Homework 6

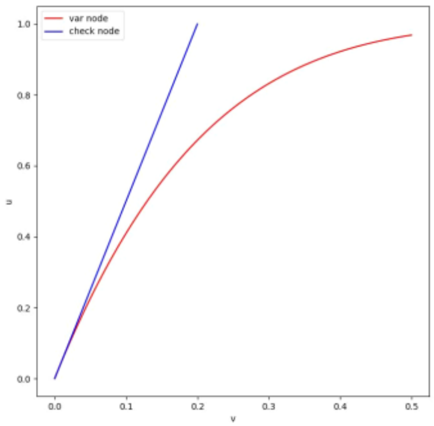
13 November 2023 19:37

6.1

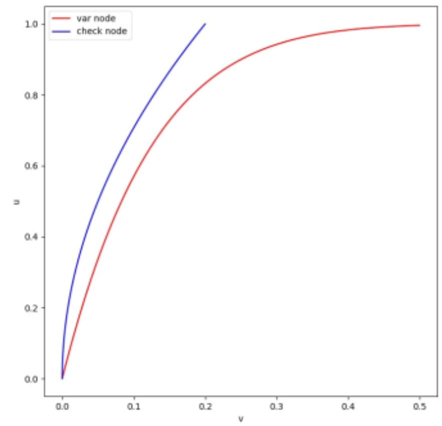
a) Any pairs (d_v, d_c) with $3d_v = d_c$ and $d_v > 1$: $(2,6); (3,9); (4,12), \ldots$

b)

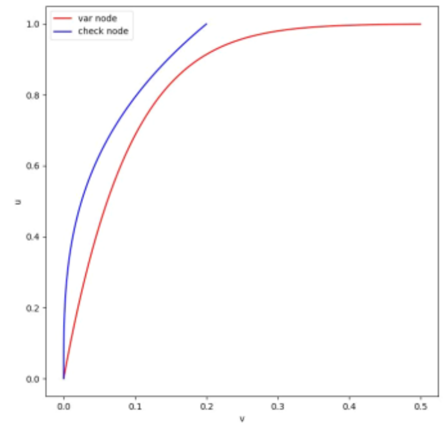
(2, 6) code



(3, 9) code



(4, 12) code



The decoder value converge to 0

c)

```
import numpy as np
import matplotlib.pyplot as plt

def check_node_out(d, v):
    return 1 - (1 - v) ** (d - 1)

def variable_node_out(d, sigma, u):
    return sigma * u ** (d - 1)

code_pairs = [(2, 6), (3, 9), (4, 12), (5, 15)]
max_iter = 1e5
for i, (dv, du) in enumerate(code_pairs):
    sigma_range = (0, 0.5)
    while True:
        _sigma = v_u = (sigma_range[0] + sigma_range[1]) / 2
        n_iter = 0
        converged = False
        while n_iter < max_iter:
            u_v = check_node_out(du, v_u)
            v_u = variable_node_out(dv, _sigma, u_v)
            if v_u < 1e-10:
                converged = True
                break
            n_iter += 1
        if converged:
            sigma_range = (_sigma, sigma_range[1])
        else:
            sigma_range = (sigma_range[0], _sigma)
        new_sigma = (sigma_range[0] + sigma_range[1]) / 2
        if np.round(new_sigma, 5) == np.round(_sigma, 5):
            break
    print(f"Noise threshold of {(dv, du)}: {_sigma}")
```

Noise threshold of (2, 6): 0.1999664306640625
Noise threshold of (3, 9): 0.28282928466796875
Noise threshold of (4, 12): 0.2570838928222656
Noise threshold of (5, 15): 0.23032760620117188

The (3,9) code has the best noise threshold

6.2

a) $$\rho(x) = \frac{R'(x)}{R'(1)} = x^{10}$$

$$\lambda(x) = \frac{L'(x)}{L'(1)} = \frac{\frac{378}{648}x + x^2 + \frac{540}{648}x^3 + \frac{162}{648}x^5 + x^7}{\frac{2376}{648}}$$

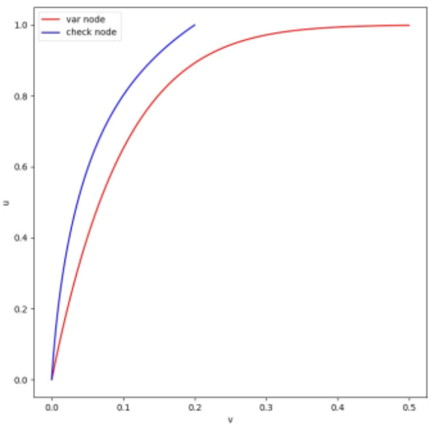
$$= 378\, x + 648\, x^2 + 540\, x^3 + 162\, x^5 + 648\, x^7$$

$L'(1)$ $\qquad \frac{2376}{648}$

$$= \frac{378}{2376}x + \frac{648}{2376}x^2 + \frac{540}{2376}x^3 + \frac{162}{2376}x^5 + \frac{648}{2376}x^7$$

$$R_d = 1 - \frac{L'(1)}{R'(1)} = \frac{2376}{648} \cdot \frac{1}{11} = \frac{2376}{7128} = \frac{1}{3}$$

b)



c)

```
import numpy as np
import matplotlib.pyplot as plt

def check_node_out(d, v):
    return 1 - (1 - v) ** (d - 1)

def variable_node_out(d_list, sigma, u):
    return sigma * np.sum([p * (u) ** (d - 1) for p, d in d_list])

du = 11
dv = [(378 / 2376, 2), (648 / 2376, 3), (540 / 2376, 4), (162 / 2376, 6), (648 / 
2376, 8)]
v_range = np.linspace(0, 0.5, num=100)
u_range = np.linspace(0, 1, num=100)
max_iter = 1e5
sigma_range = (0.0, 0.5)
while True:
    _sigma = v_u = (sigma_range[0] + sigma_range[1]) / 2
    n_iter = 0
    converged = False
    while n_iter < max_iter:
        u_v = check_node_out(du, v_u)
        v_u = variable_node_out(dv, _sigma, u_v)
        if v_u < 1e-10:
            converged = True
            break
        n_iter += 1
    if converged:
        sigma_range = (_sigma, sigma_range[1])
    else:
        sigma_range = (sigma_range[0], _sigma)
    new_sigma = (sigma_range[0] + sigma_range[1]) / 2
    if np.round(new_sigma, 5) == np.round(_sigma, 5):
        break
print(f"Noise threshold of {(dv, du)}: {_sigma}")
```

```
Noise threshold of ([(0.1590909090909091, 2),
(0.2727272727272727, 3), (0.22727272727272727, 4),
(0.06818181818181818, 6), (0.2727272727272727, 8)], 11):
0.29228973388671875
```

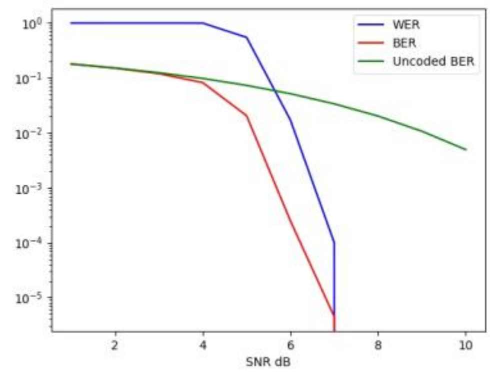
d) For BEC, the capacity is $C = 1 - \varepsilon$. If $C = R_d = \frac{1}{3}$ then $\varepsilon^* = \frac{2}{3}$

which is more than 2 time larger than the noise threshold in (c)

6.3

```
import numpy as np
```

6.5




```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

def create_quasi_cyclic_ldpc(z, proto):
    P = np.identity(z)
    H = []
    for r in range(proto.shape[0]):
        H_row = []
        for c in range(proto.shape[1]):
            if proto[r, c] == -1:
                H_row.append(np.zeros((z, z)))
            else:
                H_row.append(np.roll(P, proto[r, c], axis=1))
        H.append(np.hstack(H_row))
    return np.vstack(H)

def convert_symbol_to_input(c):
    return np.array([1 - 2 * i for i in c])

def bawgn(c, sigma2):
    x = convert_symbol_to_input(c)
    return np.random.normal(x, np.sqrt(sigma2))

def eb_snr_db_to_noise_var(snr, rate):
    return 1 / (2 * (10 ** (snr / 10)) * rate)

def sum_product_decoding(y, H, sigma2, check_nei, var_nei, max_iter):
    L = (2 / sigma2) * y
    Q = [L[check_nei[check_i]] for check_i in range(m)]
    Q_calc = [np.prod(np.tanh(np.array(Q[check_i]) / 2))
              for check_i in range(m)]
    n_iter = 0
    while n_iter < max_iter:
        R = [
            [2 * np.arctanh(Q_calc[check_i] / np.tanh(Q[check_i]
[check_nei[check_i].index(var_i)] / 2))
             for check_i in var_nei[var_i]]
            for var_i in range(n)
        ]
        R_calc = [L[var_i] + np.sum(R[var_i]) for var_i in range(n)]
        Q = [
            [R_calc[var_i] - R[var_i][var_nei[var_i].index(check_i)]
             for var_i in check_nei[check_i]]
            for check_i in range(m)
        ]
        Q_calc = [np.prod(np.tanh(np.array(Q[check_i]) / 2)) for check_i in
range(m)]
        c_hat = np.array([0 if v >= 0 else 1 for v in R_calc])
        if np.array_equal((c_hat.dot(H.T) % 2), np.zeros(m)):
            break
        n_iter += 1
    return c_hat

n, k = 648, 432
rate = 1 - (k / n)
z = 27
proto = np.array([
    [25, 26, 14, -1, 20, -1,  2, -1,  4, -1, -1,  8, -1, 16, -1,
18,  1,  0, -1, -1, -1, -1, -1, -1],
    [10, 9, 15, 11, -1, 0, -1,  1, -1, -1, 18, -1, 8, -1, 10, -1, -1, 0,
0, -1, -1, -1, -1, -1,],
    [16, 2, 20, 26, 21, -1, 6, -1, 1, 26, -1, 7, -1, -1, -1, -1, -1, -1, 0,
0, -1, -1, -1, -1],
    [10, 13, 5, 0, -1, 3, -1, 7, -1, -1, 26, -1, -1, 13, -1, 16, -1, -1, -1, 0,
0, -1, -1, -1],
    [23, 14, 24, -1, 12, -1, 19, -1, 17, -1, -1, -1, 20, -1, 21, -1,
0, -1, -1, -1, 0, 0, -1, -1],
    [6, 22, 9, 20, -1, 25, -1, 17, -1, 8, -1, 14, -1,
18, -1, -1, -1, -1, -1, -1, -1, 0, 0, -1],
    [14, 23, 21, 11, 20, -1, 24, -1, 18, -1, 19, -1, -1, -1, -1,
22, -1, -1, -1, -1, -1, -1, 0, 0],
    [17, 11, 11, 20, -1, 21, -1, 26, -1, 3, -1, -1, 18, -1, 26, -1,
1, -1, -1, -1, -1, -1, -1, 0]
])
H = create_quasi_cyclic_ldpc(z, proto)
m = H.shape[0]
check_node_neighbors = [[c for c in range(n) if H[r, c] == 1] for r in range(m)]
var_node_neighbors = [[r for r in range(m) if H[r, c] == 1] for c in range(n)]
n_mcmc = int(1e4)
max_iter = 10
u = np.array([0] * k)
c = np.array([0] * n)
snr_db_list = [i for i in range(1, 11)]
wer_list, ber_list, uncoded_ber_list = [], [], []
for snr in snr_db_list:
    sigma2 = eb_snr_db_to_noise_var(snr, rate)
    wer = ber = uncoded_ber = 0
```

```
        for _ in tqdm(range(n_mcmc)):
            y = bawgn(c, sigma2)
            c_hat = sum_product_decoding(
                y, H, sigma2, check_node_neighbors, var_node_neighbors, max_iter)
            if not np.array_equal(c, c_hat):
                wer += 1
                ber += sum([1 for i, j in zip(c, c_hat) if i != j])
            y = bawgn(u, sigma2)
            u_hat = np.array([0 if y[i] >= 0 else 1 for i in range(k)])
            if not np.array_equal(u, u_hat):
                uncoded_ber += sum([1 for i, j in zip(u, u_hat) if i != j])
        wer_list.append(wer / n_mcmc)
        ber_list.append(ber / (n * n_mcmc))
        uncoded_ber_list.append(uncoded_ber / (k * n_mcmc))
        print(f"SNR/Sigma2: {snr}/{sigma2:.04} - WER: {wer / n_mcmc:0.4} - "
              f"BER: {ber / (n * n_mcmc):0.4} - Uncoded BER: {uncoded_ber / (k *
    n_mcmc):0.4}")
    print(wer_list)
    print(ber_list)
    print(uncoded_ber_list)
    plt.plot(snr_db_list, wer_list, color='blue')
    plt.plot(snr_db_list, ber_list, color='red')
    plt.plot(snr_db_list, uncoded_ber_list, color='green')
    plt.xlabel("SNR dB")
    plt.legend(["WER", "BER", "Uncoded BER"])
    plt.yscale('log')
    plt.savefig('6.3.jpg')
```