

# Python For Data Science Cheat Sheet

Also see NumPy

## SciPy - Linear Algebra

Learn More Python for Data Science [Interactively](https://www.datacamp.com) at [www.datacamp.com](https://www.datacamp.com)



### SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.



### Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)])
```

### Index Tricks

>>> np.mgrid[0:5,0:5]	Create a dense meshgrid
>>> np.ogrid[0:2,0:2]	Create an open meshgrid
>>> np.r_[3,[0]*5,-1:10j]	Stack arrays vertically (row-wise)
>>> np.c_[b,c]	Create stacked column-wise arrays

### Shape Manipulation

>>> np.transpose(b)	Permute array dimensions
>>> b.flatten()	Flatten the array
>>> np.hstack((b,c))	Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))	Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)	Split the array horizontally at the 2nd index
>>> np.vsplit(d,2)	Split the array vertically at the 2nd index

### Polynomials

>>> from numpy import polyld	
>>> p = polyld([3,4,5])	Create a polynomial object

### Vectorizing Functions

>>> def myfunc(a): if a < 0: return a*2 else: return a/2	
>>> np.vectorize(myfunc)	Vectorize functions

### Type Handling

>>> np.real(b)	Return the real part of the array elements
>>> np.imag(b)	Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)	Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)	Cast object to a data type

### Other Useful Functions

>>> np.angle(b,deg=True)	Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)	Create an array of evenly spaced values (number of samples)
>>> g[3:] += np.pi	
>>> np.unwrap(g)	Unwrap
>>> np.logspace(0,10,3)	Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])	Return values from a list of arrays depending on conditions
>>> misc.factorial(a)	Factorial
>>> misc.comb(10,3,exact=True)	Combine N things taken at k time
>>> misc.central_diff_weights(3)	Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)	Find the n-th derivative of a function at a point

## Linear Algebra

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

### Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

### Basic Matrix Routines

#### Inverse

```
>>> A.I
>>> linalg.inv(A)
```

Inverse  
Inverse

#### Transposition

```
>>> A.T
>>> A.H
```

Tranpose matrix  
Conjugate transposition

#### Trace

```
>>> np.trace(A)
```

Trace

#### Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Frobenius norm  
L1 norm (max column sum)  
L inf norm (max row sum)

#### Rank

```
>>> np.linalg.matrix_rank(C)
```

Matrix rank

#### Determinant

```
>>> linalg.det(A)
```

Determinant

#### Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(F,E)
```

Solver for dense matrices  
Solver for dense matrices  
Least-squares solution to linear matrix equation

#### Generalized inverse

```
>>> linalg.pinv(C)
>>> linalg.pinv2(C)
```

Compute the pseudo-inverse of a matrix (least-squares solver)  
Compute the pseudo-inverse of a matrix (SVD)

### Creating Sparse Matrices

>>> F = np.eye(3, k=1)	Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2))	Create a 2x2 identity matrix
>>> C[C > 0.5] = 0	
>>> H = sparse.csr_matrix(C)	Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)	Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)	Dictionary Of Keys matrix
>>> E.todense()	Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)	Identify sparse matrix

### Sparse Matrix Routines

#### Inverse

```
>>> sparse.linalg.inv(I)
```

Inverse

#### Norm

```
>>> sparse.linalg.norm(I)
```

Norm

#### Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Solver for sparse matrices

### Sparse Matrix Functions

>>> sparse.linalg.expm(I)	Sparse matrix exponential
---------------------------	---------------------------

## Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

### Matrix Functions

#### Addition

```
>>> np.add(A,D)
```

Addition

#### Subtraction

```
>>> np.subtract(A,D)
```

Subtraction

#### Division

```
>>> np.divide(A,D)
```

Division

#### Multiplication

```
>>> A @ D
```

Multiplication operator

```
>>> np.multiply(D,A)
```

(Python 3)

```
>>> np.dot(A,D)
```

Multiplication

```
>>> np.vdot(A,D)
```

### Exponential Functions

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

Matrix exponential  
Matrix exponential (Taylor Series)  
Matrix exponential (eigenvalue decomposition)

### Logarithm Function

```
>>> linalg.logm(A)
```

Matrix logarithm

### Trigonometric Functions

```
>>> linalg.sinm(D)
>>> linalg.cosm(D)
>>> linalg.tanm(A)
```

Matrix sine  
Matrix cosine  
Matrix tangent

### Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

Hypperbolic matrix sine  
Hyperbolic matrix cosine  
Hyperbolic matrix tangent

### Matrix Sign Function

```
>>> np.signm(A)
```

Matrix sign function

### Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Matrix square root

### Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)
```

Evaluate matrix function

### Decompositions

#### Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
```

Solve ordinary or generalized eigenvalue problem for square matrix  
Unpack eigenvalues  
First eigenvector  
Second eigenvector  
Unpack eigenvalues

```
>>> l1, l2 = la
```

```
>>> v[:,0]
```

```
>>> v[:,1]
```

```
>>> linalg.eigvals(A)
```

#### Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
```

Singular Value Decomposition (SVD)

```
>>> M,N = B.shape
```

```
>>> Sig = linalg.diagsvd(s,M,N)
```

Construct sigma matrix in SVD

#### LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

LU Decomposition

### Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)
```

Eigenvalues and eigenvectors  
SVD

```
>>> sparse.linalg.svds(H, 2)
```

DataCamp

Learn Python for Data Science [Interactively](https://www.datacamp.com)

