Implementación de Laboratorio de Programación en Python en la Asignatura Mecánica de Sólidos

Edward Alfonso Ramírez González

Trabajo de Grado para Optar al Título de Ingeniero Civil

Director

David Sebastián Cotes Prieto

Magíster en Ingeniería Civil

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería Civil

Bucaramanga

2022

Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Pág.

Resumen

Título: Implementación de Laboratorio de Programación en Python en la Asignatura Mecánica de Sólidos[*]

Autor: Edward Alfonso Ramírez González[**]

Palabras clave: programación; pedagogía; Python; Mecánica de Sólidos; computación.

Descripción
El surgimiento del software y la programación computacional ha impactado directamente la forma en que se resuelven problemas en ingeniería civil. No obstante, actualmente no existe ninguna asignatura dentro del plan de estudios de ingeniería civil de la Universidad Industrial de Santander (UIS) que esté específicamente enfocada en desarrollar habilidades de programación. Por otra parte, diversas universidades alrededor del mundo están llevando a cabo cursos con Python para introducir la programación a estudiantes pertenecientes a carreras profesionales que no tienen relación directa con las ciencias de la computación. De manera similar, en este proyecto se implementó un laboratorio de programación usando Python en la asignatura Mecánica de Sólidos, la cual es una materia de ciclo básico profesional del cuarto semestre del programa de ingeniería civil de la UIS. Se llevó a cabo una metodología de clases sincrónicas virtuales de dos horas durante diez semanas del segundo semestre académico de 2021, donde se implementaron dos talleres para evaluar el desarrollo de las habilidades por parte de los alumnos. Además, se usaron diversos recursos de las Tecnologías de la Información y Comunicación (TIC) para orientar a los estudiantes en la aplicación de Python en la Mecánica de Sólidos. Se llevaron a cabo encuestas a mediados y al final del laboratorio con el fin de medir la satisfacción de los estudiantes. Los resultados obtenidos de estas encuestas permiten concluir que Python es un lenguaje adecuado para abordar la enseñanza del desarrollo de software en alumnos que no poseen experiencia previa en programación y que el laboratorio de Python contribuye al desarrollo de habilidades de programación en estudiantes de ingeniería civil.

[*] Trabajo de Grado
[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Civil. Director: David Sebastián Cotes Prieto, Magíster en Ingeniería Civil

Abstract

Title: Implementation of Python Programming Laboratory in the Solid Mechanics Course[*]

Author: Edward Alfonso Ramírez González[**]

Keywords: programming; pedagogy; Python; Solid Mechanics; computer science

Description
The rise of software and computer programming has directly impacted the way problems are solved in civil engineering. Currently, there is no subject within the civil engineering curriculum of the Industrial University of Santander (UIS) specifically focused on developing programming skills. On the other hand, several universities around the world are conducting courses with Python to introduce programming to students in professional careers that do not have direct relationship with computer science. Similarly, in this project, a programming laboratory was implemented using Python in the subject Mechanics of Solids, which is a basic professional cycle subject of the fourth semester of the UIS civil engineering program. A two-hour virtual synchronous class methodology was conducted over ten weeks of the second academic semester of 2021, where two activities were implemented to assess student skills development. In addition, various Information and Communication Technologies resources were used to guide students in the application of Python in Solid Mechanics. Surveys were conducted in the middle and the end of the laboratory to measure student satisfaction. The results obtained from these surveys conclude that Python is an appropriate language to address the teaching of software development in students who do not have previous programming experience, and that the Python lab contributes to the development of programming skills in civil engineering students.

[*] Degree work

[**] Faculty of Physicomechanical Engineering. School of Civil Engineering. Director: David Sebastián Cotes Prieto, Master in Civil Engineering

Introducción

Con la adopción masiva de la automatización, análisis de datos y la inteligencia artificial, el aprendizaje y la formación en habilidades de programación es imprescindible para que los estudiantes de ingeniería civil estén a la vanguardia de la tecnología y a su entorno cambiante (Wang, 2020). Actualmente, no existe ninguna asignatura dentro del plan de estudios de pregrado de ingeniería civil de la Universidad Industrial de Santander (UIS) que esté específicamente enfocada en desarrollar competencias en programación. Esto resulta problemático dado que la ingeniería civil se encuentra en un campo donde se están aplicando soluciones mediante computación numérica tanto en cursos como proyectos universitarios y profesionales (Ebrahimzadeh & Safai, 2019)

Por otra parte, diversas universidades alrededor del mundo han comprendido el rol que desempeña la habilidad de desarrollar software y están usando Python como herramienta de enseñanza en asignaturas enfocadas en programación para carreras profesionales que no tienen relación directa con las ciencias de la computación. Tal es el caso de la Universidad Técnica de Dinamarca, en la cual se implementó un curso completo de programación aplicada a la inteligencia artificial y análisis de datos para estudiantes de ingeniería usando Python (Sørensen & Nordfalk, 2020). Otra alternativa que han empleado entidades educativas como la Universidad Estatal de Arizona se basa en desarrollar competencias de programación en actividades dentro de las asignaturas ya existentes del plan de estudios (Wang, 2020). En esta universidad se implementó un módulo de introducción a la programación con Python en la asignatura *Introduction to Engineering*, que se da a estudiantes de primer semestre de ingeniería aeroespacial, química, eléctrica y mecánica. Dicho módulo consistió en el desarrollo de clases de 50 minutos durante 4

semanas. Al principio del módulo, más de un tercio de los estudiantes no tenían experiencia programando, lo cual generó problemas dado que la temática del curso tenía supuesto que los alumnos tenían un conocimiento básico de la programación. Esto, sumado al hecho de que el ritmo rápido de enseñanza condujo a la frustración de los estudiantes que no poseían conocimientos previos. No obstante, se realizaron encuestas de satisfacción al final del módulo, las cuales demostraron que la mayoría de los estudiantes percibieron la importancia de aprender Python y que el estudio de este lenguaje fue divertido y crucial para comprender los fundamentos de la programación.

Múltiples trabajos científicos y estudios técnicos recalcan la conveniencia de usar Python dentro del ámbito académico como primer lenguaje de programación (Javed, Zaman, Uddin, & Nusrat, 2019). Python es un lenguaje de programación interpretado de alto nivel creado por Guido Van Rossum en 1991 (Python Software Foundation, 2022). Actualmente es uno de los lenguajes de programación más populares y usados en la industria de la tecnología debido a su acceso libre y popularidad, con aplicaciones actuales como la inteligencia artificial y ciencia de datos (Nagpal & Gabrani, 2019). Tales características convierten a Python en uno de los lenguajes más amigables y sencillos de aprender para desarrolladores de software.

Teniendo en cuenta estas ventajas que ofrece Python y su amplio uso dentro de la academia, en este proyecto se implementó un laboratorio de programación usando Python por medio de clases sincrónicas y material asincrónico durante el segundo semestre académico de 2021 en la UIS, dentro de la asignatura Mecánica de Sólidos, la cual es una materia de ciclo básico profesional del cuarto semestre de la carrera profesional de ingeniería civil. Para contribuir al desarrollo de la competencia en programación en los estudiantes, se diseñó una metodología con base en el desarrollo de clases sincrónicas de dos horas cada semana. Las clases sincrónicas estuvieron

orientadas a la aplicación de conceptos de desarrollo de software en el ámbito de la Mecánica de Sólidos. En cuanto a calificación y notas, el laboratorio se basó en la entrega de dos talleres y la asistencia del estudiante. Los talleres emplearon casos de estudio relacionados con temas de la Mecánica de Sólidos, los cuales debían ser resueltos por parte del estudiante implementando Python. Además, se elaboró material didáctico de apoyo como guías, códigos de ejemplo, diapositivas y videos. Las plataformas en las cuales se publicó toda la información relacionada al curso fueron Google Classroom, Drive, Meet y GitHub, con el fin de reforzar el uso de las Tecnologías de la Información y Comunicación (TIC). Estas herramientas cumplen un papel importante como recurso pedagógico para facilitar la transferencia de la información entre alumnos y docentes y fomenta el desarrollo del conocimiento (Mosquera, Torres, & Buelvas, 2017).

En la sección 2 de este artículo, se presenta con detalle la metodología y las estrategias de enseñanza llevadas a cabo en el laboratorio. A continuación, en la sección 3 se exponen los resultados obtenidos en el proyecto y un análisis estadístico descriptivo de los datos recopilados a través de encuestas. Para terminar, en la sección 4 se presentan las conclusiones.

1. Objetivos

1.1 Objetivo General

Implementar un laboratorio de programación usando Python a través de material sincrónico y asincrónico para contribuir al desarrollo los fundamentos de pensamiento computacional orientados a la solución de problemas en estudiantes de Mecánica de Sólidos.

1.2 Objetivos Específicos

Planear la organización del laboratorio de programación mediante la creación de un cronograma del curso.

Elaborar material didáctico como guías, códigos de ejemplo, diapositivas y videos que apoyen el aprendizaje de Python.

Exponer los fundamentos de programación en Python a través de clases expositivas.

Orientar a los estudiantes de Mecánica de Sólidos en la aplicación de Python en diferentes tópicos de la asignatura a través de la solución de problemas basados en casos de estudio reales.

2. Metodología

El laboratorio de programación se implementó en 6 cursos de Mecánica de Sólidos del segundo período académico de 2021, comprendido entre el 2 de noviembre del mismo año y el 11 de marzo de 2022. Los grupos D2, H2 y J1 estaban a cargo del profesor David Sebastián Cotes Prieto, con 18 participantes, y los grupos D1, D3 y H1 a cargo del profesor José Miguel Benjumea Royero, con 17 participantes. La participación de los estudiantes en el laboratorio fue de carácter voluntario, por tanto, no afectaba la nota final de la asignatura Mecánica de Sólidos, por el contrario, se trató de una bonificación para estimular el aprendizaje. Esta bonificación consistió en otorgar hasta una unidad adicional en cada examen según la nota del laboratorio y hasta media unidad en la definitiva para quienes aprobaran la asignatura. La oferta de participación en el laboratorio se hizo durante la segunda semana de clases.

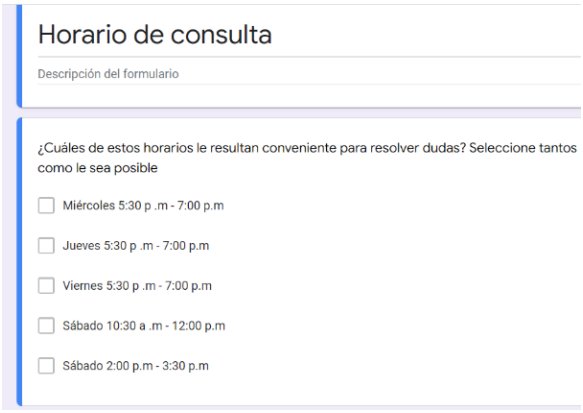
2.1 Planeación y Creación del Espacio de Trabajo

2.1.1 Introducción al Laboratorio

Se realizó una presentación del laboratorio, sus alcances y objetivos ante todos los estudiantes de Mecánica de Sólidos, de forma que se brindaron los datos de contacto del autor. Posteriormente, se implementó una encuesta de entrada en Google Forms, como muestra la Figura 1, con el fin de caracterizar la cantidad de personas interesadas en hacer parte del curso y los horarios de posible asistencia a clases de sesión virtual. Además, se caracterizó si los estudiantes tenían conocimientos previos en algún lenguaje de programación y su sistema operativo.

Figura 1.

Encuestas Realizadas a los Estudiantes

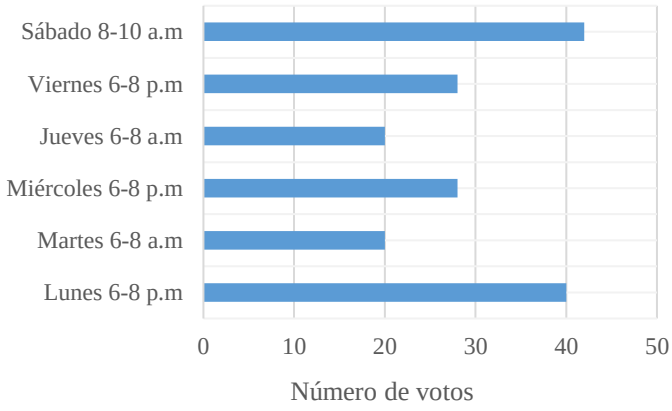


2.1.2 Determinación de Horarios y Herramientas

Con base en los resultados de esta encuesta, se determinaron dos horarios para las sesiones virtuales de mayor votación y un horario extra para llevar a cabo la consulta del laboratorio. Los horarios escogidos acorde con las encuestas fueron los lunes de 6:00 pm a 8: 00 pm y el otro grupo los sábados de 8:00 am a 10:00 am, tal y como se evidencia en la Figura 2. Después se facilitaron guías, a través del aula virtual de Google Classroom (Ramírez, 2021), para la instalación de Python y del editor de texto, Visual Studio Code, en formato de vídeo y PDF, acorde con el sistema operativo seleccionado por cada estudiante en la encuesta. Estas guías se encuentran en los Apéndices.

Figura 2.

Resultados de los Horarios más Seleccionados en la Encuesta de Entrada



2.1.3 Plan de Asignatura

Se desarrolló el programa de asignatura, según muestra la Figura 3, el cual incluyó el cronograma, las actividades, porcentajes de evaluación y condiciones para pertenecer al laboratorio. Los porcentajes de calificación se basaron en el promedio aritmético del conjunto de notas correspondientes a dos talleres y el promedio de las asistencias del estudiante a lo largo del semestre.

En cuanto al cronograma, se definieron las fechas de cada clase, con su respectivo día y hora, la temática en cuanto a teoría de programación y su respectiva aplicación en un campo de la Mecánica de Sólidos. El plan completo del laboratorio se facilitó a los estudiantes por medio del aula virtual del laboratorio y se encuentra completo en los anexos.

Figura 3.

Programa de Laboratorio de Programación.

PROGRAMA DE LABORATORIO DE PROGRAMACIÓN
ESCUELA DE INGENIERÍA CIVIL
MECÁNICA DE SÓLIDOS
SEGUNDO SEMESTRE ACADÉMICO DE 2021

Auxiliar: Edward Alfonso Ramírez González **Correo electrónico:** edward2170249@correo.uis.edu.co

Semana	Fecha	Día	Hora	Actividad	Temática Programación	Temática Mecánica de Sólidos	Trabajo especifico asignado
3	Noviembre	19 - 20	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Fundamentos de programación		
4	Noviembre	22 -25 - 26	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Funciones, operadores lógicos y estructuras cíclicas		
5	Noviembre - Diciembre	29 - 3 - 4	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Soluciones de ecuaciones lineales	Solución del sistema de equilibrio de una armadura	

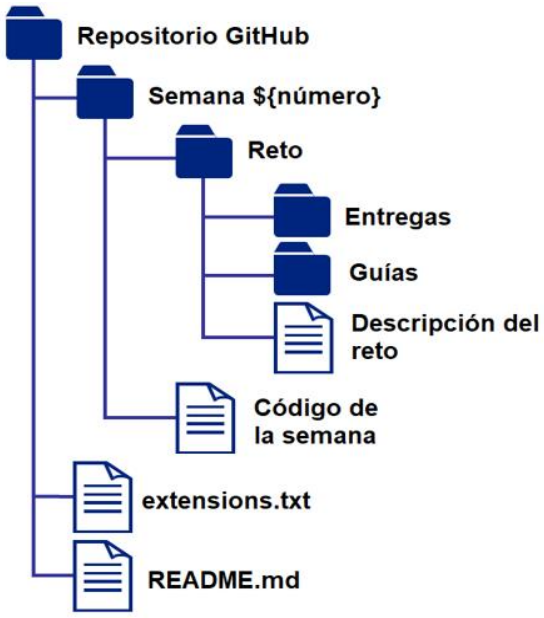
2.1.4 Organización del Espacio de Trabajo y Herramientas TIC

Se creó un repositorio público en GitHub, la cual es una plataforma de desarrollo en la nube que permite almacenar y administrar proyectos de software (GitHub Inc, 2022). De esta forma, se proporcionó el código de cada clase, material de apoyo, los talleres y los retos de programación a los estudiantes; siendo estos últimos la aplicación de la temática dada en la semana a problemas comunes en el ámbito del desarrollo de software.

Tales retos no poseían calificación ni importancia en la nota final del laboratorio, sino que el propósito de estas actividades fue el de fomentar el aprendizaje autodidacta del estudiante. En la Figura 4 se muestra la estructura de carpetas usada para organizar de forma clara los archivos y carpetas que hacen parte del proyecto.

Figura 4.

Estructura de Archivos del Repositorio en GitHub



El repositorio está compuesto por diferentes carpetas donde se almacenó todo el contenido trabajado durante el curso. En la raíz del repositorio de encuentra el *README.md*, el cual provee una descripción del proyecto y *extensions.txt*, un archivo compuesto por una lista de extensiones de Visual Studio Code que hacen del desarrollo de software una experiencia más cómoda y amena. Dentro de cada carpeta se colocó el código realizado durante las sesiones sincrónicas de clase, una carpeta con el enunciado del reto de desarrollo de software de cada semana, como lo muestra la Figura 5, una carpeta con la solución propuesta de cada estudiante de dicho reto y guías con material de apoyo para la resolución de retos o la tecnología enseñada durante la semana. Finalmente, el manejo de historial y control de versiones del proyecto en Git por parte del autor se hizo por medio de los comandos mostrados en la Tabla 1. Por parte de los estudiantes, ellos interactuaron con GitHub plenamente en su plataforma web para el envío de los retos y talleres del laboratorio. El recurso de TIC de GitHub puede encontrarse en (Ramírez, 2021)

Figura 5.

Ejemplo de Reto de Programación

README.md

Análisis de fuerzas internas de una viga

Desarrolle un software en Python, a través de la herramienta de Jupyter Notebook que analice el siguiente sistema estructural compuesto por una viga que se encuentra simplemente apoyada, es decir, el apoyo localizado en `A` es un de segundo grado y el localizado en `D` de primer grado. Determine:

> Diagrama de fuerza cortante Diagrama de momento flector Fuerza cortante máxima Momentos máximos (positivos y negativos si los hay)

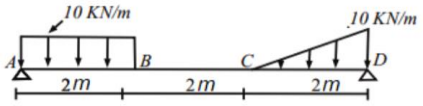


Tabla 1.

Comandos para Interactuar con el Repositorio en la Nube

Comando	Objetivo
git clone {url}	Obtener una copia del repositorio localmente
git add {archivo}	Agregar archivos o cambios en cierto punto de tiempo
git commit	Confirmar y guardar versión de los nuevos archivos o cambios
git push	Enviar nuevos cambios hechos localmente al repositorio en la nube
git pull	Traer localmente nuevos cambios que se dieron en la nube

En cuanto al aula virtual, se creó un espacio de trabajo en la plataforma Google Classroom. En esta plataforma se subieron los archivos correspondientes a la información del curso (cronograma, notas, material de apoyo, enunciados de cada taller y videos). Además, también se usó como medio para transmitir algunos comunicados y asignar las actividades con su fecha límite de entrega. Por medio de Google Classroom se creó una carpeta en Google Drive directamente conectada a este espacio de trabajo, de forma que todos los archivos y videos quedasen almacenados y expuestos a los estudiantes por medio de la nube. Finalmente, se crearon reuniones

periódicas de Google Meet para llevar a cabo las sesiones de clase sincrónicas, de forma que se utilizara la misma sala cada semana a la misma hora. Toda la información pertinente a los enlaces de cada recurso TIC usado en el laboratorio se compartió a los estudiantes vía WhatsApp, en un grupo destinado a ser el medio principal de comunicación con los participantes del laboratorio.

2.2 Desarrollo de Clases Sincrónicas

2.2.1 Dirección en el Uso de Herramientas

Se envió una guía a los estudiantes con el acceso a los recursos TIC ya mencionados. Posteriormente, se explicó de forma detallada el uso de las herramientas para desarrollar software como el editor de texto Visual Studio Code, el proceso y los comandos para ejecutar código Python en la terminal del computador y el Jupyter Notebook. Estos notebooks son una herramienta que provee Python, como ejemplifica la Figura 6, con el fin de ejecutar código de forma interactiva en el navegador, de forma que se puede separar cada parte del código en celdas y documentarlas usando la sintaxis *markdown*.

Figura 6.

Ejemplo de Código y Documentación en Jupyter Notebook

Finalmente, se expuso el proceso y los comandos necesarios para instalar librerías y código hecho por terceros, y así, evitar errores comunes durante la implementación de los paquetes más populares de acceso libre.

2.2.2 Preparación de Temática

Se dictaron 10 clases de forma sincrónica en Google Meet para dos grupos en bloques de dos horas. Estas clases fueron grabadas con el software de acceso libre OBS Studio y expuestas a través de Google Classroom. La temática fue presentada en diapositivas Power Point o en un Jupyter Notebook. El contenido de cada clase se preparó con base en recursos de acceso libre como la documentación oficial de Python, el libro Automate the Boring Stuff with Python (Sweigart, 2020) y cursos como CS50: Introduction to Computer Science (Harvard University, 2022) y FreeCodeCamp (Free Code Camp Org, 2022).

En la primera semana de clase se enseñaron los fundamentos de programación, tipos de datos básicos de Python, algoritmos y el uso del editor de texto como herramienta para instruir a la computadora. En la segunda semana se enseñó el uso de funciones básicas implementadas en Python, los operadores lógicos usados en condicionales y expresiones de verdad, así como la aplicación de estos conceptos en estructuras cíclicas como lo son los bucles *for* y *while*. Estos fundamentos de la ciencia de la computación fueron útiles para que los estudiantes comprendieran qué tipos de datos usar al momento de solucionar problemas de cálculo numérico en Mecánica de Sólidos. Además, el uso de condicionales para definir el flujo de ejecución del software, determinar la resistencia de estructuras y los bucles para realizar iteraciones hasta hacer cumplir cierta condición de equilibrio en sistemas estáticos.

Posteriormente, a partir de la tercera semana se introdujo el uso de librerías hechas por terceros como Numpy, para la solución de sistemas de ecuaciones lineales y cálculo numérico. En

la cuarta semana se explicó el uso de Matplotlib como la librería más popular para el ploteo y visualización de datos, con el fin de graficar las deformaciones de una armadura. Para terminar la primera parte del laboratorio, en la semana 5 se expuso el tema de las variables simbólicas y así determinar funciones de cortante y momento flector por medio de la integración y el cálculo diferencial en Python.

A continuación, en la sexta semana se prosiguió enseñando a determinar el equilibrio de una estructura, junto con la visualización de diagramas de fuerza cortante y momento flector usando Matplotlib. Asimismo, en la séptima semana se usó el mismo sistema estructural de la temática anterior para interactuar con Excel implementando Pandas como librería para el manejo y limpieza de datos. A continuación, en la séptima semana se explicó el tema de programación funcional y su sintaxis en Python, y así definir un software reutilizable que determine los esfuerzos internos de una viga debido a flexión biaxial. Además, se expuso un algoritmo para graficar el perfil de la viga junto con la posición del eje neutro.

De manera semejante, en la novena semana se expuso un caso de fuerza cortante de una viga sometida a una carga distribuida, y así, automatizar el cálculo de la separación mínima de pernos usados para conectar la sección cuadrada hueca de la viga con una placa metálica. Finalmente, en la última semana se expuso el manejo de librerías en Python versus una solución manual al momento de graficar el diagrama de distribución de esfuerzos de un perfil debido a momento torsional. La temática del laboratorio se presenta resumida en la Tabla 2.

Tabla 2.

Temática del Laboratorio de Programación

Semana	Temática Programación	Temática Mecánica de Sólidos
1	Fundamentos de programación	Tipos de datos útiles en sistemas estructurales
2	Funciones, operadores lógicos y estructuras cíclicas	Condicionales e iteraciones en sistemas de equilibrio
3	Soluciones de ecuaciones lineales	Solución del sistema de equilibrio de una armadura
4	Gráficas utilizando Matplotlib	Configuración deformada de una estructura con elementos sometidos a carga axial
5	Variables simbólicas, diferenciación e integración	Funciones de cortante y momento flector
6	Gráficas utilizando Matplotlib	Diagramas de cortante y momento flector
7	Manejo de Excel utilizando Pandas	Diseño a flexión con base en el módulo de sección elástica
8	Funciones y gráficas con Matplotlib	Distribución de esfuerzos por flexión biaxial
9	Variables simbólicas.	Transferencia de flujo cortante
10	Manejo de librerías en Python	Distribución de esfuerzos debido a momento torsional

2.3 Orientación de los Estudiantes

2.3.1 Notas y Actividades

Se diseñaron e implementaron dos talleres a lo largo del semestre con el fin de evaluar el desarrollo de las competencias de programación en los estudiantes. En la Figuras 7 y 8 se muestra uno de los ejercicios planteados en el taller 2. Los documentos de ambos talleres, sus soluciones y ejemplos del trabajo realizado por los estudiantes, se pueden encontrar en el recurso TIC de Google

Classroom presentado en la sección 2.1.4 y en los anexos. Estas actividades de trabajo independiente estuvieron enfocadas en la aplicación de la programación, usando Python como lenguaje para automatizar y solucionar problemas de Mecánica de Sólidos basados en casos de estudio reales. Por otra parte, cada estudiante envío su solución a través de *pull requests* en la plataforma GitHub. De esta forma, fue más sencillo revisar los archivos, dado que este mecanismo permite la visualización de código en la nube y su aprobación para anexar software al repositorio del proyecto.

Figura 7.

Instrucciones del Taller 2

ESCUELA DE INGENIERÍA Civil

Universidad Industrial de Santander
Escuela de Ingeniería Civil
Segundo periodo académico 2020
Laboratorio computacional MDS.

Universidad Industrial de Santander

INSTRUCCIONES:

1. Guardar en una carpeta "**T2_Codigo_Apellido_Nombre"** las soluciones en un archivo de Jupyter Notebook y con el nombre "**T2_solucion**", de manera que los resultados se vean de forma explícita en el navegador o al descargar y ejecutar cada celda del archivo. En la misma carpeta se debe subir el archivo Excel de la segunda parte.

2. Hacer un ***Pull Request*** en la carpeta ***/Semana9/Taller/Primera_Entrega*** en GitHub con la carpeta mencionada anteriormente. Fecha máxima de entrega: **26/02/2021**, hora límite **11:59 PM**.

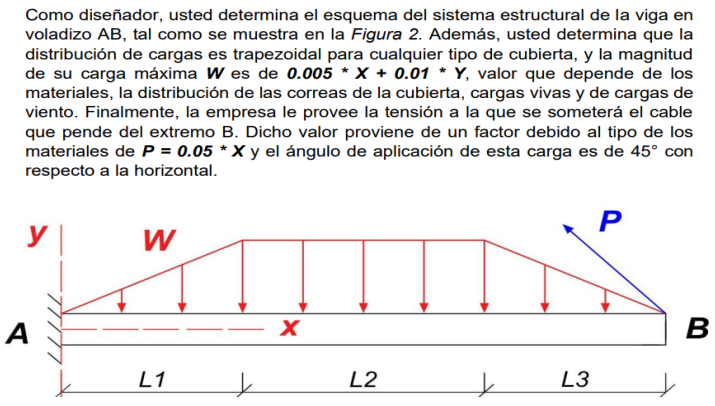
3. El día **28/02/2022** se entregará la retroalimentación personal al estudiante por medio de GitHub. Tras esto, tendrán como fecha límite **05/03/2022**, hora límite **11:59 PM** para hacer nuevamente un *Pull Request* con el mismo nombre de carpeta y archivos en la carpeta ***/Semana9/Taller/Segunda_Entrega*** en GitHub. La nota definitiva del taller se enviará a cada estudiante por medio de GitHub el **09/03/2022**.

EJERCICIO 1 - (VALOR: 2.5 UNIDADES)

Se dispone de una cubierta hecha en madera laminada apoyada en correas de

Figura 8.

Enunciado del Taller 2



Como diseñador, usted determina el esquema del sistema estructural de la viga en voladizo AB, tal como se muestra en la *Figura 2*. Además, usted determina que la distribución de cargas es trapezoidal para cualquier tipo de cubierta, y la magnitud de su carga máxima ***W*** es de ***0.005 * X + 0.01 * Y***, valor que depende de los materiales, la distribución de las correas de la cubierta, cargas vivas y de cargas de viento. Finalmente, la empresa le provee la tensión a la que se someterá el cable que pende del extremo B. Dicho valor proviene de un factor debido al tipo de los materiales de ***P = 0.05 * X*** y el ángulo de aplicación de esta carga es de 45° con respecto a la horizontal.

Cada taller fue calificado de la siguiente manera: se determinó una primera nota con base en lo realizado por el estudiante y se envió retroalimentación a través de GitHub. Posteriormente, los estudiantes tuvieron la oportunidad de realizar una segunda entrega del mismo taller en la plataforma, una semana después, con las correcciones necesarias. Finalmente, se calificó el software desarrollado en la segunda entrega y se seleccionó la mayor de las dos notas. Además, se elaboraron rúbricas de evaluación de cada taller (Ramírez, 2022), como lo muestra la Figura 9, para expresar a los estudiantes de forma clara los criterios de evaluación. El documento de la rúbrica se encuentra en los anexos. Por último, las notas definitivas del laboratorio se publicaron en Google Classroom y se reservó un espacio de dos horas, por medio de una reunión virtual en Google Meet, para realizar la retroalimentación de las calificaciones.

Figura 9.

Rúbrica Usada en Talleres

Ítem	¿Qué se espera?	Ponderación	Nota
Código legible	Nombres de variables claros ycon sentido para el valor que se está almacenando	10%	
Bases de código limpio	Que el código sea lo más reutilizable posible, es decir,con solo cambiar unos cuantos datos de entrada se debe calcular todo sin tener que cambiar más código	20%	

2.3.2 Material Extra

La temática expuesta en las clases sincrónicas fue complementada con videos publicados en Google Classroom, donde se expuso la solución a problemas comunes de desarrollo de software, la explicación de funciones y conceptos nuevos. En cuanto a material de lectura, se dispuso de guías en Jupyter Notebooks con la aplicación de las librerías en casos de estudio adicionales, además de anexar recursos electrónicos con la documentación oficial de Python y sus

paquetes desarrollados por terceros. Finalmente, se desarrollaron retos de programación acordes con la temática de cada semana, los cuales tuvieron como objetivo fomentar la práctica y el trabajo independiente, es decir, no tuvieron porcentaje de participación en el conjunto de notas obtenido en el laboratorio. Los retos, al igual que los talleres, se calificaron en forma de *pull requests* en GitHub y su retroalimentación también fue entregada por este medio.

2.3.3 Encuestas de Satisfacción de los Estudiantes

Se diseñó una encuesta en Google Forms una vez finalizada la quinta semana de clases de laboratorio, con el fin de medir la satisfacción de los estudiantes con respecto a la metodología de enseñanza, y así, tomar medidas ante la retroalimentación recibida. Posteriormente, se realizó una encuesta de satisfacción final después de la última semana del laboratorio con el fin de determinar si hubo mejoras con respecto a las medidas tomadas con base en las respuestas e inquietudes de la primera encuesta de satisfacción. En ambas encuestas se implementó una escala de calificación acorde con la escala de Likert, con el fin de cuestionar a los estudiantes sobre su nivel de acuerdo o desacuerdo con declaraciones acerca del laboratorio de programación. Las respuestas contaron con cinco opciones: 1 (totalmente en desacuerdo), 2 (en desacuerdo), 3 (ni de acuerdo ni en desacuerdo), 4 (de acuerdo) y 5 (totalmente de acuerdo). En la Tabla 3 se muestran las preguntas realizadas en las encuestas.

Tabla 3.

Primera Encuesta de Satisfacción

Preguntas Tipo Likert de las Encuestas
1. Las actividades y la temática tratada durante las horas de clases sincrónicas tienen un nivel de dificultad apropiado
2. Las herramientas utilizadas en el laboratorio están a la altura de la tecnología moderna y son adecuadas para mediar la enseñanza de las competencias fundamentales de la programación

Preguntas Tipo Likert de las Encuestas
3. La temática tratada durante las horas de clase y la metodología usada por el instructor ha sido amena y didáctica
4. Se ha visto evidenciado la utilidad de Python como herramienta de apoyo en problemas relacionados con Mecánica de Sólidos
5. Estoy dispuesto a usar Python como herramienta de programación en futuros trabajos y proyectos universitarios
6. Aprender Python fue divertido y me gustaría estudiar más de forma autodidacta, y así, aplicarlo en proyectos personales

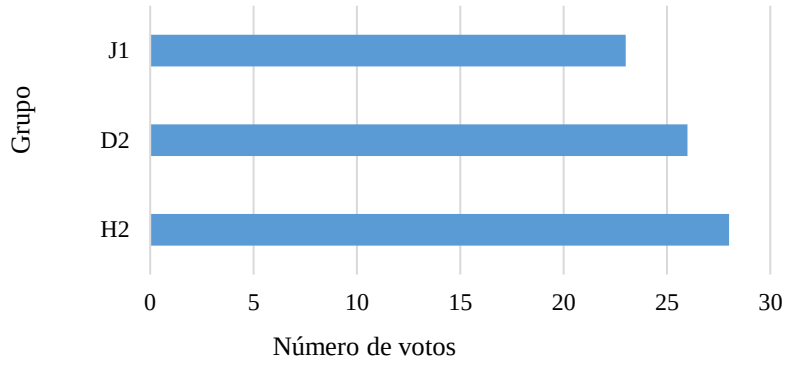
3. Resultados

3.1 Encuesta de Entrada

A continuación, se presentan los resultados obtenidos en la encuesta de entrada, la cual buscó sondear el interés de los estudiantes en hacer parte del laboratorio y sus conocimientos previos en programación. En total, 77 estudiantes de Mecánica de Sólidos respondieron la encuesta, divididos en 28 estudiantes del grupo H2, 26 estudiantes del grupo D2 y 23 estudiantes del grupo J1, todos a cargo del profesor David Sebastián Cotes Prieto, tal y como se muestra en la Figura 10. Posteriormente, se ofreció invitación libre para participar en el laboratorio a los estudiantes del profesor José Miguel Benjumea Royero.

Figura 10.

Participantes de Encuesta Inicial



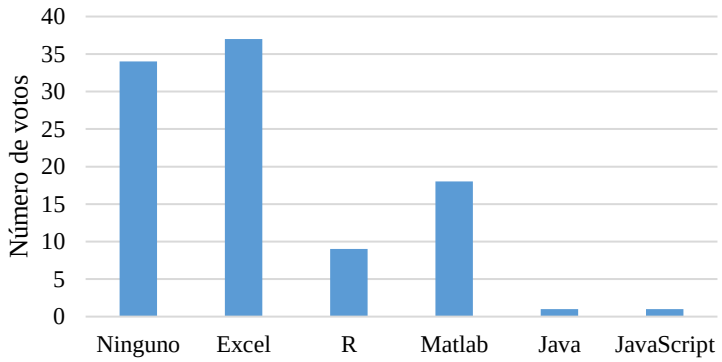
3.1.1 Conocimientos Previos de los Estudiantes

Se expuso una opción en la encuesta para que los participantes pudieran escoger tantos lenguajes de programación ya hubieran usado con anterioridad. Las opciones más escogidas fueron Excel, Matlab y R, y solo una minoría de la muestra tenía experiencia previa usando Java y JavaScript, tal y como se muestra en la Figura 11.

Es notorio mencionar que Excel se convirtió recientemente en un lenguaje de programación completo debido a la adición del paradigma de programación funcional en las hojas de cálculo (Gordon & Jones, 2021). No obstante, Excel aun presenta limitaciones y sus herramientas proveen un entorno de desarrollo para principiantes que no requieran de software con metodologías y características más allá de sus capacidades (Incerti, Thom, & Baio, 2019).

Figura 11.

Resultados de la Opción de Conocimientos Previos



Por otra parte, 34 estudiantes que componen un 45% de la muestra seleccionó la opción de no tener experiencia ninguna programando, lo que evidencia la falta de formación previa en programación en las asignaturas pertenecientes al plan de estudios de Ingeniería Civil en semestres pasados. Este porcentaje no es un buen indicador de entrada en cursos de programación intensos y de rápido avance, dado que la falta de conocimiento en los fundamentos de programación se asocia con estudiantes que sufren dificultades a la hora de comprender la temática, y, en consecuencia, genera desmotivación (Wang, 2020).

3.1.2 Sistema Operativo

De los resultados de la encuesta se encontró que dos personas (3% de la muestra) usan MacOs y el resto Windows. Por tanto, con base en estos resultados se desarrollaron dos guías diferentes para llevar a cabo la instalación de las herramientas necesarias para el laboratorio, cada una acorde con el sistema operativo que use el estudiante.

3.2 Resultados de las Encuestas de Satisfacción

Posteriormente, se realizó el análisis de la encuesta llevada a cabo al finalizar la primera parte del laboratorio, y así tener claro la satisfacción de los estudiantes después de la quinta semana

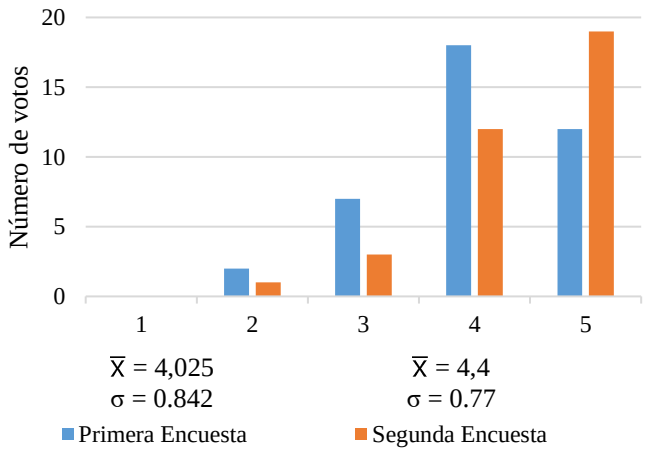
de clases. En total, 39 estudiantes de los grupos de D2, D3, J1 Y H2 de Mecánica de sólidos participaron en la encuesta.

3.2.1 Nivel de Dificultad del Laboratorio

La Figura 12 muestra las respuestas de los estudiantes a la pregunta “Las actividades y la temática tratada durante las horas de clases sincrónicas tienen un nivel de dificultad apropiado”. En cuanto a la encuesta de satisfacción realizada a mitad del laboratorio, se obtuvo un promedio de satisfacción de 4.025 y una desviación estándar 0.842. Esto se debe a que un 77% de los estudiantes de la muestra mostraron estar en acuerdo y acuerdo total con la anterior afirmación. En contraparte, solo dos personas, 5% de la muestra, demostró estar en desacuerdo y sintieron que la dificultad con la que se planteó el laboratorio fue complicada. Esto sugiere que la dificultad implementada en el laboratorio fue adecuada, debido a que es un bajo porcentaje a comparación con el 45% de estudiantes que habían expuesto no tener conocimiento alguno en programación, según lo enunciado en la sección 3.1.1 con base en la Figura 11.

Figura 12.

Percepción ante el Nivel de Dificultad del Laboratorio



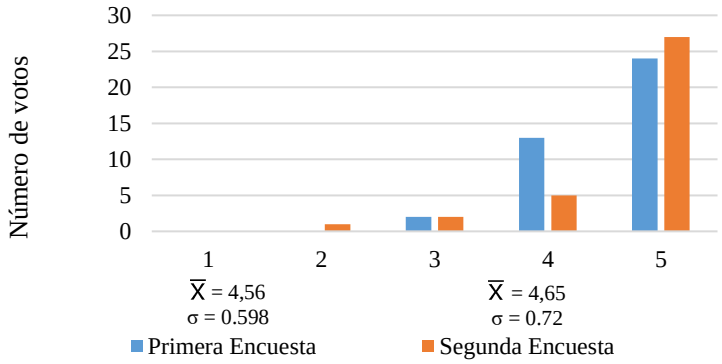
Esta cifra se mejoró en la encuesta de satisfacción final del laboratorio, donde el porcentaje de desacuerdo se redujo a 2.85%, con solo una persona estando en desacuerdo. Por otra parte, un 85% de la muestra sintió que la dificultad había sido apropiada, con un 50% total de los estudiantes estando totalmente de acuerdo, lo cual incremento el promedio de satisfacción a 4.4 y redujo la desviación estándar a 0.77. Estos indicadores evidencian que Python es un lenguaje adecuado para abordar la enseñanza de competencias en programación para estudiantes que no poseen experiencia previa con esta temática (Wang, 2020), (Nagpal & Gabrani, 2019), (Lo, Lin, & Wu, 2015).

3.2.2 Calidad de las Herramientas Empleadas en el Laboratorio

La Figura 13 muestra las respuestas de los estudiantes a la segunda pregunta “Las herramientas utilizadas en el laboratorio están a la altura de la tecnología moderna y son adecuadas para mediar la enseñanza de las competencias fundamentales de la programación”. En total, un 95% de los estudiantes de la muestra mostraron estar en acuerdo y acuerdo total con la afirmación, lo cual justifica el alto promedio de satisfacción de los estudiantes de 4.56 y una baja desviación estándar de 0.598. Por otra parte, dos personas, un 5% de la muestra, respondió de forma neutral a la afirmación, de forma que no hubo objeción alguna de ningún estudiante en cuanto al uso de las herramientas TIC implementadas según la sección 2.1.4.

Figura 13.

Percepción ante la Calidad de las Herramientas del Laboratorio



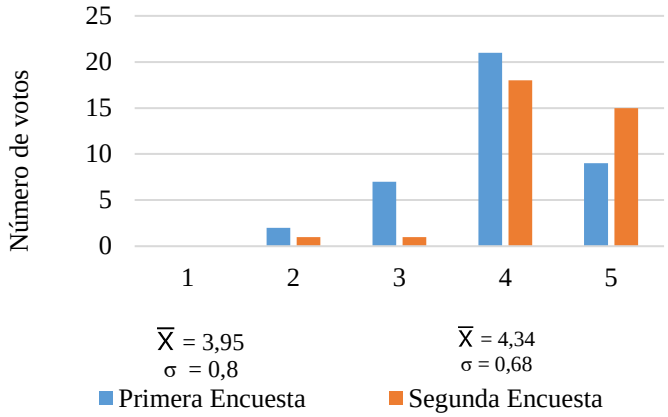
Por otra parte, en la encuesta final de satisfacción, el promedio de acuerdo se elevó hasta 4.65, al igual que la desviación estándar, la cual se elevó hasta 0.72 debido a la aparición de un dato extremo de desacuerdo. En total, un 91.4% de la muestra mostró satisfacción, con un acuerdo total por parte del 77% de la muestra. El alto porcentaje evidencia que las herramientas TIC utilizadas cumplen su papel como recursos indispensables para la transferencia de información, el desarrollo de conocimiento y la fomentación del aprendizaje en los estudiantes (Mosquera, Torres, & Buelvas, 2017).

3.2.3 Efectividad de la Temática y la Metodología

La Figura 14 muestra las respuestas de los estudiantes a la tercera pregunta “La temática tratada durante las horas de clase y la metodología usada por el instructor ha sido amena y didáctica”. En este caso, se obtuvo un promedio de satisfacción de 3.95, con una alta desviación estándar de 0.8, la cual, junto a un alto porcentaje de 77% de estudiantes que se mostraron de acuerdo con la metodología, evidencian una alta dispersión hacia el lado de satisfacción de la gráfica. Se encontró que 2 personas se mostraron en desacuerdo y el 17% de los entrevistados mostró estado neutral en cuanto a la temática y la metodología.

Figura 14.

Percepción ante la Efectividad de la Metodología del Laboratorio



Según la encuesta de satisfacción realizada durante la mitad del laboratorio, el desacuerdo se dio debido a que los estudiantes preferían que el laboratorio se hubiese dado de forma presencial, dado que el estudio presencial favorece la interacción entre profesor y alumno, evita las distracciones comunes que surgen al estudiar desde casa y problemas con la conexión a internet durante las clases. La otra razón fundamental del desacuerdo fue el ritmo de enseñanza con el cual se estaba desarrollando el curso hasta el momento. Para ayudar a estos estudiantes se propuso profundizar en la enseñanza de los fundamentos de programación, ralentizar el ritmo de las clases, incrementar la frecuencia de horas semanales y mejorar el sistema de apoyo a los estudiantes.

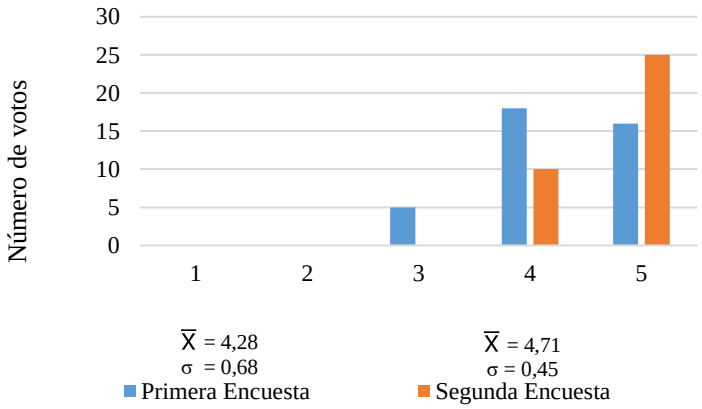
Posteriormente, se aplicaron algunas de las sugerencias mencionadas anteriormente para mejorar la experiencia de los estudiantes en el laboratorio. En consecuencia, los porcentajes de satisfacción mejoraron, teniendo un 94.28% como porcentaje de acuerdo y solo un 2.85% de la muestra en desacuerdo. Asimismo, el promedio de satisfacción aumentó a 4.34 y la desviación estándar se redujo a 0.68. Esto demuestra que la profundización y un mayor tiempo de práctica en conceptos complejos y abstractos de las ciencias de la computación mejoran los resultados y aumentan la motivación de los estudiantes (Wang, 2020).

3.2.4 Utilidad de la Programación en Mecánica de Sólidos

La Figura 15 muestra las respuestas de los estudiantes a la última pregunta “Se ha visto evidenciado la utilidad de Python como herramienta de apoyo en problemas relacionados con Mecánica de Sólidos”. En esta pregunta se obtuvo un promedio de satisfacción de 4.28 y una desviación estándar de 0.68, que se demuestra en un 87.2% de acuerdo por parte de los estudiantes en cuanto al uso de Python en la Mecánica de Sólidos y con un acuerdo total por parte del 41% de la muestra. No hubo ningún desacuerdo en cuanto a esta afirmación, siendo este un indicativo de que durante todo el laboratorio se implementaron casos de estudio de Mecánica de Solidos que fomentaron el aprendizaje de los estudiantes.

Figura 15.

Percepción ante la Relación de la Mecánica de Sólidos con la Programación



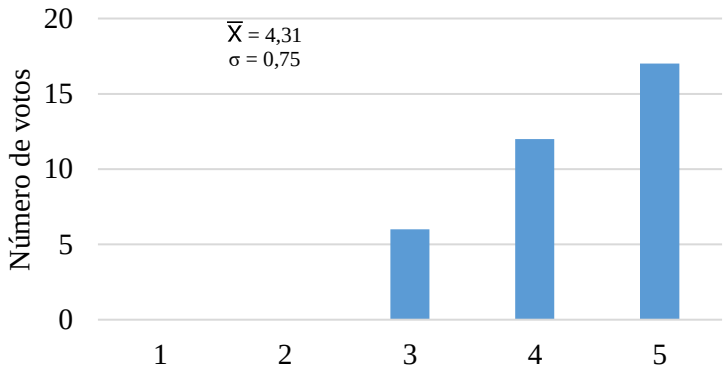
Cabe resaltar que, como muestra la Figura 15, en la encuesta de satisfacción final, el 100% de los estudiantes declararon estar de acuerdo con la afirmación y no hubo desacuerdo ni respuestas neutrales. Esto impactó directamente el crecimiento del promedio a 4.71 y la disminución de la desviación estándar a 0.45. Por tanto, queda en evidencia que los estudiantes pudieron relacionar las ciencias de la computación y su utilidad en la resolución de problemas de ingeniería civil.

3.2.5 Uso futuro de Python en el Ámbito Universitario

La Figura 16 muestra las respuestas de los estudiantes a la última pregunta “Estoy dispuesto a usar Python como herramienta de programación en futuros trabajos y proyectos universitarios”. Se obtuvo un alto promedio de satisfacción de 4.31, junto con una desviación estándar de 0.75. Además, la afirmación no tuvo ningún desacuerdo y un 82.86% de estudiantes declaró satisfacción, lo cual demuestra la adecuada implementación de la programación en casos de estudio de Mecánica de Sólidos, según lo enunciado en la sección 3.2.4, y que Python es un lenguaje que puede ser usado abiertamente en diferentes áreas de la ingeniería civil.

Figura 16.

Percepción ante el uso de Python en Proyectos Universitarios



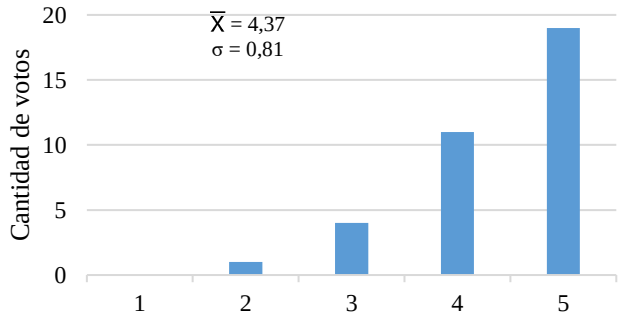
3.2.6 Uso futuro de Python en el Ámbito Personal

La Figura 17 muestra las respuestas de los estudiantes a la última pregunta “Aprender Python fue divertido y me gustaría estudiar más de forma autodidacta, y así, aplicarlo en proyectos personales”. Esta afirmación obtuvo un promedio de satisfacción de 4.37 y una desviación estándar de 0.81. Este promedio es ligeramente mayor al calculado en la sección 3.2.5, indicando así una predisposición mayor al uso de la programación en el ámbito personal y autodidacta que en el ámbito académico. Además, un 85.71% de estudiantes declaró estar de acuerdo con la afirmación, lo cual refuerza que Python es un lenguaje adecuado para aprender

programación cuando no se cuenta con conocimientos previos en las ciencias de la computación (Lo, Lin, & Wu, 2015). En consiguiente, estos resultados confirman el valor del laboratorio de Python como útil y divertido para introducir la programación a estudiantes de ingeniería civil.

Figura 17.

Percepción ante la Posibilidad de usar Python en Proyectos Personales



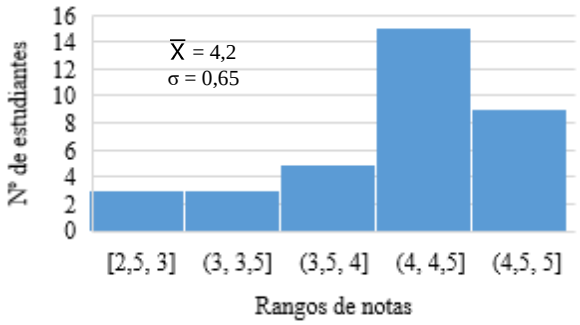
3.3 Calificación de Talleres

3.3.1 Primer Taller

La Figura 18 muestra un gráfico de distribución de las notas obtenidas por parte de los estudiantes en el primer taller del laboratorio. En este, solo un 8.57% obtuvo una nota menor a 3 y un 68.57% obtuvo una nota sobresaliente mayor a 4. Además, la nota promedio del primer taller fue de 4.2 y su desviación estándar de 0.65, lo cual indica que la mayoría de los participantes del laboratorio adquirió competencias y conocimientos básicos de programación.

Figura 18.

Notas Obtenidas en el Primer Taller

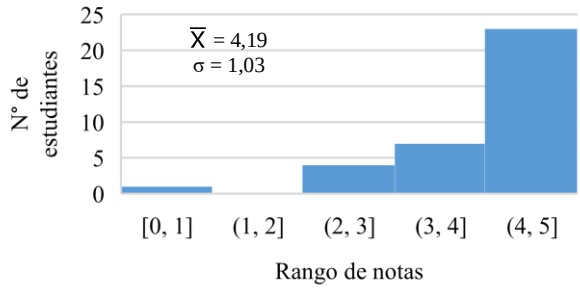


3.3.2 Segundo Taller

La Figura 19 muestra un gráfico de distribución de las notas obtenidas por parte de los estudiantes en el segundo taller del laboratorio. En este, solo un 14.28% obtuvo una nota menor a 3 y un 65.71% obtuvo una nota sobresaliente mayor a 4. Esto indica que la mayoría de los estudiantes adquirió las competencias adecuadas en cuanto al manejo de librerías de código abierto y su respectiva aplicación en la Mecánica de Sólidos. Cabe mencionar que en el segundo taller la nota promedio se mantuvo en 4.19, sin embargo, la desviación estándar aumentó a 1.03, debido a un dato atípico por parte de un estudiante que no presentó el trabajo.

Figura 19.

Notas Obtenidas en el Segundo Taller



3.3.3 Notas de Asistencia

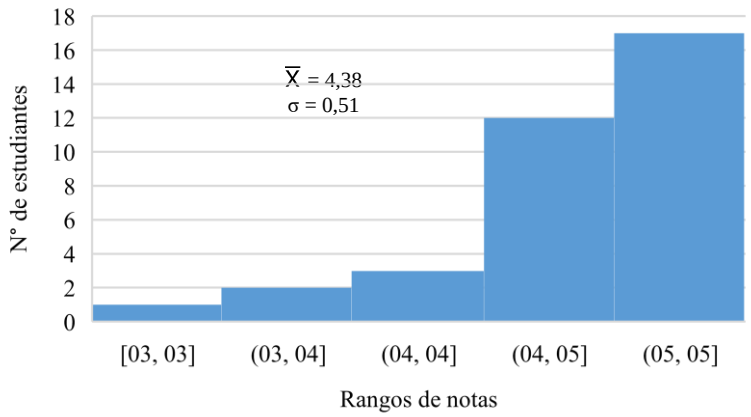
En cuanto a las notas de asistencia, el 97% de los participantes obtuvieron una nota mayor o igual a 4.5 y el resto obtuvo una nota de entre 4 y 4.5.

3.3.4 Notas Finales

La Figura 20 muestra el gráfico de distribución notas finales del laboratorio. En promedio, los estudiantes del laboratorio obtuvieron una nota sobresaliente de 4.38, con una baja desviación estándar de 0.51 y un 97.14% de aprobación del curso. Estos indicadores evidencian que la dificultad del laboratorio fue adecuada, según lo discutido en la sección 3.2.1 y que los estudiantes cumplieron con lo enunciado en el plan de asignatura mencionado en la sección 2.1.3.

Figura 20.

Notas Definitivas del Laboratorio



3.4 Encuestas de Caracterización de Estudiantes

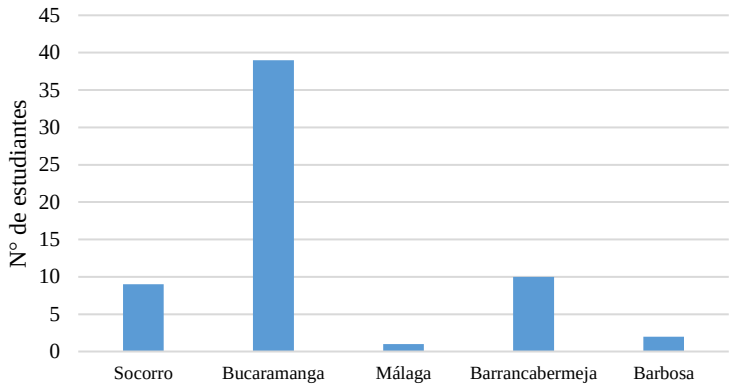
3.4.1 Sedes de Estudio

La Figura 21 muestra un gráfico de distribución de las sedes de proveniencia de los estudiantes que se inscribieron al laboratorio de programación. Por otra parte, en la Figura 22 se puede observar el porcentaje de estudiantes que culminaron el laboratorio con respecto a los que ingresaron. Con base en esta gráfica, es evidente que todos los estudiantes provenientes de las

sedes de Málaga, Barrancabermeja y Barbosa desertaron y solo las sedes de Socorro y Bucaramanga fueron las sedes provenientes de los estudiantes que culminaron el laboratorio con éxito.

Figura 21.

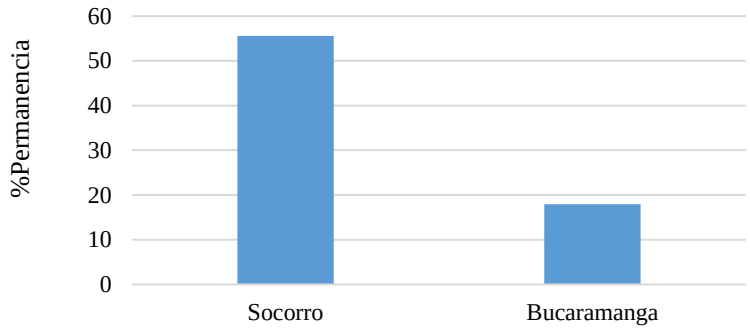
Sedes de Proveniencia de los Estudiantes Inscritos



Es notorio mencionar que un 55% de los estudiantes de la sede de Socorro inscritos permaneció en el curso, y apenas un 18% de los estudiantes de la sede de Bucaramanga pudo culminar. Sin embargo, de los estudiantes que culminaron un 58% provenía de la sede central en Bucaramanga y un 42% de la sede de Socorro, por tanto, la gran cantidad de estudiantes provenientes de la sede central ayudo a que más estudiantes de esta pudieran terminar el laboratorio, a pesar de su alto porcentaje de deserción. Esto se pudo dar debido a que el 60% de los estudiantes provenientes de la sede de Socorro que culminaron el laboratorio poseían conocimientos previos en programación, en contraste con el 42% de estudiantes de la sede de Bucaramanga.

Figura 22.

Porcentaje de Permanencia de Estudiantes con Respecto a la Sede

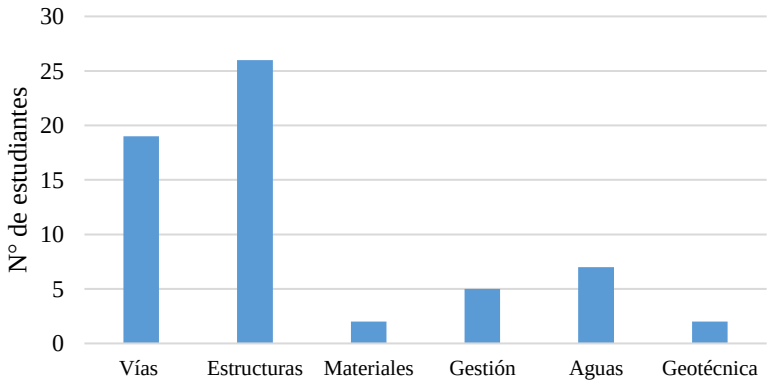


3.4.2 Áreas de Interés

La Figura 23 muestra un gráfico de distribución de las áreas de especialización que interesan a los estudiantes que se inscribieron al laboratorio de programación. Se puede observar que las áreas de mayor interés por parte de los estudiantes fueron estructuras, diseño vial y aguas. Cabe resaltar que, en las últimas décadas, el área de las estructuras ha sido impactada por el surgimiento del software y el poderoso cálculo numérico computacional (Bower, 2012).

Figura 23.

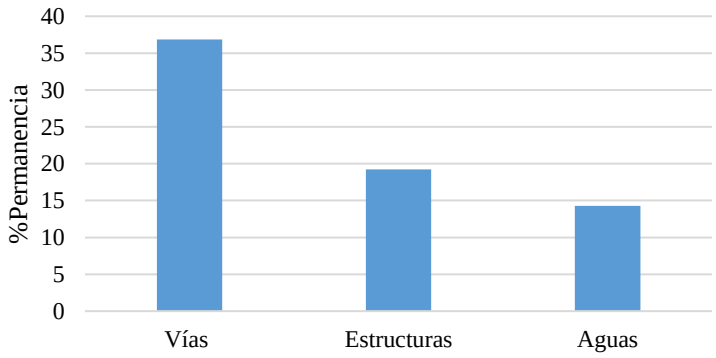
Especialidad de Interés de los Estudiantes



Finalmente, con base en la Figura 24 se puede observar que solo los estudiantes con interés en las tres áreas mencionadas anteriormente culminaron el laboratorio exitosamente. Esto se pudo dar debido a que son los tres campos de especialización que más se cubren actualmente en el plan de pregrado de ingeniería civil en la Universidad Industrial de Santander, por lo tanto, se debe incentivar el uso de la programación en las áreas de Materiales, Gestión y Geotecnia. Cabe mencionar que en el momento en que los estudiantes cursan Mecánica de Sólidos no han recibido enseñanza en ninguna de las áreas que presentaron deserción total, lo cual puede sugerir la predilección de los estudiantes hacia las estructuras y vías.

Figura 24.

Especialidad de Interés de los Estudiantes



3.4.3 Créditos Matriculados

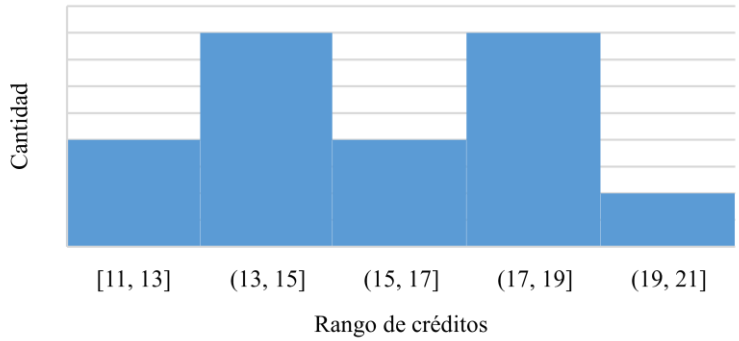
La Figura 25 muestra un gráfico de distribución de los créditos matriculados por parte de los estudiantes que culminaron el laboratorio de programación. En promedio, los estudiantes matricularon 15.76 créditos, donde un 53.85% matriculo más de 16 créditos durante el periodo académico 2021-2.

Teniendo en cuenta que un crédito académico equivale a 48 horas de trabajo académico del estudiante (División de Publicaciones, 2015), según el reglamento estudiantil UIS, se determinó una carga semanal promedio de 31,52 horas de trabajo independiente por parte del

estudiante, lo cual es una cantidad considerable de tiempo por parte de los participantes que lograron culminar el laboratorio. Con base en que la cantidad de créditos del cuarto semestre académico de ingeniería civil es de 18 (Universidad Industrial de Santander, 2022), se pudo observar que un 76.92% de los estudiantes que culminaron el laboratorio contaban con la cantidad de créditos recomendados. Además, la diferencia entre los créditos sugeridos y el promedio indica una cantidad de 9 horas disponibles para el estudio independiente, una cantidad más que suficiente para seguir con el ritmo de enseñanza del laboratorio.

Figura 25.

Créditos Matriculados por Parte de los Estudiantes que Culminaron el Laboratorio

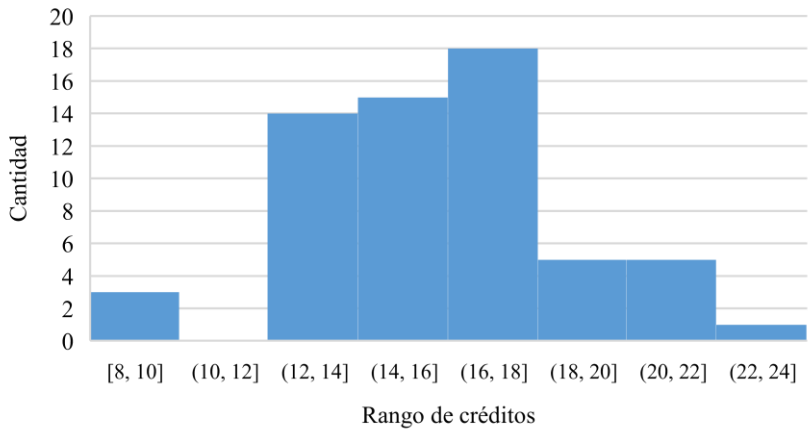


Por otra parte, la Figura 26 muestra el gráfico de distribución de los créditos matriculados por parte de los estudiantes que no culminaron el laboratorio de programación. En promedio, los estudiantes que desertaron matricularon 16.33 créditos, donde un 47.54% matriculó más de 16 créditos durante el periodo académico 2021-2. Se determinó una carga semanal promedio de 32.66 horas de trabajo independiente por parte del estudiante, siendo esta cantidad mayor a al promedio de los estudiantes que lograron culminar el laboratorio. Además, un 36.1% de los estudiantes tenían una cantidad de créditos matriculados mayor o igual a la cantidad sugerida de 18 créditos. Esto indica la posibilidad de que muchos estudiantes que se inscribieron al laboratorio desertaron debido a una alta carga de obligaciones por parte de otras materias, de forma que no vieron el

laboratorio como merecedor de un esfuerzo adicional, ya que este era de participación meramente voluntaria.

Figura 26.

Histograma de Créditos Matriculados por Estudiantes que Desertaron



Además, como ya se vio en la Figura 23, de los estudiantes que desertaron, un 85% tenía como área de interés el diseño vial, estructural e hidráulico, y las materias donde se estudian esas especializaciones tienden a tener una alta demanda académica y mayor cantidad de créditos. Finalmente, según lo discutido en la sección 3.4.1, los estudiantes de las sedes diferentes a Bucaramanga y Socorro, que desertaron en su totalidad, pudieron verse afectados por la alta cantidad de créditos matriculados y tener más problemas a la hora de adaptarse a la carga académica en la sede central. Por tanto, esto puede indicar que se debe incentivar y aplicar la programación en asignaturas dictadas desde la sede, y así evitar la sobrecarga académica que implica el no tener experiencia previa programando.

4. Conclusiones

En este trabajo se implementó un laboratorio de programación usando Python en la asignatura Mecánica de Sólidos del programa de ingeniería civil de la Universidad Industrial de Santander, en Colombia. Se diseñó una metodología de clases sincrónicas y material asincrónico usando diversos recursos TIC para orientar a los estudiantes en la aplicación de Python en diferentes componentes teóricos de la asignatura, a través de la solución de problemas de casos de estudio. Se esperaba que esta experiencia de aprendizaje, aplicada por primera vez en el curso, contribuyera al desarrollo de los fundamentos de pensamiento computacional y habilidades de programación en los estudiantes de ingeniería civil. Teniendo en cuenta los resultados discutidos en la sección 3 del artículo, las principales conclusiones extraídas del diseño e implementación del laboratorio son:

1. Los resultados de las encuestas de satisfacción de fin de semestre demuestran que la mayoría de los estudiantes se mostraron de acuerdo con respecto a las herramientas y metodología de enseñanza y disfrutaron de la experiencia de aprendizaje a través de aplicaciones en la Mecánica de Sólidos, debido a que percibieron la programación como una habilidad práctica que puede beneficiarles en el futuro. Por lo tanto, puede ser aconsejable implementar una estrategia de enseñanza similar para desarrollar actividades que fortalezcan el uso de la programación en otras asignaturas del plan de estudios de ingeniería civil, y de esa forma, incentivar la aplicación de estos conceptos en las diversas ramas que son de interés para los estudiantes de ingeniería.

2. Los resultados de las encuestas demostraron que Python es un lenguaje adecuado para abordar la enseñanza de competencias en programación para estudiantes que no poseen

experiencia previa en las ciencias de la computación. Además, estos confirman el valor de Python como un lenguaje divertido y fácil de aprender.

3. El alto porcentaje de estudiantes sin experiencia previa combinado con la alta carga académica conllevó a la frustración y deserción de estudiantes. Para la mejora futura del laboratorio, se propone profundizar en la enseñanza de los conceptos básicos de programación, agregar material extra como videos de fundación antes de las clases sincrónicas e incorporar exámenes y preguntas teóricas de programación en los talleres. Asimismo, incluir el laboratorio en el plan de estudios oficial de la asignatura, incrementar la frecuencia de horas semanales, ralentizar el ritmo de las clases, y mejorar el sistema de apoyo y resolución de dudas por medio de clases presenciales.

4. En general, los estudiantes que culminaron el laboratorio obtuvieron buenas calificaciones en los talleres y en la definitiva. Esto demuestra que la implementación del laboratorio de Python logró el objetivo de introducir la programación y contribuir al desarrollo de habilidades de las ciencias de la computación en estudiantes de ingeniería civil.

Referencias Bibliográficas

Bower, A. (2012). *"Applied Mechanics of Solids"*. Obtenido de http://solidmechanics.org/index.html

División de Publicaciones. (2015). *"Reglamento Académico-Estudiantil De Pregrado".* Bucaramanga.

Ebrahimzadeh, E., & Safai, N. (2019). *"Should 'Python for Engineers' be a Course Taught to Freshmen Engineering Majors in the U.S.A. and Abroad?"*. Obtenido de American Society for Engineering Education: doi: 10.18260/1-2--33263

Free Code Camp Org. (2022). *"Learn to code - for free"*. Obtenido de https://www.freecodecamp.org/

GitHub Inc. (2022). *"GitHub: Where the world builds software"* . Obtenido de https://github.com/

Gordon, A., & Jones, S. (2021). *"LAMBDA: The ultimate Excel worksheet function"*. Obtenido de https://www.microsoft.com/en-us/research/blog/lambda-the-ultimatae-excel-worksheet-function/

Harvard University. (2022). *"CS50's Introduction to Computer Science"*. Obtenido de https://www.edx.org/course/introduction-computer-science-harvardx-cs50x

Incerti, D., Thom, H., & Baio, G. J. (2019). *"R You Still Using Excel? The Advantages of Modern Software Tools for Health Technology Assessment"* . Value in Health, pp. 575–579.

Javed, A., Zaman, M., Uddin, M., & Nusrat, T. (Oct. de 2019). *"An analysis on python programming language demand and its recent trend in bangladesh", in ACM International Conference Proceeding Series, pp. 458–465.* . Obtenido de doi: 10.1145/3373509.3373540

Lo, C., Lin, Y., & Wu, C. (Jun. de 2015). *"Which programming language should students learn first? A comparison of Java and python", in Proceedings - International Conference on Learning and Teaching in Computing and Engineering, LaTiCE, pp. 225–226.* Obtenido de doi: 10.1109/LaTiCE.2015.15

Mosquera, C., Torres, E., & Buelvas, E. (2017). *"Uso de las tecnologías de la información y la comunicación en el proceso de enseñanza-aprendizaje de programación numérica en ingenierías para la universidad de la Costa", .* Omnia, vol. 23, no. 1, pp. 20–32.

Nagpal, A., & Gabrani, G. (2019). *Python for Data Analytics, Scientific and Technical Applications. Proc.* Obtenido de Amity Int. Conf. Artif. Intell. AICAI : doi: 10.1109/AICAI.2019.8701341

Python Software Foundation. (2022). *"The Python Tutorial"*. Obtenido de https://docs.python.org/3/tutorial/index.html

Ramírez, E. (2021). *"Aula virtual del laboratorio"*. Obtenido de https://classroom.google.com/c/NDMyNDY5ODExMDY4?hl=es&cjc=sd65soo

Ramírez, E. (2021). *"Repositorio GitHub del laboratorio"*. Obtenido de https://github.com/edwardramirez31/Python-Coding-Lab

Ramírez, E. (2022). *"Rúbrica de evaluación del laboratorio"*. Obtenido de https://classroom.google.com/u/0/c/NDMyNDY5ODExMDY4/m/NDU4Mjg5NDkwNjE2/details?hl=es

Sørensen, & Nordfalk, J. (2020). *"An Application Driven Big Data/Machine Learning Education program for Engineering Professionals – Methods and Examples", Proc. of the 2nd International Conference on Electrical, Communication and Computer Engineering.*

Sweigart, A. (2020). *"Automate the Boring Stuff with Python"*. Obtenido de https://automatetheboringstuff.com/

Universidad Industrial de Santander. (2022). *"Programa Académico: Ingeniería Civil"*. Obtenido de https://www.uis.edu.co/webUIS/es/academia/facultades/fisicoMecanicas/escuelas/ingenieriaCivil/programasAcademicos/ingenieriaCivil/planEstudios.html

Wang, C. (2020). *"A Brief Introduction of Python to Freshman Engineering Students Using Multimedia Applications", Proceedings - Frontiers in Education Conference, FIE, vol.* Obtenido de doi: 10.1109/FIE44824.2020.9273894

Apéndices

Apéndice A. *Programa de Laboratorio*

PROGRAMA DE LABORATORIO DE PROGRAMACIÓN
ESCUELA DE INGENIERÍA CIVIL
MECÁNICA DE SÓLIDOS
SEGUNDO SEMESTRE ACADÉMICO DE 2021

Auxiliar: Edward Alfonso Ramírez González **Correo electrónico:** edward2170249@correo.uis.edu.co

Semana	Fecha	Día	Hora	Actividad	Temática Programación	Temática Mecánica de Sólidos	Trabajo específico asignado
3	Noviembre	19 - 20	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Fundamentos de programación		
4	Noviembre	22 -25 - 26	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Funciones, operadores lógicos y estructuras cíclicas		
5	Noviembre - Diciembre	29 - 3 - 4	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Soluciones de ecuaciones lineales	Solución del sistema de equilibrio de una armadura	
6	Diciembre	6 - 9 - 10	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Gráficas de puntos y rectas utilizando Matplotlib	Gráfica de deformaciones en una armadura	
7	Diciembre	13 - 16 - 17	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Variables simbólicas, diferenciación e integración	Funciones de cortante y momento flector	**Presentación Taller 1**
8	Enero	10 – 14 - 15	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Gráficas de puntos y rectas utilizando Matplotlib	Diagramas de cortante y momento flector	
9	Enero	17-21-22	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Datos importados y exportados de Excel utilizando Pandas	Determinar perfil óptimo ante una solicitación	
10	Enero	24-28-29	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Variables simbólicas, diferenciación, integración	Fuerza cortante	
11	Enero Febrero	31-4-5	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	Manejo de librerías en Python	Diagrama de distribución de esfuerzos debido a momento torsional	
12	Febrero	7-11-12	06:00-08:00 pm / 08:00 - 10: 00 am	Clase expositiva	DIC (Correlación de imágenes)	Graficar distribución de esfuerzos principales y círculo de Mohr	**Presentación Taller 2**

INSTRUMENTOS DE EVALUACIÓN

- Talleres (70%)
  - Taller 1 (35%)
  - Taller 2 (35%)
- Asistencia (30%)

RESUMEN DE LAS ACTIVIDADES PROPUESTAS

• **Clase expositiva**: Consiste en sesiones magistrales de 2 horas dónde el auxiliar participa presentando un tema de forma ordenada y didáctica. El estudiante participa tomando nota de forma ordenada de lo expuesto, planteando preguntas sobre las temáticas y realizando los ejercicios propuestos por el auxiliar.

• **Talleres**: Consiste en el desarrollo de ejercicios basados en casos de estudio de la mecánica de sólidos implementando Python y utilizando los conceptos aprendidos en el curso. Para cada taller se realizarán dos entregas y se selecciona la mayor calificación. Se realizará una revisión parcial de la primera entrega y se entregarán comentarios para el desarrollo de la segunda entrega. *Nota: Su entrega posterior a la fecha y hora límite se penalizará con 1,0 en su calificación definitiva.*

SISTEMA DE CALIFICACIÓN DE TALLERES

VALORACIÓN	CARACTERÍSTICA
0,0	No realiza el ejercicio. 40°
1,5	Realiza parcial e incorrectamente el ejercicio.
3,0	Realiza parcialmente el ejercicio.
4,0	Realiza completa y correctamente el ejercicio.
5,0	Realiza completa y correctamente el ejercicio. Detalla mediante comentarios el procedimiento empleado. Los resultados de ejecución del programa son únicamente los resultados finales requeridos en cada ejercicio.

SISTEMA DE CALIFICACIÓN ASISTENCIAS

VALORACIÓN	CARACTERÍSTICA
0,0	No asiste a clase.
2,5	Ingresa a clase 10 minutos después del inicio / Abandona la clase antes de su culminación.
5,0	Ingresa a clase a tiempo.

EQUIVALENCIA CUANTITATIVA

La calificación definitiva consiste en el promedio aritmético del conjunto de notas correspondientes a cada taller y asistencia.

Apéndice B. *Guías de Instalación de Software*

Universidad Industrial de Santander

LABORATORIO DE PROGRAMACIÓN
MECÁNICA DE SÓLIDOS

ESCUELA DE INGENIERÍA Civil

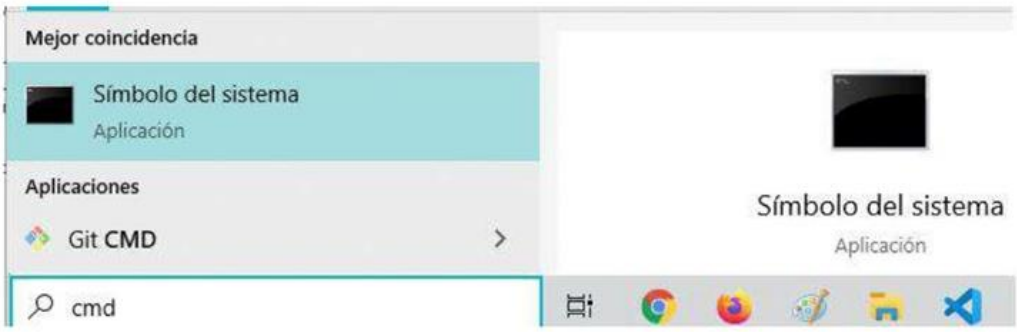
Guías de Instalación

Para el desarrollo del laboratorio se hará uso del lenguaje *Python* y de su gestor de paquetes *pip*. Además, al final de la guía se encuentra el proceso de instalación de *Visual Studio Code*, el entorno de desarrollo que se utilizará durante las clases.

- ***Instalación en Windows***

1. Acceder a la página oficial de Python *https://www.python.org/downloads/* y descargar la *última versión* en el botón *Download Python*.
2. Ejecutar instalador
3. Seleccionar la opción *Add Python 3.9 to PATH* **(IMPORTANTE)**

4. Comenzar con la instalación por medio de *Install Now*.
5. Abrir la línea de comando del computador CMD



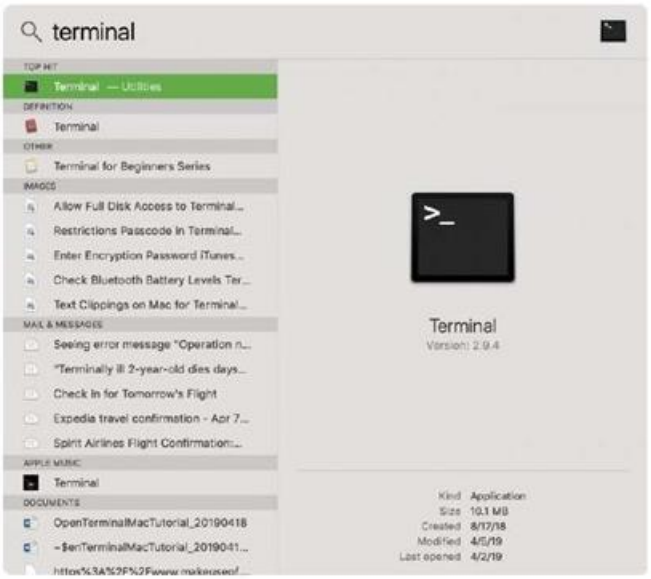
6. Escribir el siguiente comando para verificar que la instalación fue correcta.

```
python --version
```

- **Instalación en MacOS**

1. Acceder a la página oficial de Python *https://www.python.org/downloads/*
2. Descargar la última versión en el botón *Download Python*
3. Ejecutar instalador y hacer clic en *Next* a todas las opciones.
4. Verificar la instalación abriendo la consola y ejecutar el siguiente comando:

```
python3 --version
```



- **Instalación en Linux**

1. Abrir la terminal del sistema y ejecutar los siguientes comandos en orden

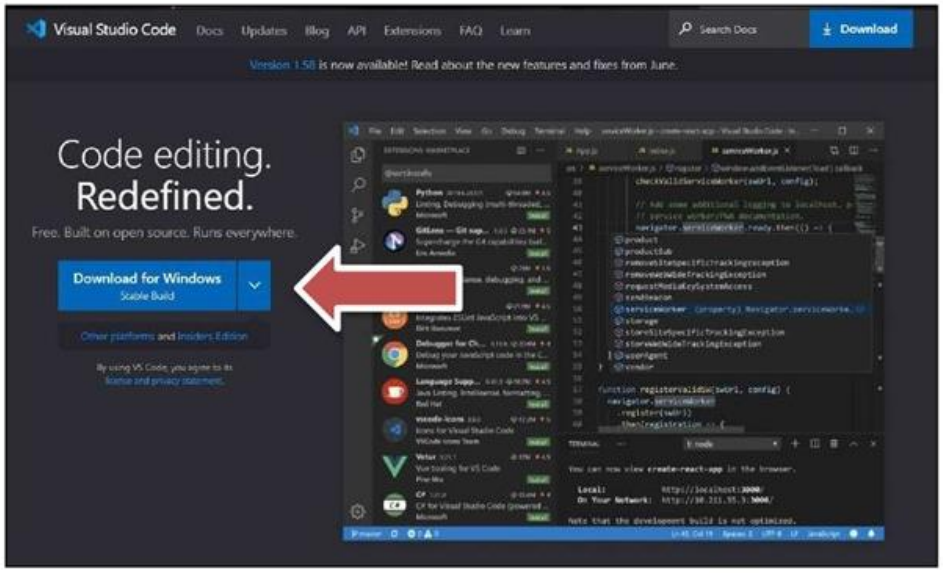
```
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.8
sudo apt install python3-pip
```

2. Verificar si la instalación fue correcta ejecutando el siguiente comando

```
python3 --version
 pip --version
```

INSTALACIÓN DE VISUAL STUDIO CODE (WINDOWS Y MAC)

1. Acceder a la página oficial *https://code.visualstudio.com/*
2. Descargar el *instalador* en el botón *Download*:



3. Ejecutar el instalador y dar *next* a todas las opciones

Apéndice C. *Rúbrica de Talleres*

UNIIVERSIDAD INDUSTRIAL DE SANTANDER
ESCUELA DE INGENIERÍA CIVIL
LABORATORIO DE PROGRAMACIÓN – MECÁNICA DE SÓLIDOS

Rúbrica para la evaluación de talleres y retos

Ítem		Evaluación			
		¿Qué se espera?	Ponderación	Nota	Comentarios
Desarrollo de software	Código legible	Nombres de variables claros y con sentido para el valor que se está almacenando	10%		
	Bases de código limpio	Que el código sea lo más reutilizable posible, es decir, con solo cambiar unos cuantos datos de entrada se debe calcular todo sin tener que cambiar más código	20%		
		Evidenciar la escritura de código limpio, dejando buen espaciado entre los iguales, llamado de variables y funciones, comentarios claros de lo que se intenta hacer e imprimir una salida que tenga sentido	20%		

Ítem		Evaluación			
		¿Qué se espera?	Ponderación	Nota	Comentarios
Mecánica de Sólidos	**Seguir las instrucciones dadas en el taller**	Se presentan un desarrollo de los ejercicios en cuadernos Jupyter localizados en la carpeta indicada	10%		
	Unidades coherentes con base en el ejercicio	Se presentan las salidas o respuestas del programa en unidades coherentes (cargas distribuidas en kN / m, N / m, longitudes en mm / m, fuerzas en N / kN y esfuerzos en MPa)	20%		
	Salidas correctas	La respuesta presentada en cada numeral del ejercicio es correcta	20%		

Apéndice D. *Primer Taller*

Universidad Industrial de Santander
Escuela de Ingeniería Civil, 2021 - II
Laboratorio de Python, Mecánica de Sólidos
Auxiliar: Edward Alfonso Ramírez González
Docentes: Ing. Jose Miguel Benjumea Royero
Ing. David Sebastián Cotes Prieto

INSTRUCCIONES:

1. Realizar individualmente cada uno de los ejercicios con los valores en las unidades propuestas.

2. Guardar en una carpeta "**T1_Codigo_Apellido_Nombre**" las soluciones en archivos de Jupyter Notebook y con el nombre "**T1_ejercicio1**" y "**T1_ejercicio2**" según corresponda, de manera que los resultados se vean de forma explícita en el navegador o al descargar y ejecutar cada celda del archivo

3. Hacer un ***Pull Request*** en la carpeta ***/Semana5/Taller/Primera_Entrega*** en GitHub con la carpeta mencionada anteriormente

4. Fecha máxima de entrega: **15/01/2021**, hora límite **11:59 PM**.

5. El día **17/01/2022** se entregará la retroalimentación personal al correo institucional de cada estudiante. Tras esto, tendrán como fecha límite **22/01/2022**, hora límite **11:59 PM** para hacer nuevamente un *Pull Request* con el mismo nombre de carpeta y archivos en la carpeta ***/Semana5/Taller/Segunda_Entrega*** en GitHub. La nota definitiva del taller se enviará al correo institucional de cada estudiante el **20/01/2022**.

EJERCICIO 1 - (VALOR: 2.3 UNIDADES)

La barra rígida AB, mostrada en la *ilustración 1*, articulada en cada extremo, pende de tres cables de acero, cuyas longitudes y diámetros son 1.5 m y ¼", respectivamente. Los pasadores A y B se encuentran a cortante doble, mientras los C, D y E a cortante simple. Todos tienen ½" de diámetro.

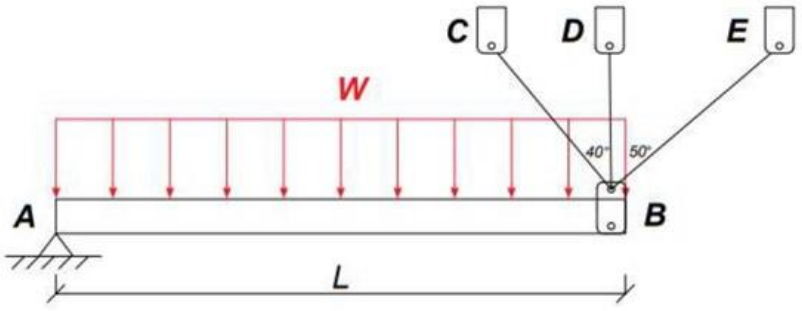


Ilustración 1 - Barra rígida AB apoyada en cables deformables. Fuente: Autor

Universidad Industrial de Santander
Escuela de Ingeniería Civil, 2021 - II
Laboratorio de Python, Mecánica de Sólidos
Auxiliar: Edward Alfonso Ramírez González
Docentes: Ing. Jose Miguel Benjumea Royero
Ing. David Sebastián Cotes Prieto

Todos los elementos se encuentran hechos de acero ASTM A-36, tal y sus propiedades se evidencian en la ilustración 2:

Material	Density kg/m³	Ultimate Strength Tension, MPa	Ultimate Strength Compression, MPa	Ultimate Strength Shear, MPa	Yield Strength Tension, MPa	Yield Strength Shear, MPa	Modulus of Elasticity, GPa	Modulus of Rigidity, GPa	Coefficient of Thermal Expansion, 10^{-6}/°C	Ductility, Percent Elongation in 50 mm
Steel										
Structural (ASTM-A36)	7860	400			250	145	200	77.2	11.7	21

Ilustración 2 - Propiedades mecánicas de ASTM A-36. Fuente: Mecánica de materiales. Beer and Johnston. Quinta Edición

Se sabe que la longitud de la barra, en metros, es **L = 0.2 * X**. Asumiendo un factor de seguridad de **1.X** respecto al esfuerzo de fluencia (*Yielding Strength*), y de **1.(X+3)** respecto al esfuerzo último (*Ultimate Strength*). Con base en esta información, evalúe las posibilidades de falla en la barra por esfuerzo normal y en los cables por esfuerzo normal (aplastamiento) y cortante, y así determinar mediante un algoritmo en Python:

1. El valor máximo que puede tener la carga distribuida ***W*** para que, con base en las posibilidades de falla enunciadas anteriormente, ninguno de los elementos (cables y pasadores) falle. ***(VALOR: 1.0)***
2. Con la carga ***W*** del numeral anterior, determine las reacciones, fuerzas y deformaciones de todos los cables. ***(VALOR: 1.0)***

Nota 1: Los cables SÓLO pueden trabajar a tracción.

Nota 2: **X** es igual a la sumatoria de los dígitos de su código estudiantil. Ejemplo: Para un código 2180140, **X** = 2+1+8+0+1+4+0 = 16.

EJERCICIO 2 - (VALOR: 2.3 UNIDADES)

Una armadura cuenta con dos apoyos, uno de primer y otro de segundo grado ubicados de forma ascendente en su extremo lateral izquierdo, como muestra la *Figura 2*. La fuerza P ubicada en el nodo C solicita la armadura y presenta un comportamiento algo especial: tiene una dirección variable entre 0° − 180° (medidos desde la línea de referencia vertical gris) y su magnitud aumenta 0.25 [kN] cada 10°, partiendo de 20 [kN] cuando su dirección es 0° (totalmente vertical con dirección hacia arriba).

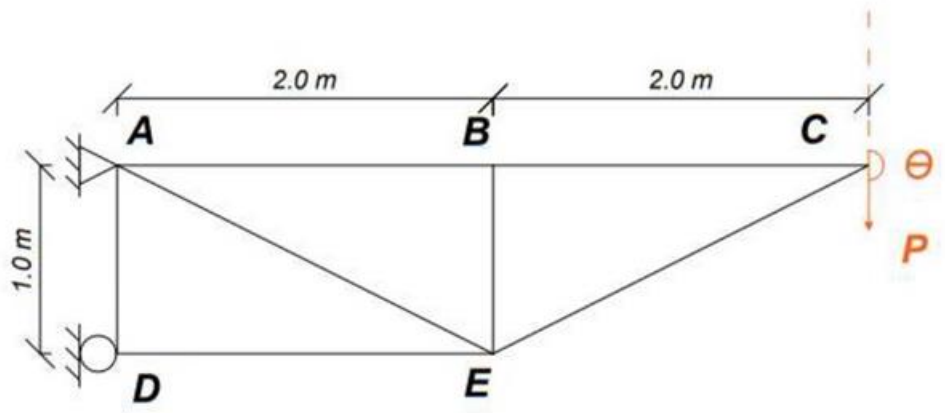


Ilustración 3. Esquema armadura. Fuente: Autor

Con base a la información presentada, desarrolle un programa de software en Python que responda las siguientes preguntas:

1. ¿Qué ángulo genera la máxima fuerza interna en cada elemento?. Generar una matriz donde se especifique el ángulo, el valor de la fuerza P y la magnitud de la fuerza interna máxima generada en el elemento. NOTA: el orden de los elementos es el siguiente: AB, BC, CD, DE, AE, AD, BD.

2. Teniendo en cuenta que el área de la sección transversal de los elementos es de 300 mm^2 y el esfuerzo normal admisible del material es de 15 $[MPa]$, responda:
   - ¿Para cuál dirección de la fuerza P se presenta la primera falla? Imprima la carga y la dirección
   - ¿Qué elemento falla y qué tipo de esfuerzo induce dicha falla? Imprima el nombre del elemento y su fuerza interna.

3. ¿Qué pasaría con las fuerzas internas de los elementos si se desplaza el nodo B hacia la derecha en el eje x con una magnitud de 0,0**L** [m], para cada variación de la carga P? Evidencie en un *array* el nombre del elemento, su fuerza interna y la carga P que fue aplicada. Ejemplo:

   array([["AB", -5kN, 20kN], ["BC", 3kN, 20kN], …, ["BD", -10kN, 24.5kN]])

4. Elaborar las gráficas de las armaduras para cada una de las iteraciones del ítem **3.**

5. Elaborar una gráfica de posición en el eje ***x*** del nodo B vs esfuerzo normal del elemento. Es decir, evidenciar en un *plot* el cambio en el esfuerzo al que se somete un elemento individual de la armadura para cada variación de la posición del nodo B. Realizar un *plot* con cada barra de la armadura, para un total de siete gráficos y colocar los siete gráficos dentro de un *subplots* de siete filas y una columna. Tome una carga P constante para cada variación de posición y $\boldsymbol{AB}°$ como su ángulo de aplicación con respecto a la vertical.

NOTA:

El valor de **L** corresponde al quinto dígito de su código estudiantil, el valor de **A** al sexto dígito y el valor de **B** al séptimo dígito.

Ejemplo:

Código estudiantil: 2170890

L:8, A:9, B:0

Aproxime el ángulo $\boldsymbol{AB}°$ en unidades de 10. Ejemplo: $\boldsymbol{AB}°$ de 95° a 100°

ENCUESTA DE SATISFACCIÓN (VALOR: 0.4 UNIDADES):

Diligencie la encuesta del siguiente link: https://forms.gle/d7vy9SK1v9C799cL6. Esto con el fin de evaluar el desempeño del curso y su satisfacción hasta este momento con respecto a la metodología usada. El formulario tendrá fecha límite hasta el mismo día de entrega del taller y su diligenciamiento otorga 0.5 unidades extra a la nota total de esta actividad.

Apéndice E. *Solución Primer Taller, Primer Ejercicio*

Solución Taller 1 - Ejercicio 1

In [1]:
```
import numpy as np
import sympy as sp
```

In [2]:
```
# metros
longitud_cables = 1.5
# pulgadas
diametro_cables = 1 / 4
# mm 2
area_cables = np.pi * (diametro_cables * 25.4 / 1000) ** 2 / 4
# pulgadas
diametro_pasadores = 1 / 2
# mm 2
area_pasadores = np.pi * (diametro_pasadores * 25.4 / 1000) ** 2 / 4

codigo = "2170249"
X = sum([int(i) for i in codigo])
# metros
L = 0.2 * X

FS_fluencia = float(f"1.{X}")
FS_resistencia_ultima = float(f"1.{X + 3}")

angulo_BC = 40 * np.pi / 180
angulo_BE = 50 * np.pi / 180
```

In [3]:
```
# propiedades material
# MPa
esfuerzo_ultimo = 400
esfuerzo_fluencia_traccion = 250
esfuerzo_fluencia_corte = 145
# GPa
E = 200
```

In [4]:
```
# Equilibrio
Ax, Ay, BC, BD, BE, W = sp.symbols('Ax Ay BC BD BE W')

Fx = Ax - BC * sp.sin(angulo_BC) + BE * sp.sin(angulo_BE)
equilibrio_X = sp.Eq(Fx, 0)
equilibrio_X
```

Out[4]: $$Ax - 0.642787609686539BC + 0.766044443118978BE = 0$$

In [5]:
```
Fy = Ay - W * L + BC * sp.cos(angulo_BC) + BE * sp.cos(angulo_BE) + BD
equilibrio_Y = sp.Eq(Fy, 0)
equilibrio_Y
```

Out[5]: $$Ay + 0.766044443118978BC + BD + 0.642787609686539BE - 5.0W = 0$$

In [6]:
```
M_a = - W * L ** 2 / 2 + BC * sp.cos(angulo_BC) * L + BE * sp.cos(angulo_BE) * L + BD * L
equilibrio_momentos = sp.Eq(M_a, 0)
equilibrio_momentos
```

Out[6]: $$3.83022221559489BC + 5.0BD + 3.2139380484327BE - 12.5W = 0$$

In [7]:
```
# por compatibilidad de desplazamientos
```

```
equivalencia_BC = sp.Eq(BC - BD * sp.cos(angulo_BC), 0)
expresion_equivalencia_BC = BD * sp.cos(angulo_BC)
equivalencia_BC
```

Out[7]: $BC - 0.766044443118978BD = 0$

In [8]:
```
equivalencia_BE = sp.Eq(BE - BD * sp.cos(angulo_BE), 0)
expresion_equivalencia_BE = BD * sp.cos(angulo_BE)
equivalencia_BE
```

Out[8]: $-0.642787609686539BD + BE = 0$

In [9]:
```
# obtener BD del equilibrio de momentos
ecuacion = sp.Eq(M_a.subs([(BC, expresion_equivalencia_BC), (BE, expresion_equivalencia_BE)])
ecuacion
```

Out[9]: $10.0BD - 12.5W = 0$

In [13]:
```
BD_vs_W = sp.solve(ecuacion)[0][BD]
BD_vs_W
```

Out[13]: $1.25W$

In [53]:
```
F_BC = expresion_equivalencia_BC.subs(BD, BD_vs_W)
F_BE = expresion_equivalencia_BE.subs(BD, BD_vs_W)

fuerzas_cables = [F_BC, BD_vs_W, F_BE]
fuerzas_cables
```

Out[53]:
```
[0.957555553898722*W, 1.25*W, 0.803484512108174*W]
```

In [74]:
```
w_max = []
for fuerza in fuerzas_cables:
    # analisis de esfuerzo normal en cables
    ecuacion_esfuerzo_normal = sp.Eq(fuerza / area_cables, esfuerzo_ultimo * 10 ** 6 / FS_res
    w_cables = sp.solve(ecuacion_esfuerzo_normal)[0] / 1000
    # cortante simple en pasadores superiores
    ecuacion_esfuerzo_cortante = sp.Eq(fuerza / area_pasadores, esfuerzo_fluencia_corte * 10
    w_pasadores = sp.solve(ecuacion_esfuerzo_cortante)[0] / 1000
    w_max.append(w_cables)
    w_max.append(w_pasadores)

print("Carga distribuida máxima en los cables [kN / m]")
min(w_max)
```

```
Carga distribuida máxima en los cables [kN / m]
```

Out[74]: 7.9173043608984

In [81]:
```
# cortante doble en los apoyos de la barra
Ay_eq = Fy.subs([(BC, expresion_equivalencia_BC), (BE, expresion_equivalencia_BE), (BD, BD_vs
Ay_vs_W = sp.solve(sp.Eq(Ay_eq, 0))
Ay_vs_W = Ay_vs_W[0][Ay]
Ay_vs_W
```

Out[81]: $2.5W$

In [91]:
```
Ax_eq = Fx.subs([(BC, expresion_equivalencia_BC), (BE, expresion_equivalencia_BE), (BD, BD_vs
Ax_vs_W = sp.solve(sp.Eq(Ax_eq, 0))
Ax_vs_W = Ax_vs_W[0][Ax]
Ax_vs_W
```

Out[91]: $-1.38777878078145 \cdot 10^{-16} W$

In [93]:
```
# fuerza en el apoyo A
F_apoyo_A = sp.sqrt(Ay_vs_W ** 2 + Ax_vs_W ** 2)
F_apoyo_A
```

Out[93]: $2.5\sqrt{W^2}$

In [100…
```
# fuerza en el punto de conexión de los cables
F_x_cortante_doble = expresion_equivalencia_BE.subs(BD, BD_vs_W) * sp.sin(angulo_BE) - expres
F_y_cortante_doble = expresion_equivalencia_BE.subs(BD, BD_vs_W) * sp.cos(angulo_BE) + expres
F_apoyo_B = sp.sqrt(F_x_cortante_doble ** 2 + F_y_cortante_doble ** 2)
F_apoyo_B
```

Out[100]: $2.5\sqrt{W^2}$

In [104…
```
fuerzas_apoyos = [F_apoyo_A, F_apoyo_B]
w_max_apoyos = []
for fuerza in fuerzas_apoyos:
    ecuacion_esfuerzo_cortante = sp.Eq(fuerza / area_pasadores, esfuerzo_fluencia_corte * 10
    w_pasadores = sp.solve(ecuacion_esfuerzo_cortante)[0] / 1000
    w_max_apoyos.append(w_pasadores)

print("Carga distribuida máxima por pasadores [kN / m]")
carga_maxima = min(w_max)
carga_maxima
```

```
Carga distribuida máxima por pasadores [kN / m]
```

Out[104]: 7.9173043608984

Cálculo de fuerzas y deformaciones

In [132…
```
valor_BE = expresion_equivalencia_BE.subs([(BD, BD_vs_W), (W, carga_maxima)])
deformacion_BE = (valor_BE * 1000 * longitud_cables) / (area_cables * E * 10 ** 9)
# fuerza en kN y deformacion en mm
valor_BE, deformacion_BE * 1000
```

Out[132]: (6.36143143162837, 1.50653346020283)

In [131…
```
valor_BC = expresion_equivalencia_BC.subs([(BD, BD_vs_W), (W, carga_maxima)])
deformacion_BC = (valor_BC * 1000 * longitud_cables) / (area_cables * E * 10 ** 9)
valor_BC, deformacion_BC * 1000
```

Out[131]: (7.58125876268484, 1.79541666356010)

In [130…
```
valor_BD = BD_vs_W.subs(W, carga_maxima)
deformacion_BD = (valor_BD * 1000 * longitud_cables) / (area_cables * E * 10 ** 9)
valor_BD, deformacion_BD * 1000
```

Out[130]: (9.89663045112300, 2.34375000000000)

In [121…
```
valor_Ax = Fx.subs([(BE, valor_BE), (BC, valor_BC)])
valor_Ax = sp.solve(sp.Eq(valor_Ax, 0))[0]
valor_Ax
```

Out[121]: $-8.88178419700125 \cdot 10^{-16}$

In [124…
```
valor_Ay = Fy.subs([(BE, valor_BE), (BC, valor_BC), (BD, valor_BD), (W, carga_maxima)])
valor_Ay = sp.solve(sp.Eq(valor_Ay, 0))[0]
valor_Ay
```

Out[124]: 19.793260902246

Apéndice F. *Solución Primer Taller, Segundo Ejercicio*

Solución Taller 1 - Ejercicio 2

In [1]:
```
import numpy as np
```

In [2]:
```
carga_inicial = 20
diferencial_de_carga = 0.25
carga_final = carga_inicial + 18 * diferencial_de_carga

distancia_AD = 1
distancia_AB = 2
distancia_BC = 2
distancia_AC = distancia_AB + distancia_BC

angulo_barra_inclinada = np.arctan(distancia_AD / distancia_BC)
angulo_barra_inclinada_AE = np.arctan(distancia_AD / distancia_AB)
```

In [3]:
```
rango_de_angulos = np.arange(0, 190, 10)
rango_de_cargas = np.arange(carga_inicial, carga_final + diferencial_de_carga, diferencial_de
rango_de_cargas
```

Out[3]:
```
array([20.  , 20.25, 20.5 , 20.75, 21.  , 21.25, 21.5 , 21.75, 22.  ,
       22.25, 22.5 , 22.75, 23.  , 23.25, 23.5 , 23.75, 24.  , 24.25,
       24.5 ])
```

In [46]:
```
def calcular_reacciones(carga, angulo, distancia_AC):
    angulo_radianes = (angulo if angulo <= 90 else 180 - angulo) * np.pi / 180
    Ay = (-1 if angulo <= 90 else 1) * carga * np.cos(angulo_radianes)
    Dx = (-1 if angulo <= 90 else 1) * distancia_AC * carga * np.cos(angulo_radianes)
    Ax = - Dx - carga * np.sin(angulo_radianes)
    return Ax, Ay, Dx
```

In [56]:
```
def obtener_fuerzas_nodo_C(carga, angulo):
    angulo_radianes = (angulo if angulo <= 90 else 180 - angulo) * np.pi / 180
    CE = (1 if angulo <= 90 else -1) * carga * np.cos(angulo_radianes) / np.sin(angulo_barra_
    BC = carga * np.sin(angulo_radianes) - CE * np.cos(angulo_barra_inclinada)
    return CE, BC

def obtener_fuerzas_nodo_B(fuerza_BC):
    BA = fuerza_BC
    BE = 0
    return BA, BE

def obtener_fuerzas_nodo_E(fuerza_CE, angulo_barra_inclinada_AE):
    EA = - fuerza_CE * np.sin(angulo_barra_inclinada) / np.sin(angulo_barra_inclinada_AE)
    ED = - EA * np.cos(angulo_barra_inclinada_AE) + fuerza_CE * np.cos(angulo_barra_inclinada
    return EA, ED

def obtener_fuerzas_nodo_D(fuerza_Dx):
    ED = - fuerza_Dx
    DA = 0
    return ED, DA



def obtener_fuerzas_nodo_A(Ax, Ay, angulo_barra_inclinada_AE):
    AE = Ay / np.sin(angulo_barra_inclinada_AE)
    AB = - Ax - AE * np.cos(angulo_barra_inclinada_AE)
    return AE, AB
```

In [42]:
```
fuerza_maxima = np.array([0, 0, 0, 0, 0, 0, 0])
for angulo, carga in zip(rango_de_angulos, rango_de_cargas):
    Ax, Ay, Dx = calcular_reacciones(carga, angulo, distancia_AC)
    CE, BC = obtener_fuerzas_nodo_C(carga, angulo)
    BA, BE = obtener_fuerzas_nodo_B(BC)
    EA, ED = obtener_fuerzas_nodo_E(CE, angulo_barra_inclinada_AE)
    DE, DA = obtener_fuerzas_nodo_D(Dx)
    AE, AB = obtener_fuerzas_nodo_A(Ax, Ay, angulo_barra_inclinada_AE)
    print(f"Reacciones angulo {angulo} grados: ")
    print(Ax, Ay, Dx)
    print("Fuerzas internas:")
    print(BA, EA, DA, BC, BE, CE, ED)
    print(f"{AE} == {EA}, {AB} == {BA}, {DE} == {ED}")
    print()
    fuerza_maxima[0] = BA if np.absolute(BA) > fuerza_maxima[0] else fuerza_maxima[0]
    fuerza_maxima[1] = BC if np.absolute(BC) > fuerza_maxima[1] else fuerza_maxima[1]
    fuerza_maxima[2] = CE if np.absolute(CE) > fuerza_maxima[2] else fuerza_maxima[2]
    fuerza_maxima[3] = ED if np.absolute(ED) > fuerza_maxima[3] else fuerza_maxima[3]
    fuerza_maxima[4] = EA if np.absolute(EA) > fuerza_maxima[4] else fuerza_maxima[4]
    fuerza_maxima[5] = DA if np.absolute(DA) > fuerza_maxima[5] else fuerza_maxima[5]
    fuerza_maxima[6] = BE if np.absolute(BE) > fuerza_maxima[6] else fuerza_maxima[6]

print(fuerza_maxima)
```

Segundo punto

In [11]:
```
# MPa / N / mm 2
esfuerzo_admisible = 21
# mm 2
area = 1200
```

In [12]:
```
# KN
fuerza_maxima = esfuerzo_admisible * area / 1000
fuerza_maxima
```

Out[12]:
```
25.2
```

In [36]:
```
fuerzas_internas = np.array([0, 0, 0, 0, 0, 0, 0])
for angulo, carga in zip(rango_de_angulos, rango_de_cargas):
    Ax, Ay, Dx = calcular_reacciones(carga, angulo)
    CE, BC = obtener_fuerzas_nodo_C(carga, angulo)
    BA, BE = obtener_fuerzas_nodo_B(BC)
    EA, ED = obtener_fuerzas_nodo_E(CE)
    DE, DA = obtener_fuerzas_nodo_D(Dx)
    AE, AB = obtener_fuerzas_nodo_A(Ax, Ay)
    fuerzas_internas = np.array([("AB", BA), ("AE", EA), ("AD", DA), ("BC", BC), ("BE", BE),
    elementos_que_fallan = np.where(np.absolute(np.array([x[1] for x in fuerzas_internas], dt
    if len(elementos_que_fallan) > 0:
        print(f"La primera falla se presenta bajo la carga de {carga} kN y la direccion de {a
        for i in elementos_que_fallan[0]:
            fuerza = fuerzas_internas[i][1].astype(float)
            causa_falla = "compresión" if fuerza < 0 else "tracción"
            print(f"El elemento {fuerzas_internas[i][0]} falla bajo la carga {np.absolute(fue
        print()
```

Tercer Punto

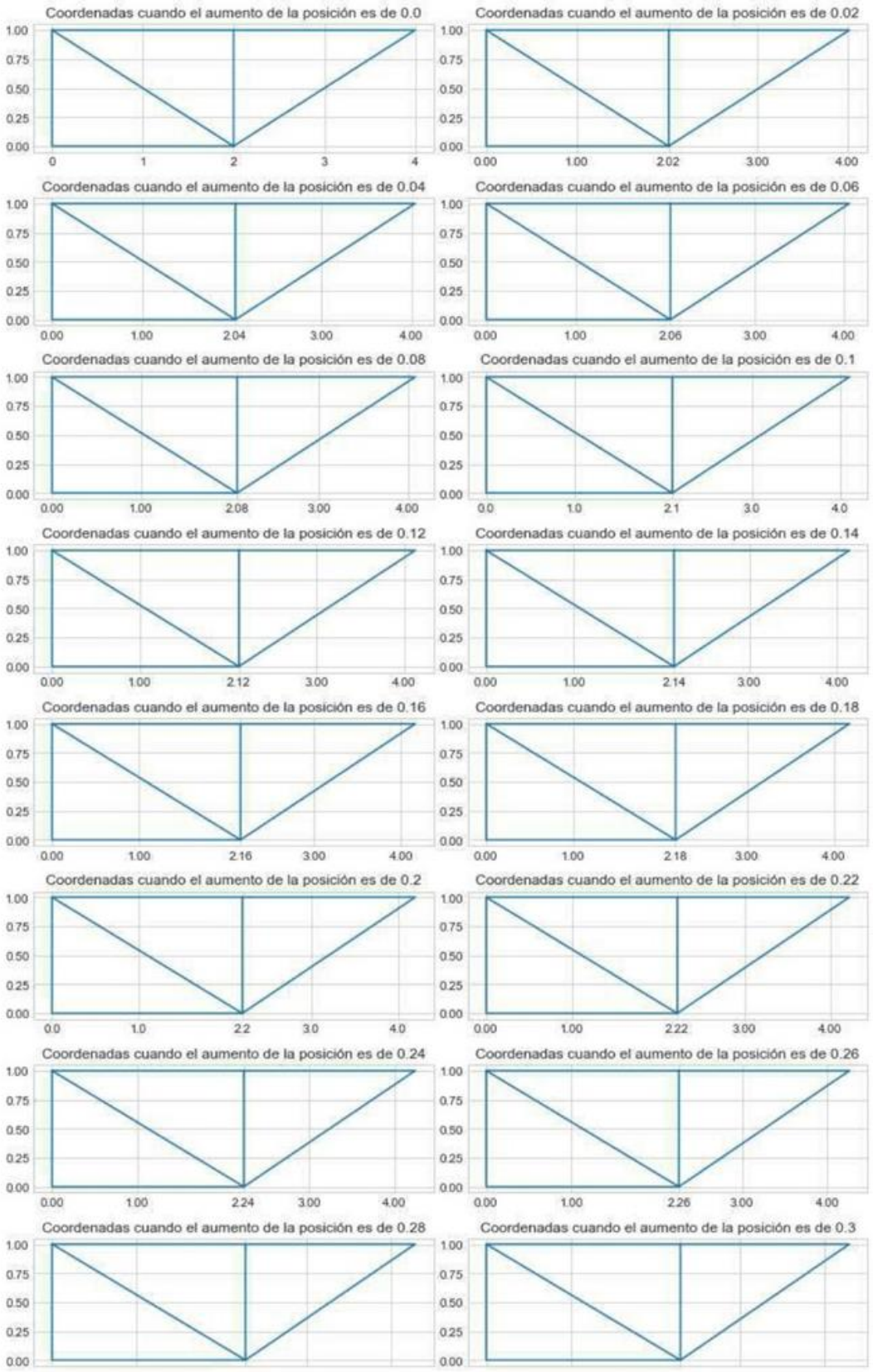
In [79]:
```
L = 2
aumento_distancia = float("0.0" + str(L))
A = 4
B = 9

AB = 49
AB = round(AB / 10) * 10
AB, aumento_distancia
```

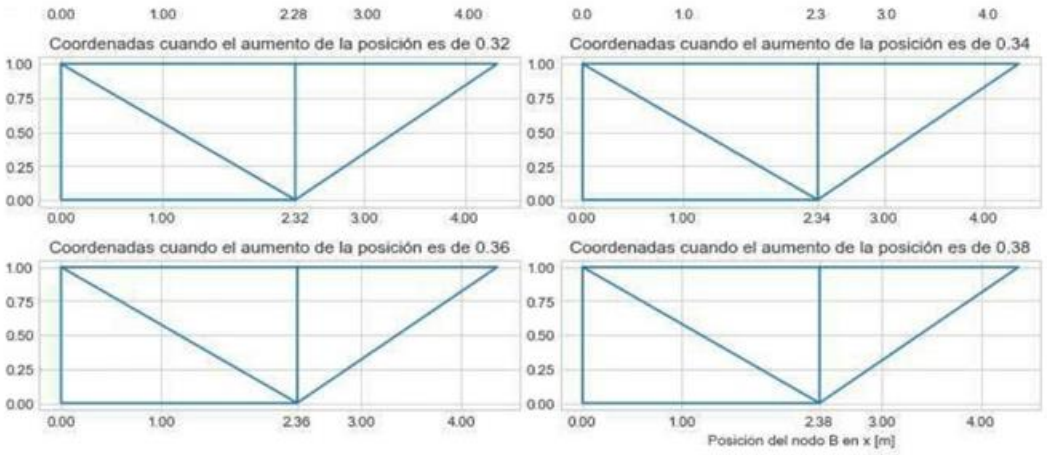
Out[79]:
```
(50, 0.02)
```

In [61]:
```
for angulo, carga in zip(rango_de_angulos, rango_de_cargas):
    distancia_AC += aumento_distancia
    distancia_AB += aumento_distancia
    angulo_barra_inclinada_AE = np.arctan(distancia_AD / distancia_AB)
    Ax, Ay, Dx = calcular_reacciones(carga, angulo, distancia_AC)
    CE, BC = obtener_fuerzas_nodo_C(carga, angulo)
    BA, BE = obtener_fuerzas_nodo_B(BC)
    EA, ED = obtener_fuerzas_nodo_E(CE, angulo_barra_inclinada_AE)
    DE, DA = obtener_fuerzas_nodo_D(Dx)
    AE, AB = obtener_fuerzas_nodo_A(Ax, Ay, angulo_barra_inclinada_AE)
    fuerzas_internas = [("AB", BA, carga), ("AE", EA, carga), ("AD", DA, carga), ("BC", BC, c
                        ("CE", CE, carga), ("ED", ED, carga)]
    print(f"Para un aumento de distancia de {distancia_AB - 2} las fuerzas calculadas son:")
    print(fuerzas_internas)
    print()
```

In [26]:
```
# CUARTO PUNTO
variaciones_distancia = np.arange(2, 2.02 + 19 * 0.02, 0.02)

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,20)
plt.style.use('seaborn-whitegrid')

fig, axes = plt.subplots(nrows=10, ncols=2)


distancia_AD = 1
distancia_BC = 2
for distancia_AB, ax in zip(variaciones_distancia, np.array(axes).flatten()):
    distancia_AC = distancia_AB + distancia_BC
    coordenadas_x = np.array([
        0, distancia_AB, distancia_AC, distancia_AB, 0, 0, distancia_AB, distancia_AB
    ])
    coordenadas_Y = np.array([
        distancia_AD, distancia_AD, distancia_AD, 0, 0, distancia_AD, 0, distancia_AD
    ])
    ax.plot(coordenadas_x, coordenadas_Y)
    ax.set_xticks([distancia_AB])
    ax.set_xticks([0, 1, distancia_AB, 3, 4])
    ax.set_title(f'Coordenadas cuando el aumento de la posición es de {round(distancia_AB - 2
ax.set_xlabel('Posición del nodo B en x [m]')
plt.tight_layout()
plt.show
```

Out[26]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```


Coordenadas cuando el aumento de la posición es de 0.0
Coordenadas cuando el aumento de la posición es de 0.02
Coordenadas cuando el aumento de la posición es de 0.04
Coordenadas cuando el aumento de la posición es de 0.06
Coordenadas cuando el aumento de la posición es de 0.08
Coordenadas cuando el aumento de la posición es de 0.1
Coordenadas cuando el aumento de la posición es de 0.12
Coordenadas cuando el aumento de la posición es de 0.14
Coordenadas cuando el aumento de la posición es de 0.16
Coordenadas cuando el aumento de la posición es de 0.18
Coordenadas cuando el aumento de la posición es de 0.2
Coordenadas cuando el aumento de la posición es de 0.22
Coordenadas cuando el aumento de la posición es de 0.24
Coordenadas cuando el aumento de la posición es de 0.26
Coordenadas cuando el aumento de la posición es de 0.28
Coordenadas cuando el aumento de la posición es de 0.3



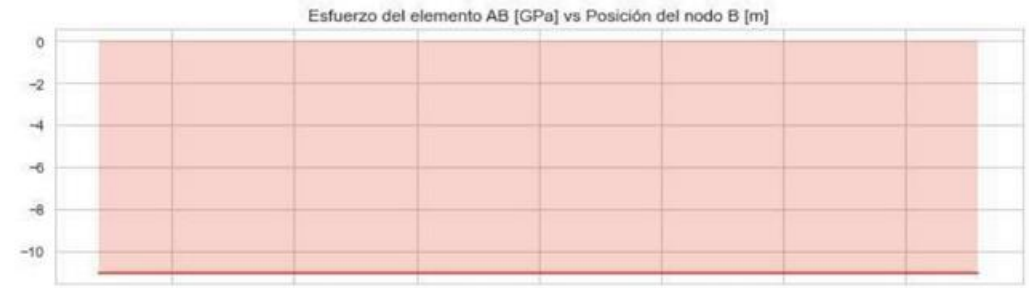
In [81]:
```
# QUINTO PUNTO
variaciones_distancia = np.arange(2.02, 2.02 + 19 * 0.02, 0.02)
fuerzas_elemento_AB = np.array([])
fuerzas_elemento_AE= np.array([])
fuerzas_elemento_AD = np.array([])
fuerzas_elemento_BC = np.array([])
fuerzas_elemento_BE = np.array([])
fuerzas_elemento_CE = np.array([])
fuerzas_elemento_ED = np.array([])
counter = 0

for distancia_AB in variaciones_distancia:
    carga = 21.25
    angulo = 50
    distancia_AC = distancia_AB + distancia_BC
    angulo_barra_inclinada_AE = np.arctan(distancia_AD / distancia_AB)
    Ax, Ay, Dx = calcular_reacciones(carga, angulo, distancia_AC)
    CE, BC = obtener_fuerzas_nodo_C(carga, angulo)
    BA, BE = obtener_fuerzas_nodo_B(BC)
    EA, ED = obtener_fuerzas_nodo_E(CE, angulo_barra_inclinada_AE)
    DE, DA = obtener_fuerzas_nodo_D(Dx)
    AE, AB = obtener_fuerzas_nodo_A(Ax, Ay, angulo_barra_inclinada_AE)
    fuerzas_elemento_AB = np.append(fuerzas_elemento_AB, BA)
    fuerzas_elemento_AE = np.append(fuerzas_elemento_AE, EA)
    fuerzas_elemento_AD = np.append(fuerzas_elemento_AD, DA)
    fuerzas_elemento_BC = np.append(fuerzas_elemento_BC, BC)
    fuerzas_elemento_BE = np.append(fuerzas_elemento_BE, BE)
    fuerzas_elemento_CE = np.append(fuerzas_elemento_CE, CE)
    fuerzas_elemento_ED = np.append(fuerzas_elemento_ED, ED)
    if counter == 5:
        print(f"Reacciones angulo {angulo} grados: ")
        print(Ax, Ay, Dx)
        print("Fuerzas internas:")
        print(BA, EA, DA, BC, BE, CE, ED)
    counter += 1
```

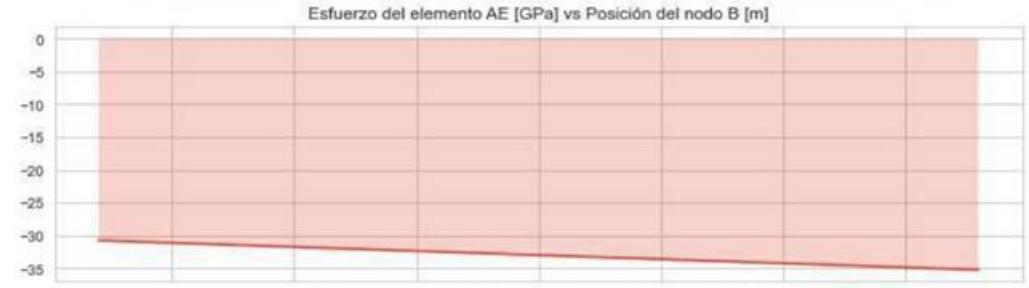
```
Reacciones angulo 50 grados:
39.99761081177824 -13.659236705838962 -56.27605522805652
Fuerzas internas:
-11.040028995399641 -32.017437312164745 0 -11.040028995399641 0 30.542981795016217 56.2760552
2805653
```

In [87]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,20)
```

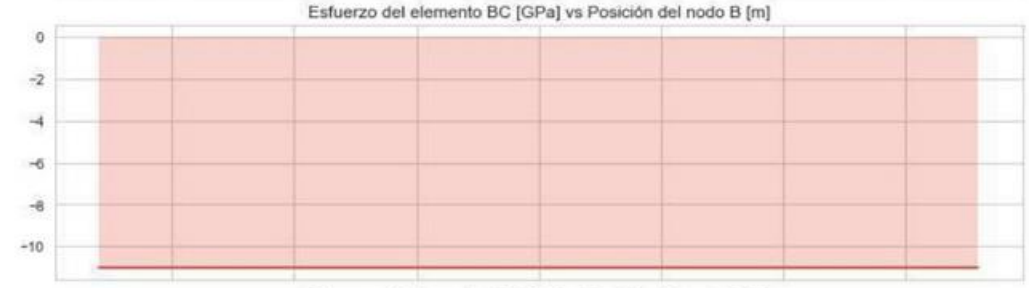
```
plt.style.use('seaborn-whitegrid')
fuerzas_elementos = [
    (fuerzas_elemento_AB, "AB"),
    (fuerzas_elemento_AE, "AE"),
    (fuerzas_elemento_AD, "AD"),
    (fuerzas_elemento_BC, "BC"),
    (fuerzas_elemento_BE, "BE"),
    (fuerzas_elemento_CE, "CE"),
    (fuerzas_elemento_ED, "ED")
]
fig, axes = plt.subplots(nrows=7, ncols=1, sharex=True)



for fuerzas, ax in zip(fuerzas_elementos, axes):
    ax.plot(variaciones_distancia, fuerzas[0])
    ax.fill_between(variaciones_distancia, fuerzas[0], alpha=0.25)
    ax.set_title(f'Esfuerzo del elemento {fuerzas[1]} [GPa] vs Posición del nodo B [m]')
ax.set_xlabel('Posición del nodo B en x [m]')
plt.tight_layout()
plt.show
```

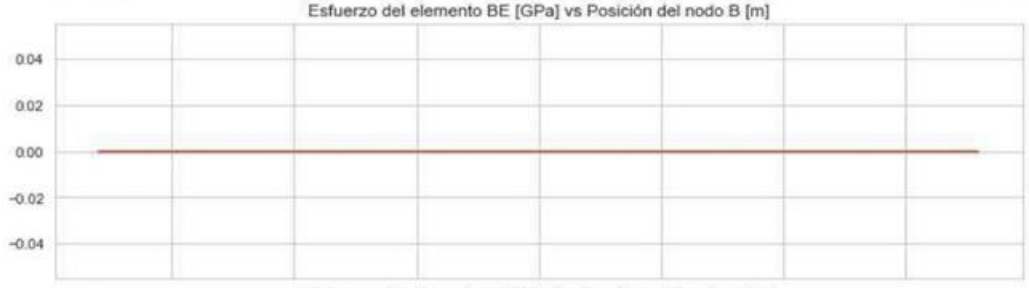
Out[87]: <function matplotlib.pyplot.show(close=None, block=None)>

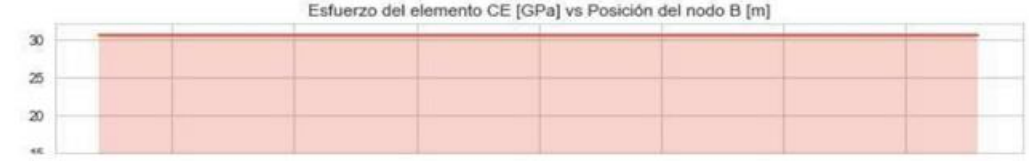

Esfuerzo del elemento AB [GPa] vs Posición del nodo B [m]
0
−2
−4
−6
−8
−10

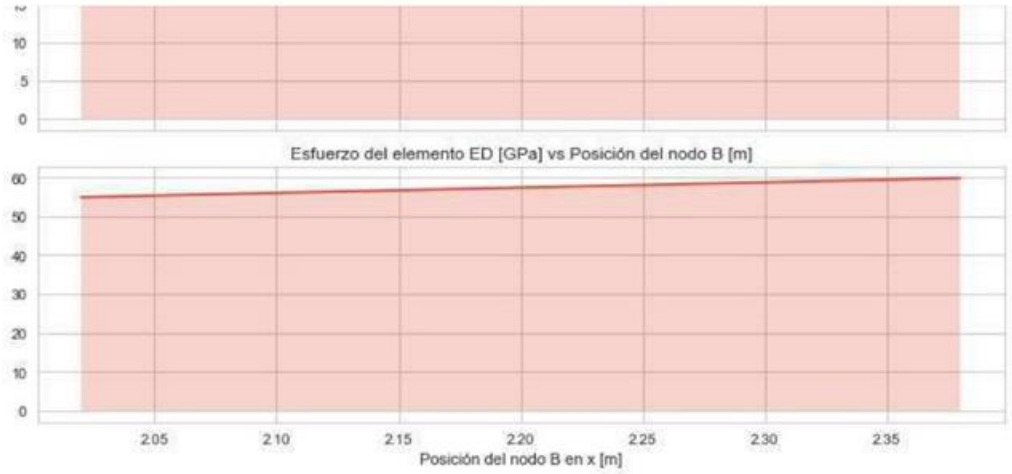

Esfuerzo del elemento AE [GPa] vs Posición del nodo B [m]
0
−5
−10
−15
−20
−25
−30
−35


Esfuerzo del elemento AD [GPa] vs Posición del nodo B [m]
0.04
0.02
0.00
−0.02
−0.04


Esfuerzo del elemento BC [GPa] vs Posición del nodo B [m]
0
−2
−4
−6
−8
−10


Esfuerzo del elemento BE [GPa] vs Posición del nodo B [m]
0.04
0.02
0.00
−0.02
−0.04


Esfuerzo del elemento CE [GPa] vs Posición del nodo B [m]
30
25
20


10
5
0
Esfuerzo del elemento ED [GPa] vs Posición del nodo B [m]
60
50
40
30
20
10
0
2.05
2.10
2.15
2.20
2.25
2.30
2.35
Posición del nodo B en x [m]

Apéndice G. *Solución Primer Taller, Propuesta por Estudiante, Primer Ejercicio*

Solución Taller 1 - Ejercicio 1

Propuesta por estudiante

In [2]:
```
#importar las bibliotecas que utilizaré
import numpy as np
import sympy as sp
import heapq as he
import math as mt
from math import pi,cos,sin
```

In [3]:
```
print("ENCISO A")
#Determinar mi valor para X y agrupar el resto de datos que necesito usar
Codigo = "2,2,0,0,2,6,9"
Lista = Codigo.split(",")
X = int(Lista[0]) + int(Lista[1]) + int(Lista[2]) + int(Lista[3]) + int(Lista[4]) + int(Lista

"Longitud del cable en metros"
LC = 1.5

"Ángulo Cable BC en radianes"
ANG_BC = mt.radians(40)

"Ángulo Cable BC en radianes"
ANG_BE = mt.radians(50)

"Diametro cables en pulgadas"
dc = 1/4

"Área de los cables en mm^2"
AC = (pi/4)*(dc*25.4)**2

"Diametro pasadores A, B, C, D, E en pulgadas"
dP = 1/2

"Área pasadores en mm^2"
AP = (pi/4)*(dP*25.4)**2

"Esfuerzo último a tensión en MPa"
UT = 400

"Esfuerzo de fluencia a tensión en MPa"
FT = 250

"Esfuerzo de fluencia a cortante en MPa"
FC = 145

"Módulo de elasticidad en GPa"
E = 200

"Longitud de la barra en metros"
LB = 0.2 * X

"Factor de seguridad para la fluencia"
FSF = str(1) + str(".") + str(X)
FSF = float(FSF)

"Factor de seguridad para el esfuerzo último"
```

```
FSU = str(1) + str(".") + str(X+3)
FSU = float(FSU)
```

```
ENCISO A
```

In [10]:
```
"""
Determinar las posibles fuerzas que se pueden generar en los cables y pasadores para cumplir
Tener en cuenta que los pasadores A y B se encuentran a cortante doble, mientras que C, D, E
"""
#Definir simbolos para hacer más comodos los despejes
NC1, NC2 = sp.symbols("NC1 NC2")
VP1, VP2 = sp.symbols("VP1 VP2")
#En este como los cables tienen la misma área y son del mismo material, para todos se puede e

"Fuerza para cumplir con el esfuerzo último a tensión en los cables en N"
Esf_admi = NC1/AC
Esf_ulti = UT/FSU
Ecu = sp.Eq(Esf_admi,Esf_ulti)
Solución = sp.solve(Ecu)
Nc1 = Solución[0]
f"Nc1 {Nc1} N"
```

Out[10]:
```
'Nc1 10215.8765947076 N'
```

In [11]:
```
"Fueza para cumplir con el esfuerzo de fluencia a tensión en los cables en N"
#Se usa la fluencia a tensión ya que nos dicen que los cables solo trabajan a tensión
Esf_admi2 = NC2/AC
Esf_Fluen = FT/FSF
Ecu2 = sp.Eq(Esf_admi2,Esf_Fluen)
Solución2 = sp.solve(Ecu2)
Nc2 = Solución2[0]
f"Nc2 {Nc2} N"
```

Out[11]:
```
'Nc2 6543.22674454413 N'
```

In [12]:
```
#Todos los pasadores tienen la misma área y el mismo material, por ello solo es necesario cal
"Fuerza para cumplir con el esfuerzo de fluencia a cortante simple en los pasadores C, D, E"
Esf_admi3 = VP1/AP
Esf_cort1 = FC/FSF
Ecu3 = sp.Eq(Esf_admi3,Esf_cort1)
Solución3 = sp.solve(Ecu3)
Vp1 = Solución3[0]
f"Vp1 {Vp1} N"
```

Out[12]:
```
'Vp1 15180.2860473424 N'
```

In [13]:
```
"Fuerza para cumplir con el esfuerzo de fluencia a cortante doble en los pasadores A y B"
Esf_admi4= VP2/(2*AP)
Esf_cort2 = FC/FSF
Ecu4 = sp.Eq(Esf_admi4,Esf_cort2)
Solución4 = sp.solve(Ecu4)
Vp2= Solución4[0]
f"Vp2 {Vp2} N"
```

Out[13]:
```
'Vp2 30360.5720946849 N'
```

In [14]:
```
"""
Ahora agruparé los valores obtenidos en una lista con el fin de determinar el valor más peque
más pequeño, porque con esto me aseguro de no pasarme en el valor de ninguno de los esfuerzos
"""
print("Las fuerzas que cumplen con los esfuerzos propuestos son:")
print()
```

```
print([f"Nc1 = {Nc1}, Nc2 = {Nc2}, Vp1 = {Vp1}, Vp2 = {Vp2}"])
Fuerzas = [Nc1,Nc2,Vp1,Vp2]
```

```
ENCISO A
Las fuerzas que cumplen con los esfuerzos propuestos son:

['Nc1 = 10215.8765947076, Nc2 = 6543.22674454413, Vp1 = 15180.2860473424, Vp2 = 30360.5720946
849']
```

In [15]:
```
#usaré la función nsmallest para encontrar el valor más pequeño de fuerza dentro de la lista
Valor_small = he.nsmallest(1,Fuerzas)
Fuerza = Valor_small[0]
print()
print(f"La fuerza en los cables será de {Fuerza} N")
print()
```

```
La fuerza en los cables será de 6543.22674454413 N
```

In [16]:
```
"los tres cables tienen la misma fuerza, con ellas podemos utilizar la estática para determin
#Determinaré el GIE en primera instancia
Incognitas_A = 3
Ecuaciones_A = 3
print(f"GIE = {Incognitas_A - Ecuaciones_A} Estructura isostática")
```

```
GIE = 0 Estructura isostática
```

In [17]:
```
# W es una fuerza distribuida, por lo que para realizar momentos debo puntualizarla
W = sp.symbols("W")

#Realizo sumatoria de momentos en A
Sumatoria_A = -(W*LB)*(LB/2) + Fuerza*cos(ANG_BC)*LB + Fuerza*cos(ANG_BE)*LB + Fuerza*LB
Momento = sp.Eq(Sumatoria_A,0)
Sln = sp.solve(Momento)
w = Sln[0]*10**-3
print()
print(f"La fuerza distribuida máxima con la que ninguno de los elementos fallará es de {w} kN
```

```
La fuerza distribuida máxima con la que ninguno de los elementos fallará es de 7.505492529062
95 kN/m
```

In [19]:
```
print("ENCISO B")
```

```
ENCISO B
```

In [20]:
```
'''
Empezamos determinando el GIE de la estructura, se sabe que en A tenemos dos reacciones, mien
fuerzas provenientes de los cables BC,BD y BE
'''
Incognitas_B = 5
Ecuaciones_B = 3
print(f"GIE = {Incognitas_B - Ecuaciones_B} Estructura hiperestática, es necesario obtener 2
```

```
GIE = 2 Estructura hiperestática, es necesario obtener 2 ecuaciones de compatibilidad
```

In [22]:
```
'''
Ahora se deben obtener las ecuaciones de equilibrio de la estructura. Realizaré momento en A
en términos de las fuerzas de los cables
'''
NBC, NBD, NBE , Ax, Ay = sp.symbols("NBC NBD NBE Ax Ay")

#Sumatoria de momentos
Sumatoria_M = -(LB/2)*LB*w + NBC*cos(ANG_BC)*LB + NBD*LB + NBE*cos(ANG_BE)*LB
Ecu_M = sp.Eq(Sumatoria_M,0)
Ecu_M
```

Out[22]: $3.21738666109971NBC + 4.2NBD + 2.69970796068347NBE - 66.1984441063352 = 0$

In [23]:
```
#Sumatoria de fuerzas en x
Sumatoria_Fx = -Ax - NBC*sin(ANG_BC) + NBE*sin(ANG_BE)
Ecu_Fx = sp.Eq(Sumatoria_Fx,0)
Ecu_Fx
```

Out[23]: $-Ax - 0.642787609686539NBC + 0.766044443118978NBE = 0$

In [24]:
```
#Sumatoria de fuerzas en y
Sumatoria_Fy = Ay + NBC*cos(ANG_BC) + NBE*cos(ANG_BE) + NBD - w*LB
Ecu_Fy = sp.Eq(Sumatoria_Fy,0)
Ecu_Fy
```

Out[24]: $Ay + 0.766044443118978NBC + NBD + 0.642787609686539NBE - 31.5230686220644 = 0$

In [35]:
```
'''
Ahora se deben obtener las ecuaciones de compatibilidad para el sistema teniendo en cuenta qu
por ello solo rota a 90 grados de su eje. Del diagrama de willot se se obtiene
'''
#Ecuación para la deformación de los cables BE, BC y BD
Def_BE = (NBE*LC)/(AC*E)
Def_BC = (NBC*LC)/(AC*E)
Def_BD = (NBD*LC)/(AC*E)

"Como se habla de un mismo material y de la misma área, las ecuaciones se pueden simplificar,
Def_BEf = NBE
Def_BCf = NBC

#Ecuaciones para el desplazamiento del punto B
BB = Def_BEf/cos(ANG_BE)
BB2 = Def_BCf/cos(ANG_BC)
BB
```

Out[35]: $1.55572382686041NBE$

In [36]:
```
BB2
```

Out[36]: $1.30540728933228NBC$

In [37]:
```
print("Primera ecuación de compatibilidad")
'Se igualan BB y BB2 para obtener la cuarta ecuación del sistema'
Ecu_Com1 = sp.Eq(BB,BB2)
Ecu_Com1
```

Primera ecuación de compatibilidad

Out[37]: $1.55572382686041NBE = 1.30540728933228NBC$

In [39]:
```
print("Segunda ecuación de compatibilidad")
'''
Se hace necesario obtener una 5 ecuación, para esto se tiene en cuenta que el desplazamiento
deformación del cable BD
'''
Def_BDf = NBD
Ecu_Com2 = sp.Eq(Def_BD,BB)
Ecu_Com2
```

Segunda ecuación de compatibilidad

Out[39]: $0.000236823028966798NBD = 1.55572382686041NBE$

In [49]:
```
#Para desarrollar el sistema 5x5 resulta efectivo crear una matriz y un vector solución
print("Matriz con ecuaciones")
print('El orden será Ax-Ay-NBC-NBD-NBE')
Matriz = sp.Matrix([[ 0,  0,  3.21738666109971, 4.2, 2.69970796068347],
                    [-1, 0, -0.642787609686539, 0,  0.766044443118978],
                    [0,  1,  0.766044443118978, 1,  0.642787609686539],
                    [0,  0 ,-1.30540728933228,  0 , 1.55572382686041],
                    [0,  0,             0,      1, -1.55572382686041]])

Matriz
```

```
Matriz con ecuaciones
El orden será Ax-Ay-NBC-NBD-NBE
```

Out[49]:

$$\begin{bmatrix} 0 & 0 & 3.21738666109971 & 4.2 & 2.69970796068347 \\ -1 & 0 & -0.642787609686539 & 0 & 0.766044443118978 \\ 0 & 1 & 0.766044443118978 & 1 & 0.642787609686539 \\ 0 & 0 & -1.30540728933228 & 0 & 1.55572382686041 \\ 0 & 0 & 0 & 1 & -1.55572382686041 \end{bmatrix}$$

In [44]:
```
print("Vector solución")
Vector = sp.Matrix([66.1984441063352, 0, 31.5230686220644, 0, 0])
Vector
```

```
Vector solución
```

Out[44]:

$$\begin{bmatrix} 66.1984441063352 \\ 0 \\ 31.5230686220644 \\ 0 \\ 0 \end{bmatrix}$$

In [47]:
```
print("Solución")
Sln_Fuerzas = Matriz.LUsolve(Vector)
Sln_Fuerzas
```

```
Solución
```

Out[47]:

$$\begin{bmatrix} 1.06581410364015 \cdot 10^{-14} \\ 15.7615343110322 \\ 6.03701788699765 \\ 7.88076715551609 \\ 5.06565948239038 \end{bmatrix}$$

In [52]:
```
print("REACCIONES")
Ax = round(Sln_Fuerzas[0],3)
Ay = round(Sln_Fuerzas[1],3)
print(f"Ax = {Ax} kN  Ay = {Ay} kN")
```

```
REACCIONES
Ax = 0.0 kN  Ay = 15.762 kN
```

In [55]:
```
print("FUERZAS EN LOS CABLES")
NBCf = round(Sln_Fuerzas[2],3)
NBDf = round(Sln_Fuerzas[3],3)
NBEf = round(Sln_Fuerzas[4],3)
print(f"NBC = {NBCf} kN  NBD = {NBDf} kN  NBE = {NBEf} kN")
print()
```

```
FUERZAS EN LOS CABLES
NBC = 6.037 kN  NBD = 7.881 kN  NBE = 5.066 kN
```

In [62]:
```
'''
Para las deformaciónes se utilizan las ecuaciones de deformación antes de ser simplificadas,
igualar los desplazamientos del punto B
'''
print("DEFORMACIÓN EN LOS CABLES")
Def_BC = Def_BC.subs(NBC,NBCf)
Def_BD = Def_BD.subs(NBD,NBDf)
Def_BE = Def_BE.subs(NBE,NBEf)
print(f"Def_BC = {round(Def_BC,4)} mm  Def_BD = {round(Def_BD,4)} mm  Def_BE = {round(Def_BE,
print()
print(f"Recordar que la deformación del cable BD es igual al desplazamiento del punto B, es d
```

```
DEFORMACIÓN EN LOS CABLES
Def_BC = 0.0014 mm  Def_BD = 0.0019 mm  Def_BE = 0.0012 mm

Recordar que la deformación del cable BD es igual al desplazamiento del punto B, es decir 0.0
019 mm
```

Apéndice H. *Solución Primer Taller, Propuesta por Estudiante, Segundo Ejercicio*

Solución Taller 1 - Ejercicio 2

Propuesta por estudiante

In [2]:
```
#Primero debo importar las bibliotecas que serán de utilidad
import math as mt
from math import cos,sin,pi
import numpy as np
import sympy as sp
import heapq as he
import matplotlib.pyplot as plt
np.set_printoptions(precision=3,suppress=True)
```

In [3]:
```
print("PUNTO 1")
#Generar arrays con los posibles valores del ángulo y de la fuerza P
Ang_P = np.arange(0,181,10)
Valor_P = np.arange(20,24.75,0.25)
print()
print("ARRAY CON ÁNGULOS")
print (Ang_P)
print()
print("ARRAY CON CARGAS P")
print(Valor_P)
#Agrupar los datos conocidos
'Distancia vertical en metros'
Dis_AD = 1

'Distancias horizontales en metros'
Dis_AB = 2
Dis_AC = 4

'Ángulo BCE en radianes (Este ángulo es el mismo ADE)'
Ang_BCE = np.arctan(Dis_AD/Dis_AB)

#Definir simbolos para las fuerzas internas
FAB,FAE,FBC,FBE,FCE,FED,FDA = sp.symbols("FAB FAE FBC FBE FCE FED FDA")

#Como se trabaja con diferentes fuerzas, lo más adecuado es usar un bucle
Contador = 0
Fuerza_Px = np.array([])
Fuerza_Py = np.array([])

'Para ángulos entre 0 y 90 grados'
while Contador <= 9:
    if Ang_P[Contador] == 0:
        Fuerza_Py = np.append(Fuerza_Py,Valor_P[0])
        Fuerza_Px = np.append(Fuerza_Px,0)

    elif Ang_P[Contador] == 90:
        Fuerza_Px = np.append(Fuerza_Px,Valor_P[9])
        Fuerza_Py = np.append(Fuerza_Py,0)
    else:
        Px = Valor_P[Contador]*sin(mt.radians(Ang_P[Contador]))
        Py = Valor_P[Contador]*cos(mt.radians(Ang_P[Contador]))
        Fuerza_Px = np.append(Fuerza_Px,Px)
        Fuerza_Py = np.append(Fuerza_Py,Py)
    Contador = Contador + 1
```

```
'Para ángulos entre 90 y 180'
Contador2 = 10
while Contador2 < 19:
    if Ang_P[Contador2] == 180:
        Fuerza_Px = np.append(Fuerza_Px,0)
        Fuerza_Py = np.append(Fuerza_Py,Valor_P[18])
    else:
        Px2 = Valor_P[Contador2]*cos(mt.radians(Ang_P[Contador2]-90))
        Py2 = Valor_P[Contador2]*sin(mt.radians(Ang_P[Contador2]-90))
        Fuerza_Px = np.append(Fuerza_Px,Px2)
        Fuerza_Py = np.append(Fuerza_Py,Py2)
    Contador2 = Contador2 + 1
```

```
PUNTO 1

ARRAY CON ÁNGULOS
[  0  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170
 180]

ARRAY CON CARGAS P
[20.   20.25 20.5  20.75 21.   21.25 21.5  21.75 22.   22.25 22.5  22.75
 23.   23.25 23.5  23.75 24.   24.25 24.5 ]
```

In [4]:
```
print("COMPONENTES EN X DE LA FUERZA P DE 0 A 180 GRADOS")
Fuerza_Px
```

```
COMPONENTES EN X DE LA FUERZA P DE 0 A 180 GRADOS
```
Out[4]:
```
array([ 0.   ,  3.516,  7.011, 10.375, 13.499, 16.278, 18.62 , 20.438,
       21.666, 22.25 , 22.158, 21.378, 19.919, 17.811, 15.106, 11.875,
        8.208,  4.211,  0.   ])
```

In [5]:
```
print("COMPONENTES EN y DE LA FUERZA P DE 0 A 180 GRADOS")
Fuerza_Py
```

```
COMPONENTES EN y DE LA FUERZA P DE 0 A 180 GRADOS
```
Out[5]:
```
array([20.   , 19.942, 19.264, 17.97 , 16.087, 13.659, 10.75 ,  7.439,
        3.82 ,  0.   ,  3.907,  7.781, 11.5  , 14.945, 18.002, 20.568,
       22.553, 23.882, 24.5  ])
```

In [6]:
```
#Para encontrar las fuerzas internas utilizaré el método de nodos, teniendo en cuenta que tod
'Nodo C'
CE = np.array([])
BC = np.array([])
Contador3 = 0

'Para ángulos entre 0 y 90 grados'
while Contador3 <= 9:
    if Contador3 == 0:
        Sum_Cx = -FBC-FCE*cos(Ang_BCE)
        Ecu_C1 = sp.Eq(Sum_Cx,0)
        Sum_Cy = Fuerza_Py[Contador3] - FCE*sin(Ang_BCE)
        Ecu_C2 = sp.Eq(Sum_Cy,0)
        Fce = sp.solve(Ecu_C2)
        Fbc = sp.solve(Ecu_C1.subs(FCE,Fce[0]))
        CE = np.append(CE,round(Fce[0],3))
        BC = np.append(BC,round(Fbc[0],3))

    elif Contador3 == 9:
        Sum_Cx = -FBC + Fuerza_Px[9] - FCE*sin(Ang_BCE)
        Ecu_C1 = sp.Eq(Sum_Cx,0)
        Sum_Cy = -FCE*sin(Ang_BCE)
        Ecu_C2 = sp.Eq(Sum_Cy,0)
        Fce = sp.solve(Ecu_C2)
        Fbc = sp.solve(Ecu_C1.subs(FCE,Fce[0]))
```

```
        CE = np.append(CE,round(Fce[0],3))
        BC = np.append(BC,round(Fbc[0],3))
    else:
        Sum_Cx2 = -FBC + Fuerza_Px[Contador3] - FCE*cos(Ang_BCE)
        Ecu_C4 = sp.Eq(Sum_Cx2,0)
        Sum_Cy2 = Fuerza_Py[Contador3]-FCE*sin(Ang_BCE)
        Ecu_C5 = sp.Eq(Sum_Cy2,0)
        Fce2 = sp.solve(Ecu_C5)
        Fbc2 = sp.solve(Ecu_C4.subs(FCE,Fce2[0]))
        CE = np.append(CE,round(Fce2[0],3))
        BC = np.append(BC,round(Fbc2[0],3))
    Contador3 = Contador3 + 1

'Para ángulos entre 90 y 180'
Contador4 = 10
while Contador4 < 19:
    if Contador4 == 18:
        Sum_Cx3 = -FBC - FCE*cos(Ang_BCE)
        Ecu_C6 = sp.Eq(Sum_Cx3,0)
        Sum_Cy3 = -Fuerza_Py[Contador4]-FCE*sin(Ang_BCE)
        Ecu_C7 = sp.Eq(Sum_Cy3,0)
        Fce3 = sp.solve(Ecu_C7)
        Fbc3 = sp.solve(Ecu_C6.subs(FCE,Fce3[0]))
        CE = np.append(CE,round(Fce3[0],3))
        BC = np.append(BC,round(Fbc3[0],3))
    else:
        Sum_Cx4 = -FBC + Fuerza_Px[Contador4] - FCE*cos(Ang_BCE)
        Ecu_C7 = sp.Eq(Sum_Cx4,0)
        Sum_Cy4 = -Fuerza_Py[Contador4]-FCE*sin(Ang_BCE)
        Ecu_C8 = sp.Eq(Sum_Cy4,0)
        Fce4 = sp.solve(Ecu_C8)
        Fbc4 = sp.solve(Ecu_C7.subs(FCE,Fce4[0]))
        CE = np.append(CE,round(Fce4[0],3))
        BC = np.append(BC,round(Fbc4[0],3))
    Contador4 = Contador4 + 1
```

In [7]:
```
CE
```

Out[7]:
```
array([44.721, 44.592, 43.075, 40.182, 35.971, 30.543, 24.038, 16.634,
       8.542, 0.0, -8.737, -17.399, -25.715, -33.418, -40.254, -45.992,
       -50.429, -53.401, -54.784], dtype=object)
```

In [8]:
```
BC
```

Out[8]:
```
array([-40.0000000000000, -36.368, -31.516, -25.565, -18.675, -11.040,
       -2.880, 5.560, 14.025, 22.250, 29.972, 36.940, 42.919, 47.700,
       51.110, 53.011, 53.314, 51.974, 49.0000000000000], dtype=object)
```

In [9]:
```
"Por la distribución de los elementos en la armadura, es claro que el elemento BE es de fuerz
BE = np.zeros(19)
BE
```

Out[9]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0.])
```

In [10]:
```
'Nodo E'
#Por la distribución de los elementos en la armadura, es claro que el elemento BE es de fuerz
ED = np.array([])
AE = np.array([])
Contador5 = 0
while Contador5 < 19:
        Sum_Ex = -FAE*cos(Ang_BCE) + CE[Contador5]*cos(Ang_BCE) - FED
        Ecu_E8 = sp.Eq(Sum_Ex,0)
        Sum_Ey = FAE*sin(Ang_BCE) + CE[Contador5]*sin(Ang_BCE)
```

```
        Ecu_E9 = sp.Eq(Sum_Ey,0)
        Fae = sp.solve(Ecu_E9)
        Fed = sp.solve(Ecu_E8.subs(FAE,Fae[0]))
        AE = np.append(AE,round(Fae[0],3))
        ED = np.append(ED,round(Fed[0],3))
        Contador5 = Contador5 + 1
```

In [11]:
```
AE
```

Out[11]:
```
array([-44.721, -44.592, -43.075, -40.182, -35.971, -30.543, -24.038,
       -16.634, -8.542, 0.0, 8.737, 17.399, 25.715, 33.418, 40.254,
       45.992, 50.429, 53.401, 54.784], dtype=object)
```

In [12]:
```
ED
```

Out[12]:
```
array([79.999, 79.769, 77.055, 71.880, 64.347, 54.637, 43.0000000000000,
       29.756, 15.280, 0, -15.629, -31.124, -46.0000000000000, -59.780,
       -72.009, -82.273, -90.210, -95.527, -98.001], dtype=object)
```

In [13]:
```
'''
Por la configuración de la armadura es claro que el elemento AB tendrá las mismas las fuerzas
es la única manera de mantener el equilibrio
'''
AB = BC
AB
```

Out[13]:
```
array([-40.0000000000000, -36.368, -31.516, -25.565, -18.675, -11.040,
       -2.880, 5.560, 14.025, 22.250, 29.972, 36.940, 42.919, 47.700,
       51.110, 53.011, 53.314, 51.974, 49.0000000000000], dtype=object)
```

In [14]:
```
'''
Si se obseva el nodo D, podemos ver que al tener un apoyo simple, este solo tiene una reacció
elemento DA también sea un elemento de fuerza cero, con esto se conservará el equilibrio del
'''
DA = np.zeros(19)
DA
```

Out[14]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0.])
```

In [15]:
```
'''
Con la función nlargest encontraré la fueza más grande de cada elemento de la armadura y con
dichas fuerzas, con la cual puedo determinar el ángulo y la fuerza P
'''
#Datos fuerza AB
AB_max = he.nlargest(1,abs(AB))[0]
PAng_AB = np.where(AB_max == AB)
Ang_AB = Ang_P[PAng_AB[0]]
P1 = Valor_P[PAng_AB[0]]
AB_maxr = AB[PAng_AB[0]]
AB_maxr = AB_maxr[0]


#Datos fuerza BC
BC_max = he.nlargest(1,abs(AB))[0]
PAng_BC = np.where(BC_max == BC)
Ang_BC = Ang_P[PAng_BC[0]]
P2 = Valor_P[PAng_BC[0]]
BC_maxr = BC[PAng_BC[0]]
BC_maxr = BC_maxr[0]

#Datos fuerza BE
'Este elemento, sin importar el ángulo de la fuerza P, será siempre fuerza cero'

#Datos fuerza CE
```

```
CE_max = he.nlargest(1,abs(CE))[0]
PAng_CE = np.where(CE_max == abs(CE))
Ang_CE = Ang_P[PAng_CE[0]]
P3 = Valor_P[PAng_CE[0]]
CE_maxr = CE[PAng_CE[0]]
CE_maxr = CE_maxr[0]

#Datos fuerza ED
ED_max = he.nlargest(1,abs(ED))[0]
PAng_ED = np.where(ED_max == abs(ED))
Ang_ED = Ang_P[PAng_ED[0]]
P4 = Valor_P[PAng_ED[0]]
ED_maxr = ED[PAng_ED[0]]
ED_maxr = ED_maxr[0]

#Datos fuerza AE
AE_max = he.nlargest(1,abs(AE))[0]
PAng_AE = np.where(AE_max == abs(AE))
Ang_AE = Ang_P[PAng_AE[0]]
P5 = Valor_P[PAng_AE[0]]
AE_maxr = AE[PAng_AE[0]]
AE_maxr = AE_maxr[0]

#Datos fuerza DA
'Este elemento al ser fuerza cero, no se ve afectado por la dirección de la fuerza P'

print("Fuerza máxima por cada elemento")
sp.Matrix([["Elemento","F_max (kN)" , "ÁNGULO (Grados)"   ,"P (kN)"],
           [   "AB"   ,   AB_maxr   ,      Ang_AB[0]      , P1[0]  ],
           [   "BC"   ,   BC_maxr   ,      Ang_BC[0]      , P2[0]  ],
           [   "BE"   ,     0       ,          0          ,   0    ],
           [   "CE"   ,   CE_maxr   ,      Ang_CE[0]      , P3[0]  ],
           [   "ED"   ,   ED_maxr   ,      Ang_ED[0]      , P4[0]  ],
           [   "AE"   ,   AE_maxr   ,      Ang_AE[0]      , P5[0]  ],
           [   "DA"   ,     0       ,          0          ,   0    ]  ])
```

Fuerza máxima por cada elemento

Out[15]:

$$\begin{bmatrix} Elemento & \mathrm{F}_{\max}\,(kN) & \text{ÁNGULO}\,(Grados) & P(kN) \\ AB & 53.314 & 160 & 24.0 \\ BC & 53.314 & 160 & 24.0 \\ BE & 0 & 0 & 0 \\ CE & -54.784 & 180 & 24.5 \\ ED & -98.001 & 180 & 24.5 \\ AE & 54.784 & 180 & 24.5 \\ DA & 0 & 0 & 0 \end{bmatrix}$$

In [16]:

```
print("PUNTO 2")
'Área sección transversal elementos en mm^2'
A_st = 300
'Esfuerzo normal admisible en Mpa'
Esf_Ad = 15

#Lo primero que se debe hacer es determinar el esfuerzo normal para cada elemento y cada dire
'Tener en cuenta que los esfuerzos del elemento AB, serán los mismos de BC'
Esf_AB = np.array([])
Contador6 = 0
while Contador6 < 19:
    EsfAB = AB[Contador6]*10**3/A_st
    Esf_AB = np.append(Esf_AB,EsfAB)
    Contador6 = Contador6 + 1
```

```
'Para los elementos BE y DA no se presenta esfuerzo, ya que son fuerza cero'

'Esfuerzos elemento CE'
Esf_CE = np.array([])
Contador7 = 0
while Contador7 < 19:
    EsfCE = CE[Contador7]*10**3/A_st
    Esf_CE = np.append(Esf_CE,EsfCE)
    Contador7 = Contador7 + 1

'Esfuerzos elemento AE'
Esf_AE = np.array([])
Contador8 = 0
while Contador8 < 19:
    EsfAE = AE[Contador8]*10**3/A_st
    Esf_AE = np.append(Esf_AE,EsfAE)
    Contador8 = Contador8 + 1

'Esfuerzos elemento ED'
Esf_ED = np.array([])
Contador9 = 0
while Contador9 < 19:
    EsfED = ED[Contador9]*10**3/A_st
    Esf_ED = np.append(Esf_ED,EsfED)
    Contador9 = Contador9 + 1
```

PUNTO 2

In [17]:
```
print("Esfuerzos que se generan en el elemento AB")
Esf_AB
```

Out[17]:
```
Esfuerzos que se generan en el elemento AB
array([-133.333333333333, -121.23, -105.05, -85.217, -62.250, -36.800,
       -9.600, 18.53, 46.750, 74.167, 99.907, 123.13, 143.06, 159.00,
       170.37, 176.70, 177.71, 173.25, 163.333333333333], dtype=object)
```

In [18]:
```
print("Esfuerzos que se generan en el elemento BC")
Esf_BC = Esf_AB
Esf_BC
```

Out[18]:
```
Esfuerzos que se generan en el elemento BC
array([-133.333333333333, -121.23, -105.05, -85.217, -62.250, -36.800,
       -9.600, 18.53, 46.750, 74.167, 99.907, 123.13, 143.06, 159.00,
       170.37, 176.70, 177.71, 173.25, 163.333333333333], dtype=object)
```

In [19]:
```
print("Esfuerzos que se generan en el elemento CE")
Esf_CE
```

Out[19]:
```
Esfuerzos que se generan en el elemento CE
array([149.07, 148.64, 143.58, 133.94, 119.90, 101.81, 80.127, 55.447,
       28.47, 0, -29.12, -57.997, -85.717, -111.39, -134.18, -153.31,
       -168.10, -178.00, -182.61], dtype=object)
```

In [20]:
```
print("Esfuerzos que se generan en el elemento AE")
Esf_AE
```

Out[20]:
```
Esfuerzos que se generan en el elemento AE
array([-149.07, -148.64, -143.58, -133.94, -119.90, -101.81, -80.127,
       -55.447, -28.47, 0, 29.12, 57.997, 85.717, 111.39, 134.18, 153.31,
       168.10, 178.00, 182.61], dtype=object)
```

In [21]:
```
print("Esfuerzos que se generan en el elemento ED")
Esf_ED
```

```
Esfuerzos que se generan en el elemento ED
```

Out[21]:
```
array([266.66, 265.90, 256.85, 239.60, 214.49, 182.12, 143.333333333333,
       99.187, 50.933, 0, -52.097, -103.75, -153.333333333333, -199.27,
       -240.03, -274.24, -300.70, -318.42, -326.67], dtype=object)
```

In [28]:
```
print(f"la primera falla se da para la carga P de {Valor_P[0]} kN con una dirección de {Ang_P
print(f'''
Los elementos que fallan son:

  * AB por compresión con una fuerza interna de {round(AB[0],3)} kN y con un esfuerzo de {rou

  * BC por compresión con una fuerza interna de {BC[0]} kN y con un esfuerzo de {round(Esf_BC

  * CE por tensión con una fuerza interna de {CE[0]} kN y con un esfuerzo de {Esf_CE[0]} MPa

  * ED por tensión con una fuerza interna de {ED[0]} kN y con un esfuerzo de {Esf_ED[0]} MPa

  * AE pr compresión con una fuerza interna de {AE[0]} kN y con un esfuerzo de {Esf_AE[0]} MP
print()
print("Para todas las direcciones los elementos fallan ya que sus esfuerzos son mayores al de
```

```
la primera falla se da para la carga P de 20.0 kN con una dirección de 0 grados, es decir tot
almente vertical hacia arriba.

Los elementos que fallan son:

  * AB por compresión con una fuerza interna de -40.0000000000000 kN y con un esfuerzo de -13
3.333 MPa

  * BC por compresión con una fuerza interna de -40.0000000000000 kN y con un esfuerzo de -13
3.333 MPa

  * CE por tensión con una fuerza interna de 44.721 kN y con un esfuerzo de 149.07 MPa

  * ED por tensión con una fuerza interna de 79.999 kN y con un esfuerzo de 266.66 MPa

  * AE pr compresión con una fuerza interna de -44.721 kN y con un esfuerzo de -149.07 MPa

Para todas las direcciones los elementos fallan ya que sus esfuerzos son mayores al de 15 MP
a.
```

In [23]:
```
print("PUNTO 3")
'Valor de la distancia que es movido el nodo B en metros'
print('Variaciones del nodo B')
D_B = np.arange(0.02,0.40,0.02)
D_B
```

```
PUNTO 3
Variaciones del nodo B
```

Out[23]:
```
array([0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 , 0.22,
       0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38])
```

In [24]:
```
print('''
Al variar la posición del nodo B, es claro que el elemento BE se empieza a inclinar cada vez
AB y BC permanecen totalmente horizontales, por esta razón, BE debe seguir comportandose como
que así no afectará el equilibrio del nodo. Lo anterior indica que cambiar la posición del no
demás elementos, es decir, para cualquier iteración, las fuerzas internas siguen siendo las y
presentan todas las fuerzas obtenidas:''')

FN =[["Carga(kN)","Ángulo(Grados)","AB","BC","BE","CE","ED","AE","DA"]]

for i in range(0,19):
    X = [Valor_P[i],Ang_P[i],AB[i],BC[i],BE[i],CE[i],ED[i],AE[i],DA[i]]
    FN.append(X)
sp.Matrix(FN)
```

Al variar la posición del nodo B, es claro que el elemento BE se empieza a inclinar cada vez más, sin embargo, los elementos
AB y BC permanecen totalmente horizontales, por esta razón, BE debe seguir comportandose como un elemento de fuerza cero, ya
que así no afectará el equilibrio del nodo. Lo anterior indica que cambiar la posición del nodo no afecta el comportamiento de los
demás elementos, es decir, para cualquier iteración, las fuerzas internas siguen siendo las y a calculadas. A continuación se
presentan todas las fuerzas obtenidas:

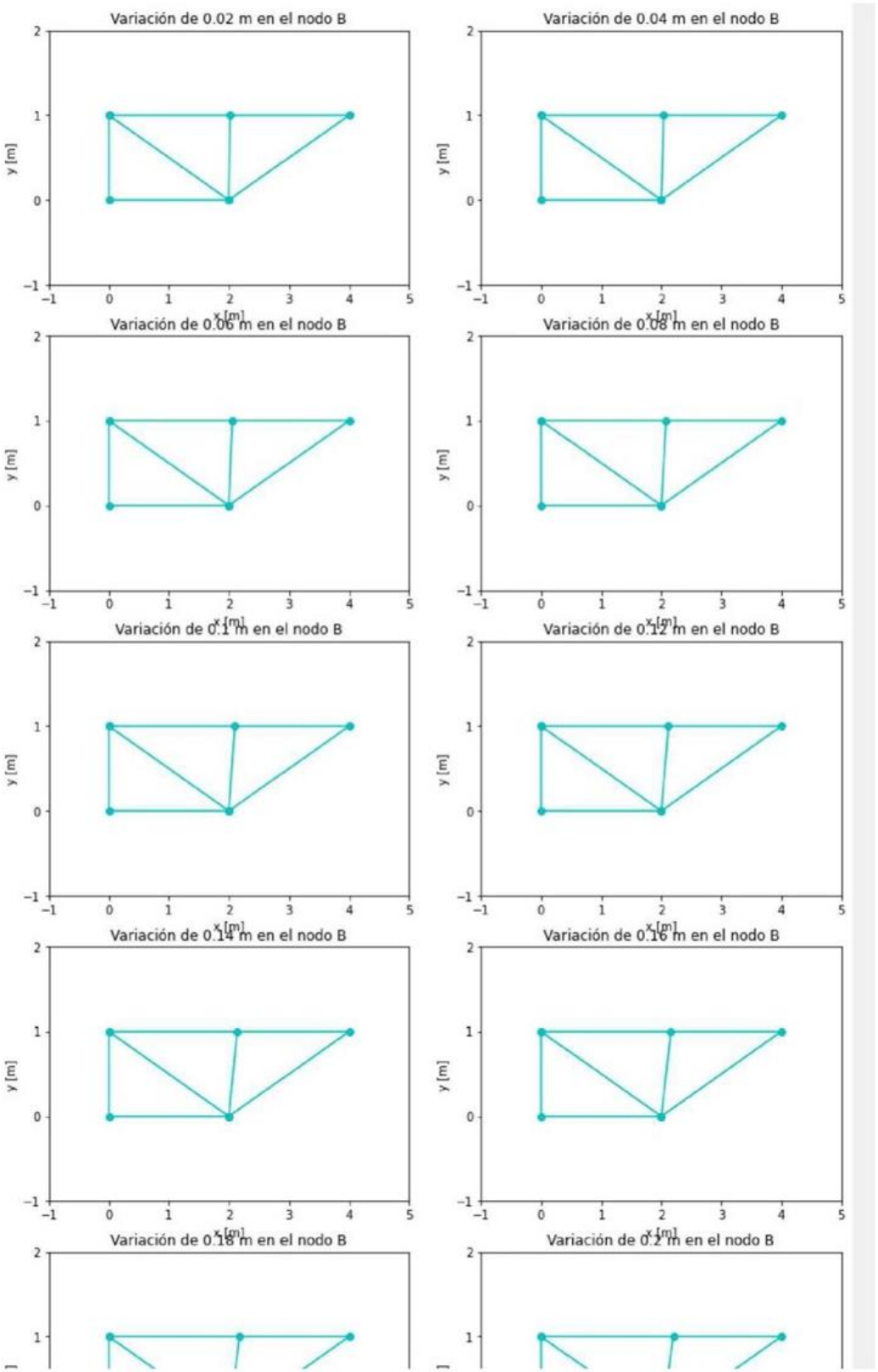
Out[24]:

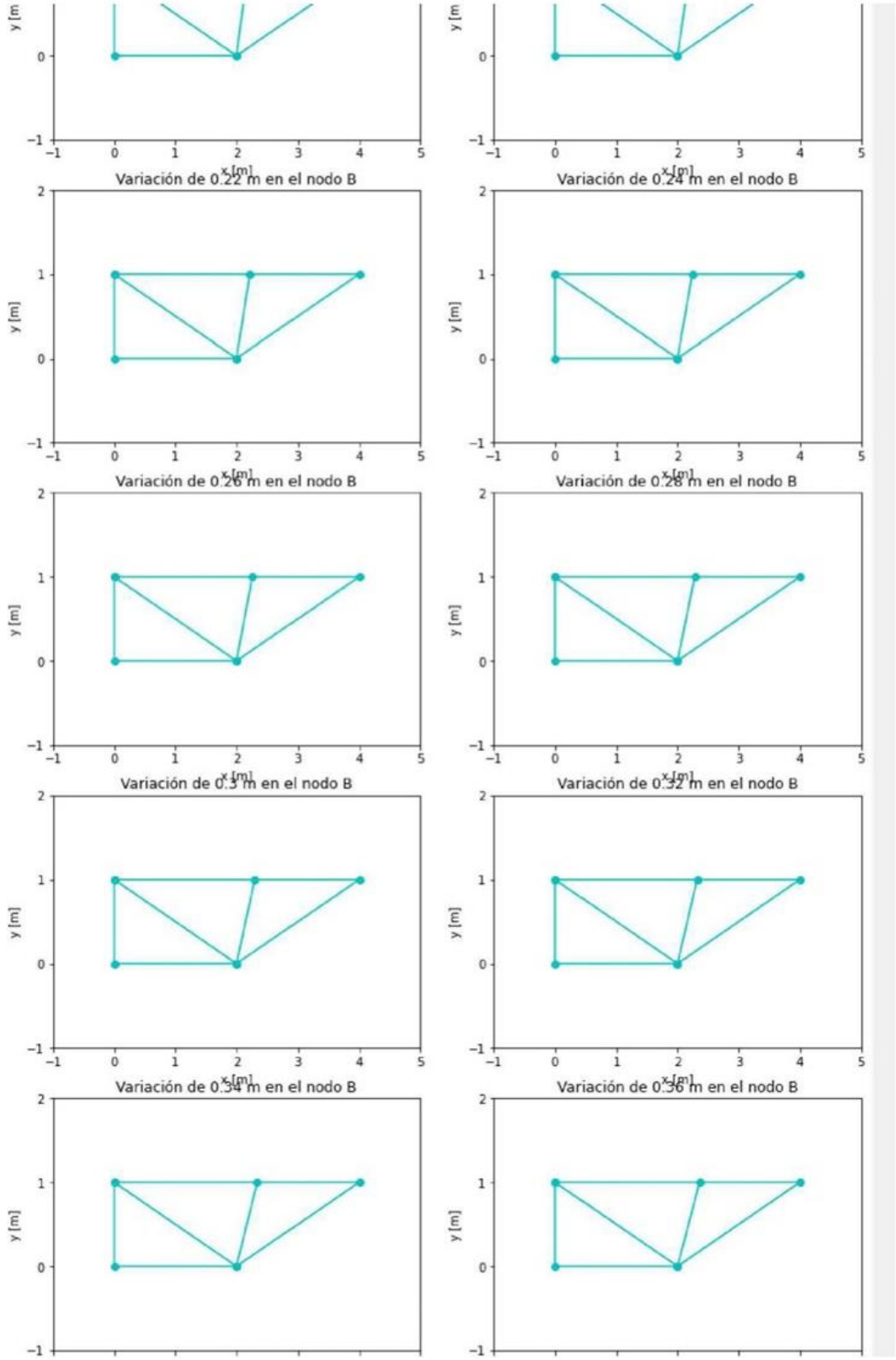
Carga (kN)	Ángulo $(Grados)$	AB	BC	BE	CE	ED	AE	DA
20.0	0	−40.0	−40.0	0	44.721	79.999	−44.721	0
20.25	10	−36.368	−36.368	0	44.592	79.769	−44.592	0
20.5	20	−31.516	−31.516	0	43.075	77.055	−43.075	0
20.75	30	−25.565	−25.565	0	40.182	71.88	−40.182	0
21.0	40	−18.675	−18.675	0	35.971	64.347	−35.971	0
21.25	50	−11.04	−11.04	0	30.543	54.637	−30.543	0
21.5	60	−2.88	−2.88	0	24.038	43.0	−24.038	0
21.75	70	5.56	5.56	0	16.634	29.756	−16.634	0
22.0	80	14.025	14.025	0	8.542	15.28	−8.542	0
22.25	90	22.25	22.25	0	0	0	0	0
22.5	100	29.972	29.972	0	−8.737	−15.629	8.737	0
22.75	110	36.94	36.94	0	−17.399	−31.124	17.399	0
23.0	120	42.919	42.919	0	−25.715	−46.0	25.715	0
23.25	130	47.7	47.7	0	−33.418	−59.78	33.418	0
23.5	140	51.11	51.11	0	−40.254	−72.009	40.254	0
23.75	150	53.011	53.011	0	−45.992	−82.273	45.992	0
24.0	160	53.314	53.314	0	−50.429	−90.21	50.429	0
24.25	170	51.974	51.974	0	−53.401	−95.527	53.401	0
24.5	180	49.0	49.0	0	−54.784	−98.001	54.784	0

In [25]:

```
print("PUNTO 4")
print("Las siguientes gráficas muestran los cambios de posición que puede presentar el nodo B
fig =plt.figure(figsize=(12,42))
fig.tight_layout()
for u in range(1,19):
    Varia_x = [0,4,2,0,0,2,2 + D_B[u-1]]
    Varia_y = [1,1,0,0,1,0,1]
    ax = plt.subplot(9,2,u)
    ax.plot(Varia_x,Varia_y,color = "c",marker = "o")
    plt.xticks(np.arange(-1,6,1))
    plt.yticks(np.arange(-1,3))
    ax.set_title(f"Variación de {round(D_B[u-1],3)} m en el nodo B")
    ax.set_xlabel("x [m]")
    ax.set_ylabel("y [m]")
```

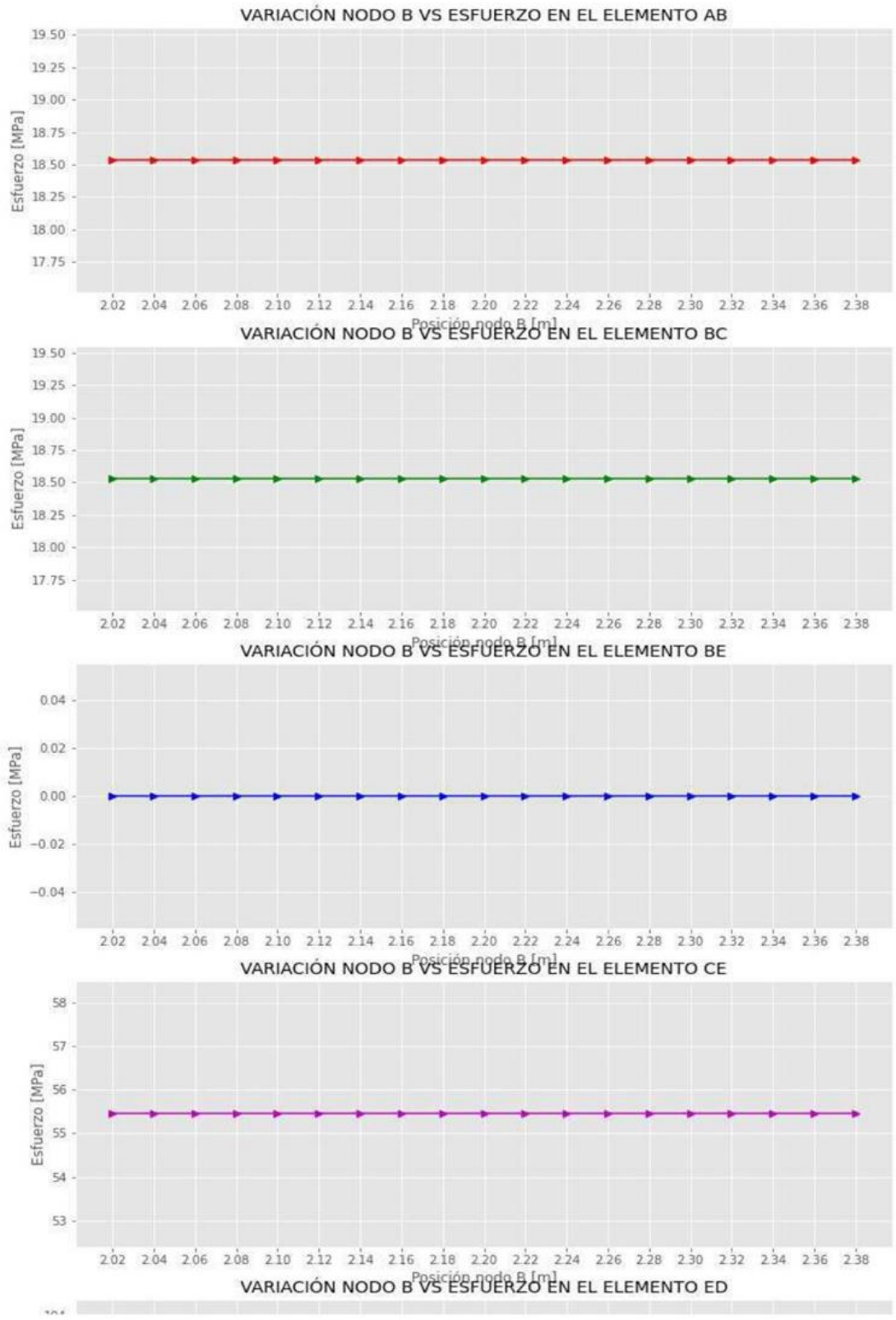
```
PUNTO 4
Las siguientes gráficas muestran los cambios de posición que puede presentar el nodo B.
```

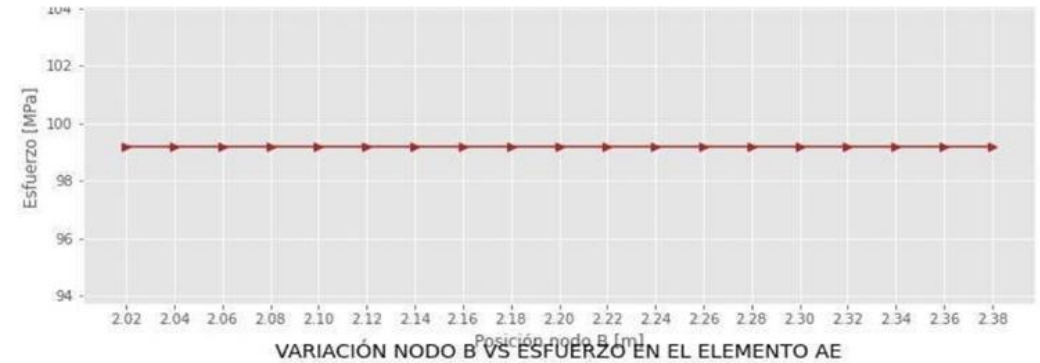

Variación de 0.02 m en el nodo B
Variación de 0.04 m en el nodo B
Variación de 0.06 m en el nodo B
Variación de 0.08 m en el nodo B
Variación de 0.1 m en el nodo B
Variación de 0.12 m en el nodo B
Variación de 0.14 m en el nodo B
Variación de 0.16 m en el nodo B
Variación de 0.18 m en el nodo B
Variación de 0.2 m en el nodo B
y [m]
x [m]


Variación de 0.22 m en el nodo B
Variación de 0.24 m en el nodo B
Variación de 0.26 m en el nodo B
Variación de 0.28 m en el nodo B
Variación de 0.3 m en el nodo B
Variación de 0.32 m en el nodo B
Variación de 0.34 m en el nodo B
Variación de 0.36 m en el nodo B
x [m]
y [m]

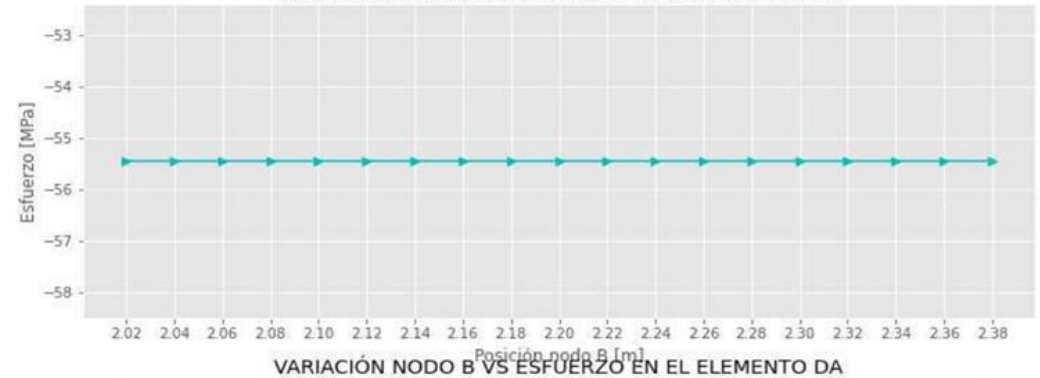
In [26]:
```
print('PUNTO 5')
plt.style.use('ggplot')
'Ángulo AB en grados'
Ang_AB = 69
Ang_AB = 70

'Valor carga P para el ángulo de 70 grados en kN'
Carga = 21.75

print('''Como ya se dijo, mover el nodo B no afecta las fuerzas internas de los elementos, po
afectados los esfuerzos, esto indica que todas las gáficas evidenciarán un comportamiento con
print()

Elementos = np.array ([Esf_AB[7],Esf_BC[7],0,Esf_CE[7],Esf_ED[7],Esf_AE[7],0])
N_Elementos = ["AB","BC","BE","CE","ED","AE","DA"]
Color = ['r','g','b','m','brown','c','grey']
fig =plt.figure(figsize=(12,35))

for i in range(1,8):
    x = np.arange(2.02,2.40,0.02)
    y = np.tile(Elementos[i-1],(19))
    bx = plt.subplot(7,1,i)
    bx.plot(x,y,color = Color[i-1],marker = ">")
    bx.set_xlabel("Posición nodo B [m]")
    bx.set_ylabel("Esfuerzo [MPa]")
    bx.set_title(f"VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO {N_Elementos[i-1]}")
    plt.xticks(np.arange(2.02,2.40,0.02))
```

```
PUNTO 5
Como ya se dijo, mover el nodo B no afecta las fuerzas internas de los elementos, por dicha r
azón, tampoco se ven
afectados los esfuerzos, esto indica que todas las gáficas evidenciarán un comportamiento con
stante.
```

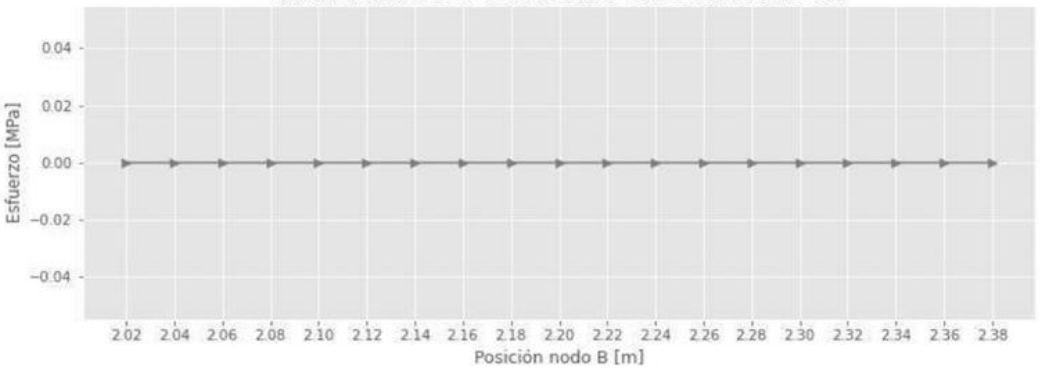

VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO AB
Esfuerzo [MPa]
Posición nodo B [m]
VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO BC
Esfuerzo [MPa]
Posición nodo B [m]
VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO BE
Esfuerzo [MPa]
Posición nodo B [m]
VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO CE
Esfuerzo [MPa]
Posición nodo B [m]
VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO ED



VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO AE



VARIACIÓN NODO B VS ESFUERZO EN EL ELEMENTO DA



Apéndice I. *Segundo Taller*

ESCUELA DE INGENIERIA Civil

Universidad Industrial de Santander
Escuela de Ingeniería Civil
Segundo periodo académico 2020
Laboratorio computacional MDS.

Universidad Industrial de Santander

INSTRUCCIONES:

1. Guardar en una carpeta "**T2_Codigo_Apellido_Nombre**" las soluciones en un archivo de Jupyter Notebook y con el nombre "**T2_solucion**", de manera que los resultados se vean de forma explícita en el navegador o al descargar y ejecutar cada celda del archivo. En la misma carpeta se debe subir el archivo Excel de la segunda parte.

2. Hacer un ***Pull Request*** en la carpeta ***/Semana9/Taller/Primera_Entrega*** en GitHub con la carpeta mencionada anteriormente. Fecha máxima de entrega: **26/02/2021**, hora límite **11:59 PM**.

3. El día **28/02/2022** se entregará la retroalimentación personal al estudiante por medio de GitHub. Tras esto, tendrán como fecha límite **05/03/2022**, hora límite **11:59 PM** para hacer nuevamente un *Pull Request* con el mismo nombre de carpeta y archivos en la carpeta ***/Semana9/Taller/Segunda_Entrega*** en GitHub. La nota definitiva del taller se enviará a cada estudiante por medio de GitHub el **09/03/2022**.

EJERCICIO 1 - (VALOR: 2.5 UNIDADES)

Se dispone de una cubierta hecha en madera laminada apoyada en correas de madera maciza, que a su vez, transmiten sus cargas a dos vigas metálicas en voladizo, según lo muestra la *Figura 1*. La empresa fabricante de estas cubiertas le contrata como diseñador estructural para que desarrolle una automatización usando Python, y de esa forma se pueda garantizar la estabilidad, el equilibrio y la seguridad de una estructura solicitada por sus clientes de forma recurrente. El objetivo es proveer a la empresa con un software que les permita determinar el perfil de viga más económico, sin importar la distribución de las correas y el peso de los materiales de los elementos que forman la cubierta.

Figura 1 – Cubierta de análisis. Fuente: https://twitter.com/pergolart/status/1201539729189228546

Como diseñador, usted determina el esquema del sistema estructural de la viga en voladizo AB, tal como se muestra en la *Figura 2*. Además, usted determina que la distribución de cargas es trapezoidal para cualquier tipo de cubierta, y la magnitud de su carga máxima ***W*** es de ***0.005 * X + 0.01 * Y***, valor que depende de los materiales, la distribución de las correas de la cubierta, cargas vivas y de cargas de viento. Finalmente, la empresa le provee la tensión a la que se someterá el cable que pende del extremo B. Dicho valor proviene de un factor debido al tipo de los materiales de ***P = 0.05 * X*** y el ángulo de aplicación de esta carga es de 45° con respecto a la horizontal.

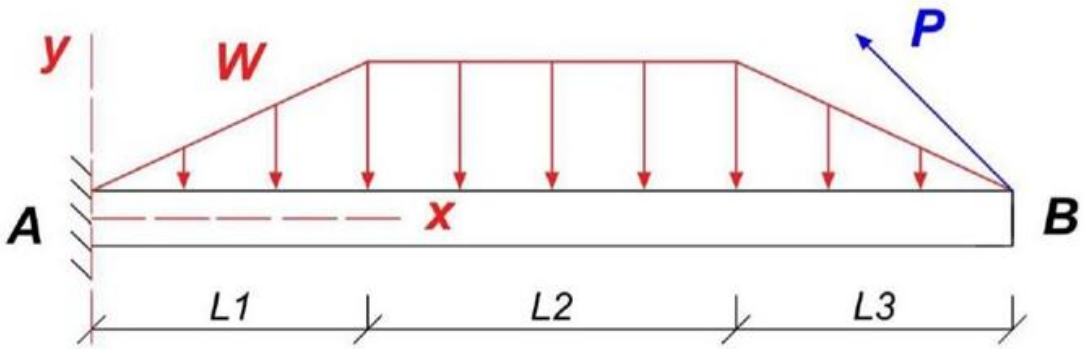


Figura 2 – Viga AB empotrada. Fuente: Autor

Se sabe que las longitudes de aplicación de cada carga distribuida están en metros, siendo **L1 = 0.05 * X**, **L2 = L1 * 1.5 y L3 = 0.5 * L1.** Con base en la información presentada, determine mediante un algoritmo en Python:

1. El diagrama de fuerza cortante [kN] y de momento flector [kN – m] de la viga. ***(VALOR: 1.25)***
2. La magnitud y posición donde se genera la máxima fuerza cortante y el momento interno máximo. ***(VALOR: 1.25)***

Nota 1: El factor de carga de la distribución de correas y los materiales **X,** es igual a la sumatoria de los dígitos de su código estudiantil. Ejemplo: Para un código 2180140, **X** = 2+1+8+0+1+4+0 = 16.

Nota 2: El factor de amplitud debido a carga viva y de viento **Y,** es igual **X / 3**.

EJERCICIO 2 - (VALOR: 2.5 UNIDADES)

El perfil tipo T mostrado en la *Figura 3* corresponde a la sección transversal de la viga de la *Figura 2*, la cual se proyecta en acero A-36 con un esfuerzo admisible de 250 MPa.

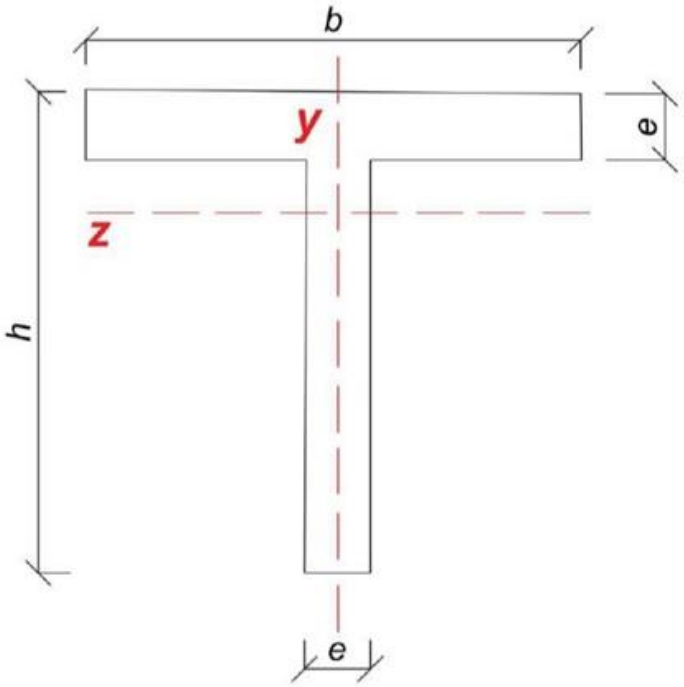


Figura 3. Perfil metálico sección T. Fuente: Autor

Con base en la información presentada y mediante un algoritmo en Python, exporte la siguiente información a un archivo Excel:

1. Una tabla con el área, inercia centroidal alrededor del eje *z*, posición del eje neutro y esfuerzos en la fibra superior e inferior para cada uno de los perfiles que usa la empresa en sus cubiertas, según la *Tabla 1*. Considere la sección de la viga con mayor demanda de fuerzas internas. ***(VALOR: 1.25)***

2. Seleccionar el perfil más económico o de menor peso que resista las solicitaciones y exportar una tabla con la información de este, con sus dimensiones, área y esfuerzo generado. ***(VALOR: 1.25)***

Tabla 1. Perfiles metálicos sección tipo T.

perfil	h [mm]	b [mm]	t [mm]
1	25	50	5
2	30	60	5.5
3	35	70	6
4	40	80	7
5	50	100	8.5
6	55	115	9
7	60	120	10
8	70	140	11.5
9	80	160	13
10	90	180	15

Apéndice J. *Solución Segundo Taller*

Solución Taller 2

In [ ]:
```
import sympy as sp
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from openpyxl import load_workbook
plt.style.use("seaborn-notebook")
```

In [2]:
```
codigo = "2172116"
X = sum([int(i) for i in codigo])
Y = X / 3
L1 = 0.05 * X
L2 = L1 * 1.5
L3 = 0.5 * L1
L_total = L1 + L2 + L3
W = 0.005 * X + 0.01 * Y
P = 0.05 * X

Ax, Ay, Ma, x = sp.symbols("Ax Ay Ma x")
angulo = 45
angulo_radianes = angulo * np.pi / 180
P_x = np.sin(angulo_radianes) * P
P_y = P_x
L_total
```

Out[2]:
```
3.0
```

EQUILIBRIO

In [3]:
```
equilibrio_y = Ay + P_y - W * (L1 + L3) / 2 - W * L2
ecuacion_y = sp.Eq(equilibrio_y, 0)
fuerza_Ay = sp.solve(ecuacion_y)[0]
# en kN
fuerza_Ay
```

Out[3]:
$$-0.332106781186548$$

In [4]:
```
equilibrio_x = Ax - P_x
ecuacion_x = sp.Eq(equilibrio_x, 0)
fuerza_Ax = sp.solve(ecuacion_x)[0]
# en kN
fuerza_Ax
```

Out[4]:
$$0.707106781186548$$

In [5]:
```
equilibrio_momentos = Ma + P_y * L_total - W * L1 / 2 * (2 * L1 / 3) - W * L2 * (L1 + L2 / 2)
                      - W * L3 / 2 * (L_total - 2 * L3 / 3)
ecuacion_momentos = sp.Eq(equilibrio_momentos, 0)
momento_A = sp.solve(ecuacion_momentos)[0]
# kN - m
momento_A
```

Out[5]:
$$-1.51715367689298$$

DIAGRAMAS DE CORTANTE Y MOMENTO

Tramo 1

In [6]:
```
# Tramo1 0 < x < L1
funcion_w = W * x / L1
cortante_1 = fuerza_Ay - sp.integrate(funcion_w, x)
momento_1 = - momento_A + sp.integrate(cortante_1, x)
```

In [7]:
```
cortante_1
```

Out[7]: $-0.0833333333333333x^2 - 0.332106781186548$

In [8]:
```
momento_1
```

Out[8]: $-0.0277777777777778x^3 - 0.332106781186548x + 1.51715367689298$

Tramo 2

In [9]:
```
# Tramo2 0 < x < L2
funcion_w = W
cortante_2 = cortante_1.subs(x, L1) - sp.integrate(funcion_w, x)
momento_2 = momento_1.subs(x, L1) + sp.integrate(cortante_2, x)
```

In [10]:
```
cortante_2
```

Out[10]: $-0.166666666666667x - 0.415440114519881$

In [11]:
```
momento_2
```

Out[11]: $-0.0833333333333333x^2 - 0.415440114519881x + 1.15726911792865$

Tramo 3

In [12]:
```
# Tramo2 0 < x < L3
funcion_w = - W * x / L3 + W
cortante_3 = cortante_2.subs(x, L2) - sp.integrate(funcion_w, x)
momento_3 = momento_2.subs(x, L2) + sp.integrate(cortante_3, x)
```

In [13]:
```
cortante_3
```

Out[13]: $0.166666666666667x^2 - 0.166666666666667x - 0.665440114519881$

In [14]:
```
momento_3
```

Out[14]: $0.0555555555555556x^3 - 0.0833333333333333x^2 - 0.665440114519881x + 0.346608946148832$

Gráfica de cortante

In [16]:
```
x_array = np.arange(0, L_total + 0.005, 0.005)
```

In [17]:
```
# ejecutar la funcion de cortante para cada elemento del array
tramo1 = sp.lambdify(x, cortante_1)(x_array[x_array < L1])
tramo2 = sp.lambdify(x, cortante_2)(x_array[(x_array >= L1) & (x_array < (L1 + L2) )] - L1)
tramo3 = sp.lambdify(x, cortante_3)(x_array[x_array >= (L1 + L2)] - L1 - L2)
# graficar diagrama de fuerza cortante
dfc = np.concatenate((tramo1, tramo2, tramo3))
plt.plot(x_array, dfc)
plt.fill_between(x_array, dfc, alpha=0.25)
# comprobar que el diagrama cierra colocando dos plots con la fuerza en cada extremo
plt.plot([0, 0], [0, float(fuerza_Ay)], marker="v", color="red")
plt.plot([L_total, L_total], [0, -P_y], marker="v", color="red")
# graficar ejes
plt.axvline(0, color="gray", linestyle='--')
plt.axhline(0, color="gray", linestyle='--')
```

Out[17]:
```
<matplotlib.lines.Line2D at 0x2571e057c10>
```

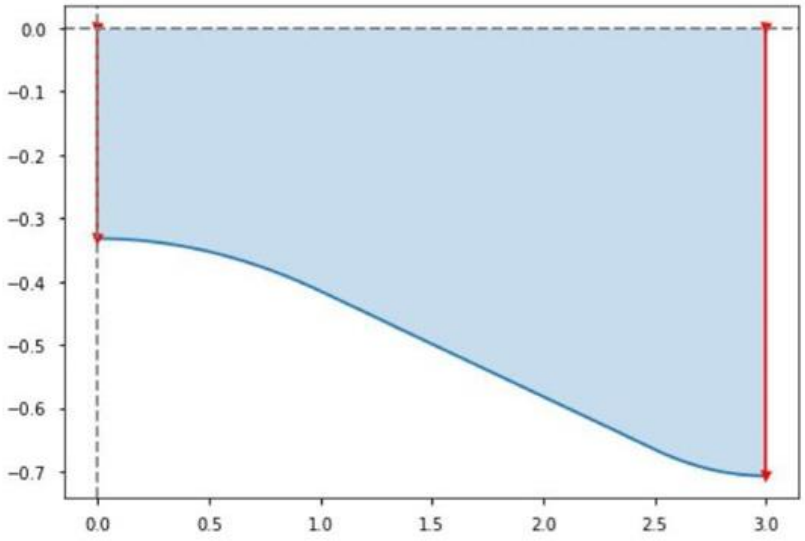
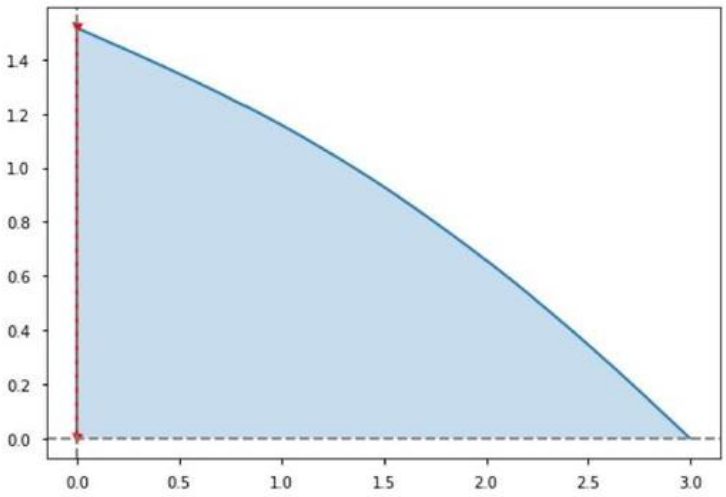


Gráfico de momento

In [18]:
```
tramo1_momento = sp.lambdify(x, momento_1)(x_array[x_array < L1])
tramo2_momento = sp.lambdify(x, momento_2)(x_array[(x_array >= L1) & (x_array < (L1 + L2) )]
tramo3_momento = sp.lambdify(x, momento_3)(x_array[x_array >= (L1 + L2)] - L1 - L2)
dmf = np.concatenate((tramo1_momento, tramo2_momento, tramo3_momento))
plt.plot(x_array, dmf)
plt.fill_between(x_array, dmf, alpha=0.25)

plt.plot([0, 0], [0, -float(momento_A)], marker="v", color="red")
# graficar ejes
plt.axvline(0, color="gray", linestyle='--')
plt.axhline(0, color="gray", linestyle='--')
```

Out[18]:
```
<matplotlib.lines.Line2D at 0x2571e145460>
```



Magnitud y posición del cortante máximo

In [18]:
```
def obtener_maximo(valores_diagrama):
    valor_max_positivo = np.amax(valores_diagrama)
    valor_max_negativo = np.amin(valores_diagrama)

    valor_maximo = valor_max_positivo
    if np.absolute(valor_max_negativo) > valor_max_positivo:
        valor_maximo = valor_max_negativo

    posicion = x_array[np.where(valores_diagrama == valor_maximo)][0]
    return valor_maximo, posicion

cortante_maximo, posicion_cortante_maximo = obtener_maximo(dfc)
print(f"El valor del cortante máximo es de {cortante_maximo} kN")
print(f"Se encuentra localizado a {posicion_cortante_maximo} metros desde el nodo A")
```

```
El valor del cortante máximo es de -0.8838834764831833 kN
Se encuentra localizado a 3.75 metros desde el nodo A
```

Magnitud y posición del momento máximo

In [19]:
```
momento_maximo, posicion_momento_maximo = obtener_maximo(dmf)
print(f"El valor del momento máximo es de {momento_maximo} kN - m")
print(f"Se encuentra localizado a {posicion_momento_maximo} metros desde el nodo A")
```

```
El valor del momento máximo es de 2.13455001597861 kN - m
Se encuentra localizado a 0.0 metros desde el nodo A
```

SEGUNDA PARTE

Obtención del perfil más económico

In [20]:
```
perfiles = pd.read_excel('T2.xlsx', usecols="B:E", nrows=11, skiprows=3, index_col='perfil')
perfiles
```

Out[20]:

perfil	h [mm]	b [mm]	t [mm]
1.0	25.0	50.0	5.0
2.0	30.0	60.0	5.5
3.0	35.0	70.0	6.0
4.0	40.0	80.0	7.0
5.0	50.0	100.0	8.5
6.0	55.0	115.0	9.0
7.0	60.0	120.0	10.0
8.0	70.0	140.0	11.5
9.0	80.0	160.0	13.0
10.0	90.0	180.0	15.0

In [21]:
```
# Agregar area
perfiles['A [mm ^ 2]'] = perfiles['b [mm]']  * perfiles['t [mm]'] + (perfiles['h [mm]'] \
                                                                    - perfiles['t [mm]']) *
perfiles
```

Out[21]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]
1.0	25.0	50.0	5.0	350.00
2.0	30.0	60.0	5.5	464.75
3.0	35.0	70.0	6.0	594.00
4.0	40.0	80.0	7.0	791.00
5.0	50.0	100.0	8.5	1202.75
6.0	55.0	115.0	9.0	1449.00
7.0	60.0	120.0	10.0	1700.00
8.0	70.0	140.0	11.5	2282.75
9.0	80.0	160.0	13.0	2951.00
10.0	90.0	180.0	15.0	3825.00

In [22]:
```
# Agregar Inercias
perfiles['centroide [mm]'] = (perfiles['b [mm]']  * perfiles['t [mm]'] * (perfiles['h [mm]']
    - perfiles['t [mm]'] / 2) + (perfiles['h [mm]'] - perfiles['t [mm]']) * perfiles['t [mm]'
    - perfiles['t [mm]']) / 2) / perfiles['A [mm ^ 2]']

perfiles['Iz [mm ^ 4]'] = perfiles['b [mm]'] * perfiles['t [mm]'] ** 3 / 12  + (perfiles['h [
    perfiles['t [mm]']) ** 3 * perfiles['t [mm]'] / 12 +  perfiles['b [mm]'] * perfiles['t [m
    perfiles['centroide [mm]'] - (perfiles['h [mm]'] - perfiles['t [mm]'] / 2)) ** 2 + (perfi
    - perfiles['t [mm]']) * perfiles['t [mm]'] * (perfiles['centroide [mm]'] - (perfiles['h [
    - perfiles['t [mm]']) / 2) ** 2

perfiles['Eje neutro [mm]'] = (P_x * 10 ** 3 / perfiles['A [mm ^ 2]']) * (perfiles['Iz [mm ^
                                                                          (-momento_maximo *
perfiles
```

Out[22]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]	centroide [mm]	Iz [mm ^ 4]	Eje neutro [mm]
1.0	25.0	50.0	5.0	350.00	18.928571	1.501488e+04	-0.017764
2.0	30.0	60.0	5.5	464.75	22.900888	2.910029e+04	-0.025928
3.0	35.0	70.0	6.0	594.00	26.873737	5.113253e+04	-0.035645
4.0	40.0	80.0	7.0	791.00	30.659292	8.866585e+04	-0.046416
5.0	50.0	100.0	8.5	1202.75	38.417845	2.115530e+05	-0.072834
6.0	55.0	115.0	9.0	1449.00	42.642857	3.036222e+05	-0.086767
7.0	60.0	120.0	10.0	1700.00	46.176471	4.318137e+05	-0.105181
8.0	70.0	140.0	11.5	2282.75	53.935139	7.908459e+05	-0.143457
9.0	80.0	160.0	13.0	2951.00	61.693833	1.337393e+06	-0.187663
10.0	90.0	180.0	15.0	3825.00	69.264706	2.186057e+06	-0.236657

In [23]:
```
perfiles['Esfuerzo Inf [MPa]'] = P_x * 10 ** 3 / perfiles['A [mm ^ 2]'] + (momento_maximo * 1
    (perfiles['centroide [mm]']) / perfiles['Iz [mm ^ 4]']

perfiles['Esfuerzo Sup [MPa]'] = P_x * 10 ** 3 / perfiles['A [mm ^ 2]'] - (momento_maximo * 1
    (perfiles['h [mm]'] - perfiles['centroide [mm]']) / perfiles['Iz [mm ^ 4]']
```

In [24]:
```
perfiles
```

Out[24]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]	centroide [mm]	Iz [mm ^ 4]	Eje neutro [mm]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
1.0	25.0	50.0	5.0	350.00	18.928571	1.501488e+04	-0.017764	2693.454638	-860.602871
2.0	30.0	60.0	5.5	464.75	22.900888	2.910029e+04	-0.025928	1681.716445	-518.828742
3.0	35.0	70.0	6.0	594.00	26.873737	5.113253e+04	-0.035645	1123.344032	-337.746396
4.0	40.0	80.0	7.0	791.00	30.659292	8.866585e+04	-0.046416	739.212141	-223.751668
5.0	50.0	100.0	8.5	1202.75	38.417845	2.115530e+05	-0.072834	388.367375	-116.127989
6.0	55.0	115.0	9.0	1449.00	42.642857	3.036222e+05	-0.086767	300.401374	-86.264223
7.0	60.0	120.0	10.0	1700.00	46.176471	4.318137e+05	-0.105181	228.780360	-67.812608
8.0	70.0	140.0	11.5	2282.75	53.935139	7.908459e+05	-0.143457	145.962027	-42.973018
9.0	80.0	160.0	13.0	2951.00	61.693833	1.337393e+06	-0.187663	98.766139	-28.918091
10.0	90.0	180.0	15.0	3825.00	69.264706	2.186057e+06	-0.236657	67.863800	-20.015657

In [25]:
```
filtro = (perfiles['Esfuerzo Inf [MPa]'] <= 250) & (perfiles['Esfuerzo Sup [MPa]'] <= 250)
perfiles[filtro]
```

Out[25]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]	centroide [mm]	Iz [mm ^ 4]	Eje neutro [mm]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
7.0	60.0	120.0	10.0	1700.00	46.176471	4.318137e+05	-0.105181	228.780360	-67.812808
8.0	70.0	140.0	11.5	2282.75	53.935139	7.908459e+05	-0.143457	145.962027	-42.973018
9.0	80.0	160.0	13.0	2951.00	61.693833	1.337393e+06	-0.187663	98.766139	-28.918091
10.0	90.0	180.0	15.0	3825.00	69.264706	2.186057e+06	-0.236657	67.863800	-20.015657

In [26]:
```
indice = perfiles.loc[filtro, 'A [mm ^ 2]'].idxmin()
perfil_optimo = perfiles.loc[[indice], ['b [mm]', 'h [mm]', 't [mm]', 'A [mm ^ 2]', 'Esfuerzo
                                        'Esfuerzo Sup [MPa]']]
perfil_optimo
```

Out[26]:

perfil	b [mm]	h [mm]	t [mm]	A [mm ^ 2]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
7.0	120.0	60.0	10.0	1700.0	228.78036	-67.812808

In [27]:
```
respuesta = perfiles.loc[:, ['b [mm]', 'h [mm]', 't [mm]', 'A [mm ^ 2]', 'Iz [mm ^ 4]', 'Eje
                             'Esfuerzo Inf [MPa]', 'Esfuerzo Sup [MPa]']]
```

In [28]:
```
respuesta
```

Out[28]:

perfil	b [mm]	h [mm]	t [mm]	A [mm ^ 2]	Iz [mm ^ 4]	Eje neutro [mm]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
1.0	50.0	25.0	5.0	350.00	1.501488e+04	-0.017764	2693.454638	-860.602871
2.0	60.0	30.0	5.5	464.75	2.910029e+04	-0.025928	1681.716445	-518.828742
3.0	70.0	35.0	6.0	594.00	5.113253e+04	-0.035645	1123.344032	-337.746396
4.0	80.0	40.0	7.0	791.00	8.866585e+04	-0.046416	739.212141	-223.751668
5.0	100.0	50.0	8.5	1202.75	2.115530e+05	-0.072834	388.367375	-116.127989
6.0	115.0	55.0	9.0	1449.00	3.036222e+05	-0.086767	300.401374	-86.264223
7.0	120.0	60.0	10.0	1700.00	4.318137e+05	-0.105181	228.780360	-67.812808
8.0	140.0	70.0	11.5	2282.75	7.908459e+05	-0.143457	145.962027	-42.973018
9.0	160.0	80.0	13.0	2951.00	1.337393e+06	-0.187663	98.766139	-28.918091
10.0	180.0	90.0	15.0	3825.00	2.186057e+06	-0.236657	67.863800	-20.015657

In [29]:
```
with pd.ExcelWriter('T2.xlsx', mode='a', engine='openpyxl') as writer:
    # configurando sheets
    book = load_workbook('T2.xlsx')
    writer.book = book
    writer.sheets = dict((ws.title, ws) for ws in book.worksheets)

    # colcando los dataframes en sus respectivas posiciones
    respuesta.to_excel(writer, startrow=3, startcol=6, sheet_name='Perfiles')
    # colocando perfiles adecuados
    perfil_optimo.to_excel(writer, startrow=18, startcol=6, sheet_name='Perfiles', index=None
```

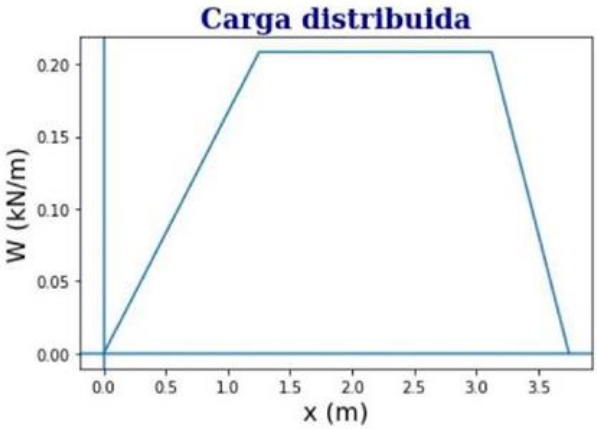
Apéndice K. *Solución Segundo Taller, Propuesta por Estudiante*

Solución Taller 2

Propuesta por estudiante

In [75]:
```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
import pandas as pd
import math

def sen(grads):
    return math.sin(math.radians(grads))
def cos(grads):
    return math.cos(math.radians(grads))
def tan(grads):
    return math.tan(math.radians(grads))
```

In [76]:
```
#GEOMETRÍA DE LA VÍGA
X=2+1+9+0+1+8+4
Y=X/3
w=0.005*X+0.01*Y #en kN/m
p=0.05*X #en kN
L1=0.05*X #en m
L2=1.5*L1 #en m
L3=0.5*L1 #en m
L=L1+L2+L3 #en m
viga={
    "L1":L1,
    "L2":L2,
    "L3":L3,
    "L":L,
    "p":p,
    "w":w,
}
#GRAFICAR CARGA W
variablex=np.linspace(0,L,999)
w1= lambda variablex: (w/L1)*variablex
w2= lambda variablex: w
w3= lambda variablex: w*(1-(variablex-(L1+L2))/L3)

carga=np.piecewise(variablex,[variablex<L1, (variablex>L1) & (variablex< L1+L2), (variablex>L
plt.plot(variablex,carga)
plt.axhline(0)
plt.axvline(0)
plt.ylabel("W (kN/m)", size = 16)
plt.xlabel("x (m)", size = 16,)
plt.title("Carga distribuida",
          fontdict={'family': 'serif',
                    'color' : 'darkblue',
                    'weight': 'bold',
                    'size': 18})
plt.show()
```



Equilibrio

In [77]:
```
# VARIABLES
Ax,Ay,Ma,x=sp.symbols("Ax Ay Ma x")
# Equilibrio Y
equilibrio_y=Ay+p*sen(45)-(w/2)*(L1+L3)-w*L2
equilibrio_x=Ax-p*cos(45)
equilibrio_m=Ma-(w*L1/2)*(L1*(2/3))-(w*L2)*(L1+(L2/2))-(w*L3/2)*(L1+L2+(L3*(1/3)))+L*p*sen(45

ecuacion_y=sp.Eq(equilibrio_y,0)
RAy=sp.solve(ecuacion_y)[0]
#En kN

ecuacion_x=sp.Eq(equilibrio_x,0)
RAx=sp.solve(ecuacion_x)[0]
#En kN

ecuacion_m=sp.Eq(equilibrio_m,0)
RMa=sp.solve(ecuacion_m)[0]
#En kN*m
```

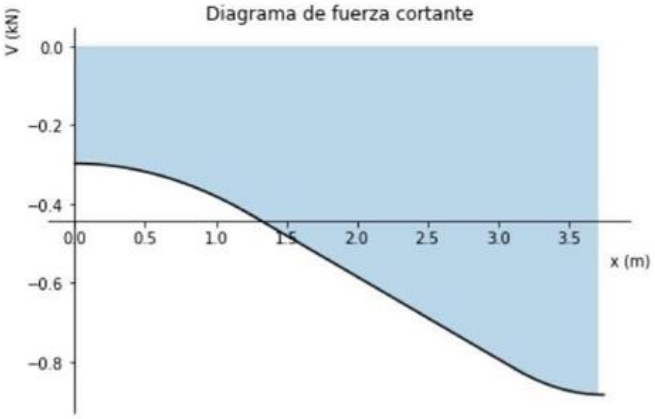
Cortante y Momento

In [78]:
```
#tramo I con 0<=X<=L1
w1=viga["w"]*x/viga["L1"]
Vx1=-sp.integrate(w1,x)+RAy
Mx1=sp.integrate(Vx1,x)-RMa
```

In [79]:
```
#tramo II con 0<=X<=L2
w2=viga["w"]
Vx2=-sp.integrate(w2,x)+Vx1.subs(x,L1)
Mx2=sp.integrate(Vx2,x)+Mx1.subs(x,L1)
```

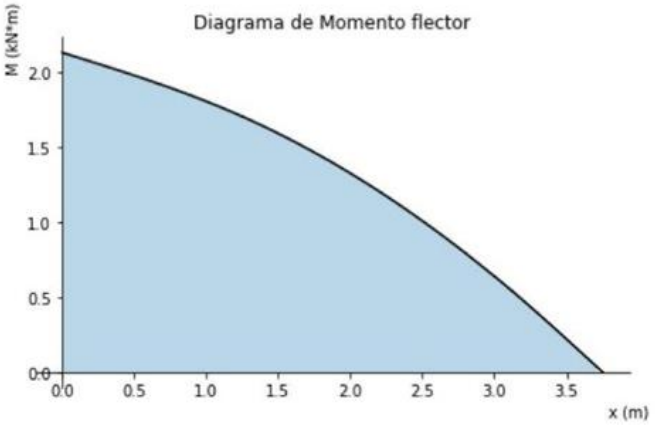
In [80]:
```
#tramo III con 0<=X<=L3
w3=viga["w"]*(1-x/viga["L3"])
Vx3=-sp.integrate(w3,x)+Vx2.subs(x,L2)
#viga["p"]*sen(45)
Mx3=sp.integrate(Vx3,x)+Mx2.subs(x,L2)
```

In [81]:
```
#gráfico de diagrama de fuerza cortante

x_array=np.arange(0,viga["L"],viga["L"]/100)
areaV1=sp.lambdify(x,Vx1)(x_array[x_array<=viga["L1"]])
areaV2=sp.lambdify(x,Vx2)(x_array[(x_array>viga["L1"]) & (x_array<=(viga["L1"]+viga["L2"]))]-
areaV3=sp.lambdify(x,Vx3)(x_array[(x_array>(viga["L1"]+viga["L2"]))&(x_array<=viga["L"])]-(vi
areaV=np.concatenate((areaV1,areaV2,areaV3))

sp.plot((Vx1,(x,0,viga["L1"])),(Vx2.subs(x,x-viga["L1"]),(x,viga["L1"],viga["L1"]+viga["L2"])
```



Out[81]:
```
<sympy.plotting.plot.Plot at 0x2b3583ba920>
```

In [82]:
```
#gráfico de diagrama de momento flector

x_array=np.arange(0,viga["L"],viga["L"]/100)
areaM1=sp.lambdify(x,Mx1)(x_array[x_array<=viga["L1"]])
areaM2=sp.lambdify(x,Mx2)(x_array[(x_array>viga["L1"]) & (x_array<=(viga["L1"]+viga["L2"]))]-
areaM3=sp.lambdify(x,Mx3)(x_array[(x_array>(viga["L1"]+viga["L2"]))&(x_array<=viga["L"])]-(vi
areaM=np.concatenate((areaM1,areaM2,areaM3))

sp.plot((Mx1,(x,0,viga["L1"])),(Mx2.subs(x,x-viga["L1"]),(x,viga["L1"],viga["L1"]+viga["L2"])
```



Out[82]:
```
<sympy.plotting.plot.Plot at 0x2b35c8ba230>
```

In [83]:
```
#Cálculo de magnitud y posición donde se genera la máxima fuerza cortante y el momento intern

#CORTANTE EN kN
Vmax=0
```

```
Vmaxpositivo=np.amax(areaV)
Vmaxnegativo=np.abs(np.amin(areaV))
if Vmaxpositivo>Vmaxnegativo:
    Vmax=Vmaxpositivo
else:
    Vmax=Vmaxnegativo
print("CORTANTE MÁXIMO = ",Vmax," kN en x = ",L , "METROS")

#MOMENTO EN kN*m
Mmax=0
Mmaxpositivo=np.amax(areaM)
Mmaxnegativo=np.abs(np.amin(areaM))
if Mmaxpositivo>Mmaxnegativo:
    Mmax=Mmaxpositivo
else:
    Mmax=Mmaxnegativo
print("MOMENTO MÁXIMO = ", Mmax, " kN*m en x = ",0 , "METROS")
```

```
CORTANTE MÁXIMO =  0.8836491014831834  kN en x =  3.75 METROS
MOMENTO MÁXIMO =  2.13455001597861  kN*m en x =  0 METROS
```

EJERCICIO 2

In [84]:
```
#Esfuerzo admisible
esfadm=250 #en MPa
#Lectura de perfiles
perfiles=pd.read_excel("T2.xlsx",usecols="B:E",nrows=10,skiprows=3,index_col='perfil')

h=perfiles["h [mm]"]
b=perfiles["b [mm]"]
t=perfiles["t [mm]"]
```

In [85]:
```
#Área
variables=perfiles
variables['A [mm ^ 2]']=b*t+(h-t)*t
#Centroide
centroide=((b*t*(h-t/2))+((h-t)*t*(h-t)/2))/(b*t+(h-t)*t)

#FIGURA DE ARRIBA ES 1
Iz1=(b*pow(t,3)/12)+b*t*pow(centroide-(h-(t/2)),2)
#FUGURA DE ABAJO ES 2
Iz2=(t*pow((h-t),3)/12)+(h-t)*t*pow(centroide-(h-t)/2,2)

#INERCIA
Iz=Iz1+Iz2
variables['Iz [mm ^ 4]']=Iz

#EJE NEURTRO
EjeNeutro=centroide
variables['Eje neutro [mm]']=EjeNeutro

# distancia más lejana al eje neutro a compresión
yc=h-EjeNeutro
# distancia más lejana al eje neutro a tensión
yt=EjeNeutro

#ESFUERZOS
#Esfuerzo máximo a tensión
Esftension=Mmax*pow(10,6)*yt/Iz
variables['Esfuerzo Inf [MPa]']=Esftension
#Esfuerzo máximo a compresión
Esfcompresion=Mmax*pow(10,6)*yc/Iz
```

```
variables['Esfuerzo Sup [MPa]']=Esfcompresion

#Tabla que se va a exportar a excel
variables_exportar=variables.loc[:,'A [mm ^ 2]':]
#Tabla con todo
variables
```

Out[85]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]	Iz [mm ^ 4]	Eje neutro [mm]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
1	25	50	5.0	350.00	1.501488e+04	18.928571	2690.929257	863.128252
2	30	60	5.5	464.75	2.910029e+04	22.900888	1679.814598	520.730590
3	35	70	6.0	594.00	5.113253e+04	26.873737	1121.856012	339.234415
4	40	80	7.0	791.00	8.866585e+04	30.659292	738.094716	224.869093
5	50	100	8.5	1202.75	2.115530e+05	38.417845	387.632489	116.862875
6	55	115	9.0	1449.00	3.036222e+05	42.642857	299.791378	86.874219
7	60	120	10.0	1700.00	4.318137e+05	46.176471	228.260429	68.332740
8	70	140	11.5	2282.75	7.908459e+05	53.935139	145.574826	43.360219
9	80	160	13.0	2951.00	1.337393e+06	61.693833	98.466619	29.217611
10	90	180	15.0	3825.00	2.186057e+06	69.264706	67.632720	20.246738

In [86]:
```
#Filtro de perfiles que soportan los esfuerzos tanto superior como inferior
filtrosup=variables['Esfuerzo Sup [MPa]']<=250
filtroinf=variables['Esfuerzo Inf [MPa]']<=250
variables[filtrosup]
variables[filtroinf]
```

Out[86]:

perfil	h [mm]	b [mm]	t [mm]	A [mm ^ 2]	Iz [mm ^ 4]	Eje neutro [mm]	Esfuerzo Inf [MPa]	Esfuerzo Sup [MPa]
7	60	120	10.0	1700.00	4.318137e+05	46.176471	228.260429	68.332740
8	70	140	11.5	2282.75	7.908459e+05	53.935139	145.574826	43.360219
9	80	160	13.0	2951.00	1.337393e+06	61.693833	98.466619	29.217611
10	90	180	15.0	3825.00	2.186057e+06	69.264706	67.632720	20.246738

In [89]:
```
#Selección del perfil óptimo
areamin=0
indice=0
#Áreas minimas que puede soportar por el esfuerzo inferior y superior
areamininf=variables.loc[filtroinf,'A [mm ^ 2]'].min()
areaminsup=variables.loc[filtrosup,'A [mm ^ 2]'].min()
if areamininf<areaminsup:
    areamin=areaminsup
    indice=variables.loc[filtrosup,'A [mm ^ 2]'].idxmin()
else:
    areamin=areamininf
    indice=variables.loc[filtroinf,'A [mm ^ 2]'].idxmin()
print("EL ÁREA MÍNIMA SERÍA = ",areamin)

#Selección de las columnas que se deben colocar en el respectivo lugar de la hoja de cálculo
```

```
perfil_optimo1=perfiles.loc[[indice],'b [mm]']
perfil_optimo2=perfiles.loc[[indice],'h [mm]']
perfil_optimo3=perfiles.loc[[indice],'t [mm]']
perfil_optimo4=perfiles.loc[[indice],'A [mm ^ 2]']
perfil_optimo5=perfiles.loc[[indice],'Esfuerzo Inf [MPa]':'Esfuerzo Sup [MPa]']
```

```
EL ÁREA MÍNIMA SERÍA =  1700.0
```

In [90]:
```
#Exportar en excel
from openpyxl import load_workbook
with pd.ExcelWriter('T2.xlsx',mode='a',if_sheet_exists='overlay',engine='openpyxl') as writer
    book=load_workbook('T2.xlsx')
    writer.book=book
    writer.sheets=dict((ws.title,ws)for ws in book.worksheets)
    #PARA PERFIL ÓPTIMO SELECCIONADO
    perfil_optimo1.to_excel(writer,startrow=18,startcol=6,sheet_name='Perfiles',index=None)
    perfil_optimo2.to_excel(writer,startrow=18,startcol=7,sheet_name='Perfiles',index=None)
    perfil_optimo3.to_excel(writer,startrow=18,startcol=8,sheet_name='Perfiles',index=None)
    perfil_optimo4.to_excel(writer,startrow=18,startcol=9,sheet_name='Perfiles',index=None)
    perfil_optimo5.to_excel(writer,startrow=18,startcol=10,sheet_name='Perfiles',index=None)

    #PARA VARIABLES SOLICITADAS DE CADA PERFIL
    variables_exportar.to_excel(writer,startrow=3,startcol=10,sheet_name='Perfiles',index=Non
```

In [ ]: