UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
TRATAMIENTO DE SEÑALES I
LABORATORIO 0
INTRODUCCIÓN A PYTHON

Justificación

Python es una potente herramienta de cálculo numérico muy usada en ingeniería, especialmente en telecomunicaciones, para simular el procesamiento de las señales, aprovechando su facilidad y versatilidad, convirtiéndola en una herramienta indispensable en el curso.

Objetivos

- Conocer las principales características de programación de Python, los comandos y funciones más comunes.
- Conocer la creación y uso de las funciones en el script de Python, al igual que su utilidad para verificar el funcionamiento de los algoritmos desarrollados.
- Mostrar diferentes herramientas de programación de Python para resolver problemas planteados, que serán útiles para desarrollar algunas guías de laboratorio de Tratamiento de Señales 1.

Teoría

Python es un lenguaje de programación fácil de aprender y poderoso. Tiene eficientes estructuras de datos de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. La elegante sintaxis y la dinámica de la escritura de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para la elaboración de scripts y el rápido desarrollo de aplicaciones en muchas áreas en la mayoría de las plataformas. El intérprete de Python se amplía fácilmente con nuevas funciones y tipos de datos implementados en C o C++ (u otros lenguajes que se pueden llamar desde C). Python también es adecuado como lenguaje de extensión para aplicaciones personalizadas. Las principales características de Python son:

- Manejo eficiente de funciones matemáticas.
- Construcción de figuras de diferentes formatos (por ejemplo gráficas 3D con contorno), con una herramienta de edición completa.
- Uso de diferentes Toolbox o caja de herramienta para aplicaciones específicas o especializadas (e.g. diseño de filtros, comunicaciones, procesamiento de señales, redes neuronales, lógica difusa, etc.)
- Integración amigable con otros lenguajes de programación (por ejemplo C, C++ y Excel).
- Los datos pueden ser almacenados en diferentes formatos (por ejemplo numeric, string, cell, double, int o entero, entre otros).
- Los datos se pueden manejar de forma vectorial, matricial, en listas y en diccionarios.

Instalación de Anaconda

1. Descargar Anaconda para Python 3.7 del siguiente enlace:
https://www.anaconda.com/products/individual

2. Si su sistema operativo es Linux:
a. Abra la carpeta donde descargó el instalador y dé click derecho en un espacio en blanco, seleccione Abrir en Terminal (o abra una terminal y con el comando cd vaya a la carpeta).

b. Instale Anaconda con el siguiente comando: bash Anaconda….
La licencia se lee oprimiendo *Espacio* hasta que se pida escribir "*yes*". Procure instalarlo en una ubicación que no tenga tildes o caracteres extraños en su ruta.

c. Al final de la instalación el programa pregunta algo más, indique sí con una "*y*" y presione *Enter*.

d. Para abrir el editor que se usará en el Laboratorio, cierre la Terminal y abra una nueva, ejecute el siguiente comando: jupyter-notebook

e. En la pestaña que se abre, dar click en *new*, y después en la opción *Python3*, debe observar algo similar a la imagen de la Figura 1:

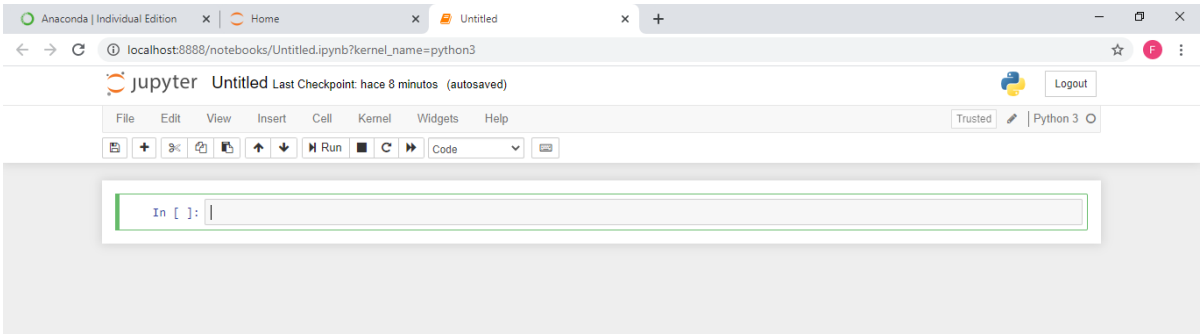

Figura 1. Entorno de Jupyter Notebook, el cual se usará en el Laboratorio.

3. Si su sistema operativo es Windows:
a. Abra la carpeta donde descargó el instalador y dé click en ejecutar. Dar en *next*, después de esto aceptar el acuerdo de licencia, elegir si instalará para el usuario actual o para todos los usuarios y finalmente elegir la ubicación donde quedará guardado Anaconda, procure elegir una ubicación que no tenga tildes o caracteres extraños en su ruta.

b. Para abrir el editor que se usará en el Laboratorio, desplazarse al buscador en Windows (parte inferior izquierda), escribir Jupyter Notebook y abrir esta aplicación.

c. En la pestaña que se abre, dar click en *new*, y después en la opción *Python3*, debe observar algo similar a la imagen de la Figura 1.

4. Si desea trabajar en Google Colab, el cual es una herramienta que te permite ejecutar y programar en Python en tu navegador, no requiere configuración, cuenta con acceso gratuito a GPUs y permite compartir fácilmente los archivos que crees con otras personas:

a. Ingresar a https://colab.research.google.com/notebooks/intro.ipynb o escribir Google Colab en el navegador y entrar a la primera opción.

b. Iniciar sesión con una cuenta de Google, preferiblemente con el correo de la Universidad.

c. En la parte superior izquierda verá la opción *Archivo*, dar click ahí y después en la opción *Nuevo cuaderno*. Veras algo similar a la Imagen de la Figura 2:

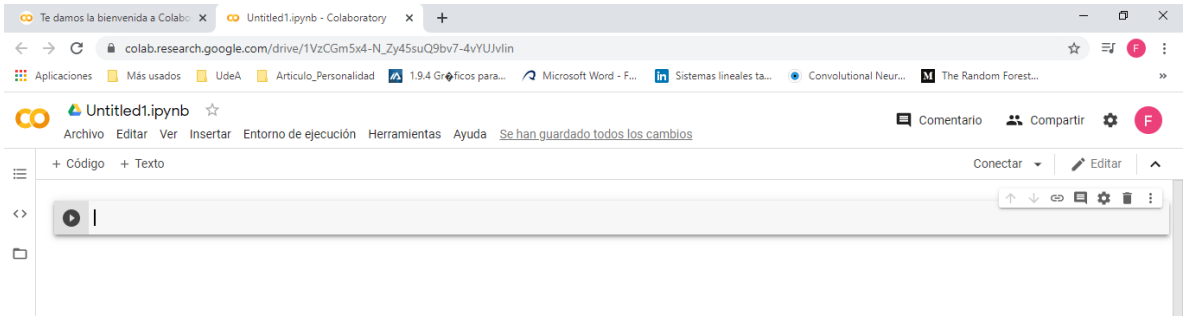

Figura 2. Entorno de Jupyter Notebook en Google Colab.

d. Todos los archivos que crees en Google Colab, quedan guardados en tu Drive en una carpeta llamada *Colab Notebooks*, donde podrás descargarlos posteriormente. **Muy importante contar con conexión a Internet para poder trabajar en Google Colab y que los cambios sean guardados.**

Manejo del Notebook

Los notebooks de Jupyter serán los informes que se entregarán de cada práctica de laboratorio. Tener en cuenta las siguientes consideraciones: Existen diferentes tipos de celdas. Las más comunes son:
Code: En estas se escribe el código que se va a ejecutar
Markdown: En estas se puede escribir texto, conclusiones, explicaciones, al igual que poner imágenes y links. En la Figura 3 se observa un ejemplo de una celda tipo Markdown.

```
# Titulo I
## Titulo II
### Titulo III

Texto normal

Se pueden agregar ecuaciones como si fuera codigo de LaTex

$$E=\frac{1}{N}\sum_{i=1}^N x[i]^2$$

Se pueden incrustar imagenes en el notebook como si fuera codigo HTML

<img src="http://cdn.memegenerator.es/imagenes/memes/full/4/10/4107176.jpg" width="500" ...
    align="middle">

Se pueden agregar enlaces a paginas web

[Introduccion a Jupyter](http://www.emilkhatib.es/introduccion-a-jupyter-ipython-notebook/)
```

Figura 3. Ejemplo de una celda tipo Markdown en Jupyter Notebook.

Comandos básicos de Python

1. Variables

Python es un lenguaje tipado de manera dinámica, es decir, que la misma variable puede tomar diferentes tipos (entero, flotante, caracteres, etc.) en el mismo programa. A continuación se muestra un ejemplo del manejo de algunas variables.

```
# Una variable puede contener una cadena de caracteres
a='Spam'
print(a)

# Se pueden concatenar cadenas con el operador '+'
b=' with eggs'
print(a+b)

#%%
# Ahora la misma variable de antes puede contener un entero
a=42
print(a)

# Naturalmente, se puede operar entre enteros
b=8
print(a+b)
```

Para mostrar información se pueden imprimir directamente las variables o se puede utilizar la función *format*.

```
c='The answer to the Ultimate Question of Life, the Universe, and Everything is:'
print(c)
print(a)

# Se pueden concatenar resultados utilizando coma (,)

print(c,a)'!'

# De una manera mas 'elegante' con la instruccion "format"
c='The answer to the Ultimate Question of Life, the Universe, and Everything is: {}!'
print(c.format(a))
d=2
e='Este es'
print('{} otro ejemplo de format con {} variables'.format(e,d))
# En Python 3, la division entre enteros produce un numero flotante
a=2
b=3
print(b/a)

# Para lograr un entero se debe utilizar el operador //
print(b//a)

# El operador "**" permite hacer potenciacion
print(a**b)
```

Los arreglos sirven para guardar datos del mismo tipo, es decir, todos sus elementos deben ser enteros, cadenas de caracteres, etc.

```
arreglo_num=[3, 1, 7, 6]
arreglo_str=['primer', 'segundo', 'tercer', 'cuarto']
print(arreglo_num)
print("El tercer elemento es: {}".format(arreglo_num[2]))
print(arreglo_str)

for i,s in enumerate(arreglo_str): # Recorrer los arreglos, mas de esto despues
    print("El {} elemento es: {}".format(s,arreglo_num[i]))
```

Las listas permiten guardar diferentes tipos de datos bajo el mismo nombre.

```
lista=[3, 'patata', 7, 2.5, 'spam']
for l in lista: # Recorrer la lista
    if isinstance(l, str): # Revisar si es una cadena de caracteres
        print(l)
    else:
        print(l/2)
```

Un diccionario permite almacenar datos como un conjunto de clave y valor sin orden.

```
diccionario={'rojo':'red', 'azul':'blue', 'cien':100, 'dos':2}
print(diccionario['rojo'])
print(diccionario['cien'])
print(diccionario['dos'])
```

2. Instrucciones de control

Las principales instrucciones en Python son:

- ***if:*** Ejecutar ciertas líneas de código si se cumple una condición.
- ***for:*** Ciclos con un número específico de iteraciones.
- ***while:*** Ciclo para repetir código hasta cumplir una condición (número NO específico de iteraciones).

Python no ofrece otras instrucciones como switch o el ciclo do-while.

```
a=2
b=3
if a>b:
    print('a es mayor que b')
elif a==b:
    print('a es igual a b')
else:
    print('a es menor que b')

# Se pueden hacer comparaciones entre diferentes tipos sin error
a=42
if a == 'The answer to the Ultimate Question of Life, the Universe, and Everything':
    print('Spam!')
else:
    print("It does not work :(")

# Se puede buscar una cadena dentro de otra
a='studying' # Intente con: 'assassin'
b='dying' # Y 'sin'
if a.find(b):
    print("You can't spell {} without {}".format(a,b))
```

Python distingue entre ciclos for y ciclos while, definiéndolos según el “libro”. Es decir, los ciclos for son para un número conocido de iteraciones, mientras que los ciclos while se utilizan cuando no se conoce de antemano el número de iteraciones, sino que se itera hasta cumplir cierta condición. Los ciclos for en Python siempre se hacen sobre secuencias almacenadas en arreglos o listas, como se muestra en los siguientes ejemplos:

```
#%%
# Los ciclos for son para secuencias almacenadas en arreglos o listas
str_seq=['Esto', 'es', 'una', 'secuencia', 'de', 'cadenas', 'de', 'caracteres']
for s in str_seq:
    print(s)

# Pueden ser secuencias de numeros
num_seq=[3, 8, 2, 5, 32, 90]
for num in num_seq:
    print("El numero es: {}".format(num))

# Si necesito el indice utilizo enumerate
for i,num in enumerate(num_seq):
    print("El numero en la posicion {} es: {}".format(i,num))

# Pueden ser tuplas
str_seq=[[2, 'Esto'], [5, 'es'], [7, 'una'], [1, 'secuencia'], [3, 'de'], [9, 'tuplas']]
for s1,s2 in str_seq:
    print("{} : {}".format(s1,s2))
```

Los ciclos while se repiten hasta que se cumple una condición.

```
a=32
while a!=42:
    a+=2
    print('Keep searching!')
c='The answer to the Ultimate Question of Life, the Universe, and Everything is: {}!'
print(c.format(a))
```

3. Funciones

Como en todos los lenguajes de programación, en Python, el programador puede definir sus propias funciones. Estas se declaran con la palabra *def + el nombre de la función*, y se definen de la siguiente forma:

```
def funcion(arg1, arg2):
    print("El primer argumento es: {}".format(arg1))
    print("El segundo argumento es: {}".format(arg2))
    return arg1*2, arg2*3

a=1
b=2

c,d=funcion(a,b)

print("El primer valor retornado es: {}".format(c))
print("El segundo valor retornado es: {}".format(d))
```

4. Librerías propias

Si tenemos alguna función que queremos utilizar en varios programas podemos aprovechar el uso de librerías para evitar tener que copiar y pegar en varios códigos. Definamos dentro de un archivo llamado MiLibreria.py las siguientes dos funciones:

```
def funcion1(arg1):
    print("Esta funcion multiplica por 2")
    return 2*arg1

def funcion2(arg1):
    print("Esta funcion multiplica por 3")
    return 3*arg1
```

Ahora, en un archivo .py o en un notebook (archivo .ipynb) que se encuentre en la misma carpeta donde creamos el archivo MiLibreria.py, podemos utilizar estas funciones de la siguiente manera:

```
import MiLibreria

a=MiLibreria.funcion1(1)
print(a)
b=MiLibreria.funcion2(2)
print(b)
```

5. Arreglos y matrices

Algunos de las formas de definir arreglos y matrices son las siguientes:

```
# manejo de arreglos

import numpy as np    # Libreria estandar para manejo de datos y senales en Python

a = np.zeros((2,2))   # Crea un arreglo tipo matriz de ceros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Crea un arreglo tipo vector fila de unos
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Crea un arreglo tipo matriz de un valor constante
print(c)              # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"
c = 7*np.ones((2,2))  # Otra forma
print(c)

d = np.eye(2)         # Crea un arreglo con la matriz identidad
print(d)             # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2)) # Crea un arreglo tipo matriz de numeros aleatorios entre 0 y 1
print(e)             # print "[[ 0.91940167  0.08143941]
                      #        [ 0.68744134  0.87236687]]"
print(e[0,1])        # Seleccionar un elemento
print(e[0,:])        # Seleccionar la primera fila
print(e[:,1])        # Seleccionar la segunda columna

xrang=np.arange(20)   # Crea un arreglo de numeros consecutivos entre 0 y 19
print(xrang)         # print "[0 1 2 3 4 5... 19]"

# Para seleccionar partes del arreglo use los siguientes comandos

print(xrang[0:5])     # print "[0 1 2 3 4]"
print(xrang[10])      # print [10]

A=np.random.random(100)
media=np.mean(A)       # calcula la media de un arreglo A
maxA=np.max(A)         # calcula el maximo valor de un arreglo
minA=np.min(A)         # calcula el minimo valor de un arreglo
print(media, maxA, minA)
```

6. Manejo de señales

a. El siguiente ejemplo genera una señal sinusoidal con una frecuencia de $1\ Hz$, una duración de 2 segundos y una frecuencia de muestreo de $f_s = 5\ Hz$.

```
import numpy as np
import matplotlib.pyplot as plt # libreria usada para graficas

# para que la grafica se incruste en el notebook
%matplotlib inline

f=1.0  # Frecuencia de la senal
fs=5.0 # Frecuencia de muestreo
t=np.arange(0, 2.0, 1.0/fs) # Vector de tiempo
x = np.sin(2*np.pi*f*t)
plt.plot(t,x)
plt.xlabel('Time',fontsize=18)
plt.ylabel('Amplitude',fontsize=18)
plt.show()
```

Ejercicio: ¿Cómo se ve la figura? Cambie la frecuencia de muestreo de la señal por $20\ Hz$, y ejecute nuevamente el script, ¿Qué cambios se observan?

b. También se pueden cargar señales de distintas fuentes de información, por ejemplo archivos de Excel. Ver el siguiente ejemplo que analiza la tasa de desempleo en Colombia desde 2002 hasta 2017.

```
import pandas as pd # libreria para el manejo de datos

# Leer datos
TD = pd.read_excel("TD.xlsx")

# Cambio de formato de los datos
TD = pd.DataFrame(TD)

# Extraigo una de las columnas del archivo de excel
TDC = TD["TD"]
index = TD["Index"]

#grafica de la tasa de desempleo
plt.plot(index, TDC)
plt.title("Tasa de desempleo en Colombia")
```

Puede ver información adicional aquí:
https://github.com/neuraldevs/ML-ND-CD/blob/master/Regression/Regression.ipynb

c. Se pueden cargar también señales de audio, tal como en el siguiente ejemplo. Primero Descargue la señal de audio de su preferencia del siguiente enlace, cópiela a la misma carpeta del Notebook y renómbrela como *senal.wav*.

https://ccrma.stanford.edu/~jos/pasp/Sound_Examples.html (en formato *.wav)

```
from scipy.io.wavfile import read # libreria para lectura de archivos de audio
from IPython.display import Audio # para escuchar la senal

file_audio=('senal.wav') # Ruta del archivo con la senal
fs, x=read(file_audio) # Cargar el archivo
x=x/float(max(abs(x))) # escala la amplitud de la senal
t=np.arange(0, float(len(x))/fs, 1.0/fs) # Vector de tiempo
plt.plot(t,x) # Dibujar la grafica

# Los siguientes dos comandos dibujan las etiquetas de los ejes
plt.xlabel('Time',fontsize=18) # Etiqueta eje X
plt.ylabel('Amplitude',fontsize=18) # Etiqueta eje Y
plt.show() # Mostrar la grafica
Audio(x, rate=fs) # para escuchar la senal, si se desea
```

Procedimiento

1. Crear una función con tres argumentos de salida y dos argumentos de entrada, donde las entradas están dadas por:

$$A = [1,1.1,1.2,\ldots,2] \text{ , } B = [sin(1),\ sin(1.2), sin(1.4),\ldots, sin(3)]$$

   Las salidas están dadas por:

$$K = A / B, L = A^2 + B^2, \text{ M } = (A \bullet B)^2 \text{ , } donde \bullet denota\ producto\ punto$$

   Ayuda: Consulte como usar la función *arange* y *linspace* de la librería *numpy* para crear vectores en Python.

2. Cree una función que reciba como argumento una matriz 3x3 generada aleatoriamente. La salida de dicha función deberá retornar lo siguiente:

   a. La transpuesta de la matriz multiplicada por su inversa.
   b. La raíz cuadrada del determinante de la matriz.
   c. La sumatoria de cada columna.

3. Generar una figura de superficie y una figura de contorno en 3D, de la siguiente ecuación:

$$z = e^{x}cos(y) + e^{y}sin(x)$$

Donde x, y son vectores que van desde -4 hasta 4 con un tamaño de paso de 0.1.

Ayuda: Consulte como usar la función *Axes3D* de la librería *mpl_toolkits.mplot3d*

4. Cree una función que reciba el periodo de una señal y retorne un tren de pulsos con un ciclo de dureza del 50%. Grafique la función obtenida entre 0 y 10s. ¿El resultado es como esperaba?

5. Repita lo anterior para una señal diente de sierra de pendiente $m = 1$. Tenga en cuenta que la función diente de cierra es creciente durante todo el periodo (en este caso no aplica el ciclo de dureza)

6. Consulte cómo usar las funciones *square* y *sawtooth* de la librería *scipy*. Compare los resultados de los numerales 4 y 5 con el resultado de usar dichas funciones.

7. Cree un vector aleatorio de “1s” y “0s” con 1000 elementos. Agregue un ruido con una desviación estándar de 0.1. Grafique el histograma del resultado.

    - ¿Qué distribución de probabilidad siguen los datos?
    - Si el resultado obtenido representa la recepción información binaria ¿Cuál sería el umbral de decisión que garantice una menor tasa de error de bit?

NOTA: Las figuras deben contener los nombres de los ejes, título y leyenda si es necesario. Todos los números y texto deben tener un tamaño de 20 pt. Use el comando *subplot* de la librería *matplotlib.pyplot* para mostrar ambas graficas en una misma figura.

Entregables

- Informe en el Jupyter Notebook (archivo .ipynb) que contiene el código (solución a lo pedido), respuesta a las preguntas y las conclusiones del laboratorio.

Bibliografía

➢ https://github.com/kuleshov/cs228-material/blob/master/tutorials/python/cs228-python-tutorial.ipynb