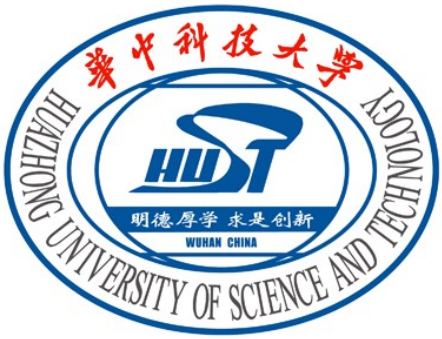


华中科技大学

课程实验报告

课程名称：随机过程

实验题目：马尔科夫模型在生物学中的两个应用

班　　级：**********

姓　　名：*******

学　　号：**********

指导老师：******

目录

摘要

马尔可夫链是由 Andrei A.Markov 于 1913 年提出，具有马尔可夫性质且时间与状态空间都是离散的随机过程，若时间连续则为马尔可夫过程。

马尔可夫链可被应用于蒙特卡罗方法中，形成马尔可夫链蒙特卡罗（Markov Chain Monte Carlo, MCMC），也被用于动力系统、化学反应、排队论、市场行为和信息检索的数学建模。此外作为结构最简单的马尔可夫模型（Markov model），一些机器学习算法，例如隐马尔可夫模型（Hidden Markov Model, HMM）、马尔可夫随机场（Markov Random Field, MRF）和马尔可夫决策过程（Markov decision process, MDP）以马尔可夫链为理论基础。

马尔可夫链的应用十分广泛，在生物信息学领域中，蛋白质空间结构和功能预测，DNA 序列分析都是十分重要的一项任务，DNA 以及蛋白质序列预测对此有重要的意义。 在微生物领域，我们也可以通过马尔可夫模型分析微生物之间的竞争生存关系。

本文主要工作便是将马尔可夫链运用在生物学中的微生物竞争模型和 DNA 序列分析上，以启发是否有将马尔可夫过程运用在更多学科方面的可能。

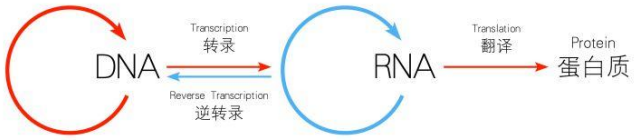
关键字：马尔科夫链，元胞自动机，隐马尔可夫模型，DNA 序列分析

一、实验背景及研究意义

碳循环即整个地球地球化学循环中碳交换的过程，是地球生命的重要组成部分。碳循环的一部分包括化合物的分解，使碳元素得以更新并以其他形式使用。该过程的这一部分的关键组成部分是植物材料和木质纤维的分解，而分解木质纤维的一些关键因素是真菌。

不同的真菌的菌丝延伸率和对木头的分解能力，形成孢子能力都不相同，同一块培养皿或是腐木上不同种类的真菌之间会相互竞争，于是它们最终的种群密度就会不同。即使是同一种真菌，不同温度适度下的菌丝延伸率和对木头的分解能力都会改变。如何确定一个在指定环境下对木头分解能力最高的真菌种类搭配，是一个对研究全球碳循环具有重要意义的课题。在本文我们研究不同真菌之间可能的竞争关系。

在各项生命活动中，不同蛋白质具有不同生物学功能，生命中的每一个过程都与生物蛋白密不可分，研究生物蛋白质对阐述生命现象的本质以及各种生命活动机理有着十分重要的意义。而由中心法则知，生物蛋白的构成和空间结构取决于生物的脱氧核苷酸即 DNA 序列，生物蛋白几乎由对应的 DNA 序列唯一的决定。因此研究生物的 DNA 序列是生命科学中十分重要的研究方向之一。



1 中心法则

当给定某生物的一串 DNA 序列时，DNA 序列上有些是会进行转录的编码区，有些则是非编码区，那么给定一段序列，能否预测是否包含基因，如果包含基因，以真核生物为例，基因结构包括启动子，外显子，内含子，剪切子，ESE，沉默子…….能够正确预测基因的结构将对研究生命过程具有重要的意义。

二、实验原理

2.1 马尔可夫链

马尔可夫链由 Andrei A.Markov 于 1913 年提出，马尔可夫链是状态与时间参数都是离散的 Markov 过程。其定义如下：

若随机过程$\{X_n, n \in T\}$对于任意的非负整数$n \in T$和任意的$i_0, i_1, \cdots, i_{n+1} \in I$，其条件概率满足

$$P\{X_{n+1} = i_{n+1} | X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n\} = P\{X_{n+1} | X_n = i_n\},$$

则称$\{X_n, n \in T\}$为马尔可夫过程，简称马氏链，称条件概率

$$p_{ij}(n) = P\{X_{n+1} = j | X_n = i\}$$

为马尔科夫链$\{X_n, n \in T\}$在时刻n的一步转移概率，简称为转移概率，其中$i, j \in T$.

所有的转移概率$p_{ij}(n)$可以构成一个概率转移矩阵$P(n)$，即：

$$P(n) = \begin{bmatrix} p_{11}(n) & \cdots & p_{1j}(n) & \cdots \\ \vdots & \ddots & \vdots & \cdots \\ p_{i1}(n) & \cdots & p_{ij}(n) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

且有$0 \le p_{ij}(n) \le 1, i, j \in I; \sum_{j \in I} p_{ij}(n) = 1, i \in I$.

称$\pi = (\pi_1, \pi_2, \cdots, \pi_n, \cdots)$为初始概率向量，其中$\pi_i = P\{X_0 = i\}, i \in I$，且同样有$0 \le \pi_i \le 1, \sum_n \pi_n = 1, i \in I$

2.2 隐马尔可夫模型

在马尔可夫模型中，每个状态都可以直接被观察到，我们可以直接确定模型在已知时刻的状态，但是在许多实际问题中，我们会遇到很复杂的情况。可能每种确定的状态，我们实际观察到的现象有多种。隐马尔可夫模型是马尔可夫链的一种，它的状态不能直接观察到，但能通过观测向量序列观察到，每个观测向量都是通过某些概率密度分布表现为各种状态，每一个观测向量是由一个具有相应概率密度分布的状态序列产生。我们也称隐马尔可夫模型为 HMM(Hidden Markov Model, HMM)。

HMM 在 20 世纪 60 年代末 70 年代初提出，是一种典型的统计方法，是一种用参数表示的、用于描述随机过程统计特征的概率模型。1970 年左右，Baum 等人建立起 HMM 的理论基础。Rabin 详细地对 HMM 做出了介绍，才使得各国的学者渐渐了解并熟悉该模型，进而开始研究。

HMM 包含一个双重随机过程，即 HMM 是由两个随机过程组成，一个是隐状态转移序列，它对应一个单纯的 Markov 过程，由初始状态概率向量π和状态转移矩阵A描述，输出的是隐状态序列；另一个是与隐状态有关的可观察序列，表示的是状态与可观察的输出符号的对应关系，由符号观察概率矩阵B描述，输出的是可观察的输出符号序列。在这里，隐状态序列是要由可观察的输出符号序列来进行推断的。其基本要素包括:

(1)N是 HMM 中的状态数，有的状态组成隐状态集，即状态空间表示为$S = \{S_1, S_2, \cdots, S_N\}$, 在时刻 t 的状态可用$q_t$来表示，其中$q_t \in S$;

(2)M 是每个状态所能产生的不同输出符号数，所有的输出符号数组成可观察的输出符号集:$V = \{V_1, V_2, \cdots, V_M\}$, 在 t 时刻观察的符号可用$o_t$来表示，其

中$o_t \in V$;

(3)状态转移概率矩阵$A = \{a_{ij}\}$,其元素a_{ij}是指 t 时刻状态为i时，t+1 时状态为j的概率，即：

$a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i], 1 \le i,j \le N$,其中$0 \le a_{ij} \le 1, \sum_{j=1}^{N} a_{ij} = 1$;

(4)符号观察矩阵$B = \{b_i(k)\}$,其元素$b_i(k)$是指 t 时刻状态为i的输出观察符号V_k的概率，即$b_i(k) = P[o_t = V_k | q_t = S_i], 1 \le i \le N, 1 \le k \le M$,其中$0 \le b_j(k) \le 1, \sum_{k=1}^{N} b_j(k) = 1$;

(5)初始状态概率向量$\pi = \{\pi_i\}$,其元素π_i是指初始时刻处于状态i的概率，即：$\pi_i = P\{q_1 = S_i\}, 1 \le i \le N$, 其中$0 \le \pi_i \le 1, \sum_{i=1}^{N} \pi_i = 1$。

故为了完整的描述一个 HMM 模型，需要模型参数$\lambda = (S, V, \pi, A, B)$,简写为模型参数$\lambda = (\pi, A, B)$。一旦确定了这些参数，一个 HMM 模型也就确定了。

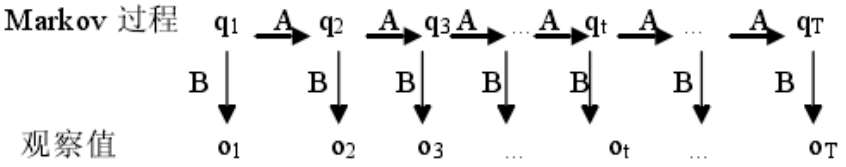


图 2 HMM 模型的示意图

对已知的 HMM 模型$\lambda = (\pi, A, B)$、状态转移序列$Q = (q_1, \cdots, q_T)$和观察序列$O = (o_1, \cdots, o_T)$，利用全概率公式可得：

$$P(O|\lambda) = \sum_{Q \in \Omega} P(O, Q|\lambda) = \sum_{Q \in \Omega} P(O|Q, \lambda) P(Q|\lambda)$$

又 $P(O|Q, \lambda) = \prod_{t=1}^{T} P(o_t|q_t, \lambda) = b_{q1}(o_1) b_{q2}(o_2) \cdots b_{qT}(o_{qT})$; $P(Q|\lambda) = \pi_{q1} a_{q1q2} \cdots a_{q_{T-1}qT}$

故

$$P(O|\lambda) = \sum_{Q \in \Omega} \pi_{q1} b_{q1}(o_1) a_{q1q2} b_{q2}(o_2) \cdots a_{q_{T-1}qT} b_{qT}(o_T)$$

式中，Ω是长度为T的所有可能存在的状态转移路径集合。

2.3 元胞自动机

元胞自动机（ Cellular Automata） 是 20 世纪 50 年代初由计算机之父冯·诺依曼（ J.von Neumann） 为了模拟生命系统所具有的自复制功能而提出来的一种动力模型。元胞自动机采用离散的空间布局和离散的时间间隔，将元胞分成有限种状态，元胞个体状态的演变仅与其当前状态以及其某个局部邻域的状态有关。

元胞自动机由以下基本元素所确定：

空间：元胞在空间中分布的空间格点，可以是一维、二维或多维。

状态集：$S = \{s_1.s_2, \cdots\}$ 每个元胞可能处于的状态。

邻居：存在与某一元胞周围，能影响该元胞在下一时刻的状态。

演化规则：任意设定， 最多 2^8=256 种不同的设定方式。若元胞以相邻的 8 个元胞为邻居，即 Moore 邻居；一个元胞的生死由其在该时刻本身的生死状态和周围八个邻居的状态。

我们知道元胞自动机中每个元胞的状态，仅有上一时刻他的邻居的状态所决

定。对于一个二维空间的元胞自动机，如果我们把整个矩阵的取值看成一个状态，则下一时刻$t+1$时矩阵的取值为i_{t+1}的概率仅有t时刻的矩阵的取值i_t所确定，即：

$$P\{X_{t+1}=i_{t+1}|\,X_0=i_0,X_1=i_1,\cdots,X_t=i_t\}$$
$$=P\{X_{t+1}\mid X_t=i_t\},$$

又元胞自动机是时间离散、状态离散的动力学模型，故元胞自动机为非齐次的马尔科夫链。但是计算元胞自动机的状态转移概率矩阵十分繁琐的一件事，故在实际演化过程中往往采用简单的演化规则。

三、元胞自动机模型

3.1 模型建立

假设在一块潮湿的木头上分布着两种真菌，给定温度和湿度的环境条件，建立元胞自动机模型来研究两种真菌在腐木上的生长情况和竞争过程。

我们假设木头是圆柱形的，且真菌只分布在木头表面。可将木头表面展开成矩阵。我们假设木头的表面积为$1000 * 1000\ cm^2$.将空间离散，每个元胞面积为$1\ cm^2$，元胞的邻居为上下左右是个元胞。

每个元胞的演化规则为：

1° 若元胞为空，且邻居中有真菌，则该元胞演化为邻居中的真菌。

2° 若元胞为真菌 A，邻居中全是真菌 A 或为空，则该元胞不变

3° 若元胞为真菌 A，邻居中存在真菌 B，以概率$\frac{s_A}{s_A+s_B}$保持不变，以概率$\frac{s_B}{s_A+s_B}$演化为真菌 B（其中s_A, s_B为真菌 A、B 分解木头的速率，也可以看成是获取养分的能力）

由 ***A trait-based understanding of wood decomposition by fungi***[6]提供的数据(完整的数据可在附录中下载)，在 10° 的环境温度下：

真菌 *Armillaria gallica OC1 A6E* 的菌丝延伸率为:$0.2\ mm * day^{-1}$，分解率为：2.29%（dry mass loss）

真菌 *Armillaria gallica SH1 A4A* 的菌丝延伸率为:$0.06\ mm * day^{-1}$，分解率为：2.18%（dry mass loss）

时间设置为 122 天，利用元胞自动机模型研究两种真菌在腐木上的生长情况和竞争过程。

3.2 模型模拟

编写如下 Matlab 程序：

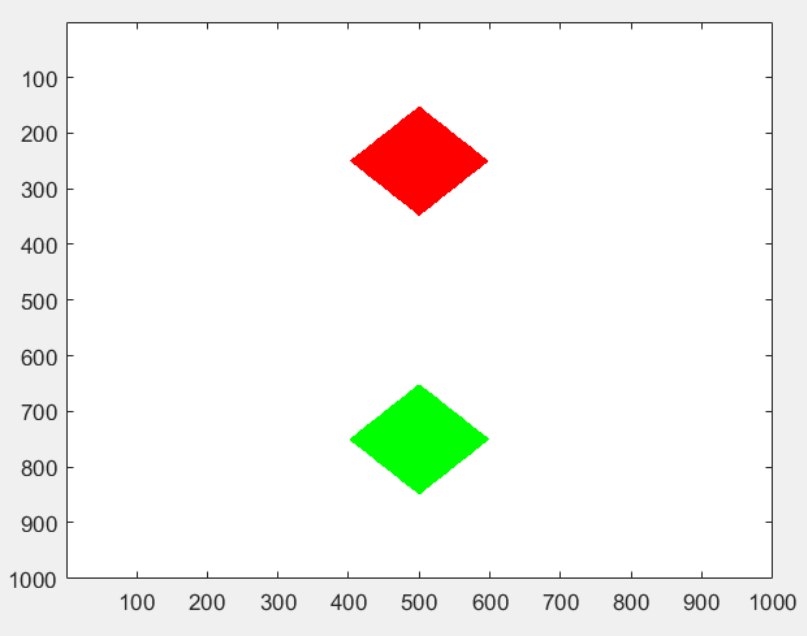
```
% simulate fungi competetion model with cellular automata
%e1 = input('Please input fungi A''s Extension rate:');
%e2 = input('Please input fungi B''s Extension rate: ');
%d1 = input('Please input fungi A''s Decomposition rate:');
%d2 = input('Please input fungi B''s Decomposition rate: ');
load('fungi.mat');
n=1000;                                                    %size of wood is n*n;
f1 = 7; f2 = 8;     %Choose the number of the two fungi
UL = [n 1:n-1]; DR = [2:n 1];                              % Up and Left neighbor; Down and
Right neighbor.
veg=zeros(n,n); E=0;A=1;B=2;                               % veg: empty=0 fungi A=1 fungi
B=2
e1 = Extension(1,f1);e2 = Extension(1,f2) ;                % Extension rate
t1= e1*100; t2=e2*100; t1=int32(t1); t2=int32(t2);
a = gcd( t1 , t2); b=t1/a;c=t2/a;                          % Discretization time
d1 = Decompostion(1,f1); d2 = Decompostion(1,f2);          %Decompostion rate
s1 = d2/d1;   s2 = d1/d2;                                  % The assumed competition
```

```
coefficient
D = ones(n);
imh = image(cat(3,D,D,D));                                  % color of empty is white
veg(250,500)=1; veg(750,500)=2;                             % Fungal colonization.
j=1;d=max(b,c); z=0;Z=0;                                    % some   state variable.
if b==d
      z=1;
end
day=0; k=0;f=k; area_a=zeros(122,1);area_b=zeros(122,1);
while(day<122)
    j=j+1;
     isE = (veg==E); isA = (veg==A); isB = (veg==B);
     if j==b
          Z=1;
          if z
               j=1;k=k+1;
          end
     end
     if j ==c
          Z=2;
          if ~z
               j=1;k=k+1;
          end
     end
     switch Z
          case 1
               sums =              (veg(UL,:)==A) + ...
                    (veg(:,UL)==A)    +     (veg(:,DR)==A) + ...   %Find out if there's any A
fungus in neighborhood.
                                        (veg(DR,:)==A);
               veg = veg + ((isE) & sums>0)-( (isB) & rand(n,n) > s1/(s1+s2)*ones(n) &
sums>0 );
                                                                                    %u
pdate veg matrix
          case 2
               sums =              (veg(UL,:)==B) + ...
                    (veg(:,UL)==B)    +     (veg(:,DR)==B) + ...   %Find out if there's any B
fungus in neighborhood.
                                        (veg(DR,:)==B);                  %update veg
matrix
               veg = veg + 2*((isE) & sums>0) + ( (isA) & rand(n,n) > s2/(s1+s2)*ones(n) &
sums>0 );
          otherwise
     end
```

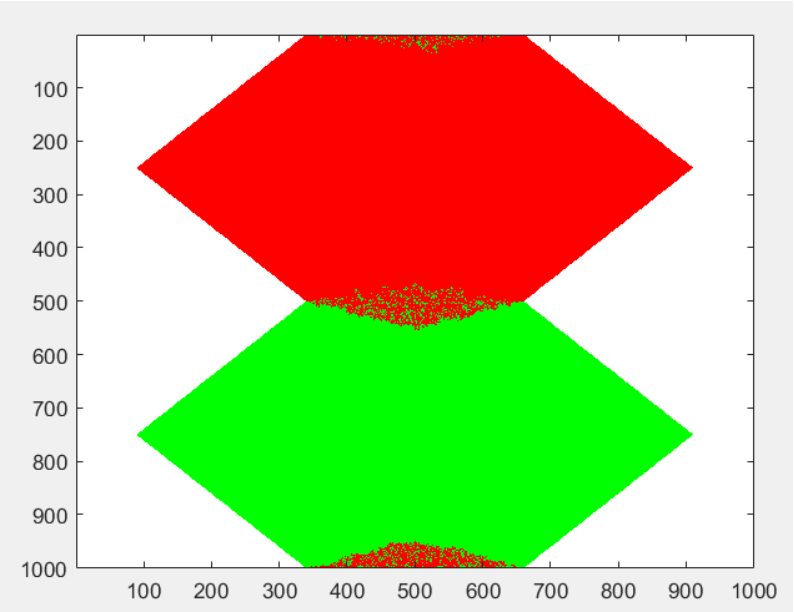
```
    Z=0;
    set(imh, 'cdata', cat(3,isE+isA,isE+isB,isE ))
    drawnow
    if k == d*a % a new day
        day=day+1;k=0;
        disp(['day:',num2str(day)]);
        %caculate area proportion (%)
        aa = sum(isA,'all'); ab = sum(isB,'all');
        area_a(day)=aa/10e3;area_b(day)=ab/10e3;
    end
end
figure(2);
plot(1:day,area_a,1:day,area_b);
title('The area proportion of Fungi');
legend(cell2mat(name(f1)),cell2mat(name(f2)));
```

运行该程序可得到两种真菌分布图（其中红色是 *Armillaria gallica 0C1 A6E*，绿色是 *Armillaria gallica SH1 A4A*）

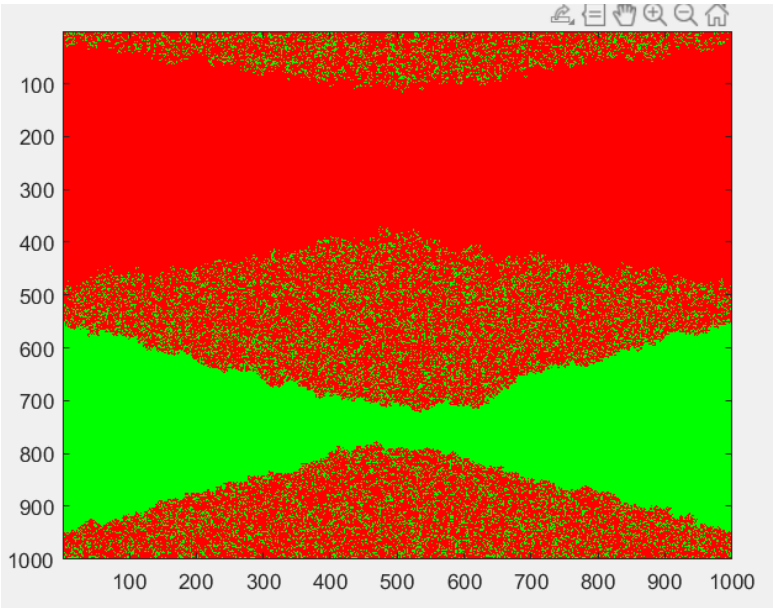
第五天时真菌的分布：



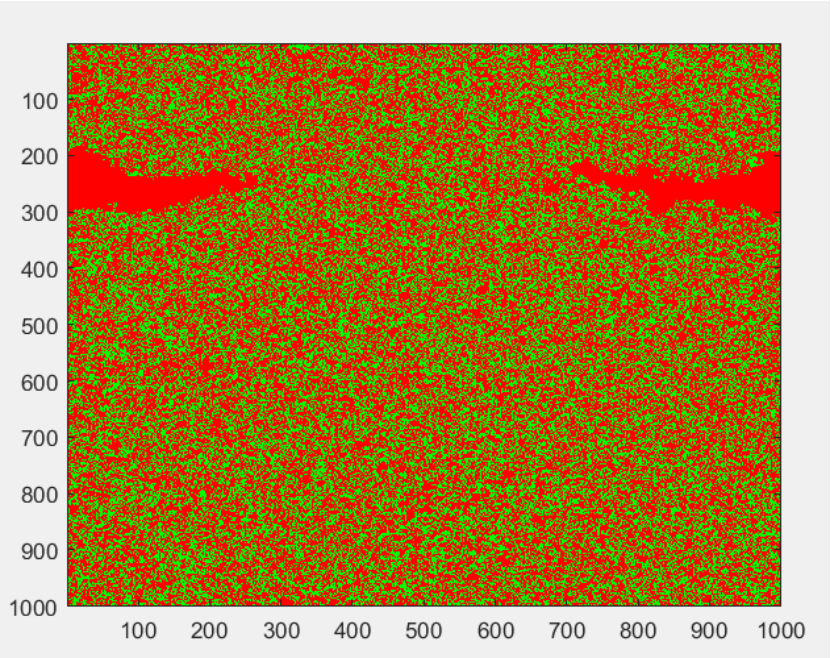
第 20 天时真菌的分布：



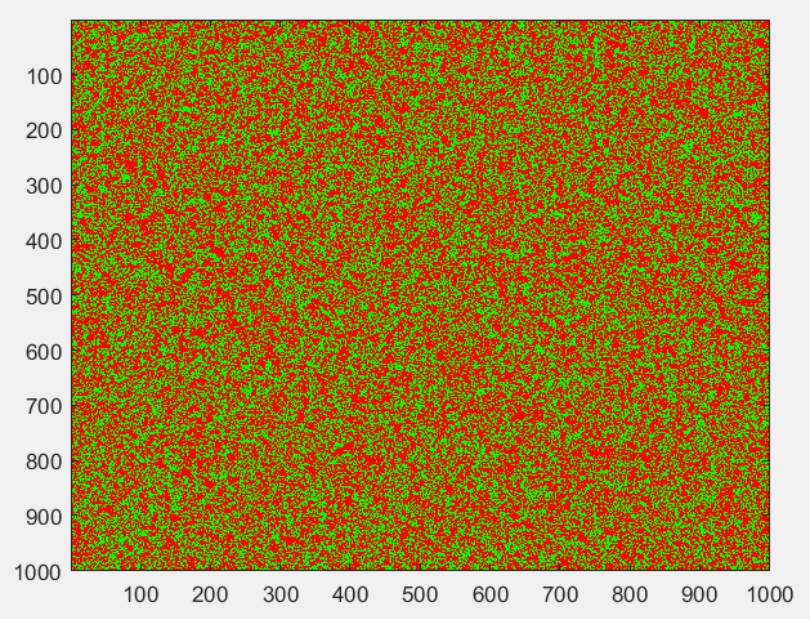
第 45 天时真菌的分布：



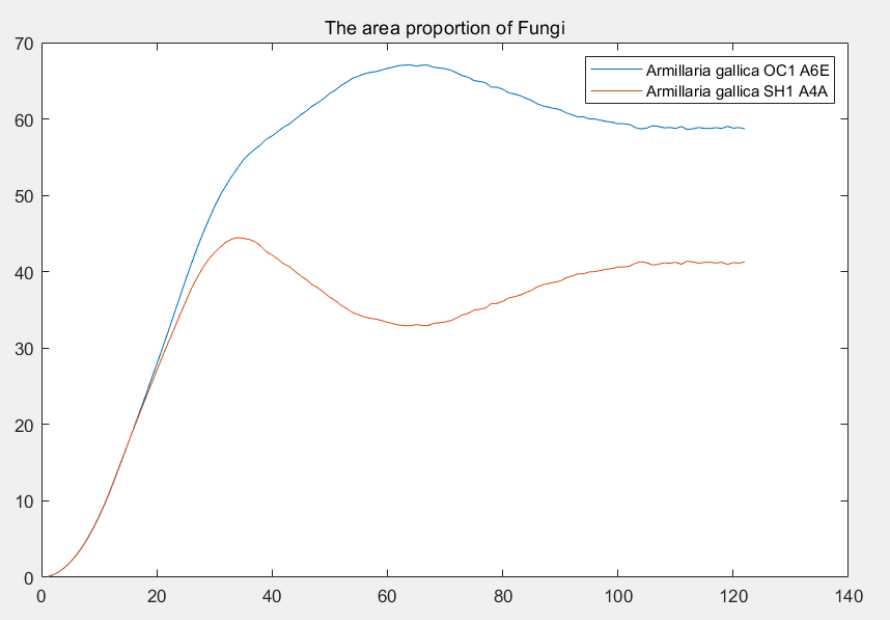
第 90 天时真菌的分布：



第 122 天时真菌的分布：



两种真菌随着时间变化的面积比例：



由此我们可以得到一个简单的真菌之间的生长情况和竞争模型。

我们可以发现真菌 OC1 A6E 和 SH1 A4A 在 10° 的环境温度下是共生的，虽然 OC1 A6E 的数量更多，但并不会导致 SH1 A4A 灭绝。对其他真菌种类进行模拟，有出现只剩下一种真菌，另一种真菌完全灭绝的情况。

但遗憾的是，由于没有实验室和培养真菌的条件，暂时无法评估该模型的精确程度如何。实际上，真菌之间除了会争夺养分，还是分泌一些化学物质可能干扰其他类型真菌的生长或是还有其他相互作用是本模型没有考虑到的。

四、基于隐马尔可夫模型的 DNA 序列分析模型

当给定某生物的一串 DNA 序列时，DNA 序列上有些是会进行转录的编码区，有些则是非编码区，那么给定一段序列，能否预测是否包含基因，如果包含基因，以真核生物为例，基因结构包括启动子，外显子，内含子，剪切子，ESE，沉默子…….能够正确预测基因的结构对生命过程的研究具有十分重要的意义。

如果直接假设 DNA 序列服从 K-oder 马尔科夫链，随着序列的增长，需要求的参数也将飞速增长，此外，如果观测值X的状态非常多，转移概率矩阵$P\{X_n|X_{n-1}\}$也会变成一个非常巨大的矩阵，占用十分多的计算机存储空间，并计算效率低，例如X有 K 个状态，那么转移概率矩阵P的大小为$K*K$，若对于连续变量进行观测，那么将更加困难。

为降低马尔可夫链中的参数数量，我们引入一个包含较少状态的隐状态，并将 DNA 序列的马尔科夫链模型转化为隐马尔可夫模型。

4.1 建立模型

我们给出一个指定的 DNA 序列，为模拟现实情况下基因测序可能出现的干扰或误差，或是生物基因表达中的干扰。我们在 DNA 序列中添加加性高斯白噪声来模拟这种干扰，然后构建 HMM 模型，对添噪后的 DNA 序列进行还原。

HMM 模型的参数包括初始概率向量π,转移概率矩阵A,符号观察矩阵B。在许多实际问题中我们能得到的信息可能只有可观察的输出符号序列$O=(o_1,o_2,\cdots,o_T)$。如何估计 HMM 的参数$\lambda=(\pi,A,B)$，使得概率$P(O|\lambda)$的值最大，是 HMM 模型训练的主要问题之一。

同时训练问题也分为 Supervised learning 和 Unsupervised learning，即监督学习和无监督学习。通常运用的方法包括 EM(expectation maximization algorithm)算法，该方法是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐性变量；粒子群优化算法(PSO：Particle swarm optimization) ，一种启发式算法，源于对鸟群捕食的行为研究，粒子群优化算法的基本思想：是通过群体中个体之间的协作和信息共享来寻找最优解。

将 EM 算法应用于 HMM 模型的算法就是 Baum-Welch 算法，也是主流的用于求 HMM 模型参数的算法。

Baum-Welch 算法和 PSO 算法的实现中对大规模样本的训练中存在收敛速度过慢、陷入局部最优和过早收敛等等问题，且实现起来相对比较复杂，且主要是运用数理统计和机器学习等相关知识，随机过程知识在其中使用较少，迫于时间紧张，作者在这里便不对其进行实现，而是直接假设出一种最简单情况下的 HMM 模型参数。

在基因组/宏基因组测序中，受测序技术读长的限制，需要将样品总 DNA 使用鸟枪法打断成小片段进行测序，拿到测序数据后再利用这些短序列来构建基因组图谱/宏基因组序列。因生物学需要，我们引入 k-mer 的概念：

k-mer：是指将一条序列分成包含 k 个碱基的子字符串，如果 reads 长度为 L，k-mer 长度设为 k，则产生的 k-mers 数目为：L-k+1，例如序列 AACTGACT，设置 k 为 3，则可以将其分割为 AAC ACT CTG TGA GAC ACT 共 6 个 k-mers。其中 k 一定是奇数，如果是偶数遇到回文序列可能会产生完全相同的 k-mers。

在本模型中我们设定 k-mer 为 3，即三个碱基组成片段为一个字符串。
进行如下假设：

（1） 每种碱基之间的转移概率均为一样，即状态转移矩阵为

$$A=\begin{bmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{bmatrix}$$

（2） 初始概率向量为:

$$\pi=\left(\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}\right).$$

（3） 观测状态转移矩阵：

$$\left(b_{q_t o_t}\right)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(o_t-q_t)^2}{\sigma^2}}$$

再令$\sigma^2=1$(最简单的情况，也可使$\sigma^2=var(O)$)

在已经 HMM $\lambda=(\pi,A,B)$和观测符号序列$O=(o_1,o_2,\cdots,o_T)$的情况下，求解一个观测序列的最佳隐状态序列$Q^*=(q_1^*,\cdots q_T^*)$。理论上可以通过枚举所有状态转换序列，并对每一个状态转换序列 Q 计算$P(O,Q|\lambda)$,其中使得$P(O,Q|\lambda)$取值最大的转换序列Q^*就是我们所需。即：

$$Q^*=arg\max_{Q\in\Omega}P(O,Q|\lambda)$$

但是在该方法计算量随着时间和样本数增加而较大的增长，因此简单的枚举不是有效的方法。

我们一般会使用 Viterbi 动态规划算法求解该解码问题，定义 Viterbi 变量$\delta_t(i)$为:

$$\delta_t(i)=\max_{q_1q_2\cdots q_{t-1}}P(q_1q_2\cdots q_t,q_t=i,o_1o_2\cdots o_t|\lambda)$$

$\delta_t(i)$是时刻 t 时沿一条路径$q_1q_2\cdots q_t$,且$q_t=i$，产生可观察的输出符号序列$o_1o_2\cdots o_t$的最大概率。由递归可得：

$$\delta_t(i)=\max_{1\le j\le N}\left[\delta_{i-1}(j)a_{ji}\right]b_i(o_t)$$

定义$\varphi_t(i)$来记录时刻 t 为i状态是最可能由时刻 t-1 的哪个状态转移而。
找到路径后再进行路径回溯，便可得到最佳隐状态序列Q^*。

Viterbi 算法的具体步骤如下：

(1)初始化

$$\delta_t(i)=\pi_i b_i(o_1),1\le i\le N$$
$$\varphi_1(i)=0,1\le i\le N$$

(2)递归计算

$$\delta_{t+1}(j) = \max_{1\le i\le N}[\delta_t(i)a_{ij}]b_j(o_{t+1}), 1 \le t \le T-1, 1 \le j \le N$$

$$\varphi_{i+1}(j) = arg\max_{1\le i\le N}[\delta_t(i)a_{ij}], 1 \le t \le T-1, 1 \le j \le N$$

(3)终止递归

$$P^* = \max_{1\le i\le N}[\delta_T(i)]$$

$$q_T^* = arg\max_{1\le i\le N}[\delta_T(i)]$$

(4)路径回溯

$$q_t^* = \varphi_{t+1}(q_{t+1}^*), t = T-1, T-2, \cdots 1$$

4.2 模型模拟

基于以上模型，给定一条 100-base DNA sequence 为：
“CATCCCTCACCTGAAGTGTCCAGCAAATACACCAAGGGTGACGCAGGACAAGCATGAGCCATTCATA
CTGCTGCAACCAGAGAGAGGGAGCAGGAAAATA”
编写的 Matlab 程序如下：

```
SNR = 30;    % SIGNAL NOISE RATIO（信噪比）

% 给定 DNA 序列（存储在文件中）
fo = fopen('dnaSequenceSample.txt', 'r');
dnaSeq = fgets(fo);
fclose(fo);

K = 3;  % K-mer 为 3

% 将 DNA 序列转化为数字信号
digitSignal = dnaSeq2Digit(dnaSeq);

% 在数字信号中添加高斯噪声
digitSignalNoise = noiseAdd(digitSignal, SNR);
% 可在 noiseAdd 函数中设置随机数生成器状态（每次噪声一样还是不同）
% ========================================================================

% 在已知 HMM 模型的参数下，可以用 Viterbi 算法求解观察序列的
% 最佳隐状态序列
dSRecovery = viterbiDecoder(digitSignalNoise);

% 将数字信号恢复成 DNA 序列
dnaSeqRecovery = digit2dnaSeq(dSRecovery);

% 将原 DNA 序列和 HMM 模型预测的序列进行对比
fprintf('原 DNA 序列为：\n');
disp(dnaSeq);
fprintf('基于 HMM 模型预测的 DNA 序列为：\n');
disp(dnaSeqRecovery);
err=sum(dnaSeq ~= dnaSeqRecovery);
```

```
fprintf('预测结果中碱基错误数量为：%d ，错误率为：%4.2f%%\n',err,err/length(dnaSeq)*100);
```

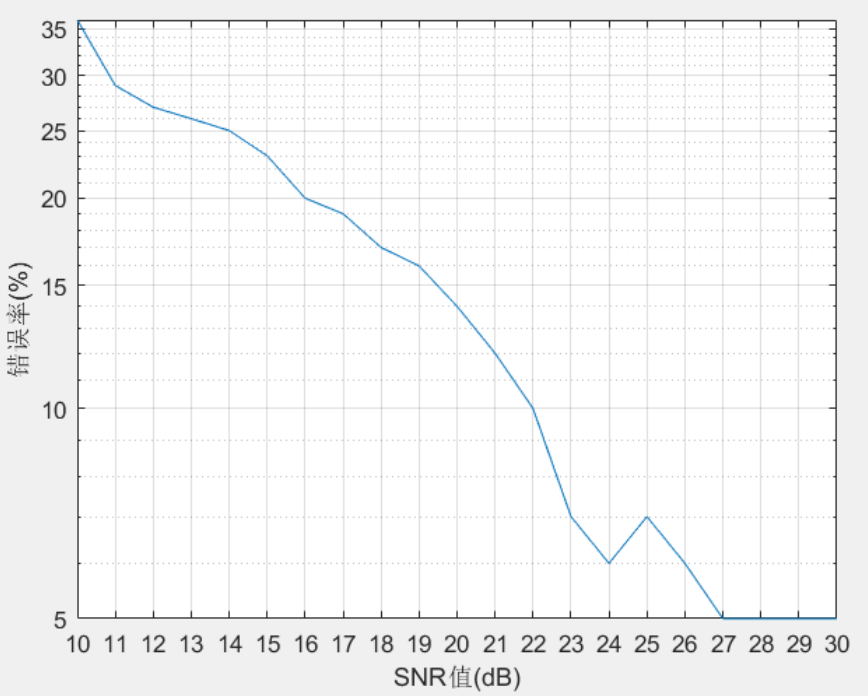
（Main 文件中所用的函数文件放在附录中）

运行结果为:

```
原 DNA 序列为:
CATCCCTCACCTGAAGTGTCCAGCAAATACACCAAGGGTGACGCAGGACAAGCATGAGCC
ATTCATACTGCTGCAACCAGAGAGAGGGAGCAGGAAAATA
基于 HMM 模型预测的 DNA 序列为:
AATCCCTCACCTGAAGTGTCCAGCAAATACACAAAGGGTGACGCAGGACAAGCATGAGCC
ATTCATACTGCTGCAACCAGAGAGAGGGAGCAGCAAAATG
预测结果中碱基错误数量为: 4 ,错误率为: 4.00%
```

将信噪比设置为 10：1：30

我们可以获得如下基于 HMM 模型的 DNA 序列预测错误率图像



4.3 结果分析

可以看出基于 HMM 模型对 DNA 序列分析的效果还是不错的，经多次实验，信噪比为 30 时，本模型的错误率在 5%左右浮动。如果运用 Baum-Welch 算法或 PSO 算法寻找最佳的 HMM 模型参数$\lambda = (\pi, A, B)$，模型的精确程度会有进一步的提高。

五、实验心得及体会

附录

1. 参考文献

[1] 刘次华，《随机过程第五版》.武汉：华中科技大学出版社
[2]AR Rao. Hidden Markov Models for biological sequences.
[3]石鸥燕，杨惠云，杨晶,等. 应用优化的隐马尔可夫模型预测蛋白质二级结构[J]. 高技术通讯，2008，018(007):738-742.
[4]周志华，机器学习，清华大学出版社，2016
[5]白雁飞. 基于隐马尔可夫模型的 J 波识别技术研究[D]. 太原理工大学，2016.
[6] Lustenhouwer N， Maynard D S， Bradford M A，et al. A trait-based understanding of wood decomposition by fungi. 2020.

同时参考了华中科技大学生命科学与技术学院薛宇教授的生物信息学(Bioinformatics)课件(http://xue.biocuckoo.org/course.html)

2. 程序代码

noiseAdd 函数：

```
function y = noiseAdd(x, snr)
% rng('default');%将随机数生成器设为默认，这样每次的噪声将相同。可以注释本行
L = length(x);
SNR = 10^(snr/10);% 将 SNR 从 dB 恢复为线性
Esym = sum(abs(x).^2)/(L);% 能量的谱密度（Energy Spectrum Density，ESD）
N0 = Esym/SNR;% 噪声能量的谱密度

noiseSigma = sqrt(N0);% 标准加性高斯白噪声(Additive White Gaussian Noise)
n = noiseSigma * randn(1, L);
y = x + n;% 实际信号（添噪后）
```

dnaSeq2Digit 函数：

```
function digitSignal = dnaSeq2Digit(dnaSeq)
%   在 K-mers 为 3 时，将 DNA 序列转化为数字信号
% Initialize
n = length(dnaSeq);
digitDNASeq = zeros(1, n);
% 设 A=0, C=1, G=2, T=2
% 对应的二进制为 A=00; C=01; G=10; T=11
for i = 1 : n
    if dnaSeq(i) == 'A'
        digitDNASeq(i) = 0;
    elseif dnaSeq(i) == 'C'
        digitDNASeq(i) = 1;
    elseif dnaSeq(i) == 'G'
        digitDNASeq(i) = 2;
    else
        digitDNASeq(i) = 3;
    end
end
digitSignal = zeros(1, n-2);
% 将三个碱基的信息存在一个八位 int 整数中
% 这个数的 bit 的每两位代表一个碱基
% bitsll：将该数 bit 位向左移动 n 步 (Bit shift left logical)
for i = 3 : n
    digitSignal(i-2) = bitsll(digitDNASeq(i-2), 4)+...
        bitsll(digitDNASeq(i-1), 2) + digitDNASeq(i);
end
```

digit2dnaSeq 函数：

```
function dnaSeq = digit2dnaSeq(digitSignal)
% 将数字信号恢复为 DNA 序列
% Initialize
n = length(digitSignal);
dnaSeq = '';
for i = 1 : n
    nib2 = bitget(digitSignal(i), 6) * 2 + bitget(digitSignal(i), 5);

    if nib2 == 0
        dnaSeq(i) = 'A';
    elseif nib2 == 1
        dnaSeq(i) = 'C';
    elseif nib2 == 2
        dnaSeq(i) = 'G';
    else
        dnaSeq(i) = 'T';
    end
end
% Decode 2 final nibbles
nib1 = bitget(digitSignal(i), 4) * 2 + bitget(digitSignal(i), 3);
nib0 = bitget(digitSignal(i), 2) * 2 + bitget(digitSignal(i), 1);
if nib1 == 0
    dnaSeq(n+1) = 'A';
elseif nib1 == 1
    dnaSeq(n+1) = 'C';
elseif nib1 == 2
    dnaSeq(n+1) = 'G';
else
    dnaSeq(n+1) = 'T';
end
if nib0 == 0
    dnaSeq(n+2) = 'A';
elseif nib0 == 1
    dnaSeq(n+2) = 'C';
elseif nib0 == 2
    dnaSeq(n+2) = 'G';
else
    dnaSeq(n+2) = 'T';
end
```

viterbiDecoder 函数：

```
function dSRecovery = viterbiDecoder(dSN)
%
dnaSeqLen = length(dSN);
% 数字信号中所有可能的状态:1 to 64
maxTri = 64;
tri = 1 : 1 : maxTri;
% 转移概率矩阵 A
A = 0.25*ones(4,4);
% Emission probability B
%sigma = var(dSN);
sigma = 1; %最简单的情况
b = 1./( sqrt(2*pi*sigma) * exp(((dSN(1) - tri).^2))/sigma);
% Previous forward path probability from the previous time step
a = ones(1,dnaSeqLen);
x = b;
% Initialize
dSRecovery = zeros(1, dnaSeqLen);

for k = 2 : dnaSeqLen
    % 跑遍每种情况
    b = 1./( sqrt(2*pi*sigma) * exp(((dSN(k) - tri).^2))/sigma);
    for i = 1 : maxTri
        % 寻找四种向前路径的概率
        for j = 0 : 3
            tmp = round((j * 16) + (i - 1)/4)+1; %1 17 33 49
            x(i) = max(x(i), a(tmp));
        end
        % 取其中最大的概率
        x(i) = x(i) * 0.25 * b(i); % 转移概率矩阵 A 的每个值均为 1/4
    end
    [~, dSRecovery(k)] = max(x);
end
```