CONTENTS

Lab Rubrics:

1. Properly formatted lab document (report writing skill)

2. Understanding of the concepts delivered.

S. No.	Rubric No.	Grading			Score
1	1	**Exemplary (100%):** Lab report was well written and properly formatted with good delivery of the required concepts.	**Adequate (70%):** Lab report was written with some needed improvements	**Poor (40%):** Poorly written lab content.	
2	2	**Exemplary (100%):** Clearly describes the objectives of the lab as well as the concepts learned.	**Adequate (70%):** Describes the objective and concepts related to lab with some deficiency.	**Poor (40%):** Inaccurate understanding of the concepts related to lab session.	

Laboratory Session No. 01

Objective:

To get introduced with fundamentals of Digital Signal Processing

Post Lab Exercises:

Question 1:

What do you mean by the term digital. Explain it briefly.

Answer:

In the context of signals, 'digital' is used to describe data that is defined at discrete, finite values of independent variable(s) and takes one of a finite set of values at each of these instances. Digital data is thus both sampled (discrete-time) and quantized (discrete-valued). A digital system is simply one that processes digital input signals to produce digital output signals.

The following figure shows four different types of signals.

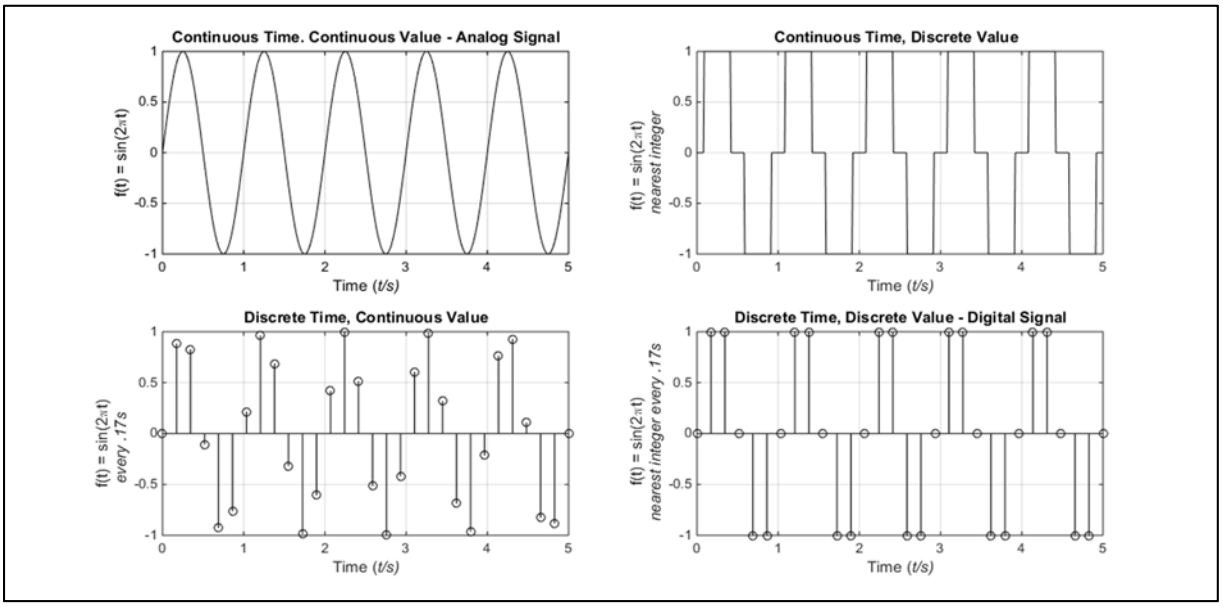


Figure 1-01: Four Different Types of Signals

Question 2:

Write some (at least three) applications of DSP related to electrical (power) engineering.

Answer:

- Instrumentation: Digital voltmeters, ammeters, and wattmeters rely on DSP principles of ADC to accurately sample, quantize, and analyze voltages, currents, and other analog quantities in power engineering.
- Noise Filters: Digital filters are used to detect and eliminate electrical noise generated by contacts, power lines, and other sources of electrical interference.
- Power System Protection: DSP algorithms are used to detect abnormal state detection, correction of errors introduced by instrument transformers, and for decision making of protective relays.
- Smart Grids: DSP algorithms are at the heart of a routing, optimization, and monitoring of electrical power generation, transmission, and distribution in smart grids.

Laboratory Session No. 02

Objective:

An Introduction to Analog to Digital Conversion (Sampling and Aliasing).

Post Lab Exercises:

Question 1:

What do you mean by the term "Sampling"? Discuss it briefly with the help of figure.

Answer:

Sampling is the process of converting a continuous time, continuous valued (CTCV) signal into one that discrete time, continuous valued (DTCV). This involves recording values or samples of an analog signal at discrete (and usually evenly-spaced) instances of time. The result is a series of numbers corresponding to the signal's value at each instance of time.

Figure 2-01 demonstrates the sampling process. 1(a) shows a CTCV signal $x_a(t) = \sin(2\pi t)$. The same signal has been sampled at a rate of 20 samples/cycle to generate the sampled DTCV signal $x_{dtcv}(n) = \sin\left(\frac{2\pi n}{20}\right)$ shown in 1(b). 1(c) demonstrates that the sampled signal data is equivalent to values of the CTCV signal recorded every 0.05s apart.

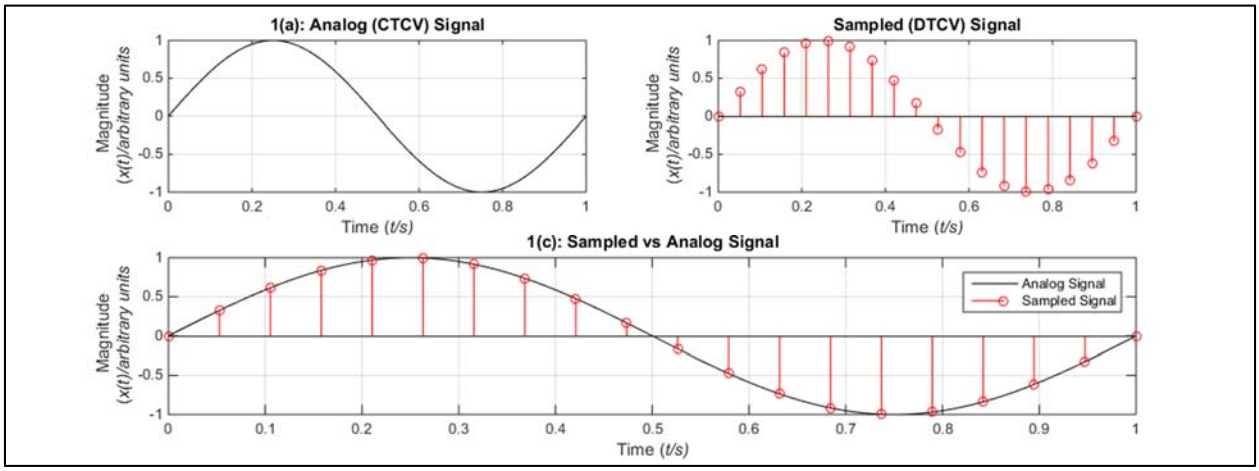


Figure 2-01: Sampling

Question 2:

What is Sampling theorem? What do you mean by the term Aliasing?

Answer:

The sampling theorem states that in order to accurately reconstruct an analog (CTCV) signal from its samples, it must be sampled at a frequency F_s at least twice as large as the highest frequency f_{max} present in the signal. Concretely,

$$F_S \geq 2f_{max}$$

This theorem is also known as the Nyqyuist Theorem, and the frequency relation is called the Nyquist criteria. Any sampling process that fails to fulfill the Nyquist criteria will result in aliasing, a phenomenon in which the samples of high frequency signals (and the resulting reconstruction) become indistinguishable from low frequency ones, effectively changing the HF signal into an alias or misidentification of a LF one.

Question 3:

Human audible frequency ranges from **20 Hz** to **20 kHz**. Human voice frequency ranges from **300 Hz** to **3.4 kHz**.

Question 4:

Record audio for 10sec and complete the following table. Show and verify the output file size through mathematical calculations.

(Hint: Check the Microphone ADC bits and use sampling frequency and the audio record time to evaluate the file size)

S No	Sampling Frequency (Hz)	File Size (MB)	Quality Comment
1	44100	1.209	Excellent audio quality.
2	22050	0.605	Minor difference in audio quality compared to sample 1. Still very good.
3	10000	0.277	Audio quality not as good as sample 2, but still easily understood. Some distortion.
4	6000	0.169	Audio sounds muffled. Can still be understood.
5	4000	0.113	Poor quality. Audio is extremely muffled.
6	2000	0.057	Audio distorted. Quality extremely poor. Speech somewhat audible.
7	1000	0.029	Poorest quality. Most distortion. Speech not audible without considerable effort.

Answer:

The file size of the audio recording can be calculated using its duration and the Simulink model's sampling frequency, number of channels, and number of bits used to encode each sample. The product of these terms is the total number of bits used to store the recording. Dividing this product by 8 (8 bits in one byte) and by 1024^2 (1MB = 1024^2 bytes). Specifically, the file size is given by

$$File\ Size = \frac{(Duration)(Sampling\ Frequency)(Bits\ per\ Sample)(Number\ of\ Chanels)}{(8)(1024)^2}$$

A comparison of the theoretical and actual file size for each recording is given below. For all recordings,

- Duration = 7 seconds
- Bits per Sample = 16
- Number of Channels = 2

S No	Sampling Frequency (Hz)	Calculations	Output file size (calculated)	Output file size (measured)
1	44100	$\frac{16 \times 44100 \times 2 \times 7}{8 \times 1024^2}$	1.18 MB	1.21 MB
2	22050	$\frac{16 \times 22050 \times 2 \times 7}{8 \times 1024^2}$	589 KB	605 KB
3	10000	$\frac{16 \times 10000 \times 2 \times 7}{8 \times 1024^2}$	267 KB	277 KB
4	6000	$\frac{16 \times 6000 \times 2 \times 7}{8 \times 1024^2}$	160 KB	169 KB
5	4000	$\frac{16 \times 4000 \times 2 \times 7}{8 \times 1024^2}$	107 KB	113 KB
6	2000	$\frac{16 \times 2000 \times 2 \times 7}{8 \times 1024^2}$	53 KB	57 KB
7	1000	$\frac{16 \times 1000 \times 2 \times 7}{8 \times 1024^2}$	27 KB	29 KB

Question 5:

If an ADC has sampling frequency =1000 Hz and receive analog signals of the following frequencies what will be the frequency of a signal which is converted back to analog by a DAC converter?

S No	Frequency (Hz)
1	100
2	750
3	1250
4	1900
5	2000
6	2500

Answer:

According Nyquist's Sampling Theorem, any signal with frequency F such that $2F \leq F_s$ can be reconstructed by a DAC without any loss of information due to aliasing. In this case, all frequencies up to and including $\frac{F_s}{2} = \frac{1000\ Hz}{2} = 500$ Hz will be reproduced by the DAC without any aliasing. Consequently, signals 2 – 6 will have different frequencies at the DAC output. With aliasing, the effective discrete-time frequency will be the first post-decrement value that fulfills the Nyquist criteria $f_d < |\frac{1}{2}|$.

S No	Signal Frequency (Hz)	Discrete-Time Frequency (cycles/sample)	Fulfills Nyquist Criteria	Effective Frequency (cycles/sample)	DAC O/P Frequency (Hz)
	F	$f_d = \frac{F}{F_s}$	$f_d \leq 0.5$	f_d'	F'
1	100	0.1	TRUE	0.1	100
2	750	0.75	FALSE	-0.25	-250
3	1250	1.25	FALSE	0.25	250
4	1900	1.9	FALSE	-0.1	-100
5	2000	2	FALSE	0	0
6	2500	2.5	FALSE	0.5	500

Laboratory Session No. 03

Objective:

An Introduction to Analog to Digital Conversion (Quantization and Coding).

Post Lab Exercises:

Question 1:

Why Quantization is needed in Digital Signal Processing?

Answer:

Quantization is the process of discretizing a signal's value at every instance of the independent variable. Without quantization, the signal's value at each instance in its domain would require infinite word length (number of bits in memory) and infinite voltage levels in a digital computer for storage and processing. As practical digital systems have both finite word length and finite memory, quantization is necessary in all practical DSP applications.

Question 2:

What is Anti-Aliasing Filter? Discuss it with some example.

Answer:

An anti-aliasing filter (AAF) is a hardware or software realization of a system that restricts a DSP system's input signal's bandwidth to specific range that satisfies the Nyquist criteria. The AAF attenuates high frequency components and prevents them from appearing at the input of a sampler. In doing so, it prevents the aliasing that would have occurred had the HF aliasing signals been sampled.

The simplest AAF is a series RC circuit as shown in 02(a). The circuit's gain is defined as $|H(j\omega)| = \frac{1}{1+(\omega RC)^2}$ and its variation as a function of the signal frequency F is shown in 3-02. As is evident from 3-02, as the gain decreases with increasing frequency, the circuit attenuates input signals with frequencies higher, and could therefore be used as an AAF for a DSP system.

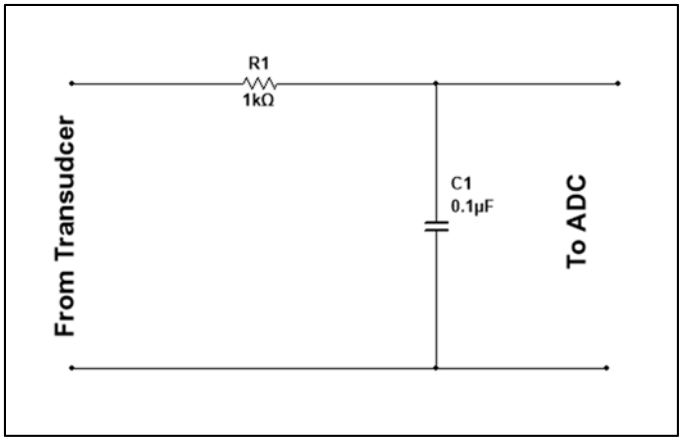


Figure 3-01: First Order Low Pass Filter

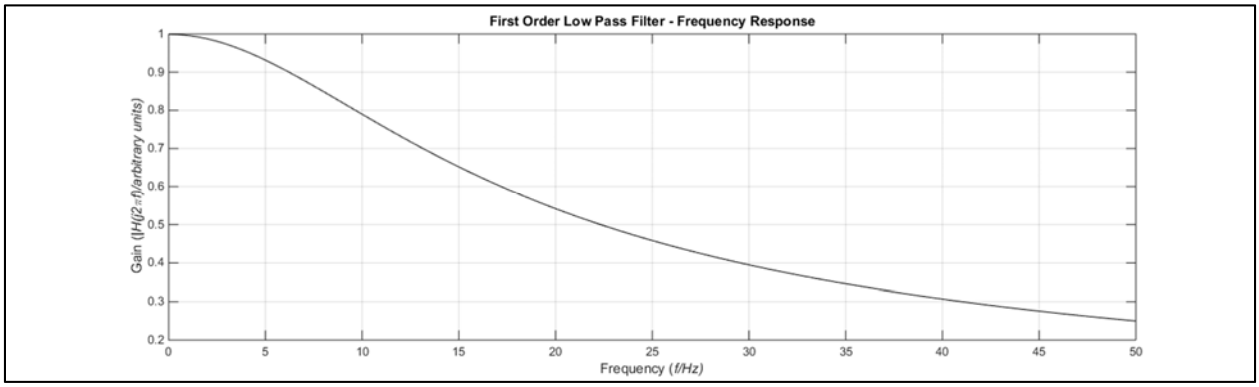


Figure 3-02: Frequency Response of First Order Low Pass Filter

Question 3:

Discuss the specifications of Arduino Uno ADC like number of bits, number of quantization levels, etc. Also, calculate the default Arduino Uno ADC resolution.

Link: https://store.arduino.cc/usa/arduino-uno-rev3

Answer:

ADC Property	Value
ADC Bits	10
Number of Quantization Levels	1024
Sampling Frequency	~10k samples/second
Total Flash Memory	32 KB
Operating Voltage Range	0V – 5V
Resolution	$\frac{5V - 0V}{2^{10}} = \frac{5V}{1024} \approx 4.88\, mV$

Question 4:

A 12-bit ADC has input values in the range of 0 – 1 V. Calculate the resolution of ADC.

Answer:

$$Resolution = \frac{ADC\ Voltage\ Range}{ADC\ Word\ Length} = \frac{1V - 0V}{2^{12}} \approx 0.244\ mV$$

Laboratory Session No. 04

(Open Ended Lab 01)

Performed with EE-164, EE-168, EE-177, EE-194, Section D

Objective:

To convert an analog (voltage) signal into digital signal using ADC (audio card) and display it on MATLAB Simulink environment.

Required Components:

1. Audio Card
2. Transformer (220V/12V)
3. Resistors (for VDR)
4. Veroboard
5. Audio jack
6. PC with MATLAB environment

Procedure:

- Using Transformer convert 220VAC from mains into 12VAC.
- Using VDR convert 12VAC to a voltage compatible to audio card (show all the calculations of resistances with their power ratings).
- Set the sampling frequency of the audio card ADC in MATLAB Simulink environment with proper justification
- Plot the acquired voltage waveform to Simulink scope.
- Mention the safe operating range of your equipment.

Introduction to ADC

Analog-to-digital conversion, abbreviated as ADC, is the process of converting an analog signal into a digital one through sampling and quantization, which discretize the value of the signal's independent and dependent variable(s) respectively. ADC may also refer to an analog-to-digital converter: a system that performs analog-to-digital conversion.

Sound Card as ADC

A sound card is a digital electronic component that is used as an audio interface in computers. Specifically, it allows a computer's operating system to control input and output of audio to and from a computer, and functions as both an ADC and DAC for audio signals.

When used as an ADC, a sound card converts analog input audio signals into digital ones through sampling and quantization. The following ADC parameters are important in determining the quality of the ADC result

- Sampling frequency
- Bit depth
- Resolution
- ADC range
- ADC gain

Programs can be used to control the sampling frequency, resolution, and ADC gain. Sampling frequency determines the total number of times a CTCV signal is sampled in a given duration (and thus determines aliasing, or lack thereof), while other factors determine quality of quantization. The ADC range and gain are important for determining when clipping occurs i.e. when the amplitudes of signals become too large to be discretized by the ADC, and are factors in determining the ADC's operating range.

Investigation 1: Sound Card Range

The range of values of an analog input signal that can be accurately digitized by the sound card is called its range or dynamic range. Concretely, the range and dynamic range can be expressed as follows:

$$Range: v_{min}(t) \leq v(t) \leq v_{max}(t)$$

$$Dynamic\ Range = v_{max}(t) - v_{min}(t)$$

Where $v_{max}(t)$ and $v_{min}(t)$ define the largest and smallest possible values of the sound card's input signal that can be faithfully converted to their digital equivalent.

The Importance of ADC Range

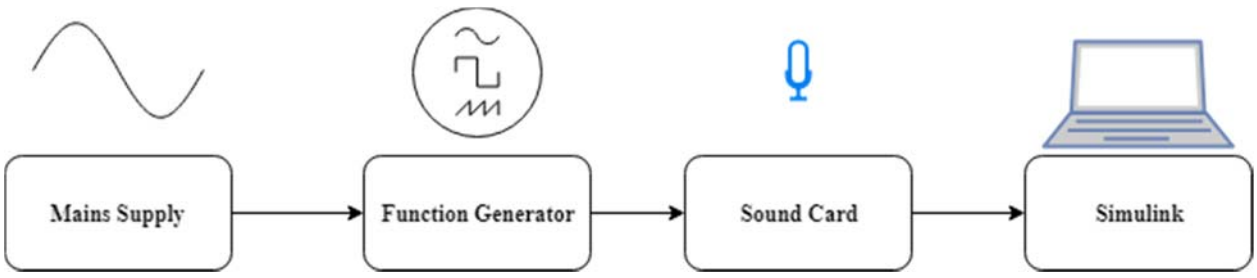
Determining the sound card's range as an ADC is important for the following reasons:

1. Clipping: The sound card will not be able to properly quantize signal values that do not lie in the interval $[v_{min}(t), v_{max}(t)]$, and any such values will distort the ADC output waveform due to clipping, affecting the quality of the digitized signal.
2. Equipment Protection: Attempting to make the sound card process signals with values outside its dynamic range can damage the sound card and connected equipment.

Apparatus

1. Function generator
2. Sound card
3. 3.5mm audio cable
4. Computer with MATLAB & Simulink

Method



The block diagram describes the experimental setup used to investigate the sound card's range when used as an ADC. The function generator supplies continuous 50 Hz sinusoidal signals of different peak-to-peak voltages to the sound card. The sound card attempts to digitize each signal, sending its data as a microphone input signal to a Simulink model on the computer which then plots the signal's variation with time on a time scope.

Maximum Value ($v_{max}(t)$)

This process is repeated for incrementally larger values of peak-to-peak voltage until clipping is first observed. The voltage is then reduced until clipping no longer occurs, and the process is repeated in a manner similar to bisection or binary search until the maximum peak-to-peak input voltage is found that does not cause clipping. This value, $v_{max}(t)$ is the maximum input signal value that can be processed by the sound card.

Minimum Value ($v_{min}(t)$)

A similar process is repeated to determine the minimum signal value that the sound card can process. The function generator is used to vary the peak-to-peak voltage for the sound card's input signal until output waveform is barely intelligible through the distortion and noise.

Observations

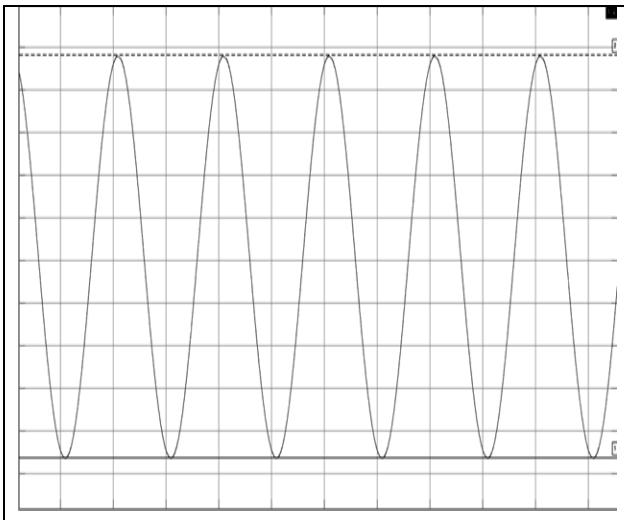
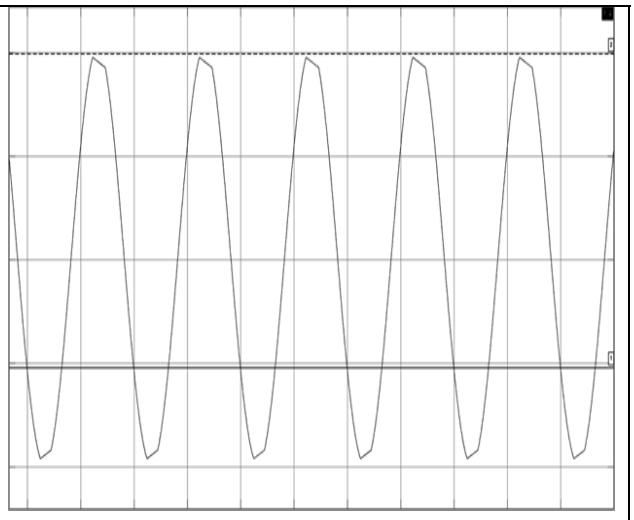
All measurements were performed for a 50 Hz sinusoid signal with ADC microphone level set to 25, sampling rate set to 44,100 samples per second, frame size set to 1024 frames, and simulation duration set to 5 seconds.

Calculating $v_{max}(t)$

Observation Number	Input Peak-to-Peak Voltage ($V_{pk\text{-}pk}$/V)	Clipping
1	1	No signs of clipping
2	2	No signs of clipping
3	4	Clipping occurs
4	3.5	No signs of clipping
5	3.6	Clipping occurs

Therefore, the upper bound of range of ADC for is 3.5 $V_{pk\text{-}pk}$.

The following figure displays digitized signals obtained by the sound card for peak-to-peak input voltages of 3.5V and 3.6V respectively. The figure clearly shows the waveform for 3.6 $V_{pk\text{-}pk}$ begins to flatten out near the peaks and troughs, and is not the smooth sinusoid of the mains supply. This 'clipping' of the waveform indicates the sound card's range has been exceeded.

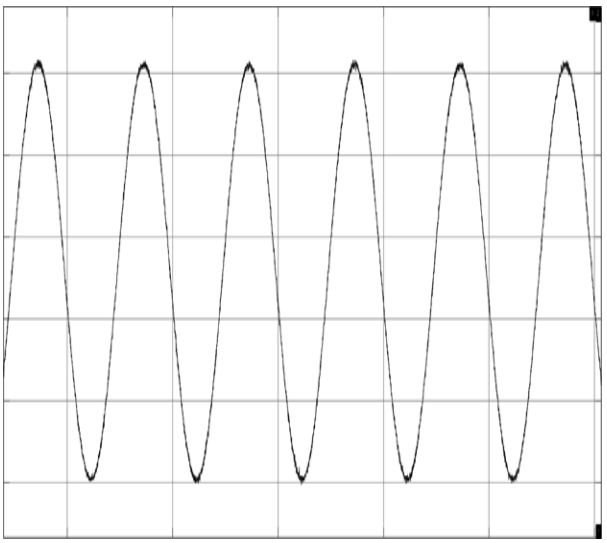
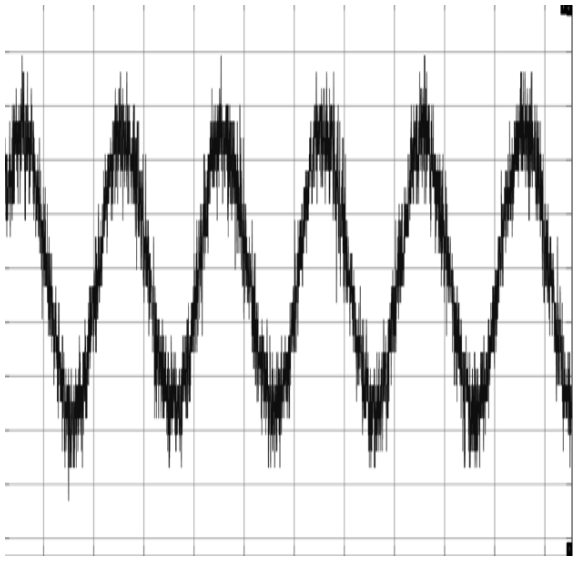
	
Output Wave in MATLAB Time scope at V_{pp} = 0.35 – good sinusoidal wave.	Output Wave in MATLAB Time scope at V_{pp} = 0.36 – clipping distorts sinusoid.

Calculating $v_{min}(t)$

Observation	Input Peak-to-Peak Voltage ($V_{pk\text{-}pk}$/mV)	Observation
1	50	Barely any noise
2	30	Very little noise, but wave still observable.
3	20	Some noise. Wave still observable.
4	1	Too much noise. Wave not observable.

Therefore, the lower bound of range of ADC for 25 sensitivity level is observed as 20 $mV_{pk\text{-}pk}$.

The following figure shows the ADC output waveforms for peak-to-peak input voltages of 20 mV and 1 mV respectively, clearly exempliying the former as the minimum possible voltage the ADC can process.

	
Output Wave form in MATLAB at 20 mV_{pp}	Output Wave in MATLAB at 1 mV_{pp}

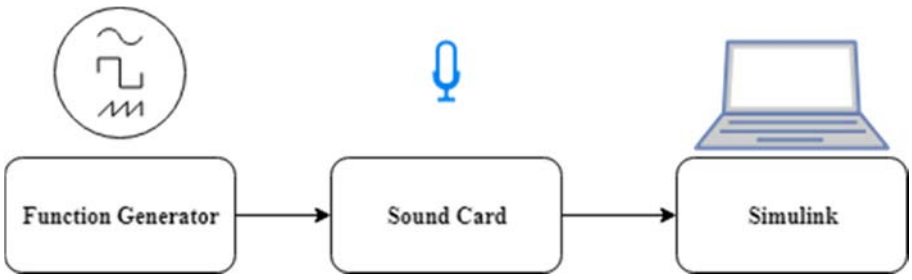
Conclusion

From our observations, $v_{max}(t) = 0.35V$ and $v_{min}(t) = 0.020V$. The operating range of the ADC is therefore $0.020\ V \leq v_{in}(t) \leq 0.35V$, and the dynamic range is $v_{max}(t) - v_{min}(t) = 0.35V - 0.020V = 0.33V$.

Investigation 2: Sound Card Gain

The gain of sound card is the ratio of its output voltage to the input voltage applied. The output voltage waveform from the soundcard is fed to the computer with the software (MATLAB). The resulting wave is a digital signal which is plotted by using time scope.

Method



A signal generator is used to provide a 50 Hz, sinusoidal AC input signal of varying peak-to-peak voltage within the sound card's operating range to the sound card via a 3.5mm audio cable. The sound card amplifies the signal by a gain factor G, digitizes it, and stores the result as microphone input data that is plotted on an oscilloscope by a Simulink model. The ratio of the sound card's input and output peak-to-peak voltages is calculated to derive its gain for microphone level set to 25%.

Microphone Levels

The sound card's gain can be controlled by the computer's operating system through the microphone level, with the latter being proportional to ADC gain. Microphone level of 25% was chosen for all investigations performed in this experiment because it provided the best tradeoff between sensitivity to input signal values and noise suppression.

Observations

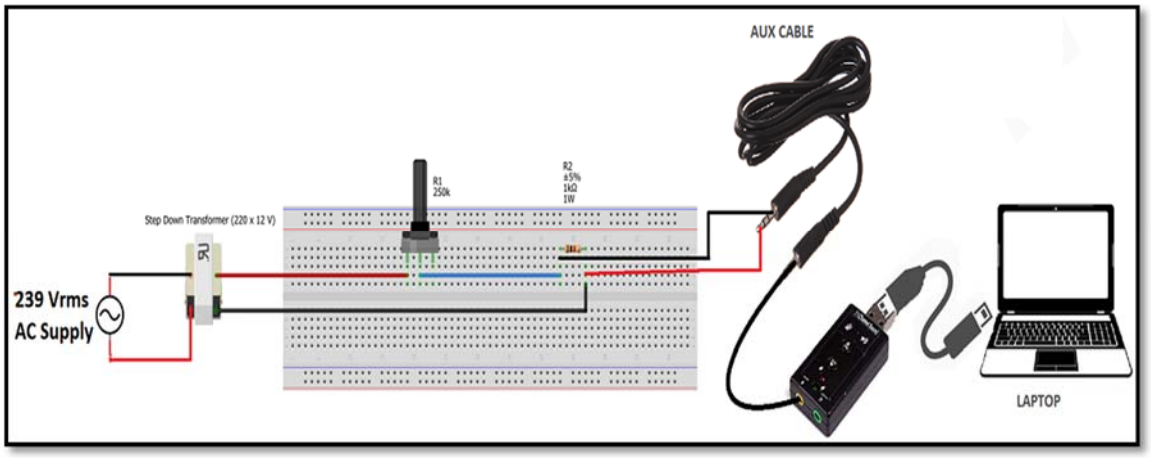
Serial No.	**Peak-Peak Input Voltage ($V_{pk\text{-}pk}in$/V)**	**Peak-Peak Output Voltage ($V_{pk\text{-}pk}out$/V)**	**Gain** $= \frac{V_{pk-pk}out}{V_{pk-pk}in}$
1	1.58	0.833	0.52
2	2.18	1.167	0.535
3	3.34	1.789	0.535

Conclusions

The sound card gain is $\frac{(0.52+0.535+0.535)}{3} = \mathbf{0.53}$. The gain remains roughly constant for all observations (with fluctuations well within the bounds of experimental error), as is expected since the input signal frequency does not change between observations and gain is a function of frequency.

Plotting Mains Supply

Circuit Diagram



Voltage Divider **(VDR)** Calculations:

In order to plot the mains supply on a scope in a Simulink model, the sound card (ADC)'s safe operating range must be taken into account. As the range was determined to be [20 $mV_{pk\text{-}pk}$, 3.5$V_{pk\text{-}pk}$], it is not safe to directly connect the 220 V_{RMS} mains supply to the ADC without decreasing it to the ADC's safe operating range.

For this purpose the mains supply is first stepped down with the help of a transformer followed by further attenuation through a voltage divider (VDR) circuit at the transformer's secondary windings. The resistors chosen for this VDR circuit are:

- Potentiometer = 250 kΩ
- Resistor = 1 kΩ (1 watt)

Both the potentiometer and the resistor were connected in series. Potentiometer was chosen to achieve a dynamic VDR output voltage. It facilitated the application of different input voltages to the sound card without repeatedly changing the resistor for calculating the gain and range.

- Output Range of VDR network = $\mathbf{0 - 15.3\ V_{RMS}}$

Calculations for **Power Rating**

$$\text{Transformer Output} = V = 15.3\ V_{rms}$$

$$\text{Max resistance} = \text{R} = 1k\Omega + 25k\Omega = 25.1k\Omega$$

$$\text{Total current in the network} = \frac{\text{V}}{\text{R}} = \frac{15.3}{25.1\text{k}\Omega} = 0.06095\ mA$$

$$\text{P}_{\text{resistor}} = \text{I}^2\text{R} = (0.06095\ \times 10^{-3})^2 * 1000 = 3.714\ \mu\text{ watt} < 1\text{ watt}$$

$$\text{P}_{\text{potentiometer}} = \text{I}^2\text{R} = (0.06095\ \times 10^{-3})^2 * 250000 = 0.928\text{ m watt}$$

Power rating of both resistances are well above the calculated power that will be dissipated by them. Hence they can operate safely in this experiment.

Gain and Resolution Calculations

At each stage the voltage is stepped down as a result the output digital waveform has small peak to peak value in order to verify the result the total gain of the system is calculated and a gain factor is connected in the Simulink model which amplifies the resultant output voltage with the same proportion as it was reduced.

Parameters:

Parameters	**V_{RMS} (Volts)**
Mains supply (V_{main})	243 V
Transformer (step down)	15.3
VDR	0.7707
Soundcard Pre-Amplification	0.7707
Soundcard (V_{rms}) Post-Amplification	237.23
Bits	16

$$Total\ Gain = \left(\frac{Transformer\ output}{Mains\ supply} \times \frac{VDR\ output}{Transformer\ Output} \times Gain\ of\ sound\ card\right)^{-1}$$

$$Total\ Gain = \left(\frac{15.3}{243} \times \frac{\frac{2.18}{2\sqrt{2}}}{15.3} \times 0.53\right)^{-1}$$

$$\boxed{Total\ Gain = 594.86}$$

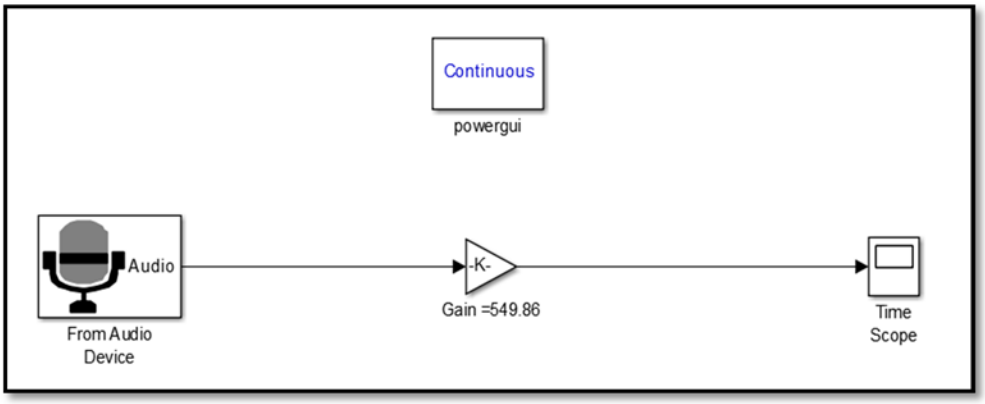
This value of total gain factor is added in our gain block in the Simulink model to counteract any scaling factor which was applied to in input mains supply in the previous stages.

$$Resolution = \Delta = \frac{X_{max} - X_{min}}{2^{bits} - 1}$$

$$Resolution = \Delta = \frac{3.55 - 0.02}{2^{16} - 1}$$

$$\boxed{Resolution = \Delta = 5.38 \times 10^{-5}\ V}$$

Simulink Model



From Audio Device Parameters:

1. Device

   To select an audio input device connected to the computer. The USB audio device option is chosen because the sound card is a connected to the computer via a USB port.

2. Number of Channels

   Channel are basically a pathway for the data to travel from its origin to destination.

   Only one of two sound card channels are used in this investigation. Dual channel recording becomes useful if its purpose is audio playback with a multi-speaker system. However, in this investigation the only requirement is to transfer the mains supply signal to the computer, which is why a single channel is enough.

3. Sampling Rate

   Sampling rate decides the number of samples the sound card records per input cycle.

   Justification:

   The signal being sampled (mains supply) is of **50 Hz**, so theoretically the minimum

sampling frequency needed to prevent aliasing and loss of information is 100 Hz. However, experimentation with the Simulink's 'From microphone' block showed that the minimum possible sampling frequency that can be set for the ADC programmatically is 1 kHz, with the maximum being 44.1 kHz. The sampling rate chosen for this investigation was

$$F_s = 44.1\ kHz$$

Primarily because it is **well above the Nyquist rate of 100 Hz** therefore **no aliasing occurs**. Given the chosen sampling duration, available memory, and processing speed, there were no memory or efficiency constraints to take into account, which is why the highest possible sampling frequency was chosen. This was done firstly to get as accurate recreation of the mains supply as possible, and secondly to test the maximum possible rate at which the sound card was capable of sampling signals.

4. Output Data Type

   Since the soundcard has a bit rate of 16 bits the output data type chosen was 'single' – which is capable of encoding each sample with up to 32 bits.

5. Frame Size

   Frame size was 1024. It doesn't affect sampling operation but only serves the purpose of bunching up the samples in frames for faster mathematical operation in real time systems.

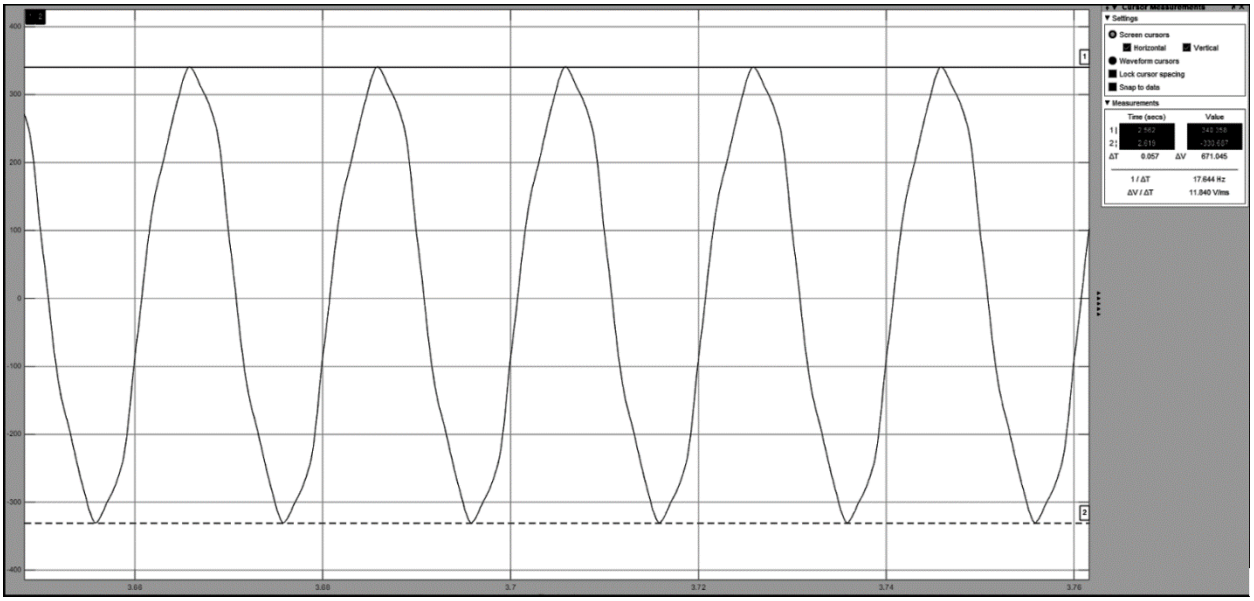
Gain Block:

This block is simply added to the model for the purpose of scaling the sound card's digitized output to the original mains value. Any desired gain factor can be used inside the block and it multiplies the signal at its input by that factor. For this experiment the gain value of the entire system was determined to be 594.86.

Time Scope:

Time scope displays and analyze signals generated during simulation and also logs their data to MATLAB's workspace. This block is used to display the digitized mains supply.

OUTPUT WAVEFORM



Calculations:

Output V_{RMS} value:

The output peak to peak value is measured using time scope in MATLAB, this value can be used to calculate V_{RMS} of the output to compare it with the main supply voltage.

$$V_{pk-pk} = 671.045\ V$$

$$V_{RMS} = \frac{V_{PP}}{2\sqrt{2}}$$

$$V_{RMS} = \frac{671.045}{2\sqrt{2}}$$

$$\boxed{V_{rms} = 237.23\ V}$$

Percentage Error:

There is slight difference between the main supplied voltages (analogue input) and the resultant output voltage (digital) due to external conditions like equipment's tolerances, temperature, noise, or slight variations in gain factor.

$V_{RMS}(measured) = 237.23\ V$

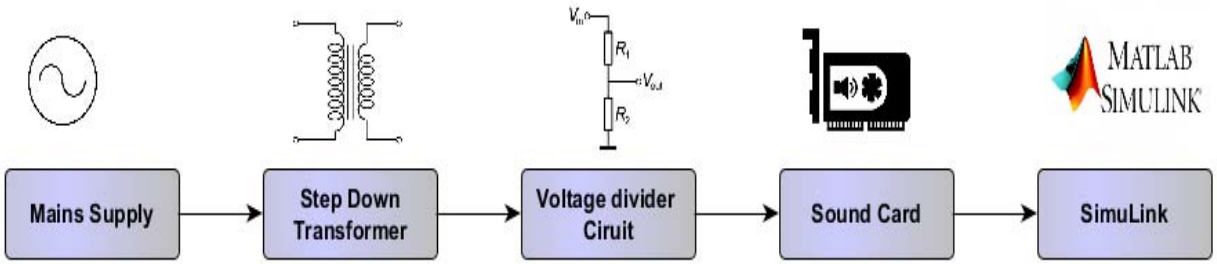
$V_{RMS}(mains) = 243\ V$

$$\% \ Error = \left(\frac{|V_{RMS}(mains) - V_{RMS}(measured)|}{V_{RMS}(mains)} \right) \times 100\%$$

$$\% \ Error = \left(\frac{|243 - 237.23|}{243} \right) \times 100\%$$

$$\% \ Error = 2.37\%$$

Results



Following the scheme above, it was proved that it is indeed possible to plot an analog signal (mains supply) as a digital one using the MATLAB Simulink environment in conjunction with an ADC (sound card). The percentage error between the RMS value of the measured mains supply and that of the digitized signal was only 2.37%, proving the digital signal is an accurate recreation of the analog one. In doing so, the role of sampling frequency, gain, and ADC operating range was also investigated and found to contribute to overall digital signal quality.

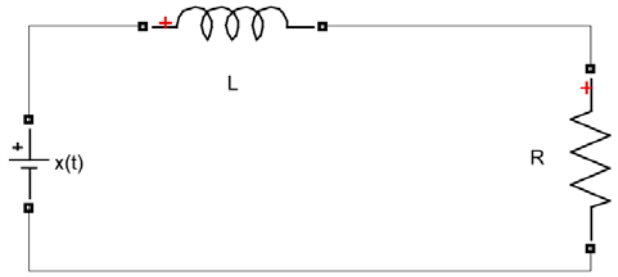
Laboratory Session No. 05

Objective:

To model a source free and sourced RL circuit in a discrete time domain and to verify it using MATLAB code and Simulink environment.

Theory:

Consider a series RL circuit excited by a voltage source $x(t)$ as shown below,



Applying KVL we get,

$$L\frac{di(t)}{dx} + Ri(t) = x(t) \quad (1)$$

For a source a source free RL circuit the above expression will become,

$$L\frac{di(t)}{dx} + Ri(t) = 0 \quad (2)$$

By using first principle, the derivate of inductor current in a discrete time domain can be evaluated as,

$$f'(a) = \lim_{x\to a}\frac{f(x)-f(a)}{x-a} \quad (3)$$

or,

$$\frac{di[(n-1)T]}{dt} = \lim_{T\to 0}\frac{i(nT)-i[(n-1)T]}{nT-(n-1)T} \quad (4)$$

From eq. (2), the derivative at the instant $(n-1)T$ can be evaluated as,

$$\frac{di[(n-1)T]}{dt} = \frac{-Ri[(n-1)T]}{L} \quad (5)$$

Equating eq. (4) and (5) we get,

$$\lim_{T\to 0} \frac{i(nT)-i[(n-1)T]}{nT-(n-1)T} = \frac{-Ri[(n-1)T]}{L} \quad (6)$$

By choosing sampling T close to zero we can evaluate the difference equation $i(n)$ as,

$$i(n) = \left(1 - T\frac{R}{L}\right) * i(n-1) \quad (7)$$

Where, nT and $(n-1)T$ are replaced by n and $(n-1)$ respectively.

Post Lab Exercise:

Question 1:

For a series RL circuit excited by a voltage source $x(t)$, derive an expression for inductor current $i(n)$, in discrete time domain. Using the derived expression, develop a MATLAB code to plot the inductor current under following system parameters:

$$R = 20\ \Omega$$

$$L = 2\ H$$

$$T_s = 0.02\ s$$

$$i_{init} = 0\ A$$

$$x(t) = discrete\ step\ input$$

$$n = any\ suitable\ range$$

Difference Equation for RL Inductor Current

Concretely, the analytical relationship between a series RL circuit's current $i(t)$, the excitation voltage $x(t)$, the resistance R, and the inductance L is as follows

$$x(t) = Ri(t) + L\frac{di(t)}{d(t)}$$

$$\Rightarrow L\frac{di(t)}{dt} = x(t) - Ri(t)$$

$$\Rightarrow \frac{di(t)}{dt} = \frac{x(t) - Ri(t)}{L}$$

Using the definition of an integral (3) and discretizing the relationship as done in (4)

$$i'(n-1) = \frac{i(n) - i(n-1)}{T} = \frac{x(t) - Ri(n-1)}{L}$$

$$i(n) - i(n-1) = \frac{T}{L}[x(t) - Ri(n-1)]$$

$$\boldsymbol{i(n) = i(n-1) + \frac{T}{L}[x(t) - Ri(n-1)]}$$

This is the difference equation which will be used to create a numerical model for the RL circuit's response.

Simulink Model: DC Powered RL Circuit

Figure 5-01 shows a Simulink model which has been used to investigate the series RL circuit's response as a discrete time signal.

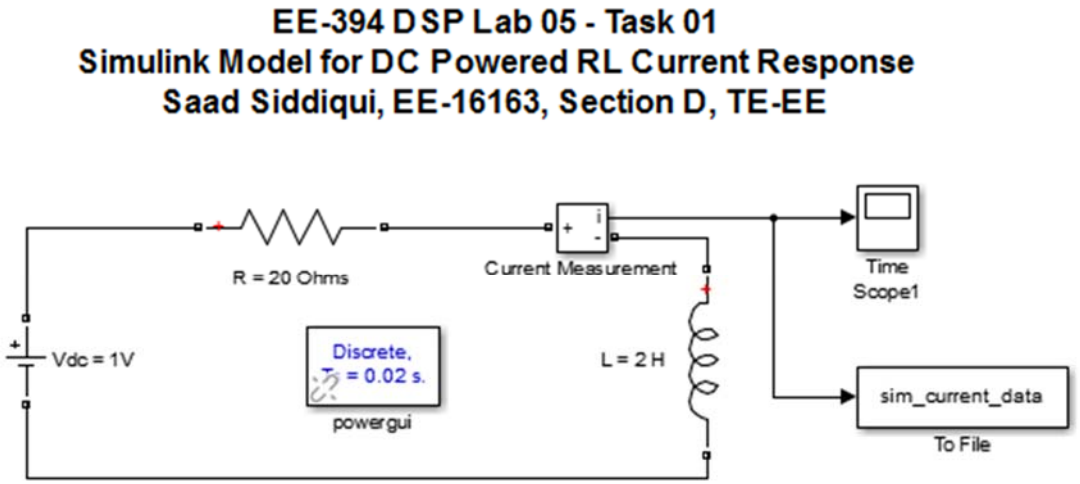


Figure 5-01: Simulink Model to Investigate DC Powered RL Circuit Current Response

The following parameters have been defined for the purposes of this investigation.

- R = 20 Ω
- L = 2 H
- V_{DC} = 1 V

- Simulation duration = 1 second
- Initial circuit current = 0 A
- $T_s = 0.02s \Rightarrow n = 50$

The `powergui` block is used to specify that a discrete (not continuous) model will be used to analyse this circuit. It also specifies the sampling period (0.02s) and the built-in differential equation solver that will be used to investigate the numerical model's current response. The current measurement data is being stored in a .mat file called `sim_current_data` while simultaneously being interpolated and plotted on a time scope. The waveform produced by the Simulink model is shown in Figure 5-02.

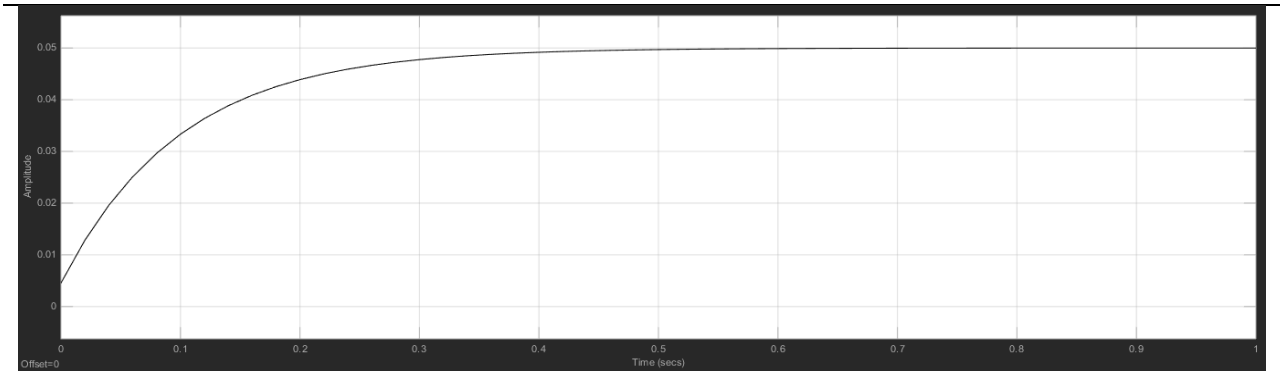


Figure 5-02: RL Circuit Current Response (Simulink)

Analysis

- The current changes its value gradually rather than instantaneously, as is expected of any current flowing through an inductor.
- The current gradually rises from its initial value of 0 A to a final steady state value of 0.05 A by t = 1s. As the transient dies out, the inductor behaves as a short circuit and has zero impedance, causing total impedance $Z = R \Rightarrow i = \frac{V_{DC}}{R} = \frac{1}{20} = 0.05\,A$.
- The data is discrete but appears continuous because adjacent samples have been joined together through MATLAB's built-in interpolation function.
- The time constant of this circuit is $\tau = \frac{L}{R} = \frac{2}{20} = 0.1$ seconds, and the Simulink model seems to verify this as the transient response dies out in approximately $5\tau = 5(0.1s) = 0.5s$, after which the current stays constant at its steady state value of 0.05 A.

This model will be used as a benchmark for the numerical model derived above.

Numerical Model: DC Powered RL Circuit

Code 01 shows a MATLAB implementation numerical model for the series RL circuit. The code defines all parameters needed for building the model, including circuit component values, sampling period, sampling duration, and total number of samples. It then calculates the value of the circuit's current for every instance in the discrete time domain, stores each sample in an array, and plots the result. The code also compares each sample value of the numerical model with its Simulink counterpart to determine the total error between each corresponding pair of sample values, as well as the root-mean-squared error between the two data sets. The latter is a way of assessing model accuracy – the smaller the RMS error, the better the model.

```
%% EE-394 Digital Signal Processing: Lab 05 - Modelling Systems
%% Task 1 - Numerical Model of DC Powered RL Circuit Current
Response
%% EE-16163, Section D, TE-EE, Fall 2018

%% Clearing all workspace data and closing any existing figures
clear all; close all;

%% Variable Declarations
t = 1;                          % sampling duration in seconds
T_s = 0.02;                     % sampling period in seconds
max_n = t / T_s;                % total number of samples
n = 1: 1 : max_n ;              % array of sample indexes
R = 20;                         % resistance in Ohms
L = 2;                          % inductance in Henries
V_s = 1;                        % DC supply voltage in Volts
i_init = 0;                     % initial circuit current in Amps
i_n = zeros( max_n, 1 );        % vector to store current values at
each sample

%% Reading Data from Simulink
% load .mat file containing Simulink data for current response
load( 'sim_current_data.mat' );

% store simulink current response - manually add 0 as first
value
% fetch only the second row - current values - until second last
col
% helps ensure both i_simulink and i_n have comparable values
and dims
i_simulink = [ 0, simulink_current( 2, 1 : max_n - 1  ) ];
```

```
%% Calculating Current According to Model
% for every sample
for var = n
    % if first sample being calculated
    if ( var == 1 )
        i_n( var ) = i_init;    % first val is always initial
current

    % if not the first sample being calculated
    else
        % using formula derived in report
        i_next = i_prev + ( T_s / L ) * ( V_s - R * i_prev );

        % store new value in the samples array
        i_n( var ) = i_next;
    end

    % remember last calculated sample value for next iteration
    i_prev = i_n( var );
end

% accounting for MATLAB's non-zero indexing
n = n - 1;

%% Plotting Model Data
figure(); stem( n, i_n ); grid on;
title( 'Current Response - Numerical Model' );
xlabel( 'Sample Index \it{(n/arbitrary units)}' );
ylabel( 'Current \it{(i(n)/A)}' );

%% Calculating RMS Error b/w Simulink and Model Data
error = i_simulink' - i_n;                          % transpose to
match dims
rmse = sqrt( mean ( error ).^2 );                   % RMS error

%% Plotting Error at Each Sample
% bar plot to display error between simulink and model at each
sample
figure(); bar( abs( error ) ); grid on;             % to fix sign
diffs
xlim( [0, 50] );                                    % default limit
includes -10
title( 'Error between Simulink and Model Currents' );
xlabel( 'Sample Index \it{(n/arbitrary units)}'); % \it{} =
italicise
ylabel( 'Error \it{(e/mA)}' );
```

```
% calculating coordinates and content for RMS error label on
graph
text_x = max ( n ) * 0.45;                % around x axis midpoint
text_y = max( abs( error ) ) * 0.95;      % around y axis top
error_label = sprintf( 'RMS Error = %.3f A', rmse );
text( text_x, text_y, error_label );      % display RMS error with
bar graph
```

Code 5-01: Numerical Model of DC RL Circuit

Figure 5-03 shows a stem plot describing the variation of the DC powered RL circuit's current as calculated by the numerical model. On inspection, the model seems to agree with Simulink data and follows the same gradual trend of increasing from 0A at index 0 to 0.05 A by sample 25, which is consistent with the Simulink time constant of 0.01s as $25(T_s) = 25(0.02) = 0.05\,s$.

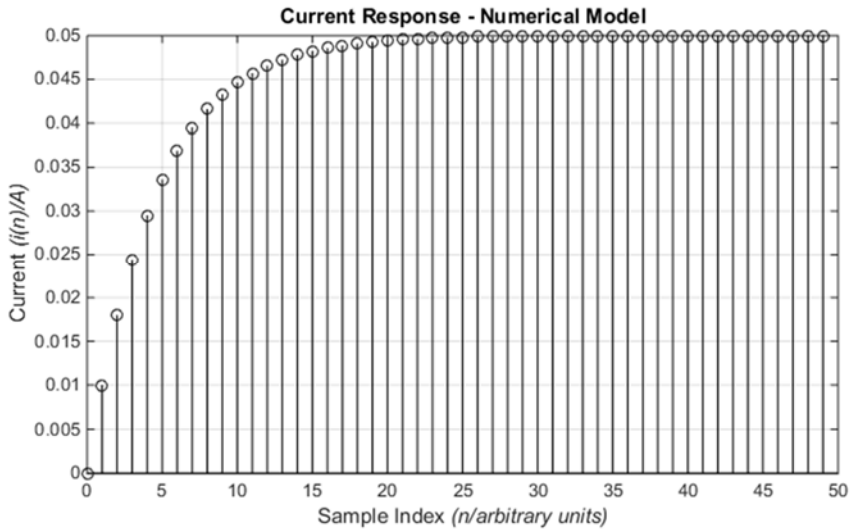


Figure 5-03: RL Circuit Current Response (Difference Equation)

Figure 5-04 shows the absolute error between the Simulink and numerical model values of the current calculated for each sample. The error decreases from 5.5 mA at sample 1 to approximately 0 mA by sample 50. The RMS error between the two sets of current response results is 0.01 A, which quantifies the similarity between the two models.

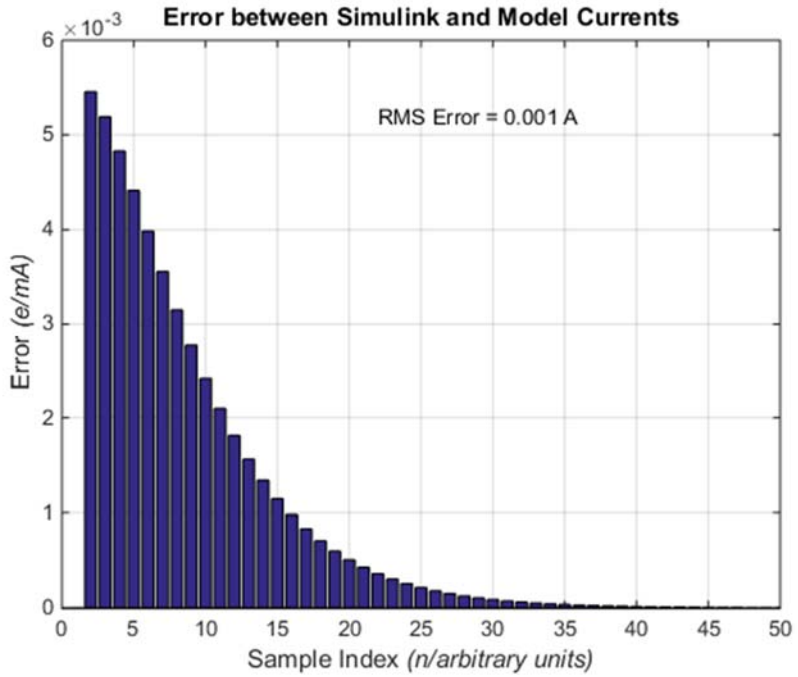


Figure 5-04: Error in Sample Values between Simulink and Difference Equation

Analysis

- The Simulink model and difference equation implemented with MATLAB predict the same current response for the RL circuit.
- The 0.01 RMS error between the two models is due to Simulink using a different differential equation-based numerical method to predict the current response.
- There is no error at sample 0 because the initial current (the first sample) has been explicitly set to be 0 A in both data sets by Code 01.
- The discrete-time RL circuit model can be used to accurately predict its current response, showing discrete time models based on difference equations are as viable options for system analysis as continuous time ones.

Laboratory Session No. 06

Objective:

Convolution for Discrete Time Signals

Post Lab Exercises:

Question 1:

We have an impulse response of a system $h(n)$= {**3**,2,1, 2,1,0, 4,0,3}, provided with an input $x(n)$={**1** 2 3 5 4 6}, find and plot $y(n)$. Also check and comment on the causality property of the system.

Question 2:

We have an impulse response of a system $h(n)$= {-3,1,2, 4,**1**,0, 4,0,3}, provided with an input x(n)={**1** 2 3 5 4 6}, find and plot y(n). Also check and comment on the causality property of the system.

Answer:

Code 01 shows a MATLAB script `myConvolutionTest.m` which is used to generate plots showing the input signal, unit impulse response, and output signal for systems defined in questions 1 and 2. The script first defines the input signal input and its origin `org_input` for use in convolution for both systems. It then defines the unit impulse response and its origin for the systems as described in questions 1 and 2. Finally, it invokes the function `myConvolution` and passes the input signal and its origin along with the unit impulse response and its origin as arguments. The resulting of running Code 01 is two figures, each showing a plot of the input signal, the unit impulse response, and the output signal for each system.

```
%% EE-394 Digital Signal Processing: Lab 06 - Convolution
%% Tasks 1 - 2: Convolution of Discrete Time Signals
%% EE-16163, Section D, TE-EE, Fall 2018

% Clearing workspace, closing all figures
clear all; close all;

% input signal for tasks 1 and 2 is the same
input = [ 1, 2, 3, 5, 4, 6 ];
org_input = 1;      % origin index
```

```
% defining impulse response + origins for tasks 1 and 2
% Task 1
impulse_1 = [ 3, 2, 1, 2, 1, 0, 4, 0, 3 ];
org_impulse_1 = 1;  % origin index

% Task 2
impulse_2 = [ -3, 1, 2, 4, 1, 0, 4, 0, 3 ];
org_impulse_2 = 5;  % origin index


% Using myConvolution function to generate subplots for each
% function arguments - input, input origin, impulse, impulse
origin
% displays input, impulse, and result of convultion in figure
myConvolution( input, org_input, impulse_1, org_impulse_1 );
myConvolution( input, org_input, impulse_2, org_impulse_2 );
```

Code 6-01: `myConvolutionTest.m` – MATLAB script to generate input, impulse, and DT convolution generated output for questions 1 and 2

Code 02 shows the implementation of the `myConvolution` function, which uses a discrete time system's input signal and origin along with the unit impulse response and its origin to perform discrete time convolution. The result is the system's output signal, which is then plotted on a figure along with the input and unit impulse response.

```
%% EE-394 Digital Signal Processing: Lab 06 - Convolution
%% Tasks 1 - 2: Convolution of Discrete Time Signals
%% EE-16163, Section D, TE-EE, Fall 2018

function myConvolution( input, org_input, impulse, org_impulse )
% accepts arrays as input and impulse response of a DT system
% along with indexes of origin in each array
% computes output using convolution and plots result

% Defining system's unit impulse response + origin, length
h = impulse;
org_h = org_impulse;      % origin sample number in h
nh = [ 0 : length( h ) - 1 ] - org_h + 1;

% Defining system's input signal + origin, length
x = input;
org_x = org_input;        % origin sample number in x
nx = [ 0 : length( x ) - 1 ] - org_x + 1;

% Output calculated with convolution
```

```
y = conv( h, x );
ny = [ nh(1) + nx(1) : nh(end) + nx(end) ] ;

% Each plot in a new figure - prevents consecutive function
calls from overwriting function plots
figure;

% Subplot 1: Plotting Impulse Response
subplot( 3, 1, 1 ); stem( nh, h );
xlabel( 'Time index n' ); ylabel( 'Amplitude' );
xlim( [ nh(1) - 1, nh(end) + 1 ] );
title( 'Impulse Response \it{h(n)}' ); grid;

% Subplot 2: Plotting Input Signal
subplot( 3, 1, 2 ); stem( nx, x );
xlabel( 'Time Index n' ); ylabel( 'Amplitude' );
xlim( [ nx(1) - 1, nx( end ) + 1 ] );
title( 'Input Signal \it{x(n)}' ); grid;

% Subplot 3: Plotting Output Signal
subplot( 3, 1, 3 ); stem( ny, y );
xlabel( 'Time Index n' ); ylabel( 'Amplitude' );
xlim( [ ny(1) - 1, ny( end ) + 1 ] );
title( 'Output Obtained by Convolution \it{y(n)}' ); grid;
```

Code 6-02: `myConvolution.m` – Function to plot system's input signal, impulse response, and output calculated through discrete time convolution

The function first uses the arguments to define the input and unit impulse signals along with their origins. It also generates an array of indexes for each signal using its length and origin. In the digital domain, signals are represented as zero-index arrays of values, but MATLAB indexes its arrays from 1 instead of 0. This is why the index array range is from 0 to `length( n ) - 1`, as the last index in a zero-indexed array is always 1 less than the length of the array. The index array is then extended by shifting all elements to account for the actual origin provided as an argument.

The built-in MATLAB `conv` function is used to generate the system output using discrete time convolution. The index array for the output signal will exists from, at most, the sum of the initial indexes to the sum of the final indexes of both input and impulse response signals.

The function then plots the input signal, the unit impulse response, and the output signal in the same figure as subplots.

Plot Generated for Question 1 and Comments on Causality

Figure 6-01 shows the plot generated by the `myConvolutionTest.m` script for the system defined in question 1. The input, unit impulse response, and output signals all start at index 0. Since the unit impulse response for this system is not defined for n < 0, the output signal y(n) does not exist until the input signal x(n) is applied to the system. Hence, this is a **causal system** – a response (output signal) can only be *caused* by first applying an excitation (input signal).

Output Signal `y(n)` from Workspace

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
$y_1(n)$	**3**	8	14	25	30	39	33	27	31	32	25	39	12	18
	↑													

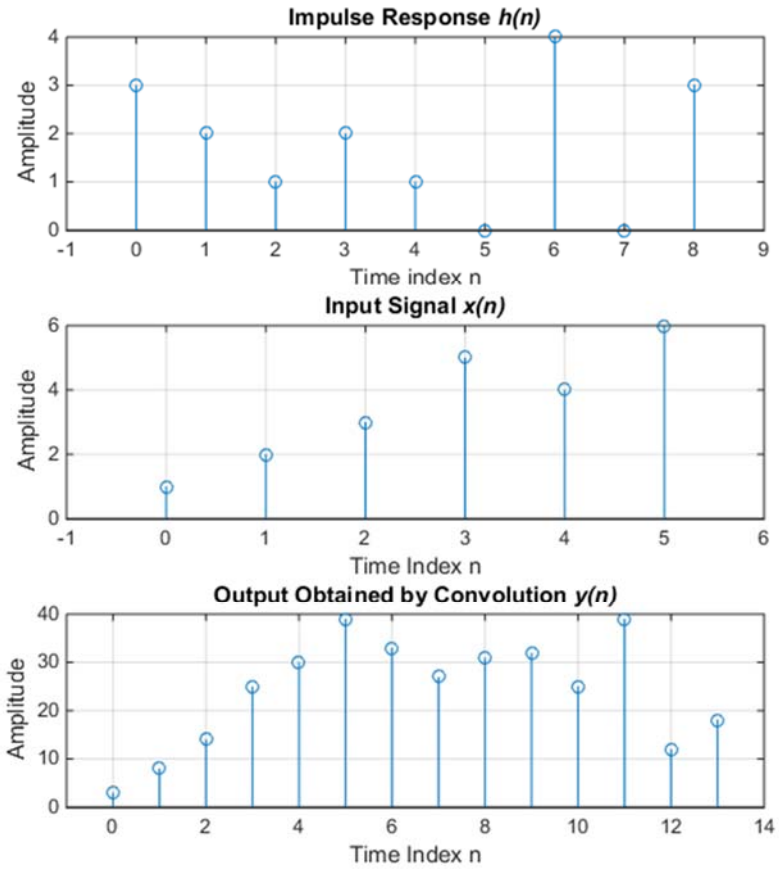


Figure 6-01: Plot Generated for System Defined in Question 1 – Causal System

Convolution Table – Question 1

Input Signal $x(n)$, Unit Impulse Response $h(k)$ and Flipped Unit Impulse Response $h(-k)$

Index	**0**	**1**	**2**	**3**	**4**	**5**
$x(n)$	**1**	2	3	5	4	6
	↑					

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$h(k)$	**3**	2	1	2	1	0	4	0	3
	↑								

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$h(-k)$	3	0	4	0	1	2	1	2	**3**
									↑

Convolution Table - Question 1

$x(n)$									**1** ↑	2	3	5	4	6						$y(n)$	
$h(-k)$	3	0	4	0	1	2	1	2	3											→0	**3**
$h(-k+1)$		3	0	4	0	1	2	1	2	3										1	8
$h(-k+2)$			3	0	4	0	1	2	1	2	3									2	14
$h(-k+3)$				3	0	4	0	1	2	1	2	3								3	25
$h(-k+4)$					3	0	4	0	1	2	1	2	3							4	30
$h(-k+5)$						3	0	4	0	1	2	1	2	3						5	39
$h(-k+6)$							3	0	4	0	1	2	1	2	3					6	33
$h(-k+7)$								3	0	4	0	1	2	1	2	3				7	27
$h(-k+8)$									3	0	4	0	1	2	1	2	3			8	31
$h(-k+9)$										3	0	4	0	1	2	1	2	3		9	32
$h(-k+10)$											3	0	4	0	1	2	1	2	3	10	25
$h(-k+11)$												3	0	4	0	1	2	1	2	11	39
$h(-k+12)$													3	0	4	0	1	2	1	12	12
$h(-k+13)$														3	0	4	0	1	2	13	18

Output calculated with convolution table is the same as that calculated through program.

Plot Generated for Question 2 and Comments on Causality

Figure 6-02 shows the plot generated by `myConvolutionTest.m` for the system defined in question 2. This is a **non-causal system**, as its unit impulse response h(n) is defined for indexes $n < 0$, which is why an output y(n) can be generated prior to application of an input signal x(n). This is evident from the figure: even though the input signal exists only for indexes ≥ 0, the impulse response and the output signal exists since index n = -4. Clearly, an input signal is not required in order to cause an output from the system, and the system is non-causal.

Output Signal `y(n)` from Workspace

Index	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$y_2(n)$	-3	-5	-5	4	**8**	10	41	41	43	32	25	39	12	18
					↑									

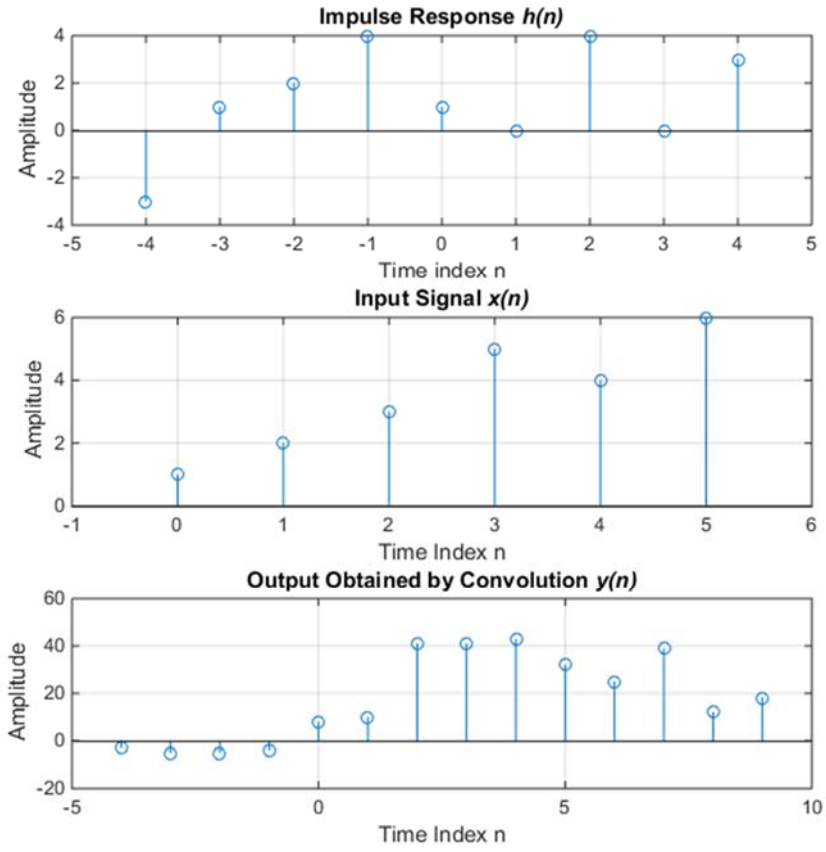


Figure 6-02: Plot Generated for System Defined in Question 2 – Non-Causal System

Convolution Table – Question 2

Input Signal $x(n)$, Unit Impulse Response $h(k)$ and Flipped Unit Impulse Response $h(-k)$

Index	**0**	**1**	**2**	**3**	**4**	**5**
$x(n)$	**1**	2	3	5	4	6
	↑					

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$h(k)$	-3	1	2	4	**1**	0	4	0	3
					↑				

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$h(-k)$	3	0	4	0	**1**	4	2	1	-3
					↑				

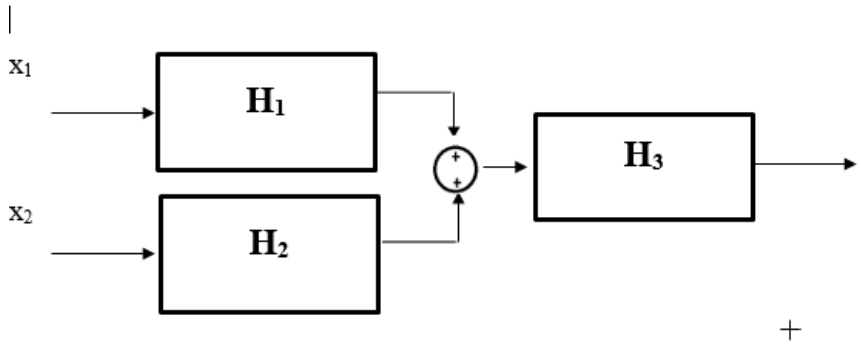
Convolution Table - Question 2

$x(n)$									**1** ↑	2	3	5	4	6								y(n)	
$h(-k-4)$	3	0	4	0	1	4	2	1	-3													-4	-3
$h(-k-3)$		3	0	4	0	1	4	2	1	-3												-3	-5
$h(-k-2)$			3	0	4	0	1	4	2	1	-3											-2	-5
$h(-k-1)$				3	0	4	0	1	4	2	1	-3										-1	-4
$h(-k)$					3	0	4	0	1	4	2	1	-3									→0	**8**
$h(-k+1)$						3	0	4	0	1	4	2	1	-3								1	10
$h(-k+2)$							3	0	4	0	1	4	2	1	-3							2	41
$h(-k+3)$								3	0	4	0	1	4	2	1	-3						3	41
$h(-k+4)$									3	0	4	0	1	4	2	1	-3					4	43
$h(-k+5)$										3	0	4	0	1	4	2	1	-3				5	32
$h(-k+6)$											3	0	4	0	1	4	2	1	-3			6	25
$h(-k+7)$												3	0	4	0	1	4	2	1	-3		7	39
$h(-k+8)$													3	0	4	0	1	2	1	2	3	8	12
$h(-k+9)$														3	0	4	0	1	2	1	2	9	18

Output obtained through convolution table is the same as that obtained through program.

Question 3:

If impulse response of the system $h_1(n)$= {**1** 1 1}, $h_2(n)$= {**0**,0, 1} and $h_3(n)$= {1,**2**,0, 1} , Input $x_1(n)$={**1** 2 3 5 4 6} and $x_2(n)$={**1** 2 3 5 4 6}, find and plot $y(n)$. Also check and comment on the causality property of the system.



Answer:

The figure shows a cascaded system – a system in which the outputs of one or more systems are used as the input(s) of another system. Code 6-03 shows a MATLAB script which uses a modified version of the `myConvolution` (Code 6-02) to compute the output of the system shown in the previous figure.

```
%% EE-394 Digital Signal Processing: Lab 06 - Convolution
%% Tasks 3: Convolution of Cascaded Systems
%% EE-16163, Section D, TE-EE, Fall 2018

% clearing workspace, closing figures
clear all; close all;

% Defining input signals and origins
x_1 = [ 1 2 3 5 4 6 ]; org_x_1 = 1;
x_2 = [ 1 2 3 5 4 6 ]; org_x_2 = 1;

% Defining impulse responses and origins
h_1 = [ 1 1 1 ];    org_h_1 = 1;
h_2 = [ 0 0 1 ];    org_h_2 = 1;
h_3 = [ 1 2 0 1 ];  org_h_3 = 2;

% Convolution of x_1 and x_2 using modified form of
% myConvolution that will return the output signal as array
output_1 = myConvolution( x_1, org_x_1, h_1, org_h_1 );
output_2 = myConvolution( x_2, org_x_2, h_2, org_h_2 );

% Creating third input signal as sum of previous outputs
```

```
x_3 = output_1 + output_2;

% Systems represented by h1 and h2 are both causal -
% org_h_1 and org_h_2 are both 1 (0 accounting for MATLAB
indexing)
% so sum of output 1 and 2 will start from 1 (0)
org_x_3 = org_x_1;

% Convolution of sum of previous outputs - cascaded
output_3 = myConvolution( x_3, org_x_3, h_3, org_h_3 );
```

Code 6-03: `cascadedConvolution.m` – MATLAB script which uses a modified version of `myConvolution` to compute output for cascaded linear system.

In addition to performing the same tasks as its counterpart in Code 02, the modified version of `myConvolution` returns the result of the convolution as an array of numbers. Code 03 makes use of this fact by computing and storing the results of convolution for subsystems H1 and H2 in `output_1` and `output_2` respectively. It then uses the algebraic sum of these output signals as the input signal for the third subsystem H3 and computes the system output by convolving this signal with H3's unit impulse response signal.

Subsystems H1 and H2 are clearly causal systems – their graphs shown in Figures 3a and 3b clearly show that their unit impulse response is not defined for $n < 0$, which means their responses cannot occur before the application of their respective input signals. Both the input signal and the unit impulse response for each system begin at index 1 (represented as 0 in the MATLAB figure), which is why the output signal for each system also begins at the same index.

Signal Data from Workspace

Index	0	1	2	3	4	5	6	7
output_1	**1**	3	6	10	12	15	10	6
	↑							
output_2	**0**	0	1	2	3	5	4	6
	↑							
x_3	**1**	3	7	12	15	20	14	12
	↑							

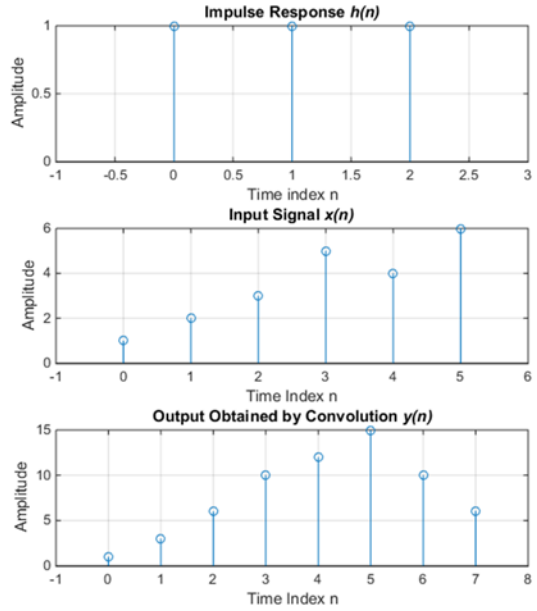


Figure 6-03a: Plots for Subsystem H1

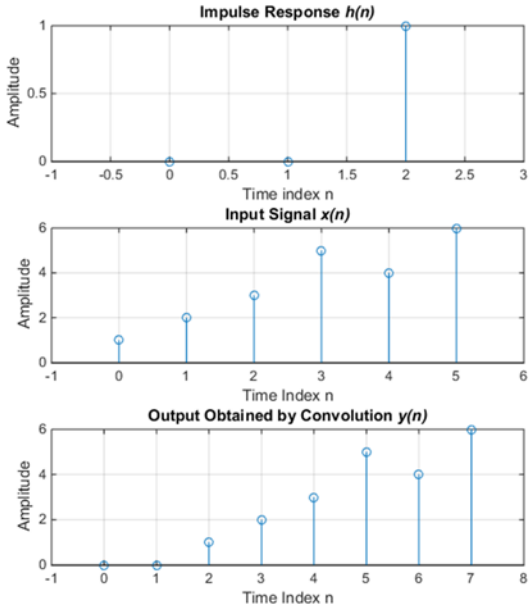


Figure 6-03b: Plots for Subsystem H2

The input to system H3 is the sum of outputs from systems H1 and H2, which means its reference or origin will also be index 0. Code 6-03 explicitly sets the third input signal `x_3` to the sum of signals `output_1` and `output_2` (there is no need for origin realignment before addition as both signals already have the same the reference point). It also sets the origin of this signal to index 1. Finally it convolves composite signal `x_3` with H3's unit impulse response to produce the final output of the system. The results of this process are shown in Figure 04.

Output `y(n)` from Workspace

Index	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
`output_3`	1	**5**	13	27	42	57	66	55	44	14	12
		↑									

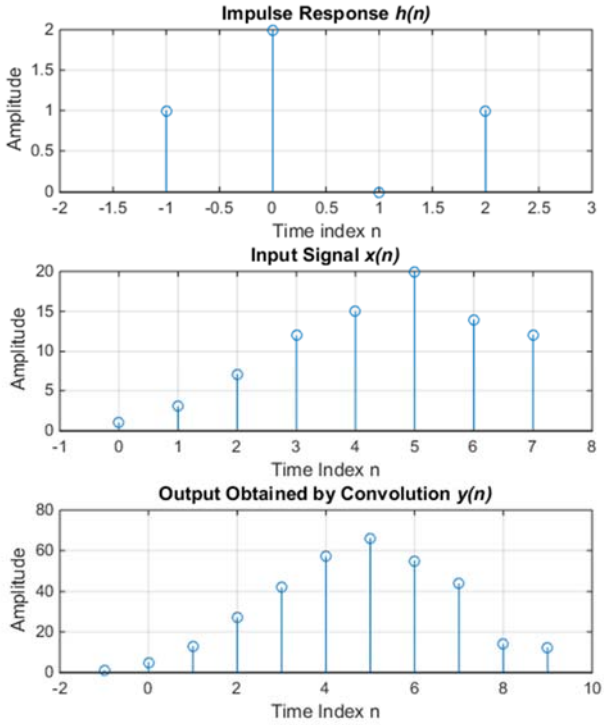


Figure 6-04: Plots Generated for Subsystem H3

Comments on Causality

- The system's inputs `x_1` and `x_2` begin at the instant represented by sample index 0, but the overall system output begins at the instant represented by index -1. This shows the system is **non-causal**.
- Even though subsystems H1 and H2 are **causal**, the presence of a **non-causal subsystem H3 makes the entire system non-causal**.

Convolution Table – Question 3: System H1 and H2

Input Signal $x_1(n)$, Unit Impulse Response $h_1(k)$ and Flipped Unit Impulse Response $h_1(-k)$

Index	**0**	**1**	**2**	**3**	**4**	**5**
$x_1(n)$	**1**	2	3	5	4	6
	↑					

Index	**0**	**1**	**2**
$h_1(k)$	1	1	1
	↑		

Index	**0**	**1**	**2**
$h_1(-k)$	1	1	1
			↑

Input Signal $x_2(n)$, Unit Impulse Response $h_2(k)$ and Flipped Unit Impulse Response $h_2(-k)$

Index	**0**	**1**	**2**	**3**	**4**	**5**
$x_2(n)$	**1**	2	3	5	4	6
	↑					

Index	**0**	**1**	**2**
$h_2(k)$	0	0	1
	↑		

Index	**0**	**1**	**2**
$h_2(-k)$	1	0	0
			↑

Convolution Table - Question 3, System H1

$x_1(n)$			**1**	2	3	5	4	6			$y_1(n)$	
			↑									
$h_1(-k)$	1	1	1								→0	**1**
$h_1(-k+1)$		1	1	1							1	3
$h_1(-k+2)$			1	1	1						2	6
$h_1(-k+3)$				1	1	1					3	10
$h_1(-k+4)$					1	1	1				4	12
$h_1(-k+5)$						1	1	1			5	15
$h_1(-k+6)$							1	1	1		6	10
$h_1(-k+7)$								1	1	1	7	6

Convolution Table - Question 3, System H2

$x_2(n)$			1	2	3	5	4	6			$y_2(n)$	
			↑									
$h_2(-k)$	1	0	0								→0	**0**
$h_2(-k+1)$		1	0	0							1	0
$h_2(-k+2)$			1	0	0						2	1
$h_2(-k+3)$				1	0	0					3	2
$h_2(-k+4)$					1	0	0				4	3
$h_2(-k+5)$						1	0	0			5	4
$h_2(-k+6)$							1	0	0		6	5
$h_2(-k+7)$								1	0	0	7	6

Convolution Table – Question 3: System H3

Input Signal $x_3(n)$ $(y_1(n) + y_2(n))$, Unit Impulse Response $h_3(k)$ and Flipped Unit Impulse Response $h_3(-k)$.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$x_3(n)$	**1**	3	7	12	15	20	14	12
	↑							

Index	**-1**	**0**	**1**	**2**
$h_3(k)$	1	**2**	0	1
		↑		

Index	**-2**	**-1**	**0**	**1**
$h_3(-k)$	1	0	**2**	1
			↑	

Convolution Table - Question 3, System 3

$x_3(n)$				**1**	3	7	12	15	20	14	12			$y_3(n)$	
				↑											
$h_3(-k-1)$	1	0	2	1										-1	1
$h_3(-k)$		1	0	2	1									→0	**5**
$h_3(-k+1)$			1	0	2	1								1	13
$h_3(-k+2)$				1	0	2	1							2	27
$h_3(-k+3)$					1	0	2	1						3	42
$h_3(-k+4)$						1	0	2	1					4	57
$h_3(-k+5)$							1	0	2	1				5	66
$h_3(-k+6)$								1	0	2	1			6	55
$h_3(-k+7)$									1	0	2	1		7	44
$h_3(-k+8)$										1	0	2	1	8	14
$h_3(-k+9)$											1	0	2	9	12

All outputs calculated from the convolution tables match those calculated by the program.

Laboratory Session No. 07

Objective:

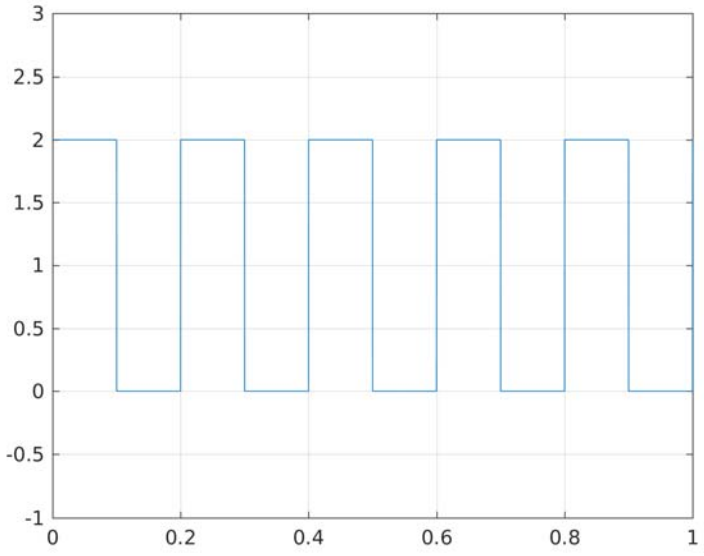
To generate a square wave in time domain of specific time period and pulse width and to apply CTFS equation to compute the spectral coefficients using MATLAB.

Post Lab Exercises:

Question 1:

Generate a square wave of magnitude 2 units with duty cycle of 50 % and have a frequency of 5 Hz using MATLAB *square()* function and then plot the frequency spectrum (magnitude vs frequency plot) of the resulting wave using MATLAB script (Evaluate five spectral coefficients). Also, verify your MATLAB results using hand calculations.

Note: Use MATLAB help to get the syntax of the *square()* function.



Answer

Code 01 is a MATLAB program which generates data for a square wave signal with amplitude of 2 units, duty cycle of 50%, and frequency of 5 Hz (same as figure on previous page). It plots the square wave signal as a function of time, then extracts and plots the data of a single period of the signal. This data is then used to perform spectral analysis of the square wave signal using a continuous time exponential Fourier series. The magnitudes of the first five complex exponential Fourier components are derived and plotted.

```
%% EE-394 Digital Signal Processing: Lab 06 - Convolution
%% Task 1: Spectral Analysis of Square Wave
%% EE-16163, Section D, TE-EE, Fall 2018
%% PART 0 - Closing all figures, clearing workspace, clearing
log command
clear all; close all; clc;

%% PART 1A - Defining parameters for square wave
T_s    = 0.001;         % sampling time in seconds
T_p    = 0.2;           % square wave period in seconds
duty   = 0.5;           % duty cycle as fraction - 50 %
f      = 1 / T_p;       % frequency of signal
ampl   = 2;             % amplitude of square wave
offset = ampl;          % to make a purely positive waveform
F0     = f;             % fundamental frequency for spectral
analysis

% Defining time domain
t_start = 0; t_end = 1;
t = t_start : T_s : t_end;

% Defining square wave
sq_wave = ampl * ( 1 + square( 2 * pi * f .* t, ...
    duty * 100 )) / 2;

%% PART 1B - Plotting the square wave
figure; plot( t, sq_wave ); grid;
ylim( [ -1, 3] ); xlabel( 'Time (\it{t/seconds})');
ylabel( 'Signal(\it{x(t)})' );title( 'Task 1 - Square Wave' );

%% PART 2A - Extracting information for one period of signal
period_start = find( round ( t * 1000 ) / 1000 == 0 ) ;
period_end = find( round ( t * 1000 ) / 1000 == T_p );
period_indices = period_start : period_end;
sq_wave_period = sq_wave( period_indices );
single_period = t( period_indices );
```

```
%% PART 2B- Plotting extracted signal
figure; plot( single_period, sq_wave_period );
xlabel( 'Time (\it{t/seconds})' );
ylabel( 'Signal(\it{x(t)})' );
title( 'Task 2a- Single Period of Square Wave' );
xlim( [ -0.01, 0.21 ] ); ylim( [-0.1, 2.1]); grid;

%% PART 3A - Computing the Continuous Time Fourier Series
Coefficients
fs_time = (-T_p / 2): T_s : ( T_p / 2 );
num_of_coeffs = 5;
coeff( 1 ) = ampl / 2;

for k = 2 : num_of_coeffs
    % Computing for complex fourier coefficient
    B = ampl * exp( -1i * 2 * pi * ( k - 1 ) * F0 .* fs_time );

    % Finding specific coefficient through discretised integral
    % Cn = 1/T int( x(t) * e^(-jnwt) ) between -T0/2, T0/2
    % wo = 2piF0t, [-T0/2, T0/2] is just fs_time
    % Because discrete, fs_time values separated by T_s
    % And integration replaced by summation
    % B is still the complex Fourier coefficient
    % length ( sq_wave_period ) = [-T0/2, T0/2]'s number of
elements
    coeff( k ) = ( sum ( sq_wave_period .* B ) ) / ...
        ( length( sq_wave_period ) );
end

%% PART 3B - Plotting Amplitude Spectrum
k =  [ 0: num_of_coeffs - 1 ];  % Frequency in Hz
coeff_mags = abs( coeff );      % Magnitudes of each component

% Creating text labels for each stem plot to show magnitude
Cn_labels = sprintfc('%.4f', coeff_mags );

% generating plot
figure; stem( k, coeff_mags ); grid; xlabel( 'K^{th} Frequency -
Hz' ); ylabel ( 'Component Amplitude - |C_{n}|' );
xlim( [ -1, num_of_coeffs ] );
title ( 'Task 2b - CT Fourier Spectrum for Square Wave' ) ;
text( k, coeff_mags, Cn_labels, 'HorizontalAlignment',
'Left',...
    'VerticalAlignment', 'Bottom' );
```

Code 7-01: Generating square wave, extracting data for period, and performing spectral analysis

Part 1 – Generating the Square Wave

```
%% PART 1A - Defining parameters for square wave
T_s     = 0.001;          % sampling time in seconds
T_p     = 0.2;            % square wave period in seconds
duty    = 0.5;            % duty cycle as fraction - 50 %
f       = 1 / T_p;        % frequency of signal
ampl    = 2;              % amplitude of square wave
F0      = f;              % fundamental frequency for spectral
analysis

% Defining time domain
t_start = 0; t_end = 1;
t = t_start : T_s : t_end;

% Defining square wave
sq_wave = ampl * ( 1 + square( 2 * pi * f .* t, ...
    duty * 100 )) / 2;

%% PART 1B - Plotting the square wave
figure; plot( t, sq_wave ); grid;
ylim( [ -1, 3] ); xlabel( 'Time (\it{t/seconds})');
ylabel( 'Signal(\it{x(t)})' );title( 'Task 1 - Square Wave' );
```

Code 7-01, Part 1 - Generating the Square Wave

Code 7-01, Part 1 is a snippet showing the MATLAB code used to generate and plot the square wave. All MATLAB arrays are inherently discrete, but can become exceedingly better approximations of continuous ranges if the separation between consecutive values becomes smaller. To generate an approximation of a continuous square wave, the sampling time `T_s` is set to 0.001. Square wave amplitude, frequency, and duty cycle are initialized to their required values of 2 units, 5 Hz, and 0.5 (50 %) respectively. The time domain consists of 1000 values between `t_start` (set to 0) and `t_end` (set to 1) in seconds. This is a good approximation of a continuous time domain, even though it is discrete.

Figure 7-01 shows the square wave generated by this code.

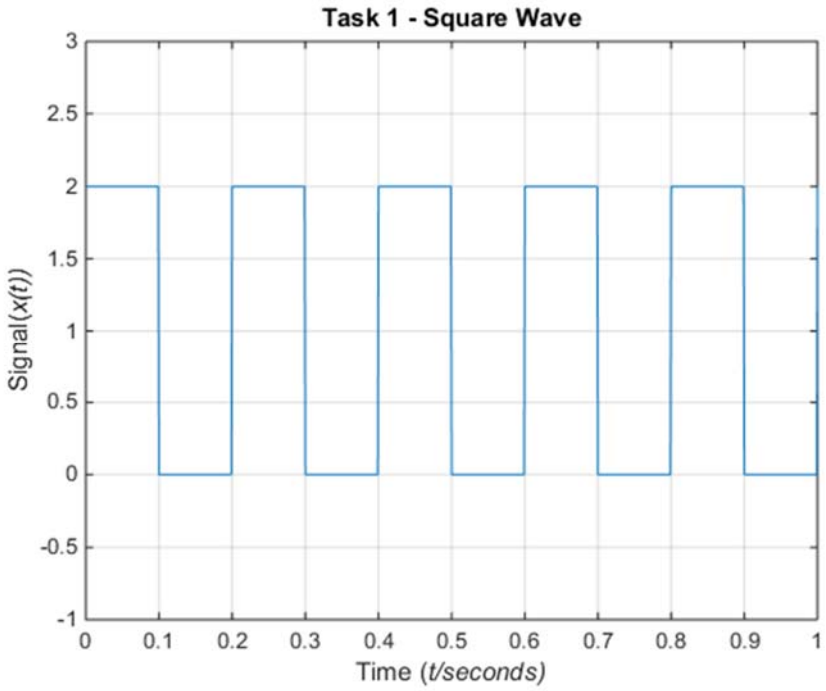


Figure 7-01 - Generating the Square Wave

The square wave itself is plotted with the built-in MATLAB `square` function, which takes two arguments

- Time Domain – expressed in terms of the duration for which to plot the signal as well as its angular frequency. `square( t )` will plot a square wave of amplitude 1 and period 2π. So `square( 2* π * f .* t)` will plot a square wave of amplitude 1 and period `T_p` for all values defined in the time domain t.
- Duty Cycle – the proportion of the time period for which the square wave is 'high' or has maximum value, expressed as a percentage (hence `duty` times 100).

The square wave is then offset (shifted upwards) and scaled by its amplitude `ampl` to ensure the wave is positive for all values of t and has the correct amplitude. The result is then divided by 2 to ensure the wave is symmetrical about the offset of 1 unit.

Part 2 – Extracting Information for One Period of the Square Wave

Part 2 examines is the portion of Code 7-01 used to extract information about a single period of the square wave. This is done because only a single period of a periodic signal is required for spectral analysis.

```
%% PART 2A - Extracting information for one period of signal
period_start = find( round ( t * 1000 ) / 1000 == 0 ) ;
period_end = find( round ( t * 1000 ) / 1000 == T_p );
period_indices = period_start : period_end;
sq_wave_period = sq_wave( period_indices );
single_period = t( period_indices );

%% PART 2B- Plotting extracted signal
figure; plot( single_period, sq_wave_period );
xlabel( 'Time (\it{t/seconds})' );
ylabel( 'Signal(\it{x(t)})' );
title( 'Task 2a- Single Period of Square Wave' );
xlim( [ -0.01, 0.21 ] ); ylim( [-0.1, 2.1]); grid;
```

Code 7-01, Part 2 – Extracting and Plotting Data for A Single Period of Square Wave

`period_start` and `period_end` define the index in the time domain array at which the period starts and ends respectively. The built-in MATLAB find function is used to find the index at which the time value, rounded to the nearest 1000th of a second, is 0 (for `t_start`) and `T_p` (for `t_end`). These extreme values are then used to generate an array of indices which will be used to access the time domain vector in order to find both time domain and square wave values corresponding to a single period, which are stored in `sq_wave_period` and `single_period` respectively. The data is then plotted on a graph, which clearly shows the square wave varies from 2 units to -2 units in a duration of approximately 0.2s, with the transition occurring at approximately 0.1s (as is expected with a duty cycle of 50 %).

The extracted signal is displayed in Figure 7-02.

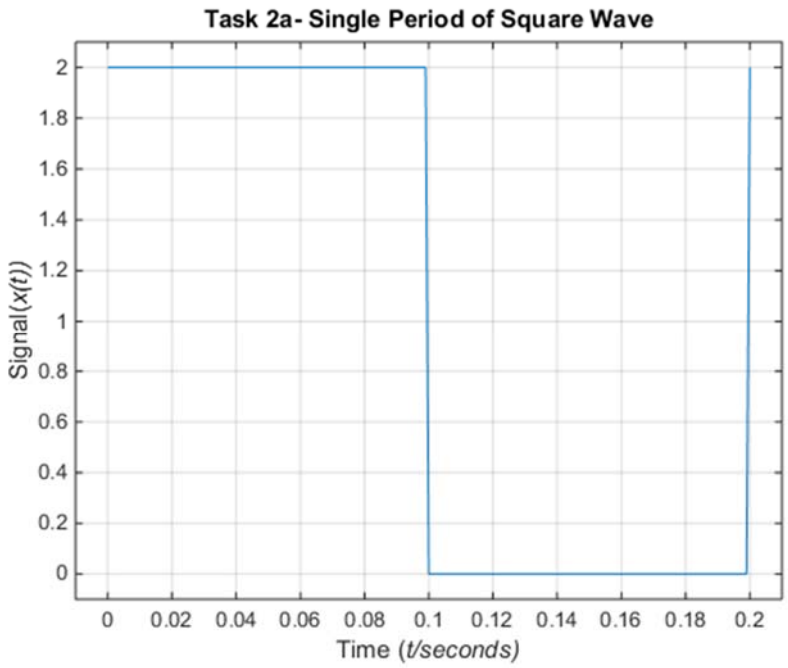


Figure 7-02 – Extracting and Plotting Data for A Single Period of Square Wave

Part 3 – Generating Fourier Series

To perform spectral analysis of the square wave, it must be decomposed into an infinite sum weighted, harmonically related sinusoids called the Fourier series. One form the Fourier series represents this infinite sum in terms of complex exponentials, with the magnitude of each exponential representing the strength of its corresponding frequency component in the signal. The continuous time complex Fourier exponential series is defined by

$$x(t) = \sum_{n=-\infty}^{\infty} B_n e^{jn\omega_0 t} \dots (1)$$

Where B_n is the n[th] Fourier coefficient, and is equal to

$$B_n = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jn\omega_0 t}\, dt \dots (2)$$

Since MATLAB, like all programming languages, can only handle discrete operations, the computation of the Fourier coefficients and the generation of the Fourier series has to be discretized. This is done in Code 01, Part 3, which first generates a discrete time domain over

which to perform the 'integration' for each coefficient. The Fourier Series time domain `fs_time` is simply an array spanning the duration of one period, with values separated by sampling time `T_s`.

```
%% PART 3A - Computing the Continuous Time Fourier Series
Coefficients
fs_time = (-T_p / 2): T_s : ( T_p / 2 );
num_of_coeffs = 5;
coeff( 1 ) = ampl / 2;

for k = 2 : num_of_coeffs
    % Computing for complex fourier coefficient
    B = exp( -1i * 2 * pi * ( k - 1 ) * F0 .* fs_time );

    % Finding specific coefficient through discretised integral
    % Cn = 1/T int( x(t) * e^(-jnwt) ) between -T0/2, T0/2
    % wo = 2piF0t, [-T0/2, T0/2] is just fs_time
    % Because discrete, fs_time values separated by T_s
    % And integration replaced by summation
    % B is still the complex Fourier coefficient
    % length ( sq_wave_period ) = [-T0/2, T0/2]'s number of
elements
    coeff( k ) = ( sum ( sq_wave_period .* B ) ) / ...
        ( length( sq_wave_period ) );
end

%% PART 3B - Plotting Amplitude Spectrum
k =  [ 0: num_of_coeffs - 1 ];  % Frequency in Hz
coeff_mags = abs( coeff );      % Magnitudes of each component

% Creating text labels for each stem plot to show magnitude
Bn_labels = sprintfc('%.4f', coeff_mags );

% generating plot
figure; stem( k, coeff_mags ); grid;
xlabel( 'K^{th} Frequency - Hz' ); ylabel ( 'Component Amplitude
- |B_{n}|' );
xlim( [ -1, num_of_coeffs ] );
title ( 'Task 2b - CT Fourier Spectrum for Square Wave' ) ;
text( k, coeff_mags, Bn_labels, 'HorizontalAlignment',
'Left',...
    'VerticalAlignment', 'Bottom' );
```

Code 7-01, Part 3 – Generating Complex Exponential Fourier Series for Square Wave

Only 6 frequency components are to be investigated – the 0^{th} harmonic to the 5^{th} harmonic. The 0^{th} harmonic – also called the DC offset – is simply half the amplitude of the wave. All other coefficients are calculated by discretizing (2) as described in the comments. At the end of each iteration, the integrand's product with the period is summed and divided by the period as a discretized way of evaluating the Fourier integral. The absolute values of the coefficients are then displayed on a stem plot to show the amplitude spectrum of the square wave (Figure 7-03).

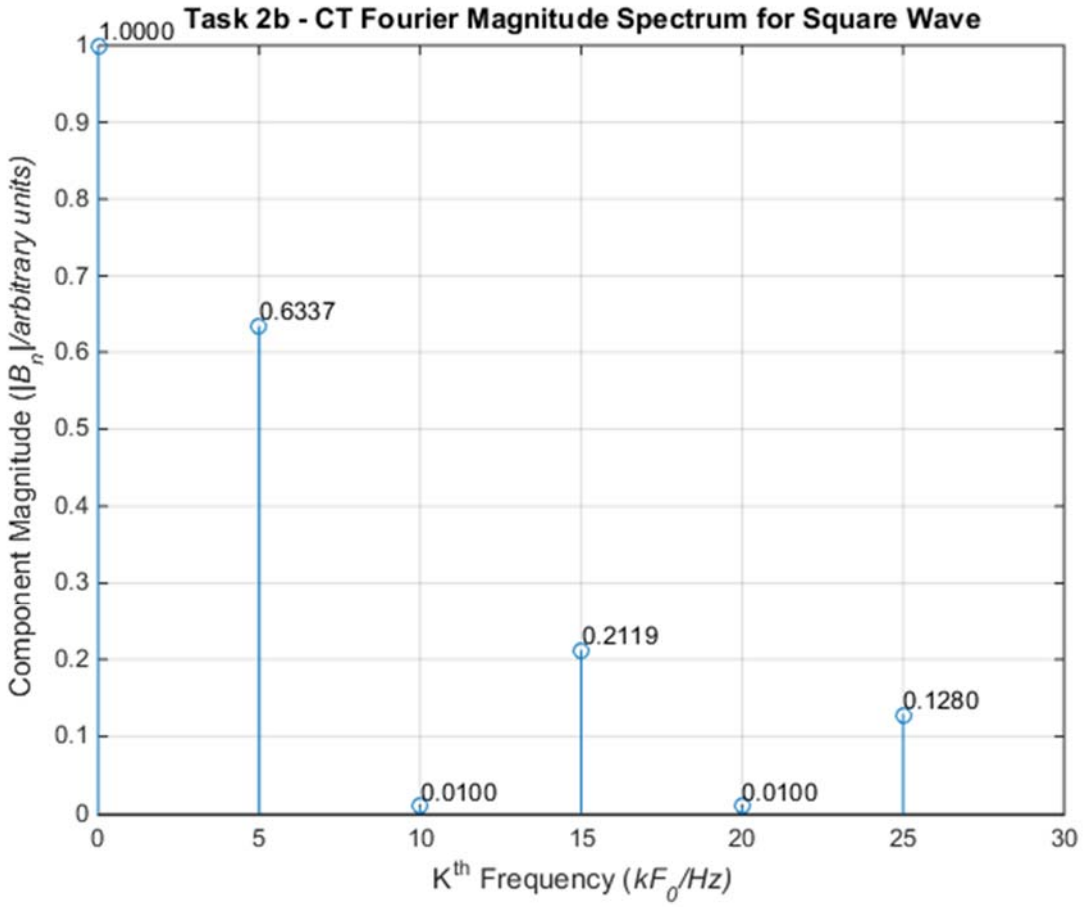


Figure 7-03: Magnitude Spectrum for Square Wave

Verifying Magnitude Spectrum

The complex Fourier series for a continuous time signal is

$$x(t) = \sum_{n=-\infty}^{\infty} B_n e^{jn\omega_0 t}$$

Where B_n is the n[th] Fourier coefficient, and is given by

$$B_n = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jn\omega_0 t}\, dt$$

For the square wave investigated in this lab session,

$$x(t) = \begin{cases} 2, & 0 < t \leq \frac{T_0}{2} = \quad 0 < t \leq 0.1 \\ 0, & \frac{T_0}{2} < t \leq T_0 = 0.1 < t \leq 0.2 \end{cases}$$

Since $x(t)$ is periodic, this means $x(t + T_0) = x(t + 0.2) = x(t)$, and $\omega_0 = \frac{2\pi}{T_0} = \frac{2\pi}{0.2} = 10\pi$

$$\forall\, n \neq 0,$$

$$B_n = \frac{1}{0.2}\int_0^{0.2} x(t) e^{-jn\omega_0 t} dt = \frac{1}{0.2}\int_0^{0.1} 2e^{-jn10\pi t} dt + \frac{1}{0.2}\int_{0.1}^{0.2} 0e^{-jn10\pi t} dt$$

$$B_n = \frac{2}{0.2}\int_0^{0.1} e^{-jn10\pi t} dt = -10\left[\frac{e^{-j10n\pi t}}{-j10n\pi}\right]_0^{0.1} = \frac{-1}{n\pi j}\left[e^{-n\pi j} - e^0\right]$$

$$B_n = \frac{-1}{n\pi j}[\cos(n\pi) - j\sin(n\pi) - 1] \because \sin(n\pi)\ is\ 0\ \forall n \in \mathbb{Z}$$

$$\boldsymbol{B_n = \frac{-1}{n\pi j}[cos(n\pi) - 1]\ \forall\ n \neq 0}$$

$$\boldsymbol{B_0 = \frac{Amplitude}{2}}$$

Comparing Magnitude Values

Fourier Coefficient	$\lvert B_n \rvert$ (theoretical)	$\lvert B_n \rvert$ (MATLAB)	Absolute Error
0[th]	1	1.000	0
1[st]	0.6366	0.6337	2.9×10^{-3}
2[nd]	0	0.0100	0.01
3[rd]	0.2122	0.2119	3×10^{-4}
4[th]	0	0.0100	0.01
5[th]	0.1273	0.1280	7×10^{-4}

Absolute between theoretical and MATLAB values of magnitudes of each Fourier coefficient is negligible for all five components investigated, with the mean absolute error being 3.983×10^{-3}. This is small enough to be negligible, hence theoretical calculations confirm Fourier magnitude spectrum for the square wave obtained through MATLAB.

Lab 07 Variant 2 –Alternate Solution – Assuming Magnitude of 2 units is V_{pk} of 2 units.

Code 7-02 is a MATLAB program which generates data for a square wave signal with amplitude of 2 units, duty cycle of 50%, and frequency of 5 Hz (same as figure on previous page). It plots the square wave signal as a function of time, then extracts and plots the data of a single period of the signal. This data is then used to perform spectral analysis of the square wave signal using a continuous time exponential Fourier series. The magnitudes of the first five complex exponential Fourier components are derived and plotted. It is assumed that magnitude of 2 units means the peak value of the square wave is 2 units.

```
%% EE-394 Digital Signal Processing: Lab 07 - Continous Time
Fourier Series
%% Task 1: Spectral Analysis of Square Wave
%% EE-16163, Section D, TE-EE, Fall 2018
%% PART 0 - Closing all figures, clearing workspace, clearing
log command
clear all; close all; clc;

%% PART 1A - Defining parameters for square wave
T_s    = 0.001;         % sampling time in seconds
T_p    = 0.2;           % square wave period in seconds
duty   = 0.5;           % duty cycle as fraction - 50 %
f      = 1 / T_p;       % frequency of signal
ampl   = 2;             % amplitude of square wave
F0     = f;             % fundamental frequency for spectral
analysis

% Defining time domain
t_start = 0; t_end = 1;
t = t_start : T_s : t_end;

% Defining square wave
sq_wave = ampl * square( 2 * pi * f .* t );
%% PART 1B - Plotting the square wave
figure; plot( t, sq_wave ); grid;
ylim( [ -3, 3] ); xlabel( 'Time (\it{t/seconds})');
ylabel( 'Signal(\it{x(t)})' );title( 'Task 1 - Square Wave' );

%% PART 2A - Extracting information for one period of signal
period_start = find( round ( t * 1000 ) / 1000 == 0 ) ;
period_end = find( round ( t * 1000 ) / 1000 == T_p );
period_indices = period_start : period_end;
sq_wave_period = sq_wave( period_indices );
single_period = t( period_indices );
```

```
%% PART 2B - Plotting extracted signal
figure; plot( single_period, sq_wave_period );
xlabel( 'Time (\it{t/seconds})' );
ylabel( 'Signal(\it{x(t)})' );
title( 'Task 2a- Single Period of Square Wave' );
margin_x = 0.01; margin_y = 0.1;
xlim( [ single_period( 1 ) - margin_x, single_period( end ) +
margin_x ] );
ylim( [ min( sq_wave_period ) - margin_y, ...
    max( sq_wave_period ) + margin_y ] ); grid;

%% PART 3A - Computing the Continuous Time Fourier Series
Coefficients
fs_time = ( -T_p / 2): T_s : ( T_p / 2 );
num_of_coeffs = 6;

for k = 1 : num_of_coeffs
    % Computing for complex fourier coefficient
    B = exp( -1i * 2 * pi * ( k - 1 ) * F0 .* fs_time );

    % Finding specific coefficient through discretised integral
    % Bn = 1/T int( x(t) * e^(-jnwt) ) between -T0/2, T0/2
    % wo = 2piF0t, [-T0/2, T0/2] is just fs_time
    % Because discrete, fs_time values separated by T_s
    % And integration replaced by summation
    % B is still the complex Fourier coefficient
    % length ( sq_wave_period ) = [-T0/2, T0/2]'s number of
elements
    coeff( k ) = ( sum ( sq_wave_period .* B ) ) / ...
        ( length( sq_wave_period ) );
end

%% PART 3B - Plotting Amplitude Spectrum
k =  F0 * ( 0 : num_of_coeffs - 1 );     % Frequency in Hz
coeff_mags = abs( coeff );               % Magnitudes of each
component

% Creating text labels for each stem plot to show magnitude
Bn_labels = sprintfc('%.4f', coeff_mags );

% generating plot
figure; stem( k, coeff_mags ); grid;
xlabel( 'Harmonic(\it{kF_0/Hz})' ); ylabel ( 'Component
Magnitude -\it{|B_{n}|}' );
title ( 'Task 2b - CT Exponential FS Magnitude Spectrum for
Square Wave' ) ;
```

```
text( k, coeff_mags, Bn_labels, 'HorizontalAlignment',
'Left',...
    'VerticalAlignment', 'Bottom' );
```

Code 7-02: Generating square wave, extracting data for period, and performing spectral analysis

Part 1 – Generating the Square Wave

Code 7-02, Part 1 is a snippet showing the MATLAB code used to generate and plot the square wave. All MATLAB arrays are inherently discrete, but can become exceedingly better approximations of continuous ranges if the separation between consecutive values becomes smaller. To generate an approximation of a continuous square wave, the sampling time `T_s` is set to 0.001. Square wave amplitude, frequency, and duty cycle are initialized to their required values of 2 units, 5 Hz, and 0.5 (50 %) respectively. The time domain consists of 1000 values between `t_start` (set to 0) and `t_end` (set to 1) in seconds. This is a good approximation of a continuous time domain, even though it is discrete.

```
%% PART 1A - Defining parameters for square wave
T_s    = 0.001;          % sampling time in seconds
T_p    = 0.2;            % square wave period in seconds
duty   = 0.5;            % duty cycle as fraction - 50 %
f      = 1 / T_p;        % frequency of signal
ampl   = 2;              % amplitude of square wave
F0     = f;              % fundamental frequency for spectral
analysis

% Defining time domain
t_start = 0; t_end = 1;
t = t_start : T_s : t_end;

% Defining square wave
sq_wave = ampl * ( 1 + square( 2 * pi * f .* t, ...
    duty * 100 )) / 2;

%% PART 1B - Plotting the square wave
figure; plot( t, sq_wave ); grid;
ylim( [ -1, 3] ); xlabel( 'Time (\it{t/seconds})');
ylabel( 'Signal(\it{x(t)})' );title( 'Task 1 - Square Wave' );
```

Code 7-02, Part 1 - Generating the Square Wave

The square wave generated by Code 7-02, Part 1 is displayed in Figure 04.

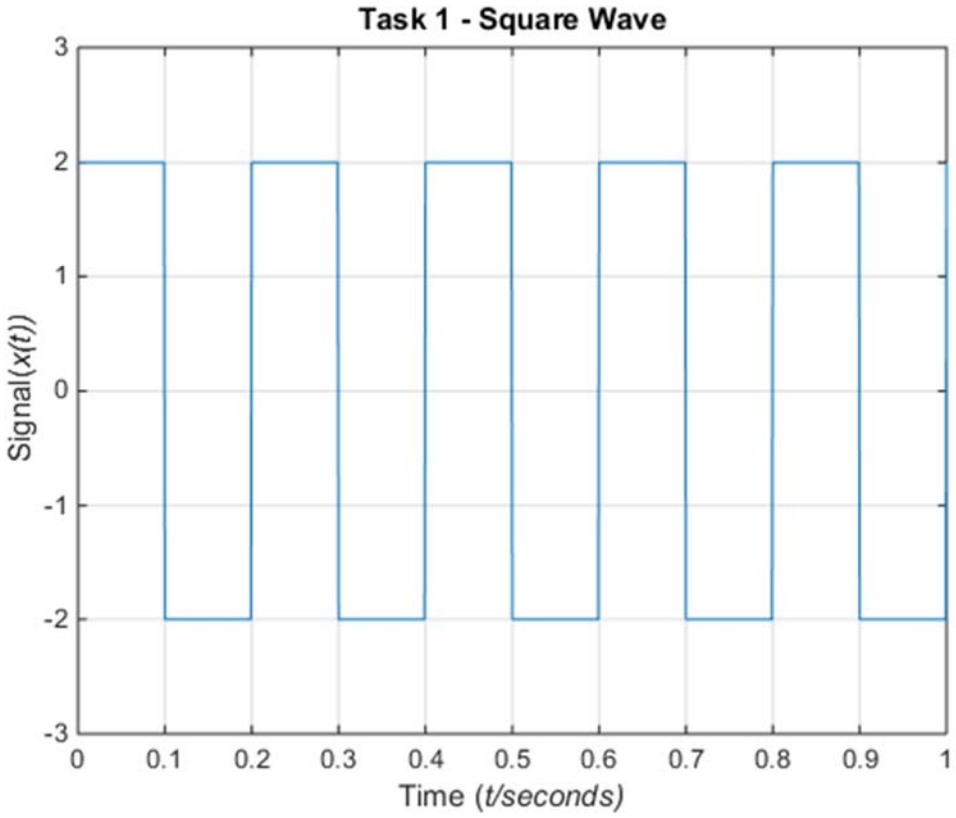


Figure 7-04 - Generating the Square Wave

The square wave itself is plotted with the built-in MATLAB `square` function, which takes two arguments

- Time Domain – expressed in terms of the duration for which to plot the signal as well as its angular frequency. `ampl * square( t )` will plot a square wave of amplitude 2 and period 2π. So 2 * `square( 2* π * f .* t)` will plot a square wave of amplitude 2 and period `T_p` for all values defined in the time domain t.
- Duty Cycle – the proportion of the time period for which the square wave is 'high' or has maximum value, expressed as a percentage (hence `duty` times 100).

Part 2 – Extracting Information for One Period of the Square Wave

Part 2 examines is the portion of Code 7-02 used to extract information about a single period of the square wave. This is done because only a single period of a periodic signal is required for spectral analysis

```
%% PART 2A - Extracting information for one period of signal
period_start = find( round ( t * 1000 ) / 1000 == 0 ) ;
period_end = find( round ( t * 1000 ) / 1000 == T_p );
period_indices = period_start : period_end;
sq_wave_period = sq_wave( period_indices );
single_period = t( period_indices );

%% PART 2B - Plotting extracted signal
figure; plot( single_period, sq_wave_period );
xlabel( 'Time (\it{t/seconds})' );
ylabel( 'Signal(\it{x(t)})' );
title( 'Task 2a- Single Period of Square Wave' );
margin_x = 0.01; margin_y = 0.1;
xlim( [ single_period( 1 ) - margin_x, single_period( end ) +
margin_x ] );
ylim( [ min( sq_wave_period ) - margin_y, ...
    max( sq_wave_period ) + margin_y ] ); grid;
```

Code 7-02, Part 2 - Generating the Square Wave

`period_start` and `period_end` define the index in the time domain array at which the period starts and ends respectively. The built-in MATLAB find function is used to find the index at which the time value, rounded to the nearest 1000th of a second, is 0 (for `t_start`) and `T_p` (for `t_end`). These extreme values are then used to generate an array of indices which will be used to access the time domain vector in order to find both time domain and square wave values corresponding to a single period, which are stored in `sq_wave_period` and `single_period` respectively. The data is then plotted on a graph, which clearly shows the square wave varies from 2 units to -2 units in a duration of approximately 0.2s, with the transition occurring at approximately 0.1s (as is expected with a duty cycle of 50 %).

The extracted signal is displayed in Figure 05.

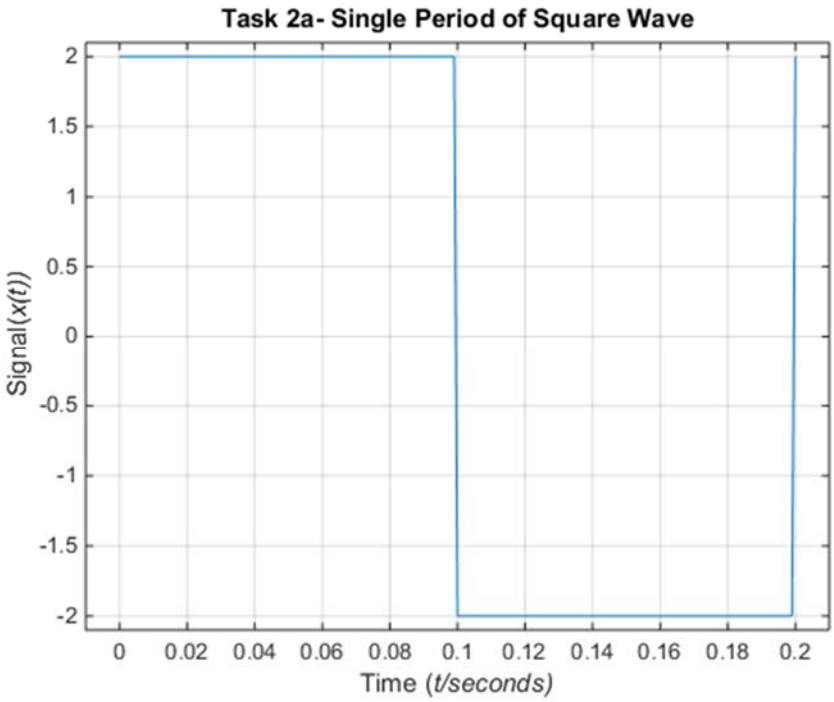


Figure 7-05 – Extracting and Plotting Data for A Single Period of Square Wave

Part 3 – Generating Fourier Series

To perform spectral analysis of the square wave, it must be decomposed into an infinite sum weighted, harmonically related sinusoids called the Fourier series. One form the Fourier series represents this infinite sum in terms of complex exponentials, with the magnitude of each exponential representing the strength of its corresponding frequency component in the signal. The continuous time complex Fourier exponential series is defined by

$$x(t) = \sum_{n=-\infty}^{\infty} B_n e^{jn\omega_0 t} \dots (1)$$

Where B_n is the n[th] Fourier coefficient, and is equal to

$$B_n = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jn\omega_0 t}\, dt \dots (2)$$

Since MATLAB, like all programming languages, can only handle discrete operations, the computation of the Fourier coefficients and the generation of the Fourier series has to be discretized. This is done in Code 02, Part 3, which first generates a discrete time domain over which to perform the 'integration' for each coefficient. The Fourier Series time domain `fs_time` is simply an array spanning the duration of one period, with values separated by sampling time `T_s`.

```
%% PART 3A - Computing the Continuous Time Fourier Series
Coefficients
fs_time = ( -T_p / 2): T_s : ( T_p / 2 );
num_of_coeffs = 6;

for k = 1 : num_of_coeffs
    % Computing for complex fourier coefficient
    B = exp( -1i * 2 * pi * ( k - 1 ) * F0 .* fs_time );

    % Finding specific coefficient through discretised integral
    % Bn = 1/T int( x(t) * e^(-jnwt) ) between -T0/2, T0/2
    % wo = 2piF0t, [-T0/2, T0/2] is just fs_time
    % Because discrete, fs_time values separated by T_s
    % And integration replaced by summation
    % B is still the complex Fourier coefficient
    % length ( sq_wave_period ) = [-T0/2, T0/2]'s number of
elements
    coeff( k ) = ( sum ( sq_wave_period .* B ) ) / ...
        ( length( sq_wave_period ) );
end

%% PART 3B - Plotting Amplitude Spectrum
k =  F0 * ( 0 : num_of_coeffs - 1 );      % Frequency in Hz
coeff_mags = abs( coeff );                % Magnitudes of each
component

% Creating text labels for each stem plot to show magnitude
Bn_labels = sprintfc('%.4f', coeff_mags );

% generating plot
figure; stem( k, coeff_mags ); grid;
xlabel( 'Harmonic(\it{kF_0/Hz})' ); ylabel ( 'Component
Magnitude -\it{|B_{n}|}' );
title ( 'Task 2b - CT Exponential FS Magnitude Spectrum for
Square Wave' ) ;
```

```
text( k, coeff_mags, Bn_labels, 'HorizontalAlignment',
'Left',...
    'VerticalAlignment', 'Bottom' );
```

Code 7-02, Part 3 – Generating Complex Exponential Fourier Series for Square Wave

Only 6 frequency components are to be investigated – the 0th harmonic to the 5th harmonic. The 0th harmonic – also called the DC offset – is simply half the amplitude of the wave. All other coefficients are calculated by discretizing (2) as described in the comments. At the end of each iteration, the integrand's product with the period is summed and divided by the period as a discretized way of evaluating the Fourier integral. The absolute values of the coefficients are then displayed on a stem plot to show the amplitude spectrum of the square wave (Figure 7-03).

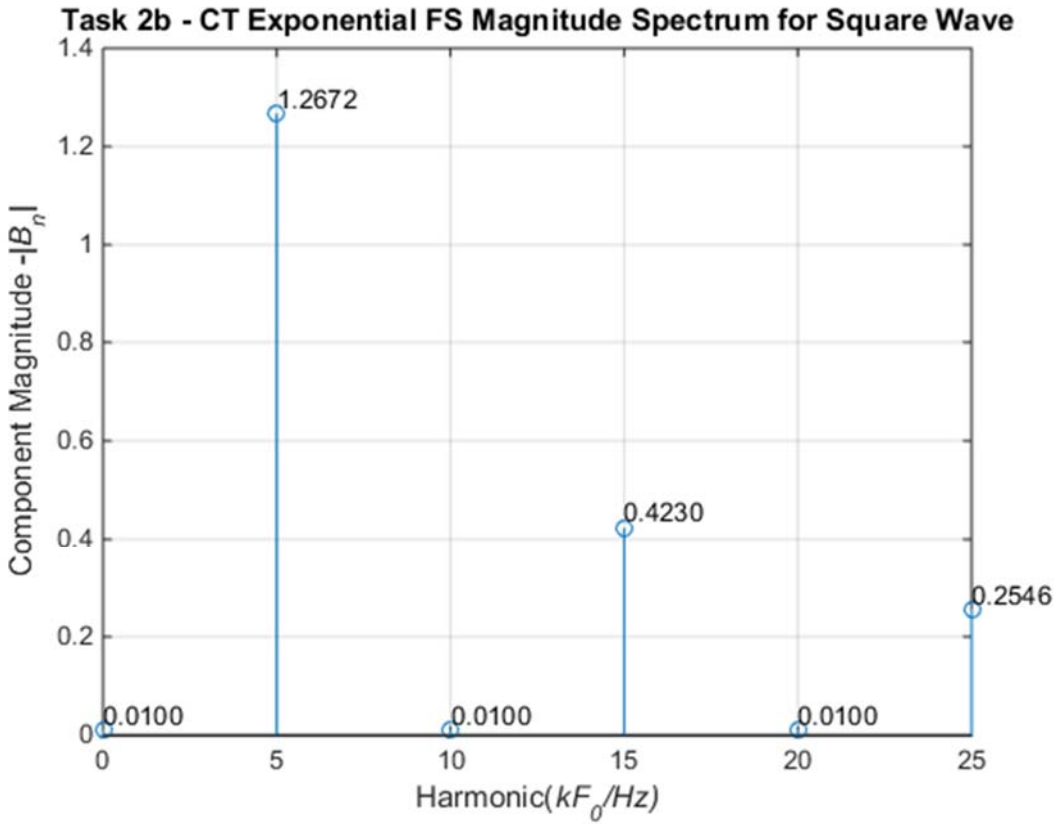


Figure 7-06: Magnitude Spectrum for Square Wave

Verifying Magnitude Spectrum

The complex Fourier series for a continuous time signal is

$$x(t) = \sum_{n=-\infty}^{\infty} B_n e^{jn\omega_0 t}$$

Where B_n is the n[th] Fourier coefficient, and is given by

$$B_n = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jn\omega_0 t}\, dt$$

For the square wave investigated in this lab session,

$$x(t) = \begin{cases} 2, & 0 < t \leq \frac{T_0}{2} = \quad 0 < t \leq 0.1 \\ -2, & \frac{T_0}{2} < t \leq T_0 = 0.1 < t \leq 0.2 \end{cases}$$

Since $x(t)$ is periodic, this means $x(t + T_0) = x(t + 0.2) = x(t)$, and $\omega_0 = \frac{2\pi}{T_0} = \frac{2\pi}{0.2} = 10\pi$

$$\forall\, n \neq 0,$$

$$B_n = \frac{1}{0.2}\int_0^{0.2} x(t) e^{-jn\omega_0 t} dt = \frac{1}{0.2}\int_0^{0.1} 2e^{-jn10\pi t} dt + \frac{1}{0.2}\int_{0.1}^{0.2} -2e^{-jn10\pi t} dt$$

$$B_n = \tfrac{2}{0.2}\textstyle\int_0^{0.1} e^{-jn10\pi t} dt - \tfrac{2}{0.2}\int_{0.1}^{0.2} e^{-jn10\pi t} = 10\left[\frac{e^{-j10n\pi t}}{-j10n\pi}\right]_0^{0.1} - 10\left[\frac{e^{-j10n\pi t}}{-j10n\pi}\right]_{0.1}^{0.2}$$

$$B_n = \frac{10}{-10jn\pi}\left\{e^{-jn\pi} - e^{0} - e^{-j2n\pi} + e^{-jn\pi}\right\} = \frac{-1}{jn\pi}\left[2e^{-jn\pi} - e^{-jn2\pi} - 1\right]$$

$$B_n = -\frac{1}{jn\pi}[2\cos(n\pi) - 2j\sin(n\pi) - \cos(2n\pi) + jsin(2n\pi) - 1]$$

$$\because \cos(n\pi) = (-1)^n and \sin(n\pi) = \sin(2n\pi) = 0\ \forall\, n \in \mathbb{Z}, n \neq 0$$

$$B_n = -\frac{1}{jn\pi}[2(-1)^n - 1 - 1\,] = -\frac{1}{jn\pi}[2(-1)^n - 2] = \frac{-2}{jn\pi}[(-1)^n - 1]$$

$$\boldsymbol{B_n = -\frac{2}{jn\pi}[(-1)^n - 1]\ \forall\, n \in \mathbb{Z}, n \neq 0}$$

$$\boldsymbol{B_0 = DC\ Offset = 0}$$

Comparing Magnitude Values

Fourier Coefficient	$\|B_n\|$ (theoretical)	$\|B_n\|$ (MATLAB)	Absolute Error
0^{th}	0	0	0
1^{st}	1.2732	1.2672	6×10^{-3}
2^{nd}	0	0.0100	0.01
3^{rd}	0.4244	0.4230	1.4×10^{-3}
4^{th}	0	0.0100	0.01
5^{th}	0.2546	0.2546	0

Absolute between theoretical and MATLAB values of magnitudes of each Fourier coefficient is negligible for all five components investigated, with the mean absolute error being 4.567×10^{-3}. This is small enough to be negligible, hence theoretical calculations confirm Fourier magnitude spectrum for the square wave obtained through MATLAB.

Laboratory Session No. 08

Objective:

Analysis and Synthesis of Signals through Discrete Fourier Transform (DFT)

Post Lab Exercises:

Question 1:

Discuss significance of "N" in N point DFT.

Answer:

The DFT of a finite-point signal $x(n)$ is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j.2\pi.\frac{kn}{N}}$$

The parameter N in a DFT of a signal $x(n)$ represents the number of frequency points which will appear in the Fourier spectra for the signal. The larger the number of data points N, the smaller the separation $\frac{k}{N}$ between consecutive frequency components in a given interval of the discrete-time frequency spectrum, and therefore the higher the frequency resolution. Thus, a higher value of N will make the DFT a better approximation of a continuous time Fourier Transform, as more of the infinitely many phasors present in a CTFT will be represented in the resulting Fourier spectra.

However, larger number of samples N also means more computations per each frequency component calculation, which can affect the speed of the DFT process.

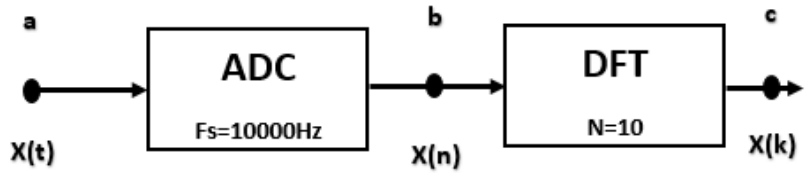
Question 2:

DFT can deal both **periodic** and **non-periodic** signals.

Question 3:

Consider the following figure, if this system receives input $x(t) = \sin(2\pi 1000t) + \sin(2\pi 3000t) + \cos(2\pi 0t) + \cos(2\pi 2000t)$ then, plot outputs at point *"b"* and *"c"*.

(Note: Both magnitude and phase plot is required at point *"c"*)



Answer:

Figure 8-01 shows the waveform at b. It consists of 10 samples of the continuous time signal $(t) = \sin(2\pi 1000t) + \sin(2\pi 3000t) + \cos(2\pi 0t) + \cos(2\pi 2000t)$ produced by ADC with sampling frequency $F_S = 10\ kHz$.

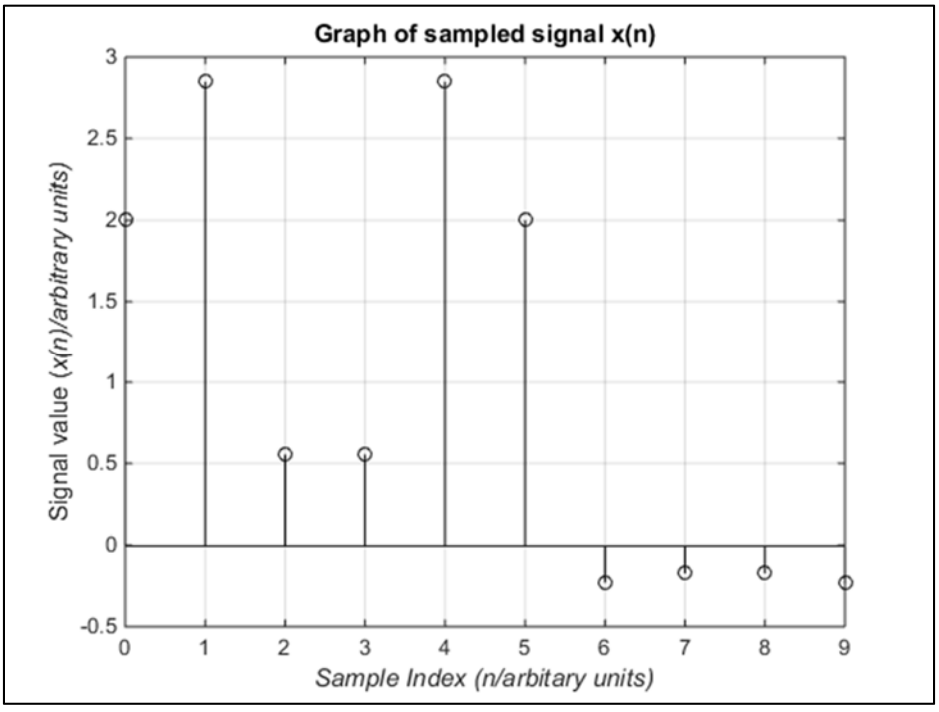


Figure 8-04: Sampled Signal $x(n)$

The 10 point DFT of this sampled signal is shown in Figures 2 and 3, which are the magnitude and phase spectrums respectively.

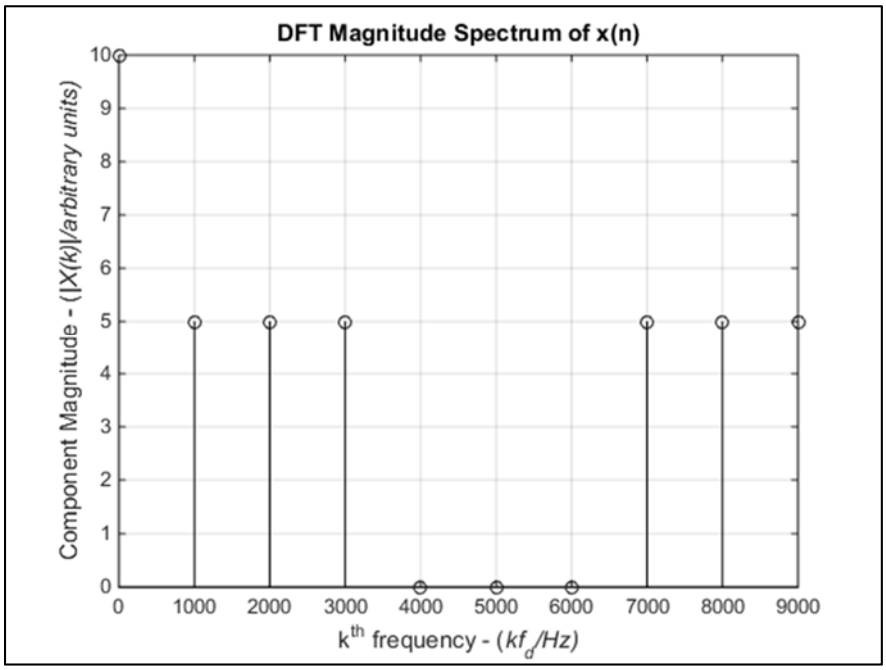


Figure 8-05:DFT Magnitude Spectrum

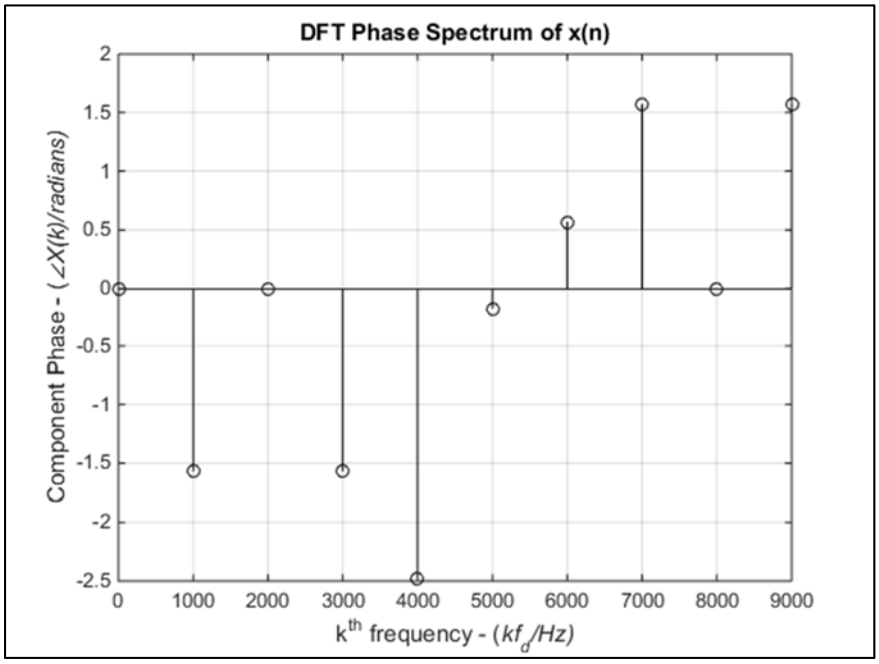


Figure 8-06: DFT Phase Spectrum

Question 4:

Discuss at least three applications of FFT in Electrical power system Engineering.

Answer:

1. Harmonic Detection: Spectral analysis with FFT can detect the presence and magnitudes of harmonics in supply voltage. This data can then be used to ascertain the quality factor of the power supply and whether or not it meets distribution guidelines – if total harmonic distortion exceeds a predefined threshold, the power quality factor is reduced and indicates a greater risk of harmonics damaging equipment due to overheating.
2. Harmonic Filter Design: FFT-based spectral analysis of an input signal can also determine the frequencies and relative magnitudes of harmonics present in supply voltage. This information can be used to decide the cutoff frequency for harmonic filter banks, which can be used to mitigate harmonic distortion in loads.
3. Smart Grids: FFT and its variants are used in smart grids to derive data about harmonic orders, fast transients, time-varying components, and steady state components from signal measurements. They are also used in secondary signal processing in smart grids to derive secondary parameters from voltage/current signals for pattern recognition, and for time-varying harmonic analysis.
4. Power Quality Event Detection: Different kinds of power quality events (short circuit faults, capacitive load switching, etc.) have different effects on the frequency spectra of both current and voltage signals. FFT can be used to identify these power quality events using spectral analysis of voltage and current signals. The spectral analysis can then be used to identify the cause of the power quality event, and can lead to an informed approach to power quality improvement.
5. Induction Motor Fault Detection: FFT can be used to analyse the frequency spectra of both the vibrations and stator of a three phase induction motor in order to detect faults. Each kind of fault in the induction motor's rotor (bearing fault, shaft fault, etc.) has characteristic frequencies that affect the frequency spectra of the two aforementioned physical quantities. Thus, analyzing the spectra of stator current or frame vibrations is a non-intrusive method of fault diagnosis for induction motors.

Laboratory Session No. 09

Objective:

To become familiar with practical constraints in calculating Fourier Transform of any real world signal.

Post Lab Exercises:

Question 1:

Briefly discuss the concept of spectral leakage.

Answer:

Spectral leakage is a phenomenon observed in the magnitude spectra obtained by applying the Fast Fourier Transform operation to a signal. It is observed in the form of peaks called side lobes that occur at frequency values that are not actually present in the original signal. These side lobes are almost always centered on a main lobe – a spike corresponding to a discrete frequency component that is the closest possible approximation to continuous one present in the signal. This phenomenon is called leakage because the energy originally confined to the main lobe 'leaks' into side lobes. In effect, this is a redistribution of the signal over the entire frequency range of investigation.

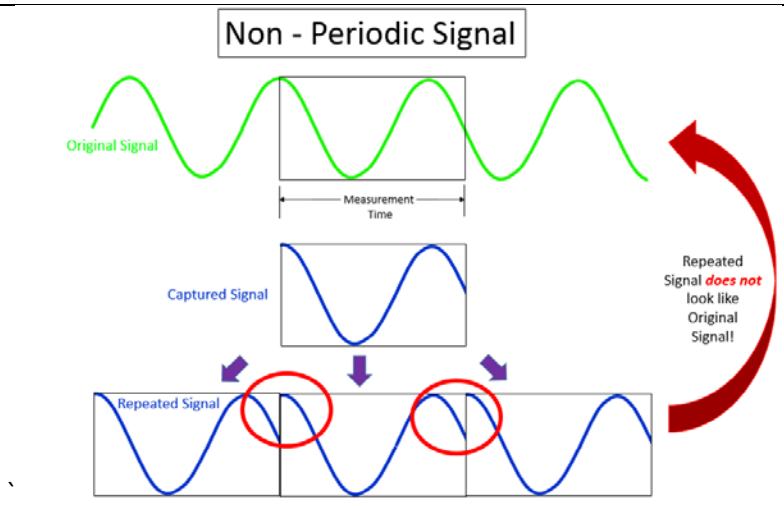


Figure 9-01: Discontinuities in a Captured Signal Prior to FFT

Source: PLM Automation, Siemens

Spectral leakage occurs due to the presence of discontinuities (Figure 9-01) between the initial and final values of a captured signal – a portion of the signal that corresponds to the N data points used in an FFT at a given sampling frequency F_S. As demonstrated in Figure 9-02, discontinuities – being transient signals in the time domain – represent broadband signals in the frequency domain, which is why the manifest as side lobes in the FFT magnitude spectrum.

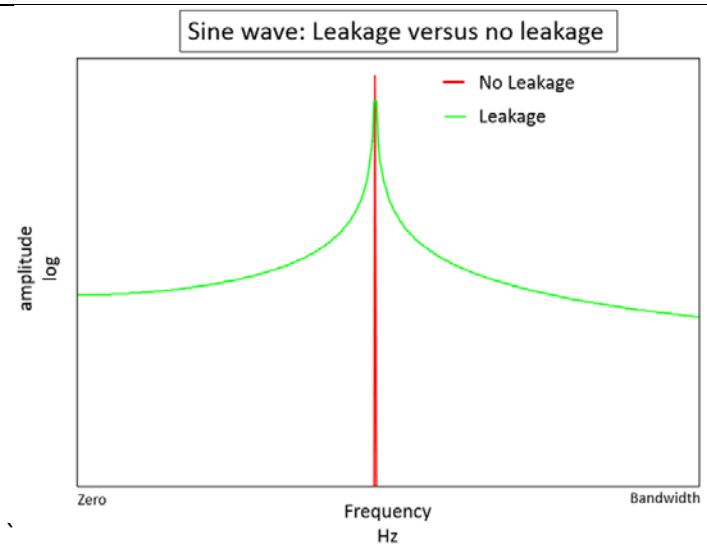


Figure 9-02: Monotonic sinusoid FFT magnitude spectrum showing spectral leakage.

Source: PLM Automation, Siemens

Question 2:

Discuss the solution of spectral leakage.

Answer:

A practical method of minimising spectral leakage is to use a windowing function (Hamming, Flattop, Exponential, etc.) to minimize discontinuities between the terminal values of the captured signal. The windowing function is multiplied with the captured signal prior to the FFT operation and in doing so normalizes its initial and final values, reducing the sharp transient in the re-created signal. This confines spectral leakage over a smaller frequency interval, and therefore improves the accuracy of the spectra.

Figure 09-03 shows the result of using a windowing function to minimise spectral leakage in the magnitude spectrum of the monotonic sine wave in Figure 09-02.

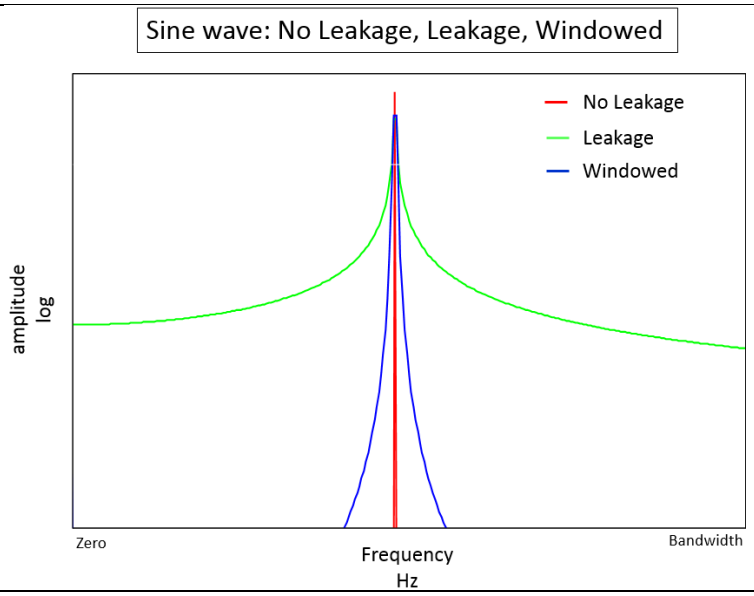


Figure 9-03: The result of using a windowing function to minimise spectral leakage

Source: PLM Automation, Siemens

Question 3:

Discuss the importance of window size (value of N) on frequency spectrum.

Answer:

The window size N is the number of data points of a signal that will be used to form a single frame to be used in computing the signal's FFT. The choice of the window size N is important because it determines how the frequency resolution of the discrete spectrum and, in doing so, also predicts how well the discrete spectrum approximates a continuous one. The resolution of a discrete spectrum evaluated using FFT is given by $f_d = \frac{F_S}{N}$, where f_d is the discrete time frequency and the between consecutive values in the spectrum, F_S is the frequency at which the signal is sampled in the time domain and N is window size in terms of signal samples. The larger the number of data points N, the lower the value of f_d, and the higher the frequency resolution. As $N \to \infty, f_d \to 0$ which essentially transforms the FFT spectrum into a continuous frequency spectrum, albeit at the cost of computation time.

Question 4:

Run the following MATLAB script, observe and discuss the resulting plots.

```
clear; close;
fs=4000;
N=400;
n=0:N-1;
y=10*cos(2*pi*1500*n/fs);
f=abs(fft(y));
k=0:N-1;
df=fs/N;
kf=k.*df;
subplot(411);
stem(kf,f);
xlabel('Frequency Hz');
title('Magnitude Spectrum Without Spectral Leakage');
% If frequency bins are not available (Spectral Leakage) %
N=50;
n=0:N-1;
nfft=1024;
y=100*cos(2*pi*1500*n/fs);
f=abs(fft(y,nfft));
k=0:nfft-1;
df=fs/nfft;
kf=k.*df;
subplot(412);
plot(kf,f);
xlabel('Frequency Hz');
title('Magnitude Spectrum With spectral leakage Leakage');

% Solution of Spectral leakage (Windowing)%
```

```
win=window(@hamming,N);
subplot(413);
stem(n,win);
title('Window');
y_win=y.*win';
f_win=abs(fft(y_win,nfft));
k=0:nfft-1;
df=fs/nfft;
kf=k.*df;
subplot(414);
plot(kf,f_win);
xlabel('Frequency Hz');
title('Magnitude Spectrum With Windowing');
```

Answer:

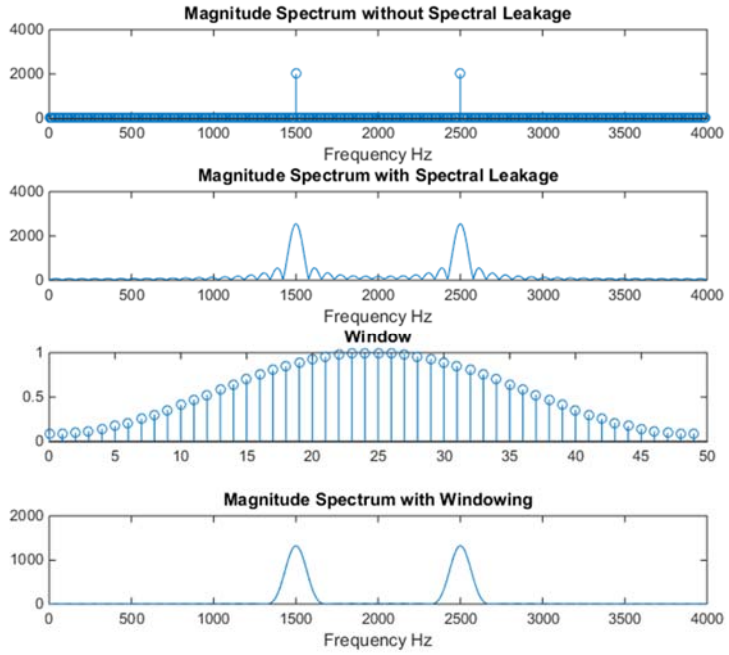


Figure 9-04: Leakage in FFT Magnitude Spectra and Windowing Functions

The given MATLAB program demonstrates how spectral leakage can occur in the FFT magnitude spectrum for a periodic signal and how a windowing function can be used to mitigate this. The program first samples the signal $y(t) = 10\cos(2\pi.1500.t)$ using 400 data points and a sampling frequency of 4 kHz. It then performs a FFT operation on the sampled signal and plots the resulting magnitude spectrum with a frequency resolution of $f_d = \frac{F_S}{N} = \frac{4000}{400} = 10\ Hz$.

First plots the FFT magnitude spectrum of the periodic signal. It shows only two non-zero spikes at 1500 Hz and 2500 Hz respectively. A 1500 Hz frequency component is indeed present in the signal (in fact, it is the only frequency component present in the signal). The second spike at 2500 Hz is a result of conjugate symmetry in the FFT operation – it does not belong to the actual spectrum of the signal. Since the sampling frequency $F_S = 4\ kHz$, the folding frequency of the spectrum is $\frac{F_S}{2} = \frac{4\ kHz}{2} = 2\ kHz$, which means any frequency components greater than 2 kHz (such as 2.5 kHz) are simply the spectrum folding in on itself. Only two spikes occur because there exists an integer k such that $k \times \frac{F_S}{N} = k \times f_d = k \times 10 = 1.5\ kHz$ i.e. a discrete frequency domain bin does exist to accommodate the spike for 1.5 kHz (and its symmetrical spectrum).

The second plot demonstrates spectral leakage in the FFT magnitude spectrum of the same signal when it is sampled using only 50 data points and the FFT is calculated with 1024 data points. The frequency resolution in this case is $\frac{F_S}{N} = \frac{4000}{1024} = 3.9063$. No integer k exists to satisfy the relation $k \times \frac{F_S}{N} = k \times 3.9063 = 1.5\ kHz$. Thus no discrete frequency bin exists in this spectrum to accommodate the spikes at 1.5 kHz and the conjugate symmetrical spike 2.5 kHz. As a result, the spectrum no longer consists of just 2 spikes – the spectrum now consists of significant non-zero peaks over the entire frequency domain.

The code then creates a plot of the Hamming function – a windowing function whose value increases from 0 at the beginning to 1 in the midband before decaying to 0 by the end again. This windowing function is multiplied with the 50 sampled signal values to create a modified version of the signal in which the initial and final values of the signal have been attenuated to zero. This eliminates discontinuities between successive captured frames of the FFT, and therefore decreases spectral leakage as shown in the new FFT magnitude spectrum. Leakage has not been eliminated entirely – it has only been confined to smaller side lobes that no longer span the

extent of the frequency domain. This was achieved without modifying the frequency resolution of the signal.

Laboratory Session No. 10

(Open Ended Lab 02)

Objective:

To convert an analog (current) signal into digital signal using ADC (audio card). Display it on MATLAB Simulink environment and perform FFT of the resulting current signal.

Required Components:

7. Audio Card
8. Current Sensor (current sensing resistor / hall effect sensor / CT)
9. Vero board
10. Audio jack
11. Harmonic producing Load (Electronic devices)
12. PC with MATLAB environment

Procedure:

- Using current sensor, convert the current flowing through load into an equivalent voltage.
- If required, using VDR to convert the voltage as obtained from current sensor to a voltage compatible to audio card (show all the calculations of resistances with their power ratings).
- Set the sampling frequency of the audio card ADC in MATLAB Simulink environment with proper justification
- Plot the acquired current waveform to Simulink scope.
- Mention the safe operating range of your equipment.
- Plot the frequency spectrum of the obtained current waveform. Use windowing function to reduce DFT leakage if required.
- Also, plot the frequency spectrum of the line voltage as obtained in open ended lab 01.

Introduction

The objective of this open-ended lab is to investigate the use of a sound card as an analog-to-digital converter (ADC) and to perform frequency analysis of the sampled current waveform using a windowed Fast Fourier Transform (FFT).

Method

Figure 01 shows a block diagram describing the steps in acquiring a digital version of AC mains supply current.

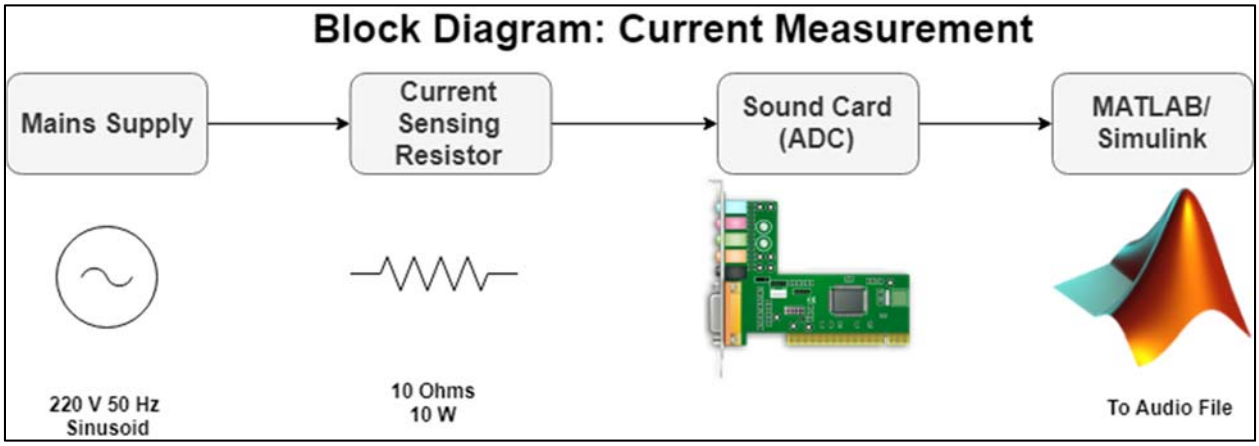


Figure 10-7: Block Diagram - Current Measurement

Mains supply is an alternating 50 Hz sinusoid voltage that is converted to an equivalent current signal by a current sensing resistor (CSR). The CSR is a high wattage, low resistance resistor that can be placed in series with a circuit branch without significantly modifying the branch's current. The power dissipated by the CSR (and hence the voltage dropped across it) are proportional to the current in the branch. The proportional relationship is simply Ohms Law $v_{CSR}(t) = i_{CSR}(t)R$.

This voltage acts as an input for a sound card which converts the continuous time, continuous value (analog) voltage signal at the terminals of the CSR to a discrete time, discrete value (digital) equivalent. A Simulink Model (Figure 02) is used to read the sampled data from a computer's audio input, rescale the samples by a factor of $\frac{1}{R}$ (where R is the CSR resistance) to extract the equivalent current signal, and write it to an audio file. The audio file is then imported into MATLAB and its frequency domain analysis is carried out with MATLAB's built-in f f t command.

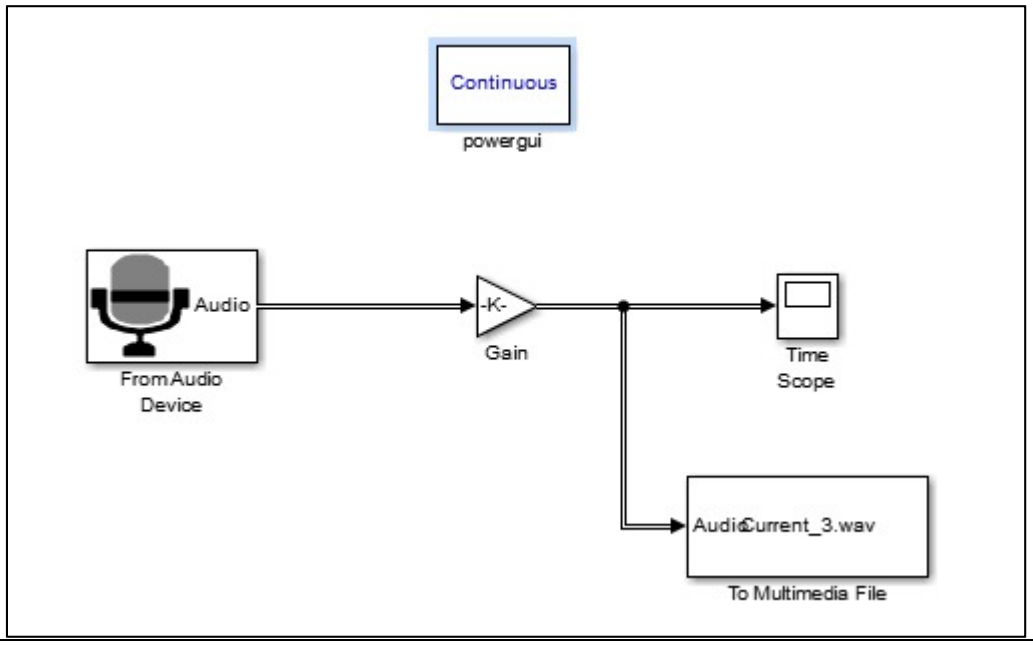


Figure 10-02 – Simulink Model for Storing Current Data

Calculations

Voltage Sensing Circuit

- Audio Card Sensitivity Level: 0
- VDR Resistors: 1 kΩ resistor, 50 kΩ potentiometer
- Mains Supply (Multimeter Measurement) = 239 V_{RMS}
- VDR Output = 0.5 V_{RMS}
- Transformer Output = 14.4 V – but this does not affect the gain as it cancels out.
- Sound Card Gain = 2.55
- Total Gain = 532.209
- Mains Supply Measured in MATLAB = 238.7 V_{RMS}

Current Sensing Circuit

Using a 4 – 5 Watt incandescent bulb and a 10 Watt, 10 Ohms current sensing resistor.

Since the same sound card was used for discretizing the voltage signal, the gain is still 0.5. As per Ohms Law, $v_{CSR} = i_{CSR} \times R_{CSR} \Rightarrow i_{CSR} = \frac{v_{CSR}}{R_{CSR}} = \frac{v_{CSR}}{10}$

$$\boldsymbol{Total\ gain = 2.55^{-1} \times 0.1}$$

Frequency Analysis

Code 01 shows a MATLAB program that is used to perform frequency analysis of the current signal extracted by the setup described earlier. Specifically, the program plots the magnitude spectrum for the current signal using a windowed-FFT operation.

```
%%  DSP Open-Ended Lab 02 - Task 2: FFT Magnitude Analysis of Current
%   This program reads values of a current waveform that has been stored
%   in the form of an audio file. It uses the recording to perform
%   frequency analysis of the current waveform using a windowed-FFT

%   Reading current data and sampling frequency from audio file
[ i_data, f_s ] = audioread( 'Current_2.wav' );

%   Calculating the number of current data samples read
n = size( i_data , 1 );

%   Defining number of points to be used in FFT analysis
win_size = 40e-3 * f_s;          % through trial and error
frame_size = win_size / 2;       % for overlap

%   creating buffer which will divide the entire sample data into chunks,
%   each of which will form its own FFT window - frame size argument
%   determines the level of overlap between consecutive windows
i_buffer = buffer( i_data, win_size, frame_size );

%   Storing number of rows and columns in buffer as counter vars
%   num of rows = FFT window length
%   num of cols = Number of windows into which the buffer has been divided
[ i_buf_row, i_buf_col ] = size( i_buffer );

%   Defining Hamming Function to be used for normalization b/w windows
%   Removes discontinuities by scaling samples near endpoints of windows to
%   0 while keeping midband values unchanged - minimizes spectral leakage
hamming_win = window( @hamming, win_size );

%   Array to store results of FFT for each window in buffer
i_mag_spectrum = [];

%   For every buffer in the buffered version of the signal
for i = 1 : i_buf_col
    %   Normalizing values for current FFT window
    buffer_data = i_buffer( :, i ) .* hamming_win;

    %   Magnitude spectrum for current FFT window
    %   Since n_fft = f_s, frequency resolution is 1?
    buffer_fft_mag = abs( fft( buffer_data, f_s ) );

    %   Appending this buffer's FFT results to rest
    i_mag_spectrum = [ i_mag_spectrum buffer_fft_mag ];
end
```

```
%   Create a figure window in which to render FFT magnitude plot
figure;

%   For every FFT window in the buffer
for i = 1 : i_buf_col
    %   clear drawing canvas before drawing new plot - prevents successive
    %   plots from overlapping
    clf;

    %   plot the magnitude spectrum for this FFT window
    plot( i_mag_spectrum( :, i ) );

    %   render the plot to the figure
    drawnow;
end

%   Adding details to graph
title( {'FFT Magnitude Spectrum - Current', ...
    '50 Hz Sinusoidal Current Spectrum over 44.1 kHz'} );
xlabel( 'Frequency (\it{f/Hz})' );
ylabel( 'Mangitude (\it{arbitrary units})' );
grid;
```

Code 10-01 – Program to plot magnitude spectrum of current signal using windowed FFT

The program reads all the current's sample values as well as the corresponding sampling frequency using MATLAB's audioread command. The program assumes the current waveform/audio file is present in the program's working directory. It then uses the MATLAB buffer command to divide the sequence of values representing the current signal into chunks or buffers approximately $F_S \times 40 \times 10^{-3} = 1764$ samples each. This the number of data points which will be used to compute a FFT of the sinusoid signal, and is called the window size.

The buffer command also uses an additional `frame_size` argument, which specifies the number of samples that will overlap or underlap between consecutive windows the signal data is divided into. In this case, the frame size is half of the window size i.e. 882. This means that all windows, except for the first and last ones, will consist of only 882 unique samples that have not been used to compute an FFT before, with the remaining 882 belonging to the previous sample i.e. an overlap of 882 samples. This is only true if $0 < frame\ size < window\ size$, as per MATLAB documentation.

The concept of overlapping windows is demonstrated in the following diagrams. A discrete signal of 8 samples is divided into 4 windows of 2 samples each, creating 4 unique non-overlapping windows.

Index		**0**	1	2	3	4	5	6	7	
Sample	…	1	2	8	10	11	12	8	3	…
Window		Window 1		Window 2		Window 3		Window 4		…

If the same procedure was to be repeated with a frame size of 1, this would double the number of unique windows present in the same sample length.

Index		**0**	1	2	3	4	5	6	7	
Sample	…	1	2	8	10	11	12	8	3	…
Window		Window 1		Window 3		Window 5		Window 7		…
			Window 2		Window 4		Window 6		Window 8	

This is why the signal, originally expected to be split into $\frac{n}{window\ size} = \frac{221184}{1764} \approx 125$ non-overlapping windows, will be split into ceil $\left(2 \times \frac{221184}{1764}\right) \approx 251$ overlapping windows.

Therefore each of the 251 columns in the matrix returned by the buffer command consist of 1764 samples from the original 221184 samples of the current waveform. Each of these windows is first multiplied by a sampled version of the Hamming function consisting of the same number of samples as the window. The result is a window in which the samples at the endpoints have been attenuated to 0, while midband samples remain unchanged due to the element-wise multiplication with corresponding Hamming function values.

While this has the benefit of removing discontinuities between endpoints of consecutive windows, it does distort the original waveform significantly, thereby minimizing spectral leakage at the cost of signal fidelity. This is precisely why overlapping windows were created using the buffer as described earlier: the overlap ensures that values attenuated by the Hamming function in one window will remain unchanged due to being in the midband of the consecutive window, and hence will never be completely eliminated from the frequency analysis operation. This both eliminates spectral leakage due to window endpoint discontinuities while also minimizing loss of signal information.

The overlapping windowed FFT is used to investigate the magnitude spectra of both current and voltage waveforms. These spectra are symmetrical about 22.05 kHz (as expected, since the folding frequency of the spectra is half of the sampling frequency), and offer insight into the nature of both current and voltage harmonics. The spectra are shown in Figures 03 and 04.

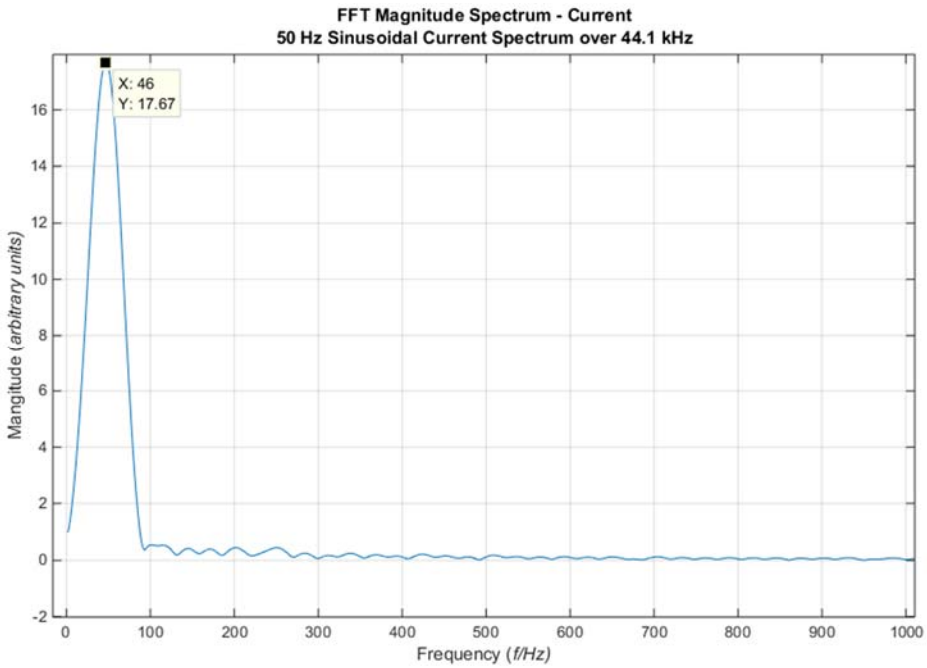


Figure 10-03 – Current Signal Magnitude Spectrum (zoomed into region of interest)

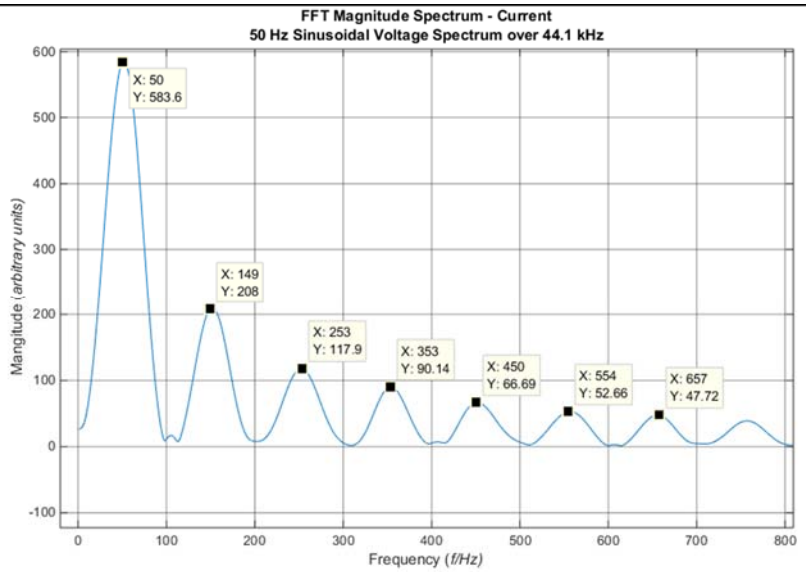


Figure 10-04 – Voltage Signal Magnitude Spectrum (zoomed into region of interest)

Analysis of Spectra

- Both the current and magnitude spectra show a large spike around 50 Hz, indicating that the highest contribution to the mains supply comes from a 50 Hz sinusoid.
- Both spectra show signs of spectral leakage: non-zero magnitudes for frequency components that are most likely not present in the original signal. This is a consequence of not using a sufficiently high frequency resolution, and manifests as side lobes around the 50 Hz peak.
- Both spectra are symmetrical about the frequency of 22.05 kHz. This is the folding frequency for a sampling frequency of 44.1 kHz. A symmetrical spike exists around 44.05 kHz as well, and represents the conjugate phasor for the 50 Hz spike in this periodic extension of the spectrum.
- Only the voltage spectrum shows harmonics distortion – significant, non-zero spikes at frequencies that are odd multiples of the fundamental frequency of 50 Hz. This is likely due to the step-down transformer being used in the voltage sensing circuit creating non-linearities due to core saturation.
- Almost no harmonic distortion can be observed in the current waveform. This may be due to the load connected to the CSR being well-designed to minimise harmonic distortion.

Laboratory Session No. 11

Objective:

To understand the concept of window overlapping.

Post Lab Exercises:

Question 1:

Run the following MATLAB script, attach and discuss the resulting plot.

```
clear all;
clc;
[y fs] = audioread('wavTones.com.unregistred.warble_1000-
2000Hz_-6dBFS_3s.wav');
n=0:5000;
win_size = 40e-3*fs;
frame_rate = 20e-3*fs;
Y = buffer(y,win_size,frame_rate);
[m n] = size(Y);
win = window(@hamming,win_size);
YYF = [];
for i = 1:n
    YY = Y(:,i).*win;
    YF = abs(fft(YY,fs));
    YYF = [YYF YF];
end
figure;
for i = 1:n
   clf;
   plot(YYF(:,i));
```

```
    drawnow;
end
```

Answer

Code Explanation

This MATLAB program performs spectral analysis of an audio file using a version of the FFT algorithm with overlapping windows to minimise loss of transient data due to conventional windowing functions. In this investigation, the two audio files analysed by the program were 3s recordings of sinusoid signals whose frequencies ranged from 1 kHz to 2 kHz and from 2 kHz to 4 kHz respectively.

The program first clears the MATLAB workspace of data left from previous executions of programs. It then uses the built-in `audioread` function to read both the values of a digital audio signal and its sampling frequency. These data are returned by the function and stored in the variables `y` and `fs` respectively.

As both audio files have been sampled at a rate of 44.1 kHz, they contain a total of $F_S \times t_{duration} = 44.1 \times 10^3 \frac{samples}{second} \times 3\ seconds = 132{,}300$ samples. Of these 132,300 samples, only $40 \times 10^{-3} \times F_S = 1764$ samples are used to perform the FFT. This is because the signal is known to be periodic, hence evaluating FFT using a subset of the total samples is perfectly adequate as long as the signal completes at least one cycle in the period spanned by these samples. This is the window size of the FFT operation – the total number of data points that will be used to perform spectral analysis of the signal.

Instead of computing the FFT for the entire 1764 sample window, the program decides to break this window into smaller frames of $20 \times 10^{-3} \times F_S = 882$ samples at a time. Furthermore, these windows of samples will overlap – with the exception of the first window, all windows will include some samples from the previous window in computing the FFT. The original signal y, the window size, and the frame size are passed to the built-in buffer function to return chunks of the actual signal that will be used to perform the final FFT. The number of rows in Y represents the number of points which will be used to evaluate the FFT while the number of columns

represents the total number of buffers or overlapping windows that will be required for this operation.

After defining a Hamming windowing function for the required window size, the program multiplies values of the signal corresponding to each frame with the windowing function (to reduce spectral leakage), computes the FFT, and appends the magnitude spectrum data to a results vector Y. The FFT spectra are then plotted on a figure that is refreshed every time the data for a new buffer is added.

The resulting spectra for each audio file are as follows.

Magnitude Spectrum Analysis: 1 kHz – 2 kHz

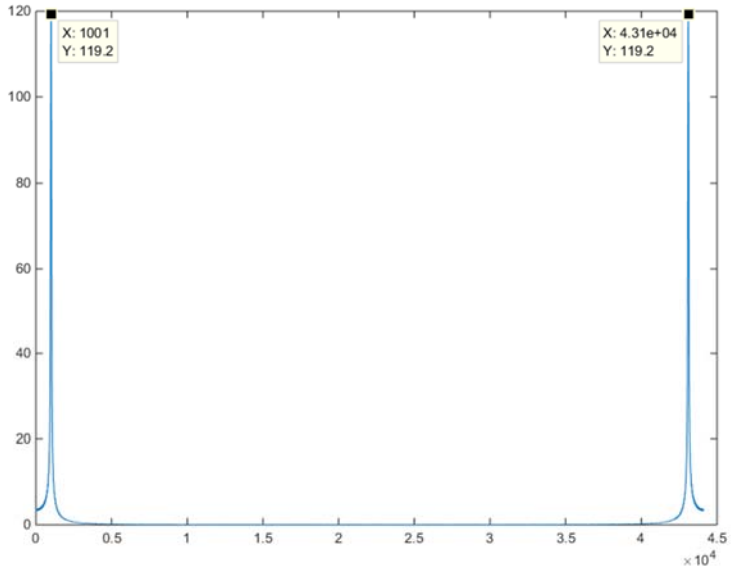


Figure 11-01: FFT Magnitude Spectrum for 1 kHz – 2 kHz sinusoid audio signal

Figure 11-01 shows the magnitude spectrum for the audio file whose frequency varies from 1 kHz to 2 kHz. As the sampling frequency for both audio files was 44.1 kHz, the folding frequency for their spectra is $\frac{F_S}{2} = \frac{44.1}{2} kHz = 22.05\ kHz$, making the range $[0\ kHz, 22.05\ kHz]$ the region of interest in both spectra.

The figure shows a spike at approximately 1101 Hz with a magnitude of 119.2 units. Some spectral leakage can be observed in the form of finite, non-zero magnitude values centered on the

1.001 Hz spike. Surprisingly, there is no spike at or around the 2 kHz frequency, indicating that the signal's frequency is at around 1 kHz for the majority of its duration.

The spike at 43.1 kHz simply corresponds to the conjugate of the 1.001 kHz spike in the next magnitude spectrum, which began at approximately 22.05 kHz and is centered at 44.1 kHz.

Magnitude Spectrum Analysis: 2 kHz – 4 kHz

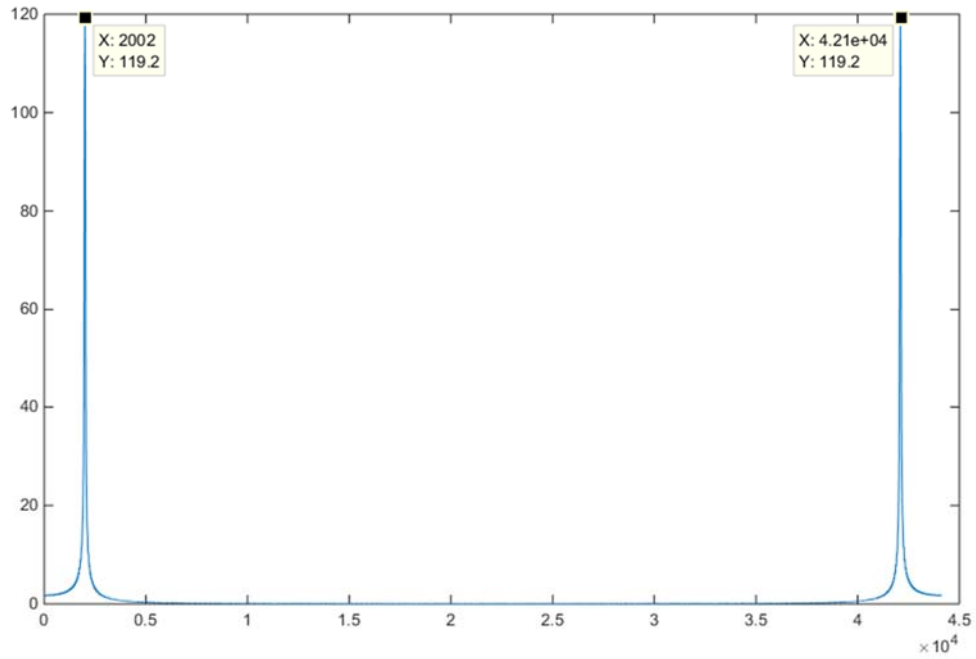


Figure 11-02: FFT Magnitude Spectrum for 2 kHz – 4 kHz sinusoid audio signal

Figure 11-02 shows the magnitude spectrum for the sinusoid signal whose frequency varies from 2 kHz to 4 kHz. As expected, the spectrum is symmetric about the folding frequency $\frac{F_S}{2} = 22.05\,kHz$ and shows some signs of spectral leakage. Again, there is a single spike at approximately 2.002 kHz, with no significant spike at the 4 kHz frequency. This suggests that the 2 kHz component is more significant in determining the overall audio signal waveform, and that for a greater part of the signal the frequency is closer to 2 kHz. The spike at 42.1 kHz is the conjugate of the 2 kHz spike in the next periodic repetition of the frequency spectrum that begins at 22.05 kHz.