Максимумы, минимумы, нули

(модуль sciру.optimize)

Ф.Я.Халили

МГУ, физический факультет

28 апреля 2009 г.

Подключение

Поиск максимумов, минимумов и нулей функций обеспечивается модулем scipy.optimize. Загружается он командами

```
from scipy import *
from scipy.optimize import *
```

либо

```
from pylab import *
from scipy.optimize import *
```

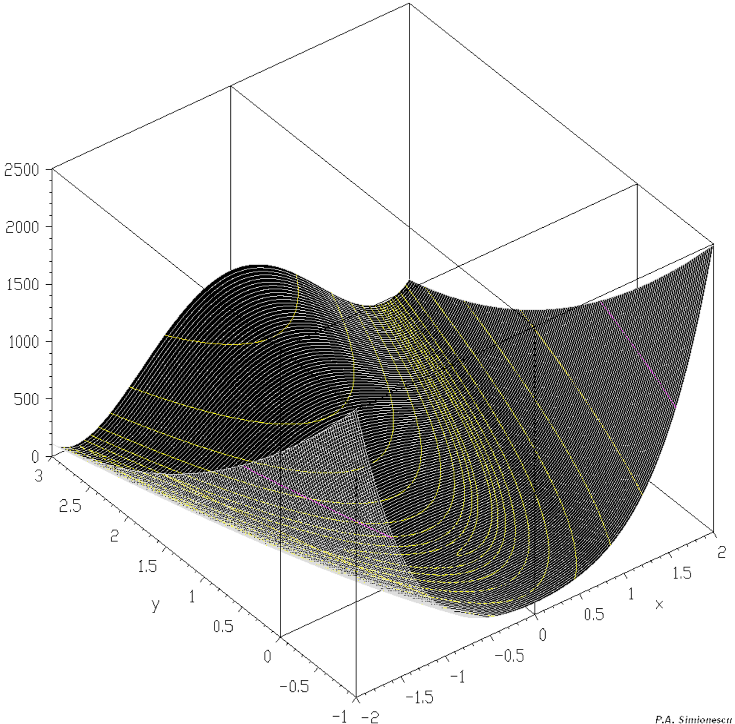
(второй вариант подключает также модуль matplotlib).

Функция Розенброка

В качестве “подопытного животного” при тестировании методов поиска экстремумов стандартно используется функция Розенброка

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2\,, \tag{1}$$

имеющая характерную длинную, узкую и искривленную долину. Ее очевидный глобальный минимум 0 достигается при $x = y = 1$.



2500
2000
1500
1000
500
0
3
2.5
2
1.5
1
0.5
0
-0.5
-1
y
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
x
P.A. Simionescu

Модуль rbrock.py

Создадим модуль, вычисляющий эту функцию и ее производные:

```
from pylab import *
from scipy.optimize import *

def rbrock(x):
  return (1-x[0])**2 + 100*(x[1]-x[0]**2)**2

def drbrock(x):
  d = array([0,0],float)
  d[0] = 2*(x[0]-1) + 400*x[0]*(x[0]**2-x[1])
  d[1] = 200*(x[1]-x[0]**2)
  return d
```

Nelder-Mead

```
fmin(<функция>,<стартовая точка>)
```

Симлексный, он же "амебный" метод, который требует задания только самой функции (градиент не требуется):

```
>>> fmin(rbrock,(0,0))
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 79
         Function evaluations: 146
array([ 1.00000439,  1.00001064])
```

Более детальное управление

```
fmin(<функция>,<стартовая точка>,\
  xtol=10^-4,ftol=10^-4,full_output=0)
```

Можно управлять точностью вычисления, а также сохранять информацию дополнительную информацию, выводимую командой fmin:

```
>>> fmin(rbrock,(0,0),xtol=1e-9,ftol=1e-9,\
... full_output=1)
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 118
         Function evaluations: 222
(array([ 1., 1.]), 2.8641094859084798e-20,118,222,0)
```

Берем в руки BFG(S)

```
fmin_bfgs(<функция>,<стартовая точка>,\
  fprime=<градиент>)
```

Алгоритм Broyden-Fletcher-Goldfarb-Shanno (или квази-метод Ньютона) требует задания как функции, так и ее градиента:

```
>>> fmin_bfgs(rbrock,(0,0),drbrock)
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 21
         Function evaluations: 26
         Gradient evaluations: 26
array([ 1.00000001,  1.00000002])
```

Выигрыш по числу итераций почти 4!

Управление точностью

fmin_bfgs(<функция>,<стартовая точка>,\
fprime=<градиент>,gtol=1^{-5},full_output=0)

```
>>> fmin_bfgs(rbrock,(0,0),drbrock,gtol=1e-20,\
...full_output=1)
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 25
         Function evaluations: 30
         Gradient evaluations: 30
(array([ 1.,  1.]), 0.0, array([ 0.,  0.]),
    array([[ 0.4992803 ,  0.99850515],
[ 0.99850515,  2.00189516]]),
    30, 30, 0)
```

1. Введение
2. Минимизация функций
3. Подгонка методом наименьших квадратов
4. Решение алгебраических уравнений

Функция leastsq

Функция `leastsq(<фунция>,<стартовая точка>)` минимизирует квадрат модуля своего первого аргумента. Как правило (но не обязательно!), он является вектором вида

$$\{\delta_j\} = \{y_j - f(x_j, \vec{p})\}\,,$$

где $f()$ – некоторая функция, $\{x_j\}$, $\{y_j\}$ – заданные вектора и $\vec{p}$ – искомый вектор, причем

$$\mathrm{size}(p) < \mathrm{size}(x) = \mathrm{size}(y)\,,$$

то есть мы имеем систему из $\mathrm{size}(x) = \mathrm{size}(y)$ уравнений с $\mathrm{size}(p)$ неизвестными.

Исходные данные

Тестовые данные (зашумленная затухающая синусоида) мы сгенерируем следующей программой:

```
from numpy import *
from numpy.random import *
tau=20.0
Omega=0.7
Dx=0.5
Dy=0.5
N=20
t=arange(N)
x=t+Dx*Dx*(rand(N)-0.5)
y=exp(-t/tau)*sin(Omega*t)+Dy*(rand(N)-0.5)
x.tofile('x.dat','\n')
y.tofile('y.dat','\n')
```

Подгонка

А подгонять будем так:

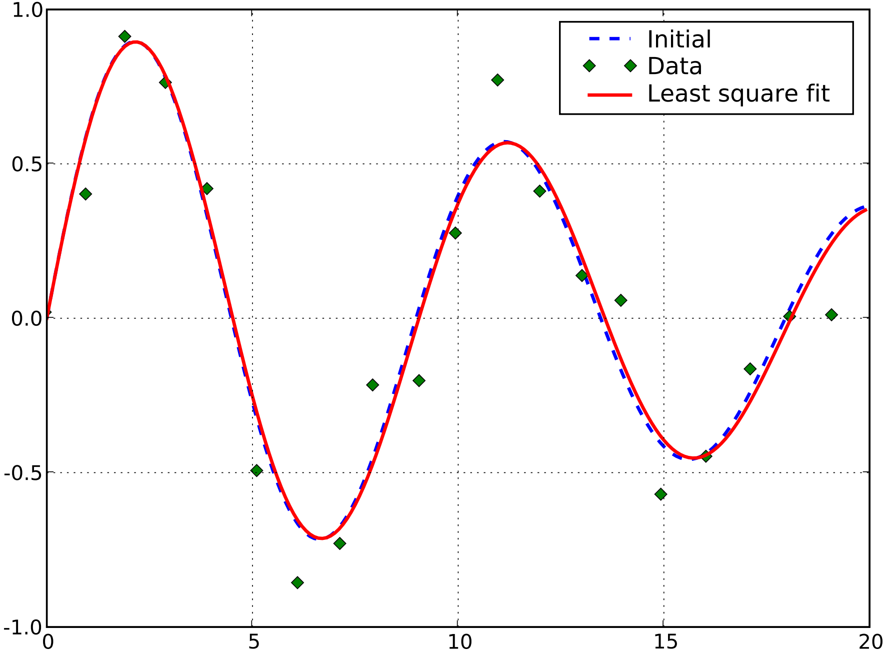
```
from pylab import *
from scipy.optimize import *

x = fromfile('x.dat',float,sep='\n')
y = fromfile('y.dat',float,sep='\n')

def f(t,Omega,tau):
  return exp(-t/tau)*sin(Omega*t)

def delta(p):
  Omega,tau = p
  return y-f(x,Omega,tau)

Omega,tau = leastsq(delta,(1,10))[0]
```


Initial
Data
Least square fit
1.0
0.5
0.0
-0.5
-1.0
0
5
10
15
20

1. Введение
2. Минимизация функций
3. Подгонка методом наименьших квадратов
4. Решение алгебраических уравнений

Уравнения с одним неизвестным

`brentq(<функция>,`a`,`b`,xtol=`10^{-12}`)`

Из нескольких доступных в Scipy методов, этот рекомендуется как наиболее универсальный. Числа a и b задают диапазон, где ищется корень, а необязательный аргумент xtol – точность вычисления:

```
>>> x=brentq(lambda x:sin(x)-x/10,pi/2,pi)
>>> x
2.852341894450158
>>> sin(x)-x/10
8.02691246803988e-14
>>> x=brentq(lambda x:sin(x)-x/10,pi/2,pi,xtol=0.1)
>>> x
2.847974673782915
>>> sin(x)-x/10
0.004619785631200912
```

Стандартный ньюанс

Нельзя слепо доверять компьютеру:

```
>>> brentq(tan,pi/2-1,pi/2+1)
1.5707963267967155
```

Тангенс меняет знак при $x = \pi/2$, что (ошибочно) воспринимается алгоритмом как корень!

Функция fsolve

Другая возможность – это использование функции

`fsolve(<функция>,<стартовая точка>,xtol=`$1.5 \cdot 10^{-8}$`)`

```
>>> x=fsolve(lambda x:sin(x)-x/10,pi/2)
>>> x
2.8523418944500918
>>> sin(x)-x/10
-1.1102230246251565e-16
```

Функция fsolve

Можно задавать сразу вектор стартовых точек, получая вектор корней:

```
>>> fsolve(lambda x:sin(x)-x/10,[0,pi/2,pi])
array([ 0.        ,  2.85234189,  2.85234189])
>>> fsolve(lambda x:sin(x)-x/10,arange(10))
array([0.00000000e+00, 7.08099439e-11, 2.85234190e+00,
       2.85234189e+00, 2.85234189e+00, 5.12414806e-10,
       7.06817436e+00, 7.06817436e+00, 8.42320393e+00,
       8.42320393e+00])
```

Системы уравнений

Та же функция fsolve используется и для поиска нулей векторных функций от векторных аргументов, то есть для решения систем уравнений. Например:

$$x_0 \cos x_1 = 4\,,$$
$$x_0 x_1 - x_1 = 05\,.$$

```
>>> def f(x):
...   r=array([0,0],float)
...   r[0]=x[0]*cos(x[1])-4
...   r[1]=x[0]*x[1]-x[1]-5
...   return r
>>> x=fsolve(f,[1,1])
>>> x
array([ 6.50409711,  0.90841421])
>>> f(x)
array([  3.73212572e-12,   1.61701763e-11])
```

Выбор начальной точки

во многом определяет успешность решения:

```
>>> x=fsolve(f,[0,0])
Warning: The iteration is not making good progress, as
  improvement from the last ten iterations.
>>> x
array([-1.33631503, -2.543248  ])
>>> f(x)
array([-2.8958441 ,  0.94182852])
```