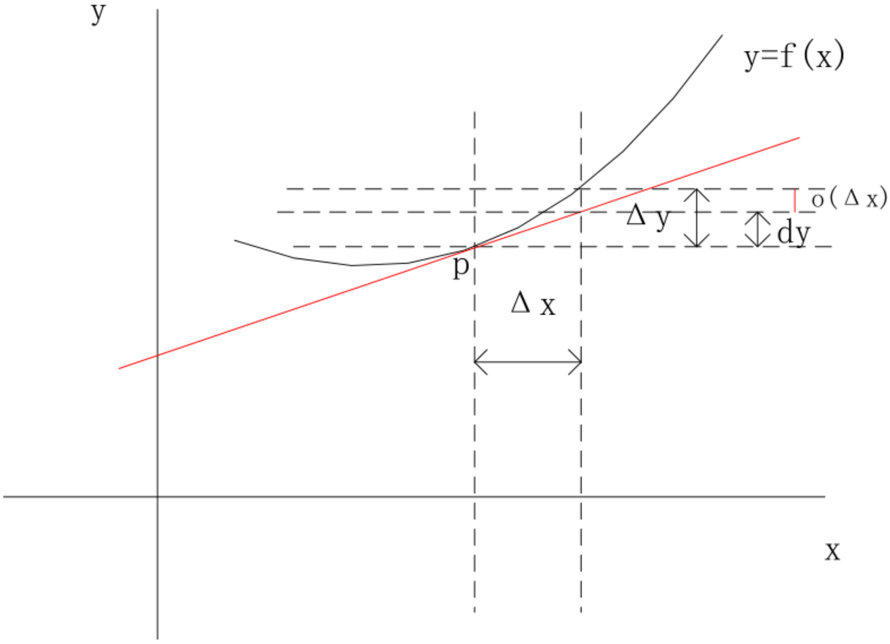
化工应用数学 第三章 数据处理 讲义

（02.数值微分）

导数、微分和数值微分：



导数代表 x 发生变化时，对应 y 值变化对应 x 变化的比率，其定义为：

$$y' = f(x)' = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

而微分代表 x 发生变化时，对应 y 值变化，著名的微分公式如下：

$$dy = f'(x_0)dx$$

注意微分 dy 与 y 值实际的变化△y 之间的差别：相差 o(△x)

而数值微分是在已知条件之下，利用数值算法对于函数导数进行求解，注意*数值微分是一种求导数的算法*。

泰勒公式：

为了更好的分析后面的内容，首先回顾一下函数的泰勒展开和泰勒公式：

函数 f(x)如果在某点附近足够平滑，则能够用其各阶导数作为系数构造多项式，从而在这一点的邻域使用这一多项式来近似函数 f(x)。

$$p_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)^2 + ... + a_n(x-x_0)^n$$

而泰勒公式则是将一个在 $x=x_0$ 处具有 n 阶导数的函数 f（x）利用关于（x-x0）的 n 次多项式来逼近的方法。

若函数 f(x)在包含 x0 的某个闭区间[a,b]上具有 n 阶导数，且在开区间(a,b）上具有（n+1）阶导数，则对闭区间[a,b]上任意一点 x，成立下式：

$$f(x)=\frac{f(x_0)}{0!}+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\ldots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+R_n(x)$$

其中：

$$R_n(x)=\frac{f^{(n+1)}(\varepsilon)}{(n+1)!}(x-x_0)^{n+1},\varepsilon\epsilon(x_0,x)$$

（具体推导可参考相关高数内容）

注意点：注意泰勒公式与上一节课所讲述的函数多项式逼近（如拉格朗日多项式）的区别：多项式逼近是一个整体上的逼近，而泰勒公式是对于某点附近邻域的问题。

数值微分-差商法：

很多情况下，无法使用传统解析的方式求导数，比如：

1）函数关系是利用离散点的方式给出的

2）函数 *f*(x)过于复杂

这种时候，我们就需要使用数值微分解决这种求导问题，这里一般可以使用导数的定义：

$$f(x)'=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

一般来说，我们有三种不同的差商方式：

向前差商$f'(\mathrm{x})=\lim\limits_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}$，即$f'(x_0)\approx\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$

向后差商$f'(\mathrm{x})=\lim\limits_{\Delta x\to 0}\frac{f(x)-f(x-\Delta x)}{\Delta x}$，即$f'(x_0)\approx\frac{f(x_0)-f(x_0-\Delta x)}{\Delta x}$

中心差商$f'(\mathrm{x})=\lim\limits_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}$，即$f'(x_0)\approx\frac{f(x_0+\Delta x)-f(x_0-\Delta x)}{2\Delta x}$

下面利用泰勒公式对上述三种差商方式的误差进行分析：

对于向前差商：

$$f(x_0+\Delta x)=f(x_0)+f'(x_0)\Delta x+\frac{\Delta x^2}{2!}f''(\theta),x_0\le\theta\le x_0+\Delta x$$

$$\mathrm{f}'(x_0)\approx\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}=\frac{f'(x_0)\Delta x+\frac{\Delta x^2}{2!}f''(\theta)}{\Delta x}=f'(x_0)+\frac{\Delta x}{2!}f''(\theta)$$

所以向前差商误差为：

$$R(x)=f'(x_0)-\left[f'(x_0)+\frac{\Delta x}{2!}f''(\theta)\right]=-\frac{\Delta x}{2!}f''(\theta)=O(\Delta x)$$

同样的，可以计算出向后差商的误差为：

$$R(x)=\frac{\Delta x}{2!}f''(\theta)=O(\Delta x)$$

中心差商的误差为：

$$R(x)=\frac{\Delta x^2}{3!}f'''(\theta)=O(\Delta x^2)$$

数值微分-插值法：

在之前的插值算法的课程中，我们已经了解到对于一个函数，我们可以使用拉格朗日多项式去表征它，我们可以使用拉格朗日多项式去逼近函数，并进一步的研究原函数的性质，比如我们这一节所关注的微分性质，即：我们可以用插值函数（如拉格朗日多项式）的导数近似为原函数的导数：

$$f^{(k)}(x)\approx L_n^{(k)}(x)$$

很容易证明，当我们使用两点形式的拉格朗日多项式时，所得到的数值微分结果与向前/后差商结果是一致的（两点插值就是线性近似），这里我们主要讨论三点形式的拉格朗日多项式所对应的数值微分结果。

根据前面插值算法部分的内容，我们知道三点形式的拉格朗日多项式为：

$$\mathrm{L(x)}=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f(x_0)+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1)+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

一般来说，我们有：$x_2-x_1=x_1-x_0=\triangle x$，那么：

$$\mathrm{L(x)}=\frac{(x-x_1)(x-x_2)}{2\Delta x^2}f(x_0)+\frac{(x-x_0)(x-x_2)}{-\Delta x^2}f(x_1)+\frac{(x-x_0)(x-x_1)}{2\Delta x^2}f(x_2)$$

显然，对于上式求导数比我们直接对函数 $f(x)$求导数要简单的多，我们有：

$$\mathrm{L}'(\mathrm{x})=\frac{x-x_1+x-x_2}{2\Delta x^2}f(x_0)+\frac{x-x_0+x-x_2}{-\Delta x^2}f(x_1)+\frac{x-x_0+x-x_1}{2\Delta x^2}f(x_2)$$

我们可以直接利用上式求解 $f'(x)$

对于这种算法的精度，针对特定值 $L'(x_1)$，利用上式可以得到：

$$\mathrm{L}'(x_1)=\frac{f(x_2)-f(x_0)}{2\Delta x}$$

这一结果与我们前面所讲的中心差商是一致的，所以针对这一特定点，三点插值算法在计算数值微分时的精度是$O(\Delta x^2)$。

利用泰勒公式我们可以严格的证明，三点插值算法计算数值微分都具有$O(\Delta x^2)$的精度。

<u>Python 导数的符号运算</u>：

使用 sympy 模块，python 能够对于函数的导数进行符号运算，如下图：

```
from sympy import diff, symbols
t = symbols('x', real=True)

def f(t):
    return t**5

for i in range(1,4):
    print(diff(f(t),t,i))
    print(diff(f(t),t,i).subs(t,i),i)
```

这里 symbols 用来定义自变量，然后使用 diff 对于函数的导数进行求解，其格式为：diff(函数，自变量，导数阶数)

结果为：

```
5*x**4
5 1
20*x**3
160 2
60*x**2
540 3
```

diff 函数使得我们能够简单的求解一些特别复杂函数的导数，比如：

```
from sympy import diff, symbols
t = symbols('x', real=True)
def f(t):
    return (t**5+3.2*t)/(t**2-2*t)*(t+2.0)+6.0*t**2.0
print(diff(f(t),t,1))
```

```
12.0*x**1.0 + (2 - 2*x)*(x + 2.0)*(x**5 + 3.2*x)/(x**2 - 2*x)**2 + (x + 2.0)*(5*x**4 + 3.2)/(x**2 - 2*x) + (x**5 + 3.2*x)/(x**2 - 2*x)
```

<u>Python 数值微分</u>：

使用 scipy.misc 模块下的 derivative 方法函数，其格式为：

$$\mathbf{derivative}(\text{函数}, \text{自变量}, \mathbf{dx} = \text{步长}, \mathbf{n} = \text{导数阶数})$$

比如：

```
from scipy.misc import derivative
def f(x):
    return x**5
for x in range(1, 4):
    print(derivative(f,x,dx=1e-6,n=1))
```

其结果为：

```
4.999999999866223
80.00000000230045
405.00000005749826
```

可以看出，上面的方法主要是针对我们已知函数表达式的情况，针对以离散数据给出的函数关系，我们可以先采用上一节所介绍的插值方法得到插值函数，然后在利用 derivative 进行导数计算

比如：

```
from scipy.misc import derivative
from scipy.interpolate import lagrange
from sympy import diff,symbols
x=[1,2,3,4,5,6]
y=[1,4,9,16,25,36]
f=lagrange(x,y)
print('function:\n',f)
t=symbols('x',real=True)
print('diff:\n',diff(f(t),t,1))
for x in range(1, 7):
    print('derivative=',derivative(f, x, dx=1e-6),'for x=',x)
```

结果显示如下：

```
function:
            5             3     2
5.551e-17 x + 1.421e-14 x + 1 x + 2.842e-14 x - 2.842e-14
diff:
 x*(x*(5.55111512312578e-17*x**2 + 1.4210854715202e-14) + 1.0) + x*(x*(5.5511151
2312578e-17*x**2 + 1.4210854715202e-14) + x*(1.66533453693773e-16*x**2 + 1.42108
54715202e-14) + 1.0) + 2.8421709430404e-14
derivative= 2.000000000002 for x= 1
derivative= 4.000000000115023 for x= 2
derivative= 6.000000001726846 for x= 3
derivative= 8.000000000230045 for x= 4
derivative= 10.00000000139778 for x= 5
derivative= 12.000000001677336 for x= 6
```