第十三周问答作业

朱士杭 231300027

2024年05月23日

（一）Sympy

1.使用Sympy解以下方程：x^2+3x-4=0

```
import sympy as sy
x=sy.Symbol("x")
exp=x**2+3*x-4
sy.solve(exp)
# e=sy.Eq(x**2+3*x-4,0)
# sy.solve(e,x)
```

2.使用SymPy库，定义一个符号化的表达式来表示二次方程ax^2+bx+c=0。然后使用SymPy的求解功能找到方程的解，并对解进行简化。接着，计算并显示该方程在x=-10到x=10范围内的导数和积分的图形表示

```
#使用sympy求解一般形式的一元二次方程组并求导数与积分
import sympy as sy
x,a,b,c=sy.symbols("x,a,b,c")
equation=sy.Eq(a*x**2+b*x+c,0)
print("一元二次方程组的通解为：",sy.solve(equation,x))
print("一元二次方程组解简化为：",[i.simplify() for i in sy.solve(equation,x)])
exp=a*x**2+b*x+c
diff=sy.diff(exp,x)
inte=sy.integrate(exp,(x,-10,10))
print("exp对x求导结果为：",diff)
print("exp在-10到10范围内积分结果为：",inte)
inte=sy.integrate(exp,x)
print("exp的不定积分为：",inte)
# 定义具体的参数值并进行替换
equation1 = exp.subs({a:1, b: -1, c: 2})
# 计算导数和积分
diff1 = sy.diff(equation1, x)
inte1 = sy.integrate(equation1, x)
# 绘制图形
sy.plotting.plot(equation1,diff1,inte1,(x,-10,10),show=True,legend=True)
sy.plot(diff1,(x,-10,10),legend=True)
sy.plot(inte1,(x,-10,10),legend=True)
```

（二）Scipy

使用SciPy的优化模块（scipy.optimize），找到函数f(x)=x^4-4x^3+x^2+6x+2)在区间[−5,5]上的全局最小值。验证你的结果是否正确，并讨论选择的优化算法及其可能的局限性。

```
#使用scipy的优化模块寻找全局最小值
from scipy.optimize import minimize_scalar
from scipy.optimize import minimize
def f(x):
    return x**4-4*x**3+x**2+6*x+2
#方法一：使用minimize_scalar优化单变量函数寻找全局最小值
res=minimize_scalar(f,bounds=(-5,5))
min_x=res.x
min_y=res.fun
print(min_x,min_y)
#方法二：使用minimize优化多变量函数优化单变量寻找全局最小值
res=minimize(f,x0=-5,bounds=((-5,5),))#x0参数是初始猜测值，bounds是一个元组或者列表表示多个变量的取值范围，注意不要漏
min_x=res.x
min_y=res.fun
print(min_x,min_y)
```

选择的优化算法好像默认为使用“梯度下降法”，如果是单变量的话说白了就直接求导即可，求出驻点然后算出极大值点与极小值点， 然后跟取值范围的边界点进行比较，找出最小值；然而梯度下降法的致命缺陷就是“梯度消失”与“维数灾难”。

1. 梯度下降法可能陷入“局部最小值”，特别是对于非凸函数。它依赖于初始参数值和学习率，可能会收敛到不同的局部最小值。
2. 梯度下降法对“学习率”的选择非常敏感。如果学习率过大，可能会导致更新步骤过大，甚至越过最小值；如果学习率过小，则收敛速度会变慢，需要更多迭代才能收敛。
3. 在高度维度的空间中，梯度为零的点不一定是局部最小值点，也可能是“鞍点”。梯度下降法在这些点的附近可能会停滞不前。
4. 梯度下降法需要计算每个参数的梯度，这在参数量很大时可能会非常耗时。

对于梯度下降法的局限性可以进行一些优化：

1. 可以考虑使用自适应学习率算法，如AdaGrad、RMSProp、Adam等，这些算法根据参数的历史梯度自动调整学习率，有助于加速收敛并减少对学习率的选择依赖。 随机梯度下降（SGD）和小批量梯度下降：这些方法通过使用训练数据的一个小子集来估计梯度，从而减少计算成本，并引入随机性，有助于跳出局部最小值。
2. 牛顿法和拟牛顿法也是不错的选择，使用二阶导数（Hessian矩阵）来加速收敛，但计算成本更高。

（三）Pandas

1、创建一个包含姓名和年龄的DataFrame，并展示如何添加一个新列“年龄分组”，根据年龄将数据分为“儿童”、“青少年”和“成人”。

```
#Pandas创建一个包含姓名和年龄的DataFrame，并添加一个新列“年龄分组”
import numpy as np
import pandas as pd
#创建包含姓名和年龄的DataFrame
```

```
df=pd.DataFrame([["朱士杭",19],["李志",46],["小朋友",4],["古稀老人",70],["Edison",13]],
                index=range(5),columns=["姓名","年龄"])
print("创建的DataFrame为: ",df)
#添加一个新列“年龄分组”
df["年龄分组"]="未知"#这里不用insert方法，直接使用[]创建新的列名
df.loc[df["年龄"]<12,"年龄分组"]="儿童"
df.loc[(df["年龄"]<18) & (df["年龄"]>=12),"年龄分组"]="青少年"
df.loc[df["年龄"]>=18,"年龄分组"]="成人"
print(df)
```

2、 有两个DataFrame，一个包含客户信息，另一个包含订单信息。编写代码将这两个DataFrame按照客户ID合并。

```
#有两个DataFrame，一个包含客户信息，另一个包含订单信息。编写代码将这两个DataFrame按照客户ID合并。
import numpy as np
import pandas as pd
# cli_info=pd.DataFrame([],columns=["客户ID","客户姓名"])
data={"客户姓名":["朱士杭","李志"],"客户ID":["231300027","1701"]}
# cli_info.loc[0]=[231300027,"朱士杭"]
# cli_info.loc[1]=[1701,"李志"]
cli_info=pd.DataFrame(data)
print("客户信息为: \n",cli_info)
data={"客户ID":["231300027","1701"],"订单编码":[340223,230243],"订单商品":["我爱南京CD","HUAWEI Mate60"]}
bill_info=pd.DataFrame(data)
print("订单信息为: \n",bill_info)
merge_info=pd.merge(cli_info,bill_info,on="客户ID")
print("合并后DataFrame信息为: \n",merge_info)
```

3、 银行客户流失数据集用于预测银行业的客户流失，其中包含离开银行或继续作为客户的银行客户的信息。给定bank.csv，请你使用pandas读取该文件，取出creditscore和Balance两列，并分别计算他们的均值和方差；

```
#银行客户流失数据集用于预测银行业的客户流失，其中包含离开银行或继续作为客户的银行客户的信息
import numpy as np
import pandas as pd
bank=pd.read_csv("bank.csv")
data=bank[["CreditScore","Balance"]]
# data=bank.loc[:,["CreditScore","Balance"]]
print(data)
#算出来的均值与方差仍然为一个DataFrame数据
m=data.mean()
v=data.var()
print("数据的均值为: \n",m)
print("数据的方差为: \n",v)
```

(四) Matplotlib

1、给定一组数据创建一个简单的折线图，显示该股票价格随时间的变化。

```
# 创建一个简单的折线图，显示该股票价格随时间的变化
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=[1.1, 2.4, 3.2, 5.9, 6.6, 10.1, 4.3, 0.4, 1.1, 4.3]
data=np.array(data,dtype=np.float32)
plt.plot(data)
plt.show()
```

2、使用subplot功能，在一个画布上绘制上一题中股票价格走向的两个不同的图表，例如一个折线图和一个柱状图。

```
#使用subplot功能，在一个画布上绘制上一题中股票价格走向的两个不同的图表，例如一个折线图和一个柱状图
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=[1.1, 2.4, 3.2, 5.9, 6.6, 10.1, 4.3, 0.4, 1.1, 4.3]
data=np.array(data,dtype=np.float32)
#先画第一个图表为折线图
plt.subplot(1,2,1)
plt.plot(data)
plt.title("The First Image of Line Chart")
plt.xlabel('x Axis')
plt.ylabel('y Axis')
#再画第二个图表为散点图
plt.subplot(1,2,2)
plt.bar(list(range(len(data))),data,color="green")
plt.title("The Second Image of Bar Chart")
plt.xlabel('a Axis')
plt.ylabel('b Axis')
#设置整体的title
plt.suptitle("Two Types of Images Here")
#最后使用show()函数将其显示出来
plt.show()
```

有第二种方法，根据PPT上有讲过add_subplot功能使用画布figure去生成多种图表

```
#使用add_subplot功能，在一个画布上绘制上一题中股票价格走向的四个不同的图表（分别为折线图、散点图、竖向柱状图、横向柱状图）
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=[1.1, 2.4, 3.2, 5.9, 6.6, 10.1, 4.3, 0.4, 1.1, 4.3]
data=np.array(data,dtype=np.float32)
figure=plt.figure()
#先画第一个图表为折线图
figure.add_subplot(2,2,1)
plt.plot(list(range(len(data))),data)
#再画第二个图表为散点图
figure.add_subplot(2,2,2)
# plt.plot(list(range(len(data))),data,"ro")#第一种方法是使用plt的参数"ro"表示red以及散点
plt.scatter(list(range(len(data))),data,color="red")#第二种方法是使用scatter函数，第一个传x，第二个传y
#再画第三个图表为柱状图
figure.add_subplot(2,2,3)
plt.bar(list(range(len(data))),data,color="brown")
#最画第四个图表为横向柱状图
```

```
figure.add_subplot(2,2,4)
plt.barh(list(range(len(data))),data,color="purple")
#最后使用show()函数将其显示出来
plt.show()
```