In [1]:
```
from functools import reduce
import numpy as np
from scipy.linalg import circulant
```

In [4]:
```
class Convolution1d:
    def __init__(self, filt):
        self.__filt = filt
        self.__r = filt.size
        self.T = TransposedConvolution1d(self.__filt)
    def __matmul__(self, vector):
        r, n = self.__r, vector.size
        return np.asarray(
            [sum(self.__filt * vector[j : j + r]) for j in range(n
        )

class TransposedConvolution1d:
    """
    Transpose of 1-dimensional convolution operator used for the
    transpose-convolution operation A.T@(...)
    """
    def __init__(self, filt):
        self.__filt = filt
        self.__r = filt.size
    def __matmul__(self, vector):
        r = self.__r
        n = vector.size + r - 1
        return np.asarray(
            [
                sum(
                    np.flip(self.__filt)[max(0, r - j - 1) : min(n
                    * vector[max(0, j - r + 1) : min(j + 1, n - r +
                )
                for j in range(n)
            ]
        )
def huber_loss(x):
    return np.sum(
        (1 / 2) * (x**2) * (np.abs(x) <= 1)
        + (np.sign(x) * x - 1 / 2) * (np.abs(x) > 1)
    )

def huber_grad(x):
    return x * (np.abs(x) <= 1) + np.sign(x) * (np.abs(x) > 1)
```

In [6]:
```
r, n, lam = 3, 20, 0.1

np.random.seed(0)
k = np.random.randn(r)
b = np.random.randn(n - r + 1)
A = Convolution1d(k)
B = circulant(np.concatenate((np.flip(k), np.zeros(n - r))))[r - 1
x = np.zeros(n)
y = np.zeros(n)
alpha = 0.01

for _ in range(100):
    x = x - alpha * (A.T @ (huber_grad(A @ x - b)) + lam * x)
    y = y - alpha * (B.T @ (huber_grad(B @ y - b)) + lam * y)

print(f'Efficient way loss: {huber_loss(A @ x - b) + 0.5 * lam * np
print(f'Inefficient way loss: {huber_loss(B @ x - b) + 0.5 * lam *
```

```
Efficient way loss: 0.4587586843129764
Inefficient way loss: 0.4587586843129765
```