Exercise 5 - (E10.1)

The required modules:

In [6]:
```python
import numpy as np
```

Let's store the problem data into variables:

In [7]:
```python
# Weight matrix
W = np.array([[1, -4, 2]])

# Inputs
yk = np.array([1, 1, 2])
```

To solve the problem we need a way to produce the inputs at each time step k, for this reason a class simulating the behavior of a tapped delay line was implemented in the following code cell:

In [8]:
```python
class TappedDelayLine(object):
    def __init__(self, y: np.ndarray, *, blocks: int):
        self.y = y                      # Input array starting from k=0
        self.length = blocks            # Length of the line
        self.num_outputs = blocks + 1   # Number of outputs
        self.k = -1                     # Initial time step

        # Initialize output array
        self.out_array = np.zeros((self.num_outputs, 1))

    def reset(self):
        # Reset timestep
        self.k = -1

    def out(self, k: int = None):
        # Initialize output array
        self.out_array = np.zeros((self.num_outputs, 1))

        # Make output function
        def make_output(kf: int):
            # Add element
            if kf < len(self.y):
                self.out_array = np.vstack(([[self.y[kf]]], self.out_array))
            else:
                self.out_array = np.vstack(([[0]], self.out_array))

            # Remove last element
            self.out_array = np.delete(self.out_array, obj=-1, axis=0)

        if k is None:  # If k is not overwritten, use the one in the class
            self.k += 1           # Increment timestep
            make_output(self.k)   # Update output
        else:  # if k is overwritten
            for i in range(k+1):  # Iterate to recreate k-th timestep
                make_output(i)    # Update output

        return self.out_array
```

An instance of this class is then created. Its behavior is shown in the output of the following code cell. To get the output of the tapped delay line at k, we simply call the ***tl.out(k)*** method, the output of the latter is the array of 3 inputs for the chosen time step:

In [9]:
```python
# Create an instance for our problem
tl = TappedDelayLine(yk, blocks=2)

# Simulation of output from k=0 through k=5
for k_iter in range(6):
    print(f"k = {k_iter} -> out = {tl.out(k_iter).squeeze()}")
```

```
k = 0 -> out = [1. 0. 0.]
k = 1 -> out = [1. 1. 0.]
k = 2 -> out = [2. 1. 1.]
k = 3 -> out = [0. 2. 1.]
k = 4 -> out = [0. 0. 2.]
k = 5 -> out = [0. 0. 0.]
```

The output of the network can now be simulated, in this case we will consider k from 0 to 9:

In [10]:
```python
# Initialize a
a = np.array([[]])

# Loop
iters = 6
for k in range(iters):
    # Get timestep p
    p = tl.out(k)

    # Simulate output
    ak = np.dot(W, p)  # Retrieve k-th output

    # Save in a array
    if not a.size:
        a = ak
    else:
        a = np.vstack((a, ak))

# Print results
print("Output: a(k) =", a.squeeze(), f"(k from 0 to {iters-1})")
```

```
Output: a(k) = [ 1. -3.  0. -6.  4.  0.] (k from 0 to 5)
```

Which is the response of the filter for each k, when the input is not zero. The output will be simply zero when all the inputs are zero as well.