**一、first\_question.py — Gauss-Legendre数值积分**

实现方法与思路：

1.定义实现Gauss-Legendre 4点法函数：

利用固定的 4 个节点（x）和对应权重（A）对函数在 [-1,1] 上进行数值积分。

def gauss\_legendre\_4(f):

    x = [-0.8611363116,-0.3399810436,0.3399810436,0.8611363116]

    A = [0.3478548451,0.6521451549,0.6521451549,0.3478548451]

    return sum(A[i] \* f(x[i]) for i in range(4))

2.定义区间变换函数：

由于 Gauss-Legendre 积分只能用于 [-1,1]，我们需要将实际区间 [a,b] 映射到 [-1,1]，使用变换函数 transform 完成这一操作，同时调整函数值乘以 (b-a)/2。

def transform(f, a, b):

    def g(t):

        return f((b + a) / 2 + (b - a) \* t / 2) \* (b - a) / 2

    return g

3.定义实现自适应区间划分函数：

integrate 函数从初始的 1 个区间开始，将 [a,b] 分成 n 个子区间，对每个子区间分别进行积分。如果与上一次积分值相比变化小于指定的误差 epsilon（1e-7），则认为收敛，终止迭代。

def div(f, a, b, n):

    h = (b - a) / n

    x = [a + i \* h for i in range(n + 1)]

    return x, h, f

def integrate(f, a, b, pre\_integral, episilon=1e-7):

    n =1

    while True:

        x, h, f = div(f, a, b, n)

        integral = sum(gauss\_legendre\_4(transform(f, x[i], x[i + 1])) for i in range(n))

        if abs(integral - pre\_integral) < episilon:

            break

        n += 1

    print(f"一共分出的区间个数: {n}")

    print(f"节点:{x}")

    print(f"积分值: {integral}")

    return integral, n

1. 定义被积函数

def f0(x):

    return 1 / (cmath.sin(x) \*\* 2 + 1/4 \* cmath.cos(x) \*\* 2)

1. 实现

a = 0

b = cmath.pi / 2

integral, n = integrate(f0, a, b, cmath.pi)

1. **second\_question.py — 最小二乘**

实现方法与思路：

1.数据导入：

原始的 x 和 ys 是一组观测数据，后者为因变量，前者为自变量。将 ys 取对数以便拟合指数函数 y = ae^(bx)。

x = [4.0, 4.2, 4.5, 4.7, 5.1, 5.5, 5.9, 6.3, 6.8, 7.1]

ys = [102.56, 113.18, 130.11, 142.05, 167.53, 195.14, 224.87, 256.73, 299.50, 326.72]

y = [np.log(i) for i in ys]

n\_1 = 3

n\_2 = 4

2.构造最小二乘法矩阵：

构造 φ 向量 phi\_i(x, i) 表示第 i 次幂的项。函数get\_G(x, n) 返回正规方程组中的系数矩阵 G。函数get\_Y(y, x, n) 返回右侧向量 Y。

def phi\_i(x, i):

    return [x[j] \*\* i for j in range(len(x))]

def get\_Y(y, x, n):

    return [np.dot(y, phi\_i(x, i)) for i in range(n)]

def get\_G(x, n):

    G = np.zeros((n, n))

    for i in range(n):

        for j in range(n):

            G[i][j] = np.dot(phi\_i(x, i), phi\_i(x, j))

    return G

1. 求解：

使用 np.linalg.solve(G, Y) 解线性方程组，获得多项式系数 c。

def get\_p(ys, x, n):

    Y = np.array(get\_Y(ys, x, n))

    G = get\_G(x, n)

    cond\_number = np.linalg.cond(G)  # 条件数计算

    print("条件数为:", cond\_number)

    try:

    # 求解线性方程组 Gc = Y

        c = np.linalg.solve(G, Y)

        p = np.poly1d(c[::-1])

        print("求解得到的向量 c 为:", c)

    except np.linalg.LinAlgError:

        print("系数矩阵 G 是奇异矩阵，无法求解线性方程组。")

        p = np.poly1d([0])

        c = np.zeros(n)

    return p, c

4.拟合并绘图：

通过 np.poly1d(c[::-1]) 构造多项式函数。对比不同阶数（n=3 和 n=4）的拟合效果。将对数后的 y 拟合为一次函数 ln(y) = bx + ln(a)，再通过指数转换回原始函数 ae^(bx)。

p\_1, c\_1 = get\_p(ys, x, n\_1)

p\_2, c\_2 = get\_p(ys, x, n\_2)

x\_vals = np.linspace(min(x), max(x), 200)

y1\_vals = p\_1(x\_vals)

y2\_vals = p\_2(x\_vals)

plt.plot(x\_vals, y1\_vals, label='Fitted Polynomial n=3', color='blue')

plt.plot(x\_vals, y2\_vals, label='Fitted Polynomial n=4', color='green')

# 绘制数据点和多项式函数

plt.scatter(x, ys, label='Data Points', color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Data Points and Polynomial Function')

plt.legend()

plt.grid(True)

plt.show()

p ,c = get\_p(y, x, 2)

b = c[1]

a = np.exp(c[0])

def fx(x):

    return a\*np.exp(b\*x)

y\_valss = fx(x\_vals)

plt.plot(x\_vals, y\_valss, label='ae^b Function', color='orange')

plt.scatter(x, ys, label='Data Points', color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Data Points and Function')

plt.legend()

plt.grid(True)

plt.show()

5.条件数计算：

打印系数矩阵的条件数，判断矩阵求解的稳定性。

cond\_number = np.linalg.cond(G)  # 条件数计算

    print("条件数为:", cond\_number)

1. 实现：

p\_1, c\_1 = get\_p(ys, x, n\_1)

p\_2, c\_2 = get\_p(ys, x, n\_2)

p ,c = get\_p(y, x, 2)

1. **third\_question.py — FFT**

实现方法与思路：

基于蝶形操作的Cooley–Tukey算法。实现逻辑分解：

1.初始化参数：

输入数组 a 长度为 2\*\*m，（m=3）即8个点，本FFT算法要求点数为2的整数次幂。

m = 3

a = [9, 7, 5, 3, 1, 4, 6, 8]

N = 2\*\*m

a\_0 = [a[i] for i in range(0, len(a))]

S = N

2.定义函数tran()实现位反转重排：

FFT 的输入数据需要进行位反转操作（bit reversal），将输入数组按照倒序二进制索引重排，这是后续蝶形操作的基础。

def tran(a):

    n = len(a)

    num\_bits = n.bit\_length() - 1

    result = [0] \* n

    for i in range(n):

        binary\_str = bin(i)[2:].zfill(num\_bits)

        rev\_i = int(binary\_str[::-1], 2)

        result[rev\_i] = a[i]

    return result

3.实现FFT核心算法：

外层循环控制分组大小 S，从全数组（8）逐步减半。每一轮中，每两个小组进行一次 蝶形操作，计算组合项和差，并乘以旋转因子。返回变换后的结果：经过位反转后的数组，即为频域结果。

def FFT(a,m):

    N = 2\*\*m

    a\_0 = a.copy()

    S = N

    while S > 1:

        T = N//S

        for k in range(1, T+1):

            for l in range((k-1)\*S, (k-1)\*S+S//2 ):

                y = a\_0[l]

                a\_0[l] = y + a\_0[l+S//2]

                a\_0[l+S//2] = (y - a\_0[l+S//2])\*cmath.exp(-2j\*cmath.pi\*l/S)

        S = S//2

    return tran(a\_0)

1. 实现：

result = FFT(a, m)

print(result)