from numpy import \*

import scipy as Sci

import scipy.linalg

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

**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.**

**General Purpose Equivalents**

|  |  |  |
| --- | --- | --- |
| **MATLAB** | **numpy** | **Notes** |
| help func | info(func) or help(func) or func? (in Ipython) | get help on the function *func* |
| which func | ([*See note 'HELP'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#whichNotes)) | find out where *func* is defined |
| type func | source(func) or func?? (in Ipython) | print source for *func* (if not a native function) |
| a && b | a and b | short-circuiting logical AND operator (Python native operator); scalar arguments only |
| a || b | a or b | short-circuiting logical OR operator (Python native operator); scalar arguments only |
| 1\*i,1\*j,1i,1j | 1j | complex numbers |
| eps | spacing(1) | Distance between 1 and the nearest floating point number |
| ode45 | scipy.integrate.ode(f).set\_integrator('dopri5') | integrate an ODE with Runge-Kutta 4,5 |
| ode15s | scipy.integrate.ode(f).\ set\_integrator('vode', method='bdf', order=15) | integrate an ODE with BDF |

**Linear Algebra Equivalents**

The notation mat(...) means to use the same expression as array, but convert to matrix with the mat() type converter.

The notation asarray(...) means to use the same expression as matrix, but convert to array with the asarray() type converter.

|  |  |  |  |
| --- | --- | --- | --- |
| **MATLAB** | **numpy.array** | **numpy.matrix** | **Notes** |
| ndims(a) | ndim(a) or a.ndim | | get the number of dimensions of a (tensor rank) |
| numel(a) | size(a) or a.size | | get the number of elements of an array |
| size(a) | shape(a) or a.shape | | get the "size" of the matrix |
| size(a,n) | a.shape[n-1] | | get the number of elements of the *n*th dimension of array a. (Note that MATLAB® uses 1 based indexing while Python uses 0 based indexing, [*See note 'INDEXING'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#indexingNotes)) |
| [ 1 2 3; 4 5 6 ] | array([[1.,2.,3.], [4.,5.,6.]]) | mat([[1.,2.,3.], [4.,5.,6.]]) or mat("1 2 3; 4 5 6") | 2x3 matrix literal |
| [ a b; c d ] | vstack([hstack([a,b]),         hstack([c,d])]) | bmat('a b; c d') | construct a matrix from blocks a,b,c, and d |
| a(end) | a[-1] | a[:,-1][0,0] | access last element in the 1xn matrix a |
| a(2,5) | a[1,4] | | access element in second row, fifth column |
| a(2,:) | a[1] or a[1,:] | | entire second row of a |
| a(1:5,:) | a[0:5] or a[:5] or a[0:5,:] | | the first five rows of a |
| a(end-4:end,:) | a[-5:] | | the last five rows of a |
| a(1:3,5:9) | a[0:3][:,4:9] | | rows one to three and columns five to nine of a. This gives read-only access. |
| a([2,4,5],[1,3]) | a[ix\_([1,3,4],[0,2])] | | rows 2,4 and 5 and columns 1 and 3. This allows the matrix to be modified, and doesn't require a regular slice. |
| a(3:2:21,:) | a[ 2:21:2,:] | | every other row of a, starting with the third and going to the twenty-first |
| a(1:2:end,:) | a[ ::2,:] | | every other row of a, starting with the first |
| a(end:-1:1,:) orflipud(a) | a[ ::-1,:] | | a with rows in reverse order |
| a([1:end 1],:) | a[r\_[:len(a),0]] | | a with copy of the first row appended to the end |
| a.' | a.transpose() or a.T | | transpose of a |
| a' | a.conj().transpose() ora.conj().T | a.H | conjugate transpose of a |
| a \* b | dot(a,b) | a \* b | matrix multiply |
| a .\* b | a \* b | multiply(a,b) | element-wise multiply |
| a./b | a/b | | element-wise divide |
| a.^3 | a\*\*3 | power(a,3) | element-wise exponentiation |
| (a>0.5) | (a>0.5) | | matrix whose i,jth element is (a\_ij > 0.5) |
| find(a>0.5) | nonzero(a>0.5) | | find the indices where (a > 0.5) |
| a(:,find(v>0.5)) | a[:,nonzero(v>0.5)[0]] | a[:,nonzero(v.A>0.5)[0]] | extract the columms of a where vector v > 0.5 |
| a(:,find(v>0.5)) | a[:,v.T>0.5] | a[:,v.T>0.5)] | extract the columms of a where column vector v > 0.5 |
| a(a<0.5)=0 | a[a<0.5]=0 | | a with elements less than 0.5 zeroed out |
| 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 | | set all values to the same scalar value |
| y=x | y = x.copy() | | numpy assigns by reference |
| y=x(2,:) | y = x[1,:].copy() | | numpy slices are by reference |
| y=x(:) | 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 [*see note 'RANGES'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#rangeliteralNotes) |
| 0:9 | arange(10.) or  r\_[:10.] or  r\_[:9:10j] | mat(arange(10.)) or  r\_[:10.,'r'] | create an increasing vector [*see note 'RANGES'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#rangeliteralNotes) |
| [1:10]' | arange(1.,11.)[:, newaxis] | r\_[1.:11.,'c'] | create a column vector |
| zeros(3,4) | zeros((3,4)) | mat(...) | 3x4 rank-2 array full of 64-bit floating point zeros |
| zeros(3,4,5) | zeros((3,4,5)) | mat(...) | 3x4x5 rank-3 array full of 64-bit floating point zeros |
| ones(3,4) | ones((3,4)) | mat(...) | 3x4 rank-2 array full of 64-bit floating point ones |
| eye(3) | eye(3) | mat(...) | 3x3 identity matrix |
| diag(a) | diag(a) | mat(...) | vector of diagonal elements of a |
| diag(a,0) | diag(a,0) | mat(...) | square diagonal matrix whose nonzero values are the elements of a |
| rand(3,4) | random.rand(3,4) | mat(...) | random 3x4 matrix |
| linspace(1,3,4) | linspace(1,3,4) | mat(...) | 4 equally spaced samples between 1 and 3, inclusive |
| [x,y]=meshgrid(0:8,0:5) | mgrid[0:9.,0:6.] or  meshgrid(r\_[0:9.],r\_[0:6.] | mat(...) | two 2D arrays: one of x values, the other of y values |
|  | ogrid[0:9.,0:6.] or  ix\_(r\_[0:9.],r\_[0:6.] | mat(...) | the best way to eval functions on a grid |
| [x,y]=meshgrid([1,2,4],[2,4,5]) | meshgrid([1,2,4],[2,4,5]) | mat(...) |  |
|  | ix\_([1,2,4],[2,4,5]) | mat(...) | the best way to eval functions on a grid |
| repmat(a, m, n) | tile(a, (m, n)) | mat(...) | create m by n copies of a |
| [a b] | concatenate((a,b),1) or  hstack((a,b)) or  column\_stack((a,b)) or  c\_[a,b] | concatenate((a,b),1) | concatenate columns of a and b |
| [a; b] | concatenate((a,b)) or  vstack((a,b)) or  r\_[a,b] | concatenate((a,b)) | concatenate rows of a and b |
| max(max(a)) | a.max() | | maximum element of a (with ndims(a)<=2 for matlab) |
| max(a) | a.max(0) | | maximum element of each column of matrix a |
| max(a,[],2) | a.max(1) | | maximum element of each row of matrix a |
| max(a,b) | maximum(a, b) | | compares a and b element-wise, and returns the maximum value from each pair |
| norm(v) | sqrt(dot(v,v)) or  Sci.linalg.norm(v) or  linalg.norm(v) | sqrt(dot(v.A,v.A))or  Sci.linalg.norm(v)or  linalg.norm(v) | L2 norm of vector v |
| a & b | logical\_and(a,b) | | element-by-element AND operator (Numpy ufunc) [*see note 'LOGICOPS'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#logicalNotes) |
| a | b | logical\_or(a,b) | | element-by-element OR operator (Numpy ufunc) [*see note 'LOGICOPS'*](http://wiki.scipy.org/NumPy_for_Matlab_Users#logicalNotes) |
| bitand(a,b) | a & b | | bitwise AND operator (Python native and Numpy ufunc) |
| bitor(a,b) | a | b | | bitwise OR operator (Python native and Numpy ufunc) |
| inv(a) | linalg.inv(a) | | inverse of square matrix a |
| pinv(a) | linalg.pinv(a) | | pseudo-inverse of matrix a |
| rank(a) | linalg.matrix\_rank(a) | | rank of a matrix a |
| a\b | linalg.solve(a,b) if a is square linalg.lstsq(a,b) otherwise | | solution of a x = b for x |
| b/a | Solve a.T x.T = b.T instead | | solution of x a = b for x |
| [U,S,V]=svd(a) | U, S, Vh = linalg.svd(a), V = Vh.T | | singular value decomposition of a |
| chol(a) | linalg.cholesky(a).T | | cholesky factorization of a matrix (chol(a) in matlab returns an upper triangular matrix, but linalg.cholesky(a) returns a lower triangular matrix) |
| [V,D]=eig(a) | D,V = linalg.eig(a) | | eigenvalues and eigenvectors of a |
| [V,D]=eig(a,b) | V,D = Sci.linalg.eig(a,b) | | eigenvalues and eigenvectors of a,b |
| [V,D]=eigs(a,k) |  |  | find the k largest eigenvalues and eigenvectors of a |
| [Q,R,P]=qr(a,0) | Q,R = Sci.linalg.qr(a) | mat(...) | QR decomposition |
| [L,U,P]=lu(a) | L,U = Sci.linalg.lu(a) or  LU,P=Sci.linalg.lu\_factor(a) | mat(...) | LU decomposition (note: P(Matlab) == transpose(P(numpy)) ) |
| conjgrad | Sci.linalg.cg | mat(...) | Conjugate gradients solver |
| fft(a) | fft(a) | mat(...) | Fourier transform of a |
| ifft(a) | ifft(a) | mat(...) | inverse Fourier transform of a |
| sort(a) | sort(a) or a.sort() | mat(...) | sort the matrix |
| [b,I] = sortrows(a,i) | I = argsort(a[:,i]), b=a[I,:] | | sort the rows of the matrix |
| regress(y,X) | linalg.lstsq(X,y) | | multilinear regression |
| decimate(x, q) | Sci.signal.resample(x, len(x)/q) | | downsample with low-pass filtering |
| unique(a) | unique(a) |  |  |
| squeeze(a) | a.squeeze() |  |  |

# Make all numpy available via shorter 'num' prefix

import numpy as num

# Make all matlib functions accessible at the top level via M.func()

import numpy.matlib as M

# Make some matlib functions accessible directly at the top level via, e.g. rand(3,3)

from numpy.matlib import rand,zeros,ones,empty,eye

# Define a Hermitian function

def hermitian(A, \*\*kwargs):

return num.transpose(A,\*\*kwargs).conj()

# Make some shorcuts for transpose,hermitian:

# num.transpose(A) --> T(A)

# hermitian(A) --> H(A)

T = num.transpose

H = hermitian

# MATLAB packages/tools and equivalent for use with NumPy

* **Plotting**: matplotlib provides a workalike interface for 2D plotting; [Mayavi](http://code.enthought.com/projects/mayavi/) provides 3D plotting
* **Symbolic calculation**: [swiginac](http://wiki.scipy.org/swiginac) appears to be the most complete current option. [sympy](http://wiki.scipy.org/sympy) is a project aiming at bringing the basic symbolic calculus functionalities to Python. Also to be noted is [PyDSTool](http://wiki.scipy.org/PyDSTool) which provides some basic symbolic functionality.
* **Linear algebra**: scipy.linalg provides the LAPACK routines
* **Interpolation**: [[/ScipyPackages](http://wiki.scipy.org/NumPy_for_Matlab_Users/ScipyPackages)/Interpolate scipy.interpolate] provides several spline interpolation tools
* **Numerical integration**: scipy.integrate provides several tools for integrating functions as well as some basic ODE integrators. Convert XML vector field specifications automatically using [VFGEN](http://www.warrenweckesser.net/vfgen/).
* **Dynamical systems**: [PyDSTool](http://pydstool.sourceforge.net/) provides a large dynamical systems and modeling package, including good ODE/DAE integrators. Convert XML vector field specifications automatically using [VFGEN](http://www.warrenweckesser.net/vfgen/).
* **Simulink**: no alternative is currently available.