Markov chains :

The type of state where the probability of coming back to itself is less than 1 or not possible is the **transient state**

The type of state where the probability of coming back to itself is 1, this is called **a recurring state.**

The Marcov chain where few states are such it is unreachable by other states or we cannot go back to that state is called **reducible** Markov chain, they can be split to further markov chains.

Unreachable and reachable states who communicate with eachother can be further divided into classes.

The Markov chain where we can every state can communicate with another states is called the **irreducible** Markov chain

Transition matrix :

Probability of reaching reach state 2 from state 1 in 2 steps can be defined as below :

$P_{ij} = A_{ij}^{2}$

For n steps it will be :

$P_{ij} = A_{ij}^{n}$

Stationary distribution : It is where the probability of states becomes same after multiplying the matrix by itself certain number of times. Conditions to satisfy this :
-irreducibility
-aperiodicity

Periodicity : without self loop 123123…
Aperiodicity : atleast one self loop will make it aperiodic 123312331233….

Markov chain reversible : need to check this and periodic states

Accept - Reject sampler :

POISSON regression

finally obtaining the following loss function

$$loss = \frac{1}{m}\sum_{i=1}^{m} \hat{y}^{(i)} - \log(\hat{y}^{(i)})y^{(i)} \quad (6)$$

X - input array
W - coefficient
B - coefficient
Y -

```
def loss(x, y, w, b):
  y_hat = np.exp(x @ w + b)
  error = (y_hat - np.log(y_hat) * y).mean()
  return error
```

Log Loss

$$\text{Log-Loss} = \sum_{i=0}^{n} -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

```
sigmoid = lambda yhat: 1/(1+np.exp(-yhat))
```

```
loss = lambda y, sigmoid:
-(y*np.log(sigmoid)+(1-y)*np.log(1-sigmoid)).mean()
```

`Linear Regression`

$$E = \frac{1}{n} \cdot \sum_{i=0}^{n} (y_i - \overline{y_i})^2$$

```python
def loss_function(m, b, points):
    total_error = 0
    for i in range(len(points)):
        x = points.iloc[i, 0]
        y = points.iloc[i, 1]
        total_error += (y - (m * x + b)) ** 2
    return total_error / float(len(points))
```