微积分基础

数学是所有工科专业的基石，其中微积分是高等数学的重点内容，也是解决现实中很多复杂问题的理论基础。在人工智能领域，微积分有着非常广泛的应用，包括图像处理、数据挖掘、深度学习、优化问题求解等。本章作为本书的开篇，将深入浅出地引领读者了解微积分的核心思想，并介绍微积分在人工智能领域中几个常见的应用案例，为后续章节的学习打下基础。

1.1 微积分的核心思想

导数和极限共同揭示了微积分的核心思想——无限分割后对一阶导数进行累加，做到“以直代曲”。为了深入浅出地阐明这一核心思想，本节将给出两个简单的案例——正弦函数面积和圆面积的累加计算，由此说明一阶导数在微积分中所发挥的作用。最后从泰勒展开式的角度，给出“以直代曲”的理论解释，循序渐进地引导读者学习微积分。

1.1.1 案例：正弦函数面积的累加计算


正弦函数面积的累加计算

当 $x\in[0,\pi]$ 时，正弦函数 $\sin x$ 与 x 轴围成一个封闭的区域，它的面积是

$$\int_0^{\pi}\sin x\,\mathrm{d}x=-\cos x\,|_0^{\pi}=-\cos\pi+\cos 0=2 \tag{1-1}$$

式(1-1)运算结果也就是这个区域面积的理论值，称为**解析解**。变量 x 在定义域$[0,\pi]$内被分成无穷多份，每份是 $\mathrm{d}x$，称为**微元**[1]。自变量 x 取不同的值，会带来 $\sin x$ 的变化。微积分的实质是分割后再累加，所以式(1-1)的积分，就是将 x 在$[0,\pi]$内无限分割，再将不同取值情况下的 $\sin x$ 做无限累加。

在计算机中实现积分计算，无法做到无限分割，毕竟物理硬件的容量有限，但是可以用有限的分割累加逼近积分的理论值，累加的结果称为**数值解**[2]。用 Python 语言编程实现这个封闭区域面积的分割累加，具体思路如下。

将定义域 $x\in[0,\pi]$ 均匀分割成 N 等份(观察 N 取 20、100、200、500 四种情况),每等份在 x 轴上的宽度为 $\Delta x=\frac{\pi}{N}$,即 $\Delta x=x_{i+1}-x_i(i=0,1,\cdots,N-1)$,每等份的面积是 $\sin x_i\cdot\Delta x$。将分割后的 N 等份面积累加,可大致得到该封闭区域的总面积,即

$$S=\sum_{i=0}^{N-1}\sin x_i\cdot\Delta x \tag{1-2}$$

代码实现如下:

【代码 1-1】

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
if__name__ == "__main__":
    mpl.rcParams['font.sans-serif'] = 'SimHei'
    N = 200              #N 分别取 20、100、200、500 进行编译运行
    begin_point = 0
    end_point = np.pi
    Delta_x = (end_point - begin_point)/N
    X = np.linspace(begin_point,end_point - Delta_x,N)
    Y = np.sin(X)
    Area = np.sum(Y) * Delta_x
    print(Area)
    plt.figure()
    plt.plot(X,Y,'r-')
    plt.bar(X,Y,width = Delta_x)
    plt.xlim([X.min() - 0.1,X.max() + 0.1])
    plt.ylim([Y.min() - 0.1,Y.max() + 0.1])
    plt.title(r'N = %d,面积 $ \approx $ %.5f' % (N,Area))
    plt.show()
```

运行代码 1-1 后,得到式(1-2)的累加结果,如图 1-1 所示。当 $N=\{20,100,200,500\}$ 时,保留小数点后 5 位,积分结果分别为 1.99589、1.99984、1.99996 和 1.99999。随着 N 的增大,累加结果越来越接近理论值。

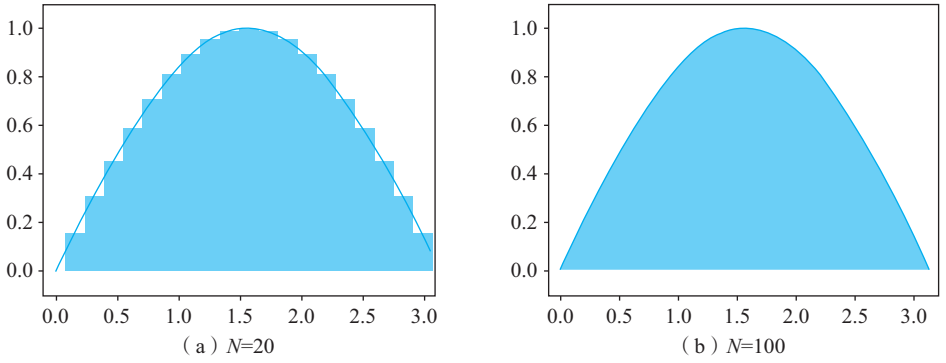
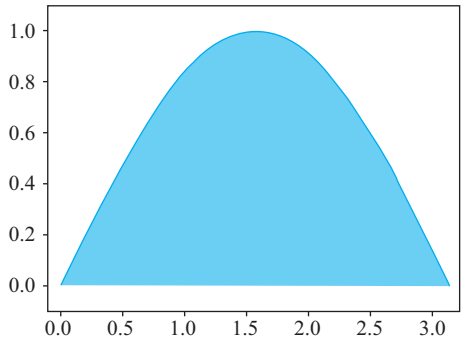
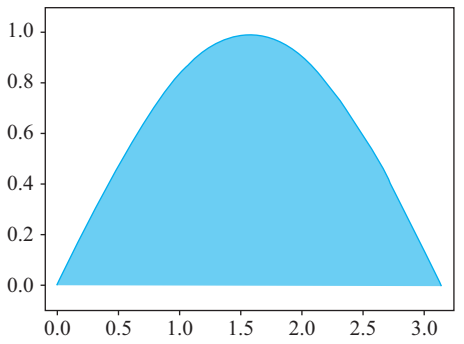


图 1-1 函数 $f(x)=\sin x$ 在 $x\in[0,\pi]$ 中的积分



（c）N=200



（d）N=500

图　1-1(续)

1.1.2　案例:圆面积的累加计算

本小节将给出圆面积的累加计算,旨在通过此案例深入分析一阶导数在微积分中所发挥的作用。由 1.1.1 小节可知,$\sin x$ 是 $-\cos x$ 的一阶导数,封闭区域的面积就是 $\sin x$ 在 $x\in[0,\pi]$内的积分结果。将$[0,\pi]$分割成 N 等份后对一阶导数进行累加,N 越大,即分割越精细,累加结果越精确。

众所周知,圆面积是关于 R 的函数(R 为圆的半径),这里记作 $S(R)=\pi R^2$,而周长 $2\pi R$ 是面积关于半径的一阶导数。类似于 1.1.1 小节,此案例把 r 看作变量,将其在$[0,R]$内等距分割成 n 小段,如图 1-2 所示。每小段长度都是 Δr,满足 $\Delta r=\dfrac{R}{n}$,即 $\Delta r=r_i-r_{i-1}(i=1,2,\cdots,n)$,其中 $r_0=0$,$r_1=\Delta r$,$r_2=2\Delta r$,以此类推,$r_n=n\Delta r=R$。

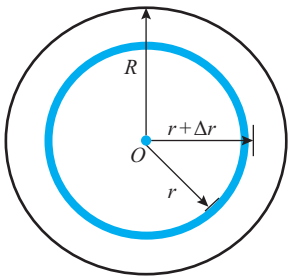


图 1-2　半径为 R 的圆

图 1-2 中粗圆环面积 $S(r+\Delta r)-S(r)=\pi(r+\Delta r)^2-\pi r^2=2\pi r\Delta r+\pi\Delta r^2$。当 $n\to\infty$时,$\Delta r\to 0$,而 $\Delta S(r)=S(r+\Delta r)-S(r)$与 Δr 的比值是

$$\lim_{\Delta r\to 0}\frac{\Delta S(r)}{\Delta r}=\frac{2\pi r\Delta r+\pi\Delta r^2}{\Delta r}=2\pi r \tag{1-3}$$

式(1-3)是 S 关于 r 的**一阶导数**。当 $r=r_i$ 时,其一阶导数为

$$\lim_{\Delta r\to 0}\frac{\Delta S(r)}{\Delta r}\bigg|_{r=r_i}=\frac{2\pi r_i\Delta r+\pi\Delta r^2}{\Delta r}=2\pi r_i \tag{1-4}$$

式(1-4)与图 1-2 一致,即 r_i 越大,圆环 $\Delta S(r_i)$的面积也越大。整个圆的面积就是将这 n 个大小不等的圆环累加起来的结果,当 $n\to\infty$时,圆面积为

$$S(R)=\pi R^2=\lim_{n\to\infty}\sum_{i=1}^{n}\Delta S(r_i)=\lim_{n\to\infty}\sum_{i=1}^{n}2\pi r_i\Delta r+\pi\lim_{n\to\infty}\sum_{i=1}^{n}\Delta r^2 \tag{1-5}$$

式(1-5)中,最右边一项 $\lim\limits_{n\to\infty}\sum\limits_{i=1}^{n}\Delta r^2=\lim\limits_{n\to\infty}n\left(\dfrac{R}{n}\right)^2=\lim\limits_{n\to\infty}\dfrac{R^2}{n}=0$,还原到微积分,即

$$S(R)=\lim_{n\to\infty}\sum_{i=1}^{n}\Delta S(r_i)=\int_0^R \mathrm{d}S(r)=\int_0^R 2\pi r\,\mathrm{d}r=\pi r^2\,\Big|_0^R=\pi R^2 \tag{1-6}$$

式(1-6)中圆面积 $S(R)$ 的积分思想是：将变量 r 在$[0,R]$内做无限分割，每一份是微元 $\mathrm{d}r$（即 $n\to\infty$时 $\Delta r\to 0$ 的情况），再将 r 在不同取值下的圆环 $\mathrm{d}S(r_i)=2\pi r_i\mathrm{d}r(i=1,2,\cdots,n)$做累加。这与 1.1.1 小节中正弦函数面积的累加计算的思想相同，即对一阶导数与变量微元的乘积结果做累加，二阶及以上导数忽略不计。

1.1.3 “以直代曲”的泰勒展开解释

微积分是将自变量在一定范围内无限分割后得到微元，再将自变量不同取值下的一阶导数与微元的乘积做累加。实际上，一阶导数只是函数的一部分，而非全部。本小节将把前两小节内容展开理论延伸，由泰勒展开式出发，给出函数的完整表达式，并以二次函数曲线为例，揭示一阶导数的实质——函数的线性变化（直线）部分。最后，本小节将直观展示一阶导数的累加如何实现曲线积分。

对于 n 阶可导的函数 $f(x)$，它在任一点 x_0 处的泰勒展开式为

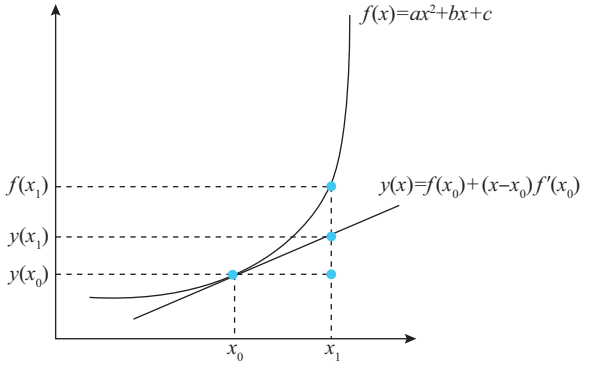
$$f(x)=f(x_0)+(x-x_0)f'(x_0)+\frac{1}{2!}(x-x_0)^2f''(x_0)+\cdots+\frac{1}{n!}(x-x_0)^nf^{(n)}(x_0) \tag{1-7}$$

式(1-7)中 x_0 已知，而 x 是变量。令 $\Delta x=x-x_0$，则 $f(x)-f(x_0)$是关于 Δx 的函数，即

$$f(x)-f(x_0)=\Delta xf'(x_0)+\frac{1}{2!}\Delta x^2f''(x_0)+\cdots+\frac{1}{n!}\Delta x^nf^{(n)}(x_0) \tag{1-8}$$

不管函数 $f(x)$几阶可导，也不管$|\Delta x|$有多大，式(1-8)都恒成立。感兴趣的读者可参阅文献[3]，其作者是已故的日本数学家高木贞治，他由拉格朗日中值定理出发，经过严格的理论推导，证明出泰勒展开式对一切函数皆成立。

由式(1-8)可知，一阶导数是函数的线性变化（直线）部分。举个例子，如图 1-3 所示，二次函数 $f(x)=ax^2+bx+c$ 呈曲线形状，令直线 $y(x)=f(x_0)+(x-x_0)f'(x_0)$，它是 $f(x)$在点 x_0 处的切线。



曲线积分

图 1-3　二次函数 $f(x)=ax^2+bx+c$ 示意图

运用式(1-8)对 $f(x)$做泰勒展开，再结合图 1-3，在 x_1 与 x_0 两点上（令 $\Delta x=x_1-x_0$），

函数值的差值 $f(x_1)-f(x_0)$包含两部分，即一阶部分和二阶部分。

一阶部分

$$y(x_1)-f(x_0)=f'(x_0)\Delta x$$

二阶部分

$$f(x_1)-y(x_1)=\frac{1}{2!}f''(x_0)\Delta x^2$$

由于 x_0 处一阶导数是 $f'(x_0)=2ax_0+b$，二阶导数是 $f''(x_0)=2a$，两部分合在一起即是 $f(x_1)-f(x_0)=(2ax_0+b)\Delta x+a\Delta x^2$。随着 x_1 不断向 x_0 靠近或远离，$|\Delta x|$会减小或增大，其中$(2ax_0+b)\Delta x$ 的变化与$|\Delta x|$呈线性关系；而 $a\Delta x^2$ 的变化与$|\Delta x|$呈二次非线性关系，这与式(1-5)最右边一项的情况完全一样。当$|\Delta x|\to 0$ 时，图 1-3 中 $f(x_1)-f(x_0)\to(2ax_0+b)\Delta x$，即一阶导数占主导作用。

图 1-3 展示的是一个二阶可导函数。对于 n 阶$(n>2)$可导的函数 $f(x)$，泰勒展开式共有 n 项，三阶部分的变化与$|\Delta x|$呈三次关系，四阶部分的变化与$|\Delta x|$呈四次关系，以此类推，当 $\Delta x\to 0$ 时，二阶导数及以上的部分都趋于 0，即

$$\lim_{\Delta x\to 0}\frac{\frac{1}{2!}\Delta x^2 f''(x_0)+\frac{1}{3!}\Delta x^3 f^{(3)}(x_0)\cdots+\frac{1}{n!}\Delta x^n f^{(n)}(x_0)}{\Delta x} \tag{1-9}$$

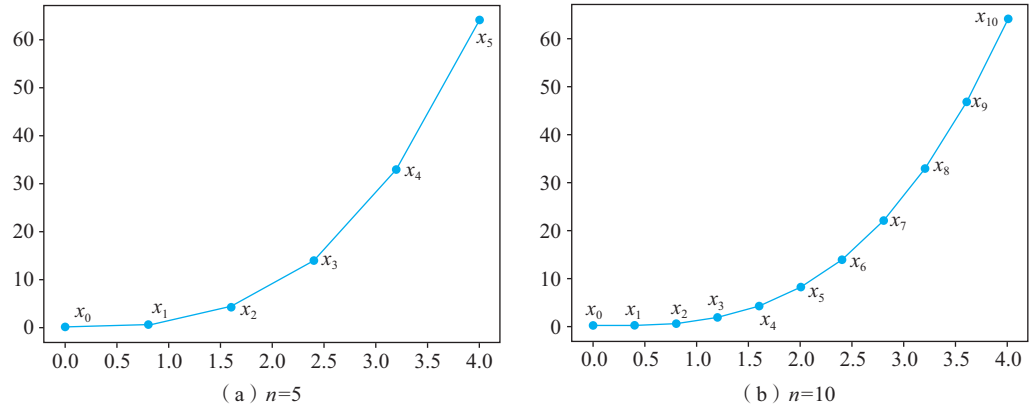
$$=\lim_{\Delta x\to 0}\Delta x\left(\frac{1}{2!}f''(x_0)+\frac{1}{3!}\Delta x f^{(3)}(x_0)+\cdots+\frac{1}{n!}\Delta x^{n-2}f^{(n)}(x_0)\right)=0$$

只有一阶导数 $f'(x_0)$对函数值变化起决定性作用，即

$$\lim_{\Delta x\to 0}f(x_1)-f(x_0)=f'(x_0)\Delta x \tag{1-10}$$

式(1-10)表明，如果将 x 在有限区间内分成 n 等份 $x_1,x_2,\cdots,x_n$，其中 $\Delta x=x_i-x_{i-1}(i=1,2,\cdots,n)$，当 $\Delta x=x_1-x_0\to 0$ 时，$f(x_1)-f(x_0)$与 Δx 呈线性关系，两者的比值是一阶导数 $f'(x_0)$。

实际上，曲线积分也采用了“以直代曲”的核心思想。以图 1-4 中函数 $f(x)=x^3+0.1$ 为例，若它在 $x\in[0,4]$的曲线长度为 l，则 l 可以近似地表达成 n 段直线累加之和。图 1-4(a)和(b)分别展示了 $n=5$ 和 $n=10$ 时，n 段直线对曲线 l 的逼近。很明显，后者的逼近效果比前者更好。



(a) n=5　　(b) n=10

图 1-4　$f(x)=x^3+0.1$ 在 $x\in[0,4]$内 n 段直线累加逼近函数曲线

根据勾股定理，若第 i 段直线的长度是 $l_i(i=1,2,\cdots,n)$，它满足

$$l_i^2=[f(x_i)-f(x_{i-1})]^2+(x_i-x_{i-1})^2$$

n 越大，Δx 越小，l_i 也就越短。当 $n\to\infty$ 时，l 就变成了对函数 $f(x)$ 的曲线积分，即

$$\begin{aligned}l&=\lim_{n\to\infty}\sum_{i=1}^{n}l_i=\lim_{n\to\infty}\sum_{i=1}^{n}\sqrt{[f(x_i)-f(x_{i-1})]^2+(x_i-x_{i-1})^2}\\&=\lim_{n\to\infty}\sum_{i=1}^{n}\sqrt{f'(x_i)^2+1}\,\Delta x\\&=\int_0^4\sqrt{f'(x)^2+1}\,\mathrm{d}x\end{aligned}$$

1.2 导数的近似估计

由 1.1 节可知，将变量的一阶导数做累加可以得到原函数。在此基础上，本节将做出相反的操作——给变量加上一点变动，根据函数值变化与变量变化的比值，近似地估计一阶导数和二阶导数，这种方法称作有限差分法。另外，本节也将给出有限差分法的两个应用案例——正弦函数导数的有限差分估计和图像边缘（轮廓）提取。

1.2.1 有限差分法

1.1 节阐明了微积分的核心思想——无限分割后将一阶导数累加，做到“以直代曲”。具体来说，它是将自变量 $x\in[a,b]$ 分割成 N 等份，当 $N\to\infty$ 时，则 $\Delta x=\dfrac{b-a}{N}\to 0$。如果 $f(x)$ 是 $F(x)$ 的一阶导数，不管 $F(x)$ 多少阶可导，式(1-11)均成立，即

$$F(b)-F(a)=\int_a^b f(x)\mathrm{d}x=\lim_{N\to\infty}\sum_{i=1}^{N}f(x_i)\Delta x\qquad(a\leqslant x_i\leqslant b)\tag{1-11}$$

这种只用一阶导数的逼近方法，在学术界称为**欧拉法**[2]。1.1.1 小节正弦函数积分的案例表明，Δx 越小，欧拉法的逼近越精确。

受到欧拉法的启发，可以进行相反的操作——将 x 取任一固定的值 x_0，稍作扰动变为 $x_1=x_0+\Delta x$。根据函数差值 $F(x_1)-F(x_0)$ 与变量差值 Δx 的比例关系，近似估计一阶导数 $f(x_0)$，这种方法称作**有限差分法**[4]。很多实际问题都可以采用有限差分法代替函数的真实导数，近似描述函数的真实变化情况。

常见的有限差分法有一阶导数的**前向差分估计**和**后向差分估计**，以及二阶导数的**中心差分估计**，计算式分别为

$$f'(x_i)\approx\frac{f(x_{i+1})-f(x_i)}{\Delta x}\tag{1-12}$$

$$f'(x_i)\approx\frac{f(x_i)-f(x_{i-1})}{\Delta x}\tag{1-13}$$

$$f''(x_i)\approx\frac{f(x_{i+1})-2f(x_i)+f(x_{i-1})}{\Delta x^2}\tag{1-14}$$

随着 $\Delta x \to 0$，一阶和二阶差分法都会收敛到相应导数，但是收敛速度不同[4]。如果固定 Δx，二阶差分估计的精度要高于一阶。究其原因，编著者认为，还是应该从泰勒展开式出发，追根溯源进行误差分析。

对于三次及以上可导的函数 $f(x)$，泰勒展开后保留前两阶导数，即

$$f(x_{i+1}) = f(x_i) + \Delta x f'(x_i) + \frac{1}{2}\Delta x^2 f''(x_i) + O\left(\sum_{j=3,4,5\cdots} \Delta x^j\right) \tag{1-15}$$

将式(1-15)变形，对照式(1-12)，一阶前向差分估计的误差为

$$\frac{f(x_{i+1}) - f(x_i)}{\Delta x} - f'(x) = \frac{1}{2}\Delta x f''(x_i) + O\left(\sum_{j=2,3,4\cdots} \Delta x^j\right) = O\left(\sum_{j=1,2,3\cdots} \Delta x^j\right) \tag{1-16}$$

二阶中心差分估计的误差分为两步，即

$$\frac{f(x_{i+1}) - f(x_i)}{\Delta x^2} = \frac{f'(x_i)}{\Delta x} + \frac{1}{2}f''(x_i) + O\left(\sum_{j=1,2,3\cdots} \Delta x^j\right) \quad ①$$

$$\frac{f(x_{i-1}) - f(x_i)}{\Delta x^2} = -\frac{f'(x_i)}{\Delta x} + \frac{1}{2}f''(x_i) + O\left(\sum_{j=1,2,3\cdots} (-1)^j \Delta x^j\right) \quad ②$$

注意： ②中，$f(x_i) - f(x_{i-1}) = \Delta x$，所以 $f(x_{i-1}) - f(x_i) = -\Delta x$。

将①②相加，对照式(1-14)可得

$$\frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2} - f''(x_i) = 2O\left(\sum_{j=2,4,6\cdots} \Delta x^j\right) \tag{1-17}$$

对比式(1-16)和式(1-17)，不难发现二阶中心差分估计的误差仅仅是 Δx 的偶数次方的阶项之和，而一阶前向差分估计的误差是关于 Δx 的所有阶次之和，故前者精度更高。至于式(1-13)的一阶后向差分估计误差，请读者仿照上述内容自行推导。

1.2.2 案例：正弦函数导数的有限差分估计

文献[5]将正弦函数 $f(x) = \sin x$ 看作原函数，在一个周期 $x \in [0, 2\pi]$ 内均匀采样 $N = 150$ 个点，分别采用前向差分估计法和二阶中心差分估计法，通过 C++与计算统一设备架构(compute unified device architecture，CUDA)相结合的并行编程方式，估算一阶导数 $f'(x) = \cos x$ 和二阶导数 $f''(x) = -\sin x$。

本案例将对此进行拓展，取 $N = 20$、50 和 100$\left(\text{即 } \Delta x = \frac{2\pi}{N}\right)$，计算这三种情况下的估计误差，并观察误差与 Δx 之间的关系。估计采用均方误差(mean square error，MSE)，即 $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$，其中 y_i 表示第 i 个点的理论值，$\hat{y}_i$ 是估计值。

代码实现如下：

【代码 1-2】

```
import numpy as np
```

```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
config = {
    "font.family" :'serif',
    "font.size" :10,
    "mathtext.fontset" :'stix',
    "font.serif" :['SimSun'],
}
mpl.rcParams.update(config)
if __name__ == '__main__':
    N = 100
    Delta = 2 * np.pi/N
    Result = np.zeros((6,N),dtype = float)
    for i in range(N):
        Result[0,i] = np.sin(Delta * i)          #原函数
        Result[1,i] = np.cos(Delta * i)          #真实的一阶导数
        Result[2,i] = (np.sin(Delta * (i + 1)) - np.sin(Delta * i))/Delta
                                                 #一阶导数的前向差分估计
        Result[3,i] = (np.sin(Delta * i) - np.sin(Delta * (i - 1)))/ Delta
                                                 #一阶导数的后向差分估计
        Result[4,i] = -np.sin(Delta * i)         #真实的二阶导数
        Result[5,i] = (np.sin(Delta * (i + 1)) - 2 * np.sin(Delta * i) + np.sin(Delta *
(i - 1)))/(Delta * * 2)                          #二阶导数的中心差分估计
    D1 = Result[1] - Result[2]
    MSE_1 = np.mean(D1 * * 2)
    print(r'一阶前向差分的误差是:%.10f' % MSE_1)
    D2 = Result[1] - Result[3]
    MSE_2 = np.mean(D2 * * 2)
    print(r'一阶后向差分的误差是:%.10f' % MSE_2)
    D3 = Result[4] - Result[5]
    MSE_3 = np.mean(D3 * * 2)
    print(r'二阶中心差分的误差是:%.10f' % MSE_3)
    fig = plt.figure(figsize = (6,4))
    plt.plot(np.linspace(0,2 * np.pi,N),Result[0],'k-')
    plt.plot(np.linspace(0,2 * np.pi,N),Result[1],'b--')
    plt.plot(np.linspace(0,2 * np.pi,N),Result[2],'c-.')
    plt.plot(np.linspace(0,2 * np.pi,N),Result[3],'g-x')
    plt.plot(np.linspace(0,2 * np.pi,N),Result[4],'r-<')
    plt.plot(np.linspace(0,2 * np.pi,N),Result[5],'m-d')
    plt.legend([r'$ f(x) $ ',r'$ cos(x) $ ',r'Diff_1',r'Diff_2',r'$ - sin(x) $ ',r'Diff_3'],loc =
0,bbox_to_anchor = (1.05,1),borderaxespad = 0,fontsize = 12)
    plt.xlim([0,2 * np.pi])
    plt.ylim([-1,1])
    plt.xlabel(r'$ x $ ',fontsize = 12)
    fig.subplots_adjust(right = 0.75)
    plt.show()
```

当 $N = 20$、50、100 时，代码 1-2 的运行效果分别如图 1-5～图 1-7 所示，其中 Diff_1、Diff_2 和 Diff_3 分别表示 N 个点的一阶前向差分估计、一阶后向差分估计和二阶中心差分估计的结果。

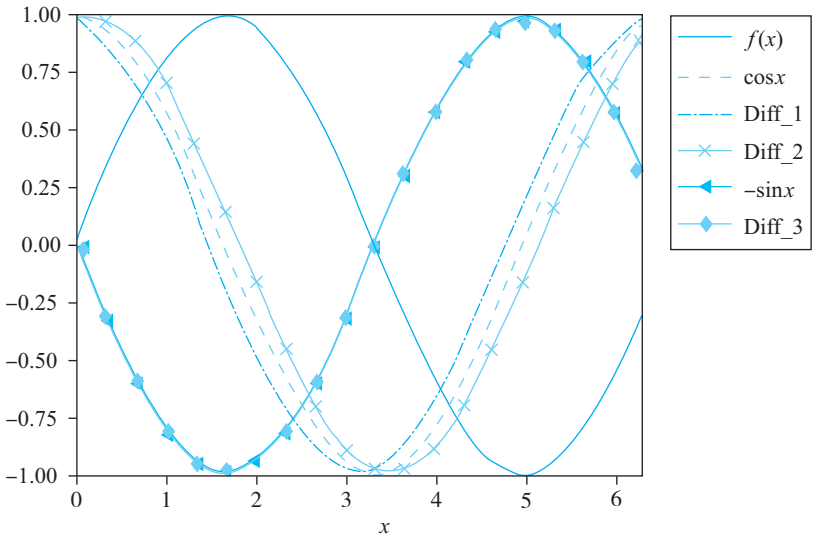


图 1-5　正弦函数的有限差分估计（$N=20$）

注：一阶前向差分的误差是 0.0122695270
　　一阶后向差分的误差是 0.0122695270
　　二阶中心差分的误差是 0.0000336008

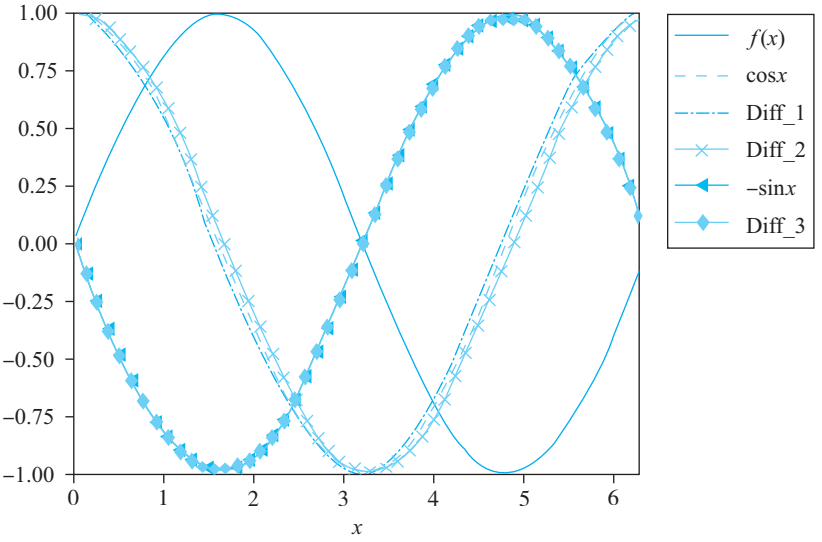


图 1-6　正弦函数的有限差分估计（$N=50$）

注：一阶前向差分的误差是 0.0019721898
　　一阶后向差分的误差是 0.0019721898
　　二阶中心差分的误差是 0.0000008649

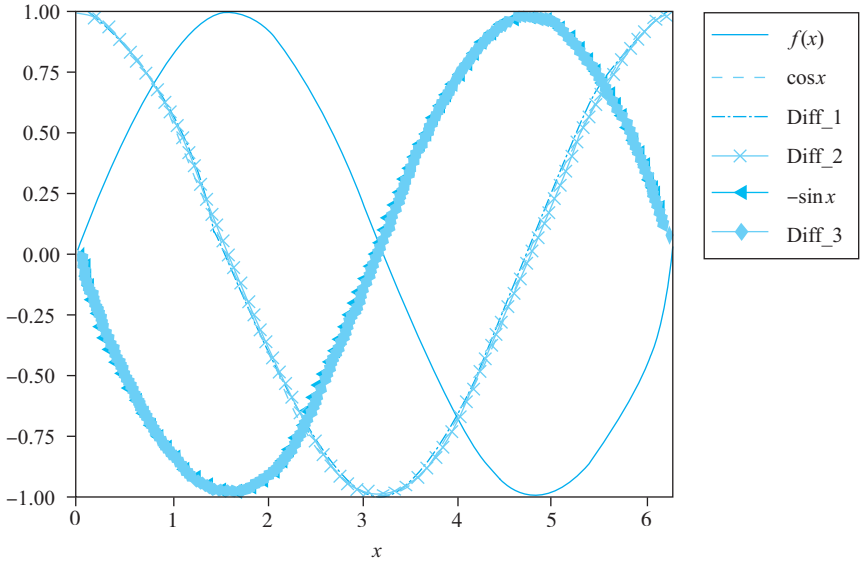


图 1-7　正弦函数的有限差分估计($N=100$)

注:一阶前向差分的误差是 0.0004933720
一阶后向差分的误差是 0.0004933720
二阶中心差分的误差是 0.0000000541

观察图 1-5～图 1-7,随着 N 的增大,Δx 在减小,这三种差分估计方法均会越来越精确。对于一阶导数,虽然前向和后向差分估计值不同,但是这 N 个点的均方误差完全一样。与此同时,二阶中心差分估计的精度要远远高于一阶的两种情况,这与 1.2.1 小节的误差分析结果一致。

1.2.3　案例:图像边缘(轮廓)提取

在人工智能领域,导数的近似估计被广泛应用在图像处理中,最典型的应用就是图像轮廓提取,也称作边缘检测。这里,首先需要了解图像在计算机中是如何表示的。图 1-8(a)是我们肉眼所见的阿拉伯数字 2 的灰度图像,分辨率为 16×16,即总共 256 个像素点。在计算机中,这张图像是一个 16×16 的二维矩阵,如图 1-8(b)所示。

通常,黑色像素点在计算机中用 0 表示,白色像素点用 255 表示。介于黑白之间的像素点即为灰色,像素值用 0～255 的整数表示,越接近白色的浅灰,像素值越接近 255,反之越接近 0。在图像处理中,将图 1-8(b)所示的二维矩阵看成一个平面坐标系,水平和垂直方向分别看成 x 轴和 y 轴,每个像素点都对应一个坐标位置(x,y),其像素值就是这个坐标位置上的函数值 $F(x,y)$。例如,第 1 行第 5 列的像素值是 76,即 $F(1,5)=76$。边缘(轮廓)提取会用到图像的导数(梯度)信息,它反映了在 x 轴和 y 轴方向上相邻两个像素值的变化程度。这其实延续了 1.2.1 小节有限差分法的思想,即根据函数值的变化近似估计导数大小。因此,边缘(轮廓)提取实际上就是遍历整个图像,找寻像素值发生突变(梯度大)的地方。

在现有的图像边缘检测提取方法中,较为常见的是 Sobel 检测、Laplacian 检测、Prewitt