基于无线传感器网络定位问题的解决方案

一、问题描述（观测量、未知量、已知量）

1.题目原貌

无线传感器网络的定位问题

物联网是新一代信息技术的重要组成部分，它是通过各种信息传感设备，如传感器、射频识别（RFID）技术、全球定位系统等各种装置与技术，实时采集声、光、热、 电、力学、化学、生物、位置等各种信息，与互联网结合形成的一个巨大网络。作为物联网的重要组成部分，无线传感器网络（WSN, Wireless Sensor networks）就是由部署在监测区域内大量的廉价微型传感器节点组成，通过无线通信方式形成的一个多跳自组织网络。

无线传感器网络的很多应用场合必须知道节点的位置，因此节点定位技术是 WSN 的关键技术和研究热点。然而，在所有节点上都配备 GPS 等定位设施成本很高。因此，一般只在部分节点通过 GPS 定位设备获得自身的精确位置，这些节点称为信标节点（beacon node）；而其它未知节点（unknown node）则通过网络连接信息和节点内部相互测距通过几何计算来估计其位置坐标。假设测距的随机误差服从正态分布，并且由于地形或设备原因，测距可能存在系统偏差。请在此假设条件下解决以下问题：

1. 对仿真算例中的未知节点进行定位（结果存储在附件表 3 中）

2. 对定位结果精度进行评定

3. 对定位结果进行总体模型检验

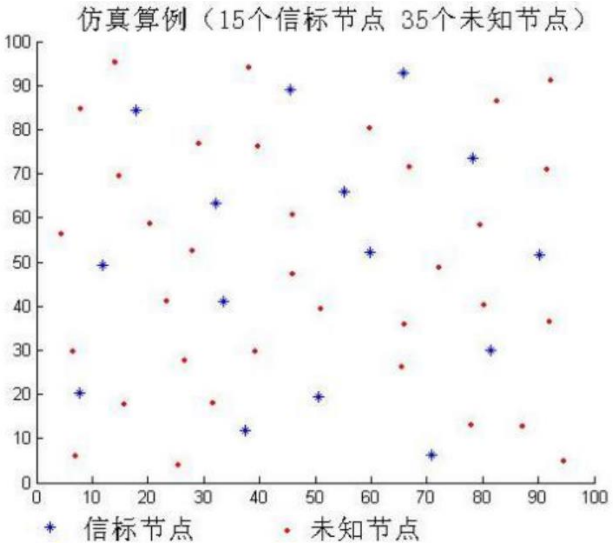

图 1：题目中的仿真算例参考图

注：附件中表 1 为未知节点到距离其最近的 6 个信标节点的测量距离，表 2 为信标节点的坐标，表 3 用来存储计算结果——未知节点的坐标。

2.变量分析

观测量
每一个未知节点与最近六个信标节点的测量距离，使用 j 和 i 来标记未知节点和信标节点的编号，未知节点 j 和已知节点 i 之间的测量距离记为 d_{ij}；
未知量
每一个未知节点的位置坐标，使用 j 来标记未知节点的编号，记为 (x_j, y_j) ；
已知量
1-15 个信标节点的 GPS 坐标，记为 (x_i, y_i)，其中 i 是信标节点的编号；信标节点的位置是通过 GPS 测定的，因此当作误差为 0 的准确值来处理。

二、数学模型（函数模型和随机模型）

1.函数模型

观测量（距离）与未知量（未知节点坐标）之间的关系可以通过欧几里得距离公式表示：

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} + \varepsilon_{ij}$$

其中，d_{ij} 是未知节点 j 与信标节点 i 之间的测量距离；

(x_j, y_j) 是未知节点 j 的坐标；

(x_i, y_i) 是信标节点 i 的坐标；

ε_{ij} 是测量未知节点 j 与信标节点 i 之间距离的误差；

2.随机模型

在测量过程中，测量误差由系统偏差和随机误差组成，表示为：

$$\varepsilon_{ij} = \mu + \delta$$

其中，ε_{ij} 是测量误差，与前面函数模型中的测量误差一致；

μ 是系统误差，是由于地形和设备原因造成的；

δ 是随机误差；

δ 服从正态分布，可以定义其服从均值为 0，方差为 σ^2 的正态分布，即

$$\delta \sim N(0, \sigma^2)$$

因此可推得 ε_{ij} 服从均值为μ，方差为 σ^2 的正态分布，即

$$\varepsilon_{ij} \sim N(\mu, \sigma^2)$$

三、参数估计（非线性问题包括线性化、迭代求解、线性问题直接用 BLUE 估计的公式求解）

在无线传感器网络节点定位中，我们的目标是通过测量未知节点与若干信标节点之间的距离来估计未知节点的位置坐标 (x_j, y_j) 。由于距离公式是非线性的，

因此需要对其进行线性化处理，并通过迭代方法求解。

0.表格预处理与数据读取

要转化为代码可以处理的数据，需要对原数据进行预处理和读取工作。我们首先拆解了原 excel 文件中的不同表，将需要读取的两张表单独提取出来成为两个单独的 excel 文件方便处理，将存储相对距离表格的表头行和纵轴编号列都删掉了，只保留了原始的数据表格，记为 1.xlsx，如图 2；将存储信标节点坐标表格的表头行删掉了，也同样只保留了原始的数据表格，记为 2.xlsx，如图 3。由于这些数据都是标准的数据格式，所以不用进行数据格式转换。

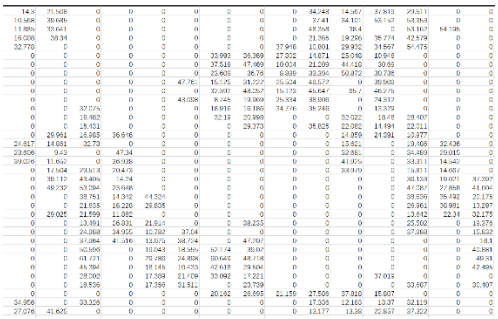

图 2：1.xlsx 表格形式

17.86	84.36
45.51	89.04
59.79	52.19
78.23	73.54
81.45	29.97
70.85	6.29
37.44	11.84
50.58	19.44
7.72	20.32
11.87	49.27
32.14	63.3
33.53	41.08
55.18	65.94
65.78	92.84
90.21	51.61

图 3：2.xlsx 表格形式

接下来就是 python 中的转换代码：

```
beacon_coords_path = "2.xlsx" #这里需要替换为运行电脑上的相对路径
beacon_coords = pd.read_excel(beacon_coords_path, header=None, names=['x', 'y']).to_numpy()
distances_path = " \1.xlsx" #这里需要替换为运行电脑上的相对路径
distances_matrix = pd.read_excel(distances_path, header=None).to_numpy()
```

1.初始估计

为了开始迭代求解，我们首先需要为未知节点的坐标提供一个初始估计值。通常可以使用信标节点坐标的质心作为初始估计。这是因为质心可以作为未知节点可能位置的一个初始估计。

$$\left(x_j^{(0)}, y_j^{(0)}\right) = \left(\frac{1}{6}\sum_{i=1}^{N} x_i, \frac{1}{6}\sum_{i=1}^{N} y_i\right)$$

其中，(x_i, y_i) 是第 i 个信标节点的坐标；信标节点的数量为 6；

代码展示：

```
def initial_estimate(distances_row, beacon_coords):
    indices = np.nonzero(distances_row)[0]
    initial_x = np.mean(beacon_coords[indices, 0])
    initial_y = np.mean(beacon_coords[indices, 1])
    return np.array([initial_x, initial_y])
```

2.雅可比矩阵

引入雅可比矩阵 J，其元素为距离对坐标的偏导数：

$$J_{ij} = \begin{bmatrix} \frac{\partial d_{ij}}{\partial x_j} & \frac{\partial d_{ij}}{\partial y_j} \end{bmatrix}$$

其中：

$$\frac{\partial d_{ij}}{\partial x_j} = \frac{x_j^{(0)} - x_i}{d_{ij}^{(0)}}, \quad \frac{\partial d_{ij}}{\partial y_j} = \frac{y_j^{(0)} - y_i}{d_{ij}^{(0)}}$$

代码展示：

```
def compute_jacobian(estimated_coords, beacon_indices, beacon_coords):
    jacobian = []
    for i in beacon_indices:
        d = np.linalg.norm(estimated_coords - beacon_coords[i])
        jacobian.append([(estimated_coords[0] - beacon_coords[i, 0]) / d,
                         (estimated_coords[1] - beacon_coords[i, 1]) / d])
    return np.array(jacobian)
```

3.线性化处理和线性化模型的矩阵表示

在每次迭代中，我们使用泰勒展开对非线性距离公式进行线性化处理。对于每个距离测量d_{ij}，我们可以展开公式如下：

$$d_{ij} \approx d_{ij}^{(0)} + \left.\frac{\partial \mathrm{d_{ij}}}{\partial \mathrm{x_j}}\right|_{\left(\mathrm{x_j^{(0)}},\mathrm{y_j^{(0)}}\right)} \left(\mathrm{x_j} - \mathrm{x_j^{(0)}}\right) + \left.\frac{\partial \mathrm{d_{ij}}}{\partial \mathrm{y_j}}\right|_{\left(\mathrm{x_j^{(0)}},\mathrm{y_j^{(0)}}\right)} \left(\mathrm{y_j} - \mathrm{y_j^{(0)}}\right)$$

其中，$d_{ij}^{(0)} = \sqrt{\left(x_j^{(0)} - x_i\right)^2 + \left(y_j^{(0)} - y_i\right)^2}$ 是使用初始估计值计算的距离

将所有测量值和偏导数整合，可以得到线性化模型的矩阵形式：

$$d \approx d^{(0)} + J\left(P_j - P_j^{(0)}\right)$$

其中：

$$d = \begin{bmatrix} d_{1j} \\ \vdots \\ d_{Nj} \end{bmatrix},\ d^{(0)} = \begin{bmatrix} d_{1j}^{(0)} \\ \vdots \\ d_{Nj}^{(0)} \end{bmatrix},\ J = \begin{bmatrix} \dfrac{x_j^{(0)} - x_1}{d_{1j}^{(0)}} & \dfrac{y_j^{(0)} - y_1}{d_{1j}^{(0)}} \\ \vdots & \vdots \\ \dfrac{x_j^{(0)} - x_N}{d_{Nj}^{(0)}} & \dfrac{y_j^{(0)} - y_N}{d_{Nj}^{(0)}} \end{bmatrix},\ P_j = \begin{bmatrix} x_j \\ y_j \end{bmatrix}$$

此处，d 是实际测量的距离向量，$d^{(0)}$ 是使用初始估计值计算的距离向量，J 是雅可比矩阵，P_j 是未知节点的实际坐标向量，$\mathrm{P_j^{(0)}}$是初始估计值向量。

这里的代码将与后面的迭代求解一起展示。

4.迭代求解

使用高斯-牛顿法进行迭代求解，每次迭代更新未知节点坐标估计值：

$$P_j^{(k+1)} = P_j^{(k)} + \Delta P_j^{(k)}$$

其中增量 $\Delta P_j^{(k)}$ 通过最小化残差平方和求解：

$$\Delta P_j^{(k)} = (J^T J)^{-1} J^T \left(d - d^{(0)}\right)$$

在此公式中，J^T 是雅可比矩阵的转置，$(J^T J)^{-1}$ 是 $J^T J$ 的逆矩阵，$d - d^{(0)}$ 是测量值和初始估计值的残差向量。

迭代过程持续进行，直到增量 $\Delta P_j^{(k)}$ 足够小未知，即达到预设的收敛标

准。经过若干次迭代，我们可以获得最终的估计值：

$$\hat{P}_j = P_j^{(k+1)}$$

代码展示：

```
def gauss_newton(distances_row, beacon_coords, initial_coords, max_iterations=100,
tolerance=1e-6):
    estimated_coords = initial_coords
    beacon_indices = np.nonzero(distances_row)[0]
    residual_history = []
    for _ in range(max_iterations):
        estimated_distances = np.array([np.linalg.norm(estimated_coords -
beacon_coords[i]) for i in beacon_indices])
        residuals = distances_row[beacon_indices] - estimated_distances
        residual_history.append(np.linalg.norm(residuals))
        jacobian = compute_jacobian(estimated_coords, beacon_indices, beacon_coords)
        delta_coords = np.linalg.inv(jacobian.T @ jacobian) @ jacobian.T @ residuals
        estimated_coords += delta_coords
        if np.linalg.norm(delta_coords) < tolerance:
            break
    sigma_squared = np.sum(residuals**2) / (len(beacon_indices) - 2)
     cov_matrix = sigma_squared * np.linalg.inv(jacobian.T @ jacobian)
    return estimated_coords, residual_history, cov_matrix, residuals

     unknown_node_coords = []
     all_residual_histories = []
     all_cov_matrices = []
     all_residuals = []
     for i, distances_row in enumerate(distances_matrix):
         initial_coords = initial_estimate(distances_row, beacon_coords)
         estimated_coords, residual_history, cov_matrix, residuals =
     gauss_newton(distances_row, beacon_coords, initial_coords)
         unknown_node_coords.append(estimated_coords)
         all_residual_histories.append(residual_history)
         all_cov_matrices.append(cov_matrix)
         all_residuals.append(residuals)

     unknown_node_coords = np.array(unknown_node_coords)
     for i, coords in enumerate(unknown_node_coords):
         print(f"未知节点 {i+1} 的估计坐标: x = {coords[0]:.4f}, y = {coords[1]:.4f}")
```

以下是未知节点的计算结果:

	1	2	3	4	5	6	7	8	9
x	28.8707	6.4854	12.3203	12.9078	2.4275	22.8678	4.3433	14.5798	24.6823
y	77.8333	86.3701	96.4435	69.7984	56.4152	41.6444	29.4808	15.9053	1.8911

	10	11	12	13	14	15	16	17	18
x	5.3289	30.8891	39.8806	52.2025	46.4410	45.6693	39.1348	37.5590	52.8101
y	4.3373	16.5428	29.6278	38.9358	46.9152	60.2077	76.2755	95.7530	99.4267

	19	20	21	22	23	24	25	26	27
x	60.1243	84.1344	93.9356	93.3091	80.4775	67.0886	72.4471	81.9224	94.3267
y	80.9001	87.6996	92.5429	71.8445	58.3157	71.6346	48.6995	39.8500	35.9463

	28	29	30	31	32	33	34	35
x	88.9736	96.4470	79.5907	66.3594	65.7418	26.1980	27.3002	19.1997
y	11.3517	3.1213	11.3681	25.9178	35.6393	27.0012	51.7889	58.2139

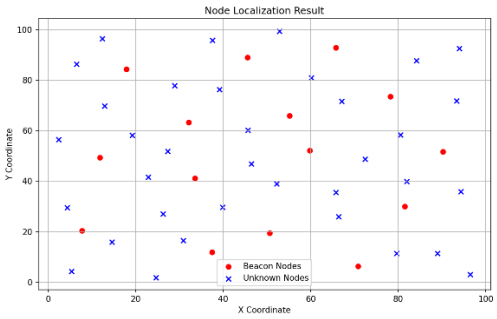

图 4：节点计算结果分布图

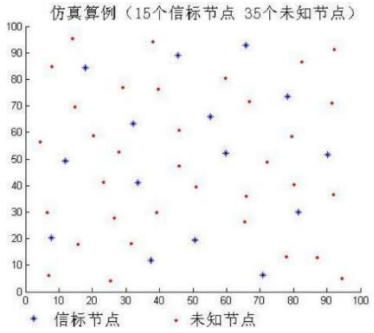

图 5：题目给定参考算例图

经过迭代求解之后，得到对应的未知点的坐标，将信标节点和未知节点求解结果一起表示在图上，如图所示，可以发现和算例的结果几乎差不多，粗略地说明了结果是正确的。下面，会根据求解结果进行更详细的参数精度评定与总体模型检验。

三、参数精度评定与总体模型检验

1.迭代残差变化曲线和残差分布图

在每次迭代中，我们计算了估计距离与实际测量距离之间的差值，称为残差。

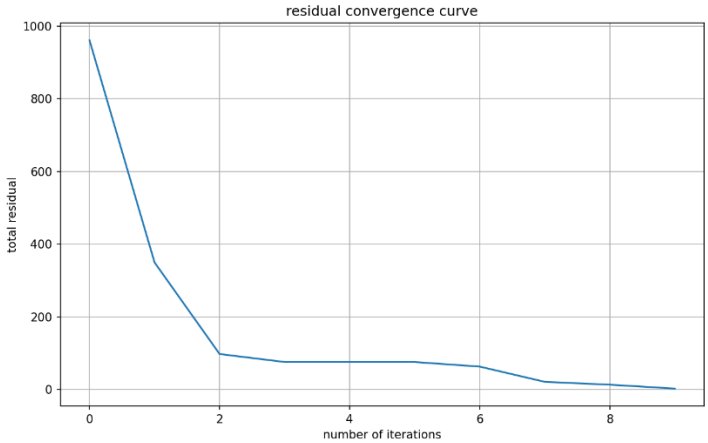

图 6：迭代总残差变化曲线

（不知道为什么 matplotlib 一直不支持中文，所以此图以及后面的图都用英文了）

把迭代求解过程中的总残差变化曲线求解出来，可以发现总残差在开始时呈现快速下降的趋势，随后出现了暂时的放缓，最后继续下降到几乎为 0 的水平，这种迭代残差变化曲线反映了计算结果是正确且有效的。

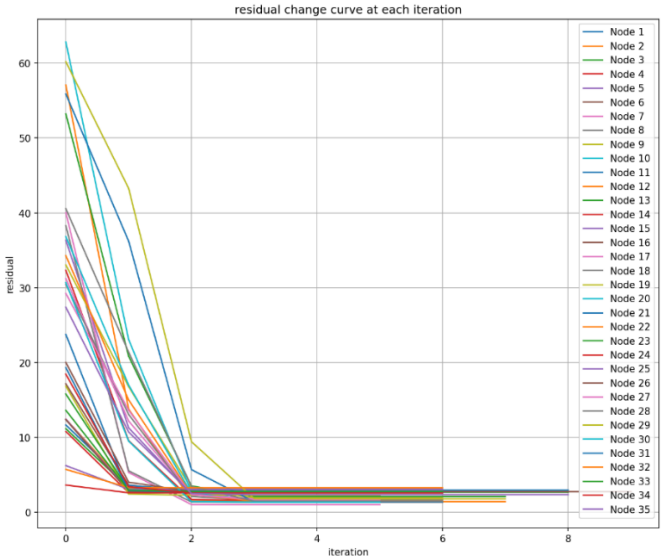

图 7：迭代的每一点残差变化曲线

此外，可以把迭代求解过程中的每一点的残差变化曲线求解出来，可以发现点与点虽然存在着一些变化的差异，但是整体上都是快速变小至接近 0 的，进一步验证了计算结果的正确性。

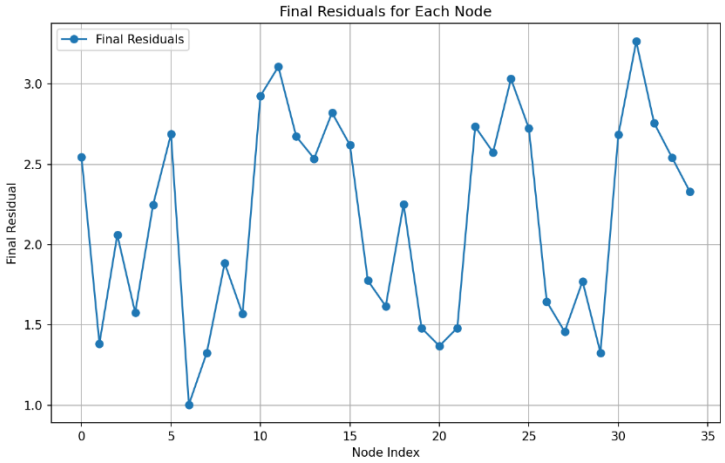

图 8：不同节点的残差分布图

这张图反映了 35 个未知节点坐标结果的残差分布图，可以发现，这 35 个残差的分布非常不均匀，而且看似没有什么规律可循，后面我们会根据这些残差进行检验。

2.协方差矩阵计算

协方差矩阵反映了参数估计的不确定性和参数之间的相关性。通过分析协方差矩阵，可以了解参数估计的精度以及参数之间的线性相关性。

对于两个随机变量 X 和 Y，他们的平均值（期望值）分别记为 $E[X]$ 和 $E[Y]$，这两个随机变量的协方差定义为

$$Cov(X,Y) = E[(X - E[X])(Y - E[Y])]$$

协方差度量了 X 和 Y 相对于它们各自的均值的联合变化趋势。如果两个变量的协方差为正，它们倾向于一起增加或减少；如果为负，一个变量的增加往往伴随着另一个变量的减少。

对于一个由 n 个随机变量 $X_1, X_2, \ldots, X_n$ 组成的向量，协方差矩阵 Σ 是一个 $n \times n$ 的方阵，其元素定义如下：

$$\Sigma = Cov(X_i, X_j)$$

协方差矩阵的对角线元素 Σ_{ii} 是变量 X_i 的方差，非对角线元素 Σ_{ij} 是 X_i 和 X_j 的协方差。

在可视化图中，协方差矩阵的对角线元素表示参数的方差，非对角线元素表示参数之间的协方差。方差越大，说明参数的不确定性越大；协方差越大，说明参数之间的相关性越强。

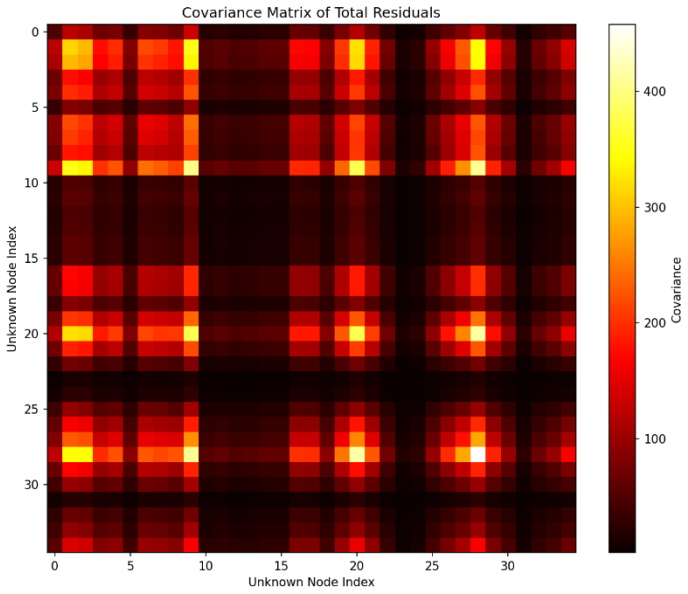

图 9：未知节点残差协方差矩阵图

这张图反映了未知节点的残差在迭代变化过程中的协方差矩阵，可以看出未知节点和未知节点残差之间的相关性并不均衡。首先看对角线上的协方差分布，这里反映的是变量自身的方差，可以看出 1 点、2 点、8-9 点、20 点、28 点的方差都相当地大，这与前面迭代残差变化图相对应，这些点的迭代变化值显然比较大。再看其它区域的协方差分布，可以看出，坐标值的迭代是有明显倾向的，对于原本初始值就接近真值的未知点，往往方差较小，而一开始初始值远离真值的未知点，往往方差较大，因为需要迭代多次才能接近真值，而且结合之间的残差变化图，这些远点往往会在快速接近真值之后在真值附近停滞一两代，然后才继续缓慢逼近真值。

3.残差正态分布的检验和判断

之前在残差分布图中，我们发现残差的分布似乎没有什么规律，为了佐证这一点，我们进行残差正态分布的检验和判断，看其是否为正态分布。这里使用了检验正态分布的两种方法，一种是直观判断的 Q-Q 图表示，一种是属于假设检验的 K-S 检验方法。

Q-Q 图是一种用于比较两组数据分布的图形工具，我们在这里进行原分布与正态分布的比较。横轴是表示理论分布分位数，是通过累计分布函数的逆函数 $\Phi^{-1}(\mathrm{p})$ 来计算分位数，其中 p 是累计概率；纵轴是表示样本数据的有序值，样本中的第 i 个值对应于经验累计概率 $\frac{i}{n+1}$，其中 n 是样本大小。一般来说，如果点显著偏移这条直线，表面样本分布与理论分布不一致。

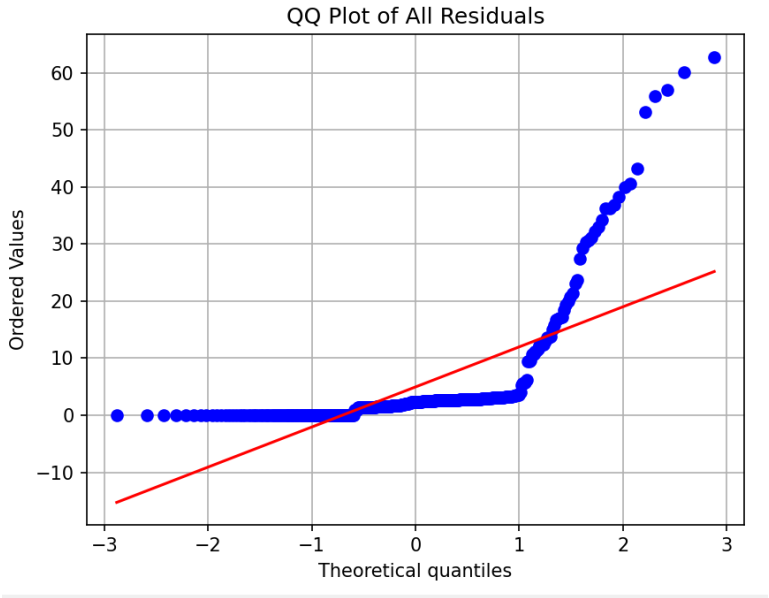

图 9：残差分布 Q-Q 图

根据 Q-Q 图来看，明显残差分布不符合正态分布情形。

K-S 检验是一种基于最大距离的统计检验，用于比较一个样本分布与一个理论分布（或两个样本分布）之间的一致性。我们现在来比较原分布是否符合

正态分布。其中定义了样本数据的理论累积分布函数，表示在该分布夏，随机变量取值小于或等于某个数值的概率，还有经验累计分布函数，是样本数据从小到大排序后，每个数据点所累积的观测值比例。对每一个数据点，计算经验累积分布函数和理论累计分布函数的绝对差值，根据 D 和样本大小确定 p 值，并根据 p 值来决定是否拒绝原假设。

根据结果，K-S 检验值在 0.39 左右，表示样本分布与理论分布之间的最大偏差是 0.39，判断显著性的概率值是一个非常接近 0 的数，再次说明了不是呈正态分布。

总的来看，残差值并不是呈现正态分布，但是这不代表着该算法是错误的。按 Q-Q 图和之前的残差分布图来看，残差分布更倾向于呈现一个双峰的融合正态分布，对于未知点而言，在迭代过程中，残差的大小与也与距离已知点的大小有关，在周边已知点都距离较远的情况下，可能使得残差也相应增加，另外，初始值的设定也有很强的关联，远点和近点的迭代快慢先后有着显著的差别，这使得残差有较大的不同。