Package

Online references

1. http://cs231n.github.io/python-numpy-tutorial/
2. https://www.tutorialspoint.com/numpy/index.htm

```
1 import numpy as np
```

Matrix operations

Matrix Creation

```
1 x = np.array([1,2,3])
2
3 print (type(x))
4
5 print (x)
6
7 print (x.shape)
```

```
<class 'numpy.ndarray'>
[1 2 3]
(3,)
```

```
1 x = np.array([ [1,2,3], [4,5,6] ])
2
3 print (type(x))
4
5 print (x)
6
7 print (x.shape)
```

```
<class 'numpy.ndarray'>
[[1 2 3]
 [4 5 6]]
(2, 3)
```

```
1 x = np.zeros( (2,3) )
2
3 print (x)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
1 x = np.ones( (2,3) )
2
```

```
_
3 print (x)
```

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

```
1 x = np.full( (2,3), 7)
2
3 print (x)
```

```
[[7 7 7]
 [7 7 7]]
```

```
1 x = np.full( (2,3), 7)
2
3 x[0][1] = -900
4 x[1][2] = 8000
5
6 print (x)
```

```
[[   7 -900    7]
 [   7    7 8000]]
```

```
1 x = np.eye(4)
2
3 print (x)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
1 x = np.random.random( (2,3) )
2
3 print (x)
```

```
[[0.51736091 0.82956735 0.10428439]
 [0.19708099 0.76812515 0.91431426]]
```

Matrix Transpose

```
1 x = np.random.random( (100, 48) )
2
3 print (x.shape)
4
5 xt = x.T
6
7 print (xt.shape)
8
9 print (x.T.T.T.T.shape)
```

```
(100, 48)
(48, 100)
(100, 48)
```

Matrix Addition

```
x1 = np.random.random( (100,48) )
x2 = np.random.random( (100,48) )

x3 = x1 + x2

print (x3.shape)
```

```
(100, 48)
```

Matrix Multiplication

```
x1 = np.random.random( (100,48) )
x2 = np.random.random( (48,607) )

x3 = np.matmul(x1,x2)

print (x3.shape)
```

```
(100, 607)
```

Elementwise Matrix Operations

```
x1 = np.random.random( (100,48) )
x2 = np.random.random( (100,48) )

x3 = np.multiply(x1,x2)

print (x3.shape)
```

```
(100, 48)
```

Matrix inversion

```
x = np.random.random( (10,10) )

xinv = np.linalg.inv(x)

print (xinv.shape)

test = np.matmul(x,xinv)
```

```
 8
 9 i10 = np.eye(10)
10
11 np.allclose(test,i10)
```

```
(10, 10)
True
```

```
1 x = np.random.random( (10,10) )
2
3 xdet = np.linalg.det(x)
4
5 print (xdet)
```

```
0.013130577693879368
```

Eigen Value

```
1 x = np.random.random( (10,10) )
2
3 w, v = np.linalg.eig(x)
4
5 print (w.shape)
6
7 print (v.shape)
```

```
(10,)
(10, 10)
```

```
1 x = np.random.random( (10,10) )
2
3 w, v = np.linalg.eig(x)
4
5 w_real = [wval for wval in w if not np.iscomplex(wval)]
6
7 print (w_real)
```

```
[(4.937632279501388+0j), (0.8147877893793526+0j), (0.6506661879985829+0j), (-0.804708
```

Matrix Factorization

```
1 x = np.random.random( (20,30) )
2
3 q,r = np.linalg.qr(x)
4
5 print (q.shape, r.shape)
6
7 xrcon= np.matmul(q,r)
8
9 np.allclose(x,xrcon)
```

```
(20, 20) (20, 30)
True
```

```
x = np.array([1,2,3])

xdiag = np.diag(x)

print (xdiag)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

```
x = np.random.random( (2,3) )

u,s,vh = np.linalg.svd(x)

print (u.shape, s.shape, vh.shape)

print (u)

print (s)

print (vh)
```

```
(2, 2) (2,) (3, 3)
[[-0.61381581 -0.78944927]
 [-0.78944927  0.61381581]]
[1.25999309 0.15729898]
[[-0.182759   -0.58738909 -0.78839914]
 [-0.22727394  0.80542572 -0.54739015]
 [-0.95652794 -0.0791421   0.28069703]]
```

Optimization

Package

```
from scipy.optimize import minimize
```

Line fitting

```
x = np.linspace(0,10,100)
```

```
y = 31.7*x + 432.693 #actual coefficients, keep them secret!
```

```
def f_error(w) :
  errval = np.sum( (w[0]*x + w[1] - y)**2 )
```

```
  return errval
```

```
w = np.array([0,0])

res = minimize(f_error,w)

print (res.x) #check if recovered the coefficients correctly?

np.allclose(res.x,np.array([31.7,432.693]))
```

```
[ 31.69999997 432.69300012]
True
```

Quadratic fitting

```
x = np.linspace(0,10,100)
```

```
y = -89.7*x**2 + 45.21*x + 9000.3 #keep these coefficients secret!
```

```
def f_error2(w) :
  errval = np.sum( (w[0]*x**2 + w[1]*x + w[2] - y)**2 )
  return errval
```

```
w = np.array([0,0,0])

res = minimize(f_error2,w)

print (res.x)

np.allclose(res.x,np.array([-89.7,45.21, 9000.3]))
```

```
[ -89.70000022   45.21000217 9000.29999649]
True
```

System of linear equations

Analytical approach

```
A = np.array([
    [4,3],
    [1,-2],
    [3,5]
])

b = np.array([
    [7],
    [-1],
    [8]
```

```
    [8]
])

print (A.shape, b.shape)
```

```
(3, 2) (3, 1)
```

```
a1 = np.matmul(A.T,A)

print (a1)
```

```
[[26 25]
 [25 38]]
```

```
a2 = np.linalg.inv(a1)
```

```
b1 = np.matmul(A.T,b)
```

```
x = np.matmul(a2,b1)
```

```
print (x)
```

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

```
b_pred = np.matmul(A,x)

print (b_pred)
print (b)
```

```
[[ 7.]
 [-1.]
 [ 8.]]
[[ 7]
 [-1]
 [ 8]]
```

Optimization approach

```
def f_error3(w) :
  w = w.reshape(2,1)
  errval = np.sum( (np.matmul(A,w) - b)**2 )
  return errval
```

```
w = np.array([100,200])
```

```
res = minimize(f_error3,w)

print (res.x)
```

```
[1.         0.99999999]
```

```
1 print (np.matmul(A,res.x))
```

```
[ 6.99999998 -0.99999999  7.99999996]
```

Trigonometric functions

```
1 x = np.linspace(-np.pi/2,np.pi/2,10000)
```

```
1 y = np.sin(x)
```

```
1 x_rcon = np.arcsin(y)
```

```
1 np.allclose(x_rcon,x)
```

```
True
```

```
1 z = np.sin(x)**2 + np.cos(x)**2
2 print (z)
```

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

```
1 x = 45
2
3 xrad = np.radians(x)
4
5 print (xrad)
6
7 xdeg = np.degrees(xrad)
8
9 print (xdeg)
```

```
0.7853981633974483
45.0
```

```
1 x = np.random.random(10)
2
3 print (np.linalg.norm(x))
4
5 print (np.linalg.norm(x,3))
```

```
1.5653332298324956
1.1905958585618095
```

Statistics

Mean

```
1 x = np.random.rand(100)
2
3 print (x.shape)
```

```
(100,)
```

```
1 x_mean = np.mean(x)
2
3 print (x_mean)
```

```
0.5029724334586698
```

```
1 x = np.random.rand(100,4)
2
3 print (x.shape)
```

```
(100, 4)
```

```
1 x_mean = np.mean(x, axis=0)
2
3 print (x_mean.shape)
```

```
(4,)
```

```
1 x_mean = np.mean(x, axis=1)
2
3 print (x_mean.shape)
```

```
(100,)
```

Variance

```
1 x = np.random.rand(100,4)
```

```
1 x_var = np.var(x)
2
3 print (x_var)
```

```
0.07944438542174038
```

```
1 x_var = np.var(x, axis=0)
2
3 print (x_var.shape)
```

```
(4,)
```

```
1 x_var = np.var(x,axis=1)
2
3 print (x_var.shape)
```

```
(100,)
```

Median

```
x = np.array([1,2,2,3,3,3,3,3,4,5,5,6,7])
```

```
np.random.shuffle(x)

print (x)
```

```
[3 7 2 3 3 4 3 5 1 3 6 5 2]
```

```
x_median = np.median(x)

print (x_median)
```

```
3.0
```

```
#REF - https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html#numpy.med

a = np.array([[10, 7, 4], [3, 2, 1]])

print (a)


print (np.median(a))

print (np.median(a, axis=0))

print (np.median(a, axis=1))

```

```
[[10  7  4]
 [ 3  2  1]]
3.5
[6.5 4.5 2.5]
[7. 2.]
```

Mode

```
x = np.array([1,2,2,3,3,3,3,3,4,5,5,6,7,6])
```

```
from scipy.stats import mode
```

```
a,b = mode(x)

print (a)

```

```
print (b)
```

```
[3]
[5]
```

```
np.random.shuffle(x)

x = x.reshape(7,2)

print (x.shape)

print (x)
```

```
(7, 2)
[[5 7]
 [6 3]
 [6 3]
 [3 5]
 [3 4]
 [1 2]
 [3 2]]
```

```
print (mode(x,axis=0))
```

```
ModeResult(mode=array([[3, 2]]), count=array([[3, 2]]))
```

Tests

```
from scipy.stats import ttest_ind
```

```
x1 = np.random.random(100)

x2 = np.random.random(200)
```

```
print (ttest_ind(x1,x2))
```

```
Ttest_indResult(statistic=0.23861786104131985, pvalue=0.8115659207763958)
```