"Responsabilidad con pensamiento positivo"

UNIVERSIDAD TECNOLÓGICA ISRAEL

TRABAJO DE TITULACIÓN EN OPCIÓN AL GRADO DE: INGENIERO EN ELECTRÓNICA DIGITAL Y TELECOMUNICACIONES

TEMA:

APLICACIONES DEL SOFTWARE LIBRE *PYTHON* PARA PRÁCTICAS DE LABORATORIO APLICADO A LA ASIGNATURA DE TRATAMIENTO DIGITAL DE SEÑALES DE LA UNIVERSIDAD TECNOLOGICA ISRAEL.

AUTOR:

JAIRO WLADIMIR CONGO PASTRANA

TUTOR:

ING. FLAVIO MORALES ARÉVALO,
MG.

QUITO, ECUADOR

2018

UNIVERSIDAD TECNOLÓGICA

ISRAEL APROBACIÓN DEL

TUTOR

En mi calidad de tutor del trabajo de titulación certifico:

Que el trabajo de titulación **"APLICACIONES DEL *SOFTWARE* LIBRE PYTHONPARA PRACTICAS DE LABORATORIO APLICADO A LA ASIGNATURA DE TRATAMIENTO DIGITAL DE SEÑALES DE LA UNIVERSIDAD TECNOLOGICA ISRAEL"**, presentado por el Sr. Jairo Wladimir Congo Pastrana, estudiante de la carrera de Electrónica Digital y Telecomunicaciones, reúne los requisitos y méritos suficientes para ser sometido a la evaluación del Tribunal de Grado, que se designe, para su correspondiente estudio y calificación.

Quito D.M. Agosto del 2018

TUTOR

……..

Ing. Flavio Morales Arévalo, Mg

AUTORÍA DEL PROYECTO DE TITULACIÓN

El abajo firmante en calidad de estudiante de la Carrera de Electrónica Digital y Telecomunicaciones, declaro que los contenidos de este Trabajo de Titulación, requisito previo a la obtención del Grado en Ingeniería en Electrónica y Telecomunicaciones, son absolutamente originales, auténticos y de exclusiva responsabilidad legal y académica del autor.

Quito, D.M. Agosto del 2018

_____

Jairo Wladimir Congo Pastrana

C.C. 1721342978

AGRADECIMIENTOS

Más que a nadie quiero dar gracias a Dios por haberme permitido llegar a esta instancia en mi vida y poder cumplir una meta tan anhelada, agradezco profundamente a mi familia quienes han sido motivo de inspiración constante convirtiéndose en un pilar fundamental en mis actividades, de igual forma a la Universidad Tecnológica Israel institución que me abrió sus puertas y me permitió nutrirme de la sabiduría de cada tutor que tuve, hoy siento con gran orgullo que sus conocimientos y experiencias transmitidas ayudaron a mi formación y culminación de una etapa en mi vida.

Quiero agradecer a mi pareja por ser parte de este proyecto y entenderme en todo, siempre seré feliz de saber que ella en todo momento de mi vida fue un apoyo incondicional, no hay palabras que describan la persona que es, fue mi lucha reflejada, es la persona a la cual yo amo demasiado, y por esos momentos que soñamos abrazar y al final los alcanzamos agradezco cada minuto a su lado.

DEDICATORIA

Este trabajo está dedicado a mis padres Iralda y Luis, a mi pareja Victoria, a mis amigos de infancia quienes con su amor, buen ejemplo, consejos, compañía y aventuras han sabido motivarme en la lucha diaria que guía al hombre a cumplir los objetivos propuestos a pesar de las circunstancias adversas, gracias por la confianza depositada en mí.

Jairo.

TABLA DE COTENIDO

LISTA DE FIGURAS

LISTA DE ECUACIONES

RESUMEN

El proyecto de titulación presentado a continuación, se enfoca en desarrollar una sugerencia de prácticas de laboratorio de Tratamiento Digital de Señales, materia impartida dentro de la carrera de Ingeniería en Electrónica Digital y Telecomunicaciones de la Universidad Tecnológica Israel. Esta sugerencia hace énfasis en el empleo del *Software* libre **Python** y servirá como introducción a la visualización de características y procesos a través de los cuales, las señales de información son tratadas previo a ser enviadas para llevar información a cualquier medio.

Con el fin de presentar una alternativa actualizada de las herramientas necesarias para un profesional en Telecomunicaciones, se ha dividido este proyecto en 5 etapas: La primera de ellas, describe con especial enfoque, los antecedentes, justificación y objetivos de promover **Python** como *software* para crear un modelo de laboratorio de Tratamiento Digital de Señales. La segunda parte contiene los aspectos conceptuales necesarios para interpretar las señales y los procesos matemáticos que deben emplearse en los circuitos que funcionan usando algoritmos para tratar señales. La tercera parte, contiene los aspectos teóricos y técnicos del *Software* sugerido para emplearse en laboratorio, mostrando sus características, líneas de ayuda y la descripción de la interfaz gráfica para el usuario. La cuarta división de este proyecto, contiene un formato de prácticas que podrían usarse para el laboratorio de la materia mencionada. El quinto capítulo contiene las conclusiones y recomendaciones que han surgido conforme se ha ido desarrollando el proyecto. Finalmente se presentan dos anexos, los cuales contienen el código de programación válidos para los script de las prácticas desarrolladas.

ABSTRACT

The degree project submitted then focused on developing a suggestion of laboratory practice of Digital signal treatment, matter taught within the career of Digital Electronics and telecommunications engineering of the University of technology Israel. This suggestion places emphasis on the use of free *Software Python* and will serve as an introduction to the display characteristics and processes through which information signals are treated prior to being sent to bring information to any means. In order to present an alternative updated the necessary tools for a telecommunication professional, I have divided this project in 5 stages: the first one, described with special focus, the background, rationale and objectives of promoting *Python* as *software* to create a model of laboratory of Digital treatment of signals. The second part contains the conceptual aspects necessary to interpret the signals and mathematical processes that should be used in circuits that work using algorithms to treat signals. The third part contains the theoretical and technical aspects of the *Software* suggested for use in laboratory, showing its features, help lines and the description of the graphical user interface. The fourth division of this project, contains a format of practices that could be used for the laboratory of the above-mentioned matter. The fifth chapter contains the conclusions and recommendations that have emerged as the project has evolved. Finally presented two annexes, which contain the valid settings for the Script code.

INTRODUCCIÓN

1. Antecedentes de la situación objeto de estudio

El manejo de aplicaciones informáticas, actualmente se ha generalizado para cualquier tipo de profesionales, esto ha hecho que no sea estrictamente necesario el dominio total de las nuevas tecnologías informáticas. La programación y simulación se ha convertido en una tarea necesaria en la sociedad y en el ámbito científico-técnico, y por supuesto dentro del ámbito académico. Debido a que la gran mayoría de estudiantes que llegan a primeros cursos no tienen ningún tipo de experiencia en programación, incluso gran porcentaje de los estudiantes en cursos superiores, no han visto la necesidad de ir más allá de los ejercicios propuestos dentro de las asignaturas que requieren el cálculo numérico ya que no encuentran herramientas de soporte gratuitos que permitan que su interés científico pueda desarrollarse con experimentos y prácticas.

2. Problema de Investigación: planteamiento y justificación

Se hace necesario que durante la formación de profesionales en telecomunicaciones algunas aplicación de la teoría impartida ligadas a la falta de laboratorios, varias universidades plantean y popularizan el uso de *software* para cómputo científico.

Existen diversas herramientas para el ámbito de computación científica una de ellas basada en *software* libre desarrollado con el popular lenguaje de programación **Python**. Para desenvolverse dentro del ámbito de computación científica, **Python** reúne un conjunto de librerías que permiten realizar diferentes tareas de tratamiento de datos, visualización, cálculo numérico y simbólico entre otras aplicaciones específicas.

Es también necesario que se incluyan herramientas que permitan desempeñar análisis prácticos de señales típicas y algoritmos matemáticos presentes en sistemas de telecomunicaciones. Simplificando el material de laboratorio mediante el uso de *software*, lograremos optimizar los recursos, además de complementar los conocimientos teóricos del estudiante en un enfoque práctico sobre la teoría de formación recibida.

La propuesta es usar las prestaciones que **Python** presenta por medio de sus librerías para computación científica como lo son: Matplotlib, Numpy, Scipy, para con ello diseñar modelos de sistemas que utilizan el procesamiento de señales para su funcionamiento y

poner en práctica de manera simultánea la teoría de la asignatura Tratamiento Digital de Señales de la Universidad Tecnológica Israel.

1. Objetivos del Trabajo de Titulación

Objetivo general:

Desarrollar por medio del *software* **Python** y sus diferentes librerías científicas, la realización de 3 Prácticas de Laboratorio para la asignatura Procesamiento Digital de Señales de la Universidad Tecnológica Israel.

Objetivos específicos:

1. Definir los tipos de herramientas y funciones que **Python** a través de sus librerías para computación científica otorga para el tratamiento digital de señales.
2. Desarrollar por medio de **Python** y sus librerías, los algoritmos que permitan identificar la clasificación de los sistemas discretos mediante sus propiedades, es decir, identificar si el sistema es: estático, dinámico, variante, invariante, lineal, no lineal, causal, no causal, estable o inestable.
3. Generar por medio de **Python** y sus librerías un algoritmo que permita obtener y graficar la convolución entre una señal impulso unitario *u*[n] y una secuencia de entrada conocida x[n], de manera que se pueda obtener la respuesta del sistema y[n] numérica y gráficamente.
4. Crear por medio de **Python** y sus librerías un algoritmo que permita obtener la respuesta en frecuencia de un filtro pasabajo FIR y un filtro pasabajo IIR, partiendo de datos conocidos como el orden del filtro y una frecuencia de corte conocida.
5. Proponer un manual de prácticas para que usando PHYTON y sus librerías de cálculo científico se pueda reproducir de manera práctica la teoría revisada en la asignatura Tratamiento digital de Señales de la Universidad Tecnológica Israel.

2. Descripción de los capítulos

- El Capítulo I presenta dentro la fundamentación teórica de la materia Tratamiento Digital de Señales de la Universidad Tecnológica Israel, que servirá como base para el desarrollo del manual de prácticas propuesto.

Dentro de este apartado se detallan los algoritmos de señales y sistemas a los cuales se hace referencia en el ámbito de Tratamiento de Señales, se describen los

conceptos básicos de los mismos, además se presentan los *software*s que en el ámbito científico son utilizados con propósito específico de creación y análisis de algoritmos útiles en Tratamiento de Señales.

- Describe las ventajas del uso de **Python**, como *software* para Tratamiento de Señales, además se muestran las librerías que transforman a este lenguaje en el entorno de programación esperado para tareas científicas, se describen los comandos y funciones básicas de cada una de las librerías científicas así como las fuentes de información donde son alojadas cada función y sintaxis de comando.

Además de lo antes descrito, el Capítulo 1 contiene la forma de instalación del *software* **Python** bajo el sistema operativo Windows utilizando el IDE **PyCharm**, el cual aloja ciertas ventajas amigables que facilitan tanto la instalación del lenguaje de programación como los componentes que se requieran.

- En el capítulo 2, contiene aspectos teóricos de los procedimientos que se emplearon para la consecución de los objetivos prácticos, en otras palabras se describen las herramientas, librerías y funciones que se utilizaron apegados a cierto diagrama de flujo simplificado para conseguir el desarrollo de la aplicación.
- El tercer capítulo, contiene una guía de prácticas sugeridas utilizando lenguaje *Python*, las cuales se apegan a el temario de la asignatura Tratamiento Digital de Señales, impartida en la Universidad Tecnológica Israel, cada práctica cuenta con un formato que incluye:
    - Objetivos**.**
    - Trabajo Preparatorio
    - Materiales a utilizar
    - Resultados obtenidos
    - Conclusiones y Recomendaciones

CAPITULO I

FUNDAMENTACIÓN TEÓRICA

1 Introducción

El área del tratamiento digital de señales es un segmento de ciencia e ingeniería que ha venido desarrollándose rápidamente dentro de los últimos cuarenta años. El rápido desarrollo ha convenido en eventos importantes tanto en la tecnología digital asociada al campo de la informática como en la fabricación de los circuitos integrados. Las computadoras digitales y el hardware digital asociado de hace cuatro décadas tenían tamaños consideradamente grandes, aparte de ello sus componentes tendían a ser muy caros y, en consecuencia, su uso estaba limitado a las aplicaciones de propósito general en tiempo no real (fuera de línea) científicos y comerciales. El rápido desarrollo de la tecnología de circuitos integrados, empezando con la integración a media escala (MSI, medium-scale integration), continuando con la integración a gran escala (LSI, large-scale integration), y actualmente con la integración a muy gran escala (VLSI, very-large-scale integration) de los circuitos electrónicos ha estimulado el desarrollo de computadoras digitales y hardware digital de propósito especial más potente, de menor tamaño, más rápido y menos costoso.

Reducir el tamaño y costo de los circuitos digitales además potenciando que son relativamente rápidos, ha hecho posible la construcción de sofisticados sistemas digitales altamente capaces de llevar a cabo funciones y tareas relacionadas con el tratamiento de señales digitales. Normalmente es bastante complejo y caro el escenario de implementar mediante circuitería analógica o sistemas de tratamiento de señales analógicas las soluciones pertinentes relacionadas. Tal es así que muchas de las tareas de tratamiento de señales que convencionalmente se realizaban por medios analógicos, en la actualidad se ejecutan empleando hardware digital que es más barato y a menudo más confiable.

No se pretende dar a entender que el tratamiento digital de señales sea la solución apropiada en todos los problemas de tratamiento de señales. Así, en el caso de muchas señales con anchos de banda muy grandes, el tratamiento en tiempo real es un requisito. Para

dichas señales, el procesamiento analógico sea quizá el más óptimo o incluso la única solución posible. Sin embargo, siempre que se disponga de circuitos digitales y se tenga la velocidad suficiente como para utilizar el tratamiento digital, será preferible emplear dichos circuitos. (Manolakis y Proakis, 2007, pág. 1)

La idea base es simple, en lugar de diseñar circuitos complejos para procesar una señal, la señal es convertida en una secuencia de números para que de esta forma sea procesada vía *software*.

El tratamiento de señales digitales (audio, voz e imágenes) es un extenso campo para el estudio teórico y experimental de una gran variedad de algoritmos, desde los más básicos e introductorios como lo son las propiedades de los sistemas LTI hasta otros más complejos, como por ejemplo los que aparecen en los problemas de compresión y descompresión de datos e imágenes o en la síntesis y análisis de patrones (generalmente con componentes que dependen de parámetros estocásticos), como los pulsos eléctricos del corazón.

Una señal representa a una cantidad que varía en el tiempo. Esta definición es muy abstracta, por tal motivo se pretende dar un ejemplo más concreto a través del sonido. El sonido es una variación de la presión del aire. Un micrófono es un dispositivo que mide las variaciones en el aire generando una señal eléctrica que representa el sonido. Por otro lado, un altavoz es un dispositivo que toma una señal eléctrica y produce un sonido.

Micrófonos y altavoces son llamados transductores, debido a que traducen, o convierten, señales que tienen una forma para llevarlas a otra representación de ellas.

El tratamiento digital de señales se puede aplicar a señales de audio y de manera similar a vibraciones mecánicas, electrónicas y señales dentro de otros dominios.

Las señales a la que se hace referencia se pueden clasificar dentro de 2 tipos, existen las señales periódicas y señales no periódicas.

1.1 Señales Periódicas

Las señales periódicas, son señales que se repiten a ellas mismas después de cierto periodo de tiempo. Por ejemplo, si alguien agita una campana, esta vibra y genera un sonido, si se tiene el dispositivo adecuado para grabar el sonido emitido por la campana luego de agitarla y además de ello se puede mostrar su representación gráfica, va a lucir tal como se muestra en la Figura 1.1.

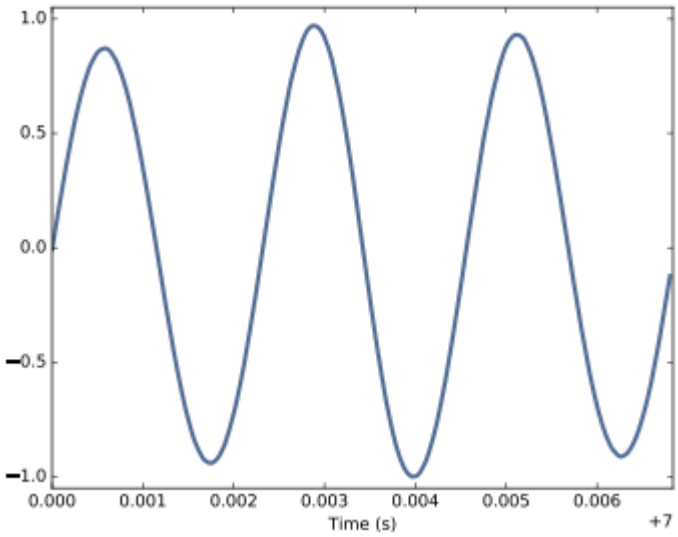


Figura 1.1. Segmento grabado del sonido emitido por una campana

Fuente: (Downey, 2014)

La señal obtenida, es una sinusoide lo que significa que tiene la misma forma que la función seno de la rama trigonométrica.

A partir de ello se pueden obtener ciertas conclusiones, ya que como se puede observar se trata de una función periódica, donde cada periodo tiene una duración cercana a los 2.3 ms.

Otro aspecto a considerar al momento de tratar con una señal sinodal es su frecuencia, la cual hace referencia al número de ciclos por segundo del instante de tiempo que ocurre la señal, esto es representada como el inverso del periodo su unidad de representación métrica es el *Hertz* ***(Hz)***.

La frecuencia de la señal representada en la figura 1.1, es aproximadamente *439 Hz* muy cercana pero menor que los *440 Hz*, que representan a la frecuencia estándar para un tono de orquesta.

1.2 Señales No Periódicas

Las señales no periódicas representan a tipos de onda en las cuales los componentes de su frecuencia cambian a lo largo del tiempo en otras palabras, muy parecidas a las señales de sonido. (Manolakis y Proakis, 2007)

La presencia de señales no periódicas son frecuentes en los sistemas de tratamiento de señales, un ejemplo de ello es el chasquido lineal, quien es una representación de una señal con frecuencia variable, un ejemplo de una señal no periódica puede ser visualizado en la Figura 1.2.

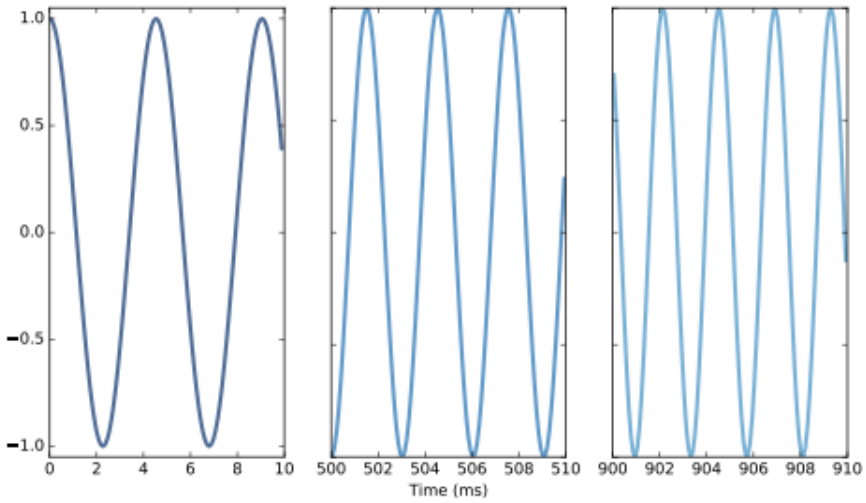


Figura 1.2. Señal No Periódica frecuencia variable

Fuente: (Downey, 2014)

2 Señales y sistemas

En términos generales del procesamiento de señales, un **Sistema** es una representación abstracta de cualquier objeta que toma una señal a su entrada y produce una diferente a su salida, la representación de un sistema simple viene ilustrado por medio de la Figura 1.3.

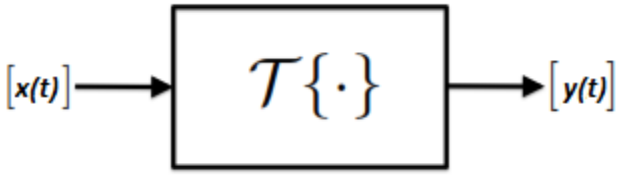


Figura 1.3. Entrada, Salida: descripción gráfica de un sistema.

Fuente: (Diepold, 2016)

Por ejemplo, un amplificador electrónico es un circuito que toma una señal eléctrica como entrada y produce una señal mucho más alta a su salida.

Citando otro ejemplo, cuando alguien escucha la música a lo largo del tiempo, imaginando que esta persona se encuentre dentro de una habitación la cual representaría al sistema, mismo que toma el comportamiento del sonido en la ubicación donde se esté

generado y producirá un sonido diferente dependiendo de la ubicación donde el escucha se encuentre y lo pueda percibir.

Resaltando lo detallado hasta este punto, cabe señalar que los sistemas a los que se hace mención se rigen y agrupan bajo varias propiedades que los distinguen unos de otros e indican bajo qué características de la señal de entrada entregaran cierto resultado en su salida.

2.1 Propiedades de los Sistemas discretos en el tiempo

(1) Causalidad.- Un sistema *S* es **Causal** si la salida en un tiempo ***t*** no depende de valores futuros de la señal de entrada, es decir únicamente contará con valores presentes y pasados de la señal de entrada.

(2) Sistemas Lineales y No Lineales.- Un sistema ***S*** puede ser:

- **Aditivo** si para dos entradas diferentes $x_1(t),\ x_2(t)$:

$$S\{x_1(t) + x_2(t)\} = S\{x_1(t)\} + S\{x_2(t)\}$$

Ecuación 1.1 Propiedad aditiva de los sistemas lineales
Fuente: (Manolakis y Proakis, 2007)

- **Homogéneo** si, para cualquier entrada $x(t)$, y cualquier numero a,

$$S\{a * x(t)\} = a * S\{x(t)\}$$

Ecuación 1.2 Homogeneidad de los sistemas lineales
Fuente: (Manolakis y Proakis, 2007)

Un sistema que cumple con ambas propiedades **Aditivo** y **Homogéneo** es conocido como **lineal**, en otras palabras, *S* es lineal si, para cualesquier 2 entradas $x_1(t),\ x_2(t)$ y cualesquier dos números $a_1\ ,\ a_2$

(3) Variabilidad en el tiempo.- Si para cualquier entrada $x(t)$, y cualquier punto en el tiempo t_1, $S\{x(t - t_1)\}$ es igual a una salida producida por la señal de entrada $y(t - t_1)$, donde $y(t)$ es la salida de la señal de excitación $x(t)$, se obtiene: $y(t) = S\{x(t)\}$ dicha ecuación responde a un sistema invariante en el tiempo.

Los sistemas que no son invariantes en el tiempo, son llamados ***Sistemas Variantes en el tiempo***.

(4) Estabilidad.- Se dice que un sistema discreto es estable si ante cualquier entrada acotada, la salida del sistema permanece acotada, cabe señalar que la señal de entrada y salida,

debe estar acotada únicamente en amplitud, es decir su amplitud se encuentra en un valor constante.

La definición matemática establecida para lo antes expuesto viene representado por la Ecuación 1.3.

$$\sum_{k=-\infty}^{\infty} |h(k)| < \infty$$

Ecuación 1.3 Definición matemática para determinar estabilidad de un sistema
Fuente: (Manolakis y Proakis, 2007)

(5) Memoria.- Un sistema ***S*** es estático o sin memoria, si la salida en cualquier punto *t*, depende solo de valores actuales de la señal de entrada *t*, en otro caso el sistema se dice tener memoria o se lo conoce como sistema **dinámico**. Por dicha definición se dice que un sistema el cual es **estático** o sin memoria es **causal,** sin embargo un sistema **causal** no necesariamente es estático.

2.2 Propiedades fundamentales de los sistemas invariantes en el tiempo LTI.

Dentro del procesamiento digital de señales generalmente el interés al tratar una señal es calcular la secuencia de salida $[y(t)]$ de un ***sistema invariante en el tiempo***, el cual viene dado en términos de una función $\tau\{.\}$ que es excitado a través de una señal de entrada $x(t)$. Como ejemplo refiérase a la Figura 1.3.

Donde:

$$[x_k] = [\ldots.[x_{k-1}, x_k, x_{k+1}], [y_k] = [\ldots.[y_{k-1}, y_k, y_{k+1}]$$

Obviamente *k* puede representar un valor entero sea 1, 2,3,…lo cual marca el índice de la escala de tiempo de la secuencia de entrada.

Para que un sistema sea lineal se requiere que el principio de superposición se sostenga, esto significa, para dos secuencias de entrada $x_1(t)$ y $x_2(t)$ las salidas correspondientes pueden sumarse, esto es definido dentro de la propiedad **Aditiva** descrito en la sección anterior.

2.3 Convolución lineal

La secuencia de salida y_k es determinada por la operación de **Convolución.** Ahora se procederá hacer una explicación de la tarea computacional de **Convolución** de dos señales de la manera más corta y eficiente. Si se considera una secuencia discreta y finita $[u_k], k = 0, 1, 2, \ldots n-1$ de longitud *n*. Cuando se alimenta con esta secuencia de entrada un **sistema invariante en el tiempo** que esta descrito bajo la asociación de la respuesta al impulso $[t_k], k = 0, 1, 2, \ldots m-1$ de longitud *m*, se puede calcular la **Convolución** lineal discreta de las dos secuencias denotadas como x_k y t_k bajo el siguiente criterio matemático, definido por la Ecuación 1.4, que da como resultado la representación gráfica de la Figura 1.4.

$$[y_k] = [u_k] * [t_k] = \sum_{k=-\infty}^{\infty} [t_{k-1}] * u_k, k = 0, 1, 2, \ldots . m + n - 2.$$

Ecuación 1.4 Expresión matemática para la convolución de dos señales.

Fuente: (Manolakis y Proakis, 2007)

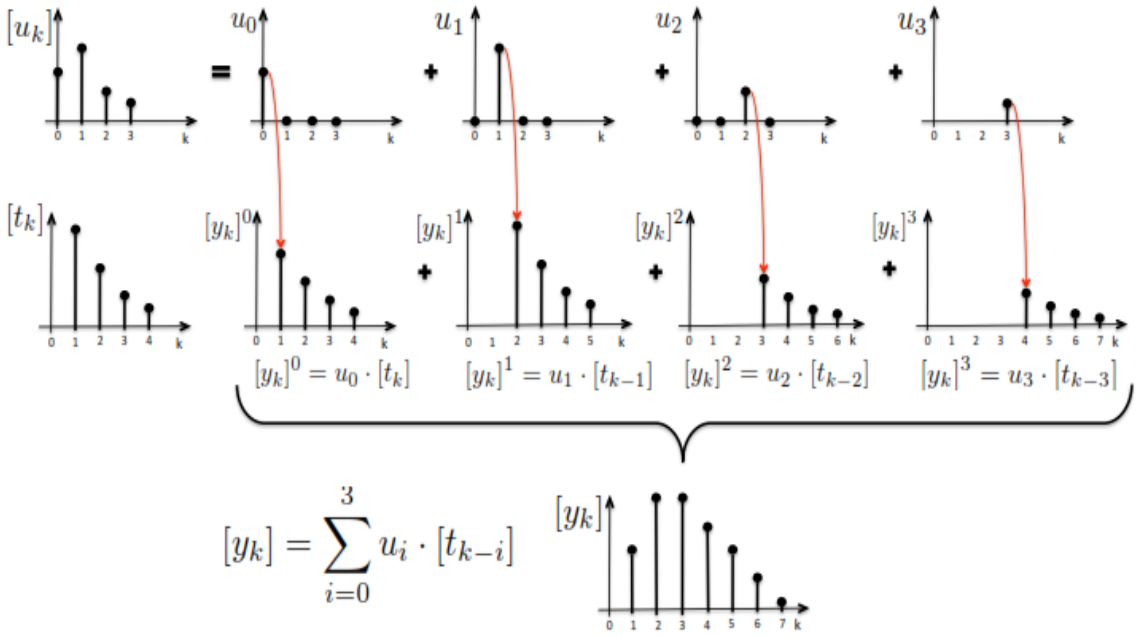


Figura 1.4. Cálculo de la Convolución de una señal de entrada y una señal con respuesta al impulso.

Fuente: *(Diepold, 2016)*

Con una secuencia de u_k de longitud *n* y una respuesta al impulso t_k de longitud *m*, la longitud de la señal de salda será marcada como *N = m+n-1*

Todas las ecuaciones o modelos matemáticos bajo los que trabajan los sistemas descritos, son insertados dentro de mecanismos electrónicos o computacionales para representar el

Algoritmo de cálculo. El concepto de vital importancia dentro del tratamiento de señales es la definición de la palabra **Algoritmo**, que implica el seguimiento lógico de una secuencia de pasos que deben ser seguidos para obtener un resultado esperado.

Se puede utilizar la definición de algoritmo en términos de propósito general para el procesamiento de información, la diferencia clave es la naturaliza de los datos que se desea sean procesados. En procesamiento de señales la secuencia de datos contiene información que no es explícitamente digital y por tal motivo usualmente es imprecisa.
La tecnología del procesamiento digital de señales confía en varios campos científicos y de ingeniería, pero los campos clave son:

- **Electrónica Analógica:** para capturar la cantidad real y pre procesarla dentro de una computadora sofisticada.
- **Representación digital:** misma que requiere un muestreo discreto de todos los valores de la señal tomada de cualquier fuente de información natural, en intervalos de tiempo discreto para únicamente dejar valores discretos y puedan ser procesados por un computador sofisticado.
- **Análisis matemático:** requerida para encontrar vías de análisis y entender señales complejas y variantes en el tiempo, las matemáticas ayudan a definir el proceso de los algoritmos requeridos.
- **Algoritmos de *software*:** requerido para implementar matemáticamente las ecuaciones que describen las señales muestreadas sobre una computadora.

3 Programas utilizados para simulación de algoritmos y Tratamiento Digital de Señales

En la actualidad existen alternativas que permiten simular y desarrollar algoritmos para interpretación del tratamiento digital de señales entre los más populares se encuentra MATLAB, que de por sí es un paquete de librerías que conforman un entorno de laboratorio bajo *software* para tratamiento digital de señales, de hecho es el *software* por excelencia elegido para este tipo de aplicaciones. Sin embargo, el costo y licenciamiento del mismo resulta una dificultad para la adquisición y prevalencia de este *software* sobre otros, para ello la alternativa que se ha popularizado de manera notable, es el uso del entorno de programación **Python** que bajo los componentes necesarios se lo puede hacer funcionar cual si de Matlab se tratase. Para ello usa un paquete de librerías que lo transforman en una

herramienta de *software* interactiva con funciones como estas, se pueden acelerar los procesos de pruebas, compilaciones y verificación de algoritmos escritos en código **Python**.

4 Python vs Matlab

Matlab (**MAT**rix **LAB**oratory, "Laboratorio de matrices"), es la abreviatura del tan popular *software* matemático que ofrece un entorno de desarrollo integrado, con lenguaje de programación propio conocido como lenguaje M.

Matlab se encuentra disponible para plataformas de Windows, Unix y Apple Mac OS X. Sus prestaciones básicas consisten en la manipulación de matrices, representación de datos y funciones, implementación de algoritmos, creación de interfaces de usuarios (GUI) además de la comunicación con programas escritos en otros lenguajes y comunicación con dispositivos en hardware como Arduino y Raspberry PI.

Matlab es un entorno de cálculo numérico comercial, el concepto de Matlab refiere a un paquete completo que incluye su propio IDE. Las librerías estandarizadas no contienen funcionalidades genéricas de programación, sin embargo incluyen una extensa librería de algebra matricial así mismo una extensa librería para funcionalidades de procesamiento de datos y graficación.

Matlab tiene varias ventajas interesantes para la programación científica, no obstante, la mayor parte de estas ventajas también las posee **Python**.

Los problemas o desventajas del uso de Matlab, son la propietariedad de sus algoritmos, esto significa que no se puede mirar dentro del código fuente de los algoritmos ya implementados y repercute en confiar que el código que el usuario se encuentre utilizando se encuentre correctamente implementado.

Sin duda alguna otra desventaja que presenta Matlab es su alto costo, esto significa que únicamente tienen acceso a su código las personas con los fondos suficientes como para adquirir su licencia. **Mathworks** empresa que se encarga de la distribución del *software* Matlab con elevados costos de licencias, ha puesto restricciones en el código de Matlab, restricciones que impiden que el código creado en el ambiente de programación Matlab puedan ser llevados hacia otro computador para que se ejecute la aplicación creada, siempre que el computador no tenga la misma versión de Matlab donde se creó el programa base.

No todo son desventajas en cuanto al *software* por excelencia para Tratamiento de Señales Matlab, este también posee grandes ventajas a las que aún no se le han presentado alternativas que impliquen usar algún otro *software* para suplantarlo, como por ejemplo el paquete ***Simulink***, que consiste en un conjunto de bloques que ejecutan determinadas funciones para la simulación de algoritmos, fácil de entender y configurar los parámetros de cada bloque el usuario encontrara gran versatilidad en la creación y simulación de algoritmos así mismo podrá validar los resultados mediante gráficos que presenta la aplicación., la Figura 1.5, muestra la diferencia de los entornos de desarrollo Matlab y Python.

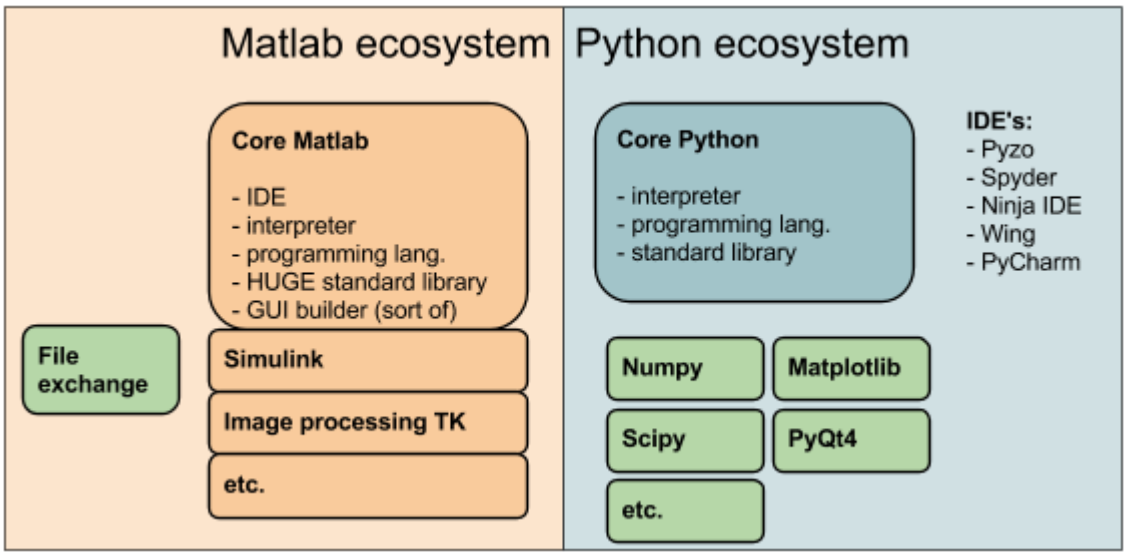


Figura 1.5. Comparación de los ecosistemas Martlab y *Python*

Fuente: *(pyzo.org, 2016)*

En alternativa para la implementación de algoritmos del Tratamiento digital de Señales, se ha popularizado el uso de Python, por definición es un lenguaje de programación y no solamente eso, Python consiste en una extensa librería estandarizada, cada librería apunta a un entorno de programación en general y contiene módulos específicos para cada necesidad ya sea el campo de Redes de Datos, Base de Datos, etc.

Para desenvolverse en el campo de computación científica en **Python**, se necesitan paquetes adicionales como (**Numpy, Scioy, Matplotlib**), además de un IDE.

Debido a que **Python** es un *software* abierto y gratuito, es muy fácil para los miembros diseñar paquetes o herramientas que extiendan las funcionalidades de **Python**, es posible crear aplicaciones usando la mayoría de librerías de GUI (Interfaces Gráficas para Usuarios).

Cada paquete es desarrollado por diferentes grupos de personas con diferentes interese de investigación, no obstante cada uno de ellos tiene su debido proceso para crear una guía de referencia, ayuda y soporte por lo que para cualquier interesado en profundizar algún campo de las librerías ofertadas en el *software* libre **Python**, encontrará documentación y código que ayudará a comprender mejor las funcionalidades de la librería investigada.

4.1 Ventajas del uso de Python

- El *software* es gratuito, no tiene costo alguno y el usuario puede mirar dentro del código de cada algoritmo y modificarlo de ser el caso.
- **Python** fue creado para convertirse en un lenguaje de programación genérico es decir, fácil de leer, mientras que Matlab se basa en la manipulación de matrices para la programación, empezar en **Python** se hará más sencillo de trabajar para estudiantes con cierto concepto en programación.
    - Los espacios para nombres, pese a que Matlab soporta espacios para nombres propios de funciones escritas por los usuarios, el núcleo de Matlab no los almacena, cada función debe ser definida en el espacio de nombres de manera global. **Python** sin embargo trabaja con módulos, con lo que se necesita solamente importar el modulo que se desea utilizar, por ejemplo (*for sympy import symbols)*
    - La manipulación de variables *String*, esto es increíblemente fácil de entender en **Python**, acotemos esta línea de código que retorna 30 caracteres justificados a la derecha: **“I code in Matlab”.replace(‘Matlab’,’Python’).rjust(30)**
    - La portabilidad que ofrece **Python**, debido a que es gratis, se pueden correr las aplicaciones desarrolladas en cualquier lugar, funciona en Windows, Linux y OS X.

4.2 ¿Qué es Python?

Python es una herramienta muy versátil, es un intérprete sofisticado y todo un entorno de programación **multi-paradigma** (sirve para programación concurrente, programación orientada a objetos, etc). Se lo puede considerar para el desarrollo de pequeñas aplicaciones o para proyectos de mayor envergadura. Además de las mencionadas funcionalidades que trae de serie este lenguaje de programación, se las puede extender, y convertirlo en un laboratorio de *software* para: tratamiento de datos, visualización, cálculo numérico y simbólico entre otras aplicaciones específicas. Para

esto, debemos instalar librerías que circulan por Internet gracias a que al ser un *software* "open source", existe una gran comunidad de usuarios dispuestos a compartir su código y mejorar sus características y potenciales de aplicación.

Cuando se empieza a trabajar en computación lo primero que llega a sorprender es la gran cantidad de lenguajes de programación, por lo que se debe realizar una cuidadosa selección. La Figura 1.6 lista los lenguajes más utilizados. (Brea, 2013, pág. 4)

Position Nov 2013	Position Nov 2012	Delta in Position	Programming Language	Ratings Nov 2013	Delta Nov 2012	Status
1	1	=	C	18.155%	-1.07%	A
2	2	=	Java	16.521%	-0.93%	A
3	3	=	Objective-C	9.406%	-0.98%	A
4	4	=	C++	8.369%	-1.33%	A
5	6	↑	C#	6.024%	+0.43%	A
6	5	↓	PHP	5.379%	-0.35%	A
7	7	=	(Visual) Basic	4.396%	-0.64%	A
8	8	=	Python	3.110%	-0.95%	A
9	23	↑↑↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.521%	+2.05%	A
10	11	↑	JavaScript	2.050%	+0.77%	A
11	15	↑↑↑↑	Visual Basic .NET	1.969%	+1.20%	A
12	9	↓↓↓	Perl	1.521%	-0.66%	A
13	10	↓↓↓	Ruby	1.303%	-0.44%	A
14	14	=	Pascal	0.715%	-0.17%	A
15	13	↓↓	Lisp	0.706%	-0.25%	A
16	19	↑↑↑	MATLAB	0.656%	+0.04%	B
17	12	↓↓↓↓↓	Delphi/Object Pascal	0.649%	-0.35%	A-
18	17	↓	PL/SQL	0.605%	-0.03%	A-
19	24	↑↑↑↑↑	COBOL	0.585%	+0.11%	B
20	20	=	Assembly	0.532%	-0.05%	B

Figura 1.6. Lenguajes de programación más utilizados en el mundo

Fuente *(Brea, 2013)*

5 PyLab, transformando Python en Matlab

Python es un lenguaje de programación de propósito general que, no se pensó en un principio para aplicaciones científicas. Con el pasar de los años bajo la filosofía GNU, se han venido desarrollando multitud de módulos científicos o librerías las cuales permiten realizar numerosas tareas de tratamiento de datos, visualización, cálculo

simbólico y aplicaciones científicas específicas. Entre la infinidad de librerías que circulan por la red gracias a la colaboración y desarrollo de varios usuarios, nos detenemos ante **PyLab** que no es una librería sino un conglomerado de varias librerías entre las que se incluyen **Numpy, Scipy, Sympy, Matplotlib, IPython**.

Con esta *suite* podremos usar de manera muy parecida al intérprete de **Python** como si de MATLAB se tratase y hacer nuestros archivos con extensión *.py* como si fueran scripts de MATLAB.

PyLab consigue mejores rendimientos en el cálculo que MATLAB en casi todas las operaciones, tiene también la ventaja de servir para Linux y para Windows y no tener que estar obligados a utilizar el sistema de Microsoft, además al estar programando en **Python** se pueden juntar programas de cálculo con todas las demás opciones que éste ofrece, sin olvidar que la sintaxis es la de **Python** conocida sin duda alguna por su simplicidad.

6 Descripción de las librerías o módulos científicos de Python

A continuación se da una breve descripción de las librerías científicas de Python.

6.1 Numpy: Librería encargada de la generación de tipos de datos científicos.

Este es uno de los módulos más importantes dentro de **Python**, su origen se debe al diseñador de *software* Jim Hugunin quien diseñó el módulo Numeric para que de esta forma **Python** adquiera capacidades de cálculo similares a otros *software*s como Matlab. Luego de ello se mejoró Numeric adquiriendo nuevas funcionalidades dando así el nacer de lo que se conoce hoy como el módulo de Numpy.

Toda la capacidad matemática y vectorial incorporada en **Python** se debe a Numpy, haciendo posible operar con cualquier dato numérico o ***Array***. Las operaciones básicas desde adiciones hasta multiplicaciones incluso más complejas como la transformada de Fourier o el álgebra lineal, son parte de la solución que brinda este módulo.

Numpy provee especificaciones de tamaño de bits para los Arrays definidos en **Python**, por ejemplo si se crea un Array de tres números, cada uno tendrá 4 bytes (32 bits en 8 bits por byte) para mostrar.

Para cargar el módulo Numpy en **Python**, se ha estandarizado el alias ***np*** para poder llamarlo.

La manera estandarizada se presenta a continuación en un ejemplo de línea de comando:

```
>>> import numpy as np
```

(NumPy developers. , 2018)

6.1.1 Arrays

Es probable que cualquier lector que haya realizado un acercamiento hacia algún lenguaje de programación este familiarizado con el termino ***Array***. Un ***Array*** es el término que traslada el concepto matemático de matriz o vector a la programación, dándole noción de almacenamiento en memoria. Cada ***Array*** contiene en su interior una serie de argumentos ordenados en filas o columnas, dependiendo de la dimensión.

La finalidad del desarrollo del módulo Numpy, es justamente la creación y modificación de ***Arrays*** multidimensionales. Para crear ***Arrays*** multidimensionales se puede utilizar la clase *nd**Array*** del inglés ***N-dimensional Array.*** En **Python** cada clase puede tener atributos que se llaman mediante métodos o escribiendo a continuación de la clase utilizada un punto y el atributo. En la mayoría de IDE al cargar la clase y escribir el punto aparecen todos los atributos disponibles en orden alfabético por lo que en el caso de dudar siempre podemos utilizar este método para escribir el comando. En el caso de *nd**Array*** los atributos principales son los que se listan a continuación:

- **nda*rray*.ndim** –> Proporciona el número de dimensiones de nuestro ***Array***. El ***Array*** identidad es un ***Array*** cuadrado con una diagonal principal unitaria.
- **nd*array*.shape** –> Devuelve la dimensión del ***Array***, es decir, una tupla de enteros indicando el tamaño del ***Array*** en cada dimensión. Para una matriz de n filas y m columnas obtendremos (n,m).
- **nda*rray*.size** –> Es el número total de elementos del ***Array***.
- **nda*rray*.dtype** –> Es un objeto que describe el tipo de elementos del ***Array***.
- **nda*rray*.itemsize** –> devuelve el tamaño del ***Array*** en bytes.
- **nda*rray*.data** –> El buffer contiene los elementos actuales del ***Array***.

Los ***Array*** en Numpy, vienen definidos en varias formas y dimensiones. Por ejemplo la Figura 1.7 muestra un ***Array*** bidimensional de 2x3 al seguir una secuencia de comandos.

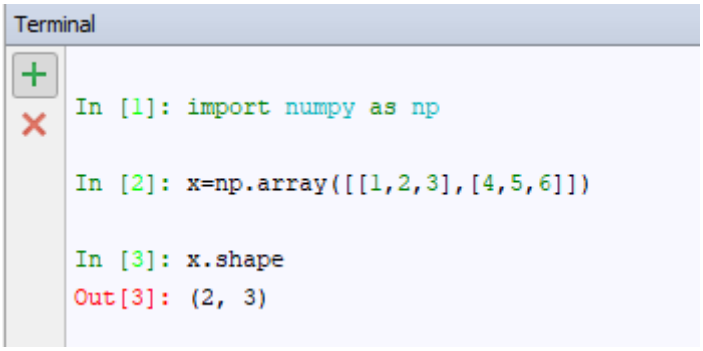


Figura 1.7. Ejemplo de un Array de dimensiones 2x3

Fuente: (Elaborado por el autor)

Se debe tener en cuenta que la librería **Numpy**, compilada debe estar localizada dentro de la memoria del intérprete de **Python,** los comandos **Numpy** que ingresan a la memoria de **Python**, son método de entrada, así mismo la salida de datos se muestran en el intérprete **Python** como comandos identificados por la librería de **Numpy**.

Trabajar con **Matrices** en **Numpy**, es muy similar a declarar ***Arrays***, pero se implementa la multiplicación de filas por columnas es decir la multiplicación por el elemento opuesto.

Si se desea multiplicar dos **Matrices**, se puede crear esta resultante directamente, o a su vez convertirla en ***Arrays*** de **Numpy**. Por ejemplificar se adjunta el siguiente procedimiento en la línea de consola representado en la Figura 1.8.

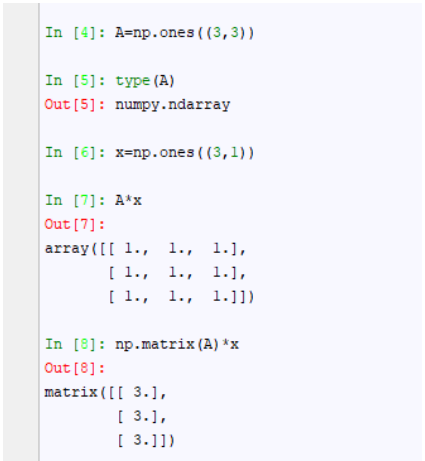


Figura 1.8. Conversión de Array a Matriz mediante multiplicación de filas por columnas

Fuente: (Elaborado por el autor)

6.2 **Scipy:** Librería con funciones científicas de uso general.

La librería Scipy, es una herramienta numérica para **Python** que se distribuye libremente.

El módulo Scipy otorga en general capacidades de cálculo numérico de gran capacidad para **Python**. Dentro del núcleo de la estructura de **Scipy** se encuentra incluida la librería de **Numpy**, las descripciones breves que se pueden acotar de las capacidades que esta librería otorga a **Python** son los módulos para optimización de funciones, integración, funciones especiales resolución de ecuaciones diferenciales ordinarias y muchos otros aspectos, la apariencia del sitio de Scipy, es ilustrado en la Figura 1.9.

(SciPy developers. , 2018)

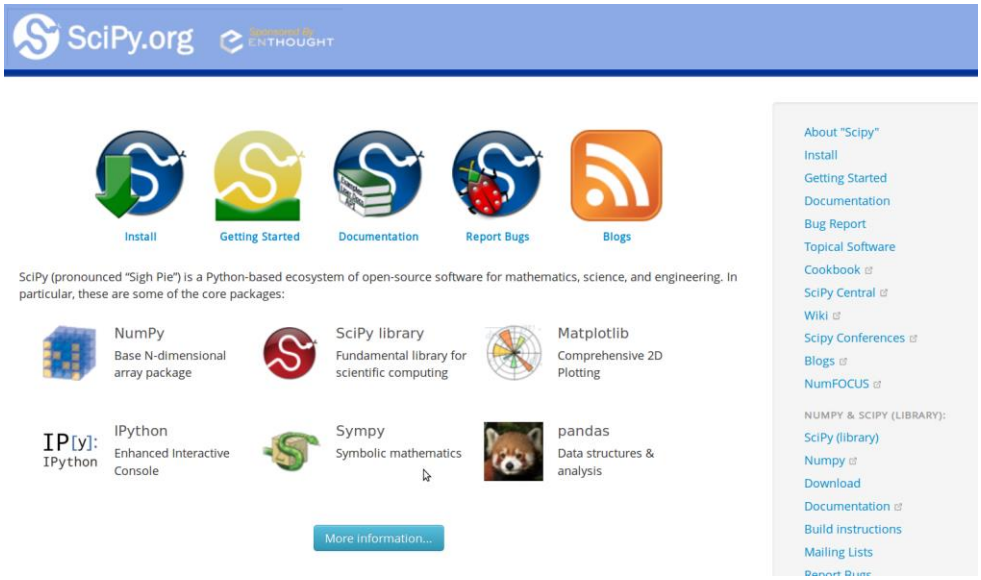


Figura 1.9. Componentes de la librería SciPy

Fuente (SciPy.org, 2018)

La organización de Scipy se encuentra dentro de una estructura de subpaquetes mismos que pueden ser considerados como especializados en dominios científicos determinados, se puede acceder a ellos según la ayuda de Scipy:

- ***linalg*** *– Algebra lineal*
- ***signal*** *– Procesamiento de señales*
- ***stats*** *– Funciones estadísticas*
- ***special*** *– Funciones especiales*
- ***integrate*** *– Integración*
- ***interpolate*** *– Herramientas de interpolación*
- ***optimize*** *– Herramientas de optimización*
- ***fftpack*** *– Algoritmos de transformada de Fourier*
- ***io*** *– Entrada y salida de datos*
- ***lib.lapack*** *– Wrappers a la librería LAPACK*

- ***lib.blas*** *– Wrappers a la librería BLAS*
- ***lib*** *– Wrappers a librerías externas*
- ***sparse*** *– Matrices sparse*
- ***misc*** *– otras utilidades*
- ***cluster*** *– Vector Quantization / Kmeans*
- ***maxentropy*** *– Ajuste a modelos con máxima entropía*

Para importar los paquetes de la librerìa Scipy, estos deben ser llamados de forma separada como se muestra en el ejemplo de la siguiente línea de comando:

```
>>> from scipy import linalg,optimize
```

6.3 SymPy: Librería con funciones para cálculo simbólico.

SymPy es una librería de **Python**, para ejecución y operación con matemáticas simbólicas. Es decir, esta librería que contiene un completo sistema algebraico computarizado con la simplicidad que el lenguaje de programación **Python** ofrece para hacerlo más expandible.

La computación simbólica llama al cálculo matemático de variables de manera simbólica o representativa. Esto implica que los objetos deben ser representados de manera exacta, no aproximada, matemáticamente las expresiones con variables no evaluables, son dejadas en forma simbólica. Para importar la librería de sympy y sus componentes, utilizamos la forma siguiente:

```
>>> from sympy import *
```

(SymPy Development Team. , 2018)

6.4 Matplotlib: Librería para generación de gráficas 2D y 3D.

El módulo de Matplotlib, es el módulo de ilustración o graficación donde se muestran gráficas 2D y 3D, las cuales se han de utilizar para la presentación de datos.

La documentación oficial se encuentra en: http://matplotlib.sourceforge.net/contents.html

Así mismo para agregar los módulos de Matplotlib, se puede utilizar la siguiente línea de comando:

```
>>> import matplotlib.pyplot as plt
```

(Matplotlib development team, 2012-2018)

Matplotlib al igual que la mayoría de módulos contiene un gran contenido de funciones, sin embargo la documentación se encuentra en el sitio web de la librería, por ello únicamente se citaran los más importantes para que se pueda dar uso rápido y adecuado de esta librería para el lector interesado.

- ***num*** = *numeración de la figura,* ***si num*** = *None, las figuras se numeran automáticamente.*
- ***figsize*** = *w, h tuplas en pulgadas. Tamaño de la figura*
- ***dpi*** = *Resolución de la imagen en puntos por pulgada.*
- ***facecolor*** = *Color del rectángulo de la figura.*
- ***edgecolor*** = *Color del perímetro de la figura.*
- ***frameon*** = *Si es falso, elimina el marco de la figura.*

A continuación la Figura 1.10 hace representación de una secuencia de comandos los cuales dibujan una gráfica, a la cual se la representa mediante la Figura 1.11.

```
In [13]: import matplotlib.pyplot as plt

In [14]: plt.plot(range(10))
Out[14]: [<matplotlib.lines.Line2D at 0x1e7b69c3dd8>]

In [15]: plt.show()
```

Figura 1.10. Uso de la librería Matplotlib, comandos para gráfica simple

Fuente: (Elaborado por el autor)

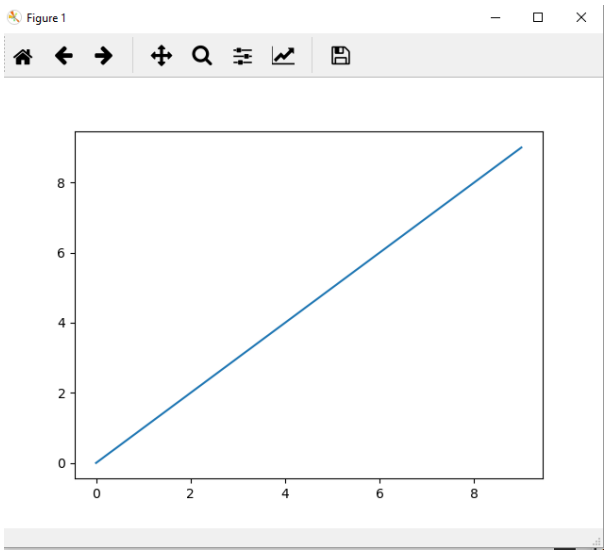


Figura 1.11. Resultado gráfico de los comandos empleados en la figura anterior

Fuente: (Elaborado por el autor)

6.5 **IPython:** Interprete de comandos **Python**, para hacer una consola interactiva con el usuario.

Se trata de una consola interactiva la cual adiciona funcionalidades extra al modo interactivo incluido con el IDE de **Python**, por ejemplo el resaltado de líneas de error con colores, I**Python** es un componente del paquete científico SciPy. El intérprete i**Python**, tiene un componente online basado en una herramienta web, a la cual de se la denomina ***NOTEBOOK*** *(LIBRETA)*, esta libreta es un documento que contiene una lista ordenada de entradas y salidas las cuales pueden tener código, texto, matemáticas, dibujos. Los notebooks de I**Python** pueden ser convertidos a otros formatos de archivos como HTML, presentaciones diapositivas, PDF, **Python**, etc. (https://es.wikipedia.org/wiki/IPython, 2018)

Por medio de la consola interactiva de I*Python*, el usuario puede hacer consultas de la especificación de algún comando, obtener ayudas en cuanto a la sintaxis empleada y la funcionalidad de dicho comando adhiriendo al final del comando el símbolo "***?***", a continuación se ejemplifica lo descrito mediante las Figuras 1.12 y 1.13.

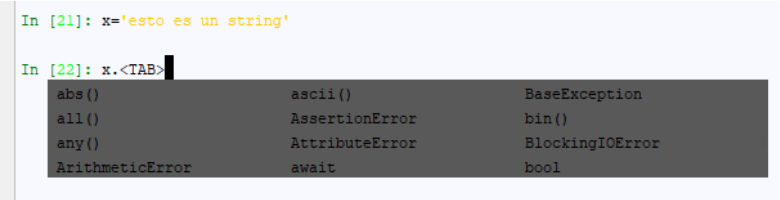


Figura 1.12. Argumentos permitidos para la variable declarada x, mostrados al presionar

Fuente: (Elaborado por el autor)

(IPython development team, 2018)

```
In [22]: x.capitalize?
Docstring:
S.capitalize() -> str

Return a capitalized version of S, i.e. make the first character
have upper case and the rest lower case.
Type:      builtin_function_or_method
```

Figura 1.13. Argumentos permitidos para la variable declarada x, mostrados al presionar

Fuente: (Elaborado por el autor)

La mayoría de usuarios de **Python** son desarrolladores de aplicaciones web, no desarrolladores en campo científico, por tal motivo la cadena de herramientas de **Python** son extensas para dichos motivos, más sin embargo el moderno grupo de desarrolladores a embebido I*Python* en los navegadores web, y se pueden usar las mismas extensiones y módulos interactivos para computación científica sin necesidad de descargar el IDE de **Python** e instalarlo en la PC.

Hace falta únicamente entrar a un navegador con acceso a internet y empezar la búsqueda de I*Python* online, con ello se abrirá el área de trabajo para entrada y salida de datos, la Figura 1.14 muestra la apariencia del Notebook de IPython online, mientras que la Figura 1.15, es la representación de su utilización, demostrando que únicamente se requiere un navegador web para acceder a las funcionalidades de Python.

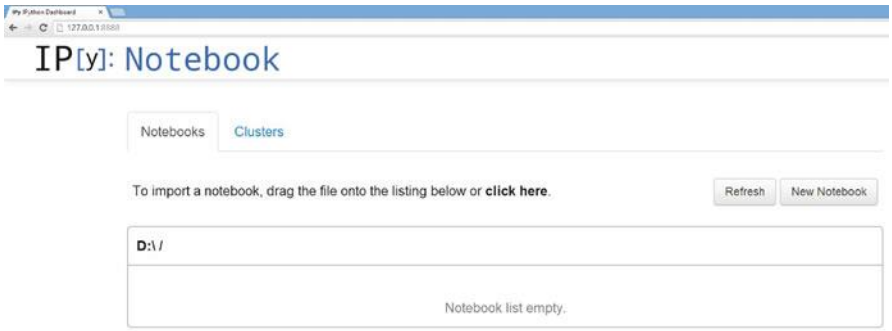


Figura 1.14. Muestra de I*Python* ONLINE embebido en un navegador web a través de sitios de búsqueda de internet

Fuente: (Elaborado por el autor)

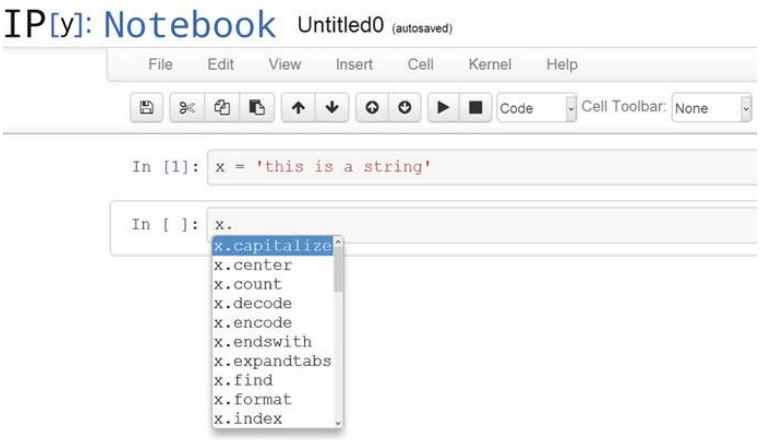


Figura 1.15. Prueba de funcionalidades de I*Python* ONLINE

Fuente: (Elaborado por el autor)

7 Puesta **en Marcha IDE para Python en Windows 10**

Para obtener un entorno de programación unificado, se debe instalar un programa informático que otorga un conjunto de herramientas para la programación, el programa al que se hace referencia puede haber sido desarrollado para interpretar un solo lenguaje de programación o para interpretar varios de ellos, en este caso se hará énfasis al **IDE (Integrated Developement Enviroment) PyCharm** de **Jetbeans**, el cual es un intérprete exclusivo para el lenguaje de programación **Python**.

PyCharm es un Entorno de Desarrollo Integrado (IDE) usado en programación de computadoras exclusivamente para **Python**. El intérprete **PyCharm** es desarrollado por la compañía Checa JetBrains. **PyCharm** provee análisis de código, compilador gráfico, una unidad de prueba integrada, soporta desarrollo web con Django.

PyCharm funciona y puede ser instalado en cualquier plataforma, con Windows, macOS y versiones Linux. La versión de **PyCharm** Comunity es mejorada constantemente bajo la licencia de Apache, existe tambien una edición profesional de **PyCharm** cuya licencia es propietaria de Jetbeans y tiene funcionalidades extras que no se encuentran en la edición Comunity, sin embargo para aspectos de este proyecto la licencia bajo Apache funciona perfectamente.

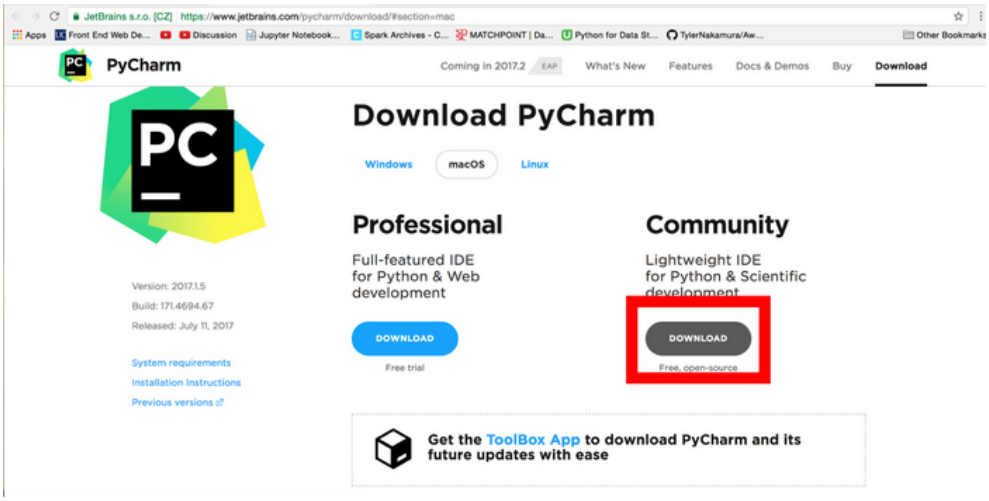
7.1 Instalación de PyCharm en Windows 10

El primer requisito para poner en marcha **PyCharm** como intérprete del lenguaje **Python** en nuestro computador bajo Windows es instalar Anaconda.

Anaconda es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con **Python**. En líneas generales "**Anaconda Distribution"** es una distribución de **Python** que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto, Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, **Anaconda Navigator**, **Anaconda Project**, Las **librerías de Ciencia de datos** y **Conda**. (Desde Linux, 2007-2018)

Se puede descargar Anaconda del enlace: https://www.anaconda.com/download/

Lo siguiente es descargar la edición Comunity de **PyCharm** (Figuras: 1.16 y 1.17) para nuestro sistema operativo, se lo puede realizar a través del siguiente enlace: https://www.jetbrains.com/**PyCharm**/download/#section=windows



Download Pycharm

Figura 1.16. Descargando PyCharm edición Comunity

Fuente: *(medium.com, 2017)*

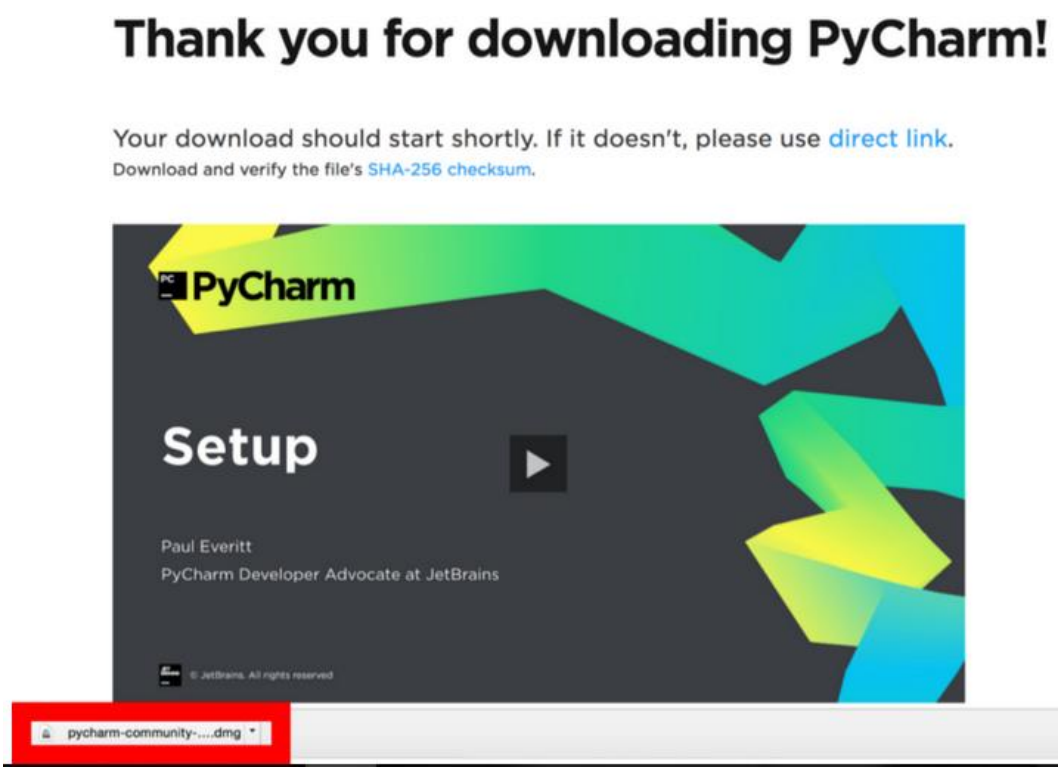


Figura 1.17. Descargando PyCharm edición Comunity

Fuente: *(medium.com, 2017)*

7.2 Descripción del IDE PyCharm

Como se mencionó en el apartado anterior, **PyCharm** es un IDE para **Python**, en general el requisito necesario para empezar el desarrollo de aplicaciones en es únicamente descargarlo, una vez que tengamos listo el programa en nuestro computador empezamos con la creación de un proyecto, cada trabajo que se realiza dentro de **PyCharm** entra en el contexto de un proyecto.

Con esto se encuentra a nuestro servicio una asistencia de código básico, es decir se puede reutilizar el código empleado en otros proyectos, también existe un corrector de sintaxis.

Para trabajar en un proyecto dentro del IDE **PyCharm**. Tenemos 3 opciones:

- **Abrir un proyecto existente:**

Trata específicamente de abrir un proyecto existente almacenado en nuestro computador, se puede acceder al elemento haciendo clic en la opción ***"Open Project"*** en la ventana de bienvenida del IDE, también se puede escoger el archivo a través de la opción ***"Open"***. La Figura 1.18 ilustra lo descrito.

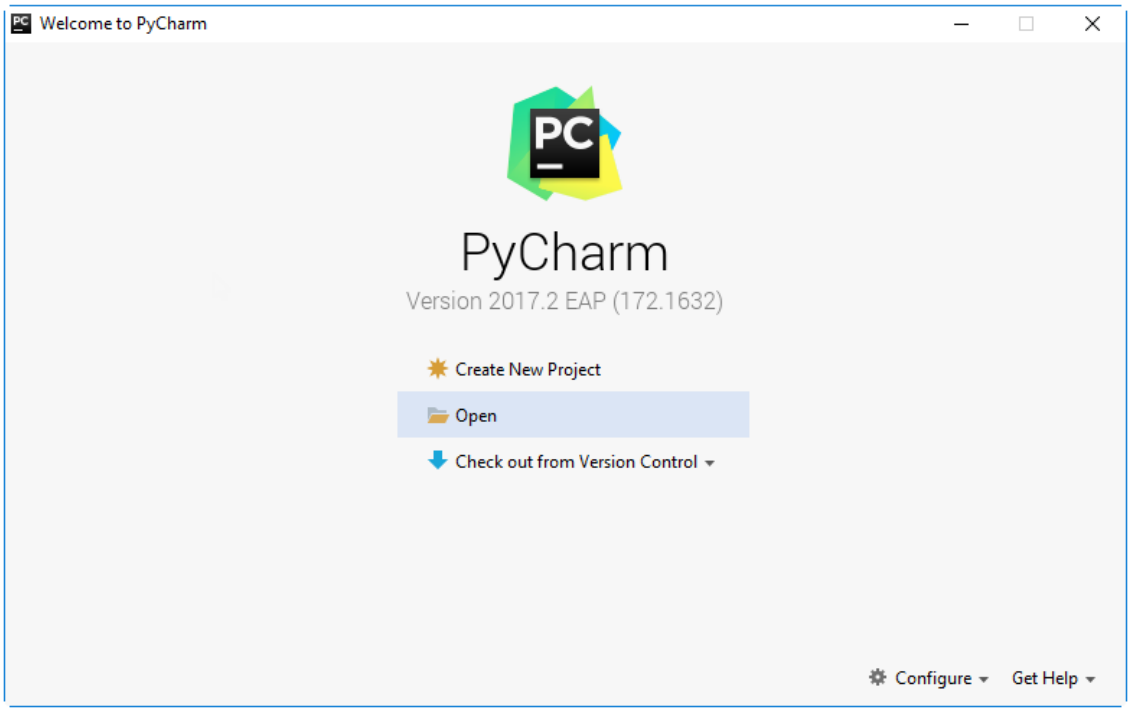


Figura 1.18. Descargando PyCharm edición Comunity

Fuente: (Elaborado por el Autor)

Por otro lado, seleccionando el comando ***"Open"*** en el menú ***"File"***, y especificando el directorio donde se encuentra la fuente de código existente refiérase a la Figura 1.19.

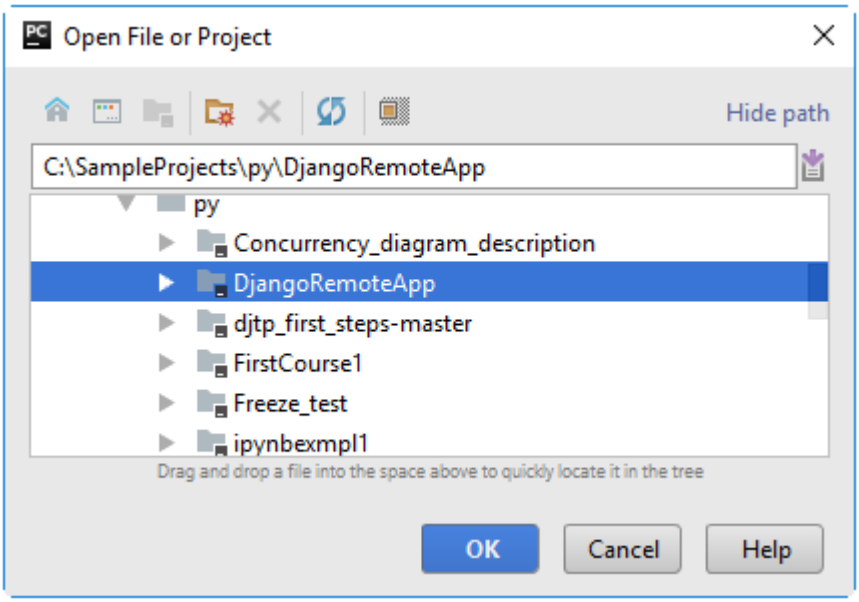


Figura 1.19. Creando un proyecto desde código existente

Fuente: (Elaborado por el autor)

PyCharm creará un proyecto para el usuario, con código existente.

7.3 Entorno de trabajo del IDE PyCharm

Cuando se lanza por primera vez **PyCharm**, o cuando no existen proyectos abiertos, este inicia en la ventana de bienvenida. La ventana de bienvenida provee varios puntos de entrada principales dentro del IDE por ejemplo: crear o abrir un proyecto, verificar un proyecto desde el control de su versión, verificar la documentación y configurar el IDE de acuerdo a la percepción y gusto del usuario.

Los elementos de la interface de usuario del IDE **PyCharm** se pueden apreciar en la Figura 1.20 y son descritos después de ello.

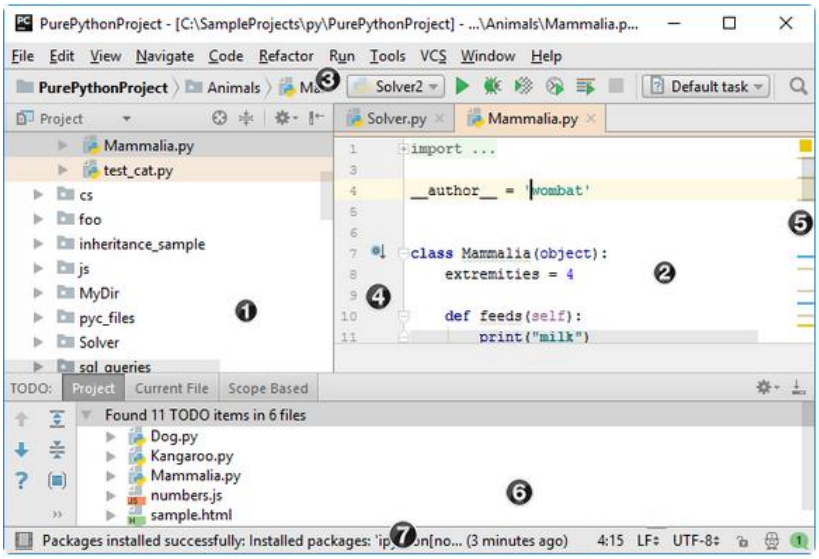


Figura 1.20. Lista de elementos de la Interface de Usuario del IDE PyCharm

Fuente: (Elaborado por el autor)

7.4 Visor de proyectos (1).- La ventana del visor de proyectos se encuentra a la izquierda de la pantalla, permite explorar un proyecto creado desde varios puntos de vista y ejecutar tareas como añadir nuevos elementos ya sean directorios, archivos, clases, etc. Otra funcionalidad que permite esta ventana es abrir los archivos creados en el ***Editor*** para navegar en su código y efectuar los cambios que se crean necesarios.

7.5 Editor (2).- Se encuentra ubicada en el lado derecho del IDE **PyCharm**, esta sección es en la que en realidad se escribe el código del programa que se esté desarrollando o donde se pueden hacer modificaciones de archivos de código Python existentes.

Contiene las características básicas de cualquier otro editor, características como marca página, puntos de interrupción de código (breakpoints), señalador de error de sintaxis, autocompletación de código entre otras. El Editor de **PyCharm** es basado en pestañas, es decir, se puede abrir múltiples archivos que contengan código **Python** en diferentes pestañas del Editor.

El Editor se encuentra dividido en varias secciones, las cuales deben ser identificadas antes de empezar a trabajar en el IDE **PyCharm**. Las secciones del Editor a las que se hace cita, se las puede identificar de acuerdo a la siguiente Figura 1.21.

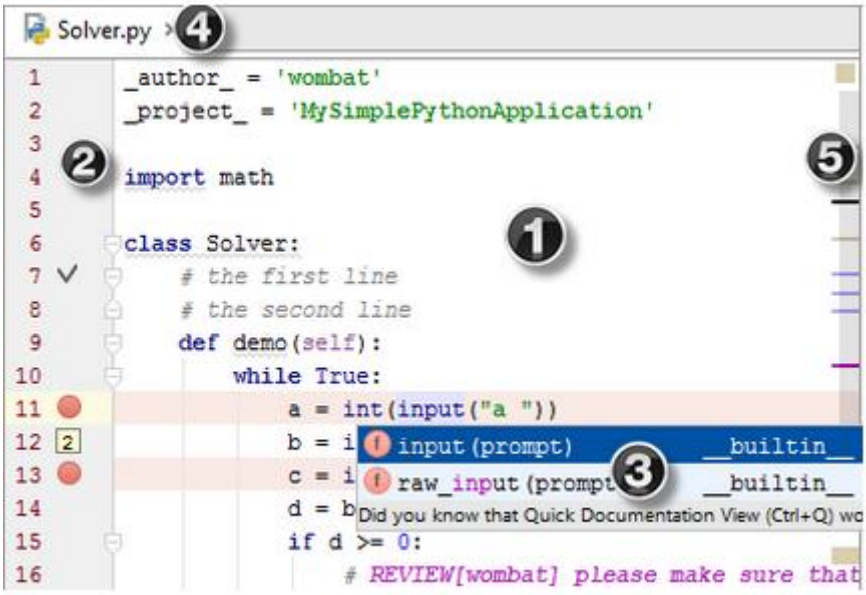


Figura 1.21. Secciones del Editor de PyCharm

Fuente: (Elaborado por el autor)

7.6 Área para editar (1).- En esta sección del Editor, se realiza la escritura o edición del código **Python**, dentro de este campo existe un asistente de código que genera numerosas ventajas para la generación de código con facilidad.

7.7 Área identificadora de línea (2).- Esta sección numera las líneas de código del programa en edición. Por defecto se encuentra con 33 líneas para código, sin embargo el número de líneas permitidas en un programa escrito en **Python**, dependerá del número de líneas necesarias para que el programa creado se ejecute, es decir, existirá un lista de números tan grande como las líneas de código escritas.

En el área identificadora de línea se puede obtener información adicional acerca del código escrito, identificando su estructura, las marcas de página, los puntos interrupción de código.

7.8 Compleción inteligente de código (3).- Es una asistencia para completar el código que se está escribiendo, con sugerencias como las clases, el nombre, el método o la etiqueta que empiece con la letra o contenga la palabra que se ha escrito.

7.9 Etiqueta de pestaña (4).- Identifica el nombre de la pestaña en la que se está trabajando con el código **Python**.

7.10 Barra de validación (5).- En esta barra ubicada al lado derecho del Editor, se muestran los colores, rojo, amarillo y verde, dependiendo de si el código escrito se encuentre con la sintaxis correcta y fuera de errores o advertencias asociará un color a la línea de código escrita . El color rojo representa que la línea escrita contiene algún error de sintaxis, el amarillo indicará que la línea escrita contiene una advertencia que debe ser revisada previo a solicitar la compilación del código, mientras que el color verde marcará un código sin novedad alguna y listo para ser ejecutado.

7.11 Barra de Navegación (3).- La barra de navegación es una alternativa rápida respecto al Visor de Proyectos, se debe usar esta barra para navegar entre proyectos creados y abrir archivos para su edición correspondiente.

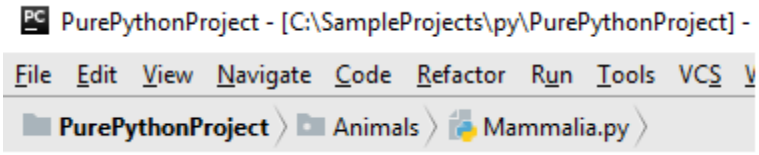


Figura 1.22. Barra de Navegación
Fuente: (Elaborado por el autor)

7.12 Identificador de línea (izquierdo (4).- La banda vertical ubicada a la izquierda del Editor, muestra los puntos de interrupción de código que se hayan colocado, por medio de este identificador se obtiene una forma conveniente de navegar a través de la jerarquía del código escrito, esto es identificar la definición de funciones o declaración de variables. El identificador de línea de código de la izquierda, muestra un número por línea.

7.13 Identificador de línea (derecho) (5).- La banda vertical ubicada a la derecha del Editor, constantemente va monitoreando la calidad del código escrito, es decir que realiza una inspección sobre el código, resaltando los errores o advertencias que contenga.

7.14 Ventana de herramientas.- Provee acceso a tareas típicas de desarrollo en el entorno de trabajo de un proyecto, puede estar ubicada en la parte inferior del entorno de **PyCharm** o en alguno de los costados del mismo, permite la administración del código

fuente, integración con sistemas de control, ejecutar el programa escrito, depuración y prueba de errores, etc.

La ventana de herramientas se encuentra siempre disponible para cualquier proyecto que se ejecute, independientemente de su contenido o configuración, ejemplo de ello es la Figura 1.23.

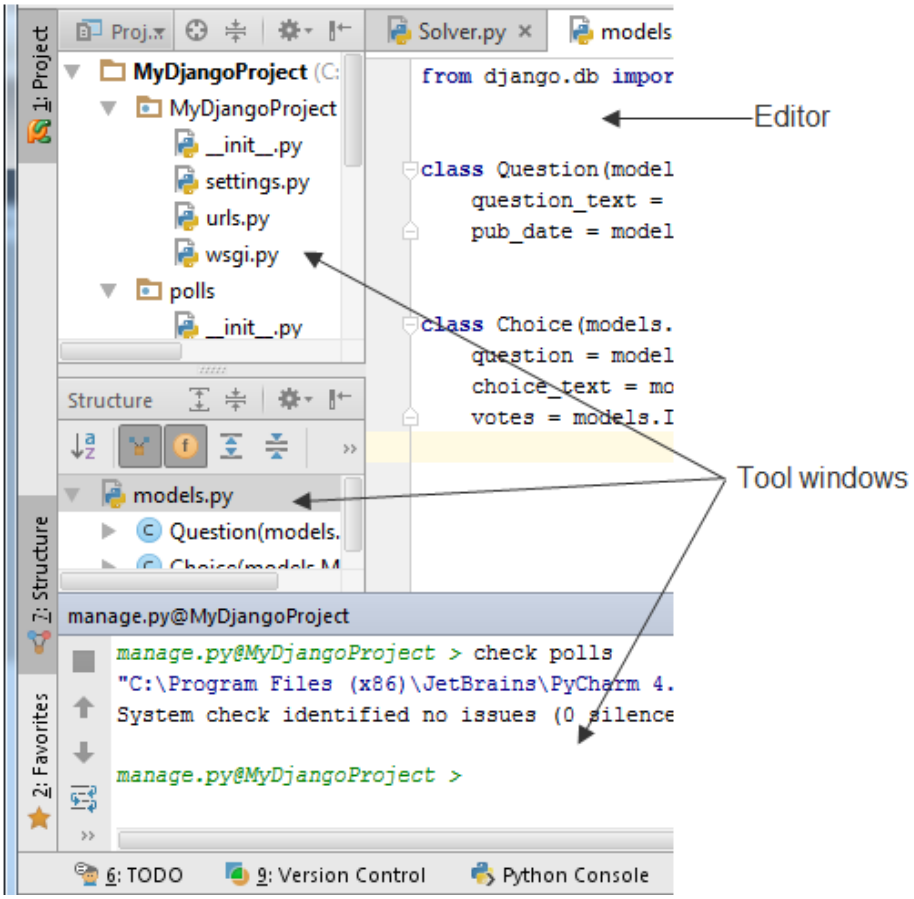


Figura 1.23. Ventana de Herramientas

Fuente: (Elaborado por el autor)

7.15 La barra de estado.- Esta sección de **PyCharm**, se indica el estado entero de un proyecto, muestra varios mensajes con información de código acerca de errores o advertencias, separadores de línea entre otros. Tambien en la parte inferior izquierda de la ventana de **PyCharm**, se pueden apreciar los botones ▭ o ▭. Estos botones muestran la lista de las herramientas disponibles, las cuales son mostradas en la Figura 1.24.

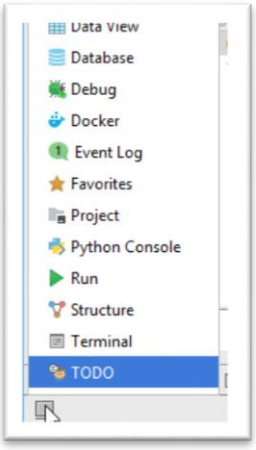


Figura 1.24. Lista de herramientas disponibles

Fuente: (Elaborado por el autor)

8 Personalización del espacio de trabajo de PyCharm

El IDE **PyCharm** otorga la libertad de personalizar el entorno de trabajo a gusto y comodidad de cada usuario, para ello brinda diferentes presentaciones en temas para la ventana del entorno de trabajo (Archivo | Configuración | Apariencia), permite también elegir el color de fondo con el que se desea trabajar en el Editor (Archivo | Configuración | Editor), adicionalmente permite seleccionar el tipo de letra con el que se quiere aparezca el código de programación (Archivo | Configuración | Editor | Estilo de código), este tipo de letra puede ser definido para cada tipo de lenguaje ya sea **Python**, CSS, HTML, entre otros.

9 Compleción de código

Es una interesante forma de ahorrar tiempo en escribir código **Python**, de manera inteligente **PyCharm** completa el comando que se esté escribiendo en código **Python**, en base a las funciones existentes. La compleción inteligente de código analiza el contexto en el que el usuario se encuentra trabajando y ofrece sugerencias que concurran alrededor del análisis efectuado, la Figura 1.25 muestra el funcionamiento de la compleción de código.

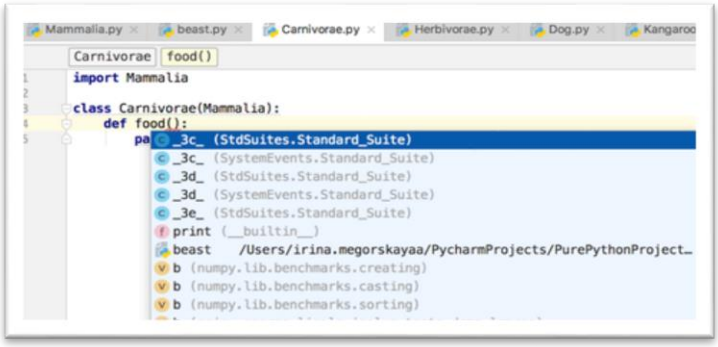


Figura 1.25. Compleción inteligente de código

Fuente: (Elaborado por el autor)

10 Acciones de Intuición del código escrito

PyCharm mantiene un monitoreo inteligente en el código que el usuario se encuentra escribiendo, de esta forma ejecuta sugerencias inteligentes para hacer compleciones de código, estas sugerencias son conocidas como Acciones de Intuición del código escrito y permiten al programador ahorrar tiempo en la generación del código. Las acciones de intuición permiten al programador aplicar cambios de manera correcta en el código y de manera automática, contrastando alguna inspección de código que se haya efectuado de manera autómata y en la que **PyCharm** haya encontrado código que seguramente estaba incorrecto. En caso que el programador olvide algún parámetro o campo de inicialización ya sea de una función o de alguna acción de flujo, al presionar en el ícono de la bombilla ó presionando el atajo Alt + Enter y seleccionando las opciones sugeridas, como ejemplo refiérase a la Figura 1.26.

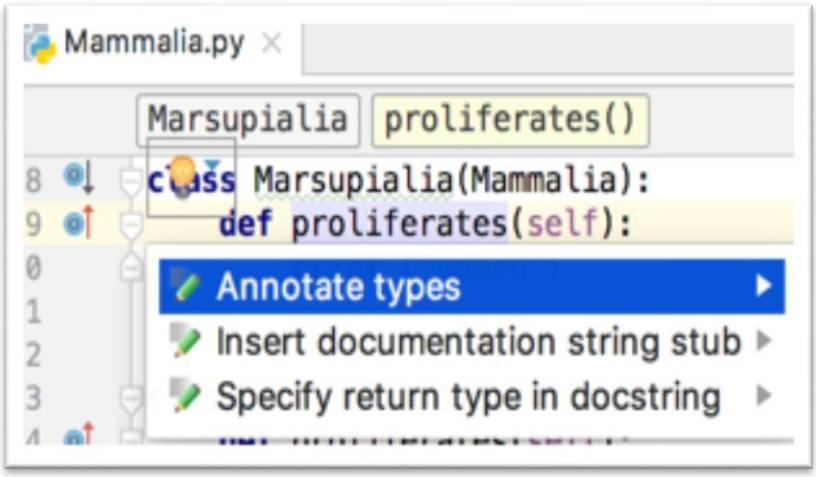


Figura 1.26. Opciones de sugerencia al usar atajo Alt+Enter en el Editor

Fuente: (Elaborado por el autor)

PyCharm ayuda a mantener el código libre de errores incluso antes de llegar a el estado de compilación y comprobación gracias a los indicadores de línea y auto compleción de código.

11 Ejecutar, compilar y comprobar

Cuando se tiene un código desarrollado en lenguaje **Python** y habiendo determinado gracias a los indicadores de línea que el contenido propuesto es correcto, se debe proceder al paso de comprobación y ejecución. Para ello la vía más sencilla que se tiene en **PyCharm** para ejecutar una aplicación es hacer clic derecho encima del Editor y seleccionar la opción Run <nombre>, donde "nombre" representa la denominación que se le haya dado al programa desarrollado, la Figura 1.27 muestra cómo se debe ejecutar un programa.

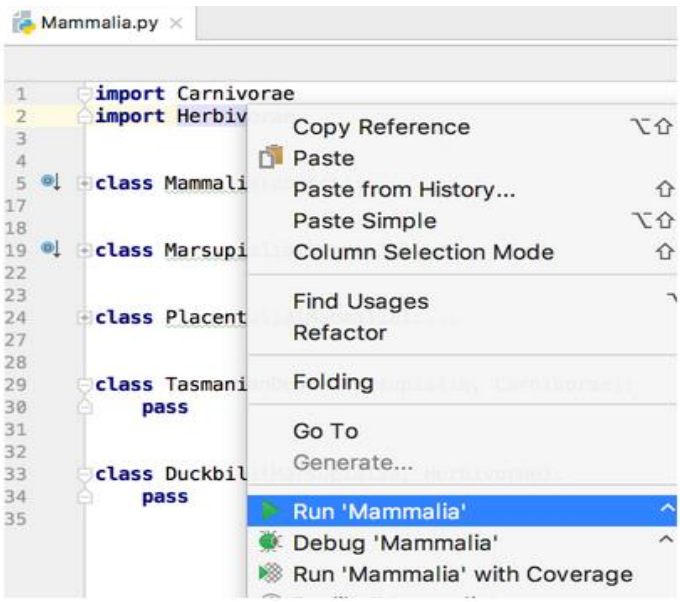


Figura 1.27. Ejecutar código del Editor

Fuente: (Elaborado por el autor)

Por otro lado la compilación de una aplicación empieza colocando puntos de interrupción de código en la ejecución del programa, se puede explorar la data del código para encontrar el error. Para depurar el código se puede presionar Shift+F9 y con esto manualmente se pueden ir colocando las variables o componentes faltantes en el código que impiden que este se ejecute con normalidad y siga el flujo de aplicación y resultado esperado la Figura 1.28 muestra como un punto de interrupción muestra donde se encuentra el error de un programa que no se ejecutó con éxito.

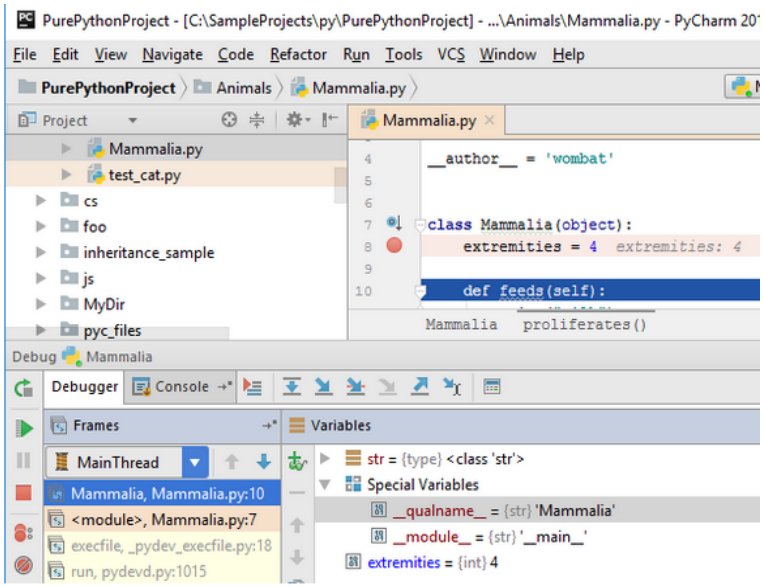


Figura 1.28. Punto de interrupción de código de una línea con código erróneo

Fuente: (Elaborado por el autor)

CAPITULO II

PROPUESTA

1. Introducción

Para alcanzar el desarrollo de un algoritmo que pueda interpretar la clasificación de los sistemas discretos es necesario utilizar simbología convencional del Tratamiento Digital de Señales, en ese caso el código fuente debe ser capaz de interpretar sistemas regidos bajo ecuaciones o variables dependientes e independientes, para ello se hará uso de las diferentes librerías que *Python* posee.

Para instalar librerías en *Python*, se debe abrir la consola mediante el menú de inicio y escribir Anaconda, como se muestra en las Figuras 2.1 y 2.2

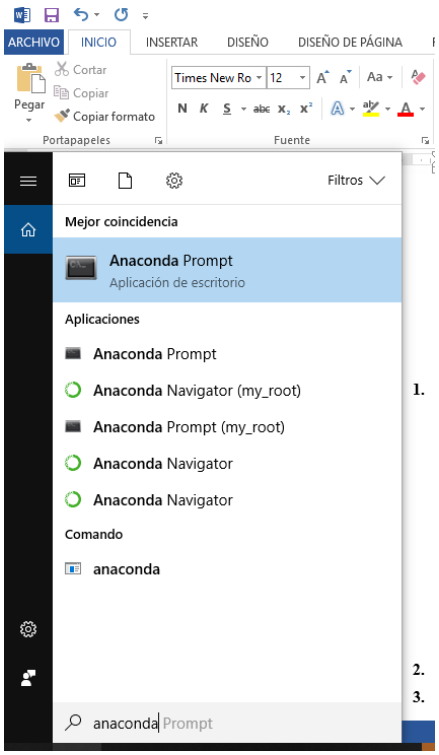


Figura 2.1. Abriendo Anaconda desde windows 10

Fuente: (Elaborado por el autor)

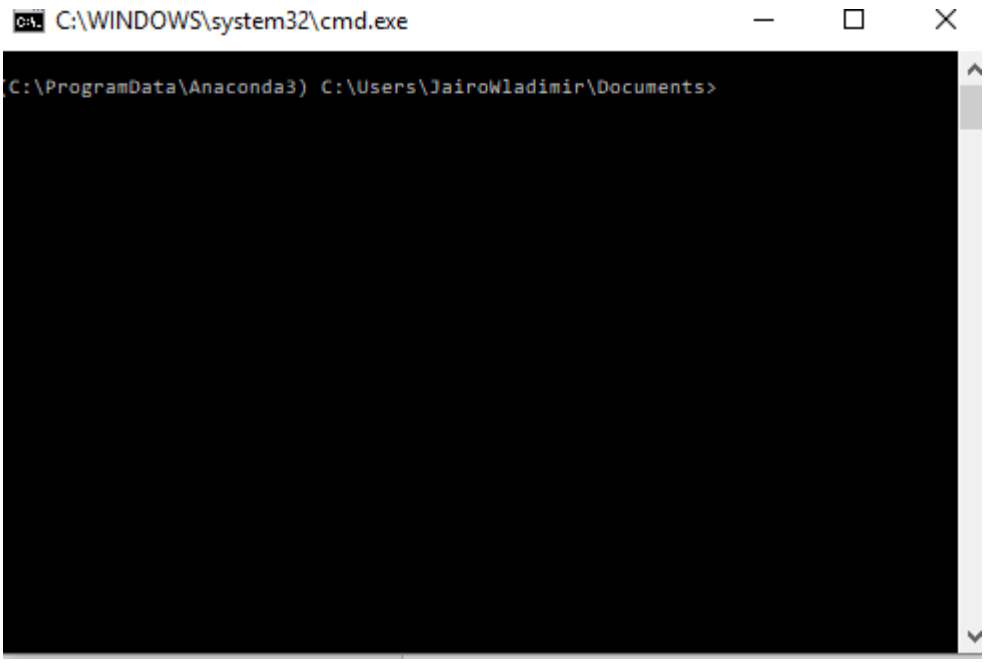


Figura 2.2. Consola interactiva Anaconda interprete de *Python*

Fuente: (Elaborado por el autor)

Para instalar cualquier librería en *Python* utilizamos el comando **pip install + nombre del paquete**

Para los fines pertinentes de este proyecto en la consola de *Python* las librerías se utilizaron los siguientes comandos en la consola del intérprete de pytjon Anaconda.

- pip install scipy
- pip install i*Python*
- pip install matplotlib
- pip install sympy
- pip install pyqt5

Para generar una interfaz gráfica de usuario que permita la interacción entre máquina y estudiante se utiliza el módulo pyqt5, el cual instala un paquete que genera interfaces gráficas. En este caso se creará una interfaz gráfica o aplicación con 3 ventadas cada una de ellas tendrá el soporte de las 3 prácticas que son objeto del desarrollo de este proyecto.

Nuevamente en el menú de inicio de Windows escribimos en buscar, esta vez la palabra clave es “designer”, refiérase a la Figura 2.3.

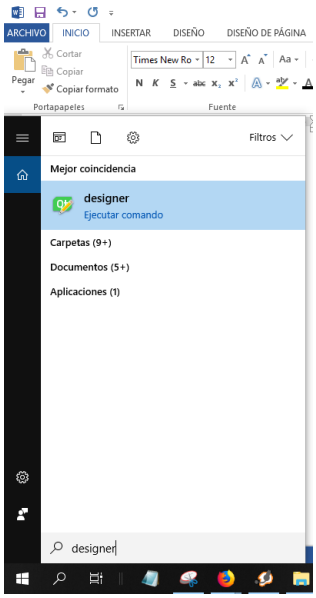


Figura 2.3. Abriendo QtDesigner

Fuente: (Elaborado por el autor)

Al abrir QtDesigner se puede elegir la apariencia que se desee para nuestra interfaz grafica de usuario, eligiendo botones, checkboxes, y editores de texto, debido a que el fin de este proyecto es mostrar las aplicaciones de las librerías útiles para el Tratamiento de Señales, no se profundizará en la descripción de QtDesigner ya que el lector deberá entender antes conceptos de programación de objetos, tema que supera los objetivos del proyecto actual, sin embargo se puede referir hacia el anexo 4, el cual contiene el manual técnico y da una breve descripción de cómo llegar a la consecución de la interfaz gráfica que se obtuvo.

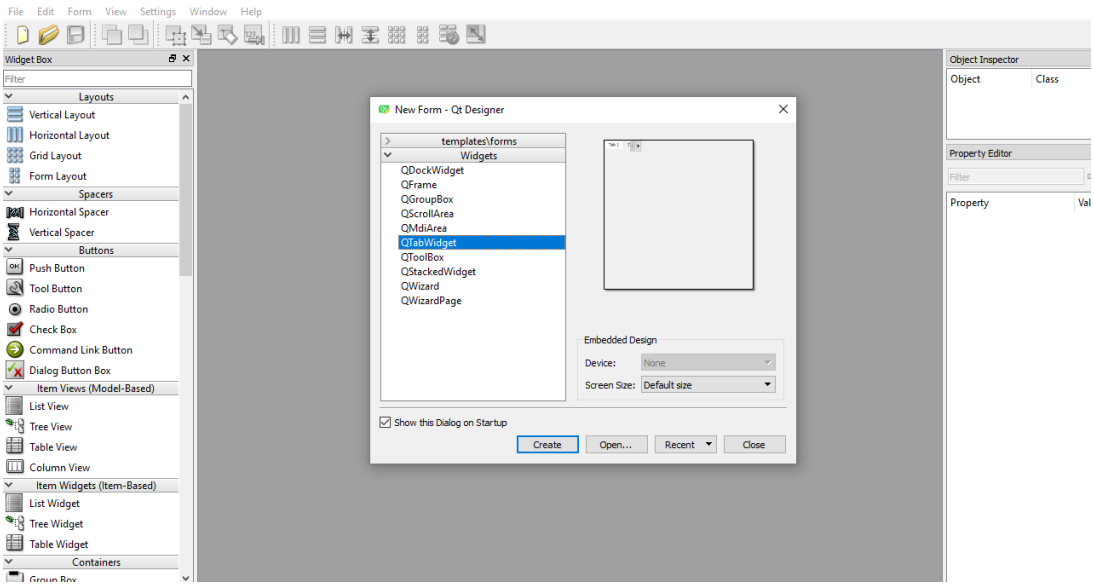


Figura 2.4. Vista de QtDesigner5

Fuente: (Elaborado por el autor)

Diseñar la aplicación mediante QtDesigner es intuitivo y relativamente fácil, lo primero es escoger el tipo de aplicación que se desea crear, para el caso de este proyecto se eligió un MainWindow, o pantalla principal, el cual abarcará mediante 3 pestañas, las 3 diferentes aplicaciones del proyecto, esto para evitar realizar 3 ventanas diferentes que aparentarían ser aplicaciones independientes la Figura 2.5, muestra las opciones que se despliegan para elegir la apariencia del programa.

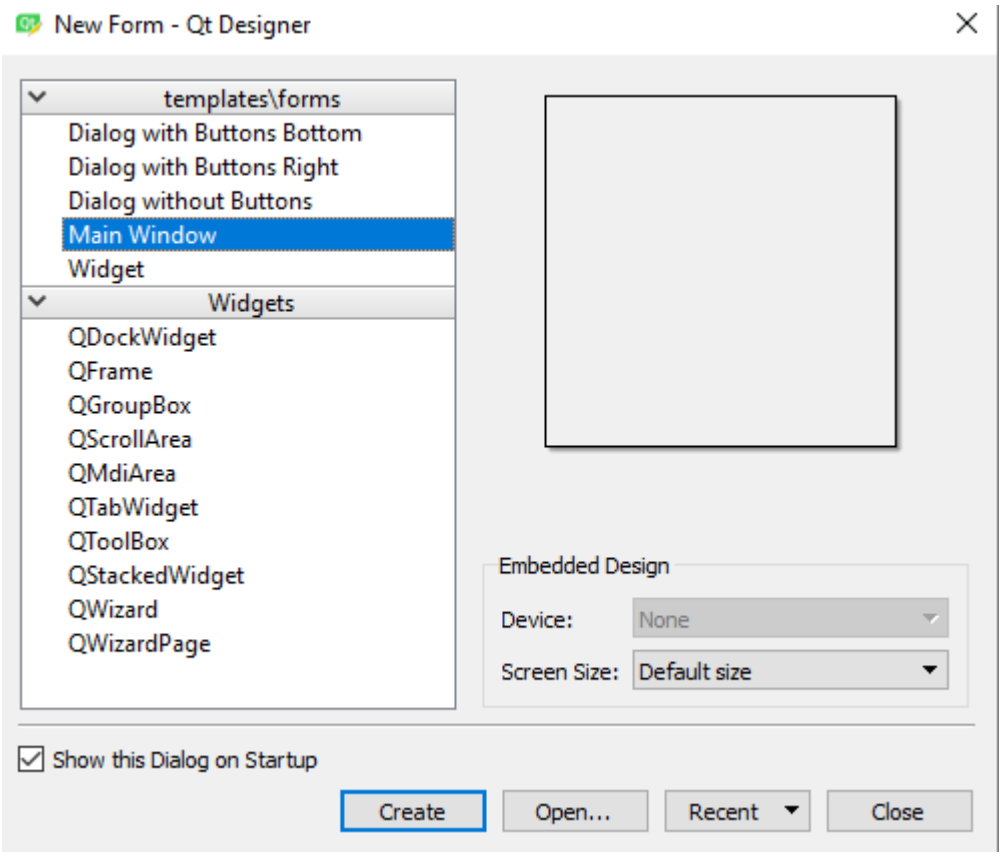


Figura 2.5. Escogiendo el tipo de aplicación en QtDesigner

Fuente: (Elaborado por el autor)

Para que nuestra ventana contenga más pestañas se utiliza el ítem TabWidget, el cual se encuentra en el lado izquierdo del QtDesigner, simplemente se lo selecciona y arrastra hacia el área de diseño de la aplicación que se está formando la Figura 2.6 representa la apariencia que irá tomando la aplicación al arrastrar el TabWidget.

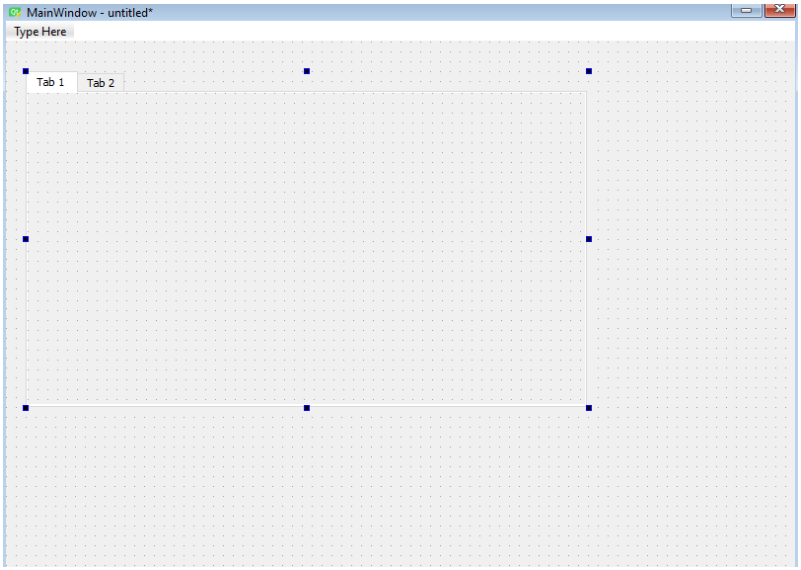


Figura 2.6. Adhiriendo TabWidget

Fuente: (Elaborado por el autor)

Por defecto el TabWidget tiene únicamente 2 pestañas, para tener las adicionales que se requieran basta con dar clic derecho sobre una de las pestañas ya creadas y dar clic en Insert Page, ejemplo de ello es la Figura 2.7

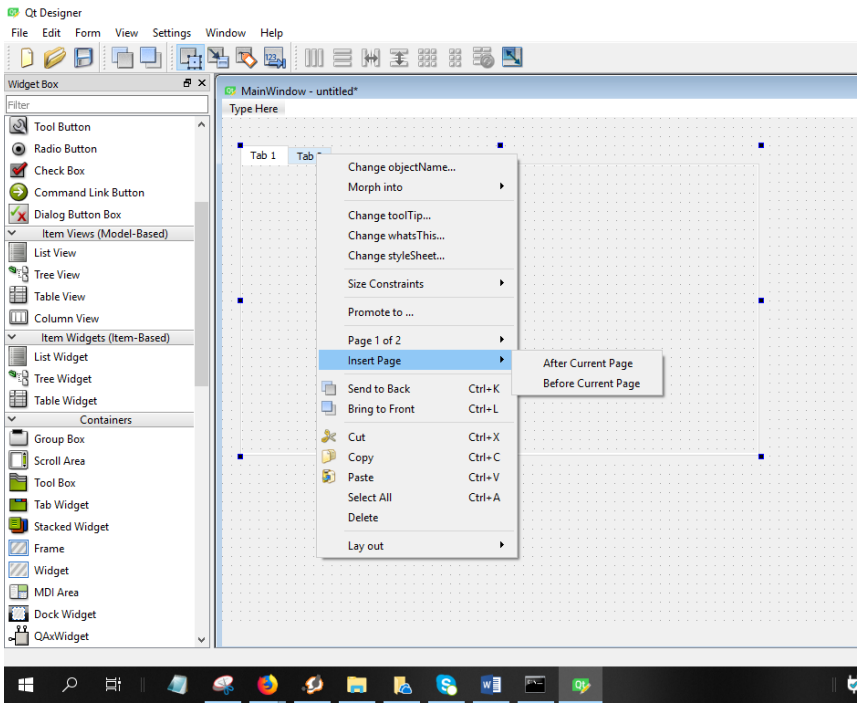


Figura 2.7. Adhiriendo nueva pestaña en TabWidget

Fuente: (Elaborado por el autor)

A criterio del programador estará la decisión de colocar antes o después de las pestañas existentes la nueva pestaña insertada.

Para la entrada de usuario se necesita captar texto, para que el programa convierta en variables y pueda calcular según los algoritmos creados la función o respuesta esperada con los datos que el usuario proporcione, para ello se escogen los TextEditor, estos serán los campos para ingreso de datos de usuario y también para visualización de los datos calculados o resultados esperados. La forma de escoger TextEditor tBoxes es idéntica y es repetitiva para cada uno de los Widgets que el programador desee usar.

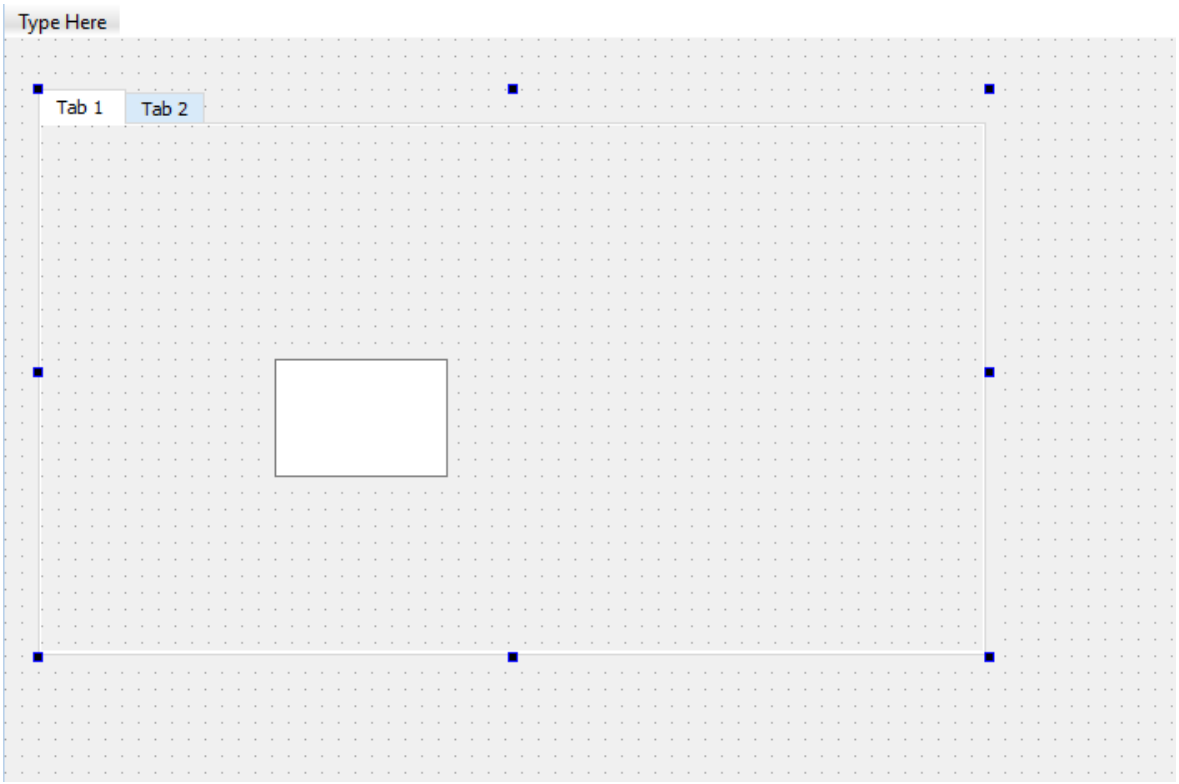


Figura 2.8. Colocando TextEditor

Fuente: (Elaborado por el autor)

Para insertar una imagen de fondo en el área de trabajo se requiere dar click el lado derecho del QtDesigner, en la sección que viene marcada como Property Editor, allí se debe colocar la abreviatura sty de style.

Al abrirse la sección de style, falta dar clic en los puntos suspensivos para desplegar la ventana emergente que sirve permite agregar una imagen para personalizar la ventana principal o MainWindow, el recurso debe estar guardado en formato .qrc, para poder ser añadido, la figura 2.9 muestra el cuadro de dialogo para agregar una imagen en formato .qrc.

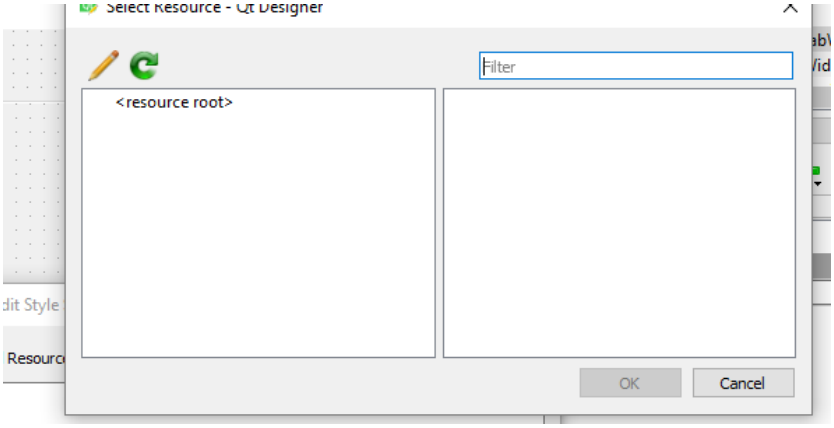


Figura 2.9. Cuadro de dialogo desplegado para insertar una imagen de fondo

Fuente: Elaborado por el autor

2. Programación y Diagramas de flujo

Una vez que se ha utilizado la herramienta QtDesigner para elaborar el diseño o presentación que tendrá la aplicación lo siguiente es importar las librerías con las que se va a trabajar para los fines pertinentes los cuales implican clasificación de los sistemas discretos, convolución de señales y diseño de filtros.

Las librerías que se utilizan para generar los algoritmos de clasificación de sistemas discretos son sympy y math.

La librería de sympy permite la interpretación de las excitaciones de entrada que escribirá el usuario mediante el GUI empleado, los sistemas se rigen bajo ecuaciones o funciones personalizadas x(n), las cuales serán interpretadas como la entrada del sistema.

La librería math, contiene funciones matemáticas tradicionales que utilizan los sistemas y las señales, funciones exponenciales, logarítmicas, trigonométricas se encuentran alojadas dentro de esta librería por lo que una función del tipo sen(x(n)), es factible de ser analizada siempre que se utilicen los algoritmos.

Para clasificar un sistema mediante la propiedad de variabilidad en el tiempo el diagrama de flujo sería el expresado en la Figura 2.10.

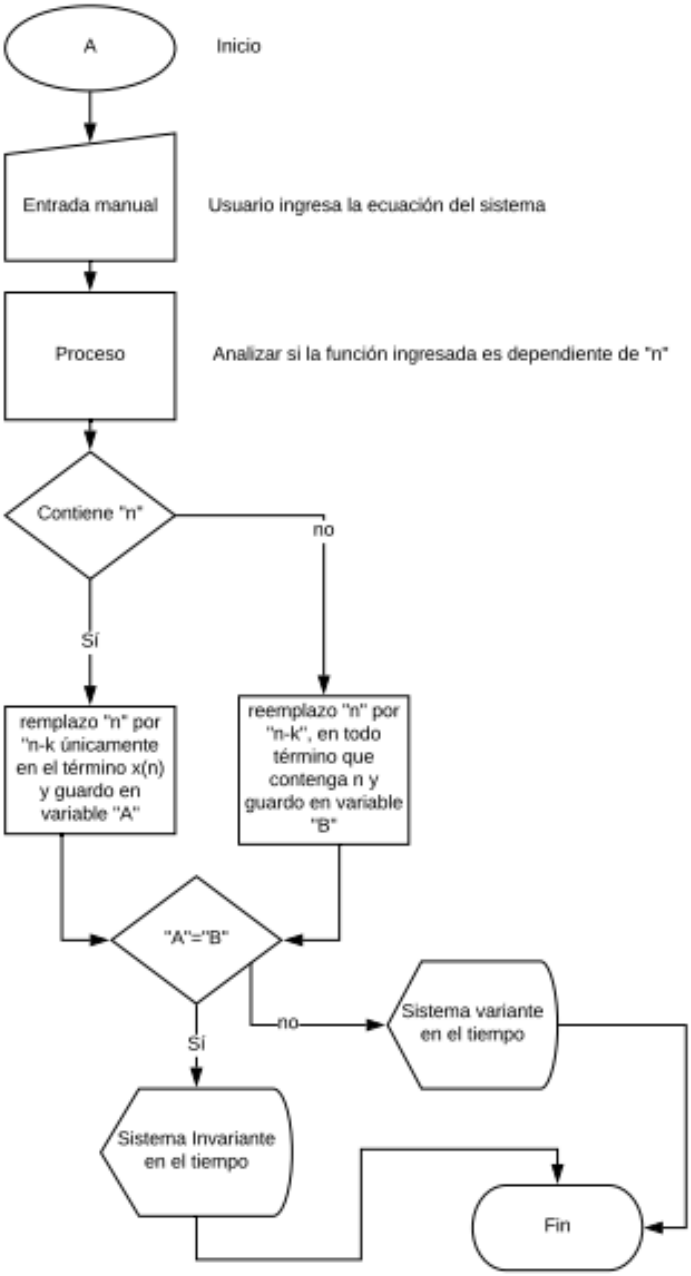


Figura 2.10. Diagrama de flujo para la clasificación de un sistema discreto invariante o variante en el tiempo
Fuente: (Elaborado por el autor)

Para conseguir el algoritmo que interprete la linealidad de un sistema ingresado en formato de ecuación por entrada del usuario, se requiere diferenciar x(n) de x1(n) y x2(n), tomando en cuenta que la excitación de entrada de los 2 sistemas x1(n) y x2(n) son representación del sistema ingresado por el usuario, únicamente se realiza el reemplazo para identificar el algoritmo que realizará el cálculo de la linealidad, el diagrama de flujo que acercara al fin esperado la Figura 2.11 ilustra lo mencionado.

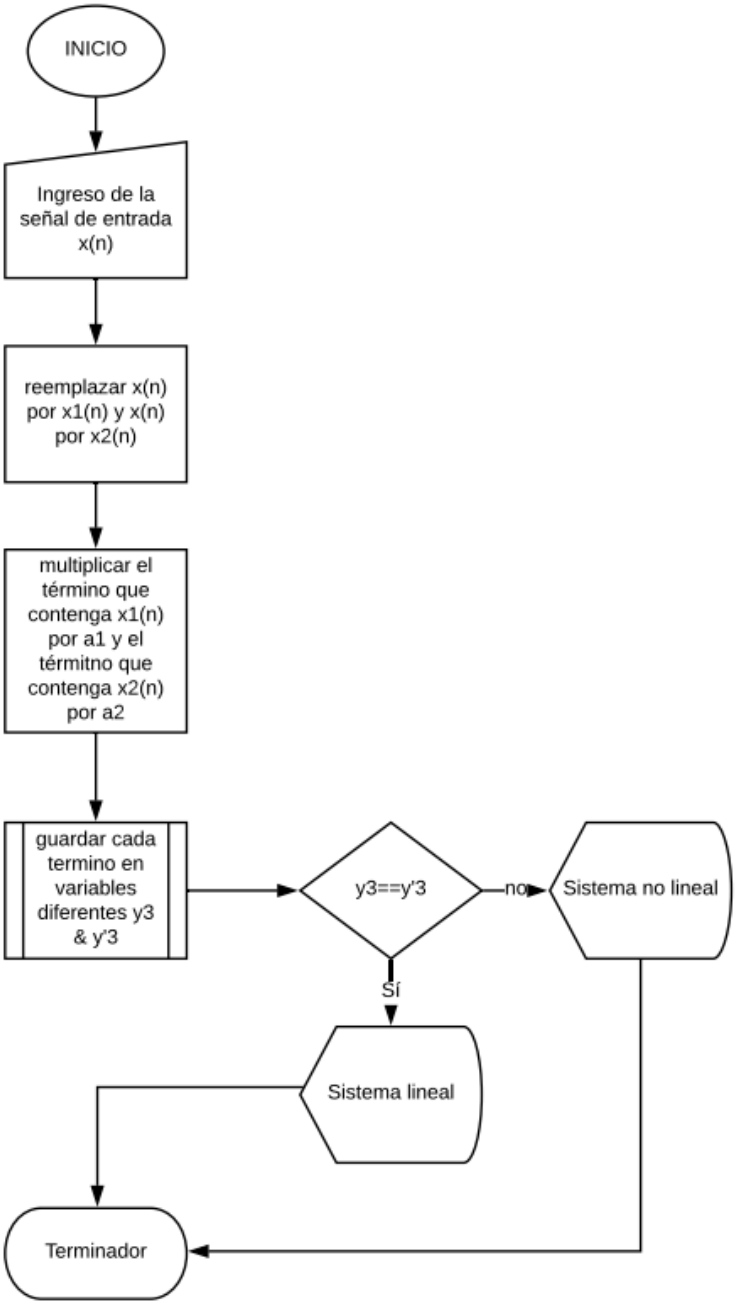


Figura 2.11. Diagrama de flujo para identificar la linealidad de un sistema

Fuente: Elaborado por el autor

Otra de las propiedades que requieren generar un algoritmo de cálculo es la de estabilidad de un sistema, para lo cual se requiere reemplazar la señal de excitación de entrada desconocida en respuesta en el tiempo, por una señal clásica conocida, es decir, se reemplaza por una función de la cual se conoce su comportamiento a lo largo del tiempo como por ejemplo la señal de impulso unitario, de la cual se conoce que su valor de amplitud es 1 para valores en los que n es mayor o igual que 0.

La Figura 2.12, ilustra el diagrama de flujo que permite llegar al resultado descrito.

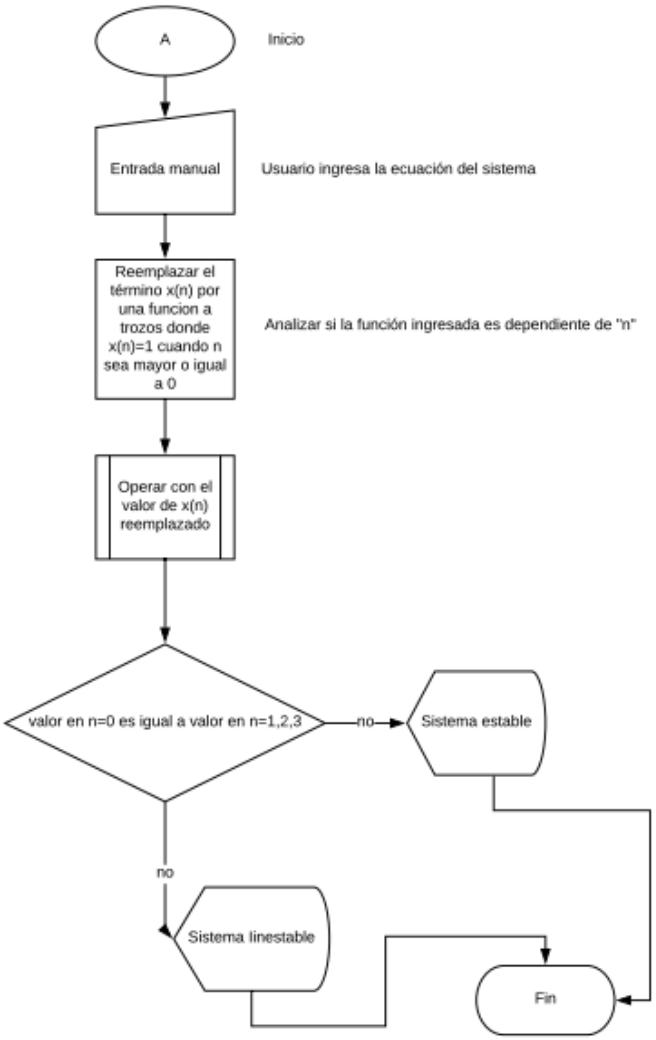


Figura 2.12. Diagrama de flujo de la estabilidad de un sistema

Fuente: (Elaborado por el autor)

Para lograr la convolución de dos señales, hay que tomar en cuenta que estas se guardan en vectores, es decir tienen diferentes módulos a lo largo del eje temporal, se debe definir el tamaño que tendrá el vector, para los fines del proyecto, un vector que almacene 9 números será suficiente con ello se podrían tener valores en el tiempo desde 0 a 8 o de -4 a 4 en caso de que sea el requerimiento.

El algoritmo diseñado para representar la convolución de dos señales en el tiempo, donde cada una de ellas almacena 9 números con amplitudes diferentes en el eje temporal, se representa mediante la Figura 2.13.

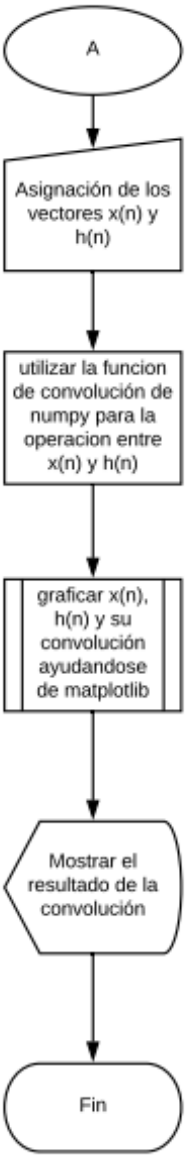


Figura 2.13. Diagrama de flujo de la convolución de señales representadas en vectores en el tiempo

Fuente: Elaborado por el autor

Por último para el diseño de filtro, se debe tener en cuenta que tipo de filtro se va a escoger es decir la elección debe ir en base a la respuesta de frecuencia que se espera obtener para el análisis, se verá si el filtro es del tipo FIR o IIR. Luego de ello se solicitara la clasificación del filtro ya que se requiere determinar si es pasa bajos, pasabanda, o pasa altos, como el objetivo de este proyecto está enfocado específicamente a filtros pasa bajo, no será necesario incluir en el diagrama de flujo la condición de decisión para que se escoja entre las 3 clases de filtros antes mencionados.

La condición de decisión en la elección de respuesta FIR e IIR es esencial ya que solo de esa manera se puede evidenciar al diferencia en respuestas de frecuencia de ambos tipos de

filtro, una vez escogido el tipo de respuesta adicionalmente se debe considerar el método de cálculo que utiliza el filtro, sea por ventas Butterwhort o Chyevyshev tipo 2. Para simplificar y no sobrepasar los objetivos del proyecto se escogen los métodos Chevishev tipo 2 y Butterworth para el diseño de filtros IIR, mientras que el método de ventanas será el tomado para el diseño de filtros FIR. La metodología utilizada se muestra en el diagrama de flujo de la Figura 2.14.

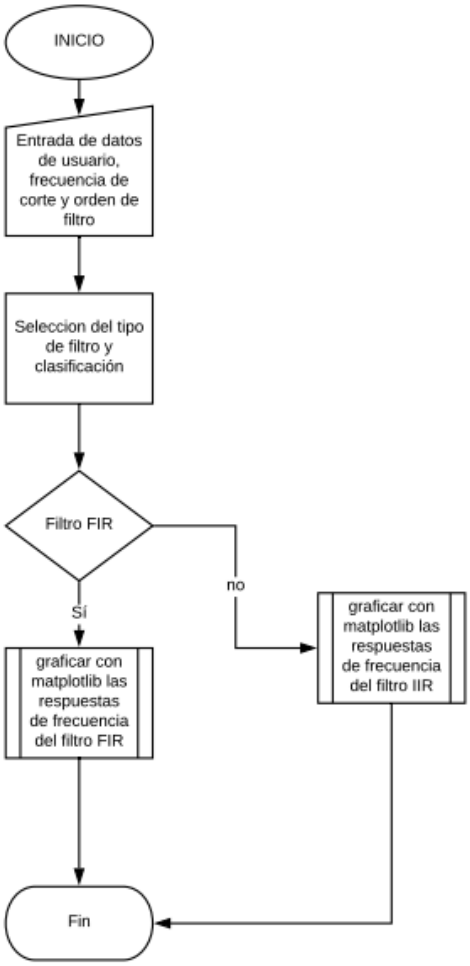


Figura 2.14. Diagrama de flujo para la respuesta de filtros FIR e IIR

Fuente: (Elaborado por el autor)

CAPÍTULO III

IMPLEMENTACIÓN

Este capítulo contiene una sugerencia de prácticas de laboratorio que ayudarán a complementar la teoría dictada en la materia de Tratamiento Digital de Señales de la Universidad Tecnológica Israel para ello se hace uso del *software* libre **Python**, con ello se permitirá al estudiante tener una mejor comprensión de la teoría impartida.

3.1 Sílabo genérico de la asignatura

Los temas específicos del syllabus de la asignatura Tratamiento Digital de Señales, donde se pretende hacer uso de Python

1. **Señales y sistemas discretos en el tiempo**
2. **Sistemas LTI discretos: la suma de la convolución**
3. **Filtros Digitales**

El syllabus completo de la materia está referenciado en la sección de Anexos.

3.2 Prácticas recomendadas

Para la correcta instalación del IDE Pycharm y evitar cualquier inconveniente los requerimientos de sistema básicos con:

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- 2 GB RAM mínimo.
- 4 GB RAM recomendado.
- 1024x768 resoluciones de pantalla mínima recomendada.
- *Python* 2.4 o superior, Jython, PyPy o Iron*Python*.

Se desarrolla a continuación un pequeño manual de prácticas que se realizaron durante la investigación presentada, recomendando que sean incluidas en el programa de estudi de la materia Tratamiento Digital de Señales de la UISRAEL, el formato sugerido presenta prácticas mediante simulaciones en el *software* **Python**, donde cada simulación contiene:

- Trabajo preparatorio correspondiente
- Materiales necesarios para la práctica

- Planteamiento de los objetivos de la práctica
- Desarrollo de la práctica
- Preguntas y conclusiones

La parte que contiene al trabajo preparatorio, se hace necesaria ya que usando este método el estudiante podrá ir con previo conocimiento acerca de la manipulación de una función o comando relacionado a **Python**, mismo que se usará para resolver el problema planteado.

Con la especificación de los materiales requeridos para cada una de las prácticas, se prepara el escenario para que no existan inconvenientes en realizarlas, y con ello obtener el resultado esperado sin contratiempo alguno.

El desarrollo de la práctica o simulación, es necesaria para evaluar el conocimiento adquirido durante la clase de Tratamiento Digital de Señales, en la Universidad Tecnológica Israel.

Después de la ejecución de cada simulación el manual sugerido contiene varias preguntas generales para que sean planteadas a los estudiantes y puedan anotar sus propias conclusiones acerca de los resultados obtenidos.

3.2.1 Desarrollo experimental 1:

Generar mediante código Python un algoritmo que permita identificar la clasificación de un sistema discreto en el tiempo, es decir si este es: estático, dinámico, variante, invariante, lineal, no lineal, causal, no causal, estable o inestable.

Trabajo Preparatorio

- Consultar en que consiste la clasificación de los sistemas discretos en el tiempo.
- Consultar como declarar y operar con variables tipo String en **Python** y que librería contiene los comandos que soportan este tipo de variables.
- Consultar con que función se puede reemplazar un elemento de un String con otro deseado.

- Consultar como declarar variables simbólicas en **Python** y que librería contiene los comandos que soportan este tipo de variables**.**
- Consultar cómo funciona el condicional **if-else** en **Python.**

Materiales:

- Computadora con *Python* instalado versiones 2.4 o superior
- Librería Sympy instalada
- PyQt5 instalado
- Librería I*Python* instalado

Objetivos:

- Definir el método de acceso y utilización de las librerías científicas de **Python.**
- Identificar variables simbólicas y como trabajar con ellas en programación científica.
- Desarrollar código **Python** simple que ayude a identificar de manera rápida la clasificación de un sistema discreto ingresado por medio de la consola interactiva del IDE **PyCharm**.

Desarrollo

Las señales son procesadas dentro de sistemas discretos, es importante conocer la clasificación de los sistemas discretos ya que con la clasificación de los mismos se pueden obviar pasos de cálculo y simplificar algoritmos.

Un sistema discreto puede ser:

- Variante o invariante en el tiempo
- Lineal o no lineal
- Estable o inestable

La entrada de un sistema o señal de excitación es conocida como $x(n)$ mientras la señal que se genera a la salida en respuesta a la excitación presentada en la entrada es conocida como $y(n)$ a continuación se evaluaran dos sistemas para conocer sus propiedades.

1er Sistema a analizar:

$$\boldsymbol{y(n) = Cos[x(n)]}$$

1. El sistema es estático, ya que de acuerdo a la definición de un sistema estático, la respuesta a la salida del sistema y(n) solamente depende de valores actuales de la excitación de entrada x(n).

2. **Linealidad**:

$$\boldsymbol{y1(n) = Cos[x1(n)]}$$
$$\boldsymbol{y2(n) = Cos[x2(n)]}$$
$$\boldsymbol{y3(n) = a1 * Cos[x1(n)] + a2 * Cos[x2(n)]}$$
$$\boldsymbol{y'3(n) = a1 * y1(n) + a2 * y2(n)]}$$
$$\boldsymbol{y'3(n) = a1 * Cos[x1(n)] + a2 * Cos[x2(n)]}$$
$$\boldsymbol{y3(n) = y'3(n)}$$

Es un sistema lineal ya que: y3(n)=y'3(n)

3. $\boldsymbol{y(n,k) = Cos[x(n-k)]}$

$\boldsymbol{y(n-k) = Cos[x(n-k)]}$
$\boldsymbol{y(n,k) = y(n-k)}$

Al haber obtenido: $\boldsymbol{y(n,k) = y(n-k)}$

Es un sistema invariante en el tiempo

4. El sistema es estático ya que únicamente depende de valores actuales en la entrada $\boldsymbol{x(n)}$

5. Al haber concluido que el sistema es **Estático** por definición también es un sistema **Causal**

6. Por simple deducción se puede observar que la entrada del sistema $\boldsymbol{x(n)}$ al estar dentro de la función **coseno** estará limitada a un solo valor que como máximo será 1, es decir la entrada está acotada por tanto genera una salida acotada lo que repercute en que el sistema cumpla con la propiedad de **Estabilidad**.

RESULTADO DE LA SIMULACIÓN

Al correr la simulación del programa desarrollado obtendremos los mismos resultados que en el análisis realizado en papel anteriormente para el sistema que rige bajo la ecuación:

$$\boldsymbol{y(n) = Cos[x(n)]}$$

Figura 3. 1 Resultados de la simulación, prueba de propiedades de los sistemas discretos

Fuente: (Elaborado por el autor)

2do Sistema a analizar:

$$\boldsymbol{y(n) = n * x(n)}$$

1. El sistema es estático, ya que de acuerdo a la definición de un sistema estático, la respuesta a la salida del sistema y(n) solo depende de valores actuales de la señal de excitación en la entrada del sistema x(n).

2. **Linealidad**:

$$\boldsymbol{y1(n) = n * x1(n)}$$
$$\boldsymbol{y2(n) = n * x2(n)}$$
$$\boldsymbol{y3(n) = a1 * n * a * x1(n) + a2 * n * x2(n)}$$
$$\boldsymbol{y'3(n) = a1 * y1(n) + a2 * y2(n)}$$
$$\boldsymbol{y'3(n) = a1 * n * x1(n) + a2 * n * x2(n)}$$
$$\boldsymbol{y3(n) = y'3(n)}$$

Es un sistema lineal ya que: y3(n)=y'3(n)

3. $\boldsymbol{y(n,k) = n * x(n-k)}$

$$\boldsymbol{y(n-k) = (n-k) * x(n-k)]}$$
$$\boldsymbol{y(n,k) = y(n-k)}$$

Al haber obtenido: $\boldsymbol{y(n,k) \neq y(n-k)}$

Es un sistema variante en el tiempo

4. El sistema es estático ya que únicamente depende de valores actuales en la entrada $x(n)$
5. Al haber concluido que el sistema es **Estático** por definición también es un sistema **Causal**
6. Por simple deducción se puede observar que la entrada del sistema $x(n)$ al ser reemplazada a lo largo del tiempo con valores de n, va a ir creciendo en el tiempo, por tanto su salida no estará limitada a un solo valor con esto el sistema no cumple con la propiedad de **Estabilidad**.

RESULTADO DE LA SIMULACIÓN

Al correr la simulación del programa desarrollado obtendremos los mismos resultados que en el análisis realizado en papel anteriormente para el sistema que rige bajo la ecuación:

$$y(n) = n * [x(n)]$$

Figura 3. 2. Resultado de la simulación, prueba de los sistemas discretos

Fuente: Elaborado por el autor

Parte del código utilizado para lograr la simulación de la práctica 1, es mostrada en los párrafos siguientes:

```
from py_expression_eval import Parser
from pybigparser import *
from sympy import *
import sys
import numpy as np
import matplotlib.pyplot as plt
from math import e
w,A,B=symbols('w A B');
n = 'n'; y = symbols('y'); k = 'k'; x=Symbol('x');a1=Symbol('a1');a2=Symbol('a2');
fnc = (input('ingrese función:    '));
if "x(n)" in fnc and "x(n-1)" in fnc: #Modificar expresion "x(n-1)" según necesidad
   fnc0 = fnc.replace("n" ,"n-k")
   expr=sympify(fnc0)
   print ("y(n,k):  ")
   pprint(expr)
```

El código completo se encuentra incluido en la sección de anexos.

Preguntas
1. ¿En qué formato expresaría una función con exponente cuadrático?
2. ¿Qué resultado se obtendría al ingresar un valor numérico como función de entrada? Explique.
3. ¿Es posible convertir un valor numérico en variable simbólica? Explique.

CONCLUSIONES

3.2.2 Desarrollo experimental 2:

Generar mediante código Python, un script que permita calcular y visualizar la convolución discreta de 2 señales conocidas, una de entrada como x(n) y una señal de excitación cualquiera h(n).

Trabajo Preparatorio

- Consultar en que consiste la convolución discreta entre dos señales.
- Consultar qué librería permite la generación de gráficas en **Python**.
- Consultar como declarar matrices en **Python.**
- Consultar la sintaxis y el cómo trabaja el bucle ***for*** en **Python**.
- Consultar como declarar funciones propias en **Python** para que estas devuelvan un cierto valor deseado.
- Consultar cómo colocar títulos, subtítulos y como manipular los ejes en las gráficas generadas en **Python**.
- Consultar la clase ***convolve*** de la librería ***Numpy.***
- Consultar la clase ***piecewise*** de la librería ***Numpy.***

Materiales:

- Computadora con *Python* instalado versiones 2.4 o superior
- Librería Scipy instalada
- PyQt5 instalado
- Librería Numpy instalada
- Librería I*Python* instalado

Objetivos:

- Familiarizar al estudiante con la generación de gráficas de funciones.
- Desarrollar código *Python* simple que ayude a obtener la convolución discreta de dos señales ingresando los datos por medio de la consola interactiva del IDE ***PyCharm***.
- Identificar como las librerías ***Numpy*** y ***Matplotlib*** se encuentran directamente asociadas en la operación y graficación de funciones.
- Conocer como declarar funciones matemáticas en **Python**.

Desarrollo

La convolución es una operación matemática entre señales definida por la Ecuación 5:

$$[y_k] = [u_k] * [t_k] = \sum_{k=-\infty}^{\infty} [t_{k-1}] * u_k, k = 0, 1, 2, \ldots . m + n - 2.$$

Ecuación 5 Ecuación suma de convolución

La interfaz de usuario solicitará los valores en el tiempo para cada punto de la función ***h(n,)*** que es la señal de excitación o de evaluación para conocer que respuesta al impulso presenta la señal de entrada ***x(n)***, se considera un rango de valores de 17 puntos, los cuales son suficientes para evaluar y conocer la respuesta a lo largo del tiempo que tendrá la señal.

Ejemplo:

Se desea conocer el resultado de convolucionar la señal de entrada:

$$x_{(n)} = \{\underset{\uparrow}{1},2,4\}$$

Con la señal impulso unitario:

$$h_{(n)} = \{\underset{\uparrow}{1},1,1,1,1\}$$

Paso 1:
Se invierte cualquiera de las dos señales reemplazando ***n*** por ***–k***, por facilidad se realiza el espejo en ***h(n)***

$$h_{(-k)} = \{1,1,1,1,\underset{\uparrow}{1}\}$$

$$h_{(-k)} = \{1,1,1,1,\underset{\uparrow}{1}\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad => Y_0 = 1$$

$$h_{(1-k)} = \{1,1,1,\underset{\uparrow}{1},1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad => Y_1 = 3$$

$$h_{(2-k)} = \{1,1,\underset{\uparrow}{1},1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad => Y_2 = 7$$

$$h_{(-k)} = \{1,\underset{\uparrow}{1},1,1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad => Y_3 = 7$$

$$h_{(1-k)} = \{\underset{\uparrow}{1},1,1,1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad => Y_4 = 7$$

$$h_{(2-k)} = \{\underset{\uparrow}{0},1,1,1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad \Rightarrow Y_5 = 6$$

$$h_{(-k)} = \{0,\underset{\uparrow}{0},1,1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad \Rightarrow Y_6 = 4$$

$$h_{(1-k)} = \{0,0,\underset{\uparrow}{0},1,1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad \Rightarrow Y_7 = 0$$

$$h_{(2-k)} = \{0,0,0,\underset{\uparrow}{0},1\} * x_{(n)} = \{\underset{\uparrow}{1},2,4\} \quad \Rightarrow Y_8 = 0$$

En consecuencia, el resultado de la convolución es:

$$y_{(n)} = \{1,3,7,7,7,6,4,0,0, \dots 0\}$$

RESULTADO DE LA SIMULACIÓN

Al correr la simulación se deberá obtener el mismo resultado mostrado anteriormente, además de ello por medio de Matplotlib, se generarán las gráficas de las señales de entrada, excitación y salida.

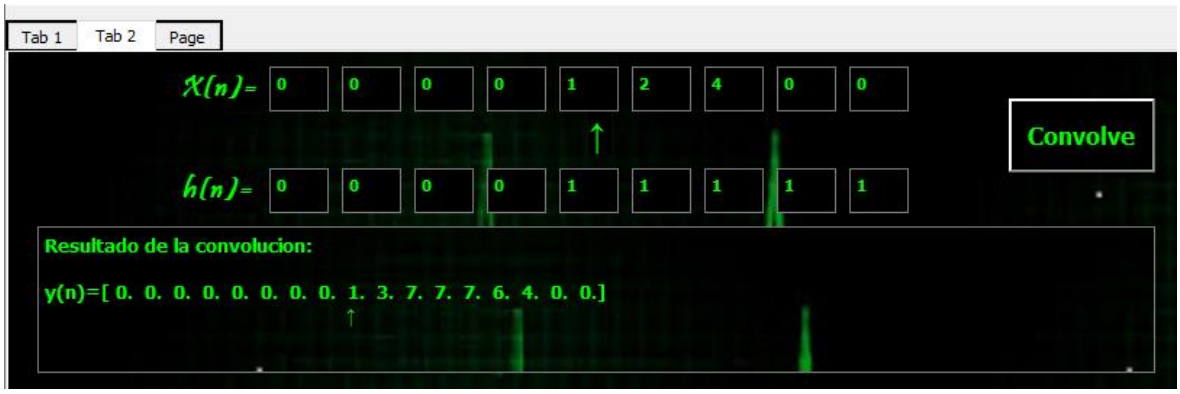


Figura 3. 3. Convolución entre dos señales dadas ingresadas por interfaz de usuario.

Fuente: (Elaborado por el autor)

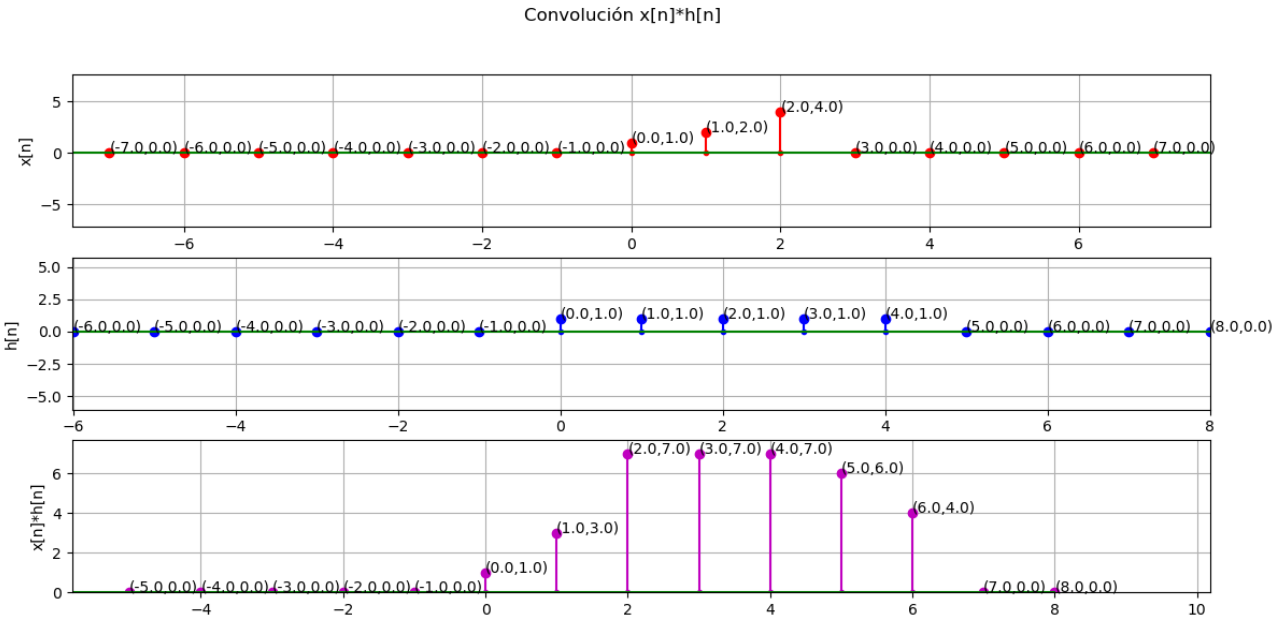


Figura 3. 4. Gráficas de las señales: x(n), h(n) y la convolución y(n)=x(n)*h(n)

Fuente: (Elaborado por el autor.)

Parte del código Python utilizado para obtener el resultado antes mostrado, está referenciado en las siguientes líneas:

```
import numpy as np
import matplotlib.pyplot as plt
global xn
global hn
# Definir la funcion para el ejemplo
def fn_x(n):
    # función matemática CAMBIAR AQUI
    xn=0
    if (n>=0 and n<=6):
        xn=np.piecewise(n,[n==0,n==1,n==3],[1,2,4],dtype=float)
        #xn=np.sin(n)
    # función matemática CAMBIAR AQUI
    return(xn)

def fn_h(n):
    # función matemática CAMBIAR AQUI
    hn=0
    if (n>=0 and n<=4):
        hn = 1
    # función matemática CAMBIAR AQUI
    return(hn)

# Programa para graficar la función
# INGRESO
```

```
print(' Rango para evaluar la convolución')
t0=int(input('valor inicial: '))
tf=int(input('valor final: '))
```

De igual forma el código completo está incluido en la sección de Anexos.

Preguntas
4. ¿Qué resultado se obtendría al ingresar los datos de *h(n)* en *x(n)* y viceversa y realizar la convolución. Explique.
5. ¿Qué resultado se obtendría al convolucionar dos entradas de impulso unitario? Explique.
6. ¿Es posible convolucionar valores expresados en variables simbólicas? Explique.

CONCLUSIONES

3.2.3 Desarrollo experimental 3:

Utilizando código Python, desarrollar un script que permita visualizar la respuesta en frecuencia de un filtro pasa bajo FIR e IIR, a partir de datos proporcionados por el usuario como: Frecuencia de Corte y el Orden del filtro.

TRABAJO PREPARATORIO

- Consultar que librerías contienen funciones que permitan realizar diseños de filtros en **Python**.
- Investigar la relación entre la frecuencia de corte y el orden del filtro.
- Encontrar la propiedad de la función ***signal*** en **Python,** además de la librería que contiene a esta función.
- Averiguar 3 argumentos de la función ***signal*** que puedan ser usados en el diseño de filtros.

Trabajo Práctico

Objetivos:

- Observar las diferentes respuestas en frecuencia de un filtro pasa bajos, al cambiar valores de frecuencia de corte y orden del filtro.
- Conocer las herramientas para diseño de filtro que **Python**.
- Diferenciar la respuesta de frecuencia entre los filtros IIR y FIR

Desarrollo:

Para conocer la respuesta en frecuencia de un filtro pasa bajo FIR, se solicitará mediante entrada de usuario los valores de orden del filtro, frecuencia de muestreo y la frecuencia de corte, se debe tener en cuenta que en caso de que los valores de frecuencia y orden no estén dentro de los rangos determinados, el programa desarrollado en *Python* devolverá un error y se cerrará la ventana por lo que los valores a experimentar en el diseño del filtro deben ser elegidos con criterio. En caso de que los valores de frecuencia de corte y orden del filtro sean correctos, se desplegara una gráfica a través de la librería MATPLOTLIB. Para verificar las respuestas de los filtros se utilizaran funciones existentes dentro de librerías de *Python*, lo cual permite para el caso de filtros IIR, partir de diseño y respuestas de filtros analógicos.

El diseño de filtro es simplemente la respuesta que tiene uno de ellos respecto a un rango de frecuencias y además de ello incluye el algoritmo que utiliza para el cálculo de la

respuesta, el tipo de filtro influye en la respuesta esperada, se permitirá al usuario escoger el tipo de filtro a través de la interfaz de usuario.

Ejemplo:

Se desea conocer la respuesta de un filtro FIR pasabajo a una frecuencia de 22050, con frecuencia de corte de 6000, comprobar con diferentes valores del orden de filtro para verificar cual causa menor rizado en la banda de paso y en las bandas atenuadas.

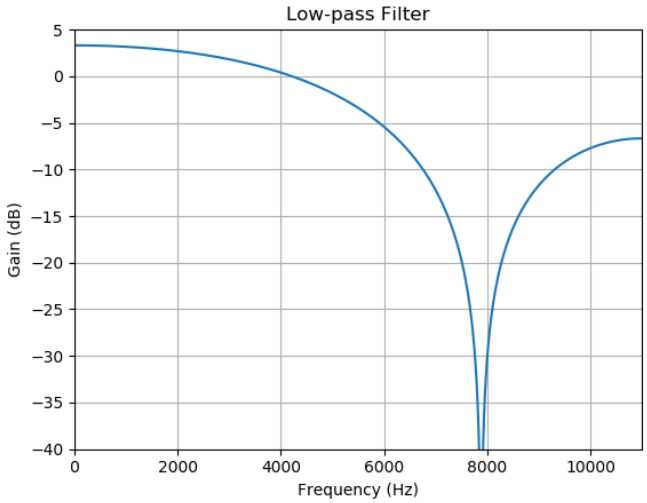


Figura 3. 5. Respuesta de un filtro FIR orden 5 y frecuencia de corte de 6000Hz

Fuente: (Elaborado por el autor.)

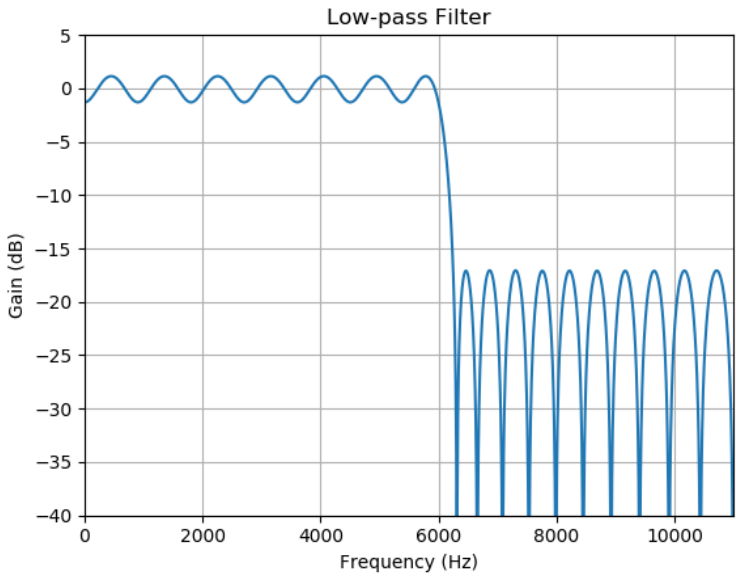


Figura 3. 6. Respuesta de un filtro FIR orden 50 y frecuencia de corte de 6000Hz

Fuente: (Elaborado por el autor)

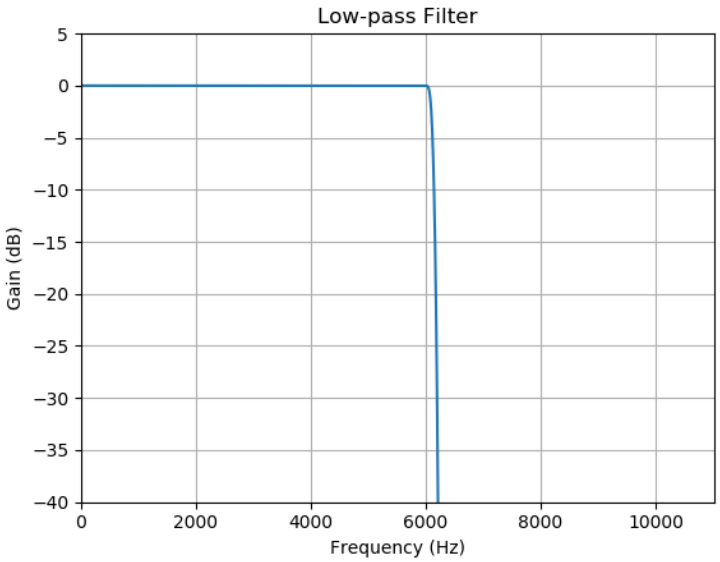


Figura 3. 7. Respuesta de un filtro FIR orden 500 y frecuencia de corte de 6000Hz

Fuente: Elaborado por el autor.

Ejemplo 2:

Se desea conocer la respuesta de un filtro IIR pasa bajo Chebyshev II a una frecuencia de, con frecuencia de corte de 6000, atenuación en la banda de corte de 300 comprobar con diferentes valores del orden de filtro para verificar cual causa menor rizado en la banda de paso y en las bandas atenuadas.

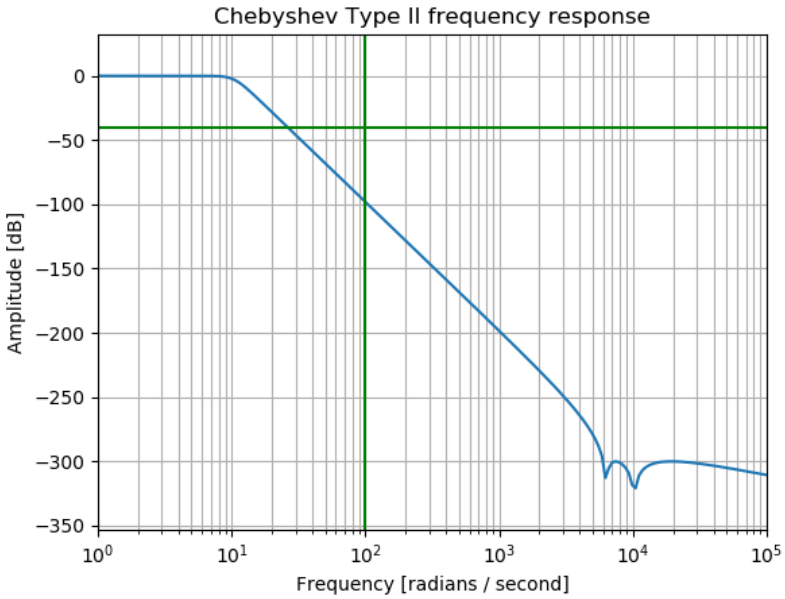


Figura 3. 8. Respuesta de un filtro IIR orden 5 y frecuencia de corte de 6000Hz

Fuente: Elaborado por el autor.

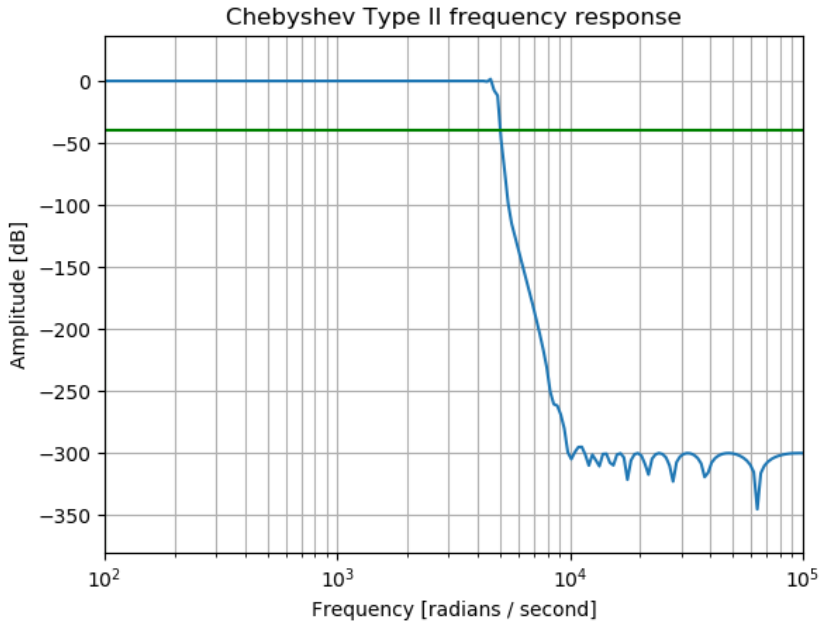


Figura 3. 9. Respuesta de un filtro IIR orden 50 y frecuencia de corte de 6000Hz

Fuente: Elaborado por el autor.

Parte del código empleado para obtener los resultados antes mostrados, son descritos en las siguientes líneas:

```
from __future__ import division, print_function

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


def plot_response(fs, w, h, title):
    plt.figure()
    plt.plot(0.5*fs*w/np.pi, 20*np.log10(np.abs(h)))
    plt.ylim(-40, 5)
    plt.xlim(0, 0.5*fs)
    plt.grid(True)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Gain (dB)')
    plt.title(title)


# Low-pass filter design parameters
fs = 22050.0       # Sample rate, Hz
fc = int (input('Ingrese el valor de la frecuencia de corte en Hz:    ')) # Desired cutoff frequency, Hz
trans_width = 250  # Width of transition from pass band to stop band, Hz
numtaps = int (input('Ingrese el orden del filtro FIR:   '))     # Size of the FIR filter.
```

De igual forma que en las prácticas anteriores, el código completo se encuentra adjunto en los Anexos.

Preguntas
1. ¿Qué sucede cuando él mientras se incrementa el número del orden de un filtro? Explique.
2. Si se desea atenuar frecuencias superiores a los 6000Hz, ¿ Cuál de los dos tipos de filtro ofrece mejor respuesta en frecuencia?
3. La disminución del rizado en la banda de frecuencias bloqueadas simboliza que la respuesta del filtro diseñado es: • Mejor • Peor Explique. .

CONCLUSIONES

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- El estudiante que desee iniciarse en **Python**, para propósitos de Tratamiento Digital de Señales, encontrara toda la ayuda necesaria gracias a la flexibilidad y facilidad que este *software* presenta, además de que se puede encontrar en internet y blogs de colaboradores, gran cantidad de contenidos y ejemplos de los diversos campos de investigación para los cuales el *software* **Python**, puede ser útil, con esto se tiene un entorno de aprendizaje, actualización e investigación permanente.
- Al tener **Python**, un entorno de computación práctico con posibilidad de mostrar resultados de forma inmediata mediante el empleo de gráficas, el estudiante será capaz de asimilar y analizar de manera criteriosa la teoría que lo acerca al campo de telecomunicaciones y Tratamiento de Señales, además, conforme vaya adquiriendo experiencia en el entorno de programación será capaz de desarrollar posibles soluciones a otros tipos de problemas como por ejemplo los algoritmos presentes en los sistemas de transmisión de fibra óptica, teorías que son expuestas en el transcurso de la carrera de Ingeniería en Electrónica Digital y Telecomunicaciones de la Universidad Tecnológica Israel, de esta manera se está colaborando con futuras generaciones y campos de investigación que el país requiere.
- El *software* libre **Python** contiene funcionalidades útiles para el Tratamiento Digital de Señales, por ello, puede ser considerado como *software* introductorio para este propósito en lugar de programas licenciados como **Matlab.**
- Al tratarse de un *software* libre y en constante crecimiento, los egresados y graduados en Electrónica Digital y Telecomunicaciones en la Universidad Tecnológica Israel, estarán en plena capacidad de proponer, ampliar y actualizar el manual de prácticas que se ha propuesto con este texto, además podrán presentar otras posibles soluciones y diseños a sistemas más complejos, como por ejemplo los campos que involucran algoritmos para reconocimiento facial.

RECOMENDACIONES

- Sugerir el *software* libre **Python**, como herramienta de apoyo para simulaciones de algoritmos matemáticos presentes en los sistemas sujetos a revisión durante la formación académica de un Ingeniero en Telecomunicaciones de la Universidad Tecnológica Israel.
- Para iniciar en el lenguaje de programación **Python** y probar sus características, se recomienda utilizar el modo interactivo, ya sea a través de consola o de la web de ***I*Python**, con ello se puede verificar la sintaxis y utilización de los comandos previo a ser utilizados en el desarrollo de alguna solución específica.
- Entre la diversidad y existencia de varios Entornos de Desarrollo Integrados para **Python**, (IDE), se recomienda la utilización de ***PyCharm***, ya que este IDE es sencillo de manejar además es el más popular para este lenguaje debido a que está representado por JetBrains, compañía que postea constantemente actualizaciones y soporte técnico, otra ventaja exclusiva es que contiene una versión profesional adecuada para quienes deseen empezar como desarrolladores.
- Se recomienda tener en cuenta las características de hardware y software descritas en el manual de prácticas, para garantizar que la aplicación presentada pueda correr con normalidad.
- Se recomienda considerar un plazo de 60 minutos por práctica, ya que es el tiempo suficiente para escribir el código y cargar los módulos necesarios.
- Se recomienda ampliar el alcance de este proyecto hacia la cuantificación y modulación de señales digitales, simulaciones que son útiles para laboratorios de Comunicaciones.

REFERENCIAS BIBLIOGRÁFICAS

Brea, L. M. (2013). *PIMCD2013-python .* Publicación 1.0.

Desde Linux. (2007-2018). *Desde Linux*. Obtenido de https://blog.desdelinux.net/

Diepold, D. -I. (2016). *Liniear Time-Invariant Systems with Discrete Time.* Múnich: Instituto de procesamiento de datos, Universidad de Múnich.

Downey, A. B. (2014). *Think DSP Digital Signal Processing in Python* (Version 1.0.9 ed.). Needham, Massachusetts: Green Tea Press.

158bd8c. (2017). *Install PyCharm and Anaconda (Windows /Mac/Ubuntu).* Medium.

IPython development team. (2018). *iPython.org*. Obtenido de https://iPython.org/

Manolakis y Proakis, J. G. (2007). *Tratamiento digital de señales.* Madrid: 4.

Matplotlib development team. (2012-2018). *matplotlib.org/*. Obtenido de https://matplotlib.org/

NumPy developers. . (2018). *numpy.org*. Obtenido de http://www.numpy.org/

SciPy developers. . (2018). *Scipy.org*. Obtenido de https://www.scipy.org/

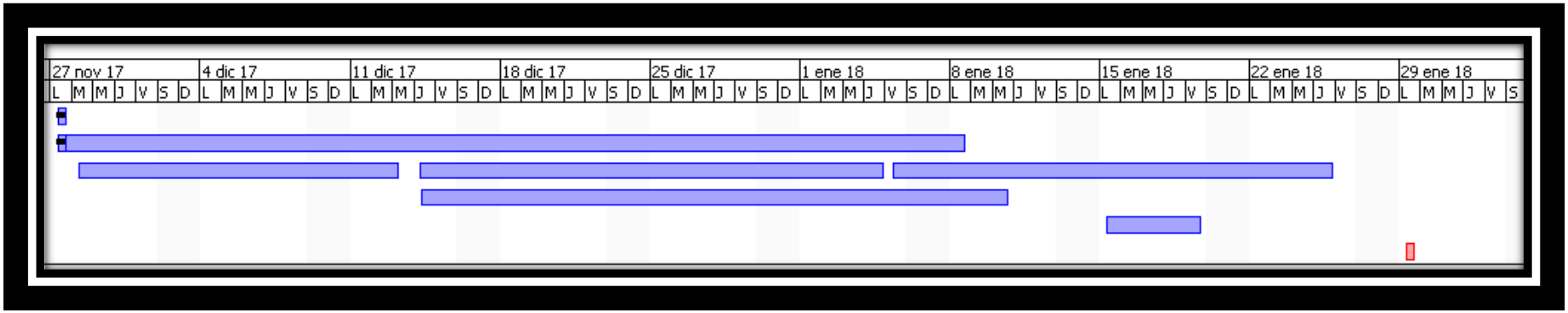
SymPy Development Team. . (2018). *sympy.org*. Obtenido de https://www.sympy.org/en/index.html

ANEXO 1. CRONOGRAMA DE ACTIVIDADES DE LAS TAREAS A REALIZAR Y RECURSOS NECESARIOS

Duración y fecha de cumplimiento de cada tarea.

	Nombre	Duracion	Inicio	Terminado
1	Presentación del plan PIC	1 day	27/11/17 8:00	27/11/17 17:00
2	Fundamentación teórica, detalle de las librerías complementarias de PYTHON	31 days	27/11/17 8:00	08/01/18 17:00
3	Realizar un diagnóstico de las funciones de PYTHON aplicado a TDS	42 days	28/11/17 8:00	25/01/18 23:00
4	Desarrollo de 3 Prácticas para Laboratorio de Tratamiento Digital de Señales	20 days	14/12/17 8:00	10/01/18 17:00
5	Validación de las prácticas realizadas según syllabus de estudio	5 days	15/01/18 8:00	19/01/18 17:00
6	Defensa del PIC	1 day	29/01/18 8:00	29/01/18 17:00

Diagrama de Grantt



ANEXO 2. CÓDIGO PYTHON

PRACTICA 1

```
# Señales discretas sistema Variante o Invariante

# Definir la funcion para el ejemplo a traves de entrada de usuario
#formato de ingreso ejemplo:  x(n)-x(n-k)
#y(n,k)# propuesta:
from py_expression_eval import Parser
from pybigparser import *
from sympy import *
import sys
import numpy as np
import matplotlib.pyplot as plt
from math import e
w,A,B=symbols('w A B');
n = 'n'; y = symbols('y'); k = 'k';
x=Symbol('x');a1=Symbol('a1');a2=Symbol('a2');
fnc = (input('ingrese función:    '));
if "x(n)" in fnc and "x(n-1)" in fnc: #Modificar expresion "x(n-1)" según
necesidad
    fnc0 = fnc.replace("n" ,"n-k")
    expr=sympify(fnc0)
    print ("y(n,k):  ")
    pprint(expr)
else:
    if  "x(n)" in fnc:
        fnc0 = fnc.replace("x(n)", "x(n-k)")
        expr = sympify(fnc0)
        print("y(n,k):  ")
        pprint(expr)
#y(n-k)
xn=fnc
n=sympify(n);k=sympify(k)
fnc2=sympify(xn)
fnc2=fnc2.xreplace({n:n-k})
print ("y(n-k):  ")
pprint(fnc2)
if (expr==fnc2):
    print (" Sistema invariante en el tiempo")
    print('\n')
else:
    print (" Sistema variante en el tiempo")
    print('\n')
#Prueba de sistema Lineal, reemplazar el exponente de la función de ser
el caso
if "x(n)**2" in fnc:
    flineal = fnc.replace("x(n)**2", "a1*x1")
    flineal2 = fnc.replace("x(n)**2", "a2*x2")
    y1 = sympify(flineal)
```

```
    y2 = sympify(flineal2)
    flineal3 = (y1 + y2) ** 2
    pprint(flineal3)
else:
    if "x" in fnc:  # Modificar expresion "x(n-1)" según necesidad
        flineal = fnc.replace("x", "a1*x1")
        flineal2 = fnc.replace("x", "a2*x2")
        y1 = sympify(flineal)
        y2 = sympify(flineal2)
        flineal3 = y1 + y2
        pprint(flineal3)

if "x" in fnc:  # Modificar expresion "x(n-1)" según necesidad
    Y1 = fnc.replace("x", "x1")
    Y2 = fnc.replace("x", "x2")
    Y1p = sympify(Y1);Y2p = sympify(Y2); A1 = sympify(a1);A2 =
sympify(a2);
    y1p = A1 * Y1p
    y2p = A2 * Y2p
    y3p = y1p + y2p
    pprint(y3p)
if (flineal3==y3p):
    print('\n')
    print('Sistema discreto LINEAL')
else:
    print('\n')
    print('Sistema discreto NO LINEAL')

 #x(n)-x(n-1)

#t = (input('ingrese valores de tiempo para "n":    '));
if "x(n)" in fnc:
    STB = fnc.replace("x(n)", "D(n)")
    STB2 = sympify(STB)
    STB3 = STB2.subs(n,0)
    STB4 = STB2.subs(n,1)
    stb4 = str(STB4)
    stb3 = str (STB3)
    if "D(0)" in stb3:
        res = stb3.replace("D(0)","1")
        res2=sympify(res)
        out=solve(res2)
    if "D(1)" in stb4:
        resp = stb4.replace("D(1)", "0")
        res2p = sympify(resp)
        outp = solve(res2p)

if outp==out:
    print('\n')
    print('Sistema discreto ESTABLE')
else:
    print('\n')
    print('Sistema discreto NO ESTABLE')
```

ANEXO 3. CODIGO PYTHON

Práctica 2

```
# Señales discretas en convolucion
# x[n]*h[n]


import numpy as np
import matplotlib.pyplot as plt
global xn
global hn
# Definir la funcion para el ejemplo
def fn_x(n):
    # función matemática CAMBIAR AQUI
    xn=0
    if (n>=0 and n<=6):
        xn=np.piecewise(n,[n==0,n==1,n==3],[1,2,4],dtype=float)
        #xn=np.sin(n)
    # función matemática CAMBIAR AQUI
    return(xn)

def fn_h(n):
    # función matemática CAMBIAR AQUI
    hn=0
    if (n>=0 and n<=4):
        hn = 1
    # función matemática CAMBIAR AQUI
    return(hn)

# Programa para graficar la función
# INGRESO
print(' Rango para evaluar la convolución')
t0=int(input('valor inicial: '))
tf=int(input('valor final: '))

# PROCEDIMIENTO
deltax=1   # Resolución 1 para discretas, <<1 para continuas
muestras=int((tf-t0)//deltax +1)   #número de muestras para gráfica
# Valores en forma discreta
t=np.zeros(muestras, dtype=float) # Dimensiona los arreglos
x=np.zeros(muestras, dtype=float)
h=np.zeros(muestras, dtype=float)
y=np.zeros(muestras, dtype=float)
for i in range(0,muestras):
    t[i]=t0+deltax*i     #para el eje x
    x[i]=fn_x(t[i])
    h[i]=fn_h(t[i])
# suma de convolucion
matriz=np.zeros(shape=(muestras,muestras), dtype=float)
for i in range(0,muestras):
    ysuma=0
    for k in range(0,muestras):
```

```
        matriz[i,k]=fn_x(t[k])*fn_h(t[i]-t[k])
        ysuma=ysuma + matriz[i,k]

    y[i]=ysuma

# SALIDA
#Escala y para el grafico
print('Resultado de la convolución: ',(y) )
plt.figure(1)        # define la grafica
plt.suptitle('Convolución x[n]*h[n]')
plt.subplot(311)     # grafica de kx1 y subgrafica 1
plt.ylabel('x[n]')
xtecho=np.max(x+1)+ 0.1*np.max(x)
xpiso =np.min(x)- 0.1*np.min(x+1)
plt.axis((t0,tf,xpiso,xtecho))
plt.stem(t, x, linefmt='r.-', markerfmt='ro', basefmt='g-')
plt.subplot(312)     # grafica de kx1 y subgrafica 1
plt.ylabel('h[n]')
htecho=np.max(h+1)+ 0.1*np.max(h)
hpiso =np.min(h)- 0.1*np.min(h+1)
plt.axis((t0,tf,hpiso,htecho))
plt.stem(t, h, linefmt='b.-', markerfmt='bo', basefmt='g-')
plt.subplot(313)     # grafica de kx1 y subgrafica 1
plt.ylabel('x[n]*h[n]')
ytecho=np.max(y+1)+ 0.1*np.max(y)
ypiso =np.min(y)- 0.1*np.min(y+1)
plt.axis((t0,tf,ypiso,ytecho))
plt.stem(t, y, linefmt='m.-', markerfmt='mo', basefmt='g-')
plt.show()
```

ANEXO 4. CÓDIGO PYTHON

Práctica 3

```
from __future__ import division, print_function

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


def plot_response(fs, w, h, title):
    plt.figure()
    plt.plot(0.5*fs*w/np.pi, 20*np.log10(np.abs(h)))
    plt.ylim(-40, 5)
    plt.xlim(0, 0.5*fs)
    plt.grid(True)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Gain (dB)')
    plt.title(title)


# Low-pass filter design parameters
fs = 22050.0       # Sample rate, Hz
fc = int (input('Ingrese el valor de la frecuencia de corte en Hz:    '))
# Desired cutoff frequency, Hz
trans_width = 250  # Width of transition from pass band to stop band, Hz
numtaps = int (input('Ingrese el orden del filtro FIR:   '))      # Size
of the FIR filter.

taps = signal.remez(numtaps, [0, fc, fc + trans_width, 0.5*fs],
                    [1, 0], Hz=fs)#ojo
w, h = signal.freqz(taps, [1], worN=2000)

plot_response(fs, w, h, "Low-pass Filter")
```

```
# High-pass filter design parameters
fs = 22050.0       # Sample rate, Hz
cutoff = 2000.0    # Desired cutoff frequency, Hz
trans_width = 250  # Width of transition from pass band to stop band, Hz
numtaps = 125      # Size of the FIR filter.

taps = signal.remez(numtaps, [0, fc - trans_width, fc, 0.5*fs],
                    [0, 1], Hz=fs)
w, h = signal.freqz(taps, [1], worN=2000)

plot_response(fs, w, h, "High-pass Filter")

# Band-pass filter design parameters
fs = 22050.0          # Sample rate, Hz
band = [2000, 5000]   # Desired pass band, Hz
trans_width = 260     # Width of transition from pass band to stop band,
Hz
numtaps = 125         # Size of the FIR filter.
```

```
edges = [0, band[0] - trans_width,
         band[0], band[1],
         band[1] + trans_width, 0.5*fs]
taps = signal.remez(numtaps, edges, [0, 1, 0], Hz=fs)
w, h = signal.freqz(taps, [1], worN=2000)

plot_response(fs, w, h, "Band-pass Filter")

# Band-stop filter design parameters
fs = 22050.0          # Sample rate, Hz
band = [6000, 8000]   # Desired stop band, Hz
trans_width = 200     # Width of transition from pass band to stop band,
Hz
numtaps = 175         # Size of the FIR filter.

edges = [0, band[0] - trans_width,
         band[0], band[1],
         band[1] + trans_width, 0.5*fs]
taps = signal.remez(numtaps, edges, [1, 0, 1], Hz=fs)
w, h = signal.freqz(taps, [1], worN=2000)

plot_response(fs, w, h, "Band-stop Filter")

plt.show()

signal.lfilter()
```

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

print ('\n')
print ('Respuesta en frecuencia de un filtro IIR')
Ord = int (input ('ingrese orden del filtro:  '))
fc= int(input ('ingrese frecuencia de corte:   '))
b, a = signal.cheby2(Ord, 40, fc, 'low', analog=True)
w, h = signal.freqs(b, a)
plt.semilogx(w, 20 * np.log10(abs(h)))
plt.title('Chebyshev Type II frequency response (rs=40)')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.axvline(100, color='green') # cutoff frequency
plt.axhline(-40, color='green') # rs
plt.show()
```

ANEXO 5. MANUAL DE USUARIO

1. Introducción

La interfaz gráfica desarrollada, permite interacción entre el computador y el estudiante o usuario, quien tendrá un intérprete y soporte para complementar ejemplos de temas tratados en la materia Tratamiento Digital de Señales de la UIsrael.

1.1. Objetivo del manual

Entregar una guía para poder utilizar la aplicación de computador desarrollada como soporte para la materia Tratamiento Digital de Señales de la UIsraael.

1.2. Alcance

Dirigido exclusivamente para temarios de Tratamiento Digital de Señales similares a los de la UIsrael.

2. Acceso al interprete para apoyo de TDS de la UIsrael

Abra el IDE Pycharm

Figura. Anexo.4.1 Icono PyCharm community
Fuente: Elaborado por el autor.

En caso de que la aplicación IDE PyCharm, se encuentre correctamente instalada según lo mostrado en el Capìtulo II, se abrirá sin inconvenientes mostrando el escritorio de la aplicación.

3. Copiar el Script de apoyo de TDS y ejecutar.

Copie el contenido del bloc de notas "TDS_App" proporcionado por el instructor.

A continuación abra un nuevo archivo en Python, dentro del IDE Pycharm.

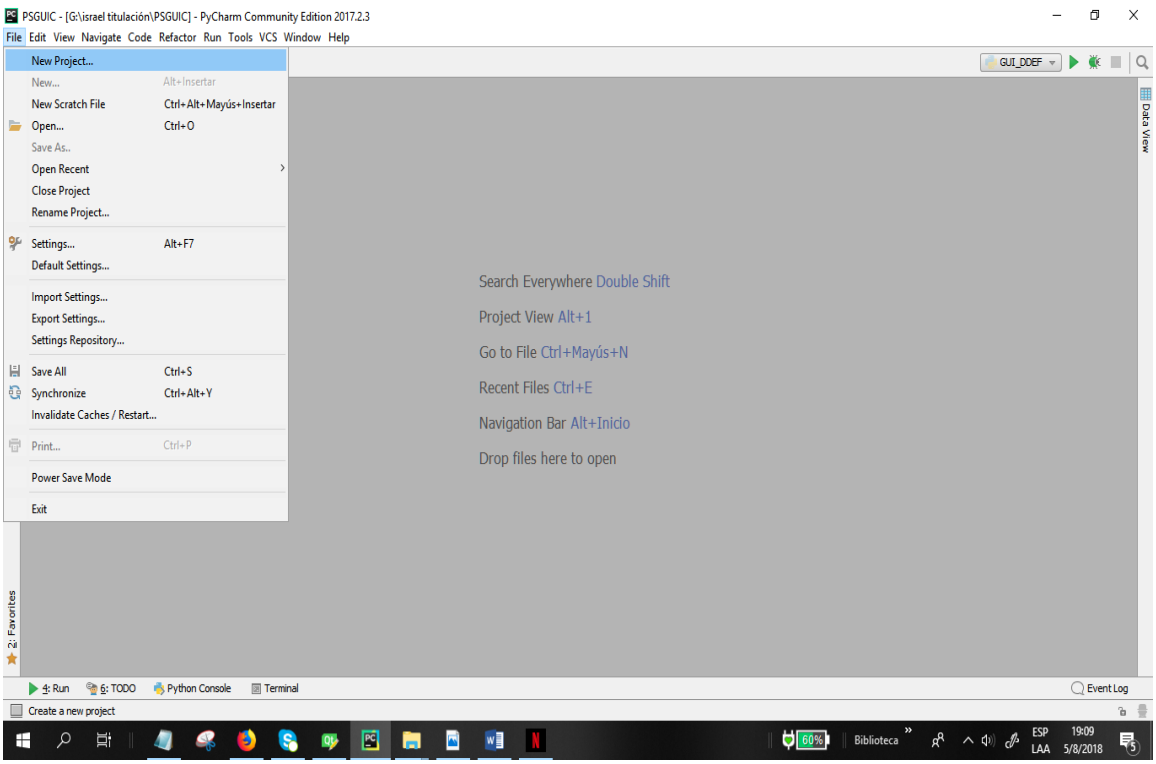


Figura. Anexo.4.2 Creando el archivo TDS.py

Fuente: Elaborado por el autor

Al dar click en *"New Project"*, se abrirá una nueva ventana

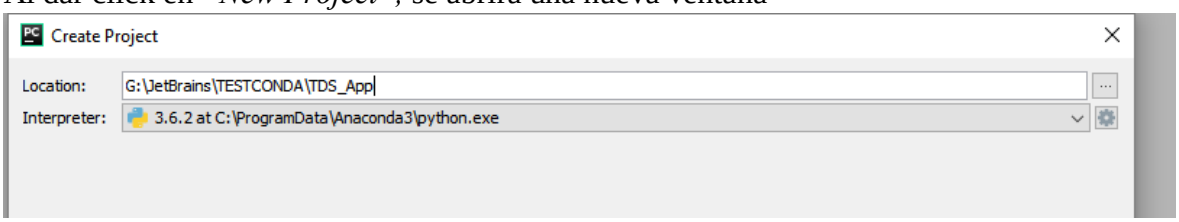


Figura. Anexo.4.3 Creando archivo TDS.py
Fuente: Elaborado por el autor

Seleccione la ruta donde desea guardar el archivo y a continuación coloque el nombre del archivo, de preferencia utilice el nombre TDS.py, para hacer referencia al script desarrollado en este proyecto.

Dar click en “Create” y a continuación en “Open in the current Window”, esto para mantener la misma ventana abierta y no crear una adicional.

Vaya a la ruta donde creó la nueva carpeta con el nombre TDS_App y en el lugar señalado pegue los archivos: “PracticaGUI” y “PracticaGUI.ui”, “ondas.qrc”, “ondas_rc”, proporcionados por el instructor.

El desplegar la pestaña debería mostrar una ruta similar a la de la siguiente figura:

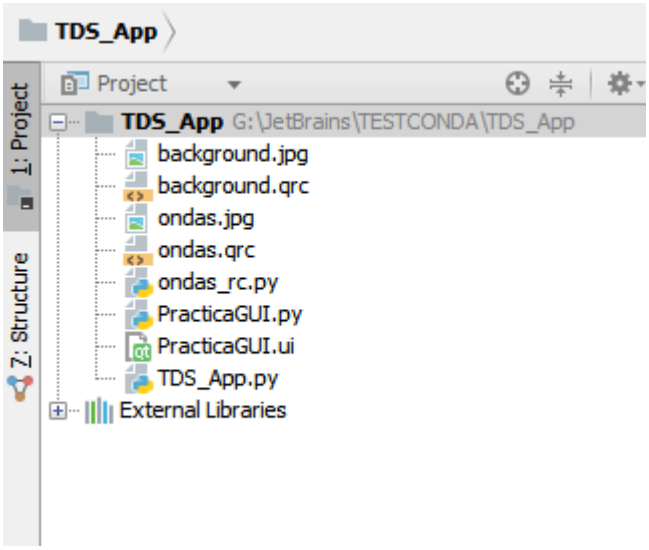


Figura. Anexo.4. 4 Ruta y contenido de la carpeta creada TDS_App
Fuente: Elaborado por el autor

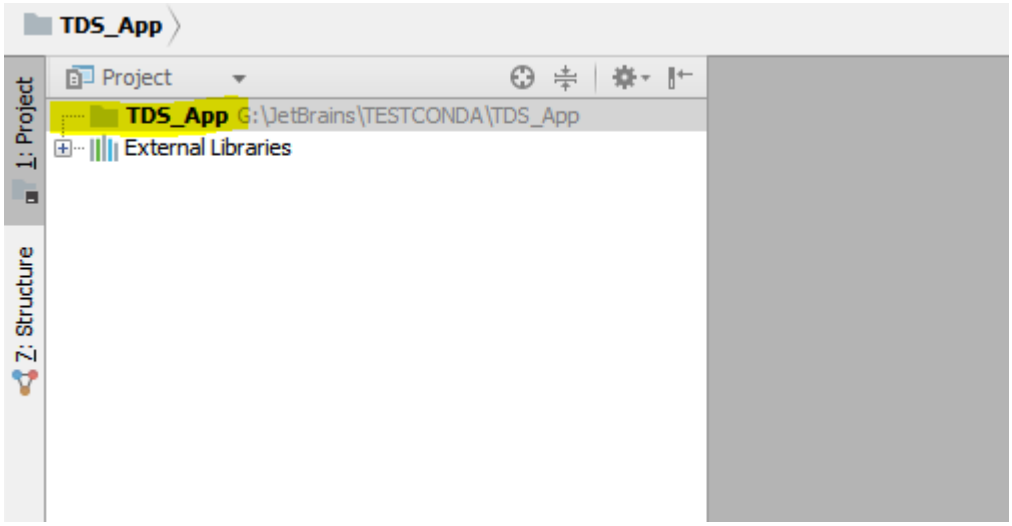


Figura. Anexo.4.5 Creando archivo TDS.py
Fuente: Elaborado por el autor

A continuación de click derecho sobre la carpeta TDS_App, y seleccione “New *Python* File”

Pegue dentro de la Pestaña TDS_App, el contenido del bloc de notas “TDS_App”.

Diríjase hacia la barra de tareas del IDE de PyCharm y seleccione Run

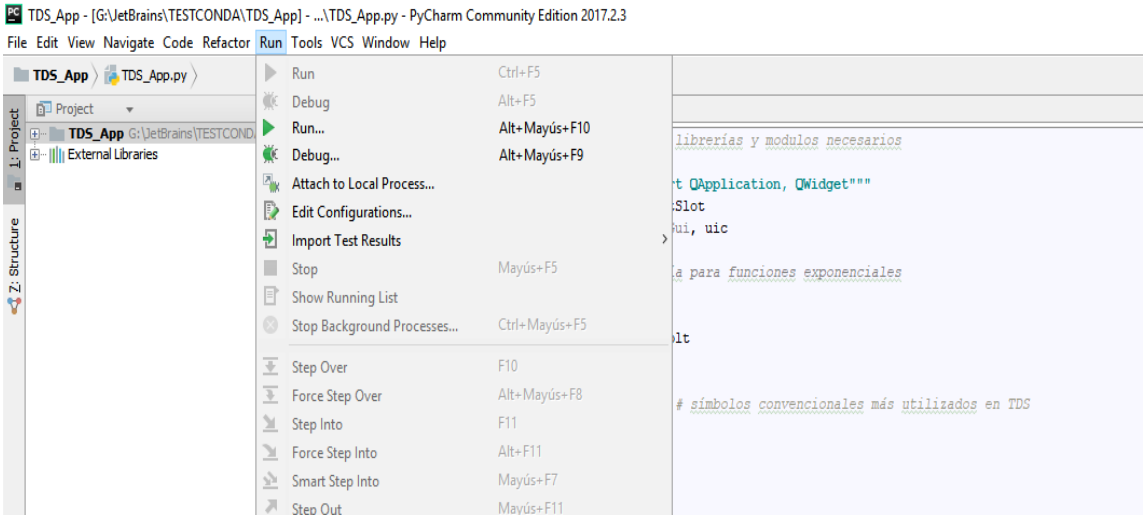


Figura. Anexo.4. 6 Ejecutando TDS_App.py
Fuente: Elaborado por el autor

A continuación seleccione la opción “Run…” y espere a que se despliegue la aplicación.

Figura. Anexo.4.7 Interfaz gráfica para apoyo de TDS en la UIsrael
Fuente: Elaborado por el autor

La aplicación tiene 3 pestañas esenciales para apoyo de la materia TDS de la UIsrael, son pestañas con contenido simple e intuitivo.

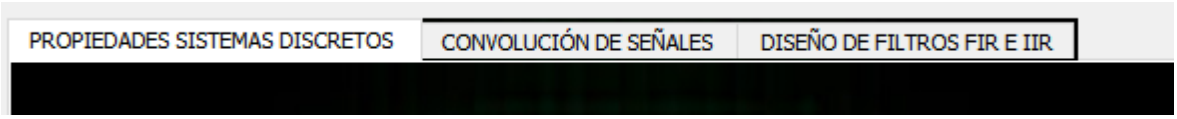


Figura. Anexo.4.8 Pestañas principales
Fuente: Elaborado por el autor

En la primera pestaña se tiene 3 campos para interacción del usuario:

- Casillero “y(n)”.
- Boton “Calcular”.
- Boton “Limpiar”.

El casillero y(n) admite toda clase de funciones a ser evaluadas, para lo cual el formato de ingreso debe ser en código *Python*, las únicas funciones no admitidas, son las que utilizan símbolo de sumatoria, por lo demás los cálculos serán efectuados siempre que el ingreso sea en formato admitido por código *Python*, es decir para ingresar una función cuadrática se debe emplear:

x(n)**2, donde **2 indica que se está elevando al cuadrado la función.

Asi mismo una función exponencial e**x(n)+1, indica que el valor del número de Euler se está elevando a un exponente determinado por la señal de entrada x(n)+1.

El botón calcular desplegará los resultados esperados al haber ingresado la función en el casillero y(n).

El botón limpiar borra todos los resultados que se muestran en los cuadros de muestra y permite una nueva evaluación de otra función de entrada o de la misma en caso de no ser reemplazada.

La segunda pestaña “Convolución de Señales”, contiene 18 cuadros para ingreso de valores y el botón para calcular la convolución.

Cada cuadro está representando un valor en el eje de coordenadas “x”, donde el máximo valor negativo es -4 y el máximo valor positivo es 4, el cero viene marcado con una flecha hacia arriba usando así connotación tradicional para quienes siguen el curso de TDS.

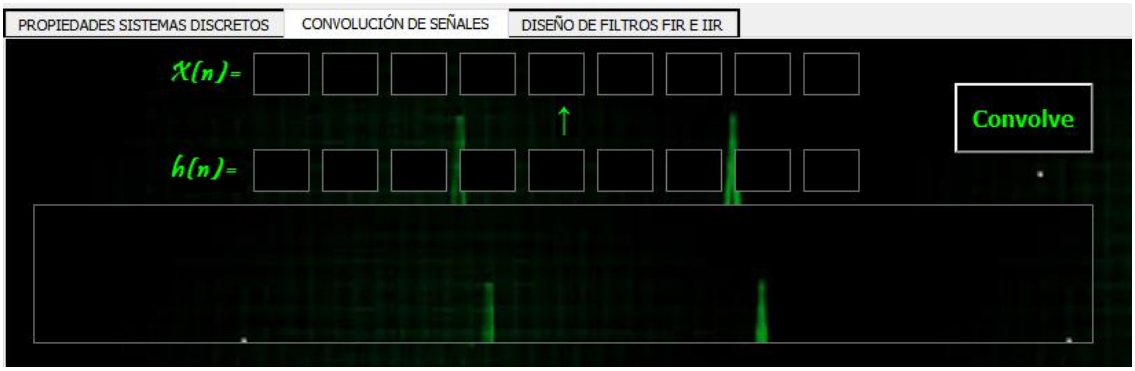


Figura. Anexo.4. 9 Pestaña Convolución de Señales
Fuente: Elaborado por el autor

La tercera pestaña “Diseño de Filtros FIR e IIR”, contiene cuadros de ingreso para datos de valores de frecuencia de corte y orden del filtro, además de el caso de Filtros FIR, el usuario puede escoger el tipo de respuesta del filtro es decir: Pasabajos, Pasa Altos, Pasa Banda.

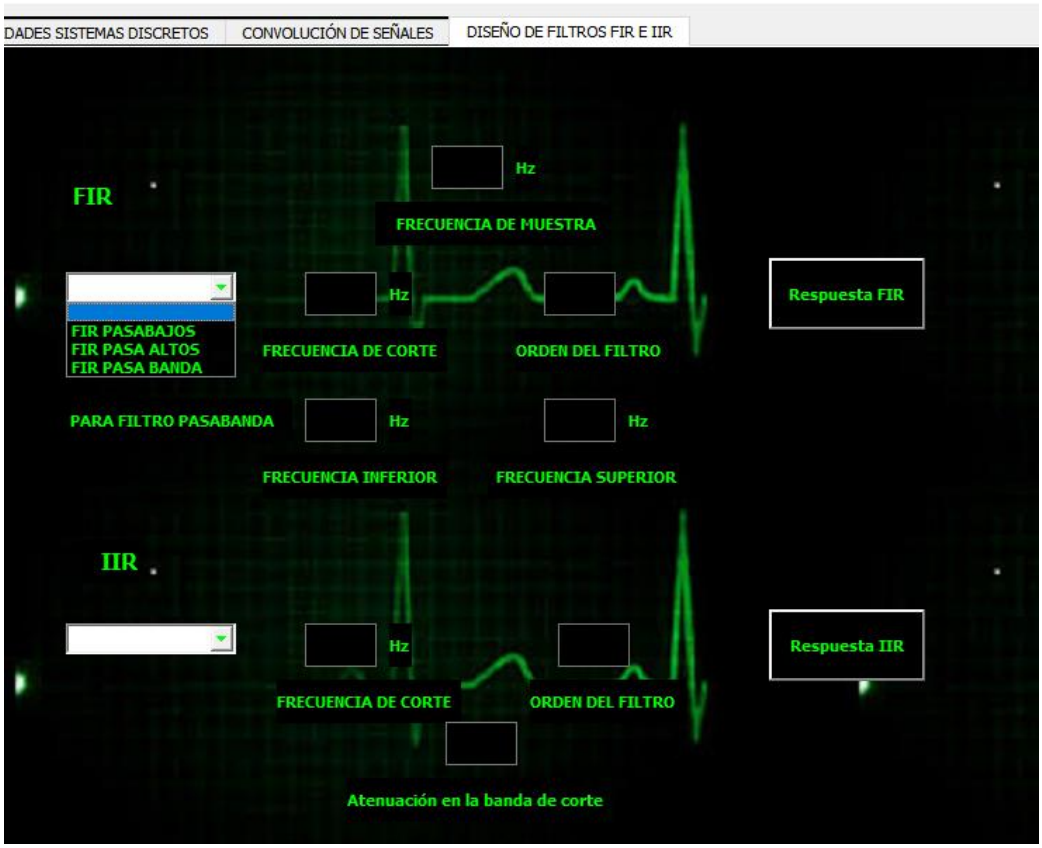


Figura. Anexo.4.10 Tipos de filtros admitidos en el diseño de Filtros FIR
Fuente: Elaborado por el autor

Para el caso de los filtros IIR, se puede elegir el algoritmo del Diseño del Filtro ya sea por medio de Butterworth o por medio de ChebyshevII, recordando que el alcance de este proyecto está dirigido únicamente a filtros Pasa Bajos.

ANEXO 6. MANUAL TÉCNICO

1. Introducción

El presente manual contiene la explicación técnica de cada una de las funciones empleadas en el desarrollo del Script TDS_App.

1.1. Librerías y Módulos

```
import matplotlib
from PyQt5 import QtWidgets
import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from sympy import *
"""from PyQt5.QtWidgets import QApplication, QWidget"""
from PyQt5.QtCore import pyqtSlot
from PyQt5 import QtCore, QtGui, uic
from sympy import *
from math import e  # Librería para funciones exponenciales
from sys import *
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
```

Para importar una librería se utiliza la palabra clave **import**, ejemplo:

import Matplotlib

Sin embargo cuando únicamente se desee realizar la importación de un módulo en específico de alguna librería, se emplea la palabra clave **from** seguido de la librería de la cual se quiere extraer el módulo y a continuación se utiliza la palabra **import** seguido de los módulos que se desean extraer. Ejemplo:

from PyQt5 **import** QtCore, QtGui, QtWidgets

Cuando no se tiene el conocimiento de que módulos contiene una librería, podemos llamar a todos sus módulos utilizando el símbolo de *. Ejemplo:

from sys **import ***

1.2. Declaración de Variables

Declarar variables en *Python* es sencillo, se debe colocar el nombre de la variable, a continuación el signo “igual (=)” luego de ello especificar el tipo de variable seguido al valor que le va a ser asignado. Ejemplo

```
y = symbols('y');
```

en el ejemplo el nombre de la variable es “y”, el tipo de variable o valor que se espera es simbólico y por tanto debe ir dentro de apóstrofes el símbolo que la representa, en este caso ‘y’.

Python admite entre otros los siguientes tipos de variables:

- Int
- Str
- Symbols
- Float

Para su declaración en el espacio local de una función o global de todo el script se puede utilizar la palabra clave **global.** Ejemplo:

```
global plot_response
```

Cuando se crea un archivo una interfaz de usuario en PyqT5 QT Designer, estos se guardan en formato .ui, con ello es necesario transformar a código *Python* la interfaz .ui a .py.

Para ello se debe utilizar la consola del IDE de PyCharm con el comando:

pyuic5 PracticaGUI.ui > PracticaGUI.py

Para llamar el módulo de la interfaz de usuario creada por medio de QT Designer, se utiliza la línea de comando siguiente:

```
qtCreatorFile = "PracticaGUI.ui"
```

las siguientes líneas son convenios para desarrollar una interfaz gráfica en *Python*, así que no se procederán aplicar, se puede decir que están estandarizadas y por tanto se pueden copiar y pegar para cualquier proyecto.

```
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)


class MyApp(QtWidgets.QMainWindow, Ui_MainWindow, QtWidgets.QTabWidget):
    def __init__(self):  # Funcion principal init, para poder utilizar
self
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```

los programas creados se subdividen en funciones, las cuales ayudan a simplificar el programa, cada función se puede acceder mediante la conexión de las señales.

Para conectar los botones colocados en la interfaz de usuario diseñada en QT Designer se utiliza el siguiente comando:

```
self.ok.clicked.connect(self.Propierties)
```

La línea conecta el botton push hacia la función Propierties, por tanto se pueden crear otras funciones y utilizar la misma línea mostrada para conectar los botones push hacia dichas funciones.

Otro de los Widgets colocados en la interfaz de usuario es el combo Box, el cual se utilizó para obtener las diferentes opciones de filtros, para adherir sus títulos se utiliza la siguiente línea:

```
self.bx1.addItem("FIR PASABAJOS")
```

El nombre del objeto “bx1”, se puede reemplazar por la que convenga.

Otro de los Widgets, es el cuadro de ingreso o TextBox, en ellos se solicitan datos en letras o números y son tratados de acuerdo al tipo de variable ingresada. La línea de comando para almacenar el dato ingresado por el usuario es la siguiente:

```
fnc = str(self.xdn.toPlainText())
```

La línea mostrada a continuación, es un condicional que identifica si alguna expresión contiene las funciones identificadas dentro de comillas, en la **variable fnc**.

```
if "x(n)" in fnc or "x(n-1)" in fnc or "x(n-2)" in fnc:
```

La línea a continuación, indentifica la función que permite reemplazar símbolos dentro de un string por otros, en este caso se reemplaza “n” por “n-k”.

```
fnc0 = fnc.replace("n", "n-k")
```

En líneas anteriores vimos como ingresar valores a un cuadro de texto o TextBox, la siguiente línea muestra como mostrar valores en los cuadros mencionados.

```
self.ydn.setText(final)
```

Donde ydn representa el cuadro de texto que mostrará el valor.

Lo anterior descrito se aplica para todas las líneas de programa que contienen las palabras clave ya explicadas.

La línea de comando siguiente hace uso de una función que permite crear un vector:

```
n = np.linspace(-8, 8, 17)
```

En este caso se crea un vector de 17 números desde -8 a 8.

La función piecewise de Numpy, crea una función por trozos.

```
s = np.piecewise(n, [n == -4, n == -3, n == -2, n == -1, n == 0, n == 1,
n == 2, n == 3, n == 4],
                 [Psm4, Psm3, Psm2, Psm1, Ps0, Ps1, Ps2, Ps3, Ps4],
dtype=float)
```

El formato y forma de interpretarlo sería lo siguiente:

Numpy.piecewise(variable que adquiere los puntos del eje x, [puntos de coordenadas del eje x], [valores de cada punto de coordenadas del eje x, tipo de dato que será guardado en las variables])

La librería Numpy tiene incorporada la función de convolución con ello, únicamente se debe llamarla y colocar dentro de sus argumentos los vectores de los cuales se espera la convolución.

```
y1 = np.convolve(s, hn,'same')
```

La línea 214 del programa ayuda a colocar texto en los cuadros de edición, ya anteriormente se detalló cómo obtener texto y asociarlo a una variable, en esta oportunidad se muestra la operación contraria que seria mostrar los datos asignados en una variable.

```
print(self.shcv.setPlainText(convolucion))
```

Las líneas que generan las gráficas a través de la librería Matplotlib, son explicadas en las siguientes definiciones.

```
fig1 = plt.figure(1)  # define la grafica
plt.suptitle('Convolución x[n]*h[n]')
ax1 = fig1.add_subplot(311)  # grafica de kx1 y
plt.ylabel('x[n]')
xtecho = np.max(n) + 0.1 * np.max(n)
xpiso = np.min(n) - 0.1 * np.min(n)
plt.axis([-8, 8, xpiso, xtecho])
plt.stem(n, s, linefmt='r.-', markerfmt='ro', basefmt='g-')
for xy in zip(n, s):
    ax1.annotate('(%s,%s)' % xy, xy=xy, textcoords='data')
plt.grid()
```

La primera línea de las mostradas anteriormente, guarda en una variable llamada fig1, el argumento *figure*, dicho argumento es el que utiliza Matplotlib para generar gráficas, en todo caso cabe recalcar que Matplotlib por convención se importa con el abreviatura *plt*.

La segunda línea por medio de la función suptitle, coloca el título superior que tendrá la gráfica creada.

Colocar add_subplot(posición), implica adherir una gráfica de menor tamaño en la posición elegida por cada programador de manera que a través de una figura creada se pueden mostrar diferentes tipos de gráficas con origen de datos que también son diferentes.

La línea que contiene el argumento ylabel de Matplotlib, únicamente añade una etiqueta al eje y con la identificación que el usuario quiera otorgarle.

El argumento máx. y mín. de Numpy, importado como np, adquieren los valores máximo y mínimo de la variable n, misma que indica el rango de valores contenidos en el eje de ordenadas.

El argumento axis de Matplorlib, importado como plt, indica los valores en los cuales se mostraran los ejes, en este caso se mostrara el máximo de n y el mínimo de n para el eje Y, mientras que el eje de abscisas irá de -8 a 8.

El argumento stem de Matplotlib, crea una gráfica discreta uniendo tallos desde el punto del eje de ordenadas hacia el eje de abscisas.

La función zip, regresa la interacción de dos vectores, en este caso el vector n que contiene los puntos del eje x y el vector s, mismo que contiene los valores del eje y. con ello en la línea siguiente utilizando el argumento annotate podemos colocar los textos que identifican cada punto mostrado en las gráficas y subgráficas creadas en Matplotlib.

Para diseño de filtros FIR, Numpy ya incluye funciones específicas que son los argumentos remezz y freqz, queda en el lector visitar la página oficial de Numpy y consultar su estructura y funcionamiento.

```
taps = signal.remez( numtaps, [0, fc - trans_width, fc, 0.5 *
fs],
                     [0, 1], Hz=fs )
w, h = signal.freqz( taps, [1], worN=2000 )
```

Para el caso de filtros IIR, así mismo se cuentan con funciones ya definidas que son las e las líneas mostradas a continuación.

```
[b, a] = signal.cheby2( Ord, ATEN, fc, 'low', analog=True )
 [b, a] = signal.butter( Ord, fc, 'low', analog=True )
```

Obviamente el caso del argumento cheby2 permite crear un filtro ChebyshevII, y el otro argumento un filtro Butterworth.