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 интеграции 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).

```
from scipy.integrate import quad
def integrand(x, a, b):
    return a*x**2 + b
a = 2
b = 1
I = quad(integrand, 0, 1,args=(a,b))
I

>>(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:

```
from scipy.integrate import dblquad
I = dblquad(lambda x, y: x**2*y, 0, 1, lambda y: 1-3*y, lambda y :
    1+2*y)
I

>>(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.

```
x = np.linspace(0, 10, num=11)
y = np.cos(-x**2 / 9.0)

xnew = np.linspace(0, 10, num=1001)
ynew = np.interp(xnew, x, y)

plt.plot(xnew, ynew, '-', label='linear interp')
plt.plot(x, y, 'o', label='data')
plt.legend(loc='best')
plt.show()
```

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:

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

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

3 Оптимизация

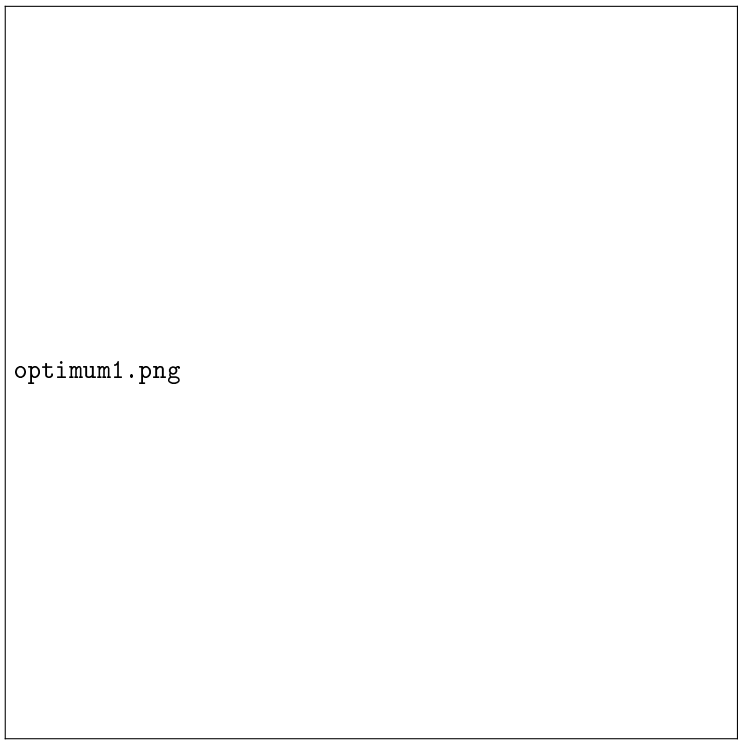
scipy.optimize предстявляет набор функций, которые реализуют популярные алгоритмы оптимизаци:

- 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()**

```
from scipy import optimize

def target_function(x):
    return x ** 2+10*np.sin(x)

plt.figure(figsize=(5,5))
x = np.linspace(-10, 10, 1000)
plt.xlabel('x')
plt.ylabel('y')
plt.title('optimize')
plt.plot(x, target_function(x),  label=r'$f(x)=x^2+10\sin(x)$')


grid = (-10, 10)
xmin_global = optimize.brute(target_function, (grid,))
print(xmin_global)

a = target_function(xmin_global)
plt.plot(xmin_global, a, 'o', label='min')
plt.legend()
plt.grid()
plt.show()

>>[-1.30640933]
```



На последок посомтрим на поиск корней. Для нахождения кореней 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 $x_0 = 0.3$.

```
from scipy.optimize import root

def target_function(x):
   return x ** 2 - 2 * np.cos(x)

x0 = 0.3
x_root=root(target_function, x0).x
print(x_root)

plt.figure(figsize=(5,5))
x = np.linspace(-2, 2, 100)
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.plot(x, target_function(x),  label=r'$f(x)=x^2-2\cos(x)$')
plt.plot(x_root, target_function(x_root), 'o', label='root')
plt.legend()
plt.grid()
plt.show()

>>[1.02168995]
```