

```
# check the number of dimensions
my_3D.ndim
'Output': 3
# get the shape of the 3-D array - this example has 2 pages, 2 rows and 3
columns: (p, r, c)
my_3D.shape
'Output': (2, 2, 3)
```

We can also create 3-D arrays with methods such as **ones**, **zeros**, **full**, and **empty** by passing the configuration for [page, row, columns] into the **shape** parameter of the methods. For example:

```
# create a 2-page, 3x3 array of ones
np.ones([2,3,3])
'Output':
array([[[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]],
       [[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])
# create a 2-page, 3x3 array of zeros
np.zeros([2,3,3])
'Output':
array([[[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]],
       [[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]])
```

Indexing/Slicing of Matrices

Let's see some examples of indexing and slicing 2-D arrays. The concept extends nicely from doing the same with 1-D arrays.

```
# create a 3x3 array contain random normal numbers
my_3D = np.random.randn(3,3)
'Output':
array([[ 0.99709882, -0.41960273,  0.12544161],
       [-0.21474247,  0.99555079,  0.62395035],
       [-0.32453132,  0.3119651 , -0.35781825]])
# select a particular cell (or element) from a 2-D array.
my_3D[1,1]    # In this case, the cell at the 2nd row and column
'Output': 0.99555079000000002
# slice the last 3 columns
my_3D[:,1:3]
'Output':
array([[ -0.41960273,  0.12544161],
       [ 0.99555079,  0.62395035],
       [ 0.3119651 , -0.35781825]])
# slice the first 2 rows and columns
my_3D[0:2, 0:2]
'Output':
array([[ 0.99709882, -0.41960273],
       [-0.21474247,  0.99555079]])
```

Matrix Operations: Linear Algebra

Linear algebra is a convenient and powerful system for manipulating a set of data features and is one of the strong points of NumPy. Linear algebra is a crucial component of machine learning and deep learning research and implementation of learning algorithms. NumPy has vectorized routines for various matrix operations. Let's go through a few of them.

Matrix Multiplication (Dot Product)

First let's create random integers using the method **np.random.randint(low, high=None, size=None,)** which returns random integers from low (inclusive) to high (exclusive).