MATLAB/Octave matrices vs. Python NumPy arrays

Note: in order to use the “np" shortcut notation for the numpy package, make sure you import the NumPy package as follows

```
>>> import numpy as np
```

	MATLAB/Octave matrices & vectors	NumPy arrays
Matrices (here: 3x3 matrix)	octave:1> A = [1 2 3; 4 5 6; 7 8 9] A = 1 2 3 4 5 6 7 8 9	>>> A = np.array([[1,2,3], [4,5,6], [7,8,9]]) >>> A array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
Access rows (here: first row)	octave:10> A(1,:) ans = 1 2 3	>>> A[0,] array([1, 2, 3])
Access columns (here: first column)	octave:11> A(:,1) ans = 1 4 7	>>> A[:,0] array([1, 4, 7]) >>> A[:,[0]] array([[1], [4], [7]])
Access elements (here: first element)	octave:8> A(1,1) ans = 1	>>> A[0,0] 1
1-D column vector	octave:3> a = [1; 2; 3] a = 1 2 3	>>> a = np.array([[1],[2],[3]]) >>> a array([[1], [2], [3]])
1-D row vector	octave:4> b = [1 2 3] b = 1 2 3	>>> b = np.array([1,2,3]) >>> b array([1, 2, 3])

	MATLAB/Octave matrices & vectors	NumPy arrays
row to column vector	octave:49> b = [1 2 3]' b = 1 2 3	>>> b = np.array([1, 2, 3]) >>> b = b[np.newaxis].T >>> b array([[1], [2], [3]])
column to row vector	octave:55> b = b' b = 1 2 3	>>> b.T array([[1, 2, 3]])
stacking vectors and matrices	octave:60> c = [a' b'] c = 1 4 2 5 3 6 octave:58> c = [a ; b] c = 1 2 3 4 5 6	>>> a = np.array([1,2,3]) >>> b = np.array([4,5,6]) >>> np.column_stack([a,b]) array([[1, 4], [2, 5], [3, 6]]) >>> np.row_stack([a,b]) array([[1, 2, 3], [4, 5, 6]])
Random m x n matrix	octave:6> rand(3,2) ans = 0.21977 0.10220 0.38959 0.69911 0.15624 0.65637	>>> np.random.rand(3,2) array([[0.29347865, 0.17920462], [0.51615758, 0.64593471], [0.01067605, 0.09692771]])
Zero-matrix, m x n	octave:16> zeros(3,2) ans = 0 0 0 0 0 0	>>> np.zeros((3,2)) array([[0., 0.], [0., 0.], [0., 0.]])

	MATLAB/Octave matrices & vectors	NumPy arrays
m x n matrix of ones	octave:36> ones(3,2) ans = 1 1 1 1 1 1	>>> np.ones([3,2]) array([[1., 1.], [1., 1.], [1., 1.]])
Identity matrix	octave:39> eye(3) ans = Diagonal Matrix 1 0 0 0 1 0 0 0 1	>>> np.identity(3) array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
Matrix diagonal (left-upper corner to right lower)	octave:40> diag(A) ans = 1 5 9	>>> np.diagonal(A) array([1, 5, 9]) >>> np.diagonal([A]) array([[1], [2], [3]])
Diagonal matrix from a column vector	octave:42> diag(a) ans = Diagonal Matrix 1 0 0 0 2 0 0 0 3	>>> np.diag(a[:,0]) array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
Matrix-scalar multiplication (*), subtraction (-), addition (+), division (/)	octave:18> A * 2 ans = 2 4 6 8 10 12 14 16 18	>>> A * 2 array([[2, 4, 6], [8, 10, 12], [14, 16, 18]])

	MATLAB/Octave matrices & vectors	NumPy arrays
Matrix element-wise power	octave:23> A.^2 ans = 1 4 9 16 25 36 49 64 81	>>> np.power(A,2) array([[1, 4, 9], [16, 25, 36], [49, 64, 81]])
Element-wise matrix multiplication	octave:32> A .* A ans = 1 4 9 16 25 36 49 64 81	>>> A * A array([[1, 4, 9], [16, 25, 36], [49, 64, 81]])
Matrix-multiplication	octave:31> A * A ans = 30 36 42 66 81 96 102 126 150	>>> np.dot(A,A) array([[30, 36, 42], [66, 81, 96], [102, 126, 150]])
Matrix transpose	octave:24> A' ans = 1 4 7 2 5 8 3 6 9	>>> A.T array([[1, 4, 7], [2, 5, 8], [3, 6, 9]])

	MATLAB/Octave matrices & vectors	NumPy arrays
Covariance Matrix of 3 random variables	`octave:36> x1 = [4.0000 4.2000 3.9000 4.3000 4.1000]'` `x2 = [2.0000 2.1000 2.0000 2.1000 2.2000]'` `x3 = [0.60000 0.59000 0.58000 0.62000 0.63000]'` `octave:44> cov( [x1,x2,x3] )` `ans =` `2.5000e-02   7.5000e-03 1.7500e-03` `7.5000e-03   7.0000e-03 1.3500e-03` `1.7500e-03   1.3500e-03 4.3000e-04`	`>>> x1 = np.array([ 4. ,  4.2,  3.9,  4.3,  4.1])` `>>> x2 = np.array([ 2. ,  2.1,  2. ,  2.1,  2.2])` `>>> x3 = np.array([ 0.6 ,  0.59,  0.58,  0.62,  0.63])` `>>> X` `array([[ 4.  ,  4.2 ,  3.9 ,  4.3 ,  4.1 ],` `       [ 2.  ,  2.1 ,  2.  ,  2.1 ,  2.2 ],` `       [ 0.6 ,  0.59,  0.58,  0.62,  0.63]])` `>>> np.cov(X)` `array([[ 0.025  ,  0.0075 ,  0.00175],` `       [ 0.0075 ,  0.007  ,  0.00135],` `       [ 0.00175,  0.00135,  0.00043]])`
Eigenvectors and Eigenvalues	`A =` `3   1` `1   3` `octave:77> [eig_vec,eig_val] = eig(A)` `eig_vec =` `-0.70711   0.70711` `0.70711   0.70711` `eig_val =` `Diagonal Matrix` `2   0` `0   4`	`>>> A = np.array([[3, 1], [1, 3]])` `>>> A` `array([[3, 1],` `       [1, 3]])` `>>> eig_val, eig_vec = np.linalg.eig(A)` `>>> eig_val` `array([ 4.,  2.])` `>>> eig_vec` `array([[ 0.70710678, -0.70710678],` `       [ 0.70710678,  0.70710678]])`

http://wiki.scipy.org/NumPy_for_Matlab_Users

array' or 'matrix'? Which should I use?

Short answer

Use arrays.

- They are the standard vector/matrix/tensor type of numpy. Many numpy function return arrays, not matrices.
- There is a clear distinction between element-wise operations and linear algebra operations.
- You can have standard vectors or row/column vectors if you like.

The only disadvantage of using the array type is that you will have to use `dot` instead of `*` to multiply (reduce) two tensors (scalar product, matrix vector multiplication etc.)