ESE 531: Homework 7

Noah Schwab

April 4, 2022

Problem solutions with figures are shown below. Work and code is shown in attachments at end of document.

5.39

a) The ROC of the system function is $|z| > \frac{1}{3}$, which means that the system is right-sided and causal. See attached work for pole-zero plot.

b) $H_{\min}(z) = \frac{-2(1-\frac{1}{2}z^{-1})}{1-\frac{1}{3}z^{-1}}$

$H_{\text{lin}}(z) = \frac{z^{-1}(z^{-1}-\frac{1}{2})}{1-\frac{1}{2}z^{-1}}$

7.15

Hann: M+1 = 81

Hamming: M+1 = 81

Blackman: M+1 = 121

7.45

a) $h_d[n] = \frac{\sin^2(\frac{\pi}{4}n)}{\pi^2 n^2}$

b) $H(e^{j\omega})$ has the form $A(e^{j\omega})e^{-j\omega M/2}$ and minimzes the integral squared difference between $A(e^{j\omega})$ and $H_d(e^{j\omega})$. Therefore, $A(e^{j\omega})$ approximates $H_d(e^{j\omega})$, and we know that $H_d(e^{j\omega})$ is an even function. $h[n]$ must be even-symmetric about $M/2$ in the range $0 \leq n \leq M$.

c)

$$h[n] = \begin{cases} \frac{\sin^2 \frac{\pi}{4}(n-M/2)}{\pi^2(n-M/2)^2} & 0 \leq n \leq M \\ 0 & otherwise \end{cases}$$

d) $\epsilon^2 = 2\sum_{M/2+1}^{\infty} |h[n]|^2$

8.14

$x_3[2] = 9$

Matlab problem 1: Frequency Response for Difference Equations

a) See function DFT_A(x). This function works by iterating through each value of k in the frequency domain, as well as each value of n in the time domain, and computes the expression $x[n]e^{-j2\pi kn/N}$. The DFT value at a given discrete frequency, $X[k]$ is the sum of all of those expressions of k for a given value of n. I tested my function with a cosine of 3 different lengths: N=50, 100, and 1000. In the table below I have reported the run time for my two-loop function, as well as the built-in fft method from the NumPy package.

N	Two-Loop Run Time (sec)	FFT Run Time(sec)
50	0.0173	0.0013
100	0.0524	0.0033
1000	4.585	0.0002

As we can see in the table above, while the two-loop DFT program exponentially increases in computational time as the size of the analyzed signal increases, while the computational time of the FFT method remains at a very short period of time. The two-loop method is clearly expensive, as it entails a computational order of N^2.

b) See function DFT_B(x). This function works by only iterating through each value of k once. For each value of k, $X[k]$ is computed by the taking the inner product of the signal and the vector $W_N = e^{-j2\pi kn/N}$. This reduces the complexity from the two-loop program because it only requires one iteration through values of order N and it executes an inner product instead of a multiplications and a sum.

In the table below I have reported the run time for my one-loop function.

N	One-Loop Run Time (sec)
50	0.0055
100	0.0169
1000	1.424

As we can see, the one-loop program performs significantly better than the two-loop program, since the number of operations is reduced from N$\hat{2}$ to N.

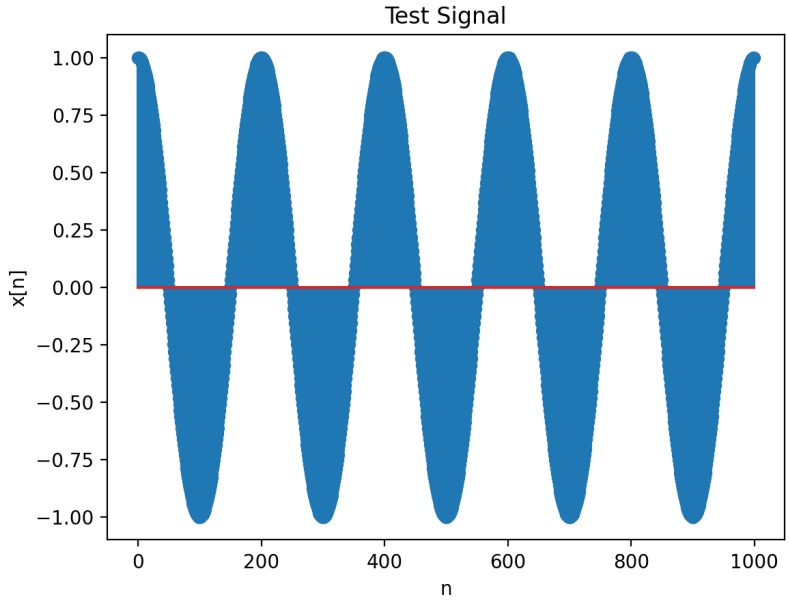
c) See function DFT_C(x). This function works by executing a matrix multiplication between a predefined matrix containing all the values of $W_N = e^{-j2\pi kn/N}$, where the columns correspond to n and the rows correspond to k, and the column vector associated with the input signal. This should improve the computational efficiency, since we are not iterating through each value from 0 to N, and we are computing the DFT

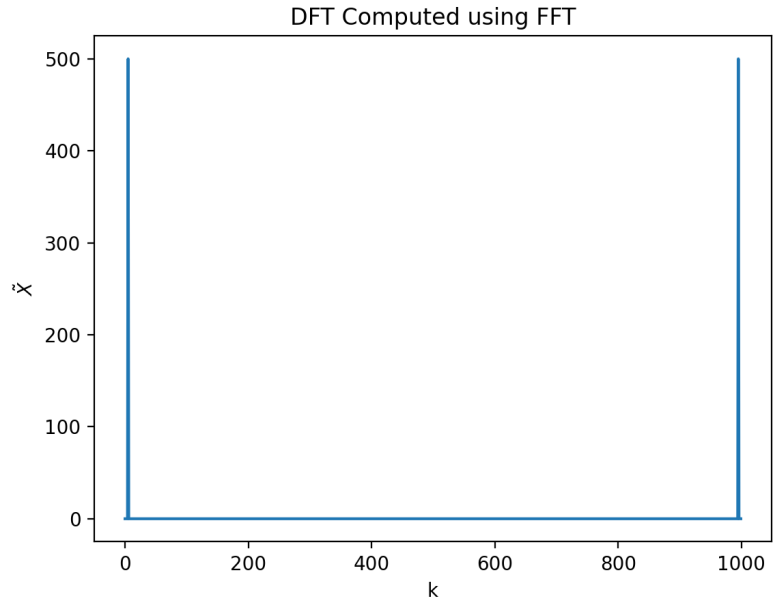
coefficients in one fell swoop via matrix multiplication. The table below reports the run time results.

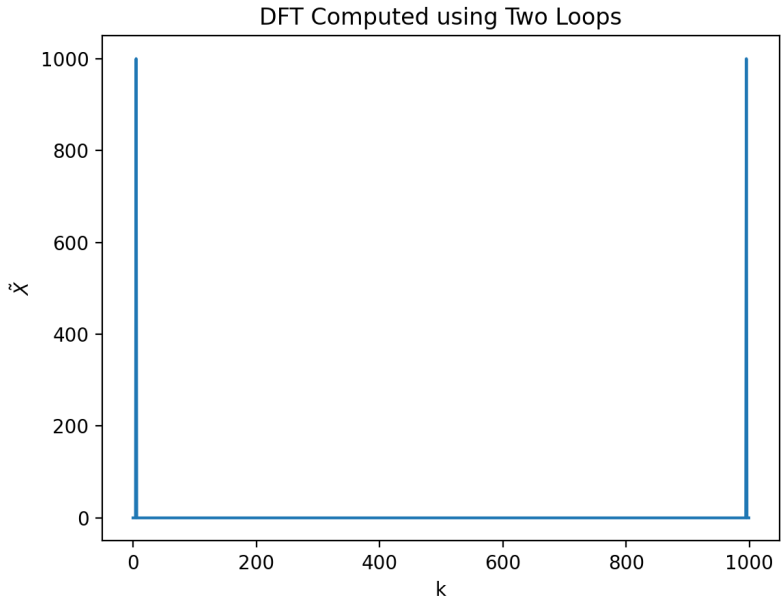
N	No-Loop Run Time (sec)
50	0.0007
100	0.0014
1000	0.0014

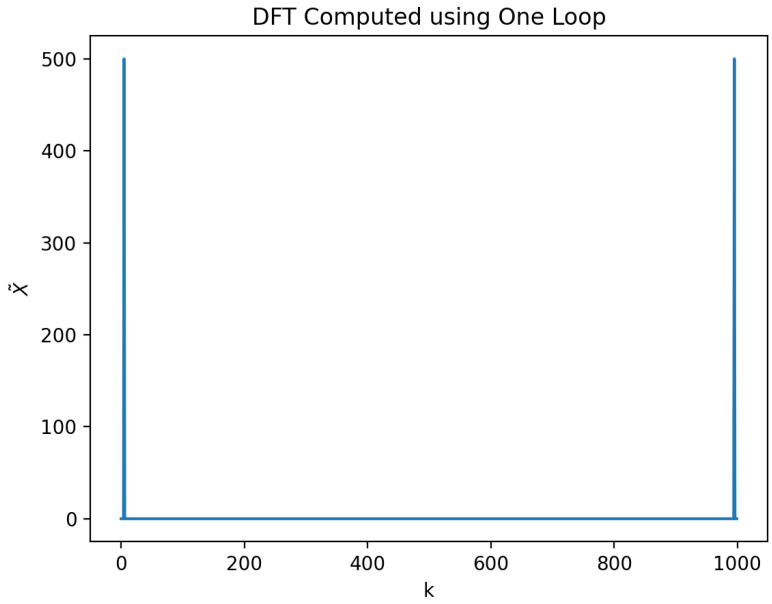
As we can see, for the signal lengths considered, the run time of the matrix method remains relatively low for the aforementioned reasons.

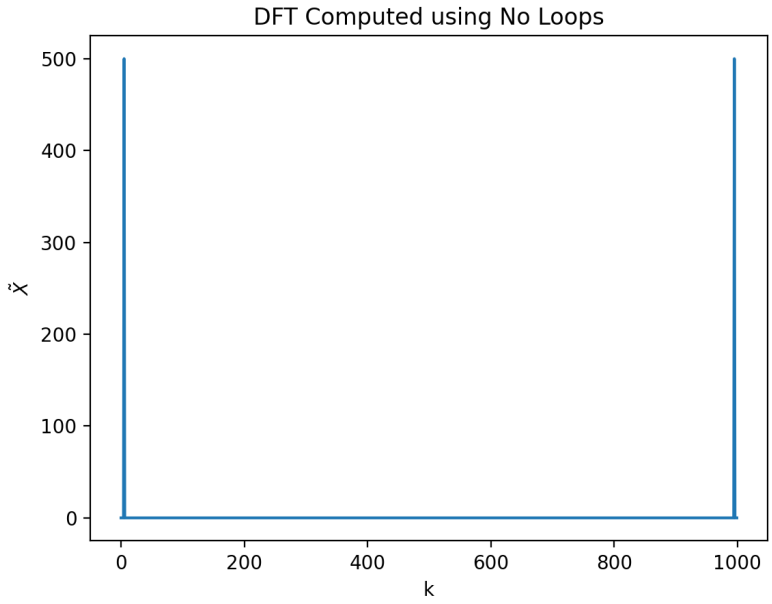
d) From the results tables, we can see that the FFT method performs the fastest, followed by the matrix method, followed by the one-loop method, followed by the two-loop method. The FFT method has a computational complexity of $N \log N$, so as N grows, the apparent difference in run time between the FFT and our own defined functions becomes more apparent. As a sanity check, I plotted the DFT of a test input signal for each of the functions considered, which are shown in the figures below.




DFT Computed using FFT
500
400
300
200
100
0
$\check{x}$
0
200
400
600
800
1000
k


DFT Computed using Two Loops
1000
800
600
400
200
0
$\check{x}$
0
200
400
600
800
1000
k





This is the expected result, since each method computes the same DFT coefficients according to the same formula. The only difference is the computational implementation, which is reflected in the reported run times.

Noah Schwab
ESE 531

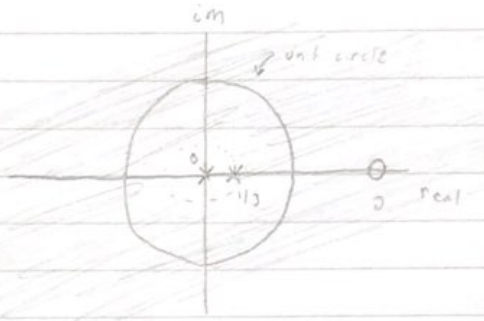
04/05/2022

HW 7: GLP Systems, Windowing, and DFT

5.39

$$H(z) = \frac{z-2}{z(z-1/3)}$$

a) To help understand the system, we can draw its pole-zero plot. We have a zero at $z=2$, and two poles at $z=0$, $z=1/3$ (and a zero at $z=\infty$).



Since $H(z)$ is stable, its ROC must include the unit circle. Therefore, the ROC is $\boxed{|z| > 1/3}$

Since the ROC extends outward to ∞, the system is causal.

b) First, consider a min-phase, all-pass decomposition of the form

$$H(z) = H_{min}(z)\, H_{ap}(z)\, z^{-1}$$

$H_{ap}(z)\, z^{-1}$ is a generalized-linear phase system since $|H_{ap}(z)| = 1$, a real-valued function of ω

$$H_{min}(z)\, H_{ap}(z) = \frac{z-2}{z-\frac{1}{3}} = \frac{1-2z^{-1}}{1-\frac{1}{3}z^{-1}}$$

$$\boxed{H_{ap}(z) = \frac{z^{-1}-\frac{1}{2}}{1-\frac{1}{3}z^{-1}}}$$

$$H_{min}(z) = \frac{z-2}{z-\frac{1}{3}} \quad \frac{1-\frac{1}{3}z^{-1}}{z^{-1}-\frac{1}{3}} = -2\,\frac{\left(1-\frac{1}{3}z^{-1}\right)}{1-\frac{1}{3}z^{-1}}$$

$$H_{lin}(z) = H_{ap}(z)\,z^{-1} = \frac{z^{-1}\left(z^{-1}-\frac{1}{3}\right)}{1-\frac{1}{3}z^{-1}}$$

7.15

$0.95 < |H(e^{j\omega})| < 1.05, \quad 0 \le |\omega| \le 0.25\pi$

$-0.1 < |H(e^{j\omega})| < 0.1, \quad 0.35\pi \le |\omega| \le \pi$

$\omega_c = 0.3\pi$

Windows from Section 7.5.1

- Rectangular
- Bartlett
- Hann
- Hamming
- Blackman

Based on the specs above, we require a max passband error of 0.05 and max stopband error of 0.1. Converting to dB:

$\delta_p = 0.05 = -26$ dB

$\delta_s = 0.1 = -20$ dB

Therefore, our filter requires a peak approximation error $\delta < -26$ dB. So we can immediately eliminate Rectangular and Bartlett.

Now we can compute $M+1$, the length of the filter, based on the width of the main lobe, which is approximately equal to the transition width.

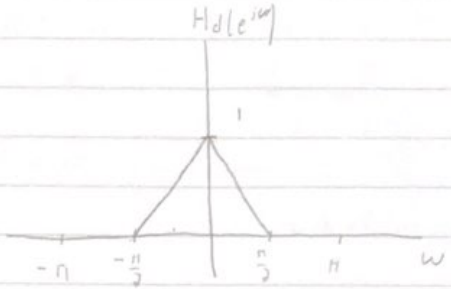
Hann: $0.1\pi = \frac{8\pi}{M} \Rightarrow M = 80$, Length = 81

Hamming: $0.1\pi = \frac{8\pi}{M} \Rightarrow M = 80$, Length = 81

Blackman: $0.1\pi = \frac{12\pi}{M} \Rightarrow M = 120$, Length = 121

7.45

$$\varepsilon_d^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |A(e^{j\omega}) - H_d(e^{j\omega})|^2 \, d\omega$$



$$H_d(e^{j\omega}) = \begin{cases} 0, & |\omega| > \frac{\pi}{2} \\ \frac{2}{\pi}\omega + 1, & -\frac{\pi}{2} < \omega < 0 \\ 1 - \frac{2}{\pi}\omega, & 0 < \omega < \frac{\pi}{2} \end{cases}$$

$H(e^{j\omega}) = A(e^{j\omega})\, e^{-j\omega M/2}$, M is an even integer

a) $h_d[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(e^{j\omega})\, e^{j\omega n}\, d\omega$

$= \frac{1}{2\pi}\int_{-\pi/2}^{0}\left(\frac{2}{\pi}\omega + 1\right) e^{j\omega n}\, d\omega + \frac{1}{2\pi}\int_{0}^{\pi/2}\left(1 - \frac{2}{\pi}\omega\right) e^{j\omega n}\, d\omega$

$= \frac{2e^{-j\frac{\pi}{2}n} + j\pi n - 2}{2\pi^2 j^2 n^2} + \frac{2e^{j\frac{\pi}{2}n} - j\pi n - 2}{2\pi^2 j^2 n^2}$

$= \frac{2e^{-j\frac{\pi}{2}n} + 2e^{j\frac{\pi}{2}n} - 4}{2\pi^2 j^2 n^2}$

$= \frac{4 - 2\left(e^{-j\frac{\pi}{2}n} + e^{j\frac{\pi}{2}n}\right)}{2\pi^2 n^2} = \frac{4 - 4\cos\left(\frac{\pi}{2}n\right)}{2\pi^2 n^2}$

$$\boxed{h_d[n] = \frac{2\left(1 - \cos\left(\frac{\pi}{2}n\right)\right)}{(\pi n)^2} = \frac{\sin^2\left(\frac{\pi}{4}n\right)}{\pi^2 n^2}}$$

b) $H(e^{jw})$ has the form $A(e^{jw})e^{-jwM/2}$ and minimizes the integral squared difference between $A(e^{jw})$ and $H_d(e^{jw})$

Therefore, $A(e^{jw})$ approximates $H_d(e^{jw})$, and we know $H_d(e^{jw})$ is an even-function. Therefore, $h[n]$ must be even symmetric about $M/2$ in the range $0 \le n \le M$

c) Since $h_d[n]$ is symmetric, we can use a symmetric window to design $h[n]$

$$h[n] = h_d[n]\,w[n]$$

$$w[n] = \begin{cases} 1, & 0 \le n \le M \\ 0, & \text{otherwise} \end{cases}$$

$$h[n] = \begin{cases} h_d[n], & 0 \le n \le M \\ 0, & \text{otherwise} \end{cases}$$

In the range $0 \le n \le M$, $h[n] = h_d[n]$, after shifting

$$\boxed{h[n] = \begin{cases} \sin^2\left(\frac{\pi}{4}(n-M/2)\right) \Big/ \pi^2(n-M/2)^2, & 0 \le n \le M \\ 0, & \text{otherwise} \end{cases}}$$

d) Citing Parseval's Theorem:

$$\varepsilon^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |A(e^{jw}) - H_d(e^{jw})|^2\,dw = \sum_{-\infty}^{\infty} |a[n] - h_d[n]|^2$$

Given $H(e^{jw}) = A(e^{jw})e^{-jwM/2}$, $a[n] = h_d[n]$ in the range $-M/2 \le n \le M/2$

$$\varepsilon^2 = \sum_{-\infty}^{-M/2-1} |a[n] - h_d[n]|^2 + \sum_{-M/2}^{M/2} |a[n] - h_d[n]|^2 + \sum_{M/2+1}^{\infty} |a[n] - h_d[n]|^2$$

Since $a[n] = h_d[n]$ in the range $-M/2 \le n \le M/2$,
and $h_d[n]$ is even symmetric, and $a[n] = 0$ outside,

$$\varepsilon^2 = 2 \sum_{M/2+1}^{\infty} |h[n]|^2$$

8.14

$$x_3[n] = x_1[n] \circledast x_2[n]$$

$$x_3[2] = \sum_{m=0}^{7} x_1[m]\, x_2[2-m]$$

$$= x_1[0]x_2[2] + x_1[1]x_2[1] + x_1[2]x_2[0] + x_1[3]x_2[-1] + x_1[4]x_2[-2] + x_1[5]x_2[-3] + x_1[6]x_2[-4] + x_1[7]x_2[-5]$$

Negative indices wrap around for circular convolution

$$x_3[2] = x_1[0]x_2[2] + x_1[1]x_2[1] + x_1[2]x_2[0] + x_1[3]x_2[7] + x_1[4]x_2[6] + x_1[5]x_2[5] + x_1[6]x_2[4] + x_1[7]x_2[3]$$

$$= (1)(3) + (2)(1) + (1)(0) + (1)(0) + (2)(0) + (1)(0) + (0)(0) + (2)(2)$$

$$= 3 + 2 + 4 = 9$$

$$\boxed{x_3[2] = 9}$$

```python
# ESE 531: HW6 Problem 2
# Author: Noah Schwab

# import libraries
import numpy as np
import time
import matplotlib.pyplot as plt

'''
DFT_A(x)

arguments:

    x: signal to be transformed, assumed to be one period

return

    X: DFT of input signal x, this function executes a nested for loop
'''
def DFT_A(x):

    # N is length of one period of x
    N = len(x)

    # instantiate empty array X to compute values
    X = np.empty(N, dtype=complex)

    # iterate through each value of k in range N
    t0 = time.time()
    for k in range(N):

        # nested loop to interate through each value of n in range N
        for n in range(N):

            # populate X with DFT value
            X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
    t1 = time.time()
    print(f'Two-Loop Run time: {t1-t0}')

    # return DFT of x, X
    return X

'''
DFT_B(x):

arguments:

    x: signal to be transformed, assumed to be one period

return

    X: DFT of input signal x, this function executes one for loop and an inner product
'''
def DFT_B(x):

    # N is length of one period of x
    N = len(x)

    # instantiate empty array X to compute values
    X = np.empty(N, dtype=complex)

    # iterate through each value of k in range N and compute X[k] via an inner product
    t0 = time.time()
    for k in range(N):

        # predefine vector W for inner product
```

```
        W = np.array([np.exp(-2j * np.pi * k * n / N) for n in range(N)])

        # comptue inner product
        X[k] = np.dot(x, W)
    t1 = time.time()
    print(f'One-Loop Run time: {t1-t0}')

    return X

'''
DFT_C(x):

arguments:

    x: signal to be transformed, assumed to be one period, must be a column vector

return:

    X: DFT of input signal, this function executes a matrix multiplication
'''
def DFT_C(x):

    # N is length of one period of x
    N = len(x)

    # predefine matrix W
    W = np.empty((N, N), complex)

    for k in range(N):
        W[k] = [np.exp(-2j * np.pi * k * n / N) for n in range(N)]

    # execute matrix multiplication
    t0 = time.time()
    X = W @ x
    t1 = time.time()
    print(f'No-Loop Run time: {t1-t0}')

    # return transpose to convert X to row vector
    return X.T

def main():


    # test signal
    N = 1000
    x = np.cos([np.pi * n / 100 for n in range(N)])
    plt.title('Test Signal')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.stem(x)
    plt.show()

    # compute DFT using built-in FFT function
    t0 = time.time()
    X0 = np.fft.fft(x)
    t1 = time.time()
    print(f'FFT Run time: {t1-t0}')


    # compute DFT using my functions
    X1 = DFT_A(x)
    X2 = DFT_B(x)
    X3 = DFT_C(x)

    # plot each DFT to show they are equivalent
    plt.plot(X0)
```

```
    plt.title('DFT Computed using FFT')
    plt.xlabel('k')
    plt.ylabel(r'$\tilde{X}$')
    plt.show()

    plt.plot(X1)
    plt.title('DFT Computed using Two Loops')
    plt.xlabel('k')
    plt.ylabel(r'$\tilde{X}$')
    plt.show()

    plt.plot(X2)
    plt.title('DFT Computed using One Loop')
    plt.xlabel('k')
    plt.ylabel(r'$\tilde{X}$')
    plt.show()

    plt.plot(X3)
    plt.title('DFT Computed using No Loops')
    plt.xlabel('k')
    plt.ylabel(r'$\tilde{X}$')
    plt.show()


if __name__ == "__main__":
    main()
```