In [ ]:

```
import numpy as np
import math
import numpy.random as rnd


def metropolis_hastings(g,N,m, burn_in = None):
    burn_in = burn_in if burn_in is not None else N // 10
    #X and Y are random integers from 0 to m
    U = np.random.uniform(0,1,N + burn_in)
    X = np.zeros(N + burn_in,dtype = int)
    X[0] = 3
    #prob_of_sampling = np.minimum(g(X)/g(Y),np.ones(N))
    #sample X with probability prob_of_sampling
    for i in range(1,N+burn_in):
        x = X[i-1]
        y = np.random.randint(0,m+1)

        if U[i] <= min(g(y) / g(x),1) :
            X[i] = y
        else:
            X[i] = x

    # X = np.where(U<prob_of_sampling,X,Y)
    X = X[burn_in:]
    return X

def truncated_poisson_samples(lam, low, high, size=1, numvars = 1):
    samples = []
    while len(samples) < size:
        x = np.random.poisson(lam, numvars)
        if low <= x <= high:
            samples.append(x)
    return np.array(samples)

def y_sampling_function(m):
        def y_sampling():
            y1 = np.random.randint(0,m+1)
            y2 = np.random.randint(0,m+1-y1)
            #create y as (y1,y2) or (y2,y1) with equal probability
            return (y1,y2) if np.random.uniform(0,1) < 0.5 else (y2,y1)
        return y_sampling


def metropolis_hastings_joint(g_joint,N, burn_in = None, y_sampling_func = y_sam
    burn_in = burn_in if burn_in is not None else N

    #X and Y are random integers from 0 to m
    U = np.random.uniform(0,1,N + burn_in)
    X = np.zeros((N + burn_in,2),dtype = float)
    X[0] = (1.0,1.0)
    #prob_of_sampling = np.minimum(g(X)/g(Y),np.ones(N))
    #sample X with probability prob_of_sampling
    for i in range(1,N+burn_in):
        x = X[i-1]
        y = y_sampling_func()
        if U[i] <= min(g_joint(y[0],y[1]) / g_joint(x[0],x[1]),1) :
            X[i] = y
        else:
            X[i] = x
    # X = np.where(U<prob_of_sampling,X,Y)
    X = X[burn_in:]
    return X

def Gibbs(As, n, x0, m):
    xs = [x0[0]]
    ys = [x0[1]]
    A1 = As[0]
    A2 = As[1]
    for k in range(1,n):
        # Generate i and sample from j
        i = xs[k-1]
        num_classes_j = int(m - i + 1)
        ps = np.zeros(num_classes_j)
        k = 0
        for j in range(num_classes_j):
            ps[j] = A2**j / math.factorial(j)
            k += A2**j / math.factorial(j)
        ps /= k
        j = rnd.choice(a=np.arange(num_classes_j), size= 1, p = ps)[0]

        ys.append(j)
        # Newest j has already been found
        num_classes_i = int(m-j + 1)
        ps = np.zeros(num_classes_i )
        k = 0
        for i in range(num_classes_i):
            ps[i] = A2**i / math.factorial(i)
            k += A2**i / math.factorial(i)
        ps /= k
        i = rnd.choice(a=np.arange(num_classes_i), size=1, p = ps)[0]
        xs.append(i)
    return np.array(xs), np.array(ys)
```