我没有手动去数，我写了一个python代码(后面附上源代码),对序列进行分析，思路是，先算出其DNA互补链和RNA链，再找起始密码子(AUG和GUG)，根据起始密码子之后计算其和CG含量平均相减的方差，方差越小越可能，然后找起始链结束链的终止密码子，看看DNA会不会异常终止结束，最后找找稀有密码子和几乎禁止的密码子(书上有)，如果有稀有密码子那么可能性存疑，最后，找其是否有SD序列和编码的AA是否常见，如果是很奇怪的RNA也认为不可能。

分析如下：

输入的起始序列：

CGCTTGCGTGCGTCTCATCTGGAGGCTGCACCTGTCCGACCACTACGGCATCTGGCAGGAATTCGGCCAGAAGTTCCTCGTCGACCCCGACATCATCAAG

输入的结束序列：

GGTGCGAGAGCGGATACTCCAGGACCTTGCGCACCTACCTGTCCTACGGCAGCAGCGACAACGCGAACCCAGTCTGCGAGACACGGCTGTGCAGGCCGTC

由于中间是用…………隔开的，所以把前后序列分开统计，前面统计起始密码子，顺便检查下有没有按照起始密码子阅读框存在下的终止密码子

下面的是DNA按照碱基互补配对原则和倒置之后存在的

将输入的结束序列逆序互补后作为起始为：

GACGGCCTGCACAGCCGTGTCTCGCAGACTGGGTTCGCGTTGTCGCTGCTGCCGTAGGACAGGTAGGTGCGCAAGGTCCTGGAGTATCCGCTCTCGCACC

将输入的起始序列逆序互补后作为结束为：

CTTGATGATGTCGGGGTCGACGAGGAACTTCTGGCCGAATTCCTGCCAGATGCCGTAGTGGTCGGACAGGTGCAGCCTCCAGATGAGACGCACGCAAGCG

给的DNA起始转录后为：

GACGGCCUGCACAGCCGUGUCUCGCAGACUGGGUUCGCGUUGUCGCUGCUGCCGUAGGACAGGUAGGUGCGCAAGGUCCUGGAGUAUCCGCUCUCGCACC

给的DNA结束转录后为：

CUUGAUGAUGUCGGGGUCGACGAGGAACUUCUGGCCGAAUUCCUGCCAGAUGCCGUAGUGGUCGGACAGGUGCAGCCUCCAGAUGAGACGCACGCAAGCG

给的DNA互补起始转录后为：

CGCUUGCGUGCGUCUCAUCUGGAGGCUGCACCUGUCCGACCACUACGGCAUCUGGCAGGAAUUCGGCCAGAAGUUCCUCGUCGACCCCGACAUCAUCAAG

给的DNA互补结束转录后为：

GGUGCGAGAGCGGAUACUCCAGGACCUUGCGCACCUACCUGUCCUACGGCAGCAGCGACAACGCGAACCCAGUCUGCGAGACACGGCUGUGCAGGCCGUC

给的链找起始子： [['GUG', 'CGC', 'AAG', 'GUC', 'CUG', 'GAG', 'UAU', 'CCG', 'CUC', 'UCG', 'CAC'], ['GUG', 'UCU', 'CGC', 'AGA', 'CUG', 'GGU', 'UCG', 'CGU', 'UGU', 'CGC', 'UGC', 'UGC', 'CGU', 'AGG', 'ACA', 'GGU', 'AGG', 'UGC', 'GCA', 'AGG', 'UCC', 'UGG', 'AGU', 'AUC', 'CGC', 'UCU', 'CGC', 'ACC']]

互补链找起始子： [['GUG', 'CGU', 'CUC', 'AUC', 'UGG', 'AGG', 'CUG', 'CAC', 'CUG', 'UCC', 'GAC', 'CAC', 'UAC', 'GGC', 'AUC', 'UGG', 'CAG', 'GAA', 'UUC', 'GGC', 'CAG', 'AAG', 'UUC', 'CUC', 'GUC', 'GAC', 'CCC', 'GAC', 'AUC', 'AUC', 'AAG']]

给的链找中止子： [['UUG', 'AUG', 'AUG', 'UCG', 'GGG', 'UCG', 'ACG', 'AGG', 'AAC', 'UUC', 'UGG', 'CCG', 'AAU', 'UCC', 'UGC', 'CAG', 'AUG', 'CCG', 'UAG'], ['UGA'], ['UGA', 'UGA'], ['UGA', 'UGA', 'UGU', 'CGG', 'GGU', 'CGA', 'CGA', 'GGA', 'ACU', 'UCU', 'GGC', 'CGA', 'AUU', 'CCU', 'GCC', 'AGA', 'UGC', 'CGU', 'AGU', 'GGU', 'CGG', 'ACA', 'GGU', 'GCA', 'GCC', 'UCC', 'AGA', 'UGA']]

互补链找中止子： []

按照起始密码子的阅读框，把前三个密码子的CG比例算出来，然后减去统计学密码子CG含量的平均数，平方之后得到方差，方差越大越可能不是真正的序列

给的链前三个密码子CG比例： [[100.0, 33.33333333333333, 66.66666666666666], [33.33333333333333, 100.0, 33.33333333333333]]

互补链前三个密码子CG比例 [[66.66666666666666, 66.66666666666666, 33.33333333333333]]

给的链前三个密码子CG的方差： [761.6666666666666, 2299.3333333333335]

互补链前三个密码子CG的方差： [1256.3333333333333]

可以看到，给的正链的密码子方差，有一个远大于其他的，直接排除，留下一条正链一条反链的

给的链最可能起始子： [['GUG', 'CGC', 'AAG', 'GUC', 'CUG', 'GAG', 'UAU', 'CCG', 'CUC', 'UCG', 'CAC']]

互补链最可能起始子： [['GUG', 'CGU', 'CUC', 'AUC', 'UGG', 'AGG', 'CUG', 'CAC', 'CUG', 'UCC', 'GAC', 'CAC', 'UAC', 'GGC', 'AUC', 'UGG', 'CAG', 'GAA', 'UUC', 'GGC', 'CAG', 'AAG', 'UUC', 'CUC', 'GUC', 'GAC', 'CCC', 'GAC', 'AUC', 'AUC', 'AAG']]

两个序列扔进代码里面分析，确实没有终止子，密码子的偏好分析如下，也没有什么稀有密码子和TTA，CTA这种几乎禁止的密码子。考虑到互补链的终止子不存在，所以优先考虑给的正链的

给定链密码子偏好分析： {'GUGCGCAAGGUCCUGGAGUAUCCGCUCUCGCAC': '稀有密码子0,禁止密码子0',

'GUGCGUCUCAUCUGGAGGCUGCACCUGUCCGACCACUACGGCAUCUGGCAGGAAUUCGGCCAGAAGUUCCUCGUCGACCCCGACAUCAUCAAG': '稀有密码子0,禁止密码子0'}

我个人倾向于认定正链上的是正确的，因为可以看到，其和标准的前几个密码子比较匹配，而且，我们把它之前的链翻出来，可以看到

AGUGGUCGGACAG GUG CAGCCUCCAGAUGAGACGCACGCAAGCG。GUG前面有一段明显的AG序列有点像SD序列。所以我认为是有一个链的，其起始的前半部分为：['GUG', 'CGC', 'AAG', 'GUC', 'CUG', 'GAG', 'UAU', 'CCG', 'CUC', 'UCG', 'CAC']

CGC------AAG------GUC------CUG----GAG-----UAU-----CCG------CUC------

Arg(46.6%)-Lys(97.7%)-Val(62.1%)-Leu(50%)-Glu(83.9%)-Tyr(3.4%)-Pro(48.2%)-Leu(48.5%)-

UCG------CAC

Ser(24.5%)-Asp(97.1%)

由此可见，这个序列里面不仅没有稀有密码子，还基本上都是常见氨基酸的常见密码子，所以，我认为以上序列就是起始序列。

# -\*- coding=utf-8 -\*-

# This is the code to deal 19.6.18's homework

# Copyleft: everybody in ECUST grades 3 major in biology could use this code do genetic engineering homework

# Written by yhy, supported by qjz

import re #引入正则库

def reverse(dna):

pairs = {'A':'T','T':'A','C':'G','G':'C'}

return ''.join([pairs[x] for x in dna])[::-1]

def transcribe(dna):

pairs = {'A':'U','T':'A','C':'G','G':'C'}

return ''.join([pairs[x] for x in dna])[::-1]

def find\_start(rna):

index1 = -1

index1\_list=[]

index2 = -1

index2\_list=[]

index3 = -1

index3\_list=[]

coden1 = re.findall(r'.{3}',rna)

coden2 = re.findall(r'.{3}',rna[1::])

coden3 = re.findall(r'.{3}',rna[2::])

for i in coden1:

index1+=1

if i == "GUG" or i == "AUG":

index1\_list.append(index1)

for i in coden2:

index2+=1

if i == "GUG" or i == "AUG":

index2\_list.append(index2)

for i in coden3:

index3+=1

if i == "GUG" or i == "AUG":

index3\_list.append(index3)

coden = []

for i in index1\_list:

coden.append(coden1[i::])

for i in index2\_list:

coden.append(coden2[i::])

for i in index3\_list:

coden.append(coden3[i::])

return(coden)

def find\_end(rna):

index1 = -1

index1\_list=[]

index2 = -1

index2\_list=[]

index3 = -1

index3\_list=[]

coden1 = re.findall(r'.{3}',rna)

coden2 = re.findall(r'.{3}',rna[1::])

coden3 = re.findall(r'.{3}',rna[2::])

for i in coden1:

index1+=1

if i == "UGA" or i == "UAG" or i == "UAA":

index1\_list.append(index1)

for i in coden2:

index2+=1

if i == "UGA" or i == "UAG" or i == "UAA":

index2\_list.append(index2)

for i in coden3:

index3+=1

if i == "UGA" or i == "UAG" or i == "UAA":

index3\_list.append(index3)

coden = []

for i in index1\_list:

coden.append(coden1[:i+1:])

for i in index2\_list:

coden.append(coden2[:i+1:])

for i in index3\_list:

coden.append(coden3[:i+1:])

return(coden)

def CG\_content(conden\_list):

cg\_content = []

for i in conden\_list:

first =(i[1].count('G')+i[1].count('C'))/3

second = (i[2].count('G')+i[2].count('C'))/3

thrid = (i[3].count('G')+i[3].count('C'))/3

cg\_content.append([first\*100, second\*100, thrid\*100])

return cg\_content

def most\_likely(codens, var):

index = 0

min\_var =min(var)

index\_list = []

likely\_coden = []

for i in var:

if i == min\_var:

index\_list.append(index)

index+=1

for i in index\_list:

likely\_coden.append(codens[i])

return likely\_coden

def suit(cg):

variance = []

for i in cg:

variance.append(((int(i[0])-66)\*\*2+(int(i[1])-53)\*\*2+(int(i[2])-93)\*\*2)/3)

return variance

def rare\_coden(coden):

dic = {}

print(coden)

for i in coden:

string = ''

for j in i:

string += j

count\_n = int(i.count("UUA"))+int(i.count("CUA"))

count\_r = int(i.count("UUU"))+int(i.count("AUU"))+int(i.count("GUU"))+int(i.count("UCU"))+int(i.count("CCA"))+int(i.count("ACU"))

dic[string] = str("稀有密码子"+str(count\_r)+",禁止密码子"+str(count\_n))

return(dic)

code\_s = 'CGCTTGCGTGCGTCTCATCTGGAGGCTGCACCTGTCCGACCACTACGGCATCTGGCAGGAATTCGGCCAGAAGTTCCTCGTCGACCCCGACATCATCAAG'

code\_e = 'GGTGCGAGAGCGGATACTCCAGGACCTTGCGCACCTACCTGTCCTACGGCAGCAGCGACAACGCGAACCCAGTCTGCGAGACACGGCTGTGCAGGCCGTC' #输入序列

pair\_code\_s = reverse(code\_e)

pair\_code\_e = reverse(code\_s)

code\_s\_rna = transcribe(code\_e)

code\_e\_rna = transcribe(code\_s)

pair\_code\_s\_rna = transcribe(pair\_code\_e)

pair\_code\_e\_rna = transcribe(pair\_code\_s)

start\_findall = find\_start(code\_s\_rna)

pair\_start\_findall = find\_start(pair\_code\_s\_rna)

end\_findall = find\_end(code\_e\_rna)

pair\_end\_findall = find\_end(pair\_code\_e\_rna)

cg\_start = CG\_content(start\_findall)

pair\_cg\_start = CG\_content(pair\_start\_findall)

start\_suit = suit(cg\_start)

pair\_start\_suit = suit(pair\_cg\_start)

most\_likely\_s = most\_likely(start\_findall, start\_suit)

pair\_likely\_s = most\_likely(pair\_start\_findall, pair\_start\_suit)

rare = rare\_coden(most\_likely\_s)

pair\_rare = rare\_coden(pair\_likely\_s)

我保证这个代码是我自己写的，没有抄也没有照搬，是我的思路我的设计！(我在班群里放出了源代码供同学们参考！)