ECE 438 - Laboratory 1
Discrete and Continuous-Time Signals

Last Updated on January 13, 2022

1. Introduction

The purpose of this lab is to illustrate the properties of continuous and discrete-time signals using digital computers and the [Python 3 (https://www.python.org/)](https://www.python.org/) environment. A continuous-time signal takes on a value at every point in time, whereas a discrete-time signal is only defined at integer values of the “time” variable. However, while discrete-time signals can be easily stored and processed on a computer, it is impossible to store the values of a continuous-time signal for all points along a segment of the real line. In later labs, we will see that digital computers are actually restricted to the storage of quantized discrete-time signals. Such signals are appropriately known as digital signals.

How then do we process continuous-time signals? In this lab, we will show that continuous-time signals may be processed by first approximating them by discrete-time signals using a process known as sampling. We will see that proper selection of the spacing between samples is crucial for an efficient and accurate approximation of a continuous-time signal. Excessively close spacing will lead to too much data, whereas excessively distant spacing will lead to a poor approximation of the continuous-time signal. Sampling will be an important topic in future labs, but for now we will use sampling to approximately compute some simple attributes of both real and synthetic signals.

2. Laboratory Ethics

Students are expected to behave ethically both in and out of the lab. Unethical behavior includes, but is not limited to, the following:

- Possession of another person's laboratory solutions from the current or previous years.
- Reference to, or use of another person's laboratory solutions from the current or previous years.
- Submission of work that is not done by your laboratory group.
- Allowing another person to copy your laboratory solutions or work.
- Cheating on quizzes.

The ECE 438 laboratory experience is meant to strengthen and deepen the student's understanding of basic concepts taught in the ECE 438 lectures and to help the student develop practical skills in applying the concepts taught in the ECE 438 course. The rules of laboratory ethics are designed to facilitate these goals. We emphasize that laboratory teaching assistants are available throughout the week to help the student both understand the basic concepts and answer the questions being asked in the laboratory exercises. By performing the laboratories independently, students will likely learn more and improve their performance in the course as a whole.

Please note that it is the responsibility of the student to make sure that the content of their graded laboratories is not distributed to other students. If there is any question as to whether a given action might be considered unethical, please see the professor or the TA before you engage in such actions.

Exercise 2.1

Please **write** the following statement in your lab report, sign and date.

"I have read and understood the Laboratory Ethics section (Section 2) of Laboratory 1. I pledge to behave ethically and with honesty in ECE 438 this semester. The reports I will hand in will be the product of original work by myself and my teammate, and no one else. I will not look at other people's laboratory. I will not use other people's code. I will not make my labs available to other students beyond my teammates, even after the semester is over. In particular, I will not post my labs on the Internet or make my files available to other people. I will not be a cheater."

Printed Name

Date

3 Introduction to Python 3

In this semester, most of the lab tasks in the ECE 438 lab will be performed using Python 3 instead of MATLAB.This section will be focusing on introducing the libraries and the basic functions in Python 3 only. The introduction of Simulink should be found in Laboratory 3.

To run and perform these labs you will need Jupyter Notebook.This course utilizes the Anaconda environment since it comes with Jupyter and a Python text editor. Anaconda will be used for the laboratory computers but you are welcome to use any other alternative for your own devices, noting that you will still be required to write your experiments in a Jupyter Notebook. Appendices A, B and C cover how to install Anaconda on 3 separate operating systems and will also provide you with a list of Python 3 libraries that you will need to install on your own device if you so choose to work on the experiments on your own device. It is still heavily recommended that you work on these experiments using the laboratory computers and/or connect remotely to them.

3.1 Python libraries tutorial

In this lab, two Python libraries will be heavily used, ***NumPy*** and ***Matplotlib***.

NumPy (https://numpy.org) supports multi-dimensional arrays and matrices, and provides numerous mathematical functions to be operated on these arrays and matrices.

Matplotlib (https://matplotlib.org) is a plotting library, and in this lab, we will be mainly using ***matplotlib.pyplot***, which provides a MATLAB-like way of plotting. Please refer to Pyplot Tutorial (https://matplotlib.org/tutorials/introductory/pyplot.html) for more detailed instructions.

First of all, let's import these libraries.

Execute the following blocks of code by pressing `shift` + `enter`.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import lab01 as lab01 # this imports the file lab01.py
```

In [2]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

You might notice that in the first block, the modules were imported using the syntax `import [module_name] as [alias]`.

Note that **np** and **plt** are the conventional aliases of **NumPy** and **matplotlib.pyplot**, respectively, and we will be following the conventions throughout this and the remaining labs.

You might have noticed that the file `lab01.py` has been imported. Throughout the course of these labs, you'll be given similar files that will contain certain functions that you will need to complete the experiment. You should not edit the files, and you will only be asked to import their functions as use them based on a specific syntax that will be given to you.

3.1 Exercise: Python Libraries

The file `lab01.py` contains a function named `get_square` that has the following synatx:

```
def get_square(x):
    return x * x
```

This function simply takes an input value `x`, and returns its square.

1. Run the following cell block to understand how the functions should be called.

In [3]:
```
a = 10
a_square = lab01.get_square(a)
print(a_square)
```

```
100
```

3.2 Basic Operations

Like other programming languages, Python 3 provides basic types including integers, floats, strings and booleans.

`print` function is usually used to print out the specified string.

In [4]:
```
a = 10                  # initialize a with the value of 10
b = a + 3               # assign a + 3 to b
c = a * 3               # multiply a by 3
d = a / 3               # divide a by 3
e = a // 3              # integer division

print(a)
print(b)
print(c)
print(d)
print(e)
```

```
10
13
30
3.3333333333333335
3
```

In [5]:
```
a = 3
b = a ** 2           # a to the power of 2
c = a ** 0.5         # a to the power of 0.5, or square root of a
d = b == 9           # return True if b is 9, otherwise False
e = "Hello World!"   # e is a string

print(a)
print(b)
print(c)
print(d)
print(e)
```

```
3
9
1.7320508075688772
True
Hello World!
```

A string can also be formatted; see the examples below.

In [6]:
```
str1 = "The value of b is {}.".format(b)
str2 = "The values of b and c are {} and {}.".format(b, c)
str3 = "The values of d is {:.5f}.".format(c)  # keep only 5 numbers after the decimal point
str4 = "The values of d is {:.2f}.".format(c)  # keep only 2 numbers after the decimal point
print(str1)
print(str2)
print(str3)
print(str4)
```

```
The value of b is 9.
The values of b and c are 9 and 1.7320508075688772.
The values of d is 1.73205.
The values of d is 1.73.
```

For loop is also very important throughout all the labs. Run the following cell block to understand how *for* loop works in Python 3.

In [7]:
```
for i in range(5): # i starts from default 0, and ends at 4 (not 5!!!), default increment by 1
    print(i)

print("-----------")

for i in range(3, 8, 1): # i starts from 3, and ends at 7 (not 8!!!), increment by 1
    print(i)

print("-----------")

for i in range(3, 7, 2): # i starts from 3, and ends at 5 (not 6 or 7!!!), increment by 2
    print(i)
```

```
0
1
2
3
4
-----------
3
4
5
6
7
-----------
3
5
```

3.3. Matrices and Operations - NumPy

In Python 3, an one-dimension array can be easily defined with the help of NumPy. For example, `A = np.array([2, 4, 5, 3, 2, 10])` creates an array with a length of 6, where the elements are $2, 4, 5, 3, 2$ and 10.

To access the `i+1` th element, use the notation `A[i]`, where `i` is the index. **Please note that in Python 3, the index starts from 0.** If the index is negative, e.g., $-i$, then `A[-i]` is retrieving the `i` th element from the right.

In [8]:
```
A = np.array([2, 4, 5, 3, 2, 10])
N = len(A) # get the length of the array A
print("The array is {}.".format(A))
print("The length of the array is {}.".format(N))

i = 4
print("The {}th element from the left of of A is {}.".format(i + 1, A[i]))
i = -2
print("The {}th element from the right of A is {}.".format(abs(i), A[i]))
```

```
The array is [ 2  4  5  3  2 10].
The length of the array is 6.
The 5th element from the left of of A is 2.
The 2th element from the right of A is 2.
```

NumPy array supports slicing, which means retrieving a subarray from the original array. The basic slicing syntax is `v[i:j:k]`, where `i` is the starting index, `j` is the stopping index, and `k` is the step.

In [9]:
```
A = np.array([2, 4, 5, 3, 2, 10])
N = len(A)
v1 = A[2:4:1]
print("A = ", A)
print("v1 = ", v1)
print("Length of v1 is {} == 4 - 2".format(len(A)))
```

```
A =  [ 2  4  5  3  2 10]
v1 =  [5 3]
Length of v1 is 6 == 4 - 2
```

If the starting index is 0, you may ignore it.

In [10]:
```
A = np.array([2, 4, 5, 3, 2, 10])
N = len(A)
v1 = A[:4:1]
print("A = ", A)
print("v1 = ", v1)
```

```
A =  [ 2  4  5  3  2 10]
v1 =  [2 4 5 3]
```

If the ending index is the length of the array, you may also ignore it.

In [11]:
```
A = np.array([2, 4, 5, 3, 2, 10])
N = len(A)
v1 = A[2::1]
print("A = ", A)
print("v1 = ", v1)
```

```
A =  [ 2  4  5  3  2 10]
v1 =  [ 5  3  2 10]
```

If the step size is 1, you may ignore it.

In [12]:
```python
A = np.array([2, 4, 5, 3, 2, 10])
N = len(A)
v1 = A[2:4]
print("A = ", A)
print("v1 = ", v1)
```

```
A =  [ 2  4  5  3  2 10]
v1 =  [5 3]
```

Lastly, let's do some basic operations on the NumPy array.

In [13]:
```python
B = np.array([4, 2, 3, 2, 5, 9, 0, 10])
C = B * 2
D = B / 2
E = -B
F = B == 2

print("Multiplying every element of B by 2: {}".format(C))
print("Dividing every element of B by 2: {}".format(D))
print("Negating every element of B: {}".format(E))
print("The element of B equals 2: {}".format(F))
```

```
Multiplying every element of B by 2: [ 8  4  6  4 10 18  0 20]
Dividing every element of B by 2: [2.  1.  1.5 1.  2.5 4.5 0.  5. ]
Negating every element of B: [ -4  -2  -3  -2  -5  -9   0 -10]
The element of B equals 2: [False  True False  True False False False False]
```

4. Continuous-Time Vs. Discrete-Time

The introduction in Section 1 mentioned the important issue of representing continuous-time signals on a computer. In the following sections, we will illustrate the process of *sampling*, and demonstrate the importance of the *sampling interval* to the precision of numerical computations.

4.1. Displaying Continuous-Time and Discrete-Time Signals in Python 3

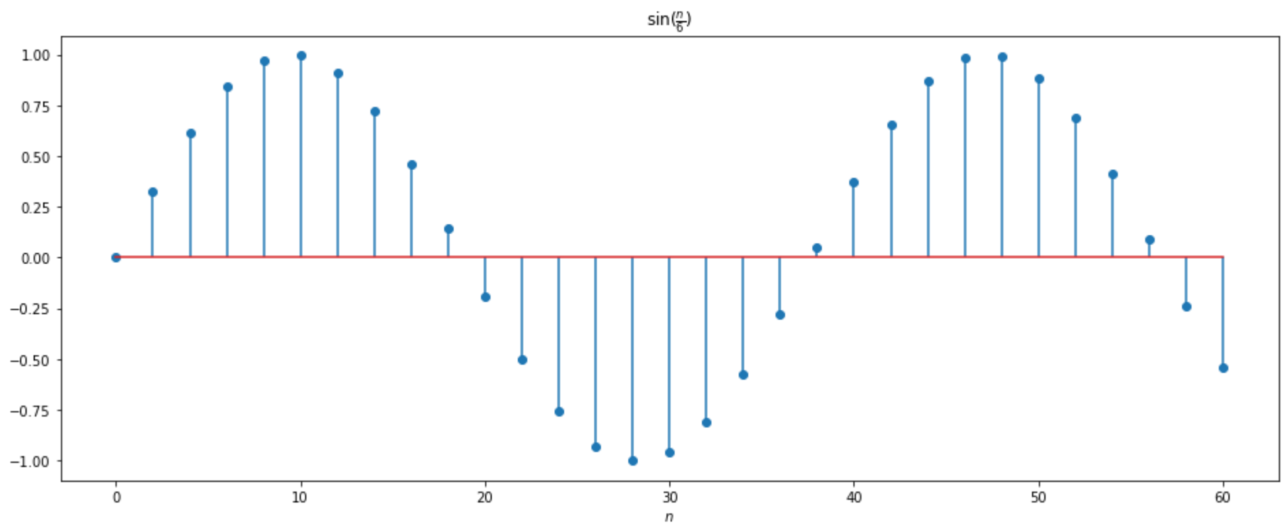
It is common to graph a discrete-time signal as dots in a Cartesian coordinate system. This can be done in Python 3 by using the *stem* function from *matplotlib.pyplot*.

1. Create an array with a length of 31, ranging from 0 to 60, and assign it to the variable n. You may find the function either np.linspace() (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html) or np.arange() (https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html) useful.
2. Apply $\sin$ function on this array and assign it to the variable y. You should use the $\sin()$ function in NumPy. Please refer to np.sin() (https://docs.scipy.org/doc/numpy/reference/generated/numpy.sin.html).
3. Stem plot this $\sin$ wave, and do not forget the title of the plot and the labels of the axes.

In [14]:
```
# 1
n = np.linspace(0, 60, 31) # create an NumPy array, [0, 2, 4, ..., 58, 60]
# or
n = np.arange(0, 61, 2)
# or
n = np.arange(61)[::2] # create an NumPy array, [0, 1, 2, ..., 59, 60], and then take every other

# 2
y = np.sin(n / 6)

# 3
plt.stem(n, y, use_line_collection=True)
plt.title(r"$\sin(\frac{n}{6})$")
plt.xlabel(r"$n$")
plt.show()
```



This plot shows the discrete-time signal formed by computing the values of the function $\sin(t/6)$ at points that are uniformly spaced at intervals of size $\frac{60-0}{31-1} = 2$.

A digital computer cannot store all points of a continuous-time signal since this would require an infinite amount of memory. It is, however, possible to plot a signal which *looks like* a continuous-time signal, by computing the value of the signal at closely spaced points in time, and then connecting the plotted points with lines. The `plt.plot` function may be used to generate such plots.

Exercise 4.1

1. Use the following instructions to generate two continuous-time plots of the signal: $\sin(t/6)$.

1. Initialize `n1` , a NumPy array starting from 0 and ending at 60, with a step size of 2 (i.e., 31 elements).
2. Initialize `z=sin(n1 / 6)` , using the function `np.sin(x)` .
3. Create the plot, using the function `plt.plot(x, y)` .

In [15]:
```
# your code goes here
```

1. Initialize `n2` , a NumPy array starting from 0 and ending at 60, with a step size of 10 (i.e., 7 elements).
2. Initialize `w=sin(n2 / 6)` , using the function `np.sin(x)` $.
3. Create the plot, using the function `plt.plot(x, y)` .

In [16]:
```
# your code goes here
```

2. Comment on the accuracy of each of the continuous time plots.

Your answer goes here

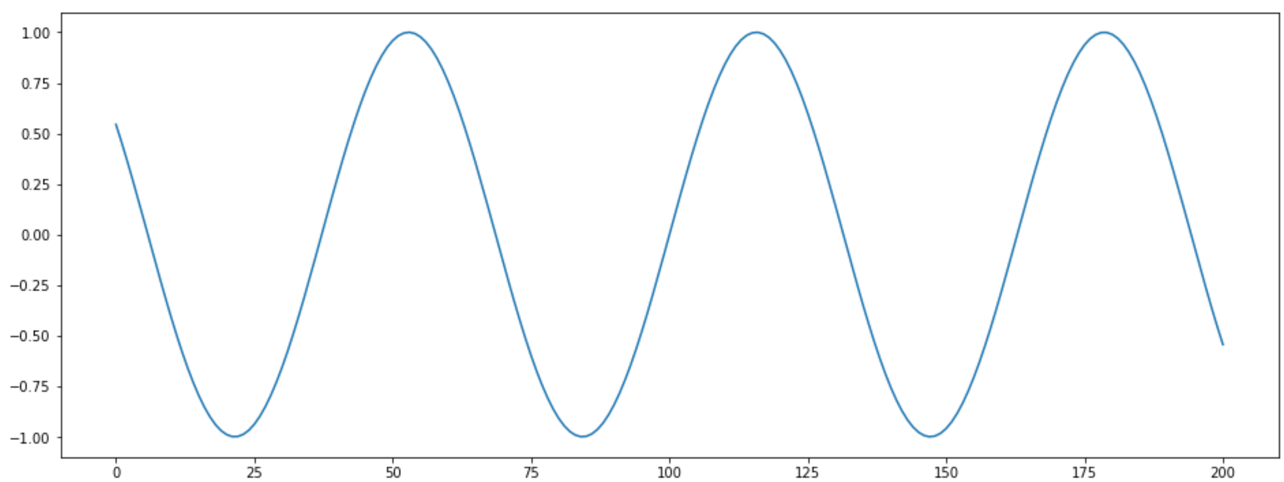
4.2. Vector Index versus Time

We saw in Section 4.1 that the samples of a continuous-time signal, say $x(t)$, can be stored in a NumPy array. It is common practice to use the same variable for the vector and the signal. So one often denotes the samples of $x(t)$ by $x[n]$, even though this is an abuse of notation and lacks rigor.

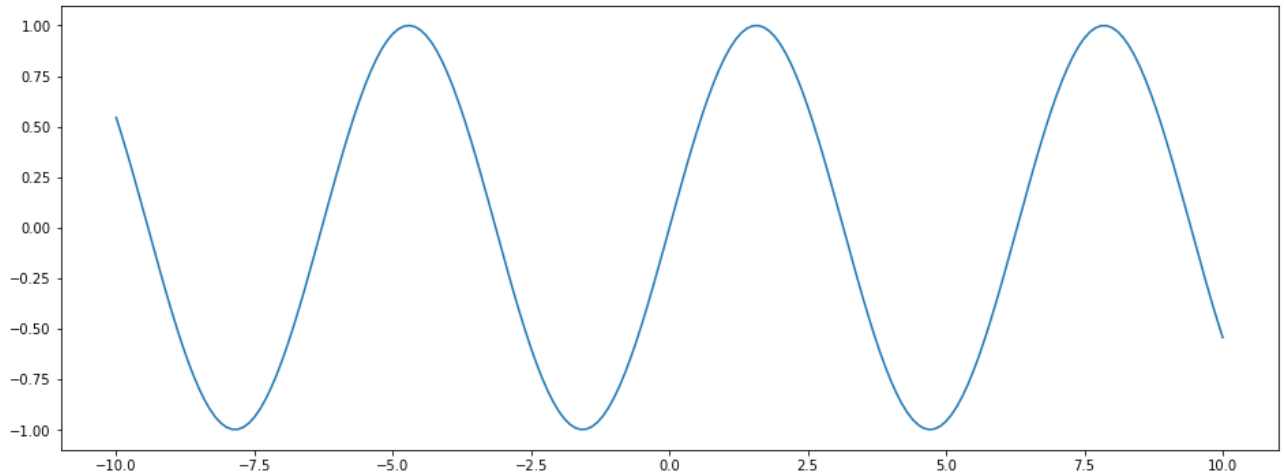
It is important not to confuse the index of a vector $x[n]$ with the value of the independent variable of a function $x(t)$. For example, Python 3 can be used to represent the function $x(t) = \sin(t)$ by sampling t at small intervals. The resulting samples may be stored in a NumPy array called `x` in your program. However, it is important to realize that the function x and the NumPy array `x` in the program are not the same things. The following code illustrates this.

In [17]:
```
t1 = np.linspace(-10, 10, 201)
x = np.sin(t1)
```

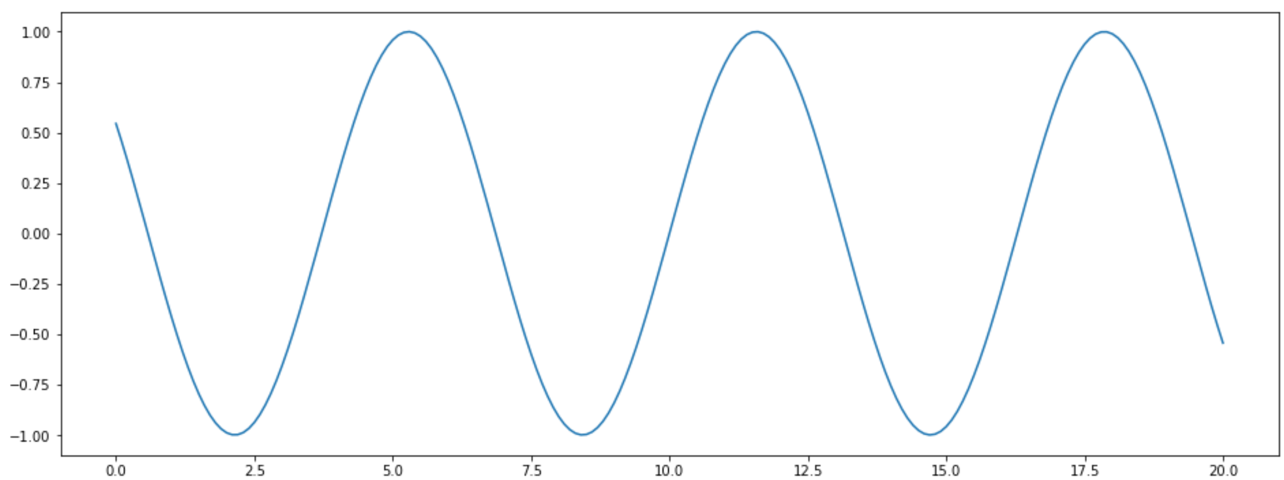
In [18]:
```
plt.plot(x)
plt.show()
```



In [19]:
```
plt.plot(t1, x)
plt.show()
```



In [20]:
```
t2 = np.linspace(0, 20, 201)
plt.plot(t2, x)
plt.show()
```



Exercise 4.2

1.Run the code above and explain the difference between the three signals represented.

Hint: Click on the cell and change it from being a `Markdown` type to `Code` type.

In [21]:
```
# your code goes here
```

your answer goes here

2. Write Python 3 code that would print the graph of $\sin(t)$ for the values of t on the interval [3.5, 4.5]. (Pick a suitable increment for t.)

```
In [22]: # your code goes here
```

Exercise 4.3

1. Compute these two integrals on a blank sheet of paper. Do the computations manually.

1. $\int_0^{2\pi} \sin^2(7t)dt$
2. $\int_0^{1} e^t dt$

4.4. Numerical Computation of Continuous-Time Signals

One common calculation on continuous-time signals is integration. Figure 1 illustrates a method used for computing the widely used Riemann integral. The Riemann integral approximates the area under a curve by breaking the region into many rectangles and summing their areas. Each rectangle is chosen to have the same width Δt, and the height of each rectangle is the value of the function at the start of the rectangle's interval.

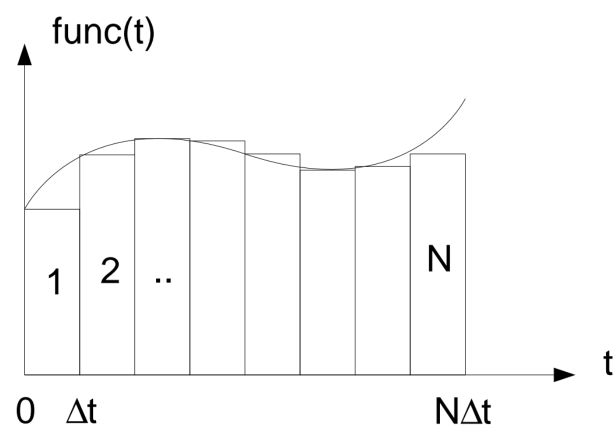


Figure 1: Illustration of the Riemann integral

Exercise 4.4

1. To see the effects of using a different number points to represent a continuous-time signal, write a Python function for numerically computing the integral of the function $\sin^2(7t)$ over the interval $[0, 2\pi]$. The syntax of the function is

```
def integ1(N):
    """
    Parameters
    ---
    N: the number of rectangles

    Returns
    ---
    I: the approximate integral
    """
    return I
```

where `I` is the result to be returned and `N` is the number of rectangles used to approximate the integral. This function should use the `np.sum` function and it should NOT contain any *for* loops!

Note: Since Python is an *interpreted* language, *for* loops are relatively slow. Therefore, we will avoid using loops whenever possible.

In [23]:
```
# your code goes here
```

2. Write a script in a different block that evaluates $I(N)$ for $1 \leqslant N \leqslant 100$, stores the result in a NumPy array and plots the resulting array as a function of N. This script may contain *for* loops.

Note: This script should call the function

```
def integ1(N)
```

you just completed.

In [24]:
```
# your code goes here
```

3. Write a second function `J=integ2(N)` which numerically computes the integral of e^t on the interval [0, 1]. The syntax of the function is

```
def integ2(N):
    """
    Parameters
    ---
    N: the number of rectangles

    Returns
    ---
    J: the approximate integral
    """
    return J
```

In [25]:
```
# your code goes here
```

4. Write a script in the next block that evaluates $J(N)$ for $1 \leqslant N \leqslant 100$, stores the result in a NumPy array and plots the resulting array as a function of N. This script may contain *for* loops.

Note: This script should call the function

```
def integ2(N)
```

you just completed.

In [26]:
```
# your code goes here
```

5. Compare your results to the analytical solutions from Section 4.3. Explain why $I(7) = I(14) = 0$.

your answer goes here

5. Processing of Speech Signals

Digital signal processing is widely used in speech processing for applications ranging from speech compression and transmission, to speech recognition and speaker identification. This exercise will introduce the process of reading and manipulating a speech signal.

Exercise 5

1. Import `soundfile` and `sounddevice` modules, and use `sf` and `sd` as the aliases respectively.

In [27]:
```
# your code goes here
```

2. Do the following:

1. Use the function `speech, fs = sf.read("speech.au")` [(https://pysoundfile.readthedocs.io/en/latest/#read-write-functions)](https://pysoundfile.readthedocs.io/en/latest/#read-write-functions) to load file *speech.au* into Python. Note that this function returns two arguments, where the first one is the data, and the second one is the sampling rate.
2. Play the signal, using the command `sd.play(speech, fs)` followed by `sd.wait()`. The second command blocks the Python interpreter until playback is finished.
3. Plot the signal on the screen as if it were a continuous-time signal (i.e., use the `plt.plot` command).

In [28]:
```
# your code goes here
```

6. Special Functions

Exercise 6.1

Plot the following two continuous-time functions over the specified intervals.

1. $\begin{cases} \frac{\sin(\pi t)}{\pi t} & t \neq 0 \\ 1 & t = 0 \end{cases} \quad \text{for } t \in [-10\pi, 10\pi]$

2. $\text{rect}(t) \quad \text{for } t \in [-2, 2]$

Hint: The function $\texttt{rect}()$ may be computed in Python by using a Boolean expression. For example, if

```
t = np.linspace(-10, 10, 201)
```

then $y = \text{rect}(t)$ may be computed using the command

```
y = (abs(t) <= 0.5).astype(float)
```

In [29]:
```
# your code goes here
```

Exercise 6.2

1. For each of the following functions, write a Python function that takes two arguments, `a` and `n`. Then write the script below to stem those 2 discrete-time functions for $a = 0.8$, $a = 1.0$ and $a = 1.5$, by calling the Python functions you just wrote.

1. $a^n(u[n] - u[n-10]) \quad \text{for } n \in [-20, 20]$

2. $\cos(\omega n)a^n u[n] \quad \text{for } \omega = \pi/4, \text{ and } n \in [-1, 10]$

The syntax of the functions should be the following:

```
def func_6_2_1(a,n):
    pass
```

and

```python
def func_6_2_2(a,n):
    pass
```

Hint: The unit step function $y = u[n]$ may be computed in Python 3 using the command

```python
y = (n >= 0).astype(float)
```

where `n` is a vector of values of time indices.

In [30]:
```python
# your code goes here
```

7. Sampling

The word *sampling* refers to the conversion of a continuous-time signal into a discrete-time signal. The signal is converted by taking its value, or sample, at uniformly spaced points in time. The time between two consecutive samples is called the *sampling period*. For example, a sampling period of 0.1 seconds implies that the value of the signal is stored every 0.1 seconds.

Exercise 7

Consider the signal $f(t) = \sin(2\pi t)$. We may form a discrete-time signal, $x[n]$, by sampling this signal with a period of T_s. In this case,

$$x(n) = f(T_s n) = \sin(2\pi T_s n)$$

1. Firstly,using the previously presented signal, complete the following function

```python
def func_7(Ts, n):
    pass
```

In [31]:
```python
# your code goes here
```

2. Use the `plt.stem` to plot the function $f(T_s n)$ defined above for the following values of T_s and n.

Note: Your code should call the function defined above.

1. $T_s = 1/10, 0 \leqslant n \leqslant 100; \quad \text{axis}([0, 100, -1, 1])$
2. $T_s = 1/3, 0 \leqslant n \leqslant 30; \quad \text{axis}([0, 30, -1, 1])$
3. $T_s = 1/2, 0 \leqslant n \leqslant 20; \quad \text{axis}([0, 20, -1, 1])$
4. $T_s = 10/9, 0 \leqslant n \leqslant 9; \quad \text{axis}([0, 9, -1, 1])$

In [32]:
```python
# your code goes here
```

3. Discuss your results obtained from the 4 plots above. How does the sampled version of the signal with $T_s = 1/10$ compare to those with $T_s = 1/3$, $T_s = 1/2$ and $T_2 = 10/9$?

your answer goes here

8. 2-D Signals

So far we have only considered 1-D signals such as speech signals. However, 2-D signals are also very important in digital signal processing. For example, the elevation at each point on a map, or the color at each point on a photograph are examples of important 2-D signals. As in the 1-D case, we may distinguish between continuous-space and discrete-space signals. However in this section, we will restrict attention to discrete-space 2-D signals.

Exercise 8

When working with 2-D signals, we may choose to visualize them as images or as 2-D surfaces in a 3-D space. To demonstrate the differences between these two approaches, we will use two different display techniques in Python. Do the following:

1. Use the `np.meshgrid` command to generate the discrete-space 2-D signal $f[m, n] = 255|\text{sinc}(0.2m)\text{sinc}(0.2n)|$ for $-20 \leqslant m \leqslant 20$ and $-20 \leqslant n \leqslant 20$.

In [33]:
```
# your code goes here
```

2. Use the following code to display the signal as a surface plot.

In [34]:
```
# refer to https://matplotlib.org/3.1.0/gallery/mplot3d/surface3d.html

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

def mesh_plot(X, Y, Z, title, xlabel, ylabel):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=2, antialiased=True)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    # Add a color bar which maps values to colors.
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()
```

3. Display the signal as an image. Use the function attribute `cmap='gray'` inside the `plt.imshow()` command to obtain a grayscale image.

In [35]:
```
# your code goes here
```

4. For which applications do you think the surface plot works better? When would you prefer the image?

your answer goes here

9. 2D Random Signals - Optional Exercise

The objective of this section is to show how to recover a signal from noisy observations of that signal.

1. Generate one 100×100 image with a 10×10 white square in the middle (pixel value 110) on a black background (pixel value 100). Add a random number to each pixel value of the image. The random number for each pixel should be generated independently following a uniform distribution on the interval $[-100, 100]$. Use the Python command `np.random.rand()` to generate these random numbers.

Hint: While command `np.random.rand()` generates a sample of a uniform random variable on the interval $[0, 1]$, the command `np.random.rand() - 3` will generate a sample from uniform random variable on the interval $[-3, -2]$, and the command `7 * np.random.rand()` will generate a sample from uniform random variable on the interval $[0, 7]$.

In [36]:
```
# your code goes here
```

2. Having added a random number to each pixel of the square image, you have created a "noisy observation" of the square image. Display the resulting 2D signal (the noisy observation) as an image using the command `plt.imshow()` (e.g., `plt.imshow(X,[])` .)

In [37]:
```
# your code goes here
```

3. Can you distinguish the square in the center of the noisy image?

your answer goes here

4. Repeat this procedure to generate 99 additional (different) noisy observations of the image of the square. Obtain a new image by averaging the pixel values of each of these 100 images. Plot the resulting new image.

Hint: Create a 3-d NumPy array by `image_arr = np.zeros((N, H, W))` , where $N = 100$ is the number of images, $H = 100$ is the height of the image and $W = 100$ is the width of the image. You may assign values to this 3-d array by using a *for* loop, then use the function `np.mean(images, axis=0)` to take the average of the pixel values of each of these 100 images.

In [38]:
```
# your code goes here
```

5. Can you distinguish the square in the center of the new image?

your answer goes here

Apendix A - Windows Anaconda Installation

Note that this is a basic tutorial on installing Anaconda and it may be subject to changes.

To install Anaconda on windows you will nedd to follow the following installation instructions:

- Download Anaconda from the Anaconda website [www.anaconda.com (http://www.anaconda.com)](http://www.anaconda.com)
- From the products tab, select the indiviual edition and downlad the windows version.
- Launch the Anaconda installer
- Follow installation steps, if you only want to use Anaconda and no other versions of Python select tge `register anaconda as your default Python` . If you have other versions of Python on your device, select the `add anaconda to my path variable option`
- Follow installation instructions as normal, you shoud not need to install PyCharm.
- Finish installation.
- To run Anaconda, you will need to open and run the `Anaconda prompt` application.
- type the command `cd direcory address` to navigate to your work space, typically where you have the jupyter files saved.
- type `Jupyter notebook` and Jupyter should launch.

- You shoud now be able to edit and run the Jupyter notebooks.

Apendix B - Mac Os Anaconda installation

Note that this is a basic tutorial on installing anaconda and it may be subject to changes.

- Download Anaconda from the Anaconda website [www.anaconda.com (http://www.anaconda.com)](http://www.anaconda.com)
- From the products tab, select the indiviual edition and downlad the Mac Os version.
- Launch the Anaconda installer

- Follow the installation instructions and choose to install Anaconda on the `~/opt` directory. Other directories might not be recomended.
- Continue the installation instructions, you do not need to install PyCharm for this course.
- Click on spotlight (or use the short cut `Cmd+Spacebar`) to open spotlight, type `terminal`
- type the command `cd/direcory address` to navigate to your work space, typically where you have the Jupyter files saved.
- type `Jupyter notebook` and Jupyter should launch.
- You shoud now be able to edit and run the Jupyter notebooks.

Apendix C - Linux Anaconda installation

Note that this is a basic tutorial on installing anaconda and it may be subject to changes.

Note Purdue uses `RedHat` as its linux operating sytem, so this instructions are geared towards such operating system.*

- You will first need to install some extended dependnecies `yum install libXcomposite libXcursor libXi libXtst libXrandr alsa-lib mesa-libEGL libXdamage mesa-libGL libXScrnSaver`
- Download Anaconda from the Anaconda website [www.anaconda.com (http://www.anaconda.com)](http://www.anaconda.com)
- From the products tab, select the indiviual edition and downlad the Linux version.
- Enter the following commands to install Anaconda on your system terminal. `bash ~/Downloads/Anaconda3-2020.02-Linux-x86_64.sh`
- Follow the installation instructions, choose the default installer location.
- Follow the installation instructions, unitl it finished installing. You do not need to install `PyCharm` for this course.
- Open system terminal
- type the command `cd/direcory address` to navigate to your work space, typically where you have the Jupyter files saved.
- type `Jupyter notebook` and Jupyter should launch.
- You shoud now be able to edit and run the Jupyter notebooks.

Python libraries you'll need to install to perfom all the labs

Note: If you are only planning on using the laboratory computers, you will not need to install these libraries.

Use the following command in a new Terminal window to install a Python library: `pip install --user <package>`

- numpy
- matplotlib
- soundfile
- sounddevice
- scipy
- json
- math

Remember to relaunch the Jupyter notebook.

Tutorials References

"Installing on Windows ." anaconda Documentation, anaconda , docs.anaconda.com/anaconda/install/windows/."

"Installing on macOS." anaconda Documentation, anaconda , docs.anaconda.com/anaconda/install/mac-os/."

"Installing on Linux." anaconda Documentation, anaconda , docs.anaconda.com/anaconda/install/linux/."

"Verifying your installation." anaconda Documentation, anaconda , docs.anaconda.com/anaconda/install/verify-install/."