uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería de Sistemas de Comunicaciones

Trabajo Fin de Grado

"Comparativa de prestaciones y rendimiento entre MATLAB y LabVIEW para entornos de comunicaciones"

Autor

Óscar Serrano García

Tutor

Víctor P.Gil Jiménez

Escuela Politécnica Superior

Septiembre 2019

Agradecimientos

Escribir este proyecto significa poner el punto final a una etapa muy importante en mi vida. Y al escribir estas líneas me vienen a la cabeza muchas de las personas que me han acompañado y me han hecho disfrutar estos años en la universidad.

En primer lugar, dar las gracias a mis profesores, ellos tienen un papel difícil, pues no es fácil transmitir los conocimientos a un alumno, y más aún cuando son de gran complejidad. En especial voy a destacar a mi tutor Víctor, que fue el primero en tenderme una mano para realizar este trabajo, y me ha ayudado siempre que lo he necesitado.

Cuando llegué a la universidad, todo era nuevo, las clases, los compañeros, los profesores, etc. Mis amigos, los de toda la vida, no lo eran. Gracias, vosotros nunca me fallasteis, ya me acompañabais antes y lo seguiréis haciendo, sin duda me hacéis mejor.

Me costó muy poco encontrar grandes personas en este nuevo camino, quiero daros las gracias a todos mis compañeros, por cada tarde de biblioteca y cada cerveza después de un examen.

La casualidad me llevó a cruzarme con ellos y muy pronto se harían imprescindibles, una mala influencia sin duda, pero que han hecho más felices estos últimos años. Gracias por todo “Pizzería”.

Y, para acabar, a mi hermana Lidia, mi madre Viky y mi padre Rafa. Son el mayor apoyo que tengo, me sirven de inspiración y estoy completamente seguro de que no estaría aquí sin ellos. Gracias por acompañarme y guiarme siempre.

Resumen

Este Trabajo Fin de Grado (TFG) se ha realizado con la idea de establecer una comparación profunda entre dos entornos de programación comúnmente utilizados: MATLAB y LabVIEW.

En el campo de la ingeniería es necesario recurrir de forma constante a herramientas que nos permitan hacer cálculos y operaciones matemáticas de forma automática que serían prácticamente inabarcables de forma escrita.

Actualmente existe una gran variedad de herramientas accesibles para todo tipo de aplicaciones, algunas de ellas gratuitas y otras con un coste de licencia. Muchas de estas herramientas son válidas para el desarrollo concreto de una misma tarea, y es ahí donde aparece la comparación, puesto que finalmente se debe elegir una para la realización del trabajo.

MATLAB y LabVIEW son dos entornos, que con sus respectivos lenguajes, permiten el cálculo algebraico y muchos tipos de simulaciones basadas en operaciones vectoriales y matriciales.

Para establecer una comparación entre ellos, y concluir cuales son las ventajas y desventajas de cada uno, se han realizado 4 pruebas y se han medido los tiempos de ejecución. Además, se han observado otros factores como la

accesibilidad del código necesario, la interfaz de trabajo y otras limitaciones en cada uno de los entornos.

Las 4 pruebas han sido: operaciones con matrices, transformada de Fourier, convolución de señales y simulación de un sistema de transmisión y recepción.

Con los datos recopilados, se ha concluido cuáles son las virtudes y cuáles los puntos débiles de cada una de las herramientas, con el objetivo de facilitar la elección a la persona que lea este trabajo.

Abstract

This project is a comparison between two well-known simulations programs, MATLAB and LabVIEW, and the main goal is to decide which one has a better performance during the simulations cases we have proposed. And not only that but remarking the positive and negative aspects of each programming environment.

The first step when we decided to start this comparation was to begin studying and reading all about these two programs. In university I had worked previously with MATLAB in many subjects that we have to simulate some exercise or some theory concepts than we had seen in class before.

That was not the case for LabVIEW, a program which I did not even know before I had started this work. So, at this point I had to read and practice more with this tool than the other one.

After this first approach to these two programming languages, I have started to look for some other investigations that might go on the same direction as mine. This was very useful for me, in order to realize way would be the best to face this comparison.

I found an interesting article from an engineering university in Rumania that was basically stablishing the same comparation as we pretend to do, so I have

taken some good ideas from this article and I try to use them in my one way. I have explained this in detail in chapter number 2 when I talk about previous studies on the matter.

With all this information in my hand, I started to design the different simulations taking into account that I have to check different capabilities from both tools, at the end we decided to do 4 computations:

- Matrix multiplication, both complex and real matrix.
- Fast Fourier Transform of a sine waveform with and without Gaussian noise added.
- Convolution of two random vectors.
- Communication system transmission and reception of and digital signal.

After designing these tests, I had to convert them in real proper code for both languages, graphical for LabVIEW and classical written line code for MATLAB, so I started developing every simulation code.

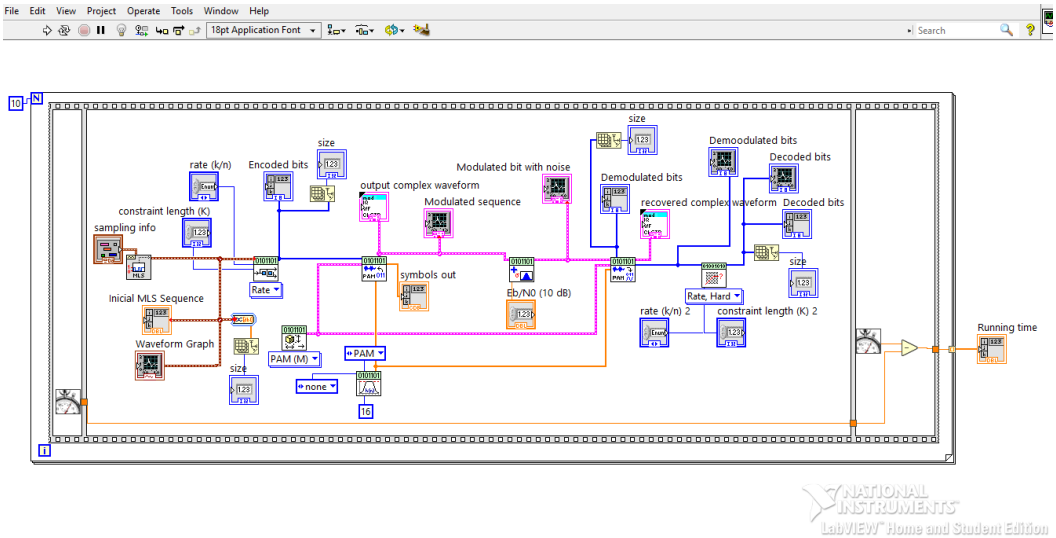

Figure A.1: Block code for LabVIEW.

Once we have the code ready, we stablish a protocol for simulations and data collect. This protocol basically consists on doing each simulation case separately, iterate 10 times in order to obtain 10 process time values and after finishing every test in each program, restart the computer to have similar conditions in every simulation.

With all the data collected we could start to analyse the results from all the study cases. In each computation test we had different results, but in the end, they all show a kind of pattern.

In the conclusions chapter it is widely explained, so I will resume the main conclusions here.

MATLAB had obtained a better performance in numerical results, because the time of execution from every simulation was better, except from the convolution, that was really optimized for LabVIEW.

Moreover, MATLAB syntax is very affordable, and has a huge variety of functions for any kind of application, what make it so easy to write code if you are a bit familiar with this tool.

On the other hand, LabVIEW provide a really interesting environment based on graphical coding interface, that make it also very comfortable and visual for users that may not be used to work with programming tools.

The next step after setting these conclusions was to explore future investigation steps. First, I would say that it is possible to go deeper in this comparation, proposing new test and focusing in other areas not necessarily related to signal communications.

It also seems reasonable to make comparations with other common languages and programming tools. I have highlighted Python in particular, because in my research, I found that many companies and publications consider it as one of the best and more popular in the recent years. And has been indicated as one of the present and future programming languages for IoT (Internet of Things), that will be definitely a huge field in technology research.

To close this project, I have written about planification and budget. This last issue is very important every time you face a project like this, because it makes no sense to start working on something if it is not profitable.

In this case, I could download the student version of both MATLAB and LabVIEW for free using the licence that the university gives to all its students. Apart from that we have the material for simulations, in this case my personal computer, and the worker expenses, including my tutor and myself.

Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

1.1. Motivación

El software y las herramientas de programación son fundamentales en el campo académico y la investigación, además del sector privado y empresarial. Elegir de forma adecuada estas herramientas puede ser tan importante como el propio desarrollo de la solución que se persigue.

En muchas ocasiones, no tenemos la capacidad de elegir entre distintas opciones, o esta capacidad se ve limitada por razones externas. Por ese motivo, encuentro muy interesante tener la posibilidad de analizar y comparar dos herramientas que pueden servir a un mismo propósito.

Durante la formación académica estos programas ofrecen un método de enseñanza más visual y que es de gran ayuda para comprender algunos de los conocimientos teóricos que se abordan. Es por ello que, en muchas asignaturas, la realización de proyectos con estas herramientas supone una parte importante de la calificación.

MATLAB y LabVIEW son dos programas muy potentes que pueden servir para hacer operaciones algebraicas, como operaciones matriciales, convoluciones, transformadas de Fourier o todo tipo de bucles e iteraciones condicionales. Y con

todo ello, en el campo de las comunicaciones, podemos simular sistemas de comunicaciones parciales o completos y ver cómo afecta el cambio de cualquier parámetro al funcionamiento y al resultado.

1.2. Objetivos

En este trabajo se pretende establecer una amplia comparación entre MATLAB y LabVIEW, dos entornos y lenguajes de programación distintos muy utilizados en entornos como el académico, de investigación y empresarial.

Durante este proceso se pretende dar respuesta, entre otras, a las siguientes preguntas: ¿Qué herramienta es más rápida para un proceso concreto? ¿Cuál tiene un diseño más sencillo? ¿Es posible desarrollar las mismas funcionalidades en ambas? ¿Son adecuadas para todos los usuarios? ¿Cuál es la más potente? ¿Cuál es mejor para una determinada tarea?

Abordaremos estas preguntas con la idea de facilitar la elección a los futuros usuarios, puesto que esta elección puede ser muy relevante a la hora de realizar sus proyectos o investigaciones. Además, se pretende entender el alcance de estas herramientas en el campo de las comunicaciones, y su versatilidad en las tareas de simulación.

Para conseguir estas respuestas, se van a analizar ambas herramientas en términos de eficiencia, rapidez, dificultad de uso, disponibilidad económica y versatilidad. Tras ello, se elaborarán unas conclusiones que sirvan como cierre de este estudio.

1.3. Estructura de la memoria

Este proyecto se organizará en capítulos de la siguiente manera:

- Capítulo 2: Estado del arte. En este capítulo se estudian y se analizan todos los agentes que intervienen en el desarrollo del trabajo, así como los estudios previos relacionados que existan.

- Capítulo 3: Definición de las pruebas. En este apartado se explican y detallan las pruebas y simulaciones que se van a llevar a cabo con las dos herramientas estudiadas.

- Capítulo 4: Análisis y resultados. Aquí se expondrán los resultados obtenidos durante el proceso de ejecución de las pruebas y se analizarán las causas y efectos de los mismos.

- Capítulo 5: Conclusión. En este capítulo se elaboran unas conclusiones, basadas en el comportamiento de ambos programas durante la realización del proyecto y sus resultados. También se expondrán las posibles líneas futuras de investigación.

- Capítulo 6: Planificación del trabajo y presupuesto. En este apartado se muestra la planificación mediante la cual se ha llevado a cabo el proyecto, y cuál es el presupuesto necesario para hacer frente a su implementación.

Capítulo 2: Estado del arte

2.1. MATLAB

MATLAB (MATrix LABoratory) es un lenguaje y entorno de programación desarrollado por la empresa MathWorks. En 1984 su cofundador, el matemático Cleve Moler sacó la primera versión de esta herramienta, si bien la creación del lenguaje se inicia a finales de los 70.

Este sofware está dirigido al cálculo matricial, la representación de funciones y datos, en forma de gráficos o tablas. A pesar de ser esta su idea inicial, actualmente cuenta con una funcionalidad muy amplia gracias a sus 'toolboxes' (cajas de herramientas). De esta manera, en la actualidad, la versatilidad de este programa va mucho más allá y lo convierte en ideal para cualquier entorno académico, de investigación o empresarial.

Se puede utilizar en las plataformas Windows, Linux, MacOS y Unix. Los archivos, tipo script, se generan con una extensión .m propia de MATLAB y pueden ejecutarse tanto en el entorno de la herramienta, como de forma independiente.

2.1.1. Simulink

Dentro del entorno de programación MATLAB, Simulink es una herramienta que nos permite un tipo de programación visual, basada en la utilización de diagramas de bloques. Es un tipo de programación a más alto nivel, que permite generar una interfaz gráfica de forma sencilla y accesible para el usuario.

Los archivos generados con Simulink tienen la extensión .mdl, y en ellos se pueden integrar otros archivos de MATLAB. Generar de forma automática código escrito es otra de las ventajas de este desarrollo, pudiendo obtener la transcripción del código en C o HDL, para luego introducirlo en otro dispositivo.

2.2. LabView

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) es un entorno de programación gráfica basado en un lenguaje propio denominado G. National Instruments creó este programa para ser utilizado en máquinas MAC, y en 1986 salió su primera versión al mercado. En 1992 incorporaron las versiones en Windows y Solaris, y en 1999 la primera versión en Linux, coincidiendo con la primera versión de LabVIEW RT (Real Time)[1].

El objetivo de este software es facilitar la creación de aplicaciones, llamadas Instrumentos Virtuales (VI), con las que poder obtener, medir o representar datos. Estos instrumentos, tienen dos partes separadas y relacionadas entre sí:

- Panel Frontal: Esta es la interfaz con la que el usuario se comunica con la aplicación, en ella inserta los controles de entrada o salida (llamados indicadores) de datos. Estos controles se pueden modificar durante la ejecución del programa, y observar los cambios que ello genere en el programa.
- Diagrama de bloques: Aquí se define la funcionalidad de la aplicación, mediante la unión de los bloques y la inserción de los módulos con una funcionalidad concreta (por ejemplo, un bucle for). Con esta parte del programa no podremos interactuar durante la ejecución del programa, pero si observar el transcurso de la ejecución.

Actualmente, las últimas versiones de LabVIEW, como la de 2019 que hemos utilizado en este proyecto, permiten insertar un VI dentro de otro. De esta manera, se pueden generar aplicaciones más complejas y similares a las que te permiten otros lenguajes y entornos. La extensión de estas aplicaciones es .vi.

[1] https://es.wikipedia.org/wiki/LabVIEW#Historial

Uno de los principales atractivos de esta herramienta, es la posibilidad de programar aplicaciones de una complejidad notable, sin tener conocimiento extenso de programación. Esto acerca la posibilidad de desarrollar a otros profesionales de sectores ajenos a la informática.

2.3. Estudios previos

En el momento de afrontar el estudio de este trabajo, el primer paso fue la búsqueda de estudios similares. Proyectos o artículos que comparasen el rendimiento y las prestaciones de ambos programas, MATLAB y LabVIEW.

Como hemos mencionado anteriormente, ambos programas tienen una larga trayectoria como herramientas de cálculo matemático, simulación y procesamiento de datos. Primera versión de MATLAB en 1984 y LabVIEW en 1986. Durante estos 30 años la comunidad de investigadores, docentes y estudiantes han utilizado ambas herramientas para desarrollar diferentes soluciones que les permitiesen obtener y tratar de forma eficiente los datos.

En cuanto empezamos la búsqueda comprobamos que no somos los primeros en establecer una comparación entre estos dos programas.

A continuación, se van a exponer algunos de los estudios que se han encontrado.

2.3.1. Clasificación y listados de los principales lenguajes de programación

En 2018, la revista IEEE spectrum, elaboró por quinto año un ranking de popularidad de los principales lenguajes de programación, siguiendo un proceso que detallan en este otro artículo[2]. Desarrollaron una aplicación[3] que permite al usuario escoger los criterios para ordenar esta lista de 48 lenguajes, además de establecer 4 categorías: web, móvil, empresarial y sistemas embebidos.

El gran ganador en esta comparativa fue Python, un lenguaje del que hablaremos más adelante. Siendo el primero en todas las categorías en las que está incluido.

En cuanto a MATLAB y LabVIEW, obtuvieron las posiciones 11 y 35, respectivamente. Figura 2.1 y 2.2.

[2] https://spectrum.ieee.org/static/ieee-top-programming-languages-2018-methods
[3] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

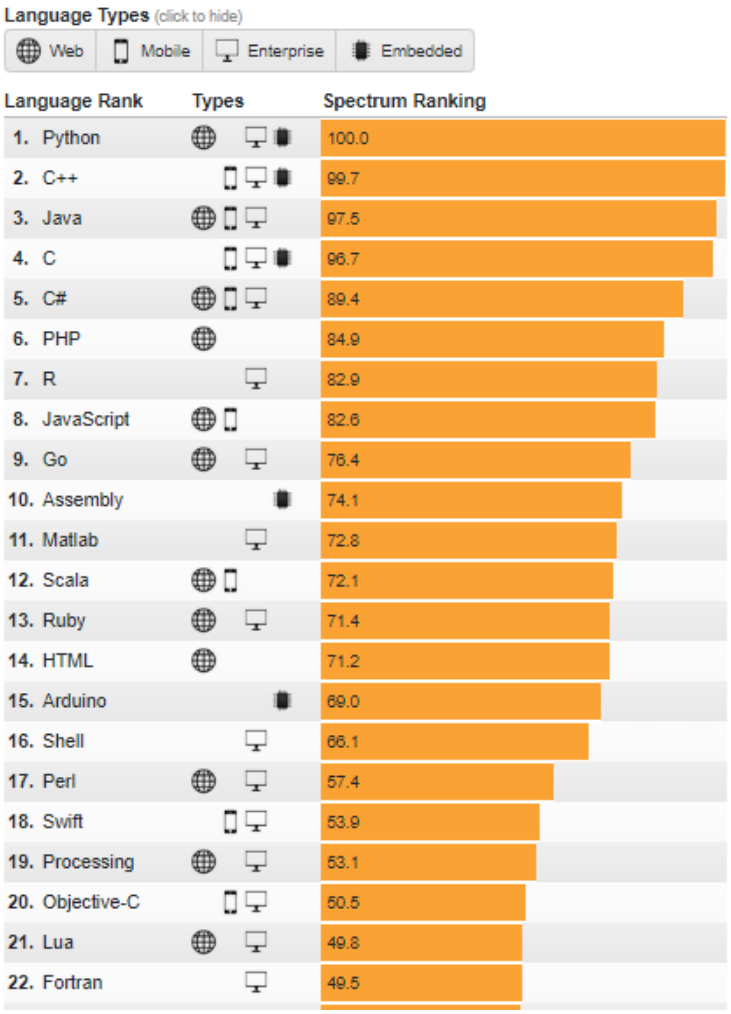

Figura 2.1: Ranking de lenguajes de programación 2018, según IEEE.

Mientras MATLAB se encuentra en las primeras posiciones, vemos que LabView no cuenta con la misma popularidad según los criterios utilizados por IEEE.

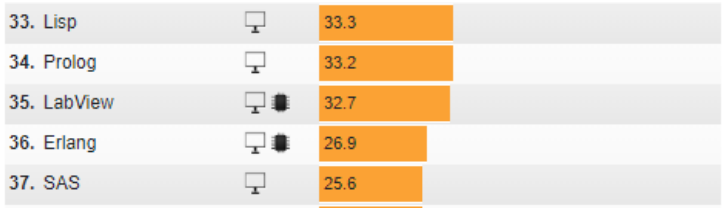

Figura 2.2: Posición de LabVIEW.

Lógicamente, el IEEE no es la única entidad que elabora estas listas, hay otras, como por ejemplo la empresa TIOBE (The Importance Of Being Earnest) que en agosto de 2019 actualizó su índice de popularidad de los principales lenguajes de programación.

Esta empresa se dedica al análisis y estudio de la calidad del código que sus clientes les envían[4]. Como en el caso de IEEE también utilizan sus propios criterios para ordenar esta lista[5].

Java es el lenguaje que se mantiene como favorito según podemos observar en la Figura 2.3, aunque más adelante en el artículo[6] encontramos que Python fue, teniendo en cuenta el crecimiento, considerado el mejor lenguaje de programación de 2018, lo cual revela una cierta tendencia común en ambas publicaciones.

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	**2**		**C**	**15.154%**	**+0.19%**
3	4	⌃	Python	10.020%	+3.03%
4	3	⌄	C++	6.057%	-1.41%
5	6	⌃	C#	3.842%	+0.30%
6	5	⌄	Visual Basic .NET	3.695%	-1.07%
7	8	⌃	JavaScript	2.258%	-0.15%
8	7	⌄	PHP	2.075%	-0.85%
9	14	⌃⌃	Objective-C	1.690%	+0.33%
10	9	⌄	SQL	1.625%	-0.69%
11	15	⌃⌃	Ruby	1.316%	+0.13%
12	13	⌃	MATLAB	1.274%	-0.09%
13	44	⌃⌃	Groovy	1.225%	+1.04%
14	12	⌄	Delphi/Object Pascal	1.194%	-0.18%
15	10	⌄⌄	Assembly language	1.114%	-0.30%

Figura 2.3: Índice TIOBE.

En esta ocasión, MATLAB obtiene una posición 12, muy similar al caso previo. Mientras que LabVIEW no consigue posicionarse entre los 50 mejores, y queda relegado a una posición entre el 50 y el 100.

Hay multitud de criterios que definen la forma de elaborar estas clasificaciones y es inevitable establecer un punto de vista, aún tratando de alcanzar la máxima objetividad. Buscando un poco más, encontramos una encuesta de 2015 que realizaron las webs EE Times[7] (Electronic Engineering Times) y EDN Network[8], en la que preguntaron a los ingenieros de su comunidad que estaban trabajando con pruebas automatizadas, sobre los lenguajes que ellos utilizaban.

[4] Más información en www.tiobe.com
[5] https://www.tiobe.com/tiobe-index/programming-languages-definition/
[6] https://www.tiobe.com/tiobe-index/
[7] https://www.eetimes.com/
[8] https://www.edn.com/

En este caso los resultados fueron muy diferentes, y LabVIEW se colocó en primer lugar, seguido de cerca por C y C++ y dejando atrás a MATLAB, que se colocó en quinta posición.

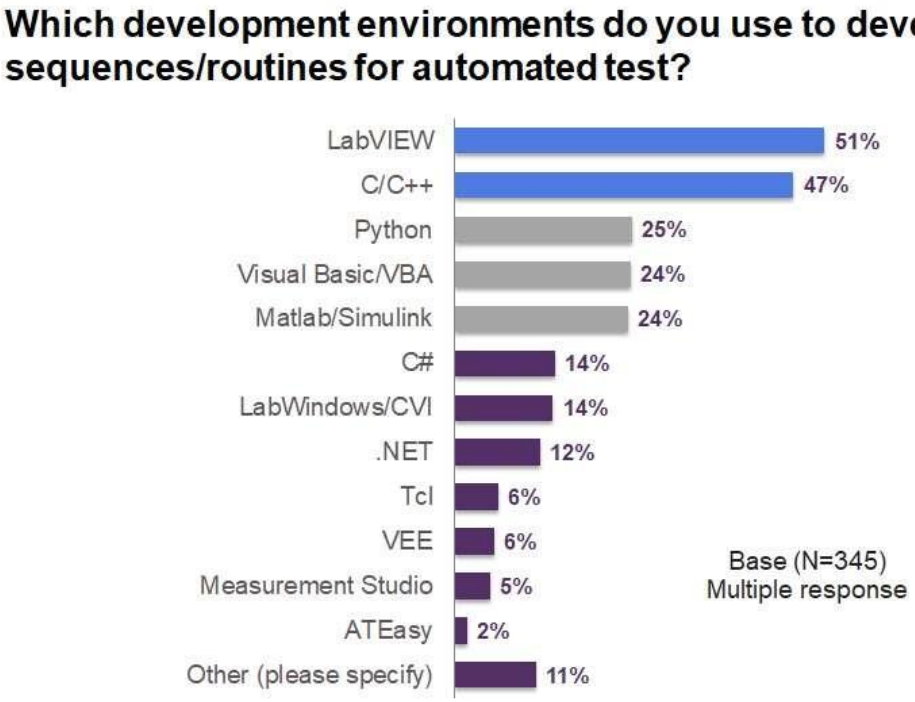

Figura 2.4: Encuesta de EE Times y EDN Network.

2.3.2. Comparaciones entre MATLAB y LabVIEW

Además de las amplias comparaciones entre los distintos tipos de lenguajes y herramientas de programación, también se han encontrado artículos y trabajos que hacen referencia directamente a la comparación explicita entre MATLAB y LabVIEW.

Son también notables las entradas en blogs y foros donde se plantea la cuestión de qué programa escoger para un determinado trabajo o se establecen comparaciones de uso, características y eficiencia.

En definitiva, todos coinciden en los siguientes puntos:

- MATLAB utiliza un lenguaje de programación tradicional, basado en líneas de código escrito para formar funciones y estructuras más complejas. LabVIEW, sin embargo, utiliza un lenguaje de programación gráfico, denominado G, basado en diagramas de bloques.
- Ambos soportan Windows, Linux y macOS, y LabVIEW además soporta multiplataforma.
- MATLAB es principalmente un entorno de cálculo matemático y algebraico. LabVIEW es un entorno de diseño, para adquisición de

datos, pruebas de automatización, control y configuración de hardware.

- En cuanto a su uso, MATLAB se orienta hacia las simulaciones gracias a la gran variedad de funciones de alto nivel disponibles en sus librerías adicionales que facilitan esta tarea. Mientras LabVIEW saca gran partido de su interfaz gráfica muy intuitiva que facilita la interacción con el hardware, incluyendo el propio de NI.

Por último, en este apartado, destacaremos un artículo encontrado en el catálogo de la biblioteca de la Universidad Carlos III. "Comparison of LabView and MATLAB for scientific research"[9] hace un estudio similar al que se va a encarar en este proyecto, por lo que sirve de referencia para encontrar posibles casos de estudio, mejoras y defectos que hagan este proyecto más completo.

Tras leer detenidamente este artículo, encontramos que establece una comparación cualitativa y cuantitativa de ambos programas, basándose en 4 supuestos; cálculo matricial, transformaciones de Fourier, funciones de transferencia con el diagrama de Bode y la simulación de un motor DC.

En este proyecto se van a realizar algunas de estas pruebas, como es el caso de las dos primeras, y además otras relacionadas con los entornos de comunicaciones como son; codificación y decodificación de señales, convoluciones de señales y la simulación de un sistema de transmisión/recepción.

Se pretende también ampliar los casos sencillos como bucles, representaciones gráficas y funciones condicionales. Todo ello, siempre enmarcado en el entorno de las comunicaciones.

[9] T. Tasner, D. Lovrec, F. Tasner y J. Edler. "Comparison of LabView and Matlab for scientific research", Annals of the Faculty of Engineering Hunedoara, vol. 10, no.3, pp. 389-394, 2012

Capítulo 3: Definición de las pruebas

En este capítulo explicaremos en qué consisten cada una de las simulaciones que hemos utilizado para medir el rendimiento de ambos entornos; MATLAB y LabVIEW.

El procedimiento a seguir para la realización de las pruebas ha sido el siguiente:

- En primer lugar, definir de forma teórica, los escenarios en los que queríamos examinar el comportamiento de los programas.
- Escribir el código correspondiente a cada una de las simulaciones, tratando de asemejarlos al máximo en ambas herramientas, teniendo en cuenta sus diferentes lenguajes.
- Ejecutar las simulaciones y almacenar los datos obtenidos; el tiempo de proceso y las gráficas correspondientes.
- Observar las diferentes posibilidades de procesamiento y memoria que ofrecen cada programa.
- Reunir y exponer los datos para obtener las correspondientes conclusiones.

Finalmente, se ha optado por realizar 4 simulaciones diferentes y orientadas al campo de las señales y sistemas de comunicación.

3.1. Equipo y software empleados

3.1.1. Software

Para este proyecto se han utilizado las siguientes versiones de software:

- NI LabVIEW 2019 (32 bit).
- MATLAB R2016a.

Ambos programas se han instalado con la licencia de estudiante facilitada por la Universidad Carlos III de Madrid por pertenecer a su comunidad académica. No siendo necesaria la compra de ninguna licencia por parte del alumno.

3.1.2. Equipo

Para la realización total de este trabajo, se ha utilizado el ordenador personal del alumno; búsqueda de información, programación y simulación de las pruebas y redacción del documento.

Este equipo tiene las siguientes características:

Marca	Lenovo
Modelo	Flex 2 (14 inch) Laptop
Procesador	Intel Core i3-4030 CPU @ 1.90GHz
RAM	4,00GB
Sistema operativo	Windows 10 Home

Tabla 3.1: Características del equipo.

3.2. Operaciones matriciales

En esta primera prueba se van a proceder a la multiplicación de dos matrices cuadradas de orden n.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} n \times n$$

Para los valores de n = 1000, 5000, 10000. Y en los siguientes supuestos:

- Matriz cuadrada compuesta por números aleatorios reales entre 0 y 1.
- Matriz cuadrada compuesta por números complejos aleatorios entre 0 + 0i y 1 + i.

Este es un supuesto básico para cualquier herramienta de cálculo algebraico, ya que las operaciones de vectores y matrices son la base de muchas de las tareas que se realizan de forma cotidiana.

3.3. Transformada rápida de Fourier

Continuamos con otra simulación básica en el procesamiento de señales, la transformada rápida de Fourier o FFT (Fast Fourier Transform). Es un algoritmo que nos permite calcular la DFT (Discrete Fourier Transform) y su inversa.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}, \qquad k = 0, \ldots, N-1.$$

Figura 3.1: Definición de la DFT.

Esta herramienta es de vital importancia en el análisis matemático por su gran variedad de posibles aplicaciones. El tratamiento digital de señales es una de estas aplicaciones, y es de gran utilidad para el estudio de sistemas de comunicaciones.

Para nuestra simulación, vamos a aplicar este algoritmo a una señal sinusoidal de frecuencia (f) 50 Hz, con una amplitud (A) de 1 y una fase inicial (φ_0) nula.

Estableceremos dos supuestos:

- En el primero calcularemos directamente la FFT de la señal y obtendremos el tiempo transcurrido desde la generación del seno hasta el final de la transformada.
- En el segundo, añadiremos previamente un ruido blanco Gaussiano a nuestra señal, estableciendo una relación señal a ruido ($\frac{Eb}{No}$) de 10dB. A continuación, calcularemos la FFT y obtendremos el tiempo transcurrido del mismo modo que en el primer supuesto.

Repetiremos la simulación en cada uno de los supuestos, variando la frecuencia de muestreo (FS) y el número de muestras o longitud de la señal (L) para los siguientes valores:

L = [100, 1000, 10000, 100000, 1000000, 10000000]

La idea es ver cómo evoluciona el rendimiento de ambas aplicaciones al aumentar el número de datos procesados.

3.4. Convolución

En nuestra siguiente prueba, realizaremos la convolución de dos funciones y obtendremos el resultado analítico y gráfico.

Una convolución es una operación matemática que transforma dos funciones *f* y *g* en una tercera función que representa la magnitud en la que se superponen *f* y una versión trasladada e invertida de *g*[10].

$$(f * g)(t) \doteq \int_{-\infty}^{\infty} f(\eta) g(t - \eta) d\eta$$

Figura 3.2: Definición de convolución.

Como muchas otras operaciones algebraicas, la convolución es muy utilizada en ingeniería, en campos como la estadística para las distribuciones de probabilidad, la óptica para las sombras, la acústica para el eco y en ingeniería eléctrica y ramas similares, donde la salida de un sistema lineal es la convolución entre la señal de entrada y la respuesta del sistema.

En este caso, para nuestro proyecto, vamos a generar dos vectores con números aleatorios entre 0 y 1 y vamos a calcular la convolución entre ambos. En este caso se tratará de una convolución discreta en el tiempo, puesto que así lo son los vectores que hemos utilizado.

$$f[m] * g[m] = \sum_{n} f[n] g[m - n]$$

Figura 3.3: Definición de convolución discreta.

De la misma forma que en las simulaciones anteriores, variaremos la longitud de los vectores progresivamente para observar los tiempos de procesamiento.

L = [1000, 10000, 100000, 1000000]

En este caso, solo hemos medido el tiempo que transcurre en el cálculo de la convolución. No se ha tenido en cuenta la generación de los vectores aleatorios para esta medida.

3.5. Sistema de transmisión y recepción

Esta es la última simulación que vamos a realizar en este proyecto, y va a consistir en la transmisión y posterior recepción de una señal digital.

En este sistema vamos a identificar los siguientes pasos o bloques:

3.5.1. Generación de una señal MLS

En primer lugar, vamos a generar una señal MLS (Maximum Length Sequence), que es una señal periódica de bits pseudo-aleatorios.

[10] Definición extraída de https://es.wikipedia.org/wiki/Convolución

3.5.2. Codificador convolucional

Codificación de la señal mediante un codificador convolucional, que genera un código para la corrección de errores con los siguientes parámetros:

- $R = \frac{1}{2}$. Ratio de bits de entrada y salida, para $k = 1$ y $n = 2$.
- $K = 3$. Longitud límite, del inglés 'Constraint length', que es el máximo número de bits que se tienen en cuenta para un bit de salida.
- $G = [5 \quad 7]$. Matriz generatriz del ejemplo utilizado.

En la Figura 3.4. mostrada a continuación se puede ver el diagrama del código convolucional utilizado en nuestra simulación.

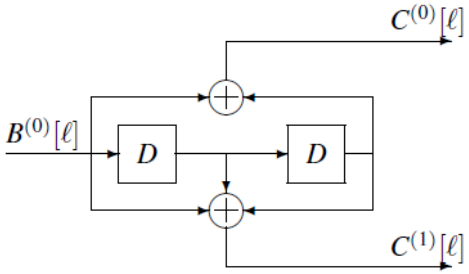

Figura 3.4: Código convolucional utilizado[11].

3.5.3. Modulador PAM

Una vez codificada la señal, procedemos a modularla mediante la modulación escogida, en este caso, una 2-PAM (Pulse Amplitud Modulation).

Esta modulación, transforma los bits de la señal codificada en uno de sus dos símbolos, -1 y +1. En la siguiente imagen, Figura 3.5., se representan las constelaciones de algunas de las modulaciones PAM más comunes.

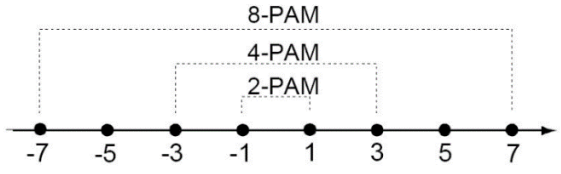

Figura 3.5: Modulaciones PAM.

Existen más tipos de modulación como; QPSK, QAM, PCM, FSK y OFDM, entre otros. Sería interesante hacer más pruebas con algunas de estas modulaciones, y queda como futura línea de investigación para este proyecto.

[11] Imagen extraída de las diapositivas de la asignatura Comunicaciones Digitales, impartida por Marcelino Lázaro, 2014. Universidad Carlos III de Madrid.

3.5.4. Canal con ruido Gaussiano

Una vez modulada la señal que estamos transmitiendo, en esta simulación vamos a establecer un canal que añada un ruido gaussiano, comúnmente denominado AWGN (Additive White Gaussian Noise), mediante el cual la señal se distorsiona según la magnitud de este ruido.

Como parámetro de relación señal a ruido, SNR (Signal to Noise Ratio) hemos establecido 10dB. Cuanto mayor sea este parámetro, mejor será la transmisión, menos ruido afectará a la señal y menos errores ocurrirán en la recepción

Este modelo de ruido trata de imitar algunas de las distorsiones aleatorias que sufren las señales durante la transmisión de forma natural.

3.5.5. Demodulador PAM

Ahora que tenemos nuestra señal distorsionada por el canal, procedemos a demodular los símbolos obtenidos para transformarlos en la secuencia de bits con la que seguiremos trabajando.

Este demodulador PAM, debe tener los mismos parámetros que el modulador que utilizamos previamente, de lo contrario no podrá descifrar la señal de forma correcta.

3.5.6. Decodificador convolucional

Llegados a este punto, tendremos una señal digital con una longitud dos veces mayor que la longitud de la señal original que generamos. Y tras la decodificación, esta longitud volverá a ser la misma.

Del mismo modo que ocurría con el demodulador, el decodificador debe también estar de acorde al tipo de codificación utilizada en el transmisor.

En este caso, para el decodificar el código convolucional utilizaremos el algoritmo de Viterbi, que se sirve del diagrama de trellis para decodificar la secuencia de bits. Viterbi ofrece dos opciones, ‘hard output’ y ‘soft output, a la hora de decidir la salida de cada bit. En la simulación se ha utilizado el ‘hard output’, que ofrece menos precisión, pero no necesita información del codificador.

El algoritmo de Viterbi se estudia en la asignatura de Comunicaciones Digitales de la Universidad Carlos III de Madrid, y en este trabajo se ha simplificado mucho su explicación, por no ser especialmente relevante para nuestro estudio, pero cabe destacar que su complejidad es mayor y su relevancia en el estudio de los sistemas digitales también.

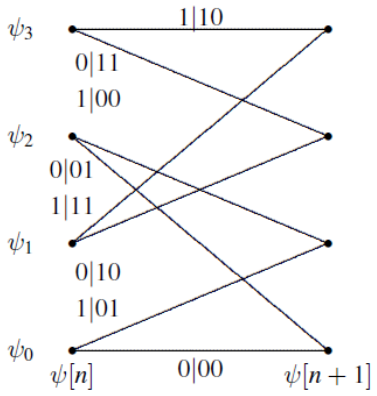

Figura 3.6: Diagrama de trellis para decodificador convolucional[12].

Una vez terminada la decodificación, obtendremos una señal digital, de longitud igual a la señal original y similitud de bits en función de los errores que hayan surgido en el proceso de recepción.

Una de las medidas más relevantes a la hora de evaluar la calidad de un sistema de comunicaciones es la tasa de error, también conocida por sus siglas en inglés: BER (Bit Error Rate). Este parámetro mide la cantidad de errores en función de la cantidad de bits transmitidos

Para deducir la BER en un sistema de comunicación sin observar directamente los bits de entrada y salida, se suele utilizar los datos de potencia de la señal y la potencia de ruido, cuanto mayor es la complejidad del sistema, más factores entran en juego en el cálculo de este parámetro.

En nuestra simulación hemos podido calcular esta tasa de forma directa, puesto que contábamos con las señales de entrada y salida, y al compararlas obteníamos directamente el resultado.

La medición del tiempo de procesado en esta última simulación ha tenido en cuenta todo el proceso, desde la generación hasta la obtención de la señal final. De la misma manera que en los supuestos anteriores, hemos variado la longitud de la señal de entrada para analizar los diferentes resultados.

L = [1000, 10000, 100000, 1000000] (bits).

[12] Imagen extraída de las diapositivas de la asignatura Comunicaciones Digitales, impartida por Marcelino Lázaro, 2014. Universidad Carlos III de Madrid.

Capítulo 4: Análisis y resultados

En este capítulo se van a exponer los resultados obtenidos en el proceso de simulación realizado en ambas herramientas, como se ha descrito en el capítulo anterior.

Antes de comenzar a analizar los resultados se va a describir brevemente el proceso de obtención de los mismos.

4.1. Medición de tiempos

Después de escribir los códigos de todas las simulaciones en los dos diferentes lenguajes, MATLAB y LabVIEW, se procede a ejecutar cada una de las simulaciones por separado.

Para ello, se enciende el ordenador, se abre el entorno correspondiente y se ejecutan los diferentes supuestos, con las variaciones pertinentes, de la misma simulación. Por ejemplo, se abre el código de MATLAB para la multiplicación de matrices y se ejecuta para los diferentes tamaños, en orden ascendente, guardando los resultados de tiempo obtenidos.

Las simulaciones se han pensado para forzar el límite de ejecución de estos programas en el equipo. De forma que algunas de ellas han provocado el bloqueo total o parcial del ordenador.

Por este motivo, después de ejecutar los supuestos de la simulación para un entorno, se cierra el programa y se reinicia el equipo para repetir el proceso de nuevo con el otro entorno. Siguiendo con el ejemplo, cerraríamos el código de multiplicación de matrices en MATLAB, reiniciamos el equipo, y a continuación abrimos y ejecutamos el código de multiplicación de matrices para LabVIEW.

De esta forma conseguimos que las condiciones iniciales del equipo al realizar las simulaciones sean lo más similares. Es conveniente añadir, que no se ha abierto ninguna otra aplicación durante la ejecución de los programas, para evitar que otros procesos disminuyeran el desempeño del equipo.

En todas las simulaciones se ha implementado un bucle que iteraba 10 veces, obteniendo un valor de tiempo en cada iteración y dando como resultado un grupo de 10 medidas de tiempo para la misma ejecución de código.

A continuación, se exponen los resultados y el análisis de cada una de las 4 simulaciones abordadas en este trabajo. Las tablas que contienen los datos comparan directamente los tiempos de ambos programas agrupados en las siguientes columnas:

- Media: Es el promedio de las 10 muestras de tiempo en cada simulación
- T. Max.: Es el mayor de los valores de las 10 muestras de tiempo.
- T. Min.: Es el menor de los valores de las 10 muestras de tiempo.
- Media - T1: Es el promedio de 9 muestras de tiempo, ignorando el valor de la primera iteración.

El motivo de este último campo es la observación de que, en muchas de las simulaciones, la primera iteración es notablemente más lenta que las sucesivas sucesiones. Esto desvirtúa en gran medida el valor promedio de las 10 muestras de tiempo. Quitando esta primera iteración, obtenemos un valor más representativo del tiempo real de procesamiento de la operación.

4.2. Multiplicación de dos matrices

Como se ha descrito anteriormente, se ha elaborado un código para multiplicar dos matrices cuadradas generadas con valores aleatorios, en el primer caso con valores reales entre 0 y 1, y en el segundo con valores complejos entre 0 + 0i y 1 + i.

4.2.1. Valores reales aleatorios

Estos son los tiempos que han tardado MATLAB y LabVIEW en generar y multiplicar dos matrices reales de tamaño n x n.

N = 1000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,148	0,535	0,087	0,105
LABVIEW	0,060	0,073	0,047	0,059

N = 5000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	6,798	7,241	6,552	6,748
LABVIEW	1,311	1,694	1,218	1,268

N = 10000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	101,62	334,24	50,382	94,435
LABVIEW	-	-	-	-

Tabla 4.1: Tiempos multiplicación de matrices reales.

En esta primera tabla, observamos que los tiempos del proceso de multiplicación de las matrices son menores en LabVIEW, sin embargo, al intentar la última operación para N = 10000 en LabVIEW, la aplicación da un fallo de memoria e impide realizar la operación.

4.2.2. Valores complejos aleatorios

Estos son los tiempos que han tardado MATLAB y LabVIEW en generar y multiplicar dos matrices complejas de tamaño n x n.

N = 1000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,553	2,179	0,327	0,372
LABVIEW	0,109	0,138	0,098	0,105

N = 5000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	27,763	42,350	24,853	27,583
LABVIEW	2,776	3,809	2,494	2,662

N = 6620	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	245,493	347,810	71,203	264,858
LABVIEW	7,135	21,706	4,570	5,516

Tabla 4.2: Tiempos multiplicación de matrices complejas.

En el caso de las matrices complejas, obtenemos tiempos notablemente más altos, y de nuevo mejores en LabVIEW. La última iteración de la simulación con matrices complejas se hizo con un tamaño N = 6620, que fue el valor que no daba ningún fallo de memoria en LabVIEW.

En esta ocasión, MATLAB tampoco logró hacer la simulación con N = 10000, dejando el equipo bloqueado antes de terminar la ejecución.

4.3. Transformada rápida de Fourier

A continuación, se muestran los resultados de la realización de una FFT de una señal sinusoidal descrita en el capítulo anterior. En esta simulación también habrá dos supuestos.

En el primero, se hará la transformada únicamente de la señal sinusoidal.

En el segundo, se añadirá un ruido Gaussiano antes de calcular la transformada.

En este caso, se ha incrementado el número de muestras en la señal. Se ha indicado en las sucesivas tablas con el valor de N.

4.3.1. Señal sinusoidal sin ruido

N = 100	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	0,528	2,4	0,042	0,505
LABVIEW	2,281	21,219	0,162	0,176

N = 1000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	8,739	81,4	0,118	0,666
LABVIEW	2,796	25,8826	0,204	0,230

N = 10000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	1,594	4,7	0,592	1,509
LABVIEW	3,256	20,111	1,163	1,383

N = 100000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	6,750	19,7	3,5	5,311
LABVIEW	14,633	31,456	12,251	12,764

N = 1000000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	80,680	110,1	58	77,411
LABVIEW	167,587	228,618	155,403	160,805

N = 10000000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	951,16	1040	911,1	941,29
LABVIEW	1.817,9	2231,4	1687,9	1.772

Tabla 4.3: Tiempos de FFT sin ruido.

En la tabla 4.3, podemos observar que, en las tres primeras simulaciones, los valores de tiempo de LabVIEW son mejores si no tenemos en cuenta la primera iteración (columna “Media - T1”), aunque el tiempo total de la simulación es casi siempre mejor en MATLAB, especialmente con los tres valores mayores de N, donde los tiempos son aproximadamente la mitad.

Además, se observa que, para esta operación en concreto, en todas las simulaciones que se han hecho con LabVIEW, el tiempo de la primera iteración ha sido el mayor de los 10 valores obtenidos. En MATLAB el comportamiento ha sido similar.

Queda reflejado que la tendencia es favorable a MATLAB y que sus tiempos de ejecución son mejores.

4.3.2. Señal sinusoidal con ruido Gaussiano

N = 100	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	0,228	0,464	0,184	0,202
LABVIEW	1,124	9,958	0,130	0,143

N = 1000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	1,524	4,5	0,3971	1,193
LABVIEW	2,326	20,577	0,281	0,298

N = 10000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	2,149	7,5	0,814	1,555
LABVIEW	5,011	29,622	1,9982	2,276

N = 100000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	7,29	8,7	6	7,344
LABVIEW	23,146	44,804	20,139	20,740

N = 1000000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	97,81	109,4	90,6	96,856
LABVIEW	250,084	307,22	235,161	243,736

N = 10000000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	1.213,6	1307,8	1131,3	1.203,2
LABVIEW	-	-	-	-

Tabla 4.4: Tiempos de FFT con ruido.

En esta simulación, donde hemos incluido un ruido Gaussiano en la señal antes de calcular la FFT, no se aprecian grandes cambios para los 3 primeros valores de N (100, 1000 y 10000).

En cambio, a medida que aumentamos la longitud de la señal, la diferencia se va haciendo más notable, especialmente en LabVIEW, donde la última iteración con N = 10000000, no se puede llevar a cabo por falta de memoria en el programa.

Como sucedía en el supuesto anterior, el desempeño de MATLAB también es superior.

4.4. Convolución de vectores

En este apartado se van a mostrar los datos de tiempo de ejecución de una operación de convolución de dos vectores aleatorios de tamaño L y valores entre 0 y 1.

Como se dijo en el capítulo anterior, no se tendrá en cuenta el tiempo de generación de los vectores, solo el de la propia convolución.

L = 1000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	1,012	4,1	0,408	0,668
LABVIEW	0,571	1,680	0,185	0,540

L = 10000	*Media (ms)*	*T. Max. (ms)*	*T. Min. (ms)*	*Media -T1 (ms)*
MATLAB	21,02	26,2	17,1	21,078
LABVIEW	4,435	5,906	4,061	4,272

L = 100000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	1,908	2,117	1,758	1,885
LABVIEW	0,061	0,085	0,048	0,0599

L = 1000000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	365,067	386,87	349,38	366,475
LABVIEW	0,561	0,615	0,521	0,561

Tabla 4.5: Tiempos de convolución.

Antes de analizar los datos obtenidos, es importante señalar que se encontraron dos funciones (bloques) en LabVIEW para llevar a cabo una convolución. Después de probar se optó por la más rápida, que además permite otras operaciones como la correlación. Ambas se encuentran en la paleta de operación de señales.

Una vez elegidas las funciones pertinentes, podemos observar que los tiempos de LabVIEW para la convolución son mucho más rápidos que los de MATLAB, especialmente cuando incrementamos la longitud de los vectores.

Por ese motivo los datos de las dos últimas iteraciones están en segundos (s) y no milisegundos (ms).

Queda patente que el desempeño de LabVIEW para esta operación es mejor que el de MATLAB.

4.5. Sistema de transmisión y recepción de una señal digital

En este último apartado, vamos a exponer los resultados obtenidos para los tiempos de simulación de un sistema de transmisión y recepción completo. Como en otros apartados, hemos dividido los resultados en dos supuestos.

En el primer supuesto vamos a calcular el tiempo transcurrido en la completa transmisión y recepción de la señal digital de longitud L, sin generar las gráficas de la señal en los distintos estados del sistema.

En el segundo supuesto, sí se incluirá en el tiempo total la generación de estas gráficas.

4.5.1. Simulación sin gráficas

L = 1000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,0269	0,1522	0,002	0,013
LABVIEW	0,0251	0,0326	0,0232402	0,0243

L = 10000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,0328	0,181	0,0066	0,0163
LABVIEW	0,261	0,291	0,251	0,258

L = 100000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,093	0,267	0,0589	0,0736
LABVIEW	2,692	2,827	2,594	2,677

L = 1000000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,645	0,799	0,592	0,628
LABVIEW	-	-	-	-

Tabla 4.6: Tiempos de transmisión y recepción sin gráficas.

En este primer supuesto, el análisis es claro, cuando generamos una señal de longitud L = 1000, los tiempos son muy similares, y a medida que aumentamos la longitud, el desempeño de LabVIEW empeora notablemente, hasta el punto de quedarse sin memoria en la última simulación de L = 1000000.

MATLAB obtiene unos resultados superiores y que aumentan paulatinamente con el incremento de la longitud de la señal como cabría esperar.

4.5.2. Simulación con gráficas

L = 1000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,129	0,421	0,0739	0,0962
LABVIEW	0,0326	0,0429	0,0269	0,0315

L = 10000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,162	0,576	0,0809	0,116
LABVIEW	0,283	0,330	0,271	0,278

L = 100000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,231	0,644	0,146	0,186
LABVIEW	2,862	3,328	2,695	2,811

L = 1000000	*Media (s)*	*T. Max. (s)*	*T. Min. (s)*	*Media -T1 (s)*
MATLAB	0,908	1,299	0,813	0,865
LABVIEW	-	-	-	-

Tabla 4.7: Tiempos de transmisión y recepción con gráficas.

Para terminar con este capítulo de análisis de resultados, tenemos los datos de la última simulación, un sistema completo de transmisión y recepción donde se han incluido hasta 5 gráficas mostrando el estado de la señal durante el proceso.

En primer lugar, hay que destacar un ligero y esperado aumento de los tiempos con respecto al primer supuesto, y un rendimiento similar al que ambos programas obtuvieron anteriormente

De nuevo LabVIEW no completa la última simulación por falta de memoria, y MATLAB se mantiene estable y con tiempos muy asumibles para este tipo de proceso.

Capítulo 5: Conclusiones

Con las simulaciones realizadas y los datos obtenidos y analizados, es el momento poner en la balanza las fortalezas y debilidades de los dos programas que hemos analizado a lo largo de este proyecto.

Dividiremos este capítulo en dos partes, la primera en la que se van a exponer las conclusiones a las que se ha llegado después y durante el proceso, y la segunda en la que se propondrán futuras líneas de investigación.

5.1. Conclusiones

El objetivo de este trabajo ha sido comparar las herramientas de trabajo MATLAB y LabVIEW en cuanto a su rendimiento y prestaciones.

Personalmente ha supuesto un reto, puesto que mi experiencia con LabVIEW era nula, hasta el punto de desconocer la existencia del programa, y con MATLAB estaba muy enfocado a las prácticas realizadas durante el grado, no siempre relacionadas con este tipo de simulaciones.

Una vez familiarizado con ambos entornos, muy similares en cuanto a posibilidades, pero realmente diferentes en su uso y manejo, puesto que uno ofrece

un tipo de programación clásica con líneas de código, mientras el otro dispone de un lenguaje de programación gráfica mediante ‘bloques’ y ‘cables’.

Primero analizaré cada una de las 4 situaciones que hemos planteado en este trabajo.

5.1.1. Multiplicación de matrices

En esta primera simulación, los resultados son claros. El proceso está mucho más optimizado en LabVIEW, y por ello los tiempos son menores. A pesar de ello, si se trabaja con matrices de gran tamaño, puede existir un fallo de memoria que no nos permita seguir trabajando.

En cuanto a la dificultad a la hora de implementar el código necesario para esta simulación, el resultado es el contrario. Generar dos matrices con números aleatorios en MATLAB se hace con una simple función ‘rand’ para números reales y ‘complex’ para números complejos. En LabVIEW es necesario calcular un numero aleatorio insertado en dos bucles para obtener una matriz aleatoria real, e introducir más variables para hacerla con números complejos.

En definitiva, y pese a obtener mejores resultados en las simulaciones, mi recomendación para la persona que quisiese trabajar con vectores y matrices, es que utilizase MATLAB. Es más sencillo, y dispone de una mayor variedad de funciones.

5.1.2. Transformada rápida de Fourier

Los resultados del cálculo de la transformada de Fourier también destacan cualidades positivas de ambos entornos.

Los tiempos de ejecución en MATLAB son a la larga mejores, y es notable el valor tan alto del tiempo de la primera ejecución en LabVIEW, que sería el tiempo al que daríamos valor si solo tenemos intención de ejecutar la transformada una vez.

En este caso, el código necesario en ambos lenguajes es muy accesible para cualquier nivel de programador. Una persona sin conocimiento previo podría hallar la solución para esta operación, puesto que ambos programas disponen de una función concreta para esta tarea.

De nuevo mi recomendación sería usar MATLAB, aunque el desempeño de LabVIEW es realmente bueno, y cabe destacar que la herramienta de generación de señales que ofrece es potente, visual y muy versátil.

5.1.3. Convolución

Llegado el momento de hacer operaciones de convolución con señales, parece cambiar la tendencia, y en los resultados observamos que el rendimiento de LabVIEW es muy superior.

Dado que ambos programas disponen una o más funciones para realizar esta operación, no puedo valorar que el uso de uno sea más sencillo que otro.

En este caso, para el usuario que desee hacer operaciones con señales del tipo convolución, mi recomendación sería LabVIEW, puesto que, además, su interfaz gráfica es sencilla, y eso puede ser de gran relevancia para el trabajo con señales.

5.1.4. Sistema de transmisión y recepción

En el último caso, los resultados vuelven a ser claros y evidencian uno de los mayores problemas que se han observado en LabVIEW, la falta de memoria en el espacio del disco designado para el programa.

Los tiempos, en general, son mejores en MATLAB, y la implementación del código es mucho más sencilla, a pesar de que en LabVIEW existen y están disponibles todas las funciones necesarias para el desarrollo del sistema de comunicaciones.

Mi recomendación para simular una comunicación de este tipo a nivel teórico es usar MATLAB, que permite desarrollar un código sencillo de forma rápida para un usuario que esté familiarizado con el tratamiento digital de señales.

Como balance positivo de LabVIEW volvería a destacar la facilidad para interactuar gráficamente con la aplicación que desarrollemos y los resultados visuales que se obtienen.

Para terminar esta reflexión, quiero señalar algunas de las sensaciones que me han trasmitido estas dos herramientas en el desarrollo del todo el trabajo.

Si no tienes un conocimiento previo de algún tipo de programación clásica, como podría ser C o Java, o cualquier otra de las muchas que existen, LabVIEW puede ser una opción interesante.

Trabajar con una herramienta gráfica como esta, te permite ver al instante qué es lo que estás desarrollando, como va a funcionar, y qué puedes esperar como resultado. A este nivel de inicio, podría considerarse divertido hacer pequeñas aplicaciones que te sirvan para entender cómo funciona, y de esta forma animar al usuario a seguir trabajando con ello.

Creo que es un programa ideal para el entorno académico, puesto que muchas de las prácticas que se realizan en la universidad, no tienen como fin la programación en sí, si no la comprensión de algún concepto estudiado que podamos visualizar en herramientas de este tipo.

Además, añadiría que National Instrumens, propietarios de LabVIEW, ofrece una gran variedad de instrumentos que son complementarios a la herramienta, y que pueden ampliar muchísimo la funcionalidad de esta. Un ejemplo podría ser un generador de señales o un microchip, que se comunican de forma directa con el programa, y que tienen ya desarrolladas funcionalidades propias de la compañía.

Trabajar con MATLAB de nuevo ha sido gratificante, puesto que el nivel que he alcanzado estos años es mayor del que creía.

Es una herramienta muy potente y muy usada en todo el mundo. Esto último facilita enormemente cualquier tarea que te propongas realizar con este lenguaje, puesto que buscar información sobre funciones, variables o algoritmos es tan sencillo como buscar en la propia página de MathWorks[13] o en algún foro de programación.

Esta comunidad tan grande de usuarios es muy útil también, para encontrar ejemplos de prácticamente todo, incluso tutoriales detallados del uso del desarrollo de alguna tarea.

Si necesitase emplear un programa de cálculo algebraico para un proyecto o aplicación, sin duda MATLAB sería mi elección.

5.2. Líneas futuras de investigación

Durante la realización de cualquier proyecto, van surgiendo diferentes dudas e intereses que se distancian de la línea marcada por la persona responsable del trabajo, pero que son igualmente interesantes.

Este apartado permite exponer de forma breve esas ideas, en forma de posibles proyectos futuros que resulten de interés o valor añadido.

Siguiendo en la línea del proyecto que hemos realizado, es evidente la multitud de posibilidades que han quedado sin analizar. Sería interesante estudiar y definir otras pruebas y simulaciones que permitiesen estudiar otras funcionalidades de estas herramientas. Como he comentado en el apartado anterior, puede ser revelador un estudio del rendimiento de ambos programas con instrumentos externos y su interacción con cada uno de los entornos.

Continuando con la comparación de software, una idea que se me vino a la cabeza al poco de empezar a estudiar para la realización de este proyecto, es comparar otros lenguajes de programación populares como Python.

Menciono Python en particular, porque es uno de los grandes señalados como lenguaje de programación favorito o más elegidos por empresas y usuarios. En especial, todo el desarrollo en este lenguaje, que tiene que ver con el IoT (Internet of Things) que es y será sin duda, uno de los mayores desafíos a nivel tecnológico de los próximos años.

[13] https://es.mathworks.com/

Capítulo 6: Planificación del trabajo y presupuesto

En este capítulo vamos a detallar cómo se ha llevado a cabo este trabajo y cuáles son los principales factores económicos que afectan a su presupuesto.

6.1. Planificación

Este proyecto se ha dividido en 4 fases principales a lo largo de su elaboración.

Fase 1: Estudio Previo

El primer paso en este proyecto fue el estudio y la recopilación de datos de los dos programas, además de los estudios similares que ya existiesen en la literatura.

El objetivo de este proceso era familiarizarse con ambos entornos de una forma teórica y encontrar ideas sobre cómo establecer una comparación efectiva en el ámbito de las comunicaciones, que era el deseado.

Fase 2: Análisis

Con la información suficiente para tener una visión global adecuada, empezamos la fase de análisis, donde vamos a estudiar de forma práctica cada uno de los programas, hasta conseguir un manejo suficiente.

A continuación, se definen de forma detallada las pruebas a las que se van a someter ambas herramientas y cuáles van a ser los criterios de comparación.

Fase 3: Desarrollo

En esta fase vamos a poner en práctica todo lo estudiado en las fases previas, y comenzamos la elaboración del código necesario para cada una de las simulaciones. Se solucionarán los problemas que surgen durante el proceso, redefiniéndose en algún caso parte de las pruebas.

Es de vital importancia para el objetivo del trabajo y de ella dependerá en gran medida el éxito o fracaso de este.

Fase 4: Memoria

Durante las tres primeras fases se irá realizando un esquema de la estructura de la memoria y se tomarán notas de todo lo que se desee incluir.

Con la fase de desarrollo ya en marcha, comienza esta última en la que se tratará de dejar por escrito todos y cada uno de los procesos seguidos, con el único objetivo de dar respuesta a las preguntas que se plantearon al inicio de este proyecto.

Este último escalón será el que más tiempo y dedicación exija al autor de este trabajo.

6.1.1. Diagrama de Gantt

A continuación, se va a mostrar en la figura 6.1 el diagrama de Gantt de este proyecto. Esta herramienta permite entender de una forma visual el proceso y la escala de tiempos de cada actividad en la elaboración de un proyecto.

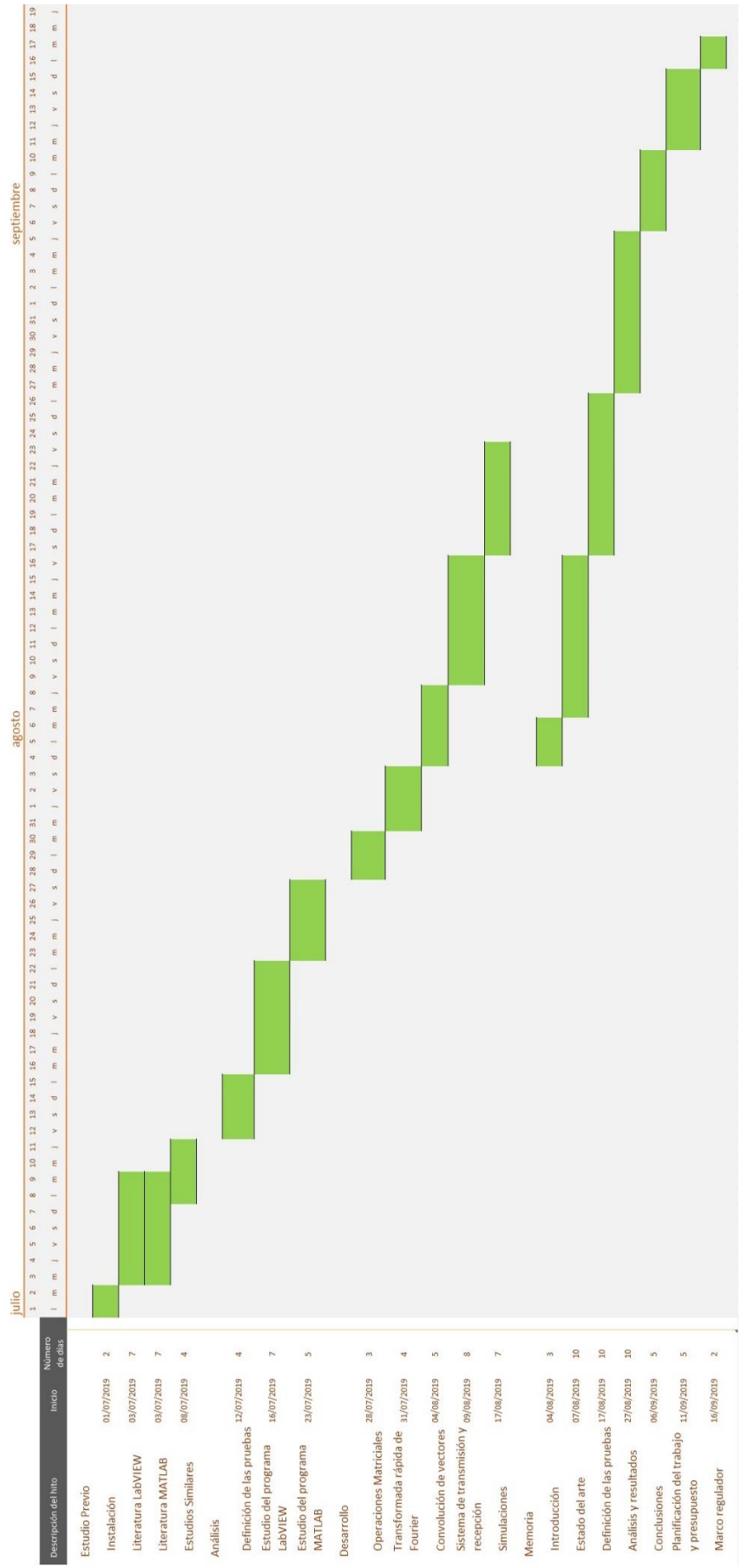

Figura 6.1: Diagrama de Gant.

6.2. Presupuesto

Es importante destacar el aspecto económico a la hora de realizar un trabajo puesto que su viabilidad depende de ello.

En el caso de nuestro trabajo, este análisis es sencillo, puesto que no conlleva la fabricación de ningún elemento físico. Estos serían los costes que afectan al proyecto.

6.2.1. Coste de personal

Este trabajo ha sido realizado un alumno de ingeniería de sistemas de comunicaciones y revisado por su tutor, profesor de la Escuela Politécnica de la Universidad Carlos III de Madrid.

La duración del proyecto ha sido de unos 80 días, con una estimación de 4 horas por día. Total, 320 horas. Por parte del tutor se estiman unas 30 horas entre tutorías y revisión por separado.

Persona	Cargo	Salario anual	Precio hora	Tiempo	Coste total
Ósca Serrano García	Autor	25.000€	14,40€/hora	320	4608€
Víctor P. Gil Jiménez	Tutor	60.000€	34,56€/hora	30	1036,8€

Tabla 6.1: Coste de personal.

6.2.2. Coste del material

En este apartado incluiremos el coste del equipo con el que se ha trabajado, puesto que el coste de las licencias lo ha asumido la universidad.

Para calcular este coste, coste imputable, es necesario establecer el precio del equipo, su vida útil y el tiempo de utilización para este proyecto.

Equipo	Coste	Vida útil	Tiempo de uso	Coste imputable
Lenovo Flex 2 (14 inch)	500 €	60 meses	3 meses	25€

Tabla 6.2: Coste del material.

6.2.3. Coste total

Teniendo en cuenta estos costes. El coste total del proyecto sería la suma del coste de personal y el coste del material.

Coste de personal	Coste del material	Coste total
5644,8€	25€	5669,8€

Tabla 6.3: Coste total.

Capítulo 7: Marco regulador

En este último capítulo del trabajo se debe establecer un marco de leyes que afectan a la realización del mismo, de forma directa o indirecta.

En el caso nuestro proyecto, en el que no hemos llevado a cabo ninguna implementación que tenga un objetivo comercial o vaya a ser utilizada por terceras personas, creo que solo es relevante destacar la ley de propiedad intelectual[14].

Esta ley ha sido respetada en la medida en que ambos softwares se han descargado y utilizado con las licencias pertinentes y como han establecido las compañías propietarias de estas tecnologías.

[14] Real decreto legislativo 1/1996, Ley de Propiedad Intelectual. (BOE Nº 97, 22 abril 1996).

Bibliografía

[1] https://es.wikipedia.org/wiki/LabVIEW#Historial

[2] https://spectrum.ieee.org/static/ieee-top-programming-languages-2018-methods

[3] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

[4] Más información en www.tiobe.com

[5] https://www.tiobe.com/tiobe-index/programming-languages-definition/

[6] https://www.tiobe.com/tiobe-index/

[7] https://www.eetimes.com/

[8] https://www.edn.com/

[9] T. Tasner, D. Lovrec, F. Tasner y J. Edler. “Comparison of LabView and Matlab for scientific research”, Annals of the Faculty of Engineering Hunedoara, vol. 10, no.3, pp. 389-394, 2012

[10] Definición extraída de https://es.wikipedia.org/wiki/Convolución

[11] Imagen extraída de las diapositivas de la asignatura Comunicaciones Digitales, impartida por Marcelino Lázaro, 2014. Universidad Carlos III de Madrid.

[12] Imagen extraída de las diapositivas de la asignatura Comunicaciones Digitales, impartida por Marcelino Lázaro, 2014. Universidad Carlos III de Madrid.

[13] https://es.mathworks.com/

[14] Real decreto legislativo 1/1996, Ley de Propiedad Intelectual. (BOE Nº 97, 22 abril 1996).