```python
import numpy as np
import cvxpy as cp

# Sigmoid function
def sigmoid(x):
    z = np.exp(-x)
    sig = 1 / (1 + z)
    return sig

# Generates a random binary signal given a
# h in U(-1,1) (uniform distribution) and adjacency matrix chi
def generate_signal(h, chi):
    P = (chi@h)
    X = [sigmoid(p) for p in P]
    for i in range(len(X)):
        test = np.random.uniform(0,1)
        if test < X[i]:
            X[i] = 1
        else:
            X[i] = 0
    return X

# Used in optimization program for creating expressions
def term_given_X(y,w,X):
    return y.T@X@w - cp.sum(cp.logistic(X@w))

def term_given_w(y,w,X):
    return y.T@X@w - cp.sum(cp.logistic(X@w))

# Given a noisy signal (Num_Nodes by Num_signals) and sparcity constraint
# Return estimated adjacency matrix
def binary_graph_learning(Y_noisy, alpha):
    # Define variables
    Y = Y_noisy
    N = Y_noisy.shape[0]
    A_var = cp.Variable((N,N), symmetric = True)
    Lap = cp.diag(A_var@np.ones((N,1))) - A_var
    h_var = cp.Variable((N,1))
    h_init = np.random.uniform(-1,1,N)
    print("CP variables built.")
    # Adjacency Matrix Constraints
    constraints = [cp.diag(A_var) == 0,
                    A_var - cp.diag(cp.diag(A_var)) >= 0,
                  sum(A_var@np.ones(N)) == N]
    A_obj = cp.Maximize(sum([term_given_w(y,h_init,A_var) for y in Y.T]) - alpha*cp.norm(Lap, 'fro'))

    P_init = cp.Problem(A_obj, constraints)
    P_init.solve()
    A_init = A_var.value
    A_vals = [A_init]
    h_vals = [h_init]
    for i in range(10):
        A_cur = cp.Variable((N,N), symmetric = True)
        Lap = cp.diag(A_cur@np.ones((N,1))) - A_cur
        h_cur = cp.Variable((N,1))
        constraints = [cp.diag(A_cur) == 0,
                    A_cur - cp.diag(cp.diag(A_cur)) >= 0]
        hconstraints = [cp.sum(h_cur) == 0,
                       cp.max(h_cur) <= 1,
                       cp.min(h_cur) >= -1]
        h_obj = cp.Maximize(sum([term_given_X(y,h_cur,A_vals[len(A_vals)-1]) for y in Y.T]))
        P_cur = cp.Problem(h_obj, hconstraints)
        P_cur.solve()
        h_vals.append(h_cur.value)
        A_obj = cp.Maximize(sum([term_given_w(y,h_vals[len(h_vals)-1],A_cur) for y in
        Y.T]) - alpha*cp.norm(Lap, 'fro'))
        P_cur2 = cp.Problem(A_obj, constraints)
        P_cur2.solve()
        A_vals.append(A_cur.value)
    return A_vals
```