```python
# Metropolis Hasting Sampling + Extras

from scipy import stats
from tqdm import tqdm
import numpy as np
from scipy.linalg import eig
import click

class metro_hast_chain:
    def __init__(self,N,E,P,PP):
        self.chain_length = N
        self.elements = np.array(E)
        self.initial_prob = np.array(P)
        self.proposal_prob = np.array(PP)
        self.chain = []
        self.X = stats.rv_discrete(name="X",values=(self.elements,self.initial_prob))
        self.Y = stats.rv_discrete(name="Y",values=(self.elements,self.proposal_prob))
        self.M = np.asmatrix(self.initial_prob*self.proposal_prob.reshape(3,1))

    def start(self):
        s = self.X.rvs(1)-1
        self.chain.append(s)
        return s

    def advance(self):
        for i in tqdm(range(self.chain_length)):
            x0 = self.chain[i-1]
            x = self.Y.rvs(1)-1
            if x > np.random.uniform(0,1):
                self.chain.append(x)
            else:
                self.chain.append(x0)

    def stationary_dist(self):
        L,V = eig(self.M,left = True,right = False)
        l1v = L.max()
        stat_dist = V[:, 0].T/sum(V[:,0])
        return l1v,stat_dist

@click.command()
@click.option(
    '--num','-n',
    default=100,
    type = int,
    show_default=True,
    help='Number of itartations in the chain'
)
@click.option(
    '--element','-e',
    default=[1,2,3],
    type = list,
    show_default=True,
    help='List of elements/states'
)
@click.option(
    '--probability','-p',
    default=[1/2,1/3,1/6],
    type=list,
    show_default=True,
    help='Probabilities of each element/state'
)
@click.option(
    '--proposal','-pp',
    default=[.99,.009,.001],
    type = list,
    show_default=True,
    help='Probability of Proposing a state'
)

def main(num,element,probability,proposal):
    mhc3 = metro_hast_chain(num,element,probability,proposal)
    start_state = mhc3.start()
    print(start_state)
    mhc3.advance()
    eigenvalue, stationary = mhc3.stationary_dist()
    print(stationary)
    print(eigenvalue)
    chain = mhc3.chain
    print(chain)


if __name__ == "__main__":
    main()
```

```python
# Sampler definitions for problem 3 and its corresping results

from scipy import stats
import numpy as np

# Defining our discrete random variable X, using a discrete random variable
def discrete_3_sample(X_W,P_W):

    X = stats.rv_discrete(name = "X",values = (X_W, P_W))
    sample = X.rvs(1)-1
    return sample

# Sampling from the same X_W and P_W using a random uniform variable
def disc_samp(val,prob):

    u = np.random.uniform(0,1)
    for w in enumerate(val):
        u -= prob[w[0]]
        if u < 0:
            return w[1]

# For problem 3(a)
X_W = (1,2,3)
P_W = (1/2,1/3,1/6)

print(discrete_3_sample(X_W,P_W))
print(disc_samp(X_W,P_W))
```

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as manimation; manimation.writers.list()
from tqdm import tqdm
import click
class IsingLattice:
    def __init__(self,size):
        self.size = size
        self.system = self._build_system()

    @property
    def sqr_size(self):
        return(self.size, self.size)

    def _build_system(self):

        system = np.random.choice([-1,1], self.sqr_size)

        return system

    def _bc(self,i):
        if i >= self.size:
            return 0
        if i < 0:
            return self.size - 1
        else:
            return i
    def node_diff(self,M,N):
        return self.system[M,N] * (self.system[self._bc(N - 1), M] + self.system[self._bc(N + 1), M]
        + self.system[N, self._bc(M - 1)] + self.system[N, self._bc(M + 1)])


    def config_change(self):
        M,N = np.random.randint(0,self.size,2)

        if self.node_diff(M,N) <= 0:
            self.system[M,N] *= -1
        elif np.exp(-2*self.node_diff(M,N)) < np.random.uniform(0,1):
            self.system[M,N] *= 1

    def same(self,N,M):
        if self.node_diff(N,M) == 4:
            return 1
        else:
            return 0




def run(lattice,burn_in,iterations,video=True):
    for b in tqdm(range(burn_in)):
        lattice.config_change()

    FFMpegWriter = manimation.writers['ffmpeg']
    writer = FFMpegWriter(fps=10)

    fig = plt.figure()

    with writer.saving(fig, "ising.mp4", 50):
        for i in tqdm(range(iterations)):
            lattice.config_change()

            if video and i % 10000 == 0:
                img = plt.imshow(lattice.system,cmap="jet",interpolation="nearest")
                writer.grab_frame()
                img.remove()

    plt.close('all')

    efw = 0
    for i in range(lattice.size):
        for j in range(lattice.size):
            efw += lattice.same(i,j)

    print(lattice.system)
    print(efw)

@click.command()
@click.option(
    '--size','-s',
    default=100,
    show_default=True,
    help='Number of sites, M, in the MxM lattice'
)
@click.option(
    '--burn_in','-b',
    default=1000,
    type=int,
    show_default=True,
    help='Number of burn in iterations to run'
)
@click.option(
    '--iterations','-i',
    default=4_000_000,
    type=int,
    show_default=True,
    help='Number of iterations to run the simulation for'
)
@click.option(
    '--video',
    default = True,
    is_flag=False,
    help='Record a video of the simulation progression'
)
def main(size,burn_in,iterations,video):
    lattice = IsingLattice(size)
    run(lattice,burn_in,iterations,video)

if __name__ == "__main__":
    plt.ion()
    main()
```