plt.plot(t2, t2, 'r--')
plt.plot(t2, 5.0-t2, 'g--')
plt.show()
```



Matplotlib

Multiple Figures | examples/3_matplotlib/multiple_2.py

```
import matplotlib.pyplot as plt

def make_ticklabels_invisible(fig):
    for i, ax in enumerate(fig.axes):
        ax.text(0.5, 0.5, "ax%d" % (i+1), va="center",
                ha="center")
        for tl in ax.get_xticklabels() + ax.get_yticklabels():
            tl.set_visible(False)


ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
ax2 = plt.subplot2grid((3,3), (1,0))
ax3 = plt.subplot2grid((3,3), (1,1))
ax4 = plt.subplot2grid((3,3), (1, 2), rowspan=2)
ax5 = plt.subplot2grid((3,3), (2, 0), colspan=2)

make_ticklabels_invisible(plt.gcf())
plt.show()
```



Gallery

SciPy

What's SciPy?

- Mathematical algorithms and convenience functions built on NumPy
- Organized into subpackages covering different scientific computing domains
- A data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab

SciPy

- Special functions (`scipy.special`)
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fftpack`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Sparse Eigenvalue Problems with ARPACK
- Statistics (`scipy.stats`)
- Multi-dimensional image processing (`scipy.ndimage`)
- File IO (`scipy.io`)
- Weave (`scipy.weave`)

SciPy.io

- Matlab files
  - `loadmat`
  - `savemat`
- IDL
  - `readsav`
- Wav sound files
- Arff files
- NetCDF
  - `createDimension`
  - `createVariable`

examples/3_scipy/netcdf.py

```
import numpy as np
from scipy.io import netcdf

#Create a new netCDF file
f = netcdf.netcdf_file('simple.nc', 'w')
f.history = 'Created for a test'
f.createDimension('time', 10)
time = f.createVariable('time', 'i',
    ('time',))
time[:] = np.arange(10)
time.units = 'days since 2008-01-01'
f.close()

#Now, open the file and read the content
fread = netcdf.netcdf_file('simple.nc', 'r')
print(fread.history)
time = fread.variables['time']
print(time.units)
print(time.shape)
print(time[-1])
fread.close()
```

SciPy.Stats

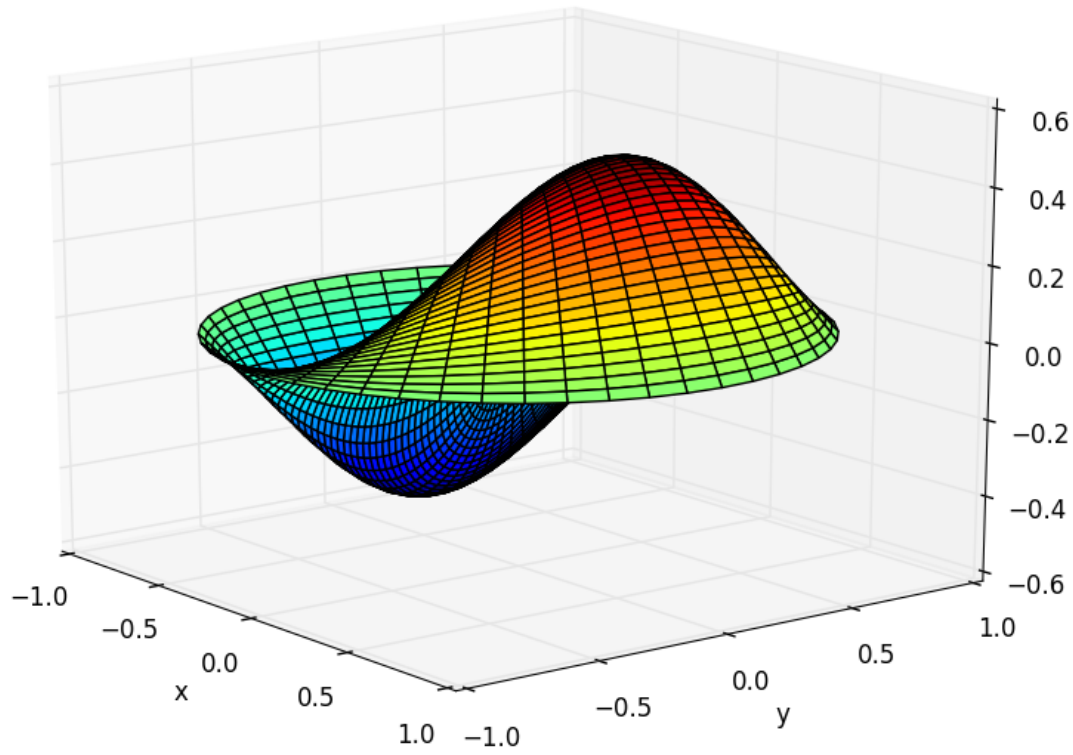
- Continuous distributions
- Discrete distributions
- Statistical functions
- Masked statistics functions

examples/3_scipy/stats.py

```
import scipy as sp
import numpy as np
s = sp.rand(50)

#Show the mean, variance, std. deviation and the median
mean = sp.mean(s)
std = sp.std(s)
print("Mean : {0:8.6f}".format(mean))
print("Variance : {0:8.6f}".format(sp.var(s)))
print("Std. deviation : {0:8.6f}".format(std))
print("Median : {0:8.6f}".format(sp.median(s)))

from scipy import stats
x = sp.linspace(-3*std, 3*std, 50)
#survival function (probability that the variate has a value
      greater than the given value
y = stats.norm.sf(x, loc=mean, scale=std)

import matplotlib.pyplot as plt
plt.plot(x,y, color="black")
plt.xlabel("Variate")
plt.ylabel("Probability")
plt.title("SF for Gaussian of mean = {0} & std. deviation =
      {1}".format(mean, std))
plt.show()
```

SciPy. Bessel Functions

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$



SciPy. Bessel Functions

examples/3_scipy/bessel.py

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from scipy import *
from scipy.special import jn, jn_zeros

def drumhead_height(n, k, distance, angle, t):
  nth_zero = jn_zeros(n, k)
  return cos(t)*cos(n*angle)*jn(n, distance*nth_zero)

theta = r_[0:2*pi:50j]
radius = r_[0:1:50j]
x = array([r*cos(theta) for r in radius])
y = array([r*sin(theta) for r in radius])
z = array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])

fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

License