

# SciPy

Гвоздев П.А.

27 Мая 2023

SciPy - это набор математических алгоритмов и удобных функций, построенный на библиотеке NumPy.

SciPy организован в подпакеты, охватывающие различные области научных вычислений. Они обобщены в следующей таблице:

Подпакет	Описание
cluster	Алгоритмы кластеризации
constants	Физические и математические константы
fftpack	Процедуры быстрого преобразования Фурье
integrate	Решатели интегрирования и обыкновенных дифференциальных уравнений
interpolate	Интерполяция и сглаживание сплайнов
io	Ввод и вывод
linalg	Линейная алгебра
ndimage	Обработка N-мерных изображений
odr	Регрессия ортогонального расстояния
optimize	Процедуры оптимизации и поиска корня
signal	Обработка сигналов
sparse	Разреженные матрицы и связанные с ними подпрограммы
spatial	Пространственные структуры данных и алгоритмы
special	Специальные функции
stats	Статистические распределения и функции

Далее рассмотрим некоторые из них.

## 1 Интегральные функции

Подпакет предоставляет несколько методов интегрирования, включая интегратор обыкновенных дифференциальных уравнений. Возвращаемое значение большинства функций представляет собой кортеж, в котором первый элемент содержит оценочное значение интеграла, а второй элемент содержит верхнюю границу ошибки.

Функция **quad** предназначена для интегрирования функции одной переменной на заданном отрезке. Например, предположим, что вы хотите вычислить вот такой интеграл.

$$I(a, b) = \int_0^1 ax^2 + b dx$$

Вычислим значение этой функции в точке (2 ; 1).

```
1 from scipy.integrate import quad
2 def integrand(x, a, b):
3     return a*x**2 + b
4 a = 2
5 b = 1
6 I = quad(integrand, 0, 1, args=(a,b))
7 I
8
9 >>(1.6666666666666667, 1.8503717077085944e-14)
```

Механизмы взятия двойных и тройных интегралов были объединены в функции **dblquad** и **tplquad**. Эти функции используют функцию для интегрирования и четыре или шесть аргументов соответственно. Пределы всех внутренних интегралов должны быть определены как функции. В качестве примера для непостоянных пределов рассмотрим интеграл

$$I = \int_0^1 dy \int_{1-3y}^{1+2y} x^2 y dx$$

Его можно вычислить так:

```
1 from scipy.integrate import dblquad
2 I = dblquad(lambda x, y: x**2*y, 0, 1, lambda y: 1-3*y, lambda y :
3     1+2*y)
4 I
5 >>(2.75, 1.2852467283249305e-13)
```

## 2 Интерполяция

В SciPy доступно несколько общих возможностей для интерполяции и сглаживания данных в 1, 2 и более высоких измерениях. Выбор конкретной процедуры интерполяции зависит от данных: являются ли они одномерными, представлены в структурированной сетке или неструктурированными. Еще одним фактором является желаемая плавность интерполятора.

Если все, что вам нужно, это линейная (или прерывистая линия) интерполяция, вы можете использовать процедуру **numpy.interp** Для интерполяции требуется два массива данных,  $x$  и  $y$ , и третий массив точек,  $xnew$  для оценки интерполяции.

```

1 x = np.linspace(0, 10, num=11)
2 y = np.cos(-x**2 / 9.0)
3
4 xnew = np.linspace(0, 10, num=1001)
5 ynew = np.interp(xnew, x, y)
6
7 plt.plot(xnew, ynew, '-', label='linear interp')
8 plt.plot(x, y, 'o', label='data')
9 plt.legend(loc='best')
10 plt.show()

```

interpol1.png

Конечно, кусочно-линейная интерполяция создает углы в точках данных, где соединяются линейные фрагменты. Для создания более плавной кривой можно использовать кубические сплайны, где интерполирующая кривая состоит из кубических фрагментов с соответствующими первой и второй производными. В коде эти объекты представлены через **CubicSpline** экземпляры класса. Экземпляр создается с использованием  $x$  и  $y$  массивов данных, а затем его можно оценить, используя целевые  $xnew$  значения:

```

1 from scipy.interpolate import CubicSpline
2 x = np.linspace(0, 10, num=11)
3 y = np.cos(-x**2 / 9.)
4 spl = CubicSpline(x, y)
5 fig, ax = plt.subplots(4, 1, figsize=(5, 7))
6 xnew = np.linspace(0, 10, num=1001)
7 ax[0].plot(xnew, spl(xnew))

```

```

8 ax[0].plot(x, y, 'o', label='data')
9 ax[1].plot(xnew, spl(xnew, nu=1), '--', label='1st derivative')
10 ax[2].plot(xnew, spl(xnew, nu=2), '--', label='2nd derivative')
11 ax[3].plot(xnew, spl(xnew, nu=3), '--', label='3rd derivative')
12 for j in range(4):
13     ax[j].legend(loc='best')
14 plt.tight_layout()
15 plt.show()

```

interpol2.png

### 3 Оптимизация

**scipy.optimize** представляет набор функций, которые реализуют популярные алгоритмы оптимизации:

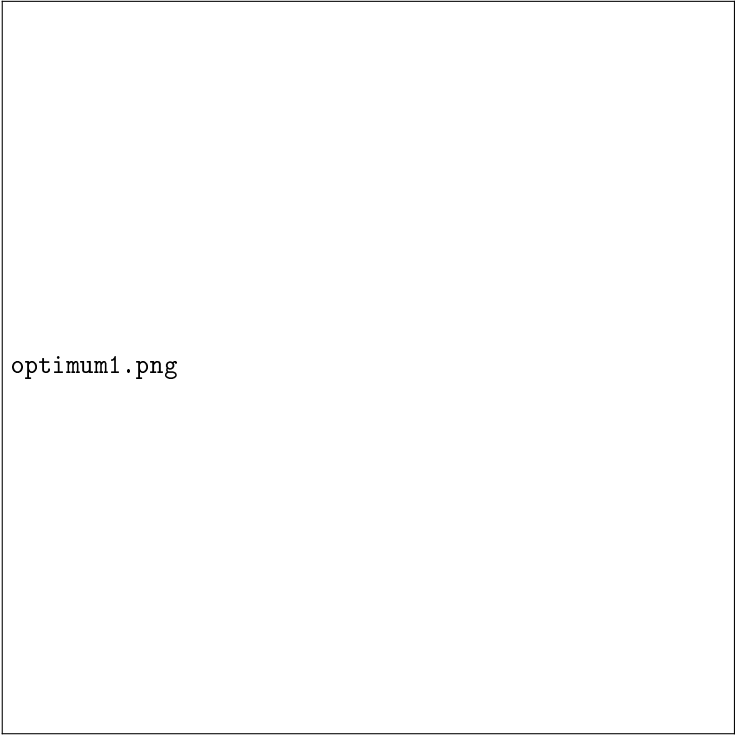
- Неограниченная и ограниченная минимизация многомерных скалярных функций (например, Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно, метод сопряженных градиентов, метод Нелдера — Мида).
- Глобальная оптимизация (дифференциальная эволюция, двойной отжиг и т. д.).
- Минимизация наименьших квадратов и подбор кривой (метод наименьших квадратов, приближение с помощью кривых);

- Минимизаторы скалярных одномерных функций и численное решение уравнений.
- Функции для решения систем многомерных уравнений с помощью таких алгоритмов, как Пауэлла, Левендберга — Марквардта и ещё куча всего.

Пусть у нас есть функция  $f(x) = x^2 + 10\sin(x)$ . Найдём её минимум. Можно использовать алгоритм поиска методом перебора, который будет оценивать каждую точку в сетке диапазонов. Если область определения функции достаточно большая, то `brute()` становится очень медленным. Можно взять функцию `scipy.optimize.anneal()`

```

1 from scipy import optimize
2
3 def target_function(x):
4     return x ** 2 + 10 * np.sin(x)
5
6 plt.figure(figsize=(5,5))
7 x = np.linspace(-10, 10, 1000)
8 plt.xlabel('x')
9 plt.ylabel('y')
10 plt.title('optimize')
11 plt.plot(x, target_function(x), label=r'$f(x)=x^2+10\sin(x)$')
12
13
14 grid = (-10, 10)
15 xmin_global = optimize.brute(target_function, (grid,))
16 print(xmin_global)
17
18 a = target_function(xmin_global)
19 plt.plot(xmin_global, a, 'o', label='min')
20 plt.legend()
21 plt.grid()
22 plt.show()
23
24 >> [-1.30640933]
```



optimum1.png

Напоследок посмотрим на поиск корней. Для нахождения корней уравнений используется функция `scipy.optimize.root(fun, x0, args=(), method, ...)`, где:

- `fun` - целевая функция.
- `x0` - начальное.
- `args` - дополнительные аргументы, которые передаются целевой функции и ее якобиану.
- `method` - метод решения.

Решим уравнение  $x^2 - 2\cos(x) = 0$  с начальным значением  $x_0 = 0.3$ .

```
1 from scipy.optimize import root
2
3 def target_function(x):
4     return x ** 2 - 2 * np.cos(x)
5
6 x0 = 0.3
7 x_root=root(target_function, x0).x
8 print(x_root)
9
10 plt.figure(figsize=(5,5))
11 x = np.linspace(-2, 2, 100)
12 plt.xlabel('x')
13 plt.ylabel('y')
```

```
14 plt.plot(x, target_function(x), label=r'$f(x)=x^2-2\cos(x)$')
15 plt.plot(x_root, target_function(x_root), 'o', label='root')
16 plt.legend()
17 plt.grid()
18 plt.show()
19
20 >>[1.02168995]
```

optimum2.png