Ex9 — FFT

Delivery deadline: **11 November** 2022.

We highly recommend you to collaborate and interact with each other when working on these tasks. Still, the exercises shall be solved independently and must be assessed in the weekly submission sessions. It is expected that everyone is familiar with the approval requirements for exercises. This is described on the course homepage: `http://www.uio.no/studier/emner/matnat/ifi/IN3190/h22/curriculum-related/`

Observe:
The sum of points in total for the tasks distributed the current week is 12.5, while the maximum score obtainable in one week is 10. The score registered will hence be obtained using the formula `min(your_score_sum, 10)`.

Exercise 1 — DFT / FFT — Points: 3

For the decimation-in-time FFT algorithm we can calculate an N-point DFT of a sequence $x[n]$ from two $N/2$-point DFTs:

$$\begin{aligned} X[k] &= X_{\mathrm{e}}[k] + W_N^k X_{\mathrm{o}}[k], \quad k = 0,\, 1,\, \ldots,\, \frac{N}{2} - 1 \\ X\left[k + \frac{N}{2}\right] &= X_{\mathrm{e}}[k] - W_N^k X_{\mathrm{o}}[k], \quad k = 0,\, 1,\, \ldots,\, \frac{N}{2} - 1. \end{aligned} \tag{1}$$

where $X_{\mathrm{e}}[k]$ and $X_{\mathrm{o}}[k]$ are the DFTs of the even- and odd-numbered samples of $x[n]$.

The computational complexity, i.e. the order of the number of operations, for direct computation of an N-point DFT using the defining formula

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0,\, 1,\, \ldots,\, N-1,$$

where twiddle factors $W_N^{kn} = e^{-j2\pi kn/N}$ are obtained from a look-up table, requires $\mathcal{O}\left(N^2\right)$ complex operations.

Fast Fourier Transform (FFT) algorithms reduce the computational complexity to $\mathcal{O}\left(N \log_2 N\right)$ operations.

a) 4-point DFTs

Let

$$x[n] = \{1,\, \underset{\uparrow}{\tfrac{\sqrt{2}}{2}},\, 0,\, -\tfrac{\sqrt{2}}{2},\, -1,\, -\tfrac{\sqrt{2}}{2},\, 0,\, \tfrac{\sqrt{2}}{2}\}.$$

be an 8-point sequence. What are the two 4-point sequences $x_{\mathrm{e}}[n]$ (even) and $x_{\mathrm{o}}[n]$ (odd) that are used in the first stage of FFT by decimation-in-time?

b) 2-point DFTs

Use Equation (1) to show that the four 2-point DFTs $X_{\mathrm{ee}}[k]$, $X_{\mathrm{oe}}[k]$, $X_{\mathrm{eo}}[k]$ and $X_{\mathrm{oo}}[k]$ calculated at the second stage of FFT by decimation-in-time are given by:

$$\begin{aligned} X_{\mathrm{ee}}[k] &= \{0,\, 2\} \\ X_{\mathrm{oe}}[k] &= \left\{0,\, \sqrt{2}\right\} \\ X_{\mathrm{eo}}[k] &= \{0,\, 0\} \\ X_{\mathrm{oo}}[k] &= \left\{0,\, -\sqrt{2}\right\}. \end{aligned}$$

c) 8-point DFT

Calculate the two 4-point DFTs $X_{\mathrm{e}}[k]$ and $X_{\mathrm{o}}[k]$ using Equation (1), and then calculate the 8-point DFT $X[k]$.

Hint: Is your final answer in accordance with what you would expect from the shape/values of the input sequence?

Exercise 2 — FFT — Points: 2

1. Peter wants to linearly convolve two complex sequences, $x(n)$ with length $N = 1024$, and h(n) with length L. He wonders which method to choose and asks Andreas for help. To be sure to get it right, Andreas asks you:

   - For which lengths L are there fewer multiplications when calculating the convolution $x(n) * h(n)$ directly, than by utilizing the radix-2 FFT algorithm?

   In the latter case, one can find the convolution by taking the inverse DFT of the product $X(k)H(k)$, where $X(k)$ is the DFT of $x(n)$ and $H(k)$ is the DFT of $h(n)$.

2. Delighted with many answers, Peter comes up with even more questions, now directly to you. He has one new sequence $y(n)$ that he is sure is complex and of length 1025 (i.e. 1 more than $N = 2^v$). Instead of disregarding the last data point, he considers zero-extending the sequence to length $N = 2^w$ so that the radix-2 FFT algorithm can be used.

   - How many multiplications and additions are needed to calculate the DFT using the radix-2 FFT algorithm?
   - How many multiplications and additions are needed to calculate a 1025-point DFT directly?

Exercise 3 — Linear vs. circular convolution — Points: 3.5

Consider the two finite length sequences

$$x_1[n] = \{\underset{\uparrow}{1}, -2, 1, -3\}$$

$$x_2[n] = \{\underset{\uparrow}{0}, 2, -1, 0, 0, 4\}$$

a) Determine the linear convolution $x_1[n] * x_2[n]$ Poeng: 1.

b) Determine the 6-point cicular convolution $x_1[n] ⑥ x_2[n]$ Poeng: 1.

c) What should be the smallest value of N so that the N-point circular convolution is equal to the linear convolution? Poeng: 0.5

d)
- When a filtering operation is done on computers it is usually done using multiplications in the frequency domain. What kind of convolution does it correspond to? Poeng: 0.5
- Explain why zero-padding is a useful technique in this case. Poeng: 0.5

Exercise 4 — Correlation, Matlab/Python — Points: 4

In signal processing, cross-correlation (or only "correlation") is a measure of similarity of two series as a function of lag / relative shift between the signals / the displacement of one relative to the other.

The concept of correlation in general quantifies the similarity of two spatial- or time-dependent signals $x[n]$ and $h[n]$. The main property of correlation is that the signals do not have to depend on each other; only statements regarding their similarity can be given. Correlation is, however, not adequate to reveal the existence of a causal relationship between $x[n]$ and $h[n]$.

The discrete (linear) cross-correlation is defined as

$$r_{xh}[n] = \sum_{k=-\infty}^{\infty} x^*[k]h[n+k], \ n \in \mathcal{Z}. \tag{2}$$

The cross-correlation is similar in nature to the convolution of two functions. Discrete convolution of two signals $x[n]$ and $h[n]$ is defined as

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x^*[k]h[n-k], \ n \in \mathcal{Z}. \tag{3}$$

In addition to (linear) cross-correlation, it is possible to define a circular cross-correlation for two finite length sequences $x[n]$ and $h[n]$ of equal length N:

$$r_{xh}^{circ}[n] = \sum_{k=0}^{N-1} x^*[k]h[(n+k)_N]. \tag{4}$$

The notation $(i)_N$ represents the modulo-operator "i mod N", i.e. the remainder when i is divided by N.

In this task you will create some Matlab or Python functions that correlates two input vectors using different methods. To help you solve the task, you can be inspired by the following function:

```
function c = my_corr(a,b,method)
%
% Function MY_CORR correlates two vectors a and b and returns
% the result in vector  c.
%
% 'metod' can have the following values:
% 'lin'    : linear correlaston
% 'circ'   : circular correlaston
% 'FFT'    : correlation using fft and ifft.
%

switch lower(method)
    case {'lin','linear'}
        c = lin_corr(a,b);
    case 'circ'
        c = circ_corr(a,b);
    case {'FFT','fft'}
        c = fft_corr(a,b);
    otherwise
        error('Unknown method');
end

% Plot result
figure(51); %
clf     % clear active figure
set(gcf,'DefaultLineLineWidth',1.5);
set(gcf,'DefaultLineMarkerSize',12);
set(gcf,'DefaultAxesFontSize',14);

subplot(3,1,1);
stem(0:length(a)-1,a);
title(['Method ',upper(method), ...
    ': Plot of ''a'', ''b'', and ''r_{a b}''']);
ylabel 'a';
axis([0 max(length(a),length(b)) -max(-min(a),max(a)) max(-min(a),max(a))]);

subplot(3,1,2);
stem(0:length(b)-1,b);
axis([0 max(length(a),length(b)) -max(-min(b),max(b)) max(-min(b),max(b))]);
ylabel 'b';

subplot(3,1,3);
switch lower(method)
    case {'lin','linear'}
        maxLag = (length(c)-1)/2;
        stem(-maxLag:maxLag,c);
        axis([-maxLag maxLag -max(-min(c),max(c)) max(-min(c),max(c))]);
    otherwise
        stem([0:length(c)-1],c);
        axis([0 length(c)-1 -max(-min(c),max(c)) max(-min(c),max(c))]);
end

ylabel 'r_{a b}';
xlabel 'lag';

% Optionally save the figure
tmp = input('Should the figure be saved as a png-file [y/n]? ... ','s');
if strcmp(tmp,'y')
    tmp = input('Please provide a figure name [figur] ... ','s');
    if length(tmp) == 0,
        tmp = 'figur';
    end
    cmd = ['print -dpng -r100 ',tmp];
    eval(cmd);
end
```

1. Write a matlab (or phyton) function 'function c = lin_corr(a,b)' which correlates two vectors 'a' and 'b'.

The function should use for-loop(s) and the implementation should be as close as possible to the definition of correlation given in (2).

Test the function on simple vectors 'a' and 'b'. Check that it gives the same answer as the matlab/phyton function 'xcorr'.

2. A circular shift is the operation of rearranging the entries in a tuple. It shifts the elements in array 'x' by 'k' positions, mathematically expressed as:

$$x'[n] = x(n-k, \text{modulo } N) \equiv x((n-k))_N$$

Write a matlab function 'function y = cirk_shift(x,k)' that takes as input a sequence (vector) 'a' and a shift N, and returns a sequence (vector) 'y' which is a circular shifted version of 'x', shifted 'k' indexes (positive or negative).

Test the function on simple vectors and check that the function gives same answer as matlab function 'circshift' or numpy function 'roll'.

3. Write a matlab (or phyton) function 'function c = circ_corr(a,b)' which circular-correlates two vectors 'a' and 'b'.

The function must use for-loop(s) and a function that circularly shifts a sequence. The implementation should be as close as possible to the definition of circular correlation given in (4).

To test your code, you can zero pad your test vectors 'a' and 'b' appropriatrly and compare the result to the one obtained with linear correlation.

4. Write a matlab (pr phyton) function 'function c = fft_corr(a,b)' which circularly correlates two vectors 'a' and 'b' using DFT (fft).

Test the function on simple vectors 'a' and 'b'. Check that the results are the same as you get with your function 'circ_corr'.

For all four tasks, report some plots showing an example and some proof that you results obtained through comparisons are equal.