南京大学
自然语言处理研究组
Natural Language Processing Group

南京大學
NANJING UNIVERSITY

SciPy和科学计算

黄书剑

SciPy生态系统

- **SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering.**

IP[y]:
IPython

NumPy Base N-dimensional array package	**SciPy library** Fundamental library for scientific computing	**Matplotlib** Comprehensive 2-D plotting
IPython Enhanced interactive console	**SymPy** Symbolic mathematics	**pandas** Data structures & analysis

数学分析

- 函数、极限、连续
- 一元函数微分学及其应用
- 一元函数积分学及其应用

- 无穷级数
- 多元函数微分学及其应用
- 多元函数积分学及其应用

高等代数

- 多项式
- 行列式
- 线性方程组
- 矩阵
- 二次型

- 线性空间
- 线性变换
- 欧几里得空间
- 双线性函数与辛空间

概率论与数理统计

- 随机变量及其分布
- 随机变量数字特征
- 集中不等式

- 参数估计
- 假设检验
- 回归分析与方差分析

最优化方法

- 凸集合
- 凸函数
- 凸优化问题
- 对偶性
- 凸函数优化
- 平滑函数优化
- 随机优化
- 分布式优化
- 在线优化

SCIPY简介

SciPy

- **The SciPy library is one of the core packages that make up the SciPy stack. It provides many user-friendly and efficient numerical routines:**
  - numerical integration 数值积分
  - interpolation 插值
  - optimization 优化
  - linear algebra 线性代数
  - statistics 统计

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

与基于符号的计算不同，SciPy侧重于针对具体数值进行计算和求解

LINEAR ALGEBRA

scipy.linalg

- See also numpy.linalg for more linear algebra functions. Note that although scipy.linalg imports most of them, identically named functions from scipy.linalg may offer more or slightly differing functionality.
  - 矩阵计算
  - 特征值
  - 矩阵分解

Basics

inv(a[, overwrite_a, check_finite])	Compute the inverse of a matrix.
solve(a, b[, sym_pos, lower, overwrite_a, ...])	Solves the linear equation set `a * x = b` for the unknown `x` for square `a` matrix.
solve_banded(l_and_u, ab, b[, overwrite_ab, ...])	Solve the equation a x = b for x, assuming a is banded matrix.
solveh_banded(ab, b[, overwrite_ab, ...])	Solve equation a x = b.
solve_circulant(c, b[, singular, tol, ...])	Solve C x = b for x, where C is a circulant matrix.
solve_triangular(a, b[, trans, lower, ...])	Solve the equation *a x = b* for *x*, assuming a is a triangular matrix.
solve_toeplitz(c_or_cr, b[, check_finite])	Solve a Toeplitz system using Levinson Recursion
matmul_toeplitz(c_or_cr, x[, check_finite, ...])	Efficient Toeplitz Matrix-Matrix Multiplication using FFT
det(a[, overwrite_a, check_finite])	Compute the determinant of a matrix
norm(a[, ord, axis, keepdims, check_finite])	Matrix or vector norm.
lstsq(a, b[, cond, overwrite_a, ...])	Compute least-squares solution to equation Ax = b.
pinv(a[, cond, rcond, return_rank, check_finite])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
pinv2(a[, cond, rcond, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
pinvh(a[, cond, rcond, lower, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a Hermitian matrix.
kron(a, b)	Kronecker product.
khatri_rao(a, b)	Khatri-rao product
tril(m[, k])	Make a copy of a matrix with elements above the kth diagonal zeroed.
triu(m[, k])	Make a copy of a matrix with elements below the kth diagonal zeroed.
orthogonal_procrustes(A, B[, check_finite])	Compute the matrix solution of the orthogonal Procrustes problem.
matrix_balance(A[, permute, scale, ...])	Compute a diagonal similarity transformation for row/column balancing.
subspace_angles(A, B)	Compute the subspace angles between two matrices.
LinAlgError	Generic Python-exception-derived object raised by linalg functions.
LinAlgWarning	The warning emitted when a linear algebra related operation is close to fail conditions of the algorithm or loss of accuracy is expected.

Eigenvalue Problems

eig(a[, b, left, right, overwrite_a, ...])	Solve an ordinary or generalized eigenvalue problem of a square matrix.
eigvals(a[, b, overwrite_a, check_finite, ...])	Compute eigenvalues from an ordinary or generalized eigenvalue problem.
eigh(a[, b, lower, eigvals_only, ...])	Solve a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eigvalsh(a[, b, lower, overwrite_a, ...])	Solves a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eig_banded(a_band[, lower, eigvals_only, ...])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigvals_banded(a_band[, lower, ...])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigh_tridiagonal(d, e[, eigvals_only, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.
eigvalsh_tridiagonal(d, e[, select, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.

Decompositions

lu(a[, permute_l, overwrite_a, check_finite])	Compute pivoted LU decomposition of a matrix.
lu_factor(a[, overwrite_a, check_finite])	Compute pivoted LU decomposition of a matrix.
lu_solve(lu_and_piv, b[, trans, ...])	Solve an equation system, a x = b, given the LU factorization of a
svd(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.
svdvals(a[, overwrite_a, check_finite])	Compute singular values of a matrix.
diagsvd(s, M, N)	Construct the sigma matrix in SVD from singular values and size M, N.
orth(A[, rcond])	Construct an orthonormal basis for the range of A using SVD
null_space(A[, rcond])	Construct an orthonormal basis for the null space of A using SVD
ldl(A[, lower, hermitian, overwrite_a, ...])	Computes the LDLt or Bunch-Kaufman factorization of a symmetric/ hermitian matrix.
cholesky(a[, lower, overwrite_a, check_finite])	Compute the Cholesky decomposition of a matrix.
cholesky_banded(ab[, overwrite_ab, lower, ...])	Cholesky decompose a banded Hermitian positive-definite matrix
cho_factor(a[, lower, overwrite_a, check_finite])	Compute the Cholesky decomposition of a matrix, to use in cho_solve
cho_solve(c_and_lower, b[, overwrite_b, ...])	Solve the linear equations A x = b, given the Cholesky factorization of A.
cho_solve_banded(cb_and_lower, b[, ...])	Solve the linear equations `A x = b`, given the Cholesky factorization of the banded Hermitian `A`.

线性代数运算示例

- **求给定线性方程组的解**

```
In [133]: a = np.array([[1, 3, 1], [2, 1, 3], [2, 2, 1]])
     ...: b = np.array([10, 13, 9])
     ...:
     ...: x = linalg.solve(a, b)
     ...: x
Out[133]: array([1., 2., 3.])
```

线性代数运算示例

- **求给定矩阵的相关特征**
  - 行列式、秩
  - 特征值等

```
In [134]: A1 = np.arange(1,
10).reshape(3, 3)
In [135]: A1
Out[135]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
In [137]: linalg.det(A1)
Out[137]: 0.0

In [138]: np.linalg.matrix_rank(A1)
Out[138]: 2

In [139]: l, v = linalg.eig(A1)
In [140]: print(l)
[ 1.61168440e+01+0.j -1.11684397e+00+0.j
-9.75918483e-16+0.j]
In [141]: print(v)
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

OPTIMIZATION

scipy.optimize

- **包含通过各种算法进行最优化（求极值）的函数；包括非线性优化、线性规划、求根、曲线拟合等**
  - minimize_scalar, minimize
  - linprog
  - curve_fit
  - root_scalar, root

https://scipy-lectures.org/advanced/mathematical_optimization/

Optimization

Scalar functions optimization

minimize_scalar(fun[, bracket, bounds, ...]) Minimization of scalar function of one variable.

The minimize_scalar function supports the following methods:

- minimize_scalar(method='brent')
- minimize_scalar(method='bounded')
- minimize_scalar(method='golden')

Local (multivariate) optimization

minimize(fun, x0[, args, method, jac, hess, ...]) Minimization of scalar function of one or more variables.

The minimize function supports the following methods:

- minimize(method='Nelder-Mead')
- minimize(method='Powell')
- minimize(method='CG')
- minimize(method='BFGS')
- minimize(method='Newton-CG')
- minimize(method='L-BFGS-B')
- minimize(method='TNC')
- minimize(method='COBYLA')
- minimize(method='SLSQP')
- minimize(method='trust-constr')
- minimize(method='dogleg')
- minimize(method='trust-ncg')
- minimize(method='trust-krylov')
- minimize(method='trust-exact')

Least-squares and curve fitting

Nonlinear least-squares

least_squares(fun, x0[, jac, bounds, ...])	Solve a nonlinear least-squares problem with bounds on the variables.

Linear least-squares

nnls(A, b[, maxiter])	Solve `argmin_x || Ax - b ||_2` for `x>=0`.
lsq_linear(A, b[, bounds, method, tol, ...])	Solve a linear least-squares problem with bounds on the variables.

Curve fitting

curve_fit(f, xdata, ydata[, p0, sigma, ...])	Use non-linear least squares to fit a function, f, to data.

Root finding

Scalar functions

root_scalar(f[, args, method, bracket, …])	Find a root of a scalar function.
brentq(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in a bracketing interval using Brent's method.
brenth(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in a bracketing interval using Brent's method with hyperbolic extrapolation.
ridder(f, a, b[, args, xtol, rtol, maxiter, …])	Find a root of a function in an interval using Ridder's method.
bisect(f, a, b[, args, xtol, rtol, maxiter, …])	Find root of a function within an interval using bisection.
newton(func, x0[, fprime, args, tol, …])	Find a zero of a real or complex function using the Newton-Raphson (or secant or Halley's) method.
toms748(f, a, b[, args, k, xtol, rtol, …])	Find a zero using TOMS Algorithm 748 method.
RootResults(root, iterations, …)	Represents the root finding result.

Linear programming

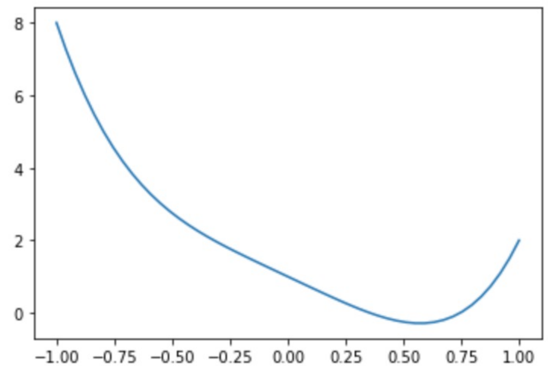
linprog(c[, A_ub, b_ub, A_eq, b_eq, bounds, ...])	Linear programming: minimize a linear objective function subject to linear equality and inequality constraints.

The linprog function supports the following methods:

- linprog(method='simplex')
- linprog(method='interior-point')
- linprog(method='revised simplex')
- linprog(method='highs-ipm')
- linprog(method='highs-ds')
- linprog(method='highs')

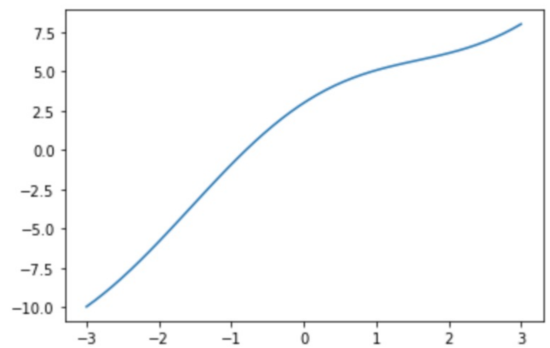
最优化示例

```
In [156]: from scipy import optimize as opt

In [157]: def objective_function(x):
     ...:     return 4 * x ** 4 - 3 * x + 1
     ...:

In [159]: res = opt.minimize_scalar(objective_function)
     ...: print(res)
     fun: -0.2878035228724982
    nfev: 16
     nit: 12
 success: True
       x: 0.5723571222032383
```



方程求根示例

```
In [164]: def func(x):
     ...:     return x * 3 + 2 * np.cos(x) + 1
     ...:
In [165]: sol = opt.root(func, 0)
In [166]: print(sol)
    fjac: array([[-1.]])
     fun: array([2.88657986e-15])
 message: 'The solution converged.'
    nfev: 8
     qtf: array([-3.42632855e-09])
       r: array([-4.43264413])
  status: 1
 success: True
       x: array([-0.79851442])
```



规划问题示例

- **生产甲、乙、丙三种产品，可以获得利润，但消耗A、B两种资源。现已知A、B的资源总量，求利润最大的生产方案。**

max x0 + x1 + x2

s.t. x0 *2 + x1 + x2 *3 <= 100

x0 *3 + x1 *2 + x2 *3 <= 120

x0, x1, x2 >= 0

optimize.linprog(...)

$$\min \quad c^T x$$

$$s.t. \begin{cases} Ax <= b \\ Aeq * x = beq \\ lb <= x <= ub \end{cases}$$

	甲	乙	丙	总量
A	2	1	3	100
B	3	2	3	120
利润	45	20	45	

规划问题示例

- **做100套钢架，用长2.5，2.3，1.5的钢条各一根，现有所有原料长度为7，通过切割得到所需钢条，求最少需要多少钢条（单位为米）。**

min x0 + x1 + x2 + x3 + x4

s.t. x0 + x1 * 2 >= 100

x0 + x2 * 2 + x3 >= 100

x0 + x2 + x3 *3 + x4*4 >= 100

2.8	2.3	1.5	
1	1	1	x0
2	0	0	x1
0	2	1	x2
0	1	3	x3
0	0	4	x4

optimize.linprog(...)

$$\min \quad c^T x$$

$$s.t. \begin{cases} Ax <= b \\ Aeq * x = beq \\ lb <= x <= ub \end{cases}$$

整数线性规划：变量取值为整数（分支定界法）

NUMERICAL INTEGRATION

Integrating functions, given function object

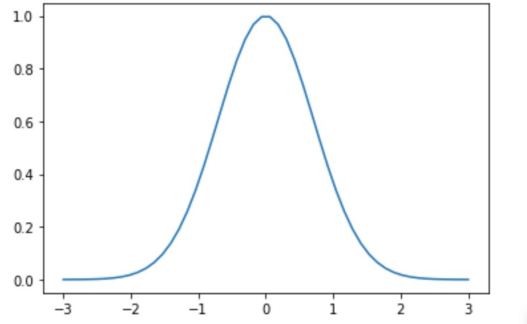
quad(func, a, b[, args, full_output, ...])	Compute a definite integral.
quad_vec(f, a, b[, epsabs, epsrel, norm, ...])	Adaptive integration of a vector-valued function.
dblquad(func, a, b, gfun, hfun[, args, ...])	Compute a double integral.
tplquad(func, a, b, gfun, hfun, qfun, rfun)	Compute a triple (definite) integral.
nquad(func, ranges[, args, opts, full_output])	Integration over multiple variables.
fixed_quad(func, a, b[, args, n])	Compute a definite integral using fixed-order Gaussian quadrature.
quadrature(func, a, b[, args, tol, rtol, ...])	Compute a definite integral using fixed-tolerance Gaussian quadrature.
romberg(function, a, b[, args, tol, rtol, ...])	Romberg integration of a callable function or method.
quad_explain([output])	Print extra information about integrate.quad() parameters and returns.
newton_cotes(rn[, equal])	Return weights and error coefficient for Newton-Cotes integration.
IntegrationWarning	Warning on issues during integration.
AccuracyWarning	

Integrating functions, given fixed samples

trapezoid(y[, x, dx, axis])	Integrate along the given axis using the composite trapezoidal rule.
cumulative_trapezoid(y[, x, dx, axis, initial])	Cumulatively integrate y(x) using the composite trapezoidal rule.
simpson(y[, x, dx, axis, even])	Integrate y(x) using samples along the given axis and the composite Simpson's rule.
romb(y[, dx, axis, show])	Romberg integration using samples of a function.

数值积分示例

```
In [167]: from scipy import integrate
In [168]: f = lambda x: np.exp(-x**2)
In [169]: integrate.quad(f, 0, 5)
#return the integration results and estimated error
Out[169]: (0.8862269254513955, 2.3183115159980698e-14)
```



Scipy回顾

- **针对特定科学计算问题的数值求解**
  - 线性代数
  - 最优化
  - 数值积分
  - ……