Diseño de actividades practicas basadas en Python para la solución de problemas de optimización en la asignatura Investigación de Operaciones.

Jerónimo Espinosa Alvernia

Trabajo de Grado para Optar el título de Ingeniera Industrial.

Director

Néstor Raúl Ortiz Pimiento

Doctor en Ingeniería

Codirector

Iván David Ortiz Pineda

Doctor en Ciencias de la computación (c)

Universidad Industrial de Santander

Facultad de Ingenierías Físicomecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2025

Tabla de Contenido

Pág.

Lista de Figuras

Pág.

Lista de Tablas

Pág.

Lista de Apéndices

Resumen

Titulo: Diseño de actividades practicas basadas en Python para la solución de problemas de optimización en la asignatura Investigación de Operaciones.

Autores: Jeronimo Espinosa Alvernia **

Palabras Clave: Python, Optimización, Gurobi, Investigación de Operaciones, Programación lineal, Pyomo, Ingeniería Industrial, Enseñanza-aprendizaje, Actividades prácticas, Herramientas computacionales.

Descripción:

En el presente proyecto tiene como propósito diseñar, implementar y evaluar actividades practicas orientadas a fortalecer la compresión de las técnicas y herramientas de solución de problemas de optimización lineal en la asignatura Investigación de Operaciones I, empleando el lenguaje de programación de Python y la biblioteca, que también cuenta con su solver, Gurobi. La propuesta se estructura a partir de la formulación de los problemas representativos, el desarrollo de material computacional interactivo y la creación de instrumentos de evaluación que permiten medir el nivel de apropiación de los contenidos por parte de los estudiantes.

La metodología incluye la elaboración de ejercicios guiados y de refuerzo, un manual de usuario para el uso de herramientas y un recurso audiovisual complementario. Las actividades principales son un taller práctico grupal y el desarrollo de un caso de estudio igualmente grupal, ambos evaluados mediante una rubrica técnica. Además, se aplica una encuesta diagnostica de satisfacción y autoevaluación para valorar la percepción de los estudiantes y el impacto de la metodología.

Los resultados muestran que la mayoría de los estudiantes alcanzan un nivel de desempeño excelente en ambas actividades, evidenciando dominio en la formulación, implementación y

análisis de modelos de optimización. La encuesta refleja una alta aceptación de la estrategia pedagógica, así como un interés significativo en seguir aplicando estas herramientas en el futuro.

En síntesis, la experiencia desarrollada demuestra que la integración de herramientas computacionales en el proceso de Enseñanza-Aprendizaje de la optimización favorece el aprendizaje significativo, potencia las competencias técnicas y fomenta la participación activa, constituyéndose en una propuesta viable y replicable en otros contextos académicos.

---

*Trabajo de grado

** Facultad de ingenierías físico-mecánicas. Escuela de estudios industriales y empresariales. Director: Dr. Ing. Néstor Raúl Ortiz Pimiento

Abstract

Title: Design of Practical Activities Based on Python for Solving Optimization Problems in the Operations Research Course.

Authors: Jeronimo Espinosa Alvernia**

Keywords: Python, Optimization, Gurobi, Linear Programming, Operations Research, Practical Activities, Teaching of Optimization, Industrial Engineering, Computational Tools.

Description:

This project aims to design, implement, and evaluate practical activities intended to strengthen the understanding of optimization problem-solving techniques in the Operations Research I course, using the Python programming language and the gurobipy library-solver. The proposal is structured around the formulation of representative problems, the development of interactive computational materials, and the creation of assessment tools that allow measuring the level of content appropriation by students.

The methodology includes the creation of guided and reinforcement exercises, a user manual for the tools, and a complementary audiovisual resource. The main activities are a group practical workshop and the development of a case study, both evaluated using a technical rubric. In addition, a diagnostic satisfaction and self-assessment survey is applied to assess students' perceptions and the impact of the methodology.

The results show that most groups achieve a performance level classified as excellent in both activities, demonstrating mastery in the formulation, implementation, and analysis of optimization models. The survey reflects a high level of acceptance of the pedagogical strategy, as well as significant interest in continuing to apply these tools in the future.

In summary, the developed experience demonstrates that integrating computational tools into the teaching-learning process of optimization fosters meaningful learning, enhances technical skills, and encourages active participation, making it a viable and replicable proposal in other academic contexts.

---

*Thesis

** Faculty of Physical and Mechanical Engineering. School of Industrial and Business Studies. Director: Dr. Ing. Néstor Raúl Ortiz Pimiento

Introducción

En la enseñanza de la educación superior, particularmente en el programa de Ingeniería Industrial de la Universidad Industrial de Santander, enfrenta el desafío de mantenerse alineada con las exigencias del mercado laboral y las tendencias tecnológicas actuales. En la asignatura de Investigación de Operaciones I, tradicionalmente se han utilizado herramientas como Microsoft Excel Solver y el software especializado GAMS (General Algebraic Modeling System) para resolver problemas de optimización. Estas herramientas, aunque efectivas en ciertos aspectos, presentan limitaciones significativas, como altos costos de licenciamiento en el caso de GAMS, y una funcionalidad limitada para problemas complejos en el caso de Excel Solver.

En paralelo, Python ha surgido como una herramienta poderosa en el ámbito académico y profesional, gracias a su código abierto, una vasta comunidad de desarrolladores, y una amplia gama de bibliotecas diseñadas para resolver problemas matemáticos y de optimización. Su facilidad de uso y la posibilidad de aplicarlo en contextos diversos han hecho que Python sea preferido por las empresas y organizaciones a nivel global. Esta tendencia plantea la necesidad de adoptar un enfoque más moderno en la enseñanza de la optimización, con el objetivo de preparar a los estudiantes para los retos reales que enfrentarán en su vida profesional.

El presente proyecto de grado busca cerrar esta brecha mediante el diseño de un conjunto de actividades prácticas y materiales pedagógicos que permitan la transición del uso de GAMS a Python en la asignatura de Investigación de Operaciones I. Para ello, se realizó una revisión exhaustiva de literatura sobre las bibliotecas matemáticas y solvers disponibles en Python, tales como NumPy, SciPy, PuLP, y Pyomo, analizando su implementación en un contexto educativo. Además, se diseñaron estrategias de aprendizaje activo que faciliten la comprensión de conceptos

teóricos y su aplicación práctica, a través de metodologías como el aprendizaje basado en problemas, estudios de caso, y evaluaciones formativas.

No solo se busca actualizar el contenido de la asignatura, sino también fomentar una enseñanza más inclusiva, accesible y alineada con las demandas del entorno profesional. Se espera que los materiales diseñados contribuyan a mejorar la experiencia de aprendizaje de los estudiantes, al tiempo que se establecen bases metodológicas replicables en otras asignaturas y programas. Además, con la implementación de este proyecto, se pretende consolidar una enseñanza coherente y homogénea dentro del programa de Ingeniería Industrial, garantizando que los estudiantes desarrollen competencias técnicas y prácticas en el uso de herramientas tecnológicas modernas. De esta manera, se responde a la necesidad de formar profesionales preparados para liderar en un entorno dinámico y competitivo, fortaleciendo tanto su perfil académico como sus oportunidades laborales.

1. Planteamiento del problema

En la actualidad, la asignatura de Investigación de Operaciones I, que forma parte del plan de estudios de la carrera de Ingeniería Industrial en la Universidad Industrial de Santander, incluye un tema de estudio relacionado con el proceso de solución de problemas de optimización utilizando GAMS (General Algebraic Modeling System). GAMS es una herramienta robusta y ampliamente utilizada tanto en la academia como en sectores profesionales específicos, especialmente en el ámbito de la investigación operativa y la optimización matemática. Sin embargo, aunque sigue siendo útil y efectivo, las empresas se han inclinado por la implementación de softwares libres que no representen altos costos, como Python, han ganado mayor aceptación debido a su flexibilidad y la vasta comunidad de desarrolladores que respalda su uso. Python ha demostrado ser una herramienta fundamental en diversas disciplinas, incluidas la ciencia de datos y la investigación operativa, y su popularidad en la industria se ha incrementado gracias a su facilidad de uso, la extensa disponibilidad de bibliotecas matemáticas, y su aplicabilidad en múltiples contextos. Además, en otras asignaturas del programa de Ingeniería Industrial, como es el área de Estadística, ya se ha realizado la transición hacia el uso de Python, lo que genera una oportunidad para alinear la enseñanza de Investigación de Operaciones I con las necesidades actuales del mercado profesional.

Este desfase entre el uso de GAMS y Python en las distintas asignaturas puede generar incoherencias en la formación de los estudiantes, quienes requieren estar preparados para enfrentar desafíos laborales donde Python es la herramienta predominante. Por esta razón, es pertinente realizar una transición en la enseñanza del proceso de solución de problemas de optimización, en Investigación de Operaciones I, incorporando Python como el lenguaje principal para su solución.

Esta transición no solo modernizará el enfoque de la asignatura, sino que también permitirá a los estudiantes adquirir competencias prácticas que están en demanda en el ámbito profesional.

El objetivo de este proyecto de grado es diseñar materiales de estudio que faciliten dicha transición, proporcionando a los estudiantes los recursos necesarios para aprender y aplicar los conceptos de optimización utilizando Python. Estos materiales permitirán consolidar este nuevo enfoque dentro del plan de clase de la asignatura, mejorando la preparación de los estudiantes y garantizando que su formación sea acorde con las exigencias actuales del entorno laboral. La transición a Python, además, promoverá una enseñanza más homogénea dentro de la carrera, ya que otras asignaturas clave ya han adoptado este lenguaje como herramienta principal de aprendizaje.

2. Diagnóstico inicial

La asignatura de Investigación de Operaciones I en la Universidad Industrial de Santander se encuentra orientada a la enseñanza de técnicas y métodos matemáticos para la solución de problemas de optimización, como la optimización lineal, problema dual, análisis de sensibilidad, Método Branch and Bound para Problemas Enteros y otros conceptos relevantes fundamentales en este ámbito, para así la exitosa formación de los estudiantes de Ingeniería Industrial. Actualmente, esta asignatura utiliza herramientas tecnológicas como Microsoft Excel con su complemento Solver, y el software especializado GAMS para enseñar a los estudiantes cómo modelar y resolver problemas de optimización en diversos contextos. Estas herramientas han sido tradicionalmente empleadas para complementar los conceptos teóricos, permitiendo a los estudiantes aplicar los métodos aprendidos en problemas prácticos de naturaleza real. Cada docente, en función de su experiencia y enfoque metodológico, adapta diferentes estrategias de enseñanza y de evaluación para así medir las competencias de sus estudiantes. Estas evaluaciones generalmente consisten en parciales escritos, talleres y ejercicios prácticos en los cuales los estudiantes deben demostrar su capacidad para modelar y resolver problemas utilizado las herramientas tecnológicas anteriormente mencionadas. Mientras que Microsoft Excel y su Solver se utilizan principalmente para introducir a los estudiantes en la resolución computacional de problemas de optimización lineal, en el caso de GAMS se emplea en problemas de mayor complejidad debido a la capacidad de manejar modelos más avanzados.

Si bien estas herramientas han resultados funcionales en el proceso de enseñanza, presentan limitaciones que justifican una necesidad de actualizar y modernizar tanto como las metodologías, como los recursos que se utilizan en la asignatura actualmente. Además, aun que GAMS es una

herramienta robusta y ampliamente reconocida, los costos asociados con su licenciamiento restringen su accesibilidad para los estudiantes, especialmente fuera del entorno universitario.

Python, es un lenguaje de programación con código abierto, ha demostrado a ser una herramienta versátil y útil en el ámbito de la optimización gracias a la disponibilidad de bibliotecas especializadas y Solvers avanzados. Su popularidad ha crecido exponencialmente en la industria debido a su capacidad para abordar problemas de diferentes niveles de complejidad, su facilidad de integración con otras tecnologías y su amplia comunidad de soporte, esto plantea la oportunidad

de implementar Python como herramienta enseñanza para la asignatura, alineando así la formación académica de los estudiantes de los estudiantes con las demandas tecnológicas del entorno profesional y empresarial. En este contexto, este proyecto tiene como propósito diseñar un conjunto de actividades prácticas que promuevan la transición del uso de GAMS hacia Python, optimizando recursos didácticos y metodológicos de la asignatura, estas actividades se desarrollaran con el objetivo de facilitar la comprensión de las técnicas de solución de problemas de optimización mediante Python, permitiendo que los estudiantes adquieran competencias modernas.

Con esta iniciativa se espera no solo modernizar las enseñanzas de la asignatura, sino también establecer una metodología que pueda ser replicada en otras asignaturas o programas, promoviendo la integración de herramientas tecnológicas de código abierto en el proceso de formación de ingenieros industriales, cumpliendo así a la necesidad de una actualización tecnológica, fomentando una enseñanza más inclusiva, accesible y alineada con las tendencias del mercado laboral contemporáneo.

3. Objetivos

3.1. Objetivo General

Diseñar actividades de tipo práctico que favorezcan la comprensión de las técnicas de solución de la solución de problemas de optimización mediante el uso de Python

3.2. Objetivos Específicos

Elaborar una revisión literaria sobre las principales bibliotecas matemáticas utilizados en Python para la implementación de algoritmos de optimización.

Realizar una revisión literaria sobre los diferentes Solvers disponibles en Python para la resolución de problemas de optimización.

Sintetizar los resultados obtenidos de la revisión de bibliotecas y Solvers para los modelos de resolución de problemas de optimización en Python

Proponer una serie de ejemplos de clase, actividades prácticas, que se utilizarán para el desarrollo de la temática propuesta.

Establecer métodos de evaluación y medir el nivel de desempeño alcanzado por los estudiantes

4. Marco de referencia

4.1. Marco Antecedentes

Peñaloza Y Arámbulo, M. (2024). *El software PHPSimplex y la capacidad de aprendizaje de la programación lineal en estudiantes universitarios*.

Este estudio evalúa cómo el uso del software PHPSimplex puede mejorar las capacidades cognitivas, procedimentales y actitudinales de estudiantes universitarios en programación lineal. Esto mediante un diseño no experimental, transversal y descriptivo, se identificó que el software apoya significativamente las capacidades procedimentales y actitudinales, promoviendo asi el aprendizaje activo y la motivación de los estudiantes. La investigación resalta la importancia de integrar herramientas tecnológicas en la educación matemática para superar barreras en el aprendizaje de conceptos abstractos como los de programación lineal

Bonito Gunsha, J. D. (2021). *Software TORA en el proceso de enseñanza-aprendizaje de programación lineal en los estudiantes de tercer semestre de la carrera de pedagogía de las ciencias experimentales matemáticas y física en el periodo noviembre 2020-abril 2021*.

Este trabajo aborda el uso del software GeoGebra para enseñar programación lineal en educación secundaria, promoviendo la construcción del conocimiento a través de la interacción con representaciones gráficas y algebraicas. El marco teórico se fundamenta en la Teoría de Registros de Representación Semiótica, destacando la transición fluida entre distintos registros. Los resultados indican que GeoGebra facilita la resolución precisa de problemas, mejora las

habilidades matemáticas y aumenta la motivación estudiantil, lo cual apoya la enseñanza de manera efectiva

Bello Durand, J. B. (2013). *Mediación del software GeoGebra en el aprendizaje de programación lineal en alumnos del quinto grado de educación secundaria*.

Esta investigación utiliza un enfoque cuasi-experimental para determinar la influencia del software TORA en el aprendizaje de programación lineal en estudiantes de pedagogía. Los hallazgos muestran una mejora significativa en el rendimiento académico al implementar TORA, destacando su capacidad para reducir la complejidad de los cálculos y permitir un enfoque en la interpretación de resultados. El estudio concluye que el software no solo optimiza el proceso de aprendizaje, sino que también mejora la receptividad y el interés de los estudiantes por el tema

Estos antecedentes son altamente relevantes para la transición de GAMS a Python en el curso de Investigación de Operaciones 1. Refuerzan la importancia de incorporar herramientas tecnológicas que fomenten un aprendizaje significativo y habilidades prácticas. El uso de software como PHPSimplex, GeoGebra o TORA demuestra cómo las TIC pueden facilitar la comprensión de problemas complejos y aumentar la motivación estudiantil. Esto respalda el diseño de materiales educativos con Python que ofrezcan similares beneficios al integrar herramientas de optimización y visualización gráfica.

4.2. Marco teórico

4.2.1. Conceptos fundamentales de los modelos de optimización

La optimización se entiende como el proceso de determinar la mejor solución posible dentro de un conjunto de alternativas factibles y bajo ciertas restricciones. Un modelo de optimización incluye tres elementos principales: variables de decisión, restricciones y una función objetivo que se busca maximizar o minimizar.

Los problemas se clasifican según la naturaleza de las variables y restricciones. En la programación lineal, la función objetivo y las restricciones son expresiones lineales y las variables toman valores continuos. En la programación entera, todas las variables deben ser enteras, mientras que en la programación binaria solo se permiten los valores 0 y 1. Finalmente, en la programación lineal entera mixta, conviven variables continuas e enteras, lo que permite representar escenarios más complejos y realistas.

Para la resolución de estos problemas se utilizan métodos exactos como el método símplex en modelos lineales continuos, o los algoritmos de ramificación y acotamiento y ramificación y corte en problemas enteros y mixtos. Actualmente, solvers como Gurobi integran estos métodos de forma eficiente, lo que facilita la resolución de problemas de gran escala en contextos académicos e industriales.

4.2.2. Optimización y su enseñanza en la educación.

La optimización es un campo fundamental de las matemáticas aplicadas y la ingeniería que busca maximizar o minimizar funciones bajo ciertas restricciones. En el contexto de la asignatura Investigación de Operaciones I, se utiliza tradicionalmente software como GAMS para modelar y resolver problemas de optimización. Sin embargo, la evolución de las herramientas tecnológicas

ha generado un interés creciente en alternativas como Python, que ofrece flexibilidad, acceso gratuito y una amplia gama de bibliotecas para resolver problemas de optimización lineal, entera y no lineal.

La enseñanza de optimización no solo debe enfocarse en la teoría, sino también en el desarrollo de competencias prácticas mediante actividades que simulen problemas del mundo real. Según el plan estratégico de la Escuela de Estudios Industriales y Empresariales (EEIE), las tendencias en la educación apuntan hacia la integración de metodologías activas y herramientas tecnológicas que fomenten el aprendizaje experiencial

4.2.3. Rol de las TIC en la enseñanza de la optimización

Las Tecnologías de la Información y la Comunicación (TIC) representan un recurso esencial en los procesos educativos contemporáneos, al facilitar la creación de entornos interactivos que promueven la autonomía, la motivación y la colaboración entre estudiantes.

En el caso de la enseñanza de la optimización, herramientas como Python, la biblioteca Gurobi y entornos en la nube como Google Colab permiten integrar la teoría con la práctica de manera accesible y dinámica. Estas plataformas posibilitan la experimentación con modelos matemáticos en tiempo real, la validación de hipótesis y la elaboración de soluciones reproducibles, contribuyendo así a un aprendizaje más profundo y significativo.

4.2.4. Python como herramienta pedagógica.

Python ha emergido como una de las herramientas más relevantes en la resolución de problemas matemáticos y de optimización, gracias a su comunidad activa y la disponibilidad de bibliotecas específicas. Entre las más destacadas se encuentran:

- *NumPy y SciPy:* Para cálculos numéricos avanzados.
- *Pandas:* Para el manejo y análisis de datos estructurados.
- *PuLP y Pyomo:* Para formular y resolver problemas de optimización lineal y no lineal.
- *Matplotlib y Seaborn:* Para visualización de datos.

El uso de Python en la enseñanza de asignaturas como Investigación de Operaciones I no solo moderniza el currículo, sino que también alinea la formación académica con las competencias requeridas en la industria, destacando su aplicabilidad en áreas como logística, producción y finanzas.

4.2.5. Ventajas de la transición a Python en la enseñanza.

La transición de GAMS a Python en la asignatura Investigación de Operaciones I responde a una necesidad de modernización curricular y coherencia en la formación de los estudiantes de ingeniería industrial. Python no solo es más accesible y económico, sino que también ofrece un ecosistema más amplio para el desarrollo de soluciones personalizadas. Esta transición garantiza que los estudiantes desarrollen competencias relevantes y alineadas con las exigencias del mercado laboral actual.

4.2.6. Metodologías de aprendizaje y estrategias activas.

El enfoque pedagógico para la integración de Python en el currículo debe considerar estrategias de aprendizaje basadas en la práctica. Entre las metodologías activas destacadas en la literatura se encuentran:

- *Aprendizajes basados en problemas*: Los estudiantes resuelven problemas reales que simulan escenarios de la industria, utilizando herramientas de optimización en Python.
- *Estudios de caso*: Se diseñan casos que combinan teoría y práctica, vinculando directamente los conceptos de optimización con sus aplicaciones prácticas**.**
- *Evaluaciones formativas*: Para medir el progreso del aprendizaje en la implementación de algoritmos y en la correcta elección de bibliotecas y Solvers.

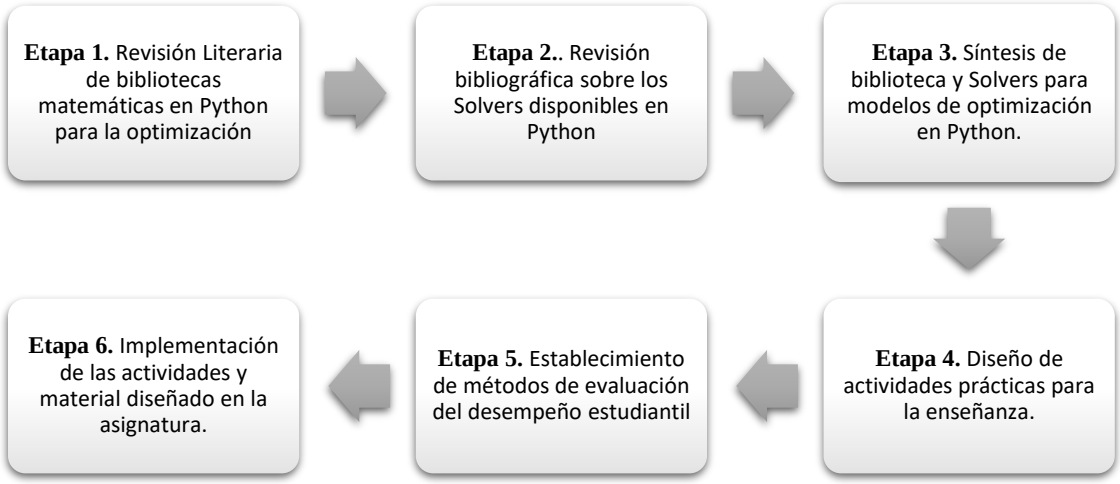
La implementación de estas metodologías fomenta el desarrollo de habilidades técnicas y blandas, como el trabajo en equipo, la comunicación efectiva y la capacidad de resolución de problemas.

5. Metodología

Para el presente proyecto pedagógico, se plantea el diseño de una metodología que permite identificar las bibliotecas y solvers más orientados al desarrollo del aprendizaje práctico y teórico de los problemas de optimización.

Figura 1.

Metodología del proyecto



5.1. Etapa 1: Revisión Literaria de bibliotecas matemáticas en Python para la optimización

Python ofrece diversas bibliotecas matemáticas que facilitan la solución de problemas de programación lineal. Entre las más destacadas se encuentran Pyomo, NumPy, CVXPY y PuLP, las cuales permiten modelar, resolver y analizar problemas de optimización de manera eficiente y flexible.

Pyomo es una biblioteca de código abierto en Python que permite formular, resolver y analizar una amplia variedad de modelos de optimización, incluyendo programación lineal, no

lineal, entera mixta y estocástica, entre otros. Se destaca por integrar sus objetos de modelado dentro de un lenguaje de programación completo, lo que facilita su uso en análisis iterativos y desarrollo de herramientas avanzadas. A diferencia de otros lenguajes como AMPL o GAMS, Pyomo aprovecha la flexibilidad de Python para crear modelos simbólicos, instanciarlos y resolverlos con solvers tanto comerciales como de código abierto.

CVXPY es un lenguaje de modelado incrustado en Python para resolver problemas de optimización convexa de forma intuitiva y cercana a la notación matemática. Además de programación convexa, admite extensiones como programación geométrica generalizada, programación convexa entera mixta y programación cuasiconvexa. Utiliza solvers de código abierto como Clarabel, OSQP y SCS, y es desarrollado activamente por una comunidad global de investigadores e ingenieros. La versión 1.6 introduce mejoras como expresiones N-dimensionales, atributos de dispersión para variables y compatibilidad con Python 3.13. CVXPY promueve la colaboración a través de Discord, GitHub y su guía para contribuidores.

NumPy es una biblioteca esencial para la computación científica en Python, que proporciona estructuras de datos eficientes como arreglos multidimensionales y un conjunto amplio de funciones para operaciones matemáticas, estadísticas y de álgebra lineal. Su capacidad para realizar cálculos vectorizados de forma rápida y optimizada la convierte en una herramienta clave en la resolución de problemas de programación lineal, ya que permite representar y manipular matrices de coeficientes, vectores de restricciones y funciones objetivo de manera eficiente, facilitando tanto la formulación como el análisis numérico de estos modelos.

PuLP es una biblioteca de Python de código abierto diseñada para modelar y resolver problemas de programación lineal y lineal entera mixta de forma sencilla e intuitiva. Su principal fortaleza radica en su sintaxis clara y cercana al lenguaje matemático, lo que permite a los usuarios

definir variables, restricciones y funciones objetivo utilizando expresiones naturales de Python, sin necesidad de aprender un nuevo lenguaje de modelado. PuLP es completamente portable, no requiere dependencias externas, y es compatible con una variedad de solucionadores tanto comerciales (como Gurobi o CPLEX) como de código abierto (como CBC).

5.2. Etapa 2: Revisión bibliográfica sobre los Solvers disponibles en Python

Los principales solvers confiables y oficiales utilizados en Python para resolver problemas de programación lineal son COIN-OR, Gurobi y CPLEX, reconocidos por su eficiencia, precisión y amplio uso en entornos académicos e industriales.

Gurobi es un solver de optimización matemática de alto rendimiento, diseñado para resolver una amplia gama de modelos, como programación lineal (LP), programación entera mixta (MIP), programación cuadrática (QP) y modelos con restricciones cuadráticas (QCP). Su principal función es encontrar soluciones óptimas a problemas de decisión bajo restricciones, y para ello emplea algoritmos avanzados como branch-and-bound, cutting planes y métodos de punto interior. A través de su biblioteca gurobipy, Gurobi se integra fácilmente con Python, lo que permite a los usuarios definir modelos, establecer restricciones, resolverlos y analizar los resultados de forma eficiente.

COIN-OR (Computational Infrastructure for Operations Research) Su objetivo es proporcionar herramientas abiertas para modelar y resolver una amplia gama de problemas de optimización, facilitando la reproducción de resultados y fomentando la colaboración entre investigadores. COIN-OR ofrece una colección de componentes integrados, como la Open Solver Interface, la Cut Generator Library y el Branch-Cut-Price Framework, que son especialmente útiles en la programación entera mixta. Al adoptar el modelo de desarrollo abierto, COIN-OR

permite que la comunidad contribuya al avance de las herramientas de optimización, promoviendo la transparencia y la innovación en el campo.

IBM ILOG CPLEX Optimization Studio es un software de optimización de alto rendimiento diseñado para resolver problemas complejos de programación lineal, entera mixta y cuadrática. Utiliza potentes algoritmos, como el método del simplex y técnicas de punto interior, para proporcionar soluciones eficientes a desafíos de planificación y asignación de recursos. Además, ofrece interfaces versátiles para el desarrollo de aplicaciones de optimización, incluyendo ILOG Concert Technology y CPLEX Callable Library, lo que facilita su integración en diversos entornos empresariales y académicos.

5.3. Etapa 3: Síntesis de biblioteca y Solvers para modelos de optimización en Python

Se lleva a cabo una revisión de la literatura con el objetivo de identificar las principales bibliotecas y solvers utilizados en la resolución de problemas de optimización mediante la herramienta Python. Para ello, se elabora una matriz en la que se registran el título del estudio, el solver empleado, la biblioteca utilizada, el tipo de aplicación o uso y los resultados obtenidos con el uso de estos.

Se utiliza la siguiente ecuación de búsqueda: Python AND (optimization OR "linear programming" OR "integer programming" OR "nonlinear programming") AND (solver OR "mathematical library" OR "optimization library") AND ("Pyomo" OR "CVXPY" OR "Numpy" OR "PuLP" OR "Gurobi" OR "COIN-OR" OR "CPLEX").

Se evidencia que el solver más utilizado por los autores en los estudios analizados para resolver problemas de optimización es Gurobi. Asimismo, se observa que los autores emplean

frecuentemente la biblioteca de Gurobi, la cual proporciona APIs (Bibliotecas de programación) que permiten interactuar con el solver desde diversos lenguajes de programación.

5.4. Etapa 4: Diseñar a las actividades prácticas que se llevaran a cabo para la enseñanza-aprendizaje utilizando Python.

Durante esta etapa se consolida y documenta el material computacional diseñado para apoyar el desarrollo temático de la asignatura Investigación de Operaciones I, haciendo uso del lenguaje de programación Python y la biblioteca oficial del solver Gurobi. La elección de estas herramientas se fundamenta en la síntesis realizada en la etapa anterior, donde se justifica el uso de Gurobi y su API como la opción más adecuada por su robustez, documentación y aceptación en la literatura especializada. Todo el desarrollo se realiza utilizando la plataforma Google Colab como entorno de programación, por su accesibilidad y facilidad de uso para los estudiantes.

El diseño de actividades inicia en el semestre 2024-2, específicamente entre los meses de septiembre y noviembre de 2024, período en el cual se formulan, codifican y validan los primeros materiales. Los problemas incluidos en este conjunto son construidos expresamente para este proyecto, tomando como referencia las estructuras conceptuales presentes en la bibliografía clásica de la asignatura. Cada modelo se ajusta al contexto académico de la asignatura, asegurando coherencia con los contenidos y las competencias que se busca desarrollar.

El proceso comienza con la construcción de modelos básicos de programación lineal, codificados en Python mediante la API de Gurobi. Estos ejercicios introductorios están conformados por problemas con únicamente dos variables y un número reducido de restricciones con valores moderados, lo que facilita la comprensión inicial del método. Su propósito es que los estudiantes se familiaricen con la lectura de datos desde hojas de cálculo, la construcción del

modelo y la interpretación de los resultados. La lógica se implementa paso a paso, con estructuras claras y comentarios explicativos que permiten comprender el flujo completo del modelo, desde la formulación hasta la solución.

De forma complementaria, se organiza un conjunto de 11 ejercicios resueltos orientados al refuerzo y la práctica individual. A diferencia de los básicos, estos incorporan más restricciones, variables, parámetros, límites e índices, con el fin de aumentar la complejidad progresivamente. Cada ejercicio contiene su enunciado acompañado de la solución desarrollada, lo que permite a los estudiantes practicar la formulación matemática, la codificación y la ejecución del modelo tanto ingresando los datos directamente en el código como cargándolos desde archivos externos en Excel. Este material está diseñado para reforzar el aprendizaje autónomo y servir como insumo en espacios de trabajo independiente o evaluaciones prácticas.

Para facilitar el uso del material se elabora un manual de apoyo, el cual explica en detalle la estructura general de un modelo de optimización en Gurobi. Este manual no solo describe la definición de variables, restricciones, función objetivo y métodos de solución, sino que además contiene las líneas de código escritas y comentadas paso a paso, con el fin de que los estudiantes puedan guiarse directamente durante la implementación en Python.

Como complemento audiovisual, Como complemento audiovisual, se desarrolla un video explicativo en formato MP4 que guía al estudiante en la implementación básica de modelos de optimización usando Python y Gurobi. Este recurso expone los ejercicios básicos y la mayoría de los ejercicios de repaso, lo que permite a los estudiantes contar con un apoyo adicional para reforzar su aprendizaje fuera del aula y apropiarse del contenido técnico.

Finalmente, todo el material generado es revisado, corregido y organizado para su entrega formal. Se verifica la funcionalidad del código, la consistencia de los resultados y la claridad de

las explicaciones, asegurando que los productos entregados sean pertinentes tanto para estudiantes como para docentes de la asignatura. El conjunto de recursos se agrupa digitalmente y queda validado para su implementación inmediata en el entorno académico.

5.5. Etapa 5: Establecer los métodos para evaluar las competencias del desempeño estudiantil.

En términos generales, esta etapa permite concretar el objetivo general del presente proyecto, al consolidar el diseño de actividades de tipo práctico orientadas a facilitar la comprensión de las técnicas de solución de problemas de optimización mediante el uso del lenguaje de programación Python. Estos métodos de evaluación y actividades de enseñanza se plantean y diseñan en el semestre correspondiente a 2024-2, en paralelo con el diseño de las actividades, como parte de las estrategias prácticas que se proponen para fortalecer la comprensión de las técnicas de solución de problemas de optimización mediante el uso de Python. En este sentido, se establecen dos actividades principales: la primera consiste en un taller práctico, compuesto por dos ejercicios enfocados en la aplicación de los conocimientos adquiridos durante el proceso formativo; la segunda corresponde al desarrollo de un caso de estudio, el cual se trabaja de manera grupal a lo largo del semestre bajo la orientación del docente de la asignatura, y cuya implementación se realiza empleando el lenguaje de programación Python. Asimismo, se avanza en el cumplimiento de varios objetivos específicos, entre ellos: la propuesta de una serie de ejemplos y ejercicios aplicados (actividades 1 y 2), diseñados para apoyar el desarrollo temático de la asignatura, y la definición de métodos de evaluación para medir el nivel de desempeño alcanzado por los estudiantes en las actividades desarrolladas. Esta evaluación se estructura a

través de una rúbrica, un cuestionario de sustentación oral y una encuesta diagnóstica aplicada al cierre del proceso formativo. Para cada una de estas actividades se define un método de evaluación específico. Adicionalmente, se diseña una encuesta diagnóstica de satisfacción y autoevaluación, con el propósito de valorar de manera integral la percepción de los estudiantes y el impacto de la propuesta pedagógica que se implementa.

5.5.1. Primera actividad: Taller práctico

Tal como se expone previamente, esta actividad está orientada al desarrollo de dos ejercicios relacionados con la programación lineal entera, con el propósito de fortalecer y afianzar los conocimientos teóricos adquiridos durante las sesiones de clase. Asimismo, se busca que los estudiantes se familiaricen con el uso del lenguaje de programación Python como herramienta para la formulación y resolución de problemas de optimización, promoviendo así el desarrollo de competencias prácticas aplicables al contexto profesional.

Cada grupo de trabajo, conformado por tres estudiantes, selecciona uno de los dos ejercicios propuestos para su implementación. El desarrollo del taller refleja tanto la correcta formulación matemática del problema como su solución computacional. Para ello, se exige la entrega del código fuente implementado en Python, el cual debe estar debidamente comentado y estructurado, así como un archivo en formato Excel que contiene la información del ejercicio.

Esta actividad tiene como finalidad no solo evaluar la apropiación de los conceptos teóricos, sino también fomentar el trabajo colaborativo, la capacidad de análisis y la aplicación práctica de herramientas computacionales en la solución de problemas reales de optimización. Los datos específicos de cada ejercicio se generan de forma aleatoria mediante un código en Python,

que se comparte previamente con todos los estudiantes. Este código funciona con las dos últimas cifras del número de identificación de cada estudiante para personalizar los datos.

- **Ejercicio 1:** Una empresa fabrica sillas y mesas, cada una con diferentes requerimientos de madera, pintura y mano de obra. El objetivo del modelo es determinar cuántas unidades de cada producto deben fabricarse para maximizar la utilidad, considerando las restricciones de disponibilidad de recursos.

Figura 2.

Ejercicio 1: Sillas y mesas.

```
[ ]  # Ingresa aquí
     d = int(input("Ingresa la suma de los 2 últimos números de su codigo estudiantil: "))

     # Parámetros personalizados
     a1 = 30 + d % 3
     a2 = 40 + d % 4
     b1 = 20 + d % 2
     b2 = 30 + (d + 1) % 3
     c1 = 40 + d % 2
     c2 = 50 + (d + 2) % 3

     M = 1000 + 100 * d
     P = 600 + 500 * d
     H = 800 + 70 * d

     u1 = 400 + 20 * d
     u2 = 700 + 30 * d

     # Mostrar parámetros
     print(f"Parámetros personalizados para el estudiante con ID terminado en {d}:")
     print(f"Silla - Madera: {a1}, Pintura: {b1}, Mano de obra: {c1}, Precio: ${u1}")
     print(f"Mesa  - Madera: {a2}, Pintura: {b2}, Mano de obra: {c2}, Precio: ${u2}")
     print(f"Disponibilidad - Madera: {M}, Pintura: {P}, Mano de obra: {H}")
```

- **Ejercicio 2:** Una empresa tecnológica produce laptops y tablets para exportación. Cada producto requiere diferentes cantidades de microchips, horas de ensamblaje y pantallas LCD. El objetivo es maximizar la utilidad semanal sin exceder la disponibilidad de recursos.

Figura 3.

Ejercicio 2: Laptops y Tablets

```
[ ]  # Ingreso del dígito
     d = int(input("Ingresa el último dígito de tu cédula: "))
     # Recursos requeridos por unidad
     chips_laptop = 10 + d % 3
     chips_tablet = 8 + d % 2
     lcd_laptop = 1
     lcd_tablet = 1
     ensam_laptop = 12 + d % 4
     ensam_tablet = 8 + d % 3
     # Recursos disponibles
     chips_total = 1000 + 100 * d
     lcd_total = 800 + 50 * d
     ensam_total = 1200 + 75 * d
     # Ingresos unitarios (en millones)
     precio_laptop = 6 + 0.4 * d
     precio_tablet = 4 + 0.3 * d
     # Mostrar parámetros
     print(f"Datos personalizados para: {d}")
     print(f"Chips disponibles: {chips_total}, LCD: {lcd_total}, Ensamblaje: {ensam_total}")
     print(f"Laptop -> chips: {chips_laptop}, LCD: 1, Ensamblaje: {ensam_laptop}, Precio: ${precio_laptop:.2f}M")
     print(f"Tablet -> chips: {chips_tablet}, LCD: 1, Ensamblaje: {ensam_tablet}, Precio: ${precio_tablet:.2f}M")
```

Para la evaluación del taller se diseña una rúbrica compuesta por 10 criterios, cada uno con el mismo peso porcentual. La calificación final del taller se obtiene como el promedio de las notas asignadas en cada uno de estos criterios.

Tabla 1.

Rubrica de evaluación.

Rúbrica	Excelente	Necesita mejorar	deficiente
Identificación correcta de variables, restricciones y función objetivo	Identifica con claridad todos los elementos del modelo, justifica su rol dentro del problema y los representa adecuadamente.	Se identifican la mayoría de elementos, pero con errores menores o explicaciones vagas.	Se omiten o confunden elementos clave del modelo, sin justificación ni coherencia.
Formulación adecuada del modelo matemático	Traduce correctamente el problema a un modelo matemático bien estructurado.	El modelo presenta errores parciales en lógica o sintaxis.	El modelo es incoherente o no corresponde al problema.

Correcta carga de datos desde archivo Excel	Integra el archivo correctamente, usando referencias bien estructuradas y comprensibles.	Se presenta la carga de datos, pero con errores de formato, rutas o estructura.	No se logra integrar el archivo o se utilizan datos mal estructurados.
Implementación del modelo en Python con gurobipy	El código es funcional, bien comentado y aprovecha la sintaxis correcta de Gurobi.	El código es funcional, pero carece de claridad o tiene errores de estilo.	El código no funciona, presenta errores críticos o no emplea gurobipy correctamente.
Ejecución exitosa del modelo sin errores	El modelo corre sin errores, devolviendo resultados óptimos y esperados.	El modelo corre con advertencias o necesita ajustes manuales.	El modelo no corre, genera errores lógicos o de compilación.
Presentación clara de la solución (valores óptimos, utilidad total, etc.)	Interpreta con precisión los resultados, explicando su significado en el contexto del problema.	La interpretación es parcial o poco clara.	La solución no se interpreta, es errónea o está ausente.
Capacidad para modificar parámetros o estructura del modelo	Realiza cambios con sentido técnico, argumentando las decisiones de manera acertada.	Realiza cambios, pero con errores menores o sin justificación sólida.	No logra modificar el modelo adecuadamente o rompe su funcionalidad.
Comprensión de los efectos de las restricciones en la solución	Explica cómo las restricciones afectan la solución y demuestra capacidad de análisis.	Tiene una comprensión general, pero superficial o con confusiones.	No comprende el rol de las restricciones o las interpreta de manera incorrecta.
Entrega del archivo Excel con datos correctamente estructurados	El archivo está completo, limpio, con etiquetas claras y consistentes con el modelo.	Tiene errores menores de estructura o presentación.	Está incompleto, desorganizado o es ilegible.
Notebook limpio, comentado y bien estructurado	La libreta es clara, con comentarios útiles y orden lógico.	Presenta desorden o comentarios escasos.	Es difícil de seguir, sin comentarios o mal organizada.

5.5.2. Segunda actividad: Desarrollo de caso de estudio en Python

A lo largo del semestre, los estudiantes desarrollan un caso de estudio asignado por el docente de la asignatura, el cual se evalúa mediante una serie de exposiciones distribuidas en diferentes etapas. Tradicionalmente, la etapa final de este caso se aborda utilizando exclusivamente el software GAMS (General Algebraic Modeling System), herramienta ampliamente empleada en la formulación y solución de modelos de optimización matemática.

No obstante, con el objetivo de fomentar la adquisición de nuevas competencias tecnológicas y promover la adaptación a entornos de programación más versátiles y de amplio uso en la industria y la investigación, en esta ocasión se propone un cambio metodológico. En lugar de utilizar GAMS, se solicita a los estudiantes implementar la solución del caso de estudio utilizando el lenguaje de programación Python.

La inclusión de Python en esta etapa responde a su creciente relevancia en el ámbito académico y profesional. Su adopción permite a los estudiantes no solo reforzar su comprensión de los modelos de optimización, sino también desarrollar habilidades prácticas en un entorno de programación ampliamente demandado, lo cual representa un valor agregado significativo en su formación.

El caso de estudio tiene como eje central el desarrollo de un modelo de negocio orientado a la creación de una pizzería. En este contexto, los estudiantes realizan una labor investigativa que les permite identificar, analizar y sustentar los distintos elementos que conforman dicho modelo. Para ello, recopilan información pertinente relacionada con aspectos clave del negocio, tales como:

- Costos de insumos y materiales necesarios para la elaboración de cada tipo de pizza
- Tiempo requerido en el proceso de producción
- Costos de mano de obra
- Costo total por unidad de producto
- Precio de venta
- Demanda mínima y máxima estimada para cada producto

Con base en esta información, los estudiantes formulan un modelo matemático de programación lineal que incluya:

- Variables de decisión
- Función objetivo (maximización de la utilidad)
- Restricciones

La valoración de esta actividad se basa en dos aspectos principales: por un lado, la correcta implementación de la solución del caso de estudio utilizando el lenguaje de programación Python; y por otro, la exposición oral del trabajo durante la tercera presentación programada en el cronograma de la asignatura. Ambos elementos evidencian el nivel de comprensión del problema, la aplicación adecuada de los conceptos aprendidos y el manejo de herramientas tecnológicas en contextos prácticos.

5.5.3. Sustentación de la segunda actividad

La sustentación se desarrolla a partir de un conjunto de preguntas previamente diseñadas, elaboradas con base en los contenidos tratados durante la implementación de los ejercicios utilizando Python y el solver Gurobi. Este cuestionario está estructurado con el propósito de evaluar no solo el dominio conceptual de los estudiantes, sino también su capacidad para aplicar dichos conocimientos en un entorno práctico. En ese sentido, se espera que los estudiantes editen y expliquen su código, validen los resultados obtenidos y argumenten de manera coherente las decisiones adoptadas en el proceso de modelado.

Tabla 2.

Cuestionario evaluativo.

CUESTIONARIO
1. Si se prefiere trabajar con variables Binarias / Enteras / Continuas, ¿Qué se debería hacer? ¿Qué consecuencias ocurrirían? 2. Si se desea crear una extensión donde el código nos imprima si el modelo no es viable ¿Qué colocaría en código? 3. Si se desea crear una extensión donde el código nos imprima si él no tiene límite superior o inferior (solución infinita) ¿Qué colocaría en código? 4. Si se desea crear una extensión donde el código nos imprima si el modelo si es factible o no acotado ¿Qué colocaría en código?

5. Si se desea crear una extensión donde el código nos imprima si el modelo no tiene forma de resolverse, pero deseamos conocer el tiempo máximo alcanzado sin resolver ¿Qué colocaría en código?
6. Supongamos que en su situación habrá unos costos fijos, ¿Cómo configuraría el código para que nos de la utilidad bruta (sin costos fijos) y la utilidad neta (después de los costos fijos)?
7. Supongamos que en su situación ya no habrá costos fijos, ¿Cómo configuraría el código para que nos dé una única utilidad?
8. ¿De qué otra forma realiza ese problema con índices?
9. ¿De qué otra forma puede definir el límite superior?
10. ¿De qué otra forma puede definir el límite inferior?
11. Supongamos que no se quiere una repuesta con decimales, ¿Qué haría?
12. Si en vez de maximizar utilidad, quisiera que minimizar costos, ¿Qué debería cambiar del código?
13. ¿Cómo está seguro que sus datos estas siendo tomados correctamente del Excel?
14. ¿El nombre (no la variable) del modelo afecta a la hora de resolver el ejercicio?
15. Supongamos que no quiero que se nombren los productos, si no que quiero que se enumeren, ¿Qué cambios debería hacer del código?
16. Sin una restricción tiene un nombre diferente a la nombrada en el código ¿Se seguirá reconociendo es misma?
17. ¿Cuál es la biblioteca que se necesita para leer las restricciones desde un Excel?
18. Si nombra la hoja de Excel diferente al nombre registrado en el código, ¿Este lo leerá?

19. Si se desea que sus restricciones fueran sumatorias, ¿Qué cambio haría?

20. Si se desea que su función objetivo fueran sumatorias, ¿Qué cambio haría?

5.5.4. Encuesta diagnostica

Se diseña una encuesta de tipo diagnóstica y evaluativa de satisfacción, cuyo propósito es conocer el nivel de comprensión que los estudiantes alcanzan respecto a las actividades desarrolladas y a la temática central del presente proyecto. La encuesta está compuesta por un total de 21 preguntas, de las cuales 3 corresponden a información personal (nombre, apellido y código estudiantil) y 18 son ítems de evaluación formulados con base en una escala tipo Likert de 5 puntos, donde 1 representa “totalmente en desacuerdo” y 5, “totalmente de acuerdo”. Con el fin de garantizar la participación de la totalidad del grupo, se asigna a la diligenciación de la encuesta un porcentaje dentro de la nota final, incentivando así la retroalimentación integral del proceso formativo.

Título de la encuesta: Encuesta de Satisfacción y Aprendizaje Modelos de Optimización con Gurobi

Preguntas de información personal:

1. Nombre
2. Apellidos
3. Código

Se diseñó Preguntas ítems de evaluación:

1. Comprendí claramente la diferencia entre variables, restricciones y función objetivo.
2. Pude identificar los elementos del modelo a partir del enunciado del problema.

3. Entendí cómo traducir los datos del Excel a parámetros dentro del modelo.
4. Comprendí cómo se interpreta el resultado óptimo del modelo.
5. Me sentí cómodo/a cargando datos desde el archivo Excel.
6. Comprendí cómo se construye el modelo en Python utilizando Gurobi.
7. Me fue fácil ejecutar el código y resolver el modelo.
8. Entendí cómo se construyen las restricciones y la función objetivo en el código. utilizar el código y resolver el modelo.
9. Me pareció fácil modificar el modelo para adaptarlo a nuevos escenarios.
10. Siento que puedo plantear y resolver un modelo de optimización por mi cuenta.
11. Considero que esta experiencia me ayudó a aplicar los conceptos teóricos de forma práctica.
12. Me gustaría seguir resolviendo problemas de optimización con este tipo de herramientas.
13. Podría explicar a otra persona cómo se resuelve un problema como los trabajados.
14. El taller o ejercicio fue claro y bien estructurado.
15. El uso de Excel para cargar datos fue útil y práctico.
16. El manual de usuario me ayudó a entender los pasos a seguir.
17. La retroalimentación del código fue útil para comprender los resultados.
18. Estoy satisfecho/a con lo que aprendí en esta actividad.

En términos generales, esta etapa permite concretar el objetivo general del presente proyecto, al consolidar el diseño de actividades de tipo práctico orientadas a facilitar la comprensión de las técnicas de solución de problemas de optimización mediante el uso del

lenguaje de programación Python. Asimismo, se avanza en el cumplimiento de varios objetivos específicos, entre ellos: la propuesta de una serie de ejemplos y ejercicios aplicados (actividades 1 y 2), diseñados para apoyar el desarrollo temático de la asignatura, y la definición de métodos de evaluación para medir el nivel de desempeño alcanzado por los estudiantes en las actividades desarrolladas. Esta evaluación se estructura a través de una rúbrica, un cuestionario de sustentación oral y una encuesta diagnóstica aplicada al cierre del proceso formativo.

5.6. Etapa 6: Implementar las actividades y materiales de estudio diseñados para el aprendizaje de problemas de optimización en Python.

Las actividades propuestas en la etapa anterior se implementan en el marco de la asignatura durante el semestre 2025-1, y el material diseñado se socializa oportunamente con los estudiantes. La primera sesión se desarrolla el 15 de mayo, en el horario habitual de la asignatura de 4:00 a 6:00 de la tarde. En esta clase se explican los conceptos básicos de la optimización lineal y las líneas de código necesarias para resolver ejercicios en Python a partir de datos cargados desde un archivo Excel. Asimismo, se analiza el significado de cada fragmento de código y se introducen ejercicios de mayor complejidad, con el fin de preparar a los estudiantes para la siguiente sesión.

La segunda clase se lleva a cabo el 22 de mayo, en el mismo horario. En esta sesión se presentan ejercicios de mayor complejidad, orientados a la preparación de los estudiantes para el desarrollo del taller práctico y del caso de estudio. En este espacio, además, se comunican las fechas de entrega: el taller práctico con plazo hasta el domingo 1 de junio, y el caso de estudio con presentaciones de justificación programadas para los días 26 y 29 de mayo, en las que los grupos son convocados en horarios específicos dentro del espacio regular de clase.

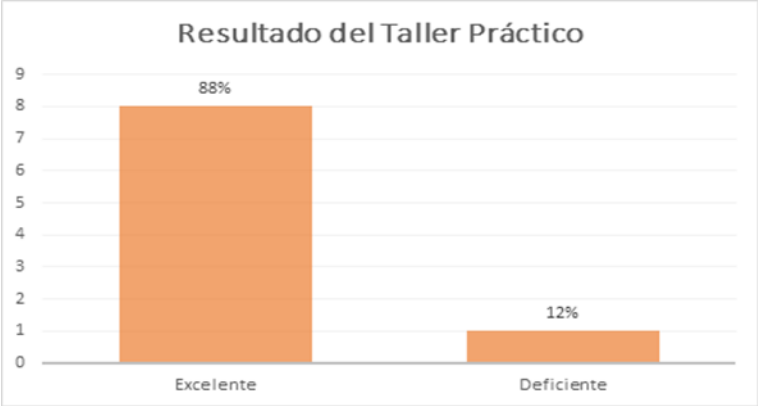
El grupo está conformado por un total de 23 estudiantes. A continuación, se presentan los resultados obtenidos a partir de la ejecución de las estrategias evaluativas, así como los principales hallazgos derivados del desarrollo e implementación de esta propuesta académica.

5.6.1. Resultados de la actividad 1 taller practico

Del total de grupos participantes, 8 de los 9 grupos evaluados alcanzaron un desempeño clasificado como "Excelente" según los criterios establecidos en la rúbrica diseñada para la actividad. Este resultado representa un 88% del total, como se evidencia en la gráfica de resultados del Taller Práctico. Solo un grupo (12%) se ubicó en la categoría de "Deficiente", lo que indica una excepción en cuanto al cumplimiento de los requerimientos mínimos establecidos. Estos resultados permiten afirmar que la Actividad 1 se desarrolló de manera satisfactoria, cumpliendo con los objetivos de aprendizaje planteados. Los estudiantes demostraron dominio en la formulación matemática del modelo, la correcta implementación del código en Python utilizando gurobipy, la carga estructurada de datos desde Excel, y la adecuada presentación e interpretación de los resultados obtenidos.

Figura 4.

Resultado Taller práctico.



La distribución mostrada en la gráfica confirma que la gran mayoría de los grupos logró alcanzar un nivel de excelencia en el desarrollo del taller práctico, lo que evidencia la efectividad del enfoque pedagógico y el diseño de las actividades implementadas. La baja proporción de desempeño deficiente sugiere que los estudiantes comprendieron adecuadamente los conceptos y herramienta trabajadas, destacando un alto nivel de apropiación del contenido abordado.

5.6.2. Resultados de la actividad 2 Caso de Estudio

En la evaluación del caso de estudio, se observó que 7 de los 9 grupos alcanzaron un desempeño clasificado como “Excelente”, lo que representa un 78% del total, mientras que 2 grupos (22%) fueron ubicados en la categoría de “Deficiente”, tal como se muestra en la gráfica de resultados del caso de estudio.

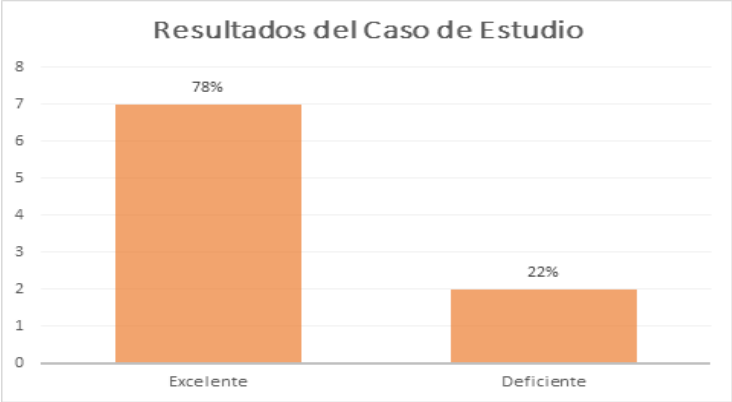
Cabe destacar que esta actividad presentó un mayor nivel de complejidad, ya que se trataba de un caso propuesto desde el inicio de la asignatura. Esto implicó que los estudiantes debían garantizar un desarrollo progresivo y técnicamente riguroso, abordando desde etapas iniciales la

recolección y análisis de datos, la formulación del modelo matemático, su implementación en Python mediante el uso de Gurobi, y finalmente, la exposición técnica del trabajo.

A pesar del grado de dificultad, los resultados evidencian un alto nivel de compromiso y apropiación por parte de la mayoría de los grupos, lo cual refleja la efectividad de la estrategia implementada. No obstante, los casos con desempeño deficiente permiten identificar oportunidades de mejora en el acompañamiento metodológico, especialmente en lo relacionado con la planificación anticipada y la integración continua de los contenidos a lo largo del semestre.

Figura 5.

Resultados Caso de Estudio.



5.6.3. Resultados de la encuesta diagnóstica

Los resultados obtenidos a partir de la encuesta diagnóstica aplicada a los estudiantes fueron, en su totalidad, positivos, lo que evidencia un alto nivel de comprensión y apropiación en relación con las actividades desarrolladas durante la implementación de las actividades establecidas en la etapa 5. Cada una de las preguntas formuladas fue respondida de manera favorable.

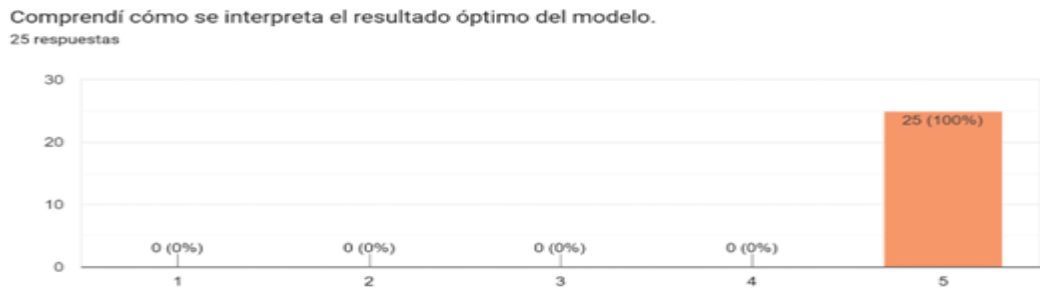
Esta encuesta no solo permitió identificar el grado de asimilación de los contenidos, sino que además se consolidó como una herramienta clave para evaluar la efectividad de la estrategia pedagógica empleada. Los resultados reflejan, a su vez, el compromiso y la participación activa de los estudiantes a lo largo del proceso, lo que refuerza la validez de los métodos utilizados y aporta insumos valiosos para futuras intervenciones educativas en entornos similares.

A continuación, se presentan una serie de gráficos representativos que ilustran visualmente algunas de las respuestas más relevantes obtenidas a través de la encuesta diagnóstica. En cada figura se puede observar tanto el porcentaje como el número de estudiantes que seleccionaron cada una de las opciones de respuesta, lo cual facilita una interpretación más precisa del nivel de aceptación, satisfacción y comprensión alcanzado por los participantes respecto a las actividades implementadas.

El 100% de los estudiantes indicó haber comprendido plenamente la interpretación del resultado óptimo del modelo, lo que evidencia una asimilación clara de este concepto clave.

Figura 6.

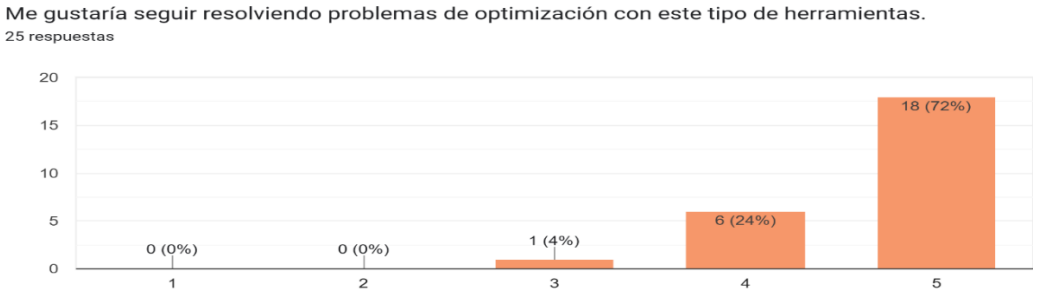
Compresión de los resultados



El 96% de los estudiantes manifestó interés en seguir resolviendo problemas de optimización con este tipo de herramientas, lo que sugiere una alta aceptación del enfoque computacional implementado.

Figura 7.

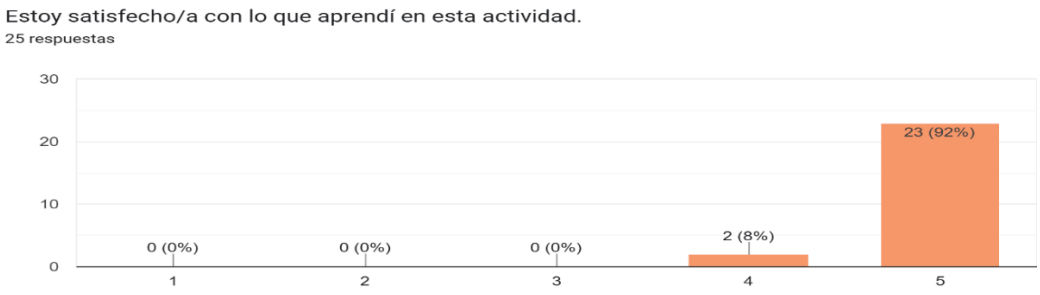
Interés en el modelo en Python.



El 92% de los estudiantes expresó estar completamente satisfecho con lo aprendido en la actividad, lo que refleja un alto nivel de valoración del proceso formativo.

Figura 8.

Nivel de satisfacción de los estudiantes.



6. Resultados

En la implementación de las actividades propuestas se evidencio un impacto positivo en el proceso de Enseñanza-Aprendizaje de la asignatura de Investigación de Operaciones I. El análisis de datos obtenidos a lo largo del semestre, tanto de las evaluaciones prácticas como de los instrumentos de medición complementarios, refleja un alto nivel de apropiación de los contenidos y el desarrollo de competencias técnicas en el uso de Python, biblioteca y solver Gurobi para la formulación y resolución de modelos de optimización.

6.1. Resultados del taller práctico

El taller práctico constituyó la primera actividad de evaluación grupal, en la cual los estudiantes aplicaron los conocimientos adquiridos en la construcción y resolución de modelos de programación lineal utilizando Python y la biblioteca Gurobi. El desempeño de los grupos fue valorado mediante la rúbrica diseñada, que contemplaba tanto la formulación matemática como la implementación computacional y la interpretación de resultados.

Tabla 3.

Calificaciones del taller práctico por grupo

Grupos	NOTA DEL TALLER:
Grupo #1	
David Nova Salazar	**5,0**
Jaime Alberto Peñuela Duarte	
Yessica Andreina Teheran Perez	
Grupo #2	
Laura Valeria Cortés González	**5,0**
Astrid Tatiana Barrera Rincón	
Gira Daniela López Deantonio	
Grupo #3	

Carlos Daniel Naranjo Rivera	**5,0**
Jesús Enrique Beltrán Méndez	
Yerly Juliana Camacho Suarez	
Grupo #4	
Sehir Julyam Guzman Rincon	**1,5**
Grupo #5	
Andres Orlando Gonzalez Prada	**5,0**
Óscar Andrés Lejarde Cifuetes	
Grupo #6	
Karen Sofia Acevedo	**5,0**
Karol Daniela Aguilar	
Yorjelys Borja	
Grupo #7	
Antonio Jose Suarez	**4,8**
Karen Sofia Ramirez	
Grupo #8	
Sary Villamizar	**5,0**
Juliana Arevalo	
Luisa Fernanda Mantilla	
Grupo #9	
Valentina Gonzalez	**5,0**
Mateo Vargas	
Juan David Aceros	

Los resultados evidencian que la mayoría de los grupos alcanzaron un nivel de desempeño calificado como “Excelente” según la rúbrica de evaluación (Tabla 1.), lo que confirma una adecuada apropiación de los conceptos básicos y una correcta aplicación de los mismos en un entorno de programación real. Este hallazgo permite concluir que la actividad cumplió su propósito formativo, brindando a los estudiantes las herramientas necesarias para enfrentar posteriormente el caso de estudio.

6.2. Resultados del caso de estudio

El caso de estudio, orientado al diseño de un modelo de negocio tipo pizzería, representó la actividad de mayor complejidad del semestre. Los grupos debieron investigar información real, estructurar un modelo matemático de programación lineal y resolverlo mediante la implementación en Python con Gurobi. Además de la correcta formulación y solución, la actividad exigió la presentación oral y la justificación de los resultados.

Tabla 4.

Calificaciones del caso de estudio por grupo.

Grupos	Notas Caso
Grupo #1	
David Nova Salazar	4,8
Jaime Alberto Peñuela Duarte	
Yessica Andreina Teheran Perez	
Grupo #2	
Laura Valeria Cortés González	5,0
Astrid Tatiana Barrera Rincón	
Gira Daniela López Deantonio	
Grupo #3	
Carlos Daniel Naranjo Rivera	5,0
Jesús Enrique Beltrán Méndez	
Yerly Juliana Camacho Suarez	
Grupo #4	
Sehir Julyam Guzman Rincon	1,0
Grupo #5	
Andres Orlando Gonzalez Prada	4,3
Óscar Andrés Lejarde Cifuetes	
Grupo #6	
Karen Sofia Acevedo	5,0
Karol Daniela Aguilar	
Yorjelys Borja	
Grupo #7	

Antonio Jose Suarez	1,0
Karen Sofia Ramirez	
Grupo #8	
Sary Villamizar	5,0
Juliana Arevalo	
Luisa Fernanda Mantilla	
Grupo #9	
Valentina Gonzalez	5,0
Mateo Vargas	
Juan David Aceros	

Los resultados muestran un desempeño positivo en la mayoría de los grupos, aunque se observa una mayor dispersión en comparación con el taller práctico. Esto se explica por la complejidad propia de la actividad, que exigía integrar investigación, modelación matemática, implementación computacional y habilidades comunicativas. Aun así, se alcanzó un nivel general satisfactorio, evidenciando que los estudiantes lograron transferir lo aprendido a un contexto más amplio y retador.

6.3. Resultados del caso de estudio

Las calificaciones finales consolidan el desempeño integral de los estudiantes en la temática trabajada, al obtenerse a partir del promedio entre las notas del taller práctico y el caso de estudio. De esta manera, se logra una valoración equilibrada que refleja tanto el dominio de los fundamentos básicos como la capacidad de aplicar los conocimientos en un contexto más complejo.

Tabla 5.

Calificaciones finales por grupo.

Grupos	NOTA FINAL
Grupo #1	
David Nova Salazar	4,9
Jaime Alberto Peñuela Duarte	
Yessica Andreina Teheran Perez	
Grupo #2	
Laura Valeria Cortés González	4,9
Astrid Tatiana Barrera Rincón	
Gira Daniela López Deantonio	
Grupo #3	
Carlos Daniel Naranjo Rivera	4,9
Jesús Enrique Beltrán Méndez	
Yerly Juliana Camacho Suarez	
Grupo #4	
Sehir Julyam Guzman Rincon	1,3
Grupo #5	
Andres Orlando Gonzalez Prada	4,7
Óscar Andrés Lejarde Cifuetes	
Grupo #6	
Karen Sofia Acevedo	5,0
Karol Daniela Aguilar	
Yorjelys Borja	
Grupo #7	
Antonio Jose Suarez	2,9
Karen Sofia Ramirez	
Grupo #8	
Sary Villamizar	5,0
Juliana Arevalo	
Luisa Fernanda Mantilla	
Grupo #9	
Valentina Gonzalez	5,0
Mateo Vargas	
Juan David Aceros	

En términos generales, los resultados muestran una clara mayoría de calificaciones altas, lo cual valida la estrategia implementada. Los estudiantes no solo alcanzan los objetivos de la asignatura, sino que también fortalecen competencias en la formulación matemática, la implementación computacional y el uso de herramientas aplicables en el ámbito profesional.

6.4. Encuesta diagnóstica y análisis global

La encuesta diagnóstica aplicada al finalizar el semestre tuvo como propósito identificar la percepción de los estudiantes respecto a las actividades implementadas y valorar el nivel de apropiación de los contenidos. Los resultados reflejaron una valoración positiva en la totalidad de los ítems, con altos niveles de satisfacción frente al uso de Python y Gurobi, así como al material de apoyo entregado.

De manera global, los hallazgos obtenidos en las distintas actividades permiten concluir que la propuesta pedagógica logra fortalecer la comprensión de los modelos de optimización, fomentar el trabajo colaborativo y desarrollar competencias tecnológicas en los estudiantes.

7. Conclusiones

El desarrollo de este proyecto permitió comprobar la viabilidad y la efectividad de implementar actividades practicas apoyadas por herramientas computacionales específicamente el lenguaje de programación de Python y la biblioteca y solver de Gurobi, para la enseñanza de técnicas de optimización en la asignatura de Investigación de Operaciones I. La propuesta logro integrar la teoría y la práctica de manera más estructurada, favoreciendo un aprendizaje significativo y la aplicación de los conceptos a problemas reales de optimización.

Los resultados obtenidos en las evaluaciones evidenciaron un alto nivel de compresión y apropiación de los contenidos por parte de los estudiantes. En el taller práctico, la mayoría de los equipos de los grupos alcanzo la categoría de “Excelente”, demostrando dominio en la formulación de modelos, implementación en Python, la carga de datos desde Excel y la interpretación de resultados. Aunque en el caso de estudio se observó una ligera disminución en el porcentaje de grupos en la categoría más alta, puesto que esta actividad contenía mayor dificultad y la exigencia de mantener coherencia y el rigor técnico durante un periodo prologado de trabajo.

La encuesta diagnostica y de satisfacción ce los estudiantes confirmó la aceptación y el impacto positivo de la metodología implementada. Los estudiantes manifestaron un alto interés en seguir utilizando estas herramientas y expresaron su satisfacción con el aprendizaje alcanzado, lo que respalda la pertenencia de incorporar este tipo de estrategias pedagógicas en el plan de estudios. Estos indicadores, junto con las calificaciones finales, validan el cumplimiento de los objetivos planteados y resalta la importancia del componente practico aplicado en la enseñanza de optimización lineal.

Asimismo, la propuesta demostró ser adaptable y replicable en otros contextos académicos, siempre y cuando se cuente con una planificación cuidadosa, un acompañamiento docente

constante y una evaluación integral que incluya tanto la valoración de resultados como la percepción de los estudiantes. La experiencia adquirida en este trabajo proporciona insumos valiosos para futuras implementaciones, orientadas a perfeccionar los materiales, optimizar el uso de recursos tecnológicos y reforzar el seguimiento de proyectos de larga duración.

En conclusión, este proyecto no solo cumplió con si objetivo principal de diseñar e implementar actividades prácticas para la enseñanza de optimación lineal, sino que también contribuyo al fortalecimiento de competencias técnicas, analíticas y de trabajo en equipo, potenciando así la formación integral de los estudiantes y ofreciendo un modelo de Enseñanza-Aprendizaje innovador y efectivo.

8. Recomendaciones

Con base en los resultados obtenidos y en la experiencia desarrollada durante la implementación del proyecto, se plantean las siguientes recomendaciones orientadas a fortalecer futuras aplicaciones y garantizar la continuidad de este tipo de iniciativas:

1. Fortalecer el acompañamiento docente

Se recomienda que los docentes de la asignatura Investigación de Operaciones I reciban capacitaciones previas en el manejo de Python y Gurobi, con el fin de dar apoyo técnico y teórico efectivo durante la formulación y codificación de los modelos, contribuyendo a que los estudiantes desarrollen las competencias requeridas para enfrentar problemas de optimización con este enfoque profesional de manea sólida.

2. Ampliar la complejidad de los ejercicios

Es muy importante que en futuros grupos de estudiantes se incorporen casos y talleres con mayor cantidad de variables, restricciones, parámetros, etc. Integrando problemas reales en los sectores de logística, producción o finanzas. Esta estrategia esta alineada con el perfil del programa de Ingeniería Industrial de la Universidad Industrial de Santander, que forma profesionales con capacidades de diseñar y controlar procesos complejos bajo criterios de productividad y competitividad.

3. Promover la integración con otras asignaturas

Se sugiere vincular las actividades diseñadas con cursos afines del plan de estudio, tales como Logística Integral, Investigación de Operaciones II y Finanzas y Presupuesto. Esta

integración favorece la formación integral y refuerza el enfoque sistemático que caracteriza la ingeniería Industrial, permitiendo que los estudiantes puedan aplicar modelos de optimización en contexto diversos y relevantes para su formación personal.

4. Actualizar y mantener los materiales de apoyo

Los recursos elaborados como el Manual de Usuario, la lista de ejercicios de repaso y básicos y el video explicativo deben ser actualizados periódicamente dependiendo de las tendencias actuales en optimización y el uso de TIC. Este proceso de actualización continua garantiza que los estudiantes desarrollen competencias acordes con las exigencias del entorno profesional, contribuyendo a mantener siempre vigente y actualizado el perfil del Ingeniero Industrial.

5. Implementar un sistema de evaluación continua

Además de la rúbrica, el taller práctico y la sustentación oral, se recomienda incorporar instrumentos de evaluación continua que permitan monitorear el progreso individual y grupal a lo largo del semestre. Esto favorecería una retroalimentación constante y un mejor seguimiento de las dificultades detectadas.

6. Fomentar la investigación aplicada

Se sugiere que los casos de estudio puedan estar asociados a problemáticas reales de empresas de la región, de manera que los estudiantes no solo apliquen conocimientos teóricos y técnicos, sino que también contribuyan a la solución de necesidades concretas del entorno productivo.

Finalizando, estas recomendaciones buscan consolidar la asignatura de Investigación de Operaciones I como una experiencia académica innovadora dentro del programa de Ingeniería Industrial de la Universidad Industrial de Santander, respondiendo a si misión institucional de formar profesionales capaces de mejorar la productividad mediante el análisis y optimización de proceso. La incorporación de estrategias actualizadas fortalecerá su coherencia con las exigencias del entorno profesional y educativo.

Referencia Bibliográfica

Anderson, D. R., Sweeney, D. J., Williams, T. A., Camm, J. D., & Martin, K. W. (2011). *Métodos cuantitativos para los negocios* (11ª ed.). Cengage Learning.

Bello Durand, J. B. (2013). *Mediación del software GeoGebra en el aprendizaje de programación lineal en alumnos del quinto grado de educación secundaria*.

Bonito Gunsha, J. D. (2021). *Software TORA en el proceso de enseñanza-aprendizaje de programación lineal en los estudiantes de tercer semestre de la carrera de pedagogía de las ciencias experimentales matemáticas y física en el periodo noviembre 2020-abril 2021*.

Hillier, F. S., & Lieberman, G. J. (2015). *Introducción a la investigación de operaciones* (9ª ed.). McGraw-Hill.

Peñaloza Y Arámbulo, M. (2024). *El software PHPSimplex y la capacidad de aprendizaje de la programación lineal en estudiantes universitarios*.

Taha, H. A. (2012). *Investigación de operaciones* (9ª ed.). Pearson Educación.

Universidad Industrial de Santander. (2017). *PROPUESTA DE MODIFICACIÓN PLAN DE ESTUDIOS DEL PROGRAMA DE INGENIERÍA INDUSTRIAL*.

Anexos

1. Ejercicios Básicos

```
!pip install gurobipy
from gurobipy import Model, GRB
import pandas as pd
```

1.1. Una empresa produce dos productos: A y B. Cada unidad de A requiere 2 horas de trabajo y 3 kg de material. Cada unidad de B requiere 4 horas de trabajo y 2 kg de material. La empresa tiene disponibles 100 horas de trabajo y 90 kg de material. Las ganancias son de 30 USD por cada unidad de A y 50 USD por cada unidad de B. ¿Cuántas unidades de A y B debe producir para maximizar la ganancia?

```
# Crear modelo
model = Model("Producción_simple")
# Variables
A = model.addVar(name="A", lb=0)
B = model.addVar(name="B", lb=0)
# Función objetivo: maximizar la ganancia
model.setObjective(30 * A + 50 * B, GRB.MAXIMIZE)
# Restricciones
model.addConstr(2 * A + 4 * B <= 100, name="Trabajo")
model.addConstr(3 * A + 2 * B <= 90, name="Material")
# Resolver
model.optimize()
# Resultados
if model.status == GRB.OPTIMAL:
```

```
        print(f"\nUnidades a producir:")
        print(f"A: {A.x}")
        print(f"B: {B.x}")
        print(f"Ganancia total: ${model.objVal:.2f}")
# Precios sombra
print("\nPrecios sombra (dual):")
for c in model.getConstrs():
    print(f"{c.ConstrName}: {c.pi:.2f}")
# Intervalos de sensibilidad
print("\nIntervalos de sensibilidad (obj):")
for v in model.getVars():
    print(f"{v.VarName}: [{v.SAObjLow}, {v.SAObjUp}]")
```

1.2. Supón que producir una unidad de A cuesta 30 USD y una de B cuesta 50 USD. Necesitamos producir al menos 10 unidades de A y al menos 5 unidades de B, cumpliendo además con las limitaciones de recursos disponibles:

- 100 horas de trabajo
- 90 kg de material

Cada unidad de A requiere 2 h de trabajo y 3 kg de material. Cada unidad de B requiere 4 h de trabajo y 2 kg de material. ¿Cuántas unidades de cada producto debe producir la empresa para cubrir la demanda al menor costo posible?

```
# Crear modelo
model = Model("Minimización_sensibilidad")
# Variables
A = model.addVar(name="A", lb=0)
B = model.addVar(name="B", lb=0)
# Objetivo
```

```
model.setObjective(30 * A + 50 * B, GRB.MINIMIZE)
# Restricciones
model.addConstr(2 * A + 4 * B <= 100, name="Trabajo")
model.addConstr(3 * A + 2 * B <= 90, name="Material")
model.addConstr(A >= 10, name="Demanda_A")
model.addConstr(B >= 5, name="Demanda_B")
# Resolver
model.optimize()
# Resultados
if model.status == GRB.OPTIMAL:
    print(f"\nUnidades a producir:")
    print(f"A: {A.x}")
    print(f"B: {B.x}")
    print(f"Costo total: ${model.objVal:.2f}")
    # Precios sombra
    print("\nPrecios sombra:")
    for c in model.getConstrs():
        print(f"{c.ConstrName}: {c.pi:.2f}")
    # Intervalos de sensibilidad
    print("\nIntervalos de sensibilidad (costos):")
    for v in model.getVars():
        print(f"{v.VarName}: [{v.SAObjLow}, {v.SAObjUp}]")
```

1.3. Una empresa fabrica 3 productos: P1, P2 y P3. Cada uno genera una utilidad diferente y consume cierta cantidad de recursos (horas de máquina y kg de material). Se tienen limitaciones en la disponibilidad de estos recursos. ¿Cuántas unidades de cada producto deben producirse para maximizar las utilidades totales?

```
# Asegúrese de subir el archivo 'datos.xlsx' en tu entorno de Colab
archivo = 'datos.xlsx'
productos_df = pd.read_excel(archivo, sheet_name="Productos")
recursos_df = pd.read_excel(archivo, sheet_name="Recursos")
consumo_df = pd.read_excel(archivo, sheet_name="Consumo")
productos = list(productos_df['Producto'])
recursos = list(recursos_df['Recurso'])
utilidad = dict(zip(productos_df['Producto'], productos_df['Utilidad']))
disponibilidad = dict(zip(recursos_df['Recurso'], recursos_df['Disponibilidad']))
consumo = {
   r: dict(zip(consumo_df['Producto'], consumo_df[r])) for r in recursos
}
model = Model("Maximizar_Utilidades")
X = model.addVars(productos, name="Producción", lb=0)
model.setObjective(sum(utilidad[p] * X[p] for p in productos), GRB.MAXIMIZE)
for r in recursos:
   model.addConstr(sum(consumo[r][p] * X[p] for p in productos) <= disponibilidad[r],
name=f"Restr_{r}")
model.optimize()
if model.status == GRB.OPTIMAL:
   print("\nProducción óptima:")
   for p in productos:
      print(f"{p}: {X[p].x:.2f} unidades")
   print(f"\nUtilidad total: ${model.objVal:.2f}")
else:
   print("No se encontró solución óptima.")
```

2. Ejercicios De Repaso

2.1. Logística de Distribución en ElectroNova S.A.

ElectroNova S.A. es una empresa colombiana que fabrica equipos electrónicos en sus dos plantas industriales: una ubicada en Medellín y otra en Bucaramanga. Estos productos deben distribuirse diariamente a tres centros logísticos ubicados en Bogotá, Cali y Barranquilla, desde donde serán entregados a clientes finales.

Para reducir costos, la dirección logística de ElectroNova desea diseñar un plan de transporte que minimice los gastos totales de envío, garantizando que cada centro reciba exactamente la cantidad requerida y que no se exceda la capacidad diaria de envío de cada planta.

Plantas	Ciudad	Capacidad diaria
Planta 1	Medellin	300 und
Planta 2	Bucaramanga	500und

- Demanda en centro de distribución:

Centro de Distribución #	Ciudad	Demanda diaria
CD1	Bogotá	250 und
CD2	Cali	300 und
CD3	Barranquilla	250d

- Costos de transporte (en miles de pesos COP por unidad):

Desde / Hacia	CD1 (Bogotá)	CD2 (Cali)	CD3 (Barranquilla)
Planta 1	18	22	30
Planta 2	25	20	16

Objetivo: ElectroNova desea minimizar el costo total de transporte, asegurando que:

- Cada centro de distribución reciba su demanda completa.
- Las plantas no envíen más unidades de las que pueden producir por día.

```
# Crear el modelo
model_electro = Model("Ejercicio_Logistica_ElectroNova")
# Conjuntos
plantas = ['Planta1', 'Planta2']
centros = ['CD1', 'CD2', 'CD3']
# Parametros
Oferta = [300, 500]
Demanda = [250, 300, 250]
Costo = [[18, 22, 30], [25, 20, 16]]
# Varaibles
x = model_electro.addVars(plantas, centros, vtype=GRB.CONTINUOUS, name="x", lb=0)
# Función objetivo
```

```
model_electro.setObjective(sum(Costo[i][j] * x[plantas[i], centros[j]] for i in range(2) for j in range(3)), GRB.MINIMIZE)
# Restricciones de demanda
for j in centros:
    model_electro.addConstr(sum(x[i,j] for i in plantas) == Demanda[centros.index(j)], f"Demanda_{j}")
# Restricciones de oferta
for i in plantas:
    model_electro.addConstr(sum(x[i,j] for j in centros) <= Oferta[plantas.index(i)], f"Oferta_{i}")
# Resolver
model_electro.optimize()
```

Respuesta del código

```
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 5 rows, 6 columns and 12 nonzeros
Model fingerprint: 0xba61172f
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+01, 3e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 5e+02]
Presolve removed 5 rows and 6 columns
Presolve time: 0.01s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
```

```
    0    1.4600000e+04   0.000000e+00   0.000000e+00      0s
Solved in 0 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.460000000e+04
```

2.2. Producción óptima de jugos naturales

Una empresa produce 3 tipos de jugos naturales: naranja (i=1), mango (i=2) y fresa (i=3). Cada jugo requiere una cantidad específica de fruta natural, agua purificada y botellas, todos recursos limitados. Cada tipo de jugo aporta una utilidad por litro vendida.

Jugo	Utilidad	Fruta	Agua	Botellas
Naranja	1.20	0.5	0.4	1
Mango	1.50	0.6	0.3	1
Fresa	1.80	0.8	0.5	1

Recursos disponibles:

- Fruta natural: 500 kg
- Agua purificada: 300 litros
- Botellas disponibles: 800 unidades
- Cada tipo de jugo debe producirse en al menos 100 litros y como máximo 300 litros.

Objetivo: Determinar cuántos litros de cada jugo producir para maximizar la utilidad total, respetando los recursos y los límites de producción.

```
# Crear el modelo
model_jugo = Model("Ejercicio_Produccion_Jugo")
# Parametros
```

```
jugos = ['Naranja', 'Mango', 'Fresa']
utilidad = [1.20, 1.50, 1.80]
fruta = [0.5, 0.6, 0.8]
agua = [0.4, 0.3, 0.5]
botellas = [1, 1, 1]
min_prod = [100, 100, 100]
max_prod = [300, 300, 300]
# Limites
lim_fruta = 500
lim_agua = 300
lim_botellas = 800
# Variables de decisión
x = model_jugo.addVars(3, vtype=GRB.CONTINUOUS, name="x", lb=min_prod,
ub=max_prod)
# Función objetivo
model_jugo.setObjective(sum(utilidad[i] * x[i] for i in range(3)), GRB.MAXIMIZE)
# Restricciones
model_jugo.addConstr(sum(fruta[i] * x[i] for i in range(3)) <= lim_fruta, "Fruta")
model_jugo.addConstr(sum(agua[i] * x[i] for i in range(3)) <= lim_agua, "Agua")
model_jugo.addConstr(sum(botellas[i] * x[i] for i in range(3)) <= lim_botellas, "Botellas")
# Resolver
model_jugo.optimize()
```

Respuesta del código

```
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 3 rows, 3 columns and 9 nonzeros
Model fingerprint: 0xb53c733a
```

```
Coefficient statistics:
  Matrix range     [3e-01, 1e+00]
  Objective range  [1e+00, 2e+00]
  Bounds range     [1e+02, 3e+02]
  RHS range        [3e+02, 8e+02]
Presolve time: 0.01s
Presolved: 3 rows, 3 columns, 9 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.2250000e+03   8.125000e+01   0.000000e+00      0s
       3    1.1700000e+03   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.02 seconds (0.00 work units)
Optimal objective  1.170000000e+03
```

```
# Resultados
if model_jugo.status == GRB.OPTIMAL:
    print("--- Solución óptima ---")
    for i in range(3):
        print(f"Litros de jugo {jugos[i]}: {x[i].X:.2f}")
    print(f"Utilidad total: ${model_jugo.ObjVal:.2f}")
else:
    print("No se encontró solución óptima.")
```

resultado del código:

```
--- Solución óptima ---
Litros de jugo Naranja: 150.00
Litros de jugo Mango: 300.00
Litros de jugo Fresa: 300.00
Utilidad total: $1170.00
```

2.3. Mezcla de suplementos alimenticios

Una empresa elabora 4 tipos de suplementos en polvo que contienen diferentes proporciones de tres ingredientes esenciales: proteína vegetal, fibra y antioxidantes. Cada suplemento se vende en bolsas de 1 kg y tiene una utilidad por unidad. Todos los productos se fabrican en la misma planta y comparten recursos.

Ingredientes disponibles:

- Proteína vegetal: 600 kg
- Fibra: 400 kg
- Antioxidantes: 300 kg
- Presupuesto máximo: $1200

Información de productos:

Productos	Utilidad	Proteína	Fibra	Antioxidantes	Costos por unidad	Producción mínima	Producción máxima
Energy Fit	4.5	0.6	0.2	0.1	2.5	50	200
Fiber Plus	3.2	0.2	0.5	0.1	2.0	40	150
Antiox Defense	5.0	0.3	0.1	0.4	3.0	30	100

Protein Boost	4.0	0.7	0.1	0.1	2.8	60	180

Objetivo: Determinar cuántas unidades producir de cada suplemento para maximizar la utilidad total, cumpliendo con las restricciones de disponibilidad de ingredientes y presupuesto.

```
# Crear modelo
model_suplementos = Model("Ejercicio_Mezcla_Suplementos")
# Parametros
Productos = ['Energy Fit', 'Fiber Plus', 'Antiox Defense', 'Protein Boost']
utilidad = [4.5, 3.2, 5.0, 4.0]
proteina = [0.6, 0.2, 0.3, 0.7]
fibra = [0.2, 0.5, 0.1, 0.1]
antiox = [0.1, 0.1, 0.4, 0.1]
costo = [2.5, 2.0, 3.0, 2.8]
min_prod = [50, 40, 30, 60]
max_prod = [200, 150, 100, 180]
# Limite
max_proteina = 600
max_fibra = 400
max_antiox = 300
max_presupuesto = 1200
# Variables de decisión
x = model_suplementos.addVars(Productos, vtype=GRB.CONTINUOUS, name="x",
lb=min_prod, ub=max_prod)
# Función objetivo
model_suplementos.setObjective(sum(utilidad[i] * x[Productos[i]] for i in range(4)),
GRB.MAXIMIZE
# Restricciones
```

```
model_suplementos.addConstr(sum(proteina[i] * x[Productos[i]] for i in range(4)) <=
max_proteina, "Proteina")
model_suplementos.addConstr(sum(fibra[i] * x[Productos[i]] for i in range(4)) <= max_fibra,
"Fibra")
model_suplementos.addConstr(sum(antiox[i] * x[Productos[i]] for i in range(4)) <= max_antiox,
"Antiox")
model_suplementos.addConstr(sum(costo[i] * x[Productos[i]] for i in range(4)) <=
max_presupuesto, "Presupuesto")
# Resolver
model_suplementos.optimize()
```

resultado del código:

```
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 4 rows, 4 columns and 16 nonzeros
Model fingerprint: 0xcd4be140
Coefficient statistics:
  Matrix range     [1e-01, 3e+00]
  Objective range  [3e+00, 5e+00]
  Bounds range     [3e+01, 2e+02]
  RHS range        [3e+02, 1e+03]
Presolve removed 3 rows and 0 columns
Presolve time: 0.01s
Presolved: 1 rows, 4 columns, 4 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.0696000e+03   3.620000e+01   0.000000e+00      0s
       1    2.0112000e+03   0.000000e+00   0.000000e+00      0s
Solved in 1 iterations and 0.02 seconds (0.00 work units)
Optimal objective  2.011200000e+03
# Resultados
```

```
if model_suplementos.status == GRB.OPTIMAL:
    print("--- Solución óptima ---")
    for i in range(4):
        print(f"Unidades de {Productos[i]}: {x[Productos[i]].X:.2f}")
    print(f"Utilidad total: ${model_suplementos.ObjVal:.2f}")
else:
    print("No se encontró solución óptima.")
```

Resultado del código:

```
--- Solución óptima ---
Unidades de Energy Fit: 200.00
Unidades de Fiber Plus: 116.00
Unidades de Antiox Defense: 100.00
Unidades de Protein Boost: 60.00

Utilidad total: $2011.20
```

2.4. Problema de producción de mezcla de jugos naturales HIT

La empresa productora de jugos naturales HIT ubicada en Bucaramanga desea planificar su producción diaria para maximizar sus ingresos. La empresa produce 10 tipos de jugos, identificados como $i \in \{ 1 , 2 , \ldots , 10 \}$. Cada jugo requiere dos tipos de insumos, fruta y agua purificada, cuyas disponibilidades diarias máximas son 500 litros para la fruta y 800 litros para el agua purificada. La cantidad de insumos necesarios para producir una unidad de cada jugo se presenta en la siguiente tabla.

Productos	Agua Purificada (L)	**Concentrado de Fruta (L)**	**Precio de Venta ($)**
Naranja	5	6	4000
Mango	4	7	4500
Fresa	6	5	4200

Mora	3	8	4300
Piña	4	6	4100
Guanábana	5	5	4400
Maracuyá	6	4	4600
Limón	4	7	4700
Manzana	5	6	4800
Durazno	3	8	4900

La empresa ha establecido que la cantidad máxima de unidades de cualquier jugo que se puede producir por día es 50 unidades. ¿Cuántas unidades de cada jugo debe producir la empresa para maximizar el ingreso total?

```
# Crear el modelo
model_hit = Model("Ejercicio_Mezcla_Productos_Hit")
# Parámetros
I = list(range(1, 10))
jugos = ['Naranja', 'Mango', 'Fresa', 'Mora', 'Piña', 'Guanábana', 'Maracuyá', 'Limón', 'Manzana', 'Durazno']
agua = [5, 4, 6, 3, 4, 5, 6, 4, 5, 3]
fruta = [6, 7, 5, 8, 6, 5, 4, 7, 6, 8]
precio = [4000, 4500, 4200, 4300, 4100, 4400, 4600, 4700, 4800, 4900]
# Limites
max_fruta = 500
max_agua = 800
max_unidades = 50
# Variables de decisión
x = model_hit.addVars(jugos, vtype=GRB.INTEGER, name="x", lb=0, ub=max_unidades)
```

```
# Función objetivo
model_hit.setObjective(sum(precio[i] * x[jugos[i]] for i in I), GRB.MAXIMIZE)
# Restricciones
model_hit.addConstr(sum(agua[i] * x[jugos[i]] for i in I) <= max_agua, "Agua")
model_hit.addConstr(sum(fruta[i] * x[jugos[i]] for i in I) <= max_fruta, "Fruta")
# Resolver
model_hit.optimize()
urobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 2 rows, 10 columns and 18 nonzeros
Model fingerprint: 0x5e673bee
Variable types: 0 continuous, 10 integer (0 binary)
Coefficient statistics:
  Matrix range     [3e+00, 8e+00]
  Objective range  [4e+03, 5e+03]
  Bounds range     [5e+01, 5e+01]
  RHS range        [5e+02, 8e+02]
Found heuristic solution: objective 351000.00000
Presolve removed 2 rows and 10 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Explored 0 nodes (0 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 1 (of 2 available processors)
Solution count 2: 492000 351000
Optimal solution found (tolerance 1.00e-04)
Best objective 4.920000000000e+05, best bound 4.920000000000e+05, gap 0.0000%
# Resultados
if model_hit.status == GRB.OPTIMAL:
  print("--- Producción óptima de jugos ---")
```

```
  for j in jugos:
    if x[j].X> 0:
        print(f"jugo{j}: {x[j].X} unidades")
  print(f"\nIngreso total maximo: ${model_hit.ObjVal:,.1f}")
else:
    print("No se encontró solución óptima.")
--- Producción óptima de jugos ---
jugoFresa: 10.0 unidades
jugoGuanábana: 50.0 unidades
jugoMaracuyá: 50.0 unidades

Ingreso total maximo: $492,000.0
```

2.5. Producción de jugos de Valle

La empresa productora de jugos naturales Valle elabora diferentes tipos de jugos utilizando tres ingredientes principales: fruta, agua y edulcorante. La empresa tiene tres plantas de producción ubicadas en diferentes regiones de Colombia y desea optimizar la cantidad de cada jugo producido en cada planta para maximizar sus ingresos.

Cada planta (p) puede producir diferentes tipos de jugos (j), pero tiene limitaciones en la disponibilidad de ingredientes y en la capacidad de producción. La cantidad de cada ingrediente requerida por litro de jugo varía según el tipo de jugo. La disponibilidad de ingredientes en cada planta también está limitada por día.

La tabla siguiente muestra la cantidad de ingredientes necesarios por litro de cada jugo y la disponibilidad máxima en cada planta:

Ingrediente	Jugo 1	Jugo 2	Jugo 3	Disponibilidad planta 1	Disponibilidad planta 2	Disponibilidad planta 3

Fruta (Kg)	2	3	1	100	150	120
Agua (L)	1	2	2	80	100	90
Edulcorante (g)	50	30	40	10.000	12.000	9.000

Cada planta tiene una capacidad máxima de producción diaria en litros:

- Planta 1: 50 litros
- Planta 2: 70 litros
- Planta 3: 60 litros

El precio de venta por litro de jugo es:

- Jugo 1: 5000
- Jugo 2: 6000
- Jugo 3: 5500

¿Cuántos litros de cada jugo debe producir cada planta para maximizar los ingresos, respetando las restricciones de disponibilidad de ingredientes y capacidad de producción?

```
# Crear el modelo
model_valle = Model("Ejercicio_Mezcla_Productos_Valle")
# Índices
plantas = [1, 2, 3]
jugos = [1, 2, 3]
# Datos
precio = {1: 5000, 2: 6000, 3: 5500}
fruta = {1: 2, 2: 3, 3: 1}
```

```
agua = {1: 1, 2: 2, 3: 2}
edulcorante = {1: 50, 2: 30, 3: 40}
# Disponibilidades por planta
disp_fruta = {1: 100, 2: 150, 3: 120}
disp_agua = {1: 80, 2: 100, 3: 90}
disp_edulcorante = {1: 10000, 2: 12000, 3: 9000}
capacidad = {1: 50, 2: 70, 3: 60}
# Variables de decisión
x = model_valle.addVars(plantas, jugos, vtype=GRB.CONTINUOUS, name="x", lb=0)
# Función objetivo
model_valle.setObjective(sum(precio[j] * x[i,j] for i in plantas for j in jugos), GRB.MAXIMIZE)
# # Restricciones por planta
for i in plantas:
    model_valle.addConstr(sum(fruta[j] * x[i,j] for j in jugos) <= disp_fruta[i], f"Fruta_{i}")
    model_valle.addConstr(sum(agua[j] * x[i,j] for j in jugos) <= disp_agua[i], f"Agua_{i}")
    model_valle.addConstr(sum(edulcorante[j] * x[i,j] for j in jugos) <= disp_edulcorante[i],
f"Edulcorante_{i}")
    model_valle.addConstr(sum(x[i,j] for j in jugos) <= capacidad[i], f"Capacidad_{i}")
# Resolver
model_valle.optimize()
Restricted license - for non-production use only - expires 2026-11-23
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 12 rows, 9 columns and 36 nonzeros
Model fingerprint: 0xf9cf24e7
Coefficient statistics:
  Matrix range     [1e+00, 5e+01]
  Objective range  [5e+03, 6e+03]
  Bounds range     [0e+00, 0e+00]
```

```
  RHS range        [5e+01, 1e+04]
Presolve removed 3 rows and 0 columns
Presolve time: 0.01s
Presolved: 9 rows, 9 columns, 27 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.0350000e+06   1.649655e+02   0.000000e+00      0s
      11    9.7000000e+05   0.000000e+00   0.000000e+00      0s
Solved in 11 iterations and 0.01 seconds (0.00 work units)
Optimal objective  9.700000000e+05
# Resultados
if model_valle.status == GRB.OPTIMAL:
    print("\n--- Producción óptima por planta y jugo ---")
    for i in plantas:
        for j in jugos:
            litros = x[i,j].X
            if litros > 0:
                print(f"Planta {i}, Jugo {j}: {litros:.2f} litros")
    print(f"\nIngresos máximos: ${model_valle.ObjVal:,.2f}")
else:
    print("No se encontró solución óptima.")
--- Producción óptima por planta y jugo ---
Planta 1, Jugo 1: 20.00 litros
Planta 1, Jugo 2: 15.00 litros
Planta 1, Jugo 3: 15.00 litros
Planta 2, Jugo 1: 40.00 litros
Planta 2, Jugo 2: 20.00 litros
Planta 2, Jugo 3: 10.00 litros
Planta 3, Jugo 1: 30.00 litros
Planta 3, Jugo 2: 15.00 litros
Planta 3, Jugo 3: 15.00 litros
```

Ingresos máximos: $970,000.00

2.6. Producción bebidas saludables Frutimax

La empresa FrutiMix S.A.S., dedicada a la producción de bebidas saludables, desea optimizar su plan de producción semanal de 8 tipos de jugos funcionales. Cada jugo requiere una combinación de tres recursos: agua purificada, concentrado de fruta, y un nuevo insumo llamado extracto funcional (como cúrcuma, jengibre o moringa).

Cada jugo tiene una demanda máxima semanal de 70 unidades y una ganancia unitaria determinada. Además, algunos jugos deben cumplir con restricciones adicionales (por ejemplo, no producir más de 50 unidades de jugos energéticos: Limón, Jengibre y Moringa combinados).

Las disponibilidades semanales son:

- Agua purificada: 1200 litros
- Concentrado de fruta: 900 litros
- Extracto funcional: 300 litros

La empresa desea saber cuántas unidades de cada jugo debe producir para maximizar la ganancia, respetando las restricciones operativas.

Producto	Agua (L)	Fruta (L)	Extracto (L)	Precio de venta	Costo total por unidad	Ganancias ($)
Naranja	4	5	0	5000	2200	2800
Mango	3	6	0	5200	2400	2800

Limón	4	4	2	5300	2700	2600
Guayaba	5	5	1	5400	2500	2900
Fresa	3	7	1	5600	2600	3000
Zanahoria	2	6	1	5500	2550	2950
Jengibre	4	3	2	5800	2900	2900
Moringa	5	4	3	6000	3000	3000

```
# Crear el modelo
model_frutimix = Model("Ejercicio_Mezcla_Productos_FrutiMix")
# Parametros
jugos = ['Naranja', 'Mango', 'Limón', 'Guayaba', 'Fresa', 'Zanahoria', 'Jengibre', 'Moringa']
agua = [4, 3, 4, 5, 3, 2, 4, 5]
fruta = [5, 6, 4, 5, 7, 6, 3, 4]
extracto = [0, 0, 2, 1, 1, 1, 2, 3]
ganancia = [2800, 2800, 2600, 2900, 3000, 2950, 2900, 3000]
costo = [2200, 2400, 2700, 2500, 2600, 2550, 2900, 3000]
precio = [5000, 5200, 5300, 5400, 5600, 5500, 5800, 6000]
max_agua = 1200
max_fruta = 900
max_extracto = 300
max_prod = 70
# Variables de decisión
x = model_frutimix.addVars(jugos, vtype=GRB.CONTINUOUS, name="x", lb=0, ub=max_prod)
# Función objetivo
model_frutimix.setObjective(sum(ganancia[i] * x[jugos[i]] for i in range(8)), GRB.MAXIMIZE)
# model.setObjective(sum(ganancia[i] * x[jugos[i]] for i in range(len(jugos))), GRB.MAXIMIZE)
# Restricciones
model_frutimix.addConstr(sum(agua[i] * x[jugos[i]] for i in range(8)) <= max_agua, "Agua")
model_frutimix.addConstr(sum(fruta[i] * x[jugos[i]] for i in range(8)) <= max_fruta, name = "Fruta")
model_frutimix.addConstr(sum(extracto[i] * x[jugos[i]] for i in range(8)) <= max_extracto, name =
"Extracto")
# Restricción adicional
model_frutimix.addConstr(x['Limón'] + x['Jengibre'] + x['Moringa'] <= 50, name = "Energeticos")
```

```
# Resolver
model_frutimix.optimize()
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 4 rows, 8 columns and 25 nonzeros
Model fingerprint: 0x33a72ade
Variable types: 0 continuous, 8 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+00]
  Objective range  [3e+03, 3e+03]
  Bounds range     [7e+01, 7e+01]
  RHS range        [5e+01, 1e+03]
Found heuristic solution: objective 475200.00000
Presolve removed 1 rows and 1 columns
Presolve time: 0.00s
Presolved: 3 rows, 7 columns, 14 nonzeros
Found heuristic solution: objective 392000.00000
Variable types: 0 continuous, 7 integer (0 binary)
Root relaxation: objective 5.685833e+05, 3 iterations, 0.00 seconds (0.00 work units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0 568583.333    0    1 475200.000 568583.333  19.7%     -    0s
H    0     0                       567600.00000 568583.333  0.17%     -    0s
H    0     0                       567700.00000 568583.333  0.16%     -    0s
H    0     0                       567750.00000 568583.333  0.15%     -    0s
H    0     0                       567800.00000 568583.333  0.14%     -    0s
*    0     0               0       567900.00000 567900.000  0.00%     -    0s
Cutting planes:
  MIR: 1
Explored 1 nodes (4 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)
Solution count 7: 567900 567800 567750 ... 392000
Optimal solution found (tolerance 1.00e-04)
```

```
Best objective 5.679000000000e+05, best bound 5.679000000000e+05, gap 0.0000%
# Resultados
if model_frutimix.status == GRB.OPTIMAL:
    print("--- Producción óptima de jugos FrutiMix ---")
    for j in jugos:
        if x[j].X > 0:
            print(f"Jugo {j}: {x[j].X:.2f} unidades")
    print(f"\nGanancia total: ${model_frutimix.ObjVal:,.2f}")
else:
    print("No se encontró solución óptima.")
--- Producción óptima de jugos FrutiMix ---
Jugo Naranja: 68.00 unidades
Jugo Guayaba: 70.00 unidades
Jugo Zanahoria: 10.00 unidades
Jugo Jengibre: 50.00 unidades

Ganancia total: $567,900.00
```

2.7. Producción de jugos naturales Frutilots S.A.S

La empresa FrutiLots S.A.S produce jugos naturales en dos plantas de fabricación, ubicadas en Cali y Medellín. Existen tres tipos de jugo: Naranja, Mango y Fresa, cada uno con requerimientos específicos de materia prima y una demanda máxima en el mercado. Cada planta tiene una capacidad máxima de producción y un costo de operación por tonelada procesada. El objetivo es maximizar el ingreso total de la empresa, cumpliendo con las restricciones de disponibilidad de materia prima y capacidad de cada planta.

Jugo	Ingrediente A (kg/ton)	Ingrediente B (kg/ton)	Demanda Máxima (ton)	Precio de Venta ($/ton)
Naranja	2	3	50	5000

Mango	4	2	40	6000
Fresa	3	5	30	7000

Se debe tener en cuenta la información de las plantas:

Planta	Capacidad Máxima (ton)	Costo Operación ($/ton)
Cali	60	1000
Medellín	50	1200

Ingrediente	Disponibilidad Máxima (kg)
A	400
B	500

```
# Crear el modelo
model_frutilots = Model("Ejercicio_Mezcla_Productos_FrutiLots")
# Índices
plantas = ['Cali', 'Medellin']
jugos = ['Naranja', 'Mango', 'Fresa']
# Parámetros
precio = {'Naranja': 5000, 'Mango': 6000, 'Fresa': 7000}
costo_operacion = {'Cali': 1000, 'Medellin': 1200}
utilidad = {(p, j): precio[j] - costo_operacion[p] for p in plantas for j in jugos}
ingrediente_A = {'Naranja': 2, 'Mango': 4, 'Fresa': 3}
```

```
ingrediente_B = {'Naranja': 3, 'Mango': 2, 'Fresa': 5}
demanda_max = {'Naranja': 50, 'Mango': 40, 'Fresa': 30}
capacidad = {'Cali': 60, "Medellin": 50}
disp_A = 400
disp_B = 500
# Variables
x = model_frutilots.addVars(plantas, jugos, vtype=GRB.CONTINUOUS, name="x", lb=0)
# Objetivo
model_frutilots.setObjective(sum(utilidad[p, j] * x[p, j] for p in plantas for j in
jugos),GRB.MAXIMIZE)
# Restricciones
for j in jugos:
    model_frutilots.addConstr(sum(x[p, j] for p in plantas) <= demanda_max[j],
name=f"Demanda_{j}")
for p in plantas:
    model_frutilots.addConstr(sum(x[p, j] for j in jugos) <= capacidad[p],
name=f"Capacidad_{p}")
model_frutilots.addConstr(sum(ingrediente_A[j] * x[p, j] for p in plantas for j in jugos) <=
disp_A, name="IngredienteA")
model_frutilots.addConstr(sum(ingrediente_B[j] * x[p, j] for p in plantas for j in jugos) <=
disp_B, name="IngredienteB")
# Resolver modelo
model_frutilots.optimize()
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 7 rows, 6 columns and 24 nonzeros
Model fingerprint: 0xe9d0f9ed
Coefficient statistics:
  Matrix range     [1e+00, 5e+00]
```

```
  Objective range  [4e+03, 6e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e+01, 5e+02]
Presolve time: 0.01s
Presolved: 7 rows, 6 columns, 24 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.9400000e+34   1.900000e+31   2.940000e+04      0s
       7    5.3000000e+05   0.000000e+00   0.000000e+00      0s
Solved in 7 iterations and 0.01 seconds (0.00 work units)
           Optimal objective  5.300000000e+05
# Resultados
if model_frutilots.status == GRB.OPTIMAL:
    print("--- Producción óptima con nombres explícitos ---")
    for p in plantas:
        for j in jugos:
            val = x[p, j].X
            if val > 0:
                print(f"Planta {p} debe producir {val:.2f} toneladas de Jugo {j}")
    print(f"\nUtilidad total máxima: ${model_frutilots.ObjVal:,.2f}")
else:
    print("No se encontró solución óptima.")
--- Producción óptima con nombres explícitos ---
Planta Cali debe producir 30.00 toneladas de Jugo Mango
Planta Cali debe producir 30.00 toneladas de Jugo Fresa
Planta Medellin debe producir 40.00 toneladas de Jugo Naranja
Planta Medellin debe producir 10.00 toneladas de Jugo Mango
           Utilidad total máxima: $530,000.00
```

2.8. Optimización de producción y transporte de baterías - ElectroPower S.A.S

La empresa ElectroPower S.A.S. fabrica 3 tipos de baterías recargables** en **3** centros de producción: Bogotá, Medellín y Barranquilla. Estas baterías se distribuyen a **4 centros de distribución regionales: Norte, Centro, Occidente y Sur.

Cada planta tiene una capacidad máxima de producción, disponibilidad limitada de litio y cobre, y cada batería tiene diferentes requerimientos.

El objetivo es maximizar la utilidad total teniendo en cuenta:

- Costos de producción
- Costos de transporte
- Precio de venta
- Restricciones de capacidad y materiales

Datos plantas

Planta	Capacidad (unid)	Litio (kg)	Cobre (kg)
Bogotá	400	600	800
Medellín	300	500	600
Barranquilla	350	550	700

Demanda Mínima por centro

Centro	Demanda (unid)
Norte	250
Centro	300
Occidente	200

Sur	250

Características de baterías

Tipo	Litio (kg)	Cobre (kg)	Precio ($)	Costo prod.
Hogar (B1)	1	2	80	40
Indus (B2)	2	2.5	120	60
EV (B3)	2.5	3	150	75

Costos de transporte planta-centro (por unidad)

Planta / Centro	Norte	Centro	Occidente	Sur
Bogotá	10	8	12	15
Medellín	9	7	14	20
Barranquilla	12	10	8	25

```
# Índices
plantas = ['Bogota', 'Medellin', 'Barranquilla']
centros = ['Norte', 'Centro', 'Occidente', 'Sur']
baterias = ['B1', 'B2', 'B3']
# Parámetros
capacidad = {'Bogota': 400, 'Medellin': 300, 'Barranquilla': 350}
litio_disp = {'Bogota': 600, 'Medellin': 500, 'Barranquilla': 550}
cobre_disp = {'Bogota': 800, 'Medellin': 600, 'Barranquilla': 700}
demanda = {'Norte': 250, 'Centro': 300, 'Occidente': 200, 'Sur': 250}
precio = {'B1': 80, 'B2': 120, 'B3': 150}
costo_prod = {'B1': 40, 'B2': 60, 'B3': 75}
litio = {'B1': 1, 'B2': 2, 'B3': 2.5}
cobre = {'B1': 2, 'B2': 2.5, 'B3': 3}
costo_trans = {
```

```
    ('Bogota', 'Norte'): 10, ('Bogota', 'Centro'): 8, ('Bogota', 'Occidente'): 12, ('Bogota', 'Sur'): 15,
    ('Medellin', 'Norte'): 9, ('Medellin', 'Centro'): 7, ('Medellin', 'Occidente'): 14, ('Medellin', 'Sur'): 20,
    ('Barranquilla', 'Norte'): 12, ('Barranquilla', 'Centro'): 10, ('Barranquilla', 'Occidente'): 8, ('Barranquilla', 'Sur'): 25,
}
# Crear modelo
model = Model("ElectroPower")

# Variables de decisión: x[p,b,c] = unidades de batería b fabricadas en planta p y enviadas al centro c
x = model.addVars(plantas, baterias, centros, vtype=GRB.CONTINUOUS, name="x", lb=0)
# Función objetivo: maximizar utilidad
model.setObjective(
    sum((precio[b] - costo_prod[b] - costo_trans[p,c]) * x[p,b,c] for p in plantas for b in baterias for c in centros),
    GRB.MAXIMIZE
)
# Restricciones por planta: capacidad
for p in plantas:
    model.addConstr(sum(x[p,b,c] for b in baterias for c in centros) <= capacidad[p], name=f"Capacidad_{p}")
    model.addConstr(sum(litio[b] * x[p,b,c] for b in baterias for c in centros) <= litio_disp[p], name=f"Litio_{p}")
    model.addConstr(sum(cobre[b] * x[p,b,c] for b in baterias for c in centros) <= cobre_disp[p], name=f"Cobre_{p}"
# Restricciones de demanda mínima por centro
for c in centros:
    model.addConstr(sum(x[p,b,c] for p in plantas for b in baterias) >= demanda[c], name=f"Demanda_{c}")
```

```
# Resolver modelo
model.optimize()
# Resultados
if model.status == GRB.OPTIMAL:
    print("\nPlan de producción y distribución óptimo:")
    for p in plantas:
        for b in baterias:
            for c in centros:
                val = x[p,b,c].X
                if val > 0:
                    print(f"{val:.2f} unidades de batería {b} desde {p} a {c}")
    print(f"\nUtilidad máxima total: ${model.ObjVal:,.2f}")
else:
    print("No se encontró solución óptima.")
```

2.9. Problema de producción de jugos naturales (Análisis de sensibilidad)

Una empresa productora de jugos naturales embotellados cuenta con tres fábricas (A, B y C) encargadas de producir cinco tipos de jugos (Naranja, Mango, Piña, Fresa y Guayaba). Cada fábrica tiene una capacidad máxima de producción medida en litros diarios. Para la producción de los jugos, se requiere una combinación específica de ingredientes naturales, cuya disponibilidad es limitada. Además, cada tipo de jugo tiene una demanda mínima diaria en el mercado y un precio de venta establecido por litro.

La tabla siguiente muestra la cantidad de materia prima requerida por litro de jugo en cada fábrica, la disponibilidad de materia prima y la capacidad máxima de producción por fábrica:

Ingrediente	Naranja	Mango	Piña	Fresa	Guayaba	Disponibilidad máx (litros)
Agua	2	1.5	2	1.2	1.8	5000
Pulpa	1	2	1.5	2.5	1.2	4000
Azúcar	0.5	0..7	30	0.9	0.6	3000

Cada fábrica tiene una capacidad máxima de producción diaria, siendo de 2000 litros para la Fábrica A, 3000 litros para la Fábrica B y 2500 litros para la Fábrica C. Los precios de venta por litro de jugo en el mercado varían según el sabor, con la Naranja a 1500, Mango a 1800, Piña a 1600, Fresa a 2000 y Guayaba a 1700.

El objetivo es determinar cuántos litros de cada tipo de jugo debe producir cada fábrica para maximizar los ingresos de la empresa, cumpliendo con las restricciones de capacidad y disponibilidad de insumos. Se debe realizar un análisis de sensibilidad, obteniendo los precios sombra de las restricciones y los intervalos de sensibilidad para la capacidad de producción de cada fábrica y la disponibilidad de materia prima.

```
# Conjuntos
Productos = ['Naranja', 'Mango', 'Pina', 'Fresa', 'Guayaba']
Fabricas = ['A', 'B', 'C']
Ingredientes = ['Agua', 'Pulpa', 'Azúcar']
# Parámetros
capacidad_fabrica = {'A': 2000, 'B': 2500, 'C': 1500}
disp_ingr = {'Agua': 5000, 'Pulpa': 4000, 'Azúcar': 3000}
precio_venta = {'Naranja': 1500, 'Mango': 1800, 'Pina': 1300, 'Fresa': 2000, 'Guayaba': 1700}
demanda_min = {'Naranja': 500, 'Mango': 400, 'Pina': 300, 'Fresa': 600, 'Guayaba': 450}
uso_ingr = {
    'Agua': {'Naranja': 2, 'Mango': 1.5, 'Pina': 2, 'Fresa': 1.2, 'Guayaba': 1.8},
```

```
    'Pulpa': {'Naranja': 1, 'Mango': 2, 'Pina': 1.5, 'Fresa': 2.5, 'Guayaba': 1.2},
    'Azúcar': {'Naranja': 0.5, 'Mango': 0.7, 'Pina': 3, 'Fresa': 0.9, 'Guayaba': 0.6}
}
# Crear modelo
model = Model("Sensibilidad_Jugos")
# Variables de decisión
X = model.addVars(Fabricas, Productos, name="X", lb=0)
# Función objetivo
model.setObjective(sum(precio_venta[p] * X[f, p] for f in Fabricas for p in Productos),
GRB.MAXIMIZE)
# Restricciones de capacidad por fábrica
for f in Fabricas:
    model.addConstr(sum(X[f, p] for p in Productos) <= capacidad_fabrica[f],
name=f"Capacidad_{f}")
# Restricciones de disponibilidad de ingredientes
for i in Ingredientes:
    model.addConstr(sum(uso_ingr[i][p] * X[f, p] for f in Fabricas for p in Productos) <=
disp_ingr[i], name=f"Ingr_{i}")
# Restricciones de demanda mínima
for p in Productos:
    model.addConstr(sum(X[f, p] for f in Fabricas) >= demanda_min[p], name=f"Demanda_{p}")
# Resolver modelo
model.optimize()
# Imprimir resultados
if model.status == GRB.OPTIMAL:
    print("\n--- Producción óptima ---")
    for f in Fabricas:
        for p in Productos:
            cantidad = X[f, p].x
            if cantidad > 0:
```

```
            print(f"Fábrica {f} produce {cantidad:.2f} litros de jugo de {p}")
    print(f"\nIngreso total: ${model.objVal:,.2f}")
    # Precios sombra
    print("\n--- Precios Sombra ---")
    for i in Ingredientes:
        restr = model.getConstrByName(f"Ingr_{i}")
        print(f"{restr.constrName}: {restr.pi:.2f}")
    for f in Fabricas:
        restr = model.getConstrByName(f"Capacidad_{f}")
        print(f"{restr.constrName}: {restr.pi:.2f}")
    # Intervalos de sensibilidad
    print("\n--- Intervalos de Sensibilidad (rango de precios aceptables) ---")
    for f in Fabricas:
        for p in Productos:
            var = model.getVarByName(f"X[{f},{p}]")
            if var.x > 0:
                print(f"{var.VarName}: [{var.SAObjLow:.2f}, {var.SAObjUp:.2f}]")
else:
    print("No se encontró una solución óptima.")
```

```
Gurobi Optimizer version 12.0.1 build v12.0.1rc0 (linux64 - "Ubuntu 22.04.4 LTS")
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 11 rows, 15 columns and 75 nonzeros
Model fingerprint: 0x62ed6f94
Coefficient statistics:
  Matrix range     [5e-01, 3e+00]
  Objective range  [1e+03, 2e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e+02, 5e+03]
Presolve time: 0.01s
```

```
Presolved: 11 rows, 15 columns, 75 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0   2.4900000e+34   3.817500e+31   2.490000e+04      0s
       9   4.1400000e+06   0.000000e+00   0.000000e+00      0s
Solved in 9 iterations and 0.01 seconds (0.00 work units)
Optimal objective  4.140000000e+06
--- Producción óptima ---
Fábrica A produce 710.00 litros de jugo de Naranja
Fábrica A produce 400.00 litros de jugo de Mango
Fábrica A produce 300.00 litros de jugo de Pina
Fábrica A produce 450.00 litros de jugo de Guayaba
Fábrica B produce 600.00 litros de jugo de Fresa
Ingreso total: $4,140,000.00
--- Precios Sombra ---
Ingr_Agua: 0.00
Ingr_Pulpa: 1500.00
Ingr_Azúcar: 0.00
Capacidad_A: 0.00
Capacidad_B: 0.00
Capacidad_C: 0.00
--- Intervalos de Sensibilidad (rango de precios aceptables) ---
X[A,Naranja]: [1500.00, inf]
X[A,Mango]: [1800.00, 3000.00]
X[A,Pina]: [1300.00, 2250.00]
X[A,Guayaba]: [1700.00, 1800.00]
        X[B,Fresa]: [2000.00, 3750.00]
```

2.10. Optimización de jugos Vital jugos desde Excel

La empresa VitalJugos S.A.S. produce cinco jugos: Manzana, Uva, Mora, Papaya, Kiwi en tres fábricas: Centro, Norte, Sur.

Tienes un archivo Excel llamado datos_jugos.xlsx que contiene:

```
# 1. Leer los datos del archivo Excel
archivo = 'datos_jugos.xlsx'
productos_df = pd.read_excel(archivo, sheet_name='Productos')
fabricas_df = pd.read_excel(archivo, sheet_name='Fabricas')
ingredientes_df = pd.read_excel(archivo, sheet_name='Ingredientes')
consumo_df = pd.read_excel(archivo, sheet_name='Consumo')
# 2. Crear listas
Productos = list(productos_df['Producto'])
Fabricas = list(fabricas_df['Fabrica'])
Ingredientes = list(ingredientes_df['Ingrediente'])
# 3. Crear parámetros
precio_venta = dict(zip(productos_df['Producto'], productos_df['PrecioVenta']))
demanda_min = dict(zip(productos_df['Producto'], productos_df['DemandaMinima']))
capacidad_fabrica = dict(zip(fabricas_df['Fabrica'], fabricas_df['Capacidad']))
disp_ingr = dict(zip(ingredientes_df['Ingrediente'], ingredientes_df['Disponibilidad']))
consumo = {
   i: dict(zip(Productos, consumo_df.loc[consumo_df['Ingrediente'] == i,
Productos].values.flatten()))
   for i in Ingredientes
}
# 4. Crear el modelo
model = Model("Optimización_Jugos_Desde_Excel")
# 5. Variables de decisión
X = model.addVars(Fabricas, Productos, name="Producción", lb=0)
# 6. Función objetivo: maximizar ingresos
```

```
model.setObjective(sum(precio_venta[p] * X[f, p] for f in Fabricas for p in Productos),
GRB.MAXIMIZE)
# 7. Restricciones
# Capacidad de fábricas
for f in Fabricas:
    model.addConstr(sum(X[f, p] for p in Productos) <= capacidad_fabrica[f],
name=f"Capacidad_{f}")
# Disponibilidad de ingredientes
for i in Ingredientes:
    model.addConstr(sum(consumo[i][p] * X[f, p] for f in Fabricas for p in Productos) <=
disp_ingr[i], name=f"Ingr_{i}")

# Demanda mínima
for p in Productos:
    model.addConstr(sum(X[f, p] for f in Fabricas) >= demanda_min[p], name=f"Demanda_{p}")
# 8. Resolver el modelo
model.optimize()
# 9. Mostrar resultados
if model.status == GRB.OPTIMAL:
    print("\n--- Producción óptima ---")
    for f in Fabricas:
        for p in Productos:
            cantidad = X[f, p].x
            if cantidad > 0:
                print(f"Fábrica {f} produce {cantidad:.2f} litros de {p}")
    print(f"\nIngreso total: ${model.objVal:,.2f}")
else:
    print("No se encontró solución óptima.")
```

```
Optimize a model with 11 rows, 15 columns and 75 nonzeros
Model fingerprint: 0xe15dc0d2
```

```
Coefficient statistics:
  Matrix range     [5e-01, 2e+00]
  Objective range  [2e+03, 2e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+02, 6e+03]
Presolve time: 0.01s
Presolved: 11 rows, 15 columns, 75 nonzeros
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.7900000e+34   3.877500e+31   2.790000e+04      0s
      10    5.8200000e+06   0.000000e+00   0.000000e+00      0s
Solved in 10 iterations and 0.02 seconds (0.00 work units)
Optimal objective  5.820000000e+06
--- Producción óptima ---
Fábrica Centro produce 500.00 litros de Uva
Fábrica Centro produce 400.00 litros de Mora
Fábrica Centro produce 700.00 litros de Kiwi
Fábrica Norte produce 900.00 litros de Manzana
Fábrica Norte produce 700.00 litros de Papaya
Ingreso total: $5,820,000.00
```

2.11. Producción óptima de computadoras

La empresa TecnoFab S.A.S. fabrica tres productos electrónicos: Portátiles Básicos, Portátiles Gaming, y PCs de Escritorio en tres plantas: Planta Norte, Planta Sur y Planta Centro.

Cada tipo de producto requiere ciertas cantidades de Microprocesadores, Memoria RAM y Unidades SSD, y cada planta tiene una capacidad máxima diaria de ensamblaje (en unidades).

La empresa desea maximizar los ingresos totales por ventas, asegurando que se cumplan las siguientes condiciones:

- No se puede exceder la capacidad de ensamblaje de cada planta.
- La disponibilidad total de componentes es limitada.
- Debe cumplirse una demanda mínima diaria para cada tipo de producto.

A continuación, se presentan las tablas con la información correspondiente.

Requerimientos de componentes por producto (por unidad)

Ingrediente	Portátil Básico	Portátil Gaming	Pc escritorio
Microprocesador	1	1	1
RAM	2	4	3
SSD	1	2	2

Capacidad de las plantas

Fábrica	Capacidad (unidades/día)
Planta Norte	300
Planta Sur	250
Planta Centro	350

Disponibilidad diaria de componentes

Componente	Disponibilidad
Microprocesador	1000
RAM	1500
SSD	900

Precio de venta y demanda mínima diaria

Producto	Precio de venta	Demanda mínima
Portátil básico	3.200.000	100
Portátil gaming	6.500.000	80
PC escritorio	4.500.000	120

```
archivo = 'datos_computadoras.xlsx'
productos_df = pd.read_excel(archivo, sheet_name='Productos')
fabricas_df = pd.read_excel(archivo, sheet_name='Fabricas')
ingredientes_df = pd.read_excel(archivo, sheet_name='Ingredientes')
consumo_df = pd.read_excel(archivo, sheet_name='Consumo')
# Listas
Productos = list(productos_df['Producto'])
Fabricas = list(fabricas_df['Fabrica'])
Ingredientes = list(ingredientes_df['Ingrediente'])
# Parámetros
precio_venta = dict(zip(productos_df['Producto'], productos_df['PrecioVenta']))
demanda_min = dict(zip(productos_df['Producto'], productos_df['DemandaMinima']))
capacidad_fabrica = dict(zip(fabricas_df['Fabrica'], fabricas_df['Capacidad']))
disp_ingr = dict(zip(ingredientes_df['Ingrediente'], ingredientes_df['Disponibilidad']))
# Consumo por ingrediente-producto
consumo = {
    i: dict(zip(Productos, consumo_df.loc[consumo_df['Ingrediente'] == i,
Productos].values.flatten()))
    for i in Ingredientes
}
# Crear modelo
```

```
model = Model("Computadoras_Producción")
X = model.addVars(Fabricas, Productos, name="Producción", lb=0)
model.setObjective(sum(precio_venta[p] * X[f, p] for f in Fabricas for p in Productos),
GRB.MAXIMIZE)
# Restricciones
for f in Fabricas:
   model.addConstr(sum(X[f, p] for p in Productos) <= capacidad_fabrica[f],
name=f"Capacidad_{f}")
for i in Ingredientes:
   model.addConstr(sum(consumo[i][p] * X[f, p] for f in Fabricas for p in Productos) <=
disp_ingr[i], name=f"Ingr_{i}")
for p in Productos:
   model.addConstr(sum(X[f, p] for f in Fabricas) >= demanda_min[p], name=f"Demanda_{p}")
model.optimize()
# Resultados
if model.status == GRB.OPTIMAL:
   print("\n--- Producción óptima ---")
   for f in Fabricas:
      for p in Productos:
         cantidad = X[f, p].x
         if cantidad > 0:
            print(f"{f} produce {cantidad:.2f} unidades de {p}")
   print(f"\nIngreso total: ${model.ObjVal:,.2f}")
else:
   print("No se encontró solución óptima.")
```

3. Manual de Usuario: Modelos de Optimización Lineal con Gurobi y Python

1. Introducción

Este manual proporciona una guía práctica para construir, resolver e interpretar modelos de optimización lineal utilizando Python y el solver Gurobi. Está dirigido a estudiantes y docentes que deseen desarrollar modelos como los empleados en la producción de jugos o mezcla de productos con restricciones de capacidad, insumos y demanda.

2. Estructura Básica de un Modelo de Optimización

2.1 Importación de librerías

from gurobipy import Model, GRB

import pandas as pd (si se usan datos desde Excel)

2.1 Creación del modelo

Esto crea un objeto de modelo de Gurobi, el cual contendrá todas las variables, restricciones y la función objetivo.

2.2 Definición de conjuntos e índices

Los conjuntos representan los grupos de elementos sobre los cuales se construye el modelo, como productos, plantas, ingredientes, etc.

2.3 Parámetros

Variables fijas que no cambian durante la ejecución del modelo, como precios, capacidades, demandas.

2.4 Variables de decisión

Se crean con model.addVars(...). Representan las cantidades a determinar, como cuánto producir de cada jugo. Puede definirse el limite inferior (lb=n) y definir

2.5 Función objetivo

Se define con model.setObjective(...). Puede ser de maximización (GRB.MAXIMIZE) o minimización (GRB.MINIMIZE).

2.6 Tipo de variables numéricas

Se define con vtype=. Puede ser Continuos (GRB. CONTINUOUS), Enteros (GRB.INTEGER) y Binarias (GRB.BINARY).

2.7 Restricciones

Se definen con model.addConstr(...) o model.addConstrs(...). Limitan las decisiones posibles según recursos, demanda, etc.

2.8 Solución del modelo

model.optimize(): Esta instrucción resuelve el modelo usando el solver de Gurobi.

3. Interpretación de Resultados

X[f, p].x: Devuelve el valor óptimo de la variable de decisión 'X[f, p]' tras resolver el modelo.

model.objVal: Devuelve el valor óptimo de la función objetivo.

v.VarName: Nombre de la variable, útil para imprimir resultados o escribirlos a archivos.

model.getConstrByName('Nombre'): Permite obtener una restricción para acceder a sus propiedades.

c.pi: Precio sombra de la restricción 'c'. Muestra cuánto mejoraría la función objetivo si se relaja una unidad esta restricción.

v.SAObjLow, v.SAObjUp: Límite inferior y superior del intervalo de sensibilidad para el coeficiente objetivo de una variable.

4. Lectura de Datos desde Excel

Se usa pandas: pd.read_excel('archivo.xlsx', sheet_name='Hoja') para importar conjuntos, parámetros o consumos.

5. Recomendaciones Finales

- Verifica que todos los parámetros y conjuntos estén correctamente definidos.
- Usa nombres explícitos para variables y restricciones.
- Interpreta los resultados óptimos y los precios sombra para obtener valor del análisis de sensibilidad