

# Python For Data Science Cheat Sheet

## 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

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

#### Shape Manipulation

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

#### Polynomials

<pre>&gt;&gt;&gt; from numpy import polyld &gt;&gt;&gt; p = polyld([3,4,5])</pre>	Create a polynomial object
---	----------------------------

#### Vectorizing Functions

<pre>&gt;&gt;&gt; def myfunc(a):     if a &lt; 0:         return a*2     else:         return a/2 &gt;&gt;&gt; np.vectorize(myfunc)</pre>	Vectorize functions
---	---------------------

#### Type Handling

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

#### Other Useful Functions

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

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

<pre>&gt;&gt;&gt; sparse.linalg.expm(I)</pre>	Sparse matrix exponential
---	---------------------------

### Asking For Help

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

[Also see NumPy](#)

### 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 (Python 3)

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

Multiplication

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

Dot product

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

Vector dot product

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

Inner product

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

Outer product

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

Tensor dot product

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

Kronecker product

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

Eigenvalues and eigenvectors  
SVD

DataCamp

Learn Python for Data Science [Interactively](#)

