COMP0080 - Graphical Models
Assignment 3
Due: January 15, 2019

UCL
Machine Learning MSc
Computational Statistics and Machine Learning MSc

Author: Dorota Jagnesakova
Student ID: 16079098
E-mail: dorota.jagnesakova.19@ucl.ac.uk
Questions: 2

Author: Agnieszka Dobrowolska
Student ID: 16034489
E-mail: agnieszka.dobrowolska.16@ucl.ac.uk
Questions: 2

Author: Oliver Slumbers
Student ID: 19027699
E-mail: oliver.slumbers.19@ucl.ac.uk
Questions: 1

Author: Tom Grigg
Student ID: 19151291
E-mail: tom.grigg.19@ucl.ac.uk
Questions: 1

1 LDPC Codes

1.1 Systematic encoding matrix

Relevant code in Appendix part 1
We used the form of the generator matrix provided in the lecture notes:

$$G = \begin{bmatrix} P \\ I_K \end{bmatrix} \tag{1}$$

Therefore using the H provided our function outputs the following results:

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2}$$

$$\hat{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{3}$$

Where you can see that $\hat{H}$ is in the form $\begin{bmatrix} I_{N-K} & P \end{bmatrix}$ where N is the number of columns and K is the number of rows. We get our final generator matrix:

$$G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

Which can be verified by checking $\hat{H}Gt$ for all t (all calculations done in F_2) which in our case $= 0$ as expected.

1.2 Factor Graphs

The factor graph representing the matrix is as follows where $\boxplus$ represents computations in mod 2:

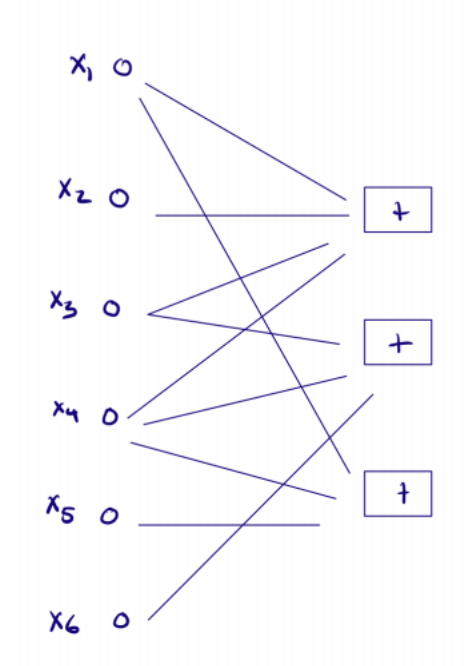

Figure 1: Factor graph representing H

The distribution corresponding to the factor graph is as follows nothing that computations are in mod 2:

$$\begin{aligned} P(x_1, ..., x_6) &= \frac{1}{Z} \prod_{m=1}^{3} f(x_{s_m}) \qquad (5) \\ &= \frac{1}{Z} \prod_{m=1}^{3} \mathbb{1}[\sum_{n \in \mathcal{N}(m)} x_n mod 2] \qquad (6) \end{aligned}$$

Updates used for the factor to variable messages as described in LDPC Codes: An Introduction by A. Shokrollahi:

$$m_{fv}^{(\ell)} = ln \frac{1 + \prod_{v' \in V_f \setminus v} tanh(\frac{m_{v'f}^{(\ell)}}{2})}{1 - \prod_{v' \in V_f \setminus v} tanh(\frac{m_{v'f}^{(\ell)}}{2})} \qquad (7)$$

Updates used for the variable to factor messages as described in LDPC Codes: An Introduction by A. Shokrollahi:

$$m_{vf}^{(\ell)} = \begin{cases} m_v & \text{if } \ell = 0 \\ m_v + \sum_{f' \in F_v \setminus f} m_{f'v}^{(\ell-1)} & \text{if } \ell \geqslant 1 \end{cases} \qquad (8)$$

Where ℓ is the round of decoding and m_v is the log-likelihood of the node v. Additionally F_v is the set of factor nodes connected to the variable node v, and V_f is the set of variable nodes connected to the factor node f.

1.3 LDPC-decoder

Relevant code in Appendix part 1

The algorithm converged in 8 iterations and the result of the decoding is the following result:

```
array([0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 0])
```

Figure 2: Decoded vector

1.4 Message

Relevant code in Appendix part 1
The original message is: 'Happy Holidays! Dmitry & David :)'.

1.5 Empirical Study

Relevant code in Appendix part 1
We carried out this study on 100 randomly generated sequences of length 252 bits for each value of p between 0.01 and 0.20 with intervals of 0.01. The plot of percentage of successful decodings is below:

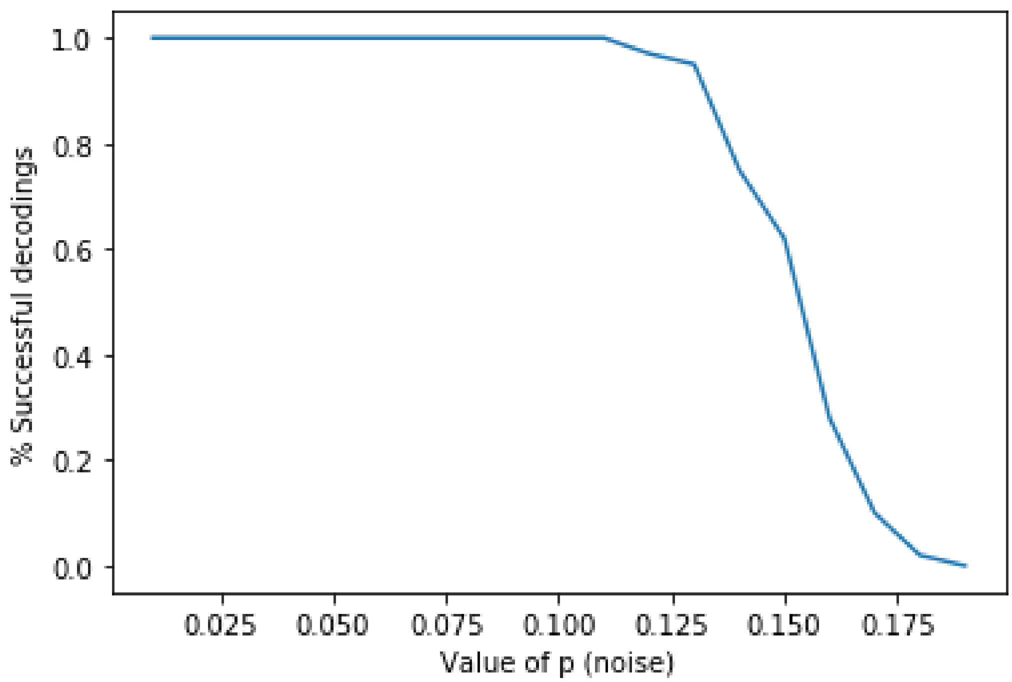

Figure 3: Successful decodings

Appendix Part 1

January 15, 2020

[1]:
```
import numpy as np
import time
from tqdm import tqdm
import matplotlib.pylab as plt
```

0.1 Question 1

[14]:
```
H1 = np.array([[1, 1, 1, 1, 0, 0], [0, 0, 1, 1, 0, 1], [1, 0, 0, 1, 1, 0]])
```

[15]:
```
def systematic_h(parity):
    for col in range(parity.shape[1] - parity.shape[0]):
        if parity[col, col] == 1: #check if pivot
            for r in range(col+1, parity.shape[0]):
                if parity[r, col] == 1:
                    parity[r, :] = parity[col, :]^parity[r, :]
                else:
                    continue
        else:
            for r in range(col+1, parity.shape[0]):
                if parity[r, col] == 1:
                    parity[[col, r]] = parity[[r, col]]

    ##backward pass
    for col in range(1, parity.shape[1] - parity.shape[0]):
        for r in range(0, col):
            if parity[r, col] == 1:
                parity[r, :] = parity[col, :]^parity[r, :]
            else:
                continue

    return parity
```

[16]:
```
# assumes parity check shape of [I P] as described in the lecture notes and␣
↪returns a stacked [P I] matrix
def generator(parity):
    K = parity.shape[0]
    N = parity.shape[1]
```

```
    I = np.identity(K)
    P = parity[:, (N-K):N]
    G = np.vstack((P, I))
    return G
```

Print outputs of the functions

[17]:
```
H_hat = systematic_h(H1)
G = generator(H_hat)
print('Generator Matrix: \n{}'.format(G))
print('H_hat Matrix: \n{}'.format(H_hat))
```

```
Generator Matrix:
[[1. 1. 0.]
 [1. 1. 1.]
 [1. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
H_hat Matrix:
[[1 0 0 1 1 0]
 [0 1 0 1 1 1]
 [0 0 1 1 0 1]]
```

Verify that G above is correct using $\hat{H}Gt = 0$ for any t

[18]:
```
verif = H_hat.dot(G)
verif%2
```

[18]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

0.2 Question 3

[19]:
```
H = np.loadtxt('H1.txt')
y = np.loadtxt('y1.txt')
G = np.loadtxt('sys_g.txt')
```

Initialisation step

[20]:
```
def initialise_vectorised(y, H, p=0.1):
    y_probs = np.zeros(len(y))
    p0 = np.log(1-p) - np.log(p) # if node = 0
    p1 = -p0 # if node = 1
    y_probs = np.where(y == 1, p1, p0)
    Msg = np.zeros((H.shape))

    #find indices where H = 1
    indices = np.where(H == 1)
```

```python
    #initial var-to-fac message
    Msg[indices[0], indices[1]] = y_probs[indices[1]]

    return y_probs, Msg
```

Factor to variable step

[21]:
```python
def factor_to_var(H, Msg):
    m,n = H.shape
    f2v = np.zeros((m,n))
    indices = np.where(H == 1)[1]
    idx_split = np.split(indices, m)
    for i in range(m):
        for j in idx_split[i]:
            result = 1
            temp_id = [x for x in idx_split[i] if x != j]
            temp_res = np.tanh(Msg[i,:][temp_id]/2)
            temp_fin = 1*np.prod(temp_res)
            f2v[i,j] = np.log((1+temp_fin)/(1-temp_fin))
    return f2v
```

Variable to factor steps

[22]:
```python
def var_to_factor_check(y_probs, f2v):
    m,n = f2v.shape
    v2f = np.zeros(n)
    v2f = y_probs + np.sum(f2v,0)

    return v2f
```

[23]:
```python
def var_to_factor_update(y_probs, msg, f2v, H):
    m,n = msg.shape
    for i in range(n):
        indices = np.where(H[:, i] == 1)
        for j in range(3):
            temp_id = [x for x in indices[0] if x != indices[0][j]]
            temp_res = f2v[:, i][temp_id]
            temp_fin = np.sum(temp_res)
            msg[indices[0][j],i] = y_probs[i] + temp_fin

    return msg
```

Full decoder function

[24]:
```python
def decoder_loop(y, H, p, maxiter=20):
    y_probs, Msg = initialise_vectorised(y, H)[:]

    for i in range(maxiter):
        fac_to_var = factor_to_var(H, Msg)
        var_to_fac = var_to_factor_check(y_probs, fac_to_var)
```

```python
        # check if successful decoding
        decoded = np.where(var_to_fac > 0, 0, 1)
        if sum(H.dot(decoded)%2)==0:
            output = 0
            return (i+1), decoded, output
            break

        # if unsuccessful update message
        Msg = var_to_factor_update(y_probs, Msg, fac_to_var, H)

    #if max iterations reached output -1
    output = -1
    return maxiter, decoded, output
```

Decode message

[25]:
```python
iters, decoded, output = decoder_loop(y, H, 0.1)
print('iterations: {}'.format(str(iters)))
print('output: {}'.format(str(output)))
print('result: {}'.format(decoded))
```

```
iterations: 8
output: 0
result: [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1
1
 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 0 0 1
 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1
 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 1 1 0
 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 0 0
 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1
 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0
 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1
 0 1 0 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1
 1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0
 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1
 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 1
 1 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0
 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 0
 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 1
 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1
 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 0 1
 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1
 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 1
 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 0 1
 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 1
 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1
 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 0
```

```
0 1 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1
1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 1 1
0]
```

0.3 Question 4

[26]:
```
def message_decode(decoded):
    sequence = [decoded[i:i+8] for i in range(0,248,8)]
    decimals = [int(''.join(map(str, sequence[i])), 2) for i in range(0,␣
↪len(sequence))]
    text_list = [chr(decimals[i]) for i in range(0, len(decimals))]
    message = ''.join(text_list)
    return message
```

[27]:
```
message_decode(decoded)
```

[27]:
```
'Happy Holidays! Dmitry&David :)'
```

0.4 Question 5

[28]:
```
#generate fake xs of size 252 bits
def fake_xs(data_size):
    fake = []
    for i in range(data_size):
        temp = list(np.random.randint(2, size=252))
        fake.append(temp)
    return np.array(fake)
```

[29]:
```
#Encode the fake xs using the generator matrix
def encode_G(xs):
    temp = G.dot(xs)
    return (temp % 2)
```

[30]:
```
#randomly flip bits dependent on the probability value given
def flip_bits(p, x):
    bits_flip = []
    for i in range(1000):
        bits_flip.append(np.random.choice(np.arange(0, 2), p=[1-p, p]))
    bits = np.array(bits_flip)
    fin = (x + bits) % 2
    return fin
```

[31]:
```
#Full run where fake xs are generated, encoded and then bits randomly flipped␣
↪depending on the value of p
#A decoding is succesful if the output value = 0
def bonus_q(p, data_size=100):
    gen_xs = fake_xs(data_size)
    correct = 0
    for i in gen_xs:
```

```
        x = encode_G(i)
        y = flip_bits(p, x)
        iters, result, output = decoder_loop(y, H, p)
        if output == 0:
            correct += 1
    return (correct/data_size)
```

[32]:
```
#Run the empirical experiment
ps = list(np.arange(0.01, 0.20, 0.01))
accuracy = {p: [] for p in ps}
for p in tqdm(ps):
    res = bonus_q(p, data_size=100)
    accuracy[p].append(res)
```

Plot the accuracies against p values

[33]:
```
#Plot the percentage of successful encodings against the value of p
accuracies = sorted(accuracy.items())
p, acc = zip(*accuracies)
plt.plot(p, acc)
plt.xlabel("Value of p (noise)")
plt.ylabel("% Successful decodings")
plt.show
```

[33]: <function matplotlib.pyplot.show(*args, **kw)>