This documentation provides an illustration (and also a proof) of Metropolis algorithm of sampling. The proof follows Metropolis et al (1953), but is modified and generalized in its morden face.

Notation 1. *[Discrete Version]*[1] *Let $\mathcal{X}$ state-space of a given stochastic system. Suppose $\mathcal{X}$ is discrete or has been discretized, within which state is denoted by x_r, or simply r. Let $p(r)$ denotes the target distribution to be mimicked. Let $q(r \to s)$ (or $q(s|r)$ by statistics) the proposed transition-distribution of Markov process from state r to s. Suppose, for any $r, s \in \mathcal{X}$, $q(r \to s) \neq 0$ and $p(r) \neq 0$ (thus they are positive). That is, in $\mathcal{X}$ any r and s are "connected". And for Metropolis algorithm, $q(r \to s) = q(s \to r)$ for $\forall r, s \in \mathcal{X}$ is supposed.*

Algorithm 1

[Metropolis]

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random

class MetropolisSampler:
    """
    Args:
        iterations: int
        initialize_state: (None -> State)
        markov_process: (State -> State)
        burn_in: int

    Attributes:
        accept_ratio: float
            Generated only after calling MetropolisSampler.

    Methods:
        sampling:
            Do the sampling by Metropolis algorithm.

    Remarks:
        The "State" can be any abstract class.
    """


    def __init__(self, iterations, initialize_state, markov_process, burn_in):

        self.iterations = iterations
        self.initialize_state = initialize_state
        self.markov_process = markov_process
        self.burn_in = burn_in


    def sampling(self, target_distribution):
        """
        Do the sampling.

        Args:
            target_distribution: (State -> float)
```

1. [Continuum Version] Let $\mathcal{X}$ state-space of a given stochastic system. Let $p(x)$ denotes the target distribution to be mimicked. Let $q(x \to y)$ (or $q(y|x)$ by statistics) the proposed transition-distribution of Markov process from state x to y. Suppose, for any $x, y \in \mathcal{X}$, $q(x \to y) \neq 0$ and $p(x) \neq 0$ (thus they are positive). That is, in $\mathcal{X}$ any x and y are "connected". And for Metropolis algorithm, $q(x \to y) = q(y \to x)$ for $\forall x, y \in \mathcal{X}$ is supposed.

```
Returns:
    list of State, with length being iterations - burn_in.
"""

init_state = self.initialize_state()
assert target_distribution(init_state) > 0

chain = [init_state]
accepted = 0

for i in range(self.iterations):

    next_state = self.markov_process(init_state)

    alpha = target_distribution(next_state) / target_distribution(init_state)
    u = random.uniform(0, 1)

    if alpha > u:

        accepted += 1
        chain.append(next_state)

        init_state = next_state.copy()

    else:

        chain.append(init_state)

self.accept_ratio = accepted / self.iterations
print('accept-ratio = {0}'.format(self.accept_ratio))

return chain[self.burn_in:]
```

Lemma 2. *Define, for $\forall r \in \mathcal{X}$, $J(r) := \{\forall s \in \mathcal{X}: p(r) \geqslant p(s)\}$. Let $h_i(r)$ the distribution of $\forall r \in \mathcal{X}$ at the ith Markov epoch. For any given h_i, we have*

$$h_{i+1}(r) - h_i(r) = \sum_{s \in \mathcal{X}} \left\{ q(s \to r) \left[\frac{\delta_{s \in J(r)}}{p(r)} + \frac{\delta_{s \notin J(r)}}{p(s)} \right] \right\} \{h_i(s)\, p(r) - h_i(r)\, p(s)\}.$$

Note that the first $\{...\}$ are symmetric for s and r, while the second is anti-symmtric.

Proof. Directly from Metropolis algorithm, we have,

$$\begin{aligned} h_{i+1}(r) - h_i(r) &= \sum_{s \in J(r)} h_i(s)\, q(s \to r) + \sum_{s \notin J(r)} h_i(s)\, q(s \to r) \frac{p(r)}{p(s)} \\ &- \sum_{s \in J(r)} h_i(r)\, q(r \to s) \frac{p(s)}{p(r)} + \sum_{s \notin J(r)} h_i(r)\, q(r \to s). \end{aligned}$$

Since, in Metropolis algorithm, $q(s \to r) = q(r \to s)$ is supposed,

$$\begin{aligned} h_{i+1}(r) - h_i(r) &= \sum_{s \in J(r)} h_i(s)\, q(s \to r) + \sum_{s \notin J(r)} h_i(s)\, q(s \to r) \frac{p(r)}{p(s)} \\ &- \sum_{s \in J(r)} h_i(r)\, q(s \to r) \frac{p(s)}{p(r)} + \sum_{s \notin J(r)} h_i(r)\, q(s \to r); \end{aligned}$$

then by direct simplification, we reach

$$h_{i+1}(r)-h_i(r)=\sum_{s\in\mathcal{X}}\left\{q(s\to r)\left[\frac{\delta_{s\in J(r)}}{p(r)}+\frac{\delta_{s\notin J(r)}}{p(s)}\right]\right\}\{h_i(s)\,p(r)-h_i(r)\,p(s)\}. \qquad \square$$

Corollary 3. *If $h_i=p$, then $h_{i+1}=p$.*

Theorem 4. *[Metropolis] Samples generated by algorithm 1 approximately obeys the target distribution $p(x)$. That is, algorithm 1 creates a sampler of $p(x)$.*

Proof. [Intuitive]

Let $\forall r_1\in\mathcal{X}$ given. We have a Markov chain generated by Metropolis algorithm starting at r_1, say $\{r_1,r_2,...,r_N\}$. We want to prove that $\{r_1,r_2,...,r_N\}$ approximately obeys distribution $p(x)$. This proof has three steps.

S1) Define h by $\{r_1, r_2, ..., r_{N-1}\}\sim h$, and h' by $\{r_2, r_3, ..., r_N\}\sim h'$. That is, the histogram of $\{r_1,r_2,...,r_{N-1}\}$, after normalization, can be fitted by h, and the same for h'. We will proof first that $h=h'+o(1)$ on $\mathcal{X}$ as $N\to+\infty$. Indeed, if N is large enough, $\{r_1, r_2, ..., r_{N-1}\}\sim h$, since dropping a single element will affect little on the fitting of histogram (will be reviewed later in remark 5). So, $h=h'+o(1)$ on $\mathcal{X}$ as $N\to+\infty$.

S2) Next is to proof that $h'=h\Rightarrow h=p$ on $\mathcal{X}$. (This proof temporally employs the Perron–Frobenius theorem.)

S2.1) Indeed, we shall not forget that $\{r_1, r_2, ..., r_N\}$ are generated by a Markov chain obeying Metropolis algorithm. Thus, the Metropolis algorithm as a Markov process brings $r_i\to r_{i+1}$ for $\forall i=1,...,N-1$, s.t. $\{r_1,r_2,...,r_{N-1}\}\to\{r_2,r_3,...,r_N\}$. That is to say, h' can be regarded as being generated by Metropolis algorithm from h. So, h and h' are related by lemma 2.

S2.2) Since h and h' are related so, as in the proof of lemma 2 before inserting the condition $q(s\to r)=q(r\to s)$,

$$\begin{aligned}
h'(r) &= h(r)\\
&+ \sum_{s\in J(r)} h(s)\,q(s\to r)+\sum_{s\notin J(r)} h(s)\,q(s\to r)\,\frac{p(r)}{p(s)}\\
&- \sum_{s\in J(r)} h(r)\,q(r\to s)\,\frac{p(s)}{p(r)}-\sum_{s\notin J(r)} h(r)\,q(r\to s)\\
&:= \sum_{s\in\mathcal{X}} K(r,s)\,h(s),
\end{aligned}$$

where

$$\begin{aligned}
K(r,s) &= \delta_{s\in J(r)}\,q(s\to r)+\delta_{s\notin J(r)}\,q(s\to r)\,\frac{p(r)}{p(s)}\\
&+ \delta_{r,s}\times\left[1-\sum_{t\in J(r)} q(r\to t)\,\frac{p(t)}{p(r)}-\sum_{t\notin J(r)} q(r\to t)\right].
\end{aligned}$$

Or in matrix form $\boldsymbol{h}=K\boldsymbol{h}$. That is, $\boldsymbol{h}$ is the eigen-vector of K with eigen-value 1.

Since, for $\forall t\in J(r)$, $0<p(t)/p(r)\leqslant 1$, we have

$$\begin{aligned}
&1-\sum_{t\in J(r)} q(r\to t)\,\frac{p(t)}{p(r)}-\sum_{t\notin J(r)} q(r\to t)\\
\geqslant\ &1-\sum_{t\in J(r)} q(r\to t)-\sum_{t\notin J(r)} q(r\to t)\\
=\ &1-1=0;
\end{aligned}$$

also since, for $\forall r, s \in \mathcal{X}$, both $q(s \to r)$ and $p(r)$ are positive, we thus conclude that, for $\forall r, s \in \mathcal{X}$

$$K(r, s) > 0,$$

that is, K is a positive real square matrix. Recall Perron–Frobenius theorem for positive matrices states that "given positive matrix[2] A, the Perron–Frobenius eigenvector[3] is the only (up to multiplication by constant) non-negative eigenvector for A". As in corollary 3, by letting $\boldsymbol{h} = \boldsymbol{p}$, we have gained an eigen-value of K such that all components are real and non-negative[4]. So, as a distribution (thus all components have to be real and non-negative), $\boldsymbol{h}$ has no choice but be $\boldsymbol{p}$, which is what we want to prove, that is $h' = h \Rightarrow h = p$ on $\mathcal{X}$. □

Remark 5. [Burn-in]

Within this intuitive proof, we have to ensure that dropping r_1 from $\{r_1, r_2, ..., r_{N-1}\}$ affects little on the fitting of h. This does affect h if r_1 happens to be the state where $p(r) \ll 1$, so that causes a "Poisson error". After all, r_1 is initialized randomly. For this reason, "burn-in" mechanism is introduced in. While adding r_N will affects little on $h' - h$, since the probability of being in the "important region" of p for r_N is large, after all, r_N is not initialized randomly as r_1.

2. I.e. positive real squre matrix.

3. I.e. that unique eigen-vector (up to multiplication by constant) of the Perron-Frobenius eigen-value, which is defined herein.

4. Recall that this corollary requires the Metropolis condition $q(s \to r) = q(r \to s)$ for $\forall r, s \in \mathcal{X}$. This is where this condition is employed in the proof of Metropolis algorithm.