数学实验 exp1 实验报告

计65 赖金霖 2016011377

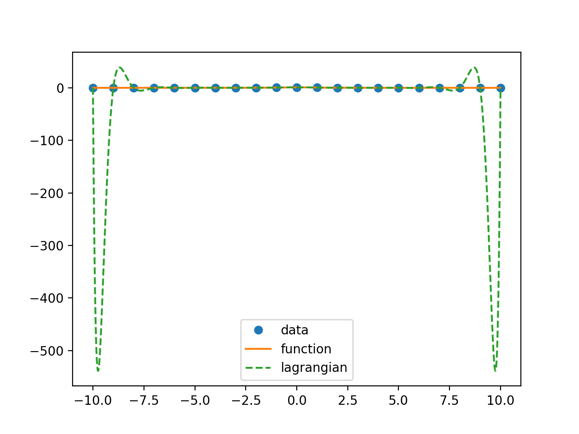
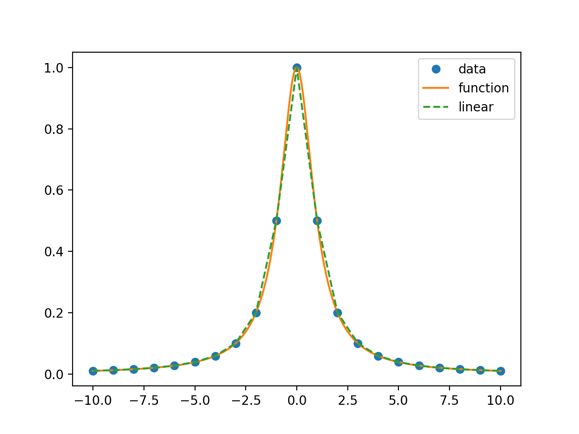
**〇、实现方法**

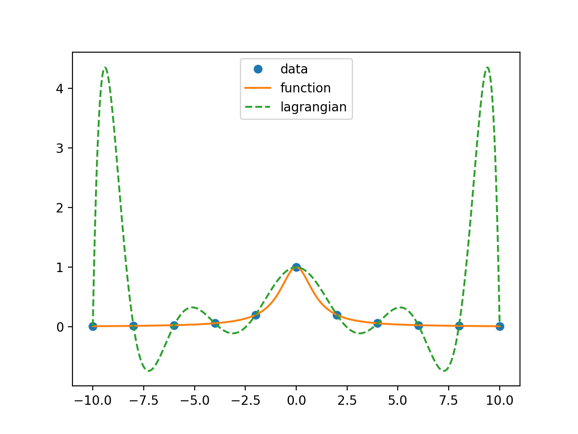
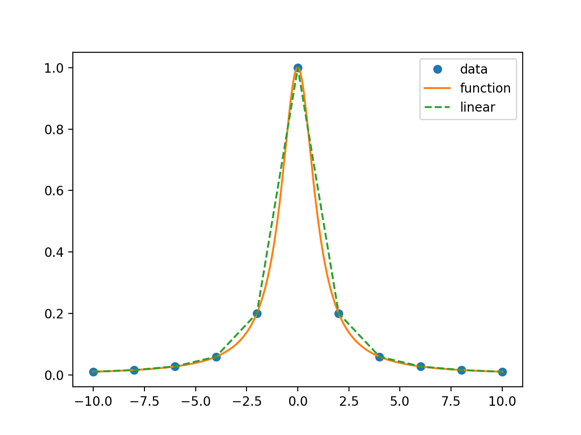
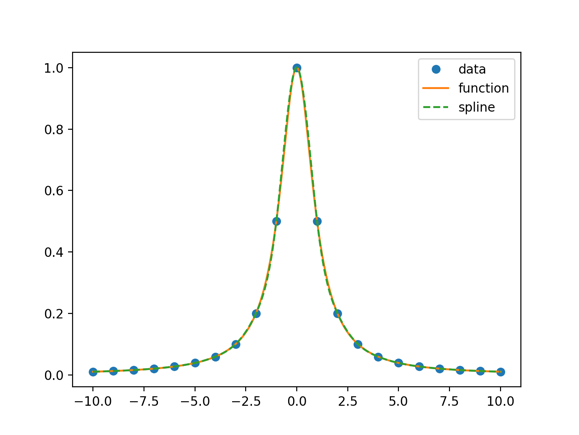
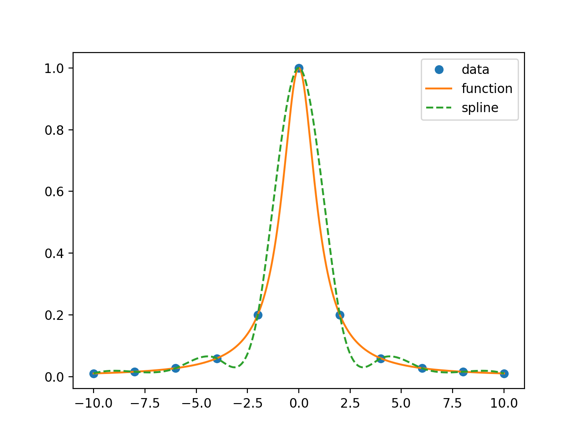
我使用python3完成了本次实验，实验代码可在https://github.com/lll6924/math\_exp的exp1文件夹下找到，各函数实现细节在utils文件夹下。除手动实现部分公式外，用于计算的库为numpy和scipy，用于画图的库为matplotlib。

**一、函数插值练习**

**1.** 等间距插值

可以预见，随着节点数目增加，Lagrange插值的高次项将影响到函数形状，而分段线性和三次插值会有较好的形状。

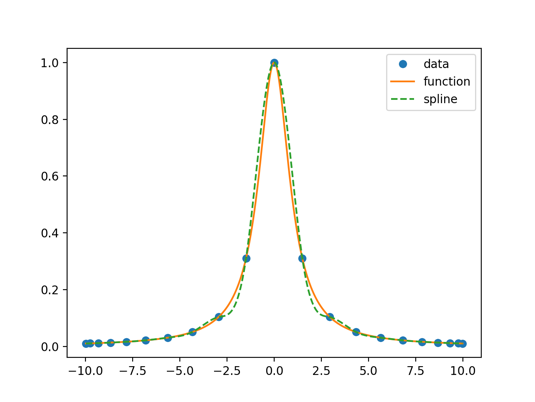
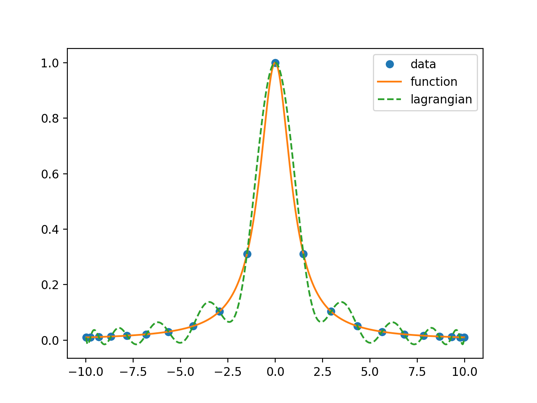
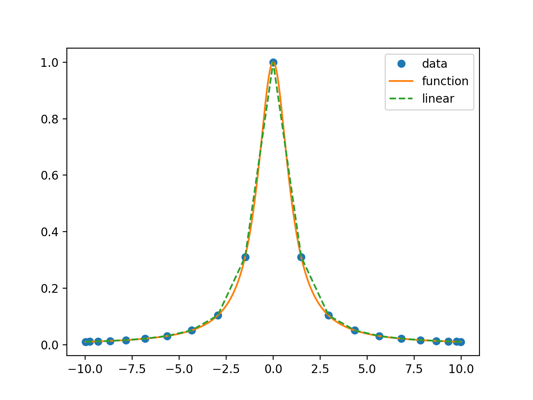
 在节点数目为11时，三种插值结果如左下，节点数目为21时，插值结果如右下：



从上至下依次为Lagrange插值、分段线性插值、三次样条插值。

容易看出，次数较高时Lagrange插值已经崩溃，而分段线性插值和三次样条插值变得接近原始函数。而用肉眼观察，三次样条插值在采样点数为21时已经和原始函数没有什么区别了。

**2.** 非均匀插值

 插值节点****的性质是两侧稠密，中间稀疏，可以猜想这种插值在横坐标绝对值较大的点能保持较好的函数形状，以n=21为例，三种插值方法的图像如下（从左至右分别为Lagrange插值，分段线性插值，三次样条插值）。

容易看出，使采样密度随坐标绝对值增大而增加拯救了高次的Lagrange插值，从插值函数中可以看出原始函数的大致形状。而与此同时，分段线性插值和三次样条插值失去了较多中间部分的信息，而不如均匀采样拟合得好。

这次插值实验说明了不同的插值方法适用不同的采样方法，同时，插值方法也要视具体情况而决定。

本部分绘图代码结构如下：

samp=isometry\_sampler(-10.,10.,8)  
x=samp.getAll()  
func=function1()  
int=lagrangian\_interpolator(x=x,y=func.get(x),fun=func)  
int.plot()

其中isometry\_sampler(等距采样)可替换成strange\_sampler(非均匀采样)，lagrangian\_interpolator可替换成linear\_interpolator或spline\_interpolator。Sampler的第三个参数是采样数。function1是描述被拟合函数的函数对象。各函数的实现细节详见exp1/interpolation.py和utils/function.py。

**二、数值积分练习**

假设复合辛普森公式的结果为S(x)，quad命令的结果为Q(x)，误差为Q\_error(x)，真实值为Φ(x)，通过程序计算x∈Z∩[0,5]时的答案如下表。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | S(x) | Q(x) | Q\_error(x) | Φ(x) |
| 0 | 0.5000000011 | 0.4999999988 | 1e-10 | 0.5 |
| 1 | 0.8413447649 | 0.8413447458 | 2e-10 | 0.8413447461 |
| 2 | 0.9772498538 | 0.9772498704 | 3.3e-09 | 0.9772498681 |
| 3 | 0.998650099 | 0.9986501047 | 2.6e-09 | 0.998650102 |
| 4 | 0.9999683269 | 0.9999683256 | 4.4e-09 | 0.9999683288 |
| 5 | 0.9999997133 | 0.9999997008 | 3.5e-09 | 0.9999997133 |

在具体实现上，由于无法做到从负无穷开始积分，程序采取了在[-100,x]上积分。这里的依据是Φ(-100)已经小到不能被计算机表示。对复合辛普森公式,，程序采取了m=1000的采样，自适应积分设置的限制是最多取2000个区间，但实际没有采到这么多区间。

容易看出，在x=1,2时自适应辛普森要更接近真实值，在x=0,3,4,5时复合辛普森更接近真实值，这是因为调用库限制，无法控制采样点这一变量的结果。在采样点相同的情况下，复合辛普森公式的结果应该更准确。

代码结构如下：

**def** get(x):  
 res1=simpson\_integration(x)  
 print(round(res1,15))  
 (res2,err)=quad\_integration(x)  
 print(round(res2,15))  
 print(round(err,15))  
 print(round(normal\_cdf(x),15))  
 **return** (res2,err)

此函数输入x，分别输出S(x),Q(x),Q\_error(x)和Φ(x)，并返回Φ(x)的估计值和误差，其中simpson\_integration, quad\_integration和normal\_cdf函数实现如下：

**def** simpson\_integration(x):  
 int=simpson\_integrator(SIMPSON\_LEFT,x,function2(),SIMPSON\_M)  
 (res,err)=int.calc()  
 **return** res  
  
**def** quad\_integration(x):  
 int=quad\_integrator(SIMPSON\_LEFT,x,function2(),2\*SIMPSON\_M)  
 (res,err)=int.calc()  
 **return** (res,err)  
  
**def** normal\_cdf(x):  
 res=norm.cdf(x)  
 **return** res

所用到的simpson\_integrator和quad\_integrator见utils/function.py中的实现。norm.cdf是scipy中的函数，function2()是表示正太概率分布的函数对象。