Monte Carlo Sampling and MCMC

Rejection Sampling

taken from [https://www.deep-teaching.org/notebooks/monte-carlo-simulation/sampling/exercise-sampling-rejection]

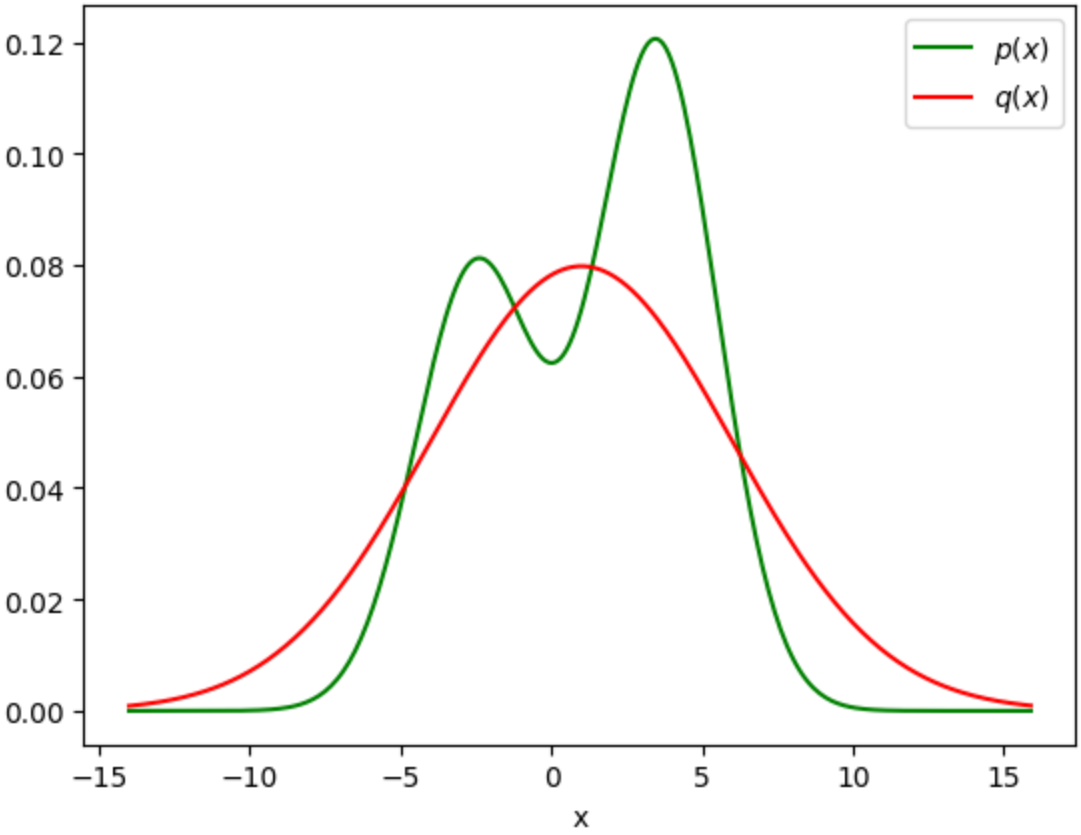
Rejection sampling is a simple and straightforward algorihtm to generate samples for distributions, which a hard or impossible to sample from, using a second enclosing distribution. Using the enclosing function we will sample points and accept them as sample points for our desired distribution if they lie under the desired distribution curve and otherwise reject them.

In [393…

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm

%matplotlib inline
```

In [394…

```
size=40
sigma_square_1 = 4.0
sigma_square_2 = 4.0
mu_1, sigma_1 = -2.5,np.sqrt(sigma_square_1)
mu_2, sigma_2 = 3.5,np.sqrt(sigma_square_1)
prob_1 = 0.4

rv_1 = norm(loc = mu_1, scale = sigma_1)
rv_2 = norm(loc = mu_2, scale = sigma_2)
x_ = np.arange(-14, 16, .1)

p_green = lambda x: prob_1 * rv_1.pdf(x) + (1-prob_1) * rv_2.pdf(x)
plt.plot(x_, p_green(x_) , "g-",label='$p(x)$')

sigma_red,mu_red = 5. , 1.
q_red = lambda x: norm(loc = mu_red, scale = sigma_red).pdf(x)

plt.plot(x_, q_red(x_) , "r-",label='$q(x)$')
plt.legend()

_ = plt.xlabel("x")
```



We want to sample from the green distribution p above but unfortunately we have no way to do so. We will use the normal distribution q to obtain a sample from p using rejection sampling.

Exercise

- Implement rejection sampling to get a sample from $p(x)$.

Let's first hypothesize that we identified a value M such that: $\forall x \in \mathbb{R}, p(x) < M \cdot q(x)$

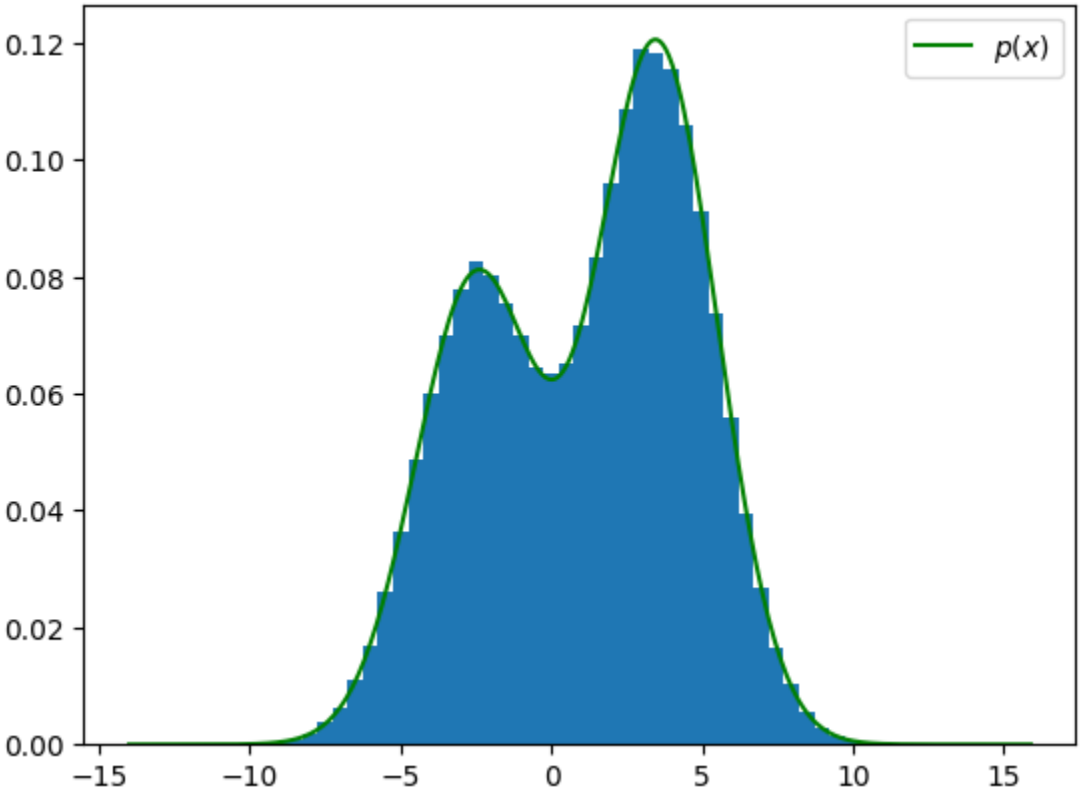
The algorithm for rejection sampling works as follow:

- Sample a value x from q
- Draw an value u using an uniform distribution $\mathcal{U}(0, 1)$ and compare u to $\frac{p(x)}{M \cdot q(x)}$
  - if $u < \frac{p(x)}{M \cdot q(x)}$ then accept the sample x
  - if $u \geq \frac{p(x)}{M \cdot q(x)}$ discard the sample.

You can use an histogram to verify that your sample is indeed approaching the p distribution.

In [395…

```
## Your code can go here.

M = 2
samps = 10**6

x = np.random.normal(mu_red, sigma_red, samps)
u = np.random.uniform(0, 1, samps)
mask = u < (p_green(x) / M / q_red(x))
accepted = x[mask]

plt.hist(accepted, bins=50, density=True)
plt.plot(x_, p_green(x_) , "g-",label='$p(x)$')
plt.legend()
plt.show()
```



Text decoding using Metropolis-Hasting

In this exercise, we will apply the Metropolis-Hastings algorith, to the problem of decoding encoded by substitution, assuming we known the language of the text.

We assume also that we can model this language using bigrams (an order 1 Markov chain on the letters). More formally, this language is written over the alphabet Λ. For example, in English, Λ contains upper and lower case letters, punctuation marks, numbers, etc...

The bigram model is given by μ and A:

- μ is the stationary distribution of single letters probabilities
- A is the transition matrix between letters: it gives for each letter the probability of the next letter.

This model can easily be estimated from a large corpus of text. An encoding (or decoding) function by substitution is a bijective function $\tau : \Lambda \to \Lambda$. If T' is a text, the encoded text $T = \tau(T')$ is obtained by replacing each letter c of T' by $\tau(c)$.

Our formal problem is thus: given and encoded text $T = (c_1, c_2, \ldots, c_{|T|})$ ($c_i \in \Lambda, \forall i$), retrieve the initial decoded text.

Preliminaries

1. How do you compute the likelihood of a sequence $T = (c_1, c_2, \cdots, c_{|T|})$ using a bigram model of parameter (μ, A)?

We note the likelihood this way: $L(T, \mu, A) = P(c_1, c_2, \ldots, c_{|T|} \mid \mu, A)$

One way to find the code would be to use simple Monte Carlo and sample encoding functions, using a distribution proportional to the likelihood of the decoded text.

In [396…

```
# The probability of a sequence is calculated as in a normal markov chain model:
# P(c1,c2,...,ct) = P(c1) * P(c2|c1) * ... * P(ct|ct-1) = mu(c1) * A(c1,c2) * ... *
```

2. How many encoding functions are there? Is it possible to sample?

As the encoding function is bijective on Λ, there are $|\Lambda|!$ such functions. Sampling seems impossible.

As we cannot sample directly a encoding function we will use MCMC. This method only needs the to know sampling probabilities up to a factor, which is the case here.

We will thus construct a Markov chain on the space of all decoded text, by sampling decoding functions τ. The Metropolis-Hastings method works as follow:

- Choose an initial state τ_0 arbitrarily
- Repeat N times the following steps:
  - Compute τ from τ_t by swapping exactly 2 letters c_1 and c_2 in the encoding (see below for details on the function)
  - Accept the transition from τ_t to τ with the probability $\alpha(\tau_t, \tau)$:

$$\alpha(\tau_t, \tau) = \min\left(1, \frac{L(\tau(T), \mu, A) \cdot M(\tau_t, \tau)}{L(\tau_t(T), \mu, A) \cdot M(\tau, \tau_t)}\right)$$

    This mean that ones draw $u \sim \mathcal{U}(0, 1)$:
    - if $u < \alpha(\tau_t, \tau)$ then $\tau_{t+1} = \tau$
    - otherwise $\tau_{t+1} = \tau_t$

After a sufficient large number of of iterations, the decoding function is drawn according to the probabilities of the decoded text.

3. Show that the log probability of acceptation can be written as:

$$\log \alpha(\tau_t, \tau) = \min(0, \log(\mu(\tau(c_1))) + \sum_{i=1}^{|T|} \log(A(\tau(c_i), \tau(c_{i+1})) + \log(\mu(\tau_t(c_1))) - \sum_{i=1}^{|T|}$$

A3

We reform as follows

$$\begin{aligned}
&\log \alpha(\tau_t, \tau) \\
&= \log \min \left(1, \frac{L(\tau(T), \mu, A)}{L(\tau_t(T), \mu, A)} \cdot \frac{M(\tau_t, \tau)}{M(\tau, \tau_t)}\right) \\
&= \min \left(0, \log \frac{L(\tau(T), \mu, A)}{L(\tau_t(T), \mu, A)} + \log \frac{M(\tau_t, \tau)}{M(\tau, \tau_t)}\right) \\
&= \min \left(0, \log L(\tau(T), \mu, A) - \log L(\tau_t(T), \mu, A) + \log M(\tau_t, \tau) - \log M(\tau, \tau_t)\right) \\
&= \min \left(0, \log(\prod_{i=1}^{|T|} A(\tau(c_i), \tau(c_{i+1}))) - \log(\prod_{i=1}^{|T|} A(\tau_t(c_i), \tau_t(c_{i+1}))) + \log M(\tau_t, \tau) - \log M\right. \\
&= \min \left(0, (\sum_{i=1}^{|T|} \log A(\tau(c_i), \tau(c_{i+1}))) - (\sum_{i=1}^{|T|} \log A(\tau_t(c_i), \tau_t(c_{i+1}))) + \log M(\tau_t, \tau) - \log M\right.
\end{aligned}$$

We know that

$$M = \frac{M(\tau, \tau_t)}{M(\tau_t, \tau)}$$

and therefore

$$L(\tau_t(T), \mu, A) < L(\tau(T), \mu, A) \cdot \frac{M(\tau, \tau_t)}{M(\tau_t, \tau)}$$

Which holds for

$$M(\tau, \tau_t) = \log(\mu(\tau_t(c_1))), M(\tau_t, \tau) = \log(\mu(\tau(c_1)))$$

And hence

$$\log \alpha(\tau_t, \tau) = \min \left(0, (\sum_{i=1}^{|T|} \log A(\tau(c_i), \tau(c_{i+1}))) - (\sum_{i=1}^{|T|} \log A(\tau_t(c_i), \tau_t(c_{i+1}))) + \log(\mu(\tau(\right.$$

Not sure why one of the my subterms is negative rather than positive - maybe a typo?

Project

1. Download the file `countWar.pkl` which contains the template of bigrams learned from Tolstoy's novel *War and Peace*. This text was chosen because it contains more than 3 million characters, which is long enough to learn a representative model of a language. Also download the `secret.txt` message to decode (or this one `secret2.txt`, same message coded differently in case you would have problems with the first file).

2. Execute the following code to load in python the bigram model and the secret message.

    - The variable `count` is a dictionary: for each letter of the novel *War and Peace*, it provides its number of occurrences in the novel.
    - The pattern of bigrams is described by the variables `mu` and `A`.
    - The variable `mu` is a vector that contains the initial probability distribution over the letters.
    - `A` is a matrix that gives the probabilities of a letter given another letter.
    - `secret` is a variable containing the message to be decoded.

In [397…

```
import pickle as pkl

# Open the dictionary
with open("./countWar.pkl", 'rb') as f:
    (count, mu, A) = pkl.load(f, encoding='latin1')

with open("./secret.txt", 'r') as f:
    secret = f.read()[0:-1] # -1 to suppress the line break

with open("./secret2.txt", 'r') as f:
    secret2 = f.read()[0:-1] # -1 to suppress the line break
```

3. The decoding functions will be represented as a dictionary where the key and the stored value are both of type character (one letter is encoded/decoded into another letter).

Write a function `swapF : (char,char) dict -> (char,char) dict`
which takes as argument a decoding function τ_t and returns a new decoding function τ constructed by swapping two letters c_1 and c_2 as described in the previous question:

- $\tau(c) = \tau_t(c)$ for any c such as $c \neq c_1$ and $c \neq c_2$;
- $\tau(c_1) = \tau_t(c_2)$
- $\tau(c_2) = \tau_t(c_1)$.

You can test your function with the following dictionary:

In [398…
```
import random
import copy

def swapF(taut: dict[str, str], c1: str = None, c2: str = None) -> dict[str, str]:
    if c1 == None: c1 = random.choice(list(taut.keys()))
    if c2 == None: c2 = random.choice(list(taut.keys()))
    tau = copy.deepcopy(taut)
    tau[c1] = taut[c2]
    tau[c2] = taut[c1]
    return tau
```

In [399…
```
tau = {'a' : 'b', 'b' : 'c', 'c' : 'a', 'd' : 'd' }
print(tau, "\n", swapF(tau))
```

```
{'a': 'b', 'b': 'c', 'c': 'a', 'd': 'd'}
 {'a': 'a', 'b': 'c', 'c': 'b', 'd': 'd'}
```

4. Write a function `decrypt: string x (char,char) dict -> string` which, given a `mess` string and a decoding function `tau`, returns the string obtained by decoding `mess` by `tau`.

You can test your function with the following code:

In [400…
```
def decrypt(mess: str, tau: dict[str, str]) -> str:
    return "".join([tau[c] for c in list(mess)])
```

In [401…
```
tau = {'a' : 'b', 'b' : 'c', 'c' : 'a', 'd' : 'd' }
print(decrypt("aabcd", tau))
print(decrypt( "dcba", tau))
```

```
bbcad
dacb
```

which will produce the following output:

```
>>> decrypt ("aabcd", tau )
 bbcad
>>> decrypt ("dcba", tau )
 dacb
```

5. Create a dictionary (a hash table) associating to each character its index in `mu` or `A`. The code is simply the following:

In [402…
```
chars2index = dict(zip(list(count.keys()), range(len(count.keys()))))
```

`chars2index['a']` simply gives access to the index corresponding to the letter `a` in `mu` or `A`

If you prefer, you can also use the index file that has already been generated: `fileHash.pkl`

6. Write a function `logLikelihood: string x float np.array x float np.2D-array x (char,int) dict-> string` which, given a `mess` message (string), the arrays `mu` and `A` created in question 2 from `pickle` and the previous dictionary `chars2index`, returns the log-likelihood of the `mess` message with respect to the big diagram model `(mu, A)`.

You can test your function with the following code:

In [403…

```
def logLikelihood(mess: str, mu: np.ndarray[float], A: np.ndarray[float], c2i: dict
    ids = np.vectorize(lambda s: c2i[s])(list(mess))
    return np.log(mu[ids[0]]) + np.sum(np.log(np.vectorize(lambda i: A[ids[i], ids[

print(logLikelihood("abcd", mu, A, chars2index), -24.600258560804818)
print(logLikelihood("dcba", mu, A, chars2index), -26.274828997400395)
```

```
-24.600258560804814 -24.600258560804818
-26.274828997400398 -26.274828997400395
```

logLikelihood("abcd", mu, A, chars2index) logLikelihood("dcba", mu, A, chars2index)

which will produce the following output:

```
>>> logLikelihood( "abcd", mu, A, chars2index )
-24.600258560804818
>>> logLikelihood( "dcba", mu, A, chars2index )
-26.274828997400395
```

7. Code the Metropolis-Hastings method seen in TD as a function called `MetropolisHastings(mess, mu, A, tau, N, chars2index)` using the `swapF`, `decrypt` and `logLikelihood` functions:

- The `mess` parameter is the coded message.
- Parameters `mu` and `A` represent the bigram model.
- The argument `tau` is the initial decoding function to start the Metropolis-Hastings algorithm.
- The argument `N` is the maximum number of iterations of the algorithm.
- The argument `chars2index` has already been seen.

The method is a simple loop where we do the following steps:

- draw a new decoding function `tau'` by applying swapF with the current decoding function tau as parameter
- calculation of the log-likelihood of the decoded message thanks to `tau'`.
- draw to accept or not to accept the transition to `tau'` given the the ratio of likelihoods
- if the transition is accepted, save the decoded message with the highest likelihood.

The function returns the most likely decoded message. You will display the log-likelihood each time it is improved and the decoded message is saved.

the function returns the most likely decoded message. You will display the log-likelihood each time it is improved and the corresponding decoded message so that you can observe the evolution of the algorithm.

In order to test your function, you can execute the following code:

In [404…

```
import tqdm
def MetropolisHastings(mess: str, mu: np.ndarray[float], A: np.ndarray[float], tau:
    decode = decrypt(mess, tau)
    loglike = logLikelihood(decode, mu, A, c2i)
    best_msg = decode
    best_loglike = loglike
    for i in tqdm.tqdm(range(N), total=N):
        tau_t = swapF(tau)
        decode_t = decrypt(mess, tau_t)
        loglike_t = logLikelihood(decode_t, mu, A, c2i)
        if np.log(np.random.rand()) < (loglike_t - loglike):
            if loglike_t > loglike:
                best_msg, best_loglike = decode_t, loglike_t
                # print(f"Iteration {i}, loglikelihood {loglike:.4f} to {loglike_t:
            tau, decode, loglike = tau_t, decode_t, loglike_t
    print(f"loglikelihood {loglike_t:.4f}, message is {best_msg}")
    return best_msg
```

In [405…

```
def identityTau(count):
    tau = {}
    for k in list(count.keys()):
        tau[k] = k
    return tau
```

In [406…

```
MetropolisHastings(secret2, mu, A, identityTau(count), 10000, chars2index)
```

```
100%|██████████| 10000/10000 [00:17<00:00, 557.96it/s]
loglikelihood -6181.8521, message is V2245UQ184*67/86/*451L/4031L/2/X41298J405104Z10
5Y4V337L78J410478=7870/*7Q12*(4Q105/Q1076*(405/4Q9*04/W16049=4*67/86/*(41E18X98*40
5/4Z396/**49=41812)*7*418X4/80/3*498405/48/:4Z396/**49=405/4780/J31079849=4U8C89:8(4
78=7870/2)4*Q122(4"U180707/*Y4VE18X9878J405/46986/Z079849=461U*/(4Q105/Q1076*4*//C*4
21:(4051047*(405/4Z39Z/30)469QQ9840941224U8C89:8(478=7870/2)4*Q122(4/2/Q/80*Y4T84189
05/34=93Q4EU041298J405/4*1Q/4Z10549=43/=2/60798405/4905/34*67/86/*451L/4Z396//X/XY4O
5/84F/:0984/8U86710/X405/421:49=4J31L70)45/4X7X48904*1)40510405/4*U8493405/4/1305451
X414Z39Z/30)49=41003160798j45/4*17X4051041224E9X7/*4=39Q405/4213J/*0409405/4*Q122/*0
451L/405/4Z39Z/30)49=4100316078J498/418905/3(4051047*(42/1L78J41*7X/405/4"U/*079849=
405/461U*/49=405/4Q9L/Q/8049=405/4E9X7/*(45/4/WZ3/**/X405/4Z39Z/30)469QQ9840941224E9
X7/*4=39Q405/478=7870/2)4213J/409405/478=7870/2)4*Q122Y4R5/4*1Q/47*4X98/4E)405/4810U
3124*67/86/*G42/1L78J41*7X/405/4"U/*079849=461U*/(405/)4*//C4=93421:*Y4D7*093)4*018X
*498405/4*1Q/4Z105Y4V8X47=457*093)451*4=93470*49Em/60405/4*0UX)49=405/4Q9L/Q/8049=40
5/4810798*418X49=45UQ1870)418X4890405/481331079849=4/Z7*9X/*478405/427L/*49=478X7L7X
U12*(4704099(4*/0078J41*7X/405/46986/Z079849=461U*/(4*59U2X4*//C405/421:*469QQ984094
122405/478*/Z131E2)4780/36988/60/X478=7870/*7Q124/2/Q/80*49=4=3//4:722Y
```

Out[406]:
```
'V2245UQ184*67/86/*451L/4031L/2/X41298J405104Z105Y4V337L78J410478=7870/*7Q12*(4Q10
5/Q1076*(405/4Q9*04/W16049=4*67/86/*(41E18X98*405/4Z396/**49=41812)*7*418X4/80/3*4
98405/48/:4Z396/**49=405/4780/J31079849=4U8C89:8(478=7870/2)4*Q122(4"U180707/*Y4VE
18X9878J405/46986/Z079849=461U*/(4Q105/Q1076*4*//C*421:(4051047*(405/4Z39Z/30)469Q
Q9840941224U8C89:8(478=7870/2)4*Q122(4/2/Q/80*Y4T8418905/34=93Q4EU041298J405/4*1
Q/4Z10549=43/=2/60798405/4905/34*67/86/*451L/4Z396//X/XY4O5/84F/:0984/8U86710/X40
5/421:49=4J31L70)45/4X7X48904*1)40510405/4*U8493405/4/1305451X414Z39Z/30)49=410031
60798j45/4*17X4051041224E9X7/*4=39Q405/4213J/*0409405/4*Q122/*0451L/405/4Z39Z/30)4
9=4100316078J498/418905/3(4051047*(42/1L78J41*7X/405/4"U/*079849=405/461U*/49=40
5/4Q9L/Q/8049=405/4E9X7/*(45/4/WZ3/**/X405/4Z39Z/30)469QQ9840941224E9X7/*4=39Q40
5/478=7870/2)4213J/409405/478=7870/2)4*Q122Y4R5/4*1Q/47*4X98/4E)405/4810U3124*67/8
6/*G42/1L78J41*7X/405/4"U/*079849=461U*/(405/)4*//C4=93421:*Y4D7*093)4*018X*49840
5/4*1Q/4Z105Y4V8X47=457*093)451*4=93470*49Em/60405/4*0UX)49=405/4Q9L/Q/8049=405/48
10798*418X49=45UQ1870)418X4890405/481331079849=4/Z7*9X/*478405/427L/*49=478X7L7XU1
2*(4704099(4*/0078J41*7X/405/46986/Z079849=461U*/(4*59U2X4*//C405/421:*469QQ984094
122405/478*/Z131E2)4780/36988/60/X478=7870/*7Q124/2/Q/80*49=4=3//4:722Y'
```

Warning: this (silly) code doesn't work with `secret.txt`, only with `secret2.txt`!

7. To speed up the calculations, we'll start from a decoding function taking the frequencies of occurrence of the letters (i.e. the most frequent letter of the coded message will be decoded to the most frequent letter observed in Tolstoy's novel; then the second most frequent letter of the coded message will be decoded to the second most frequent letter observed in the novel; and so on...). You can use the following code to build such a decoding function, named here `tau_init`.

In [407…
```
# WARNING: mu = proba of the init characters, not the stationary proba.
# => find frequent characters = sort (count) !
# stationary character distribution
freqKeys = np.array(list(count.keys()))
freqVal = np.array(list(count.values()))
# character index: +freq => - freq in references
rankFreq = (-freqVal).argsort()
# secret message analysis: index of the most frequent => least frequent
keys = np.array(list(set(secret2))) # all characters of secret2
rankSecret = np.argsort(-np.array([secret2.count(c) for c in keys]))
# ATTENTION: 37 keys in secret, 77 in general...
# Only the most frequent characters of mu are encoded,
# so much the worse for the others.
# alignment of + freq in mu VS + freq in secret

tau_init = dict([(keys[rankSecret[i]], freqKeys[rankFreq[i]]) for i in range(len(ra
```

In [408…
```
MetropolisHastings(secret2, mu, A, tau_init, 50000, chars2index)
```

```
100%|██████████| 50000/50000 [01:42<00:00, 488.25it/s]
```

```
loglikelihood -3194.5644, message is All human sciences have traveled along that pat
h. Arriving at infinitesimals, mathematics, the most exact of sciences, abandons the
process of analysis and enters on the new process of the integration of unknown, inf
initely small, Buantities. Abandoning the conception of cause, mathematics seeks law
, that is, the property common to all unknown, infinitely small, elements. In anothe
r form but along the same path of reflection the other sciences have proceeded. Then
Hewton enunciated the law of gravity he did not say that the sun or the earth had a
property of attraction' he said that all bodies from the largest to the smallest hav
e the property of attracting one another, that is, leaving aside the Buestion of the
cause of the movement of the bodies, he expressed the property common to all bodies
from the infinitely large to the infinitely small. "he same is done by the natural s
ciences! leaving aside the Buestion of cause, they seek for laws. Pistory stands on
the same path. And if history has for its ob-ect the study of the movement of the na
tions and of humanity and not the narration of episodes in the lives of individuals,
it too, setting aside the conception of cause, should seek the laws common to all th
e inseparably interconnected infinitesimal elements of free will.
```

Out[408]:
```
'All human sciences have traveled along that path. Arriving at infinitesimals, mat
hematics, the most exact of sciences, abandons the process of analysis and enters
on the new process of the integration of unknown, infinitely small, Buantities. Ab
andoning the conception of cause, mathematics seeks law, that is, the property com
mon to all unknown, infinitely small, elements. In another form but along the same
path of reflection the other sciences have proceeded. Then Hewton enunciated the l
aw of gravity he did not say that the sun or the earth had a property of attractio
n\' he said that all bodies from the largest to the smallest have the property of
attracting one another, that is, leaving aside the Buestion of the cause of the mo
vement of the bodies, he expressed the property common to all bodies from the infi
nitely large to the infinitely small. "he same is done by the natural sciences! le
aving aside the Buestion of cause, they seek for laws. Pistory stands on the same
path. And if history has for its ob-ect the study of the movement of the nations a
nd of humanity and not the narration of episodes in the lives of individuals, it t
oo, setting aside the conception of cause, should seek the laws common to all the
inseparably interconnected infinitesimal elements of free will.'
```

In [409…
```
print(tau_init)
```

```
{'T': ' ', '(': 'e', 'g': 't', 'I': 'a', 'V': 'o', ')': 'n', ':': 'i', '2': 'h', 'S'
: 's', '"': 'r', 'm': 'd', 'B': 'l', '9': 'u', '0': 'c', '3': 'm', 'h': 'w', 'K': 'f
', 'x': 'g', '=': 'y', 'X': ',', 'v': 'p', 'a': 'b', 'D': '.', 'q': 'v', 'P': 'k', '
,': '"', 'd': "'", 'M': 'I', 'o': '-', '?': 'T', '4': 'A', ' ': 'P', 'R': '!', 'C':
'x', '7': 'H', 'J': 'B'}
```

The message will normally be intelligible when you reach a log-likelihood of more than -3090. However, there are usually still some errors in the translation... Do you notice some specific errors and could you explain why they are being observed?

In [410…
```
# some letters may be close to equivalent to the algo (almost equally often followi
# letters such as 'N' for 'Newton' are not available within tau as they are not in
# see below: i added missing keys to tau, which ought to allow for perfect decoding
```

In [411…

```
freqKeys = np.array(list(count.keys()))
freqVal = np.array(list(count.values()))
rankFreq = (-freqVal).argsort()

s2keys = np.array(list(set(secret2)))
missingkeys = [k for k in list(set(freqKeys) - set(s2keys))]

keys = np.array(list(set(secret2)))
rankSecret = np.argsort(-np.array([secret2.count(c) for c in keys]))

tau_init = dict([
    (keys[rankSecret[i]], freqKeys[rankFreq[i]])
    if i < len(rankSecret) else
    (missingkeys[i - len(rankSecret)], freqKeys[rankFreq[i]]) # add missing keys to
    for i in range(len(rankFreq))
    ])
MetropolisHastings(secret2, mu, A, tau_init, 50000, chars2index)
```

```
100%|██████████| 50000/50000 [01:56<00:00, 429.89it/s]
loglikelihood -3324.6968, message is All human sciences have traveled along that pat
h. Arriving at infinitesimals, mathematics, the most exact of sciences, abandons the
process of analysis and enters on the new process of the integration of unknown, inf
initely small, quantities. Abandoning the conception of cause, mathematics seeks law
, that is, the property common to all unknown, infinitely small, elements. On anothe
r form but along the same path of reflection the other sciences have proceeded. Then
Hewton enunciated the law of gravity he did not say that the sun or the earth had a
property of attraction? he said that all bodies from the largest to the smallest hav
e the property of attracting one another, that is, leaving aside the question of the
cause of the movement of the bodies, he expressed the property common to all bodies
from the infinitely large to the infinitely small. She same is done by the natural s
ciences! leaving aside the question of cause, they seek for laws. Mistory stands on
the same path. And if history has for its object the study of the movement of the na
tions and of humanity and not the narration of episodes in the lives of individuals,
it too, setting aside the conception of cause, should seek the laws common to all th
e inseparably interconnected infinitesimal elements of free will.
```

Out[411]:

```
'All human sciences have traveled along that path. Arriving at infinitesimals, mat
hematics, the most exact of sciences, abandons the process of analysis and enters
on the new process of the integration of unknown, infinitely small, quantities. Ab
andoning the conception of cause, mathematics seeks law, that is, the property com
mon to all unknown, infinitely small, elements. On another form but along the same
path of reflection the other sciences have proceeded. Then Hewton enunciated the l
aw of gravity he did not say that the sun or the earth had a property of attractio
n? he said that all bodies from the largest to the smallest have the property of a
ttracting one another, that is, leaving aside the question of the cause of the mov
ement of the bodies, he expressed the property common to all bodies from the infin
itely large to the infinitely small. She same is done by the natural sciences! lea
ving aside the question of cause, they seek for laws. Mistory stands on the same p
ath. And if history has for its object the study of the movement of the nations an
d of humanity and not the narration of episodes in the lives of individuals, it to
o, setting aside the conception of cause, should seek the laws common to all the i
nseparably interconnected infinitesimal elements of free will.'
```