Efe
Tarhan
22002840

EEE424 Digital Signal Processing Homework 4

```
%% Generate h[n]:

h(1:120) = 0.96.^(0:119);
```

h[n] is generated by using the equation described in the homework procedure: $h[n] = 0.96^n$ for $n = 0, 1, 2 \dots 119$. After that a sound file has been read with MATLAB's "audioread()" function and stored in three different $x_i[n]$ for $i = 1, 2, 3$. These input signals are for parts a, b and c separately. Zeros were added to the input signal from the beginning and the end by at least the amount of the length of the impulse response to avoid any algorithm-based problems.

```
%% Generate x[n]: Nx is exactly 120.000 with 119 zeros
% before 0 and 120 zeros after  120.000
%x[0] = x(120)
[x,fsh] = audioread("eric_clapton.mp3");
i = 2.9*10^6;
x = x(i:i+120000-1+240-2);
x(1:119) = 0;
x(120120:120239) = 0;
x1 = x(1:120239);
x2 = x(1:120239);
x3 = x(1:120239);
% The nonzero part of the x is between indexes 120-120119 inclusive
```

Part A)

In part A, the output of the input to the given impulse response was found by using linear convolution.
Since the output signal y[n] is 120120 points long and there are i = 1, …, 120 multiplications for each data point which has been done with a for loop, the number of multiplications is the following:

$$120.120 \cdot 120 = 14.414.400$$

This is a measure of the algorithmic complexity of the filtering operation.

```
%% PART A
% Both x and h has real values therfore there will be
% 120 multiplications for each array element therefore;
% there will be 120120*120 = 14.414.400  = (Nx+Nh) . Nh
y1(1:120120) = 0;

for i = (1:120)
    y1(1:120120) = y1(1:120120) + h(i).*x1((120:120239)-i+1);
end
```

Part B)

For circular convolution to be equal to the linear convolution, the circular convolution size N must satisfy the following relation:

$$N \geq N_h + N_s - 1$$

Where N_h is the length of the impulse response and N_s is the length of the segment of the input for the case where the impulse response is finite and the input signal is indefinitely long, then if there is a fixed size of N and the selectable variable is the length of the N_s. The size of the N_s will satisfy the following criteria for $N = 2048$ and $N_h = 120$:

$$1929 \geq N_s$$

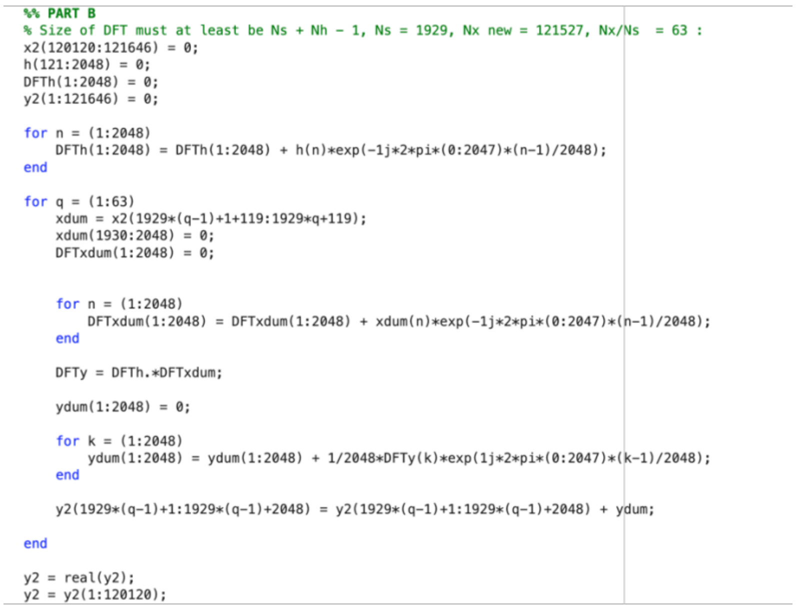
Therefore, the input segments will have a at most a length of 1929 for circular convolution to be equal to the linear convolution. 1527 zeros will be added to the end of the input signal to make it possible for it to be divided to pieces with length 1929. After that the resultant output piece $y_r[n]$ can be found with the following operations:

$$y_r[n] = IDFT_{2048}\{DFT_{2048}\{x_r[n]\} \cdot DFT_{2048}\{h[n]\}\}$$

$$x_r[n] = \begin{cases} x[n - r \cdot 1929], & n \in 0 \ldots 1928 \\ 0, & else \end{cases}$$

$$y_r[n] = \frac{1}{2048} \sum_{n=0}^{2047} \left[\left(\sum_{n=0}^{2047} x_r[n] \cdot e^{\frac{-j2\pi kn}{2048}} \right) \cdot \left(\sum_{n=0}^{2047} h[n] \cdot e^{\frac{-j2\pi kn}{2048}} \right) \right] \cdot e^{\frac{j2\pi kn}{2048}}$$

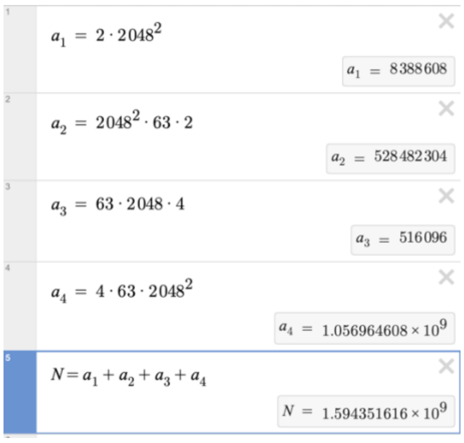
After finding the output result of the given input segment $x_r[n]$, the output pieces are combined by adding them on top of each other by shifting them. The algorithm is *overlap-and-add* method. Code of the described operation is the following:



```
%% PART B
% Size of DFT must at least be Ns + Nh - 1, Ns = 1929, Nx new = 121527, Nx/Ns  = 63 :
x2(120120:121646) = 0;
h(121:2048) = 0;
DFTh(1:2048) = 0;
y2(1:121646) = 0;

for n = (1:2048)
    DFTh(1:2048) = DFTh(1:2048) + h(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
end

for q = (1:63)
    xdum = x2(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = 0;


    for n = (1:2048)
        DFTxdum(1:2048) = DFTxdum(1:2048) + xdum(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
    end

    DFTy = DFTh.*DFTxdum;

    ydum(1:2048) = 0;

    for k = (1:2048)
        ydum(1:2048) = ydum(1:2048) + 1/2048*DFTy(k)*exp(1j*2*pi*(0:2047)*(k-1)/2048);
    end

    y2(1929*(q-1)+1:1929*(q-1)+2048) = y2(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y2 = real(y2);
y2 = y2(1:120120);
```

Again, the complexity of the algorithm can be found by counting the number of real multiplications of the code segment. The contributions to multiplications are the following:

- $a_1 = \# \ of \ DFT_{2048} \ of \ h[n] : 2048^2 \ complex \ multiplications$. Where one signal is purely real and the other is complex. Therefore, the contribution is: $2 \cdot 2048^2$.
- $a_2 = \# \ of \ DFT_{2048} \ of \ x_r[n]$: The loop turns for 63 times so, there are $63 \cdot 2048^2$complex multiplications. Where one signal is purely real and the other is complex. Therefore, the contribution is: $2 \cdot 63 \cdot 2048^2$.
- $a_3 = \# \ of \ X_r[k] \cdot H[k]$: The loop turns for 63 times so, there are $63 \cdot 2048^2$ complex multiplications. Where both signals are complex. Therefore, the contribution is: $4 \cdot 63 \cdot 2048^2$.
- $a_4 = \# \ of \ IDFT_{2048} \ of \ y_r[n]$: The loop turns for 63 times so, there are $63 \cdot 2048^2$ complex multiplications. Where both signals are complex. Therefore, the contribution is: $4 \cdot 63 \cdot 2048^2$.



$a_1 = 2 \cdot 2048^2$

$a_1 = 8388608$

$a_2 = 2048^2 \cdot 63 \cdot 2$

$a_2 = 528482304$

$a_3 = 63 \cdot 2048 \cdot 4$

$a_3 = 516096$

$a_4 = 4 \cdot 63 \cdot 2048^2$

$a_4 = 1.056964608 \times 10^9$

$N = a_1 + a_2 + a_3 + a_4$

$N = 1.594351616 \times 10^9$

As a total sum there are nearly 1.6 billion multiplications in this algorithm to find the output of the system.

Part C)

MATLAB uses an FFT algorithm called FFTW (The Fastest Fourier Transform in the West) which selects the best or the most efficient algorithm to organize the codelets (name of the highly optimized composable blocks of C code) depending on the size of the data and computational hardware capabilities of the computer that the MATLAB will be operated on. The main analytical FFT algorithm that FFTW uses is the radix-$\sqrt{N}$ Cooley-Tukey algorithm which is a decimation-in-time type of algorithm. The radix of the algorithm can change depending on the hardware properties of the device. The code 'fftw('planner')' and 'fftw('wisdom')' selects the best FFT algorithm type depending on the properties of the computer and the input data. [1]

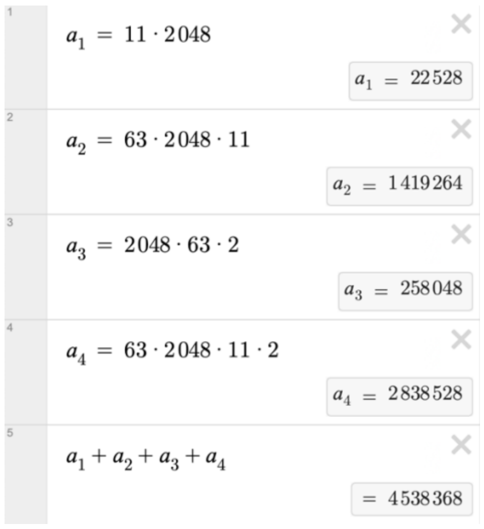
The 2048-pt FFT conducted with radix-2 Cooley-Tukey Algorithm has a complexity of $\frac{N}{2} log_2(N)$ which is the number of complex multiplications. Complexity of this algorithm can be found with the the number of real multiplications in the following algorithm.

- $a_1 = \#\ of\ FFT_{2048}\ of\ h[n]\ :\ 11 \cdot 2048\ real\ multiplications.$

- $a_2 = \#\ of\ FFT_{2048}\ of\ x_r[n]$: The loop turns for 63 times so, there are $63 \cdot 2048 \cdot 11$ real multiplications.

- $a_3 = \#\ of\ X_r[k] \cdot H[k]$: The loop turns for 63 times so, there are $63 \cdot 2048 \cdot 2$ real multiplications.

- $a_4 = \#\ of\ IFFT_{2048}\ of\ y_r[n]$: The loop turns for 63 times so, there are $63 \cdot 2048 \cdot 11 \cdot 2$ real multiplications.



$a_1 = 11 \cdot 2048$

$a_1 = 22528$

$a_2 = 63 \cdot 2048 \cdot 11$

$a_2 = 1419264$

$a_3 = 2048 \cdot 63 \cdot 2$

$a_3 = 258048$

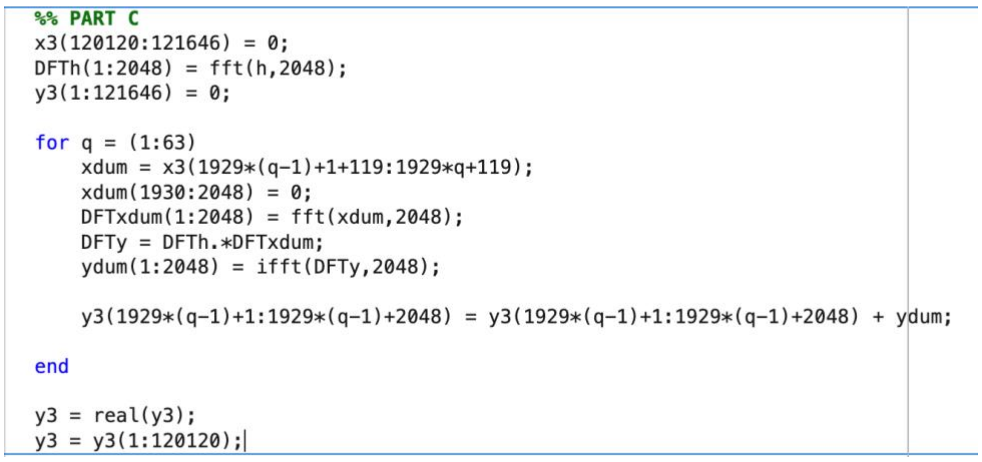
$a_4 = 63 \cdot 2048 \cdot 11 \cdot 2$

$a_4 = 2838528$

$a_1 + a_2 + a_3 + a_4$

$= 4538368$

As a result, it can be stated that the number of real multiplications is nearly equal to 4.54 million which shows that algorithmic complexity is fairly decreased with the usage of FFT compared with the other algorithms.



```
%% PART C
x3(120120:121646) = 0;
DFTh(1:2048) = fft(h,2048);
y3(1:121646) = 0;

for q = (1:63)
    xdum = x3(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = fft(xdum,2048);
    DFTy = DFTh.*DFTxdum;
    ydum(1:2048) = ifft(DFTy,2048);

    y3(1929*(q-1)+1:1929*(q-1)+2048) = y3(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y3 = real(y3);
y3 = y3(1:120120);
```

If we compare all three algorithms it can be stated that the algorithmic complexity of FFT based method is the least while the method with direct DFT calculation has the most.

Part D)

The errors between the outputs of different computation techniques are calculated numerically by using MATLAB with the given formulas the results are the following:

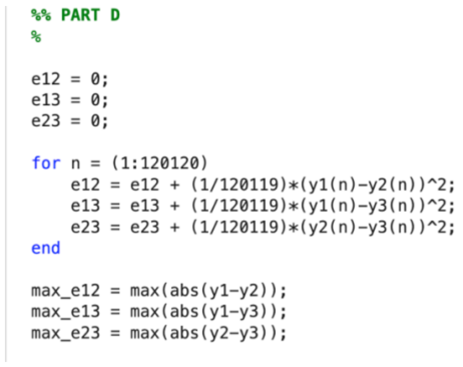
$$\epsilon_{12} = 4.3983 \cdot 10^{-25}$$
$$\epsilon_{13} = 1.455 \cdot 10^{-30}$$
$$\epsilon_{12} = 4.3979 \cdot 10^{-25}$$

The first source of error is the numerical limitation of the storage of the real numbers where only several digits can be stored for each number. This problem occurs for the impulse response since it is not a short rational number or a whole number. Also, when computing the DFT for the impulse response and the input pieces the complex exponentials are estimated with Taylor Series and therefore will introduce numerical errors. If we consider that the error in the impulse response effects all three outputs in a similar manner, the main source of the error as the differences between the output is the operated algorithm. The difference between the result of the FFT algorithm that MATLAB uses, and the result of the convolution is 10000 times smaller than the result obtained by the handwritten DFT algorithm that gives the second output. Therefore, it can be argued that MATLAB uses a more precise and accurate algorithm for the computation of the DFT with FFT than the handwritten code. But the main source of error in this case is mostly the multiplication with complex exponentials.

The errors defined in the second half of this part are also given below:

$$\max(|y_1 - y_2|) = \epsilon_{12} = 4.3983 \cdot 10^{-25}$$
$$\max(|y_1 - y_3|) = \epsilon_{13} = 1.455 \cdot 10^{-30}$$
$$\max(|y_2 - y_3|) = \epsilon_{23} = 4.3979 \cdot 10^{-25}$$

Since the found errors are small it can be stated that both algorithms are correct and well-defined.



```
%% PART D
%

e12 = 0;
e13 = 0;
e23 = 0;

for n = (1:120120)
    e12 = e12 + (1/120119)*(y1(n)-y2(n))^2;
    e13 = e13 + (1/120119)*(y1(n)-y3(n))^2;
    e23 = e23 + (1/120119)*(y2(n)-y3(n))^2;
end

max_e12 = max(abs(y1-y2));
max_e13 = max(abs(y1-y3));
max_e23 = max(abs(y2-y3));
```

Part E)

The analytic steady state output to the system can be found by taking the Fourier Transform if it is defined, i.e, if the system is BIBO stable which requires the condition that the $|z| = 1 \in ROC$, where $x[n] = \cos(0.05\pi n)$ and $y[n] = 0.96^n$. The result can be found with the following computations.

$$x[n] = \cos(0.05\pi n) = \frac{1}{2}e^{j0.05\pi n} + \frac{1}{2}e^{-j0.05\pi n}$$

At first look x[n] might be thought to be an eigenfunction of the LTI system therefore finding the output is easy but it must be noted that x[n] is finite duration and therefore is not a periodic complex exponential.

However, the sinusoidal steady state response of the LTI system can still be found by using the Fourier Transform since it eliminates the decaying exponentials, but it becomes a hard work compared to the direct convolution when both the x[n] and the y[n] appears to be finite. So, the steady state output of the system can be found by finding the convolution part that h[n] appears fully inside the length of x[n]. Then the computation of the steady state part of the output $y_4[n]$ can be done by the following:

$$y_{ss4}[n] = \sum_{k=119}^{119999} h[n-k] \cdot x[k]$$

k is taken from n-119 to n since it is the interval that the h[n-k] fully locates inside the signal x[n] and therefore the response is the steady state. It can also be proven that this method gives the same result with fully taking the convolution and eliminating the decaying exponentials.

$$= y_{ss4}[n] = \sum_{k=n-119}^{n} 0.96^{n-k} \cdot \cos(0.05 \cdot \pi \cdot k)$$

$$= \sum_{k=n-119}^{n} 0.96^{n-k} \cdot \cos(0.05 \cdot \pi \cdot k)$$

$$= \frac{1}{2} \sum_{k=n-119}^{n} 0.96^{n-k} \cdot \left(e^{j0.05\pi k} + e^{-j0.05\pi k}\right)$$

$$= \frac{1}{2} \cdot 0.96^n \sum_{k=n-119}^{n} \left(e^{j0.05\pi} \cdot 0.96^{-1}\right)^k + \left(e^{-j0.05\pi} \cdot 0.96^{-1}\right)^k$$

$$= \frac{1}{2} \cdot 0.96^n \left[\frac{\left(e^{j0.05\pi} \cdot 0.96^{-1}\right)^{n-119} - \left(e^{j0.05\pi} \cdot 0.96^{-1}\right)^{n+1}}{1 - (e^{j0.05\pi} \cdot 0.96^{-1})} + \frac{\left(e^{-j0.05\pi} \cdot 0.96^{-1}\right)^{n-119} - \left(e^{-j0.05\pi} \cdot 0.96^{-1}\right)^{n+1}}{1 - (e^{-j0.05\pi} \cdot 0.96^{-1})}\right]$$

$$= \frac{1}{2}\left[\frac{0.96^{119}(e^{j0.05\pi})^{n-119} - 0.96^{-1}(e^{j0.05\pi})^{n+1}}{1 - (e^{j0.05\pi} \cdot 0.96^{-1})} + \frac{0.96^{119}(e^{-j0.05\pi})^{n-119} - 0.96^{-1}(e^{-j0.05\pi})^{n+1}}{1 - (e^{-j0.05\pi} \cdot 0.96^{-1})}\right]$$

$$= \frac{1}{2}\left[\frac{0.96^{119}\left(e^{j0.05\pi(n+1)}\right) - 0.96^{-1}\left(e^{j0.05\pi(n+1)}\right)}{1-\left(e^{j0.05\pi}\cdot 0.96^{-1}\right)} + \frac{0.96^{119}\left(e^{-j0.05\pi(n+1)}\right) - 0.96^{-1}\left(e^{-j0.05\pi(n+1)}\right)}{1-\left(e^{-j0.05\pi}\cdot 0.96^{-1}\right)}\right]$$

$$= \frac{\left(0.96^{119} - 0.96^{-1}\right)}{2}\left[\frac{\left(e^{j0.05\pi(n+1)}\right)}{1-\left(e^{j0.05\pi}\cdot 0.96^{-1}\right)} + \frac{\left(e^{-j0.05\pi(n+1)}\right)}{1-\left(e^{-j0.05\pi}\cdot 0.96^{-1}\right)}\right]$$

$$= \frac{\left(0.96^{119} - 0.96^{-1}\right)}{2}\cdot\frac{\left(e^{j0.05\pi(n+1)}\right)\cdot\left(1-\left(e^{-j0.05\pi}\cdot 0.96^{-1}\right)\right) + \left(e^{-j0.05\pi(n+1)}\right)\cdot\left(1-\left(e^{j0.05\pi}\cdot 0.96^{-1}\right)\right)}{1+0.96^{-2} - 2\cdot 0.96^{-1}\cdot\cos(0.05\pi n)}$$

$$= \frac{0.96^{119} - 0.96^{-1}}{1+0.96^{-2} - 2\cdot 0.96^{-1}\cdot\cos(0.05\pi n)}\cdot\left[\cos\left(0.05\pi(n+1)\right) - 0.96^{-1}\cos(0.05\pi n)\right]$$

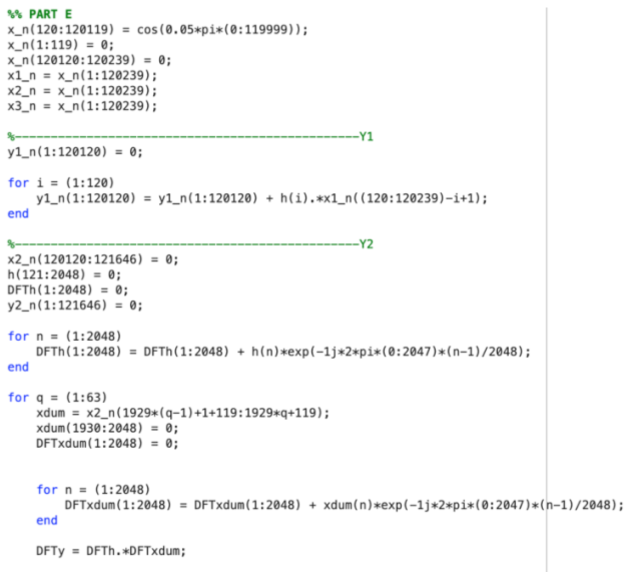
$$= \boxed{\frac{1-0.96^{120}}{1+0.96^{-2} - 2\cdot 0.96^{-1}\cdot\cos(0.05\pi n)}\cdot\left[\cos(0.05\pi n) - 0.96\cdot\cos\left(0.05\pi(n+1)\right)\right]} = y_{ss4}[n]\ for\ n \in 199,\dots,119999$$

After finding the result analytically, the steady state outputs are also computed numerically with the previously defined algorithms. The outputs are compared with a technique like the previous part and the results are the following:

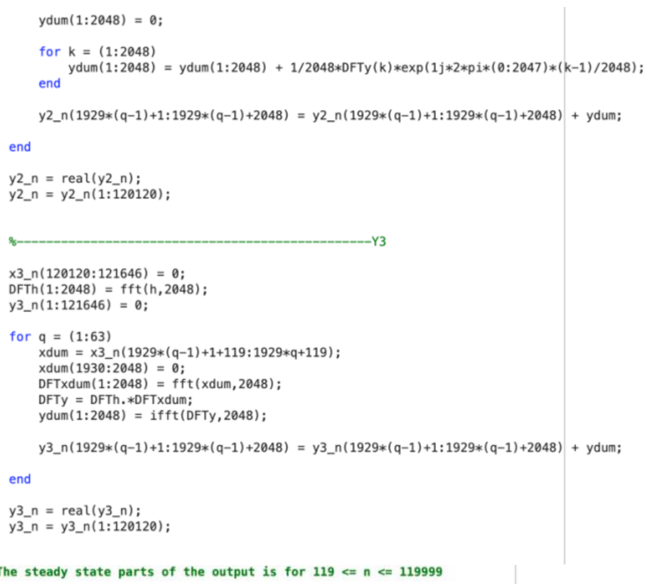
$$\begin{aligned}\epsilon_{41} &= 2.8540\cdot 10^{-22}\\ \epsilon_{42} &= 2.8557\cdot 10^{-22}\\ \epsilon_{43} &= 2.8540\cdot 10^{-22}\end{aligned}$$

And the maximum error component in the difference of the outputs are the following for three comparisons of the analytical result with the numerical results:

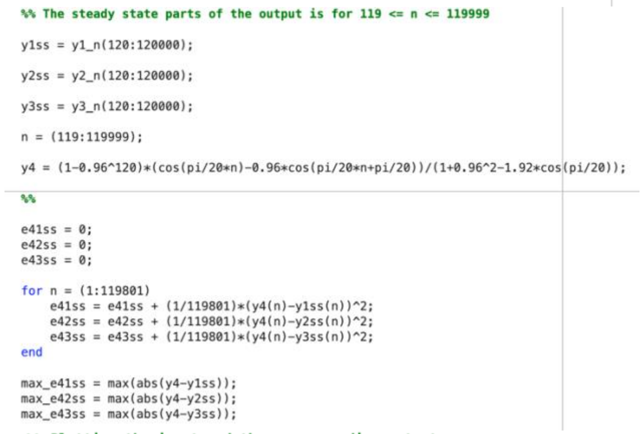
$$\begin{aligned}\max(|y_4 - y_1|) &= \epsilon_{41} = 6.7373\cdot 10^{-11}\\ \max(|y_4 - y_2|) &= \epsilon_{42} = 6.8370\cdot 10^{-11}\\ \max(|y_4 - y_3|) &= \epsilon_{43} = 6.7367\cdot 10^{-11}\end{aligned}$$



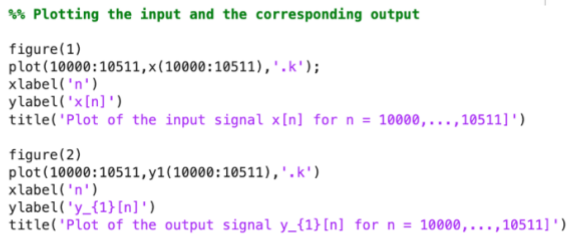
```
%% PART E
x_n(120:120119) = cos(0.05*pi*(0:119999));
x_n(1:119) = 0;
x_n(120120:120239) = 0;
x1_n = x_n(1:120239);
x2_n = x_n(1:120239);
x3_n = x_n(1:120239);

%-----------------------------------------------Y1
y1_n(1:120120) = 0;

for i = (1:120)
    y1_n(1:120120) = y1_n(1:120120) + h(i).*x1_n((120:120239)-i+1);
end

%-----------------------------------------------Y2
x2_n(120120:121646) = 0;
h(121:2048) = 0;
DFTh(1:2048) = 0;
y2_n(1:121646) = 0;

for n = (1:2048)
    DFTh(1:2048) = DFTh(1:2048) + h(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
end

for q = (1:63)
    xdum = x2_n(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = 0;


    for n = (1:2048)
        DFTxdum(1:2048) = DFTxdum(1:2048) + xdum(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
    end

    DFTy = DFTh.*DFTxdum;
```



```
        ydum(1:2048) = 0;

        for k = (1:2048)
            ydum(1:2048) = ydum(1:2048) + 1/2048*DFTy(k)*exp(1j*2*pi*(0:2047)*(k-1)/2048);
        end

        y2_n(1929*(q-1)+1:1929*(q-1)+2048) = y2_n(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

    end

    y2_n = real(y2_n);
    y2_n = y2_n(1:120120);


    %-----------------------------------------------Y3

    x3_n(120120:121646) = 0;
    DFTh(1:2048) = fft(h,2048);
    y3_n(1:121646) = 0;

    for q = (1:63)
        xdum = x3_n(1929*(q-1)+1+119:1929*q+119);
        xdum(1930:2048) = 0;
        DFTxdum(1:2048) = fft(xdum,2048);
        DFTy = DFTh.*DFTxdum;
        ydum(1:2048) = ifft(DFTy,2048);

        y3_n(1929*(q-1)+1:1929*(q-1)+2048) = y3_n(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

    end

    y3_n = real(y3_n);
    y3_n = y3_n(1:120120);
```



```
%% The steady state parts of the output is for 119 <= n <= 119999

y1ss = y1_n(120:120000);

y2ss = y2_n(120:120000);

y3ss = y3_n(120:120000);

n = (119:119999);

y4 = (1-0.96^120)*(cos(pi/20*n)-0.96*cos(pi/20*n+pi/20))/(1+0.96^2-1.92*cos(pi/20));
%%

e41ss = 0;
e42ss = 0;
e43ss = 0;

for n = (1:119801)
    e41ss = e41ss + (1/119801)*(y4(n)-y1ss(n))^2;
    e42ss = e42ss + (1/119801)*(y4(n)-y2ss(n))^2;
    e43ss = e43ss + (1/119801)*(y4(n)-y3ss(n))^2;
end

max_e41ss = max(abs(y4-y1ss));
max_e42ss = max(abs(y4-y2ss));
max_e43ss = max(abs(y4-y3ss));
```



```
%% Plotting the input and the corresponding output

figure(1)
plot(10000:10511,x(10000:10511),'.k');
xlabel('n')
ylabel('x[n]')
title('Plot of the input signal x[n] for n = 10000,...,10511]')

figure(2)
plot(10000:10511,y1(10000:10511),'.k')
xlabel('n')
ylabel('y_{1}[n]')
title('Plot of the output signal y_{1}[n] for n = 10000,...,10511]')
```

There are several sources of error in the implementation of these algorithms and they will be explained one by one:

- *64-Bit Storage:* The register sizes in a computer is the hardware limitation that causes errors when conducting digital signal processing or any calculation tasks. After the filling the 64-bit digit limit the numbers will be rounded or cut depending on the defined data size. Therefore, the number precision can only go up to a certain limit.

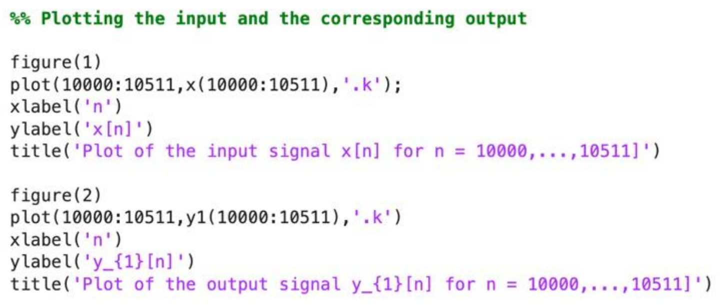
- *Taylor Approximation:* Computation of the DFT or FFT of a sequence or working with any type of complex exponential signal requires the usage of the values of sine and cosine functions. Since the values of these functions cannot be found like polynomials the computer uses Taylor Approximation for the calculation. Again because of the limitations of the computer CPU and memory the computer can only approximate them up to a certain differential order. This might also create problems.

When three techniques for the given task are compared it can be stated that the regular linear convolution is the best technique for the most accurate numerical data computation since it doesn't need the usage of complex exponentials if the input doesn't contain any. The second-best algorithm is the FFT since the usage of complex exponentials and multiplications among them are less than the direct DFT calculation. The direct DFT calculation is the worst technique for result accuracy because it requires the greatest number of multiplications or other operations with complex exponentials.

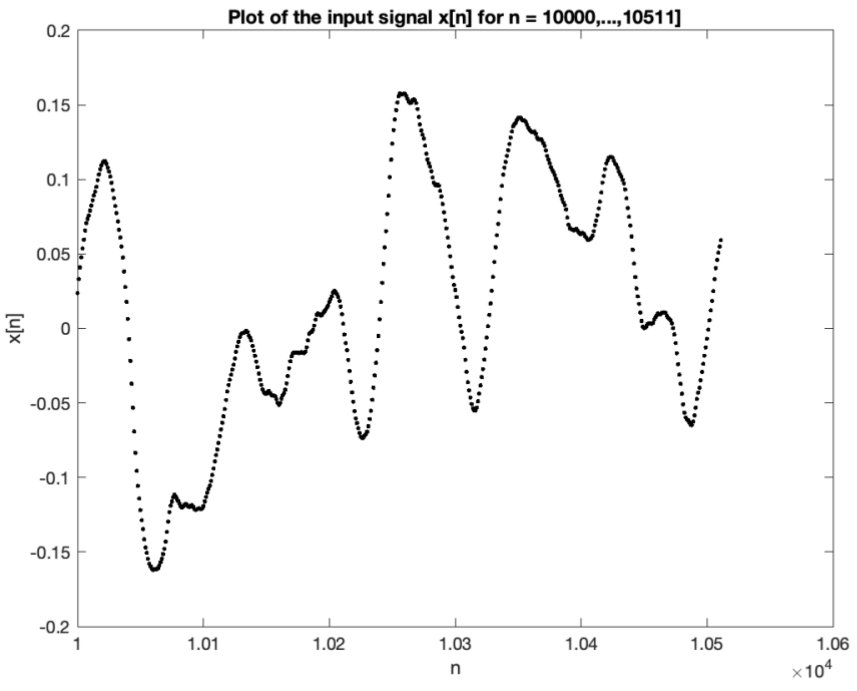
For an additional speed comparison FFT operates the fastest since the number of real multiplications for FFT that determines the algorithmic complexity is less than the other two algorithms. The second-best technique for speed is convolution while the last is again the direct DFT calculation.

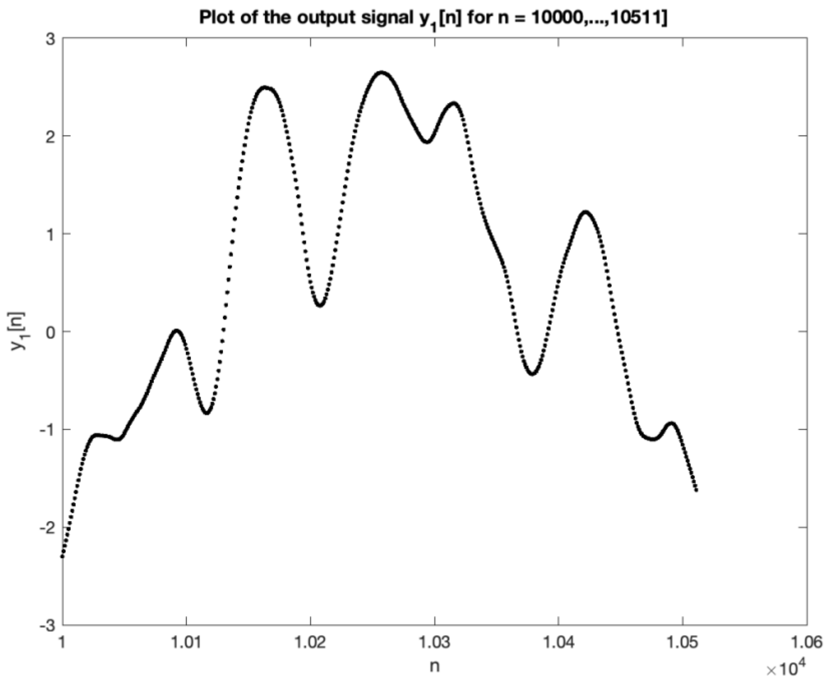
Part F)

For the plot the main input x[n] and its corresponding output obtained in Part A is used for the interval $n \in 10000, \ldots, 10511$.



```
%% Plotting the input and the corresponding output

figure(1)
plot(10000:10511,x(10000:10511),'.k');
xlabel('n')
ylabel('x[n]')
title('Plot of the input signal x[n] for n = 10000,...,10511]')

figure(2)
plot(10000:10511,y1(10000:10511),'.k')
xlabel('n')
ylabel('y_{1}[n]')
title('Plot of the output signal y_{1}[n] for n = 10000,...,10511]')
```





Sources

[1] F. Matteo and S.G. Johnson, 'The Fastest FFT Algorithm in the West', *Massachusetts Institute of Technology*. Sep 11, 1997.[Online] Available: https://www.fftw.org/fftw-paper.pdf. [Accessed: Apr 15 2023].

The Written MATLAB Code:

```
%%                    EEE424 HOMEWORK4 EFE TARHAN



%% Generate h[n]:


h(1:120) = 0.96.^(0:119);


%% Generate x[n]: Nx is exactly 120.000 with 119 zeros
% before 0 and 120 zeros after  120.000
%x[0] = x(120)
[x,fsh] = audioread("eric_clapton.mp3");
i = 2.9*10^6;
x = x(i:i+120000-1+240-2);
```

```
x(1:119) = 0;
x(120120:120239) = 0;
x1 = x(1:120239);
x2 = x(1:120239);
x3 = x(1:120239);
% The nonzero part of the x is between indexes 120-120119 inclusive
%% PART A
% Both x and h has real values therfore there will be
% 120 multiplications for each array element therefore;
% there will be 120120*120 = 14.414.400  = (Nx+Nh) . Nh
y1(1:120120) = 0;

for i = (1:120)
    y1(1:120120) = y1(1:120120) + h(i).*x1((120:120239)-i+1);
end

%% PART B
% Size of DFT must at least be Ns + Nh - 1, Ns = 1929, Nx new = 121527, Nx/Ns  = 63 :
x2(120120:121646) = 0;
h(121:2048) = 0;
DFTh(1:2048) = 0;
y2(1:121646) = 0;

for n = (1:2048)
    DFTh(1:2048) = DFTh(1:2048) + h(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
end

for q = (1:63)
    xdum = x2(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = 0;



    for n = (1:2048)
        DFTxdum(1:2048) = DFTxdum(1:2048) + xdum(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
    end
    DFTy = DFTh.*DFTxdum;

    ydum(1:2048) = 0;

    for k = (1:2048)
        ydum(1:2048) = ydum(1:2048) + 1/2048*DFTy(k)*exp(1j*2*pi*(0:2047)*(k-1)/2048);
    end

    y2(1929*(q-1)+1:1929*(q-1)+2048) = y2(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y2 = real(y2);
y2 = y2(1:120120);
%% PART C
x3(120120:121646) = 0;
DFTh(1:2048) = fft(h,2048);
y3(1:121646) = 0;

for q = (1:63)
    xdum = x3(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = fft(xdum,2048);
    DFTy = DFTh.*DFTxdum;
    ydum(1:2048) = ifft(DFTy,2048);

    y3(1929*(q-1)+1:1929*(q-1)+2048) = y3(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y3 = real(y3);
y3 = y3(1:120120);
%% PART D
%
```

```
e12 = 0;
e13 = 0;
e23 = 0;

for n = (1:120120)
    e12 = e12 + (1/120119)*(y1(n)-y2(n))^2;
    e13 = e13 + (1/120119)*(y1(n)-y3(n))^2;
    e23 = e23 + (1/120119)*(y2(n)-y3(n))^2;
end

max_e12 = max(abs(y1-y2));
max_e13 = max(abs(y1-y3));
max_e23 = max(abs(y2-y3));

%% PART E
x_n(120:120119) = cos(0.05*pi*(0:119999));
x_n(1:119) = 0;
x_n(120120:120239) = 0;
x1_n = x_n(1:120239);
x2_n = x_n(1:120239);
x3_n = x_n(1:120239);

%------------------------------------------------Y1
y1_n(1:120120) = 0;

for i = (1:120)
    y1_n(1:120120) = y1_n(1:120120) + h(i).*x1_n((120:120239)-i+1);
end

%------------------------------------------------Y2
x2_n(120120:121646) = 0;
h(121:2048) = 0;
DFTh(1:2048) = 0;
y2_n(1:121646) = 0;

for n = (1:2048)
```

```
    DFTh(1:2048) = DFTh(1:2048) + h(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
end

for q = (1:63)
    xdum = x2_n(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = 0;


    for n = (1:2048)
        DFTxdum(1:2048) = DFTxdum(1:2048) + xdum(n)*exp(-1j*2*pi*(0:2047)*(n-1)/2048);
    end

    DFTy = DFTh.*DFTxdum;

    ydum(1:2048) = 0;

    for k = (1:2048)
        ydum(1:2048) = ydum(1:2048) + 1/2048*DFTy(k)*exp(1j*2*pi*(0:2047)*(k-1)/2048);
    end

    y2_n(1929*(q-1)+1:1929*(q-1)+2048) = y2_n(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y2_n = real(y2_n);
y2_n = y2_n(1:120120);


%-----------------------------------------------Y3

x3_n(120120:121646) = 0;
DFTh(1:2048) = fft(h,2048);
y3_n(1:121646) = 0;

for q = (1:63)
```

```
    xdum = x3_n(1929*(q-1)+1+119:1929*q+119);
    xdum(1930:2048) = 0;
    DFTxdum(1:2048) = fft(xdum,2048);
    DFTy = DFTh.*DFTxdum;
    ydum(1:2048) = ifft(DFTy,2048);

    y3_n(1929*(q-1)+1:1929*(q-1)+2048) = y3_n(1929*(q-1)+1:1929*(q-1)+2048) + ydum;

end

y3_n = real(y3_n);
y3_n = y3_n(1:120120);


%% The steady state parts of the output is for 119 <= n <= 119999

y1ss = y1_n(120:120000);

y2ss = y2_n(120:120000);

y3ss = y3_n(120:120000);

n = (119:119999);

y4 = (1-0.96^120)*(cos(pi/20*n)-0.96*cos(pi/20*n+pi/20))/(1+0.96^2-1.92*cos(pi/20));

%%

e41ss = 0;
e42ss = 0;
e43ss = 0;

for n = (1:119801)
    e41ss = e41ss + (1/119801)*(y4(n)-y1ss(n))^2;
    e42ss = e42ss + (1/119801)*(y4(n)-y2ss(n))^2;
    e43ss = e43ss + (1/119801)*(y4(n)-y3ss(n))^2;
```

```
end

max_e41ss = max(abs(y4-y1ss));
max_e42ss = max(abs(y4-y2ss));
max_e43ss = max(abs(y4-y3ss));

%% Plotting the input and the corresponding output

figure(1)
plot(10000:10511,x(10000:10511),'.k');
xlabel('n')
ylabel('x[n]')
title('Plot of the input signal x[n] for n = 10000,...,10511]')

figure(2)
plot(10000:10511,y1(10000:10511),'.k')
xlabel('n')
ylabel('y_{1}[n]')
title('Plot of the output signal y_{1}[n] for n = 10000,...,10511]')
```