Lab Report: *03*

Title: *Use of Google Colab and Code in Python*

Course Title: *Data and Telecommunication Laboratory*

Course Code: *CSE-260*

2nd Year 2nd Semester Examination 2023

Submitted to :-

Sarnali Basak
Associate Professor

Dr. Md. Imdadul Islam
Professor

Dr. Md. Abul Kalam Azad
Professor

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Class Roll	Name
371	Md. Ahad Siddiki

Date of Submission: 08/12/2024

Lab Report: Adding **Google Colab** to Drive File

Objective:

To learn how to create, save, and manage Google Colab notebooks in Google Drive for efficient access and collaboration.

Materials Required:

- Google account
- Access to Google Drive
- Internet connection

Procedure:

Step 1: Open Google Drive

1. Open your preferred web browser.
2. Navigate to Google Drive.
3. Log in with your Google account credentials.

Step 2: Create a New Google Colab Notebook

1. Create a new folder named “CoLab_lab”Click the **"New"** button on the left-hand side.
2. Hover over **"More"** and look for **"Google Colaboratory"** in the dropdown list.
    - If **Google Colaboratory** does not appear, follow Step 3.

Step 3: Add Google Colab to Google Drive

1. Click **"More"** at the bottom of the dropdown list.
2. Select **"Connect more apps"** to open the Google Workspace Marketplace.
3. In the search bar, type **"Google Colaboratory"** and press Enter.
4. Click on the **Google Colaboratory** app and hit **"Install"** (or **"Connect"**).

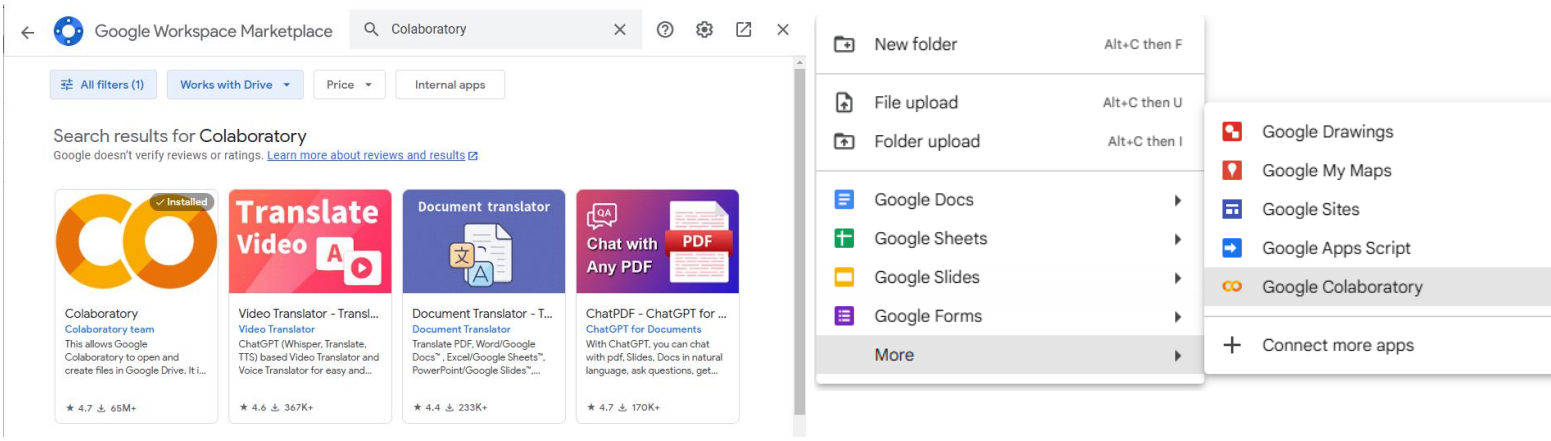
5. Grant permissions if prompted.
6. Once added, the **"Google Colaboratory"** option will appear under the **"More"** section when creating a new file in Drive.

Step 4: Save and Access Colab Files in Drive

1. After creating or opening a Colab notebook, save it to your Drive:
    - Navigate to **File > Save a copy in Drive** within the Colab notebook or created folder.
2. To access Colab files, open **Google Drive** and navigate to the **Colab Notebooks** folder (created automatically) or created folder “CoLab_lab”.

Results:

- Successfully added Google Colaboratory as an app to Google Drive.
- Learned how to create and save Colab notebooks in Google Drive for easy access.



Conclusion:

By integrating Google Colab with Google Drive, users can seamlessly create and manage Python notebooks, ensuring data is stored securely in the cloud and is accessible from anywhere.

Lab 01 starts Here

Assigining Two Variable, Basic Operations

In [ ]:
```
a=20
b=30
```

Addition in Python

In [ ]:
```
a+b
```

Out[ ]:
```
50
```

Subtraction

In [ ]:
```
a-b
```

Out[ ]:
```
-10
```

Multiplication of Two Variable

In [ ]:
```
a * b
```

Out[ ]:
```
600
```

Division

In [ ]:
```
a/b
```

Out[ ]:
```
0.6666666666666666
```

Differentiate: f(x)=sin(5.pie.x)+e^4x+x^2 , 1 time derivative

In [ ]:
```
import sympy as sp
x=sp.Symbol('x')
f=sp.sin(5*sp.pi*x)+sp.exp(4*x)+x**2
print(sp.diff(f,x,1))
```

```
2*x + 4*exp(4*x) + 5*pi*cos(5*pi*x)
```

Differentiate: d^2/dx^2=sin(5.pie.x) , 2 times derivative

In [ ]:
```
import sympy as sp
x=sp.Symbol('x')
f=sp.sin(5*sp.pi*x)
print(sp.diff(f,x,2))
```

```
-25*pi**2*sin(5*pi*x)
```

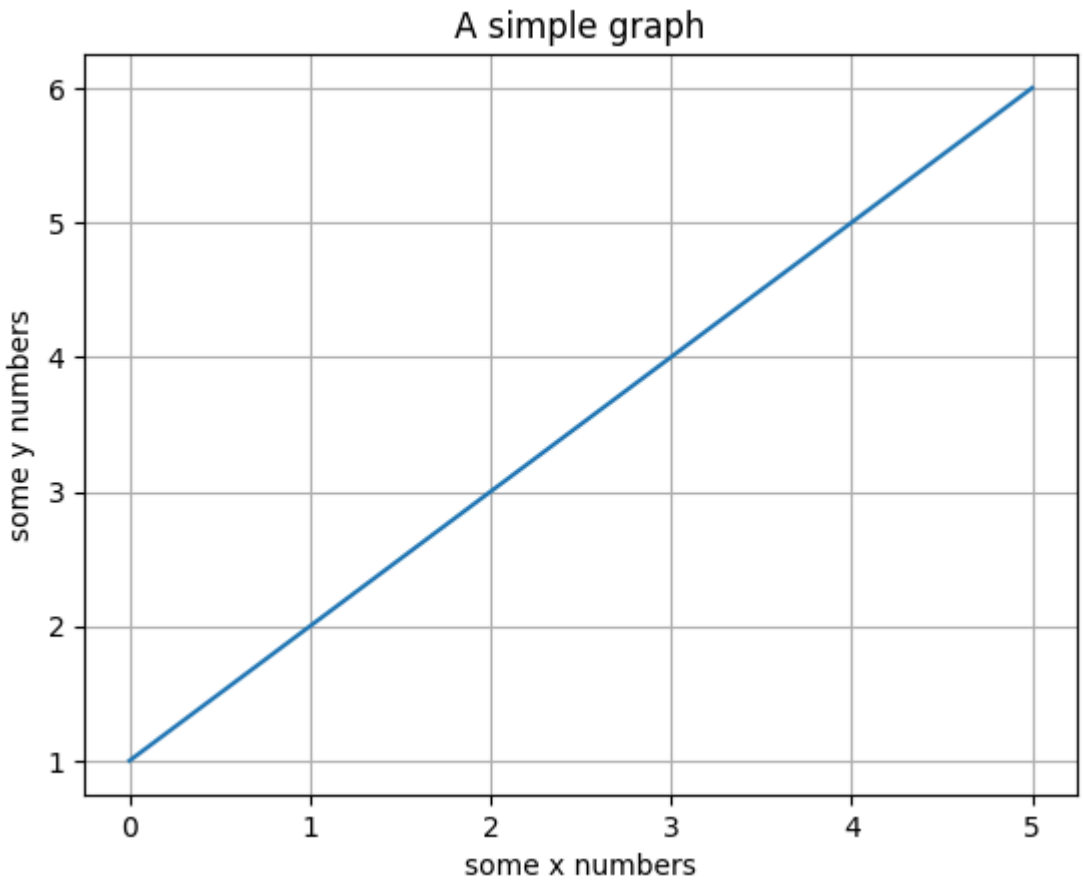
indefinite integral: sin(pi*x)

In [ ]:
```
from sympy import*
x=sp.Symbol('x')
f=sin(pi*x)
print(integrate(f,x))
```

```
-cos(pi*x)/pi
```

Simple Graph Plot using matplotlib

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4,5,6])
plt.ylabel('some y numbers')
plt.xlabel('some x numbers')
plt.title('A simple graph')
plt.grid(True, which='both')
plt.show()
```



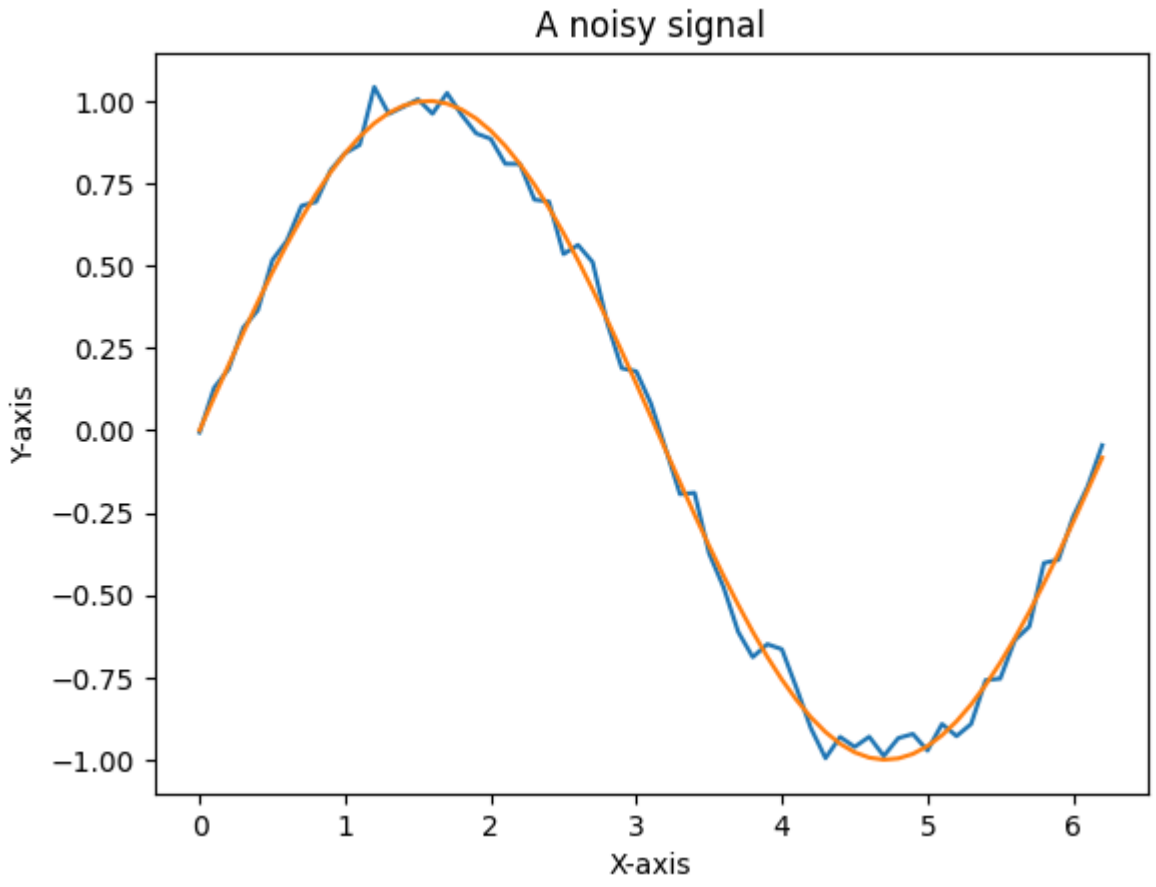
Observations:

- The plot displays a simple line graph with six data points on the x-axis, ranging from 1 to 6.
- The y-axis represents corresponding values for these x-values, which are the same as the x-values (1, 2, 3, 4, 5, 6).

- The graph includes labels for both axes: "some x numbers" for the x-axis and "some y numbers" for the y-axis.
- The title of the graph is "A simple graph".
- A grid is added to both axes, improving the readability of the graph and helping to identify the data points more clearly.

adding noise to signal

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
x=np.arange(0,2*(np.pi),0.1)
y=np.sin(x)
sigma=0.05 # noise variance
y_noisy=y+sigma*np.random.randn(y.size)
plt.plot(x,y_noisy)
plt.plot(x,y)
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('A noisy signal')
plt.show()
```



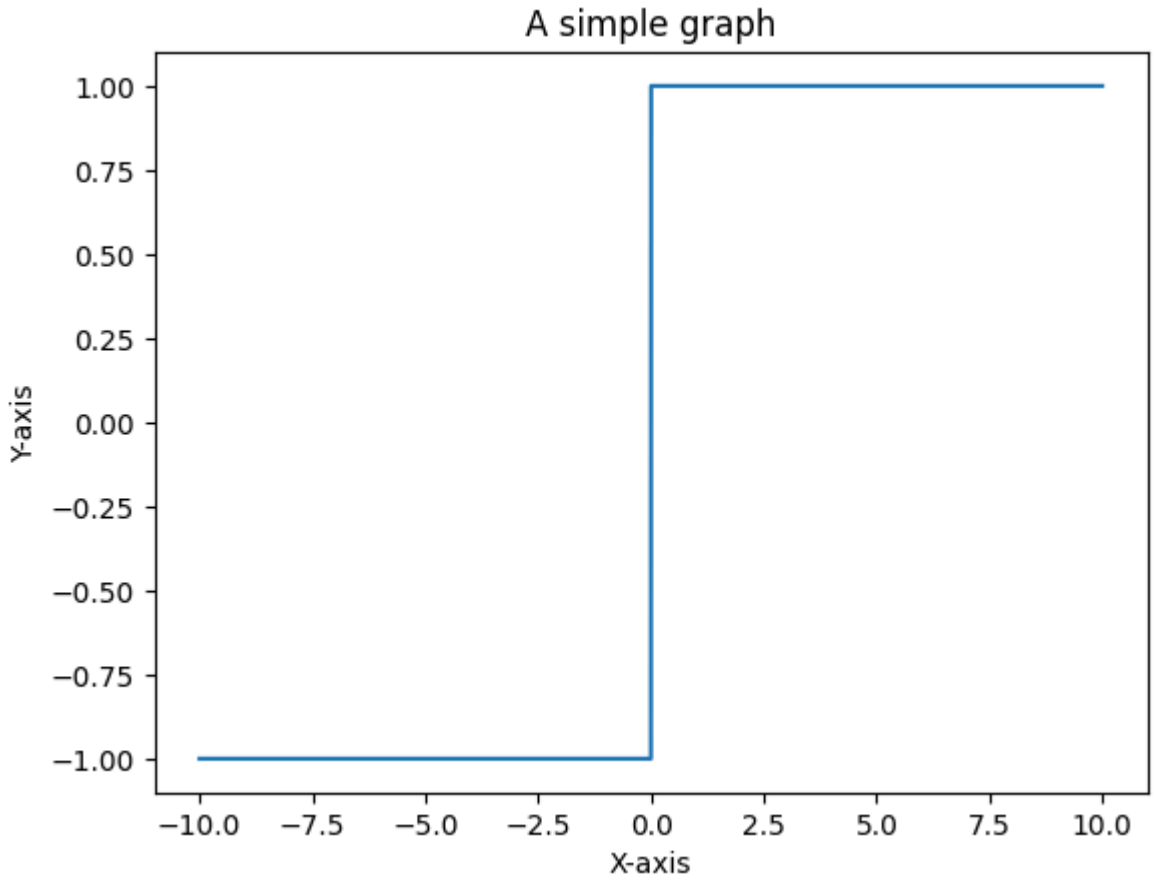
Observations:

- The plot visualizes a sine wave along with its noisy version.
- The x-axis represents the input values from 0 to 2π, while the y-axis shows the corresponding amplitudes.

- The noisy signal, obtained by adding random noise to the sine wave, appears as a slightly distorted version of the original smooth wave.
- The original sine wave is also plotted, providing a clear comparison between the ideal signal and the noisy one.
- The plot effectively demonstrates the impact of noise on a periodic signal.

Plot of Signum function

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-10, 10, 0.001)
y = np.sign(x)
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('A simple graph')
plt.show()
```



Observations:

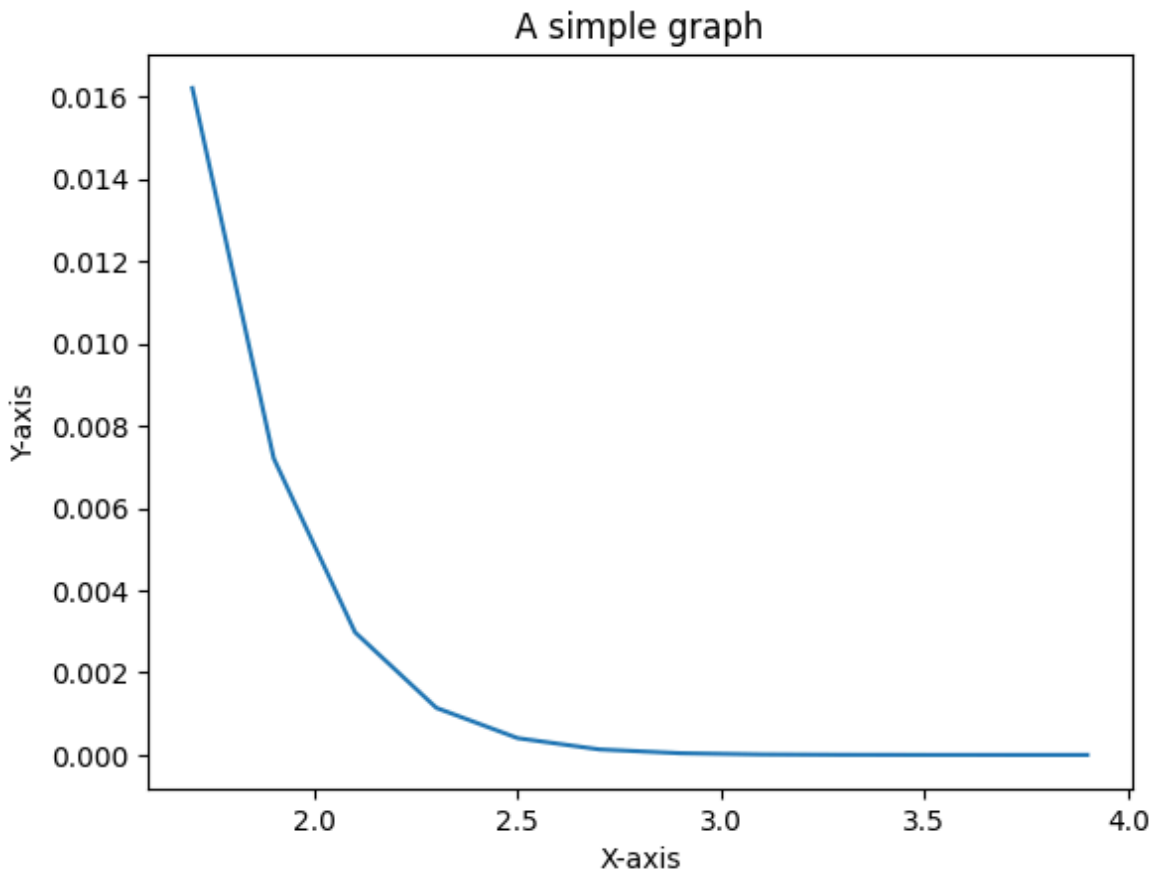
- The plot illustrates the behavior of the sign function, which has three distinct output regions: negative values for inputs less than zero, zero at the origin, and positive values for inputs greater than zero.
- The graph shows a sharp transition at the origin, creating a step-like structure.
- The x-axis represents the input values, and the y-axis shows the output of the function, highlighting its discontinuous nature.

- This visualization clearly depicts the abrupt change in the function's value at the origin.

Plot of erfc(x) function

In [ ]:
```
import math
import matplotlib.pyplot as plt
import numpy as np

k=1.5
y=[0]*12
x=[0]*12
for i in range(0, 12):
 k=k+0.2
 y[i]= math.erfc(k)
 x[i]=k
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('A simple graph')
plt.show()
```



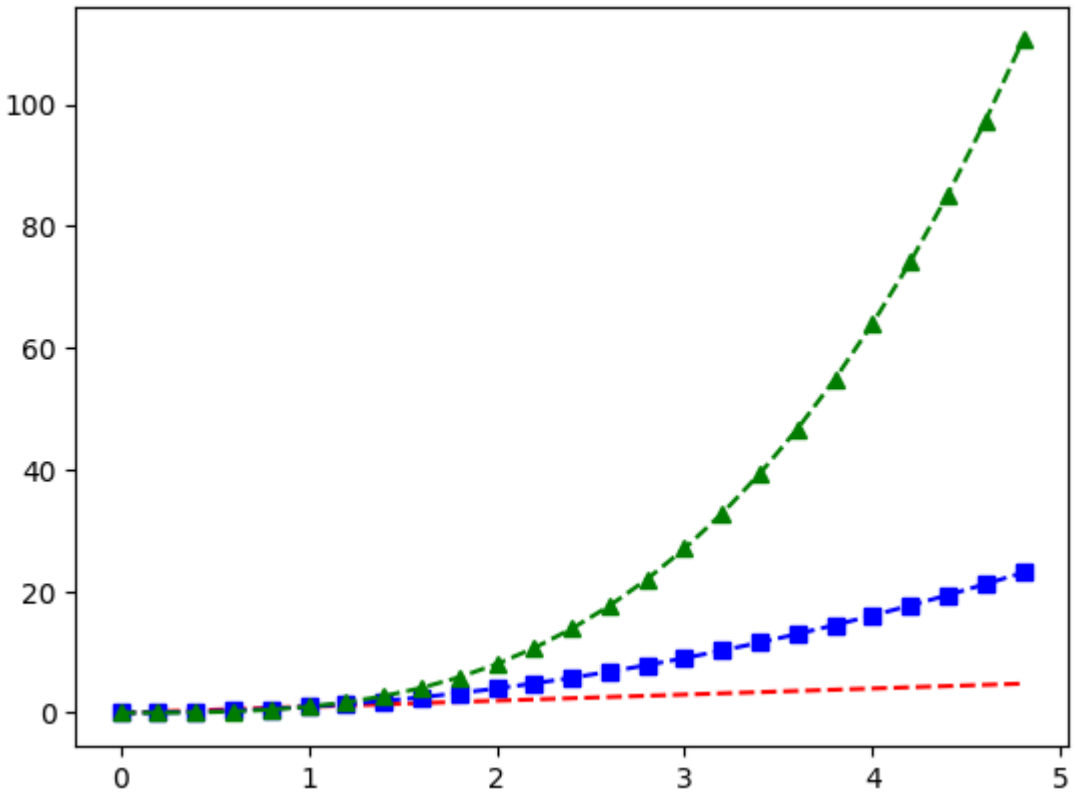
Observations:

- The plot visualizes the complementary error function y=erfc(k) as k increases.
- The values of k range from approximately 1.5 to 3.9, with increments of 0.2.
- The graph shows a monotonic decrease in y as k increases, consistent with the behavior of the complementary error function.

- The x-axis represents k, and the y-axis represents the corresponding erfc(k) values.
- The plot effectively demonstrates the diminishing nature of the complementary error function for larger values of k.

:linear, quadratic, and cubic functions graphs.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs--', t, t**3, 'g^--')
plt.show()
```



Observations:

The plot visualizes three functions of t:

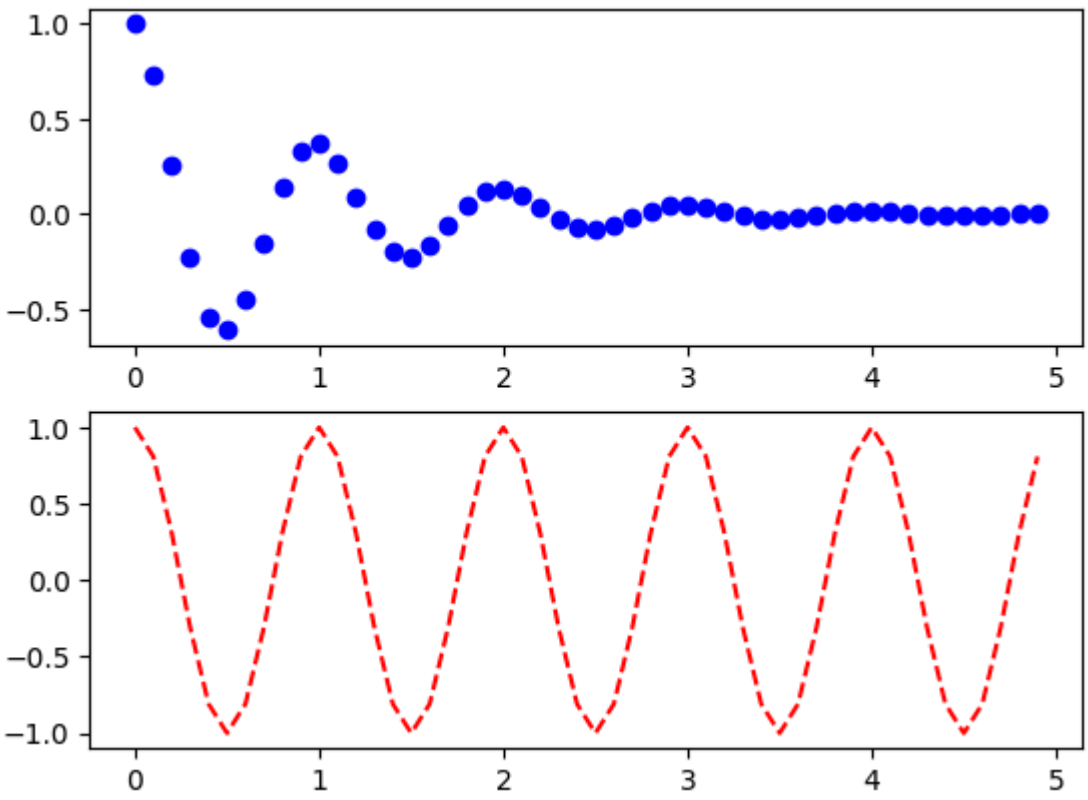
- Linear Function: y=t, represented by red dashed lines ('r--').
- Quadratic Function: y=t^2, represented by blue squares with dashed lines ('bs--').
- Cubic Function: y=t^3, represented by green triangles with dashed lines ('g^--').

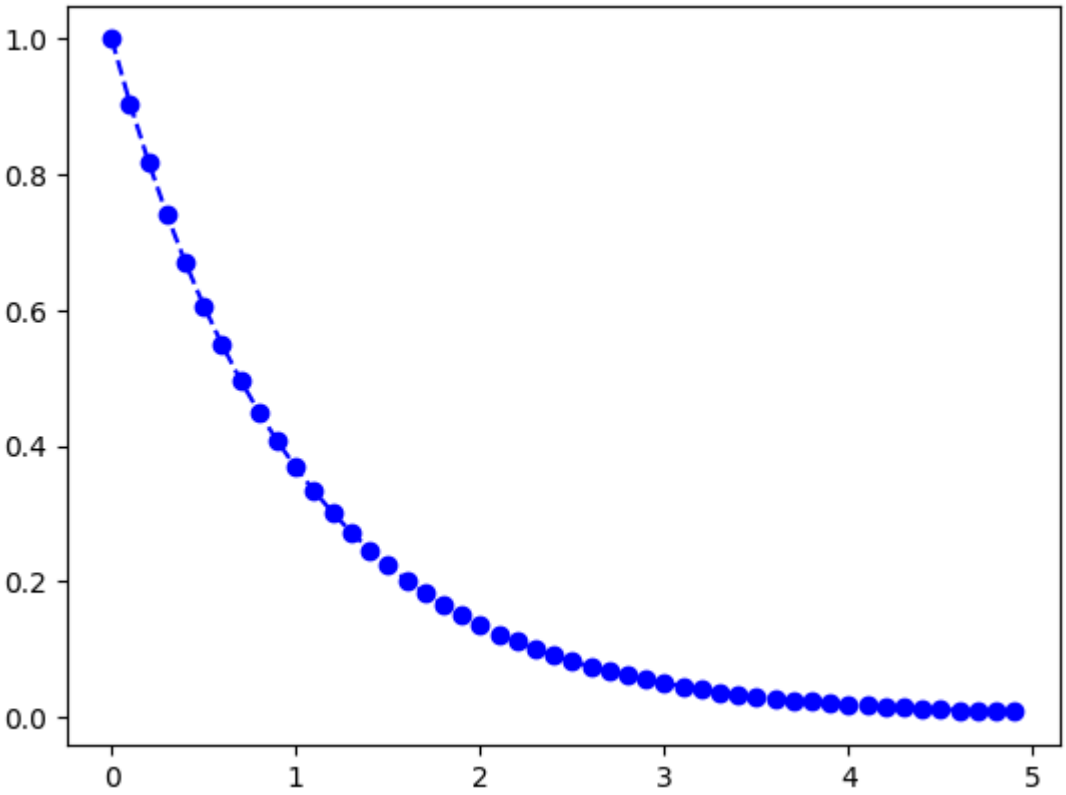
The plot highlights the differences in growth rates of these functions as t increases.

A damped oscillation vs. a pure oscillation.

Exponential decay.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 5.0, 0.1)
y1 = np.exp(-t) * np.cos(2*np.pi*t)
y2 = np.cos(2*np.pi*t)
y3 = np.exp(-t)
plt.figure(1)
plt.subplot(211)
plt.plot(t, y1, 'bo')
plt.subplot(212)
plt.plot(t, y2, 'r--')
plt.figure(2)
plt.plot(t, y3, 'bo--')
plt.show()
```





Observations:

Figure 1, Subplot 1:

- The plot displays a damped cosine wave y_1 = e^(-t) cos(2πt).
- The amplitude decays exponentially over time due to the e^(-t) factor.
- Blue circle markers ('bo') represent the points, emphasizing the damping effect.

Figure 1, Subplot 2:

- The plot shows a pure cosine wave y_2 =cos(2πt).
- The wave oscillates with a constant amplitude and is visualized using a red dashed line ('r--').

Figure 2:

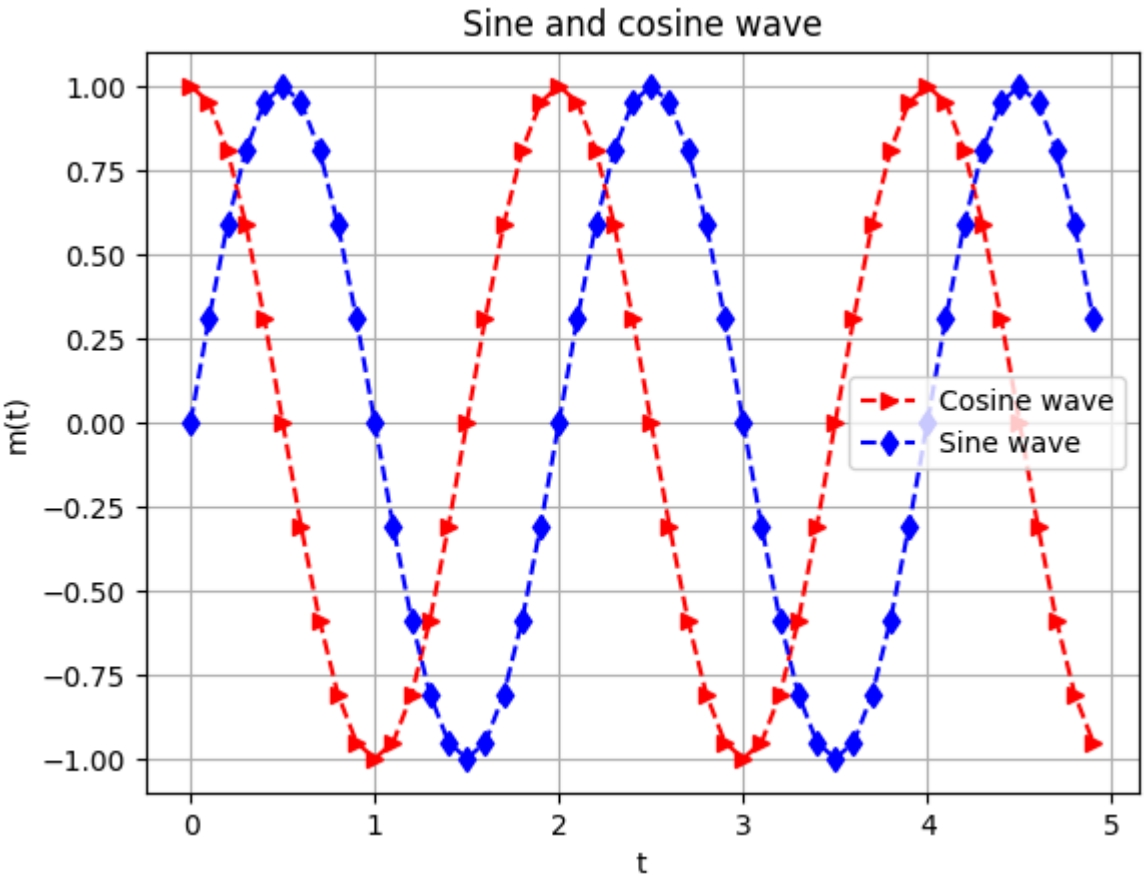
- The plot represents the exponential decay function y_3=e^(-t)
- Blue circle markers connected with a dashed line ('bo--') emphasize the smooth decay over time.

General Characteristics:

- The first figure highlights the effect of damping on a cosine wave, while the second figure illustrates the exponential decay separately.
- Multiple subplots and figures provide clear comparisons between the damped wave, pure wave, and decay function.
- The plots effectively demonstrate the interplay of exponential and trigonometric functions.

trigonometric wave comparison graph

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 5.0, 0.1)
y1 = np.cos(np.pi*t);
y2 = np.sin(np.pi*t);
plt.plot(t,y1, 'r>--',t,y2, 'bd--')
plt.xlabel('t')
plt.ylabel('m(t)')
plt.title('Sine and cosine wave')
plt.legend(['Cosine wave', 'Sine wave'])
plt.grid(True, which='both')
plt.show()
```



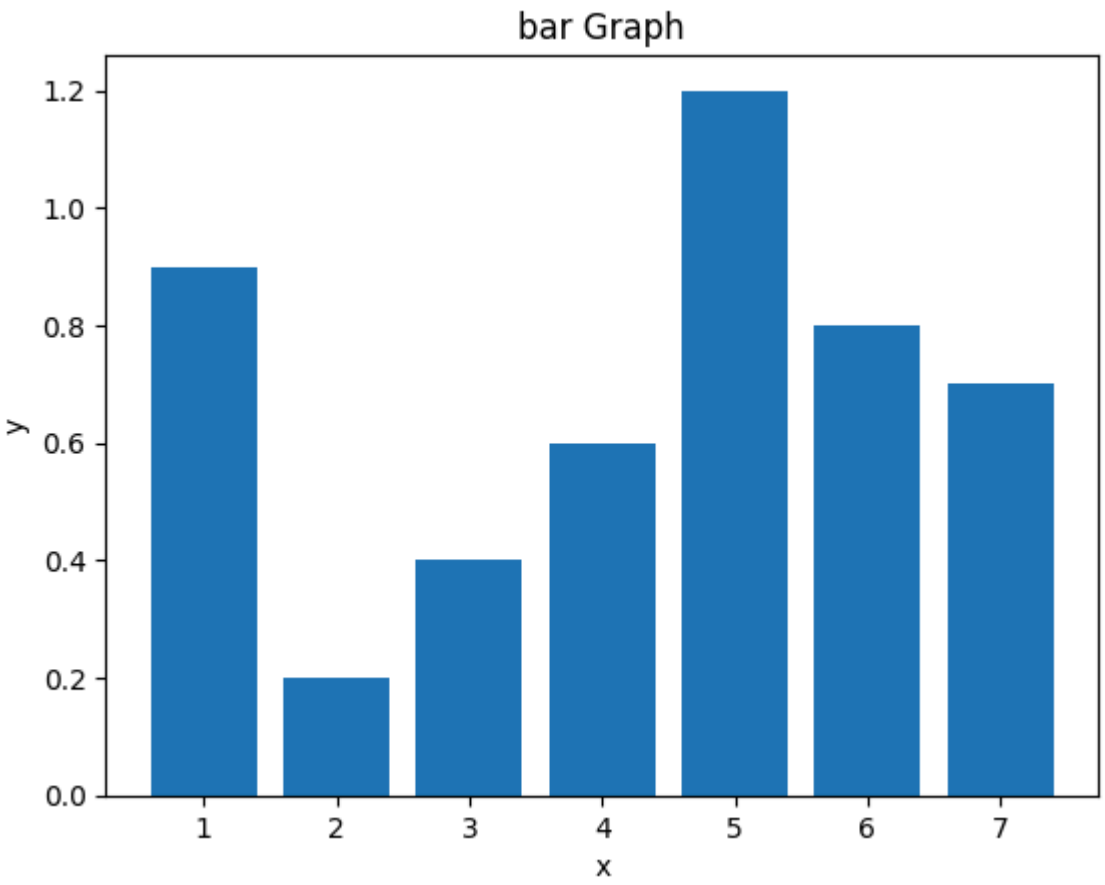
Observations: line.

- The plot visualizes a cosine wave and a sine wave over the time range from 0 to 5 with a step size of 0.1.
- The cosine wave is represented using red triangle markers ('r>--') with a dashed line.
- The sine wave is depicted with blue diamond markers ('bd--') and a dashed
- The graph is labeled with x-axis (time) and y-axis (amplitude) along with a title "Sine and cosine wave".
- A legend distinguishes the cosine wave from the sine wave, and a grid enhances readability by marking major and minor intervals.

- The plot effectively showcases the periodic and complementary nature of sine and cosine waves.

Bar Graph

In [ ]:
```
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt
x = [1,2,3,4,5,6,7]
y=[0.9, 0.2, 0.4, 0.6,1.2,0.8,0.7]
plt.bar(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('bar Graph')
plt.show()
```

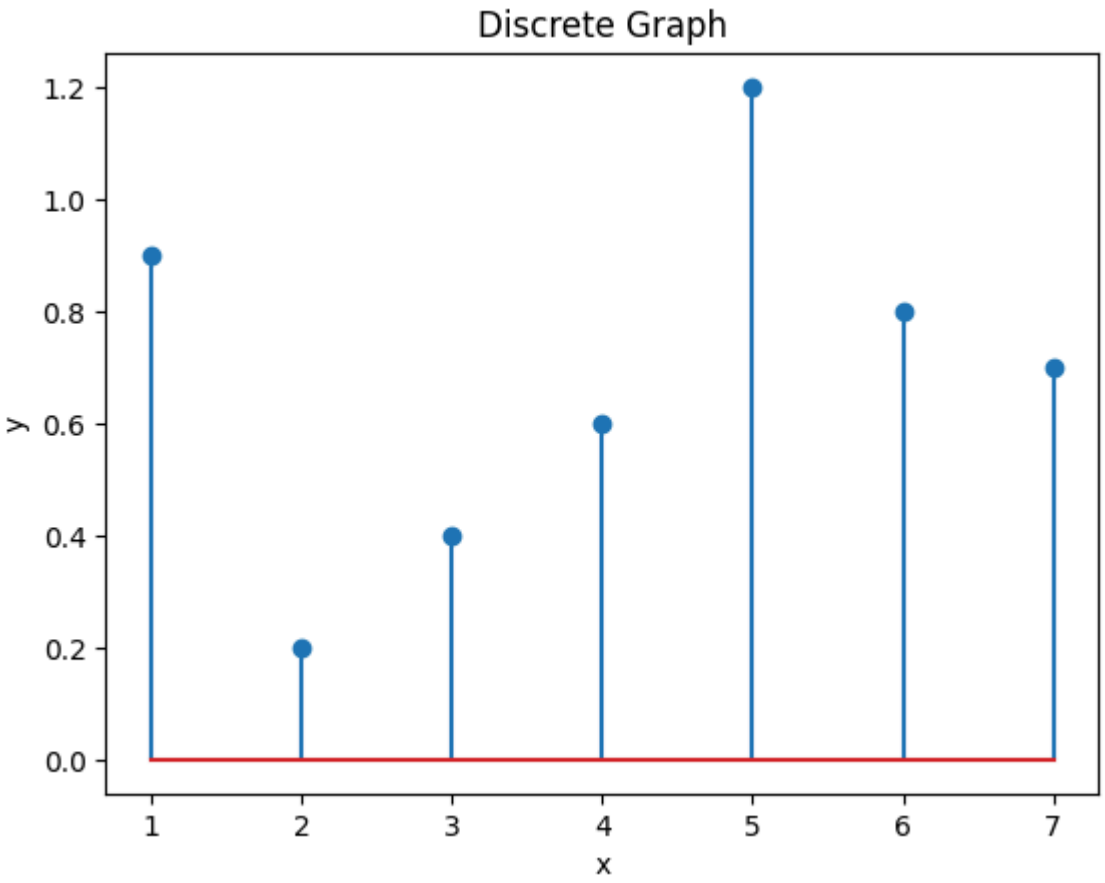


Observations:

- The plot displays a bar graph representing the relationship between x and y.
- Each bar corresponds to an x-value (1 to 7) and its respective y-value (ranging from 0.2 to 1.2).
- The height of each bar reflects the magnitude of the y-value for each x-coordinate.
- The graph is labeled with x and y axes and a title "bar Graph" for clarity.
- The bar graph effectively visualizes the data distribution and comparisons between individual y-values across the x-values.

Discrete Graph

In [ ]:
```
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt
x = [1,2,3,4,5,6,7]
y=[0.9, 0.2, 0.4, 0.6,1.2,0.8,0.7]
plt.stem(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Discrete Graph')
plt.show()
```



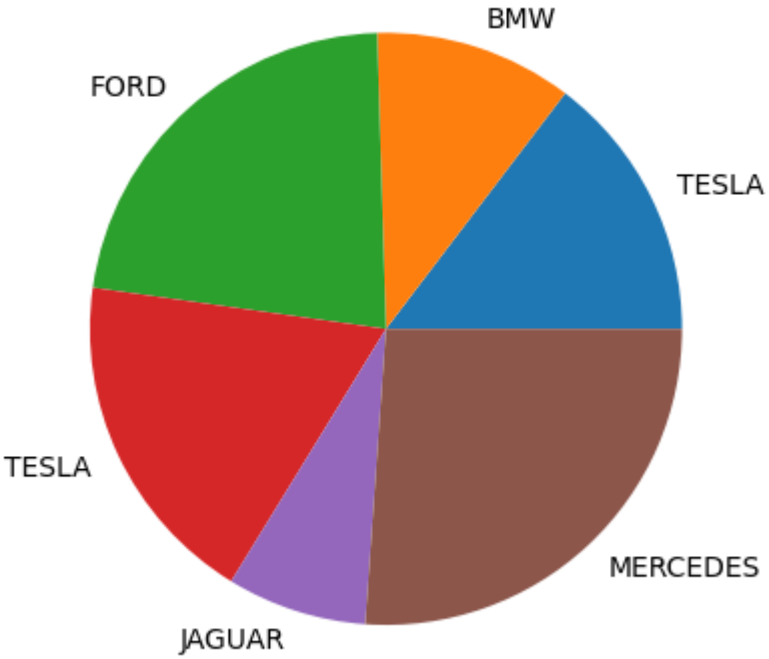
Observations:

- The plot displays a discrete graph using a stem plot to represent the relationship between x and y.
- The x-values are integers from 1 to 7, and the y-values vary between 0.2 and 1.2.
- Each data point is represented by a vertical line (stem) with a marker at the tip to indicate the corresponding yyy-value.
- The graph is labeled with x and y axes and a title "Discrete Graph" for clarity.
- The stem plot is useful for visualizing individual data points and their magnitude, especially in discrete datasets.

Pie Chart

In [ ]:
```
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt
cars = ['TESLA', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
```

```
data = [23, 17, 35, 29, 12, 41]
plt.pie(data, labels = cars)
# show plot
plt.show()
```

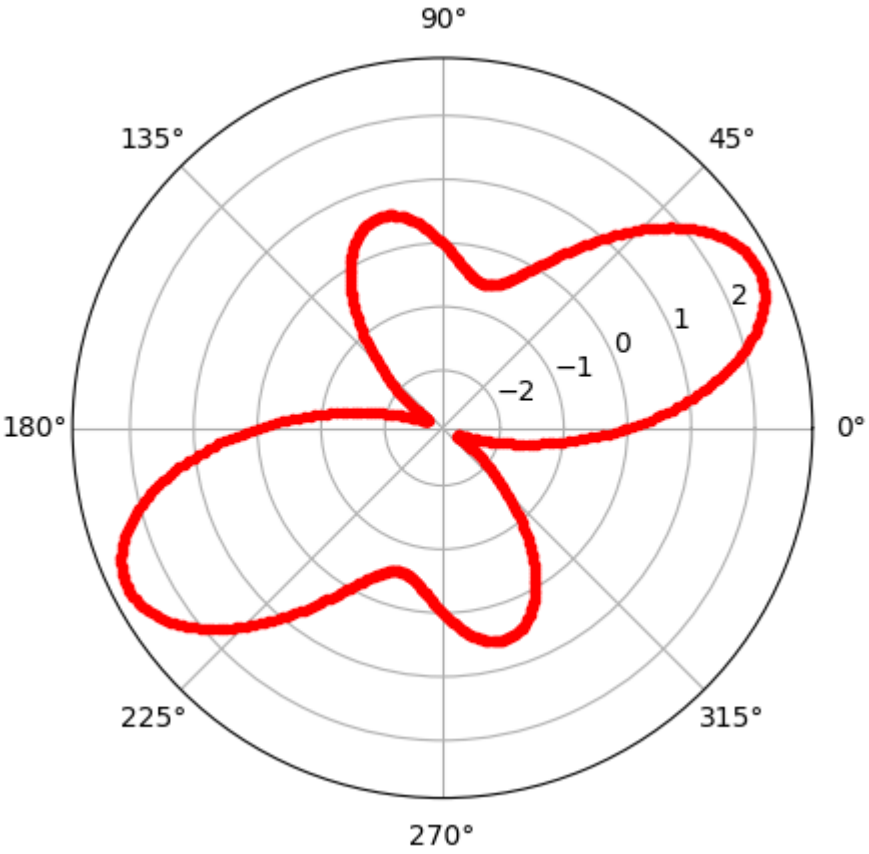


Observations:

- The plot visualizes the distribution of data values for different car brands using a pie chart.
- Each segment represents a car brand, with the size of the segment proportional to the corresponding data value.
- The brands displayed are TESLA, BMW, FORD, JAGUAR, and MERCEDES, with TESLA appearing twice.
- The pie chart helps in easily comparing the relative sizes of the data values for each brand.
- The chart is automatically labeled with the car brand names for clarity, though the actual percentages are not displayed

Polar plot of r = 3sin(3θ)cos(θ)

In [ ]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
# radian values of theta and corresponding value of r
angle = np.arange(0, (2 * np.pi), 0.01)
r=3*np.sin(3*angle)*np.cos(angle)
plt.polar(angle, r, 'r.')
# display the Polar plot
plt.show()
```

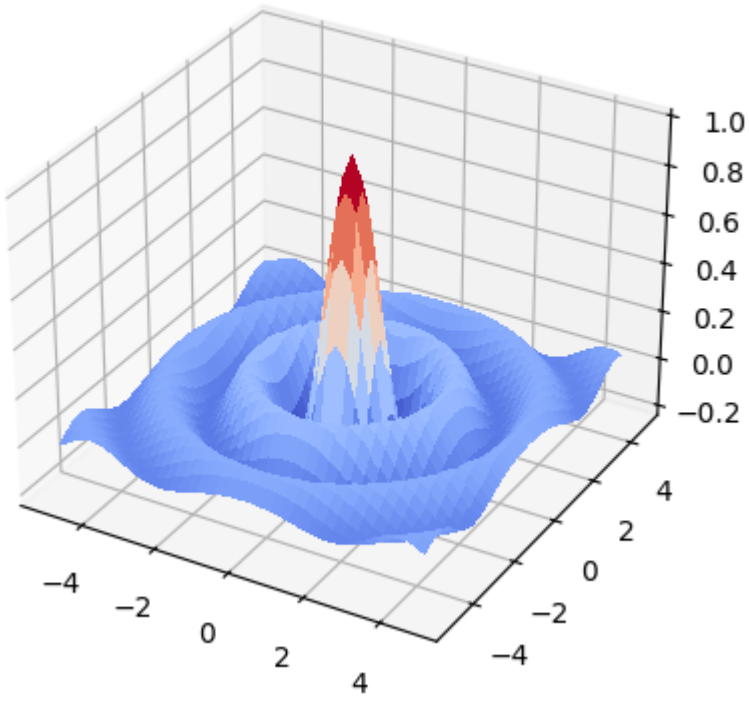


Observations:

- The plot visualizes the polar function $r = 3sin(3\theta)cos(\theta)$ with radial distance r as a function of angle θ.
- It exhibits 3-fold rotational symmetry due to the sin(θ) term, forming a flower-like structure.
- The radial distance oscillates between positive and negative values, creating alternating peaks and valleys.
- The plot uses red dots ('r.') to highlight each point, providing clear visualization.
- The polar coordinate system is used to represent the periodic nature of the function in a circular format.

surface plot

In [ ]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import numpy as np
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sinc(R)
ax.plot_surface(X, Y, Z, cmap = cm.coolwarm, linewidth = 0, antialiased =False)
plt.show()
```



Observations:

3D Surface Plot:

- The plot visualizes the sinc function Z=sinc(X2+Y2) as a smooth surface.

Radial Symmetry:

- The sinc function exhibits radial symmetry, with concentric oscillations around the origin.

Amplitude Decay:

- The amplitude of the surface decreases as the distance from the origin (RRR) increases, showcasing a smooth decay.

Peaks and Troughs:

- The plot shows periodic peaks and troughs that gradually diminish in size, forming a wavelike structure.

Color Mapping:

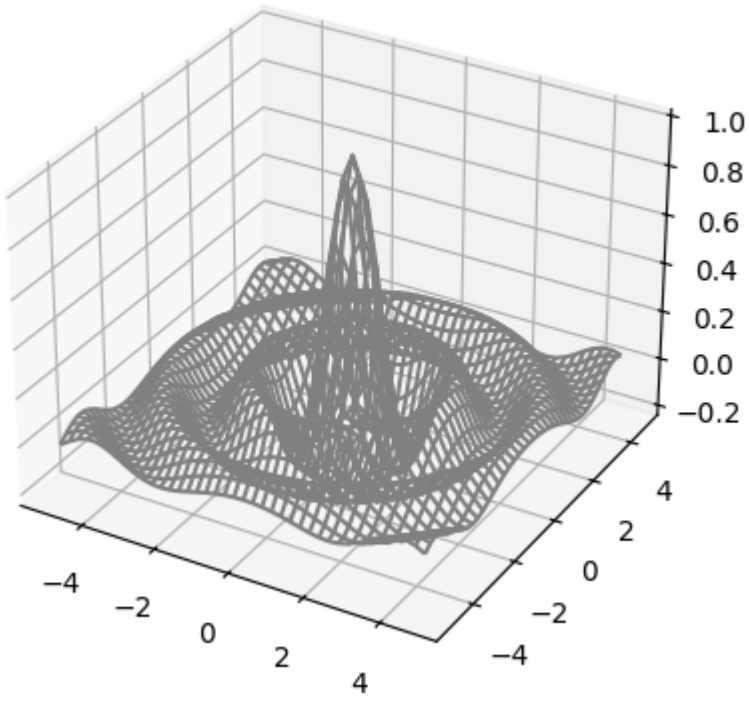
- The coolwarm colormap enhances visualization by assigning warm colors (reds) to higher values and cool colors (blues) to lower values.

Smoothness:

- The surface is smooth and antialiased, making the structure of the sinc function visually clear and appealing.

mesh (wireframe) plot.

In [ ]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import numpy as np
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sinc(R)
ax.plot_wireframe(X, Y, Z, color = 'gray')
plt.show()
```

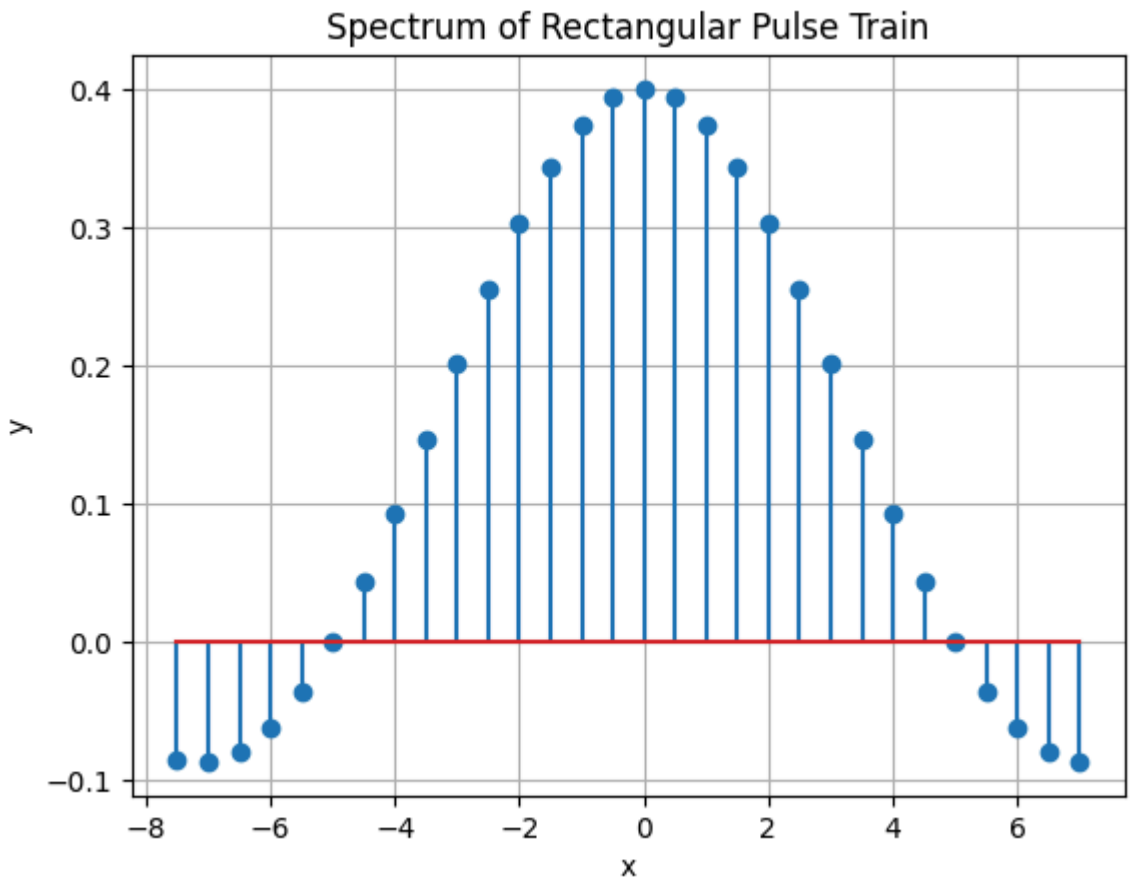


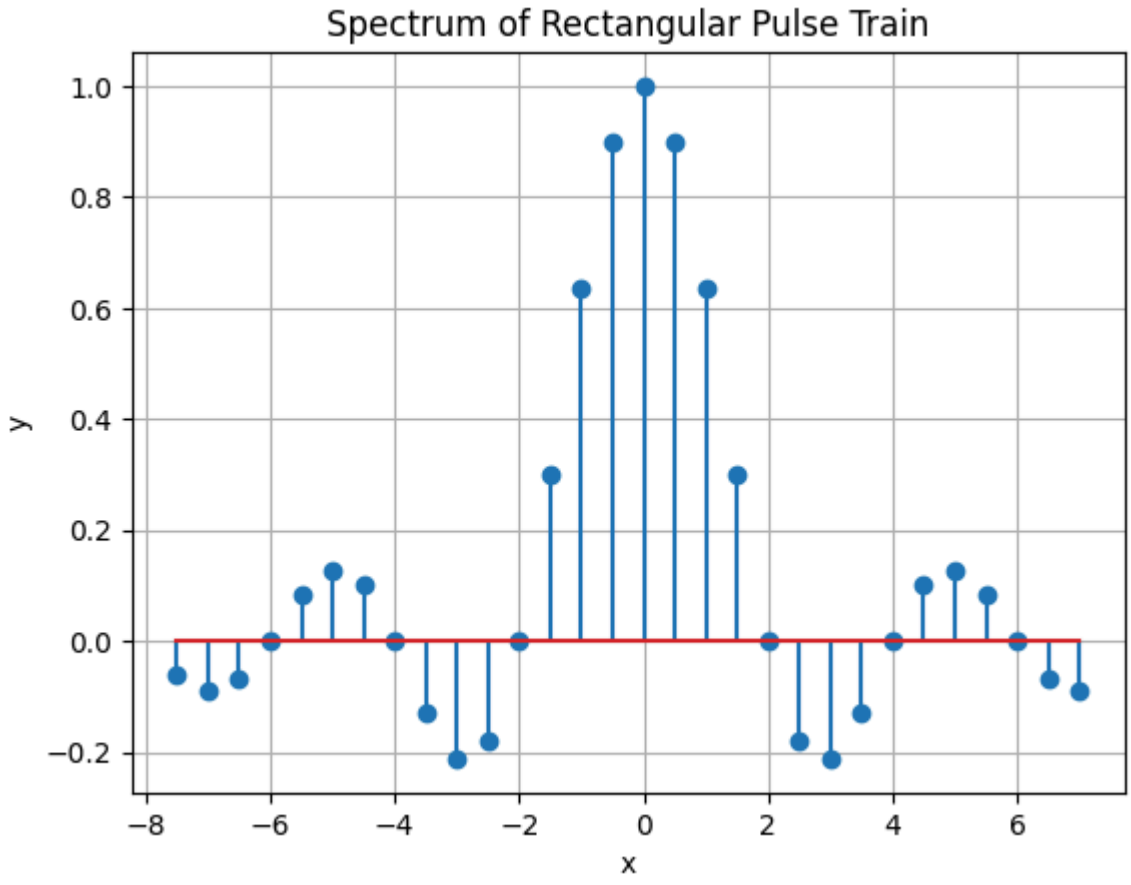
- The plot represents the 3D sinc function $Z = sinc(\sqrt{(X^2 + Y^2)})$
- It exhibits radial symmetry with oscillations forming concentric rings around the origin.
- Amplitude decreases as the distance from the origin increases, showcasing a smooth decay.
- The wireframe style provides clarity in visualizing peaks and troughs.
- Useful in applications like signal processing and waveform analysis.

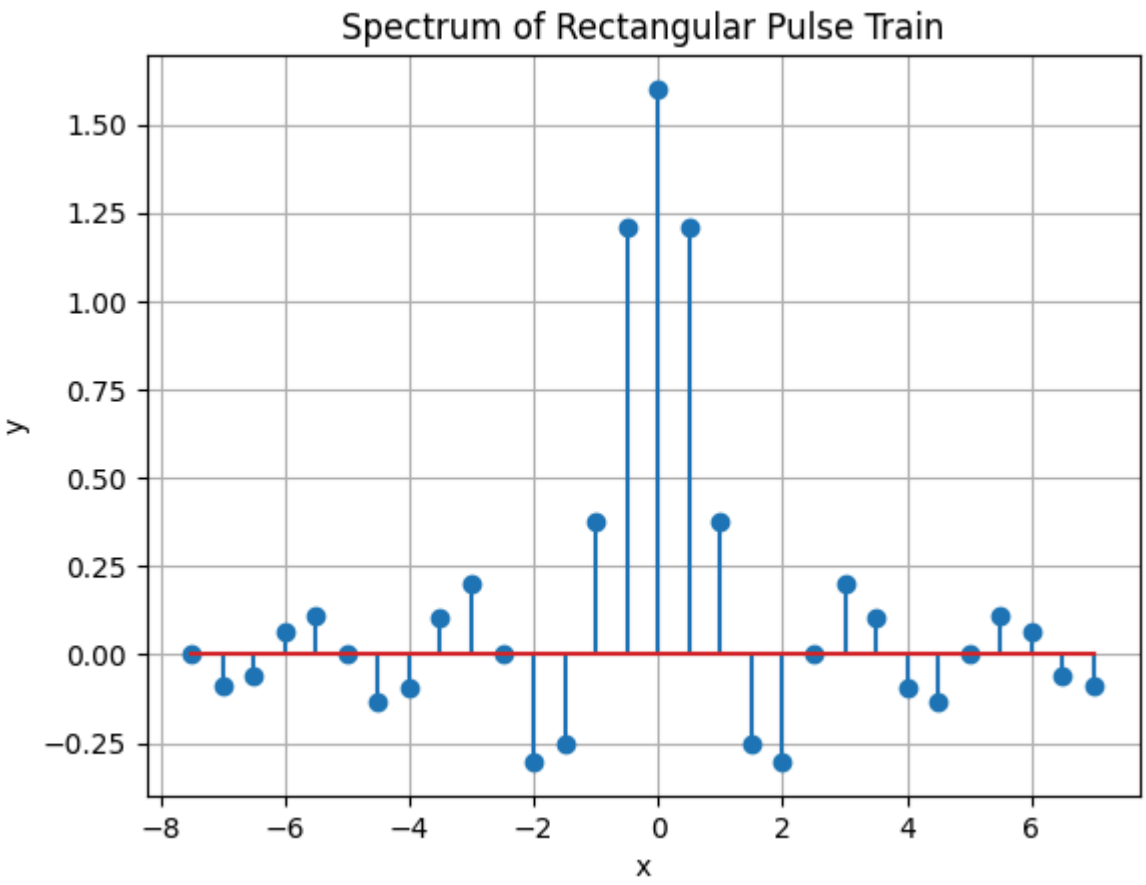
Lab 02 starts Here

Spectrum of Rectangular Pulse Train

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
#subplot 1
A=2
n = np.arange(-15, 15, 1)
T=2;
tau=0.2;
d=tau/T;
Cn= A*tau*np.sinc(n*tau/T)
plt.stem(n/T,Cn)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Spectrum of Rectangular Pulse Train')
plt.grid(True, which='both')
plt.show()
#subplot 2
A=2
n = np.arange(-15, 15, 1)
T=2;
tau=0.5;
d=tau/T;
Cn= A*tau*np.sinc(n*tau/T)
plt.stem(n/T,Cn)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Spectrum of Rectangular Pulse Train')
plt.grid(True, which='both')
plt.show()
#subplot 3
A=2
n = np.arange(-15, 15, 1)
T=2;
tau=0.8;
d=tau/T;
Cn= A*tau*np.sinc(n*tau/T)
plt.stem(n/T,Cn)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Spectrum of Rectangular Pulse Train')
plt.grid(True, which='both')
plt.show()
```


Spectrum of Rectangular Pulse Train
0.4
0.3
0.2
0.1
0.0
−0.1
y
−8
−6
−4
−2
0
2
4
6
x


Spectrum of Rectangular Pulse Train
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
−8
−6
−4
−2
0
2
4
6
x



- The code demonstrates the trade-off between the time-domain pulse width (τ) and the frequency-domain bandwidth. A wider pulse in time results in a narrower frequency spectrum, and vice versa. This aligns with the time-frequency duality principle.

Observations:

Pulse Width and Spectrum Relationship:

- As the pulse width (τ) increases, the spectrum narrows, concentrating energy in lower frequencies.
- As τ decreases, the spectrum widens, spreading energy to higher frequencies.

Energy Distribution:

- For smaller τ, higher-frequency components are more prominent.
- For larger τ, most energy is concentrated in the central, lower frequencies.

Amplitude Behavior:

- The amplitude of the spectrum is proportional to both A (pulse amplitude) and τ (pulse width).

Side Lobes:

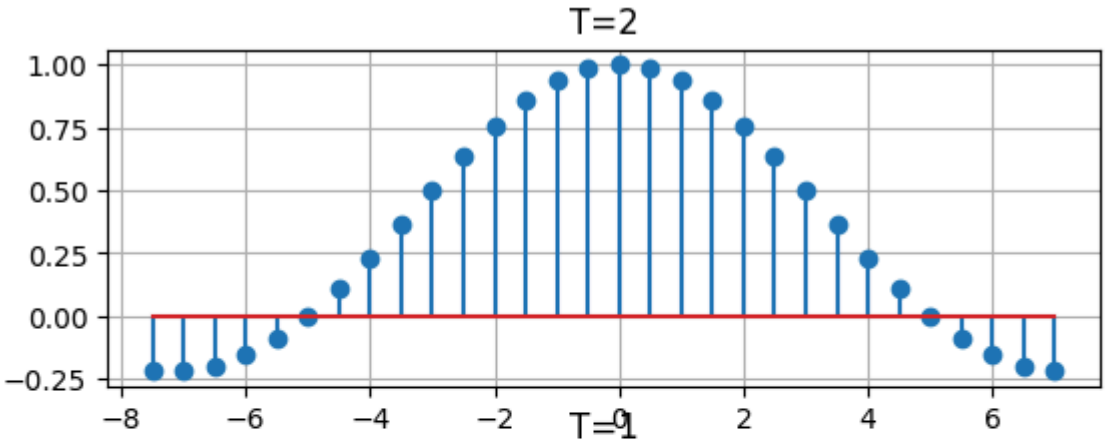
- Side lobes are more pronounced for narrower pulses (smaller τ) and diminish for wider pulses.

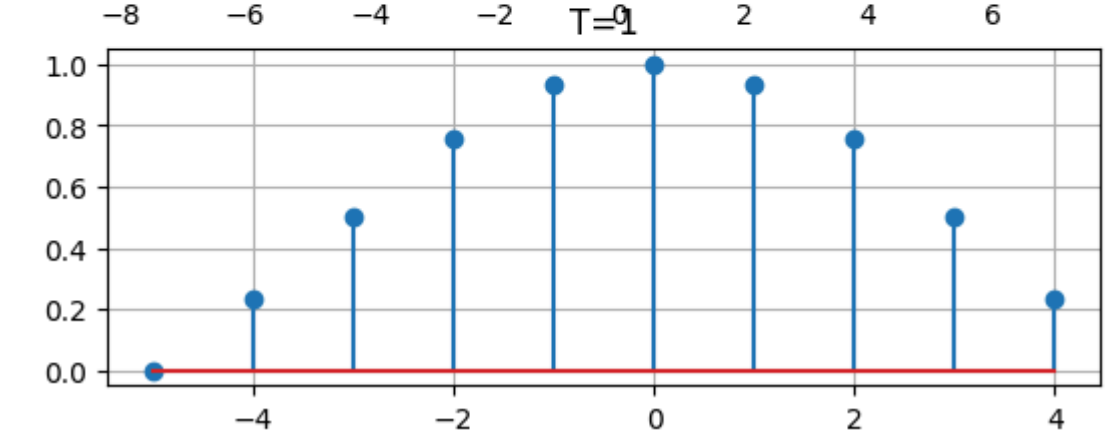
Symmetry:

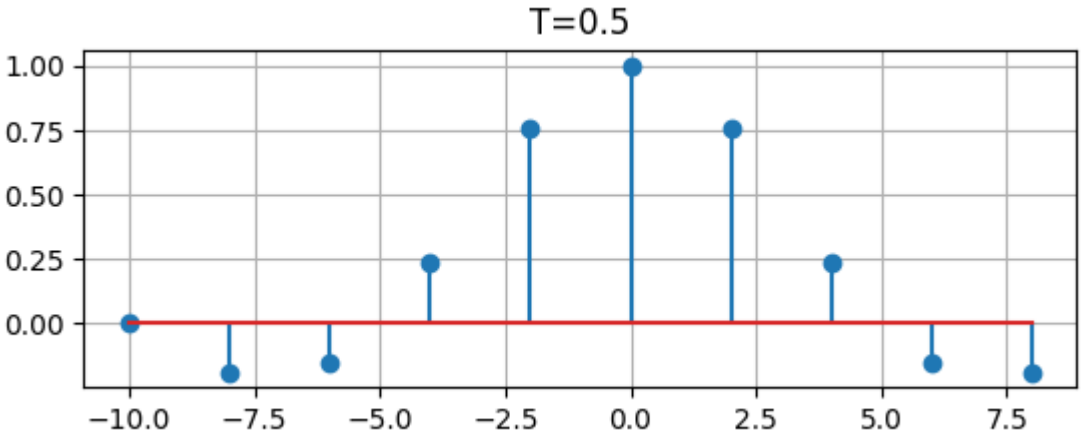
- The spectrum is symmetric about n=0, reflecting the symmetry of the rectangular pulse in time.

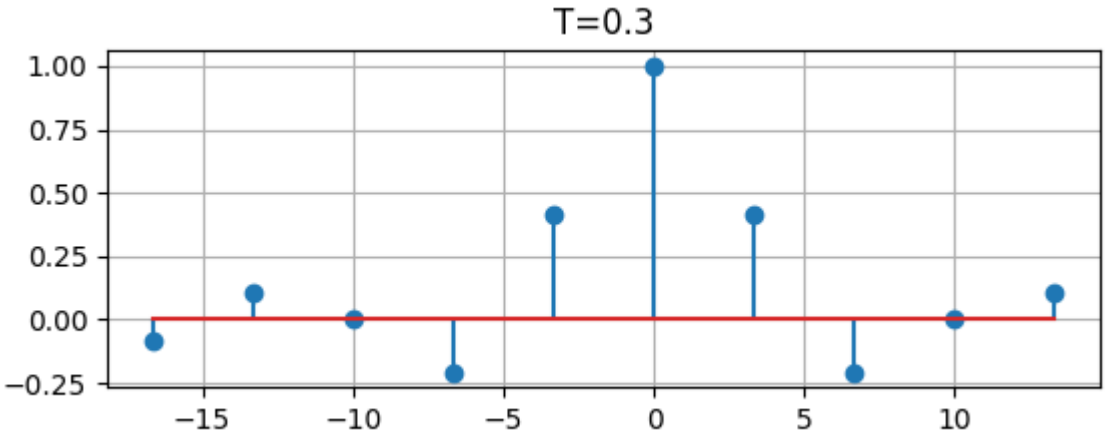
Impact of T

In [ ]:
```
import numpy as np
from matplotlib import pyplot as
    #subplot 1
plt.subplot(211)
n = np.arange(-15, 15, 1)
T=2;
tau=0.2;
d=tau/T;
y= np.sinc(n*d);
plt.stem(n/T,y)
plt.title('T=2')
plt.grid(True, which='both')
    #subplot 2
plt.subplot(212)
n = np.arange(-5, 5, 1)
T=1;
tau=0.2;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('T=1')
plt.grid(True, which='both')
plt.show()
    #subplot 3
plt.subplot(212)
n = np.arange(-5, 5, 1)
T=0.5;
tau=0.2;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('T=0.5')
plt.grid(True, which='both')
plt.show()
    #subplot 4
plt.subplot(212)
n = np.arange(-5, 5, 1)
T=0.3;
tau=0.2;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('T=0.3')
plt.grid(True, which='both')
plt.show()
```







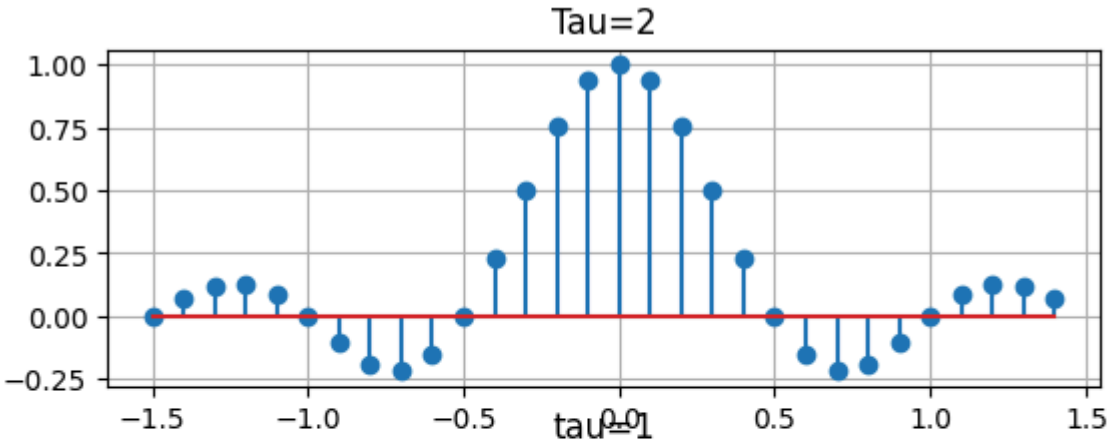


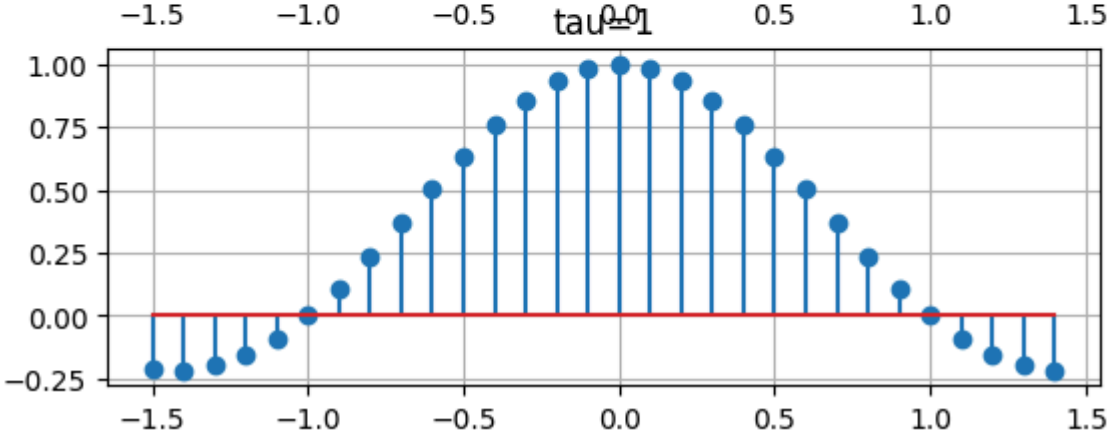
Observation :

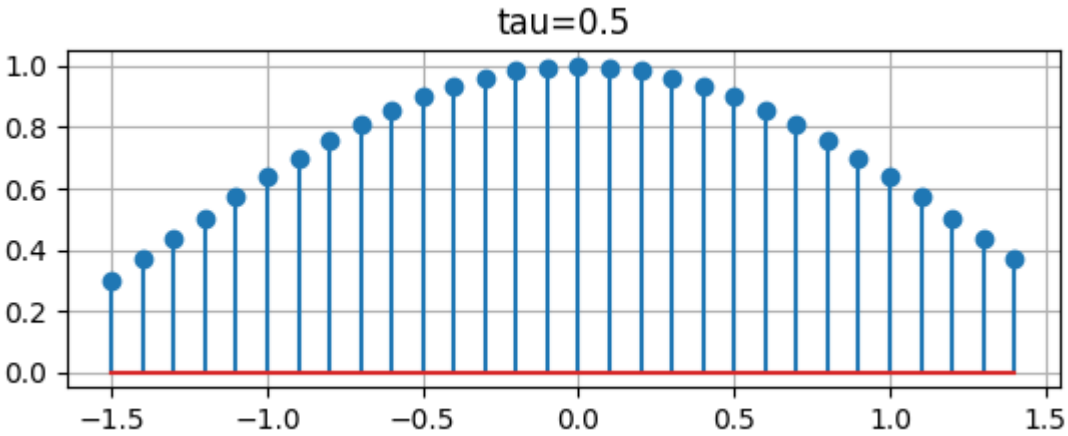
- As *T* increasing the graph getting **dense**.
- *T* reducing make the graph **sparse**.

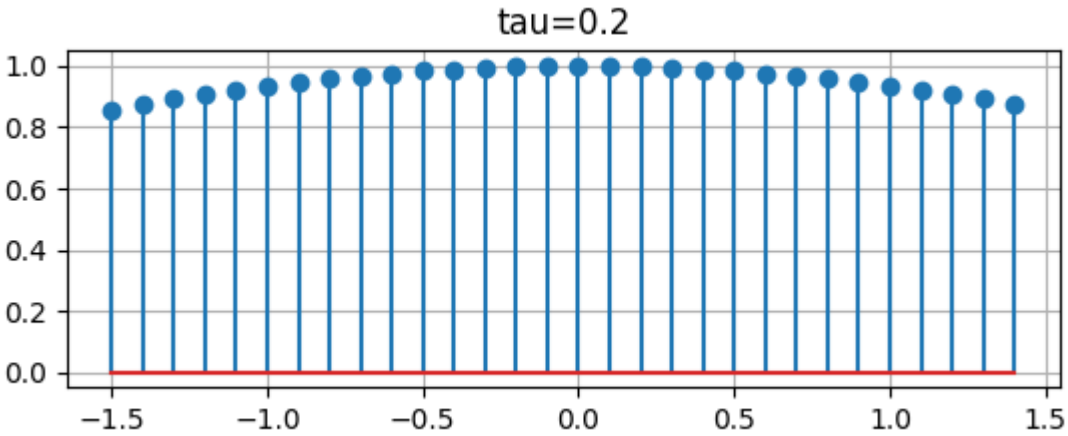
impact of tau

In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt
    #subplot 1
plt.subplot(211)
n = np.arange(-15, 15, 1)
T=10;
tau=2;
d=tau/T;
y= np.sinc(n*d);
plt.stem(n/T,y)
plt.title('Tau=2')
plt.grid(True, which='both')
    #subplot 2
plt.subplot(212)
tau=1;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('tau=1')
plt.grid(True,  which='both')
plt.show()
    #subplot 3
plt.subplot(212)
tau=0.5;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('tau=0.5')
plt.grid(True,  which='both')
plt.show()
    #subplot 4
plt.subplot(212)
tau=0.2;  #reduced
d=tau/T;
y=np.sinc(n*d);
plt.stem(n/T,y)
plt.title('tau=0.2')
plt.grid(True,  which='both')
plt.show()
```









Observation :

- As *Tau* reducing the graph getting **wider**.
- *Tau* reducing make the **crossing zero points further**.
- Higher *Tau* makes more **zero crossing points closer.**