3. NumPy

Author : Dongsoo S. Kim (dskim@iupui.edu (mailto:dskim@iupui.edu))
Date : Feb. 8, 2016, Sep.5, 2019

In [1]:
```
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
```

In [2]:
```
x = np.array([6, 7, 8])         # create an np array
x
```

Out[2]:
```
array([6, 7, 8])
```

In [3]:
```
# printing an NumPy object
print (x)
```

```
[6 7 8]
```

In [4]:
```
# multi-dimensional array
y = np.array([[0,1,2],[3,4,5],[6,7,8]])
print ('y =', y)
print ('size: ', y.size)             # number of elements
print ('type: ', y.dtype)            # data type of elements
print ('ndim: ', y.ndim)             # dimension of the array
print ('shape:', y.shape)            # shape of the array
```

```
y = [[0 1 2]
 [3 4 5]
 [6 7 8]]
size:  9
type:  int32
ndim:  2
shape: (3, 3)
```

In [5]:
```
# various types
print (np.array([ [1,2], [3,4] ], dtype=int))
print (np.array([ [1,2], [3,4] ], dtype=float))
print (np.array([ [1,2], [3,4] ], dtype=complex))
x = 1+5j
print (x)
```

```
[[1 2]
 [3 4]]
[[1. 2.]
 [3. 4.]]
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]]
(1+5j)
```

In [6]:
```python
# numerous array functions
print ('identity:\n', np.identity(3))
```

```
identity:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [7]:
```python
print ('zeros:\n', np.zeros((3,4)))
```

```
zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [8]:
```python
print ('ones:\n', np.ones( (2,3,4), dtype=np.int16 ) )
```

```
ones:
 [[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
```

vectorized computing

In [9]:
```python
# simple array arithmetic
a = np.array([ [1,2], [3,4] ])
b = np.array([ [5,6], [7,8] ], dtype=complex)

print ('a + b\n', a + b)
print ('a - b\n', a - b)
print ('a * b\n', a * b)
print ('a / b\n', a / b)

```

```
a + b
 [[ 6.+0.j  8.+0.j]
 [10.+0.j 12.+0.j]]
a - b
 [[-4.+0.j -4.+0.j]
 [-4.+0.j -4.+0.j]]
a * b
 [[ 5.+0.j 12.+0.j]
 [21.+0.j 32.+0.j]]
a / b
 [[0.2       +0.j 0.33333333+0.j]
 [0.42857143+0.j 0.5       +0.j]]
```

In [10]:
```
print ('a ** 2\n', a ** 2)
print ('10 * sin(a)\n', 10 * np.sin(a))
```

```
a ** 2
 [[ 1  4]
 [ 9 16]]
10 * sin(a)
 [[ 8.41470985  9.09297427]
 [ 1.41120008 -7.56802495]]
```

In [11]:
```
# numerous aux functions
a = np.random.random((2,5))
print ('a =\n', a)
print ('a.sum =', a.sum())
print ('a.min =', a.min())
print ('a.max =', a.max())
```

```
a =
 [[0.64458286 0.15065244 0.25598586 0.02459517 0.2319529 ]
 [0.82608774 0.43347058 0.67035519 0.92331687 0.15584371]]
a.sum = 4.316843342203911
a.min = 0.02459517452385529
a.max = 0.92331687346472
```

powerful indexing

In [12]:
```
# sub-array in row
a = np.random.random((5,3))
print ('a =\n', a)
print ('a[1:4] =\n', a[1:4])

```

```
a =
 [[0.69963029 0.60533353 0.45818071]
 [0.85565126 0.18453609 0.9522594 ]
 [0.16555263 0.84277372 0.94436354]
 [0.88798277 0.29993386 0.95844613]
 [0.70249742 0.16514484 0.42562092]]
a[1:4] =
 [[0.85565126 0.18453609 0.9522594 ]
 [0.16555263 0.84277372 0.94436354]
 [0.88798277 0.29993386 0.95844613]]
```

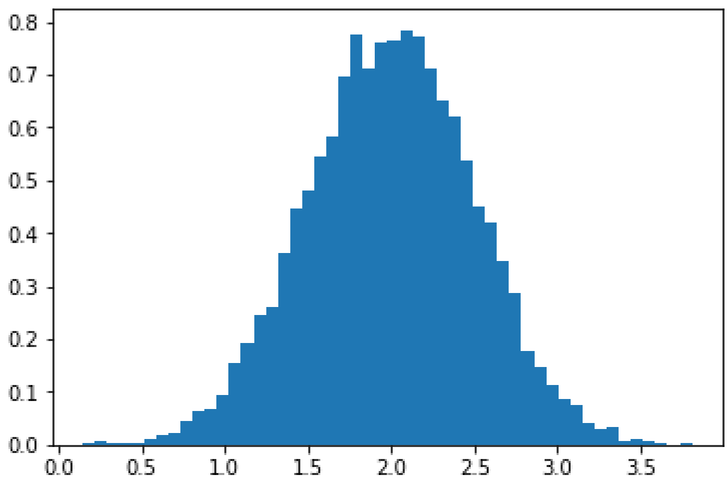
In [13]:
```
# sub-array in col
print ('a[:,1:] =\n', a[:,1:])
```

```
a[:,1:] =
 [[0.60533353 0.45818071]
 [0.18453609 0.9522594 ]
 [0.84277372 0.94436354]
 [0.29993386 0.95844613]
 [0.16514484 0.42562092]]
```

numpy has its own powerful random library

In [14]:
```
# work well with graphic library
mu, sigma = 2, 0.5
v = np.random.normal(mu,sigma,10000)
```

In [15]:
```
# Plot a normalized histogram with 50 bins
plt.hist(v, bins=50, density=1)     # Plot a normalized histogram with 50 bi
plt.show()
```



Numpy vs. MatLab

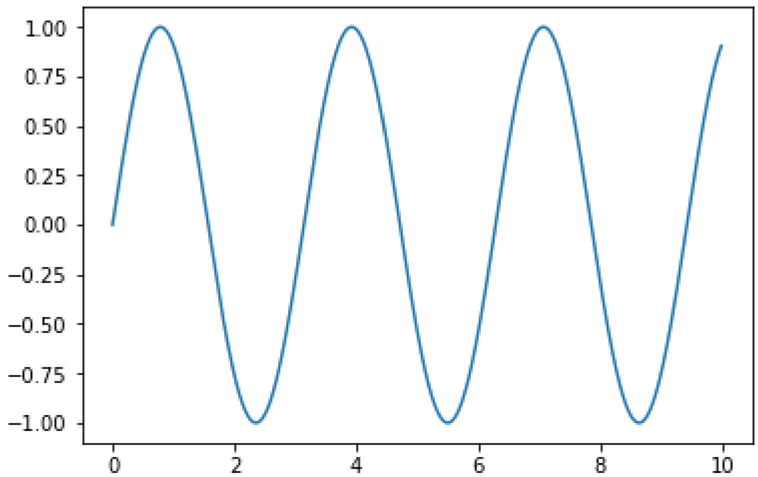
I found the following from a MatLab tutorial

```
x = [0:.01:10];
y = sin(2 * x);
plot(x,y)
```

Its equivalent NumPy code will be as below:

In [16]:
```
x = np.arange(0,10, 0.01)
y = np.sin(2 * x)
plt.plot(x,y)
```

Out[16]: [<matplotlib.lines.Line2D at 0x22d292c0ef0>]

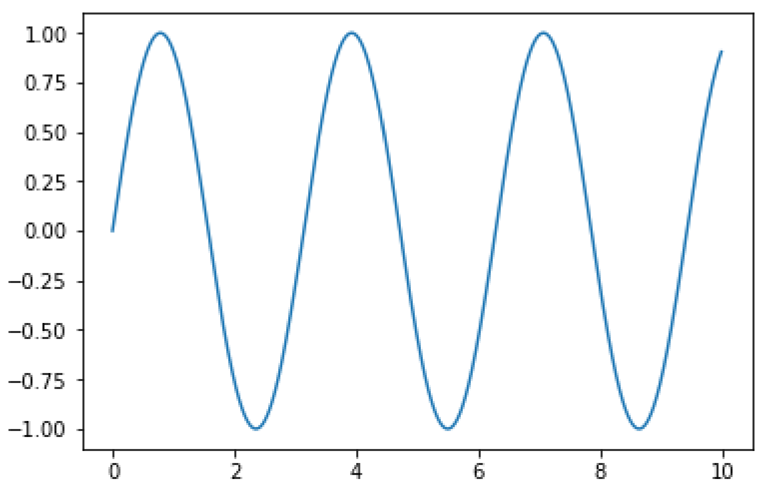


even without the namespace ...

In [17]:
```
from numpy import *
from matplotlib.pylab import *

x = arange(0,10, 0.01)
y = sin(2 * x)
plot(x,y)
```

Out[17]: [<matplotlib.lines.Line2D at 0x22d293302e8>]



Linear algebra (numpy.linalg)

[https://docs.scipy.org/doc/numpy/reference/routines.linalg.html (https://docs.scipy.org/doc/numpy/reference/routines.linalg.html)](https://docs.scipy.org/doc/numpy/reference/routines.linalg.html)

In [18]:

```
import numpy as np
r = np.array([
    [0, 1, 1, 1],
    [1, 0, 0, 1],
    [1, 0, 0, 1],
    [1, 1, 1, 0]])
w,v = np.linalg.eig(r)
print ("w =", w)
print ("v =", v)
```

```
w = [ 2.56155281e+00 -1.00000000e+00 -1.56155281e+00 -5.49681202e-18]
v = [[-5.57345410e-01  7.07106781e-01 -4.35162146e-01  1.98492213e-17]
 [-4.35162146e-01 -3.97111101e-16  5.57345410e-01 -7.07106781e-01]
 [-4.35162146e-01 -7.79219809e-17  5.57345410e-01  7.07106781e-01]
 [-5.57345410e-01 -7.07106781e-01 -4.35162146e-01 -6.87807092e-17]]
```

In [ ]:

```

```