


 datacamp

Python For Data Science

SciPy Cheat Sheet

Learn SciPy online at [www.DataCamp.com](https://www.datacamp.com)

SciPy

 **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
<pre>>>> 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)]])</pre>		
Index Tricks		
<pre>>>> 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,-1:1:10j]] #Stack arrays vertically (row-wise) >>> np.c_[b,c] #Create stacked column-wise arrays</pre>		
Shape Manipulation		
<pre>>>> 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.vpsplit(d,2) #Split the array vertically at the 2nd index</pre>		
Polynomials		
<pre>>>> from numpy import poly1d >>> p = poly1d([3,4,5]) #Create a polynomial object</pre>		
Vectorizing Functions		
<pre>>>> def myfunc(a): if a < 0: return a*2 else: return a/2 >>> np.vectorize(myfunc) #Vectorize functions</pre>		
Type Handling		
<pre>>>> np.real(c) #Return the real part of the array elements >>> np.imag(c) #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</pre>		
Other Useful Functions		
<pre>>>> 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</pre>		

>	Linear Algebra	Also see NumPy
You'll use the <code>linalg</code> and <code>sparse</code> modules. Note that <code>scipy.linalg</code> contains and expands on <code>numpy.linalg</code> .		
<pre>>>> from scipy import linalg, sparse</pre>		
Creating Matrices		
<pre>>>> 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]])</pre>		
Basic Matrix Routines		
<div>Inverse</div> <pre>>>> A.I #Inverse >>> linalg.inv(A) #Inverse >>> A.T #Transpose matrix >>> A.H #Conjugate transposition >>> np.trace(A) #Trace</pre> <div>Norm</div> <pre>>>> linalg.norm(A) #Frobenius norm >>> linalg.norm(A,1) #L1 norm (max column sum) >>> linalg.norm(A,np.inf) #L inf norm (max row sum)</pre> <div>Rank</div> <pre>>>> np.linalg.matrix_rank(C) #Matrix rank</pre> <div>Determinant</div> <pre>>>> linalg.det(A) #Determinant</pre> <div>Solving linear problems</div> <pre>>>> linalg.solve(A,b) #Solver for dense matrices >>> E = np.mat(a).T #Solver for dense matrices >>> linalg.lstsq(D,E) #Least-squares solution to linear matrix equation</pre> <div>Generalized inverse</div> <pre>>>> linalg.pinv(C) #Compute the pseudo-inverse of a matrix (least-squares solver) >>> linalg.pinv2(C) #Compute the pseudo-inverse of a matrix (SVD)</pre>		
Creating Sparse Matrices		
<pre>>>> 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</pre>		
Sparse Matrix Routines		
<div>Inverse</div> <pre>>>> sparse.linalg.inv(I) #Inverse</pre> <div>Norm</div> <pre>>>> sparse.linalg.norm(I) #Norm</pre> <div>Solving linear problems</div> <pre>>>> sparse.linalg.spsolve(H,I) #Solver for sparse matrices</pre>		
Sparse Matrix Functions		
<pre>>>> sparse.linalg.expm(I) #Sparse matrix exponential</pre>		
Sparse Matrix Decompositions		
<pre>>>> la, v = sparse.linalg.eigs(F,1) #Eigenvalues and eigenvectors >>> sparse.linalg.svds(H, 2) #SVD</pre>		

>

Asking For Help

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

Matrix Functions
<div>Addition</div> <pre>>>> np.add(A,D) #Addition</pre> <div>Subtraction</div> <pre>>>> np.subtract(A,D) #Subtraction</pre> <div>Division</div> <pre>>>> np.divide(A,D) #Division</pre> <div>Multiplication</div> <pre>>>> 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</pre> <div>Exponential Functions</div> <pre>>>> linalg.expm(A) #Matrix exponential >>> linalg.expm2(A) #Matrix exponential (Taylor Series) >>> linalg.expm3(D) #Matrix exponential (eigenvalue decomposition)</pre> <div>Logarithm Function</div> <pre>>>> linalg.logm(A) #Matrix logarithm</pre> <div>Trigonometric Functions</div> <pre>>>> linalg.sinm(D) Matrix sine >>> linalg.cosm(D) Matrix cosine >>> linalg.tanm(A) Matrix tangent</pre> <div>Hyperbolic Trigonometric Functions</div> <pre>>>> linalg.sinhm(D) #Hypperbolic matrix sine >>> linalg.coshm(D) #Hyperbolic matrix cosine >>> linalg.tanhm(A) #Hyperbolic matrix tangent</pre> <div>Matrix Sign Function</div> <pre>>>> np.sigm(A) #Matrix sign function</pre> <div>Matrix Square Root</div> <pre>>>> linalg.sqrtm(A) #Matrix square root</pre> <div>Arbitrary Functions</div> <pre>>>> linalg.funm(A, lambda x: x*x) #Evaluate matrix function</pre>
Decompositions
<div>Eigenvalues and Eigenvectors</div> <pre>>>> la, v = linalg.eig(A) #Solve ordinary or generalized eigenvalue problem for square matrix >>> l1, l2 = la #Unpack eigenvalues >>> v[:,0] #First eigenvector >>> v[:,1] #Second eigenvector >>> linalg.eigvals(A) #Unpack eigenvalues</pre> <div>Singular Value Decomposition</div> <pre>>>> 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</pre> <div>LU Decomposition</div> <pre>>>> P,L,U = linalg.lu(C) #LU Decomposition</pre>

Learn Data Skills Online at

[www.DataCamp.com](https://www.datacamp.com)

 datacamp