```
import numpy as np

# Generate patterns
def gen_patterns(p, N):
    mat = np.random.randint(low = 0, high = 2, size = (p,N))
    mat[mat == 0] = -1
    return mat

# Hebb's rule
def hebbs(P):
    N = P.shape[1]
    W = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
            if (j != i):
                W[i,j] = 1/N * np.dot(P[:,i],P[:,j])
    return W

# Asynchronous update
def p_(b):
    beta = 2
    return 1/(1+np.exp(-2*beta*b))
def stochastic_asynch_update(W, s_in, t):
    t = t%W.shape[0]
    b = W[t,:].dot(s_in)
    s_out = np.copy(s_in)
    r = np.random.rand(1)[0]
    prob = p_(b)
    if (r < prob):
        s_out[t] = 1
    else:
        s_out[t] = -1
    return s_out
def calc_order_param_for_t(s, x_mu):
    return s.dot(x_mu) / (x_mu.shape[0])
def m_experiment(p):
    T = 2*10**5
    P = gen_patterns(p,200)
    W = hebbs(P)
    s_old = P[0,:]
    m_mu = 0
    for it in range(T):
        s_new = stochastic_asynch_update(W,s_old,it)
        m_mu += calc_order_param_for_t(s_new, P[0,:])
        s_old = s_new
    return m_mu / T
def m_average(p,times):
    m_tot = 0
    for ix in range(times):
        m_tot += m_experiment(p)
    return m_tot / times

# Computation for p = 7
m_avg_7 = m_average(7, 100)
```

```
print(m_avg_7)
# Computation for p = 45
m_avg_45 = m_average(45, 100)
print(m_avg_45)
```