HW2

June 4, 2024

0.1 Codes

There are ready made commands for coding theory, including: * constructions of "famous" codes, e.g., Hamming codes and Golay codes * minimum distance * generator and parity check matrix * "famous" bounds such as the sphere packing bound

We illustrate the commands with Hamming codes. q-ary Hamming codes have parameters $n = (q^r - 1)/(q - 1)$, dimension $n - r$ and minimal distance 3.

[1]:
```
C1 = codes.HammingCode(GF(4), 2)
C1
```

[1]:
```
[5, 3] Hamming Code over GF(4)
```

[2]:
```
(4^2-1)/(4-1), C1.minimum_distance()
```

[2]:
```
(5, 3)
```

[3]:
```
C1.generator_matrix()
```

[3]:
```
[     1      0      0 z2 + 1     z2]
[     0      1      0      1      1]
[     0      0      1     z2 z2 + 1]
```

This is the "famous" binary Hamming code, note that columns represent all possible binary vectors apart 0.

[4]:
```
C2 = codes.HammingCode(GF(2), 3)
C2.parity_check_matrix()
```

[4]:
```
[1 0 1 0 1 0 1]
[0 1 1 0 0 1 1]
[0 0 0 1 1 1 1]
```

This is the sphere packing bound. So C_1 contains 4^3 codewords and is perfect. Also C_2 contains 2^4 codewords and is perfect.

[5]:
```
# parameters: n,q,d
codes.bounds.hamming_upper_bound(5,4,3), 4^3
```

[5]: (64, 64)

[6]:
```
# parameters: n,q,d
codes.bounds.hamming_upper_bound(7,2,3), 2^4
```

[6]: (16, 16)

0.2 Exercise

The goal of this exercise is to construct a generator matrix for the binary Golay [23.12] code. This is based on Example 5.9.1 in the notes. We suggest the following steps: * check that 2 has horder 11 modulo 23, * factorize $X^{23}-1$ over $\mathbb{F}_2$, * compute the multiplicative subset I of $(\mathbb{Z}/23\mathbb{Z})^\times$ generated by 2, * construct an extension of $\mathbb{F}_2$ to find a primitive 23rd root of unity, * using I as defining set, find a generator polynomial * use the generator polynomial to obtain a generator matrix.

Once done, one could further: * put this matrix in systematic form * compute the minimum distance * check the code is perfect.

Exercise 73 in the notes: Check that 2 has order 11 modulo 23 and that $X^{23}-1$ over $\mathbb{F}_2$ is the product of three irreducible polynomials.

[7]:
```
for i in range(1,12):
    print(i,(2^i)%23)
```

```
1 2
2 4
3 8
4 16
5 9
6 18
7 13
8 3
9 6
10 12
11 1
```

[8]:
```
R.<x> = PolynomialRing(GF(2))
factor(x^(23)-1)
```

[8]: (x + 1) * (x^11 + x^9 + x^7 + x^6 + x^5 + x + 1) * (x^11 + x^10 + x^6 + x^5 + x^4 + x^2 + 1)

Example 5.9.1 in the notes. Compute the multiplicative subset I of $(\mathbb{Z}/23\mathbb{Z})^\times$ generated by 2.

[9]:
```
I = []
for i in range(1,12):
    I.append((2^i)%23)
sorted(I)
```

[9]: [1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18]

Construct an extension of $\mathbb{F}_2$ to find a primitive 23rd root of unity.

[10]:
```
[(2^t-1)%23 for t in range(1,15)], (2^11-1)/23
```

[10]: ([1, 3, 7, 15, 8, 17, 12, 2, 5, 11, 0, 1, 3, 7], 89)

[11]:
```
F2_11.<a> = GF(2^11, modulus="primitive")
b = a^89
g = 1
for i in I:
    g = g*(x-b^i)
g
```

[11]: x^11 + x^9 + x^7 + x^6 + x^5 + x + 1

Construct a generator matrix for the polynomial of degree 11 computed above.

[12]:
```
rw1 = list(x^11 + x^9 + x^7 + x^6 + x^5 + x + 1)
rw1
```

[12]: [1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1]

[13]:
```
rw1 = rw1 + [0]*(23-len(rw1))
rw1
```

[13]: [1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[14]:
```
def cyclicshift(l):
    '''
        input: list
        output: a cyclic shift by 1 to the right
    '''
    ll = [l[-1]]
    ll += l[0:-1]

    return ll
```

[15]:
```
rws = [rw1]
rw = rw1
for i in range(11):
    nxtrow = cyclicshift(rw)
    rws.append(nxtrow)
    rw = nxtrow

G1 = matrix(GF(2),rws)
G1
```

[15]:
```
[1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0]
[0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0]
[0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1]
```

Put the generator matrix in systematic form.

[16]:
```
G1.echelon_form()
```

[16]:
```
[1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0]
[0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1]
[0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1 0]
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1]
[0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 1 1]
[0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1]
[0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1]
[0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1]
[0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1]
```

Compute the minimum distance.

[17]:
```
LinearCode(G1)
```

[17]:
```
[23, 12] linear code over GF(2)
```

[18]:
```
LinearCode(G1).minimum_distance()
```

[18]:
```
7
```

Check that the code is perfect.

[19]:
```
# parameters: n,q,d
codes.bounds.hamming_upper_bound(23,2,7), 2^12
```

[19]:
```
(4096, 4096)
```