```
import numpy as np
import scipy.integrate as integrate
import math

## Functions to estimate the one-step error probability
def gen_patterns(p, N):
    mat = np.random.randint(low = 0, high = 2, size = (p,N))
    mat[mat == 0] = -1
    return mat
def choose_bit(N):
    return np.random.randint(low = 0, high = N)
def choose_pattern(p):
    return np.random.randint(low = 0, high = p)
def asynch_update1(P, bit, pattern): # For w_ii = 0
    c = 0
    for j in range(np.shape(P)[1]):
        for mu in range(np.shape(P)[0]):
            if (j != bit and mu != pattern ):
                c += P[mu, bit] * P[mu, j] * P[pattern, j] *
P[pattern, bit]
    return (-1/np.shape(P)[1]) * c
def isIncorrect(c):
    if c > 1: return 1
    else: return 0
def one_step_error1(n_trials, N, p): # For w_ii = 0
    c_sum = 0
    for it in range(n_trials):
        P = gen_patterns(p,N)
        bit = choose_bit(N)
        pattern = choose_pattern(p)
        c = asynch_update1(P, bit, pattern)
        c_sum += isIncorrect(c)
    print(c_sum)
    return c_sum/n_trials

# Functions for w_ii != 0
def hebbs(P):
    N = P.shape[1]
    W = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
                W[i,j] = 1/N * np.dot(P[:,i],P[:,j])
    return W
def asynch_update2(W, bit, s_vec): # For w_ii != 0
    return np.sign(W[bit,:].dot(s_vec) )
def one_step_error2(n_trials, N, p): # For w_ii != 0
    n_incorrect = 0
    for it in range(n_trials):
        P = gen_patterns(p,N)
        W = hebbs(P)
        bit = choose_bit(N)
        pattern = choose_pattern(p)
        s_m = asynch_update2(W, bit, P[pattern])
        if (s_m != P[pattern,bit] ):
```

```
            n_incorrect += 1
    print(n_incorrect)
    return n_incorrect/n_trials

## Functions for computing theoretical one-step error
def e(x):
    return np.exp(-(x**2))
def erf(z):
    temp = integrate.quad(e,0,z)[0]
    temp = temp * (2/np.sqrt(math.pi))
    return temp
def error_prob(N,p):
    return 1/2 * (1-erf(np.sqrt(N/(2*p))))

## Function for one-step error for all p when w_ii = 0
def one_step_error_pvec1(p):
    result = np.zeros((2,np.shape(p)[0]))
    for it in range(np.shape(p)[0]):
        result[0,it] = error_prob(120,p[it])
        result[1,it] = one_step_error1(10**5, 120, p[it])
    return result

## Function for one-step error for all p when w_ii != 0
def one_step_error_pvec2(p):
    result = np.zeros((2,np.shape(p)[0]))
    for it in range(np.shape(p)[0]):
        result[0,it] = error_prob(120,p[it])
        result[1,it] = one_step_error2(10**5, 120, p[it])
    return result

## Computing estimated one-step error for all values of p together
with
## theoretical one-step error prob.
p = np.array([12, 24, 48, 70, 100, 120])
result_zero_diag = one_step_error_pvec1(p)
print(result_zero_diag)
result_non_zero_diag = one_step_error_pvec2(p)
print(result_non_zero_diag)
```