UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Septiembre de 2019

Proyecto 1

Recombinacion de ADN con el método de Monte Carlo via Cadenas de Markov

---

Hecho por:
Quintana Silva, Brenda Paola

Contents

1 Motivación y Antecendentes

Desde los 70's se han desarrollado metodologías que permiten las modificaciones genéticas. Estas son las llamadas técnicas recombinantes de ADN, las cuales tienen el objetivo de modificar el genoma; consiten en cortar, unir y reintroducir ciertos fragmentos de ADN en forma de cromosomas. A continuación se mencionarán algunas vetajas de modificación génetica a diferentes seres vivos y así resaltar la importancia de conocer la estructura del genoma.

1.1 Animales

La técnica de modificación genéticas más utilizada en animales es la microinyección de embriones. Se inyecta el nuevo ADN que incorpora un gen en el núcleo de las células de un embrión. Estas modificaciones resultan de gran utiles, por ejemplo, la modificación genéticas de ratones ayuda para el estudio de enfermedades humanas y sus tratamientos.

También se ha utilizado la modificación genética de animales para poder ser utilizados para trasplantes de órganos. Por último, otra utilidad es para que ciertas especies produzca una sustancia. Por ejemplo, la cabra ha sido modificada para que su leche sea terapéutica.

Figure 1: En Japón, en 2007, crearon unas ranas translúcidas con el objeto de poder estudiar el efecto de químicos en sus órganos.

1.2 Bacterias y Levaduras

Las modificaciones genéticas a bacterias y levaduras se llevan a cabo incorporando los genes que se desea introducir en forma de pequeños cromosomas. Estos métodos permiten una ampliación del gen y su información; así se puede hacer la síntesis del gen que está codificado en el ADN para producir proteínas como la insulina o algunas hormonas.

Una aplicación destacada a este tipo de técnicas es la llamada Biología Sintética; esta consiste en aprovechar la información que se tiene de genes. En un principio se podrían introducir combinaciones

de estos genes en bacterias, y así dar origen a nuevas bacterias que sirvan para producir nuevas sustancias como combustibles o fármacos que sirvan para eliminar contaminantes. Sin embargo también podrían crearse o modificarse enfermedades patógenas y usarlas con fines militares.

1.3 Humanos

Esta es la modificación más discutida; alterar el material genético de un embrión humano está prohibida pues la incorporación de nuevos genes en el genoma humano es muy impredecible en la actualidad. Se permite modificaciones en plantas y animales porque se pueden obtener más individuos de una forma 'fácil', el cual no es el caso de embrión humano. Aunque con un poco de visión futurista estos métodos pueden ayudar a mejora del genoma humano, corrigiendo enfermedades genéticas desde antes de nacer y eliminando fenotipos débiles.

Un caso distinto es el modificar el tejido de un paciente que tiene una enfermedad debido a un gen defectuoso. La terapia genética consiste en extraer tejido donde se da la enfermedad y corregir el defecto genético que provoca, el tratamiento es extraer células precursoras de la médula ósea, corregir el defecto genético, volverlas a introducir y eliminar las células no corregidas. Algunas de la enfermedades en las que se utiliza esta técnica es en el tratamiento del cáncer, el SIDA, fibrosis quística, la enfermedad de Crohn, enfermedades oculares, etc.

Figure 2: Corea de Huntington, se caracteriza por la degeneración de neuronas y células del sistema nervioso central.

1.4 Plantas

Permiten incorporar caracteres de interés pero que no se encuentran en las poblaciones existentes de plantas, como el tamaño, color, oxidación, las semillas, etc.

Figure 3: Manzana modificada para que no se oxide.

Resulta gran ventaja hacer modificaciones genéticas y para llevarlas acabo es necesario disponer de la secuencia del gen a insertar o modificar y su ubicación; por está razón es importante hacer una aproximaciones genéticas (los posibles lugares de los genes) ayudaría a facilitar las modificaciones genética.

2 Propuesta

2.1 Misión

Con la intención de hacer una mejora a una primer distribución aproximada de las regiones de recombinación, en este proyecto se propone una forma de aproximación sin tener que hacer la estimación a través de métodos experimentales que consisten en la falla y error, los cuales que pueden resultar costosos y poco eficientes. Esta primera distribución puede ser mejorada con métodos experimentales.

2.2 Estrategia

El procedimiento consiste en un modelo matemático probabilístico y estadístico, el cual estima la distribución de los nucleótidos dado un cierto gen. El esquema que se utilizará como base es el llamado Modelos Ocultos de Markov, el cual se encarga modelar la secuencia de la cadena ADN dado los

nucleótidos anteriores y posteriores y que también toma en cuenta el gen en que se está.

Se usará un algoritmo tipo Monte Carlo vía Cadenas de Markov (MCMC), el que se encargará de maximizar la probabilidad de una cadena de ADN dada una cierta distribución de probabilidad, esta distribución depende del gen en que se encuentre, y así obtener la distribución de genes más verosímil posible.

La idea central es mejorar una primera estimación estadística con el algoritmo MCMC y tras un serie de pasos poder encontrar el argumento máximo de la función de verosimilitud el cual corresponde a la distribuciones de nucleótidos de cada gen y así poder obtener una distribución que caracteriza a cada gen.

Las ventajas de usar este método es que no resulta costoso el hacer una estimación, solo se necesita una base de datos y una computadora. El programa que se incluye en el ápendice se encarga de distinguir la distribución de nucleótidos de un cierto tipo de gen, y así obtener una primera aproximación.

2.3 Objetivos

- Mejorar una aproximaciones genéticas para economizar recursos.
- Identificar los fragmentos de ADN que corresponde a un cierto tipo de gen.
- Poder construir un mapa genético que identifique en qué parte de los cromosomas se encuentra un gen. En un principio este resultado dependerá del tipo de especie del que provienen el genoma que se este analizando.
- Poder identificar anomalías genéticas.

2.3.1 Mercado

Este algoritmo ya implementado puede ser vendido a farmacéuticas; haciendo las modificaciones necesarias para trabajar con el genoma de bacterias. El algoritmo puede identificar genes particular y con la tecnología adecuada hacer una recombinación de ADN conveniente la cual puede llevar a crear ciertos tipos de sustancias útiles para fabricar medicamentos o anestésicos.

3 Problema

El ADN es una secuencia de nucleótidos muy larga; la cual tiene contine subsecuencias específicas llamadas exones e intrones. Un exón codifica una porción información específica de la proteína (al unir cierto conjunto de exones se tiene la codificación en ADN de una proteína en particular). El conjunto de exones forma la región codificante del gen. Los exones están separados por intrones que son regiones largas de ADN que no se codifican.

Sin pérdida de generalidad podemos decir que el ADN tiene la siguiente forma: $... - intron - exon - intron - exon - ....$ El número de intrones en un genoma varía de acuerdo a la especie. Por ejemplo, en el genoma del pez globo el número de intrones es bajo, pero por otro lado, en el genoma de los mamíferos abundan intrones.

3.1 Clasificación

El proyecto tiene como objetivo encontrar la regiones que codifican los genes. Con la idea anterior se clasificará los fragmentos en regiones codificantes y regiones no codificantes, los cuales corresponden a: exones e intrones respectivamente. Posteriormente los exones se separan en tipos de genes.

El problema surge al querer hacer la clasificación pues se necesita cierto criterios científicos que sean válidos para hacer una partición del genoma. Los criterios que se tomarán son los siguientes:

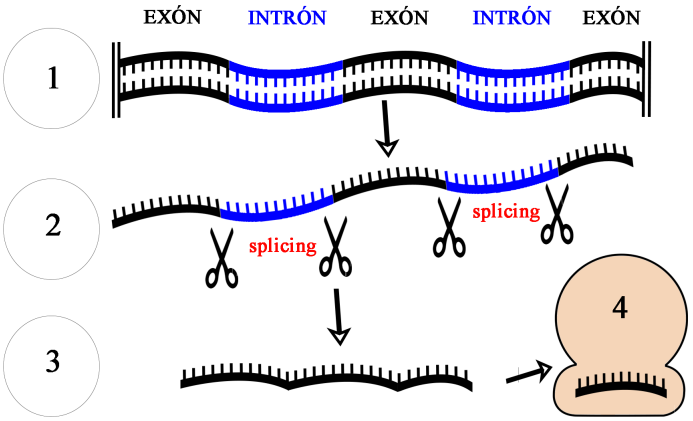

Figure 4: Subsecuencia de ADN: Intrones y exones.

3.1.1 Stops y Start

Ciertas proteínas se encargan de leer las secuencias de ADN de las regiones codificantes para formar proteínas. Esta es leída de 3 en 3 pues estos trios codifican aminoácidos (en el caso de los humanos existen 20 aminoácidos que son usados para hacer proteínas) y son llamados codones. El orden y tipo de nucleótidos dan origen a una gran variedad de proteínas. En particular hay 2 aminoácidos importantes los cuales son stop y start, el tipo start tiene la función de indicar dónde inician las regiones codificantes y solo hay uno, el cual es la secuencia TAC y los tipo stop terminan las regiones de codificación, de este tipo hay 3: UAG, UGA y UAA .

3.1.2 islas CpG

Los CpG son nucleótidos los cuales están compuestos de Citosina-fosfato-Guanina, esta secuencia tomara una gran importancia, pues en el genoma humano este dúo tiende a ser modificando por metilación.

Las islas CpG son regiones de ADN donde se encuentra esta secuencia más de la esperada. Son de gran importancias pues entre el 50% y el 70% de los genes tienen una isla CpG asociada a sus promotores. Un promotor es una región de ADN que controla la iniciación de una determinada porción de ADN o ARN 'promueve la transcripción de un gen'.

El modelo con el que se trabajará asumiendo que las regiones de codificación tienes una diferente distribución de las no codificantes, en las regiones no recombinante pues podría encontrarse cadenas de start sin un stop y puede que no se encuentren islas CpG.

4 Modelo

El modelo que se utilizará es un modelo estadístico llamado modelos ocultos de Markov en cual se asume que sistema es un sistema de Markov con parámetros desconocidos. Este modelo es muy útil para reconocer patrones y por está razón se va utilizar, teniendo como objetivo reconcecer la distribución de cada gen.

4.1 Modelos Ocultos de Markov (HMM)

Un Modelo Oculto de Markov (Hidden Markov Model) es un doble proceso estocástico, el primer proceso es una cadena de Markov que contiene una sucesión de estados ocultos; en nuestro caso corresponderá a si la variable aleatoria se encuentra en estado intrón o exón (y es así a qué gen corresponde). El número de estados ocultos puede ser variable pues en general este número no se conoce; en nuestra simulación no es la excepción, se desconoce la cantidad de genes que se encuentren codificadas en una porción de ADN dada .

El segundo proceso consiste en lo siguiente:
cada estado genera una secuencia observable que se ajusta una distribución multinomial. Las frecuencias observadas en cada estado oculto son independie ntes entre sí.

Los parámetros que definen un HMM:

- El número de estados ocultos r, $r \geq 2$.
- El número de estados que aparecen en la secuencia observable M.
- Las probabilidades de transición de la cadena oculta. Es una matriz r r donde se recoge la probabilidad de que la cadena de Markov cambie de uno de sus estados ocultos a otro (o de que no cambie).

- Las probabilidades de transición de la cadena observable: es una matriz r M donde se recoge la frecuencia con la que se observa cada estado en cada uno de los estados ocultos. Cada estado oculto es capaz de generar los mismos estados observables, pero con probabilidades distintas (Estas probabilidades son independiente entre si).
- Las probabilidades del estado inicial. Es un vector que recoge la frecuencia con que se puede encontrar cada estado observable en la primera posición de la secuencia

Sea $X_t = (X_1, ..., X)$ una cadena de Markov a tiempo discreto, esta corresponderá a los estados observables, la cual será la secuencia de nucleótidos que se quiere analizar.

Su espacio de estados es $S = \{A, T, G, C\}$

Con matriz de transición asociada:

$$P = \begin{array}{c} \\ I \\ G1 \\ \vdots \\ Gr \end{array} \begin{array}{c} \begin{array}{cccc} A & C & T & G \end{array} \\ \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ \vdots & \vdots & \vdots & \vdots \\ p_{r1} & p_{r2} & p_{r3} & p_{r4} \end{pmatrix} \end{array}$$

Sea $Y = (Y_1, ..., YN)$ una cadena de Markov a tiempo discreto, esta jugará el rol de la cadena oculta de Markov y representará dada una secuencia de nucleotidos observados el posible correpondiente.

Su espacio de estados corresponde $S = \{I, G1, G2, ..., Gr\}$ donde Gi corresponde al gen i-ésimo y donde r es el número de estados ocultos.

Con su matriz de transición asociada:

$$\Lambda = \begin{array}{c} \\ I \\ G1 \\ \vdots \\ Gr \end{array} \begin{array}{c} \begin{array}{cccc} I & G1 & \dots & Gr \end{array} \\ \begin{pmatrix} \lambda_{11} & \lambda_{12} & \dots & \lambda_{1r} \\ \lambda_{21} & \lambda_{22} & \dots & \lambda_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{r1} & \lambda_{r2} & \dots & \lambda_{rr} \end{pmatrix} \end{array}$$

Por ejemplo, dada una cadena X_t observable se le asocia un proceso Y_t el cual va a estar oculto e indica en que gen se encuentra la cadena X_t:

t	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	$i+7...$
Y_t	$...G1$	$G1$	$G1$	I	$G3$	$G3$	$G4$	$I...$
X_t	$...C$	A	T	G	T	T	A	$C..$

El problema está en dada una secuencia $X_{OBS} = \{X_i, ..., X_n\}$ con una cadena oculta asociada $Y_t^* = \{Y_1^*, ..., Y_n^*\}$ se quiere maximizar la función de verosimilitud, pues donde el argumento se maximice será las distribución más probable y así poder conseguir identificar un segmento que codifique algún

gen y además se obtendrá todas las segmentos de un gen dispersos el una cierta alineación de ADN dada.

$$(Y_1^*, ..., Y_n^*) = argmax \{P(Y_1, ..., Y_n | X_1, ..., Y_n)\}$$

Se tiene que:

$$P(r, P, \Lambda) = P(r)P(\lambda|r)P(P|r)$$

Usando el teorema de Bayes es posible poder maximizar la función de verosimilitud dada una cadena observada X_{OBS} y número de estados ocultos dado S.

$$P(r, P, \Lambda, s|X_{OBS}) \propto p(r, \Lambda, P)P(X_{OBS}|s, r, P)P(s|r, P, \Lambda) \tag{1}$$

$$P(X_{OBS}, s|r, \Lambda, P) \propto \prod_{i \in S} \prod_{j \in} \lambda_{ij}^{m_{ij}} \prod_{k \in K} \prod_{l \in L} P_{kl}^{n_{kl}}$$

donde

$$m_{ij} = \sum_{t \geq o} 1(X_{t-1} = i, X_t = i), n_{kl} = \sum_{t \geq o} 1(y_t = k, X_t = l)$$

Obtener muestras aleatorias de (1) para poder maximizar la función de verosimilitud resulta ser difícil; por esta razón se usarán algoritmo de simulación de tipo MCMC para lograr el objetivo del proyecto que se explicará en siguiente subsección.

4.2 Método Monte Carlo via Cadenas de Markov

Los métodos MCMC tienen como objetivo crear una cadena de markor Ergódica la cual converge a su distribución estacionaria (por el teoriema Ergódico) y esta coincide con la función objetivo f.

Teorema Ergódico:
Sea $\{X_n, n \geq 0\}$ una cadena de Markov irreducible con matriz de transición P. Sunponga además que P tiene distribución estacionaria π. Sea $f : \varepsilon \to R$, tal que:

$$\pi |f| = \sum_{i \in \varepsilon} \pi(i) |f(i)| < \infty$$

entonces,

$$\lim_{n \to \infty} \frac{1}{n} \sum_{j=0}^{n-1} f(X_j) = \pi f = \sum_{i \in \varepsilon} \pi(i) f(i)$$

con probablidad 1.

Este método es útil al momento de simular distribuciones que son muy raras o complicadas de simular. En el caso de querer una muestra aleatoria la distribución complicada esta tomará el papel de la función objetivo; así creando la cadena de Markov adecuado a partir de una cierta M (M es número de iteraciones necesarias para que convenga la cadena a la distribución objetivo o estacionaria) las

muestras obtenidas se distribuyen de la forma deseada.

4.2.1 Método Metropolis Hasting

El algoritmo Metropolis Hasting es algoritmo de tipo MCMC más popular y eso se debe a que las condiciones que debe satisfacer la función objetivo son mínimas.
Este comienza con una distribución objetivo f, de alguna manera se le asocia $q(x|y)$ la cual es una densidad condicional que es fácil de simular o que conozcamos analíticamente, esta densidad es arbitraria el único requerimento es $f(y)/q(x|y)$ es conocida. A q se le conoce como la distribución instrumental.

Algoritmo Metropolis Hasting

1. Generar Y_t $q(x|y)$
2. $X_{t+1} = \begin{cases} Y_t & \textit{con probabilidad } \rho(X^t, Y_t) \\ X_t & \textit{con probabilidad } 1 - \rho(X^t, Y_t) \end{cases}$

$\rho(X,Y) = \textit{mín}\left\{\frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}, 1\right\}$

$\rho(X,Y)$ es la probabilidad de aceptar el valor generado.

4.3 Muestreo de Gibbs

El muestreo de gibbs es un caso particular de Metropolis Hasting, este no necesita una distribución instrumental; solo se necesita poder generar muestras de la distribución posterior condicional de cada uno de los parámetros individuales.

Es decir, se puede generar muestras de la distribución:

$$p(\theta_1, \theta_2, ..., \theta_k|x)$$

si se conoce:

$$\begin{aligned} p(\theta_1|\theta_2, ..., \theta_k, x) \\ p(\theta_1|\theta_2, \theta_3, ..., \theta_k, x) \\ p(\theta_2|\theta_1, \theta_3, ..., \theta_k, x) \\ .... \\ p(\theta_k|\theta_1, \theta_2, ..., \theta_{k-1}, x) \end{aligned}$$

El muestreo de Gibbs es una caminata aleatoria a lo largo del espacio de parámetros. La caminata inicia en un punto arbitrario y en cada tiempo el siguiente paso depende únicamente de la posición actual.

4.4 Distribuciones

Dirichlet
Sea $Q = [Q_1, Q_2, .., Q_k]$ tal que $0 \leq Q_i$ para $i = 1, ..., n$ y $\sum_{i=1}^{k} Q_i = 1$ y $\alpha = [\alpha_1, \alpha_2, ..., \alpha_k]$ con $\alpha_i > 0$ para cada i y defina $\alpha_0 = \sum_{i=1}^{k} \alpha_i$, entonces se dice $Q \sim Dirichlet(\alpha)$

$$f(q; \alpha) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} q_i^{\alpha_i - 1}$$

Esta distribución es usada para obtener la probabilidad de k eventos, que se puede representar como una distribución de probabilidad. Además es distribución conjungada con la binomial.
Poisson
Sea $\lambda \in R$, se dice $X \sim Poisson(\lambda)$

$$f_x(t; \lambda) = \frac{\lambda^t \exp -\lambda}{t!}, \quad t \in N$$

Esta distribucipon se usa para calcular la probabilidad de ocurrencia de eventos clasificados como éxito o fracaso dentro de un intervalo de tiempo ponderado con un decaimiento exponencial.

4.5 Simulando con muestreo de Gibbs

Como ya se mencionó se quieren muestras aleatoria de la función de probabilidad:

$$P(r, \lambda, \widehat{P} | X*)$$

donde $X*$ es la muestra o dartos obtenidos.
Siguiendo la idea del muestreo de Gibbs se tiene que simular tres condicionales.
Condicional 1:

$$p(r|\Lambda, \widehat{P}, X*) \sim Poisson(r)$$

Se supondrá que el número de genes real no está muy alejado del estimado y por eso que $p(r|\lambda, \widehat{P}, X*) \sim Poisson(r)$ tiene una distribución Poisson de parámetro r, donde r es el número de estados estimado.

Sea r2 = el número de genes simulados, r = el número de genes estimado, la simulación se divide en dos casos:

- El número de estados simulado es menor al original:
  En este caso, se tiene que encontrar una manera donde se junten genes con un criterio lo más científico posible. Se propone crear una matriz VAR, esta será una matriz que toma las distribuciones de los genes y hace el producto cartesiano aplicando la distancia 2 o distancia Euclidiana. Se obtendrá una matriz cuadrada de rxr que dará una aproximación de que tan parecidas son dos distribuciones y así se recogerán r2 -r genes aleatorias y se juntaron con su distribución más parecida, obviando el hecho que la más parecida a sí misma es ella misma.

- El número de estados simulado es mayor al original:
  En este caso se tiene una situación un poco más difícil pues se tiene que separar algún gen para forma r2 -r genes nuevos. La solución que se propone no es la más científica posible, por esta razón al analizar los resultados se dividirá en dos: una donde se considere esta situación y en la otra se excluirá este caso.
  El método que se propone consiste en elegir r2 - r genes al azar del total de genes disponibles, extraer un fragmento de ADN y colocarlo como nuevo gen.

Codicional 2:

$$p(\lambda|r, \widehat{P}, X*) \sim Dir(\alpha) \quad \alpha = [\alpha_1, \alpha_2, ..., \alpha_r]$$

Las propiedades de la distribución Dirichlet serán muy útiles de ayuda para simular la matriz de transición de los estados oculto, pues cada muestra suma 1, todas las entradas son mayores o iguales a cero, es decir, es una distribución de probabilidad. Por otro lado la distribución Dirichlet además que es conjugada con la distribución Multinomial lo que la hace ideal para simular la nueva matriz.

Se simula la matriz de transición de los estados oculta; tomando como estimador de parámetro ($\alpha = [\alpha_1, ..., \alpha_r]$) la distribución de transición correspondiente el gen i-ésimo.

Condicional 3:

$$p(\lambda|r, \widehat{P}, X*) \sim Dir(\alpha), \quad \alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$$

Por razones análogas a las de la condición anterior se toma la distribución Dirichlet como la distribución de la condicional.

Se simula la matriz de transición de los estados oculta; tomando como estimador de parámetro ($\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$), la distribución de bases nitrogenadas correspondiente el gen i-ésimo.

5 Resultados

Se utilizó como prueba el genoma de la bacteria Buchnera aphidicola, esta tiene como característica ser un genoma mínimo, es decir, el genoma más pequeño capaz de dar soporte a la vida. La ventaja de este, es que al ser una base de datos 'pequeña' podrá ser más manipulables que a comparación de una base de base de datos enorme como es el genoma humano.

Buchnera aphidicola	
Num. de nucleotidos	655725
Num. de genes proteinicos	574
Num. de RNA genes	36

La base de datos se obtuvo de la NCBI (National Center for Biotechnology Information) almacena y constantemente actualiza secuencias genómicas en GenBank.

El programa que se utilizó para el implementar el algoritmo de muestreo de gibbs se pondrá al final del trabajo en el apéndice; fue escrito en Python y ejecutado el Jupyter. Resulta útil programar con python, ya que cuenta con la paquetería 'Biopython' especializada para la manipulación de secuencias genómicas.

Solo se estudiará un parte de secuencia de ADN pues la memoria del editor online con el que se trabajó es pequeña y a la hora de ejecutar el programa con toda la cadena el ordenador se traba con la cantidad de información. Además como se menciona anteriormente el análisis de resultados se divide en dos, de acuerdo al tipo de simulación ejecutada.

Simulación del tipo 1: En esta simulación se analizá el caso donde la simulación de r solo decrece.
Tamaño de la secuencia: 9216

Asumiendo el supuesto de los genes se encuentran distribuidos uniformemente en el genoma se tienes, se puede obtener el número de genes esperados en una secuencia de 9216, a través de un sencillo cálculo.

$$num\ de\ genes\ esperado = \frac{(genes\ estimados)(574)}{655725}$$

El número de genes esperado es 8, si el algoritmo fuera efectivo el número de genes estimados final fuera cercano a este número.

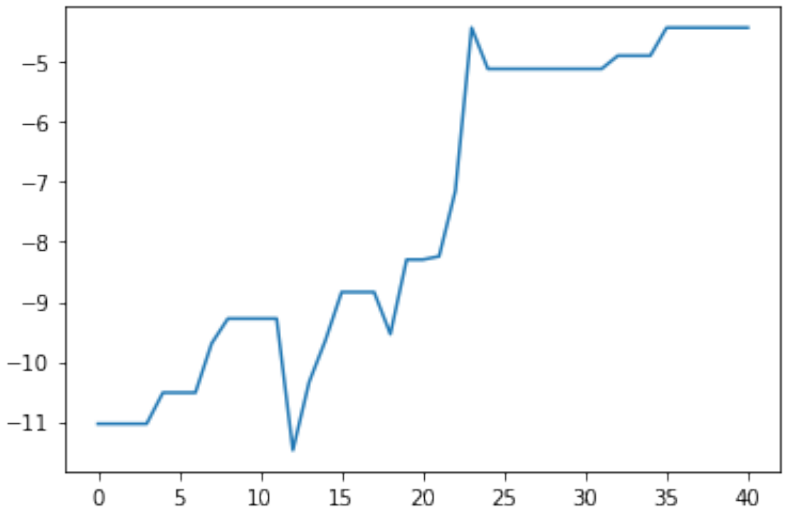

Figure 5: Prueba de tipo 1, Verosimilitud, iteraciones 40

La figura 5 y 6 corresponden a la simulación de tipo 1 en 40 iteraciones.

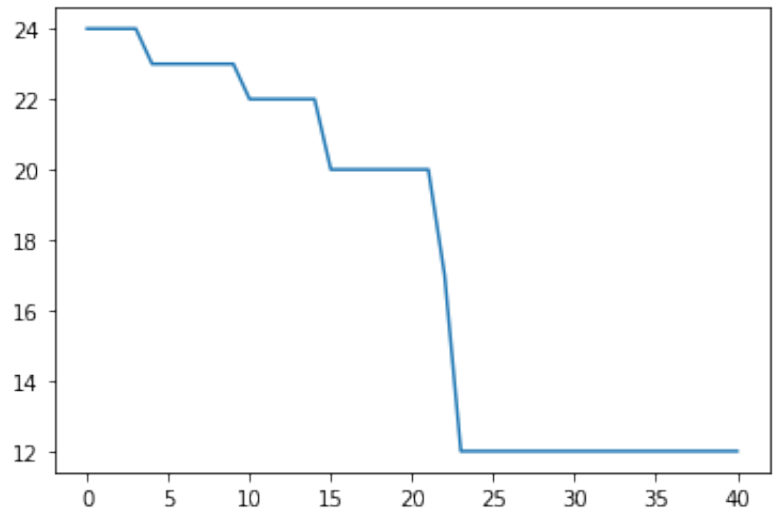

Figure 6: Prueba de tipo 1, número de genes, iteraciones 40

En la figura 5 se puede ver como mejora la función de verosimilitud se puede observar que mejora rápidamente y se estabiliza parece ser bueno pues son solo 40 iteraciones.
La figura 6 muestra el número de genes estimado en cada iteración y lo importante es que al final de las 40 iteraciones se llega a 12 que es cercano al número de genes esperados.

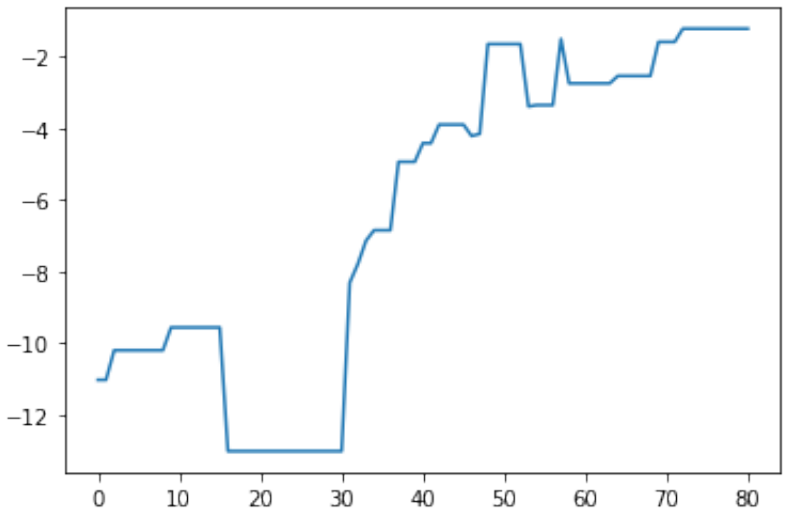

Figure 7: Prueba de tipo 1, número de genes, iteraciones 80

En la figura 7 y 8 se hace exactamente lo mismo que en la ejucación anterior pero aumentando en el número de iteraciones y se obtuvo que el número de genes estimados es 7 el cual es bastante cercano.

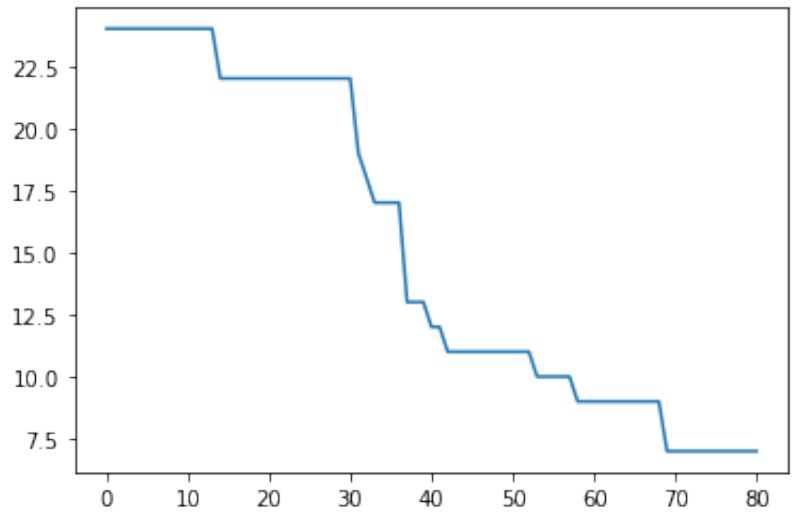

Figure 8: Prueba de tipo 1, número de genes, iteraciones 80

Finalmente para concluir con este tipo de simulación, se obtendra el promedio del número de genes simulados. Se ejecuta este algoritmo 50 veces con la cadena de tamaño 9216 y con 40 iteraciones correspondientes al muestro de Gibbs. El promedio resulto de 9.3333 el cual es cercano al número de genes esperado. Haciendo lo mismo pero ahora con 80 iteraciones se obtuvo que el promedio es de 6.26666.

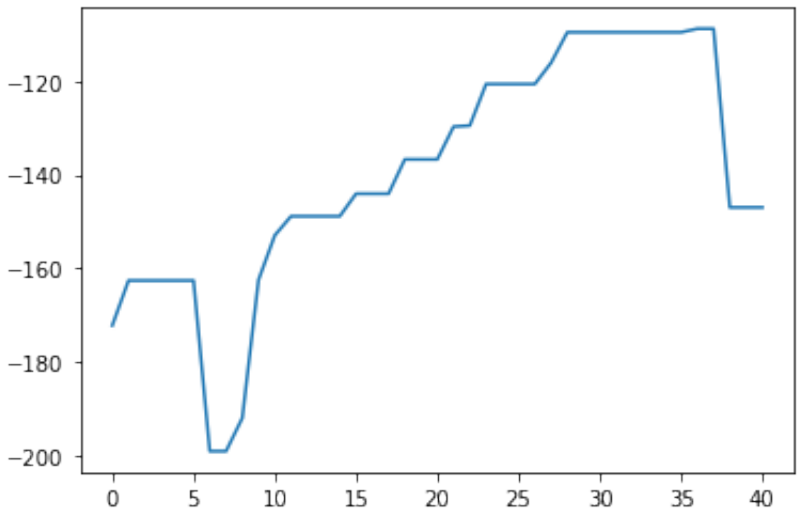

Figure 9: Prueba de tipo 1, función de verosimilitud, iteraciones 80

En la figura 9 y 10 se cambio la cadena a una más larga de 149 583 caracteres y el número de genes esperados es 130, para observar como funcina el algoritmo con cadenas más largas. No se puede simular

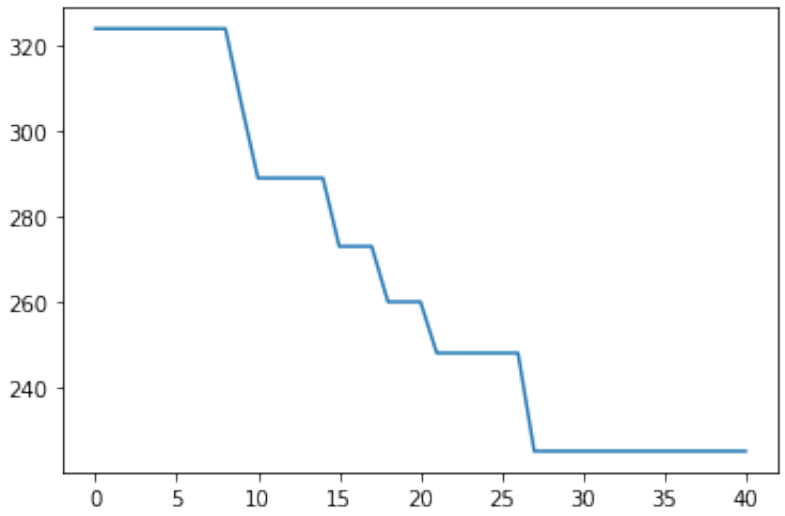

Figure 10: Prueba de tipo 1, función de verosimilitud, iteraciones 80

tantas veces este tipo de muestras pues el tiempo requerido aumenta muy radicalmente para esta en particular tardó casi una hora. Como resultado de la simulación se obtuvo 225 genes estimados el cuál está alejado del número que se desea obtener, pero hay que considerar que solo fueron 40 iteraciones.

Simualción de tipo 2

Consiste en la simulación del parámetro r quitando y añadiendo genes, pero al agregar genes no es un método científico.

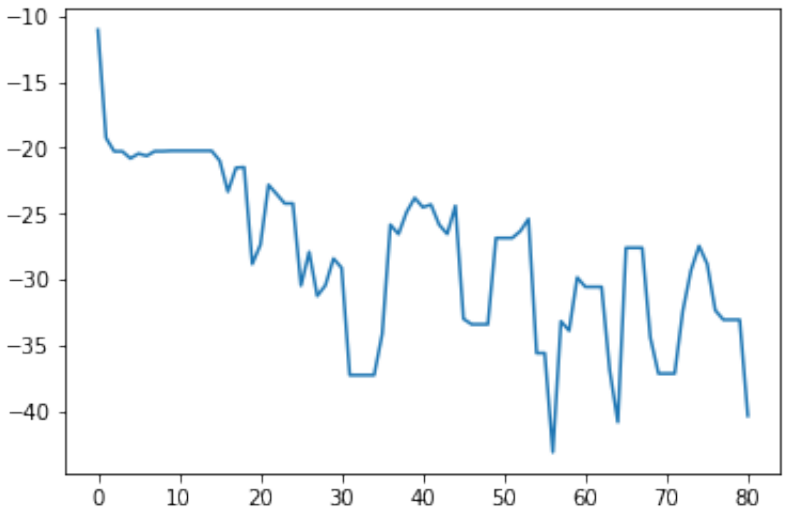

Figure 11: Prueba de tipo 2, función de verosimilitud, iteraciones 40

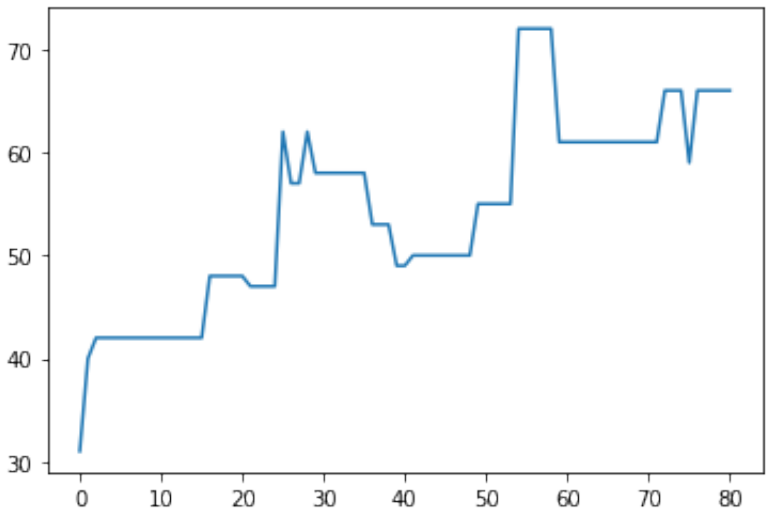

Figure 12: Prueba de tipo 2, número de genes, iteraciones 40

La figura 11 que corresponde a la gráfica de la función de verosimilitud la que parece no mejorar progresivamente, más bien empeora. En la figura 12 se puede observar que el número de genes parece aumentar y disminuir con una gran variabilidad yno llegar a una estabilidad.

Con este se puede concluir que para forma nuevos genes extrayendo fragmentos de forma aleatoria no es una buena para estimar. Si se encontrará una mejor forma de implementar este aumento de genes, basados en argumentos científicos se podría hacer una mejor estimación.

Ahora analisis en tiempo que tan efectivo es el algoritmo.

6 Conclusiones

Conclusiones: Los resultados que se obtuvieron no fueron tan satisfactorios como se planeó en un principio, pueden ser mejorados:

- Dar un criterio científico para la creación de un nuevo gen a la hora de simular.
- Combinar el muestreo de Gibbs con el algoritmo de enfriamiento estocástico para no tener cambios tan radicales en los parámetros en las iteraciones finales.
- Hacer una separación de genes más efectiva, pues en este paso al tener cadenas muy grandes el programa puede tardar demasiada o trabar la computadora.

7 Bibliografia

References

[1] Dirk Husmeier y and Grainne McGuire (2002), *Detecting recombination with MCMC*, Limusa, Oxford University.

[2] Nicholas C. Grassly y Edward C. Holmes (1997), *A Likelihood Method for the Detection of Selection and Recombination Using Nucleotide Sequences*, Oxford University.

[3] Pardoux E. (2008), *Markov Processes and Applications: Algorithms, Networks, Genome and Finance (pp. 77-97)*, 1ra edición, Inglaterra, DUNOD.

[4] Puigdoménech P., *II Congreso sobre las Nuevas Tecnologías y sus repercusiones en el seguro: Internet, Biotecnología y Nanotecnología (pp. 155-162)*, 1ra edición, España, SEIDA.

[5] Boys R. J. y Henderson D. A. (2001), *A comparison of reversible jump MCMC algorithms for DNA sequence segmentation using hidden Markov models*, Department of Statistics, University of Newcastle, U.K.

[6] Robert C. P. y Casella G. (2010), *Introducing Monte Carlo Methods with R*, Unites States, Springer.

[7] Ortega Sánchez J. y Rivero Mercado V., *Modelos Estocásticos I: Notas de Curso (pp.47 y 48)* ,CIMAT, A.C.

[8] González Mañas J.M. , *La molécula de ADN, vista por un bioinformático*, URL : $http$: $//www.ehu.eus/biofisica/juanma/bioinf/pdf/dna_molecule_2.pdf$.

[9] Frigyik B. A., Kapila A. y Gupta M. R. (2010), *Introduction to the Dirichlet Distribution and Related Processes*, Department of Electrical Engineering, University of Washington Seattle, URL : $https$: $//web.archive.org/web/20150219021331/https$: $//www.ee.washington.edu/techsite/papers/documents/UWEETR-2010-0006.pdf$.

[10] Bautista Otero P. (2007), *Experimentación con un algoritmo de MCMC multiescala y autoajustable*, CIMAT, Guanajuato.

[11] JAL , *Biología molecular y celular. Bioquímica. Biomedicina*,Fundación Conocimiento, Madrid, URL : $https$: $//www.madrimasd.org/blogs/biocienciatecnologia/2018/07/26/134057$.

8 Apéndice

Código utilizado

```
import Bio
from Bio import SeqIO
from Bio import Entrez
```

```
import re
from Bio.Seq import Seq
import math
import pandas as pd
import scipy.stats as ss
import numpy as np
import matplotlib.pyplot as plt
from timeit import timeit

#Actualizamos la matrix de estados acultos dados un conjunto de genes
def ActMatOcu(genes):
    r = len(genes)
    oculta = list(range(len(genes)))
    for i in range(len(genes)):
        oculta[i] =[]
        for j in range(len(genes)):
            oculta[i].append(2/3*1/(r))
        oculta[i].append(1/3)

    oculta.append([1/len(genes)]*(r+1))

    return(oculta)

def ActMatObs(genes): #Funcion que atualiza la matriz observada
    dist = list(range(len(genes)))

    for i in range(len(genes)):
        temp = ''
        for j in range(len(genes[i])):
            temp = temp + genes[i][j]
        aux = Seq(temp)
        dist[i] = [aux.count('A'), aux.count('C'), aux.count('T'), aux.count('G')]

    #Ditribucion de los intrones
    AI = Seq(cadena).count('A')
    CI = Seq(cadena).count('C')
    TI = Seq(cadena).count('T')
    GI = Seq(cadena).count('G')

    totales = []

    for j in range(4):
        conta = 0
        for i in range(len(dist)):
```

```
            conta = conta + dist[i][j]
        totales.append(conta)

    I = [AI - totales[0], CI -totales[1], TI -totales[2], GI -totales[2]]
    dist.append(I)

    matrizTrans = list(range(len(dist))) #np.zeros((len(dist),4))
    for i in range(len(dist)):
        total = sum(dist[i])
        matrizTrans[i] = []
        for j in range(4):
            matrizTrans[i].append(dist[i][j]*(1/total))

    return(matrizTrans)

def ActCadOcu(genes): #Funcion qeuc crea la nueva cadena oculta de acuerdo a los nuevos genes estimados
    gen = list(range(len(genes)))
    cadOculta = ''

    for i in range(len(genes)):
        for j in range(len(genes[i])):
            laLista = re.split('ATG'+ genes[i][j], cadena)
            cadOculta = cadOculta + len(laLista[0])*'I' + (len(genes[i][j]) + 3)*(str(gen[i]))
            cade = laLista[1]
    cadOculta = cadOculta + 'I'*len(cade)

    return(cadOculta)

def ObtenerGenes(cadena):
    posible = re.split('ATG', cadena)
    posible2 = re.split('ATG', cadena)
    posible.pop(0)
    posible2.pop(0)
    for i in range(len(posible)):
        posible[i] = 'ATG' + posible[i]
    patrone = '((TAA)|(TGA)|(TAG))'

    #Separamos por los STOPS encontramos los posibles genes
    posgen = list(range(len(posible2)))
    posgen2 = list(range(len(posible2)))

    for i in range(len(posible)):
        if posible2[i] == re.split(patrone, posible2[i])[0]:
```

```
            posgen[i] = 'rep'
            posgen2[i] = 'rep'
        else:
            posgen[i] = 'ATG' + re.split(patrone, posible2[i])[0] + 'TAA'
            posgen2[i] =re.split(patrone, posible2[i])[0]

cuenta = posgen.count('rep')
for i in range(cuenta):
    posgen.remove('rep')
    posgen2.remove('rep')

#Estos son nuestros posbles genes candidatos
#posgen pero eliminiemos alas secuencias que no pueden ser frgmentos como los de longitud 1

ayuda = []
for i in range(len(posgen2)):
    if len(posgen2[i]) < 3:
        ayuda.append(i)

        #posgen.remove(i)

cuen = 0
for i in range(len(ayuda)):
    posgen2.pop(ayuda[i]-cuen)
    posgen.pop(ayuda[i]-cuen)
    cuen = cuen + 1

#Localizamos el dinucleotido CpG para poder agrupar genes
info = []
for i in range(len(posgen2)):
    info.append(Seq(posgen2[i]).count('CG')/len(posgen2[i]))

#Ya agrupamos los genes con sus fragmentos asociados
genes = []
aux = []
for i in range(len(info)):
    if info[i] == 0:
        aux.append(posgen2[i])
    else:
        genes.append(aux)
        aux = [posgen2[i]]

genes2 = []
aux2 = []
```

```
    for i in range(len(info)):
        if info[i] == 0:
            aux2.append(posgen[i])
        else:
            genes2.append(aux2)
            aux = [posgen[i]]

    return(genes)
#Distancia Eclidania
def dist2(x, y):
    cuento = 0
    for i in range(len(x)):
        cuento = cuento + (x[i]-y[i])**2
    return math.sqrt(cuento)

#Si un exon tiene una distribucion similar a otra los asociaremos si queremos disminuir el
    nmero estimado de r

def varia(matrizTrans):
    var = list(range(len(matrizTrans)))

    for i in range(len(matrizTrans)):
        var[i] = []
        for j in range(len(matrizTrans)):
            var[i].append(dist2(matrizTrans[i], matrizTrans[j]))

    return(var)

def SimR(r):
    rnueva = ss.poisson.rvs(r)
    return(rnueva)

def rep(num, n):
    x = list(range(n))
    for i in range(n):
        x[i] = num
    return(x)

def verosimilitud(cadena, cadOculta, matrizTrans, oculta):
    gen = list(range(len(oculta)-1))

    for i in range(len(gen)):
        gen[i] = str(i)
    gen = gen + ['I']
```

```
    matObs = pd.DataFrame(matrizTrans)
    matObs.columns = ['A', 'C', 'T', 'G']
    matObs.index = gen

    matOcu = pd.DataFrame(oculta)
    matOcu.columns = gen
    matOcu.index = gen

    #print(matObs)
    #print(matOcu)

    cuentaOcu = 2 + 1/len(gen) #Probabilidad de iniciar en cualquier estado oculto
    edoOcu = re.split(' ', cadOculta)
    edoObs = re.split(' ', cadena)
    cuentaObs = 2 + matObs[edoObs[1]][edoOcu[1]]

    for i in range(len(oculta)-1):
        cuentaOcu = np.log(matOcu[edoOcu[i+2]][edoOcu[i+1]]+2)*cuentaOcu
        cuentaObs = cuentaObs*np.log(matObs[edoObs[i+1]][edoOcu[i+1]]+2)
        #print([cuentaOcu, cuentaObs])

    total = np.log(cuentaOcu*cuentaObs)
    return(total)

#Optimizando
#Elegimos un parametro y lo volvemos a siomular
def SimPar(r, cadOcu, matrizTrans, oculta, genes):
    unif = np.random.rand()
    var = varia(matrizTrans)
    var2 = varia(matrizTrans)

    gen = list(range(len(genes)))
    for i in range(len(gen)):
        gen[i] = str(i)
    gen = gen + ['I']

    if unif < 1/3:
        r2 = SimR(r) #Tambien tenemos que volver a simular los otras variables pues
        if r2 < r:
            elegidos = list(set(list(np.random.randint(0, r, r-r2)))) #Borrrar a colapasar uno
                al azar

            var = varia(matrizTrans)
```

```
            var2 = varia(matrizTrans)

            for i in range(len(elegidos)):
                var2[elegidos[i]].remove(min(var2[elegidos[i]]))
                masCercana = min(var2[elegidos[i]])
                indice = var[elegidos[i]].index(masCercana)

                aux = list(range(len(genes)))
                if (indice in aux) and (elegidos[i] in aux):
                    genes[indice] = genes[indice] + list(genes[elegidos[i]])
                    genes.pop(elegidos[i])

            matrizTrans = ActMatObs(genes)
            oculta = ActMatOcu(genes)
            cadOcu = ActCadOcu(genes)
            r = len(genes)

        elif (r < r2 and r >r2): #if (r2 < (len(genesglob)-1)) and (r2>r): #Creamos otro
            renglon
            div = list(np.random.randint(0, r, r2-r))
            div = list(set(div))
            #print(div)
            for i in range(len(div)):
                #print(genes[div[i]])
                nuevoGen = int(np.random.randint(0, len(genes[div[i]]), 1))
                #print(genes[div[i]][nuevoGen])
                aux = [genes[div[i]][nuevoGen]]
                genes.append(aux)

            matrizTrans = ActMatObs(genes)
            oculta = ActMatOcu(genes)
            cadOcu = ActCadOcu(genes)
            r = len(genes)

        else:
            matrizTrans = matrizTrans
        #print('simulamos r')

    elif (1/3 <= unif) and (unif < 2/3):
        #Sim Ocul
        nueva = []
        for i in range(len(oculta)):
            alphai = list(oculta[i])
            if 0 in alphai:
```

```
                alphai = tuple(rep(1/len(oculta[i]), len(oculta[i])))
                nueva.append(list(np.random.dirichlet(alphai, 1)[0]))
            else:
                alphai = tuple(oculta[i])
                nueva.append(list(np.random.dirichlet(alphai, 1)[0]))
        oculta = nueva
        #print('simulamos MatOcu')
    else:
        nueva = []

        for i in range(len(matrizTrans)):
            alphai = list(matrizTrans[i])
            if 0 in alphai:
                alphai = tuple(rep(1/4, 4))
                nueva.append(list(np.random.dirichlet(alphai, 1)[0]))
            else:
                alphai = tuple(matrizTrans[i])
                nueva.append(list(np.random.dirichlet(alphai, 1)[0]))

        matrizTrans = nueva
        #print('simulamos MatObs')

    return([r, cadOcu, matrizTrans, oculta, genes])

#Muestreo de Gibbs
def MuestreoGibs(cadena, cadOculta, matrizTrans, oculta, genes, num):
    compara = verosimilitud(cadena, cadOculta, matrizTrans, oculta) #nmero de la verosimilitud
        :v
    graf = [compara]
    r = len(genes)
    numGen = [r]

    for i in range(num):
        theta = SimPar(r, cadOculta, matrizTrans, oculta, genes) #lista de esta forma: [r,
            cadOcu, matrizTrans, oculta, genes]
        compara2 = verosimilitud(cadena, theta[1], theta[2], theta[3])

        if compara > compara2: #Si no mejora le damos chance tal vez ponerna

            proba = compara2/compara
            proba = abs(proba) - abs(int(proba))
            #if proba > 1:
            #    proba = 1/proba
```

```
            desci = int(np.random.binomial(1, 1-proba, 1))

            if desci == 1:
                r = theta[0]
                cadOculta = theta[1]
                matrizTrans = theta[2]
                oculta = theta[3]
                genes = theta[4]
                compara = compara2
        else: #Si mejora nos quedamos con la nueva
            r = theta[0]
            cadOculta = theta[1]
            matrizTrans = theta[2]
            oculta = theta[3]
            genes = theta[4]
            compara = compara2
        graf.append(compara)
        numGen.append(len(genes))

        #print(compara)
    return([cadena, cadOculta, matrizTrans, oculta, graf, numGen])

instanteInicial = datetime.now()

genesglob = ObtenerGenes(cadena)
ocultaglob = ActMatOcu(genesglob)
matrizTransglob = ActMatObs(genesglob)
cadOcuglob = ActCadOcu(genesglob)
instanteFinal = datetime.now()
tiempo = instanteFinal - instanteInicial

s
```