
Universidad de Valladolid

# Markov Chain Monte Carlo Metropolis-Hastings en Criptografia

*Alejandro Barón García*

Algoritmos y Computación

Trabajo individual

Curso 2018-2019

# Índice

# 1. Introduccion

En este informe se expone el fundamento teórico sobre métodos de Monte Carlo aplicado a simulación en una cadena de Markov así como un ejemplo de aplicación del algoritmo de Metropolis-Hastings a un problema de desencriptado.

## 1.1 Markov chain Monte Carlo

Llamamos algoritmos de Markov chain Monte Carlo (en adelante MCMC) a aquellos algoritmos utilizados para muestrear(simular) una funcion de probabilidad. Esto tiene especial interés en la problemática de distribuciones con alta dimensionalidad, ya que de otro modo serían imposibles de simular.

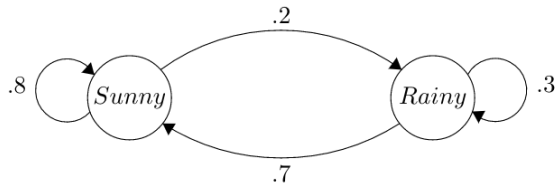
La idea detrás de estos algoritmos es la de generar un camino aleatorio sobre los valores de la distribución con el objetivo de ir muestreando la misma. La manera de construir el algoritmo de MCMC es empleando una cadena ergódica cuya distribución límite sea la de la distribución objetivo.

## 1.2 Breve introducción a las cadenas de Markov

### Cadenas de Markov

Pero, ¿qué es una cadena de Markov? Decimos que un proceso estocástico sigue una cadena de Markov si la probabilidad del siguiente evento solo depende del inmediatamente anterior. Un ejemplo de esto es un juego de azar:

*"Supongamos que tienes 10 euros. Lanzamos una moneda al aire. Si sale cara, ganas 1 euro, si sale cruz lo pierdes. Sea Xn el dinero que tienes en cada momento n. Xn+1 estará únicamente determinado por el dinero que tenías en Xn"*




*Cadena de Markov en la que el clima de un día depende solo del día anterior*

Esta propiedad, llamada la propiedad de Markov términos de probabilidad, se expresa:

$$P(X_n = k|X_n = a, X_{n-1} = b) = P(X_n = k|X_n = a)$$

**Matriz de transicion**

Un elemento clave a la hora de trabajar con Cadenas de Markov, es la matriz de transición (P).Consiste en una matriz cuadrada con tantas filas/columnas como estados posibles tenga la cadena, tal que para cada par de estados i,j se tiene que:

$$P(i,j) = P(X_{n+1} = j|X_n = i) = \textit{Probabilidad de ir del estado } i \textit{ al } j$$

**Distribución estacionaria**

Supongamos que la distribución inicial de los estados es aleatoria,esto es, que se puede empezar en cada estado con una probabilidad p. Por ejemplo, en una cadena con 3 estados, tendremos que se puede empezar en los estados [1,2,3] con probabilidades q=[0.4,0.5.0.1].

Podemos decir que,dada esa aleatoriedad de comienzo, en el siguiente paso estaremos en cada estado con una probabilidad $q * P = t$. Si seguimos multiplicando $t * P$ sucesivamente, en algunas cadenas se da una convergencia hasta llegar a una probabilidad $\pi$ de tal manera que se cumple que $\pi P = \pi$ . A esta distribución se la denomina **distribución estacionaria**.Se interpreta como la probabilidad de volver a un mismo estado a lo largo del tiempo así como la frecuencia de visitas a cada estado. En nuestra problemática de MCMC serán las probabilidades de la distribución que trataremos de simular.

**Periodicidad**

Otro concepto necesario es lo que se llama periodicidad. Sea el periodo el número de pasos necesario para retornar a un estado. Decimos que una cadena es aperiódica si su periodo es 1, o si hay dos estados cuyo mínimo común múltiplo de sus periodos sea 1. Esto garantiza algunas propiedades como la existencia de distribución estacionaria, lo cuál es necesario para el algoritmo de Metropolis-Hastings.

### 1.2.1 Monte Carlo en Cadenas de Markov

Pero, ¿cómo puede sernos de utilidad métodos de Montecarlo? La idea base es simular un **Camino Aleatorio** sobre una cadena de Markov cuya distribución estacionaria converja a la distribución a simular. De este modo, podremos muestrear la distribución con la certeza de que la frecuencia relativa de los estados en la muestra se aproximará a las probabilidades a priori de la distribución a muestrear.

# 2. Algoritmo de Metrópolis-Hastings

El algoritmo de Metrópolis Hastings es uno de los algoritmos MCMC más comunes. Este algoritmo fue implementado por primera vez en 1953 por Nicholas Metropolis para distribuciones simétricas, y fue ampliado en 1970 por Wilfred Hastings.

## 2.1 Funcionamiento

Sea $\pi$= ($\pi$1, $\pi$2,...) una distribución de probabilidad (discreta).El algoritmos simulará una cadena de Markov cuya distribución estacionaria es $\pi$.

Sea J una matriz de transición de una cadena de Markov muestreable, finita e irreducible(todos los estados son accesibles desde todos los demás en un número finito de pasos) con el mismo número de estados que la distribución a simular a la que denominamos matriz subyacente o de propuesta .

Simulando la matriz J, iremos generando estados de la distribución a simular en función del os estados de J. De forma genérica, siendo Xn=i el estado actual y Xn+1=j el **candidato**:

- Elegir un estado j a partir del estado actual i de J con probabilidad J(i,j). Este estado se llama estado propuesta
- Se decide si pasar al estado j o permanecer en el i según la función de aceptación $a(i, j)$

### 2.1.1 Función de Aceptación

A partir del los estados de J, podemos simular estados de la distribución objetivo según sus probabilidades $\pi$. Defininimos la función de aceptación $a(i, j)$ como:

$$a(i, j) = \frac{\pi_j J_{ji}}{\pi_i J_{ij}}$$

Aceptaremos el estado j si $a(i, j) \geq 1$. Si $a(i, j) < 1$, se acepta j con probabilidad $a(i, j)$. En caso de rechazar j, seguimos en i. Esto es equivalente a, siendo runif(0,1) un valor aleatorio uniformemente distribuido entre 0 y 1:

$$X_{n+1} = j \text{ si } runif(0, 1) < a(i, j)$$
$$X_{n+1} = i \text{ si } runif(0, 1) \geq a(i, j)$$

El camino simulado por este algoritmo corresponde a una cadena de Markov cuya distribución estacionaria es $\pi$ (ver apartado 4.1 de convergencia para la demostración)

# 3. Ejemplo en Criptografía

Vamos a ver un ejemplo de aplicación del Algoritmo a la decodificación de un mensaje encriptado con un cifrado de sustitución.

Un cifrado de sustitución consiste en sustituir una letra del abecedario por otra según dictamine la función de sustitución.Así por ejemplo, con la función de encriptado $f = (b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a)$, la palabra "hola" pasará a ser "ipmb". Para toda función de cifrado por sustitución, habrá una de descifrado. El objetivo de nuestro algoritmo será encontrar cuál es esa función de descifrado, haciendo un recorrido sobre todas las funciones de desencriptado asignándolas un estado de una cadena de Markov. ¿Cuál será esta cadena? Como hemos visto anteriormente, necesitamos un vector de probabilidades discretas. Para el alfabeto inglés (A-Z sin caracteres especiales) tenemos 26! (403291461126605635584000000) posibles funciones de desencriptado .

A cada función le asignaremos un score,siendo este una estimación según cuántas veces aparecen los pares sucesivos de caracteres (después de desencriptar el texto) en el lenguaje anglosajón. A más veces aparezca cada par $f(c_i)f(c_{i+1})$, más score tendrá la función.

$$score(f) = \prod_i^{LongTexto} M[f(c_i), f(c_{i+1})]$$

Para recorrelas y muestrearlas,a las funciones de desencriptado les asignaremos una probabilidad $\pi_j$ en función de su "score".La idea es ir muestreando funciones cada vez más "probables como correctas" con un $\pi_j$ más alto. Los datos de la matriz de ocurrencias M los obtenemos de los recursos de la obra de Robert Dobrow [3] los cuales se han calculado procesando textos de Jane Austen, de tal manera que los $\pi_j$ representarán las probabilidades de los estados a muestrear:

$$\pi_j = score(f_j)/\sum_g score(f_g)$$

**Distribuciones desconocidas o incalculables**

Un apunte interesante es que no es necesario conocer el vector de probabilidades $\pi$. Al basarse en $a(i, j)$ en el cociente de los $\pi_k$, nos basta con conocer el ratio o proporción de aparición de un estado frente al otro $\frac{\pi_j}{\pi_i}$.

Esto es muy ventajoso para nuestro ejemplo, ya que el denominador del cálculo de los $\pi_j$ no es fácilmente gestionable por un sistema computacional (es la suma de 27! términos), y en el cálculo de los ratios este sumatorio se simplificará.

Además, si pasamos de una función $f_i$ a otra $f_j$ permutando únicamente dos posiciones de cada función,tendremos que $J_{ij} = J_{ji}$ (J es lo que se llama un camino simétrico) y podremos calcular la función de aceptación a(i,j):

$$a(i,j) = \frac{\pi_j J_{ji}}{\pi_i J_{ij}} = \frac{\pi_j}{\pi_i} = \frac{score(f_j)}{score(f_i)}$$

El algoritmo actúa de la siguiente manera (pseudocódigo en apartado 4)

1. Seleccionar una función inicial $f_i$ (con i=0 en el primer paso)
2. Permutar 2 letras al azar de $f_i$ para obtener $f_j$
3. Calcular a(i,j) y decidir si aceptar $f_j$ como se vio en la seccion 2.1.1
4. Volver al paso 1, iterando tantas veces como sean necesarias hasta llegar a la solución

A continuación se muestra un ejemplo de ejecución con el número de iteración junto al texto desencriptado en cada paso. En 2000 iteraciones, consiguió desencriptarlo.

0 ablhl whl abhll kdgpn me fdln fdln pwsg fdln wgp nawadnadyn swhk avwdg d ymqfp gma yfwds abwa d vwn nswhalh abwg ndrao edjl mablh zqon iqa abl wjlhwzl me ndrao edjl mablh zqon ylhawdgfo elogswg w ndgzfl plwab dn w ahwzlpo w sdffdmg plwabn dn w nawadnady xmnlcb nawfdg ymdgydplgyln dg zlglhwf whl zhlwa naqsifdgz ifmykn dg abl vwo me abwa yfwnn me abdgklhn vbm bwjl illg lpqywalp am kgmv gmabdgz me abl ablmho me chmiwidfdadln abwa ablmho am vbdyb abl smna zfmhdmqn mixlyan me bqswg hlnlwhyb whl dgplialp emh abl smna zfmhdmqn me dffqnahwadmgn lpzwh wfflg cml smhzql smhcblqn abdn dn omqh fwna ybwgyl wealh abdn ablhl dn gm aqhgdgz iwyk omq awkl abl ifql cdff abl namho lgpn omq vwkl qc dg omqh ilp wgp ilfdljl vbwaljlh omq vwga am ilfdljl omq awkl abl hlp cdff omq nawo dg vmgplhfwgp wgp d nbmv omq bmv pllc abl hwiida bmfl zmln

1000 ulete ate ultee yirds ok mies mies dacr mies ard suauisuibs caty upair i bonmd rou bmaic ulau i pas scatuet ular siquf kive oulet wnfs hnu ule avetawe ok siquf kive oulet wnfs betuairmf kefrcar a sirwme deaul is a utawedf a cimmior deauls is a suauisuib josegl suamir boirbiderbes ir weretam ate wteau sunchmirw hmobys ir ule paf ok ulau bmass ok uliryets plo lave heer ednbaued uo yrop roulirw ok ule uleotf ok gtohahimiuies ulau uleotf uo plibl ule cosu wmotions ohjebus ok lncar teseatbl ate irdehued kot ule cosu wmotions ok immnsutauiors edwat ammer goe cotwne cotglens ulis is font masu blarbe akuet ulis ulete is ro untrirw haby fon uaye ule hmne gimm ule suotf erds fon paye ng ir font hed ard hemieve plauevet fon paru uo hemieve fon uaye ule ted gimm fon suaf ir pordetmard ard i slop fon lop deeg ule tahhiu lome woes

2000 there are three kinds of lies lies damn lies and statistics mark twain i could not claim that i was smarter than sixty five other guys but the average of sixty five other guys certainly feynman a single death is a tragedy a million deaths is a statistic joseph stalin coincidences in general are great stumbling blocks in the way of that class of thinkers who have been educated to know nothing of the theory of probabilities that theory to which the most glorious objects of human research are indebted for the most glorious of

illustrations edgar allen poe morgue morpheus this is your last chance after this there is no turning back you take the blue pill the story ends you wake up in your bed and believe whatever you want to believe you take the red pill you stay in wonderland and i show you how deep the rabbit hole goes

## 3.1 Reflexión propia:Problemas del Algoritmo

Quiero decir de antemano que las fallas del algoritmo aquí mostradas son fruto de mi propia experiencia, encontradas de forma empírica.

#### Mensaje de entrada

El mensaje de entrada del algoritmo es crucial para que este consiga desencriptar el texto. Debemos aplicar el algoritmo a un texto rico, con diversas palabras que sean una representación general del idioma a desencriptar. Es por esto que para frases cortas, no se consigue desencriptar el texto.

Hubo ocasiones que probando textos como los primeros versículos del génesis o poemas,quizás por ser un lenguaje algo arcaico, no conseguía descifrarlos si los añadía solos. Sin embargo, al adjuntarlos al texto de muestra (mezcla de textos propuestos por los autores), sí que conseguía traducirlos, gracias a que este texto muestra es rico en variedad de palabras.

#### Lenguaje Castellano

En los scripts de Python3 adjuntos, se encuentra `text_processor.py`, el cual calcula la matriz de ocurrencias para textos en castellano. A pesar de verificar que son correctas(salvo por pequeños valores atípicos que no perturbarían el análisis), no he conseguido aplicar el algoritmo a textos en castellano.Quizás esto se deba a que el espacio muestral aumenta considerablemente (hablamos de 28! si añadimos la letra ñ) o a que los textos que proporcioné no eran suficientemente nutridos como se menciona anteriormente.

#### Fallo del algoritmo

En ocasiones, el algoritmo converge a una solución que no es la correcta, es decir, no llega a muestrear la función verdaderamente más probable (con mayor score) sino que se queda en un máximo local. Aún sin ser muy frecuente esta situación, sí que se ha dado en algunas ocasiones. Esto se debe a que el camino que sigue le hace llegar a una función f cuyas funciones adyacentes (aquellas a las que puede acceder permutando dos letras) no poseen un score mayor, por lo que muestrea como $\pi_i$ mayor una que no lo es. También puede deberse a que los scores fueron estimado con gran variedad palabras, pero sigue siendo una estimación con su consecuente posible error.

# 4. Convergencia y Coste

## 4.1 Pseudocódigo

**Algorithm 1** Desencriptado MCMC

**Require:** Mensaje encriptado en minúsuculas, solo letras y sin puntuación

$f_i \leftarrow funcion\_identidad$
$score_i \leftarrow score(funcion\_identidad)$
**for** $i < n\_iteraciones; i++$ **do**
  $f_j = permutar2(f_i)$
  $score_j = score(f_j)$
  $a(i,j) = score_j / score_i$
  **if** $runif(0,1) < a(i,j)$ **then**
    $f_i = f_j$
    $score_i = score_j$
  **end if**
**end for**

## 4.2 Coste

### 4.2.1 Caso genérico del algoritmo de Metropolis Hastings

En este apartado se describe el coste para el algoritmo MH descrito en el apartado 2.1. Supondremos inicializadas las estructuras necesarias: una matriz de propuesta J con k estados (tantos como tenga la distribución) y el vector $\pi$ de probabilidades.

Sea n el número de iteraciones. Para cada iteración:

- Suponemos que generar un estado tiene coste $g$
- Suponemos coste $c$ el cálculo de la función de aceptación a(i,j), en el caso genérico serán 2 accesos (i,j y j,i) a la matriz J de transición y dos accesos al vector $\pi$ (i y j)
- Suponemos coste 1 en la generación de runif(0,1) y en la comprobación de menor o igual que a(i,j)

Todo esto nos lleva a afirmar que cada iteración tiene un coste de $\mathcal{O}(c+v+1)$, iterando n veces, tenemos que el coste es $\mathcal{O}(n(c+1))$

### 4.2.2 Caso específico: criptografía

Basándonos en lo anteriormente expuesto, en nuestro caso el coste de cada iteración será:

- Tenemos un mensaje de longitud l
- Calcular a(i,j) tiene ahora un coste de calcular $score(f_j)$ .
- El coste de la función score(f), que conlleva iterar sobre los l caracteres del **mensaje desencriptado**, es el acceso l veces a la matriz de ocurrencias(coste de acceso 1).
- El coste de desencriptar el mensaje es traducir los l caracteres del mensaje original según la función f (coste de traducción 1=acceder al vector f)
- Luego el $coste\_a(i,j) = coste\_score(f_j) = coste\_desencriptar + coste\_calculo = l + l = 2 * l$
- La permutación la suponemos de coste 1
- La comprobación también la consideramos de coste unitario

  Lo que se traduce en $\mathcal{O}(1 + 2 * l + 1)$, por ende, con n iteraciones tendremos que el coste es $\mathcal{O}((2 * l + 2) * n)$.

  En caso de tener en cuenta el cálculo del score para $f_i$ inicial (al ser la función identidad no hay que desencriptar puesto que mapea $c_i \rightarrow c_i$ para todo $c_i$ del mensaje) estaríamos hablando de un coste de $\mathcal{O}((2*l+2)*n+l)$.Podemos despreciar operaciones de coste 1 e inicialización y simplificarlo a $\mathcal{O}(2l * n)$

## 4.3 Convergencias

### 4.3.1 Prueba de convergencia a la distribución simulada

Sea P la matriz de transición de la cadena de Markov generada por el algoritmo de Metrópolis Hastings a partir de J (matriz propuesta) y $\pi$ (distribución a simular). Por la definición de la función de aceptación, tenemos que

$$P_{ij} \text{ tomará los valores para } i \neq j$$
$$\begin{array}{ll} a(i,j)J_{ij} & si\ \pi_j J_{ji} \leq \pi_i J_{ij} \\ J_{ij} & si\ \pi_j J_{ji} > \pi_i J_{ij} \end{array}$$

Para la demostración, necesitamos conocer qué son las ecuaciones de equilibrio

#### Ecuaciones de equilibrio

Se dice que un proceso de Markov cumple las ecuaciones de equilibrio si para todo $x, y$ del conjunto de estados se tiene que

$$\pi_x P_{xy} = \pi_y P_{yx}$$

Si $\pi$ cumple esto,por ser una condición más restrictiva que $\pi P = \pi$ significa que será la distribución estacionaria de $P$. Veamos si se cumple

Para el caso de $\pi_j J_{ji} \leq \pi_i J_{ij}$

$$\pi_i P_{ij} = \pi_i J_{ij} a(i,j) = \pi_i J_{ij} \frac{\pi_j J_{ji}}{\pi_i J_{ij}} = \pi_j J_{ji} = \pi_j P_{ji}$$

Para el caso $\pi_j J_{ji} > pi_i J_{ij}$ la demostración es la misma,por lo que se cumplen las ecuaciones de balance para P, ergo $\pi$ es la distribución estacionaria

### 4.3.2 Convergencia del error

#### Aproximación 1

La convergencia del error para el algoritmos de Metrópolis-Hastings trata de estudiar la convergencia de la matriz P generada por el algoritmo. Es decir, cuán larga ha de ser la cadena para que $P^n(x,y) \to \pi_y$. Una forma de medir la distancia entre dos distribuciones de probabilidad es la función de TV(total variation). Basándose en esto, Perci Diaconis (primero en plantear la aplicación criptográfica) [2] propone la siguiente aproximación:

$$\|P_x^n - \pi\| = \frac{1}{2} \sum_y |P^n(x,y) - \pi_y| = \max_{A \subseteq X} |P^n(x,A) - \pi_A|$$

De forma intuitiva, se puede analizar como que la distancia máxima en términos de $TV$ entre dos distribuciones de probabilidad $P$ y $\pi$ es el máximo de las distancias, en valor absoluto, desde los estados X a todos los sucesos(subconjuntos) del espacio de llegada

Lo que plantea, dados $P, \pi, x$ y un error $\epsilon > 0$, ¿cómo de grande ha de ser $n$ tal que:

$$\|P_x^n - \pi\| < \epsilon \ ?$$

Como comenta Diaconis en el documento referenciado, hay pocas situaciones prácticas donde este problema sea solucionado.De hecho, no se conoce respuesta en el caso de la criptografía.Este es un buen ejemplo de cómo la convergencia de un algoritmo puede no ser si quiera calculable.

#### Aproximación 2

Dobrow propone otro cálculo de la convergencia:

Tenemos nuestra matriz generada por MH, $P$, y la distribución $\pi$ al la que converge.

Sea $Q = \sqrt{\pi} I$, esto es, una matriz diagonal con elementos $\sqrt{\pi_i}$. La inversa de esta matriz será $Q^{-1} = \frac{1}{\sqrt{\pi}} I$.

Sea $A = QPQ^{-1}$ de tal manera que $A_{ij} = \sqrt{\pi_i} P_{ij} \frac{1}{\sqrt{\pi_j}}$ Se puede ver que como $P$ es reversible por cumplir las ecuaciones de equilibrio, $A_{ij} = A_{ji}$. A es simétrica,y por ser simétrica tiene autovalores reales y diagonalizable (ortogonalmente) por lo que $\exists S / A = SDS^T$ siendo una $S$ matriz ortonormal de autovalores correspondiente a esos autovalores.

Se puede probar que los autovalores de una cadena ergódica están comprendidos entre -1 y 1, valiendo el 1 el máximo de ellos (no lo vamos a hacer porque no es el objetivo principal).

$$1 = \lambda_1 > \lambda_2 \geq \lambda_3 \geq .... \geq -1$$

Como $A = Q * P * Q^{-1}$, $P$ y $A$ tienen los mismos autovalores por ser matrices semejantes.

Entonces,podemos decir que:

$$P = Q^{-1}AQ = Q^{-1}(SDS^T)Q = (Q^{-1}S)D(S^TQ)$$

Con D siendo la matriz diagonal de autovalores de P.

La potencia n-ésima de $P$ ($P^n$ es conocida como la matriz de transición en n pasos) será de la forma

$$P^n = (Q^{-1}S)D^n(S^TQ)$$

Por lo que los $P_{ij}^n = \frac{\sqrt{\pi_j}}{\sqrt{\pi_i}} \sum_{t=1}^{k} \lambda_t^n S_{it} S_{tj}$. Como $P^n$ converge a $\pi$ con $n \to \infty$, podemos decir que :

$$\pi_j = \frac{\sqrt{\pi_j}}{\sqrt{\pi_i}} S_{i1} S_{1j}$$

(dado que el $\lambda_1 = 1$, el resto de $\lambda$ convergerán a 0 con n grande).Entonces tenemos que el error $\epsilon$ :

$$\epsilon = |P_{ij}^n - \pi_j| \leq \frac{\sqrt{\pi_j}}{\sqrt{\pi_i}} \sum_{t=2}^{k} |\lambda_t^n S_{it} S_{tj}| \leq \frac{\sqrt{\pi_j}}{\sqrt{\pi_i}} |\lambda_{MAX}|^n \sum_{t=2}^{k} S_{it} S_{tj} = |\lambda_{MAX}|^n * c_{ij}$$

Con $|\lambda_{MAX}| = \max_{t \neq 1} |\lambda_t|$($2^o$ autovalor más grande en valor absoluto) y $c_{ij}$ una constante que no depende de n. Es por esto que se puede decir que el ratio de convergencia depende del ese segundo autovalor de $P$ más grande.

En muchos problemas (como el de la criptografía) no se puede conocer los autovalores de $P$, pero es interesante ver cómo otro tipo de aproximación nos puede dar la convergencia del error en MCMC.

# 5. Conclusiones

## 5.1 Archivos adjuntos a esta memoria

Los scripts adjuntos han sido programados en Python 3 (*https://www.python.org/download /releases/3.0/* debido al procesado de textos en UTF-8. Se incluye

- `decoder.py`: implementación del algoritmo de Metrópolis-Hastling, basta con ejecutar este fichero con Python 3. Es necesario el uso de las librerías `pandas` y `numpy`.
- `AustenCount.txt`: Contiene la matriz de ocurrencias sobre los pares de caracteres+espacios en el lenguaje inglés. Obtenida de la obra de Robert Dobrow [3].

Además, se incluye una carpeta `textprocessor.py` que contiene lo necesario para sacar la matriz de ocurrencia de los textos que se tengan, en la que se incluyen:

- `Source_Books`: carpeta contenedora de los libros a analizar, en formato PDF.
- `text_processor.py`: script para procesar textos en castellano. Genera un fichero `matrix.txt` y otro `outfile.csv` con las ocurrencias. Es necesario el uso de las librerías `PyPDF2,pandas` y `numpy`

## 5.2 Reflexión final

Este trabajo es una muestra de una posible aplicación de un algoritmo probabilista que quizás con otro tipo de algoritmos no sería tan fácil de implementar.

No obstante, como todo algoritmo de Montecarlo, puede fallar (de hecho, en ocasiones no muestrea la función de desencriptado verdadera), y depende mucho del mensaje de entrada como se comentaba en la sección 3.1. Se invita al lector a probar el algoritmo con distintos textos, y a tratar de conseguir unas buenas ocurrencias y mensaje de entrada para el castellano.

Gracias por detenerse a leer esta memoria, espero que la haya disfrutado.

# 6. Bibliografía


[1] Decrypting Classical Cipher Text Using Markov Chain Monte Carlo
*Jian Chen and Jeffrey S. Rosenthal Department of Statistics, University of Toronto*
http://probability.ca/ jeff/ftpdir/decipherart.pdf

[2] The Markov Chain Monte Carlo Revolution
*Persi Diaconis, Departments of Mathematics and Statistics, Stanford University* https://math.uchicago.edu/ shmuel/Network-course- readings/MCMCRev.pdf
http://statweb.stanford.edu/ cgates/PERSI/papers/MCMCRev.pdf

[3] Introduction to Stochastic Processes with R,First Edition.
*Robert P. Dobrow. 2016 John Wiley & Sons, Inc. Published 2016 by John Wiley & Sons, Inc.*

[4] Experimentación con un algoritmo de MCMC multiescala y autoajustable
*Patricia Bautista Otero, CIMAT Méjico,2007*
https://cimat.repositorioinstitucional.mx/jspui/bitstream/1008/81/2/TE%20238.pdf

[5] Understanding the Metropolis-Hastings Algorithm
*Siddhartha Chib and Edward Greenberg,The American Statistician, November 1995, Vol. 49 No.4*
https://eml.berkeley.edu/reprints/misc/understanding.pdf

[6] http://probability.ca/jeff/ftpdir/clement.pdf

[7] https://en.wikipedia.org/wiki/Metropolis-Hastings_algorithm