DSP Lab 05: Discrete systems as functions

1. Objective

Students should check basic properties of digital systems for easy systems implemented in Matlab

2. Theoretical aspects

2.1 Functions in Matlab

Each function in Matlab is created in its own file, according to the following template:

In [ ]:
```
function y = function_name(x, a, b)
% func_name is the name if the function. It must be saved in a file
func_name.m
% x, a, b = the input arguments of the function
% y = the output value of the function.
% If the function returns multiple outputs, write them like: [y1, y
2, y3] = function_name(x, a, b)

end
```

A discrete system can be implemented as a function which takes as input one vector (x) and produces as output another vector (y). The output vector is computed according to the system equation, inside the function.

Example: what is the following function doing?

In [ ]:
```
function y = mistery_function(x)

N = length(x);

y(1) = x(1);
y(2) = x(2) - 2*x(0);
for i=3:N
    y(i) = x(i) - 2*x(i-1) + 0.5*x(i-2);
end
```

Question: why do we need to treat $y(1)$ and $y(2)$ separately, before the `for` loop?

2.2 Functions as arguments to another function

A function can have an input argument another function.

Let's define first a simple function which squares a number:

In [ ]:
```
function y = my_square(x)

y = x^2;

end
```

Let's define now another function, which takes another function as input :

function y = foo(a, b, somefunc)

y = a + somefunc(b);

end

The 3rd argument of the function `foo()` **is a function handle**. Its another function, which is received here under the name `somefunc` , and can be as a function inside our `foo()` function, i.e. by calling `somefunc(b)` .

We don't know yet what `somefunc()` does. This depends on what function is passed as 3rd argument to `foo()` when calling it.

We can pass `my_square()` as the argument to `foo()` as follows:

In [ ]:
```
y = foo(4, 6, @my_square);
```

Inside `foo()` , `my_square` becomes `somefunc` , and the result is computed as $y = 4 + 6^2$.

Note **the special sign @** before the name of the function. It represents **the handle (address)** of the function `my_square()` . It means we're not calling `my_square()` , we just want its location.

Question: what is the result of the call `foo(4, 6, @sqrt)` ?

2.3 Properties of discerte systems

Two fundamental properties of discrete systems are **linearity** and **time-invariance**. You can find more about them in the lectures.

A system is **linear** if it satisfies the following relation:

$$H\{a \cdot x_1[n] + b \cdot x_2[n]\} = a \cdot H\{x_1[n]\} + b \cdot H\{x_2[n]\}$$

A system is **time-invariant** if it satisfies the following relation:

$$H\{x[n-k]\} = y[n-k], \text{ where } y[n] = H\{x[n]\}$$

3. Exercises

1. Create a function `mysys1()` that implements the following system H_1:

$$y[n] = H_1\{x[n]\} = \frac{1}{4}x[n] - \frac{1}{2}x[n-1] + \frac{1}{4}x[n-2]$$

- The function takes one input argument `x` and outputs one vector `y`
- Test the function by running it in on the following input signal x: 20 zeros, followed by 20 ones, repeated 5 times
- Plot the original signal x and the output signal y on the same graph.

1. Check the linearity of the system in `mysys1()` , by checking if the linearity equation holds, in the following way:

- generate two random vectors `x1` and `x2` of some length (e.g. 500) and two random numbers `a` and `b`
- apply the system (e.g. the function `mysys1`) to `a*x1` , `b*x2` , and `a*x1 + b*x2` , and check if the results verify the linearity equation: the sum of the first two results must be equal to the third

1. Create a function to test the linearity of a system, `test_linear()` , in the manner described above.

- the function shall take one input argument, a function handle of the system function, e.g. the function will be called as `test_linear(@mysys1)`
- inside, the function shall do exactly the same procedure as above: generate two random vectors and two constants, apply the system to `a*x1` , `b*x2` , and `a*x1 + b*x2` , and shall check if the results verify the linearity equation
- the check shall be repeated for 5 times, with 5 different randomly generated data
- if the linearity equation holds every time, the function shall return 1; otherwise the return value shall be 0

  Run the function for the `mysys1()` function in Exercise 1, and check whether it is linear or not.

2. Create functions for the following systems as well, and check if they are linear:

$$y[n] = H_1\{x[n]\} = n \cdot x[n] + 5$$

$$y[n] = x[n] + 0.5x[n-1] + 1$$

$$y[n] = (x[n])^2 + 4$$

3. Implement a similar function to **check the time invariance** of a system, following the same approach. We can check time invariance in the following way:

- Apply the system to some random vector `x` . Let's call the result `y` .
- Apply the system to `x` prepended with K values zeros (i.e. delayed by K samples). K can be anything between 1 and whatever. Let's call the result `y2` .
- If the system is time invariant, the vector `y` should be identical to the vector `y2` starting after position K (from (K+1) onwards).

4. Final questions

TBD