Пакет специальных функций

- **scipy.специальный** пакет содержит многочисленные функции математической физики.
- Специальная функция SciPy включает кубический корень, экспоненту, экспоненту логарифмической суммы, Ламберта, перестановку и комбинации, гамму, Бесселя, гипергеометрическую, Кельвина, бета, параболический цилиндр, экспоненту относительной ошибки и т. д.
- Для описания всех этих функций в одну строку введите в консоли Python:

```
help(scipy.special)
Output:
NAME
    scipy.special

DESCRIPTION
    ========================================
    Special functions (:mod:`scipy.special`)
    ========================================

    .. module:: scipy.special

    Nearly all of the functions below are universal functions and follow
    broadcasting and automatic array-looping rules. Exceptions are noted.
```

Функция кубического корня

Функция Cubic Root находит кубический корень значений.
Синтаксис:

```
scipy.special.cbrt(x)
```

Это критически важно для анализа и выбора наиболее эффективных ключевых слов для улучшения рейтинга вашего сайта.

```
from scipy.special import cbrt
#Find cubic root of 27 & 64 using cbrt() function
cb = cbrt([27, 64])
#print value of cb
print(cb)
```

Вывод: массив([3., 4.])

Экспоненциальная функция:

Экспоненциальная функция вычисляет элемент 10**x.wise.
Пример:

```
from scipy.special import exp10
#define exp10 function and pass value in its
exp = exp10([1,10])
print(exp)
```

Вывод: [1.e+01 1.e+10]

Перестановки и комбинации

SciPy также предоставляет функции для расчета перестановок и комбинаций.
Комбинации - scipy.special.comb(N,k**)**
Пример:

```
from scipy.special import comb
#find combinations of 5, 2 values using comb(N, k)
com = comb(5, 2, exact = False, repetition=True)
print(com)
```

Вывод: 15.0
Перестановки –

```
scipy.special.perm(N,k)
```

Пример:

```
from scipy.special import perm
#find permutation of 5, 2 using perm (N, k) function
per = perm(5, 2, exact = True)
print(per)
```

Вывод: 20

Экспоненциальная функция логарифмической суммы

Log Sum Exponential вычисляет журнал входного элемента экспоненциальной суммы.
Синтаксис:

```
scipy.special.logsumexp(x)
```

Функция Бесселя

Функция вычисления N-го целочисленного порядка
Синтаксис:

```
scipy.special.jn()
```

Линейная алгебра с SciPy

- Линейная алгебра SciPy — это реализация библиотек BLAS и ATLAS LAPACK.
- Производительность линейной алгебры очень высокая по сравнению с BLAS и LAPACK.
- Процедура линейной алгебры принимает объект двумерного массива, и выходные данные также представляют собой двумерный массив.

Теперь давайте проведем тест с **scipy.linalg,**
Расчет **определитель** двумерной матрицы,

```
from scipy import linalg
import numpy as np
#define square matrix
two_d_array = np.array([ [4,5], [3,2] ])
#pass values to det() function
linalg.det( two_d_array )
```

Вывод: -7.0
Обратная матрица –

```
scipy.linalg.inv()
```

Обратная матрица Scipy вычисляет обратную любую квадратную матрицу. Давайте посмотрим,

```
from scipy import linalg
import numpy as np
# define square matrix
two_d_array = np.array([ [4,5], [3,2] ])
#pass value to function inv()
linalg.inv( two_d_array )
```

Вывод:

```
array( [[-0.28571429,  0.71428571],
       [ 0.42857143, -0.57142857]] )
```

Собственные значения и собственный вектор

scipy.linalg.eig()

- Самая распространенная проблема в линейной алгебре — это собственные значения и собственный вектор, которые можно легко решить с помощью **Эйг()**функции.
- Теперь давайте найдем собственное значение (**X**) и соответствуют собственному вектору двумерной квадратной матрицы.

Пример

```
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[5,4],[6,3]])
#pass value into function
eg_val, eg_vect = linalg.eig(arr)
#get eigenvalues
print(eg_val)
#get eigenvectors
print(eg_vect)
```

Вывод:

```
[ 9.+0.j -1.+0.j] #eigenvalues
 [ [ 0.70710678 -0.5547002 ] #eigenvectors
   [ 0.70710678  0.83205029] ]
```

Дискретное преобразование Фурье — scipy.fftpack

- ДПФ — это математический метод, который используется для преобразования пространственных данных в частотные данные.
- БПФ (быстрое преобразование Фурье) — это алгоритм вычисления ДПФ.
- БПФ применяется к многомерному массиву.
- Частота определяет количество сигналов или длин волн в определенный период времени.

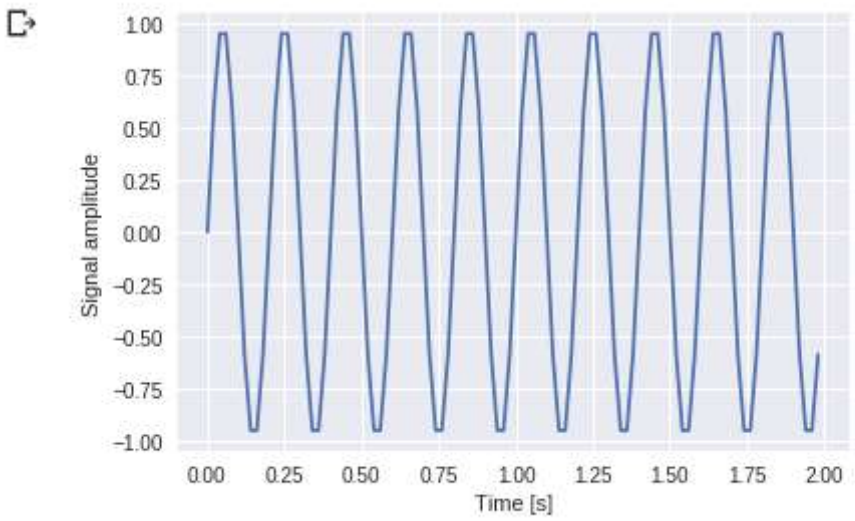
Пример: Возьмите волну и покажите, используя библиотеку Matplotlib. мы возьмем пример простой периодической функции sin(20 × 2πt)

```
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np

#Frequency in terms of Hertz
fre  = 5
```

```
#Sample rate
fre_samp = 50
t = np.linspace(0, 2, 2 * fre_samp, endpoint = False )
a = np.sin(fre  * 2 * np.pi * t)
figure, axis = plt.subplots()
axis.plot(t, a)
axis.set_xlabel ('Time (s)')
axis.set_ylabel ('Signal amplitude')
plt.show()
```

Вывод:



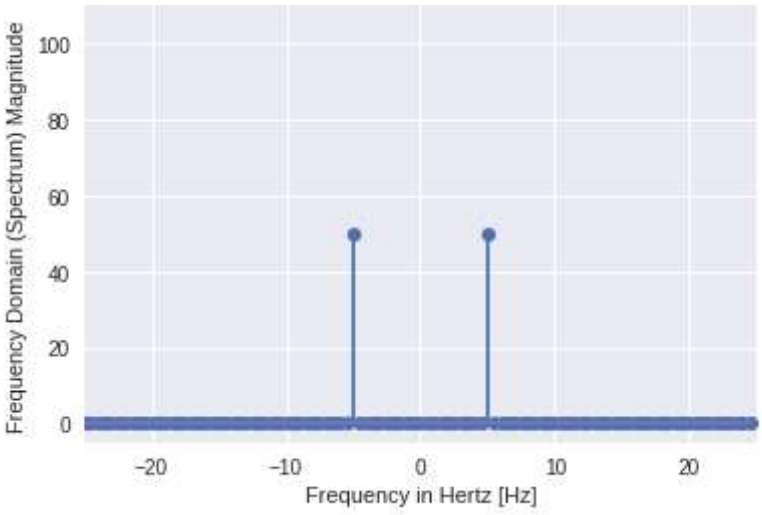
Вы можете это увидеть. Частота 5 Гц, сигнал повторяется через 1/5 секунды – это называется определенный период времени.
Теперь давайте воспользуемся этой синусоидальной волной с помощью приложения DFT.

```
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()
```

Вывод:



- Вы можете ясно видеть, что выходные данные представляют собой одномерный массив.
- Ввод, содержащий complex значения равны нулю, за исключением двух точек.
- В примере ДПФ мы визуализируем величину сигнала.

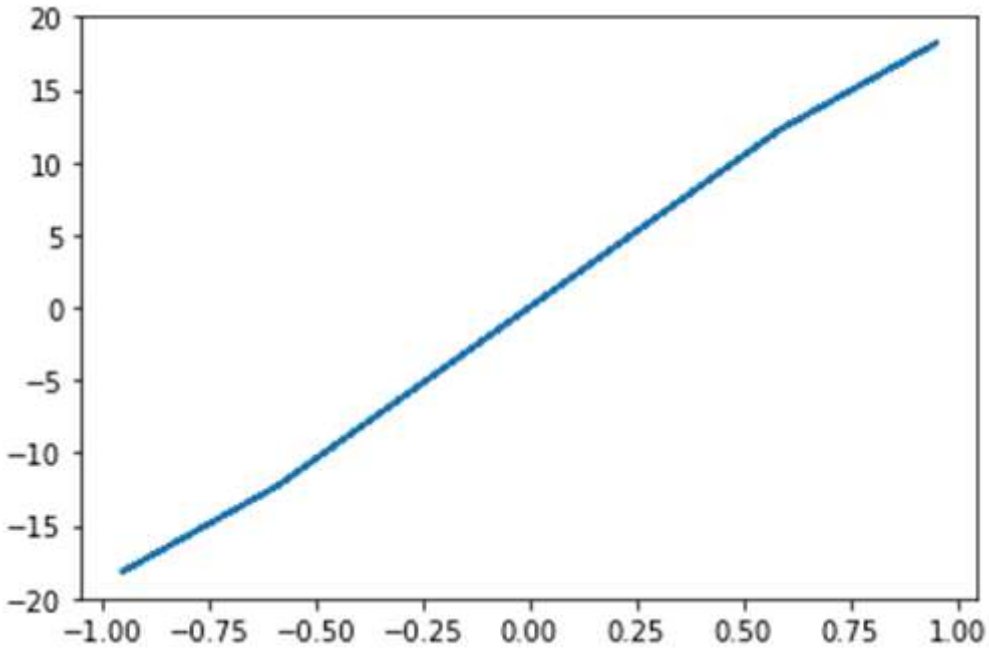
Оптимизация и соответствие SciPy – scipy.optimize

- Оптимизация предоставляет полезный алгоритм для минимизации подгонки кривой, многомерной или скалярной, и подгонки корня.
- Возьмем пример **скалярная функция,**найти минимальную скалярную функцию.

```
%matplotlib inline
import matplotlib.pyplot as plt
from scipy import optimize
import numpy as np

def function(a):
       return   a*2 + 20 * np.sin(a)
plt.plot(a, function(a))
plt.show()
#use BFGS algorithm for optimization
optimize.fmin_bfgs(function, 0)
```

Вывод:



Оптимизация успешно завершена.
Текущее значение функции: -23.241676
Итераций: 4
Функциональных оценок: 18
Оценок градиента: 6
массив([-1.67096375])

- В данном примере оптимизация производится с помощью алгоритма градиентного спуска от начальной точки.
- Но возможная проблема заключается в локальных минимумах, а не в глобальных минимумах. Если мы не находим соседа глобальных минимумов, нам нужно применить глобальную оптимизацию и найти функцию глобальных минимумов, используемую как **плавание по бассейну()** который объединяет локальный оптимизатор.

оптимизировать.basinhopping(функция, 0)
Вывод:

```
fun: -23.241676238045315
 lowest_optimization_result:
      fun: -23.241676238045315
 hess_inv: array([[0.05023331]])
      jac: array([4.76837158e-07])
  message: 'Optimization terminated successfully.'
     nfev: 15
      nit: 3
     njev: 5
   status: 0
  success: True
        x: array([-1.67096375])
                    message: ['requested number of basinhopping
iterations completed successfully']
```

```
minimization_failures: 0
                  nfev: 1530
                   nit: 100
                  njev: 510
             x: array([-1.67096375])
```

Алгоритм Нелдера-Мида:

- Алгоритм Нелдера-Мида выбирает через параметр метода.
- Он обеспечивает наиболее простой способ минимизации функции добросовестного поведения.
- Алгоритм Нелдера – Мида не используется для оценки градиента, поскольку поиск решения может занять больше времени.

```
import numpy as np
from scipy.optimize import minimize
#define function f(x)
def f(x):
    return .4*(1 - x[0])**2

optimize.minimize(f, [2, -1], method="Nelder-Mead")
```

Вывод:

```
final_simplex: (array([[ 1.        , -1.27109375],
       [ 1.        , -1.27118835],
       [ 1.        , -1.27113762]]), array([0., 0., 0.]))
           fun: 0.0
       message: 'Optimization terminated successfully.'
          nfev: 147
           nit: 69
        status: 0
       success: True
             x: array([ 1.        , -1.27109375])
```

Обработка изображений с помощью SciPy – scipy.ndimage

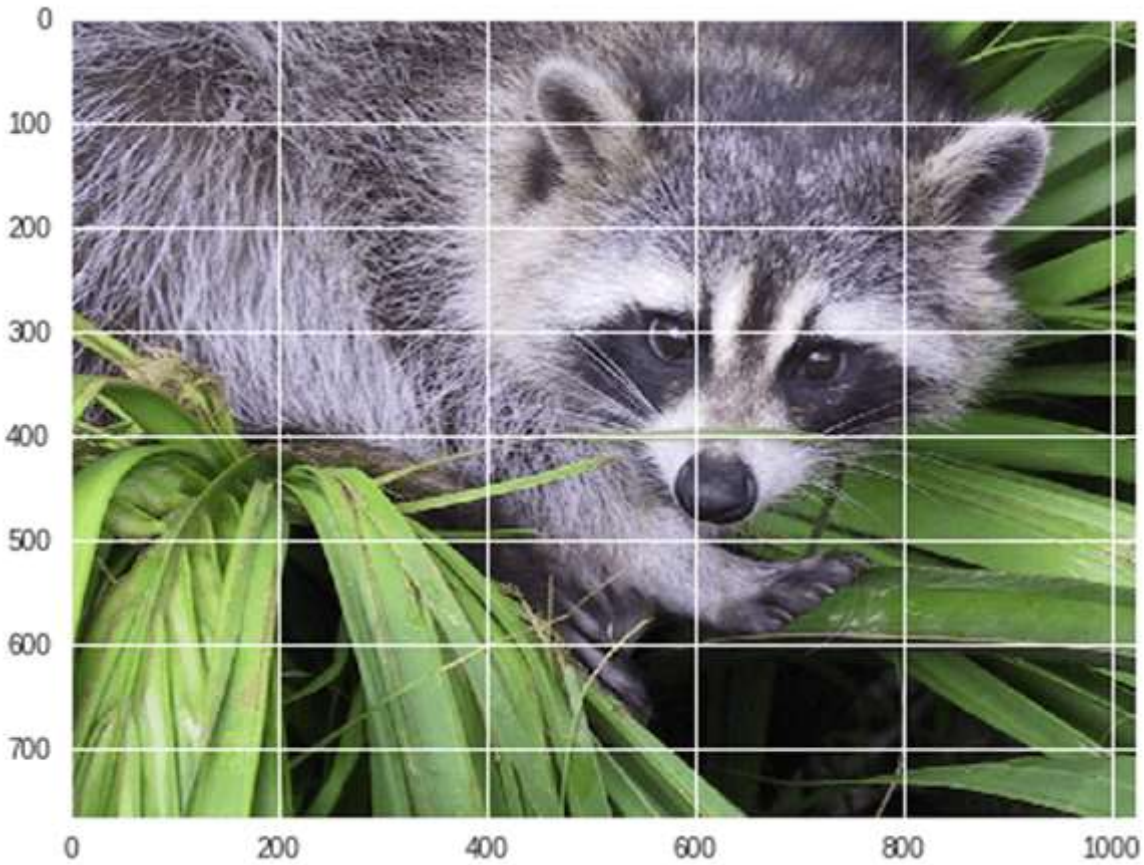
- scipy.ndimage — это подмодуль SciPy, который в основном используется для выполнения операций, связанных с изображением.
- ndimage означает «n»-мерное изображение.

- SciPy Image Processing обеспечивает геометрическое преобразование (поворот, обрезка, переворот), фильтрацию изображений (резкость и удаление), отображение изображения, сегментацию изображения, классификацию и извлечение признаков.
- **Пакет «Разное»** в SciPy содержит готовые изображения, которые можно использовать для выполнения задачи манипулирования изображениями.

Пример: Возьмем пример геометрического преобразования изображений.

```
from scipy import misc
from matplotlib import pyplot as plt
import numpy as np
#get face image of panda from misc package
panda = misc.face()
#plot or show image of face
plt.imshow( panda )
plt.show()
```

Вывод:

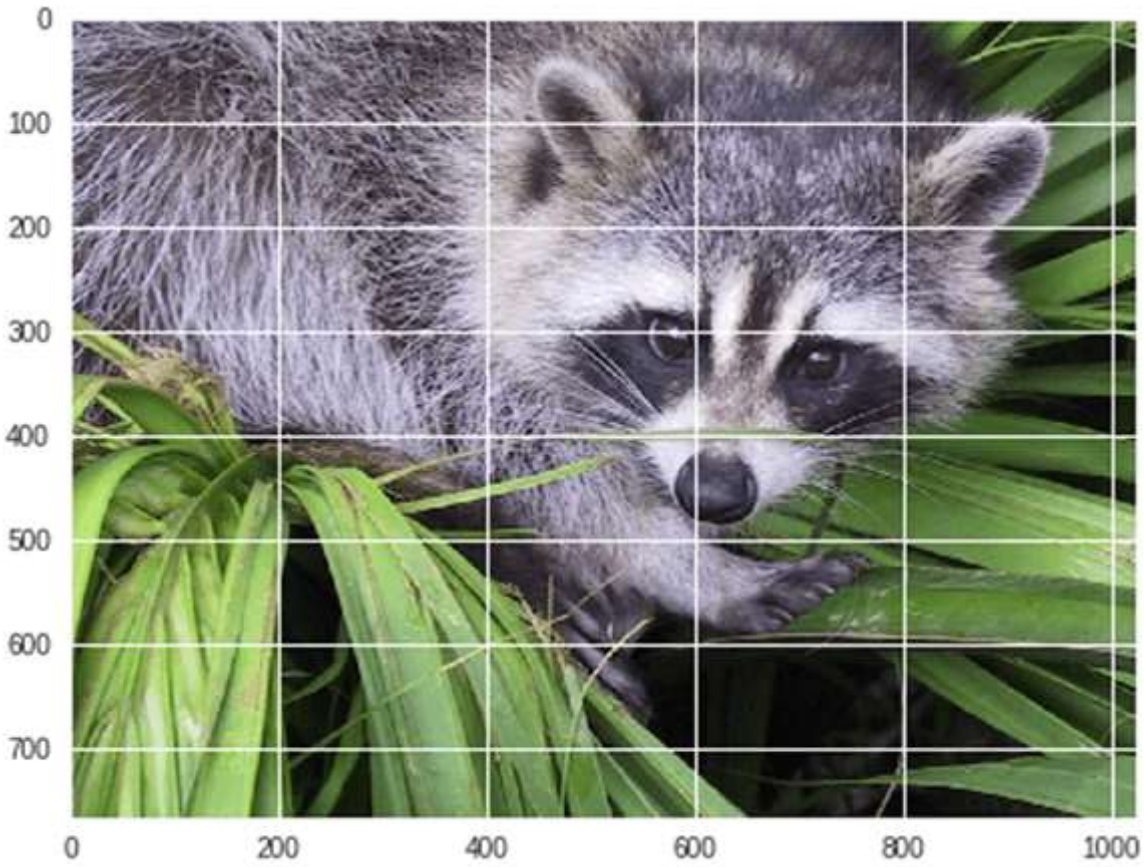


Теперь мы **Откидной вниз** текущее изображение:

```
#Flip Down using scipy misc.face image
flip_down = np.flipud(misc.face())
```

```
plt.imshow(flip_down)
plt.show()
```

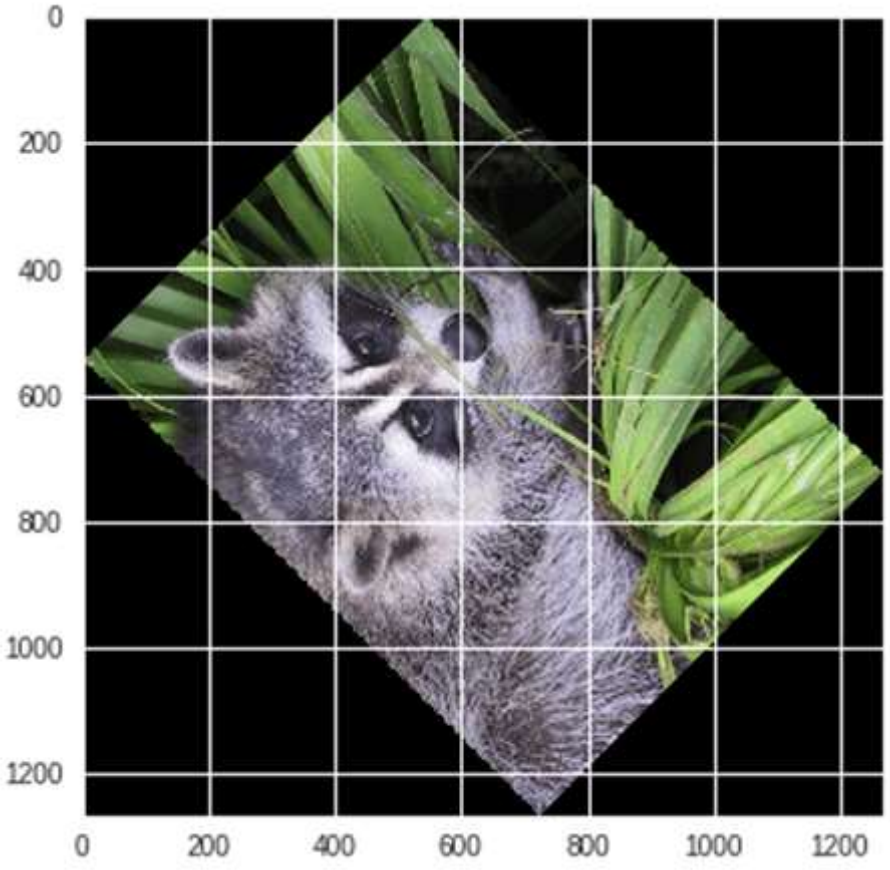
Вывод:



Пример: Поворот изображения с использованием Scipy,

```
from scipy import ndimage, misc
from matplotlib import pyplot as plt
panda = misc.face()
#rotatation function of scipy for image – image rotated 135 degree
panda_rotate = ndimage.rotate(panda, 135)
plt.imshow(panda_rotate)
plt.show()
```

Вывод:



Интеграция со Scipy – численное интегрирование

- Когда мы интегрируем любую функцию, аналитическое интегрирование которой невозможно, нам нужно обратиться к численному интегрированию.
- SciPy предоставляет функциональные возможности для интеграции функций с численным интегрированием.
- **scipy.интегрировать** библиотека имеет единую интеграцию, double, тройка, кратность, квадрат Гаусса, правила Ромберга, трапеции и правила Симпсона.

Пример: Теперь возьмем пример **Единая интеграция**

$$\int_a^b f(x)dx$$

Здесь **a** является верхним пределом и **b** это нижний предел

```
from scipy import integrate
# take f(x) function as f
f = lambda x : x**2
#single integration with a = 0 & b = 1
integration = integrate.quad(f, 0 , 1)
print(integration)
```

Вывод:
(0.33333333333333337, 3.700743415417189e-15)

Здесь функция возвращает два значения, в которых первое значение представляет собой интегрирование, а второе значение — оценочную ошибку в интеграле.
Пример: Теперь возьмем пример SciPy **double интеграция.** Мы находим double интеграция следующегowing уравнение,

$$\int_0^{2/4} \int_0^{\sqrt{1-2y^2}} 64xy\, dx$$

```
from scipy import integrate
import numpy as np
#import square root function from math lib
from math import sqrt
# set  fuction f(x)
f = lambda x, y : 64 *x*y
# lower limit of second integral
p = lambda x : 0
# upper limit of first integral
q = lambda y : sqrt(1 - 2*y**2)
# perform double integration
integration = integrate.dblquad(f , 0 , 2/4,  p, q)
print(integration)
```

Вывод:
(3.0, 9.657432734515774e-14)
Вы видели этот вывод выше, как и предыдущий.

Выводы

- SciPy (произносится как «Sigh Pi») — это библиотека с открытым исходным кодом на основе Python, которая используется в математике, научных вычислениях, инженерии и технических вычислениях.
- SciPy содержит множество подпакетов, которые помогают решить наиболее распространенные проблемы, связанные с научными вычислениями.
- SciPy построен поверх NumPy.