Decoding of Golay codes, in Sagemath

April 8, 2024

1 MAT005 - Coding Theory, FS 24, Sheet 6

Lecturer: Prof. Dr. Joachim Rosenthal; *TA:* Giacomo Borin.

1.1 Ex 04: Golay codes

Consider the $[23, 12, 7]$ binary golay code G_{23} with systematic generator matrix $\mathbf{G} = (\mathbf{I}_{12} \mid \mathbf{A})$ for

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Decode the following received vectors:

$$\mathbf{r}_0 = (1,\ 0,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0)\ ;$$

$$\mathbf{r}_1 = (0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1)\ ;$$

$$\mathbf{r}_2 = (1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 1,\ 0)\ .$$

You can decode with **any** method you want, look anywhere you like for the one that fits you better, but you have to insert a reference and briefly describe the procedure you perform.

[6]:
```
codes.GolayCode?
```

```
Init signature: codes.GolayCode(base_field, extended=True)
Docstring:
   Representation of a Golay Code.

   INPUT:
```

```
   * "base_field" -- The base field over which the code is defined.
     Can only be "GF(2)" or "GF(3)".

   * "extended" -- (default: "True") if set to "True", creates an
     extended Golay code.

   EXAMPLES:

      sage: codes.GolayCode(GF(2))
      [24, 12, 8] Extended Golay code over GF(2)

   Another example with the perfect binary Golay code:

      sage: codes.GolayCode(GF(2), False)
      [23, 12, 7]  Golay code over GF(2)
File:           /private/var/tmp/sage-10.2-current/local/var/lib/sage/venv-python3.
↪11.1/lib/python3.11/site-packages/sage/coding/golay_code.py
Type:           type
Subclasses:     GolayCode_with_category
```

[8]:
```python
F = GF(2)
C = codes.GolayCode(base_field = F,extended = False)
n = C.length()
k = C.dimension()
C
```

[8]: `[23, 12, 7]  Golay code over GF(2)`

[2]:
```python
show(C.systematic_generator_matrix())
```

$$\left(\begin{array}{ccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Question: Do something change if we consider another generator matrix? What can you deduce from the following:

[46]:
```python
show(C.generator_matrix())
```

$$\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}$$

We can proceed in the generation of the three examples of the exercise:

[21]:
```
y = C.random_element(); show(y)
z = copy(y)
z[2] += 1; show(z)
show(C.syndrome(z))
```

$(1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1)$

$(1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1)$

$(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$

[168]:
```
M = F**12
D = F**23
CODEWORDS = []
ERRORS = []
RECEIVED = []
for (i,key) in enumerate([1997,1861,1]):
    with seed(key):
        m = M.random_element() # sample a random message
        error_positions = [randint(0,22) for _ in range(3)]
        error_positions = list(dict.fromkeys(error_positions)) # removal of␣
 ↪duplicates
        while len(error_positions) < 3:
            error_positions.append(randint(0,22))
            error_positions = list(dict.fromkeys(error_positions))

    codeword = C.encode(m)
    CODEWORDS.append(codeword)
    ERRORS.append(error_positions)
    received = copy(codeword)
    for idx in error_positions:
        received[idx] += 1
    RECEIVED.append(received)
    show('\mathbf{c}_'+ str(i)+' =' + latex(codeword))
```

```
    show('\mathbf{e}_'+ str(i)+' =' + latex(set(error_positions)))
    show('\mathbf{r}_'+ str(i)+' =' + latex(received))
    # print('$$ % key = ' + str(key) + '\n \mathbf{r}_'+ str(i)+' =' +␣
↪latex(received))
    # print('$$ %  err = ' + str(set(error_positions)))
    # print(' % codeword = ' + str(codeword))
    print('')
```

$\mathbf{c}_0 = (1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0)$

$\mathbf{e}_0 = \{18,\ 19,\ 13\}$

$\mathbf{r}_0 = (1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0)$

$\mathbf{c}_1 = (0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1)$

$\mathbf{e}_1 = \{19,\ 6,\ 15\}$

$\mathbf{r}_1 = (0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1)$

$\mathbf{c}_2 = (0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0)$

$\mathbf{e}_2 = \{0,\ 10,\ 13\}$

$\mathbf{r}_2 = (1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0)$

[169]:
```
for key in range(10,500):
    with seed(key):
        m = M.random_element() # sample a random message
        error_positions = [randint(0,22) for _ in range(3)]
        error_positions = list(dict.fromkeys(error_positions)) # removal of␣
↪duplicates
        while len(error_positions) < 3:
            error_positions.append(randint(0,22))
            error_positions = list(dict.fromkeys(error_positions))

    codeword = C.encode(m)
    CODEWORDS.append(codeword)
    ERRORS.append(error_positions)
    received = copy(codeword)
    for idx in error_positions:
        received[idx] += 1
    RECEIVED.append(received)
```

[171]:
```
received = RECEIVED[2]
received
```

[171]: (1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0)

Let's focus on the last one, but still writing techniques that hopefully work in general. For simplicity we write here the Hamming distance function.

[53]:
```
def distH(x,y):
    if not len(x) == len(y):
        raise ValueError('Input vectors of different lenght')
    else:
        return sum([x[i] != y[i] for i in range(len(x))])

distH(vector([0,1,1,8]),vector([0,1,2,8]))
```

[53]: 1

Brute force This is the most easy and less efficient technique, we simply go through all the errors until we find the right one. It is important to note that it is only becaues the code is perfect that we can prove this procedure always terminates, with other codes this may output nothing.

[28]:
```
def check_err(err,r,code = C):
    buff = copy(r)
    for pos in err:
        buff[pos] += 1
    return buff in C,buff
```

[29]:
```
def brute_force(received,t = 3, code = C):
    for err in Subsets(set([0..22]),k = t):
        buff = copy(received)
        decoded,decoded_codeword = check_err(err,buff,code = code)
        if decoded:
            break
    return err,decoded_codeword

err,decoded_codeword = brute_force(received)

show('\mathbf{e} =' + latex((err)))
show('\mathbf{c} =' + latex(decoded_codeword))
```

$\mathbf{e} = \{1, 20, 5\}$

$\mathbf{c} = (1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0)$

[32]:
```
import time
t_start = time.time()
for r,e,c in zip(RECEIVED,ERRORS,CODEWORDS):
    errors, decoded_codeword = brute_force(r)
    if c != decoded_codeword:
        print('error')
        break
```

```
print(time.time() - t_start)
```

95.81962323188782

Syndrome Decoding The idea is to compute for every possible error vector $\mathbf{e}$ of weight $t \leq 3$ the syndrome:

$$\mathbf{e} \cdot \mathbf{H}^{\perp}$$

and store them in a table with the errors.

[45]:
```
dict_synd = {}
buff = D.zero_vector()
buff[0] += 1
show(C.syndrome(buff))
dict_synd[str(C.syndrome(buff))] = {0}
print(dict_synd)
```

(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

```
{'(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)': {0}}
```

[46]:
```
for err in Subsets(set([0..22]),k = 3):
    buff = D.zero_vector()
    for pos in err:
        buff[pos] = 1
    dict_synd[str(C.syndrome(buff))] = err


# Let's see the first 5 entries
iterator = iter(dict_synd.items())
for i in range(7):
    print(next(iterator))
```

```
('(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)', {0})
('(1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)', {0, 1, 2})
('(1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0)', {0, 1, 3})
('(1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0)', {0, 1, 4})
('(1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0)', {0, 1, 5})
('(1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0)', {0, 1, 6})
('(1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)', {0, 1, 7})
```

[47]:
```
err = dict_synd[str(C.syndrome(received))]
check_err(err,received)
```

[47]: (True, (1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0))

[48]:
```
import time
t_start = time.time()
for r in RECEIVED:
```

```
    err = dict_synd[str(C.syndrome(r))]
    if not check_err(err,r):
        print('error')
        break

print(time.time() - t_start)
```

0.22689414024353027

Information Set Decoding Here we sample random information sets $I \subset \{1, ..., n\}$ until we find one such that

$$\mathsf{supp}(\mathbf{e}) \cap I = \emptyset$$

At this point we would have that:

$$\mathbf{r}_I = \mathbf{c}_I = \mathbf{m} \cdot \mathbf{G}_I \ .$$

So we can solve the system for $\mathbf{m}$ and find the codeword!

[44]:
```
def rand_infoset(code = C):
    n = code.length()
    k = code.dimension()
    # here we add elements to the set until we get k of them
    info_set = {randint(0,n-1)}
    G = C.generator_matrix()
    while len(info_set) < k:
        info_set.add(randint(0,n-1))
    sub_G = G[:,list(info_set)]

    # if it is singular we repeat
    while sub_G.is_singular():
        info_set = {randint(0,n-1)}
        while len(info_set) < k:
            info_set.add(randint(0,n-1))
        sub_G = G[:,list(info_set)]

    # translate to list
    info_set = list(info_set)
    info_set.sort()
    return info_set

rand_infoset(code = C)
```

[44]: [0, 2, 3, 5, 6, 7, 9, 13, 18, 19, 21, 22]

[51]:
```
def invert_info_set(info_set, received, code = C):
    G = C.systematic_generator_matrix()
    sub_G = G[:,info_set]
    sub_received = vector([x for (idx,x) in enumerate(received) if idx in␣
 ↪info_set])
```

```
    m = sub_received / sub_G # system solve on the right!
    return m*G
```

[56]:
```
def info_set_decoding(received, t, code = C):
    iter = 1
    # take a random info set
    info_set = rand_infoset(code = code)
    # try to solve the system
    sol = invert_info_set(info_set, received, code = code)
    # check if the distance is less or equal then the threshold
    while distH(received,sol) > t:
        iter += 1
        info_set = rand_infoset(code = code)
        sol = invert_info_set(info_set, received, code = code)
    return sol,iter

info_set_decoding(received, t = 3, code = C)
```

[56]:
```
((1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0), 9)
```

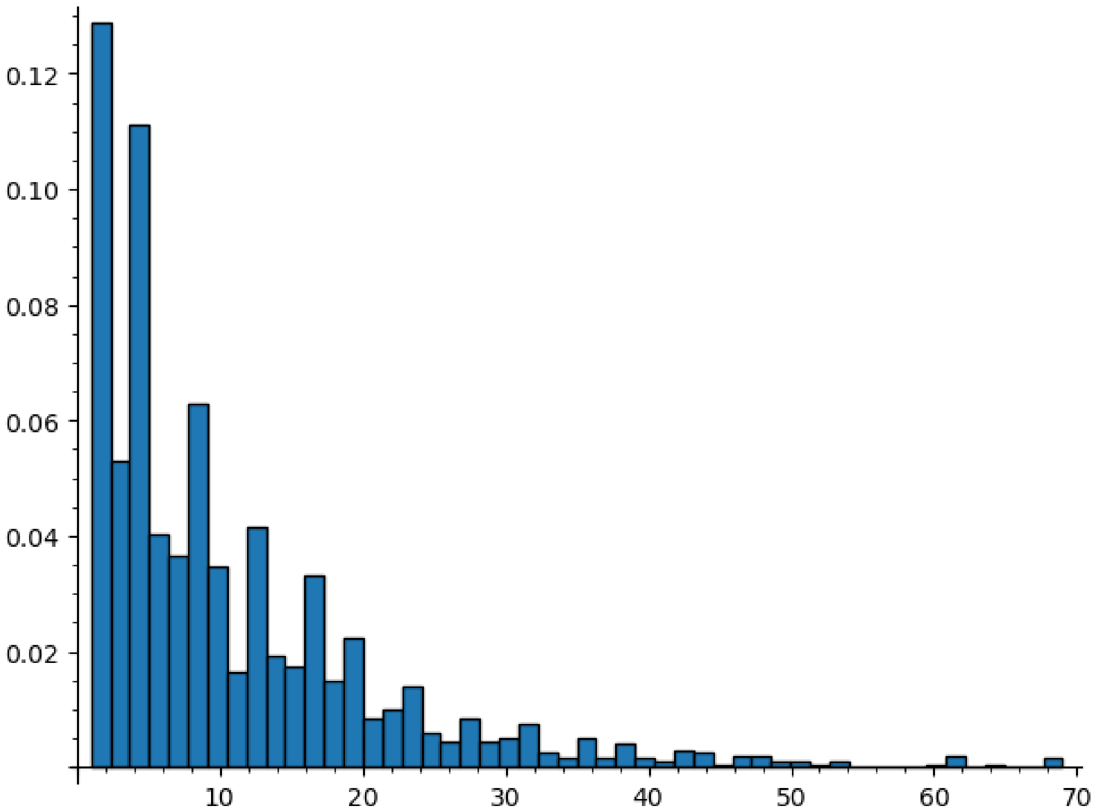
[59]:
```
import time
t_start = time.time()
iters = []
for r,e,c in zip(RECEIVED,ERRORS,CODEWORDS):
    sol,iter = info_set_decoding(r, t = 3, code = C)
    iters.append(iter)
    if not sol == c:
        print('error')
        break

print(time.time() - t_start)
```

```
3.517163038253784
```

[73]:
```
from sage.plot.histogram import Histogram
plot(histogram(iters,bins=50, density=True))
```

[73]:



1.1.1 Algebraic decoding

Now we would like to exploit the algebraic structure of cyclic code that we have for $\mathcal{G}_{23} = \langle g(x) \rangle$, for

$$g(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

We know that, if $r(x) = c(x) + e(x)$ with $g(x) \mid c(x)$ and error positions $\{e_1, e_2, e_3\}$ (so $e(x) = x^{e_1} + x^{e_2} + x^{e_3}$, then for any root α of $g(x)$ we have

$$r(\alpha) = c(\alpha) + e(\alpha) = 0 + e(\alpha) = e(\alpha) = \alpha^{e_1} + \alpha^{e_2} + \alpha^{e_3} =: s$$

[214]:
```
R.<x> = GF(2)['x']
f = x**11 + x**2 + 1
beta = GF(2**11,name = 'beta', modulus = f).gen()
alpha = beta**-89
```

[215]:
```
show(alpha)
```

$\beta^{10} + \beta^5 + \beta^4 + \beta^3 + \beta^2 + 1$

[216]:
```
show(alpha.multiplicative_order())
```

23

[217]:
```
show(alpha.minimal_polynomial())
```

$x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$

[260]:
```
C.generator_matrix()
```

[260]:
```
[1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0]
[0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0]
[0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0]
[0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1]
```

[220]:
```
received = RECEIVED[2]
r = R(list(received))
show(r)
synd = r(alpha)
show('s= ',synd)
```

$x^{21} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^9 + x^6 + x^4 + x^2 + 1$

s=$\beta^{10} + \beta^8 + \beta^5$

[221]:
```
Multi.<S> = R['S']
ex = x**11 + x**2 + 1 + S
ex
```

[221]:
```
S + x^11 + x^2 + 1
```

Consider now an *error locator polynomial* $L(x)$ such that:

$$L(x) = (x - \alpha^{e_1})(x - \alpha^{e_2})(x - \alpha^{e_3})$$

We hope to find it starting from the *syndrome* $s = \alpha^{e_1} + \alpha^{e_2} + \alpha^{e_3}$, for example from a direct calculation we have that the coefficient of x^2 is actually:

$$-\alpha^{e_1} - \alpha^{e_2} - \alpha^{e_3} = -s$$

so we have hope that some relation can be found…

Now let's apply some magic (i.e. we compute some *Groebner Basis* https://en.wikipedia.org/wiki/Gröbner_basis) to get the secret algebraic relation:

[226]:
```
LL = S^277 + S^276*x^22 + S^273*x^19 + S^272*x^18 + \
             S^261*x^7 + S^260*x^6 + S^257*x^3 + S^256*x^2 + \
             S^70 + S^68*x^21 + S^66*x^19 + S^64*x^17 + S^47 + \
             S^46*x^22 + S^45*x^21 + S^44*x^20 + S^43*x^19 + \
             S^42*x^18 + S^41*x^17 + S^40*x^16 + S^39*x^15 + \
             S^38*x^14 + S^37*x^13 + S^36*x^12 + S^35*x^11 + \
             S^34*x^10 + S^33*x^9 + S^32*x^8 + S^21*x^20 + \
             S^20*x^19 + S^17*x^16 + S^16*x^15 + S^15*x^14 + \
             S^14*x^13 + S^13*x^12 + S^12*x^11 + S^11*x^10 + \
             S^10*x^9 + S^9*x^8 + S^8*x^7 + S^7*x^6 + \
             S^4*x^3 + S^3*x^2 + S

show(LL)
```

$$S^{277} + x^{22}S^{276} + x^{19}S^{273} + x^{18}S^{272} + x^{7}S^{261} + x^{6}S^{260} + x^{3}S^{257} + x^{2}S^{256} + S^{70} + x^{21}S^{68} + x^{19}S^{66} + x^{17}S^{64} + S^{47} + x^{22}S^{46} + x^{21}S^{45} + x^{20}S^{44} + x^{19}S^{43} + x^{18}S^{42} + x^{17}S^{41} + x^{16}S^{40} + x^{15}S^{39} + x^{14}S^{38} + x^{13}S^{37} + x^{12}S^{36} + x^{11}S^{35} + x^{10}S^{34} + x^{9}S^{33} + x^{8}S^{32} + x^{20}S^{21} + x^{19}S^{20} + x^{16}S^{17} + x^{15}S^{16} + x^{14}S^{15} + x^{13}S^{14} + x^{12}S^{13} + x^{11}S^{12} + x^{10}S^{11} + x^{9}S^{10} + x^{8}S^{9} + x^{7}S^{8} + x^{6}S^{7} + x^{3}S^{4} + x^{2}S^{3} + S$$

[230]:
```
gcd(LL(synd),x**23 + 1)
```

[230]:
```
x^3 + (beta^10 + beta^9 + beta^6 + beta^3 + beta^2 + beta)*x^2 + (beta^9 +
beta^8 + beta^7 + beta^5 + beta^3)*x + beta^10 + beta^6 + beta^5 + beta^4 +
beta^3 + beta^2
```

[259]:
```
r = list(RECEIVED[2])
r = R(r)
synd = r(alpha)

show(gcd(polynomial(synd),x**23 + 1))
print()
print('Error positions :',end=' ')
for i in range(3):
    root = gcd(LL(synd,x),x**23 + 1).roots()[i][0]
    print(log(root,alpha**-1), end=' ')
```

$$x^3 + \left(\beta^{10} + \beta^8 + \beta^5\right) x^2 + \left(\beta^{10} + \beta^8 + \beta^5\right) x + 1$$

```
Error positions : 0 13 10
```

[258]:
```
def algebraic_solutions(received):
    r = list(received)
    r = R(r)
    synd = r(alpha)
    err = []
    for i in range(3):
        root = gcd(LL(synd),x**23 + 1).roots()[i][0]
        err.append((23-log(root,alpha)) % 23)
```

```
    err.sort()
    return err

algebraic_solutions(RECEIVED[0]),ERRORS[0]
```

[258]: ([13, 18, 19], [13, 18, 19])

[257]:
```
import time
t_start = time.time()
iters = []
for r,e,c in zip(RECEIVED,ERRORS,CODEWORDS):
    err = algebraic_solutions(r)
    if not set(err)==(set(e)):
        print('error',err,e)


print(time.time() - t_start)
```

15.653797149658203

Some references for this magic:

1. Chapter 10 of the wonderful book: *Codes, Cryptology and Curves with Computer Algebra*, Ruud Pellikaan, Xin-Wen Wu, Stanislav Bulygin, Relinde Jurrius (in case give also a look to **chapter 12**).
2. NASA decoding procedure from 1988, paper
3. *The Golay codes*, Mario de Boer and Ruud Pellikaan, from TU/e
4. Gröbner Basis, any Computational Algebra Book
5. A small presentation on this topic on my own.