Implementação

Começamos por importar a biblioteca do solver e o método *randint* da biblioteca *random* que será responsável pela criação dos valores pseudo-aleatórios. Além disso, criemos o modelo que será utilizado pelo solver. Este modelo é capaz de trabalhar com inequações não lineares.

In [1]:
```
# solver
from ortools.sat.python import cp_model

# Cria o modelo CP-SAT
model = cp_model.CpModel()

from random import randint
```

Inputs

1. O N corresponde ao número de colunas da matriz L (a matriz L será referenciada mais a frente) e é um inteiro qualquer maior do que 30 em teoria
2. O M corresponde ao tamanho do vetor E e ao número de linhas da matriz L (o vetor E e a matriz L serão referenciados mais a frente) e é de tal forma que |M| > |N| + 1
3. O Q é um primo qualquer maior ou igual que 3, tal que |Q| > |M|

In [2]:
```
N = 3
M = 15
Q = 73
```

Seja D $\equiv \frac{Q-1}{2}$ Como Q é um primo maior ou igual a 3, então Q é ímpar, logo Q-1 é par e portanto esta divisão da um inteiro

Consideremos agora uma matriz L, com M linhas e N colunas, tal que seus valores são gerados aleatória e uniformemente no intervalo inteiro {-d,...,d}

In [3]:
```
D = int((Q-1)/2)
L = {}

for m in range(M):
    for n in range(N):
        L[m,n] = randint(-D, D)
```

Consideremos um vetor E com M elementos. Cada elemento pode ser -1, 0 ou 1, ou seja, está no intervalo inteiro [-1,1]. Estes elementos serão variáveis que nos permitirão resolver aquilo que é proposto no trabalho prático.

In [4]:
```
E = {}

for m in range(M):
    E[m] = model.NewIntVar(-1, 1, f'E[{m}]')
```

Seja Z um vetor com M elementos e seja Z_m um elemento do vetor Z e E_m um elemento do vetor E, ambos com índice m, queremos algo do género:

$$Z_m = 1 \text{ se e só se } E_m = 0$$

Os elementos de Z serão variáveis booleanas do nosso problema que respeitarão a equivalência acima

In [5]:
```
Z = {}

for m in range(M):
    Z[m] = model.NewBoolVar(f'Z[{m}]')
    model.Add(E[m] == 0).OnlyEnforceIf(Z[m])
    model.Add(E[m] != 0).OnlyEnforceIf(Z[m].Not())
```

Queremos garantir que o vetor E que buscamos como solução, não é nulo, ou seja, há pelo menos um elemento que é diferente zero, isto é, como o vetor E tem M elementos, ele tem no máximo M-1 zeros:

$$\sum_{m<M} Z_m \leq M - 1$$

In [6]:
```
model.Add(sum([Z[m] for m in range(M)]) <= M - 1)
pass
```

Caso tenhamos uma solução não nula ao problema, queremos minimizar o número de zeros:

$$\sum_{m<M} Z_m$$

In [7]:
```
model.Minimize(sum([Z[m] for m in range(M)]))
pass
```

Queremos determinar o vetor E de forma a respeitar a seguinte relação matricial:

$$\forall_{n<N} \sum_{m<M} E_m \times Lm, n \equiv 0 \bmod \mathrm{q}$$

isto é,

$$\forall_{n<N}. \exists_{k \in \mathbb{Z}}. \sum_{m<M} E_m \times Lm, n = q \times k$$

In [8]:
```
K = {}

for n in range(N):
    K[n] = model.NewIntVar(-1000000, 1000000, f'K[{n}]')
    model.Add(sum([E[m] * L[m,n] for m in range(M)]) == Q * K[n])
```

Testes

In [9]:
```
solver = cp_model.CpSolver()

status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
    for m in range(M):
        # print(solver.Value(E[m]))
        pass
else:
    # print('No solution found.')
    pass
```