SciPy Cheatsheet

Basic Concepts

SciPy is a library used for scientific and technical computing. It builds on NumPy and provides advanced mathematical functions.

- Modules Overview:
  - `scipy.integrate`: Integration and ODE solvers.
  - `scipy.linalg`: Linear algebra beyond NumPy.
  - `scipy.optimize`: Optimization and root finding.
  - `scipy.stats`: Probability distributions and statistical tests.
  - `scipy.signal`: Signal processing.
  - `scipy.sparse`: Sparse matrices.

Optimization (`scipy.optimize`)

Find global or local minima of functions and find roots of equations.

```
from scipy.optimize import minimize, root

# Minify a function
res = minimize(f, x0, method='BFGS')

# Find roots of an equation (f(x) = 0)
sol = root(f, x0)
```

Integration (`scipy.integrate`)

Compute numerical integrals and solve ordinary differential equations.

```
from scipy.integrate import quad, solve_ivp

# Single integration
result, error = quad(lambda x: x**2, 0, 1)

# Solve Initial Value Problem (ODE)
# dy/dt = f(t, y)
sol = solve_ivp(f, [0, 10], [y0])
```

Statistics (`scipy.stats`)

Extensive library of probability distributions and advanced statistical tests.

```
from scipy.stats import norm, ttest_ind

# Normal distribution
rv = norm(loc=0, scale=1) # Mean 0, Std 1
p = rv.pdf(0) # Probability density at 0
cdf = rv.cdf(1.96) # Cumulative distribution function

# Independent t-test
t_stat, p_val = ttest_ind(sample1, sample2)
```

Linear Algebra (`scipy.linalg`)

Advanced linear algebra operations for efficiency and stability.

```
from scipy.linalg import inv, det, eig, solve

# Calculate inverse and determinant
A_inv = inv(M)
val_det = det(M)

# Solve linear system Ax = b
x = solve(A, b)

# Eigenvalues and Eigenvectors
vals, vecs = eig(M)

# Matrix Decompositions (LU, QR, SVD)
from scipy.linalg import lu, qr, svd
P, L, U = lu(M)
```

Signal Processing (`scipy.signal`)

Tools for filtering, spectral analysis, and LTI systems.

```
from scipy.signal import butter, lfilter

# Create a low-pass filter
b, a = butter(N=4, Wn=0.2, btype='low')
filtered_data = lfilter(b, a, data)
```

Sparse Matrices (`scipy.sparse`)

Efficient storage and calculation for matrices where most elements are zero.

```
from scipy.sparse import csr_matrix, csc_matrix

# Compressed Sparse Row matrix
S = csr_matrix(dense_matrix)
```

Interpolation (`scipy.interpolate`)

Construct new data points within the range of a discrete set of known data points.

```
from scipy.interpolate import interp1d

f = interp1d(x, y, kind='cubic')
y_new = f(x_new)
```