DIGITAL SIGNAL PROCESSING

SIMULATION ASSIGNMENT

FREQUENCY ANALYSIS OF SIGNALS

Author
Daniel V Mathew

Roll No: 28
Semester: 5
Department of Electronics and Communication
Rajiv Gandhi Institute of Technology

Supervisor
Prof. Premson Y

Assistant Professor
Department of Electronics and Communication
Rajiv Gandhi Institute of Technology

Abstract

This paper analyse various *parameters* that affect the *generation* of DTFTs and DFTs and tries to find an *optimal* mix of all of these *parameters* in giving a good *frequency spectrum* that *carries* the *sufficient spectral information* to *reconstruct* or *interpolate back* the *original input signal*.

It is found that the *zero padding* and taking a *higher point* TRANSFORM does *enhance* the DFTs but *not so much* in the case of DTFTs. In both the TRANSFORMs, *increasing* the *sample count* and taking a *higher point* TRANSFORM seems to *increase* the *frequency resolution*.

It is also found that these TRANSFORMs only *carry spectral information* upto *half* the *sampling frequency*. The other half *mirrors* the first half. *Increasing* just the *sampling frequency* can also *negatively* affect the TRANSFORM if *sample count* is not *sufficient*.

The *real* and the *imaginary* parts of the TRANSFORMs are found to be crucial in *reconstructing* the *input signal*, as together they carry the *amplitude* and *phase* information.

September 18, 2024

1 Introduction

This analysis aims to analyse bunch of *complex signals*[1], using DTFT and DFT in order to find their *frequency components*. Thereby *investigating* different *parameters* that affect the *computation* of DTFTs and DFTs. And to see how we can *adjust* these *parameters* to yield a usable *frequency spectrum*. We will also *investigate* how *zero padding* affects the generated DTFTs and DFTs. We will also see the *information content* of the *real part* and the *imaginary part* of the generated DTFTs and DFTs, and how they will help us to *perfectly reconstruct*[2] the *input signal* by giving us the *sufficient spectral information* about the *input signal*.

Tools used: For the *computation* of the sequences and DTFTs and DFTs we will use PYTHON and we will save the these sequences as *csv* files[3]. Then they can be *plotted* using LaTeX's PGFPLOTS package. And all the *figures* in this report are drawn using LaTeX's Ti*k*Z package, as this entire report is written using LaTeX[4].

An overview: First we will *generate* some *real* sequences. Then combine them into bunch of *complex* sequences. Then we will take DTFTs and DFTs. Then we will *zero pad* them, then again we will find the corresponding DTFTs and DFTs. And we will analyse them in their respective sections. And in the end we will conclude all of our *interesting* observations.

[1] with *real sine signals* with different *frequencies* as their *real* and *imaginary* parts.
[2] we won't be *reconstructing* them in this analysis though.
[3] they are not actually *comma separated*, but are *space separated*, they just have the *.csv* extension.
[4] for all *source files*, see **https://github.com/thepenguinn/notes/tree/master/s5/dsp/work_01**.

2 Generating Real Sequences

For our *analysis* we need generate some *complex sequences*, but before we generate these *complex sequences* we need to decide the *frequency components* that goes into the *real* and *imaginary* parts. So select some *frequencies* for that. Let the following be those *frequencies*.

$$f_1 = 28\text{Hz} \tag{1}$$
$$f_2 = 2 \times 28 = 56\text{Hz} \tag{2}$$
$$f_3 = (2 \times 28) + 0.1 = 56.1\text{Hz} \tag{3}$$

Now we need to decide whether we need to generate a *sine* signal or one with 90° phase shift, ie, the *cosine* signal. If we mix[5] *sine* and *cosine* we may get a different kind of *complex* sequence than if we mix *sine* to *sine* or *cosine* to *cosine*. For our analysis, let's stick with having zero phase shift between the signals that we are mixing. So let's just mix *sine* signals with the above frequencies.

To get the continuous feel of these sequences let's generate them with 2500 samples. And these signals will range from 0 to $\frac{\pi}{4}$.

Therefore these signals will be:

$$x_1(t) = \left\{ \sin(2\pi 28t) \quad ; \text{for } 0 \leq t \leq \frac{\pi}{4} \right. \tag{4}$$

$$x_2(t) = \left\{ \sin(2\pi 56t) \quad ; \text{for } 0 \leq t \leq \frac{\pi}{4} \right. \tag{5}$$

$$x_3(t) = \left\{ \sin(2\pi 56.1t) \quad ; \text{for } 0 \leq t \leq \frac{\pi}{4} \right. \tag{6}$$

2.1 Generating Sequences using Python

```
import numpy as np
import pandas as pd

X = 28
t = np.linspace(0, np.pi / 4, 2500)

x1 = np.sin( 2 * np.pi * X * t)
x2 = np.sin( 2 * np.pi * (2 * X) * t)
x3 = np.sin( 2 * np.pi * ((2 * X) + 0.1) * t)

data = pd.DataFrame(
    {
        "x1": x1,
        "x2": x2,
        "x3": x3,
    }
)
```

[5] like, real part and imaginary part having a phase difference.

```
data.to_csv(
    "../data/three_sequences.csv",
    sep = " ", index_label = "time"
)
print(data)
```

Output

```
             x1        x2        x3
0      0.000000  0.000000  0.000000
1      0.055264  0.110359  0.110555
2      0.110359  0.219369  0.219754
3      0.165116  0.325699  0.326259
4      0.219369  0.428051  0.428765
...         ...       ...       ...
2495 -0.273262 -0.525724 -0.060838
2496 -0.219684 -0.428635  0.049885
2497 -0.165435 -0.326311  0.159997
2498 -0.110680 -0.220000  0.268147
2499 -0.055586 -0.111001  0.373009

[2500 rows x 3 columns]
```

2.2 Real Sequences

Figures from 1 to 3 plots above generated *real valued sequences*. Among them 2 and 3 almost seems exactly the same. But if we look closer at the $\frac{\pi}{4}$ mark, we can see the slight difference of 0.1 in frequency causing them to deviate.

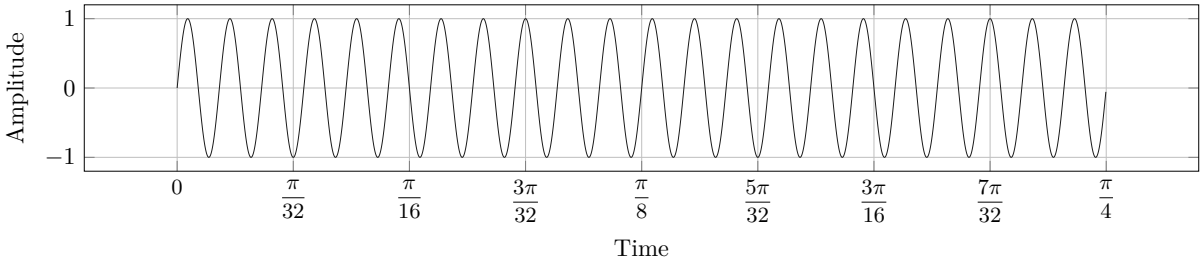

Figure 1: Sine signal x_1 with frequency of 28Hz

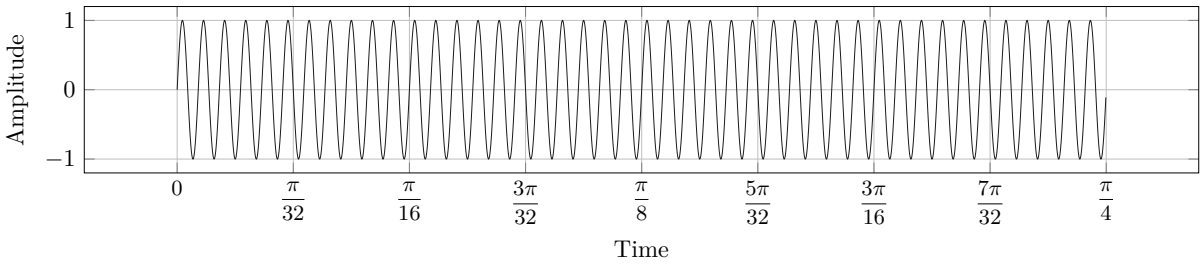

Figure 2: Sine signal x_2 with frequency of 56Hz

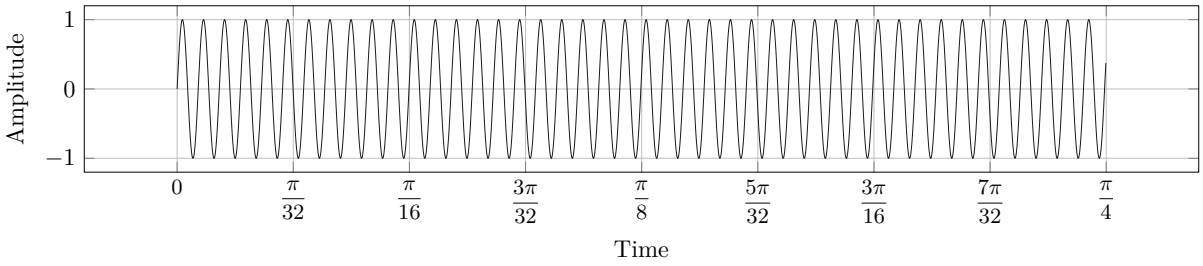

Figure 3: Sine signal x_3 with frequency of 56.1Hz

3 Generating Complex Sequences

In section 2.1 we generated three signals with three different frequencies. The important thing to notice is that, they all are real valued sequences. In this section, we will see how we can combine[6] these signals into *complex* sequences.

3.1 Mixing of Signals

We have three different distinct sequences, so how we can mix these signals into more diverse *complex* sequences? This can be simply done by taking one of our *real valued* sequence as the *real part* of the *complex* sequence and another *real valued* sequence as the *imaginary part* of the *complex* sequence. By doing so, one may encounter that there are many ways to arrange these sequences. Or so to say, there are many different combinations[7], thus generating there many distinct *complex* sequences.

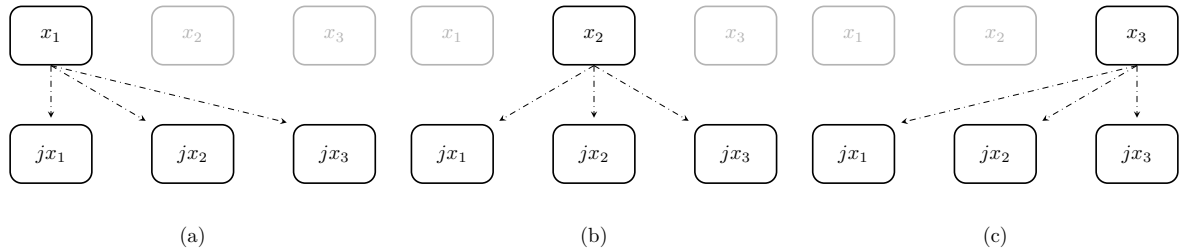

Figure 4: Different combinations of *complex sequences*. (a): Taking x_1 as real part and x_1, x_2, and x_3 as imaginary parts. (b): Taking x_2 as real part and x_1, x_2, and x_3 as imaginary parts. (c): Taking x_3 as real part and x_1, x_2, and x_3 as imaginary parts.

As we can see from Figure 4 there is a total of $3 \times 3 = 9$ different ways to combine these three signals. We will pick some of them depending on how *distinct* they look and we will see each of those selected ones in detail in the following sections before further analysis.

But before that, let's use python to combine our *real* sequences into *complex* sequences.

3.2 Combining Sequences using Python

```
import numpy as np
import pandas as pd

X = 28
```

[6] as real and imaginary part of a complex sequence.
[7] of different real and imaginary sequences.

```
t = np.linspace(0, np.pi / 4, 2500)

x1 = np.sin( 2 * np.pi * X * t)
x2 = np.sin( 2 * np.pi * (2 * X) * t)
x3 = np.sin( 2 * np.pi * ((2 * X) + 0.1) * t)

real = [x1, x2, x3]
imag = [x1, x2, x3]

pd.DataFrame( {
    "x1": x1[0:625], "x2": x2[0:625], "x3": x3[0:625],
}).to_csv(
    "../data/short_sequences.csv",
    sep = " ", index_label = "time"
)

cplx_labels = [
    "complex_a", "complex_b", "complex_c",
    "complex_d", "complex_e", "complex_f",
    "complex_g", "complex_h", "complex_i",
]

cplx = []

for r in real:
    for i in imag:
        cplx.append(pd.DataFrame({
            "real": r,
            "imag": i,
        }))

for i in range(len(cplx)):
    cplx[i].to_csv(
        "../data/" + cplx_labels[i] + ".csv",
        sep = " ", index_label = "time"
    )
```

Combination Name	Complex Sequence	Combination Name	Complex Sequence	Combination Name	Complex Sequence
Complex A	$x_1 + jx_1$	Complex E	$x_2 + jx_1$	Complex G	$x_3 + jx_1$
Complex B	$x_1 + jx_2$	Complex D	$x_2 + jx_2$	Complex H	$x_3 + jx_2$
Complex C	$x_1 + jx_3$	Complex F	$x_2 + jx_3$	Complex I	$x_3 + jx_3$

Table 1: Different combinations of *mixed complex* sequences. The selected sequences for further analysis can also be seen as *highlighted*.

3.3 Complex A

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 28t)$

Inference: Here, both the *real part* and the *imaginary part* are the same, hence from front view they seem to oscillate back and forth between $(-1-1j)$ and $(1+1j)$ through a straight line passing through the origin.

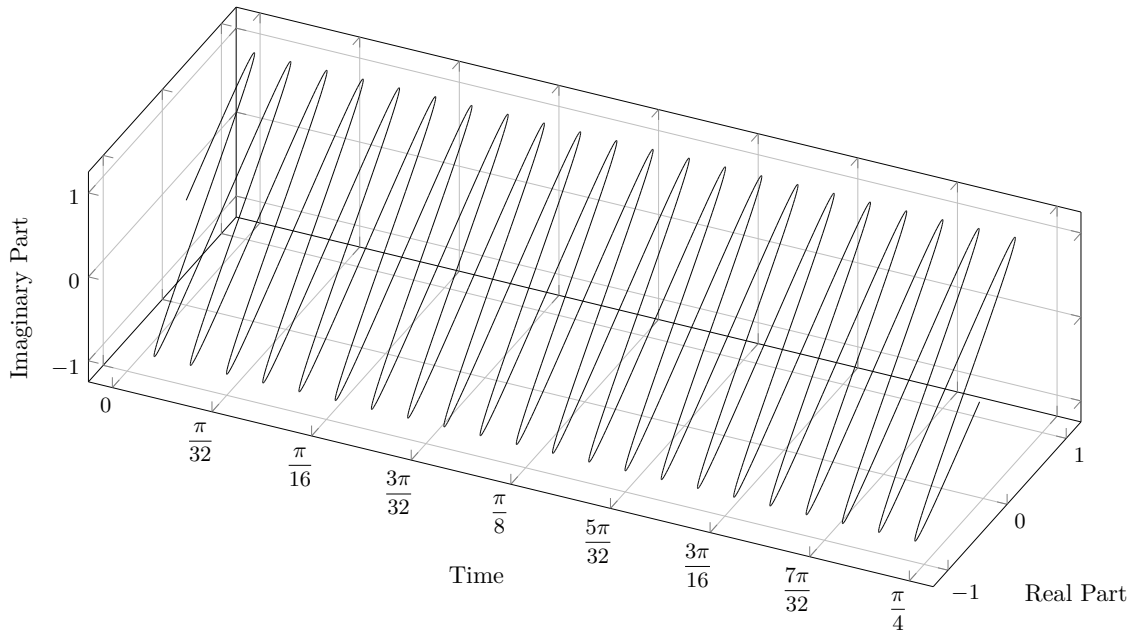

Figure 5: 3D view of *complex sequence* A.

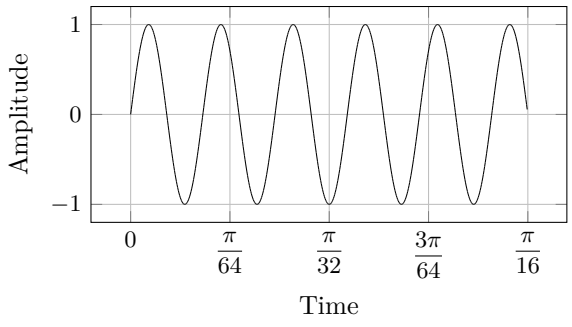

Figure 6: *Real part* $\sin(2\pi 28t)$.

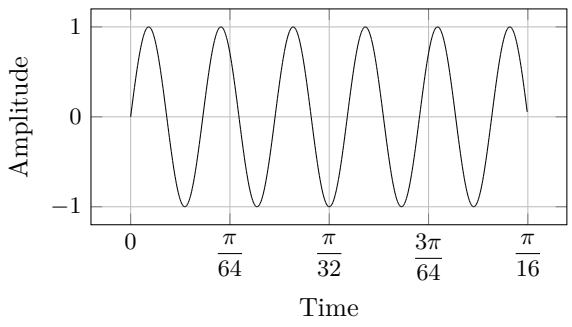

Figure 7: *Imaginary part* $\sin(2\pi 28t)$.

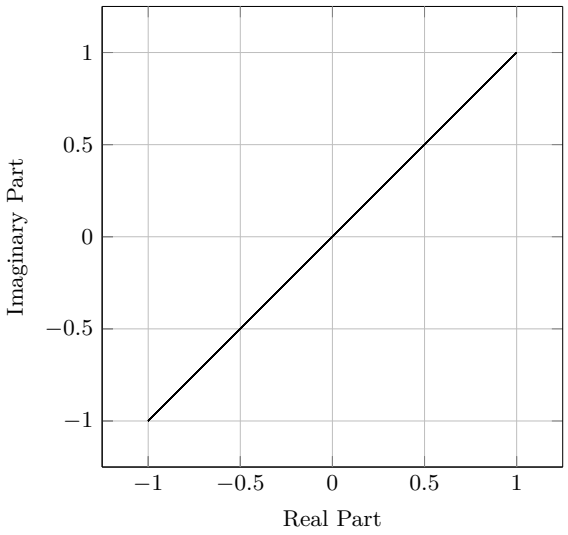

Figure 8: View with *x axis* perpendicular.

3.4 Complex B

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

Inference: Here the frequency of the *imaginary part* is exactly the double that of the *real part*. Hence forming an infinity[8] shape from the front view.

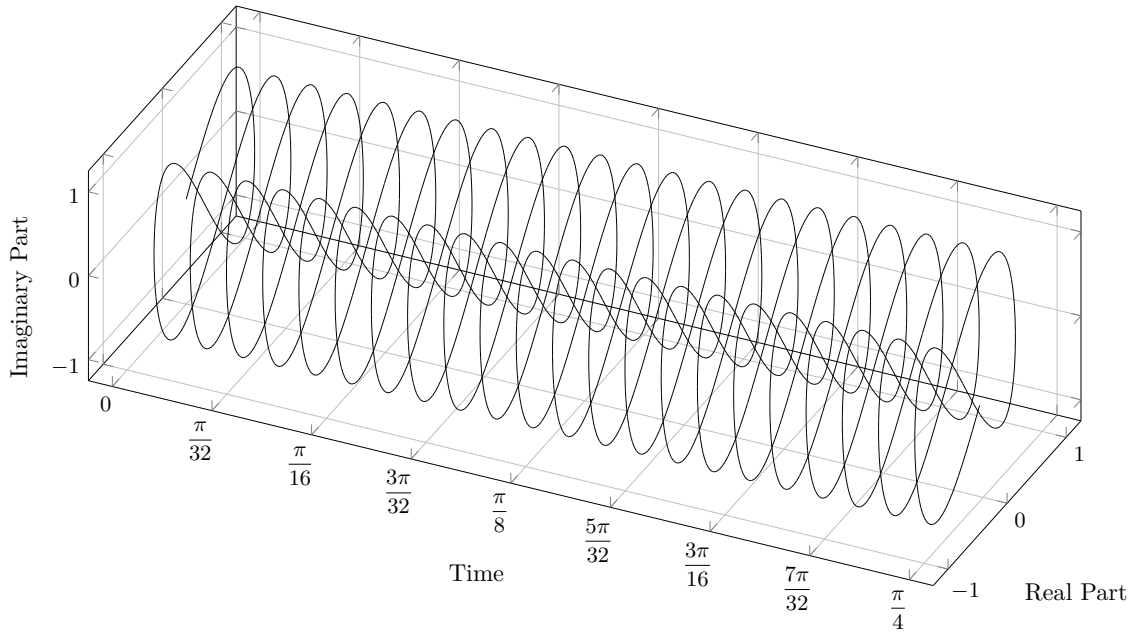

Figure 9: 3D view of *complex sequence* B.

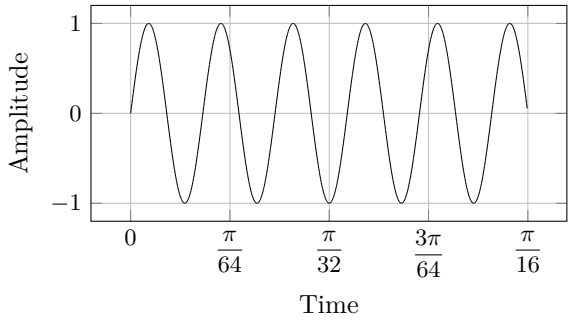

Figure 10: *Real part* $\sin(2\pi 28t)$.

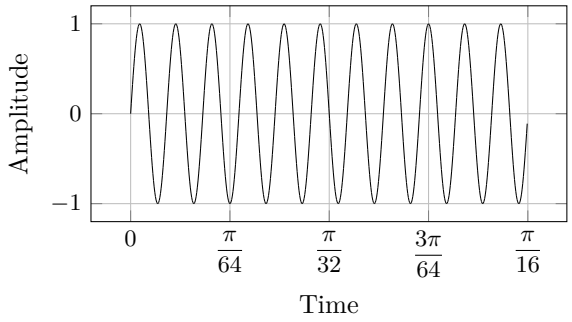

Figure 11: *Imaginary part* $\sin(2\pi 56t)$.

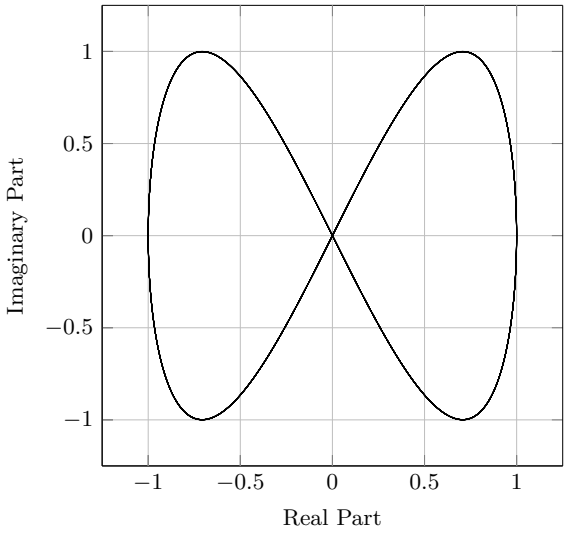

Figure 12: View with *x axis* perpendicular.

[8] in mathematics, it's called a *lemnicate* (see the **Wikipedia article**).

3.5 Complex C

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56.1t)$

Inference: Here the *real* and *imaginary* parts are similar to that of Complex B from Section 3.4. The only difference is that the *frequency* of the *imaginary part* is not exactly the double that of the *real part*. This causes the signal to deviate from that infinity shape. This becomes evident as we observe the front view from the Figure 16.

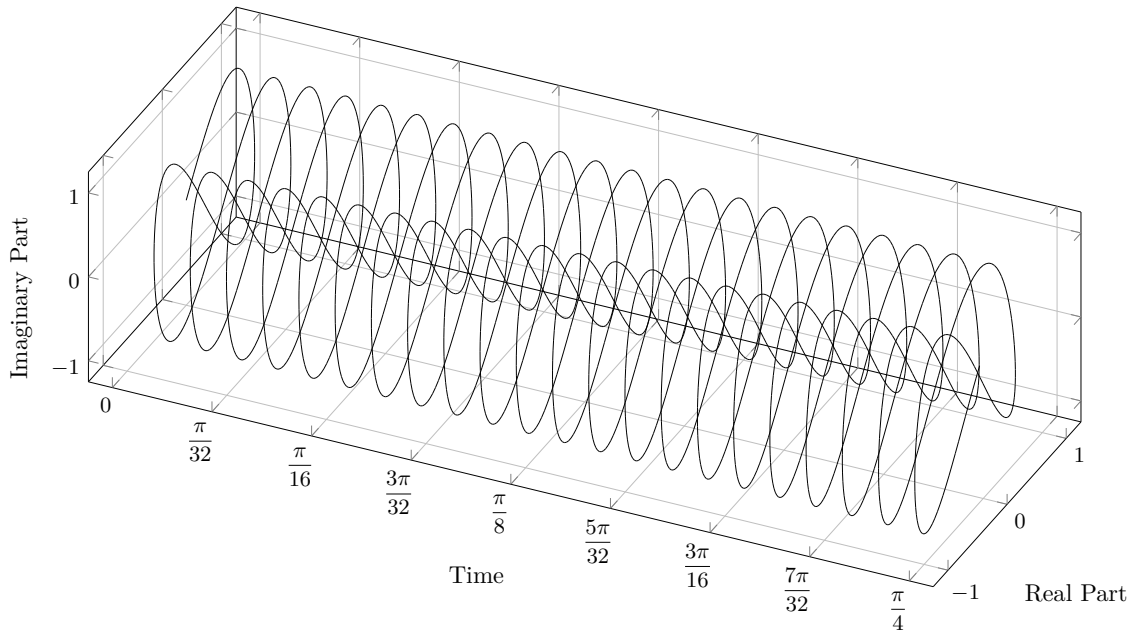

Figure 13: 3D view of *complex sequence* C.

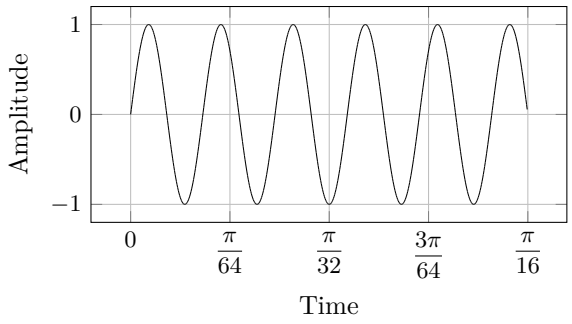

Figure 14: *Real part* $\sin(2\pi 28t)$.

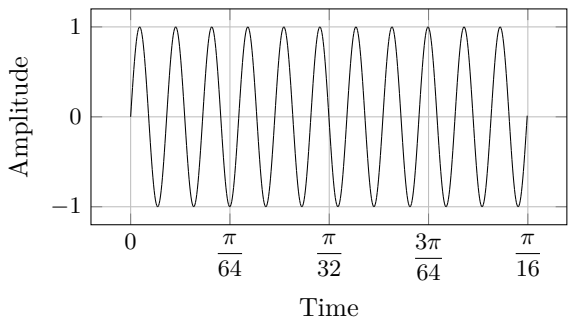

Figure 15: *Imaginary part* $\sin(2\pi 56.1t)$.

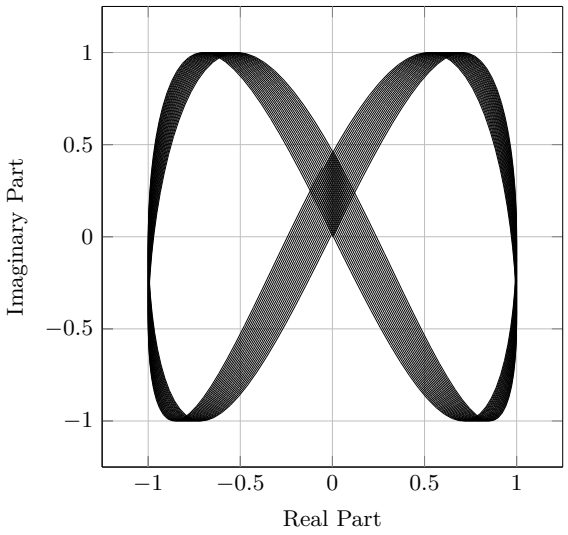

Figure 16: View with *x axis* perpendicular.

3.6 Complex F

Sequence: $\sin(2\pi 56t) + j\sin(2\pi 56.1t)$

Inference: Here the *imaginary part* is higher in frequency than that of the *real part*. This is similar to that of Complex A from Section 3.3, but this slight frequency difference will deviate the signal ever so slightly from oscillating through the straight line as viewed from the front view, and making the front view as shown in Figure 20.

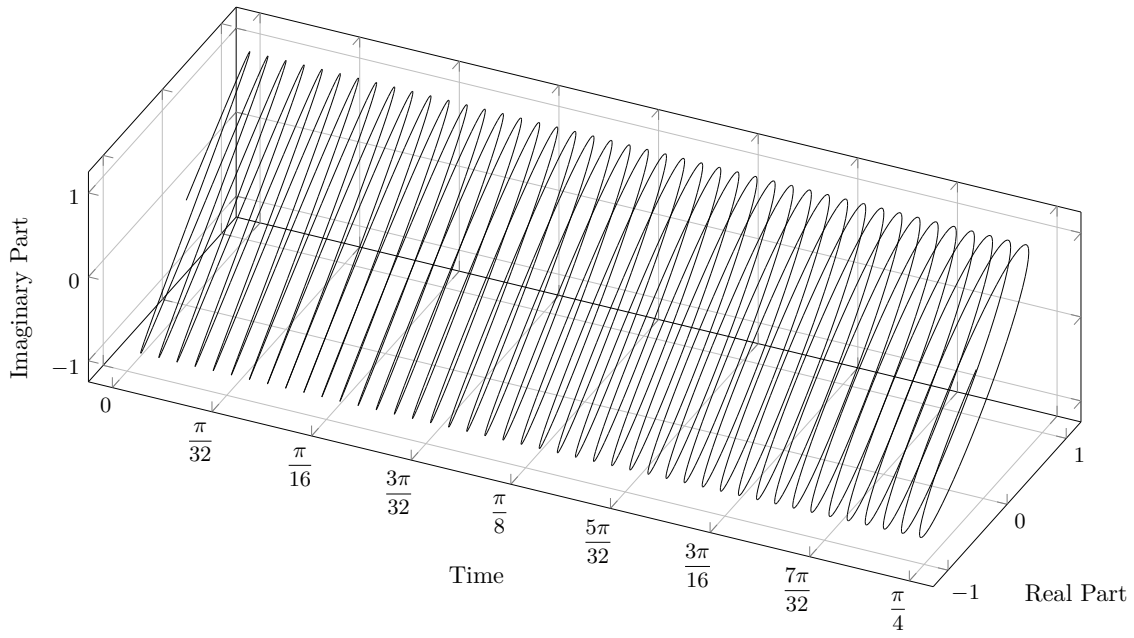

Figure 17: 3D view of *complex sequence* F.

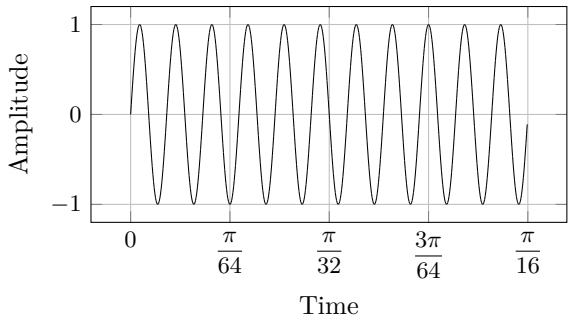

Figure 18: *Real part* $\sin(2\pi 56t)$.

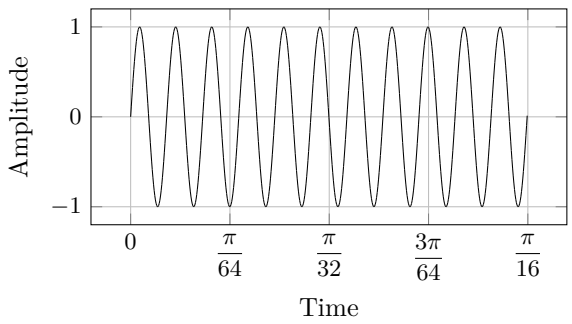

Figure 19: *Imaginary part* $\sin(2\pi 56.1t)$.

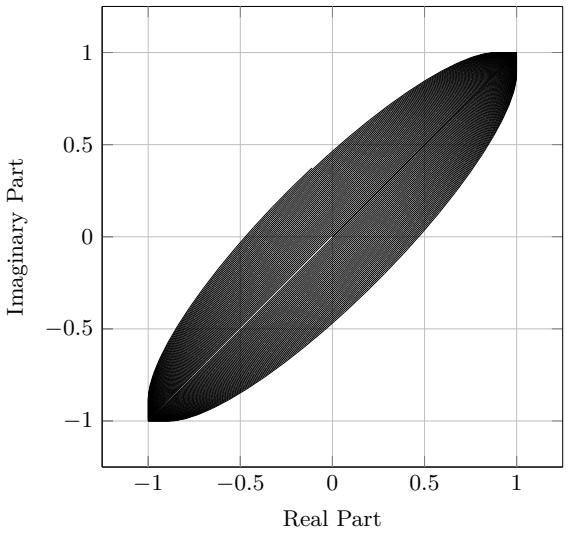

Figure 20: View with *x axis* perpendicular.

4 Taking DTFT of the Complex Sequences

In the previous section, we generated a bunch of *complex sequences*. And picked 4 sequences for further analysis. In this section we will be taking DTFT of those sequences. But before that, we need to take a look at what a DTFT is and how to compute them using PYTHON.

4.1 Discrete Time Fourier Transform

Discrete Time Fourier Transform, aka, DTFT transforms *discrete time* samples into a *continuous* signal in the *frequency* domain. DTFT is basically a special case of another transform known as $\mathcal{Z}$-TRANSFORM. $\mathcal{Z}$-TRANSFORM can be mathematically described as,

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{7}$$

where,

$\mathcal{Z}\{x[n]\}$: is the $\mathcal{Z}$-TRANSFORM.

$x[n]$: is the input sequence, which was sampled with a particular *sampling frequency*, f_s.

z: is a *complex number*, in the form $Ae^{-j\omega}$.

where,

A: is the *real part* or the *amplitude*.

ω: is the *complex argument* in radians.

DTFT is a special case of $\mathcal{Z}$-TRANSFORM, where we take A as 1. Mathematically, we can describe DTFT as,

$$X(e^{j\omega}) = X(z)|_{z=e^{j\omega}} \tag{8}$$

$$= \sum_{n=-\infty}^{\infty} x[n]z^{-n}\Big|_{z=e^{jw}} \tag{9}$$

$$= \sum_{n=-\infty}^{\infty} x[n]e^{-jwn} \tag{10}$$

Where ω can be substituted with $2\pi f$, then the DTFT becomes,

$$X(e^{j2\pi f}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j2\pi fn} \tag{11}$$

Now eq. (10) becomes a function of *frequency* f instead of *angular frequency* ω. One thing to NOTE is that, *sweeping* f in eq. (11) from 0 to 1 will result in same output as *sweeping* ω in eq. (10) from 0 to 2π. Another interesting thing about this *transform* is that the $e^{-j2\pi fn}$ will *scales* and *wraps* the *frequencies* from 0 to ∞ around the UNIT CIRCLE in the $\mathcal{Z}$-PLANE. This will become important as we compute the DTFTs. The Figure 21 attempts to visualizes this interesting concept.

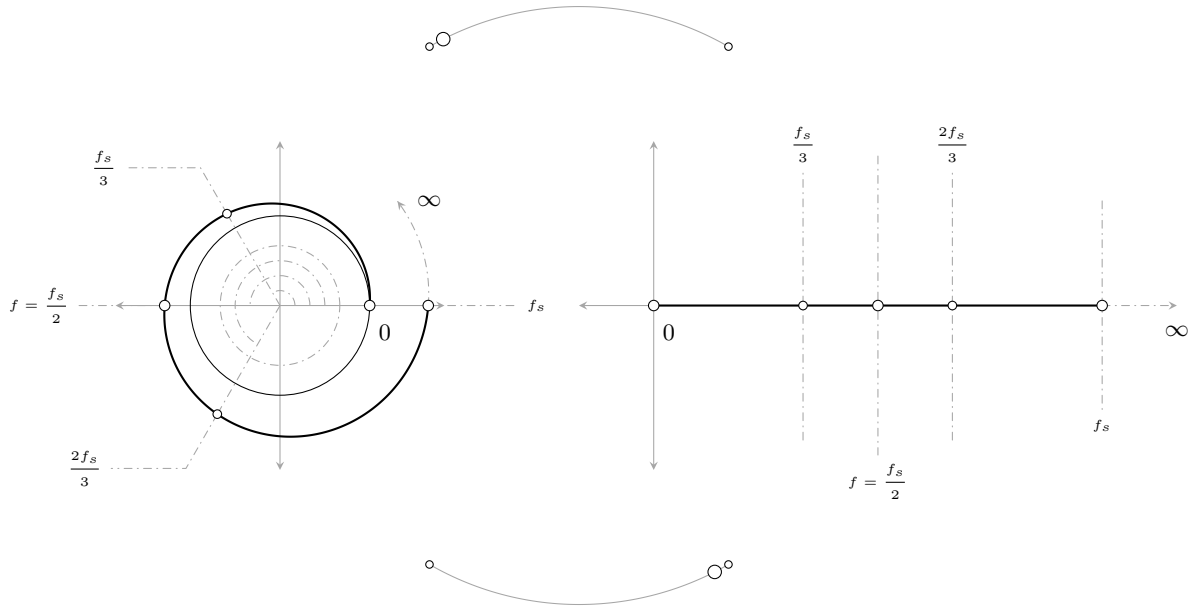

Figure 21: Wrapping frequencies around the UNIT CIRCLE in $\mathcal{Z}$-PLANE

4.2 Implementing DTFT using Python

We essentially need to take DTFT of several[9] sequences. So it will be handy to implement a *factory*, that will take a *sequence* and returns a *dtft function* that can be called with different *frequencies* and returns it's corresponding value. Let us call this *function* the `dtft_factory`.

```
import numpy as np

def dtft_factory(cplx_seq):

    def dtft(freq):
        sum = 0
        # see eq. (11)
        for n, cplx in enumerate(cplx_seq):
            sum = sum + cplx * np.exp(-1j * 2 * np.pi * freq * n)
        return sum

    return dtft
```

Now, let's implement another *function* that will returns another *function*[10] that can be called with $n = 0, 1, 2, ...$ to generate the *sequence* with specified *sampling frequency*, *real* and *imaginary* parts. Let's call this *function* `cplx_factory`

```
import numpy as np

def cplx_factory(samp_freq, real_freq, imag_freq):
    samp_period = 1 / samp_freq
    return lambda n: np.sin(
        2 * np.pi * real_freq * (n * samp_period)
    ) + 1j * np.sin(
```

[9] actually 4, still more than 1.
[10] yes, of course.

```
        2 * np.pi * imag_freq * (n * samp_period)
    )
```

Let's implement yet another *function* that will make our life easier by *mixing* and *generating* the DTFT for two given *frequencies*[11]. As depicted in Figure 21 we only need to evaluate the DTFT from 0 to 1 range[12] and need to *rescale* it back.

```
def gen_dtft_seq(
    samp_count, samp_freq, real_freq, imag_freq, dtft_samp_count
    ):

    cplx_fn = cplx_factory(
        samp_freq = samp_freq,
        real_freq = real_freq,
        imag_freq = imag_freq
    )

    cplx_seq = np.ndarray(samp_count, dtype = np.cdouble)

    for i in range(samp_count):
        cplx_seq[i] = cplx_fn(i)

    freq = np.linspace(0, 1, dtft_samp_count)

    dtft_fn = dtft_factory(cplx_seq = cplx_seq)
    dtft_seq = dtft_fn(freq)

    # scaling back the freq from (0 - 1) to (0 - f_s)
    return freq * samp_freq, dtft_seq, cplx_seq
```

Now we could just call `gen_dtft_seq` with appropriate *arguments* and it will return the `dtft_seq` and its corresponding *frequencies*. Let's take a bunch of DTFTs with different *sampling frequencies*.

```
import pandas as pd

X = 28

cplx_seqs = [
    {
        "name": "complex_a",
        "real_freq": X,
        "imag_freq": X,
        # we will fill this later, for taking 64 point DTFT
        "sequences": {},
    },
    {
        "name": "complex_b",
        "real_freq": X,
        "imag_freq": 2 * X,
```

[11] one as *real* and one as *imaginary*.
[12] in the case of eq. (11).

```
        "sequences": {},
    },
    {
        "name": "complex_c",
        "real_freq": X,
        "imag_freq": (2 * X) + 0.1,
        "sequences": {},
    },
    {
        "name": "complex_f",
        "real_freq": 2 * X,
        "imag_freq": (2 * X) + 0.1,
        "sequences": {},
    },
]

samp_freqs = {
    "normal":           int(4 * X),
    "slightly_greater": int(4 * X + 10),
    "much_greater":     int(4 * X * 6),
    "much_lesser":      int(4 * X / 2),
}

samp_count = 32

dtft_samp_count = 2000

for cplx in cplx_seqs:

    for samp_freq in samp_freqs:

        freq, dtft_seq, cplx_seq = gen_dtft_seq(
            samp_count = samp_count,
            samp_freq = samp_freqs[samp_freq],
            real_freq = cplx["real_freq"],
            imag_freq = cplx["imag_freq"],
            dtft_samp_count = dtft_samp_count
        )

        # keeping generated sequences for taking 64 point DTFT
        cplx["sequences"][samp_freq] = cplx_seq

        data = pd.DataFrame(
            data = {
                "real": dtft_seq.real,
                "imag": dtft_seq.imag,
            },
            index = freq
        )

        data.to_csv(
```

```
            "../data/dtft_" + cplx["name"] + "_" + samp_freq + "_32.csv",
            sep = " ", index_label = "freq"
        )
```

We have generated DTFTs for our *sequences* with different *sampling frequencies*, and `cplx_seqs` dictionary has all those generated *input sequences*. Now we need to pad them with 32 zeros and find the DTFTs of those sequences. Let's quickly implement another *function* for that. Let it be `gen_dtft_seq_64`.

```
def gen_dtft_seq_64(
    cplx_seq, samp_freq, dtft_samp_count
    ):

    cplx_seq = np.concatenate(
        [cplx_seq, np.zeros(32, dtype = np.cdouble)]
    )

    freq = np.linspace(0, 1, dtft_samp_count)

    dtft_fn = dtft_factory(cplx_seq = cplx_seq)
    dtft_seq = dtft_fn(freq)

    # scaling back the freq from (0 - 1) to (0 - f_s)
    return freq * samp_freq, dtft_seq, cplx_seq
```

Now let's take the DTFTs of those sequences using this `gen_dtft_seq_64`.

```
dtft_samp_count = 2000

for cplx in cplx_seqs:
    for seq_name in cplx["sequences"]:
        freq, dtft_seq, _ = gen_dtft_seq_64(
            cplx_seq = cplx["sequences"][seq_name],
            samp_freq = samp_freqs[seq_name],
            dtft_samp_count = dtft_samp_count
        )

        data = pd.DataFrame(
            data = {
                "real": dtft_seq.real,
                "imag": dtft_seq.imag,
            },
            index = freq
        )

        data.to_csv(
            "../data/dtft_" + cplx["name"] + "_" + seq_name + "_64.csv",
            sep = " ", index_label = "freq"
        )
```

4.3 DTFT of Complex A

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 28t)$

DTFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

To much *aliasing*, can barely distinguish 28Hz.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Real and *imaginary* parts have same frequency, therefore that *bulge* at the pulse near 28.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

Similar to (b).

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Similar to (b) and (c).

Inference: We can distinguish almost two of the *frequencies* from four of the different *sampling frequencies*.

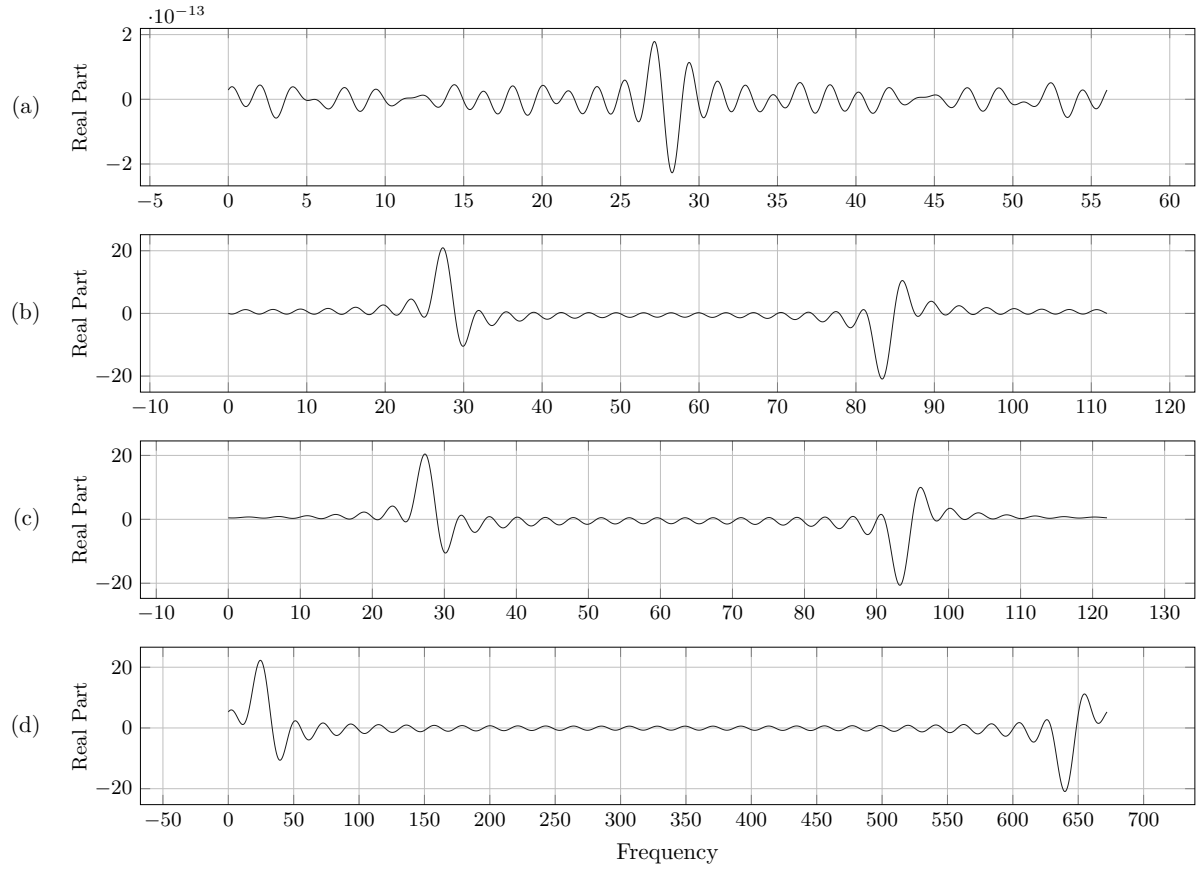

Figure 22: Plot of DTFTs of Complex A with different sampling frequencies

DTFT with 64 Samples

(a): $f_s = \frac{4 \times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Looks like this is similar to 32 samples. The padding of 32 zero didn't make any difference.

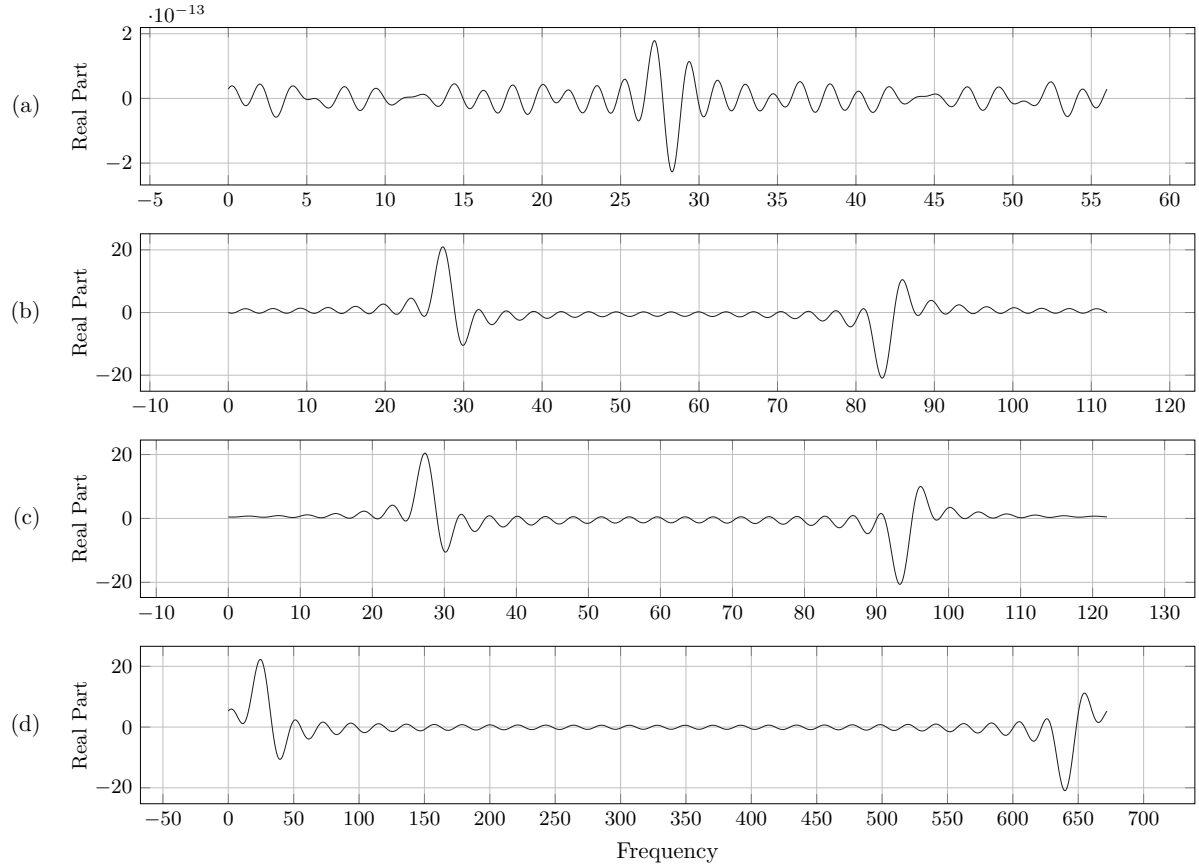

Figure 23: Plot of DTFTs of Complex A with different sampling frequencies

4.4 DTFT of Complex B

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

DTFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

To much *aliasing*, nothing can be inferred.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

We can see 28Hz but no sign of 56Hz

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

We can see both the frequencies. *Real* frequency 28Hz is at the *zero crossing* of the first impulse, and the *imaginary* frequency 56Hz is at the *tip* of the second pulse.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Similar to (c), but a bit hard to distinguish.

Inference: It's looks like the frequency of *sine* in the *real* part can be found as a *zero crossing N* shaped pulse. And a frequency of *sine* in *imaginary* part can be found at the *tip* of an *impulse* shaped pulse.

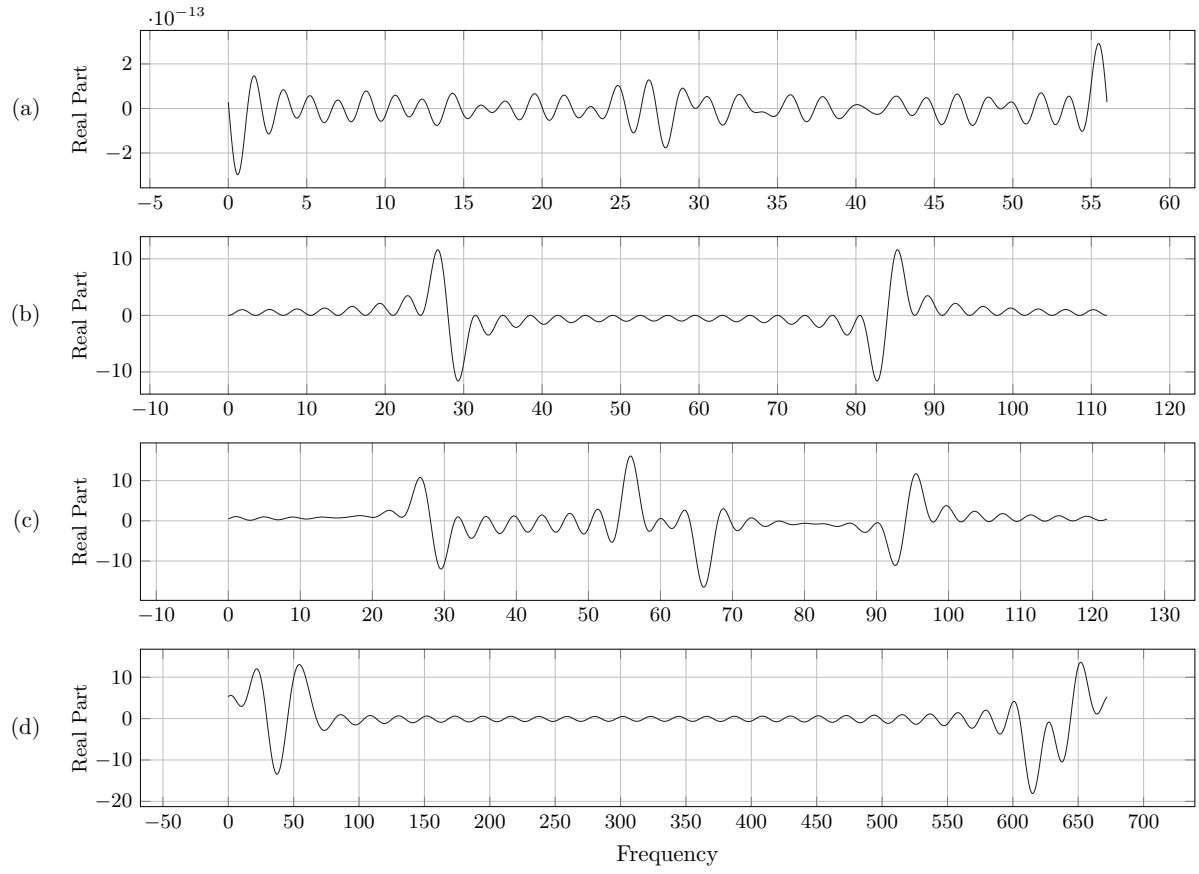

Figure 24: Plot of DTFTs of Complex B with different sampling frequencies

DTFT with 64 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Similar to 32 point, padding made no difference.

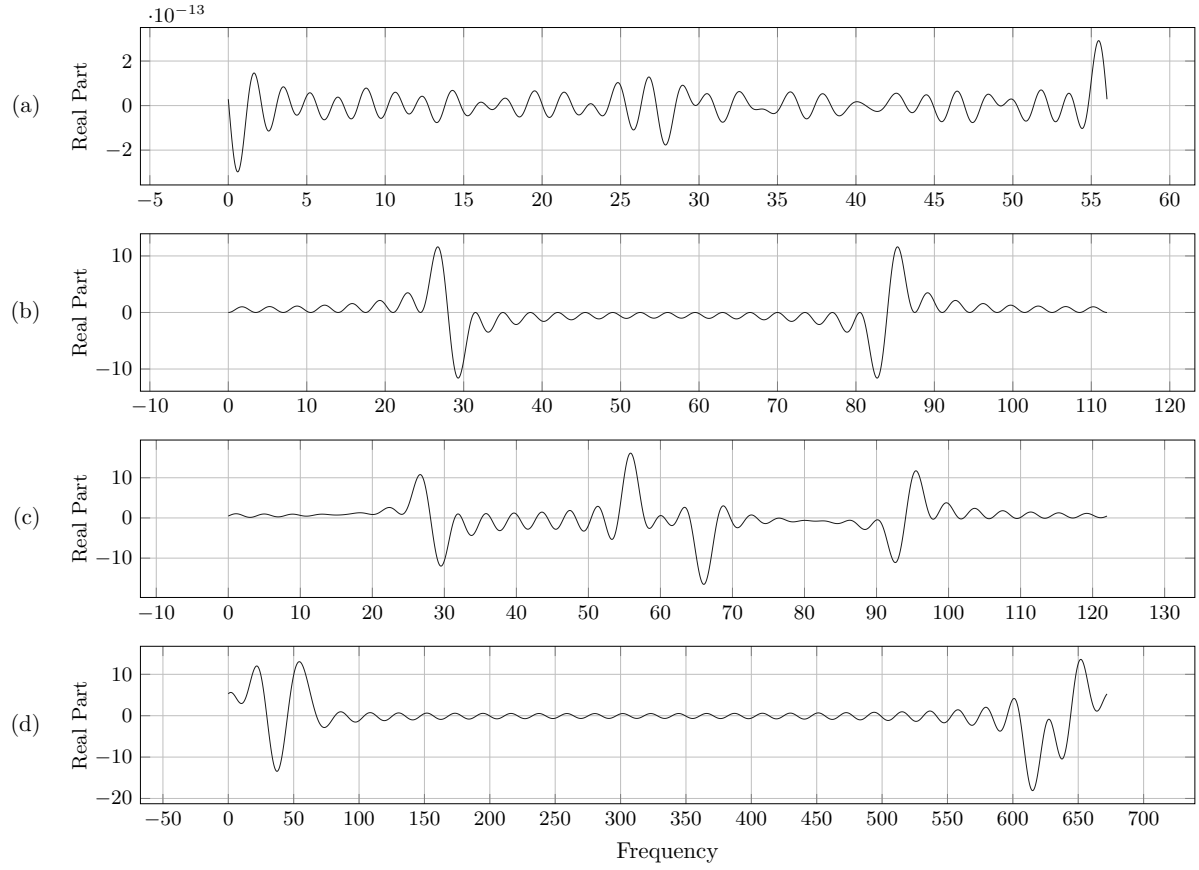

Figure 25: Plot of DTFTs of Complex B with different sampling frequencies

4.5 DTFT of Complex C

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56.1t)$

DTFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

Too much *aliasing*.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Only 28Hz can be inferred.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

Both can be inferred, but maybe hard to pinpoint the exact frequency of *imaginary* part as 56.1Hz

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Both can be inferred, kind of...

Inference: Similar to Complex B.

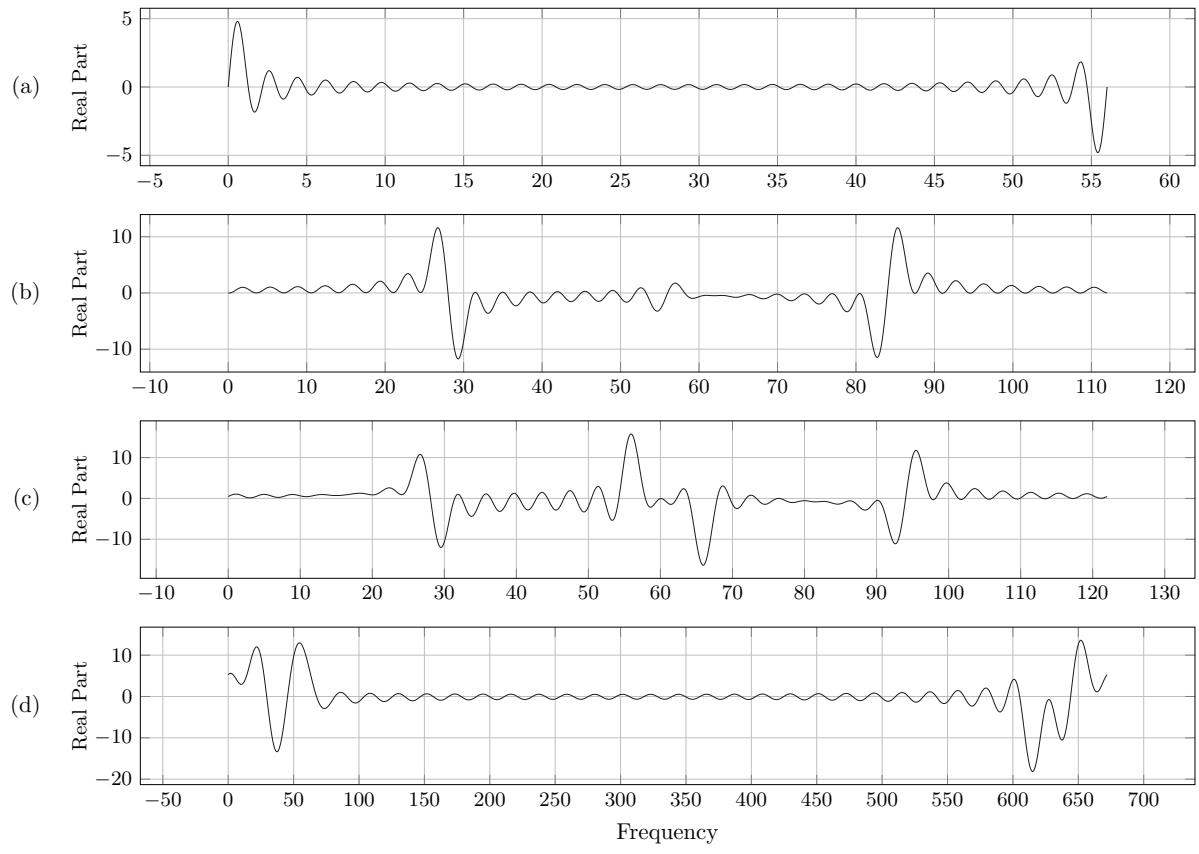

Figure 26: Plot of DTFTs of Complex C with different sampling frequencies

DTFT with 64 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Similar as 32 point.

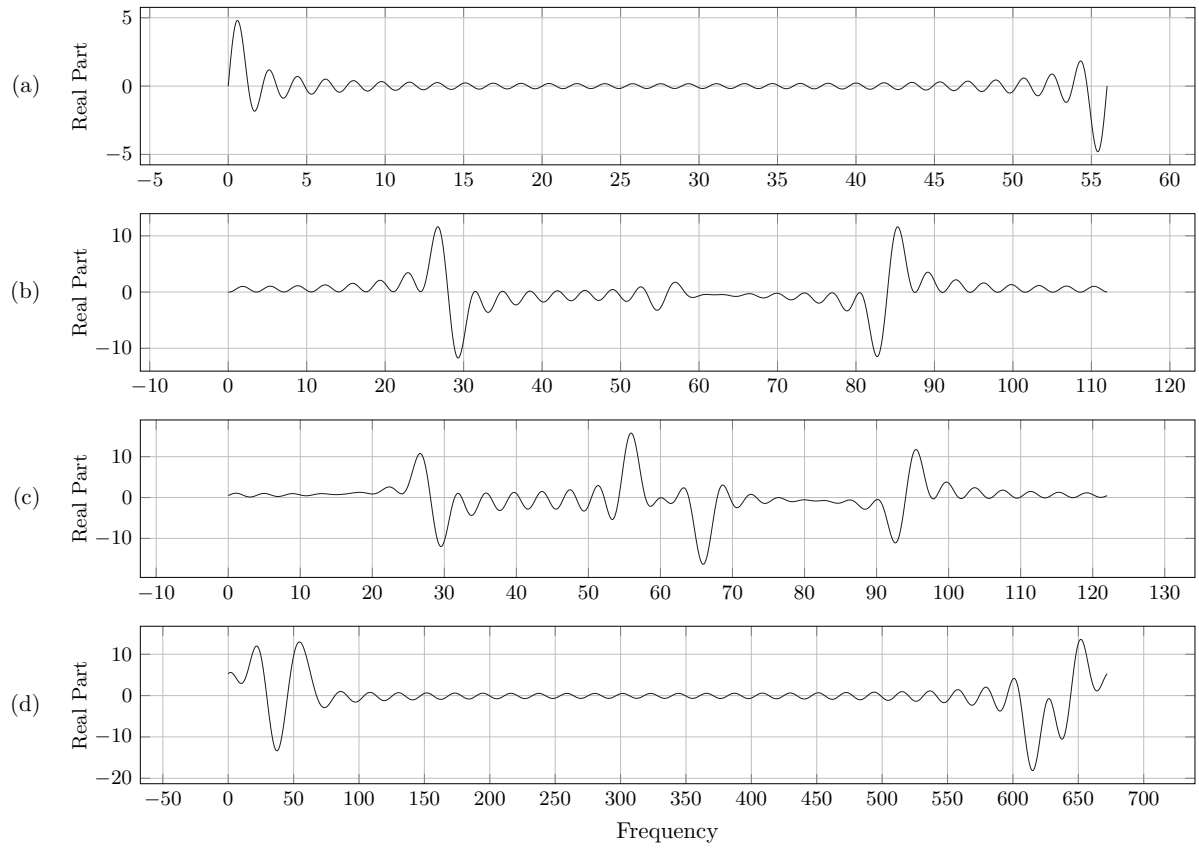

Figure 27: Plot of DTFTs of Complex C with different sampling frequencies

4.6 DTFT of Complex F

Sequence: $\sin(2\pi 56t) + j\sin(2\pi 56.1t)$

DTFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

Too much *aliasing*, causing to misidentify 1Hz as one of the frequency components.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Has *aliasing*, barely identify 56Hz.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

Can identify a bulging, but it would be hard to 56.1Hz from 56Hz.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Similar to (c).

Inference: Maybe we need much more *samples* of the input sequence, to identify, closer *frequencies*.

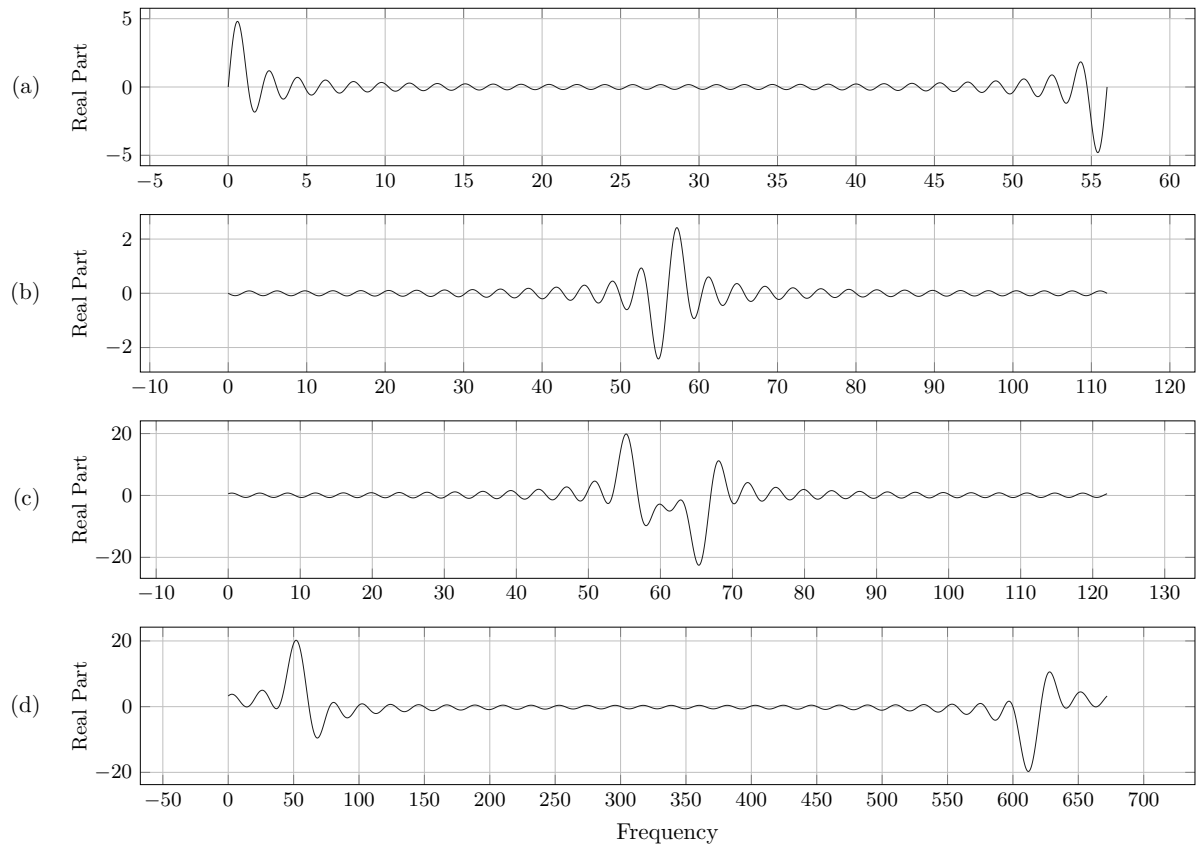

Figure 28: Plot of DTFTs of Complex F with different sampling frequencies

DTFT with 64 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Similar to 32 point.

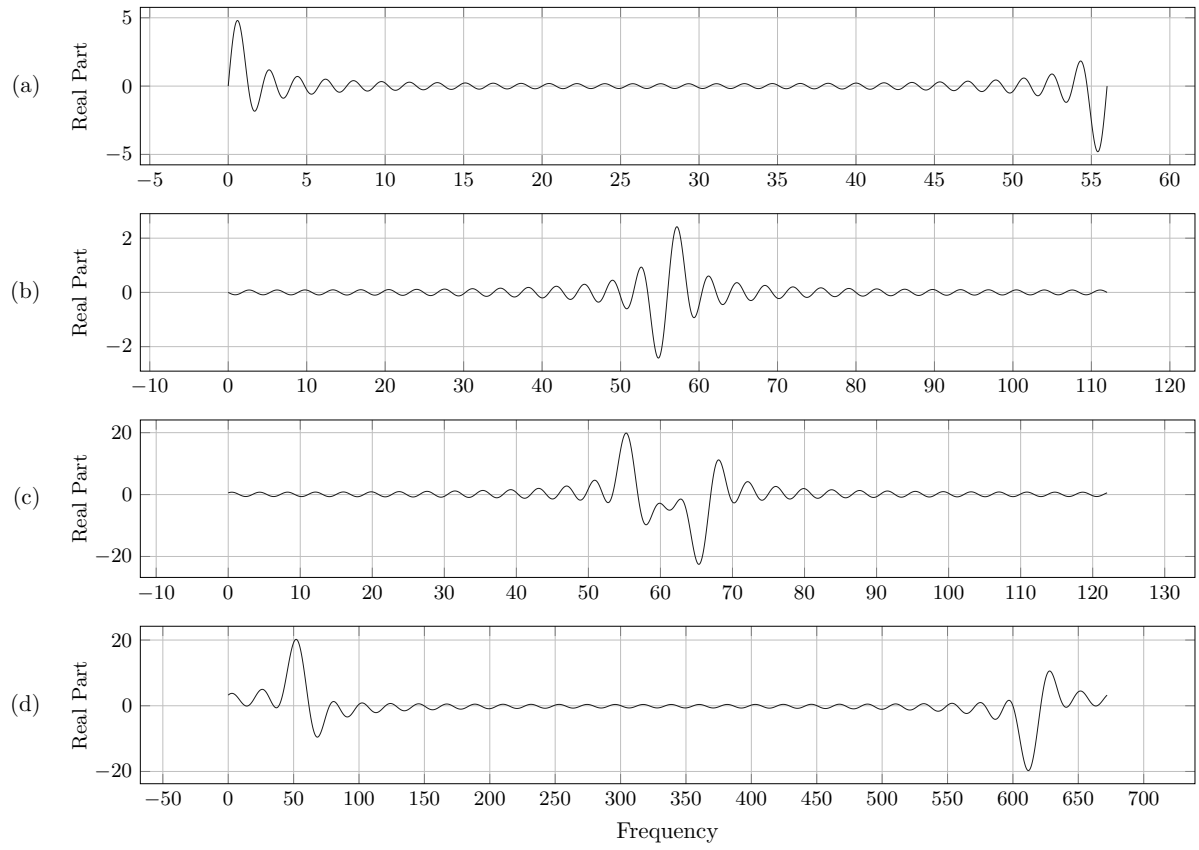

Figure 29: Plot of DTFTs of Complex F with different sampling frequencies

5 Taking DFT of the Complex Sequences

In the previous Section, we took DTFTs of the *sequences* we have generated. In this Section, we will be taking DFTs of those same signals and we can compare our findings. But before that let's take a look at what DFTs are, and how to compute them.

5.1 Discrete Fourier Transform

Discrete Fourier Transforms or DFTs are just a special case of previously mentioned DTFTs. DTFTs take a *discrete* input sequence and produces a *continuous* signal. In the case of DFT, they take *discrete* input but produces a signal in the *discrete* form itself. In other words, if we *sample* DTFT we will get DFT. Mathematically they can be described as,

$$X[k] = X(e^{j\omega})|_{\omega=\frac{2\pi k}{N}} \tag{12}$$

$$= \sum_{n=0}^{N-1} x[n]e^{-jwn}\Big|_{\omega=\frac{2\pi k}{N}} \tag{13}$$

$$= \sum_{n=0}^{N-1} x[n]e^{\left(-j\frac{2\pi k}{N}n\right)} \tag{14}$$

where,

$X[k]$: is the DFT itself.

where,

k: varies from 0 to $N-1$

N: is the total *sample count*.

$x[n]$: is the *input sequence*.

5.2 Implementing DFT using Python

Just like we did with DTFTs let's implement a similar `dtf_factory` that would take *complex sequence* and returns a *function* that can be called with k for each of the values of DFT.

```
import numpy as np

def dft_factory(cplx_seq):

    sample_count = len(cplx_seq)
    def dft(k):
        sum = 0
        # see eq. (14)
        for n, cplx in enumerate(cplx_seq):
            sum = sum + cplx * np.exp(-1j * 2 * np.pi * k * n / sample_count)
        return sum

    return dft
```

Now we need the `cplx_factory` from the previous section.

```
def cplx_factory(samp_freq, real_freq, imag_freq):
    samp_period = 1 / samp_freq
    return lambda n: np.sin(
        2 * np.pi * real_freq * (n * samp_period)
    ) + 1j * np.sin(
        2 * np.pi * imag_freq * (n * samp_period)
    )
```

Now let's implement a *function* similar to that of `gen_dtft_seq` for DFT that will generate the DFT sequence for our *complex sequences*. Let us call it..., you guessed it, the `gen_dft_seq`!

```
def gen_dft_seq(
    samp_count, samp_freq, real_freq, imag_freq
    ):

    cplx_fn = cplx_factory(
        samp_freq = samp_freq,
        real_freq = real_freq,
        imag_freq = imag_freq
    )

    cplx_seq = np.ndarray(samp_count, dtype = np.cdouble)

    for i in range(samp_count):
        cplx_seq[i] = cplx_fn(i)

    freq = np.arange(samp_count)

    dft_fn = dft_factory(cplx_seq = cplx_seq)
    dft_seq = dft_fn(freq)

    # we need to scale freq back
    return freq * samp_freq / samp_count, dft_seq, cplx_seq
```

Now that we have `gen_dft_seq`, next we need a to compute 32 point DFTs using all of these functions.

```
import pandas as pd

X = 28

cplx_seqs = [
    {
        "name": "complex_a",
        "real_freq": X,
        "imag_freq": X,
        # we will fill this later, for taking 64 point DFT
        "sequences": {},
    },
    {
        "name": "complex_b",
        "real_freq": X,
```

```
        "imag_freq": 2 * X,
        "sequences": {},
    },
    {
        "name": "complex_c",
        "real_freq": X,
        "imag_freq": (2 * X) + 0.1,
        "sequences": {},
    },
    {
        "name": "complex_f",
        "real_freq": 2 * X,
        "imag_freq": (2 * X) + 0.1,
        "sequences": {},
    },
]

samp_freqs = {
    "normal":           int(4 * X),
    "slightly_greater": int(4 * X + 6),
    "much_greater":     int(4 * X * 6),
    "much_lesser":      int(4 * X / 2),
}

samp_count = 32

for cplx in cplx_seqs:

    for samp_freq in samp_freqs:

        freq, dft_seq, cplx_seq = gen_dft_seq(
            samp_count = samp_count,
            samp_freq = samp_freqs[samp_freq],
            real_freq = cplx["real_freq"],
            imag_freq = cplx["imag_freq"]
        )

        # keeping generated sequences for taking 64 point DFT
        cplx["sequences"][samp_freq] = cplx_seq

        data = pd.DataFrame(
            data = {
                "real": dft_seq.real,
                "imag": dft_seq.imag,
            },
            index = freq
        )

        data.to_csv(
            "../data/dft_" + cplx["name"] + "_" + samp_freq + "_32.csv",
            sep = " ", index_label = "freq"
```

```
    )
```

Next thing to do is to convert the above 32 sized samples to 64 sized samples by *zero padding*. Then we need to find 64 point DFTs. Let's quickly implement another *function* called `gen_dft_seq_64`.

```
def gen_dft_seq_64(
    cplx_seq, samp_freq
    ):

    cplx_seq = np.concatenate(
        [cplx_seq, np.zeros(32, dtype = np.cdouble)]
    )

    freq = np.arange(64)

    dft_fn = dft_factory(cplx_seq = cplx_seq)
    dft_seq = dft_fn(freq)

    # we need to scale freq back
    return freq * samp_freq / 64, dft_seq, cplx_seq
```

Now let's take the DFTs of our sequences using `gen_dft_seq_64`.

```
for cplx in cplx_seqs:
    for seq_name in cplx["sequences"]:
        freq, dft_seq, _ = gen_dft_seq_64(
            cplx_seq = cplx["sequences"][seq_name],
            samp_freq = samp_freqs[seq_name]
        )

        data = pd.DataFrame(
            data = {
                "real": dft_seq.real,
                "imag": dft_seq.imag,
            },
            index = freq
        )

        data.to_csv(
            "../data/dft_" + cplx["name"] + "_" + seq_name + "_64.csv",
            sep = " ", index_label = "freq"
        )
```

5.3 DFT of Complex A

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 28t)$

DFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

There is no *hope* in *alias land*, we could see something between 25 30.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

We could see a spike at somewhat near the vicinity of 28Hz.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

There is a little bit of bulge can be seen near 28Hz.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

There is little bit bulge at the left end.

Inference: It seems like if we only increase the *sampling frequency*, but keep the *sample count* same, the *uncertainty* also increases.

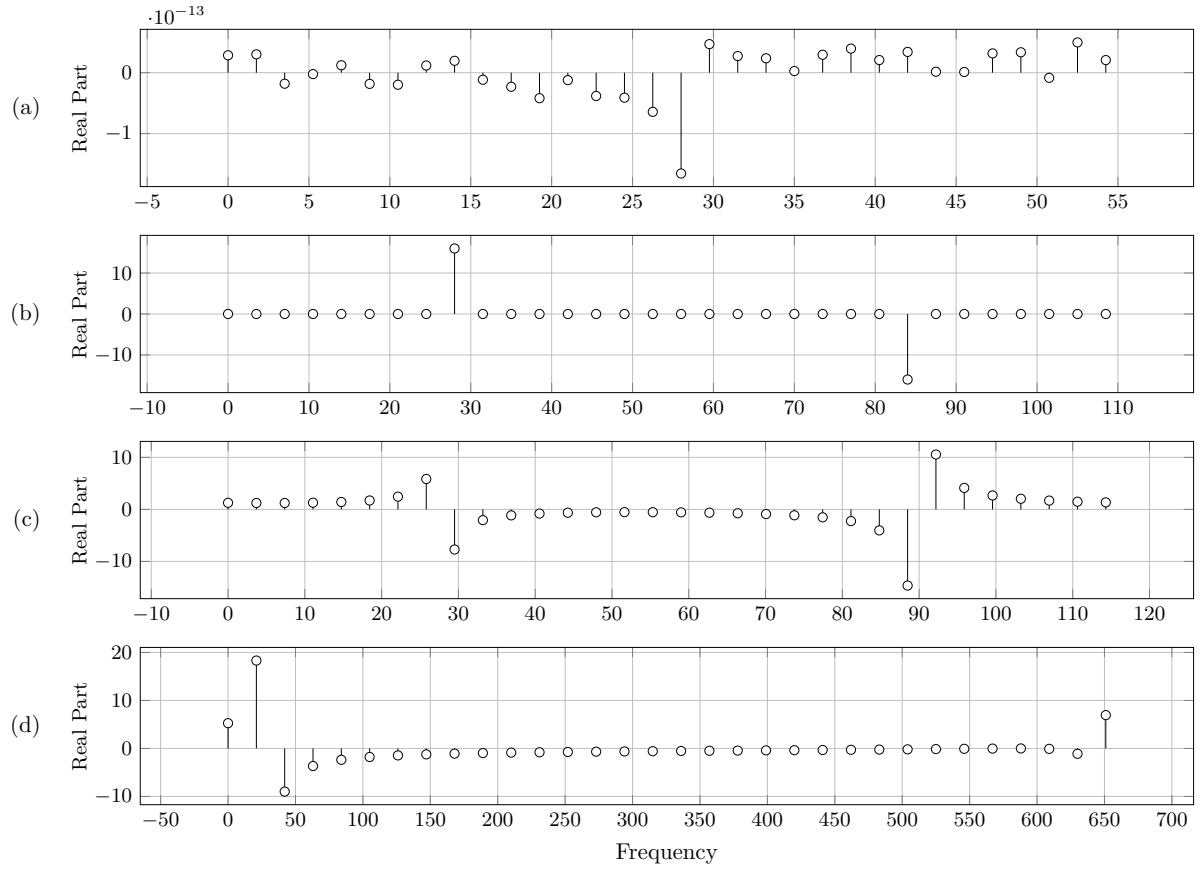

Figure 30: Plot of DFTs of Complex A with different sampling frequencies.

DFT with 64 Samples

(a): $f_s = \frac{4 \times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Surprisingly, 64 point increases the resolution to *double*. And we can now see more information from the *spectrum*.

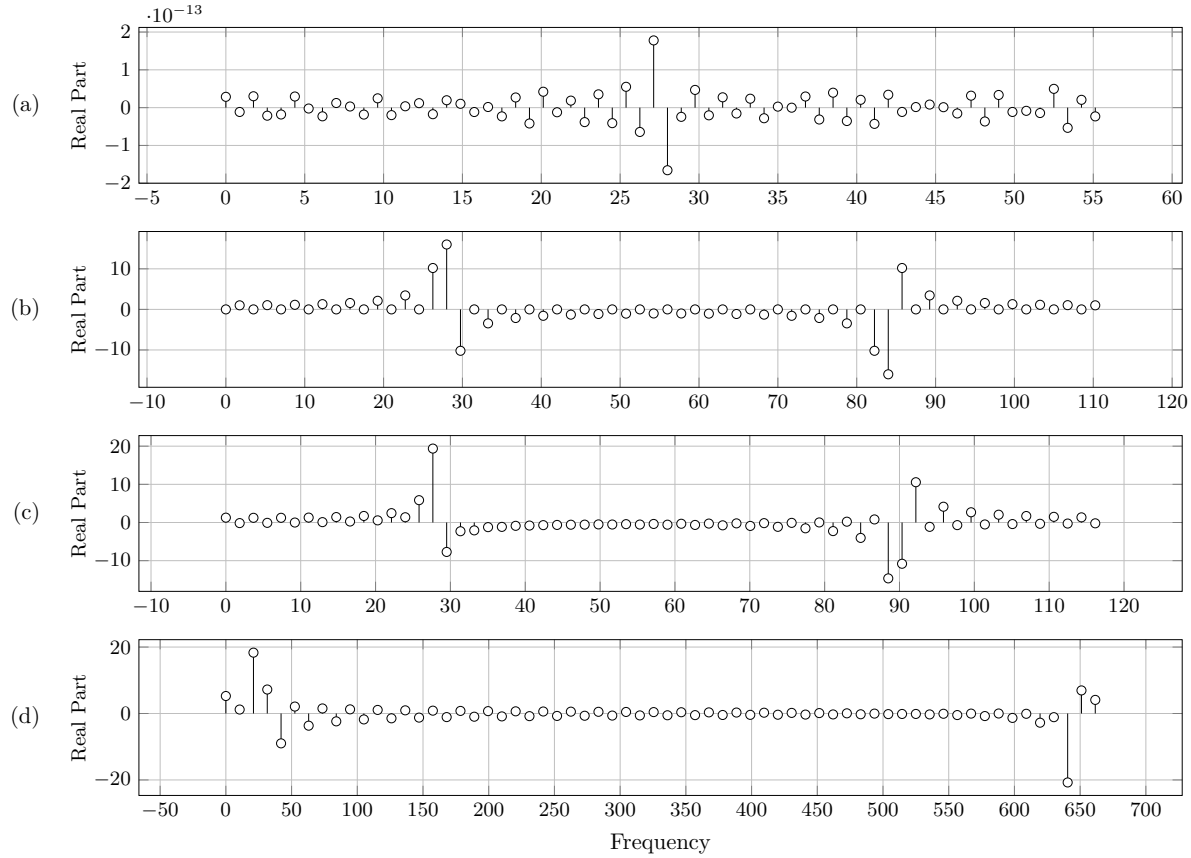

Figure 31: Plot of DFTs of Complex A with different sampling frequencies.

5.4 DFT of Complex B

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

DFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

Alias land, no hope.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Could see some noise.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

We can kind of see two of the frequencies.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Uncertainty increases.

Inference: Same as Complex A.

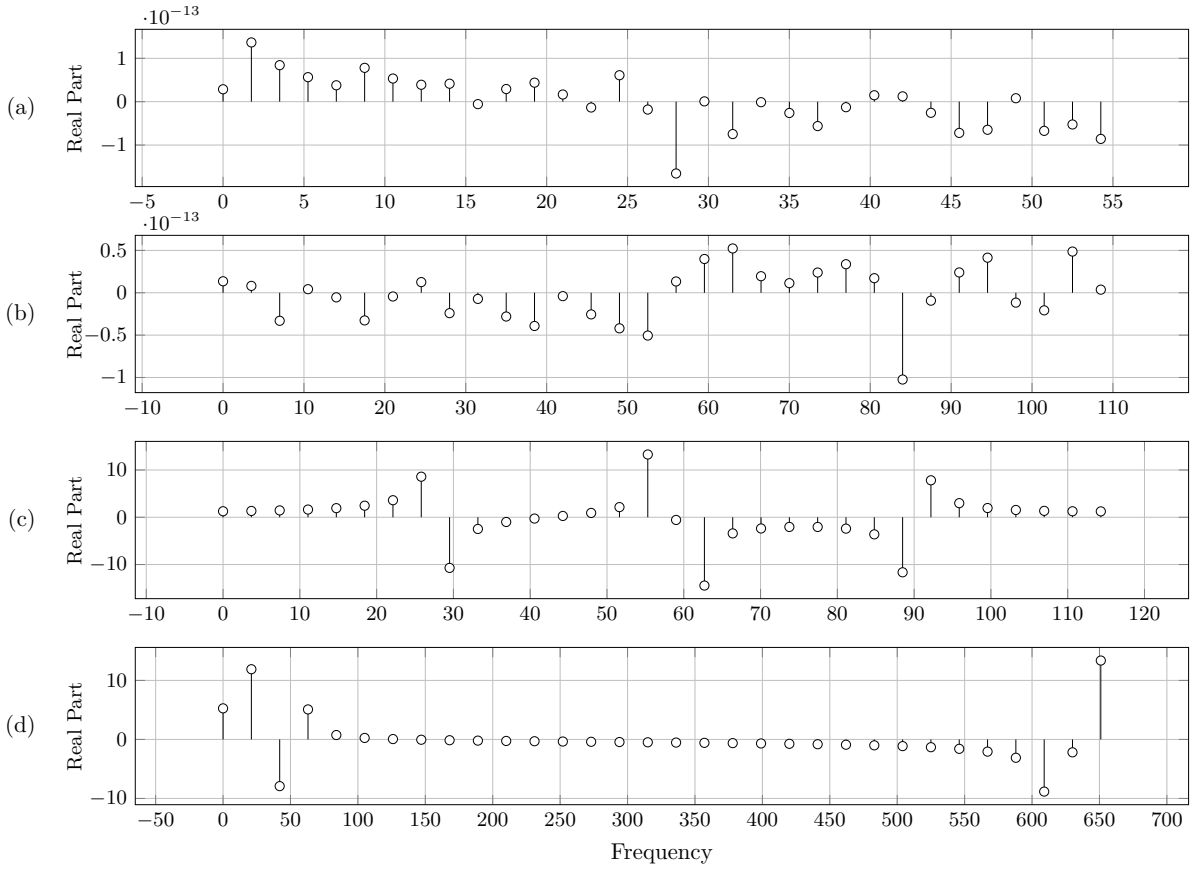

Figure 32: Plot of DFTs of Complex B with different sampling frequencies.

DFT with 64 Samples

(a): $f_s = \frac{4 \times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Similar to COMPLEX A. Doubles the resolution.

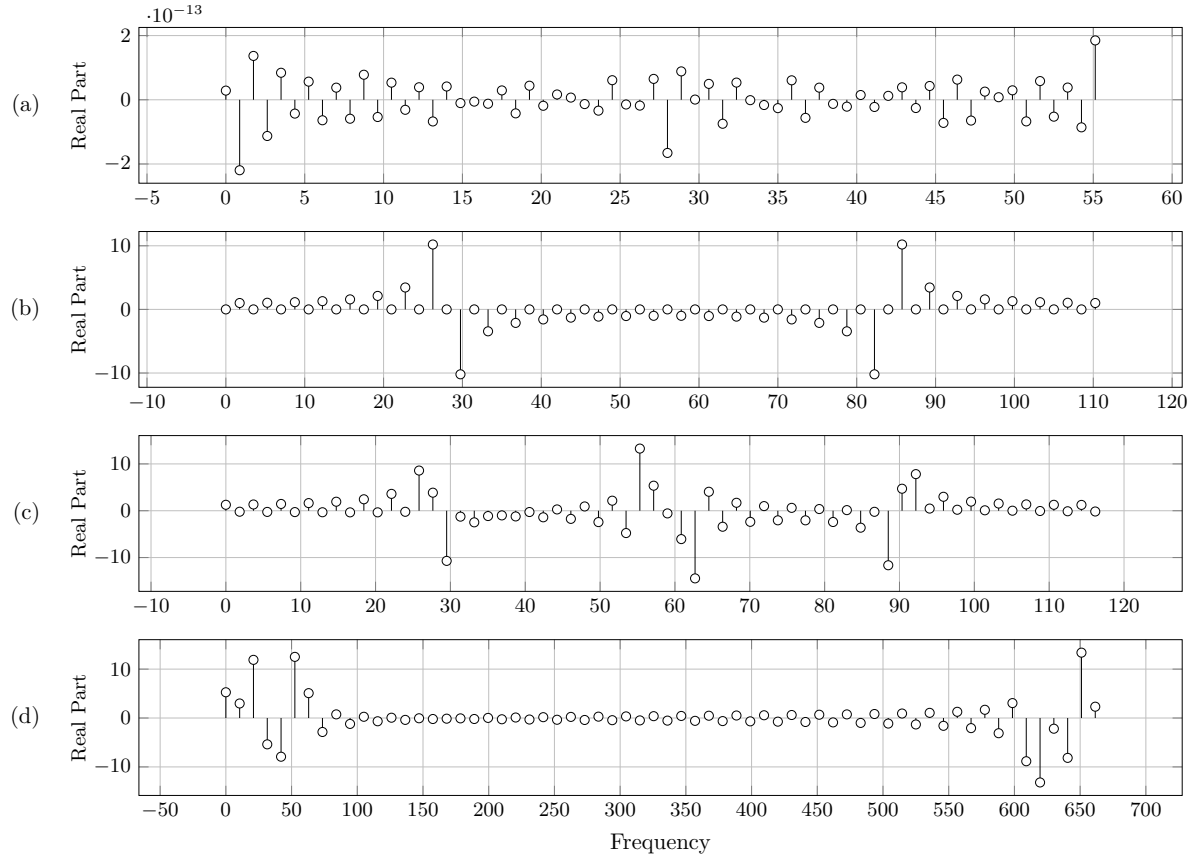

Figure 33: Plot of DFTs of COMPLEX B with different sampling frequencies.

5.5 DFT of Complex C

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56.1t)$

DFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

Aliasing causing, misinformation.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Cant's see 28Hz but something is present near the 50s ranges.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

Can almost see both.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Similar as above.

Inference: Similar to above two *sequences*.

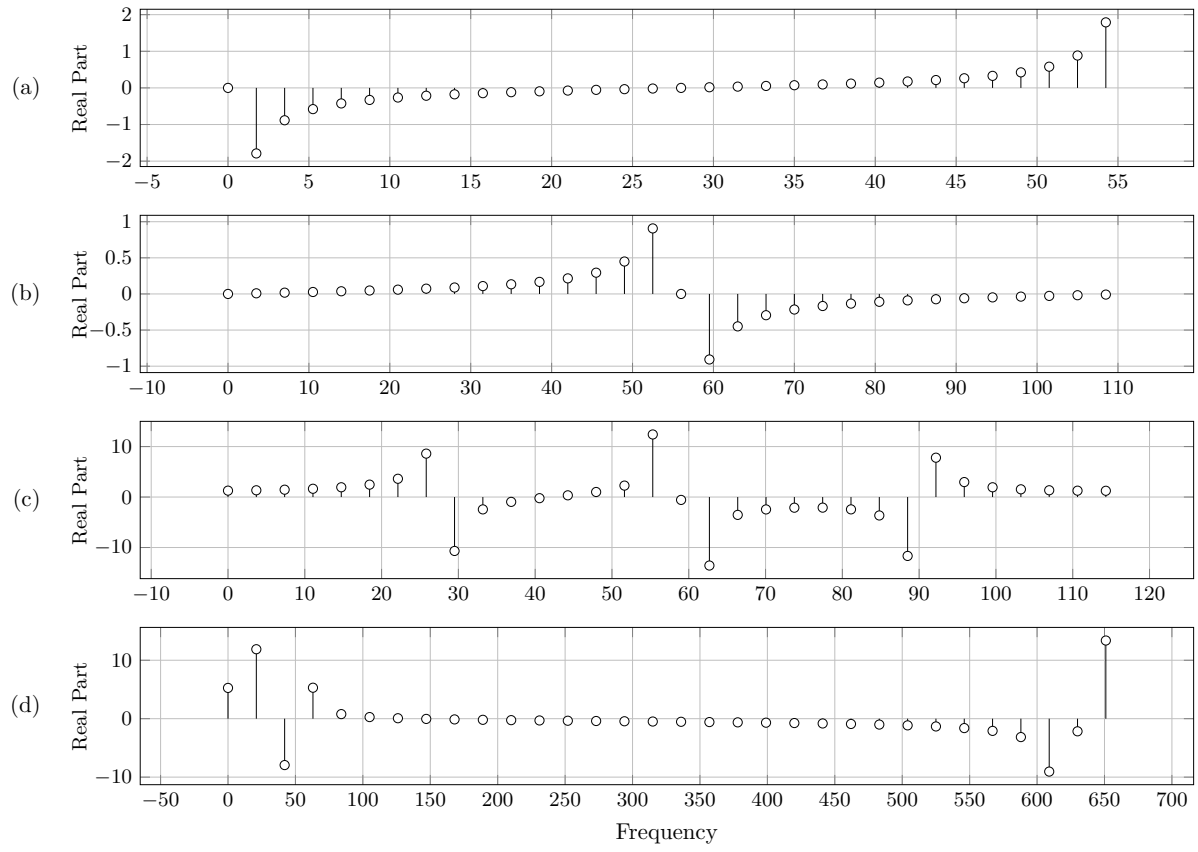

Figure 34: Plot of DFTs of Complex C with different sampling frequencies.

DFT with 64 Samples

(a): $f_s = \frac{4 \times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Resolution increases, so does if there is *aliasing*.

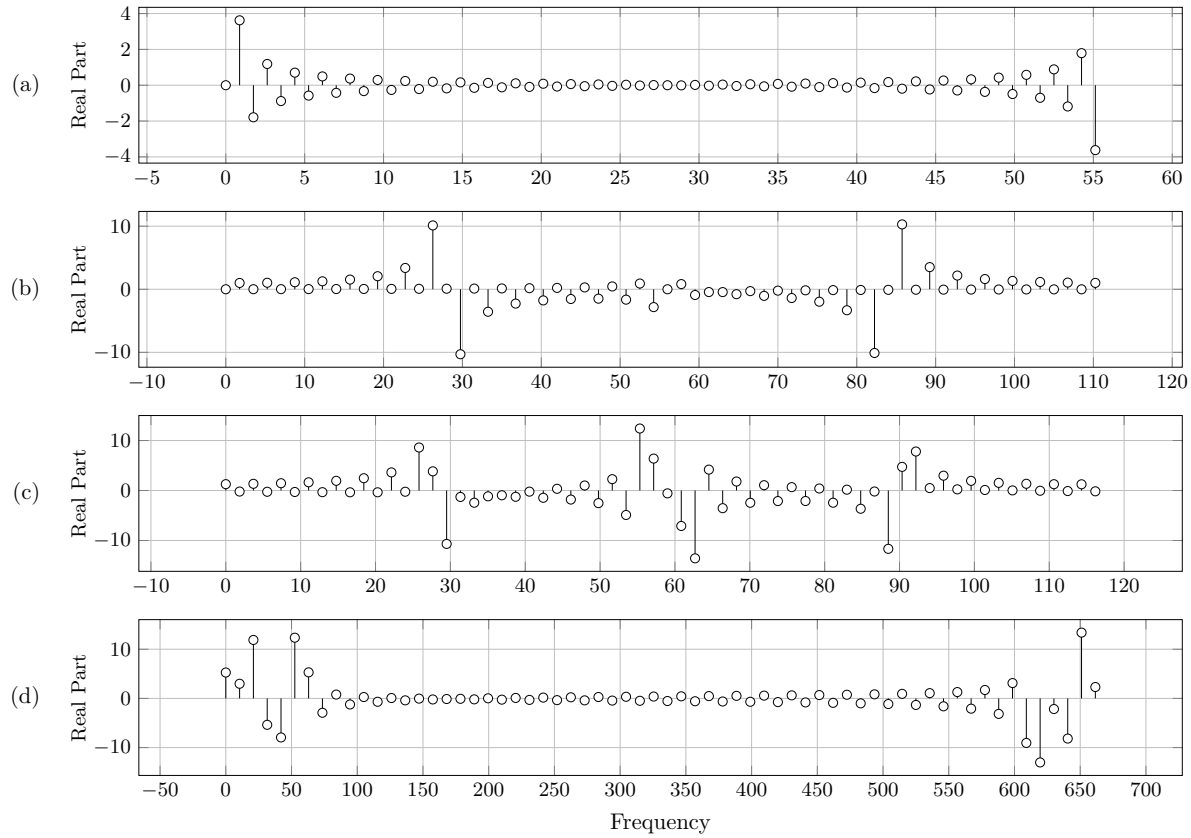

Figure 35: Plot of DFTs of Complex C with different sampling frequencies.

5.6 DFT of Complex F

Sequence: $\sin(2\pi 56t) + j\sin(2\pi 56.1t)$

DFT with 32 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

Aliasing.

(b): $f_s = 4 \times 28 = 112\text{Hz}$

Unable to distinguish between two of the similar frequencies.

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

Unable to distinguish between two of the similar frequencies.

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Same as above.

Inference: Similar to above sequences.

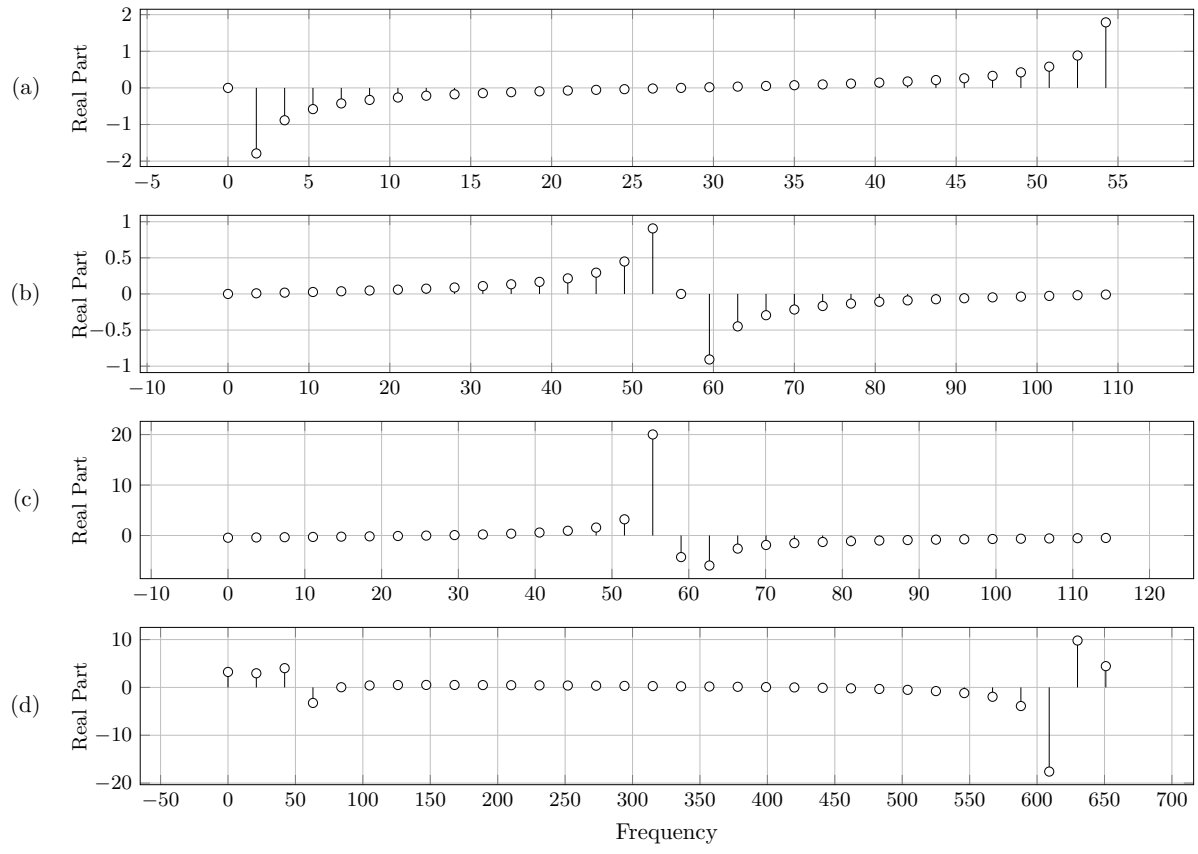

Figure 36: Plot of DFTs of Complex F with different sampling frequencies.

DFT with 64 Samples

(a): $f_s = \frac{4\times 28}{2} = 56\text{Hz}$

(b): $f_s = 4 \times 28 = 112\text{Hz}$

(c): $f_s = (4 \times 28) + 10 = 122\text{Hz}$

(d): $f_s = 4 \times 28 \times 6 = 672\text{Hz}$

Inference: Resolution increases.

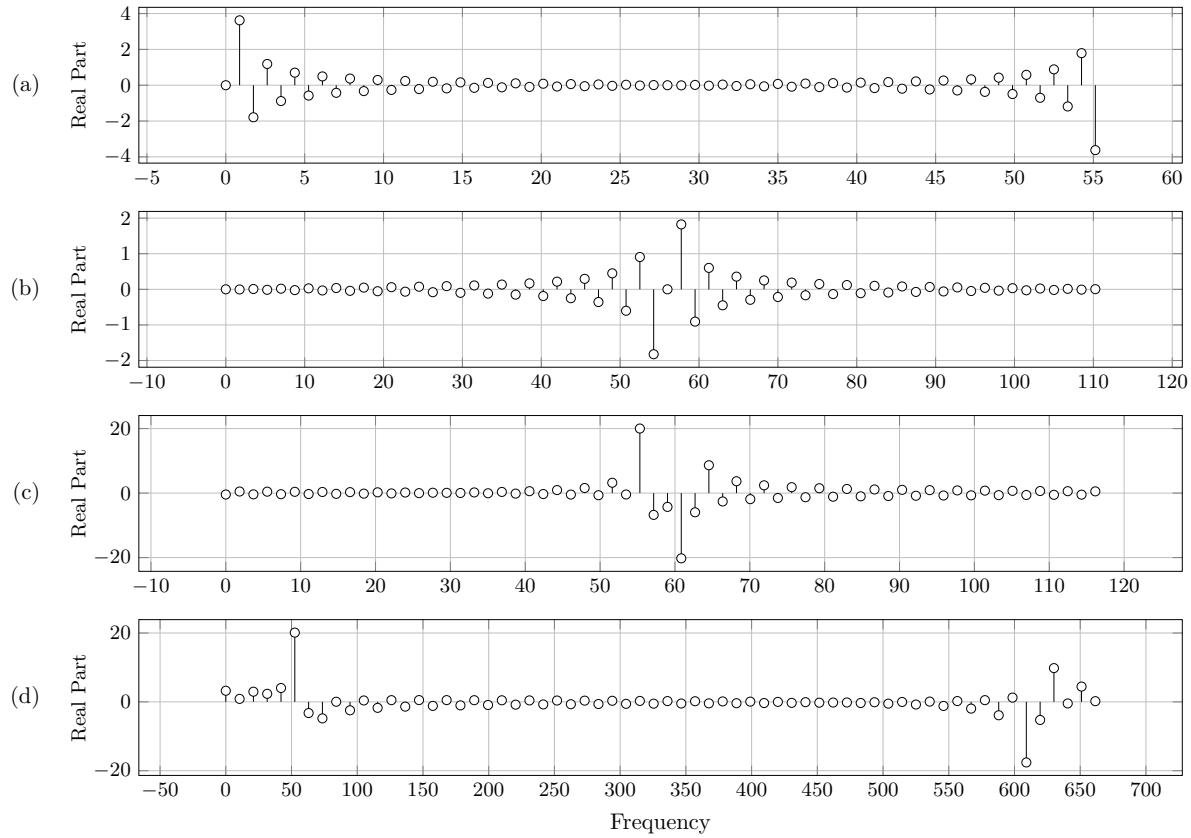

Figure 37: Plot of DFTs of Complex F with different sampling frequencies.

6 Conclusion

In the previous sections we have generated a bunch of *complex signals*. And we have sampled them and analysed their *frequency spectrum* with different *sampling frequencies*. And we took the 32 point DTFTs and DFTs for all of these *sequences*. Then we have padded them with 32 more zeros and found the corresponding DTFTs and DFTs. And we've observed some *interesting* observations. In this section, we will be seeing some of those observation.

Zero Padding: In the case of DTFTs, *zero padding* is not that useful, but in the case of DFTs *zero padding* and essentially taking a 64 point DFT seem to improve the *frequency spectrum* in some sense. And it seems like if we keep *zero pad* more and more and take *higher* point DFT, the DFT will slowly turn into the corresponding DTFT.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

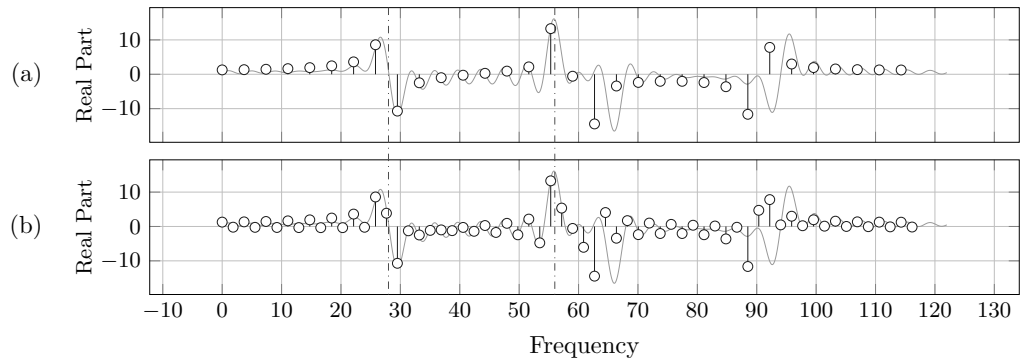

Figure 38: Comparing 32 point DFT and 64 point DFT of COMPLEX B, both having *sampling frequency* of 112Hz. Also the 64 point DTFT with the same *sampling frequency* is given as a reference. (a): 32 point DFT of COMPLEX B. (b): 64 point DFT of COMPLEX B.

Mirroring of Frequencies: Figure 21 depicts an interesting physical property of DTFTs. DTFTs and DFTs map the *frequencies* in the $\mathcal{Z}$-PLANE around the UNIT CIRCLE, essentially causing them to *repeat* after a *period*. From all the DTFTs and DFTs that we have taken, it is evident that the values got from *sweeping* from π to 2π range[13] is essentially a *mirror* of values obtained[14] from *sweeping* ω from 0 to π range.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

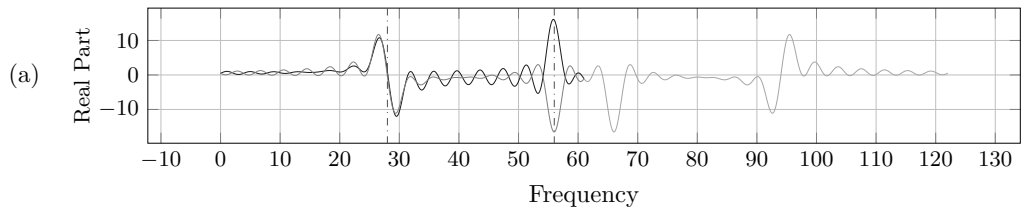

Figure 39: Frequencies from 61Hz to 122Hz mirroring frequencies from 0Hz to 61Hz in DTFT of COMPLEX B, having *sampling frequency* of 122Hz.

In Figure 39 we can see that they don't *exactly* overlap. This is because in our case the *input sequence* was not a *real sequence* instead it was a *complex sequence*. If

[13] in the case of ω.

[14] not *always*, but the *information content* will be the same.

the *input* was a *real sequence*, the DTFT would have shown so called a *hermitian symmetry*[15]. Neither less, they convey the same *information*. Or so to say, the *information content* in range 0Hz to 61Hz *mirrors into* 61Hz to 122Hz range.

Half of Sampling Frequency: The importance of above *observation*[16] is that we can only *retrieve* or *extract* the *frequencies* upto $\frac{f_s}{2}$ in the case of a signal *sampled* with a *sampling frequency* of f_s. The rest will be the *mirror* of *frequencies* ranging from 0 to $\frac{f_s}{2}$. In the case COMPLEX B with *sampling frequency* 122Hz, we can only *extract frequencies* upto 61Hz.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

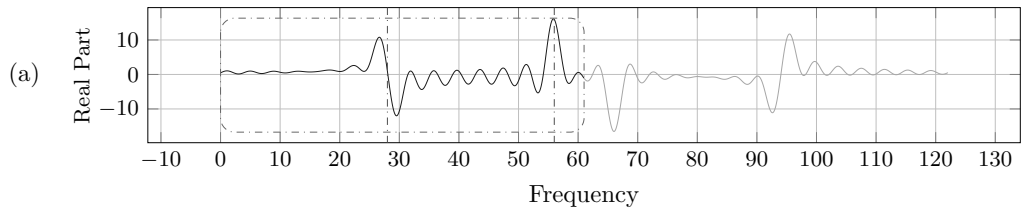

Figure 40: Usable *information* from 0Hz to 61Hz range in 32 point DTFT of COMPLEX B having *sampling frequency* of 122Hz.

Information Content: All the plots of DTFTs and DFTs that we've plotted are actually only the *real part* of the corresponding *frequency spectrum*. But what about the *imaginary part*? Let's plot the *real part* and corresponding *imaginary part* of one of the *frequency spectrum*. Let's take COMPLEX B.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

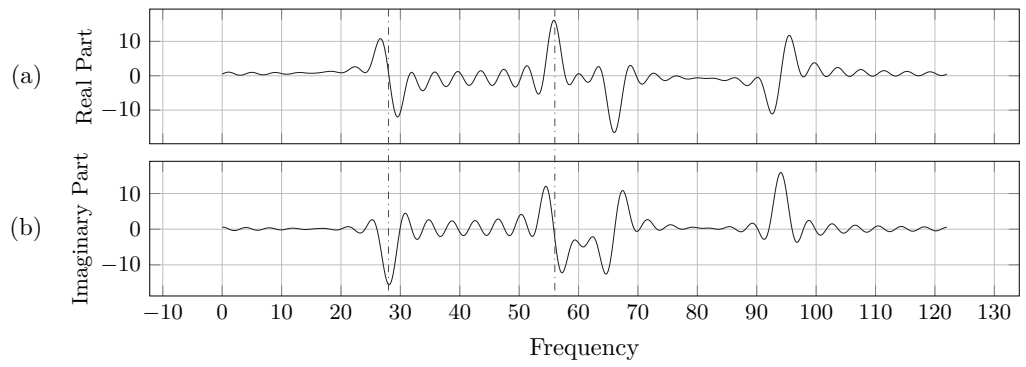

Figure 41: Comparing *real* and *imaginary* parts of 32 point DTFT of COMPLEX B having *sampling frequency* of 122Hz. (a): *Real part* of DTFT. (b): *Imaginary part* of DTFT.

From Figure 41 we can see that we actually get some sense about each of the *frequency components* from both the *real* and *imaginary* parts equally, but if we want to know where each *frequency* belong[17] or the *phase* of these *frequency components*, we need to analyse both of them. If we are only *interested* in the *frequency components*, we could just take the *absolute value* of our *frequency spectrum*. That would give us the

[15] symmetry that seen in *hermitian functions* (see the **Wikipedia article**).
[16] mirroring of frequencies.
[17] like, to the *real part* or the *imaginary part*.

amplitude spectrum of the *frequency spectrum*. *Amplitude spectrum* will have *clean pulses*[18] corresponding to each of the *frequency components*.

Real Part Information: If we see any of the features[19] like in Figure 42, in the *real part* of the *frequency spectrum*, we can conclude the *existence* of that *spectral components* in the *input sequence*. But where[20] or what[21] they are can't be determined just alone from the *real part*.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

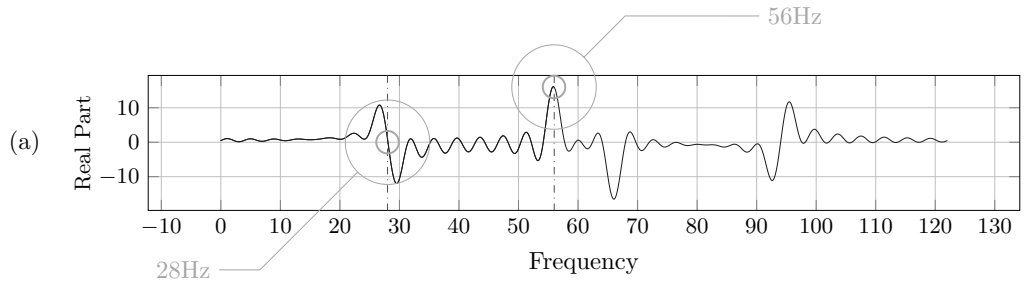

Figure 42: Features in the *real part* of 32 point DTFT that corresponds to the two *frequencies* in COMPLEX B having *sampling frequency* of 122Hz.

Imaginary Part Information: If we see any of the features[22] like in Figure 43, in the *imaginary part* of the *frequency spectrum*, we can conclude the *existence* of that *spectral components* in the *input sequence*. But where[23] or what[24] they are can't be determined just alone from the *imaginary part*.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

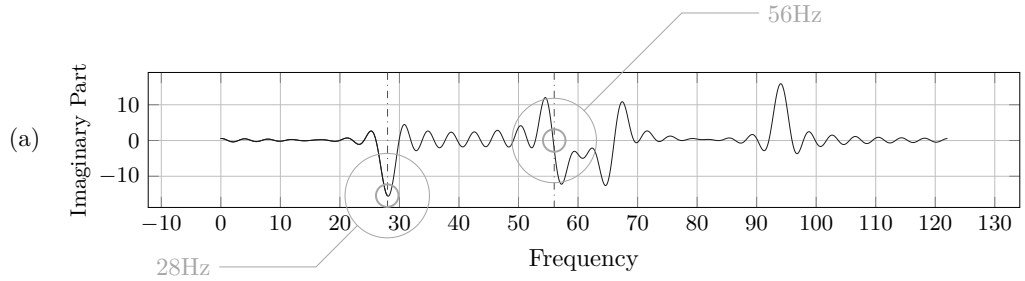

Figure 43: Features in the *imaginary part* of 32 point DTFT that corresponds to the two *frequencies* in COMPLEX B having *sampling frequency* of 122Hz.

Way Higher Sampling Frequency: In Figure 44 we can almost see the two *frequencies*. But if we *increase* the *sampling frequency* again without *increasing* the *sample count*[25], the DTFT or DFT might lose those *frequency information*. It seems like we need to *increase* the *sample count* as well as the *sampling frequency* in order to get a proper *frequency spectrum*.

[18] the *width* and the *height* of these pulses will vary depending on the *sample count*, ie, if we take *higher* point DFT, the *pulses* will be sharp.
[19] as inside the *marking circles*.
[20] see footnote 17.
[21] like, *sine* or *cosine* component.
[22] see footnote 19
[23] see footnote 17.
[24] see footnote 21.
[25] in our case it was 32 and 64.

Sequence: $\sin(2\pi 28t) + j\sin(2\pi 56t)$

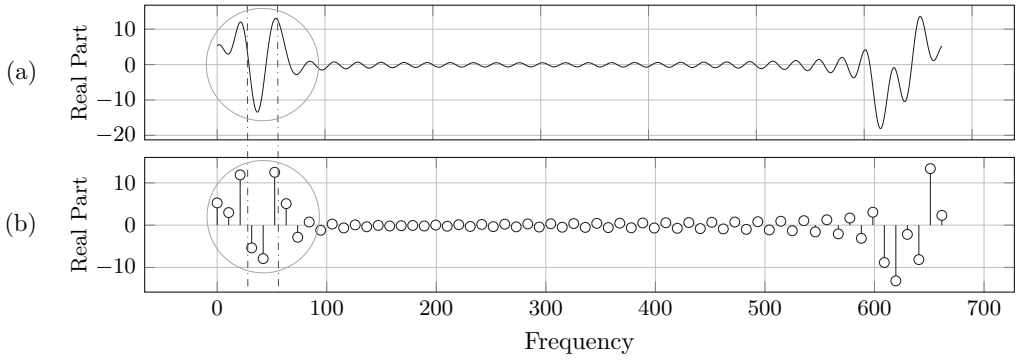

Figure 44: 64 point DTFT and DFT of COMPLEX B, both having *sampling frequency* of 122Hz. (a): 64 point DTFT. (b): 64 point DFT.

Detection of Closer Frequencies: From the DTFTs and DFTs of COMPLEX F from Section 3.6, we can see that it is hard to distinguish between the *closer frequencies* like 56Hz and 56.1Hz.

Sequence: $\sin(2\pi 56t) + j\sin(2\pi 56.1t)$

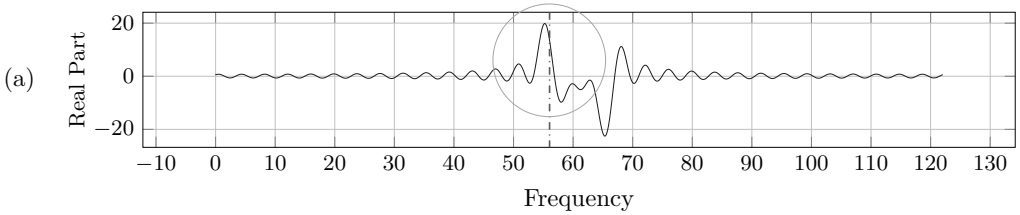

Figure 45: 56Hz and 56.1Hz being *undistinguishable* from 32 point DTFT of COMPLEX F having *sampling frequency* of 122Hz.

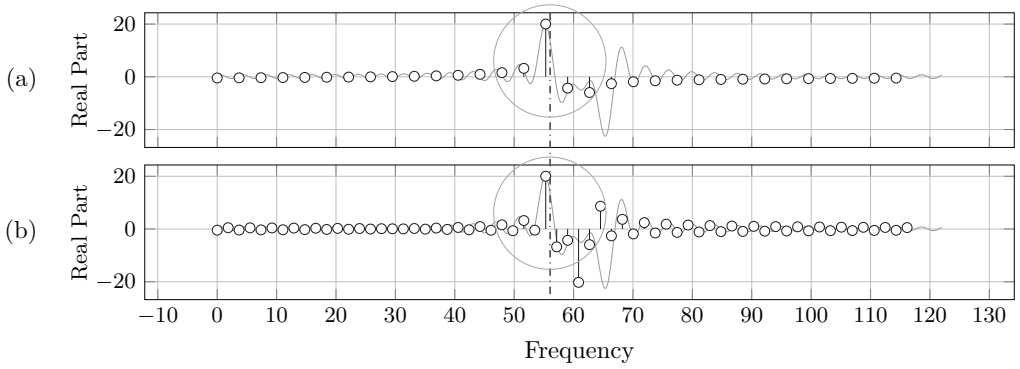

Figure 46: 32 point DFT and 64 point DFT with corresponding N point DTFTs of COMPLEX F with *sampling frequency* of 122Hz, still having *undistinguishable frequency components*. (a): 32 point DFT with 32 point DTFT. (b): 64 point DFT with 64 point DTFT.

As we can see from Figure 45 and 46, the *close frequencies* 56Hz and 56.1Hz are *undistinguishable* from each other. As we can see from Figure 46 the *zero padding* doesn't seem to help much in this regard. Even if we *zero pad* to a *higher length* and took the DFT, we will only going to get something similar to the DTFT in Figure 45. But there is another way to be able to *distinguish* between *closer frequencies*, that is

to take a *higher number* of *samples*[26] and then take a *higher point* DFT. Instead of taking 32 or 64 *samples*, if we take a *higher sample count*[27] and then take a *higher point* DFT we might be able to get that high of a *frequency resolution*.

[26] not as same as *zero padding*.
[27] let's say 1024 samples.