## 2.3.1 编程微积分

微积分（Calculus），是研究极限、微分、积分和无穷级数等的一个数学分支。本质的讲，微积分是一门研究变化的学问。在本书中多处涉及到微积分的知识，例如阐述回归部分对残差平方和关于回归系数求微分，另微分结果为0解方程组得回归系数值，构建回归方程；在梯度下降法中梯度就是分别对每个变量求偏微分；本部分SIR传播模型的阐述中则通过建立易感人群、恢复人群和受感人群的微分方程建立SIR传播模型。可见微积分在数据分析中具有重要的作用，因此有必要以代码的途径结合图表表述阐释微积分的基础知识，为相关数据分析预备。以《7天搞定微积分》[1]和《漫画微积分》[2]的内容为讲述的结构，主要使用[SymPy（Calculus）](https://docs.sympy.org/latest/tutorial/calculus.html)①库解释微积分。

### 2.3.1.1 导数（Derivative）与微分（Differentiation）

* 导数

导数是用来分析变化的，即曲线（函数图像）在某点处的斜率，表示倾斜的程度。对于直线函数求导会得到直线的斜率，对曲线函数求导则能得到各点的斜率（即瞬间斜率）。下述代码使用SymPy库的diff方法计算了曲线上采样点各处的斜率，具体的过程是先建立曲线图形的函数表达式为：， 由diff方法关于x求微分结果为：，通过该微分方程，给定任意一点的横坐标，就可以计算获得曲线对应点的斜率。为了清晰表述采样点各处斜率的变化情况，由导数（斜率值）derivative\_value，采样点横坐标sample\_x，采样点纵坐标sample\_y，假设采样点横坐标的固定变化值为，计算处的纵坐标，从而绘制各点处的切线。为了能够清晰的看到各个采样点处导数的倾斜程度，即变化趋势的强弱，对齐所有切线原点于横坐标上，保持各点横轴变化量不变，计算各个结束点的纵坐标，通过向量长度的变化可以确定各点变化趋势的大小，通过向量方向的变化可以确定各点变化趋势的走势。

* 极限

上述计算斜率的方法是直接使用了SymPy库，为了更清晰的理解计算的过程，需要首先了解什么是极限。极限可以描述一个序列的指标愈来愈大时，数列中元素的性质变化趋势，也可以描述函数的自变量接近某一个值时，相对应函数值变化的趋势。例如对于数列（sequence） nanf(x)c*xcf(x)Lxcf(x)L*n无限接近1时，无限接近1-1，即无限接近0。又如，即无限接近1时，无限接近2。

如果要求函数图形点的斜率，点的坐标为，将点向右移动，即横向长度差，则纵向长度差为，过点的斜率，即在处的导数（导函数）为：，即：y(f(x))xd x的导数”的求导计算。

已知曲线图形的函数表达式为，在SymPy下建立表达式，根据上述求导极限方程可以得到函数在点处的求导表达式（导函数），即下述代码中变量limit\_x\_0\_expr为， dxx $时的极限值即为导数/斜率，计算结果为。这与使用SymPy的diff方法求得的关于求导方程，代入的结果一致。

* 误差率

获得导函数，可以求任一点的斜率，例如在点处斜率为。欲建立该点处的切线方程，需要求得截距，根据，其中为斜率，为截距，则有，此时，而斜率已知，，求得截距则可建立该点的切线方程。误差率则是以为起点进行变化时，和值之间的差异，占的变化量的百分比，即。离越近，误差率越小。所谓近似成一次函数，就是令原函数的误差率局部为0的情况。所以在讨论局部性质时，可以用一次函数替代原函数进而推导出正确的结论。

* 求导的基本公式

1. ;
2. ;
3. (af(x))‘=a f’ (x) # 常系数微分
4. ；# 和的微分
5. ( x^{n} )’ =n x^{n-1} ; # 幂函数的导数
6. {f(x)g(x)}‘= f(x)’g(x) + f(x)g(x)’; # 积的微分
7.  # 商的微分
8. {g(f(x))}’ = g’(f(x)) f’ (x) # 复合函数的微分
9.  # 反函数的微分方程

* 微分

对于微分的理解可以拓展为函数图像中，某点切线的斜率和函数的变化率。微分是对函数的局部变化率的一种线性描述，其可以近似的描述当函数自变量的取值足够小的改变时，函数的值是怎样变化的。

* 由“微分=0”可知极值

极大点和极小点是函数增减性发生变化的地方，对研究函数的性质来说是很重要的。极大点、极小点常常会变成最大点、最小点，是求解某些（最优解）问题时十分关键的点。极值条件：在处为极大点或极小点，则有。即求极大点和极小点，只需找到满足的即可。

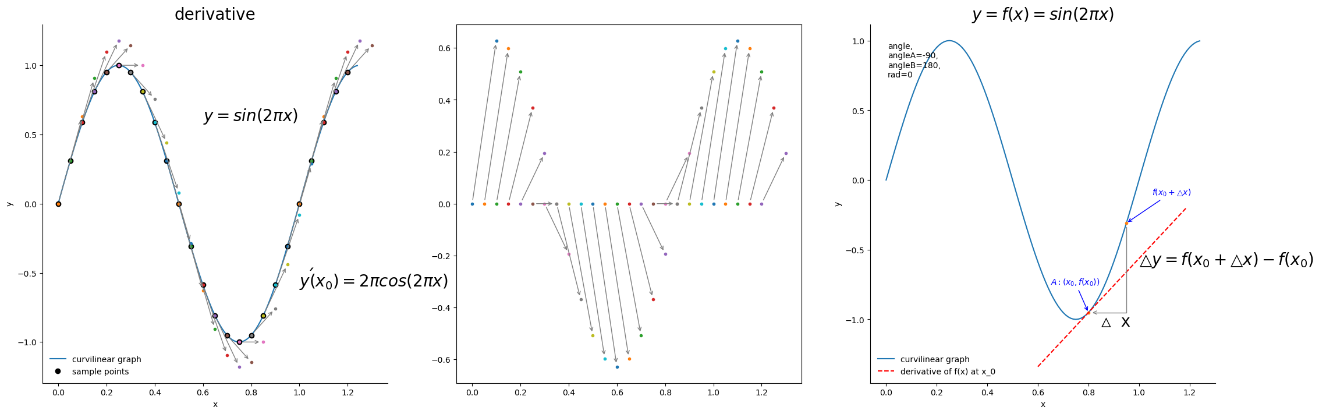
增减性的判断条件：当时，所近似的一次函数在处呈现递增的趋势，因此可知同样呈现递增趋势；同样当时，处于下降的状态，即不在顶端也不在谷底。

* 平均值定理

对于来说，存在一个*ζ*a<*ζ*<bf(b)= f’ (*ζ*) (b-a)+f(a)。

import numpy as np  
import matplotlib.pyplot as plt  
import sympy  
from sympy import diff,pprint,limit  
  
x=sympy.symbols('x')  
curvilinear\_expr=sympy.sin(2\*sympy.pi\*x) # 定义曲线函数  
  
# A-使用SymPy库diff方法求导  
derivative\_curvilinear\_expr=diff(curvilinear\_expr,x) # curvilinear\_expr 关于x求微分/导数方程  
print("curvilinear\_expr 关于x求微分/导数方程:")  
pprint(derivative\_curvilinear\_expr,use\_unicode=True)   
  
curvilinear\_expr\_=sympy.lambdify(x,curvilinear\_expr,"numpy")  
derivative\_expr=sympy.lambdify(x,derivative\_curvilinear\_expr,"numpy")  
  
t=np.arange(0.0,1.25,0.01)  
y=curvilinear\_expr\_(t)  
  
fig, axs=plt.subplots(1,3,figsize=(26,8))  
axs[0].plot(t, y,label="curvilinear graph")  
axs[0].set\_title(r'derivative', fontsize=20)  
axs[0].text(1, -0.6, r'$y\'(x\_{0}) =2 \pi cos(2 \pi x)$', fontsize=20) # $y'(x\_{0}) =2 \pi cos(2 \pi x)$ ; $\sum\_{i=0}^\infty x\_i$  
axs[0].text(0.6, 0.6, r'$y=sin(2 \pi x)$',fontsize=20)  
axs[0].set\_xlabel('x')  
axs[0].set\_ylabel('y')  
axs[0].spines['right'].set\_visible(False)  
axs[0].spines['top'].set\_visible(False)  
  
# 采样原点  
sample\_x=t[::5]  
sample\_y=curvilinear\_expr\_(sample\_x)  
axs[0].plot(sample\_x,sample\_y,'o',label='sample points',color='black')  
  
# 采样终点  
derivative\_value=derivative\_expr(sample\_x) # 求各个采样点的导数（斜率）  
delta\_x=0.1 #x向变化量  
sample\_x\_endPts=sample\_x+delta\_x  
sample\_y\_endPts=derivative\_value\*delta\_x+sample\_y  
  
def demo\_con\_style\_multiple(a\_coordi,b\_coordi,ax,connectionstyle):  
 '''  
 function - 在matplotlib的子图中绘制多个连接线  
 reference：matplotlib官网Connectionstyle Demo :https://matplotlib.org/3.3.2/gallery/userdemo/connectionstyle\_demo.html#sphx-glr-gallery-userdemo-connectionstyle-demo-py  
  
 Params:  
 a\_coordi - 起始点的x，y坐标；tuple  
 b\_coordi - 结束点的x，y坐标；tuple  
 ax - 子图；ax(plot)  
 connectionstyle - 连接线的形式；string  
   
 Returns:  
 None  
 '''  
 x1, y1=a\_coordi[0],a\_coordi[1]  
 x2, y2=b\_coordi[0],b\_coordi[1]  
  
 ax.plot([x1, x2], [y1, y2], ".")  
 for i in range(len(x1)):  
 ax.annotate("",  
 xy=(x1[i], y1[i]), xycoords='data',  
 xytext=(x2[i], y2[i]), textcoords='data',  
 arrowprops=dict(arrowstyle="<-", color="0.5",  
 shrinkA=5, shrinkB=5,  
 patchA=None, patchB=None,  
 connectionstyle=connectionstyle,  
 ),  
 )  
   
demo\_con\_style\_multiple((sample\_x,sample\_y),(sample\_x\_endPts,sample\_y\_endPts),axs[0],"arc3,rad=0.")   
demo\_con\_style\_multiple((sample\_x,sample\_y\*0),(sample\_x\_endPts,sample\_y\_endPts-sample\_y),axs[1],"arc3,rad=0.")   
  
# B-使用极限方法求导  
axs[2].set\_title(r'$y=f(x)=sin(2 \pi x)$', fontsize=20)  
axs[2].plot(t, y,label="curvilinear graph")  
import util\_A  
dx=0.15  
x\_0=0.8  
util\_A.demo\_con\_style((x\_0,curvilinear\_expr\_(x\_0)),(x\_0+dx,curvilinear\_expr\_(x\_0+dx)),axs[2],"angle,angleA=-90,angleB=180,rad=0")   
axs[2].text(x\_0+0.05, curvilinear\_expr\_(x\_0)-0.1, "△ x", family="monospace",size=20)  
axs[2].text(x\_0+0.2,curvilinear\_expr\_(x\_0+dx)-0.3, r"$△ y=f(x\_{0}+△ x)-f(x\_{0})$", family="monospace",size=20)  
color='blue'  
axs[2].annotate(r'$A:(x\_{0},f(x\_{0}))$', xy=(x\_0, curvilinear\_expr\_(x\_0)), xycoords='data',xytext=(x\_0-0.15, curvilinear\_expr\_(x\_0)+0.2), textcoords='data',weight='bold', color=color,arrowprops=dict(arrowstyle='->',connectionstyle="arc3",color=color))  
axs[2].annotate(r'$f(x\_{0}+△ x)$', xy=(x\_0+dx, curvilinear\_expr\_(x\_0+dx)), xycoords='data',xytext=(x\_0+dx+0.1, curvilinear\_expr\_(x\_0+dx)+0.2), textcoords='data',weight='bold', color=color,arrowprops=dict(arrowstyle='->',connectionstyle="arc3",color=color))  
axs[2].set\_xlabel('x')  
axs[2].set\_ylabel('y')  
axs[2].spines['right'].set\_visible(False)  
axs[2].spines['top'].set\_visible(False)  
  
d=sympy.symbols('d')  
limit\_x\_0\_expr=(curvilinear\_expr.subs(x,x\_0+d)-curvilinear\_expr.subs(x,x\_0))/d # 函数f(x)在点x\_0处的极限方程  
print("f(x)在x\_0处的求导方程：")  
pprint(limit\_x\_0\_expr)  
limit\_x\_0=limit(limit\_x\_0\_expr,d,0)  
print(r"f(x)在x\_0处的导数/斜率为：")  
pprint(limit\_x\_0)  
  
t\_=np.arange(0.6,1.2,0.01)  
intercept=curvilinear\_expr\_(x\_0)-limit\_x\_0\*x\_0  
axs[2].plot(t\_,limit\_x\_0\*t\_+intercept,'--r',label="derivative of f(x) at x\_0") # limit\_x\_0\*t\_+intercept即为x\_0处的切线方程  
  
# C-（x\_0）误差率  
gx=limit\_x\_0\*x+intercept  
x\_1=x\_0+dx  
err=(curvilinear\_expr.subs(x,x\_1)-gx.subs(x,x\_1))/dx  
print("x\_0点导函数，在x\_1点的误差率：%.2f"%err)  
  
axs[0].legend(loc='lower left', frameon=False)  
axs[2].legend(loc='lower left', frameon=False)  
plt.show()

curvilinear\_expr 关于x求微分/导数方程:  
2⋅π⋅cos(2⋅π⋅x)  
f(x)在x\_0处的求导方程：  
sin(π⋅(2⋅d + 1.6)) - sin(1.6⋅π)  
───────────────────────────────  
 d   
f(x)在x\_0处的导数/斜率为：  
 π √5⋅π  
- ─ + ────  
 2 2   
x\_0点导函数，在x\_1点的误差率：2.34



### 2.3.1.2 积分（Integrate）

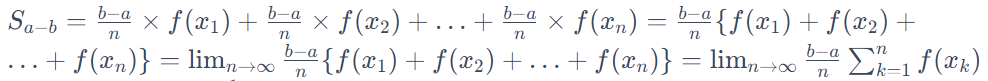
积分是导数的逆运算（针对计算方式而言），利用积分可以求出变化的规律和不规整图形的面积。积分和导数通常配套使用，合称为微积分。积分通常分为定积分和不定积分两种。对于定积分，给定一个正实值函数，在一个实数区间上的定积分为 *o*xy x=a，x=bxf(x)f(x)F(x)f(x)F(x)f(x)F(x)+Cf  f(x)x f(x)yxf(x)f(x)f(x)f(x)的原函数”，这三种表达方式意思相同。

是基础函数，f(x)f’(x)f(x) 表示不定积分。

* 不定积分、定积分和面积

实际上表示将进行（积分），而是“summation（合计）”的开头字母的变形，表示对的合计之意。是“与对应的轴坐标”，表示延轴的最小增量。因此就是变化横轴增量下矩形的面积。当对所有位于区间下变化增量为的矩形面积求积分（合计）后（宽度极限小的长方形的集合），即为区间为的横轴与曲线围合的面积。

* 区分求积法

对于函数，给定区间，假设进行次分割，长方形从左向右依次为x1 ,x2,x3, ……,xk,……,xn  **)。

下述代码在使用区分求积法时，给定的函数为f’ (x)= x² 的导函数），已知区间为，依据上述公式则有。在使用极限计算求和公式时，需要使用doit()方法计算不被默认计算的对象（极限、积分、求和及乘积等），否则不能计算极限。

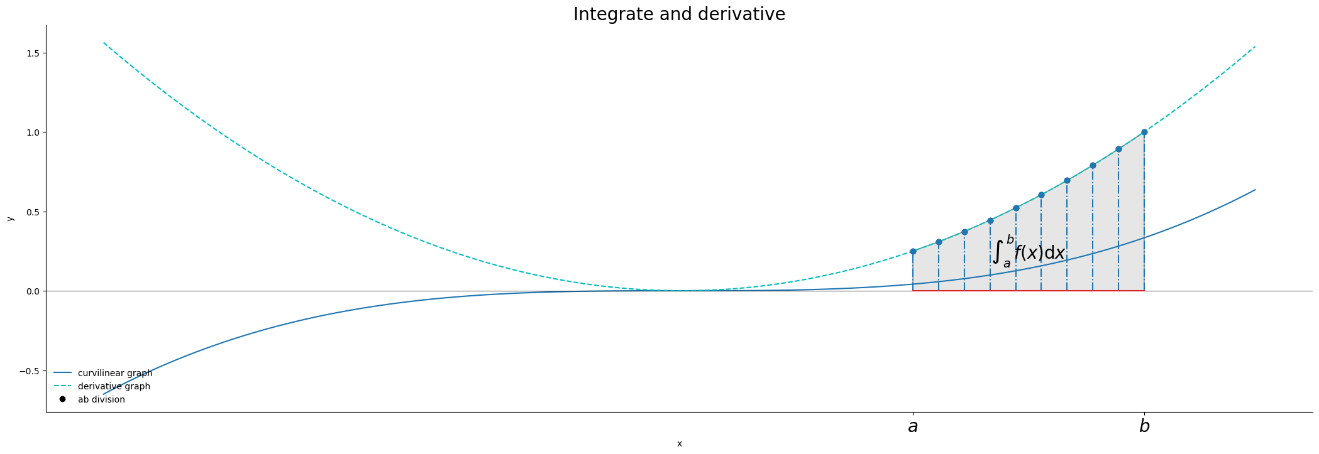
定积分求给定区间的面积，直接使用SymPy提供的integrate方法，给定区间计算结果约为0.29，与区分求积法计算结果相同。

* 换元积分公式

对于，将变量替换为一个关于变量的函数，即时，对于的定积分的值，用表示为：。

import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.patches import Polygon  
import sympy  
from sympy import diff,pprint,integrate,oo,Sum,limit # oo 为正无穷  
  
x,n,k=sympy.symbols('x n k')  
curvilinear\_expr=x\*\*3/3 # 定义曲线函数  
  
derivative\_curvilinear\_expr=diff(curvilinear\_expr,x) # curvilinear\_expr 关于x求微分/导数方程  
print("曲线函数curvilinear\_expr为：")  
pprint(curvilinear\_expr)  
print("curvilinear\_expr 关于x求微分/导数，导函数为:")  
pprint(derivative\_curvilinear\_expr,use\_unicode=True)   
  
integrate\_derivative\_curvilinear\_expr=integrate(derivative\_curvilinear\_expr,x)  
print("curvilinear\_expr导函数的积分：")  
pprint(integrate\_derivative\_curvilinear\_expr)  
  
curvilinear\_expr\_=sympy.lambdify(x,curvilinear\_expr,"numpy")  
derivative\_expr=sympy.lambdify(x,derivative\_curvilinear\_expr,"numpy")  
  
t=np.arange(-1.25,1.25,0.01)  
y=curvilinear\_expr\_(t)  
  
fig=plt.figure(figsize=(26,8))  
ax=fig.add\_subplot(111)  
ax.plot(t, y,label="curvilinear graph")  
ax.plot(t, derivative\_expr(t),'--c',label="derivative graph")  
ax.set\_title(r'Integrate and derivative', fontsize=20)  
ax.set\_xlabel('x')  
ax.set\_ylabel('y')  
ax.spines['right'].set\_visible(False)  
ax.spines['top'].set\_visible(False)  
ax.axhline(0,color='black',linewidth=0.5)  
  
a,b=0.5,1.0 # 定义区间  
ix=np.linspace(a,b,10)  
iy=derivative\_expr(ix)  
ax.plot(ix,iy,'o',label='ab division',color='black')  
verts=[(a, 0)]+list(zip(ix, iy))+[(b, 0)]  
poly=Polygon(verts, facecolor='0.9', edgecolor='0.5') # 绘制面积区域  
ax.add\_patch(poly)  
plt.text(0.5 \* (a + b), 0.2, r"$\int\_a^b f(x)\mathrm{d}x$",horizontalalignment='center', fontsize=20)  
ax.set\_xticks((a, b))  
ax.set\_xticklabels(('$a$', '$b$'))  
ax.stem(ix,iy,linefmt='-.')  
  
# A-使用区分求积法求取面积  
Sum\_ab=(b-a)/n\*Sum(derivative\_curvilinear\_expr.subs(x,a+k\*(b-a)/n),(k,0,n-1)) # 面积求和公式  
print("所有长方形面积之和的公式：\n")  
pprint(Sum\_ab)  
print("doit():\n")  
pprint(Sum\_ab.doit())  
S\_ab=limit(Sum\_ab.doit(),n,oo)  
print("区分求积法计算的面积=",S\_ab)  
  
# B-使用定积分求面积（函数）  
S\_ab\_integrate=integrate(derivative\_curvilinear\_expr,(x,a,b))  
print("定积分计算的面积=",S\_ab\_integrate)  
  
ax.legend(loc='lower left', frameon=False)  
plt.xticks(fontsize=20)  
plt.show()

曲线函数curvilinear\_expr为：  
 3  
x   
──  
3   
curvilinear\_expr 关于x求微分/导数，导函数为:  
 2  
x   
curvilinear\_expr导函数的积分：  
 3  
x   
──  
3   
所有长方形面积之和的公式：  
  
 n - 1   
 \_\_\_\_   
 ╲   
 ╲ 2  
 ╲ ⎛k ⎞   
0.5⋅ ╱ 0.25⋅⎜─ + 1⎟   
 ╱ ⎝n ⎠   
 ╱   
 ‾‾‾‾   
 k = 0   
───────────────────────  
 n   
doit():  
  
 ⎛ ⎛ 2 ⎞ ⎛ 3 2 ⎞⎞  
 ⎜ ⎜n n⎟ ⎜n n n⎟⎟  
 ⎜ 0.5⋅⎜── - ─⎟ 0.25⋅⎜── - ── + ─⎟⎟  
 ⎜ ⎝2 2⎠ ⎝3 2 6⎠⎟  
0.5⋅⎜0.25⋅n + ──────────── + ──────────────────⎟  
 ⎜ n 2 ⎟  
 ⎝ n ⎠  
────────────────────────────────────────────────  
 n   
区分求积法计算的面积= 7/24  
定积分计算的面积= 0.291666666666667



### 2.3.1.3 泰勒展开式（Taylor expansion）

上述阐释误差率时，在曲线局部使用一次函数替代（近似）曲线，例如对于函数，令，则在距很近的地方，能够将近似为。使用一次函数其误差率相对较高，如果近似为二次函数或者三次函数，是否可以降低误差率？泰勒展开就是将复杂的函数改写成多项式。

一般函数（要能够无限次的进行微分），则可以表示成如下形式， f(x)= f(x)x=0f(x)例如对于函数 x=0.1，及右边为x=2-1 < x < 1x才成立。

* 泰勒展示的求解方法——确定系数

对于       式(1)

首先，带入，由f(0)= a0 a0f(0)。——（A）

然后对式(1)进行微分， f’ (x)=       式(2)

将带入式(2),由f’ (0)= a1 a1f’ (0)。——（B）

继续对式(2)进行微分， f’’ ’ (x)=       式(3)

代入，可知二次系数为。——（C）

对式(3)进行微分，，

由此可知，三次系数为。

持续进行这种运算，次微分后，就应该得到，nf(x)。

由此可知，次系数为。 ，读作“的阶乘”，它表示。

对进行泰勒展开，便有

上述公式中，

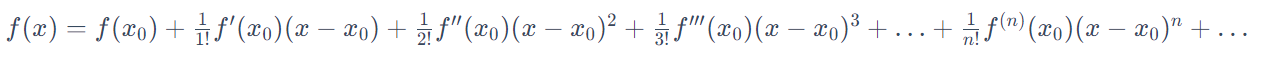
<——0次的常数项， 即 a\_{0} =f(0) ——（A）

<——1次项， 即 a1 =f’ (0)——（B）

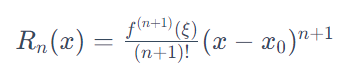
<——2次项， 即 a2 =——（C）

<——3次项， 即 ——（D）

泰勒展开，不一定非要从的地方开始，也可以从处开始，此时只需要将替换为,展开方法同上，得：

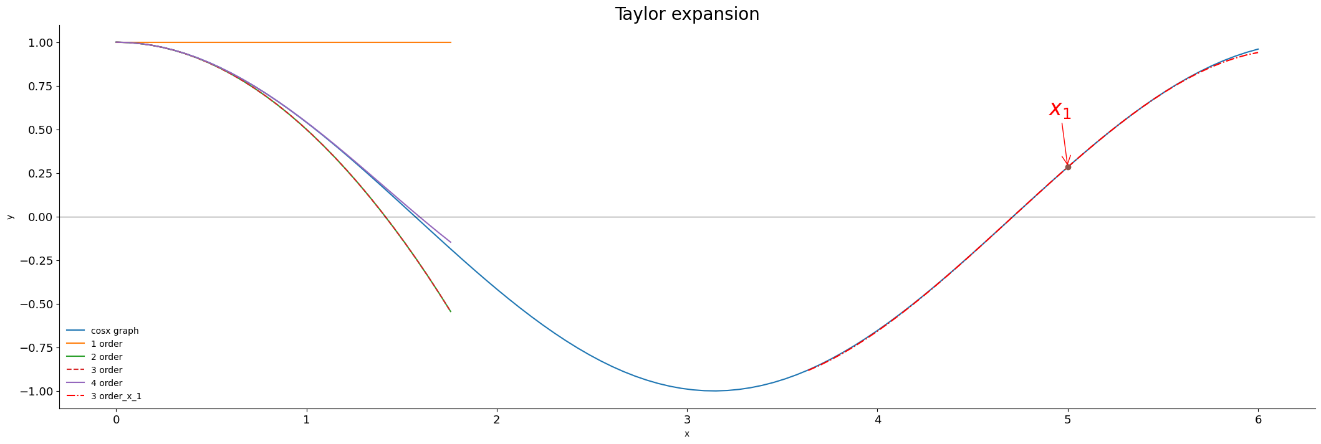


* 误差项

 （推导过程略）

import numpy as np  
import matplotlib.pyplot as plt  
import sympy  
from sympy import pprint,solve,diff,factorial  
  
x,a\_1,b\_1,x\_i=sympy.symbols('x a\_1 b\_1 x\_i')  
  
# 定义原函数  
cos\_curve=sympy.cos(x)  
cos\_curve\_=sympy.lambdify(x,cos\_curve,"numpy")  
# 定义区间  
a,b=0,6   
ix=np.linspace(a,b,100)  
  
fig=plt.figure(figsize=(26,8))  
ax=fig.add\_subplot(111)  
ax.plot(ix,cos\_curve\_(ix) ,label="cosx graph")  
  
# A-x=0位置点近似多项式  
x\_0=0  
# 近似曲线系数计算  
a\_0=cos\_curve.subs(x,x\_0)  
a\_1=diff(cos\_curve,x).subs(x,x\_0)/factorial(1)  
a\_2=diff(cos\_curve,x,x).subs(x,x\_0)/factorial(2)  
a\_3=diff(cos\_curve,x,x,x).subs(x,x\_0)/factorial(3)  
a\_4=diff(cos\_curve,x,x,x,x).subs(x,x\_0)/factorial(4)  
  
ix\_=ix[:30]  
# 1阶近似  
f\_1=a\_1\*x+a\_0  
print("1阶函数：",f\_1)  
ax.plot(ix\_,[1]\*len(ix\_),label="1 order")  
  
# 2阶近似  
f\_2=a\_2\*x\*\*2+a\_1\*x+a\_0  
f\_2\_=sympy.lambdify(x,f\_2,"numpy")  
ax.plot(ix\_,f\_2\_(ix\_),label="2 order")  
print("2阶函数：")  
pprint(f\_2)  
  
# 3阶近似  
f\_3=a\_3\*x\*\*3+a\_2\*x\*\*2+a\_1\*x+a\_0  
f\_3\_=sympy.lambdify(x,f\_3,"numpy")  
ax.plot(ix\_,f\_3\_(ix\_),'--',label="3 order")  
print("3阶函数：")  
pprint(f\_3)  
  
# 4阶近似  
f\_4=a\_4\*x\*\*4+a\_3\*x\*\*3+a\_2\*x\*\*2+a\_1\*x+a\_0  
f\_4\_=sympy.lambdify(x,f\_4,"numpy")  
ax.plot(ix\_,f\_4\_(ix\_),label="4 order")  
print("4阶函数：")  
pprint(f\_4)  
  
# B-x任一点近似多项式(3阶为例)  
x\_1=5  
f\_x=cos\_curve.subs(x,x\_i)+diff(cos\_curve,x).subs(x,x\_i)\*(x-x\_i)/factorial(1)+diff(cos\_curve,x,x).subs(x,x\_i)\*(x-x\_i)\*\*2/factorial(2)+diff(cos\_curve,x,x,x).subs(x,x\_i)\*(x-x\_i)\*\*3/factorial(3) #近似多项式  
f\_x\_1=f\_x.subs(x\_i,x\_1)  
f\_x\_1\_=sympy.lambdify(x,f\_x\_1,"numpy")  
ax.plot(ix[60:],f\_x\_1\_(ix[60:]),label="3 order\_x\_1",c='red',ls='-.')  
ax.plot(x\_1,cos\_curve.subs(x,x\_1),'o')  
ax.annotate(r'$x\_1$', xy=(x\_1, cos\_curve.subs(x,x\_1)), xycoords='data',xytext=(x\_1-0.1,cos\_curve.subs(x,x\_1)+0.3), textcoords='data',weight='bold', color='red',arrowprops=dict(arrowstyle='->',connectionstyle="arc3",color='red'),fontsize=25)  
  
# 误差项(3阶)  
xi=x\_1+0.25  
c,d=x\_1-0.5,x\_1+0.5  
error=diff(cos\_curve,x,x,x,x).subs(x,xi)\*(d-c)\*\*4/factorial(4)  
print("3阶多项式区间[%.2f,%.2f]内位置点%.2f的误差为：%.2f"%(c,d,xi,error))  
  
ax.set\_title(r'Taylor expansion', fontsize=20)  
ax.set\_xlabel('x')  
ax.set\_ylabel('y')  
ax.spines['right'].set\_visible(False)  
ax.spines['top'].set\_visible(False)  
ax.axhline(0,color='black',linewidth=0.5)  
  
ax.legend(loc='lower left', frameon=False)  
plt.xticks(fontsize=13)  
plt.yticks(fontsize=13)  
plt.show()

1阶函数： 1  
2阶函数：  
 2  
 x   
1 - ──  
 2   
3阶函数：  
 2  
 x   
1 - ──  
 2   
4阶函数：  
 4 2   
x x   
── - ── + 1  
24 2   
3阶多项式区间[4.50,5.50]内位置点5.25的误差为：0.02



### 2.3.1.4 偏微分（偏导数 Partial derivative）

* 偏微分

函数，在某个邻域内的所有点都可以关于进行偏微分时，在点处，关于的偏微分系数所对应的函数被称为关于的偏导数。可表示为：。

同样，在这个邻域内的所有点都可以关于进行偏微分时，所对应的被称为关于的偏导数。可表示为：。求偏导数的过程叫做偏微分。

偏微分计算直接使用SymPy库的diff方法。

* 全微分

由在处的近似一次函数可知

，可以将其改写为：

      式(1)

意味着，当点向变化时，高度的增量，效仿一元函数的情况写作。另外，为，为。

此时，式(1)可以写作

      式(2)，时，这个式子意味着：对于函数，当由增加了，由增加了后，就相应增加了。

ybx方向上的增量”，

xay方向上的增量”。

说明“”的增量可以分解为方向上的增量与方向上的增量之和。

将式(2)作理想化（瞬时化）处理了，得，

      式(3)

或者，      式(4)

式(3)(4)被称为全微分公式。即，

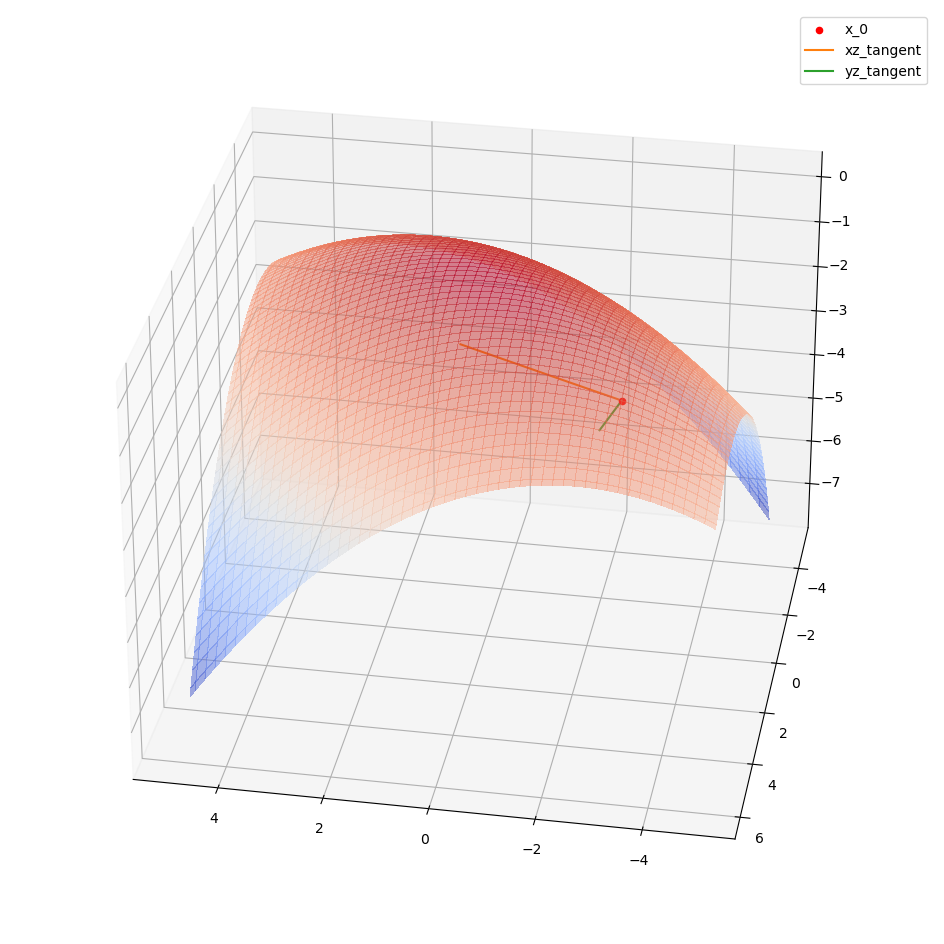
* 链式法则公式(Chain rule)

当时，。（推导过程略）

（偏）微分在机器学习领域中广泛应用，具体可以参看“梯度下降法”部分，对寻找极值的解释。

import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import sympy  
from sympy import pprint,diff  
  
x,y=sympy.symbols('x y')  
f=-(x\*\*2+x\*y+y\*\*2)/10  
f\_=sympy.lambdify([x,y],f,"numpy")  
  
fig=plt.figure(figsize=(12,12))  
ax=fig.add\_subplot(111,projection='3d')  
  
x\_i=np.arange(-5,5,0.1)  
y\_i=np.arange(-5,5,0.1)  
x\_mesh,y\_mesh=np.meshgrid(x\_i,y\_i)  
mesh=f\_(x\_mesh,y\_mesh)  
surf=ax.plot\_surface(x\_mesh,y\_mesh ,mesh, cmap=cm.coolwarm,linewidth=0, antialiased=False,alpha=0.5,)  
  
# 偏微分  
px=diff(f,x)  
py=diff(f,y)  
print("偏微分x：∂𝑓/∂𝑥=")  
pprint(px)  
print("偏微分y：∂𝑓/∂y=")  
pprint(py)  
  
x\_0,y\_0=-3,3  
z\_0=f.subs([(x,x\_0),(y,y\_0)])  
print(x\_0, y\_0,z\_0)  
ax.scatter(x\_0, y\_0,float(z\_0), marker="o",color="red",label="x\_0")  
  
# 平行于xz面，绘制点（x\_0,y\_0）的切线。关于x的偏导数  
xz\_dx=3   
xz\_dz=px.subs([(x,x\_0),(y,y\_0)])\*xz\_dx  
ax.plot((x\_0,x\_0+xz\_dx),(y\_0,y\_0),(z\_0,z\_0+xz\_dz),label="xz\_tangent")  
  
# 平行于yz面，绘制点（x\_0,y\_0）的切线。关于y的偏导数  
yz\_dy=3  
yz\_dz=py.subs([(x,x\_0),(y,y\_0)])\*yz\_dy  
ax.plot((x\_0,x\_0),(y\_0,y\_0+yz\_dy),(z\_0,z\_0+xz\_dz),label="yz\_tangent")  
  
ax.view\_init(30,100) # 可以旋转图形的角度，方便观察  
ax.legend()  
plt.show()

偏微分x：∂𝑓/∂𝑥=  
 x y   
- ─ - ──  
 5 10  
偏微分y：∂𝑓/∂y=  
 x y  
- ── - ─  
 10 5  
-3 3 -9/10



注释（Notes）：

① SymPy（Calculus），在SymPy中计算基本的微积分，例如导数、积分、极限和数列展开（series expansions）（<https://docs.sympy.org/latest/tutorials/intro-tutorial/calculus.html>）。

参考文献（References）:

[1] [日]石山平,大上丈彦著.李巧丽译.7天搞定微积分[M].南海出版公司.海口.2010.8.

[2] [日]小岛宽之著,十神 真漫画绘制,株式会社BECOM漫画制作,张仲恒译.漫画微积分[M].科学出版社.北京.2009.8.