Lab 3: Signals and Systems

The central theme of this third (and last) lab session are the *FFT* (Fast Fourier Transform) and the *NTT* (Number Theoretic Transform) and their application to computing convolutions and correlations. The Fast Fourier Transfrom is an algorithm that computes the DFT (Discrete Fourier Transform) in O(N log N) time, where N is the number of samples of the input signal. The DFT is defined as:

$$y[k] = \Sigma_{n=0}^{N-1} x[n] \cdot e^{j2\pi nk/N}, \quad k = 0, 1, ...$$

Its inverse is given by:

$$x[k] = \frac{1}{N} \Sigma_{n=0}^{N-1} y[n] \cdot e^{-j2\pi nk/N}, \quad k = 0, 1, ...$$

Note that this can be rewritten into (where $\omega = e^{j2\pi/N}$):

$$\begin{aligned} y[k] &= \Sigma_{n=0}^{N-1} x[n] \cdot \omega^{nk}, \\ x[k] &= \frac{1}{N} \Sigma_{n=0}^{N-1} y[n] \cdot \omega^{-nk} \end{aligned}$$

We call ω a primitive N-th root of unity since $\omega^N = 1$ and $\omega^x \neq \omega^y$ for all $0 \leq x < y < N$. Note that this property is one of the main reasons why we use complex numbers: the only primitive root that exists in the set of real numbers is -1 for N=2.
However, other domains exist in which we can find primitive roots. For example, if N=10, we can decide to compute modulo the prime number 11 using $\omega = 2$:

$$\begin{aligned} &2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4(= 16) = 5, 2^5 = 10, 2^6(= 20) = 9, \\ &2^7(= 18) = 7, 2^8(= 14) = 3, 2^9 = 6, 2^{10}(= 12) = 1 \end{aligned}$$

This observation is the basis of the NTT, which is nothing else but an FFT using modular arithmetic. For more details, you are referred to the slides (see Nestor) of lecture 6.

Exercises

- Make a function `myconv(x,y)` that returns the convolution of the signals `x` and `y`. Your function should run in O(N log N) time, where N is the maximum of the length (number of samples) of `x` and `y`. You will need the convolution theorem to achieve this. Use the built-in functions `fft` and `ifft`. Your algorithm should produce the same result as the built-in function `conv`.
- Make a (very slow) DFT by constructing a vanderMonde matrix, and multiplying the input with this matrix. Compare your result with the built-in function FFT. Make also the inverse IFT.
- Repeat the previous exercise using prime numbers and roots of unity. On Nestor you can find two C-programs that can be used to find primes, roots of unity and inverses modulo a prime.
- Implement the FFT and IFFT yourself (use a recursive impementation: see slide 36 of lecture 5). Compare with the built-in FFT.
- Modify the code of the previous exercise to implement the NTT and its inverse.
- Repeat the first exercise (`myconv`), but now using NTTs.
- Write a small report about the above exercises, and hand it in before the deadline.