In [1]:
```
import itertools
```

In [2]:
```
# Hamming weight function
def weight(v):
    return sum(x!=0 for x in v)
```

In [3]:
```
# finite field
F = GF(19)
```

In [4]:
```
# generator matrix for [18, 2, 17] RS code
alpha = [i for i in range(1, 19)]
G = Matrix(F, [[1 for i in range(18)], [alpha[i] for i in range(18)]])
pretty_print(G)
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{pmatrix}$$

In [5]:
```
# basic code and Sudan parameters
k=2
n=18
Rp = (k-1)/n
ell = 4
print(f"k: {k}   n: {n}   R': {Rp}   ell: {ell}")
```

```
k: 2   n: 18   R': 1/18   ell: 4
```

In [6]:
```
# derived parameters
Theta = ell/(ell+1) - (ell/2)*Rp
pretty_print(Theta)
tau = ceil(n*Theta)-1
pretty_print(tau)
```

$$\frac{31}{45}$$

$$12$$

In [7]:
```
# message vector
u = vector(F, [18, 14])
pretty_print(u)
# codeword
c = u*G
pretty_print(c)
```

(18, 14)

(13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 5, 0, 14, 9, 4)

In [8]:
```
# received word
y = vector(F, [5, 5, 1, 10, 10, 7, 2, 18, 6, 6, 1, 15, 13, 5, 14, 3, 1, 0])
pretty_print(y)
```

(5, 5, 1, 10, 10, 7, 2, 18, 6, 6, 1, 15, 13, 5, 14, 3, 1, 0)

In [9]:
```
# error vector
ev = y - c
pretty_print(ev)
```

(11, 16, 17, 12, 17, 0, 0, 2, 14, 0, 0, 0, 3, 0, 14, 8, 11, 15)

In [10]:
```
# Hamming weight of error vector
weight(ev)
```

Out[10]: 12

In [11]:
```
# Bivariate polynomial ring
P.<x,z> = PolynomialRing(F)
# degree constraints
zdeg = ell
xdeg = lambda r: n - tau - 1 - (k-1)*r
# monomials vector
ml = [[x^j*z^i for j in range(xdeg(i)+1)] for i in range(zdeg+1)]
ml = list(itertools.chain(*ml))
mv = vector(P, ml)
pretty_print(mv)
print(f"length={len(mv)}")
```

$$\left(1,\ x,\ x^2,\ x^3,\ x^4,\ x^5,\ z,\ xz,\ x^2z,\ x^3z,\ x^4z,\ z^2,\ xz^2,\ x^2z^2,\ x^3z^2,\ z^3,\ xz^3,\ x^2z^3,\ z^4,\ xz^4\right)$$

```
length=20
```

In [12]:
```
# Create the interpolation matrix
M = Matrix(F, [mv.subs(x=j, z=y[j-1]) for j in range(1,n+1)])
pretty_print(M)
```

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 11 & 11 & 11 & 17 & 17 \\
1 & 2 & 4 & 8 & 16 & 13 & 5 & 10 & 1 & 2 & 4 & 6 & 12 & 5 & 10 & 11 & 3 & 6 & 17 & 15 \\
1 & 3 & 9 & 8 & 5 & 15 & 1 & 3 & 9 & 8 & 5 & 1 & 3 & 9 & 8 & 1 & 3 & 9 & 1 & 3 \\
1 & 4 & 16 & 7 & 9 & 17 & 10 & 2 & 8 & 13 & 14 & 5 & 1 & 4 & 16 & 12 & 10 & 2 & 6 & 5 \\
1 & 5 & 6 & 11 & 17 & 9 & 10 & 12 & 3 & 15 & 18 & 5 & 6 & 11 & 17 & 12 & 3 & 15 & 6 & 11 \\
1 & 6 & 17 & 7 & 4 & 5 & 7 & 4 & 5 & 11 & 9 & 11 & 9 & 16 & 1 & 1 & 6 & 17 & 7 & 4 \\
1 & 7 & 11 & 1 & 7 & 11 & 2 & 14 & 3 & 2 & 14 & 4 & 9 & 6 & 4 & 8 & 18 & 12 & 16 & 17 \\
1 & 8 & 7 & 18 & 11 & 12 & 18 & 11 & 12 & 1 & 8 & 1 & 8 & 7 & 18 & 18 & 11 & 12 & 1 & 8 \\
1 & 9 & 5 & 7 & 6 & 16 & 6 & 16 & 11 & 4 & 17 & 17 & 1 & 9 & 5 & 7 & 6 & 16 & 4 & 17 \\
1 & 10 & 5 & 12 & 6 & 3 & 6 & 3 & 11 & 15 & 17 & 17 & 18 & 9 & 14 & 7 & 13 & 16 & 4 & 2 \\
1 & 11 & 7 & 1 & 11 & 7 & 1 & 11 & 7 & 1 & 11 & 1 & 11 & 7 & 1 & 1 & 11 & 7 & 1 & 11 \\
1 & 12 & 11 & 18 & 7 & 8 & 15 & 9 & 13 & 4 & 10 & 16 & 2 & 5 & 3 & 12 & 11 & 18 & 9 & 13 \\
1 & 13 & 17 & 12 & 4 & 14 & 13 & 17 & 12 & 4 & 14 & 17 & 12 & 4 & 14 & 12 & 4 & 14 & 4 & 14 \\
1 & 14 & 6 & 8 & 17 & 10 & 5 & 13 & 11 & 2 & 9 & 6 & 8 & 17 & 10 & 11 & 2 & 9 & 17 & 10 \\
1 & 15 & 16 & 12 & 9 & 2 & 14 & 1 & 15 & 16 & 12 & 6 & 14 & 1 & 15 & 8 & 6 & 14 & 17 & 8 \\
1 & 16 & 9 & 11 & 5 & 4 & 3 & 10 & 8 & 14 & 15 & 9 & 11 & 5 & 4 & 8 & 14 & 15 & 5 & 4 \\
1 & 17 & 4 & 11 & 16 & 6 & 1 & 17 & 4 & 11 & 16 & 1 & 17 & 4 & 11 & 1 & 17 & 4 & 1 & 17 \\
1 & 18 & 1 & 18 & 1 & 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

In [13]:
```
B = M.right_kernel().basis()
print(B)
```

```
[
(1, 0, 11, 13, 1, 0, 3, 16, 8, 9, 10, 11, 11, 9, 10, 13, 6, 16, 4, 12),
(0, 1, 11, 14, 13, 9, 16, 18, 6, 11, 10, 7, 18, 5, 3, 18, 10, 9, 8, 15)
]
```

In [14]:
```
s0 = vector(F, B[0])
s1 = vector(F, B[1])
```

In [15]:
```
# first solution in Q
Q0 = mv.dot_product(s0)
pretty_print(Q0)
```

$$10x^4z + 10x^3z^2 + 16x^2z^3 + 12xz^4 + x^4 + 9x^3z + 9x^2z^2 + 6xz^3 + 4z^4 + 13x^3 + 8x^2z + 11xz^2 + 13z^3 + 11x^2 + 16xz + 11$$

In [16]:
```
factor(Q0)
```

Out[16]:
```
(-7) * (5*x + z + 1) * (-8*x + z - 8) * (-2*x^2*z + x*z^2 - 4*x^2 - x*z - 6*z^2 + 6*x + 5*z - 1)
```

In [17]:
```
u00 = 18 + 14*x  # the original codeword
u01 = 8 + 8*x  # a second codeword
```

In [18]:
```
# the second codeword
c1 = vector(F, [u01.subs(x=alpha[i]) for i in range(18)])
```

In [19]:
```
# distance from received word
weight(c1-y)
```

Out[19]:
```
12
```

In [20]:
```
# second solution in Q
Q1 = mv.dot_product(s1)
pretty_print(Q1)
```

$$9x^5 + 10x^4z + 3x^3z^2 + 9x^2z^3 + 15xz^4 + 13x^4 + 11x^3z + 5x^2z^2 + 10xz^3 + 8z^4 + 14x^3 + 6x^2z - xz^2 - z^3 + 11x^2 - xz + 7$$

In [21]:
```
factor(Q1)
```

Out[21]:
```
(-4) * (5*x + z + 1) * (-8*x + z - 8) * (-6*x^3 - 4*x^2*z + x*z^2 - 4*x^2 + 8*x*z - 2*z^2 + 3*x - 9*
z)
```

In [22]:
```
# another solution
s2 = 4*s0 + 12*s1
Q2 = mv.dot_product(s2)
factor(Q2)
```

Out[22]:
```
(-2) * (3*x + z + 5) * (5*x + z + 1) * (-8*x + z - 8) * (9*x^2 + 4*x + z + 1)
```

In [23]:
```
# a third codeword?
u2 = 14 + 16*x
c2 = vector(F, [u2.subs(x=alpha[i]) for i in range(18)])
c2
```

Out[23]:
```
(11, 8, 5, 2, 18, 15, 12, 9, 6, 3, 0, 16, 13, 10, 7, 4, 1, 17)
```

In [24]:
```
weight(c2-y)
```

Out[24]:
```
15
```

This is outside the decoding radius tau=12!

BACK