

NumPy for Matlab Users





1. Introduction
2. Some Key Differences
3. Table of Rough Equivalents
4. Notes
5. Links


Introduction

MATLAB® and NumPy/SciPy have a lot in common. But there are many differences. NumPy and SciPy were created to do numerical and scientific computing in the most natural way with Python, not to be MATLAB® clones. This page is intended to be a place to collect wisdom about the differences, mostly for the purpose of helping proficient MATLAB® users become proficient NumPy and SciPy users.

Some Key Differences

<p>In MATLAB®, the basic data type is a multidimensional array of double precision floating point numbers. Most expressions take such arrays and return such arrays. Operations on the 2-D instances of these arrays are designed to act more or less like matrix operations in linear algebra.</p>	<p>In NumPy the basic type is a multidimensional <code>array</code>. Operations on these arrays in all dimensionalities including 2D are elementwise operations. However, there is a special <code>matrix</code> type for doing linear algebra, which is just a subclass of the <code>array</code> class. Operations on matrix-class arrays are linear algebra operations.</p>
<p>In MATLAB®, arrays are pass by value. Slice operations copy parts of the array.</p>	<p>In NumPy arrays are pass by reference. Slice operations are views into an array.</p>
<p>MATLAB®'s scripting language was created for doing linear algebra. The syntax for basic matrix operations is nice and clean, but the API for adding GUIs</p>	<p>NumPy is based on Python, which was designed from the outset to be an excellent general-purpose programming language. While Matlab's syntax for some array manipulations is more compact than NumPy's, NumPy (by virtue of being an</p>

<p>and making full-fledged applications is more or less an afterthought.</p>	<p>add-on to Python) can do many things that Matlab just cannot, for instance subclassing the main array type to do both array and matrix math cleanly.</p>
<p>MATLAB® has an active community and there is lots of  code available for free. But the vitality of the community is limited by MATLAB®'s cost; your MATLAB® programs can be run by only a few. In contrast, Python programs can be redistributed and used freely.</p>	<p>NumPy/SciPy also has an active community, based right here on this web site! It is smaller, but it is growing very quickly. See Topical Software for a listing of free add-on application software, Mailing Lists for discussions, and the rest of this web site for additional community contributions. We encourage your participation!</p>
<p>MATLAB® has an extensive set of optional, domain-specific add-ons ('toolboxes') available for purchase, such as for signal processing, optimization, control systems, and the whole SimuLink® system for graphically creating dynamical system models.</p>	<p>There's no direct equivalent of this in the free software world currently, in terms of range and depth of the add-ons. However the list in Topical Software certainly shows a growing trend in that direction.</p>
<p>MATLAB® has a sophisticated 2-d and 3-d plotting system, with user interface widgets.</p>	<p>Addon software can be used with Numpy to make comparable plots to MATLAB®.  Matplotlib is a mature 2-d plotting library that emulates the MATLAB® interface.  PyQwt allows more robust and faster user interfaces than MATLAB®. And MayaVi allows 3-d plots via VTK. See the Topical Software page for more options, links, and details.</p>
<p>MATLAB® provides a full development environment with command interaction window, integrated editor, and debugger.</p>	<p>Numpy does not have one standard IDE. However, the  IPython environment provides a sophisticated command prompt with full completion, help, and debugging support, and interfaces with the Matplotlib library for plotting and the Emacs/XEmacs editors.</p>
<p>MATLAB® itself costs thousands of</p>	

dollars if you're not a student. The source code to the main package is not available to ordinary users. You can neither isolate nor fix bugs and performance issues yourself, nor can you directly influence the direction of future development. (If you are really set on Matlab-like syntax, however, there is  Octave, which is free.)

NumPy and SciPy are free (both beer and speech), whoever you are.

Table of Rough Equivalents

The table below gives rough equivalents for some common MATLAB® expressions. **These are not exact equivalents**, but rather should be taken as hints to get you going in the right direction. For more detail read the built-in documentation on the NumPy functions.

Some care is necessary when writing functions that take arrays or matrices as arguments --- if you are expecting an `array` and are given a `matrix`, or vice versa, then `*` (multiplication) will give you unexpected results. You can convert back and forth between arrays and matrices using

- **`asarray`**: always returns an object of type `array`
- **`asmatrix`**: always returns an object of type `matrix`
- **`asanyarray`**: always returns an `array` object or a subclass derived from it, depending on the input. For instance if you pass in a `matrix` it returns a `matrix`.

These functions all accept both arrays and matrices (among other things like Python lists), and thus are useful when writing functions that should accept any array-like object.

In the table below, it is assumed that you have executed the following commands in Python:

```
from numpy import *
import scipy as Sci
from scipy.linalg import lu
```

[The last line is necessary as of scipy release 0.4.6 to get access to scipy's linear algebra sub-module. It will probably be fixed in a future release so that the last line is unnecessary.]

Also assume below that if the Notes talk about "matrix" that the arguments are rank 2 entities.

The notation `asmatrix(...)` means to use the same expression as `array`, but convert to matrix with the `asmatrix()` type converter. (Note that `mat(...)` also converts to a matrix but `mat(...)` makes an extra copy in the process, and so is generally slower)

THIS IS AN EVOLVING WIKI DOCUMENT. If you find an error, or can fill in an empty box, please fix it! If there's something you'd like to see added, just add it.

MATLAB	numpy.array	numpy.matrix	Notes
<code>help func</code>	<code>help(func)</code>	<code>help(func)</code>	get help on the function <i>func</i>
<code>which func</code>	<i>(See note 'HELP')</i>		find out where <i>func</i> is defined
<code>type func</code>	<i>(See note 'HELP')</i>		print source for <i>func</i> (if not a native function)
<code>1*i,1*j,1i,1j</code>	<code>1j</code>	<code>1j</code>	complex numbers
<code>ndims(a)</code>	<code>a.ndim</code>	<code>a.ndim</code>	get the number of dimensions of a (tensor rank)
<code>size(a,n)</code>	<code>a.shape[n]</code>	<code>a.shape[n]</code>	get the number of elements of the n'th dimension of array a
<code>[1 2 3; 4 5 6]</code>	<code>array([[1.,2.,3.],</code>	<code>mat([[1.,2.,3.],</code> <code>[4.,5.,6.]])</code> or	2x3 matrix literal

	[4.,5.,6.])	mat("1 2 3; 4 5 6")	
a(2,5)	a[1,4]	a[1,4]	access element in second row, fifth column
a(2,:)	a[1] or a[1,:]	a[1] or a[1,:]	entire second row of a
a(1:5,:)	a[0:5] or a[:5] or a[0:5,:]	a[0:5] or a[:5] or a[0:5,:]	the first five rows of a
a(end-5:end,:)	a[-5:]	a[-5:]	the last five rows of a
a(1:2:end,:)	a[::2,:]	a[::2,:]	every other row of a, starting with the first
a(end:-1:1,:)	a[::-1,:]	a[::-1,:]	a with rows in reverse order
a.'	a.transpose()	a.T	transpose of a
a'	a.conj().transpose()	a.H	conjugate transpose of a
a * b	matrixmultiply(a,b)	a * b	matrix multiply
a .* b	a * b	mat(a.A * b.A)	element-wise multiply
(a>0.5)	(a>0.5)	(a>0.5)	matrix whose i,jth element is (a_ij > 0.5)
find(a>0.5)	where(a>0.5)	where(a>0.5)	find the indices where (a > 0.5)
a .* (a>0.5)	a * (a>0.5)	mat(a.A * (a>0.5).A)	a with elements less than 0.5 zeroed out
a(:) = 3	a[:] = 3	a[:] = 3	set all values to the same scalar value
y=x	y = x.copy()	y = x.copy()	numpy assigns by reference
y=x(2,:)	y = x[2,:].copy()	y = x[2,:].copy()	numpy slices are by reference
y=x(:)	y = x.flatten(1)	y = x.flatten(1)	turn array into vector (note that this forces a copy)

1:10	arange(1.,11.) or r_[1.:11.] or r_[1:10:10j]	mat(arange (1.,11.)) or r_[1.:11.,'r']	create an increasing vector <i>see note 'RANGES'</i>
0:9	arange(10.) or r_[:10.] or r_[:9:10j]	mat(arange(10.)) or r_[:10., 'r']	create an increasing vector <i>see note 'RANGES'</i>
[1:10]'	arange(1.,11.) [:, NewAxis]	r_[1.:11.,'c']	create a column vector
zeros(3,4)	zeros((3,4), float)	asmatrix(...)	3x4 rank-2 array full of zeros
zeros(3,4,5)	zeros((3,4,5), float)	asmatrix(...)	3x4x5 rank-3 array full of zeros
ones(3,4)	ones((3,4), float)	asmatrix(...)	3x4 rank-2 array full of ones
eye(3)	eye(3)	asmatrix(...)	3x3 identity matrix
rand(3,4)	rand(3,4)	asmatrix(...)	random 3x4 matrix
linspace (1,3,4)	linspace(1,3,4)	asmatrix(...)	4 equally spaced samples between 1 and 3, inclusive
[x,y] =meshgrid (0:8,0:5)	mgrid[0:9.,0:6.]	asmatrix(...)	two 2D arrays: one of x values, the other of y values
	ogrid[0:9.,0:6.]	asmatrix(...)	the best way to eval functions on a grid
repmat(a, m, n)	Sci.linalg.kron(ones ((m,n)),a)	asmatrix(...)	create m by n copies of a
max(max(a))	a.max()	a.max()	maximum element of a (with ndims(a)<=2 for matlab)
max(a)	a.max(0)	a.max(0)	maximum element of each column of matrix a
max(a,[],2)	a.max(1)	a.max(1)	maximum element of each row of matrix a

[a b]	concatenate((a,b),1) or hstack((a,b))	concatenate ((a,b),1)	join
[a; b]	concatenate((a,b)) or vstack((a,b))	concatenate ((a,b))	join
a && b	a and b	a and b	short-circuiting logical AND operator (Python native operator)
a b	or	or	short-circuiting logical OR operator (Python native operator)
a & b	logical_and(a,b)	logical_and(a,b)	element-by-element AND operator (Numpy ufunc) <i>see note 'LOGICOPS'</i>
a b	logical_or(a,b)	logical_or(a,b)	element-by-element OR operator (Numpy ufunc) <i>see note 'LOGICOPS'</i>
bitand(a,b)	a & b	a & b	bitwise AND operator (Python native?)
bitor(a,b)	a b	a b	bitwise OR operator (Python native?)
inv(a)	linalg.inv(a)	linalg.inv(a)	inverse of square matrix a
pinv(a)	linalg.pinv(a)	linalg.pinv(a)	pseudo-inverse of matrix a
a\b	linalg.solve(a,b)	linalg.solve(a,b)	solution of $a x = b$ for x <i>see note 'SOLVE'</i>
a/b			solution of $x a = b$ for x <i>see note 'MATDIV'</i>
[U,S,V]=svd (a)	(U, S, V) = linalg.svd (a)	(U, S, V) = linalg.svd(a)	singular value decomposition of a
chol(a)	linalg.cholesky(a)	linalg.cholesky(a)	cholesky factorization of a
			eigenvalues and

<code>[V,D]=eig(a)</code>	<code>linalg.eig(a)</code>	<code>linalg.eig(a)</code>	eigenvectors of a
<code>[V,D]=eigs(a,k)</code>			find the k largest eigenvalues and eigenvectors of a
<code>[Q,R,P]=qr(a,0)</code>	<code>(Q,R)=Sci.linalg.qr(a)</code>	<code>asmatrix(...)</code>	QR decomposition
<code>[L,U,P]=lu(a)</code>	<code>(L,U)=Sci.linalg.lu(a)</code> or <code>(LU,P)=Sci.linalg.lu_factor(a)</code>	<code>asmatrix(...)</code>	LU decomposition
<code>conjgrad</code>	<code>Sci.linalg.cg</code>	<code>asmatrix(...)</code>	Conjugate gradients solver
<code>fft(a)</code>	<code>fft(a)</code>	<code>asmatrix(...)</code>	Fourier transform of a
<code>ifft(a)</code>	<code>ifft(a)</code>	<code>asmatrix(...)</code>	inverse Fourier transform of a
<code>sort(a)</code>	<code>sort(a)</code> or <code>a.sort()</code>	<code>asmatrix(...)</code>	sort the matrix
<code>sortrows(a,i)</code>	<code>a[argsort(a[:,0],i)]</code>		sort the rows of the matrix

Notes

HELP: There is no direct equivalent of MATLAB's `which` command, but often the `__module__` field of an object will give you the module it is defined in, then the module's `__file__` will give you the file. Python also has an `inspect` module (do `import inspect`) which provides functions like `getfile` and `getsource` that are similar to MATLAB's `which` and `type`. In particular `print(inspect.getsource(func))` is similar to `type func`.

RANGES: In Matlab, `0:5` can be used as both a range literal and a 'slice' index (inside parenthesis); however, in Python, constructs like `0:5` can *only* be used as a slice index (inside square brackets). Thus the somewhat quirky `r_` object was created to allow numpy to have a similar terse range construction mechanism. Note that `r_` is not called like a function or a constructor, but rather *indexed* using square brackets, which

allows the use of Python's slice syntax in the arguments.

LOGICOPS: `&` or `|` in Numpy is bitwise AND/OR, while in Matlab `&` and `|` are logical AND/OR. The difference should be clear to anyone with significant programming experience. The two can appear to work the same, but there are important differences. If you would have used Matlab's `&` or `|` operators, you should use the Numpy ufuncs `logical_and`/`logical_or`. The notable differences between Matlab's and Numpy's `&` and `|` operators are:

- Non-logical `{0,1}` inputs: Numpy's output is the bitwise AND of the inputs. Matlab treats any non-zero value as 1 and returns the logical AND. For example `(3 & 4)` in Numpy is 3, while in Matlab both 3 and 4 are considered logical true and `(4 & 3)` returns 1 (logical true).
- Precedence: Numpy's `&` operator is higher precedence than logical operators like `<` and `>`; Matlab's is the reverse.

SOLVE: `linalg.solve` only works on square matrices. For over-determined systems use `linalg.lstsq(a,b)`.

MATDIV. You can define `/` in terms of solve and transpose: `x/y = (y.\x).'`




Links

See <http://www.37mm.no/matlab-python-xref.html> for another Matlab/NumPy cross-reference.

See <http://urapiv.wordpress.com> for an open-source project (URAPIV) that attempts to move from MATLAB to Python (PyPIV) <http://sourceforge.net/projects/pypiv> with SciPy / NumPy.

In order to create a programming environment similar to the one presented by MATLAB, the following are useful:

- [IPython](#): an interactive environment with many features geared towards efficient work in typical scientific usage very similar (with some enhancements) to MATLAB console.
- [Matplotlib](#): a 2D plotting package with a list of commands similar to the ones found in matlab. Matplotlib is very well integrated with IPython.
- [SPE](#) is a good free IDE for python. Has an interactive prompt.

-  Eclipse: as one nice option for python code edition via the  pydev plugin.
-  Wing IDE: a commercial IDE available for multiple platforms. The professional version has an interactive debugging prompt similar to MATLAB's.

An extensive list of tools for scientific work with python is in the link: [Topical Software](#).

MATLAB® and SimuLink® are registered trademarks of The MathWorks.

last edited 2006-05-13 21:42:21 by Dan