置乱的阶的分析

PB21000004　吴越

1　问题一

编写一个程序，对于N个数的置乱：

1. 求最大阶T。它意味着密文连续再加密T-1次就会恢复成明文。
2. 绘制p(K)曲线。p(K)是任意一个置乱，其阶小于K的概率。
3. 你的程序能处理的N越大越好。

1.1　实验原理与算法

任意置乱的阶，就是进行不相交循环分解后各个循环长度的最小公倍数。

这里我们分两种情况讨论

1. 算法1：n值较小的情况

   可以考虑直接遍历求解，即：遍历所有可能的置乱，计算每个置乱的阶，最大值即为最大阶T。

   要绘制p(K)曲线，可以直接遍历所有可能的置乱，对每个置乱的阶做统计，然后分别计算各种阶大小出现的概率，最后累加起来就是概率分布函数，也就是p(K)。

2. 算法2：n值较大的情况

   对于较大的n值，置乱有n!种，不可能再采用遍历的方式，而要采用更好的算法。

   这里利用了置乱阶的特殊性质：将n分解成若干数字的和，这些数字的最小公倍数的最大值，就是最大阶。对于这个问题，查找相关文献 A000793 - OEIS 后，发现有可供参考的高效算法可以利用。

   要绘制p(K)曲线，可以考虑用概率统计的方法计算概率。对给定的K，随机取1000（或更多）个参考点，每个参考点取1000（或更多）个随机置乱，然后统计阶小于K的数量，计算阶小于K的概率，由大数定律即得到p(K)概率的分布函数。

1.2　代码实现

1.2.1　n值较小的情况计算T与p(K)

设定一个阶的最大值，直接遍历所有可能的置乱，对每个置乱的阶做统计，T就是阶的最大值。然后分别计算各种阶大小出现的概率，最后累加起来就是概率分布函数，也就是p(K)。

```
def samll_n_plot_p_k_curve(N,T=10000):
    orders = [0] * (T + 1)
    total_permutations = 0
    # 遍历循环
    for p in itertools.permutations(range(N)):
        order = find_order(p)
        orders[order] += 1
        total_permutations += 1
    # 计算最大阶T
    for i in range(T,0,-1):
        if(orders[i]!=0):
            T=i
            break
    # 统计频率
```

```
    cumulative_probabilities = [0] * (T + 1)
    for i in range(1, T + 1):
        cumulative_probabilities[i] = cumulative_probabilities[i - 1] + orders[i] /
total_permutations
    # 绘制p(K)
    print(f'Maximum order T for N = {N}: {T}')
    plt.plot(range(T + 1), cumulative_probabilities)
    plt.xlabel('K')
    plt.ylabel('p(K)')
    plt.title(f'p(K) curve for N = {N}; T = {T}')
    plt.show()
```

首先估计T的最大值（如10000），利用Python中itertools.permutations遍历所有可能的置乱做统计，保存在数组中，最大的不为0的下标即为最大阶T。统计各阶频率后累加即可得到概率分布函数，最后利用Python中的matplotlib.pyplot绘图将p(K)可视化展现出来。

1.2.2 n值较大的情况计算T

这里计算最大阶参考了 https://oeis.org/A000793 中的相关算法与文献，用到了Landau's function g(n)：

> Landau's function g(n): largest order of permutation of n elements. Equivalently, largest LCM of partitions of n.

具体的算法如下：

```
from sympy import primerange, sqrt, log, Rational
def f(N): # compute terms a(0)..a(N)
    V = [1 for j in range(N+1)]
    if N < 4:
        C = 2
    else:
        C = Rational(166, 125)
    for i in primerange(C*sqrt(N*log(N))):
        for j in range(N, i-1, -1):
            hi = V[j]
            pp = i
            while pp <= j:
                hi = max(V[j-pp]*pp, hi)
                pp *= i
            V[j] = hi
    return V
# Philip Turecek, Mar 31 2023
```

该算法可以获得：将n分解成若干数字的和，这些数字的最小公倍数的最大值。利用置乱阶的特殊性质：将n分解成若干数字的和，这些数字的最小公倍数的最大值，就是最大阶。因此可以直接得到最大阶T。

1.2.3 n值较大的情况计算p(K)

由大数定律，对给定的K，随机取100000（或更多）个置乱，计算其阶为K的概率，就可以得到p(K)概率密度函数的近似值。最后累加得到分布函数，绘制p(K)曲线即可。具体代码实现如下：

```
def large_n_plot_p_k_curve(N,T):
    # 统计基数
    radix=1000
    freq=1000
    cumulative_probabilities = [0] * (freq + 1)
```

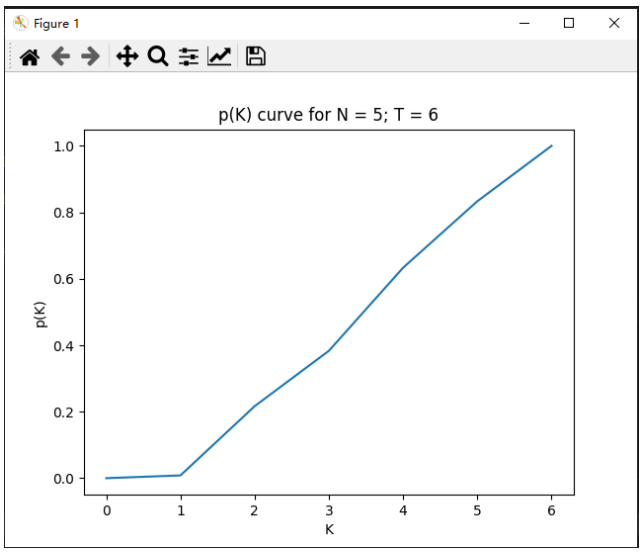
```
    i=0
    for K in range(0,T,T//freq):
        count=0
        for j in range(radix):
            p=generate_permutation(N)
            order = find_order(p)
            if order < K:
                count+=1
        cumulative_probabilities[i]=count/radix
        i+=1

    # 绘制p(K)
    print(f'Maximum order T for N = {N}: {T}')
    plt.plot(range(0,T,T//freq), cumulative_probabilities)
    plt.xlabel('K')
    plt.ylabel('p(K)')
    plt.title(f'p(K) curve for N = {N}; T = {T}')
    plt.show()
```

对给定的K，随机取1000（或更多）个参考点，每个参考点利用random.shuffle()随机产生1000（或更多）个置乱，然后统计阶小于K的数量，计算阶小于K的概率，存放在数组cumulative_probabilities中，由大数定律即得到p(K)概率的分布函数，。最后利用Python中的matplotlib.pyplot绘图将p(K)绘制出来。

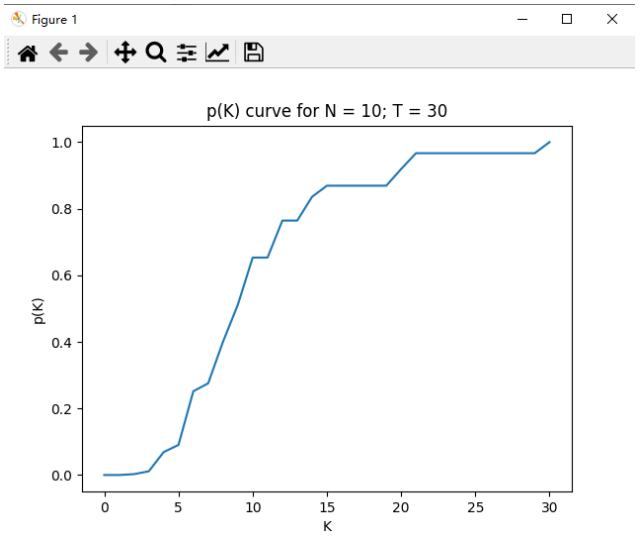
1.3 实验测试

1. 当n值较小时，使用算法1

   输入n=5，输出

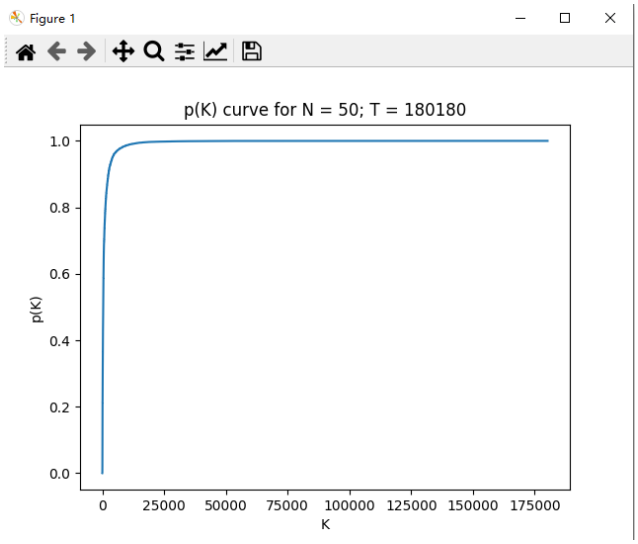


   输入n=10，输出

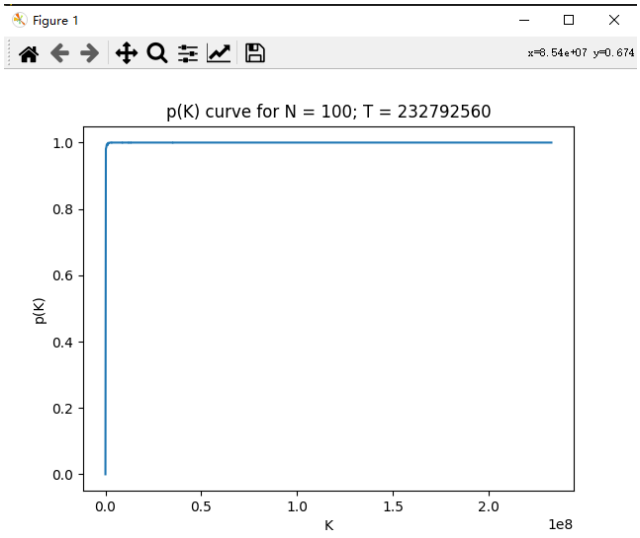


2. 当n值较大时，使用算法2

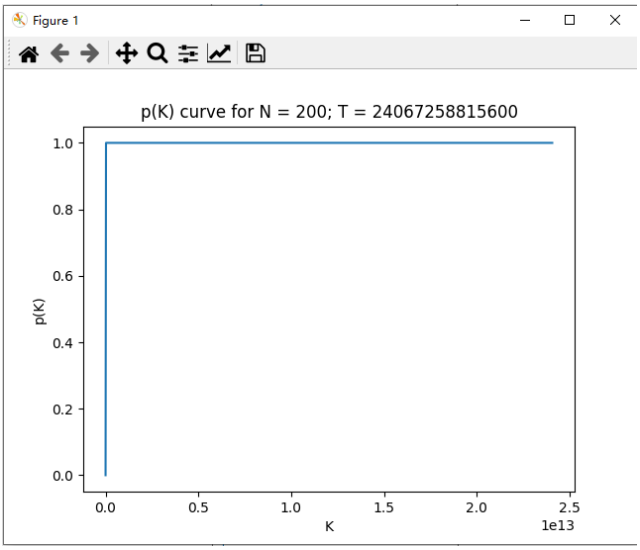
输入n=50，输出



输入n=100，输出



输入n=200，输出



1.4 结果分析

分析得到的p(K)图像以及最大阶T，我们可以得到以下结论：

1. 随着N的增长，最大阶T的变化幅度是巨大的，以较大幅度增长。
2. 随机置乱的阶大部分集中于较小的位置，而接近最大阶的置乱占比较小。这可以由p(K)函数图像看出：在阶数K较小时，p(K)快速增长，说明阶大部分集中于较小的位置。在阶数K较大时，p(K)增长较为缓慢趋于1，说明接近最大阶的置乱数量较小。
3. 随着N的增大，函数图像逐渐向左上角凸，这也是置乱的阶大部分集中于较小的位置，而接近最大阶的置乱占比较小的体现。可以猜测当N更大时，图像趋于一个直角。

2 问题二

以下是使用Logistic混沌映射构造置乱的一种常用方法：

◦ 映射函数为$x_{n+1} = \mu x_n(1 - x_n)(0 < x < 1)$，当$3.57 < \mu < 4$时，呈现混沌特性
◦ 选定参数μ，任选初始值x_0，迭代计算$x_1 - x_N$。将这N个数排序，以每个数的位置为置乱索引。例如，若x_i被排在第j位，则置乱中将第i个数移至第j位
◦ 编写一个程序，评测用该方法得到的置乱
  • 约定混沌函数的计算使用64位计算机，double类型
  • 自行考虑评测中所需要考虑的自它参数

2.1 实验原理与算法

1. 使用Logistic混沌映射构造置乱

按题目中所给的算法构造即可。

2. 评测用该方法得到的置乱

当评估置乱算法时，主要可以考虑以下指标：

1. **entropy(熵)：**熵是一个序列的不确定性度量，可以用来评估置乱后的序列的随机性。熵越高，序列的随机性越强，置乱效果越好。
2. **Correlation Coefficient(相关系数)：**相关系数是一个序列与另一个序列之间的线性相关性度量。可以用来评估置乱后的序列与原序列之间的相关性。相关系数越低，置乱效果越好。
3. **Move distance(位置差异度)：**原序列和置乱序列在位置上的差异程度，计算方法为原序列中每个数在置乱序列中的位置差的绝对值之和的平均值。位置差异度越大，表示置乱效果越好。

4. **Euclidean Distance (欧几里得距离)**: 这是衡量两个序列之间距离的常用方法。它计算了两个序列之间的直线距离的平均值。较小的欧几里得距离表示两个序列更相似。
5. **Inversion Count (逆序数)**: 这个指标计算了置乱序列中逆序对的数量。逆序对是指在原始序列中位置靠前的元素在置乱序列中位置靠后。逆序数越大，置乱程度越高。
6. **Normalized Inversion Count (归一化逆序数)**: 这个指标是逆序数除以序列长度的最大可能逆序数。它的值范围在0到1之间，其中0表示完全正相关，1表示完全负相关。

为分析Logistic混沌映射置乱，可以考虑如下比较：

1. 不同长度的指标差异
2. 不同μ参数的指标差异
3. 不同置乱方法的指标差异

2.2 代码实现

2.2.1 Logistic混沌映射构造置乱

```python
def logistic_map(mu, x, n):
for i in range(n):
x = mu * x * (1 - x)
return x

def logistic_pemutation(n, mu):
seq = [logistic_map(mu, 0.5, i) for i in range(1, n+1)]
# 排序并生成置乱序列
scramble_seq = [seq.index(x) for x in sorted(seq)]
return scramble_seq
```

首先根据映射函数$x_{n+1} = \mu x_n(1 - x_n)(0 < x < 1)$，$3.57 < \mu < 4$，构造$x_i$序列。

再将这N个数排序，以每个数的位置为置乱索引。这样就实现了Logistic混沌映射构造置乱。

2.2.2 评测Logistic混沌映射得到的置乱

对几个指标评估的具体函数如下：

```python
def entropy(seq1,seq2):
seq=np.abs(np.array(seq1) - np.array(seq2))
hist, _ = np.histogram(seq, bins=len(seq))
p = hist / len(seq)
return -np.sum(p * np.log2(p + 1e-12))

def correlation_coefficient(seq1, seq2):
return np.corrcoef(seq1, seq2)[0, 1]

def move_distance(seq1, seq2):
return np.sum(np.abs(np.array(seq1) - np.array(seq2)))

def euclidean_distance(seq1, seq2):
return np.sqrt(np.sum((np.array(seq1) - np.array(seq2)) ** 2))

def count_inversions(seq1, seq2):
inv_count = 0
for i in range(n):
for j in range(i+1, n):
```

```
    if seq1[i] < seq1[j] and seq2.index(seq[i]) > seq2.index(seq[j]):
        inv_count += 1
return inv_count
```

各个指标具体的实现思路如下：

1. **entropy(熵)：** 思路是以每个数字移动的相对距离为指标，计算该数据集的熵，公式如下：

$$H(p) = -\Sigma plog(p)$$

2. **Correlation Coefficient(相关系数)：** 相关系数是一个序列与另一个序列之间的线性相关性度量。这里调用了np.corrcoef的函数计算。
3. **Move distance(位置差异度)：** 原序列和置乱序列在位置上的差异程度，计算方法为原序列中每个数在置乱序列中的位置差的绝对值之和的平均值。

$$d = \frac{1}{N}\Sigma|d_1 - d_2|$$

4. **Euclidean Distance (欧几里得距离)**: 这是衡量两个序列之间距离的常用方法。它计算了两个序列之间的直线距离的平均值。

$$d = \frac{1}{N}\sqrt{\Sigma|d_1 - d_2|^2}$$

5. **Inversion Count (逆序数)**: 这个指标计算了置乱序列中逆序对的数量。这里采用了遍历统计的方法统计了所有在原始序列中位置靠前的元素在置乱序列中位置靠后的元素，当n较大时该指标不便于统计，故舍去。
6. **Normalized Inversion Count (归一化逆序数)**: 这个指标是逆序数除以序列长度的最大可能逆序数。计算公式为`inv_count / (len(seq) * (len(seq) - 1) / 2)`。

此外，采用random.shuffle以及Logistic混沌映射分别评测，比较两者性能。

2.3 实验测试

实验中选取的参数如下：

```
num=[100,1000,10000]
mu=[3.6,3.7,3.8]
method=['random','logistic']
```

分别对不同长度，不同参数μ，不同算法的置乱评测参数作比较，结果如下：

```
Mode:random|Number:100
Entropy: 5.71101406453672
Correlation coefficient: -0.08206420642064205
Move distance: 34.24
Euclidean_distance: 4.2464808959890545
count_inversions: 2606
Normalized Inversion Count: 0.5264646464646464

Mode:logistic|Number:100|Mu:3.6
Entropy: 5.703465189516528
Correlation coefficient: -0.015001500150015001
Move distance: 33.74
Euclidean_distance: 4.1127849445357585
count_inversions: 2500
Normalized Inversion Count: 0.5050505050505051
```

```
Mode:logistic|Number:100|Mu:3.7
Entropy: 5.714917659789467
Correlation coefficient: -0.009624962496249624
Move distance: 32.36
Euclidean_distance: 4.101877618847252
count_inversions: 2484
Normalized Inversion Count: 0.5018181818181818

Mode:logistic|Number:100|Mu:3.8
Entropy: 5.781209564618931
Correlation coefficient: -0.023090309030903085
Move distance: 33.8
Euclidean_distance: 4.12914034636751
count_inversions: 2505
Normalized Inversion Count: 0.5060606060606061

Mode:random|Number:1000
Entropy: 8.932316428941057
Correlation coefficient: 0.00042184842184842187
Move distance: 334.532
Euclidean_distance: 12.907214726655786
count_inversions: 249723
Normalized Inversion Count: 0.49994594594594594

Mode:logistic|Number:1000|Mu:3.6
Entropy: 8.956655125362923
Correlation coefficient: -0.0015000015000015002
Move distance: 334.012
Euclidean_distance: 12.919616867384264
count_inversions: 250000
Normalized Inversion Count: 0.5005005005005005

Mode:logistic|Number:1000|Mu:3.7
Entropy: 8.967149516464854
Correlation coefficient: 0.0016525816525816528
Move distance: 333.258
Euclidean_distance: 12.89926625820244
count_inversions: 249478
Normalized Inversion Count: 0.49945545545545544

Mode:logistic|Number:1000|Mu:3.8
Entropy: 8.96946167596013
Correlation coefficient: 0.007145371145371145
Move distance: 328.278
Euclidean_distance: 12.863732195595492
count_inversions: 248534
Normalized Inversion Count: 0.4975655655655656

Mode:random|Number:10000
Entropy: 12.257269962188008
Correlation coefficient: 0.002447422812474228
Move distance: 3325.306
```

```
Mode:logistic|Number:10000|Mu:3.6
Entropy: 12.265108591516949
Correlation coefficient: -0.00015000000150000004
Move distance: 3333.5318

Mode:logistic|Number:10000|Mu:3.7
Entropy: 12.262482761125717
Correlation coefficient: -0.00022409214224092144
Move distance: 3342.6976

Mode:logistic|Number:10000|Mu:3.8
Entropy: 12.263297064246139
Correlation coefficient: 0.0026743270347432705
Move distance: 3335.4622
```

2.4 结果分析

根据得到的数据评测Logistic混沌映射得到的置乱如下：

1. **entropy(熵)：** 熵越高，序列的随机性越强，置乱效果越好。对比数据可知，Logistic混沌映射以相对距离为参数的熵相对较高，随机性较好。此外，N的长度越大，熵有所提高，说明N越大Logistic混沌映射序列的随机性越强。而对于不同的μ参数，熵则相差不大。
2. **Correlation Coefficient(相关系数)：** 相关系数越低，置乱效果越好。对比数据可知，Logistic混沌映射与原有序序列的相关系数相对较低，随机性较好。此外，N的长度越大，相关系数有所降低，同样说明N越大Logistic混沌映射序列的随机性越强。而对于不同的μ参数，相关系数也有所差异，比较三个参数，$\mu = 3.8$时相关系数较高，随机性略差。
3. **Move distance(位置差异度)：** 位置差异度越大，表示置乱效果越好。对比数据可知，Logistic混沌映射与原有序序列的位置差异度相对较高，随机性较好。此外，N的长度越大，位置差异度越高，同样说明N越大Logistic混沌映射序列的随机性越强。而对于不同的μ参数，位置差异度差别不大。
4. **Euclidean Distance (欧几里得距离)**: 与位置差异度相似，欧几里得距离越大，表示置乱效果越好。对比数据可知，Logistic混沌映射与原有序序列的欧几里得距离相对较高，随机性较好。
5. **Inversion Count (逆序数)**: 逆序数越大，置乱程度越高。对比数据可知，Logistic混沌映射逆序数对数相对较高，随机性较好。
6. **Normalized Inversion Count (归一化逆序数)**: 这个指标是逆序数除以序列长度的最大可能逆序数。它的值范围在0到1之间，其中0表示完全正相关，1表示完全负相关。对于越随机的变量，两个元素顺序与逆序的概率应相同，即归一化逆序数区域0.5。对比数据可知，Logistic混沌映射以归一化逆序数非常接近0.5，说明随机性较好。

综上，Logistic混沌映射的随机性较好。且N越大随机性越高。此外对不同的参数μ随机性有差异，实际使用时应当选择合适的μ作为参数。

此外，对比Logistic混沌映射以及调用random.shuffle的数据，在N较小时，random.shuffle的数据相关系数高于Logistic混沌映射。而当N较大时，两者的指标相差不大，因此可以认为在N较小时Logistic混沌映射得到的置乱优于random.shuffle得到的置乱，而当N较大时，两者性能相差不大。

3 实验总结

在问题一中，我们对于N个数的置乱计算了其最大阶T，并绘制了任意一个置乱阶小于K的概率分布p(K)曲线。在问题二中，我们通过Logistic混沌映射构造置乱，并从多角度的评测指标评测了Logistic混沌映射构造置乱的性能，最终得出结论Logistic混沌映射随机性较好，适合于构造随机的置乱序列。