CHƯƠNG 5: TÍNH GẦN ĐÚNG ĐẠO HÀM & TÍCH PHÂN

Tài liệu:

1. Giải Tích Số, Phạm Kỳ Anh
2. Elementary Numerical Analysis, Atkinson & Han
3. Numerical methods, Greenbaum & Chartier

Tác giả: TS. Hà Phi
Khoa Toán – Cơ - Tin học
ĐHKHTN, ĐHQGHN

BÀI TOÁN 1: TÍNH GẦN ĐÚNG ĐẠO HÀM $\frac{d^n f}{dx^n}$ hay $f^{(n)}(x)$

- Đặt vấn đề: chúng ta đã cho hàm y = f (x) và muốn nhận một trong các đạo hàm của nó tại điểm x = x_k. Thuật ngữ “đã cho” có nghĩa là chúng ta có ***một thuật toán*** để tính toán hàm hoặc chúng ta có ***một tập hợp các điểm dữ liệu rời rạc*** (x_i, y_i), i = 0, 1,. . . , n.
- Trong cả hai trường hợp, chúng ta có quyền truy cập vào một số lượng hữu hạn các cặp dữ liệu (x, y) để tính toán đạo hàm.
- Công cụ 1: khai triển chuỗi Taylor của f (x) tại điểm x_k, công cụ này có ưu điểm là cung cấp cho chúng ta thông tin về sai số **=> *Phương pháp sai phân hữu hạn***
- Công cụ 2: đa thức nội suy, i.e. ta có thể đi tính đạo hàm của đa thức nội suy P(x) xấp xỉ f(x) và sau đó tính P’(x_k). **=> *Phương pháp nội suy***
- Google search: automatic differentiation

CÁC CÔNG THỨC SAI PHÂN TRUNG TÂM

Dựa trên các khai triển chuỗi Taylor tiến và lùi của f (x) tại x, chẳng hạn như

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + \cdots \quad \text{(a)}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - \cdots \quad \text{(b)}$$

Do đó ta có

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{12}f^{(4)}(x) + \cdots \quad \text{(e)}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f'''(x) + \ldots \quad \text{(f)}$$

Từ (e) và (f) ta có công thức ***sai phân trung tâm*** cho f'(x) với sai số O(h^2)

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2)$$

Q? Sử dụng thêm 2 khai triển Taylor nữa của f(x) tại x như (c) & (d) ở dưới đây. Hãy đi tìm công thức sai phân trung tâm cho f'(x) với sai số O(h^4)

$$f(x+2h) = f(x) + 2hf'(x) + \frac{(2h)^2}{2!}f''(x) + \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) + \cdots \quad \text{(c)}$$

$$f(x-2h) = f(x) - 2hf'(x) + \frac{(2h)^2}{2!}f''(x) - \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) - \cdots \quad \text{(d)}$$

CÁC CÔNG THỨC SAI PHÂN TRUNG TÂM

Tương tự ta có

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{h^2}{12} f^{(4)}(x) + \dots \quad \Rightarrow \quad f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^2)$$

Thay thế các xấp xỉ của f'(x) và f''(x) vào (e) và (f) ta được các xấp xỉ đạo hàm cấp cao hơn

$$f'''(x) = \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} + \mathcal{O}(h^2) \qquad (5.3)$$

$$f^{(4)}(x) = \frac{f(x+2h) - 4f(x+h) + 6f(x) - 4f(x-h) + f(x-2h)}{h^4} + \mathcal{O}(h^2) \qquad (5.4)$$

	$f(x-2h)$	$f(x-h)$	$f(x)$	$f(x+h)$	$f(x+2h)$
$2hf'(x)$		-1	0	1	
$h^2 f''(x)$		1	-2	1	
$2h^3 f'''(x)$	-1	2	0	-2	1
$h^4 f^{(4)}(x)$	1	-4	6	-4	1

Bảng tổng hợp các hệ số của xấp xỉ sai phân trung tâm với sai số O(h^2)

CÁC CÔNG THỨC SAI PHÂN 1 PHÍA VỚI SAI SỐ O(h)

Dựa trên các khai triển chuỗi Taylor tiến và lùi của f (x) tại x, chẳng hạn như

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + \cdots \quad \text{(a)}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - \cdots \quad \text{(b)}$$

Do đó ta có các công thức sai phân 1 phía sau

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

Sai phân tiến (forward difference)

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h)$$

Sai phân lùi (backward difference)

Tương tự ta có CT sai phân tiến cho f''(x)

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + \mathcal{O}(h)$$

Chú ý: Thực tế đòi hỏi các công thức có độ chính xác tốt hơn O(h), i.e. O(h^p) với p>1.

CÁC CÔNG THỨC SAI PHÂN 1 PHÍA VỚI SAI SỐ O(h^p)

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \cdots$$

$$f(x+2h) = f(x) + 2hf'(x) + 2h^2 f''(x) + \frac{4h^3}{3}f'''(x) + \frac{2h^4}{3}f^{(4)}(x) + \cdots$$

Chúng ta loại bỏ f "(x) bằng cách lấy PT2 – 4 * PT1. Kết quả là

$$f(x+2h) - 4f(x+h) = -3f(x) - 2hf'(x) + \frac{2h^3}{3}f'''(x) + \cdots$$

=>

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \frac{h^2}{3}f'''(x) + \cdots$$

=>

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \mathcal{O}(h^2)$$

Xấp xỉ sai phân tiến bậc 2

CÁC CÔNG THỨC SAI PHÂN 1 PHÍA VỚI SAI SỐ O(h^p)

Các xấp xỉ sai phân hữu hạn cho các đạo hàm cấp cao hơn f'(x) liên quan thêm nhiều chuỗi Taylor. Do đó, xấp xỉ sai phân tiến cho f "(x) sử dụng chuỗi Taylor cho f (x + h), f (x + 2h) và f (x + 3h); xấp xỉ cho f "'(x) bao gồm các chuỗi cho f (x + h), f (x + 2h), f (x + 3h), f (x + 4h), v.v.

	$f(x)$	$f(x+h)$	$f(x+2h)$	$f(x+3h)$	$f(x+4h)$	$f(x+5h)$
$2hf'(x)$	−3	4	−1			
$h^2 f''(x)$	2	−5	4	−1		
$2h^3 f'''(x)$	−5	18	−24	14	−3	
$h^4 f^{(4)}(x)$	3	−14	26	−24	11	−2

Table 5.3a. Coefficients of Forward Finite Difference Approximations of $\mathcal{O}(h^2)$

	$f(x-5h)$	$f(x-4h)$	$f(x-3h)$	$f(x-2h)$	$f(x-h)$	$f(x)$
$2hf'(x)$				1	−4	3
$h^2 f''(x)$			−1	4	−5	2
$2h^3 f'''(x)$		3	−14	24	−18	5
$h^4 f^{(4)}(x)$	−2	11	−24	26	−14	3

Table 5.3b. Coefficients of Backward Finite Difference Approximations of $\mathcal{O}(h^2)$

NGOẠI SUY RICHARDSON

- Phép ngoại suy Richardson là một phương pháp đơn giản để tăng độ chính xác của một số phương pháp số nhất định, bao gồm các phép xấp xỉ sai phân hữu hạn.
- Giả sử rằng chúng ta có một phương pháp tính toán gần đúng một đại lượng G.

  Hơn nữa, giả sử rằng kết quả phụ thuộc vào một tham số h (step/bước của phương pháp).

  Ký hiệu xấp xỉ bằng g (h), ta có G = g(h) + E(h), trong đó E(h) đại diện cho sai số.
- Phép ngoại suy Richardson có thể giảm sai số, miễn là nó có dạng $E(h) = ch^p$ trong đó c và p là các hằng số.
- Chúng ta bắt đầu bằng cách tính g (h) với một số giá trị của h

$$G = g(h_1) + ch_1^p$$ và $$G = g(h_2) + ch_2^p$$

Khi đó ta có thể khử c và tính ra G bằng ***CT ngoại suy Richardson***

$$G = \frac{(h_1/h_2)^p g(h_2) - g(h_1)}{(h_1/h_2)^p - 1}$$

TH đặc biệt phổ biến là sử dụng
h_2 = h_1/2

$$G = \frac{2^p g(h_1/2) - g(h_1)}{2^p - 1}$$

Let us illustrate Richardson extrapolation by applying it to the finite difference approximation of $(e^{-x})''$ at $x = 1$. We work with six-digit precision and use the results in Table 5.4. Because the extrapolation works only on the truncation error, we must confine h to values that produce negligible roundoff. In Table 5.4 we have

$$g(0.64) = 0.380\,610 \qquad g(0.32) = 0.371\,035$$

The truncation error in central difference approximation is $E(h) = \mathcal{O}(h^2) = c_1 h^2 + c_2 h^4 + c_3 h^6 + \ldots$. Therefore, we can eliminate the first (dominant) error term if we substitute $p = 2$ and $h_1 = 0.64$ in Eq. (5.9). The result is

$$G = \frac{2^2 g(0.32) - g(0.64)}{2^2 - 1} = \frac{4(0.371\,035) - 0.380\,610}{3} = 0.367\,843$$

which is an approximation of $(e^{-x})''$ with the error $\mathcal{O}(h^4)$. Note that it is as accurate as the best result obtained with eight-digit computations in Table 5.4.

PHƯƠNG PHÁP SỬ DỤNG ĐA THỨC NỘI SUY

- Nếu f (x) được cho dưới dạng một tập các điểm dữ liệu rời rạc, thì phép nội suy có thể là một công cụ rất hiệu quả để tính toán các đạo hàm của nó.
- Ý tưởng là xấp xỉ đạo hàm của f (x) bằng đạo hàm của đa thức nội suy

$$P_{n-1}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

đi qua n + 1 điểm dữ liệu

- Ưu điểm: Phương pháp này đặc biệt hữu ích nếu các điểm dữ liệu được phân bố không đều, và do đó không thể áp dụng phương pháp sai phân hữu hạn.
- Hạn chế: Các phương pháp nội suy Lagrange hay Newton đều không phù hợp ở đây, do việc lấy đạo hàm của các đa thức nội suy (dạng Lagrange/Newton) là rất khó & không rẻ (chi phí tính toán đắt).
- Giải pháp: Phương pháp bình phương tối thiểu (sẽ trở thành nội suy khi bậc của đa thức = n, và ta có chính xác n+1 mốc nội suy).
- Trường hợp dữ liệu bị nhiễu (bậc của đa thức = m < n) thì vẫn dùng bình phương tối thiểu nbt.

EXAMPLE 5.4

Given the data

x	1.5	1.9	2.1	2.4	2.6	3.1
$f(x)$	1.0628	1.3961	1.5432	1.7349	1.8423	2.0397

compute $f'(2)$ and $f''(2)$ using (1) polynomial interpolation over three nearest-neighbor points, and (2) a natural cubic spline interpolant spanning all the data points.

Solution of Part (1). The interpolant is $P_2(x) = a_0 + a_1 x + a_2 x^2$ passing through the points at $x = 1.9, 2.1$, and 2.4. The normal equations, Eqs. (3.22), of the least-squares fit are

phương trình chính tắc

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \end{bmatrix}$$

After substituting the data, we get

$$\begin{bmatrix} 3 & 6.4 & 13.78 \\ 6.4 & 13.78 & 29.944 \\ 13.78 & 29.944 & 65.6578 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 4.6742 \\ 10.0571 \\ 21.8385 \end{bmatrix}$$

which yields $\mathbf{a} = \begin{bmatrix} -0.7714 & 1.5075 & -0.1930 \end{bmatrix}^T$.

The derivatives of the interpolant are $P_2'(x) = a_1 + 2a_2 x$ and $P_2''(x) = 2a_2$. Therefore,

$$f'(2) \approx P_2'(2) = 1.5075 + 2(-0.1930)(2) = 0.7355$$

$$f''(2) \approx P_2''(2) = 2(-0.1930) = -0.3860$$

PYTHON BUILT-IN FUNCTIONS: scipy.optimize.approx_fprime (1/2)

```
def approx_fprime(xk, f, epsilon=_epsilon, *args):
    """Finite difference approximation of the derivatives of a
    scalar or vector-valued function.

    If a function maps from :math:`R^n` to :math:`R^m`, its derivatives form
    an m-by-n matrix
    called the Jacobian, where an element :math:`(i, j)` is a partial
    derivative of f[i] with respect to ``xk[j]``.
```

Returns: **jac** : ***ndarray***

The partial derivatives of *f* to *xk*.

PYTHON BUILT-IN FUNCTIONS: scipy.optimize.approx_fprime (2/2)

Notes

The function gradient is determined by the forward finite difference formula:

$$f'[i] = \frac{f(xk[i] + epsilon[i]) - f(xk[i])}{epsilon[i]}$$

Examples

```
>>> from scipy import optimize
>>> def func(x, c0, c1):
...     "Coordinate vector `x` should be an array of size two."
...     return c0 * x[0]**2 + c1*x[1]**2
```

```
>>> x = np.ones(2)
>>> c0, c1 = (1, 200)
>>> eps = np.sqrt(np.finfo(float).eps)
>>> optimize.approx_fprime(x, func, [eps, np.sqrt(200) * eps], c0, c1)
array([   2.        ,  400.00004198])
```

PYTHON BUILT-IN FUNCTIONS: scipy.misc.derivative (1/2)

scipy.misc.derivative

scipy.misc.derivative(*func, x0, dx=1.0, n=1, args=(), order=3*) [source]

Find the nth derivative of a function at a point.

Given a function, use a central difference formula with spacing *dx* to compute the nth derivative at *x0*.

Notes

Decreasing the step size too small can result in round-off error.

Examples

```
>>> from scipy.misc import derivative
>>> def f(x):
...     return x**3 + x**2
>>> derivative(f, 1.0, dx=1e-6)
4.9999999999217337
```

PYTHON BUILT-IN FUNCTIONS: scipy.misc.derivative (2/2)

```
if n == 1:
    if order == 3:
        weights = array([-1,0,1])/2.0
    elif order == 5:
        weights = array([1,-8,0,8,-1])/12.0
    elif order == 7:
        weights = array([-1,9,-45,0,45,-9,1])/60.0
    elif order == 9:
        weights = array([3,-32,168,-672,0,672,-168,32,-3])/840.0
    else:
        weights = central_diff_weights(order,1)
elif n == 2:
    if order == 3:
        weights = array([1,-2.0,1])
    elif order == 5:
        weights = array([-1,16,-30,16,-1])/12.0
    elif order == 7:
        weights = array([2,-27,270,-490,270,-27,2])/180.0
    elif order == 9:
        weights = array([-9,128,-1008,8064,-14350,8064,-1008,128,-9])/5040.0
```

Inside the code scipy.misc.derivative

PYTHON BUILT-IN FUNCTIONS: scipy.optimize.check_grad

```
def check_grad(func, grad, x0, *args, epsilon=_epsilon,
               direction='all', seed=None):
    """Check the correctness of a gradient function by comparing it against a
    (forward) finite-difference approximation of the gradient.
```

Returns: **err** : *float*

The square root of the sum of squares (i.e., the 2-norm) of the difference between `grad(x0, *args)` and the finite difference approximation of *grad* using func at the points *x0*.

```
>>> def func(x):
...     return x[0]**2 - 0.5 * x[1]**3
>>> def grad(x):
...     return [2 * x[0], -1.5 * x[1]**2]
>>> from scipy.optimize import check_grad
>>> check_grad(func, grad, [1.5, -1.5])
2.9802322387695312e-08  # may vary
```