高数作业答案

**第一题**

问题：给出函数,确定其单调性区间、极值点以及凹凸性。

答案：

1.求导：

1. 确定单调性区间：

当0,函数增加；当, 函数减少。

因此，在区间上递减，在区间上递增。

1. 确定极值点：

是一个极小值点，对应的值为-4。

1. 确定凹凸性：

"(x)=6x-6。当"(x)>0,函数是凸的；当"(x)<0, 函数是凹的。

"(x)=0 时x=1 。 因此，(x)在区间上是凹的，在区间上是凸的。

**第二题**

问题：如果且,求

答案：

使用链式法则，我们有：

*=*

*= 2u 和 = 3*

将 u 的表达式代入，我们得到：

**第三题**

问题：对于函数*, 求 和* 。

答案：

1. 对 x 求偏导数：

*=*

1. 对y 求偏导数：

**第四题**

问题：对于函数, 求点(1,1)处沿向量 的方向导数。

解：计算梯度：

先求f 关于x 和y的偏导数:

在点(1,1)处，偏导数的值分别为2和2。因此，梯度 (1,1)=

标准化方向向量：

我们需要将给定的方向向量转化为单位向量。给定向量 ,其长度为 。单位向量为：

计算方向导数：

方向导数定义为梯度与单位方向向量的点积：

所以，点(1,1)处沿着方向的方向导数是2。

**第五题**

问题：

对于函数,写出其在处的麦克劳林级数的前三项。

答案：

使用泰勒公式，我们有：

对于 f(x)=e², 所有导数在x=0处的值都是1。因此，麦克劳林级数的前三项为：

以上是每个问题的详细解答步骤。如果您还需要其他问题的详细解答或有其他问题，请告诉我。

**第六题**

问题：

最大化函数, 受到约束。 使用拉格朗日乘子法求解。

解：

1. 设置拉格朗日函数：

引入拉格朗日乘子入，我们有：

1. 求偏导并设为0:

为了找到极值，我们对 和入求偏导，并令它们为 0。

*=* 1- 2*x =* 0

*=* 1- 2*y =* 0

*=* 1- *- =* 0

1. 解偏导方程：

从(i)和(ii), 我们有：

*,和 y =*

将这两者代入（iii）得到：

*1 - -*  = 0

*⇒ =*

*⇒ =±*

取= ，我们得*x=y=*，这唯一满足约束*x²+y²=1*的解，因此在此处达到最大值2。

第七题

问题：

计算不定积分

答案：

不定积分的计算基于反导数的概念。对于每一个单独的项，我们增加指数并除以新的指数：

1.对于 3

2.对 于:

3.对于常数2:

结合所有的项，我们得到：

其中C是积分常数。

第八题

问题：

使用牛顿-莱布尼兹公式计算定积分

答案：

牛顿-莱布尼兹公式给出了计算定积分的方法：找到函数的原函数，然后在积分的上下限处求差值

1.对的不定积分为：

2.对常数1的不定积分为：

因此，原函数为：

使用牛顿-莱布尼兹公式：

所以，定积分的结果为：

**第九题**

此函数首先定义了一个内部函数 function\_at\_point，它用于计算函数在给定点的值。然后，它定义了另一个内部函数partial\_derivative，用于计算偏导数。最后，这个偏导数函数被添加到一个列表中，该列表最终被返回。这样，我们就得到了一个每个变量的偏导数函数的列表。

1. def manual\_gradient(f, variables, h=1e-5):
2. """Compute the gradient of a function with respect to the given variables without using external libraries."""
3. gradient\_values = []  # 初始化一个空列表来存储每个变量的偏导数函数
5. **for** var in variables:  # 对于输入的每个变量
7. def function\_at\_point(\*point):
8. """Substitute the variables in the function with the given point."""
9. substitutions = {var: value **for** var, value in zip(variables, point)}  # 创建一个字典，将变量映射到点的坐标
10. **return** eval(f, {}, substitutions)  # 使用eval来计算函数在给定点的值
12. def partial\_derivative(point, var\_index):
13. """Compute the partial derivative using the central difference method."""
14. point\_before = list(point)  # 创建点的一个拷贝
15. point\_before[var\_index] -= h  # 将对应变量的值减小一个小量 h
16. point\_after = list(point)   # 再次创建点的一个拷贝
17. point\_after[var\_index] += h  # 将对应变量的值增加一个小量 h
19. # 使用中心差分方法计算偏导数
20. **return** (function\_at\_point(\*point\_after) - function\_at\_point(\*point\_before)) / (2 \* h)
22. gradient\_values.append(partial\_derivative)  # 将偏导数函数添加到列表中
24. **return** gradient\_values  # 返回偏导数函数的列表

## **第十题**

链式法则涉及到两个函数，内部函数和外部函数。

1. def chain\_rule\_derivative(outer\_function, inner\_function, variable, h=1e-5):
2. """使用链式法则计算导数，不使用外部库。"""
4. # 使用有限差分法计算外部函数在内部函数处的导数
5. def outer\_derivative\_at\_inner(point):
6. # 在给定的点处评估内部函数的值
7. inner\_value = eval(inner\_function, {variable: point})
8. # 为有限差分法在稍微移动的点上评估内部函数的值
9. inner\_value\_h = eval(inner\_function, {variable: point + h})
10. # 计算外部函数值的差分，并除以 h 来近似导数
11. **return** (eval(outer\_function, {"u": inner\_value\_h}) - eval(outer\_function, {"u": inner\_value})) / h
13. # 使用有限差分法计算内部函数的导数
14. def inner\_derivative(point):
15. # 计算内部函数值的差分，并除以 h 来近似导数
16. **return** (eval(inner\_function, {variable: point + h}) - eval(inner\_function, {variable: point})) / h
18. # 结合两个导数来应用链式法则
19. def combined\_derivative(point):
20. # 链式法则通过相乘两个导数来结合它们
21. **return** outer\_derivative\_at\_inner(point) \* inner\_derivative(point)
23. # 返回表示组合导数的函数
24. **return** combined\_derivative
26. # Example usage:
27. outer\_func = "u\*\*2"
28. inner\_func = "3\*x + 7"
29. variable = "x"
31. # Get the combined derivative function using chain rule
32. chain\_derivative\_func = chain\_rule\_derivative(outer\_func, inner\_func, variable)
34. # Evaluate the derivative at the point x=1
35. derivative\_at\_point = chain\_derivative\_func(1)
36. derivative\_at\_point