DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

18ECS301J – APPLIED PROGRAMMING

SEMESTER – VI

2021 – 2022 (EVEN)

LABORATORY MANUAL

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR – 603 203

CHENGALPATTU DISTRICT

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

18ECS301J – APPLIED PROGRAMMING

LABORATORY MANUAL

Course Team

Course Teacher(s)

Dr. M. S. Vasanthi

Dr. M. Susila

December 2021

Revision : 00

SRM Institute of Science and Technology

Department of Electronics and Communication Engineering

Vision of the Department

1. To create and disseminate knowledge in the area of Electronics and Communication Engineering through national and international accredited educational process.

2. To facilitate a unique learning and research experience to the students and faculty.

3. To prepare the students as highly ethical and competent professionals.

Mission of the Department

1. Build an educational process that is well suited to local needs as well as satisfies the national and international accreditation requirements.

2. Attract the qualified professionals and retain them by building an environment that fosters work freedom and empowerment.

3. With the right talent pool, create knowledge and disseminate, get involved in collaborative research with reputed universities and produce competent graduands.

Program Educational Objectives (PEO)

The Program Educational Objectives for the Electronics and Computer Engineering program describe accomplishments that graduates are expected to attain within a few years of graduation.

PEO-1: Expertise using their mathematical and scientific knowledge to solve emerging real-world problems, design and create novel products and solutions related to Electronics and Computer System Design that are technically sound, economically feasible and socially acceptable.

PEO-2: Broad knowledge to establish themselves as creative practicing professionals, locally and globally, in technical/managerial roles ranging from design, development, problem solving to production support in software industries and R&D sectors.

PEO-3: Communication skills (in both written and oral forms) and critical reasoning skills in bridging the divide between advanced technology and end users in the practice of Electronics and Computer Engineering.

PEO-4: Sustained learning and adapting to a constantly changing field through graduate work, professional development, self-study and collaborative activities.

PEO-5: Leadership and initiative to ethically advance professional and organizational goals, facilitate the achievements of others, and obtain substantive results.

PEO-6: Ability to work productively as individuals and in groups (teamwork) of diverse cultural and multidisciplinary backgrounds.

Course Learning Rationale (CLR):

The purpose of learning this course is to:

CLR-1: Recognize a powerful high level language that implements a deliberately clear syntax
CLR-2: Identify a highly coherent programming model
CLR-3: Apply knowledge on features of portability, productivity and extensive support libraries
CLR-4: Analyze the seamless integration with components coded in any other programming language
CLR-5: Employ the scientific computing skills
CLR-6: Prepare efficient software models.

Course Outcomes (CO):

At the end of this course, learners will be able to:

CO-1: Apply the basic and advanced features of core language built-ins
CO-2: Demonstrate control system/OS level features
CO-3: Compute software models for client-server communication
CO-4: Categorize using sockets, the client and server-side scripts
CO-5: Produce applications using database.
CO-6: Interpret the basic applications with database connectivity

Performance Criteria

The committee has decided that the following performance criteria (Key topics/questions) will be used for evaluation of CLR and CO pertaining to this course.

PROGRAM OUTCOMES (PO)

PO 1: Engineering Knowledge
PO 2: Problem Analysis
PO 3: Design & Development
PO 4:Analysis,Design,Research
PO 5:Modern Tool usage
PO 9:Individual &Teamwork
PO 10:Communication
PO11: Progrect Management & Finance
PO 12:Life long learning

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Design of Intelligent computing systems
PSO2: Project Management Techniques
PSO3: Adapting to changes in ICT

Course Articulation Matrix

18ECS301J – APPLIED PROGRAMMING		Program Outcomes												PSO		
Course Outcomes (CO): :		1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
CO-1	Apply the basic and advanced features of core language built- ins	*3*	-	-	-	*3*	-	-	-	-	-	*3*	-	*3*	-	-
CO-2	Demonstrate control system/OS level features	*2*	-	-	-	*3*	-	-	-	-	-	*3*	-	*3*	-	-
CO-3	Compute software models for client-server communication	*3*	-	-	-	*2*	-	-	-	-	-	*2*	-	*3*	-	-
CO-4	Categorize using sockets, the client and server-side scripts	*2*	-	-	-	*3*	-	-	-	-	-	*3*	-	*3*	-	-
CO-5	Produce applications using database.	*3*	-	-	-	*3*	-	-	-	-	-	*2*	-	*3*	-	-
CO-6	Interpret the basic applications with database connectivity	*3*	-	-	-	*2*	-	-	-	-	-	*3*	-	*3*	-	-

Session Plan

Learning Unit/ Module	Session	Description of Topic (Theory)	No. of Contact hours	CO	PO	BL	Reference
Introduction, Types and Operations	1	Introduction to Python, Python Interpreter and its working, Syntax and Semantics	1	1	1	1-4	[1]Part 1 – Chapter 1,2,3
	2	Data Types Assignment statements and Expression statements	1	1	1	1-4	[1] Part 1I– Chapter 4,11 Part III – Chapter 10
	3	Control Flow Statements Sequences	1	1	1	1-4	[1] Part 1I – Chapter 4
	4	Lists Dictionaries	1	1	1	1-4	[1] Part 1I – Chapter 4,8
	5	Tuples Files	1	1	1	1-4	[1] Part 1I– Chapter 4,9
	6	Functions Lambda expressions	1	1	1	1-4	[1] Part 1V – Chapter 16,19
Classes and OOPS	7	Iteration Protocol List Comprehensions	1	2	2	1-4	[1] Part 1II Chapter 14
	8	Handling text files	1	2	2	1-4	[1] Part V– Chapter 23,26

	9	Modules Class coding basics	1	2	2	1-4	[1] Part V– Chapter 23,26
	10	Designing with Classes Multiple Inheritance	1	2	2	1-4	[1] Part V– Chapter 30
	11	Exception Objects Designing with Exceptions	1	2	2	1-4	[1] Part 1II– Chapter 8
	12	Strings Regular Expressions	1	2	2	1-4	[1] Part 1– Chapter 7
Expectations and Tools	13	OS modules Lower-Level file tools in the OS module	1	3	3	1-4	[2] Part 1I– Chapter 5
	14	Sys modules Directory Traversal tools	1	3	3	1-4	[2] Part 1I– Chapter 4
	15	Parallel System tools: Forking processes Parallel System tools: thread module	1	3	3	1-4	[2] Part 1I– Chapter 5
	16	Running Multiple threads Synchronizing access to shared objects	1	3	3	1-4	[1] Part 1II– Chapter 10
	17	Parallel System tools : queue module Arguments Vs globals	1	3	3	1-4	[2] Part 1I– Chapter 5
	18	Program Exits-sys module exits, OS module exits, shell command exits Process exit status and shared state	1	3	3	1-4	[2] Part 1I– Chapter 5
Script Execution Tools	19	Introduction to Socket Programming Interprocess Communication	1	4	3	1-4	[2] Part IV- Chapter 12
	20	Handling Multiple Clients Client side scripting, urlib	1	4	3	1-4	[2] Part 1V– Chapter 13
	21	Server Side Scripting : CGI Script Running Server side	1	4	3	1-4	[2] Part 1V– Chapter 15
	22	Climbing the CGI Learning curve Passing parameters in Hardcoded URLs	1	4	3	1-4	[2] Part 1V– Chapter 15
	23	Saving state Information in CGI Scripts Refactoring Code for Maintainability	1	4	3	1-4	[2] Part 1V– Chapter 15
	24	URL and HTML Escapes Transferring Files to Clients and Servers	1	4	3	1-4	[2] Part 1V– Chapter 15
Graphical User Interface	25	Introduction to tkinter Top Level Windows	1	5, 6	2,3	1-4	[2] Part III– Chapter 8
	26	Dialogs, Message and Entry Event Handling, Menus	1	5,6	2,3	1-4	[2] Part III– Chapter 8
	27	Listboxes and Scrollbars, Text Canvas, Grids	1	5,6	2,3	1-4	[2] Part III– Chapter 9
	28	SQL Database interfaces with sqlite3 : Basic operations SQLite Tools	1	5,6	2,3	1-4	[2] Part V– Chapter 17
	29	SQLite functions SQLite Interfaces	1	5,6	2,3	1-4	[1] Part VI– Chapter 25
	30	Object Relational mappers Table load scripts	1	5,6	2,3	1-4	[1] Part VI– Chapter 24

Practical Session:

List of Experiments

SI. No	Description of Experiments	Contact hours	Reference
1	Solving linear equations- Least squares method	2	5
2	Solving linear equations- Least squares method using Lists	2	5
3	Solving linear equations- Least squares method using functions	2	5
4	Simulating in time- Differentiator	2	5
5	Simulating in time- Integrator	2	5
6	Simulating in time – Square wave generator	2	5
7	Simulating a device- Bistable multivibrator	2	5
8	Simulating a device- Monostable multivibrator	2	5
9	Simulating a device – Astable multivibrator	2	5
10	Using the system module to solve for step and impulse response of op-amp circuits	2	5
11	Using the DFT to obtain steady state response of linear (and op-amp) circuits	2	5
12	Simulating Noise in Circuits	2	5
13	Low pass filtering of signals using digital filters	2	5
14	High pass filtering of signals using digital filters	2	5
15	Effect on SNR	2	5

References

1. Mark Lutz ,"Learning Python", O Reily, 4thEdition, 2009, ISBN: 978-0-596-15806-42.
2. Mark Lutz ,"Programming Python ", O Reily, 4thEdition, 2010, ISBN 97805961581183
3. Tim Hall and J-P Stacey ,"Python 3 for Absolute Beginners" , 2009, ISBN:97814302163224.
4. Magnus Lie Hetland , "Beginning Python: From Novice to Professional", 2ndEdition, 2009, ISBN:9781590599822
5. Lab Manual

Assessment

Continuous Internal Evaluation (50% Weightage)								Semester End Examination (50% Weightage)	
CLA -1		CLA -2		CLA -3		CLA -4			
Theory	Practical	Theory	Practical	Theory	Practical	Theory	Practical	Theory	Practical
5%	5%	7.5%	7.5%	7.5%	7.5%	5%	5%	25%	25%

PREFACE

The **18ECS301J – Applied Programming lab** is designed to help students understand the basic principles of communication techniques as well as giving them the insight on how to use Python program for design and simulation of electronic circuits and system. The main aim is to provide programming experience to the students so that they are able to put theoretical concepts to practice through programming.

The experiments are designed in such a way students will simulate through program the basic linear equations, simulation with respect to time, devices, system modules and generation of noise and its effect in the communication channel. Open Source Software tool " Python (Jupyter Notebook) and Google CoLab is utilized.

Students will program the experiments in an interactive way and using data visualization tools for plotting graphs. Students are exposed to various ways of writing the programs through different functions and modules. Students will have the exposure to program circuit-based simulations and system modules.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

College of Engineering and Technology

Department of Electronics and Communication Engineering

Course Code :18ECS301J **Semester: V**

Course Title :Applied Programming **Course Duration: Jan- May '22**

Pre- requisite : 18CSC101J **Co-requisite: NIL**

Program Outcomes

PO1: Engineering Knowledge & PO5: Modern Tool Usage

1 Solving linear equations- Least squares method

2 Solving linear equations- Least squares method using Lists

3 Solving linear equations- Least squares method using functions

4 Simulating in time- Differentiator

5 Simulating in time- Integrator

6 Simulating in time – Square wave generator

7 Simulating a device- Bistable multivibrator

8 Simulating a device- Monostable multivibrator

9 Simulating a device – Astable multivibrator

10 Using the system module to solve for step and impulse response of op-amp circuits

11 Using the DFT to obtain steady state response of linear (op-amp) circuits

12 Simulating noise in circuits

13 Low pass filtering of signals using digital filters

14 High pass filtering of signals using digital filters

15 Effect on SNR

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

College of Engineering and Technology

Department of Electronics and Communication Engineering

Course Code :18ECS301J **Semester: V**

Course Title :Applied Programming **Course Duration: Jan- May '22**

Pre- requisite : 18CSC101J **Co-requisite: NIL**

Mapping of Course Outcomes with Experiments

CO1: Apply the basic and advanced features of core language built-ins
CO2: Demonstrate control system/OS level features
CO3: Compute software models for client-server communication
CO4: Categorize using sockets, the client and server-side scripts
CO5: Produce applications using database.
CO6: Interpret the basic applications with database connectivity

List of Experiments	CO1	CO2	CO3	CO4	CO5	CO6
Solving linear equations- Least squares method	*					*
Solving linear equations- Least squares method using Lists	*					*
Solving linear equations- Least squares method using functions	*					*
Simulating in time- Differentiator		*				*
Simulating in time- Integrator		*				*
Simulating in time – Square wave generator		*				*
Simulating a device- Bistable multivibrator			*			*
Simulating a device- Monostable multivibrator			*			*
Simulating a device – Astable multivibrator			*			*
Using the system module to solve for step and impulse response of op-amp circuits				*		*
Using the DFT to obtain steady state response of linear (and op-amp) circuits				*		*
Simulating Noise in Circuits				*		*
Low pass filtering of signals using digital filters					*	*
High pass filtering of signals using digital filters					*	*
Effect on SNR					*	*

LABORATORY ORIENTATION

I. OVERALL PURPOSE

The laboratory component of this course is designed to give the student coding experience in in Python for applications related to electronics and communication. The theory cum lab component helps the students to learn and experience the course content.

II. GENERAL COMMENTS

One week before each lab class, students are instructed about the purpose and outcome of doing the experiment. Students are informed to prepare about the basics, functions and operation of the experiment. The student can refer to the text book and other resources as prescribed in the course description for the fundamental theory.

Your grade will reflect how well you have prepared for the lab.

III LABORATORY AND SYSTEM MAINTENANCE

Student should take the responsibility of properly switching on during the lab session and shut down the system properly before leaving the lab. Students should take the responsibility not download any software without the knowledge of the Lab In-charges. Students should enter their details and the duration of their working in the log book maintained in the lab.

IV. LABORATORY NOTEBOOK

Each student should maintain a laboratory (Observation) notebook according to the following guidelines:

1. Obtain a printed material whose pages are consecutively numbered.
2. Details such as name, register number, course code and title, semester, class & section, lab venue, and Faculty Incharge on the cover sheet.
3. All data should be recorded by **pen only.**
4. Label the axes of a graph with variable names, units, origin, and scales.
5. Demonstrate to the lab staff your understanding of the experiment objectives.
6. Once the experiment is successfully done, the data and output should get attested by the faculty. It is the responsibility of both the staff and the student the experiment is done successfully with any error.

V. PRE-LAB WORK

There are pre-lab works for each experiment. The pre work must be completed in the laboratory (observation) notebook *before* entering the laboratory. The pre work usually consists of some questions that is closely related to the fundamentally and understanding of experimental work, that is intended to prepare students for the lab.

VI. LABORATORY REPORTS

Lab reports are maintained by individual students to note done their findings and results that they carry out during each program/ experiment. The same will be verified by the course teachers for its correctness. The verified reports will be submitted in the next lab session for evaluation for the experiment. The report will be evaluated and marks will be awarded based on clarity, legibility, results and deliverables.

LABORATORY POLICIES AND REPORT FORMAT:

Each experiment should have the following contents but not limited to.

Title Page (Evaluation Sheet):

Date:

Experiment # & Name

Report Content:

I. Objective

II Theory.)

- include pertinent theory for experiment

III. Algorithm

IV. Program

V. PreLab Questions

V. Postlab Questions

Answering all the prelab and post questions will help the students to gain fundamental knowledge as well as able to analyse the experiment in different perspective.

VI. Result and Conclusion

Summarize results in the introductory sentence. Relate results to objective. Present the results in the easiest way for reader to understand

Laboratory Report Cover Sheet

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
College of Engineering and Techn
ology
Department of Electronics and Communication Engineering

18ECS301J – Applied Programming Lab

VI Semester, 2021 - 22 (Even Semester)

Name :

Register No. :

Class :

Venue :

Title of Experiment :

Date of Conduction :

Date of Submission :

Particulars	Max. Marks	Marks Obtained
Pre Lab	05	
Lab Performance	10	
Post Lab	10	
Record Submission	5	
Total	**30**	

REPORT VERIFICATION

Staff Name :

Signature :

TABLE OF CONTENTS

SI. No.	Title of the Experiment	Page No.
1	Solving linear equations- Least squares method	15
2	Solving linear equations- Least squares method using Lists	17
3	Solving linear equations- Least squares method using functions	23
4	Simulating in time- Differentiator	25
5	Simulating in time- Integrator	28
6	Simulating in time – Square wave generator	31
7	Simulating a device- Astable multivibrator	34
8	Simulating a device- Monostable multivibrator	38
9	Low pass filtering of signals using digital filters	45
10	High pass filtering of signals using digital filters - Step Response and Impulse Response	53
11	Simulating Additive White Gaussian Noise	60

EXPERIMENT #1 Solving Linear Equations

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

1	Solving Linear equations efficiently

Aim

Write python code to solve the following linear equation efficiently:

1. 2x+3=x-6

2. 2x +3-4x=8(x+6)

3. 5y+0.2(23y/5-2)+15 = 0.4y

Theory:

Linear Equation: It must be solvable in its pure form. This means that it can only have one variable, usually written as x. A two-variable equation would require multiple linear equations (a system of equations) to be solved. A linear equation consists of three primary components — constants, variables, and multipliers. Any number or combination of operations — addition, subtraction, multiplication, and division — are valid, with any parenthesis scope. As long as it abides by these definitions of a linear equation, it is solvable by our function.

- expression = equation.replace("="," - (")+")"

  This transforms the equation into an expression to be evaluated by moving the entire expression on the right side of the equation to the left side.

- grouped = eval(expression.replace(var,'1j')

  The eval function takes in an expression. By substituting an unknown variable x with the natively understood j (i), Python treats two categories of expression elements — variables and constants — as separate. When the expression is evaluated, the answer comes out to be a*j + b, which Python believes is a complex number. Because j was used as a substitute for x, the result is a simplified and easily solvable linear equation.

- return -grouped.real/grouped.imag

  returns the negative of the real component of the complex number divided by the imaginary component.

Example: **2x+3 = x-6**

2x+3 – (x-6) =0

x=-9

Program and Input Data

1a.

```
# Expt 1a: Solving Linear equations
# Class: III Year EKE
# Course: 18ECS301J – Applied Programming Lab
# Student Name and Reg Number:

from math import *
def solve_linear(equation,var='x'):
    expression = equation.replace("=","-(")+")"
    grouped = eval(expression.replace(var,'1j'))
    return -grouped.real/grouped.imag
solve_linear("2*x+3=x-6")
```

1b.

```
# Expt 1b: Solving Linear equations
# Class: III Year EKE
# Course: 18ECS301J – Applied Programming Lab
# Student Name and Reg Number

from math import *
def solve_linear(equation,var='x'):
    expression = equation.replace("=","-(")+")"
    grouped = eval(expression.replace(var,'1j'))
    return -grouped.real/grouped.imag
solve_linear("2*x +3-4*x=8*(x+6)")
```

1c.

```
# Expt 1c: Solving Linear equations
# Class: III Year EKE
# Course: 18ECS301J – Applied Programming Lab
# Student Name and Reg Number:

from math import *
def solve_linear(equation,var='x'):
    expression = equation.replace("=","-(")+")"
    grouped = eval(expression.replace(var,'1j'))
    return -grouped.real/grouped.imag
solve_linear("5*y+0.2*(23*y/5-2)+15 = 0.4*y”)
```

Pre lab Questions:

1. What is Python?
2. What is an interpreted Language
3. What are the supported data types in python

Post Lab Questions:

1. Name three features of python.
2. What is Jupyter Notebook?
3. Name the Python IDEs?

Results and Conclusion

Solved linear equations efficiently with python.

EXPERIMENT#2 Solving Polynomial Equation Using List

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

2	**Compute a polynomial equation given that the coefficients of the polynomial are stored in a List.**

Program and Input Data

```
# i.  Evaluate value of  px=-1+2x+6x2+2x3    for x = 3

# 2x3 - 6x2 + 2x - 1 for x = 3
poly = [2, -6, 2, -1]
x = 3
n = len(poly)

# Declaring the result
result = 0

# Running a for loop to traverse through the list
for i in range(n):

# Declaring the variable Sum
Sum = poly[i]

# Running a for loop to multiply x (n-i-1)
# times to the current coefficient
for j in range(n - i - 1):
Sum = Sum * x

# Adding the sum to the result
result = result + Sum

# Printing the result
print(result)
```

SCREEN SHOT:

2a(i). Evaluate value of px=-1+2x+6x2+2x3 for x = 3

In [1]:
```
# 2a (i): 2x3 - 6x2 + 2x - 1 for x = 3
poly = [2, -6, 2, -1]
x = 3
n = len(poly)

# Declaring the result
result = 0

# Running a for loop to traverse through the list
for i in range(n):

    # Declaring the variable Sum
    Sum = poly[i]

    # Running a for loop to multiply x (n-i-1)
    # times to the current coefficient
    for j in range(n - i - 1):
        Sum = Sum * x

    # Adding the sum to the result
    result = result + Sum

# Printing the result
print(result)
```

```
5
```

2a (ii) Evaluate value of px=-1+x2+3x7 for x = 3

In [6]:
```
#2a
#(ii) Evaluate value of  px=-1+x2+3x7    for x = 3

# 3x7 +x2 - 1 for x = 3

poly = [3,0,0,0,0,1,0, -1]
x = 3
n = len(poly)

# Declaring the result
result = 0

# Running a for loop to traverse through the list
for i in range(n):

    # Declaring the variable Sum
    Sum = poly[i]

    # Running a for loop to multiply x (n-i-1)
    # times to the current coefficient
    for j in range(n - i - 1):
        Sum = Sum * x

    # Adding the sum to the result
    result = result + Sum

# Printing the result
print(result)
```

```
6569
```

2b	**Solving polynomial expressions with dictionaries**

Aim / Objective. To write a function to solve the following polynomial expressions for x=2 (i) px=-1+x2+3x7 and (ii) px=-4+2x+5x2+8x3 with dictionaries and *perform the program in Jupyter Notebook.*

Program and Input Data

$$p(x) = -1 + x^2 + 3x^7 .$$

```
p = {0: -1, 2: 1, 7: 3}
```

```
p
```

```
{0: -1, 2: 1, 7: 3}
```

```
def eval_poly_dict(poly, x):
    sum = 0.0
    for power in poly:
        sum += poly[power]*x**power
    return sum
eval_poly_dict(p,2)
```

```
387.0
```

Screen shot:

2b

(i) px=-1+x2+3x7

```
In [4]: # 2b(i) px=-1+x2+3x7 , x=2
        p = {0: -1,2: 1,7: 3}
        def eval_poly_dict(poly, x):
            sum = 0.0
            for power in poly:
                sum += poly[power]*x**power
            return sum
        eval_poly_dict(p,2)

Out[4]: 387.0
```

2b

(ii) px=-4+2x+5x2+8x3

```
In [5]: # 2b(ii) px=-4+2x+5x2+8x3 , x=2
        p = {0: -4,1 :2,2: 5,3: 8}
        def eval_poly_dict(poly, x):
            sum = 0.0
            for power in poly:
                sum += poly[power]*x**power
            return sum
        eval_poly_dict(p,2)

Out[5]: 84.0
```

PRE-LAB

1. dict={"name": "Plato", "country": "Ancient Greece", "born": -427, "teacher": "Socrates", "student": "Aristotle"}

   (i)When was Plato born?

   (ii) Who was Plato student?

   (iii) Change Plato's birth year from B.C. 427 to B.C. 428.

2. dict={"son's name": "Lucas", "son's eyes": "green", "son's height": 32, "son's weight":25}

   (i) Add 2 inches to the son's height.

   (ii) Using .get() method print the value of "son's eyes".

3. Write a for loop which appends the type of each element in the first list to the second list.

POST LAB:

1. Change the value of a specific item by referring to its key name:

   Change the “year” to 2020 in a given program:

   thisdict ={ "brand": "Ford", "model": "Mustang", "year": 1972}

2. Using a for loop and .append() method, append each item with a Dr. prefix to the lst.

   Lst=[‘Varun’,’Sanjay’,’Ravi’]

Results and Conclusion

Found the result of the given polynomial with list and dictionary using python program.

EXPERIMENT #3 Roots of Quadratic Equation

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

3	**Write a python program to compute the roots of a quadratic equation**

Aim / Objective. To write a python function to compute the roots of the following quadratic expressions (i) $2x^2 + x + 1 = 0$, (ii) $x^2 + 5x + 6 = 0$ and (iii) $4x^2 + 12x + 9 = 0$ and *simulate the program in Jupyter Notebook.*

Theory:
Standard form of quadratic equation is $ax^2 + bx + c = 0$
where, a, b, and c are coefficient and real numbers and also $a \neq 0$. If a is equal to 0 that equation is not valid quadratic equation.
Using the below quadratic formula we can find the root of the quadratic equation.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

There are following important cases.
If $b^2 < 4ac$, then roots are complex , (not real).
If $b^2 > 4ac$, then roots are real and different
If $b^2 = 4ac$, then roots are real and same

Program and Input Data

 # Give all the comment statements

```
# Python program to find roots of quadratic equation
import math

# function for finding roots
def anyname( a, b, c):

    # calculating discriminant using formula
    dis = b * * b - 4 * a * c
    sqrt_val = math.sqrt(abs(dis))
```

```
 # Checking the nature of roots
    if dis > 0:
        print(" real and different roots ")
        print((-b + sqrt_val)/(2 * a))
        print((-b - sqrt_val)/(2 * a))

    elif dis == 0:
        print(" real and same roots")
        print(-b / (2 * a))

    # when discriminant is less than 0
    else:
        print("Complex Roots")
        print(- b / (2 * a), " + i", sqrt_val)
        print(- b / (2 * a), " - i", sqrt_val)

# Driver Program to give the inputs of quadratic equations
# To check for valid quadratic equation
if a == 0:
    print("Given equation is not a valid quadratic equation")
else:
    anyname(a, b, c)
```

PRE-LAB

1. Write a function named "evens" which returns True if a number is even and otherwise returns False
2. Which is the keyword used to define a function in python.
3. Which of the following function headers is correct? Explain
   (i) def fun(a = 2, b = 3, c)
   (ii) def fun(a = 2, b, c = 3)
   (iii) def fun(a, b = 2, c = 3)
   (iv) def fun(a, b, c = 3, d)

POST LAB:

1. What will the following python function give as output if input argument is a string

```
def count_l(a):
    c = 0
    for i in a.split():
        if "e" in i:
            c = c+1
        else:
            pass
    return c
print(count_l(str))
```

2. What is a recursive function? Give an example

Results and Conclusion

Solved the given quadratic equation efficiently with python.

(i) $2x^2 + x + 1 = 0$; (ii) $x^2 + 5x + 6 = 0$ and (iii) $4x^2 + 12x + 9 = 0$

EXPERIMENT#4 Compute Derivative of an Equation

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

4	**Write a python program to compute and evaluate the derivative of an equation**

Aim / Objective.

To write a python function to compute the derivative of the following expressions (i) 2x2+x+1 (ii) x2+5x+6 and (iii) 4x2+12x+9 and evaluate the derivative for x=4. S*imulate the program in Jupyter Notebook.*

Theory:

Standard form of quadratic equation is $ax^2 + bx + c$, where, a, b, and c are coefficient. The derivate with respect to x will be 2ax+b.

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python. SymPy only depends on mpmath, a pure Python library for arbitrary floating point arithmetic, making it easy to use.

Derivatives are the Fundamental tools of Calculus. It is very useful for optimizing a loss function using **sympy.Derivative()** method, we can create an unevaluated derivative of a SymPy expression. It has the same syntax as **diff()** method. To evaluate an unevaluated derivative, use the **doit()** method.

Syntax: *Derivative(expression, reference variable)*

Parameters:

expression *– A SymPy expression whose unevaluated derivative is found.*

reference variable – *Variable with respect to which derivative is found.*

Returns: *Returns an unevaluated derivative of the given expression.*

Program and Input Data

 # Students are encouraged to write the program through understanding the concept and by checking the correctness of the output.

 # Explore different methods/syntax to write the program, , no restrictions to stick on to this program.

Example Program 1

 # Give all the comment statements

 # Python program to compute the derivative of the equation

```
# import sympy for evaluating
from sympy import *

x, y = symbols('x y')
expr = x**2 + 2 * y + y**3
print("Expression : {} ".format(expr))

# Use sympy.Derivative() method
expr_diff = Derivative(expr, x)

print("Derivative of expression with respect to x : {}".format(expr_diff))
print("Value of the derivative : {} ".format(expr_diff.doit()))
```

Output:

```
Expression : x**2 + y**3 + 2*y
Derivative of expression with respect to x : Derivative(x**2 + y**3 + 2*y, x)
Value of the derivative : 2*x
```

In [2]:

```
# Expt 4 PRG 1 :program to compute the derivative of the equation
# Class: III Year EKE
# Course: 18ECS301J - Applied Programming Lab
# Student Name and Reg Number: Merrill_054
# Python program to compute the derivative of the equation
# import sympy for evaluating

from sympy import *

x, y = symbols('x y')
expr = x**2 + 2 * y + y**3
print("Expression : {} ".format(expr))

# Use sympy.Derivative() method
expr_diff = Derivative(expr, x)

print("Derivative of expression with respect to x : {}".format(expr_diff))
print("Value of the derivative : {} ".format(expr_diff.doit()))
```

```
Expression : x**2 + y**3 + 2*y
Derivative of expression with respect to x : Derivative(x**2 + y**3 + 2*y, x)
Value of the derivative : 2*x
```

Example Program 2 - Solving a differential with Sympy diff()

```
#Importing sympy
 from sympy import *
 # create a "symbol" called x
x = Symbol('x')
#Define function
f = x**2
#Calculating Derivative
```

derivative_f = f.diff(x)
derivative_f

In [5]:
```
# Expt 4 PRG 2:program to compute the derivative of the equation
# Class: III Year EKE
# Course: 18ECS301J - Applied Programming Lab
# Student Name and Reg Number: Merrill_054

#import sympy
from sympy import *
 # create a "symbol" called x
x = Symbol('x')
#Define function
f = x**2
#Calculating Derivative
derivative_f = f.diff(x)
derivative_f
```

Out[5]: $2x$

Example Program 3 – Solving derivatives in Python
 # Calculate the derivative of the function at x = 2 sympy has lambdify function in which we pass the symbol and the function.

```
from sympy import *

# create a "symbol" called x
x = Symbol('x')
#Define function
f = x**2
f1 = lambdify(x, f)
#passing x=2 to the function
f1(2)
```

PRE-LAB

1. Write a program using sympy.diff() to find the differentiation of sinx + cosx with respect to x.
2. Mention three features of Sympy module.

POST LAB:

1. What is a derivative.
2. Mention any two basic derivative rules in python Sympy

Results and Conclusion

Executed a python program to compute the derivative of the following expressions and simulated the program in Jupyter notebook.

EXPERIMENT#5 Compute Integral of an Equation

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

5	**Write a python program to compute and evaluate the integral of an equation**

Aim / Objective. To write a python function to integrate the following expressions

(i) $\int_0^2 3x^2 + 1 \; dx$

(ii) $\int_0^{2\pi} e^{-x} \sin\,(3x)\, dx$

S*imulate the program in Jupyter Notebook.*

Theory:

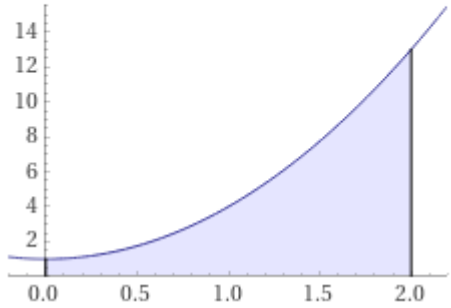
Standard form of quadratic equation is $ax^2 + bx + c$ where, a, b, and c are coefficient.

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python. SymPy only depends on mpmath, a pure Python library for arbitrary floating point arithmetic, making it easy to use.

Example:

$$\int_0^2 (3x^2 + 1)\, dx = 10$$

Visual representation of the integral



Program and Input Data

 # Students are encouraged to write the program through understanding the concept and by checking the correctness of the output.

 # Explore different methods/syntax to write the program, , no restrictions to stick on to this program.

Program 1: $\int_0^2 3x^2 + 1 \; dx$

(i) Using Sympy module

 # Give all the comment statements

 # Python program to compute the integral of the equation

```
# import sympy for evaluating

import sympy as sp
x= sp.Symbol ('x')
Expression=3.0*x**2+1
print(Expression)
 print(sp.integrate(Expression,x))
```

Output:

3.0*x**2+1

x^3+x

(ii) Using scipy module

```
from scipy.integrate import quad
def f(x):
   return 3.0*x**2+1
i=quad(f,0,2)
print(i[0])
print(i[1])
```

Output:

10.000000000000002
1.1102230246251568e-13

Progam 2: $\int_0^{2\pi} e^{-x} \sin(3x)\, dx$

i) ***Using Sympy module***

```
# import sympy for evaluating
#i
import sympy as sp
from sympy import Symbol,integrate,exp,pprint,sin,pi
x= sp.Symbol ('x')
Expression= exp(-x)*((sin(3*x)))
print(Expression)
print(sp.integrate(Expression,x))
```

```
exp(-x)*sin(3*x)
-exp(-x)*sin(3*x)/10 - 3*exp(-x)*cos(3*x)/10
```

ii) **Using scipy module**

```
#(ii) Using scipy module
from sympy import Symbol,integrate,exp,pprint,sin,pi
from scipy.integrate import quad
import math
x=sp.Symbol('x')
def f(x):
    return exp(-x)*((sin(3*x)))
i=quad(f,0,2*pi)
print(i[0])
print(i[1])
```

```
0.29943976718048754
5.050152988254293e-13
```

PRE-LAB

1. Mention any four methods for integrating functions in scipy.integrate module
2. Mention features of the quad function and give the syntax.

POST LAB:

1. How analytical Integration is different from Numerical integration
2. Discuss the error occurring when performing numerical integration

Results and Conclusion

Executed a python program to compute and evaluate the integral of an equation and simulated the program in Jupyter notebook.

EXPERIMENT #6 Square Wave Generation

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**		
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

Title of the Experiment

6	**Write a python program to generate a square wave and plot the same.**

Aim / Objective. To write a python function to generate a square wave for the following parameters where frequency is given by **f**, amplitude by **A** and duty cycle by **d**. (i) f=2; A = 5, d=0.5 (50% duty cycle) and (ii) f=5; A = 5, d=0.75 (75% duty cycle). S*imulate the program in Jupyter Notebook.*

Theory:

Square waves are period waveforms. However, Square waves are non-sinusoidal. The transition between the peak values is instantaneous in a square wave. The period of the square wave is also called the pulse width.

To draw a square wave using **matplotlib, scipy** and **numpy** following details are required

- Frequency of the square wave (eq. 10 Hz ; i.e.10 cycles per second)
- Amplitude of the square wave.
- Square waves have a duty cycle of 50%. That is, the percentage of the waveform that occurs above zero axes is 50% for a square wave.

By default the **signal.square()** function of **scipy** takes a duty value of **0.5.**

Square wave can be generated using 'square' tool from the scipy/signal library. The arguments Amplitude and frequency will define square wave and the optional argument 'duty' defines which fraction of the whole duty cycle the signal will be in its 'high' state. The plotting commands will help to label the figure and the command 'figsize', defines the size of the figure.

Program and Input Data

 # Students are encouraged to write the program through understanding the concept and by checking the correctness of the output.

 # Explore different methods/syntax to write the program, no restrictions to stick on to this program.

Program and Input Data

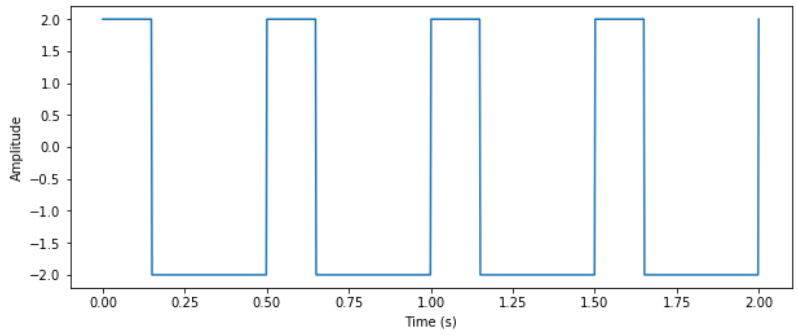
 # Students are encouraged to write the program through understanding the concept and by checking the correctness of the output.

 # Explore different methods/syntax to write the program, no restrictions to stick on to this program.

Program 1

```
# Python program to generate Square Wave
# First, call the necessary Python libraries

import numpy as np
from scipy import signal as sg
import matplotlib.pyplot as plt

# Use input statements to get the inputs from the user
freq = 2
amp = 2
time = np.linspace(0, 2, 1000)

# Plotting of Square Wave with duty cycle 30%
sqsignal = amp*sg.square(2*np.pi*freq*time, duty=0.3)

# Plotting the wave within a window or frame of specified size
plt.figure(figsize=(10,4))

# Labelling the figure
plt.plot(time, sqsignal)
plt.xlabel('Time (s)')  # Give x axis label for the square wave plot
plt.ylabel('Amplitude')  # Give y axis label for the square wave plot
plt.show() # Display the square wave
```

Output:

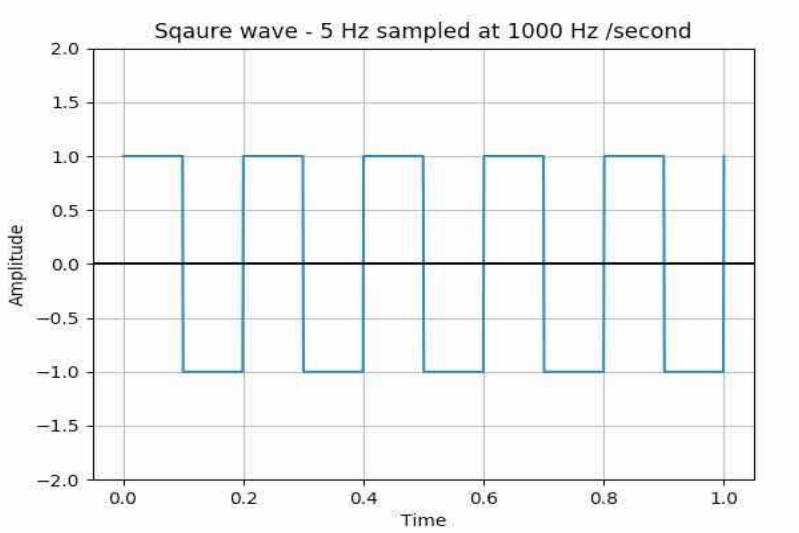


Program 2

```
# Program to plot Square Wave
from scipy import signal
import matplotlib.pyplot as plot
import numpy as np

# Sampling rate 1000 hz / second
t = np.linspace(0, 1, 1000, endpoint=True)
```

```
# Plot the square wave signal
plot.plot(t, signal.square(2 * np.pi * 5 * t))

# Give a title and labels for the square wave plot
plot.title('Sqaure wave - 5 Hz sampled at 1000 Hz /second')
plot.xlabel('Time')
plot.ylabel('Amplitude')
plot.grid(True, which='both')

# Provide x axis and line color
plot.axhline(y=0, color='k')

# Set the max and min values for y axis
plot.ylim(-2, 2)

# Display the square wave
plot.show()
```

Output:



PRE-LAB

1. Write the syntax to define the size of the figure while plotting.
2. Write the command to display the grid lines in the figure.

POST LAB:

1. Write a program to generate and plot a sine wave with frequency =10, amplitude = 5.

2. Write a program to generate triangular wave with amplitude=2 and frequency = 5.

Result:

Thus, Square wave is generate and plotted using Python program.

EXPERIMENT#7 Astable Multivibrator

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Teacher :					

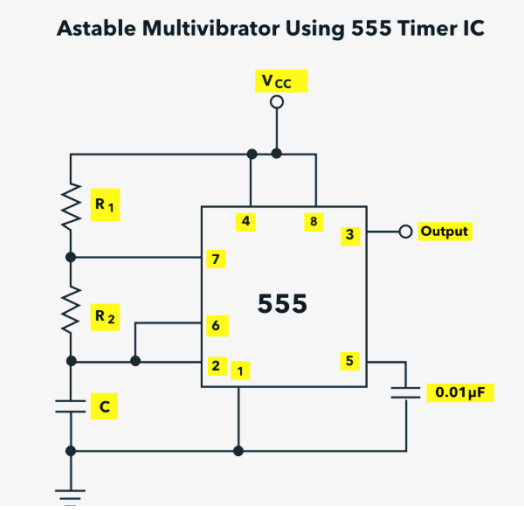
Title of the Experiment

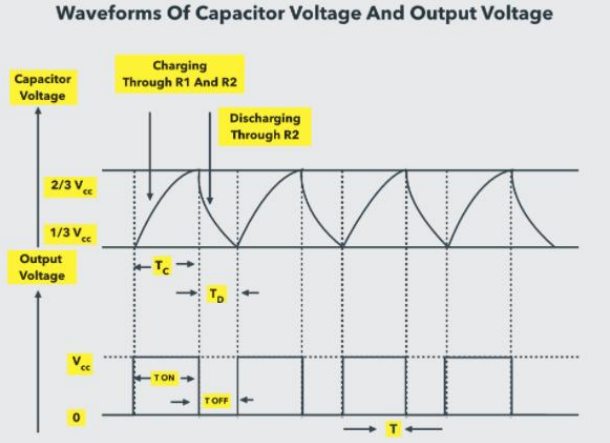
7	**Write a python program to design an Astable Multivibrator and plot the output voltages**

Aim / Objective: To write a python code to design an Astable multivibrator for frequency (i) f=35 Hz (ii) 70 Hz and plot the output voltage and capacitor voltage. S*imulate the program in Jupyter Notebook.*

Theory: **Astable Multivibrator Mode of 555 Timer IC Circuit**

Astable multivibrator is also called as Free Running Multivibrator. It has no stable states and continuously switches between the two states without application of any external trigger. The IC 555 can be made to work as an astable multivibrator with the addition of three external components: two resistors (R1 and R2) and a capacitor (C). The schematic of the IC 555 as an astable multivibrator along with the three external components is shown below.





Design

The frequency of oscillations in Astable Multivibrator is given by f= 1.44(R1+2R2)*C

$T_{OFF} = T_D = 0.693 * R_2C$

$T_{ON} = T_C = 0.693 * (R_1 + R_2) C$

$T = T_{ON} + T_{OFF}$

%Duty Cycle = $(T_{ON} / T)*100$

Typical values

C_1	R_2 = 10KΩ R_1 = 1KΩ	R_2 = 100KΩ R_1 = 10KΩ	R_2 = 1MΩ R_1 = 100KΩ
0.001µF (102)	68 KHz	6.8 KHz	680 Hz
0.01µF (103)	6.8 KHz	680 Hz	68 Hz
0.1µF (104)	680 Hz	68 Hz	6.8 Hz
1µF	68 Hz	6.8 Hz	0.68 Hz
10µF	6.8 Hz	0.68 Hz (41 per min.)	0.068 Hz (4 per min.)

Selection of R1, R2, C.

Given f= 35Hz, Let C = 2 µF,

R_1 and R_2 should be in the range 1KΩ to 1MΩ.

$R_2 = 0.693 /(f * C)$ implies $R_2 = 10K\ \Omega$

Choose R_1 to be about a tenth of R_2 (1KΩ min.)

Therefore, $R_1 = 1\ K\ \Omega$

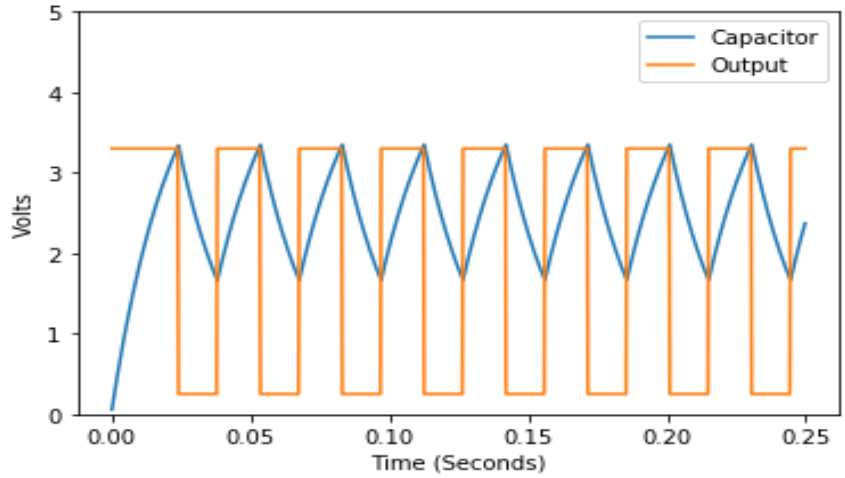
Program and Input Data

#Implemeting and plotting the output voltages of the ASTABLE MULTIVIBRATOR

```
import math
import matplotlib.pyplot as plt
vin = 5.0                 # Voltage in

out_high = 3.3            # Voltage output from pin 3 of  IC 555 when "high"
out_low  = 0.25           # Voltage output from pin 3 of IC 555 when "low"

farads = 2/1000000        # Capacitance of primary capacitor, which connects both the
                          # threshold (pin 6) and trigger (pin 2) to ground.

ohms_R1 = 1000            # Resistance in ohms of resistor R1, which connects the
                          # discharge (pin 7) and resistor to positive voltage.
                          # The primary capacitor charges through this resistor,
                          # but does not discharge through it.

ohms_R2 = 10000           # Resistance in ohms of resistor R2, which both discharges
                          # the capacitor through pin 7 and charges it from the end
                          # of resistor A. Connects to the threshold (pin 6), trigger
                          # (pin 2) and the primary capacitor.

sim_time = 0.25                 # Length of time to run the simulation (in seconds)
step = sim_time / 1000          # Time increment for each simulation step (in seconds)

#### Utility functions ##########################################################
def update_latch(latch, trigger, threshold, vin):
    if trigger < vin / 3:
      latch = True
  if threshold > vin * 2 / 3:
      latch = False
  return latch
```

```
def update_capacitor(cap, farads, res_1, res_2, vin, latch, step):
     if latch:
       tc = farads * (res_1 + res_2)       # Time proportion
       tp = step / tc                 #Percent change from existing voltage to full voltage (vin)
       pc = 1 - (1/math.e**tp)#e-t
       cap =cap+ (vin - cap) * pc
   else:
       tc = farads * res_2                 # Time proportion
       tp = step / tc                      # Percent change from existing voltage down to zero
       pc = 1 - (1/math.e**tp)
       cap =cap- cap * pc
 return cap

####  Setup ################################################################
time = 0
cap = 0
latch = True
vout = out_high
# Vectors to store simulated states
cap_list = []
out_list = []
time_list = []

####  Simulate  ############################################################
while time < sim_time:
   cap = update_capacitor(cap, farads, ohms_R1, ohms_R2, vin, latch, step)
   latch = update_latch(latch, cap, cap, vin)
   if latch:
      vout = out_high
   else:
      vout = out_low
   cap_list.append(cap)
   out_list.append(vout)
   time_list.append(time)
   time =time+ step

####  Plot  ################################################################
plt.plot(time_list, cap_list, label='Capacitor')          # Capacitor voltages
plt.plot(time_list, out_list, label='Output')             # Output (Q) voltages
plt.xlabel('Time (Seconds)')
plt.ylabel('Volts')
plt.ylim(0, vin)
plt.legend(loc='upper right')
plt.show()
```

OUTPUT



PRE-LAB

1. An Astable multivibrator has _____ states
2. _____ is used as a- pulse position modulation, frequency modulation
3. What is the duty cycle of the output of an astable multivibrator?
   a) 50%
   b) 100%
   c) 75%
   d) 55%
   Ans: 50%

POST LAB

1. How can the duty cycle be changed for an astable multivibrator?
   a) By adding another capacitor to the circuit
   b) By adding diodes to the circuit
   c) By adding an inductor to the circuit
   d) The duty cycle cannot be changed
2. A multivibrator operating at 150Hz has a discharge time of 2.5ms. Find the duty cycle?
   a) 50%
   b) 95.99%
   c) 75%
   d) 37.5%

Results and Conclusion

Executed a python program to design an Astable Multivibrator and plot the output voltages for the given parameters and simulated the program in Jupyter notebook.

EXPERIMENT#8 Monostable Multivibrator

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Instructor:					

Title of the Experiment

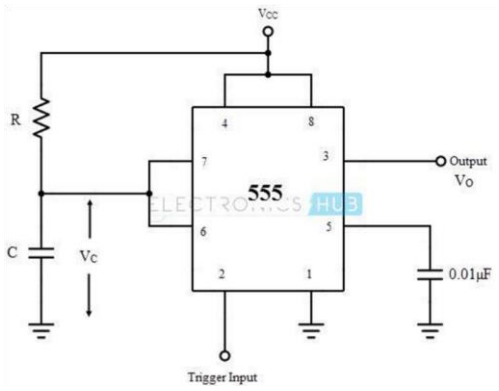
8	**Write a python program to design an Monostable Multvibrator and plot the output voltages**

Aim / Objective: To write a python code to design an Monostable multivibrator required to produce a minimum output time delay of (i) 500ms (ii) 250ms and plot the output voltage and capacitor voltage. S*imulate the program in Jupyter Notebook.*

Theory: **Monostable Multivibrator using 555 Timer**

A monostable multivibrator has only one stable state. When a trigger input is applied, a pulse is produced at the output and returns back to the stable state after a time interval. The duration of time for which the pulse is high will depend on the timing circuit that comprises of a resistor (R) and a capacitor (C). The details of the connection are as follows. The pins 1 and 8 are connected to ground and supply (V_{CC}) respectively. Output is taken at pin 3. To avoid accidental reset of the circuit, pin 4 is connected to the V_{CC}. Pin 5, which is the control voltage input, should be grounded when not in use. To filter the noise, it is connected to the ground via a small capacitor of capacitance 0.01µF.

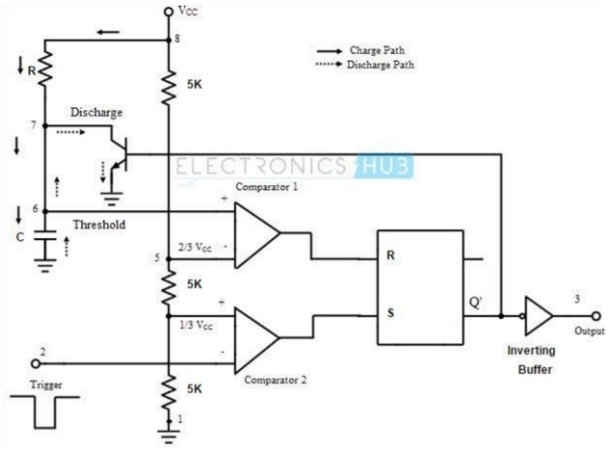
The monostable multivibator can be used as a Frequency Divider, Linear Ramp Generator, Pulse Width modulator, Relay switch etc…



Operation

The monostable mode is also called “one-shot” pulse generator. The sequence of events starts when a negative going trigger pulse is applied to the trigger comparator. When this trigger comparator senses the short negative going trigger pulse to be just below the reference voltage (1/3 VCC), the device triggers and the output goes HIGH.

The discharge transistor is turned OFF and the capacitor C that is externally connected to its collector will start charging to the max value through the resistor R. The HIGH output pulse ends when the charge on the capacitor reaches 2/3 V_{CC}. The internal connection of the IC 555 in monostable mode along with the RC timing circuit is shown below.



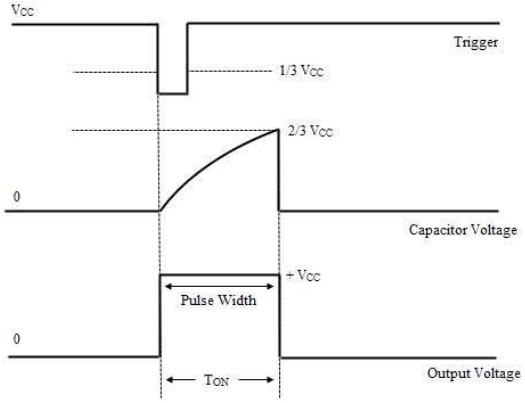
The detailed operation can be explained as follows. Initially, the flip-flop is RESET. This will allow the discharge transistor to go to saturation. The capacitor C, which is connected to the open collector (drain in case of CMOS) of the transistor, is provided with a discharge path. Hence the capacitor discharges completely and the voltage across it is 0. The output at pin 3 is low (0).

When a negative going trigger pulse input is applied to the trigger comparator (comparator 2), it is compared with a reference voltage of 1/3 V_{CC}. The output remains low until the trigger input is greater than the reference voltage. The moment trigger voltage goes below 1/3 VCC, the output of comparator goes high and this will SET the flip-flop. Hence the output at pin 3 will become high.

At the same time, the discharge transistor is turned OFF and the capacitor C will begin to charge and the voltage across it rises exponentially. This is nothing but the threshold voltage at pin 6. This is given to the comparator 1 along with a reference voltage of 2/3 V_{CC}. The output at pin 3 will remain HIGH until the voltage across the capacitor reaches 2/3 VCC.

The instance at which the threshold voltage (which is nothing but the voltage across the capacitor) becomes more than the reference voltage, the output of the comparator 1 goes high. This will RESET the flip-flop and hence the output at pin 3 will fall to low (logic 0) i.e. the output returns to its stable state. As the output is low, the discharge transistor is driven to saturation and the capacitor will completely discharge.

Hence it can be noted that the output at pin 3 is low at start, when the trigger becomes less than 1/3 VCC the output at pin 3 goes high and when the threshold voltage is greater than 2/3 VCC the output becomes low until the occurrence of next trigger pulse. A rectangular pulse is produced at the output. The time for which the output stays high or the width of the rectangular pulse is controlled by the timing circuit i.e. the charging time of the capacitor which depends on the time constant RC.



Pulse Width Derivation

We know that the voltage across the capacitor C rises exponentially. Hence the equation for the capacitor voltage VC can be written as

$V_C = V_{CC} (1 – e^{-t/RC})$

When the capacitor voltage is 2/3 V_{CC}, then, $2/3\ V_{CC} = V_{CC} (1 – e^{-t/RC})$ ∴

$t \approx 1.1\ RC$

The pulse width of the output rectangular pulse is W = 1.1 RC.

Design

A Monostable 555 Timer is required to produce a time delay within a circuit. If a 10μF timing capacitor is used, calculate the value of the resistor required to produce a minimum output time delay of 500ms.

Given C = 10μF

R = t/1.1C = 45.5 kΩ

Program and Input Data

```
# Libraries ********** import
math     # Just to get Euler's constant import
matplotlib.pyplot as pplot

import random   # For bouncing

####  Constants  ########################################################################

vin = 5.0          # Voltage in
```

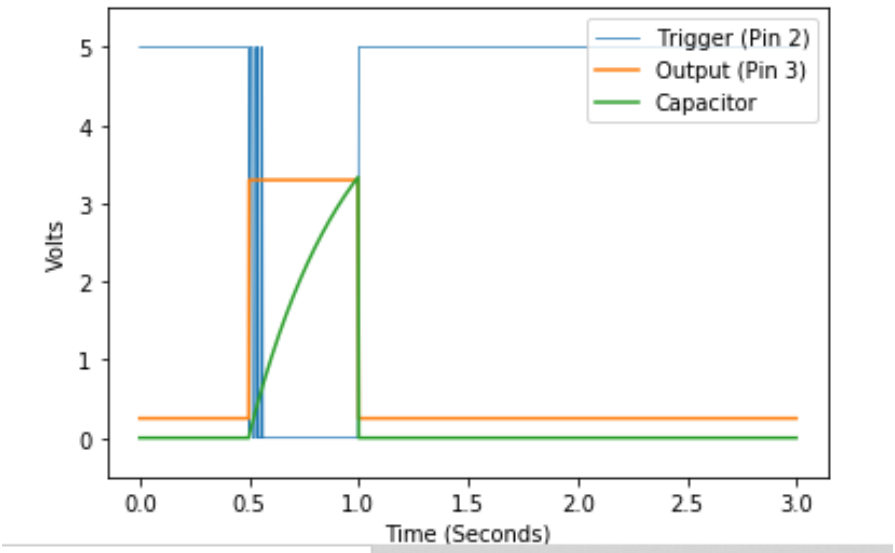
```
out_high = 3.3      # Voltage output from pin 3 when "high"
out_low  = 0.25     # Voltage output from pin 3 when "low"
# (These are not simply vin and zero. See datasheet.)

farads = 2/1000000  # Capacitance of primary capacitor, which connects both the
# threshold (pin 6) and discharge (pin 7) to ground. It is                    # fed
voltage through one end of the primary resistor.

ohms = 1000000      # Resistance in ohms of the primary resistor. This connects
# the input voltage to pins 6 & 7 as well as the capacitor.

sim_time = 3        # Length of time to run the entire simulation (in seconds)
step = sim_time / 1000      # Time increment for each simulation step (seconds)
button_start_time = 0.5     # At what point do we fire off the timer (seconds)
button_hold_time = 0.5      # How long do we press the button? (seconds)
bounce_time = 0.07  # How long to simulate switch bounce after button press

                # (Set to zero to turn off bounce simulation)


bounce_prob = 0.33  # Chance that switch will bounce open during each step


unbounce_prob = 0.5 # Chance that a bounced switch will re-close during a step

# Bounce time should not exceed button hold time
if bounce_time > button_hold_time:
bounce_time = button_hold_time


def update_latch(latch, trigger, threshold, vin):


    # Latch is "set" if the trigger voltage is less than 1/3 vin

    # (But not necessarily "unset" if it's not!)
if trigger < vin / 3:         latch = True


    # Latch is "unset" if the threshold voltage is above 2/3 vin

    # (But not necessarily "set" if it's not!)
if threshold > vin * 2 / 3:         latch =
False

    # For voltages between these, the latch stays at whatever it was
return latch


def update_capacitor(cap, farads, ohms, vin, latch, step):

        if
latch:

        # Charging... Time constant
tc = farads * ohms         # Time
proportion
```

```
        tp = step / tc

        # Percent change from existing voltage to full voltage (vin)
pc = 1 - (1/math.e**tp)         cap += (vin - cap) * pc     else:

        # Discharging... Resistor is bypassed, so discharge is essentially immediate
cap = 0         # That was easy!

    return cap


#### Set up    trigger = vin latch = False vout =
out_low     cap     =     0     pre_trigger_steps     =
round(button_start_time / step) trigger_list =
[trigger] * pre_trigger_steps cap_list = [cap] *
pre_trigger_steps      out_list      =      [vout]      *
pre_trigger_steps

time_list = [i * step for i in range(pre_trigger_steps)]

time = button_start_time


#### Simulate

# Seed for bounce randomness

random.seed(8675309)
# Press the button! :-)
trigger = 0 button_time
= 0 while time <
sim_time:

    latch = update_latch(latch, trigger, cap, vin)     cap =
update_capacitor(cap, farads, ohms, vin, latch, step)     if
latch:         vout = out_high     else:

        vout = out_low
trigger_list.append(trigger)
cap_list.append(cap)
out_list.append(vout)
time_list.append(time)     time +=
step     button_time += step     #
Should we check for bounce?     if
button_time <= bounce_time:         r
= random.randint(0, 100) / 100

        # If trigger is low (button pressed) will it bounce open?
if trigger == 0 and r < bounce_prob:

            trigger = vin

        # If trigger is high (button has bounced open), will it re-close?
elif trigger != 0 and r < unbounce_prob:

            trigger = 0
else:

        # If we're not bouncing, button is either pressed or unpressed
if button_time > button_hold_time:
```

```
        trigger = vin        else:        trigger = 0
# Print constants print() print('Capacitor: ' +
str(farads * 1000000) + 'mf') print('Resistor: ' +
str(ohms/1000) + 'K') print('Vcc: +' + str(vin) +
'V') print()

# Print out timing info (formula from the datasheet)
time_high = round(1.1 * ohms * farads, 2)
print('Time high: ' + str(time_high*1000) + 'ms')
print('Periods simulated: ' + str(len(time_list)))
print('Bounce time: ' + str(bounce_time*1000) + 'ms')
print()

# Trigger (pin 7) voltages

pplot.plot(time_list, trigger_list, linewidth=0.7, label='Trigger (Pin 2)')

# Output (Q) voltages

pplot.plot(time_list, out_list, label='Output (Pin 3)')

# Capacitor voltages

pplot.plot(time_list, cap_list, label='Capacitor')
pplot.xlabel('Time (Seconds)')
pplot.ylabel('Volts') pplot.ylim(0-vin*0.1,
vin*1.1) pplot.legend(loc='upper right')
pplot.show()
```

OUTPUT

```
Capacitor: 10.0uf
Resistor: 45.5K
Vcc: +5.0V

Time high: 500.0ms
Periods simulated: 1001
Bounce time: 70.0ms
```



PRE-LAB

1. An Monostable multivibrator has _____ states
2. Monostable multivibrator is used as a _____
3. Which among the following can be used to detect the missing heart beat?
    a) Monostable multivibrator
    b) Astable multivibrator
    c) Schmitt trigger
    d) None of the mentioned

POST LAB

1. A monostable multivibrator has R = 120kΩ and the time delay T = 1000ms, calculate the value of C?

    a) 0.9µF

    b) 1.32µF

    c) 7.5µF

    d) 2.49µF

2. How can a monostable multivibrator be modified into a linear ramp generator?

    a) Connect a constant current source to trigger input
    b) Connect a constant current source to trigger output
    c) Replace resistor by constant current source
    d) Replace capacitor by constant current source

Results and Conclusion

Executed a python program to design a Monostable Multvibrator required to produce a minimum output time delay of

(i) 500ms.

(ii) 250ms
and plot the output voltage and capacitor voltage.

EXPERIMENT# 9 Magnitude Response of Active Low Pass Filter

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Instructor :					

Title of the Experiment

9	**Write a python program to evaluate the Step Response ans Impulse Response of Low Pass filter .**

Aim / Objective. To write a python code to design. S*imulate the program in Jupyter Notebook.*

Theory: Low Pass Filter

Presented below is a circuit for an active lowpass filter. The analysis is in Laplace domain and use SymPy to obtain the solutions for various input signals. Here the values of the components are given to be
G = 1.586, R1 = R2 = 10kΩ, C1 = C2 = 1nF

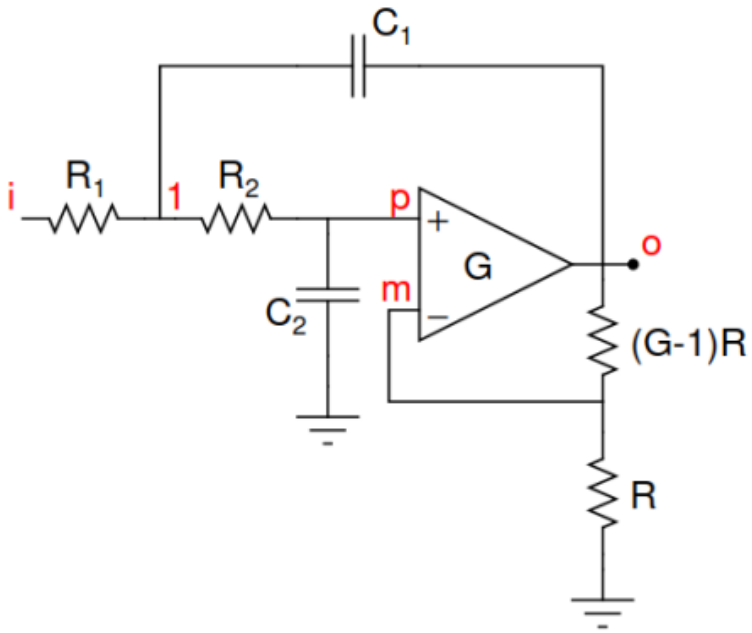


Fig. 9.1 Active Low Pass Filter

Analysing in Laplace domain we obtain the following equations:

$$V_m = \frac{V_o}{G} \tag{1}$$

$$V_1(\frac{1}{R_2} + \frac{1}{R_1} + C_1 s) - \frac{V_i}{R_1} - C_1 s V_o - \frac{V_p}{R_2} = 0 \tag{2}$$

$$V_p = \frac{V_1}{1 + R_2 C_2 s} \tag{3}$$

$$V_o = G(V_p - V_m) \tag{4}$$

Expressing these equations in matrix form, we get

$$\begin{bmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{1}{1+R_2C_2s} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ (\frac{1}{R_2} + \frac{1}{R_1} + C_1 s) & -\frac{1}{R_2} & 0 & -C_1 s \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{V_i}{R_1} \end{bmatrix}$$

Av = b

Now we can solve this matrix equation for any given input signal Vi to obtain the matrix v. The last element of v is the output. We shall now solve this using SymPy for a variety of input signals.

• We first define the function lowpass() which generates the matrices A and b for given system parameters, solves for v and returns all 3 matrices.
s is defined as a SymPy symbol here.

```
# lowpass filter circuit
def lowpass ( R1 , R2 , C1 , C2 , G , Vi ):
A = sy . Matrix ([[0 , 0 , 1 , -1/ G ] , [ -1/(1+ R2 * C2 * s ) , 1 , 0 , 0] ,
[0 , -G , G , 1] , [(1/ R1 + 1/ R2 + C1 * s ) , -1/ R2 , 0 , - C1 * s ]])
b = sy . Matrix ([0 , 0 , 0 , Vi / R1 ])
V = A . inv () * b
return A , b , V
```

• Then we initialize this system with the given parameters, and extract the actual output signal in terms
of s.

```
A ,b , V = lowpass (10000 ,10000 ,1 e -9 ,1 e -9 ,1.586 ,1)
Vo = V [3]
```

• We initialize our plotting space in frequency domain to be a logarithmic space from frequency ω of
1rad/s to 108
rad/s.

We then obtain the values for s we need for the frequency response by populating an array with the imaginary values jω that is the variable ss.

```
w = p . logspace (0 ,8 ,801)
ss = complex (0 , 1)* w
```

• We obtain the frequency response by using the function lambdify() which converts the SymPy transfer function to a lambda function which can be evaluated for all ω in our logspace.

```
hf = sy . lambdify (s , Vo , ' numpy ')
```

• We obtain the magnitude response by evaluating the absolute value of frequency response for our logspace ω. We then generate the magnitude plot of this system.

```
v = hf ( ss )
p . loglog (w , abs ( v ), lw =2)
```

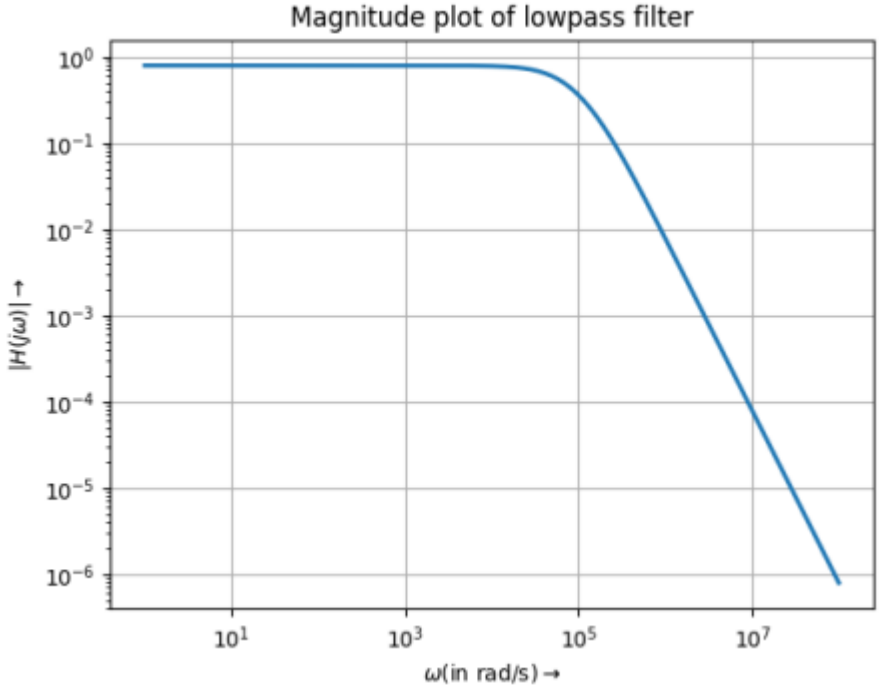


Fig. 9.2 Magnitude of Frequency Response of Active Low Pass Filter

As expected, it is a lowpass filter.

• We shall now evaluate the step response of this lowpass filter.

• We know that the laplace transform of the unit step function is $\frac{1}{s}$. For step response, we will simply multiply the function with $\frac{1}{s}$ to obtain the laplace transform of the output.

• We want to obtain the time domain output. Therefore we must first convert the SymPy transfer function into a scipy.signal.lti object. We will then use scipy.signal.impulse() to obtain the time domain function for a given time interval.

t = p . linspace (0 , 0.001 , 1000)
StepResponse_SDomain = convertSympyToLTI (Vo * 1/ s)
t , StepResponse = sp . impulse (StepResponse_SDomain , None , t)
• The conversion from SymPy function to scipy.signal.lti object is quite simple. We have made a
function convertSympyToLTI() for this purpose. We simply obtain the coefficients on both numerator
and denominator side by first extracting the polynomials, then obtaining all coefficients.
• We then create our new scipy.signal.lti object with these arrays
# converts a sympy transfer function to a scipy . signal . lti object
def convertSympyToLTI (H):
# fraction function : Returns a pair with
# expressions numerator and denominator .
n , d = sy . fraction (H)
# convert those to polynomials
polynum = sy . poly (n)
4
polyden = sy . poly (d)
# get arrays for their coefficients
numCoeff = polynum . all_coeffs ()
denCoeff = polyden . all_coeffs ()
# feed the coefficient arrays into sp . lti to get an
# lti system object with the transfer function H
H_lti = sp . lti (p . array (numCoeff , dtype = float) ,
p . array (denCoeff , dtype = float))
return H_lti
• We have obtain the time domain function for step response. We now plot it.

Program and Input Data

```
# Libraries **********
# Using System module - Low Pass Filter, High Pass Filter
# Impulse and Step Response

import sympy as sy
import matplotlib.pylab as p
import scipy.signal as sp

s = sy.symbols('s')

# lowpass filter circuit
def lowpass(R1, R2, C1, C2, G, Vi):
    A = sy.Matrix([[0, 0, 1, -1/G], [-1/(1+R2*C2*s), 1, 0, 0], [0, -G, G, 1], [(1/R1 + 1/R2 +
C1*s), -1/R2, 0, -C1*s]])
```

```
    b = sy.Matrix([0, 0, 0, Vi/R1])

    V = A.inv() * b

    return A, b, V

# highpass filter circuit
def highpass(R1, R3, C1, C2, G, Vi):
    A = sy.Matrix([[0, 0, 1, -1/G], [-R3*C2*s/(1+R3*C2*s), 1, 0, 0], [0, -G, G, 1], [(C1*s +
C2*s + 1/R1), -C2*s, 0, -1/R1]])
    b = sy.Matrix([0, 0, 0, Vi*s*C1])

    V = A.inv() * b

    return A, b, V

# converts a sympy transfer function to a scipy.signal.lti object
def convertSympyToLTI(H):
    # fraction function: Returns a pair with expression's numerator and denominator.
    n, d = sy.fraction(H)
    # convert those to polynomials
    polynum = sy.poly(n)
    polyden = sy.poly(d)
    # get arrays for their coefficients
    numCoeff = polynum.all_coeffs()
    denCoeff = polyden.all_coeffs()

    # feed the coefficient arrays into sp.lti to get an lti system object with the transfer funciton
H
    H_lti = sp.lti(p.array(numCoeff, dtype=float), p.array(denCoeff, dtype=float))

    return H_lti

def lowpassAnalysis():
    # first of all show the magnitude plot
    A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
    print('G=1000')
    Vo = V[3]
    print(Vo)
    w = p.logspace(0,8,801)
    ss = complex(0, 1)*w
    hf = sy.lambdify(s, Vo, 'numpy')
    v = hf(ss)
```

```
    p.loglog(w,abs(v),lw=2)
    p.title("Magnitude plot of lowpass filter")
    p.ylabel("$|H(j\\omega)|\\rightarrow$")
    p.xlabel("$\\omega$(in rad/s)$\\rightarrow$")
    p.grid(True)
    p.show()

    # Q1
    # now we obtain the step response
    t = p.linspace(0, 0.001, 1000)
    StepResponse_SDomain = convertSympyToLTI(Vo * 1/s)
    t, StepResponse = sp.impulse(StepResponse_SDomain, None, t)

    p.plot(t, StepResponse)
    p.title("Step response of lowpass filter")
    p.ylabel("$v_{o}(t)$(in V)$\\rightarrow$")
    p.xlabel("$t$(in s)$\\rightarrow$")
    p.grid(True)
    p.show()

    # Q2
    # response for sum of sinusoids
    t = p.linspace(0,0.01,100000)
    Vinp = p.sin(2e3*p.pi*t)+p.cos(2e6*p.pi*t)
    t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)

    p.plot(t, Vinp, label='$V_{in}$')
    p.plot(t, Vout, label='$V_{out}$')
    p.title("Response for superposition of sinusoids of frequency $10^3s^{-1}$  and $10^6s^{-
1}$")
    p.ylabel("$v(t)$(in V)$\\rightarrow$")
    p.xlabel("$t$(in s)$\\rightarrow$")
    p.grid(True)
    p.legend()
    p.show()

def highpassAnalysis():
    # first of all show the magnitude plot
    A,b,V = highpass(10000,10000,1e-9,1e-9,1.586,1)
    Vo = V[3]
    print(Vo)
    w = p.logspace(0,8,801)
    ss = complex(0, 1)*w
    hf = sy.lambdify(s, Vo, 'numpy')
```

```
v = hf(ss)

p.loglog(w,abs(v),lw=2)
p.title("Magnitude plot of highpass filter")
p.ylabel("$|H(j\\omega)|\\rightarrow$")
p.xlabel("$\\omega$(in rad/s)$\\rightarrow$")
p.grid(True)
p.show()

# Q4
# output for a decaying sinusoid with
# freq = 1
# decay factor = 0.5
t = p.linspace(0, 5, 1000)
Vinp = p.exp(-0.5*t) * p.sin(2*p.pi*t)
t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)

p.plot(t, Vinp, label='$V_{in}$')
p.plot(t, Vout, label='$V_{out}$')
p.title("Response for decaying sinusoid of frequency $10^3$")
p.ylabel("$v(t)$(in V)$\\rightarrow$")
p.xlabel("$t$(in s)$\\rightarrow$")
p.grid(True)
p.legend()
p.show()

# output for a decaying sinusoid with
# freq = 1e6
# decay factor = 0.5
t = p.linspace(0, 0.0001, 10000)
Vinp = p.exp(-0.5*t) * p.sin(2e6*p.pi*t)
t, Vout, svec = sp.lsim(convertSympyToLTI(Vo), Vinp, t)

p.plot(t, Vinp, label='$V_{in}$')
p.plot(t, Vout, label='$V_{out}$')
p.title("Response for decaying sinusoid of frequency $10^6$")
p.ylabel("$v(t)$(in V)$\\rightarrow$")
p.xlabel("$t$(in s)$\\rightarrow$")
p.grid(True)
p.legend()
p.show()

#Q5
# now we obtain the step response
```

```
    t = p.linspace(0, 0.001, 1000)
    StepResponse_SDomain = convertSympyToLTI(Vo * 1/s)
    t, StepResponse = sp.impulse(StepResponse_SDomain, None, t)

    p.plot(t, StepResponse)
    p.title("Step response of lowpass filter")
    p.ylabel("$v_{o}(t)$(in V)$\\rightarrow$")
    p.xlabel("$t$(in s)$\\rightarrow$")
    p.grid(True)
    p.show()

# Q1, Q2
lowpassAnalysis()
# Q3, Q4, Q5
highpassAnalysis()
```

PRELAB:

1. What will be the output of the program?

```
list_1 = [1,2,3,4,5,6,7,8,9]
cubed = map(lambda x: pow(x,3), list_1)
list(cubed)
```

2. Write the syntax for impulse response of continuous time system using scipy.
3. What is the sytax for finding the response of continuous-time system using scipy.

POST LAB

1. Compute the impulse response of a second order system with a repeated root: x''(t) + 2*x'(t) + x(t) = u(t)
2. Correct the errors in the program and Plot the output

```
from scipy import signal
lti =lti([1.0], [1.0, 1.0])
t, y = signal.step(lti)
plt.plot(t, y)
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Step response for 1. Order Lowpass'
```

Result

- The Active Low pass is analysed using the Laplace Transform.
- The analysis was performed using symbolic algebra in the SymPy module for Python, and Magnitude plots of the frequency response were plotted for the filter.

EXPERIMENT#10 Magnitude Response Of Active Op-Amp High Pass Filter

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Instructor :					

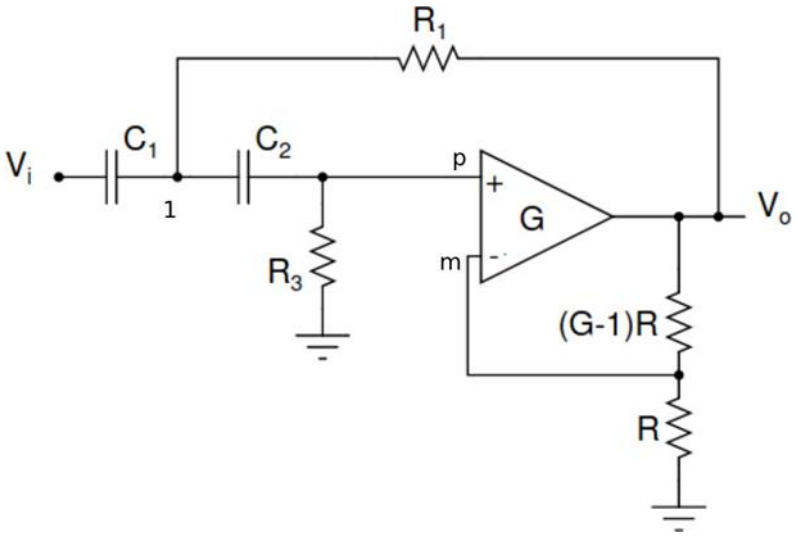
Title of the Experiment

10.	**To plot the magnitude response of active op-amp High Pass Filter.**

Aim / Objective. To write a python code to design and plat the step response and impulse response of High Pass Filter, S*imulate the program in Jupyter Notebook.*

Theory: High Pass Filter

Presented below is a circuit for an active highpass filter. We shall analyse this in Laplace domain and use SymPy to obtain the solutions for various input signals. Here the values of the components are given to be G = 1.586, R1 = R3 = 10kΩ, C1 = C2 = 1nF .



Analysing in laplace domain we obtain the following equations:

$$V_m = \frac{V_o}{G} \tag{5}$$

$$V_1(C_2 s + \frac{1}{R_1} + C_1 s) - C_1 s V_i - \frac{V_o}{R_1} - C_2 s V_p = 0 \tag{6}$$

$$V_p = \frac{R_3 C_2 s V_1}{1 + R_3 C_2 s} \tag{7}$$

$$V_o = G(V_p - V_m) \tag{8}$$

Expressing these equations in matrix form, we get:

$$\begin{bmatrix} 0 & 0 & 1 & -\frac{1}{G} \\ -\frac{R_3C_2s}{1+R_3C_2s} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ (C_2s+\frac{1}{R_1}+C_1s) & -C_2s & 0 & -\frac{1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ C_1sV_i \end{bmatrix}$$

$$Av = b$$

Now we can solve this matrix equation for any given input signal Vi to obtain the matrix v. The last element of v is the output. We shall now solve this using SymPy for a variety of input signals.

We create a highpass() function for the transfer function which accepts circuit parameters and solves for the v matrix. It returns A, b as well as v matrices.

```
# highpass filter circuit
def highpass ( R1 , R3 , C1 , C2 , G, Vi ):
A = sy. Matrix ([[0 , 0 , 1 , -1/ G], [- R3 * C2 * s /(1+ R3 * C2 * s), 1 , 0 , 0],
                [0 , -G, G, 1], [( C1 * s + C2 * s + 1/ R1 ), - C2 *s, 0 , -1/ R1 ]])
b = sy. Matrix ([0 , 0 , 0 , Vi* s* C1 ])
V = A. inv () * b
return A, b, V
```

We use the same method as explained for lowpass to find and generate the magnitude plot of the transfer function using the lambdify() function

```
# first of all show the magnitude plot
A,b, V= lowpass (10000 ,10000 ,1 e -9 ,1 e -9 ,1.586 ,1)
print (' G =1000 ')
 Vo = V [3]
print ( Vo)
w = p. logspace (0 ,8 ,801)
ss = complex (0 , 1)* w
hf = sy. lambdify (s, Vo , ' numpy ')
v  =  hf( ss )
 p. loglog (w, abs ( v), lw =2)
```

We now analyse the output of the system for the input signal of a decaying sinusoid. First consider a sinusoid of low frequency, here 1Hz

$$V_i(t) = e^{-0.5t}sin(2\pi t)$$

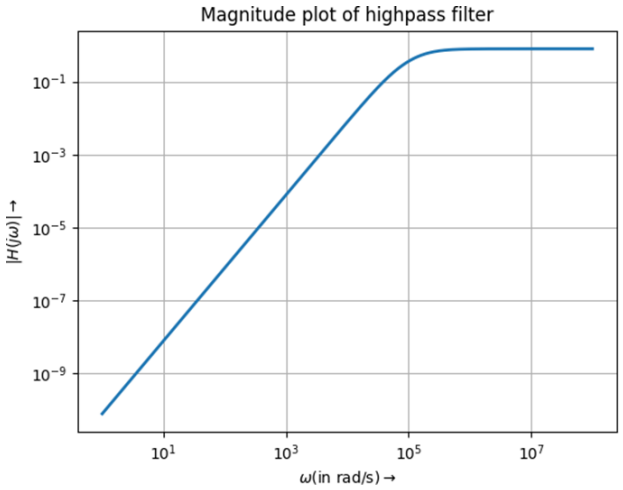


Figure 10.1: Magnitude plot of high pass

using the scipy.signal.lsim() function as follows:

```
# output for a decaying sinusoid with freq = 1
# decay factor = 0.5
t = p. linspace (0 , 5 , 1000)
Vinp = p. exp ( -0.5* t) * p. sin (2* p. pi* t)
t, Vout , svec = sp. lsim ( convertSympyToLTI ( Vo), Vinp , t)
```

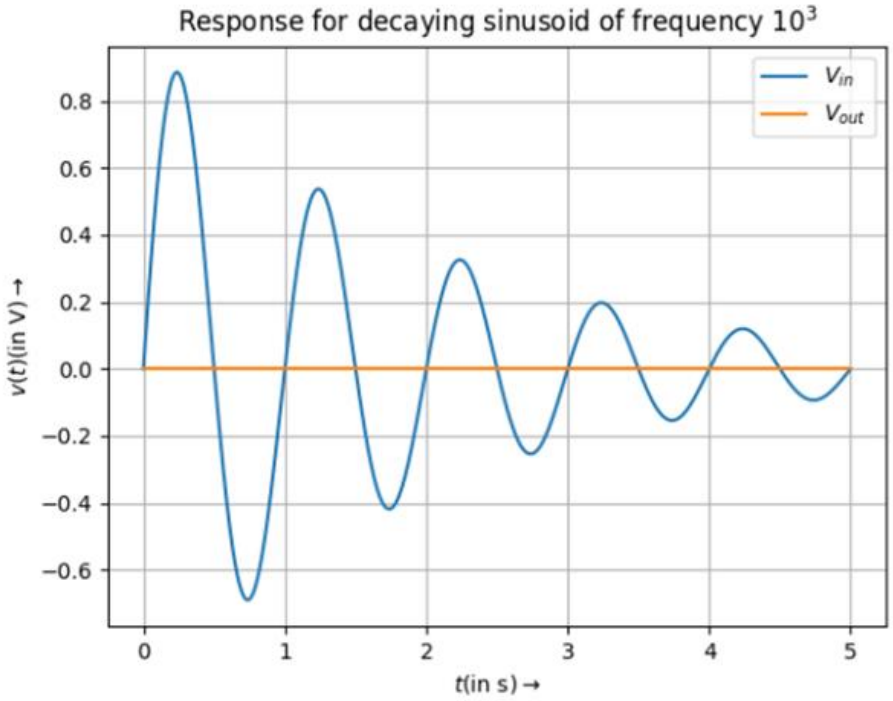


Figure 10.2: Time domain output $V_o(t)$ for the input

$$V_i(t) = e^{-0.5t} sin(2\pi t)$$

• The system almost completely cuts out this frequency because 2πrad/s is very low frequency which is attenuated heavily, hence the highpass filter shows no output.

• Let us now test this for a high frequency (within the passband). Let us consider $10^6 Hz$.

$$V_i(t) = e^{-0.5t} sin(2 \times 10^6 \pi t)$$

```
# output for a decaying sinusoid with
 # freq = 1e6
# decay factor = 0.5
t = p. linspace (0 , 0.0001 , 10000)
Vinp = p. exp ( -0.5* t) * p. sin (2 e6 * p. pi* t)
t, Vout , svec = sp. lsim ( convert Sympy To LTI ( Vo), Vinp , t)
```

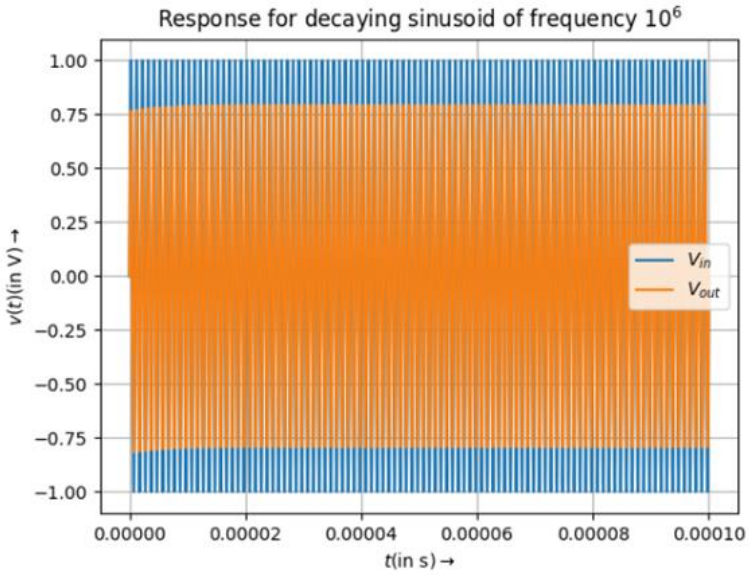


Figure 10.3: Time domain output $V_o(t)$ for the input

$$V_i(t) = e^{-0.5t} sin(2 \times 10^6 \pi t)$$

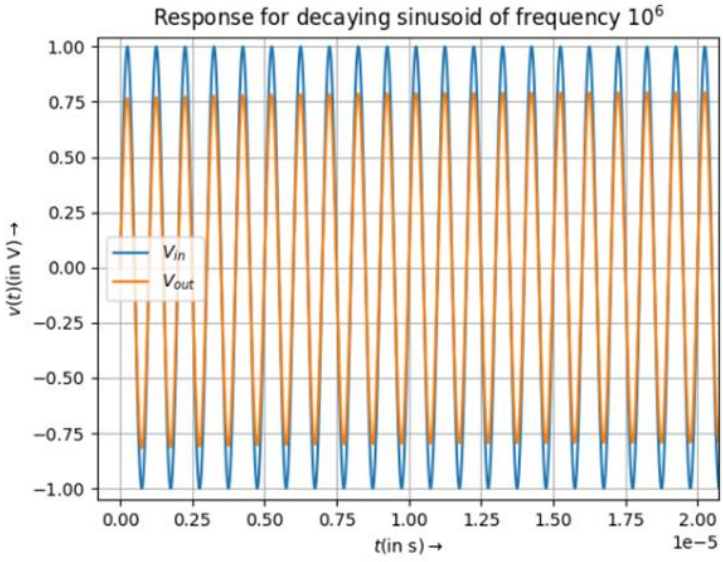


Figure 10.4: A zoom in look of figure 10.3

- We can see that the signal survives, albeit it is slightly attenuated because at at $2 \times 10^6 \pi rad/s$ the gain is just smaller than 0dB.
- This is expected, because the frequency is within the passband of the highpass filter.
- We now analyse the step response of this system in the same way we did for lowpass filter, using scipy.signal.impulse()

```
# now we obtain the step response
t = p. linspace (0 , 0.001 , 1000)
Step Response_SDomain = convertSympyToLTI ( Vo * 1/ s)
t, Step Response = sp. impulse ( StepResponse_SDomain , None , t)
```

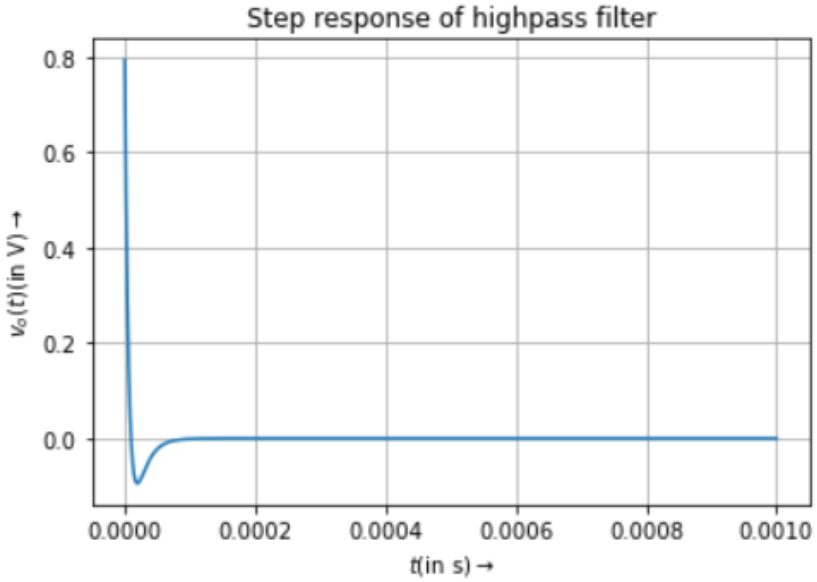


Figure 10.5: Step response of highpass filter

- The reason for the initial peak is the initial conditions. The capacitor does not allow instantaneous changes in the voltage across it. Before t = 0, both capacitors have no charge, and the voltage across both is 0, and this will not change instantaneously.

- Refering to the circuit diagram, the instant the 1V of the unit step is applied ($t = 0^+$), the drop acrossthe capacitors C_1 and C_2 will still be zero. Therefore $V_m = 1V$.

- $V_m = \frac{V_o}{G}$ and $V_o = G(V_p - V_m)$, therefore we get

$$V_0 = G(1 - \frac{V_o}{G})$$

$$2V_o = G$$

$$V_o = \frac{G}{2} = \frac{1.586}{2} = 0.793 \approx 0.8V$$

- This is what explains the initial peaking of 0.8V. It then quickly decays and settles down

to the expected 0V because it is a highpass filter, and DC will give no output on such a filter

Program and Input Data

```
import sympy as sy
import matplotlib.pylab as p
import scipy.signal as sp

s = sy.symbols('s')
# highpass filter circuit
def highpass(R1, R3, C1, C2, G, Vi):
    A = sy.Matrix([[0, 0, 1, -1/G], [-R3*C2*s/(1+R3*C2*s), 1, 0, 0], [0, -G, G, 1], [(C1*s + C2*s + 1/R1), -C2*s, 0, -1/R1]])
    b = sy.Matrix([0, 0, 0, Vi*s*C1])

    V = A.inv() * b

    return A, b, V


# converts a sympy transfer function to a scipy.signal.lti object
def convertSympyToLTI(H):
    # fraction function: Returns a pair with expression's numerator and denominator.
    n, d = sy.fraction(H)
    # convert those to polynomials
    polynum = sy.poly(n)
    polyden = sy.poly(d)
    # get arrays for their coefficients
    numCoeff = polynum.all_coeffs()
    denCoeff = polyden.all_coeffs()

    # feed the coefficient arrays into sp.lti to get an lti system object with the transfer funciton H
    H_lti = sp.lti(p.array(numCoeff, dtype=float), p.array(denCoeff, dtype=float))

    return H_lti

def highpassAnalysis():
    # first of all show the magnitude plot
    A,b,V = highpass(10000,10000,1e-9,1e-9,1.586,1)
    Vo = V[3]
    print(Vo)
    w = p.logspace(0,8,801)
    ss = complex(0, 1)*w
```

```
    hf = sy.lambdify(s, Vo, 'numpy')
    v = hf(ss)

    p.loglog(w,abs(v),lw=2)
    p.title("Magnitude plot of highpass filter")
    p.ylabel("$|H(j\\omega)|\\rightarrow$")
    p.xlabel("$\\omega$(in rad/s)$\\rightarrow$")
    p.grid(True)
    p.show()

# Q3, Q4, Q5
highpassAnalysis()
```

PRELAB

1. Write the syntax for finding the transfer function using scipy and control
2. What is the significance of using control module / library?

POST LAB

1. Plot the output of the program and interpret the result

   ```
   import control
   import matplotlib.pyplot as plt
   s = control.TransferFunction.s
   H = (2)/(3*s + 1)
   print ('H(s) =', H)
   t, y = control.step_response(H)
   plt.plot(t,y)
   plt.title("Step Response")
   plt.grid()
   ```

2. For the given transfer function, write the python code

   $$H(s) = \frac{s^2+3s+3}{s^2+2s+1};$$

Result

- The Active High Pass Filters is analysed using the Laplace Transform.
- The analysis was performed using symbolic algebra in the SymPy module for Python, and Magnitude plots of the frequency response were plotted for the filter.

ESPERIMENT#11 Generation of Additive White Gaussian Noise

Course Code	**:**	18ECS301J	**Course Title**	**:**	Applied Programming
Reg. No.	**:**		**Name**	**:**	
Semester	**:**	VI Semester	**Year**	**:**	III Year
Date of Expt.	**:**		**Date of Submission**	**:**	
Name of the Lab Instructor :					

Title of the Experiment

11	**Write a python program to simulate Additive White Gaussian Noise**

Aim / Objective: To write a python function to simulate Standard Additive White Gaussian Noise

Theory:

A single sample of Additive White Gaussian Noise (AWGN) is a realization of a random variable whose probability density function is a scaled standard normal distribution. A normal distribution is a symmetric, bell-shaped curve that describes the distribution of many types of data. The normal distribution has two parameters, mean and standard deviation. All AWGN samples are identical and independent which is why their spectrum is uniform across all frequencies (i.e white). Each sample of AWGN is modelled as directly being added to the signal of interest (i.e Additive).

If the inputs to the channel are samples Xi, then the output Yi of the AWGN channel with AWGN is expressed as :

$$Yi=Xi+Zi$$

Zi is drawn from a zero mean normal distribution with variance N. Zi is uncorrelated with Xi. Power of the real AWGN signal Zi is N. The complex AWGN signal may be represented as two separate and uncorrelated streams of real AWGN. The power of the complex AWGN is 2N.

Program and Input Data

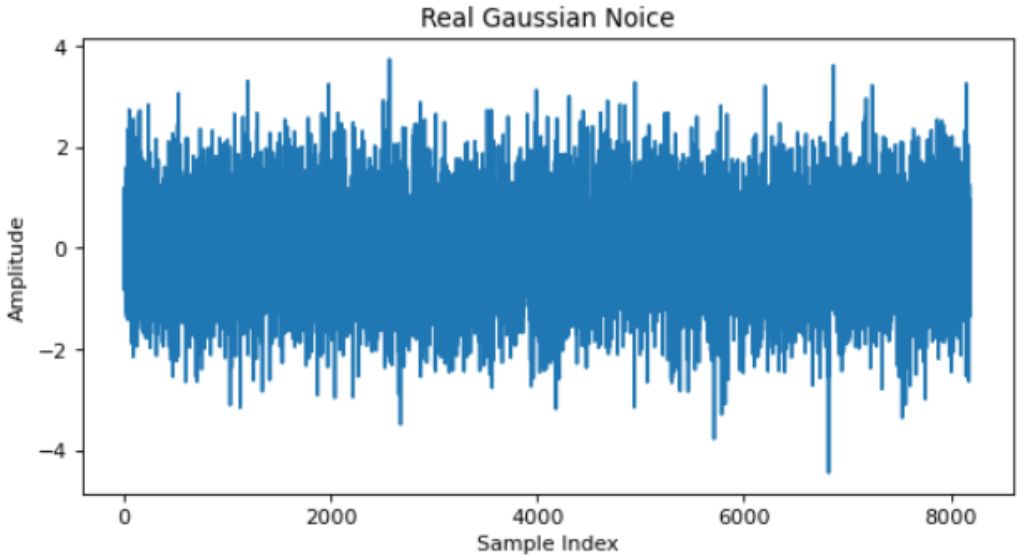
```
# Program to generate AWGN
# Import libraries

import numpy as np
import random as rand
import matplotlib.pyplot as plt
#enter number of symbols

num_symbol = 8192
#samples drawn from a random normal distribution of variance
Z=np.random.normal(0,1,size=num_symbol)
Power=np.mean(np.abs(Z**2))
print("Mean Square Power = ", Power)
```

fig=plt.figure(figsize=(0,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(Z)
plt.ylabel('Amplitude')
plt.xlabel('Sample Index')
plt.title('Real Gaussian Noise')

[6]:
```
import numpy as np
import random as random
import matplotlib.pyplot as plt

num_s=8192
z=np.random.normal(0,1,size=num_s)
power=np.mean(np.abs(z**2))
print("Mean Square Power=",power)

fig=plt.figure(figsize=(8,4),dpi=80,facecolor='w',edgecolor='k')
plt.plot(z)
plt.ylabel("Amplitude")
plt.xlabel("Sample Index")
plt.title("Real Gaussian Noice")
```

```
Mean Square Power= 1.0164031756828413
```

[6]:
```
Text(0.5, 1.0, 'Real Gaussian Noice')
```



PRE-LAB

1. What is the difference between autocorrelation and cross correlation
2. What is modulation and why do we need modulation?
3. Although light travels much faster than sound, then also why the sound is heard first and the picture appear later when the TV is switched on?
4. What is white noise?

POST LAB:

1. Draw a normal distribution of zero mean and unit variance.

2. Which of the following are two key parameters of a normal distribution?

a. Mean and standard deviation
b. Mean and mode
c. Median and standard deviation
d. Mean and median

3. Standard normal distribution have mean as ____ and standard deviation as _____

a. (1, 0) b. (0, 1) c. (0, 2) d. (1, 1)

Results and Conclusion

Executed a python program to simulate Standard Additive White Gaussian Noise.

Not : Students are encourage to do Mini Project pertaining to the application of the lab and the same to be validated through simulation and verification.