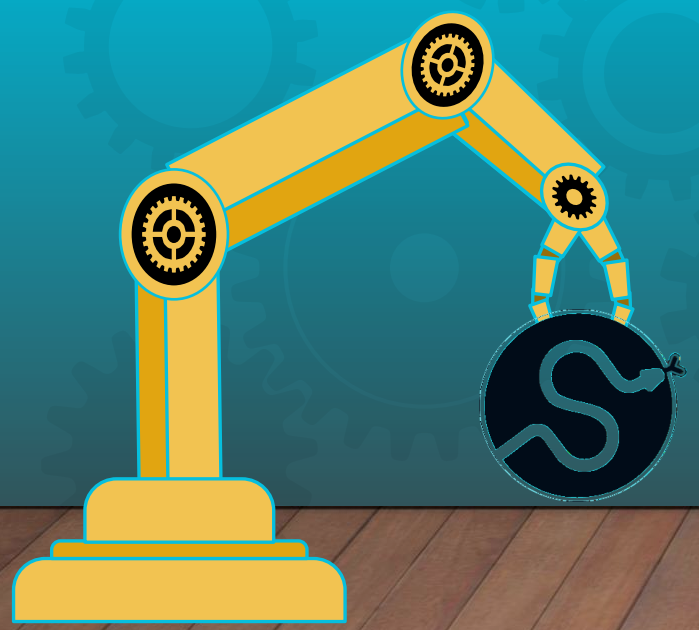
SciPy



Content Table

SciPy چیست

پکیج **SciPy** (سای پای) یک پکیج علمی و اوپن سورس مبتنی بر زبان پایتون است و برای انجام محاسبات علمی و مهندسی مورد استفاده قرار می‌گیرد. کتابخانه‌ی **SciPy** بر مبنای کتابخانه‌ی **NumPy** است و امکان کار با آرایه‌های **n** بُعدی را فراهم می‌کند. این کتابخانه برای کار با آرایه‌های **Numpy** ایجاد شده است و بسیاری از عملیات محاسباتی و بهینه‌سازی را به طور کارا ممکن می‌کند. هر دو پکیج **Numpy** و **Scipy** روی سیستم عامل‌های موجود کار می‌کنند و به راحتی نصب می‌شوند. در این ارائه به آموزش کتابخانه‌ی **SciPy** در پایتون خواهیم پرداخت و در پایان شما متوجه خواهید شد که **SciPy** چیست و چگونه میتوانید از ان استفاده کنید.

Sub-package در SciPy

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

SciPy زیر پکیج‌هایی دارد که هرکدام به منظور پوشش اعمال محاسباتی در زمینه‌ی خاصی توسعه یافته‌اند. این زیرپکیج‌ها در جدول روبهرو با توضیح مختصری از کارکرد آن‌ها آمده است.

هر کدام از این زیر مجموعه ها همانند یک کتابخانه مجزا کد و داکیومنت دارند و طی سال ها توسعه این زیر مجموعه های به کتابخانه سای پای اضافه شدند و این کتابخانه را به یک کتابخانه جامع تبدیل کردند.

ساختار داده‌های SciPy

ساختار داده‌ی پایه‌ای که در **SciPy** مورد استفاده قرار گرفته آرایه‌های چند بُعدی در **NumPy** است. **NumPy** برخی توابع را برای کار با جبرخطی، تبدیل فوریه و تولید اعداد تصادفی فراهم کرده‌است. اما این توابع به گستردگی توابع ارائه شده‌ی متناظر در پکیج **SciPy** نیست. همانطور که در کتابخانه های قبلی دیدیم کتابخانه **NumPy** نقش مهمی در توسعه اکثر کتابخانه ها دارد و بیشتر کتابخانه ها از نیازمند این کتابخانه برای پردازش های پایه ریاضیاتی هستند.

نصب SciPy

به صورت کلی اکثر کتابخانه های پایتون با دستور pip نصب میشوند و کتابخانه Scipy نیز با همین روش نصب میشود. برای نصب کتابخانه از دستور زیر استفاده میکنیم

```
pip install pandas
```

اگر ما از توزیع پکیج آناکوندا استفاده کنیم، Scipy به طور پیش‌فرض در آن نصب شده است و نیازی به نصب مجدد ان نیست.

توابع پایه‌ای در SciPy

به طور کلی توابع NumPy در پکیج Scipy موجود است و نیازی به وارد کردن کتابخانه‌ی NumPy نیست. از مهم‌ترین توابع موجود درnumpy آرایه‌های چند بعدی یک‌دست است. این آرایه‌ها مشابه جدولی از اجزائی (عموماً عدد) است که همگی نوع یکسان دارند و اندیس‌های عددی صحیح و مثبت دارند.

درNumpy به ابعاد داده محور axes گفته می‌شود و تعداد محورها مرتبه‌ی آرایه یا rank نامیده می‌شود. در ادامه بردارها و ماتریس‌های عددی درNumPy را به صورت سریع و کلی مرور می‌کنیم. از آنجایی کهScipy روی Numpy بنا شده است.

توابع پایه‌ای در SciPy

یک بردار در NumPy می‌تواند به چند طریق مختلف ایجاد شود. برخی از این روش‌ها در زیر آمده است: تبدیل لیست پایتون به آرایه

مثال زیر را در نظر بگیرید:

```
import numpy as np
list = [1,2,3,4]
arr = np.array(list)
print(arr)
```

توابع پایه‌ای در SciPy

```
import numpy as np
list = [1,2,3,4]
arr = np.array(list)
print(arr)
```

خروجی کد ما به شکل زیر است:

```
[1,2,3,4]
```

در Numpy توابع داخلی برای ایجاد آرایه‌ها ساخته شده است. برخی از این توابع در صفحات بعدی مورد بررسی قرار میگیرد.

توابع پایه‌ای در SciPy

ماتریس صفر

تابع **zeros** به شکل **zeros(shape)** آرایه‌ای است که با مقادیر صفر به تعداد **shape** تعریف شده، پر شده است. نوع پیش‌فرض **float64** بیتی است. مثال زیر را در نظر بگیرید:

```
import numpy as np
print( np.zeros((2, 3)) )
```

خروجی کد بالا آرایه‌ای به شکل زیر است:

```
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
```

توابع پایه‌ای در SciPy

ماتریس

یک ماتریس آرایه‌ای دوبعدی است که در طی انجام انواع عملیات ماهیت دو بعدی خود را حفظ می‌کند. نحوه‌ی تعریف ماتریس به شکل زیر است:

```
import numpy as np
Print( np.matrix('1 2; 3 4') )
```

خروجی، ماتریسی به شکل زیر است:

```
matrix([[1, 2],
        [3, 4]])
```

توابع پایه‌ای در SciPy

به شکل زیر نیز می‌توان ترانهاده‌ی ماتریس را محاسبه کرد:

```
import numpy as np
mat = np.matrix('1 2; 3 4')
print(mat.H)
```

خروجی، ماتریسی به شکل زیر است:

```
matrix([[1, 3],
        [2, 4]])
```

همانطور که در مثال بالا و مثال های قبلی دیدید خیلی از ساختار های پایه نامپای در این کتابخانه هم استفاده میشوند و قدرت سای پای در به کارگیری این ساختار ها برای پرداز های پیچیده است.

خوشه‌بندی یا Clustering

خوشه‌بندی **K-means** روشی برای یافتن خوشه‌ها و مراکز آن‌ها در یک مجموعه داده‌ی بدون برچسب است. خوشه‌ها مجموعه‌ای از نقاط داده‌ای هستند که مشابه همدیگرند و فاصله‌ی میان آن نقطه داده‌ها با همدیگر کمتر از سایر خوشه هست. با **K** مرکز خوشه‌ای داده شده، الگوریتم **K-means** دو مرحله‌ی زیر را تکرار می‌کند

برای هر مرکز، زیرمجموعه‌ای از نقاط آموزشی که به آن مرکز، نسبت به سایر مراکز نزدیک‌ترند، انتخاب می‌شوند. میانگین مقدار هر ویژگی برای هر مرکز، براساس مقادیر نقاط داده برای آن ویژگی محاسبه می‌شود و بردار حاصل از میانگین ویژگی‌ها در هر خوشه، به عنوان مرکز جدید خوشه‌ای در نظر گرفته می‌شود.

این دو مرحله تا زمانی که مقادیر مراکز خوشه تغییر چندانی نکند، تکرار می‌شوند. در نهایت هر نمونه داده‌ی جدید مانند **x**، به خوشه‌ی مشابه‌اش تعلق می‌گیرد.

خوشه‌بندی یا Clustering

اجرای K-means با SciPy

کتابخانه‌ی SciPy امکان اجرای بهینه‌ی الگوریتم k-means را با پکیج خوشه‌بندی فراهم می‌کند.
وارد کردن K-means نحوه‌ی وارد کردن و استفاده از تابع k-means به شکل زیر است:

```
from SciPy.cluster.vq import kmeans,vq,whiten
```

تولید داده برای اجرای خوشه‌بندی، به داده‌هایی نیاز داریم. می‌توانیم این داده‌ها را به شکل زیر ایجاد کنیم:

```
from numpy import vstack,array
from numpy.random import rand
# data generation with three features
data = vstack((rand(100,3) + array([.5,.5,.5]),rand(100,3)))
```

خوشه‌بندی یا Clustering

اجرای K-means با SciPy

```
from numpy import vstack,array
from numpy.random import rand
# data generation with three features
data = vstack((rand(100,3) + array([.5,.5,.5]),rand(100,3)))
```

کد بالا داده‌هایی به شکل زیر تولید می‌کند:

```
array([[ 1.48598868e+00, 8.17445796e-01 , 1.00834051e+00],
       [ 8.45299768e-01  , 1.35450732e+00, 8.66323621e-01],
       [ 1.27725864e+00 , 1.00622682e+00, 8.43735610e-01],. . . ]
```

خوشه‌بندی یا Clustering

اجرای K-means با SciPy

در خوشه‌بندی به کمک Kmeans بهتر است داده‌های مربوط به هر ویژگی را نرمال‌سازی کنیم. داده‌ها اگر در مقیاس مشابهی باشند و پراکندگی یکسانی داشته باشند، خوشه‌بندی کاراتر است.

پاک‌سازی داده برای این‌کار کد زیر را اجرا می‌کنیم:

```
#whitening of data
data = whiten(data)
```

محاسبه‌ی kmeans با سه خوشه

برای تقسیم داده‌ها به سه خوشه کد زیر را اجرا می‌کنیم:

```
#computing K-Means with K = 3 (3 clusters)
centroids,_ = kmeans(data,3)
```

خوشه‌بندی یا Clustering

محاسبه‌ی kmeans با سه خوشه

```
centroids,_ = kmeans(data,3)
```

کد بالا الگوریتم kmeans را روی مجموعه داده‌ها اجرا کرده و سه خوشه تولید می‌کند. الگوریتم kmeans مراکز خوشه‌ها را تا زمان رسیدن به مقدار مناسب محاسبه می‌کند. تغییر مقادیر خوشه‌ها تا مرحله‌ای تکرار می‌شود که تفاوت مقدار خوشه در آن مرحله با مرحله‌ی قبلی‌اش از یک مقدار آستانه کمتر باشد. می‌توانیم مقدار مراکز خوشه‌ها را که در متغیر centroids قرار داده شده است را به کمک کد زیر به دست آوریم:

```
print(centroids)
```

مراکز سه خوشه‌ی بالا سه بردار زیر است که هر بردار سه مقدار دارد که همان تعداد ویژگی‌های داده‌های ما هستند.

خوشه‌بندی یا Clustering

محاسبه‌ی **kmeans** با سه خوشه

هر داده‌ی جدید را می‌توان با کد زیر به خوشه‌ی مناسبش، اختصاص داد.

```
#assign each sample to a cluster
clx,_ = vq(data,centroids)
```

تابع **vq** هر بردار داده را با مراکز خوشه‌ها مقایسه می‌کند و به نزدیک‌ترین خوشه اختصاص می‌دهد. این تابع خوشه‌ای را که داده به آن تعلق می‌گیرد را، برمی‌گرداند. با دستور زیر می‌توانیم چگونگی خوشه‌بندی داده‌ها و اینکه متعلق به کدام یک از سه خوشه‌ی ۰، ۱ یا ۲ هستند را ببینیم.

خوشه‌بندی یا Clustering

محاسبه‌ی kmeans با سه خوشه

```
#check clusters of observation
print (clx)
```

خروجی کد بالا به شکل زیر است:

```
array([1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 2, 0, 2, 0, 1, 1,
1,0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...], dtype=int32)
```

همانطور که در خروجی بالا دیدید خروجی شامل خوشه بندنی مجموعه عظیمی از عددهاست

مقادیر ثابت در SciPy

پکیج scipy.constantمقادیر ثابت متعددی را می‌تواند تولید کند. باید مقدار ثابت درخواستی را با دستور importوارد کنیم و سپس از آن استفاده کنیم. برای مثال عدد پی (Piرا به شکل زیر استفاده می‌کنیم:

```
#Import pi constant from both the packages
from scipy.constants import pi
from math import pi
print("sciPy - pi = %.16f"%scipy.constants.pi)
print("math - pi = %.16f"%math.pi)
```

مقادیر ثابت در SciPy

```
#Import pi constant from both the packages
from scipy.constants import pi
from math import pi
print("sciPy - pi = %.16f"%scipy.constants.pi)
print("math - pi = %.16f"%math.pi)
```

خروجی کد بالا برای هر دو کتابخانه‌ی scipy و mathبه شکل زیر است:

```
sciPy - pi = 3.1415926535897931
math - pi = 3.1415926535897931
```

مقادیر ثابت در SciPy

لیست مقادیر ثابت موجود

مقادیر ثابت فراوانی در ریاضیات و فیزیک وجود دارد. همچنین اختصارات زیادی وجود دارد که همیشه معنی یکسانی را می‌رساند. به طور مثال واحدهای تعریف شده برای جرم و طول را با kg و inch نشان می‌دهند. به خاطر سپاری اختصارات مربوط به مقادیر ثابت کار سختی است. برای اینکه اطلاعات بیشتر در این زمینه کسب کنید و بدانید که اختصارات و علائم تعریف شده در SciPy چیست، به document آن مراجع کنید. راه حل آسان برای فهمیدن اینکه هر کلمه‌ی کلیدی در SciPy چیست و به چه مقدار ثابتی اختصاص یافته است، به کارگیری متد scipy.constant.find() است.

مقادیر ثابت در SciPy

به مثال زیر توجه کنید:

```
import scipy.constants
res = scipy.constants.physical_constants["alpha particle mass"]
print(res)
```

برنامه‌ی بالا خروجی زیر را تولید می‌کند که لیستی از کلیدهای منطبق با کلید مورد جستجوی ماست.

```
['alpha particle mass','alpha particle mass energy equivalent',
'alpha particle mass energy equivalent in MeV',
'alpha particle mass in u',
'electron to alpha particle mass ratio' ]
```

تبدیل فوریه FFTPack

تبدیل فوریه محاسباتی به منظور تبدیل تابع از دامنه‌ی زمانی به دامنه‌ی فرکانسی است و رفتار تابع را در حوزه‌ی فرکانسی بررسی می‌کند. تبدیل فوریه در مواردی همچون پردازش سیگنال و نویز، پردازش تصویر، پردازش سیگنال صوتی و... کاربرد دارد.**SciPy** ماژول **FFTPack** که نام کامل ان میشود **(Fast Fourier Transform)** را برای محاسبه‌ی تبدیل فوریه به کار می‌گیرد و به کاربر امکان تبدیل فوریه از طریق الگوریتم تبدیل سریع فوریه را می‌دهد.

در اسلاید های بعدی مثالی از یک تابع سینوسی را که تبدیل فوریه را با استفاده از ماژول **FFTPack** انجام می‌دهد، و فرایند ان را مورد بررسی قرار میدهیم.

تبدیل فوریه FFTPack

تعریف تبدیل سریع فوریه:

تبدیل سریع فوریه را با جزییات بیشتری شرحی می‌دهیم. تبدیل فوریه گسسته یکی از تکنیک‌های ریاضی و محاسباتی است که می‌تواند داده‌ها را به داده‌هایی در محدوده‌ی فرکانسی تبدیل کند. تبدیل فوریه سریع یا همان **FFT** یکی از روش‌های محاسبه برای تبدیل فوریه‌ی گسسته است.

FFT می‌تواند روی آرایه‌های چند بعدی اعمال شود. این نکته را یادآور می‌شویم که منظور از فرکانس، تعداد سیگنال یا طول موج در یک بازه‌ی زمانی مشخص می‌باشد.

تبدیل فوریه FFTPack

تبدیل فوریه‌ی گسسته‌ی یک بعدی

تابع خروجی **y** نتیجه‌ی اعمال تبدیل فوریه‌ی سریع،یا همان تابع **fft** بر روی ورودی **x** است و طول دامنه در هر دو تابع **x** و **y** یکسان است.

```
#Importing the fft and inverse fft functions from fftpackage
from scipy.fftpack import fft
#create an array with random n numbers
x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
#Applying the fft function
y = fft(x)
print(y)
```

تبدیل فوریه FFTPack

خروجی کد بالا به شکل زیر است:

```
[2.08155948-1.65109876j , -.۰+۴٫۵۰۰۰۰۰۰۰j
1.83155948+1.60822041j ,-1.83155948-1.60822041j ,
2.08155948+1.65109876j ]
```

برای محاسبه‌ی معکوس تبدیل فوریه از ifft() استفاده می‌کنیم:

```
#FFT is already in the workspace, using the same workspace to for
inverse transform
yinv = ifft(y)
print(yinv)
```

خروجی این کد به شکل زیر است:

```
[ 1.0+0.j   2.0+0.j   1.0+0.j   -1.0+0.j   1.5+0.j ]
```

تبدیل فوریه FFTPack

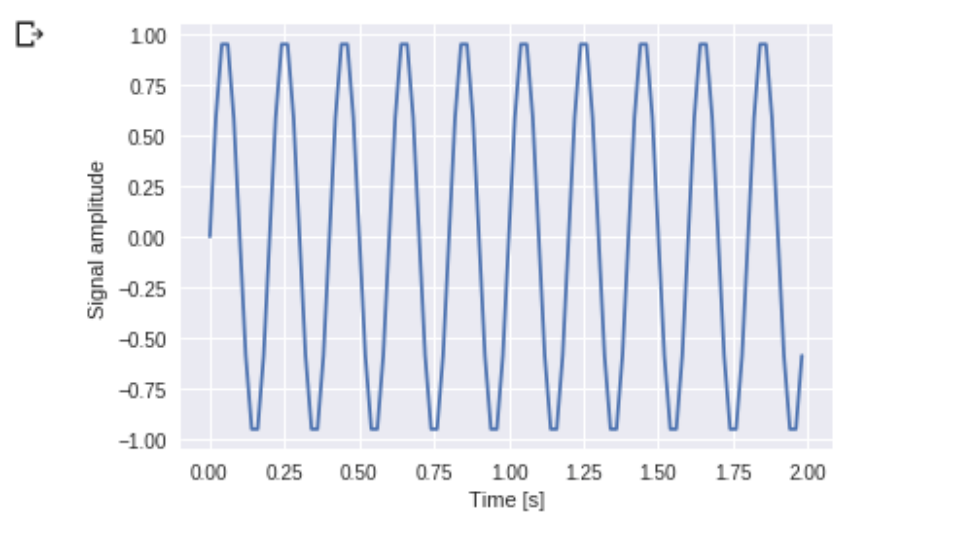
حال مثال کاربردی‌تری را باهم بررسی می‌کنیم. به تابع زیر که به کمک کتابخانه‌ی matplotlib آن را ترسیم کرده‌ایم، توجه کنید. شکل حاصل نشانگر تابع متناوب سینوسی $\sin(20 \times 2\pi t)$ است.

```
from matplotlib import pyplot as plt
import numpy as np
fre  = 5  #Frequency in terms of Hertz
fre_samp = 50 #Sample rate
t = np.linspace(0, 2, 2 * fre_samp, endpoint = False )
a = np.sin(fre  * 2 * np.pi * t)
```

تبدیل فوریه FFTPack

```
figure, axis = plt.subplots()
axis.plot(t, a)
axis.set_xlabel('Time (s)')
axis.set_ylabel('Signal amplitude')
plt.show()
```

با استفاده از کد بالا خروجی تبدیل را نمایش میدهیم. همان طور که می‌بینید فرکانس ۵ هرتز است و سیگنال در ۱/۵ ثانیه تکرار می‌شود که دوره‌ی زمانی تناوب این سیگنال است.:



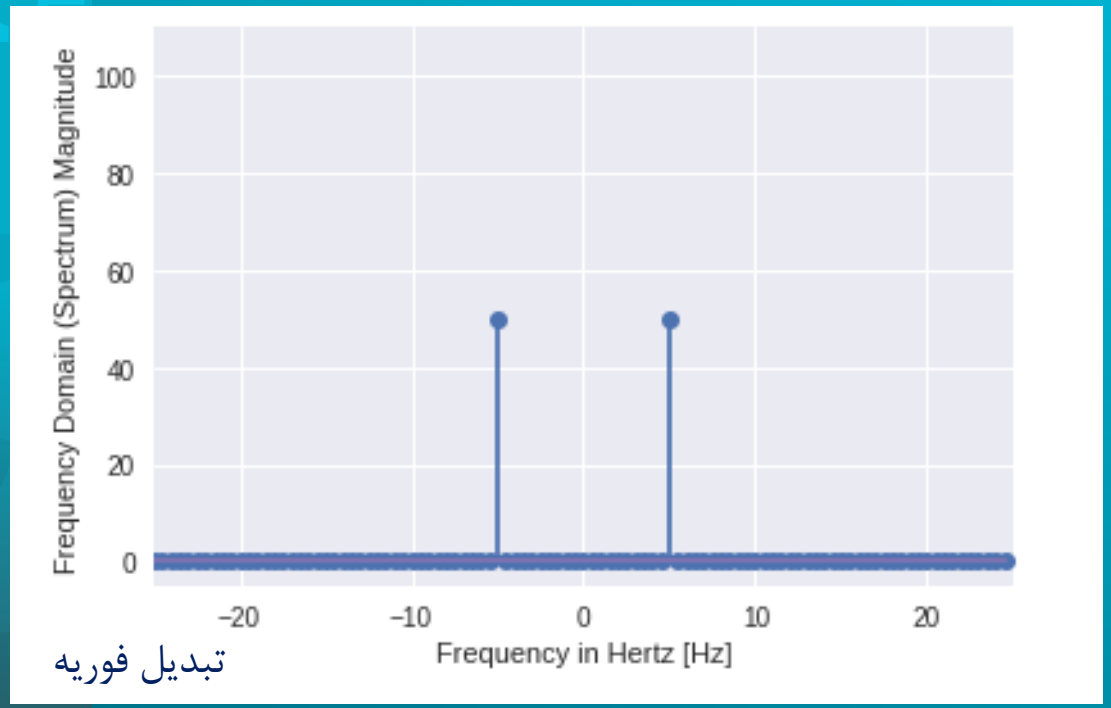
تبدیل فوریه FFTPack

حال به کمک ابزار ارئه شده در scipy از این موج سینوسی برای تبدیل فوریه استفاده می‌کنیم:

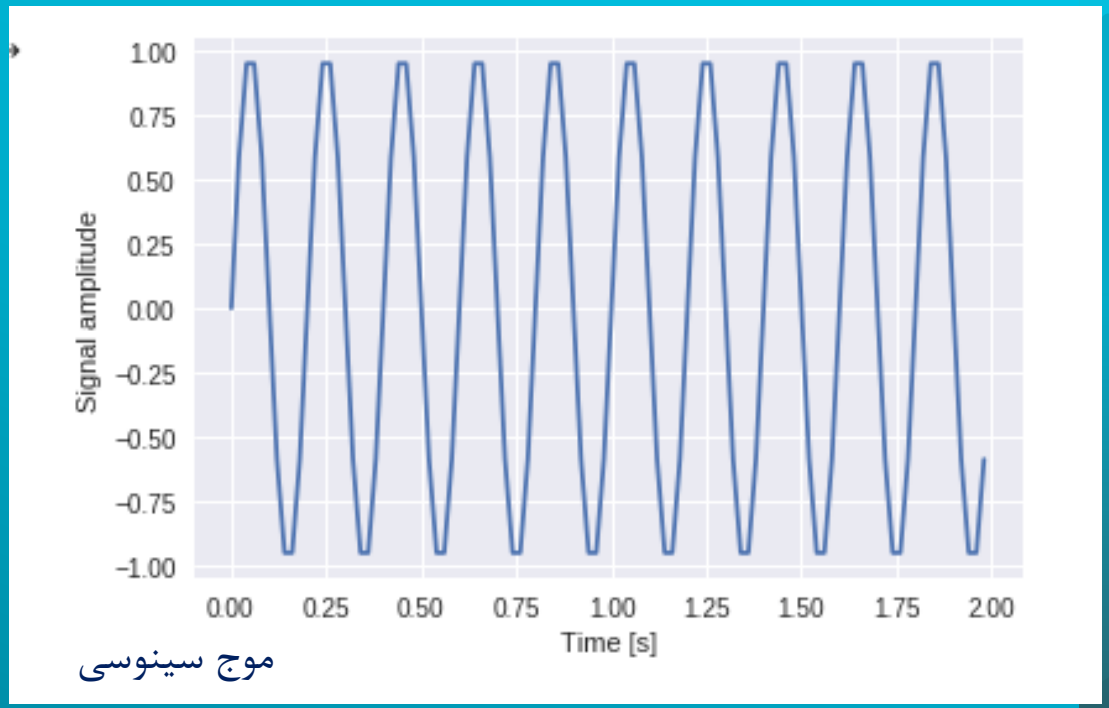
```
from scipy import fftpack
A = fftpack.fft(a)
frequency = fftpack.fftfreq(len(a)) * fre_samp
figure, axis = plt.subplots()
axis.stem(frequency, np.abs(A))
axis.set_xlabel('Frequency in Hz')
axis.set_ylabel('Frequency Spectrum Magnitude')
axis.set_xlim(-fre_samp / 2, fre_samp/ 2)
axis.set_ylim(-5, 110)
plt.show()
```

تبدیل فوریه FFTPack

خروجی قبلی در سمت راست و خروجی نهایی در سمت چپ قرار دارد:



تبدیل فوریه



موج سینوسی

نحوه‌ی کار با ورودی و خروجی در SciPy

پکیج **scipy.io** توابع بسیار گسترده‌ای را برای کار با انواع مختلفی از فرمت‌های داده در اختیار می‌گذارد. داده‌ها می‌توانند فرمت‌هایی مانند **Matlab**، **IDL**، **MatrixMarket**، **Wave**، **Arff**، **Netcdf** و غیره داشته باشند. در ادامه در خصوص داده‌ی **Matlab** جزییات بیشتری را شرح می‌دهیم.

در جدول زیر توابع به کار رفته برای بارگذاری و ذخیره‌ی یک فایل متلب **.mat** را می‌بینید:

Sr. No.	Function & Description
1	**loadmat** Loads a MATLAB file
2	**savemat** Saves a MATLAB file
3	**whosmat** Lists variables inside a MATLAB file

نحوه‌ی کار با ورودی و خروجی در SciPy

مثال زیر را در نظر بگیرید:

```
import scipy.io as sio
import numpy as np
#Save a mat file
vect = np.arange(10)
sio.savemat('array.mat', {'vect':vect})
#Now Load the File
mat_file_content = sio.loadmat('array.mat')
Print(mat_file_content)
```

نحوه‌ی کار با ورودی و خروجی در SciPy

خروجی دستورات بالا به شکل زیر است که آرایه‌ی ورودی را به همراه متادیتا، نمایش می‌دهد.

```
{
'vect'          : array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]),
'__version__' : '1.0',
'__header__' : 'MATLAB 5.0 MAT-file Platform: posix, Created on: Sat Sep 30 09:49:32 2017',
'__globals__' : []
}
```

نحوه‌ی کار با ورودی و خروجی در SciPy

اگر بخواهیم محتویات فایل متلب را بدون خواندن داده‌ها در حافظه بررسی کنیم از دستور whosmat به شکل زیر استفاده می‌کنیم:

```
import scipy.io as sio
mat_file_content = sio.whosmat('array.mat')
print(mat_file_content)
```

کد بالا خروجی زیر را تولید می‌کند:

```
[('vect', (1, 10), 'int64')]
```

جبر خطی در SciPy

جبر خطی:

SciPy با استفاده از کتابخانه‌های **ATLASLAPACK** و**BLAS** ساخته شده است و قابلیت‌های بسیار خوبی در زمینه‌ی جبر خطی دارد. در تمامی این روال‌های جبر خطی، انتظار می‌رود که هر شی‌ای قابل تبدیل به یک آرایه‌ی دو بعدی باشد و خروجی تمامی این روال‌ها نیز آرایه‌ای دو بعدی است.

جبر خطی در SciPy

تفاوت جبر خطی در NumPy و SciPy چیست؟

کتابخانه‌ی SciPy شامل تمامی موارد و توابع جبر خطی تعریف شده در NumPy است. علاوه بر آن جبر خطی در SciPy که آن را با scipy.linalg فراخوانی می‌کنیم، توابع پیشرفته‌ای دارد که در NumPy موجود نیست. کتابخانه‌ی SciPy همواره با پشتیبانی BLAS/LAPACK، کامپایل می‌شود در حالی که این گزینه در NumPy اختیاری است. در نتیجه بسته به نحوه‌ی نصب NumPy، ممکن است نسخه‌ی SciPy سریع‌تر باشد.

جبر خطی در SciPy

معادلات خطی

برای حل معادله‌ی a*x+b*y=z به ازای مقادیر مجهول x و y از ویژگی scipy.linalg.solve استفاده می‌کنیم. برای مثال فرض کنید باید معادله‌ی زیر را حل کنید:

x + 3y + 5z=10
2x + 5y + z=8
2x + 3y + 8z= 3

برای حل معادله‌ی بالا برای مقادیر مجهول x، y و z ماتریس ضرایب را در متغیر a و ماتریس سمت راست معادلات را در متغیر b قرار می‌دهیم. متغیرهای a و b را به عنوان پارامتر به تابع linalg.solve ارسال می‌کنیم و از پاسخ معادله‌ی حل شده به کمک دستور print خروجی می‌گیریم.

جبر خطی در SciPy

```
from scipy import linalg
import numpy as np
#Declaring the numpy arrays
a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
b = np.array([10, 8, 3])
x = linalg.solve(a, b) #Passing the values to the solve function
print(x) #printing the result array
```

برنامه‌ی بالا خروجی زیر را برای متغیرهای **x** و **y** و **z** تولید می‌کند.

```
array([ 2., -2., 9.])
```

جبر خطی در SciPy

محاسبه‌ی دترمینان:

دترمینان ماتریس مربعی **A** به شکل **|A|** نمایش داده می‌شود و مقداری است که در بسیاری از محاسبات جبری به کار گرفته می‌شود. در پکیج **SciPy** با دستور **det()** این مقدار را محاسبه می‌کنیم. این تابع ماتریسی را به عنوان ورودی گرفته، و مقداری عددی را به عنوان خروجی محاسبه و برمی‌گرداند.

در صفحه بعد یک مثال برای حل یک مسئله توسط کد داریم که مورد بررسی قرار میدهیم.

جبر خطی در SciPy

به مثال زیر توجه کنید

```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np
A = np.array([[1,2],[3,4]]) #Declaring the numpy array
x = linalg.det(A) #Passing the values to the det function
print (x) #printing the result
```

خروجی برنامه به شکل زیر است.

```
-2.0
```

پردازش تصویر با Scipy

زیرمجموعه‌ی **scipy_ndimage** به پردازش تصویر اختصاص داده شده است و منظور از عبارت **ndimage**، تصویر **n** بعدی است. برخی از کارهای معمول که در پردازش تصویر انجام می‌گیرد، شامل موارد زیر است:

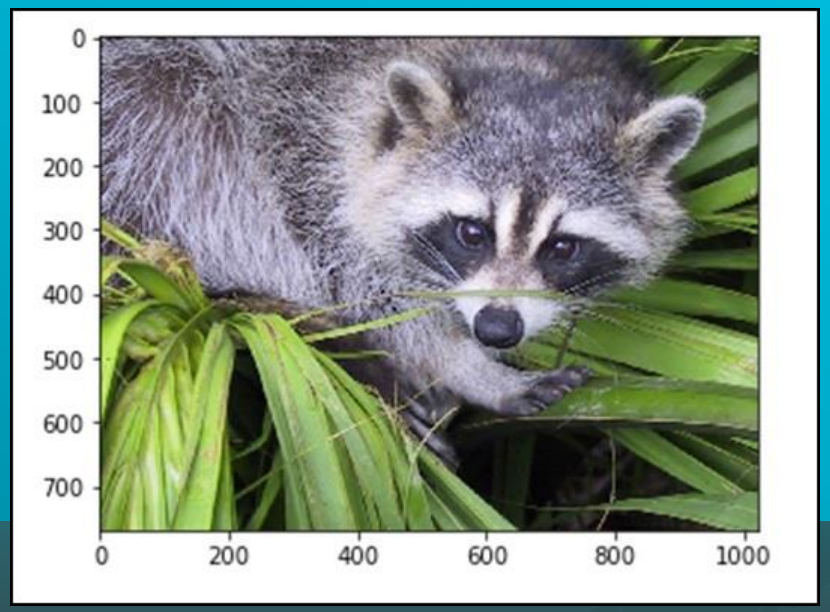
- ورودی/خروجی و نمایش تصویر
- دست‌کاری‌های اولیه در عکس مثل برش تصویر، چرخش، تقسیم و ...
- اعمال فیلترهای روی عکس مانند حذف نویز و **sharp** کردن و ...
- قطعه‌بندی تصاویر یا سگمنت کردن: برچسب‌گذاری پیکسل‌ها متناسب با آبجکت‌های مختلف در عکس
- طبقه‌بندی یا کلاس‌بندی
- استخراج ویژگی
- رجیستر کردن تصاویر

پردازش تصویر با Scipy

باز کردن و نوشتن در فایل‌های تصویری

پکیج miscدر Scipyمجموعه‌ای از عکس‌ها را با خود به همراه دارد. می‌توان از آن عکس‌ها برای آموزش کار با تصاویر و دست‌کاری و تغییر آن‌ها استفاده کرد.

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f)
import matplotlib.pyplot as plt
# plot the image
plt.imshow(f)
plt.show()
```



پردازش تصویر با Scipy

هر تصویری در فرمت خام خود ترکیبی از رنگ‌هاست که توسط اعداد در قالب ماتریس نمایش داده می‌شوند. کامپیوتر تنها بر اساس آن اعداد تصاویر را درک و دست‌کاری می‌کند. روش RGBجزو روش‌های بسیار پرکاربرد و مرسوم در نمایش داده‌هاست.

اطلاعات آماری در خصوص تصویر بالا با کد زیر قابل استخراج است:

```
from scipy import misc
face = misc.face(gray = False)
print(face.mean(), face.max(), face.min())
```

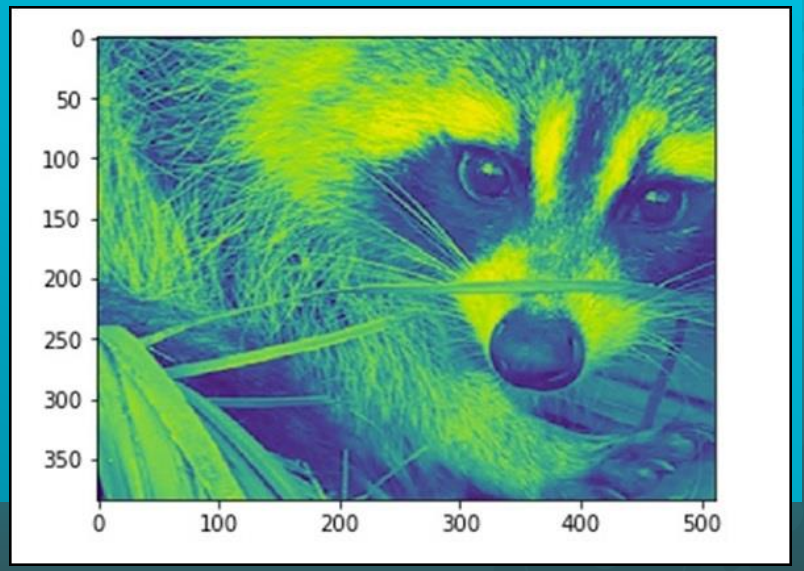
برنامه‌ی بالا خروجی زیر را تولید می‌کند که میانگین و ماکزیمم و مینیمم مقادیر از بردار RGB را استخراج می‌کند.

```
110.16,  255,  0
```

پردازش تصویر با Scipy

حال می‌دانیم که هر عکس از اعدادی ساخته شده است، پس هر تغییری در مقدار این اعداد، منجر به تغییر تصویر می‌شود. بیایید چند تغییر هندسی روی تصویر اعمال کنیم. از پایه‌ای‌ترین تغییرات هندسی روی تصاویر، برش یا **crop**کردن تصویر است. در کد زیر تصویر را از هر طرف به اندازه‌ی ۱/۴ برش می‌زنیم.

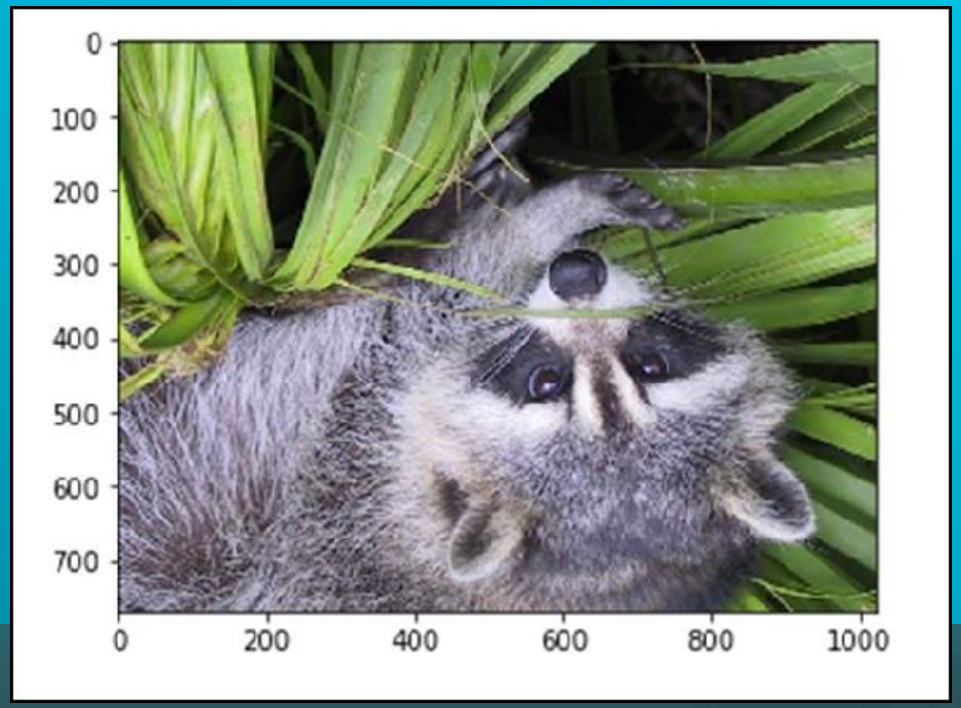
```
from scipy import misc
face = misc.face(gray = True)
lx, ly = face.shape # Cropping
crop_face = face[lx / 4: - lx / 4, ly / 4: - ly / 4]
import matplotlib.pyplot as plt
plt.imshow(crop_face)
plt.show()
```



پردازش تصویر با Scipy

می‌توانیم تصویر خود را در جهت بالا به پایین بچرخانیم. به کد زیر توجه کنید:

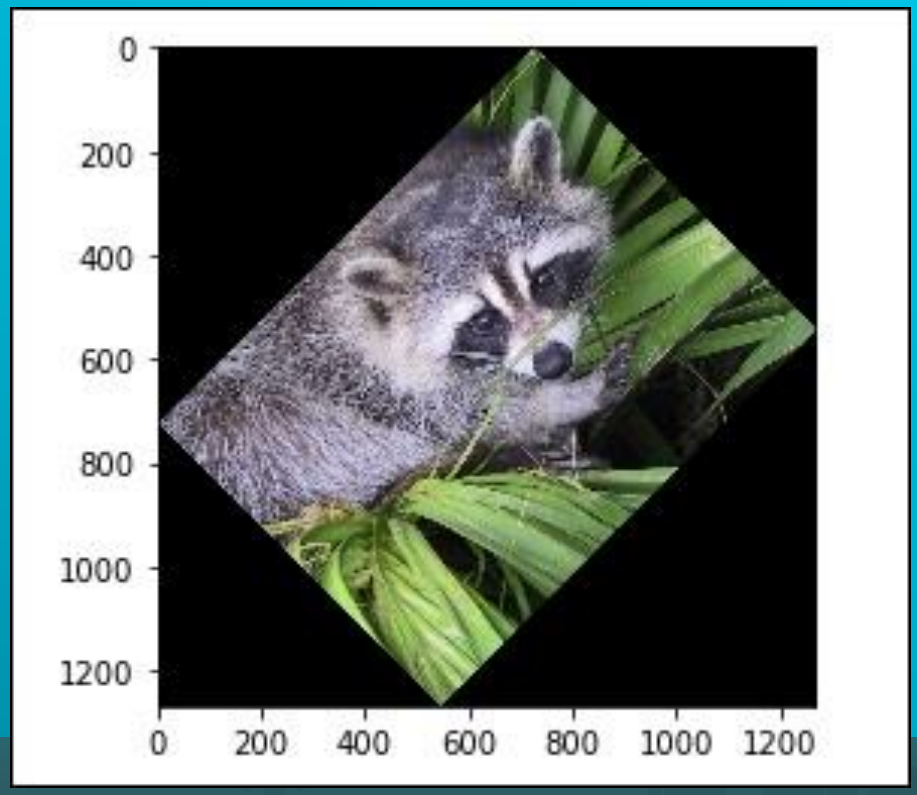
```
# up <-> down flip
from scipy import misc
face = misc.face()
flip_ud_face = np.flipud(face)
import matplotlib.pyplot as plt
plt.imshow(flip_ud_face)
plt.show()
```



پردازش تصویر با Scipy

همچنین از تابع rotate() می‌توان برای چرخاندن عکس با مقدار زاویه‌ی دلخواه استفاده کرد:

```
# rotation
from scipy import misc,ndimage
face = misc.face()
rotate_face = ndimage.rotate(face, 45)
import matplotlib.pyplot as plt
plt.imshow(rotate_face)
plt.show()
```



پردازش تصویر با Scipy

فیلترگذاری در تصاویر

بیایید در مورد اینکه فیلترها چگونه در پردازش تصویر به ما کمک می‌کنند، بحث کنیم. فیلترگذاری تکنیکی برای اصلاح یا تقویت یک تصویر است. برای مثال، می‌توانید تصویری را به منظور تاکید روی ویژگی‌های خاص یا حذف برخی ویژگی‌های دیگر، فیلتر کنید. عملیات پردازش تصویر که با فیلتر انجام می‌شود شامل صاف کردن (smoothing)، تیز کردن(sharpening) و تقویت لبه‌های تصویر (Edge Enhancement) است.

فیلترینگ تصاویر یک عملیات همسایگی(neighborhood operation) محسوب می‌شود، به این معنی که مقدار هر پیکسل در تصویر خروجی از طریق اعمال الگوریتم‌هایی روی پیکسل ورودی و پیکسل‌های موجود در همسایگی آن ورودی تعیین می‌شود. بگذارید اکنون عملیات جدیدی به کمک Scipy.ndimage انجام دهیم.

پردازش تصویر با Scipy

Bluring : محو و تارشدن تصویر

Blur کردن تصاویر به منظور کاهش داده‌ی نویز است. می‌توان فیلتر را اعمال و تغییرات را مشاهده کنید.

مثال زیر را در نظر بگیرید:

```
from scipy import misc
face = misc.face()
blurred_face = ndimage.gaussian_filter(face, sigma=3)
import matplotlib.pyplot as plt
plt.imshow(blurred_face)
plt.show()
```

پردازش تصویر با Scipy

تشخیص لبه‌های تصویر: Edge Detection

تشخیص لبه‌ی تصاویر تکنیکی در پردازش تصویر است که برای یافتن مرز و محدوده‌ی اشیا (Object) موجود در تصویر به کار می‌آید. این کار از طریق شناسایی ناپیوستگی‌ها در مقدار روشنایی پیکسل صورت می‌گیرد. تشخیص لبه برای قطعه‌بندی تصاویر و استخراج داده در مباحثی همچون پردازش تصویر، بینایی کامپیوتر و بینایی ماشین مورد استفاده قرار می‌گیرد.

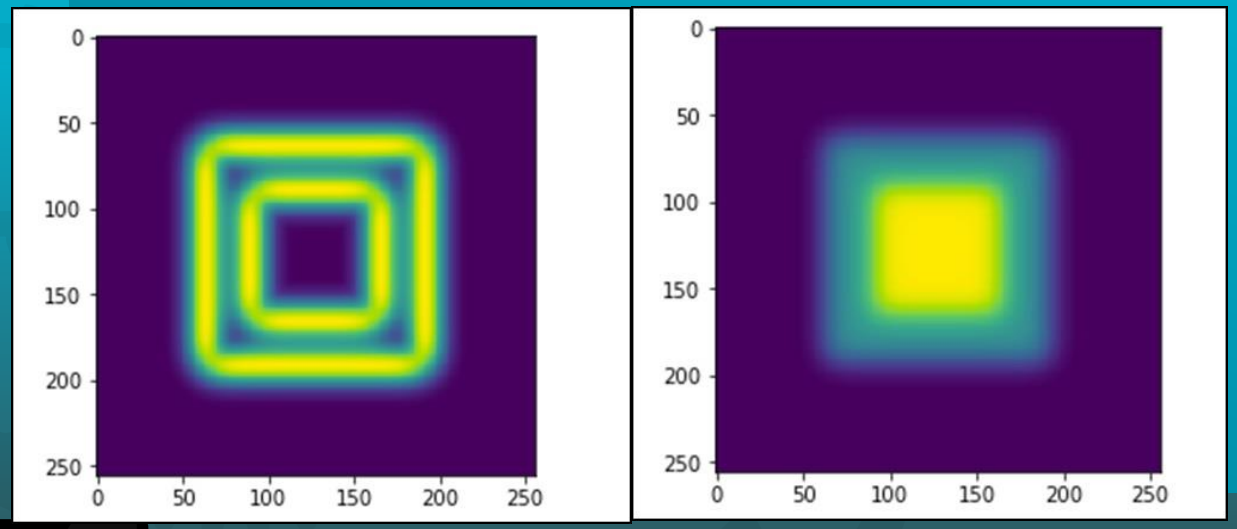
بیشترین الگوریتم‌های مورد استفاده در تشخیص لبه‌ی تصاویر عبارتند از:

- Sobel
- Canny
- Prewitt
- Roberts
- Fuzzy Logic Methods

پردازش تصویر با Scipy

تشخیص لبه‌های تصویر: Edge Detection

به مثال زیر توجه کنیدو که تصویر ورودی تصویر سمت راست و تصمیر خروجی بعد از پیاده سازی الگوریتم sobel تصویر سمت چپ است که در تصویر راست دو لبه داریم یکی برای مربع زرد وسط و دومی برای مربع کم رنگ اطراف در نتیجه بعد از پیاده سازی الگوریتم فقط دو لبه باقی می مانند. و بقیه بخش ها حذف میشود.

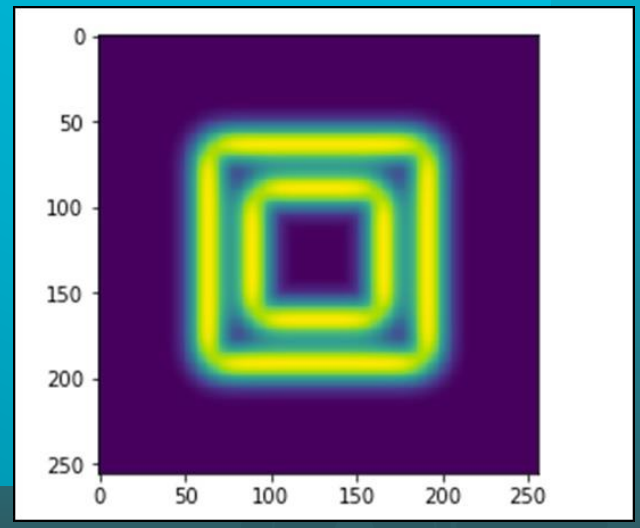


پردازش تصویر با Scipy

تشخیص لبه‌های تصویر: Edge Detection

کد مثال صفحه قبل در پایین قرار دارد:

```
import scipy.ndimage as nd
import matplotlib.pyplot as plt
.....
sx = ndimage.sobel(im, axis = 0, mode = 'constant')
sy = ndimage.sobel(im, axis = 1, mode = 'constant')
sob = np.hypot(sx, sy)
plt.imshow(sob)
plt.show()
```



نتیجه گیری

همانطور که دیدید کتابخانه **SciPy** یکی از بهترین ابزار ها برای پردازش های ریاضیاتی است و همین قابلیت قدرست بسیار زیادی به این کتابخانه برای کار کردن با داده های جدولی و چند بعدی مثل پردازش تصویر میدهد و این کتابخانه یکی از مهم ترین ابزار در زمینه هوش مصنوعی محسوب میشود.

SciPy

Thank You