SciPy

Introduction :

SciPy مجموعه ای از الگوریتم های ریاضی و توابع است که بر روی کتابخانه NumPy نوشته شده است. SciPy مخفف Scientific Python است. مزیتی که کتابخانه SciPy بر زبان پایتون افزوده است این است که زبان پایتون را تبدیل به یک زبان برنامه نویسی قدرتمند برای استفاده در توسعه برنامه های پیچیده و برنامه های کاربردی کرده است. با استفاده از SciPy می توان کدهای پروژه های علمی را توسعه بخشید. مزیت SciPy نسبت به NumPy ، وجود یک سری توابع اضافه تر و بهینه سازی است که در Data Science استفاده می شوند.

عمدتا این کتابخانه به زبان پایتون نوشته شده است اما بخش هایی نیز دارد که به زبان C نوشته شده است.

SciPy Sub-package :

SciPy دارای Sub-package های مختلف برای پوشش محاسبات علمی گوناگون است. جدول زیر خلاصه شده ی این Sub-package ها را نشان می دهد.

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

نکته : Sub-package باید به صورت جداگانه import شوند.

Special Functions :

ویژگی اصلی scipy.spacial تعریف توابع ویژه متعدد ریاضیاتی و فیزیکی است. توابعی مانند : گاما ، بسل ، بتا ، Hypergeometric و... . همچنین برخی از توابع سطح پایین آماری که برای استفاده عمومی در نظر گرفته نشده اند زیرا یک رابط ساده تر برای این توابع توسط ماژول آمار ارائه شده است. اکثر این توابع می توانند آرگمان های از جنس آرایه بگیرند و نتایجی از جنس آرایه را برگردانند مانند کتابخانه NumPy. بسیاری از این توابع نیز اعداد مختلط را به عنوان ورودی دریافت می کنند.

Integration :

Sub-package انتگرال SciPy شامل چندین متد مختلف برای انتگرال گیری است که یکی از این متدها ODE یا Ordinary Differential Equation یا معادله دیفرانسیل معمولی است.

❖ General Integration :

برای گرفتن انتگرال از دستور quad استفاده می کنیم. اولین آرگمان این دستور باید به عنوان تابعی که قرار است انتگرال گیری شود مقداردهی بشود. آرگمان دوم و سوم به ترتیب کران پایین و بالای انتگرال است.

✓ **مثال :**

```
import scipy.integrate as integrate

result = integrate.quad(lambda x: x**2 + x, 0, 4)
result
```

نکته : خروجی این دستور یک تاپل است که عضو اول این تاپل حاصل انتگرال است و عضو دوم آن کران بالا را در محدوده خطا نگه می دارد.

✓ **مثال :**

```
def integrand(x, a, b):
    return a*x**2 + b

a = 2
b = 1
result = integrate.quad(integrand, 0, 1, args = (a, b))
result
```

نکته : آرگمان args مربوط به ورودی های تابعی که قرار است انتگرال گیری شود ، می باشد.

✓ **مثال :** فرض کنید می خواهیم انتگرال تابع زیر را محاسبه کنیم.

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n}\, dt.$$

توسط پایتون و SciPy می توانیم به شکل زیر حل کنیم :

```
import numpy as np

def integrand(t, n, x):
    return np.exp(-x*t) / t**n

def expint(n, x):
    return integrate.quad(integrand, 1, np.inf, args=(n, x))[0]

vec_expint = np.vectorize(expint)

vec_expint(3, np.arange(1.0, 4.0, 0.5))
```

❖ **General Multiple Integration (انتگرال چندگانه) :**

برای گرفتن انتگرال دوگانه و سه گانه به ترتیب از دستورات dblquad و tplquad و نیز برای انتگرال گیری nگانه از nquad استفاده می شود.

نکته : کران های بالا و پایین انتگرال درونی باید حتما به صورت یک تابع تعریف شوند.

✓ **مثال :** فرض کنید می خواهیم انتگرال زیر را محاسبه کنیم.

$$I_n = \int_0^\infty \int_1^\infty \frac{e^{-xt}}{t^n}\, dt\, dx = \frac{1}{n}.$$

توسط پایتون و SciPy می توانیم به شکل زیر حل کنیم :

```
def I(n):
    return integrate.dblquad(lambda t, x: np.exp(-x*t)/t**n,
                             0, np.inf,
                             lambda x: 1, lambda x: np.inf)

I(4)
```

اگر بخواهیم همین مثال را با استفاده از دستور nquad حل کنیم داریم :

```
N = 5
def f(t, x):
   return np.exp(-x*t) / t**N

integrate.nquad(f, [[1, np.inf],[0, np.inf]])
```

روش های گوناگون دیگر نیز برای محاسبه انتگرال وجود دارد مانند Simpson و... .

Linear Algebra :

این Sub-package برای انجام کارهای جبرخطی ساخته شده است.

❖ SciPy.linalg vs NumPy.linalg :

اولا توابع جبرخطی SciPy به نسبت NumPy کامل تر است. بعضا به دلایل خاص که بسته به نحوه نصب NumPy دارد ممکن است کتابخانه SciPy سریعتر باشد.

❖ Finding the Inverse :

ماتریس وارون ، ماتریسی است که حاصلضرب یک ماتریس در وارونش برابر با ماتریس همانی شود. ماتریس همانی ماتریسی است که تمام المان های قطر اصلی آن برابر ۱ است و قطر فرعی آن برابر صفر است.

✓ **مثال :** فرض کنید می خواهیم وارون ماتریس زیر را پیدا کنیم.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$$

```
from scipy import linalg

A = np.array([[1,3,5],[2,5,1],[2,3,8]])

A_1 = linalg.inv(A)

A.dot(A_1)
```

❖ **Solving a Linear System :**

برای حل یک دستگاه معادلات می توان از روش زیر استفاده کرد.

✓ **مثال :** فرض کنید دستگاه معادلات زیر را می خواهیم حل کنیم.

$$\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}$$

برای حل این سوال دو روش وجود دارد.

۱- روش اول : استفاده از حاصل ضرب وارون ماتریس ضرایب در ماتریس ثابت ها.

```
A = np.array([[1, 2, 2], [3, 5, 3], [5, 1, 8]])

b = np.array([[10], [8], [3]])

print(linalg.inv(A).dot(b))

print(A.dot(linalg.inv(A).dot(b)) - b)
```

۲- روش دوم : استفاده از دستور solve.

```
print(np.linalg.solve(A, b))

print(A.dot(np.linalg.solve(A, b)) - b)
```

نکته : روش دوم به نسبت روش اول سریعتر و دقیق تر است.

❖ Finding the Determinant :

دترمینان ماتریس مربعی A را به شکل |A| نمایش می دهند. برای بدست آوردن مقدار دترمینان از فرمول زیر می توان استفاده کرد.

$$|\mathbf{A}| = \sum_j (-1)^{i+j} a_{ij} M_{ij}$$

✓ **مثال :** فرض کنید دترمینان ماتریس زیر را می خواهیم بدست بیاوریم.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}$$

جواب این مسئله به صورت ریاضیاتی به شکل زیر است.

$$\begin{aligned} |\mathbf{A}| &= 1 \begin{vmatrix} 5 & 1 \\ 3 & 8 \end{vmatrix} - 3 \begin{vmatrix} 2 & 1 \\ 2 & 8 \end{vmatrix} + 5 \begin{vmatrix} 2 & 5 \\ 2 & 3 \end{vmatrix} \\ &= 1\,(5 \cdot 8 - 3 \cdot 1) - 3\,(2 \cdot 8 - 2 \cdot 1) + 5\,(2 \cdot 3 - 2 \cdot 5) = -25. \end{aligned}$$

و جواب این سوال با استفاده از SciPy به شکل زیر است.

```
A = np.array([[1,3,5],[2,5,1],[2,3,8]])

linalg.det(A)
```

❖ Special Matrices :

کتابخانه SciPy شامل برخی از ماتریس های خاص می باشد.

Type	Function	Description
block diagonal	`scipy.linalg.block_diag`	Create a block diagonal matrix from the provided arrays.
circulant	`scipy.linalg.circulant`	Create a circulant matrix.
companion	`scipy.linalg.companion`	Create a companion matrix.
convolution	`scipy.linalg.convolution_matrix`	Create a convolution matrix.
Discrete Fourier	`scipy.linalg.dft`	Create a discrete Fourier transform matrix.
Fiedler	`scipy.linalg.fiedler`	Create a symmetric Fiedler matrix.
Fiedler Companion	`scipy.linalg.fiedler_companion`	Create a Fiedler companion matrix.
Hadamard	`scipy.linalg.hadamard`	Create an Hadamard matrix.
Hankel	`scipy.linalg.hankel`	Create a Hankel matrix.
Helmert	`scipy.linalg.helmert`	Create a Helmert matrix.
Hilbert	`scipy.linalg.hilbert`	Create a Hilbert matrix.

Inverse Hilbert	`scipy.linalg.invhilbert`	Create the inverse of a Hilbert matrix.
Leslie	`scipy.linalg.leslie`	Create a Leslie matrix.
Pascal	`scipy.linalg.pascal`	Create a Pascal matrix.
Inverse Pascal	`scipy.linalg.invpascal`	Create the inverse of a Pascal matrix.
Toeplitz	`scipy.linalg.toeplitz`	Create a Toeplitz matrix.
Van der Monde	`numpy.vander`	Create a Van der Monde matrix.

Optimization :

این Sub-package برای انجام کارهای بهینه سازی ساخته شده است.

❖ Root Finding :

۱- Scalar Functions :

اگر یک معادله تک متغیره داشته باشیم چندین الگوریتم ریشه یابی مختلف برای حل معادله وجود دارد. به طور کلی استفاده از Brentq بهترین انتخاب است. ممکن است روش های دیگر مانند Newton و... در شرایط خاص نیز مفید باشند.

۲- Sets of Equations :

به واسطه متد root از SciPy می توان ریشه معادلات غیرخطی را بدست آورد.

✓ **مثال :** فرض کنید می خواهیم معادله زیر را حل کنیم.

$$x + 2\cos(x) = 0$$

با استفاده از SciPy داریم :

```
from scipy.optimize import root
def func(x):
    return x + 2 * np.cos(x)
sol = root(func, 0.3)
sol.x
```

دستور root دو آرگمان می گیرد که اولین آرگمان یک تابع است و دومین آرگمان یک حدس اولیه برای ریشه معادله می باشد.

❖ Minimize :

تابع شامل منحنی یا منحنی هایی می باشد که نقاطی با بالاترین مقدار و پایین ترین مقدار دارد. بالاترین مقدار را ماکزیمم و پایین ترین مقدار را مینیمم می نامند.
اگر از آن نقطه بالاتر در سرتاسر منحنی نباشد ماکزیمم کلی (Global Max) می گویند و همچنین برای مینیمم پایین ترین باشد مینیمم کلی می گویند و اگر نقطه مد نظر فقط نسبت به اطراف خود در منحنی بالاتر باشد و مقداری در کل نمودار باشد که از آن بالاتر باشد ماکزیمم محلی (Local Max) می گویند و همچنین برای مینیمم نیز به همین شکل.

✓ **مثال :** فرض کنید می خواهیم مینیمم تابع زیر را بدست بیاوریم.

$$f(x) = x^2 + x + 2$$

با استفاده از SciPy داریم :

```
from scipy.optimize import minimize
def F(x):
    return x**2 + x + 2
sol = minimize(F, 0, method = "BFGS")
sol.x
```

متدهای مختلفی برای پیدا کردن مینیمم یک تابع وجود دارد.