

# 中国科学技术大学

# 本科学位论文



英语字母频率分析

|       |            |
|-------|------------|
| 作者姓名: | 吴立凡        |
| 学科专业: | 网络空间安全     |
| 导师姓名: | 李卫海        |
| 完成时间: | 2020 年 5 月 |

University of Science and Technology of China  
A dissertation for bachelor's degree



English letter frequency analysis

|                 |                     |
|-----------------|---------------------|
| Author :        | Lifan Wu            |
| Speciality :    | Cyberspace Security |
| Supervisor :    | Weihai Li           |
| Finished Time : | May, 2020           |



## 致谢

感谢 CSDN 的前辈们, 他们的论文博客为本次实验提供了丰富的经验与建议。如果不是他们的帮助, 这篇文章的完成也会变得无比艰难。

另外也要感谢所有教过我的老师们, 他们所教授的许多内容都成为了我实践的坚实基础。在此特别感谢李卫海老师, 他所教授的密码学导论为这篇论文提供了莫大的帮助。

## 目录

|                                   |    |
|-----------------------------------|----|
| 摘要 .....                          | 5  |
| 关键词 .....                         | 6  |
| Abstract .....                    | 6  |
| Keywords: .....                   | 6  |
| 正文 .....                          | 7  |
| 题目要求 .....                        | 7  |
| 算法分析 .....                        | 7  |
| 完成效果 .....                        | 8  |
| 分析《圣经》中字母统计分布 .....               | 9  |
| 三字符的可视化处理 .....                   | 12 |
| 用维吉尼亚密码加密这本书 .....                | 15 |
| 使用 kasiski 方法破解 vigenere 密码 ..... | 17 |
| 参考文献 .....                        | 19 |

## 摘要

对于代换密码，统计分析是一种很有效的攻击方式，这是因为简单代换密码加密后的结果往往有很多统计特征，给攻击方许多攻击的机会。本次研究开发了一个用 c++语言和 python 开发的文字统计分析工具，用于统计一段文字中单字符、双字符、三字符的出现频率，并使用图表表示出来。然后使用简单的代换密码：维吉尼亚密码对《圣经》进行加密，对密文进行统计分析，并用 Kasiski 方法分析密钥长度。

## 关键词

字母 频率 双字符 密度图 维吉尼亚密码 Kasiski 方法

## Abstract

For substitution passwords, statistical analysis is a very effective attack method. This is because the result of simple substitution password encryption often has many statistical characteristics, giving the attacker many opportunities for attack. In this research, a text statistical analysis tool developed in c++ language and python was developed to count the frequency of single, double, and three characters in a text, and use graphs to show it. Then use a simple substitution password: the Virginia cipher encrypts the "Bible", performs statistical analysis on the ciphertext, and uses the Kasiski method to analyze the key length.

## Keywords:

Letter frequency    two-character    density map    Virginia cipher    Kasiski method

# 正文

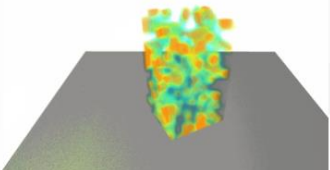
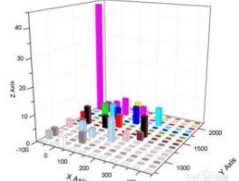
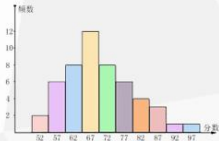
## 题目要求

课程实践

94 / 94

题目1：统计分析工具

- 编写一个软件，实现以下功能：统计一段文字中单字符、双字符、三字符的出现频率，分别用你直方图、三维直方图、三维密度图表示出来。字符范围应可以指定；画图部分可以包含在的软件之内，也可以单独使用某个工具实现。



- 自选一本英文书籍，用该工具分析字母统计分布（字母改为小写，删除所有非字母的符号）
- 用维吉尼亚密码加密这本书，对得到的密文进行统计分析
- 根据密文统计结果，用Kasiski方法分析密钥长度。（你发现了多少可用的串？是否遇到了偶然的干扰？）

使用语言：c 语言、python

需求分析：读取一个文件并统计单字符、双字符、三字符的出现频率，分别用你直方图、三维直方图、三维密度图表示出来。字符范围应可以指定；画图部分可以包含在的软件之内，也可以单独使用某个工具实现。

本实验的所有代码都已经放在了附件里

## 算法分析

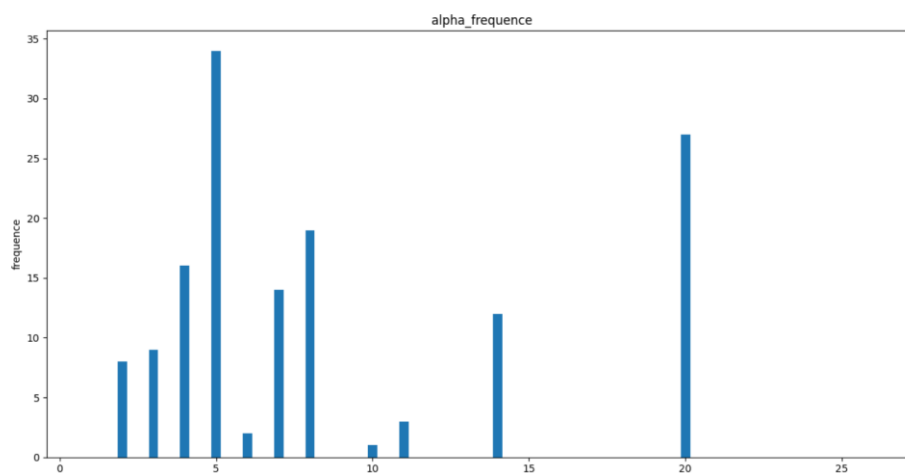
大致思路：统计部分使用 c 语言编写，先使用 fileload 函数读取文件内容存放在 cipher 数组里，再对 cipher 里的内容经行遍历，使用一维二维三维数组统计词频，每遇到一个字母或者双字符将对应位置的统计数组值加一，最后将文件保存

在 `alpha_frequency.txt` 中，（实现以上功能的代码都已经封装成函数并保存在 `frequency.h` 库文件中）

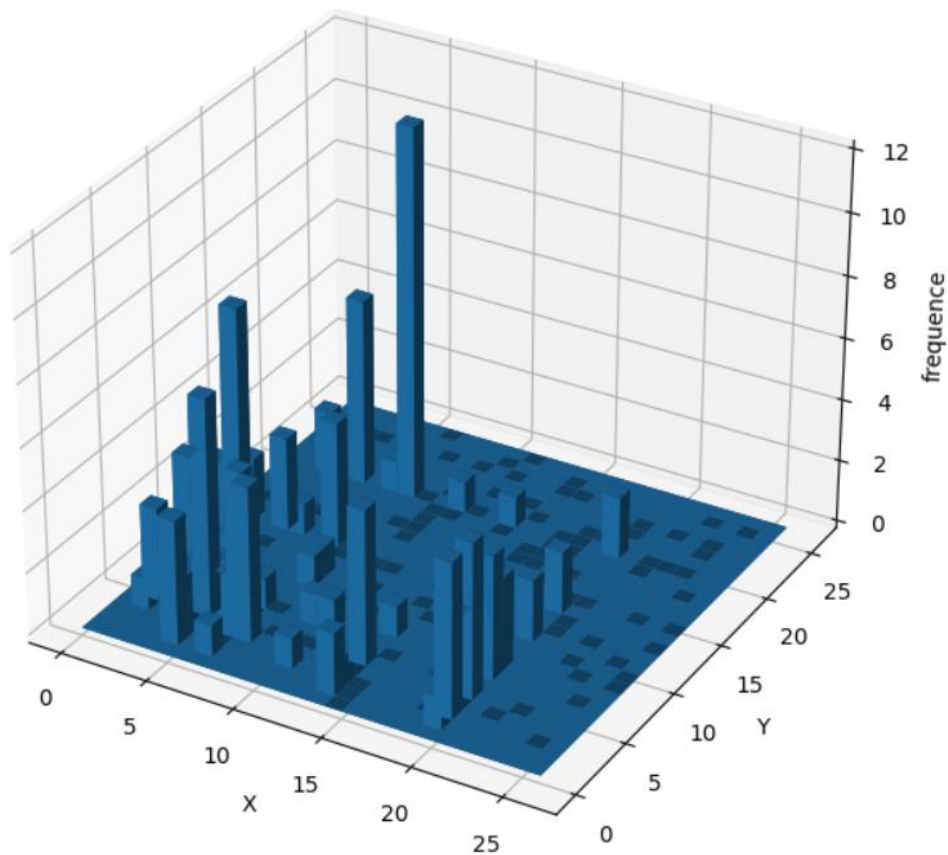
再执行 `system("python alpha_frequency.py");`来使用 python 绘制图表，使用了 `matplotlib` 库来进行表格的绘制。以实现数据的可视化处理。

## 完成效果

将要统计的文件命名为 `plaintext.txt` 文件里，执行程序后，会统计文件里的字母频数、双字母频数，并绘制出如下表格



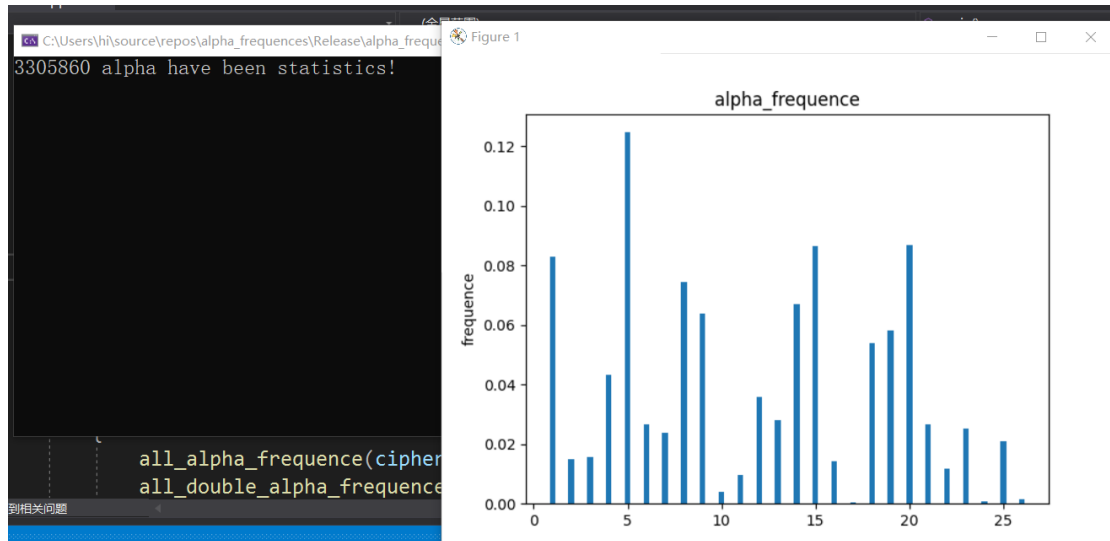




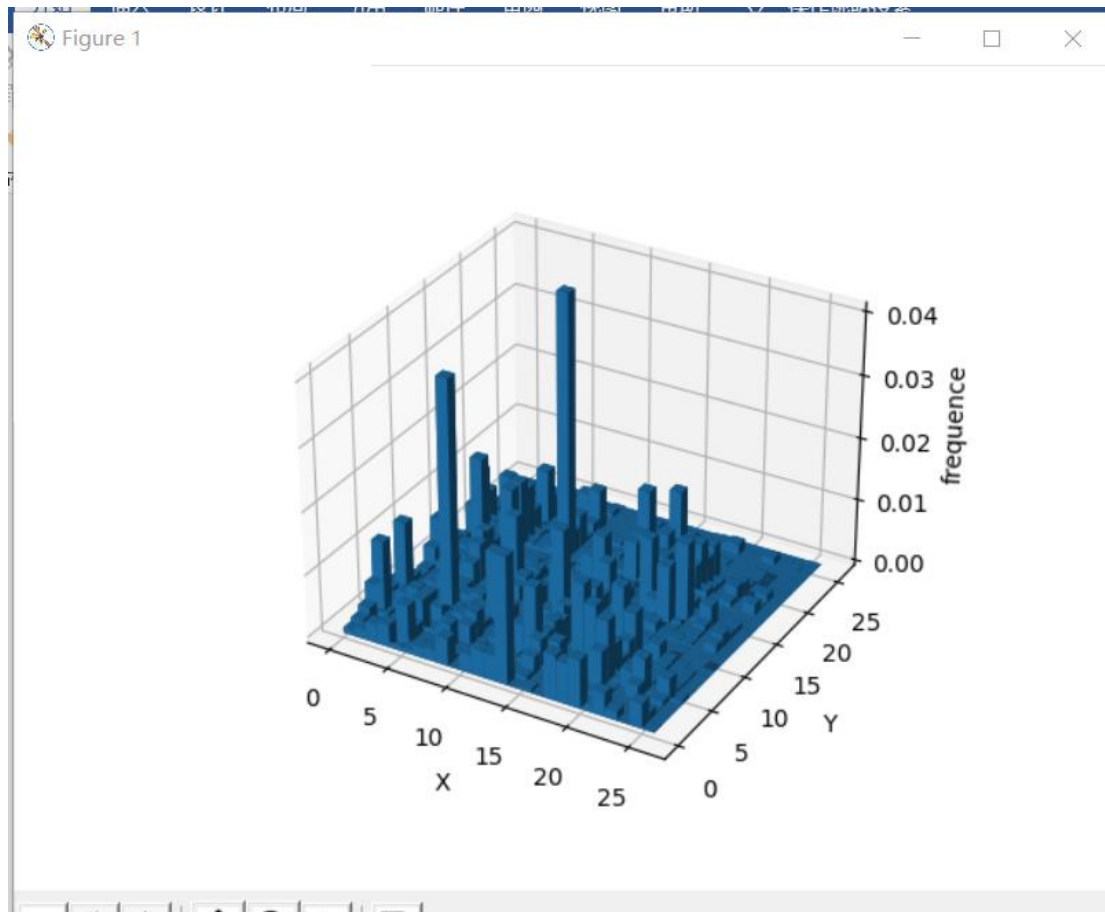
## 分析《圣经》中字母统计分布

(字母改为小写, 删除所有非字母的符号)

这里选用一本很长的的书《圣经》，一共 3046103 个字符，统计字母频率如下：



可以看出，字母 e 频率最高，其次是 t、o、a，与英文字母出现频率相符合。  
双字母频率如下：



我们可以顺便分析一下文件 double\_alpha\_frequency.txt，看一下出现频率前几的双字母：

```
test.py > ...
1
2 import numpy as np
3 all_double_alpha=list(map(int,input().split()))
4 alpha_frequency = np.array(all_double_alpha)
5 a=alpha_frequency.argsort()[-10:][::-1]
6
7 for i in a:
8     alpha1=int(i/26)
9     alpha2=i%26
10     print(chr(alpha1+ord('a')),chr(alpha2+ord('a')))
11 #print(a)
```

问题 1 输出 调试控制台 终端

```
n d
e r
i n
e n
o f
e s
t o
o u
[501 186 341 121 221 117 369 122 508 384]
wusar@LAPTOP-31I88NQC:~/crypt_homework$ cat double_alpha_frequency.txt | python3 test.py
t h
h e
n d
e r
i n
e n
o f
e s
t o
o u
wusar@LAPTOP-31I88NQC:~/crypt_homework$
```

出现频率前几的双字母有：th、er、in 等，这些都是英文中常见的双字符，也是最常见的英文词根。

我们采用同样的方法分析一下三字母出现的频率：

```

2 import numpy as np
3 all_double_alpha=list(map(int,input().split()))
4 alpha_frequence = np.array(all_double_alpha)
5 a=alpha_frequence.argsort()[-10:][::-1]
6
7 for i in a:
8     alpha1=int((i/(26*26)))
9     alpha2=int((i/(26))%26)
10    alpha3=i%26
11    print(chr(alpha1+ord('a')),chr(alpha2+ord('a')),chr(alpha3+ord('a')))
12 print(a)

```

问题 2 输出 调试控制台 终端

```

y * u
o ã h
f ñ r
n ð h
h è r
h ġ s
f ö h
[13030 341 5752 16608 9965 3761 9289 4853 4958 3881]
wusar@LAPTOP-31I88NQC:~/crypt_homework$ cat triple_alpha_frequence.txt | python3 test2.py
the
and
ing
you
oth
for
nth
her
his
fth
[13030 341 5752 16608 9965 3761 9289 4853 4958 3881]
wusar@LAPTOP-31I88NQC:~/crypt_homework$

```

Interpreter 0.2

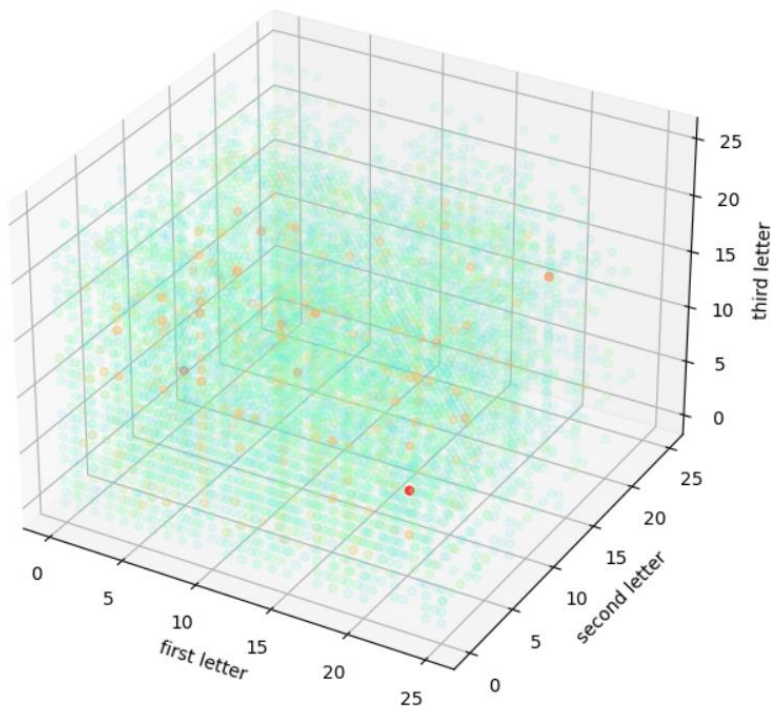
可以看到，出现频率前几的三字符为：the and ing 等

## 三字符的可视化处理

至于三字符的可视化处理，就要麻烦很多

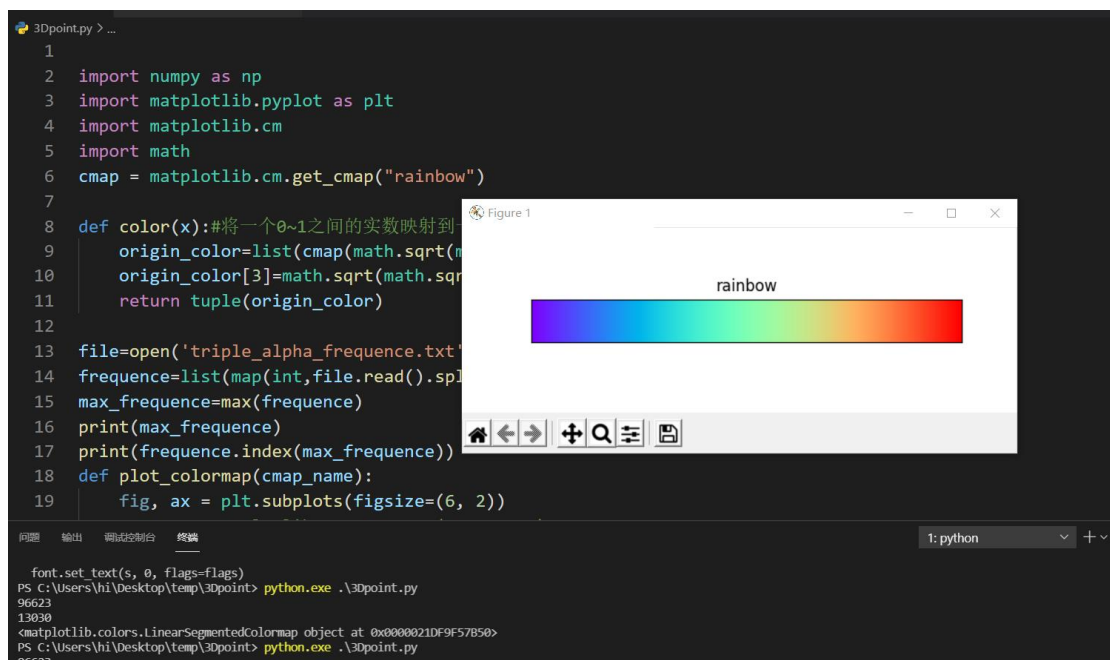
我们采用一个三维图中的一个点表示一个三字符的统计信息。

点的  $x$ ,  $y$ ,  $z$  三个坐标轴表示第 1、2、3 个字母，使用点的颜色和透明度代表对应三字符的出现频率。实现效果如下：

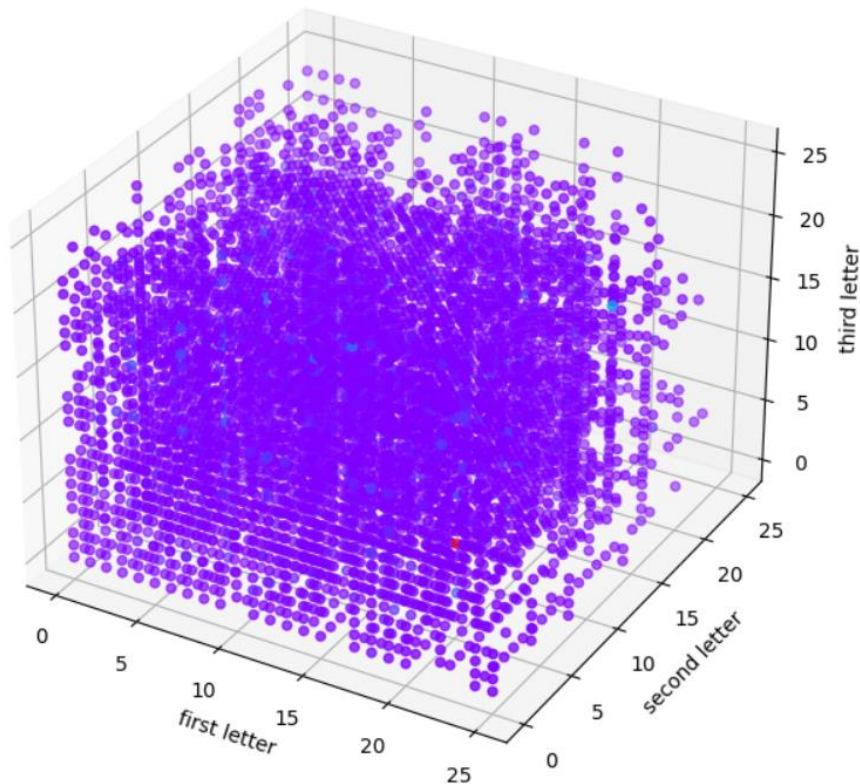


这里面最困难的部分在于从频率到颜色的映射，我使用 matplotlib 里面的 colormap 来实现。

Colormap 将一个 0~1 之间的实数映射到一个 rgb 颜色三元组，如图：最左边的蓝色表示 0，最右边的红色表示 1，这样就实现了颜色的表示



但是如果直接使用颜色表示，不经任何其他处理的话，图像就是这样：



可视化效果很差！

这是由于实际的三字符分布方差很大，出现频数高的三字符如“the”出现了96623次之多！而绝大部分三字符的出现频率都是0。

因此我们需要对颜色进行一些修正，修正的方法采用老师们经常使用的调分大法，直接对频率进行多次开方处理，并根据开方后的大小设置颜色，具体代码如下：

```
def color(x):#将一个0~1之间的实数映射到一个颜色点，实数越大颜色越红越深
    origin_color=list(cmap(math.sqrt(math.sqrt(math.sqrt(x)))))
    origin_color[3]=math.sqrt(math.sqrt(x))#设置透明度
    return tuple(origin_color)#返回颜色的三元组
```

经测试，这样的调整能较好看的展现出三字母的频率分布。

## 用维吉尼亚密码加密这本书

加密的代码写在 `vigenere.cpp` 中，读取 `plaintext.txt` 文件，根据 `key` 使用维吉尼亚密码加密该文件，将加密后的结果保存在 `cipher.txt` 中，再写一个 `vigenere_decode.cpp` 代码，将加密后的 `cipher.txt` 文件进行解密，将解密后的文件保存在 `decode.txt` 中，验证一下，可以看到 `decode.txt` 与 `plaintext.txt` 中字母相同。说明加密解密算法写的没有问题。

使用上面的字母频率统计工具，统计 `cipher.txt` 里的字母频率、双字母频率，如下：

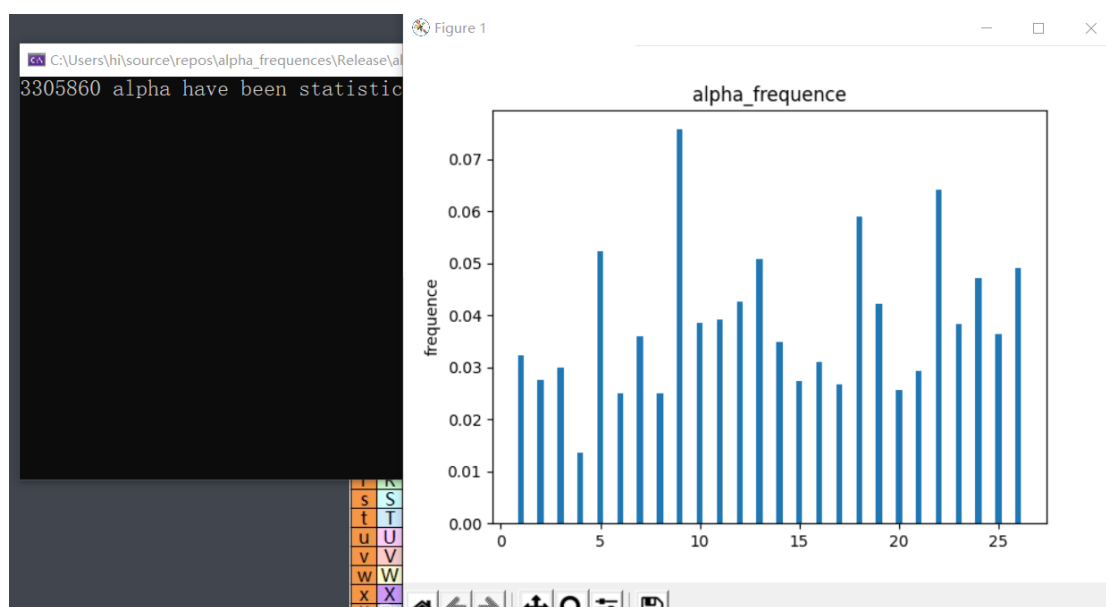
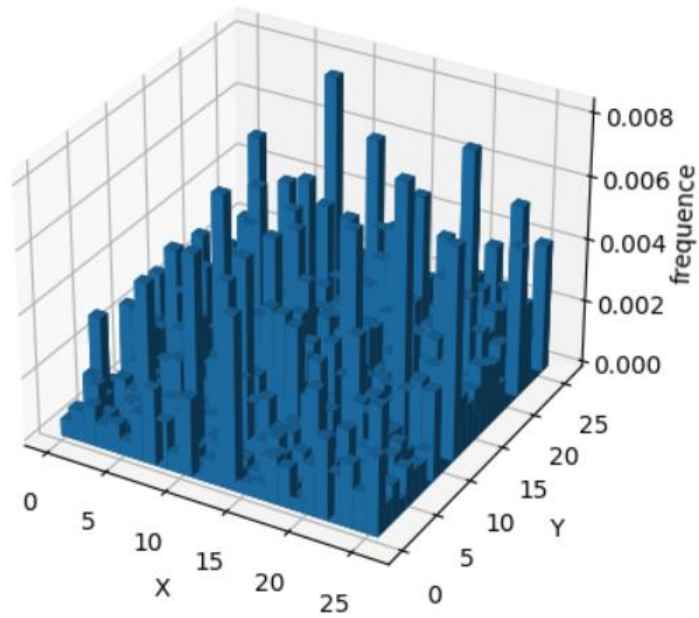
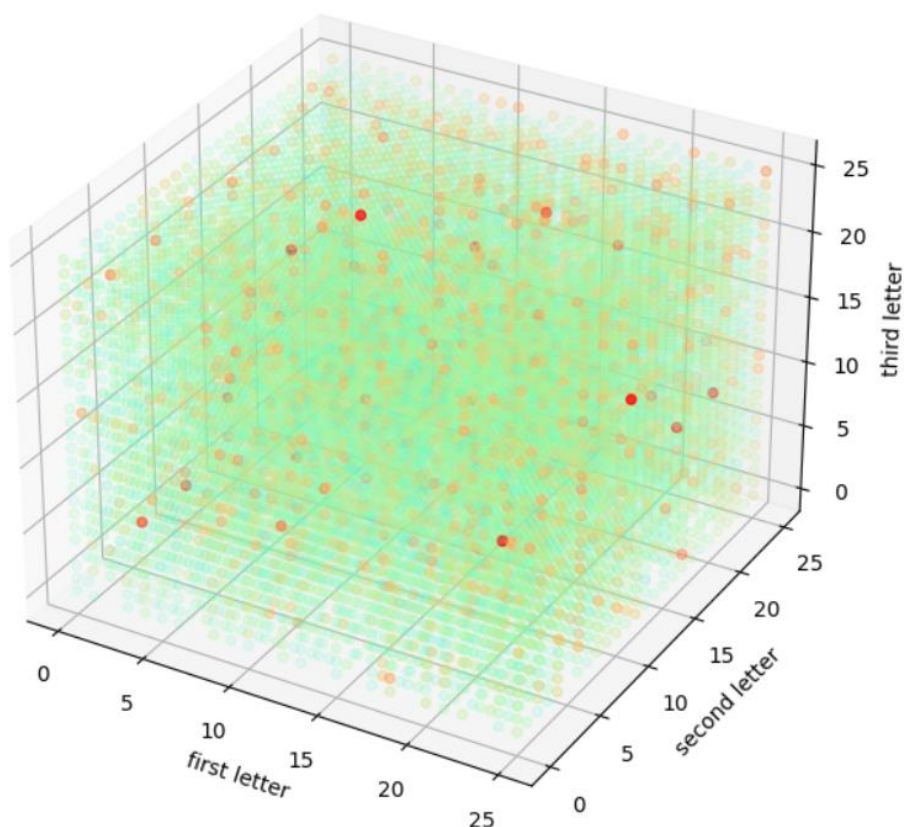


Figure 1



三字符的统计分布图：





在这里，我发现使用维吉尼亚密码加密后的密文字母不是平均分布的！

在单字母频率分析中，密文里面字母 i 出现的频率最大，其原因也很好理解，在明文中字母 e 出现频率最大，密钥选用“vigenere”，e 出现的次数也最多，而当明文为 e，密钥为 e 时，加密出来的字母就是 i！所以可以看出来，维吉尼亚密码并不能消除明文中的频率统计特征。

但是我们也看出，加密后密文的分布方差要明显比明文小很多，我们可以从信息熵的角度考虑这个问题：

$$H(M)=H(C|K)\leq H(C)$$

M、C、K 为明文、密文、密钥，密文的熵是要小于明文的熵的。所以比起明文，密文会显得更加随机。

## 使用 kasiski 方法破解 vigenere 密码

这里我们选取密文中出现频率最大的三字符进行分析

```
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequency$ cat cipher_triple_alpha_frequency.txt | python3 test2.py
x y i
g l v
b n i
z l r
k l z
x u i
x c m
o p k
v v j
g r q
[16180 4363 1022 17203 7071 16076 15612 9864 14751 4514]
```

但是当我们直接求这个三字符的出现位置的差的最大公因数的时候, 就会发现求出来的结果是 1。

```
kasiski.py > ...
10 def multi_gcd(array):
11     l = len(array)
12     if l == 1:
13         return array[0]
14     elif l == 2:
15         return gcd(array[0], array[1])
16     else:
17         return gcd(multi_gcd(array[:l//2]), multi_gcd(array[l//2:]))
18
19 cipher=input()
20 pos=[]
21 for i in range(len(cipher)-2):
22     if cipher[i]=='x' and cipher[i+1]=='y' and cipher[i+2]=='i':
23         pos.append(i)
24 div=[]
25 for i in range(len(pos)-1):
26     div.append(pos[i+1]-pos[i])
27
28 print(multi_gcd(div))

问题 输出 调试控制台 终端 1: bash
36, 112, 280, 24, 288, 272, 232, 256, 200, 72, 112, 208, 192, 136, 488, 16, 744, 24, 112, 88, 120, 24, 72, 184, 96, 56, 240, 112, 112, 88, 80,
256, 696, 408, 160, 236, 52, 192, 16, 256, 64, 344, 136, 104, 88, 128, 240, 152, 8, 104, 904, 480, 424, 504, 120, 200, 136, 32, 56, 88, 264,
752, 768, 488, 96, 320, 176, 296, 488, 40, 232, 368, 8, 264, 152, 160, 376, 72, 496, 344, 88, 48, 80, 102, 306, 664, 416, 256, 352, 208, 72,
16, 344, 272, 48, 800, 416, 40, 296, 744, 72, 80, 280, 160, 64, 280, 312, 72, 64, 24, 120, 112, 376, 376, 528, 544, 56, 360, 216, 440, 336, 6
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequency$ cat cipher.txt | python3 kasiski.py
1
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequency$
```

这是因为这个三字符在密文中出现的次数太多了, 所以不满足 kasiski 方法的使用条件: 不同的明文获得相同密文的巧合很少发生。

所以我们可以只选择较少的一部分密文进行分析, 如图:

```
kasiski.py > ...
10 def multi_gcd(array):
11     l = len(array)
12     if l == 1:
13         return array[0]
14     elif l == 2:
15         return gcd(array[0], array[1])
16     else:
17         return gcd(multi_gcd(array[:l//2]), multi_gcd(array[l//2:]))
18
19 cipher=input()
20 pos=[]
21 for i in range(len(cipher)-2):
22     if cipher[i]=='x' and cipher[i+1]=='y' and cipher[i+2]=='i':
23         pos.append(i)
24 div=[]
25 for i in range(25,30):
26     div.append(pos[i+1]-pos[i])
27
28 print(multi_gcd(div))

问题 输出 调试控制台 终端
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequence$ cat cipher.txt | python3 kasiski.py
24
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequence$ cat cipher.txt | python3 kasiski.py
2
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequence$ cat cipher.txt | python3 kasiski.py
8
wusar@LAPTOP-31I88NQC:~/crypt_homework/alpha_frequence$
```

这样就可以求出来密钥长度应该是 8

## 参考文献

无