UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA, LEÓN.

ÁREA DE CONOCIMIENTO CIENCIAS Y TECNOLOGÍA.

ÁREA DE CONOCIMIENTO ESPECÍFICO DE MATEMÁTICA Y ESTADÍSTICA.

TRABAJO MONOGRÁFICO PARA OPTAR AL TÍTULO DE LICENCIADO EN MATEMÁTICA.

UTILIZACIÓN DE LOS PROGRAMAS, SCILAB, PYTHON, GEOGEBRA Y R EN LA SOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL.

Autoras:

Br. Lourdes Nayelis Benavidez Monjarrez.

Br. Jessy Juniette Centeno Aguirre.

Br. Martha Lorena Olivas Ramos.

Tutor:

Lic. Cesar Frederick Ruiz Chávez.

León, Nicaragua, 2024.

2024: 45/19 ¡LA PATRIA, LA REVOLUCIÓN!

RESUMEN

La programación lineal es una herramienta matemática esencial para optimizar recursos, con aplicaciones en áreas como la ingeniería, la economía y la logística. En este trabajo monográfico se explora la aplicación de diferentes programas como Scilab, Python, GeoGebra y R en la solución de problemas de programación lineal utilizando el método gráfico y el método simplex. Estos programas ofrecen plataformas versátiles para el análisis y resolución de sistemas lineales, permitiendo a los usuarios modelar y optimizar problemas de manera eficiente. Scilab una plataforma de software libre, que se destaca por su capacidad para realizar cálculos numéricos y manejar matrices grandes, facilitando la solución de modelos lineales mediante métodos como el simplex. Python, con sus bibliotecas como SciPy y Pulp ofrece un entorno flexible para implementar algoritmos avanzados la implementación de optimización, estas bibliotecas simplifican la formulación y la solución de problemas de programación lineal. GeoGebra, aunque es conocido principalmente como una herramienta educativa, permite visualizar gráficamente soluciones y restricciones en el plano, esto es muy útil para enseñar los conceptos fundamentales de programación lineal, y finalmente, R, con paquetes como lpSolve, permite resolver problemas lineales de manera eficiente y analizar los resultados estadísticamente. A lo largo de esta investigación se realiza un análisis comparativo de las características, ventajas y limitaciones de cada uno de estos programas, se evalúa su facilidad de uso, precisión en los resultados y aplicabilidad en problemas del mundo real.

AGRADECIMIENTO

Agradecemos primeramente a Dios por darnos la oportunidad y la fortaleza para culminar nuestro trabajo monográfico, su guía ha sido fundamental en este proceso.

A nuestros padres y familiares por el amor y apoyo incondicional, sincero y valioso de todos los días sin ustedes no habríamos podido alcanzar esta grandiosa etapa de nuestras vidas.

A nuestros compañeros por el trabajo en equipo, por compartir este viaje académico con nosotros, cada uno de ustedes contribuyeron a hacer este proceso algo muy valioso y enriquecedor.

A nuestro tutor Lic. Cesar Frederick Ruiz Chávez por habernos guiado con su conocimiento y brindarnos su confianza a lo largo de nuestra investigación, su orientación y apoyo a sido invaluable y ha enriquecido nuestro trabajo monográfico.

DEDICATORIA

A Dios Todopoderoso

Dedicamos el presente trabajo monográfico por ser fuente inagotable en nuestras vidas por habernos permitido llegar hasta este punto y darnos salud para lograr nuestros objetivos, además de su infinita bondad y amor.

A Nuestros Padres

Por habernos apoyado en todo momento, por sus consejos, sus valores, por la motivación constante que nos han brindado. Su amor incondicional, apoyo moral y respaldo económico durante nuestra vida académica, han sido pilares fundamentales en nuestro camino. Sin ustedes no hubiéramos podido alcanzar esta importante etapa de nuestras vidas y su compromiso ha sido fuente de inspiración en cada uno de nuestros logros.

A Nuestros Maestros

Por habernos brindado su conocimiento y apoyo incondicional, su dedicación y compromiso han sido esenciales en nuestra formación académica, sus enseñanzas dejan una huella en nuestra vida.

ÍNDICE GENERAL.

Capítulo 1

1.1. INTRODUCCIÓN

La programación lineal ha sido, desde su introducción en la década de 1940, una herramienta clave para la toma de decisiones en la optimización de recursos limitados, este método matemático se utiliza para maximizar o minimizar una función objetivo, sujeta a una serie de restricciones lineales. Su campo de aplicación se extiende desde la planificación de producción y la distribución de bienes hasta la investigación operativa y la gestión empresarial. La capacidad de resolver problemas complejos de manera eficiente y precisa ha hecho que la programación lineal sea indispensable en diversas industrias y disciplinas académicas. (Fedossova, Buitrago & Britto, 2011)

En este contexto programas como Scilab, Python, GeoGebra y R han surgido como operaciones accesibles y potentes para la solución de problemas de programación lineal. Estos entornos de software no solo permiten la implementación de algoritmos como el método simplex o método gráfico, sino que también ofrece la capacidad de modelar, simular y visualizar soluciones a problemas de optimización. Además, de carácter libre y de código abierto estas herramientas las convierte en una alternativa económica y flexible frente a soluciones propietarias.

Cada uno de estos programas presenta características particulares que los hacen útiles en diferentes escenarios. Scilab con su estructura similar a Matlab, facilita la solución numérica de problemas de algebra lineal y programación lineal. Python con sus múltiples bibliotecas especializadas, ofrece una plataforma versátil para modelar y resolver modelos complejos. GeoGebra conocido por su uso en la enseñanza en matemáticas. R principalmente utilizado en análisis estadístico,

cuenta con paquetes específicos que permiten abordar problemas de programación de optimización de forma eficiente.

Este trabajo monográfico se propone explorar y comparar el uso de estos cuatro programas en la solución de problemas de programación lineal, destacando sus fortalezas, limitaciones y casos prácticos de aplicación. A lo largo del trabajo se presentarán ejemplos concretos que ilustran el proceso de modelo y solución, así como una evaluación para el ámbito educativo y profesional.

1.2. OBJETIVOS

1.2.1. Objetivo General

- Aplicar de forma general el uso de los programas Scilab, Python, GeoGebra y R en la solución de problemas de programación lineal.

1.2.2. Objetivos Específicos

- Interpretar y plantear modelos de programación lineal aplicado a la economía, salud y administración.
- Generar correctamente problemas de programación lineal mediante el uso de los programas Scilab, Python, GeoGebra y R, así como posibles errores que se puedan cometer en dichos programas.
- Encontrar la solución óptima del problema de programación lineal.
- Presentar ideas básicas en la solución de problemas de programación lineal.

Capítulo 2

MARCO TEÓRICO

2.1. Programación lineal

La programación lineal es una herramienta valiosa en la toma de decisiones empresariales, permite encontrar soluciones óptimas a problemas complejos con múltiples variables. La programación lineal consiste en la optimización (maximización o minimización) de una función lineal, denominada función objetivo, sujeta a un conjunto de restricciones expresada como desigualdades o igualdades lineales, el objetivo es encontrar los valores de las variables que optimicen la función objetivo respetando las restricciones del problema. (Taha, 2004)

El problema típico de programación lineal se fórmula de la siguiente manera:

Maximizar o Minimizar:

$$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

Sujeta a:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = (\geq, \leq)\ b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = (\geq, \leq)\ b_2$$

$$. \quad . \quad + \cdots + \quad . \quad .$$

$$. \quad . \quad + \cdots + \quad . \quad .$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = (\geq, \leq)\ b_m$$

$$x_1, x_2, \ldots, x_n \geq 0$$

Donde:

$Z = C1X1 + C2CX2 + \cdots . + CnXn$: Función objetivo.

$Xj =$ Variables o actividades $j(para\ j = 1,2, \dots, n)$.

$Cj =$ Coeficiente de costo $j(para\ j = 1,2, \dots, n)$.

$bi =$ Disponibilidades o requerimientos $(para\ i = 1,2, \dots, m)$.

$aij =$ Coeficientes tecnológicos.

$Xij \geq 0 =$ Restricciones de no negatividad.

La formulación del problema de programación lineal (PPL) anterior esta descrita de forma de extensión y general.

2.1.1. Un problema de programación lineal consta:

CRITERIO	**FUNCION OBJETIVO**
Max o Min	$\boldsymbol{f = \sum_{j=1}^{n} c_j x_j}$
RESTRICCIÓN	$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i = \overline{1,m}$
s. a.	
	$x_j \geq 0 \qquad j = \overline{1,n}$
$\boldsymbol{n.n.}$	

Aquí la representación es de sumatoria y la siguiente presentación es matricial.

2.1.1.2. Formas de PPL:

Forma	Problema de Minimización	Problema de Maximización
Canónica	Min $F = CX$ $s.a. \quad AX > B$ $n.n. \quad X > 0$	Max $F = CX$ $s.a. \quad AX < B$ $n.n. \quad X > 0$
Estándar	Min $F = CX$ $s.a. \quad AX = B$	Max $F = CX$ $s.a. \quad AX = B$

	$X > 0$	$X > 0$
Mixta	Min $F = CX$ $s.a.$ $A1\,X < B1$ $A2\,X > B2$ $A3\,X = B3$ $n.n.$ $X > 0$	Max $F = CX$ $s.a.$ $A1\,X < B1$ $A2\,X > B2$ $A3\,X = B3$ $n.n.$ $X > 0$

2.1.1.3. Propiedades para la transformación de PPL:

1. $Max\,F \leftrightarrow -Min - F$ $\quad Min\,F \leftrightarrow -Max - F$
2. $ax \leq -b \leftrightarrow -ax \geq b$ $\quad ax \geq -b \leftrightarrow -ax \leq b$
3. $ax = b \leftrightarrow ax \leq b \,\wedge\, -ax \leq -b$ $\quad \leftrightarrow ax \geq b \,\wedge\, -ax \geq -b$
4. $x \leq 0 \leftrightarrow x = -x_1 \;/\; x_1 \geq 0$
5. $x\ irrestringida \leftrightarrow x = x_1 - x_2 \;/\; x_1 \geq 0,\ x_2 \geq 0$
6. $ax \leq b \leftrightarrow ax + h = b \;/\; h \geq 0$
7. $ax \leq b \leftrightarrow ax - h = b \;/\; h \geq 0$

La solución de problemas de programación lineal puede obtenerse mediante métodos, como el método gráfico (para problemas con dos variables) y el método simplex (para problemas con más de dos variables), además existen métodos computacionales más avanzados, como el método de los puntos interiores, que son esenciales para resolver problemas grandes o complejos.

2.1.2. Métodos.

2.1.2.1. Método Gráfico

El método gráfico es una técnica utilizada para resolver problemas de programación lineal con dos variables. Aunque no es muy práctico para una gran cantidad de variables, es muy útil para interpretar y analizar los resultados y la sensibilidad del problema. Este método permite encontrar el máximo o el mínimo de una función.

El objetivo mediante la representación gráfica de las restricciones del análisis de la región factible. Aunque es sencillo su uso se limita a problemas de dos variables debido a la complejidad de representar más dimensiones gráficamente. (Hillier, 1993)

2.1.2.2. Método gráfico para problemas de programación lineal.

Gráfico de las ecuaciones lineales

- La recta $ax + by = c$, con $a, b, c \neq 0$, pasa por los puntos $(\frac{c}{a}, 0)$ y $(0, \frac{c}{b})$.
- La recta $x = c$ es una recta perpendicular al eje x y lo corta en "c"
- La recta $y = c$ es una recta perpendicular al eje y y lo corta en "c"
- La recta $ax + by = 0$, con $a, b \neq 0$ para los puntos $(0,0)$ y $(1, -\frac{a}{b})$, o bien por $(0,0)$ y $(-\frac{b}{a}, 1)$

2.1.2.3. Región de las desigualdades

- Gráfico de la desigualdad $ax + by \leq c$, se construye escogiendo como la región de la desigualdad la mitad del plano que se extiende de la recta en el sentido contrario con el valor normal de $\vec{r} = \overrightarrow{(a, b)}$.
- El gráfico de la desigualdad $ax + by > c$, se construye escogiendo como región de la desigualdad la mitad del plano que se extiende de la recta en el sentido coincidente con el valor normal de $\vec{r} = \overrightarrow{(a, b)}$.

2.1.2.4. Procedimiento de solución método gráfico

- Se gráfica el vector de costos $\vec{C} = \overrightarrow{(c_1, c_2)}$.
- Se trazan rectas perpendiculares al vector de costos que corten a la región factible, y se desplazan en el sentido del vector $\vec{C}$, si el problema es de maximizar, o en el sentido contrario al del vector $\vec{C}$, si el problema es de minimizar. El último de los puntos de la región factible cortadas por las rectas que se desplaza es el punto óptimo.

A veces es necesario resolver un sistema de ecuaciones para obtener el punto óptimo.

El método gráfico es una herramienta didáctica y visual, aunque su aplicación es limitada a problemas simples, su aplicación en programas como, Scilab, Python, GeoGebra y R permite a los usuarios aprovechar sus capacidades gráficas y de optimización para explorar el problema, visualizar la solución y automatizar cálculos.

2.1.3. Método Simplex

El método simplex es una técnica iterativa utilizada para resolver problemas de programación lineal con más de dos variables. Se emplea principalmente cuando el número de variable y restricciones es grande y el método gráfico no es aplicable. En este método se construye una tabla que muestra las variables y las restricciones, y se realiza una serie de iteraciones para encontrar la solución óptima, el método simplex busca maximizar o minimizar una función objetivo lineal bajo un conjunto de restricciones lineales. (Salazar, 2019)

2.1.3.1. El método simplex (para PPL maximizar)

Transformar PPL en forma estándar

- Todas las variables de decisión mayor o igual que cero, $x \geq 0$.
- Todo el lado derecho mayor o igual que cero, $b \geq 0$.
- A cada restricción con $\leq$ se le suma en su lado izquierdo una variable llamada variable de holgura $h_i \geq 0$, y se cambia el $\leq$ por $=$.
- A cada restricción con $\geq$ se le resta en su lado izquierdo una variable llamada variable de holgura $h_i \geq 0$, y se cambia el $\geq$ por $=$.

$$Max \;\; 1f - c_1x_1 - c_2x_2 - \cdots - c_nx_n + 0h_1 + 0h_2 + 0h_3 + \cdots + 0h_n = 0$$

$$s.a \qquad 0f + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + 1h_1 + 0h_2 + 0h_3 + \cdots + 0h_n = b_1$$

$$0f + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + 0h_1 + 1h_2 + 0h_3 + \cdots + 0h_n = b_2$$

$$0f + a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n + 0h_1 + 0h_2 + 1h_3 + \cdots + 0h_n = b_3$$

$$\dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots$$

$$0f + a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + 0h_1 + 0h_2 + 0h_3 + \cdots + 1h_n = b_n$$

$$n.n \quad x_1, x_2, \dots, x_n, h_1, h_2, h_3 \dots h_n \geq 0$$

2.1.3.2. La tabla inicial simplex

- Se revisa cada restricción en busca de una variable que tenga $+1$ de coeficiente y cero en las otras restricciones, si tal variable existe se le llamara variable básica. Si en una restricción no se encuentra una variable con estas características se le suma en su lado izquierdo una variable llamada artificial, $A_j \geq 0$ y también variable básica.
- Para cada variable artificial A_j agregada en alguna restricción se le suma a la función objetivo el termino $M\ A_j$ en caso de minimizar o $-\ M\ A_j$ en el caso de maximizar, donde M se considera tan grande como sea conveniente.

2.1.3.3. Construcción de la tabla inicial

<table>
<tr><td rowspan="2">C.B</td><td rowspan="2">V.B</td><td>Lista de coeficientes en la Función objetivo</td><td rowspan="2">VALOR</td></tr>
<tr><td>Lista de las variables del problema</td></tr>
<tr><td>Lista de los coeficientes de las variables básicas</td><td>Lista de variables básicas</td><td>Matriz del sistema de Restricciones (A)</td><td>Lado derecho del sistema de restricciones</td></tr>
<tr><td></td><td>Z_j</td><td>Producto de la columna C.B por las columnas de la matriz del sistema de restricciones</td><td></td></tr>
<tr><td></td><td>$C_j - Z_j$</td><td>Resta de los coeficientes en la función objetivo con sus correspondientes en la fila Z</td><td>Valor de la función objetivo</td></tr>
</table>

2.1.3.4. El algoritmo simplex

Después de haber transformado el problema inicial a la forma conveniente descrita y construido la tabla simplex inicial.

1) Escoja el mínimo de los negativos del Reglón cero, R_0, la variable que corresponda a ese mínimo llámela variable que entra a la base.
2) Si el mínimo elegido en el paso anterior es mayor o igual que cero, $R_0 \geq 0$, fin y diga:
    a. Que el problema original tiene óptimo finito (solución) en el punto que se obtiene asignándole a las variables básicas el valor correspondiente del lado derecho, y cero a las restantes; todo eso si todas las variables artificiales tienen valor cero.
    b. Que el problema original es no factible (no tiene solución), si existen variables artificiales básicas con valor mayor que cero.
3) Si los elementos en el cuerpo principal debajo de la variable de entrada son todos menores o iguales que cero, $a_{ij} \geq 0$, termine el proceso diciendo que:
    a. El problema original es no factible, si hay variables básicas artificiales con valores mayores que cero.

En caso contrario continue.

4) Calcule la menor de las razones de los elementos del lado derecho entre los correspondientes elementos positivos de la columna debajo de la variable que entra, $\min\left\{\frac{b_j}{a_{ij}} / a_{ij} > 0\right\}$. Al elemento de la columna debajo de la variable que entra donde ocurre el mínimo llámelo pivote, y a la variable básica que corresponde a esa fila llámela variable que sale de la base.
5) Pivotee en la tabla, realice operaciones entre Reglones con el Reglón cero, el cuerpo principal y el lado derecho (R_0, a_{ij} y b_j) de tal manera que la columna de pivote se transforme en una columna de ceros y un único 1 en la posición del pivote.
6) Cambie en la lista de variables básicas la variable que sale de la base por la variable que entra a la base.

7) Regrese al paso 1)

2.1.3.5. El método simplex (para PPL minimización)

Para problemas de minimización se cambia

- Escoja el máximo de los elementos positivos del Reglón cero y la variable que corresponda a ese máximo llámele variable que entra a la base.
- Si el máximo elegido en el paso anterior es menor o igual que cero, $R_0 \leq 0$, fin y diga: ….

El método simplex para resolver problemas de programación lineal cuando hay variables artificiales una vez obtenida la tabla inicial del simplex (Antes de aplicar el algoritmo simplex), se busca como eliminarlas usando dos métodos: El método de la M grande y el método de las dos fases.

$$Min \quad 1f - c_1x_1 - c_2x_2 - .. - c_nx_n + 0h_1 + 0h_2 + 0h_3 + .. + 0h_n - MA_1 - MA_2 = 0$$

$$s.a \; 0f + a_{11}x_1 + a_{12}x_2 + . + a_{1n}x_n - 1h_1 + 0h_2 + 0h_3 + .. + 0h_n + 1A_1 + 0A_2 = b_1$$

$$0f + a_{21}x_1 + a_{22}x_2 + . + a_{2n}x_n + 0h_1 - 1h_2 + 0h_3 + .. + 0h_n + 0A_1 + 1A_2 = b_2$$

$$0f + a_{31}x_1 + a_{32}x_2 + . + a_{3n}x_n + 0h_1 + 0h_2 + 1h_3 + .. + 0h_n + 0A_1 + 0A_2 = b_3$$

… ….

$$0f + a_{m1}x_1 + a_{m2}x_2 + . + a_{mn}x_n + 0h_1 + 0h_2 + 0h_3 + .. + 1h_n + 0A_1 + 0A_2 = b_n$$

$$n.n \quad x_1x_2, \ldots, x_n, h_1, h_2, h_2 \ldots h_n, A_1, +A_2 \geq 0$$

El método simplex y su implementación en los programas, Scilab, Python, GeoGebra y R ofrece una amplia gama de posibilidades para resolver problemas de optimización en diferentes contextos. Python y R son ideales para resolver problemas complejos y automatizados, mientras que GeoGebra es más adecuado para presentar gráficamente problemas simples y didácticos. Scilab es una opción intermedia para quienes buscan una solución más libre y matemática.

2.2. Programas y su aplicación en programación lineal.

2.2.1. Scilab y su Aplicación en Programación Lineal.

El software de libre distribución Scilab es un programa para reconocer y experimentar con el uso de variables, es un programa de cálculo numérico que trabaja con una ventana de comandos, permite realizar operaciones con cálculo matriciales, con polinomios, operaciones con ecuaciones lineales y diferenciales, graficar funciones en 2D y 3D, y además programa sus propias funciones. Es utilizado principalmente en matemáticas, física y ciencias de la ingeniería. Es popular en el campo académico por su similitud con Matlab y su capacidad para manejar cálculos numéricos avanzados. (Ferrari, 2015)

El software Scilab funciona a través de la ejecución de comandos en una ventana o consola que se abre al seleccionar el programa en el menú de programas de su sistema operativo. La forma de trabajo consiste en escribir un orden en la ventana de comandos para indicar al ordenador que tarea debe realizar.

En el contexto de la programación lineal, Scilab ofrece funciones y bibliotecas que permiten la formulación y solución de problemas de optimización. Uno de los enfoques más comunes es el uso del paquete Optimization, que incluye el método simplex para resolver problemas de programación lineal. La función simplex implementa el método simplex, una técnica clásica para resolver problemas de programación lineal. Estas son algunas ventajas de Scilab:

- Capacidad para manejar grandes volúmenes de datos y sistema de ecuaciones lineales.
- Buen rendimiento en la resolución de problemas numéricos.
- Una amplia gama de bibliotecas dedicada a la optimización.

2.2.2. Python y su Aplicación en Programación lineal.

Python es un lenguaje de programación de código abierto, orientado a objetos, fácil de interpretar y con una sintaxis que permite leerlo de manera semejante a como se lee en inglés, es un leguaje interpretado, significa que el código de

programación se convierte en bytecode y luego se ejecuta por el intérprete, en este caso, la máquina virtual de Python. El código escrito en Python puede ejecutarse en sistemas operativos como $Windows$, $macOs$ y $Linux$, sin necesidad de modificaciones importantes. Así mismo se a consolidado como unos de los lenguajes de programación más utilizado en la ciencia y la ingeniería, debido a su simplicidad y flexibilidad. (Frisoli, 2022)

Python es uno de los lenguajes de programación más versátiles y populares, es una herramienta potente y muy flexible que facilita la creación de modelos de optimización matemática en programación lineal. En la actualidad su simplicidad sintáctica, junto con una extensa colección de bibliotecas, lo hace adecuado para una amplia gama de aplicaciones, incluida la programación lineal. Las bibliotecas que más destacan son:

- **SciPy:** Aunque esta más enfocada en problemas de optimización no lineal, también puede manejar problemas de programación lineal.
- **CVXPY:** Una de las bibliotecas más potentes que permite formular y resolver problemas de optimización convexa, incluyendo programación lineal.
- **Pulp:** Una biblioteca específica para modelar y resolver problemas de programación lineal y de programación entera.
- **Numpy y Pandas:** Son útiles para la manipulación de datos y matrices, que son componentes fundamentales en la formulación de problemas de programación lineal.

Python destaca por su capacidad de integración con otras herramientas y su facilidad para realizar análisis de datos complejos, visualizaciones y automatización de procesos.

2.2.3. GeoGebra y su Aplicación en Programación Lineal.

GeoGebra es un software matemático que ofrece la posibilidad de asociar objetos geométricos y algebraicos para resolver problemas complejos, permite fundamentalmente la interactividad, estimula el desarrollo afectivo, social, cognitivo, motriz, del lenguaje verbal-simbólico y de emprendimiento por el hecho que pueden

crear modelos prácticos para la transformación de la realidad, diseñado inicialmente para la enseñanza de la geometría, pero que también incluye herramientas para algebra, cálculo y optimización. (Manco, 2020)

En el caso de la programación lineal, GeoGebra es especialmente útil para la visualización de problemas con dos variables, permitiendo a los estudiantes y usuarios observar gráficamente las restricciones y la función objetivo. Es una herramienta pedagógica poderosa, ya que facilita la comprensión del concepto de regiones factibles y la localización de la solución óptima en problemas simples.

Algunas características en la solución de problemas de programación lineal incluyen:

- Visualización interactiva de restricciones y soluciones.
- Interpretación gráfica de los problemas, ideal para el método gráfico de programación lineal.
- Accesibilidad a usuarios sin conocimientos avanzados de programación.

Aunque sus capacidades son limitadas para problemas de gran escala es una excelente herramienta didáctica para problemas con pocas variables.

2.2.4. R y su Aplicación en Programación Lineal.

R es un lenguaje de programación ampliamente utilizado en el ámbito del análisis de datos y la estadística. Es un entorno de software de código abierto y gratuito que ofrece herramientas y bibliotecas especializadas para el procesamiento, la visualización y el modelado de datos. R se utiliza en una amplia variedad de aplicaciones en el campo del análisis de datos, la estadística y la investigación científica, permite a los profesionales manipular, visualizar y analizar grandes conjuntos de datos de manera eficiente. Aunque es conocido por su enfoque en la estadística, también cuenta con paquetes que permiten resolver problemas de optimización lineal. (Paz & Minguillón, 2019)

R proporciona paquetes para resolver problemas mediante programación lineal, uno de los más sencillos linprog, que se usa el algoritmo simplex para resolver el

problema propuesto mediante una notación matricial para la función objetivo y las restricciones. El paquete más común para abordar problemas de programación lineal en R es lpSolve, que implementa el método simplex y ggplot2 para resolver problemas de método gráfico. Además, R ofrece otras ventajas en el análisis de programación lineal:

- Manejo eficiente de datos y modelos estadísticos.
- Capacidad para crear modelos complejos y analizar los resultados mediante técnicas estadísticas avanzadas.
- Generación de gráficos y visualizaciones que ayudan a interpretar los resultados de los problemas de optimización.

Al ser una herramienta ampliamente utilizada en análisis de datos, R permite integrar la resolución de problemas de programación lineal dentro de un contexto más amplio de análisis y procesamiento de información.

2.3. Ejercicio 1. Scilab.

Una empresa produce dos productos, A y B. El producto A tiene un beneficio por unidad de $3\$$, el requerimiento de recurso uno es de 2 unidades y el requerimiento de recurso dos es de 1 unidad. El producto B tiene un beneficio por unidad de $4\$$, el requerimiento de recurso uno es de 1 unidad y el requerimiento de recurso dos es de 3 unidades. La empresa dispone de 100 unidades del recurso uno y 90 unidades del recurso dos. La empresa quiere maximizar el beneficio total.

2.3.1. Planteamiento del problema.

1. **Variables de decisión:**
   Definamos las variables que representan la cantidad de cada producto que se va a producir
   - x_A: cantidad de producto A a producir.
   - x_B: cantidad de producto B a producir.
2. **Función objetivo:**
   La función objetivo representa el beneficio total que queremos maximizar:

$$Maximizar\ Z = 3x_A + 4x_B$$

Donde:

- $3x_A$: Beneficio total por las unidades de producto A producidas.
- $4x_B$: Beneficio total por las unidades de producto B producidas.

3. **Restricciones:**

Las restricciones se derivan de los requerimientos de recursos para la producción de ambos productos y la disponibilidad de estos.

Recursos disponibles:

Recurso 1:

- Producto A requiere 2 unidades de recurso 1.
- Producto B requiere 1 unidad de recurso 1.

Por lo tanto, la restricción para recurso 1 es:

$$2x_A + 1x_B \leq 100$$

Recurso 2:

- Producto A requiere 1 unidad de recurso de recurso 2.
- Producto B requiere 3 unidades de recurso 2.

La restricción para recurso 2 es:

$$1x_A + 3x_B \leq 90$$

Restricción de no negatividad:

Además de las restricciones de recurso tenemos que asegurarnos de que las cantidades producidas no sean negativas:

$$x_A \geq 0$$

$$x_B \geq 0$$

Resumen del planteamiento.

El problema de maximización del beneficio total se puede resumir como:

$$Maximizar\ Z = 3x_A + 4x_B$$

Sujeta a:

$$2x_A + 1x_B \leq 100$$

$$1x_A + 3x_B \leq 90$$

$$x_A \geq 0, \qquad x_B \geq 0$$

2.3.2. Solución en Scilab.

Utilizando el Método Gráfico.

Paso 1: Definir las variables.

Productos A y B

```
// Definición de variables
profit_A = 3;   // Beneficio por unidad de A
resource1_A = 2; // Requerimiento de recurso 1 para A
resource2_A = 1; // Requerimiento de recurso 2 para A

profit_B = 4;   // Beneficio por unidad de B
resource1_B = 1;  // Requerimiento de recurso 1 para B
resource2_B = 3;  // Requerimiento de recurso 2 para B
```

Aquí se definen los beneficios por unidad de cada producto y los recursos necesarios para producir una unidad de cada uno.

Disponibilidad de recursos

```
available_resource1 = 100;  // Total de recurso 1 disponible
available_resource2 = 90;  // Total de recurso 2 disponible
```

Se establece la cantidad total de cada recurso disponible para la producción.

Paso 2: Definir el rango para las variables de decisión.

```
// Definición del rango para las variables de decisión
x_A = 0:1:50;  // Cantidad de producto A (de 0 a 50)
```

Definir un rango de valores posibles para la cantidad de producto A que se puede producir (de 0 a 50 unidades).

Paso 3: Calcular las cantidades de producto B según las restricciones.

```
// Cálculo de x_B (producto B) según las restricciones
x_B1 = (available_resource1 - resource1_A * x_A) / resource1_B; // Restricción para recurso 1
x_B2 = (available_resource2 - resource2_A * x_A) / resource2_B; // Restricción para recurso 2
```

Se calculan las cantidades máximas de producto B que se pueden producir en función de las cantidades de producto A, según las restricciones de los recursos.

Paso 4: Limitar los valores de x_B a no negativos.

```
// Limitar los valores de x_B a no negativos
x_B1(x_B1 < 0) = 0;
x_B2(x_B2 < 0) = 0;
```

Se asegura que las cantidades de producto B no sean negativas, ya que no se puede producir una cantidad negativa de un producto.

Paso 5: Graficar las restricciones.

```
// Graficar las restricciones con líneas más gruesas
clf; // Limpiar la figura
plot(x_A, x_B1, 'r', 'thickness', 4); // Restricción recurso 1 con grosor 3
plot(x_A, x_B2, 'b-', 'thickness', 4); // Restricción recurso 2 con grosor 3
xlabel('Cantidad de producto x_A');
ylabel('Cantidad de producto x_B');
title('Método Gráfico para Maximizar Beneficios');
xgrid();
legend('Restricción Recurso 1', 'Restricción Recurso 2');
```

Se grafican las restricciones en un gráfico, donde:

- $plot$ dibuja las restricciones.
- $xlabel, ylabel,$ y $title$ agregan etiquetas para los ejes y el título.
- Se añaden etiquetas a los ejes y una leyenda para identificar las restricciones.

Paso 6: Definir la región factible.

```
// Definir la región factible
// Coordenadas de la región factible
fgx = [0, 42, 50, 0]; // Puntos en x
fgy = [30, 16, 0, 0]; // Puntos en y

// Definir el color dodgerblue1 (30, 144, 255 en RGB)
dodgerblue1 = color(30, 144, 255);

// Graficar la región factible con dodgerblue1
xfpoly(fgx, fgy, dodgerblue1); // Rellenar la región factible con dodgerblue1
```

Se definen los vértices de la región factible (donde se cumplen todas las restricciones) y se grafica esta región.

Paso 7: calcular puntos de intersección.

```
// Resolver el sistema de ecuaciones para encontrar la intersección
A = [2, 1; 1, 3];
b = [100; 90];
solution = A \ b;
intersec_x_A = solution(1);
intersec_x_B = solution(2);
```

Se calcula el punto de intersección de las dos restricciones, lo que puede ser un vértice de la región factible.

Paso 8: Mostrar el resultado del punto de intersección.

```
// Mostrar los resultados
disp("Intersección de restricciones:");
disp("x_A = " + string(intersec_x_A));
disp("x_B = " + string(intersec_x_B));
```

Se muestran los valores de x_A y x_B en el punto de intersección, los cuales serán utilizados para maximizar el beneficio.

Paso 9: Calcular el beneficio total.

```
// Calcular el beneficio total
beneficio_total = profit_A * intersec_x_A + profit_B * intersec_x_B;
disp("Beneficio total = " + string(beneficio_total));
```

Se identifica el máximo beneficio a partir de los resultados evaluados, así como las cantidades óptimas de A y B que deben producirse. Se calcula usando los beneficios por unidad de A y B, multiplicados por las cantidades óptimas de x_A y x_B.

2.4. Ejercicio 2. Python.

Jens fabrica dos tipos de juguetes de madera: soldados y trenes. Un soldado se vende por 27 dólares y utiliza 10 dólares en materia prima. Cada soldado que se fabrica aumenta la mano de obra variable y los costos generales en 14 dólares. Un tren se vende por 21 dólares y utiliza 9 dólares en cada materia prima, cada tren construido aumenta la mano de obra variable y los costos generales en 10 dólares. La fabricación de soldados y trenes de madera requiere dos tipos de mano de obra calificada: carpintería y acabado. Un solo soldado requiere 2 horas de trabajo de acabado y 1 hora de trabajo de carpintería. Un tren requiere 1 hora de trabajo de acabado y 1 hora de trabajo de carpintería. Cada semana, Jens puede obtener toda la materia prima necesaria pero solo 100 horas de acabado y 80 horas de carpintería. La demanda de trenes es limitada pero como máximo se compran 40 soldados cada semana. Jens quiere maximizar las ganancias semanales (ingresos-costos).

2.4.1. Planteamiento del problema.

1. **Variables de decisión:**
   - x: número de soldados producidos cada semana.
   - y: número de trenes producido cada semana.
2. **Función objetivo:**

   La función objetivo es maximizar las ganancias. Cada soldado vendido genera 27 dólares de ingresos, pero tiene un costo de 10 dólares en materia prima y 14 dólares en mano de obra variable y costos generales, lo que deja una ganancia de:

   $$Ganancia\ por\ soldado = 27 - (10 + 14) = 3$$

   Cada tren vendido genera 21 dólares de ingresos, pero tiene un costo de 9 dólares en materia prima y 10 dólares en mano de obra variable y costos generales, lo que deja una ganancia de:

   $$Ganancia\ por\ ten = 21 - (9 + 10) = 2$$

La función objetivo es entonces:

$$Z = 3x + 2y$$

Donde Z representa la ganancia total.

3. **Restricciones:**
   - **Horas de trabajo de carpintería:** La cantidad total de horas de carpintería es limitada a 80 horas por semana. Cada soldado requiere 1 hora de carpintería, y cada tren también requiere 1 hora. Por lo tanto:
   $$x + y \leq 80$$
   - **Horas de trabajo de acabado:** La cantidad total de horas de trabajo de acabado es limitada a 100 horas por semana. Cada soldado requiere 2 horas de acabado y cada tren 1 hora. Por lo que:
   $$2x + y \leq 100$$
   - **Demanda de soldados:** Como máximo, se pueden vender 40 soldados cada semana:
   $$x \leq 40$$
   - **No negatividad:** No es posible producir un número negativo de soldados o trenes:
   $$x \geq 0, \qquad y \geq 0$$
4. **Resumen del planteamiento.**

$$Maximizar: Z = 3x + 2y$$

Sujeta a:

$$x + y \leq 80$$
$$2x + y \leq 100$$
$$x \leq 40$$
$$x \geq 0, \qquad y \geq 0$$

2.4.2. Solución en Python.

Utilizando Método Gráfico.

Paso 1: Instalar bibliotecas.

Para instalar la biblioteca $numpy$

- Abrir Terminal en el menú de inicio.
- Escribir el siguiente comando y presionar Enter.

```
TERMINAL   PORTS   JUPYTER
\programas tesis> pip install numpy
```

Biblioteca $matplotlib$ el mismo procedimiento.

```
UT   TERMINAL   PORTS   JUPYTER
os\programas tesis> pip install matplotlib
```

Importar las bibliotecas.

```
import numpy as np
import matplotlib.pyplot as plt
```

Estas dos bibliotecas son necesarias para:

- $Numpy(np)$: para mejorar arreglos y hacer cálculos matemáticos.
- $matplotlib.pyplot(plt)$: para generar los gráficos que representarán las restricciones y la región factible.

Paso 2: Definir las restricciones.

```
# Definición de las restricciones
x = np.linspace(0, 50, 400)
```

Aquí se define un rango de valores para cada variable x, que corresponde al número de soldados producidos. La función $np.linspace(0{,}50{,}400)$ crea un arreglo de 400 valores entre 0 y 50. Esto nos permite trazar las restricciones de forma continua.

Restricciones gráficas:

- **Restricción 1:**

```
# Horas de trabajo de acabado
y1 = 100 - 2*x
```

Esta es la ecuación que representa la restricción de horas de acabado $2x + y \leq 100$ despejar y en términos de x para graficar la recta.

- **Restricción 2:**

```
# Horas de trabajo de carpintería
y2 = 80 - x
```

Esta es la ecuación que representa la restricción de horas de carpintería: $x + y \leq 80$. Nuevamente despejar y en términos de x.

- **Restricción 3:**

```
# Limite de producción de soldados
y3 = np.full_like(x, 0)  # Para la restricción x <= 40
```

Esta restricción simplemente establece que el número de soldados no puede exceder de 40: $x \leq 40$. Esto se representa como una línea vertical en $x = 40$.

Paso 3: Graficar las restricciones.

```
# Graficar las restricciones
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label=r'$2x + y \leq 100$', color='blue')
plt.plot(x, y2, label=r'$x + y \leq 80$', color='red')
plt.axvline(40, color='green', linestyle='--', label='Soldados (x ≤ 40)')
```

Este bloque genera el gráfico con las tres restricciones:

- La línea azul es la primera restricción: $2x + y \leq 100$.
- La línea roja es la segunda restricción: $x + y \leq 80$.
- La línea verde discontinua es la restricción de la demanda, que limita la producción de soldados a 40.

Configurar los límites del gráfico:

```
# Configurar límites del gráfico
plt.xlim(0, 50)
plt.ylim(0, 100)
plt.xlabel('Número de Soldados (x)')
plt.ylabel('Número de Trenes (y)')
plt.title('Método Gráfico para Maximizar Ganancias')
```

Se definen los rangos del gráfico en los ejes x (número de soldados) y y (número de trenes), así como etiquetas y el título.

Paso 4: Rellenar la región factible.

```
plt.fill_between(x, 0, np.minimum(y1, y2), where=(x <= 40), color='gray', alpha=0.5)
```

Aquí se sombrea la región factible. La región factible es la zona del gráfico que cumple con todas las restricciones simultáneamente:

- $x \leq 40$ (límite de producción de soldados).
- $2x + y \leq 100$ (restricción de horas de acabado).
- $x + y \leq 80$ (restricción de horas de carpintería).

Se usa $np.minimum(y1, y2)$ para asegurarse de que se sombreen las áreas por debajo de ambas líneas, respetando las restricciones.

Paso 5: Encontrar los puntos de intersección.

```
# Identificar los puntos de intersección
# Corregido: Intersección de 2x + y = 100 y x + y = 80
x_inter = 20
y_inter = 60
```

Este es el punto donde dos restricciones se intersecan (ya lo habíamos calculado manualmente):

- Intersección de $2x + y = 100$ y $x + y = 80$ nos da $x = 20$ y $y = 60$.

Paso 6: Definir los vértices de la región factible.

```
# Vértices de la región factible
vertices = np.array([[0, 0], [0, 80], [40, 0], [x_inter, y_inter]])
```

Los vértices de la región factible son los puntos donde las líneas de las restricciones se intersecan o tocan los ejes. Los vértices importantes para este problema son:

- $(0,0)$: punto de origen.
- $(0,80)$: intersección de la segunda restricción con el eje y.
- $(40,0)$: intersección de la restricción $x \leq 40$ con el eje x.
- $(20,60)$: intersección de las dos restricciones.

Paso 7: Calcular la función objetivo en los vértices.

```
# Calcular la función objetivo en los vértices
Z = 3 * vertices[:, 0] + 2 * vertices[:, 1]
```

La función objetivo es maximizar la ganancia, que se calcula con la fórmula:

$$Z = 3x + 2y$$

Donde 3 es el beneficio por soldado y 2 es el beneficio por tren. Aquí se calcula la ganancia para cada vértice.

Paso 8: Imprimir los resultados.

```
# Imprimir resultados
for i, (vx, vy) in enumerate(vertices):
    print(f"Vértice {i+1}: Soldados = {vx}, Trenes = {vy}, Ganancia = {Z[i]}")
```

Este bloque recorre los vértices y muestra cuantos soldados x y trenes y se producen en cada vértice, junto con la ganancia correspondiente.

Paso 9: Mostrar el gráfico y la solución óptima.

```
# Mostrar leyenda y gráfico
plt.legend()
plt.grid()
plt.show()
```

Se muestra el gráfico con las restricciones y la región factible sombreada.

Finalmente, el bloque siguiente encuentra el vértice donde se obtiene la ganancia máxima.

```
# Mejor solución
max_profit_index = np.argmax(Z)
print(f"\nGanancia máxima se obtiene en Vértice {max_profit_index+1}:")
print(f"Soldados = {vertices[max_profit_index, 0]}, Trenes = {vertices[max_profit_index, 1]}, Ganancia = {Z[max_profit_index]}")
```

Se utiliza $np.argmax(Z)$ para identificar el índice del vértice con la mayor ganancia.

2.5. Ejercicio 3. GeoGebra.

Un fabricante de tapices dispone de $500\ kg$ de hilo de seda, $400\ kg$ de hilo de plata y $225\ kg$ de hilo de oro. Desea fabricar dos tipos de tapices que llamaremos A y B. para los del tipo A se necesita $1\ kg$ de hilo de seda y $2\ kg$ de hilo de plata, y para los del tipo B, $2\ kg$ de hilo de seda, $1\ kg$ de hilo de plata y $1\ kg$ de hilo de oro. Cada tapiz del tipo A se vende a $2000\ euros$ y cada tapiz del tipo B a $3000\ euros$. Si se vende todo lo que fabrica, ¿Cuántos tapices de cada tipo ha de fabricar para que el beneficio sea máximo y cuál es el beneficio?

2.5.1. Planteamiento del problema.

1. **Variables de decisión:**
   - x : tapiz del tipo A.
   - y : tapiz del tipo B.
2. **Función objetivo:**

   El objetivo es maximizar las ganancias. Sabemos que:
   - $2000\ euros$ para cada tapiz del tipo A.
   - $3000\ euros$ para cada tapiz del tipo B.

   Por lo tanto, la función objetivo a maximizar es:

   $$Z = 2000x + 3000y$$

   Donde Z representa el beneficio total.
3. **Restricciones:**

- **Hilo de seda:**

  Cada tapiz del tipo A utiliza $1\,kg$ de hilo de seda y cada tapiz de tipo B utiliza $2\,kg$ de hilo de seda. Hay $500\,kg$ de hilo de seda disponible, por lo que la restricción será:

  $$x + 2y \leq 500$$

- **Hilo de plata:**

  Cada tapiz del tipo A utiliza $2\,kg$ de hilo de plata y cada tapiz del tipo B utiliza $1\,kg$ de hilo de plata, hay $400\,kg$ de hilo de plata disponible, por lo que la restricción será:

  $$2x + y \leq 400$$

- **Hilo de oro:**

  Cada tapiz del tipo B utiliza $1\,kg$ de hilo de oro. Hay $225\,kg$ de hilo de oro disponible, por lo que la restricción será:

  $$y \leq 225$$

- **Restricciones de no negatividad:**

  Los valores de x y y deben ser no negativos, es decir:

  $$x \geq 0, \qquad y \geq 0$$

4. **Resumen del planteamiento.**

$$Maximizar\colon \quad Z = 2000x + 3000y$$

Sujeta a:

$$x + 2y \leq 500$$
$$2x + y \leq 400$$
$$y \leq 225$$
$$x \geq 0, \qquad y \geq 0$$

2.5.2. Solución Gráfica en GeoGebra.

Paso 1. Introducir las restricciones como ecuaciones de desigualdad.

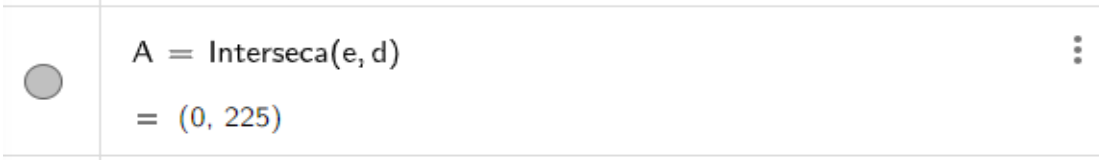
Paso 2: Dibujar las restricciones en el plano cartesiano con x y y como ejes.

Paso 3: Determinar la región factible.

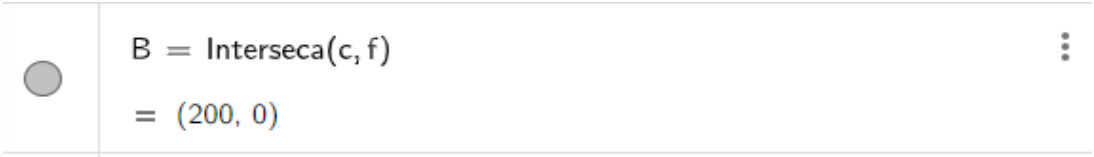
La región factible está delimitada por estas líneas y las restricciones $x \geq 0$. Esta es la región donde se cumplen todas las restricciones.

Paso 4: Identificar los vértices de la región factible: los vértices de la región factible son los puntos de intersección entre las rectas:

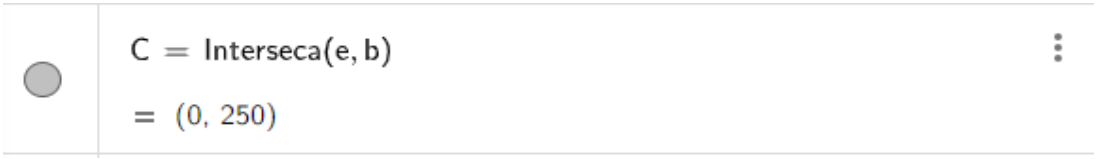
1. **Punto A**: Intersección de $y = 225$ y el eje x. coordenadas $(0, 225)$.



2. **Punto B**: Intersección de $2x + y = 400$ y el eje y. coordenadas $(200, 0)$.



3. **Punto C**: intersección de $x + 2y = 500$ y el eje y. coordenadas $(0, 250)$.



4. **Punto D**: intersección de $x + 2y = 500$ y $2x + y = 400$.
    - Resolver es sistema de ecuaciones:

$$x + 2y = 500$$

$$2x + y = 400$$

- Despejando x de la segunda ecuación:

$$2x + y = 400 \Longrightarrow y = 400 - 2x$$

- Sustituyendo en la primera ecuación:

$$x + 2(400 - 2x) = 500$$

$$x + 800 - 4x = 500$$

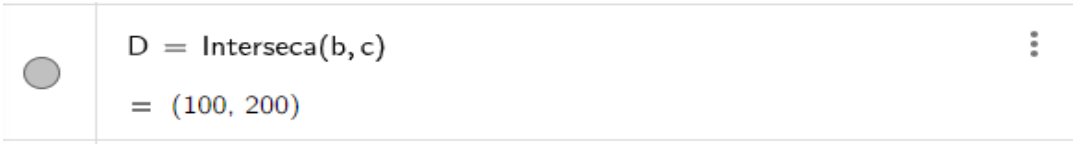
$$-3 + 800 = 500$$

$$-3x = -300 \Longrightarrow x = 100$$

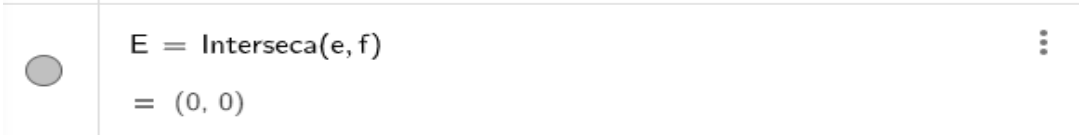
- Sustituyendo $x = 100$ en $y = 400 - 2x$:

$$y = 400 - 2(100) = 200$$

- Coordenadas de $D = (100, 200)$.



5. **Punto E**: intersección de los ejes $x = 0$ y $y = 0$, que da el origen $(0, 0)$.



Paso 5: evaluar la función objetivo en los vértices:

La función objetivo es:

$$Z = 2000x + 3000y$$

Evaluar Z en cada uno de los vértices identificados:

1. En el punto $A(0, 225)$:

$$Z = 2000(0) + 3000(225) = 675000$$

2. En el punto $B(200, 0)$:

$$Z = 2000(200) + 3000(0) = 400000$$

3. En el punto $C(0, 250)$:

$$Z = 2000(0) + 3000(250) = 750000$$

4. En el punto $D(100{,}200)$:

$$Z = 2000(100) + 3000(200) = 800000$$

5. En el punto $E(0, 0)$:

$$Z = 2000(0) + 3000(0) = 0$$

2.6. Ejercicio 4. R.

Laboratorios Novartis desea preparar un medicamento de tal manera que cada frasco contenga al menos 32 unidades de vitamina A, 10 de vitamina B y 40 de vitamina C. para suministrar estas vitaminas, el laboratorio emplea el aditivo $X1$, a un costo de 2 $dólares$ por onza, el cual contiene 15 unidades de la vitamina A, 2 de B y 4 de C, un aditivo $X2$ a un costo de 4 $dólares$ por cada onza, que contiene 4 unidades de vitamina A, 2 de B y 14 de C. ¿Cuántas onzas de cada aditivo se deben incluir en el frasco para minimizar el costo?

2.6.1. Planteamiento del problema.

1. **Variables de decisión:**
   - x: Onza de aditivo $X1$.
   - y: Onza de aditivo $X2$.
2. **Función objetivo:**

El objetivo es minimizar el costo sabemos que:

- $2\ dólares$ por onza.
- $4\ dólares$ por onza.

Por lo tanto, la función objetivo a minimizar es:

$$Minimizar: Z = 2x + 4y$$

Donde Z representa es costo mínimo.

3. **Restricciones:**

- **Vitamina A:**

  Cada frasco debe contener 32 unidades de vitamina A, el laboratorio emplea el aditivo $X1$ que contiene 15 unidades de vitamina A y el aditivo $X2$ que contiene 4 unidades de vitamina A.

$$15x + 4y \geq 32$$

- **Vitamina B:**

  Cada frasco debe contener 10 unidades de vitamina B, el laboratorio emplea el aditivo $X1$ que contiene 2 unidades de vitamina B y el aditivo $X2$ que contiene 2 unidades de vitamina B.

$$2x + 2y \geq 10$$

- **Vitamina C:**

  Cada frasco debe contener 40 unidades de vitamina C, el laboratorio emplea el aditivo $X1$ que contiene 4 unidades de vitamina C y el aditivo $X2$ que contiene 14 unidades de vitamina C.

$$4x + 14y \geq 40$$

- **Restricciones de no negatividad:**

  Los valores de x y y deben ser no negativos, es decir:

$$x \geq 0, \qquad y \geq 0$$

4. **Resumen del planteamiento.**

$$Minimizar\colon\ Z = 2x + 4y$$

Sujeta a:

$$15x + 4y \geq 32$$

$$2x + 2y \geq 10$$

$$4x + 14y \geq 40$$

$$x \geq 0, \qquad y \geq 0$$

2.6.2. Solución en R.

Utilizando Método Gráfico.

Paso 1: Instalar y cargar las librerías necesarias.

```
# Cargar la librería ggplot2
install.packages("ggplot2")
library(ggplot2)
```

Se carga la librería $ggplot2$, que es una herramienta poderosa en R para la visualización de datos, especialmente para crear gráficos elegantes y personalizados.

Paso 2: Definir las funciones de las restricciones.

```
# Definir las funciones de las restricciones en términos de x (X1) y y (X2)
f1 <- function(x) { (32 - 15*x) / 4 }   # Vitamina A
f2 <- function(x) { (10 - 2*x) / 2 }    # Vitamina B
f3 <- function(x) { (40 - 4*x) / 14 }   # Vitamina C
```

Se definen tres funciones que representan las restricciones del problema:

- Vitamina A: La función $f1$ indicá la relación entre la cantidad de aditivo $X1$ (denotado como x) y la cantidad mínima de vitamina A necesaria en el frasco.

- Vitamina B: La función $f2$ establece la relación para la vitamina B.
- Vitamina C: La función $f3$ se refiere a vitamina C.

Paso 3: Crear valores de x para graficar.

```
# Crear valores de x para graficar
x_values <- seq(0, 5, length.out = 400)
```

Se generan 400 valores equidistantes entre 0 y 5 para x (onzas de aditivo $X1$). Esto proporciona una gama continua de valores para graficar.

Paso 4: Calcular valores de y para cada x según las restricciones.

```
# Calcular valores de y para cada x según las restricciones
y_f1 <- sapply(x_values, f1)
y_f2 <- sapply(x_values, f2)
y_f3 <- sapply(x_values, f3)
```

Se utiliza $sapply$ para aplicar cada función de restricción a todos los valores de x y calcular el correspondiente valor de y (onzas de aditivo $X2$).

Paso 5: Limitar valores negativos en las restricciones.

```
# Limitar valores negativos en las restricciones
y_f1[y_f1 < 0] <- NA
y_f2[y_f2 < 0] <- NA
y_f3[y_f3 < 0] <- NA
```

Se remplazan los valores negativos de y por NA, ya que no son validos (no se puede tener un valor negativo de onzas de aditivos).

Paso 6: Crear el $Data\ Frame$ con los resultados.

```
# Crear el gráfico con las restricciones
df <- data.frame(x = x_values, f1 = y_f1, f2 = y_f2, f3 = y_f3)
```

Se crea un $data\ frame\ df$ que contiene los valores de x y las tres funciones de restricciones. Esto facilita el manejo y visualización de los datos.

Paso 7: Definir el punto óptimo y su costo mínimo.

```
# Definir el punto óptimo
x_val <- 3
y_val <- 2
min_cost <- 2 * x_val + 4 * y_val
```

Se especifica el punto óptimo $(x = 3, y = 2)$, calculado previamente fuera del gráfico, y se determina su costo mínimo usando la función objetivo: $2x1 + 4x2$.

Paso 8: Crear un $data.frame$ para el punto óptimo.

```
# Crear un dataframe para el punto óptimo
optimo <- data.frame(x_val = x_val, y_val = y_val)
```

$data.frame$ llamado óptimo almacena el punto óptimo, que luego se graficara como un punto especifico en el gráfico.

Paso 9: Graficar las restricciones y sombrear el área factible.

```
# Graficar las restricciones y sombrear el área factible
ggplot(df, aes(x = x)) +
  geom_line(aes(y = f1), color = "darkorange1", linewidth = 1, linetype = "solid") +
  geom_line(aes(y = f2), color = "red", linewidth = 1, linetype = "solid") +
  geom_line(aes(y = f3), color = "green2", linewidth = 1, linetype = "solid") +

  # Sombreado de la región factible
  geom_ribbon(aes(ymin = pmax(f1, f2, f3, na.rm = TRUE), ymax = 10), fill = "cadetblue", alpha = 0.5) +
  geom_point(data = optimo, aes(x = x_val, y = y_val), color = "purple", size = 4) +  # Punto óptimo
  annotate("text", x = x_val, y = y_val + 0.5, label = "Punto óptimo (3, 2)", color = "darkorchid4", size = 5, hjust = 0.5) +

  # Ajustes estéticos
  coord_cartesian(xlim = c(0, 5), ylim = c(0, 10)) +
  labs(x = "Aditivo X1", y = "Aditivo X2") +
  ggtitle("Minimización del Costo de Aditivos") +
  theme_minimal(base_size = 14) +
  theme(plot.title = element_text(hjust = 0.5),
        panel.grid.minor = element_blank())
```

Se utiliza $ggplot$ para crear un gráfico que muestras las líneas de las restricciones.

- $geom\_line$ se usa para graficar las líneas de las restricciones en diferentes colores y estilos.
- $geom\_ribbon$ se utiliza para sombrear la región factible (el área donde se cumplen todas las restricciones)

- $labs\ y\ ggtitle$ añaden etiquetas y un título al gráfico.
- $theme\_minimal$ y otros ajustes estéticos mejoran la apariencia del gráfico.

Paso 10: Mostrar los resultados.

```
# Mostrar los resultados
cat("El punto óptimo es:\n")
cat("X1 =", x_val, "\n")
cat("X2 =", y_val, "\n")
cat("Costo mínimo =", min_cost, "$\n")
```

Finalmente, se imprimen los vértices junto con sus costos para analizar cuál combinación de $x1$ y $x2$ resulta en el costo mínimo.

2.7. Aplicación del Método Simplex con tres variables.

2.7.1. Ejercicio 5.

Una empresa fábrica tres productos A, B y C. la empresa tiene un máximo de 100 horas de trabajo disponible por día, tiene un máximo de 80 unidades de materia prima disponible por día y tiene un máximo de 60 unidades de espacio de almacenamiento disponibles por día. La empresa desea maximizar sus ganancias mediante la producción de estos productos. Las ganancias por cada producto son de $\$\ 3, \$\ 2$ y $\$\ 4$.

2.7.1.1. Planteamiento del problema.

1. **Variables de decisión:**
   - x_A: Número de unidades producidas del producto A.
   - x_B: Número de unidades producidas del producto B.
   - x_C: Número de unidades producidas del producto C.
2. **Función objetivo:**

   El objetivo es maximizar las ganancias de la producción.

$$Maximizar:\ Z = 3x_A + 2x_B + 4x_C$$

3. **Restricciones:**

- **Horas de trabajo disponibles:** La suma total de horas de trabajo requeridas por los productos, no pueden exceder 100 horas.

$$x_A + x_B + x_C \leq 100$$

- **Materia prima disponible:** La cantidad total de materia prima utilizada no puede exceder 80 unidades.

$$2x_A + x_B + 3x_C \leq 80$$

- **Espacio de almacenamiento disponible:** La cantidad total de espacio utilizado no puede exceder 60 unidades.

$$x_A + 2x_B + x_C \leq 60$$

- **Restricciones de no negatividad:** No se puede producir cantidades negativas de productos.

$$x_A \geq 0, \qquad x_B \geq 0, \qquad x_C \geq 0$$

4. **Resumen del planteamiento.**

$$Maximizar\colon\ Z = 3x_A + 2x_B + 4x_C$$

Sujeta a:

$$x_A + x_B + x_C \leq 100$$
$$2x_A + x_B + 3x_C \leq 80$$
$$x_A + 2x_B + x_C \leq 60$$
$$x_A \geq 0, \qquad x_B \geq 0, \qquad x_C \geq 0$$

2.7.2. Ejercicio 5. Solución en Scilab.

Utilizando el Método simplex.

Paso 1: Definir los coeficientes de los productos.

```
// Definir los coeficientes de los productos A, B y C (para maximizar)
cost_A = 3; // Beneficio por unidad de A
cost_B = 2; // Beneficio por unidad de B
cost_C = 4; // Beneficio por unidad de C
```

El código define las ganancias por unidad de cada uno de los tres productos.

Paso 2: Definir la matriz de coeficientes de restricciones.

```
// Definir la matriz de coeficientes de las restricciones
A = [1 1 1;  // Restricción 1: horas de trabajo
     2 1 3;  // Restricción 2: materia prima
     1 2 1]; // Restricción 3: espacio de almacenamiento
```

La matriz A contiene los coeficientes de las variables A, B y C en las restricciones. Cada fila representa una restricción:

- La primera fila es para las horas de trabajo, donde la cantidad de producto consume un número determinado de horas.
- La segunda fila es para la materia prima, donde cada producto utiliza una cantidad de materia prima.
- La tercera fila es para el espacio de almacenamiento, donde cada producto requiere espacio de almacenamiento.

Paso 3: Definir los términos independientes.

```
// Definir el vector de términos independientes
b = [100; 80; 60]; // Recursos disponibles
```

Este vector b define los recursos disponibles para cada restricción.

- 100 horas de trabajo.
- 80 unidades de materia prima.
- 60 unidades de espacio de almacenamiento.

Paso 4: Definir la función objetivo.

```
// Definir los coeficientes de la función objetivo (negados para maximizar)
c = [-cost_A; -cost_B; -cost_C; 0; 0; 0]; // Los tres ceros son para las variables de holgura
```

El vector c representa los coeficientes de la función objetivo, pero negados. Esto se hace porque el algoritmo simplex esta diseñado para trabajar con problemas de minimización. Si se niegan los coeficientes, se puede convertir un problema de maximización en uno de minimización. Se agregan tres ceros al final para las variables de holgura.

Paso 5: Añadir variables de holgura a las restricciones.

```
// Añadir variables de holgura a las restricciones
A = [A, eye(3, 3)]; // Añadir una matriz identidad de 3x3 para las variables de holgura
```

Se agregan las variables de holgura a la matriz A para convertir las restricciones en igualdades. las variables de holgura representan la cantidad de recursos no utilizados en cada restricción.

Paso 6: Construir la tabla inicial del Simplex.

```
// Crear la tabla inicial del simplex
tableau = [A, b; c', 0];
```

Se construye la tabla inicial del Simplex. Las primeras filas contienen los coeficientes de las restricciones (con las variables de holgura añadidas). La última fila contiene los coeficientes de la función objetivo (negados), y un valor de cero al final (para representar la constante de la función objetivo).

Paso 7: Mostrar la tabla inicial.

```
// Mostrar la tabla inicial
disp("Tabla inicial:");
disp(tableau);
```

Se imprime la tabla inicial, mostrando los coeficientes de las restricciones, los coeficientes de la función objetivo y los términos independientes.

Paso 8: Encontrar la columna y la fila pivote.

```
// Función para encontrar la columna pivote (mínimo costo negativo)
function col_pivot = find_col_pivot(tableau)
    [min_val, col_pivot] = min(tableau($, 1:$-1)); // Buscar la columna con el costo más negativo
endfunction

// Función para encontrar la fila pivote (mínima razón)
function row_pivot = find_row_pivot(tableau, col_pivot)
    ratios = tableau(1:$-1, $) ./ tableau(1:$-1, col_pivot); // Calcular las razones
    ratios(ratios <= 0) = %inf; // Ignorar ratios no positivos
    [min_val, row_pivot] = min(ratios); // Encontrar la fila con la razón mínima
endfunction
```

1. La columna pivote, es la columna que tiene el mínimo valor negativo en la fila de la función objetivo (los coeficientes de la función objetivo). Esta columna es la que indica cual variable entra en la base.
2. La función $find\_row\_pivot$ calcula las razones entre los términos independientes y los coeficientes de la columna pivote. La fila pivote se selecciona como la que tiene la razón mínima positiva, que indica que variable básica debe salir de la base.

Paso 9: Iteración del Algoritmo Simplex.

```
// Iterar hasta que no haya más columnas pivote negativas
while sum(tableau($, 1:$-1) < 0) > 0
    col_pivot = find_col_pivot(tableau);  // Encontramos la columna pivote
    row_pivot = find_row_pivot(tableau, col_pivot); // Encontramos la fila pivote
    
    // Actualizar la tabla usando el elemento pivote
    pivot = tableau(row_pivot, col_pivot);
    tableau(row_pivot, :) = tableau(row_pivot, :) / pivot; // Normalizar fila pivote
    
    // Actualizar el resto de las filas
    for i = 1:size(tableau, 1)
        if i <> row_pivot then
            tableau(i, :) = tableau(i, :) - tableau(i, col_pivot) * tableau(row_pivot, :);
        end
    end

    // Mostrar la tabla después de cada iteración
    disp("Tabla intermedia:");
    disp(tableau);
end
```

Aquí realiza las iteraciones del algoritmo simplex. Mientras haya coeficientes negativos en la fila de la función objetivo, el algoritmo sigue iterando.

Paso 10: Mostrar la tabla final y calcular la ganancia máxima.

```
// Mostrar la tabla final
disp("Tabla final:");
disp(tableau);

// Costo máximo (es el valor en la última celda)
max_cost = tableau($,$);
disp("Ganancia máxima: " + string(max_cost));
```

Después que el algoritmo Simplex termina, se imprime la tabla final. El valor de la ganancia máxima se encuentra en la última celda de la tabla, que es el valor de la función objetivo.

Paso 11: Extraer las soluciones óptimas.

```
// Inicializar el vector de soluciones
x_values = zeros(3, 1);

// Identificar las columnas correspondientes a las variables básicas (A, B y C)
for i = 1:3
    columna = tableau(1:$-1, i);
    if sum(columna == 1) == 1 && sum(columna == 0) == (size(tableau, 1) - 2)
        fila = find(columna == 1);
        x_values(i) = tableau(fila, $); // Extraemos el valor de la última columna (solución)
    end
end

// Mostrar las cantidades de productos A, B y C
disp("Cantidad de Producto A: " + string(x_values(1)));
disp("Cantidad de Producto B: " + string(x_values(2)));
disp("Cantidad de Producto C: " + string(x_values(3)));
```

Se extrae las cantidades óptimas de los productos A, B y C de la tabla final. Las columnas que contienen un único 1 en su columna corresponden a las variables básicas, y los valores de esas filas en la última columna son las cantidades óptimas.

2.7.3. Ejercicio 5. Solución en Python.

Utilizando el Método Simplex.

Paso 1: Instalar librerías.

- Abrir Terminal en el menú de inicio.
- Escribir el siguiente comando y presionar Enter.

```
TERMINAL    PORTS    JUPYTER
\programas tesis> pip install scipy
```

Importar librerías.

```
import numpy as np
from scipy.optimize import linprog
```

Usar $numpy$ para manejar matrices y $linprog$ de $scipy.optimize$ para resolver problemas de optimización lineal.

Paso 2: Definir el problema de optimización lineal

1. **Definir la matriz de coeficientes de las restricciones.**

```
# Definir la matriz de coeficientes de las restricciones
A = np.array([
    [1, 1, 1],  # Restricción de horas de trabajo
    [2, 1, 3],  # Restricción de materia prima
    [1, 2, 1]   # Restricción de espacio de almacenamiento
])
```

La matriz A representa las restricciones de los recursos disponibles, cada fila corresponde a una restricción.

- La primera fila indica que el producto $A, B\ y\ C$ consumen cada uno una unidad de “horas de trabajo”.
- La segunda fila representa el uso de “materia prima”.
- La tercera fila representa el uso de “espacio de almacenamiento”.

Cada columna representa la cantidad que cada producto $(A, B\ o\ C)$ utiliza de cada recurso.

2. **Definir el vector de términos independientes.**

```
# Definir el vector de términos independientes (recursos disponibles)
b = np.array([100, 80, 60])
```

El vector b indica la cantidad total de recursos disponibles.

- 100 horas de trabajo disponible por día.
- 80 unidades de materia prima disponibles.

- 60 unidades de espacio de almacenamiento.

3. **Definir el vector de la función objetivo.**

```
# Definir la función objetivo (negativa para maximización)
c = np.array([-3, -2, -4])
```

La función objetivo es la ganancia obtenida al producir productos $A, B\ y\ C$. los valores son negativos por que $linprog$ resuelve problemas de minimización, al usar valores negativos linprog maximizará la ganancia.

- La ganancia del producto A es 3.
- La ganancia del producto B es 2.
- La ganancia del producto C es 4.

Paso 3: Resolver el problema con linprog.

```
# Resolver el problema usando el método HiGHS
res = linprog(c, A_ub=A, b_ub=b, method='highs')
```

Resolver el problema de maximización utilizando $linprog$, especificando:

- c como la función objetivo.
- A_ub y b_ub como la matriz de restricciones y los recursos disponibles.
- $method = 'highs'$ para usar un método eficiente de optimización.

Paso 4: Mostrar la configuración inicial del problema.

Imprimir la configuración inicial

```
# Mostrar la tabla inicial
print("Tabla inicial del simplex:")
print("Coeficientes de las restricciones (A):")
print(A)
print("Recursos disponibles (b):")
print(b)
print("Función objetivo (c):")
print(c)
```

- A: matriz de coeficientes de restricciones.
- b: vector de recursos disponibles.

- c: vector de coeficientes de la función objetivo.

Paso 5: Crear funciones auxiliares para el Método Simplex.

1. **Función para encontrar la columna pivote.**

```python
# Función para encontrar la columna pivote
def encontrar_columna_pivote(tabla):
    return np.argmin(tabla[-1, :-1])
```

Localiza el índice de la columna pivote en la última fila (función objetivo) buscando el valor más negativo, esto indica la variable con mayor potencial de incremento en el objetivo.

2. **Función para encontrar la fila pivote.**

```python
# Función para encontrar la fila pivote
def encontrar_fila_pivote(tabla, col_pivote):
    filas = tabla[:-1, -1] / tabla[:-1, col_pivote]
    filas[filas <= 0] = np.inf
    return np.argmin(filas)
```

Esta función encuentra la fila pivote dividiendo el valor de la última columna (los recursos disponibles) entre el valor de la columna pivote para cada fila, solo se consideran cocientes positivos (las filas con cocientes negativos se descartan usando $np.inf$) la fila pivote corresponde al menor cociente positivo.

Paso 6: Implementar el Método Simplex.

Definir la función simplex.

```python
# Iterar hasta que no haya más columnas pivote negativas
def simplex(tabla):
    while np.any(tabla[-1, :-1] < 0):
        col_pivote = encontrar_columna_pivote(tabla)
        fila_pivote = encontrar_fila_pivote(tabla, col_pivote)
        pivote = tabla[fila_pivote, col_pivote]
        tabla[fila_pivote, :] = tabla[fila_pivote, :] / pivote
        for i in range(len(tabla)):
            if i != fila_pivote:
                tabla[i, :] -= tabla[i, col_pivote] * tabla[fila_pivote, :]
        print("\nTabla actualizada:")
        print(tabla)
    return tabla
```

La función $simplex$ ejecuta iterativamente el Método Simplex hasta que no haya valores negativos en la última fila de la tabla.

- Seleccionar columna y fila pivote.
- Actualizar la fila pivote dividiendo por el valor pivote.
- Operaciones de fila para hacer ceros en la columna pivote en todas las demás filas.

Crear la tabla inicial del Simplex

```
# Crear la tabla inicial del simplex
tabla_inicial = np.hstack([A, b.reshape(-1, 1)]).astype(float)
tabla_inicial = np.vstack([tabla_inicial, np.hstack([c, 0])])
```

$tabla\_inicial$ se crea agregando b como la última columna y c como la última fila, convirtiéndose en la tabla de entrada para el Método Simplex.

Paso 7: Ejecutar el Método Simplex.

```
# Ejecutar el método simplex
tabla_final = simplex(tabla_inicial)
```

Llamar a la función $simplex$ con la tabla inicial, la función muestra la tabla actualizada en cada iteración.

Paso 8: Mostramos los resultados finales.

1. **Ganancia máxima:**

```
# Mostrar los resultados
print("Ganancia máxima:", -res.fun)
```

$res.fun$ es el valor mínimo encontrado por $linprog$, por lo que invierte su signo para obtener la ganancia máxima.

2. **Cantidad óptima de cada producto:**

```
print("Cantidad de productos A, B y C a producir:", res.x)
```

$res.x$ contiene las cantidades óptimas de cada producto $(A, B\ y\ C)$ que maximizan las ganancias bajo las restricciones dadas.

2.7.4. Ejercicio 5. Solución en R.

Utilizando el Método Simplex.

Paso 1: Instalar la paquetería lpSolve.

Para instalar y cargar esta biblioteca en R, usar los siguientes comandos:

```
install.packages("lpSolve")
library(lpSolve)
```

La biblioteca $lpSolve$ está diseñada específicamente para resolver problemas de programación lineal y programación lineal entera.

Paso 2: Definir los datos del problema.

1. **Matriz de restricciones:**

```
# Definir la matriz de coeficientes de las restricciones
A <- matrix(c(1, 1, 1,  # Restricción de horas de trabajo
              2, 1, 3,  # Restricción de materia prima
              1, 2, 1), # Restricción de espacio de almacenamiento
            nrow = 3, byrow = TRUE)
```

   La matriz A recoge información sobre como cada producto $(A, B\ y\ C)$ consume recursos en el proceso de fabricación, las filas representan las restricciones de recursos y las columnas representan los productos.

2. **Definir el vector de recursos disponibles:**

```
# Definir el vector de términos independientes (recursos disponibles)
b <- c(100, 80, 60)
```

   El vector b contiene los recursos máximos disponibles para cada restricción.

   - 100 horas de trabajo.
   - 80 unidades de materia prima.
   - 60 unidades de espacio de almacenamiento.

3. **Definir el vector de coeficientes de la función objetivo:**

```
# Definir la función objetivo (positiva para maximización)
c <- c(3, 2, 4)
```

   c define la ganancia de cada producto.

   - $\$3$ por unidad de A.
   - $\$2$ por unidad de B.

- $\$4$ por unidad de C.

Paso 3: Crear la tabla inicial del Simplex.

```
# Crear la tabla inicial del simplex
simplex_table <- function(A, b, c) {
  tableau <- cbind(A, diag(nrow(A)), b)
  tableau <- rbind(tableau, c(-c, rep(0, nrow(A) + 1)))
  return(tableau)
}

# Mostrar la tabla inicial
initial_tableau <- simplex_table(A, b, c)
```

La función $simplex\_table$ es la que crea la tabla inicial del Método Simplex, que combina las restricciones y la función objetivo en una sola matriz.

- $cbind$: Añade las variables de holgura y los valores del lado derecho (b) a A.
- $rbind$: Añade la fila de la función objetivo al final de la tabla, usando $-c$ para invertir los signos preparando la tabla para maximizar.

Paso 4: Encontrar la columna pivote.

```
# Función para encontrar la columna pivote
pivot_column <- function(tableau) {
  return(which.min(tableau[nrow(tableau), 1:(ncol(tableau) - 1)]))
}
```

La función $pivot\_column$ encuentra la columna pivote, que es la más negativa en la fila de la función objetivo (última fila). Este paso es fundamental para seleccionar la variable de entrada en cada iteración.

Paso 5: Encontrar la fila pivote.

```
# Función para encontrar la fila pivote
pivot_row <- function(tableau, pivot_col) {
  ratios <- tableau[1:(nrow(tableau) - 1), ncol(tableau)] / tableau[1:(nrow(tableau) - 1), pivot_col]
  ratios[ratios <= 0] <- Inf
  return(which.min(ratios))
}
```

La función $pivot\_row$ determina la fila pivote dividiendo los valores de recursos disponibles (b) entre los coeficientes de la columna pivote para encontrar el mínimo positivo, aplicando la regla del cociente mínimo. La fila pivote seleccionada indica qué variable dejará la base.

Paso 6: Función principal del Método Simplex.

```
# Iterar hasta que no haya más columnas pivote negativas
simplex <- function(tableau) {
  while (any(tableau[nrow(tableau), 1:(ncol(tableau) - 1)] < 0)) {
    pivot_col <- pivot_column(tableau)
    pivot_row <- pivot_row(tableau, pivot_col)
    pivot_element <- tableau[pivot_row, pivot_col]

    tableau[pivot_row, ] <- tableau[pivot_row, ] / pivot_element
    for (i in 1:nrow(tableau)) {
      if (i != pivot_row) {
        tableau[i, ] <- tableau[i, ] - tableau[i, pivot_col] * tableau[pivot_row, ]
      }
    }

    print(paste("Tabla después de iteración con pivote en columna", pivot_col, "y fila", pivot_row, ":"))
    print(tableau)
  }
  return(tableau)
}
```

La función $simplex$ realiza iteraciones hasta que todos los elementos en la fila de la función objetivo sean no negativos, lo cual indica que se ha encontrado la solución.

Paso 7: Ejecutar el Método Simplex y obtenemos la ganancia máxima.

```
# Ejecutar el método simplex
final_tableau <- simplex(initial_tableau)

# Mostrar los resultados finales
print("Tabla final del simplex:")
print(final_tableau)
ganancia_maxima <- final_tableau[nrow(final_tableau), ncol(final_tableau)]
cat("Ganancia máxima:", ganancia_maxima, "$\n")
```

Una vez completada las iteraciones, podemos leer la ganancia máxima en la última celda de la última fila de la tabla.

Paso 8: Calcular las cantidades de productos A, B y C.

```
# Calcular las cantidades de productos A, B y C
num_variables <- length(c)
solucion <- rep(0, num_variables)
for (i in 1:num_variables) {
  columna <- final_tableau[1:(nrow(final_tableau) - 1), i]
  if (sum(columna == 1) == 1 && sum(columna == 0) == (nrow(final_tableau) - 2)) {
    fila <- which(columna == 1)
    solucion[i] <- final_tableau[fila, ncol(final_tableau)]
  }
}

cat("Cantidad de productos A, B y C a producir:", round(solucion, 4), "\n")
```

Para encontrar las cantidades de cada producto en esta parte del código se buscan columnas que contengan una sola entrada de 1 y el resto 0.

2.8. Aplicación del Método Gráfico.

2.8.1. Ejercicio 6.

Una compañía de auditores se especializa en preparar liquidaciones y auditorías de empresas pequeñas. Tiene interés en saber cuántas auditorías y liquidaciones pueden realizar mensualmente para maximizar sus ingresos. Se dispone de 800 horas de trabajo directo y 320 horas para revisión. Una auditoría en promedio requiere de 40 horas de trabajo directo y 10 horas de revisión, además aporta un ingreso de $300\,\$$. Una liquidación de impuestos requiere de 8 horas de trabajo directo y de 5 horas de revisión produce un ingreso de $100\,\$$. El máximo de las liquidaciones mensuales disponibles es de 60. Maximizar el ingreso total

2.8.1.1. Planteamiento del problema.

1. **Variables de decisión:**
   - x: Número de auditorías a realizar.
   - y: Número de liquidaciones a realizar.
2. **Función objetivo:**
   La compañía quiere maximizar el ingreso total.
   - Cada auditoría genera $300\,\$$.
   - Cada liquidación genera $100\,\$$.

$$Maximizar: \quad Z = 300x + 100y$$

3. **Restricciones:**
   - **Trabajo directo:** Cada auditoria requiere 40 horas de trabajo directo, cada liquidación requiere 8 horas de trabajo directo, la compañía dispone de un total de 800 horas de trabajo directo al mes.

$$40x + 8y \leq 800$$

- **Horas de revisión:** Cada auditoría requiere de 10 horas de revisión, cada liquidación requiere 5 horas de revisión, la compañía dispone de un total de 320 horas de revisión al mes.

$$10x + 5y \leq 320$$

- **Máximo de liquidaciones:** La compañía puede realizar un máximo de 60 liquidaciones al mes.

$$y \leq 60$$

- **Restricciones de no negatividad:** Las variables de decisión x y y deben ser mayores o iguales a 0, ya que no se pueden hacer auditorías o liquidaciones negativas.

$$x \geq 0, \qquad y \geq 0$$

4. **Resumen del planteamiento.**

$$Maximizar\colon\ Z = 300x + 100y$$

Sujeta a:

$$40x + 8y \leq 800$$
$$10x + 5y \leq 320$$
$$y \leq 60$$
$$x \geq 0, \qquad y \geq 0$$

2.8.2. Ejercicio 6. Solución en Scilab.

Utilizando el Método Gráfico.

Paso 1: Definir las variables.

```
//·Definición·de·variables
income_audit·=·300;····//·Ingreso·por·auditoría
income_settlement·=·100;·//·Ingreso·por·liquidación

//·Requerimientos·de·horas
hours_audit·=·40;·····//·Horas·requeridas·para·una·auditoría
hours_settlement·=·8;·//·Horas·requeridas·para·una·liquidación

//·Disponibilidad·de·horas
available_hours_direct·=·800;··//·Total·de·horas·de·trabajo·directo·disponibles
available_hours_review·=·320;···//·Total·de·horas·de·revisión·disponibles

//·Límite·de·liquidaciones
max_settlements·=·60;·//·Máximo·de·liquidaciones·mensuales
```

Estas lineal inicializan las variables que contienen los datos del problema: ingresos por cada tipo de servicio, horas requeridas por cada uno, el total de horas disponibles y el máximo de liquidaciones mensuales.

Paso 2: Definir el rango para las variables de decisión.

```
// Definición del rango para las variables de decisión
x_audits = 0:0.1:30; // Cantidad de auditorías (de 0 a 30)
```

Definir un rango de posibles valores para las auditorías (x_audits), de 0 a 30 en incrementos de 0.1. Esto permitirá construir la gráfica evaluando el número de liquidaciones (y) en función del número de auditorías.

Paso 3: Calcular las restricciones para y (liquidaciones).

```
// Cálculo de y según las restricciones
y1 = (available_hours_direct - hours_audit * x_audits) / hours_settlement; // Restricción de horas de trabajo directo
y2 = (available_hours_review - 10 * x_audits) / 5; // Restricción de horas de revisión
y3 = max_settlements * ones(x_audits); // Restricción de máximo de 60 liquidaciones
```

Calcular $y1$, $y2$ y $y3$, que representaran las restricciones de las horas de trabajo directo, horas de revisión y el máximo de liquidaciones.

- $y1$ es la cantidad máxima de liquidación permitidas según las horas de trabajo directo.
- $y2$ es la cantidad máxima de liquidaciones permitidas según las horas de revisión.
- $y3$ es una restricción fija para límite de 60 liquidaciones.

Paso 4: Limitar valores negativos de y.

```
// Limitar los valores negativos de y para representar solo la región factible
y1(y1 < 0) = 0;
y2(y2 < 0) = 0;
```

Ajustar $y1$ y $y2$ para que no tomen valores negativos, ya que y representan la cantidad de liquidaciones, que no puede ser negativa.

Paso 5: Graficar las restricciones.

```
// Graficar las restricciones
clf; // Limpiar la figura
plot(x_audits, y1, 'b','thickness',2);
plot( x_audits, y2, 'r','thickness',2);
plot(x_audits, y3, 'g--','thickness',2);
xlabel("Número de Auditorías (x)");
ylabel("Número de Liquidaciones (y)");
title("Método Gráfico para Maximizar Ingresos");
legend(["$40x_1 + 8x_2 \leq 800$", "$10x_1 + 5x_2 \leq 320$", "$x_2 \leq 60$"], "location", "northwest");
xgrid();
```

Estas líneas crean la gráfica de las restricciones

- $y1$ se grafica en azul para la restricción de horas de trabajo directo.
- $y2$ se grafica en rojo para la restricción de horas de revisión.
- $y3$ se grafica en verde (con línea descontinua) para el límite de 60 liquidaciones.
- Los ejes y el título también se etiquetan para mayor claridad, y se muestra una leyenda para identificar cada restricción.

Paso 6: Rellenar el área factible.

```
// Rellenar el área factible
x_fill = [0, 0, x_audits(find(x_audits >= 0 & x_audits <= 30))];
y_fill = [0, 60, min(min(y1(find(x_audits >= 0 & x_audits <= 30)), y2(find(x_audits >= 0 & x_audits <= 30))),
y3(find(x_audits >= 0 & x_audits <= 30)))];
xfpoly(x_fill, y_fill, color(144, 238, 144));
```

Esta sección crea una región sombreada que representa el área factible (o región de soluciones posibles) en la gráfica.

Paso 7: Calcular vértices y función objetivo.

```
// Calcular los vértices de la región factible y la función objetivo
vertices = [0, 0; 0, 60; 12, 40; 20, 0]; // Vértices de la región factible
Z = income_audit * vertices(:, 1) + income_settlement * vertices(:, 2); // Calcular Z = 300*x1 + 100*x2
```

Los vértices de esta región factible se definen, y luego se calcula el ingreso (z) en cada uno. Se calculan los puntos de intersección de las restricciones (vértices) y se evalúa la función de ingreso en cada uno para identificar el valor máximo.

Paso 8: Imprimir los resultados en la consola.

```
// Mostrar los resultados en la consola
for i = 1:size(vertices, 1)
    mprintf("Vértice %d: Auditorías = %d, Liquidaciones = %d, Ingreso = $%d\n", i, vertices(i, 1), vertices(i, 2), Z(i));
end
```

Este paso permite ver el ingreso potencial en cada combinación posible dentro de la región factible.

Paso 9: Encontrar y Mostrar el ingreso máximo.

```
// Encontrar el vértice con ingreso máximo
[max_profit, max_index] = max(Z);
optimal_audits = vertices(max_index, 1);
optimal_settlements = vertices(max_index, 2);

mprintf("\nIngreso máximo se obtiene en el Vértice %d:\n", max_index);
mprintf("Auditorías = %d, Liquidaciones = %d, Ingreso = $%d\n", optimal_audits, optimal_settlements, max_profit);
```

El código identifica el vértice que maximiza el ingreso total (z) y muestra la solución óptima en la consola, indicando la cantidad ideal de auditorías y liquidaciones.

Paso 10: Marcar el punto óptimo en la gráfica.

```
// Marcar el punto óptimo en la gráfica
plot(optimal_audits, optimal_settlements, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'y');
xstring(optimal_audits + 0.5, optimal_settlements, sprintf("(%d, %d)", optimal_audits, optimal_settlements), 1);
```

Finalmente, el punto óptimo se marca en la gráfica y se etiqueta, esto permite visualizar el punto que maximiza el ingreso en la gráfica.

2.8.3. Ejercicio 6. Solución en Python.

Utilizando el Método Gráfico

Paso 1: Importación de bibliotecas.

```
import numpy as np
import matplotlib.pyplot as plt
```

Se importan $numpy$ para el manejo de $arrays$ y operaciones matemáticas, y $matplotlib.pyplot$ para la creación de gráficos.

Paso 2: Definición de restricciones.

```
# Definición de las restricciones
x = np.linspace(0, 30, 400)
```

Se define el rango de valores de x (cantidades de auditorías), que va de 0 a 30 con 400 puntos para un trazo suaves de las curvas de restricción.

1. **Restricción del problema.**

```
# Restricciones
y1 = (800 - 40 * x) / 8   # Restricción de trabajo directo
y2 = (320 - 10 * x) / 5   # Restricción de horas de revisión
y3 = np.full_like(x, 60)   # Restricción de máximo 60 liquidaciones
```

Las restricciones de recursos y límites del problema se formulan como:

- $y1$: Cantidad de liquidaciones en función de x, dada por la restricción de horas de trabajo.
- $y2$: Cantidad de liquidaciones en función de x, derivada de la restricción de horas de revisión.
- $y3$: Restricción de máximo 60 liquidaciones, constantes.

Paso 3: Graficación de las restricciones.

```
# Graficar las restricciones
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label=r'$40x_1 + 8x_2 \leq 800$', color='blue')
plt.plot(x, y2, label=r'$10x_1 + 5x_2 \leq 320$', color='red')
plt.plot(x, y3, label=r'$x_2 \leq 60$', color='green', linestyle='--')
```

Cada restricción se grafica con un color en específico y una etiqueta de ecuación:

- $y1$ en azul representa el límite por horas de trabajo directo.
- $y2$ en rojo representa el límite por hora de revisión.
- $y3$ en verde con línea discontinua representa el máximo de 60 liquidaciones.

Paso 4: Configuración del gráfico.

```
# Configurar límites del gráfico
plt.xlim(0, 30)
plt.ylim(0, 70)
plt.xlabel('Número de Auditorías ($x_1$)')
plt.ylabel('Número de Liquidaciones ($x_2$)')
plt.title('Método Gráfico para Maximizar Ingresos')
```

Se establece límites en el eje x y y para enfocar la región de interés. Los ejes y el titulo se etiqueta para mayor claridad.

Relleno de la región factible.

```
plt.fill_between(x, 0, np.minimum(np.minimum(y1, y2), y3), where=(x <= 30), color='lawngreen', alpha=0.3)
```

La región factible que es la zona en la que todas las restricciones se cumplen.

Paso 5: Vértices de la región factible.

```
# Vértices de la región factible
vertices = np.array([[0, 0], [0, 60], [12, 40], [20, 0]])
```

Se definen los vértices de la región factible, que son los puntos donde las restricciones se cruzan y que representan soluciones posibles bajo las restricciones del problema.

Paso 6: Cálculo de la función objetivo en el vértice.

```
# Calcular la función objetivo de los vértices (Z = 300x_1 + 100x_2)
Z = 300 * vertices[:, 0] + 100 * vertices[:, 1]
```

Se calcula el ingreso para cada vértice usando la función objetivo.

Imprimir los resultados.

```
# Imprimir los resultados
for i, (vx, vy) in enumerate(vertices):
    print(f"Vértice {i+1}: Auditorías = {vx}, Liquidaciones = {vy}, Ingreso = {Z[i]}")
```

Se imprime el número de auditorías y liquidaciones para cada vértice, junto con el ingreso correspondiente.

Paso 7: Identificación del ingreso máximo.

```
# Solución: encontrar el máximo ingreso
max_profit_index = np.argmax(Z)
print(f"\nIngreso máximo se obtiene en el Vértice {max_profit_index+1}:")
print(f"Auditorías = {vertices[max_profit_index, 0]}, Liquidaciones = {vertices[max_profit_index,1]}, Ingreso  = $ {Z[max_profit_index]}")
```

Se utiliza $np.argmax$ para encontrar el vértice con el ingreso máximo, junto con las cantidades de auditorías y liquidaciones óptimas.

Paso 8: Marcado del punto óptimo en el gráfico.

```
# Marcar el punto óptimo en el gráfico
plt.plot(x_opt, y_opt, 'bo', label='Punto Óptimo')
plt.text(x_opt + 0.5, y_opt, f'({x_opt}, {y_opt})', fontsize=12, color='magenta')
```

Paso 9: Mostrar leyenda y etiquetas.

```
# Mostrar leyenda y gráfico
plt.legend()
plt.grid()
plt.show()
```

Se muestra el gráfico con una leyenda, que facilita la interpretación de cada curva de restricción y de la región factible.

2.8.4. Ejercicio 6. Solución Gráfica en GeoGebra.

Utilizando el Método Gráfico.

Paso 1: Introducir las restricciones como ecuaciones de desigualdad.

●	a : $40\,x + 8\,y \leq 800 \land 10\,x + 5\,y \leq 320 \land y \leq 60 \land x \geq 0 \land y \geq 0$

Paso 2: Dibujar las restricciones en el plano cartesiano con x y y como ejes.

●	b : $40\,x + 8\,y = 800$
●	c : $10\,x + 5\,y = 320$
●	d : $y = 60$
●	e : $x = 0$
●	f : $y = 0$

Paso 3: Determinar la región factible

La región factible es el área donde se cumplen todas las restricciones al mismo tiempo, considerando también las restricciones de no negatividad $x \geq 0$ y $y \geq 0$.

Paso 4: Identificar los vértices de la región factible.

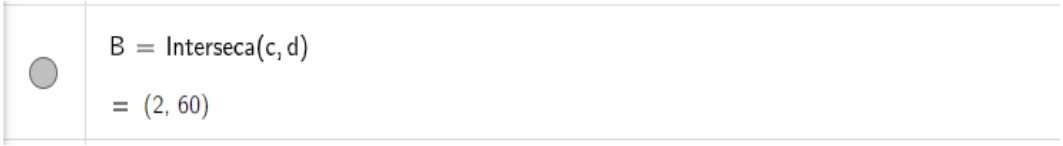
1. **Punto A:** Intersección de $y = 60$ y el eje x. coordenadas $(0, 60)$.

2. **Punto B:** Intersección de las líneas $10x + 5y = 320$ y $y = 60$.
   - Sustituyendo $y = 60$ en $10x + 5(60) = 320$:

$$10x + 300 = 320$$

$$10x = 20 \Longrightarrow x = 2$$

   - Coordenadas de $B = (2,60)$.



3. **Punto C:** Intersección de las líneas $40x + 8y = 800$ y $10x + 5y = 320$.

   - **Resolviendo el sistema de ecuaciones:**

$$40x + 8y = 800$$

$$10x + 5y = 320$$

   - **Despejamos y en la segunda ecuación:**

$$10x + 5y = 320 \Longrightarrow y = 64 - 2x$$

   - **Sustituimos en la primera ecuación:**

$$40x + 8(64 - 2x) = 800$$

$$40x + 512 - 16x = 800$$

$$24x = 288 \Longrightarrow x = 12$$

   - **Sustituyendo** $x = 12$ en $y = 64 - 2x$:

$$y = 64 - 2(12) = 40$$

   - Coordenadas de $C = (12, 40)$.

C = Interseca(b, c)
= (12, 40)

4. **Punto D:** Intersección de $40x + 8y = 800$ y el eje y.
   - Si $y = 0$, entonces $x = 20$.
   - Coordenadas de $D = (20, 0)$.

D = Interseca(b, f)
= (20, 0)

5. **Punto E:** Intersección de los ejes $x = 0$ y $y = 0$, que da el origen: $(0, 0)$.

E = Interseca(e, f)
= (0, 0)

Paso 5: Evaluar la función objetivo en los vértices.

La función objetivo es:

$$Z = 300x + 100y$$

Evaluar Z en cada uno de los vértices identificados:

1. En el punto $A(0, 60)$:
$$Z = 300(0) + 100(60) = 6000$$
2. En el punto $B(2, 60)$:
$$Z = 300(2) + 100(60) = 600 + 6000 = 6600$$
3. En el punto $C(12, 40)$:
$$Z = 300(12) + 100(40) = 3600 + 4000 = 7600$$
4. En el punto $D(20, 0)$:

$$Z = 300(20) + 100(0) = 6000$$

5. En el punto $E(0, 0)$:

$$Z = 300(0) + 100(0) = 0$$

2.8.5. Ejercicio 6. Solución en R.

Utilizando Método Gráfico.

Paso 1: Cargar las bibliotecas necesarias.

```
# Cargar librerías necesarias
library(ggplot2)
```

Se carga $ggplot2$, que es una librería para crear gráficos sofisticados en R.

Paso 2: Definir el rango de valores para las variables.

```
# Definimos el rango de valores para las variables
x <- seq(0, 30, length.out = 400)  # Número de auditorías
```

Se crea una secuencia de valores de x (número de auditorías) que va de 0 a 30 con 400 valores. Este rango será utilizado para evaluar las restricciones y trazar las líneas en el gráfico.

Paso 3: Definir las restricciones en la función de y.

```
# Restricciones
y1 <- (800 - 40 * x) / 8   # Restricción de trabajo directo
y2 <- (320 - 10 * x) / 5   # Restricción de horas de revisión
y3 <- rep(60, length(x))   # Restricción de máximo 60 liquidaciones
```

Cada restricción se convierte en función de y:

- $y1$: Restricción de trabajo directo.
- $y2$: Restricción de horas de revisión.
- $y3$: límite máximo de liquidaciones.

Paso 4: Graficar las restricciones.

```
# Graficar las restricciones
plot(x, y1, type = "l", col = "blue", ylim = c(0, 70), lwd = 3,
     xlab = "Número de Auditorías (x)", ylab = "Número de Liquidaciones (y)",
     main = "Método Gráfico para Maximizar Ingresos")

lines(x, y2, col = "red", lwd = 3)
lines(x, y3, col = "green", lty = 2, lwd = 3)
```

Este bloque genera el gráfico y las líneas de las restricciones:

- $plot$ inicia el gráfico con la restricción $y1$ en azul.
- $lines$ añade las restricciones $y2$ en rojo y $y3$ (límite de liquidaciones) en verde.

Paso 5: Definir el rango para la región factible.

```
# Definir el rango para la región factible
x_fill <- seq(0, 20, length.out = 200)  # Limitar hasta el máximo de auditorías

# Calcular los límites de la región factible
y_fill1 <- (800 - 40 * x_fill) / 8   # Trabajo directo
y_fill2 <- (320 - 10 * x_fill) / 5   # Revisión
y_fill3 <- rep(60, length(x_fill))   # Máximo 60 liquidaciones

# Limitar valores negativos para las restricciones
y_fill1[y_fill1 < 0] <- 0
y_fill2[y_fill2 < 0] <- 0
```

Se crea una secuencia más corta para x (hasta 20) y se calcula las restricciones correspondientes para acotar la región factible, los valores negativos de y se fijan en cero para limitar la región factible al primer cuadrante.

Paso 6: Rellenar la región factible.

```
# Rellenar la región factible solo en el área válida
polygon(c(x_fill, rev(x_fill)),
        c(rep(0, length(x_fill)), rev(pmin(pmin(y_fill1, y_fill2), y_fill3))),
        col = rgb(0.5, 1, 0.5, 0.5))
```

Se crea una región sombreada para la región factible usando $polygon$.

- Se trazan los límites de la región factible usando los valores de y más bajos (mínimos entre $y\_fill1, y\_fill2, y\_fill3$) en cada punto de x.

- Se usa rgb para definir un color semitransparente, resaltando visualmente la región factible.

Paso 7: Calcular las intersecciones.

```
# Identificar los puntos de intersección
# Solucionando 40x + 8y = 800 y 10x + 5y = 320
A <- matrix(c(40, 8, 10, 5), nrow = 2, byrow = TRUE)
b <- c(800, 320)
sol <- solve(A, b)

# Puntos óptimos
x_opt <- sol[1]
y_opt <- sol[2]
```

Se resuelven las ecuaciones $40x + 8y = 800$ y $10x + 5y = 320$ para encontrar el punto de intersección (el valor óptimo de x y y) de estas dos restricciones. $Solve$ resuelve el sistema de ecuaciones y almacena las soluciones en x_opt y y_opt.

Paso 8: Definir los vértices de la región factible.

```
# Vértices de la región factible
vertices <- rbind(c(0, 0), c(0, 60), c(x_opt, y_opt), c(20, 0))

# Calcular la función objetivo de los vértices (Z = 300x + 100y)
Z <- 300 * vertices[, 1] + 100 * vertices[, 2]
```

Se identifican los vértices de la región factible y se calculan los valores de la función objetivo en cada vértice.

- vértices contiene los puntos $(0, 0), (0, 60)$, el punto de intersección calculado anteriormente y $(20,\ 0)$.
- Z calcula el ingreso de cada vértice usando la función objetivo $300x + 100y$.

Paso 9: Imprimir los resultados de los vértices.

```
# Imprimir los resultados
for (i in 1:nrow(vertices)) {
  cat(sprintf("Vértice %d: Auditorías = %.2f, Liquidaciones = %.2f, Ingreso = %.2f\n",
              i, vertices[i, 1], vertices[i, 2], Z[i]))
}
```

for imprime los valores de auditorías, liquidaciones e ingresos en cada vértice de la región factible, facilitando la interpretación de los posibles puntos.

Paso 10: Marcar y etiquetar el punto óptimo en el gráfico.

```
# Marcar el punto óptimo en el gráfico
points(x_opt, y_opt, col = "blue4", pch = 19)

# Etiquetar el punto óptimo al lado del punto
text(x_opt + 0.5, y_opt, labels = paste("(", round(x_opt, 1), ",", round(y_opt, 1), ")", sep = ""),
     col = "magenta", pos = 4)  # pos = 4 para que se posicione a la derecha
```

Se marca el punto óptimo (la intersección de $40x + 8y = 800$ y $10x + 5y = 320$) en el gráfico y se etiqueta con sus coordenadas.

Paso 11: Agregar la leyenda.

```
# Agregar la leyenda
legend("topright", legend = c("Trabajo Directo", "Revisión", "Máximo Liquidaciones"),
       col = c("blue", "red", "green"),
       lty = c(1, 1, 2),
       bty = "o", cex = 0.6)
```

Paso 12: Imprimir los resultados.

```
# Mostrar los resultados finales
cat("Número óptimo de auditorías (x):", round(x_opt, 2), "\n")
cat("Número óptimo de liquidaciones (y):", round(y_opt, 2), "\n")
cat("Ingreso total óptimo (Z):", max(Z), "$\n")  # Se muestra el ingreso máximo encontrado
```

Se muestra en la consola los valores óptimos de auditorías y liquidaciones, así como el ingreso total óptimo.

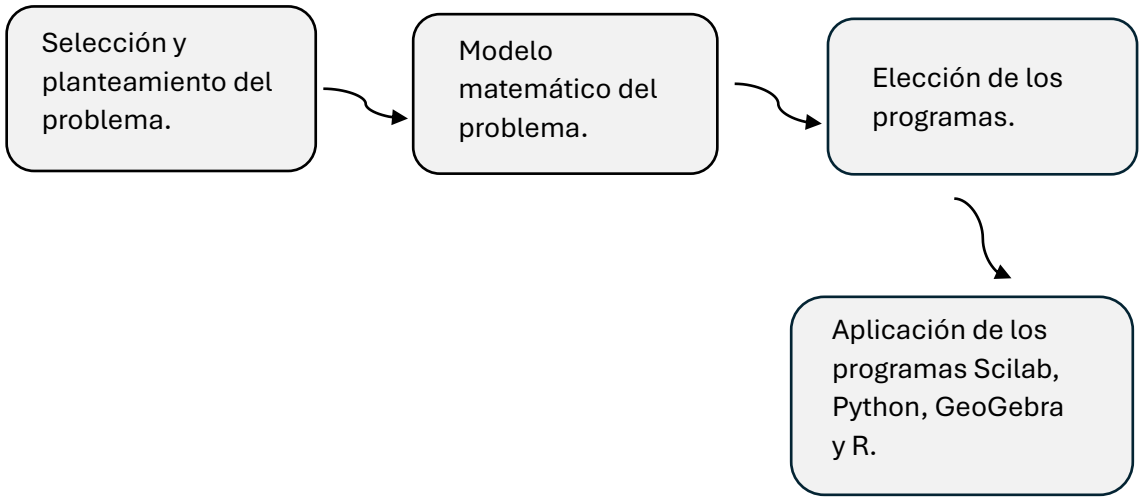
Capítulo 3

DISEÑO METODOLÓGICO

La investigación es de tipo cuantitativa y comparativa, debido a que se utiliza diferentes herramientas computacionales para resolver problemas de programación lineal y se realiza una comparación de su efectividad y eficiencia. Se identifica cuál de ellas ofrece mejores resultados en términos de precisión, se analizan las ventajas y limitaciones de cada programa, proporcionando así una base sólida para futuras investigaciones y aplicaciones prácticas en el campo de la programación lineal.

Para el desarrollo de la investigación fue necesaria una metodología basada en una serie de pasos lógicos que permiten determinar los procedimientos necesarios para analizar y evaluar la efectividad de los programas.

3.1. Metodología.



Selección y planteamiento del problema.

Se identifica y se define el problema de programación lineal.

- Se determina si el objetivo del problema es de maximización o de minimización.
- Se definen las variables de decisión.
- Formular la función objetivo en términos de estas variables, que representa el valor que se desea optimizar.
- Se definen y formulan las restricciones del problema, expresadas como desigualdades o igualdades lineales.

Modelo matemático del problema.

Este proceso implica formular el problema en términos de programación lineal mediante la función objetivo y las restricciones identificadas.

- **Función objetivo:** Se expresa como una combinación lineal de las variables de decisión.
- **Restricciones:** Cada limitación se formula en términos de desigualdades o igualdades.

Elección de los programas.

La elección dependió de varios factores, como la disponibilidad de recursos, el nivel de complejidad del problema, los conocimientos sobre programación y la familiaridad con los programas.

- **Scilab:** Es un software de código abierto para cálculos numéricos.
- **Python:** Proporciona diversas bibliotecas, es versátil y ampliamente utilizado en ciencia de datos.
- **GeoGebra:** Es excelente para el método gráfico en problemas de dos variables y para la visualización de la región factible.
- **R:** Se utiliza principalmente en análisis estadístico, pero cuenta con paquetes para resolver problemas de optimización.

Aplicación de los programas Scilab, Python, GeoGebra y R.

Scilab

1. **Descarga e Instalación**
    - Descargar Scilab desde su página oficial: https://www.scilab.org.
    - Selecciona la versión compatible con tu sistema operativo (Windows, macOS o Linux).
    - Ejecuta el archivo de instalación y sigue las instrucciones en pantalla.
    - Verifica la instalación abriendo Scilab y ejecutando un comando básico, como $disp$(“Instalación correcta”).
2. **Configurar entorno:** Asegurarse de tener instalada la versión más resiente de Scilab y las bibliotecas necesarias.
3. **Ingreso de parámetros:** Definir los coeficientes de la función objetivo y las restricciones.
4. **Solución usando el método gráfico y el simplex**.
5. **Visualización gráfica:** Traza las restricciones y la región factible para obtener una solución gráfica.

Python.

1. **Descarga e instalación.**
    - Descargar Python desde su sitio oficial: https://www.python.org.
    - Durante la instalación, asegurarse de marcar la casilla “Agregar Python a la variable $PATH$”.
    - Instalar las bibliotecas necesarias para optimización.
2. **Configurar las bibliotecas:** Importar $Scipy$ y $Pulp$ para resolver problemas de programación lineal.
3. **Definición del problema:** Plantear la función objetivo y restricciones en el entorno de código.
4. **Ejecutar el algoritmo:** Llamar a las funciones de optimización para obtener los valores óptimos de las variables.

5. **Interpretación de los resultados:** Mostrar y analizar los resultados de la función objetivo y las variables óptimas.

GeoGebra.

1. **Descarga e instalación.**
    - Descargar GeoGebra desde su sitio web oficial: https://www.geogebra.org.
    - Selecciona la versión de escritorio y sigue las instrucciones de instalación.
    - Una vez instalado, abrir GeoGebra y verifica que la herramienta gráfica funcione correctamente ingresando restricciones en el área de gráficos.
2. **Representación de las restricciones:** Utilizar el área de gráficos de GeoGebra para dibujar las restricciones.
3. **Determinar la región factible:** Sombrear la región que satisface todas las restricciones.
4. **Maximización o minimización de la función objetivo:** Mover la línea de la función objetivo en la dirección de optimización para encontrar el punto extremo óptimo.

R.

1. **Descarga e instalación.**
    - Descargar R desde su sitio oficial: https://cran.r-project.org.
    - Descargar e instalar RStudio (Recomendado) desde la página oficial: https://www.rstudio.com.
2. **Instalar paquete lpSolve y ggplot2:** Dependiendo del problema se resuelve por método simplex o método gráfico.
3. **Definición de la función objetivo y las restricciones.**
4. **Interpretación de resultados y gráficas.**

Capítulo 4

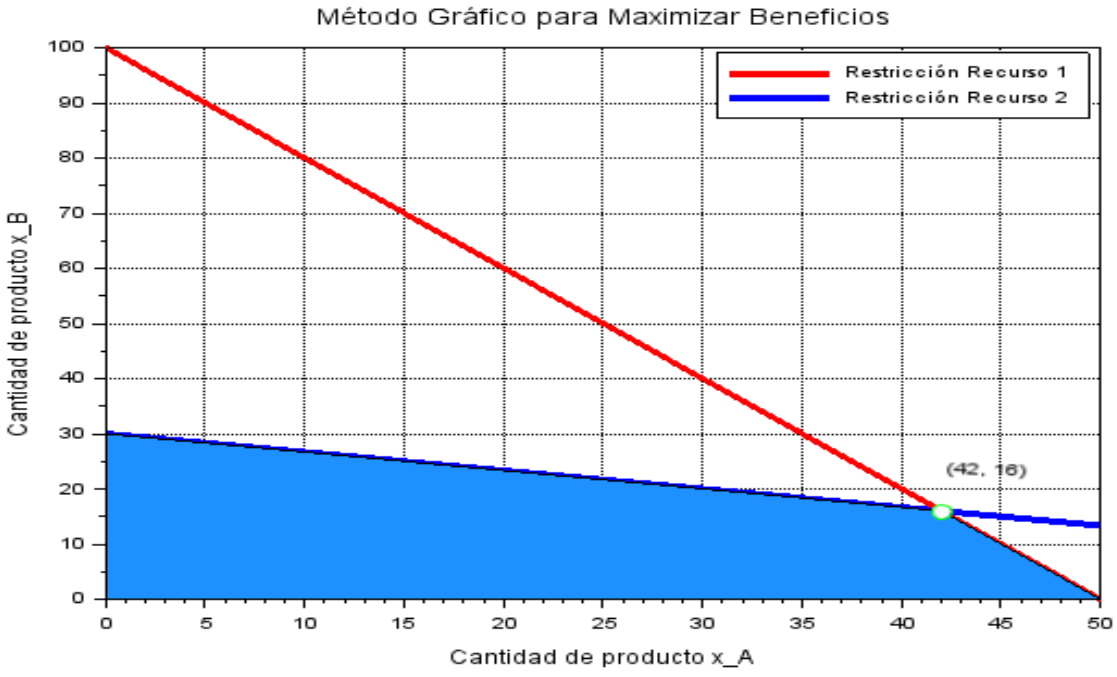
RESULTADOS Y DISCUSIONES

Ejecución del (2.3. Ejercicio 1. Scilab.)

```
"Intersección de restricciones:"

"x_A = 42"

"x_B = 16"

"Beneficio total = 190"
```

Esta es la mejor estrategia para la empresa, dados los recursos disponibles (recurso 1 y recurso 2) y los beneficios unitarios de los productos, es producir producto A (42 unidades) y producto B (16 unidades) para obtener un beneficio máximo de 190 $.

Solución Gráfica. (2.3. Ejercicio 1. Scilab).



- **Región factible:** El área azul es la región donde se puede producir tanto producto A como B respetando las restricciones de los recursos.
- **Líneas de restricciones:** Indican los límites máximos impuestos por la disponibilidad de recursos.
- **Vértices:** (42, 16) Son los puntos donde se evalúa la función objetivo.

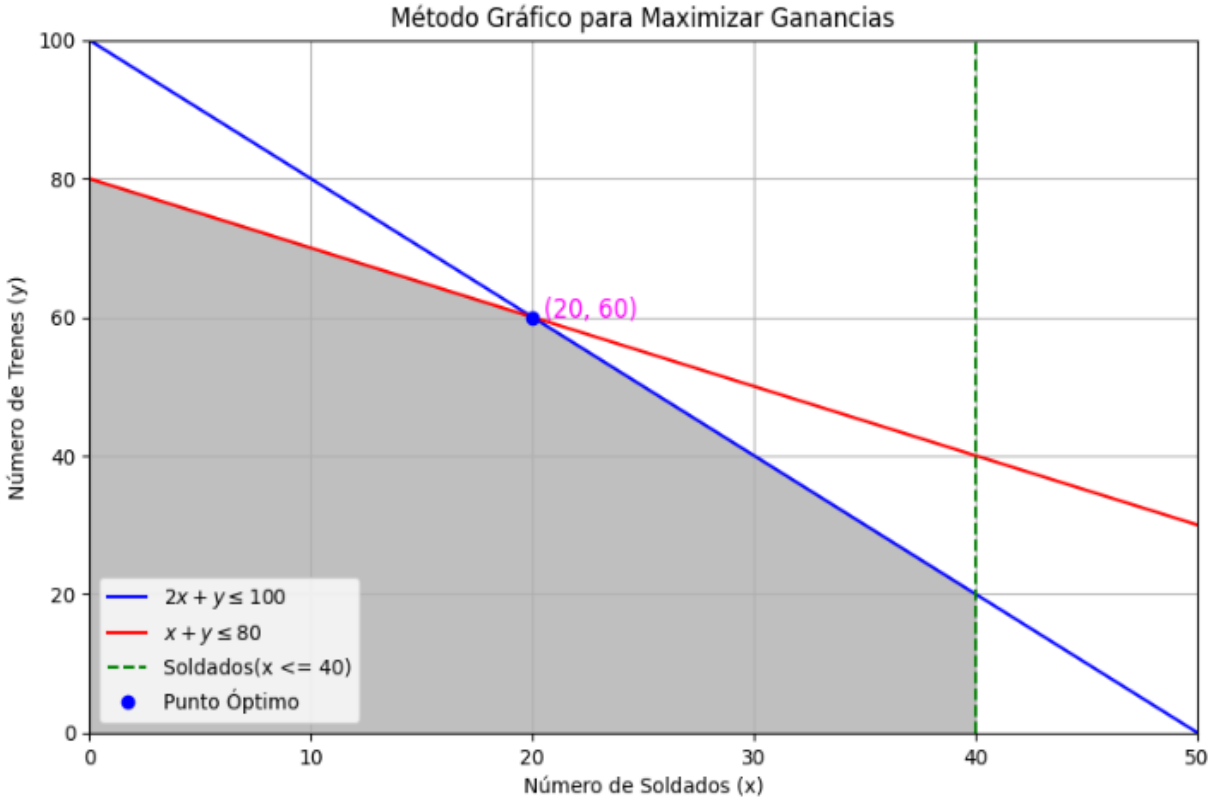
Ejecución del (2.4. Ejercicio 2. Python).

```
...   vértice1: Soldados = 0, Trenes = 0, Ganancia = 0
      vértice2: Soldados = 0, Trenes = 80, Ganancia = 160
      vértice3: Soldados = 40, Trenes = 0, Ganancia = 120
      vértice4: Soldados = 20, Trenes = 60, Ganancia = 180
```

```
...
      Ganancia máxima se obtiene en Vértice 4:
      Soldados = 20, Trenes = 60, Ganancia = 180
```

Para poder obtener la ganancia máxima de 180 se debe asignar 20 soldados y 60 trenes.

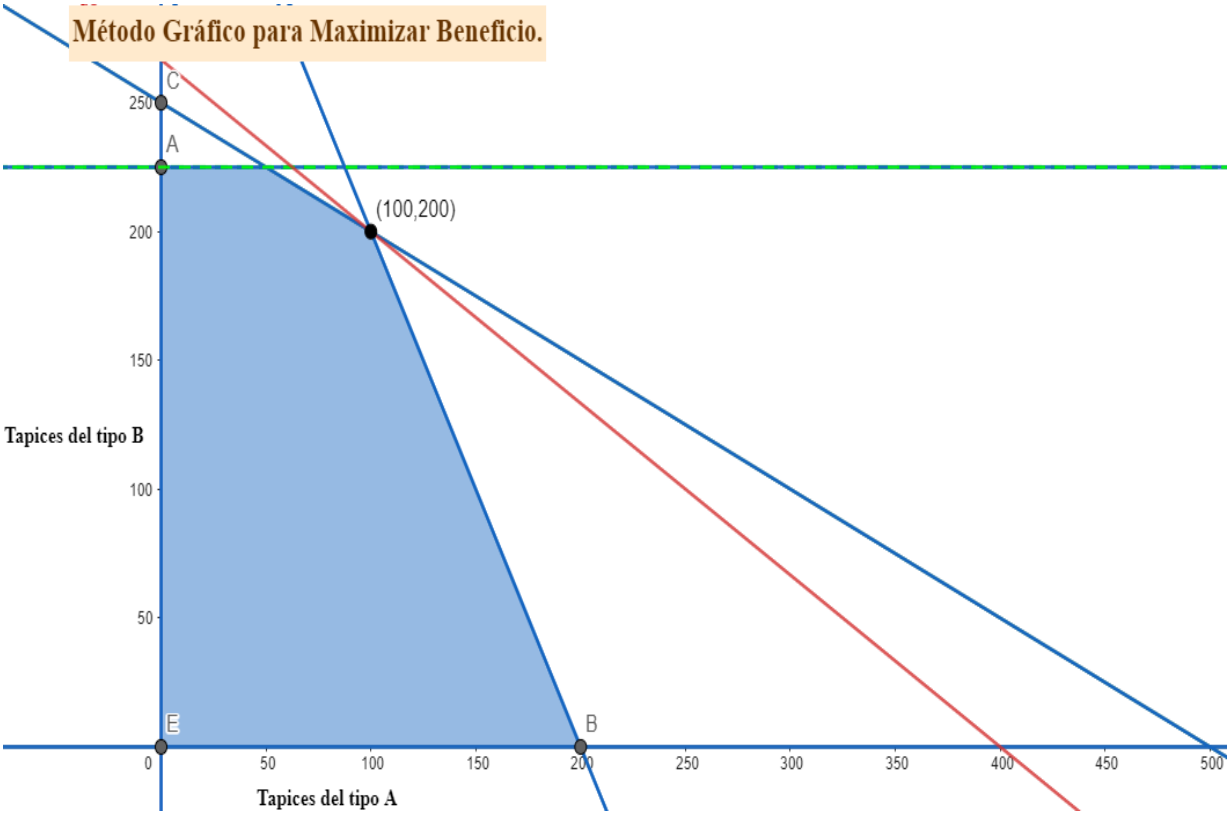
Solución Gráfica (2.4. Ejercicio 2. Python).



- **Región factible:** El área que está en azul es la región factible donde se producen soldados y trenes respetando las restricciones.
- **Líneas de restricciones:** Las líneas rojas y azul representan restricciones y la línea verde discontinua representa el límite de fabricación de soldados.
- **Punto óptimo:** Es la intersección de (20, 60).

Solución Gráfica (2.5. Ejercicio 3. GeoGebra).

El punto que interseca (100, 200) indica que la empresa debe producir 100 tapices del tipo A y 200 tapices del tipo B, se obtiene un beneficio máximo de 80000 euros.



El objetivo es maximizar el beneficio total para la fabricación de cada tipo de tapiz A y B.

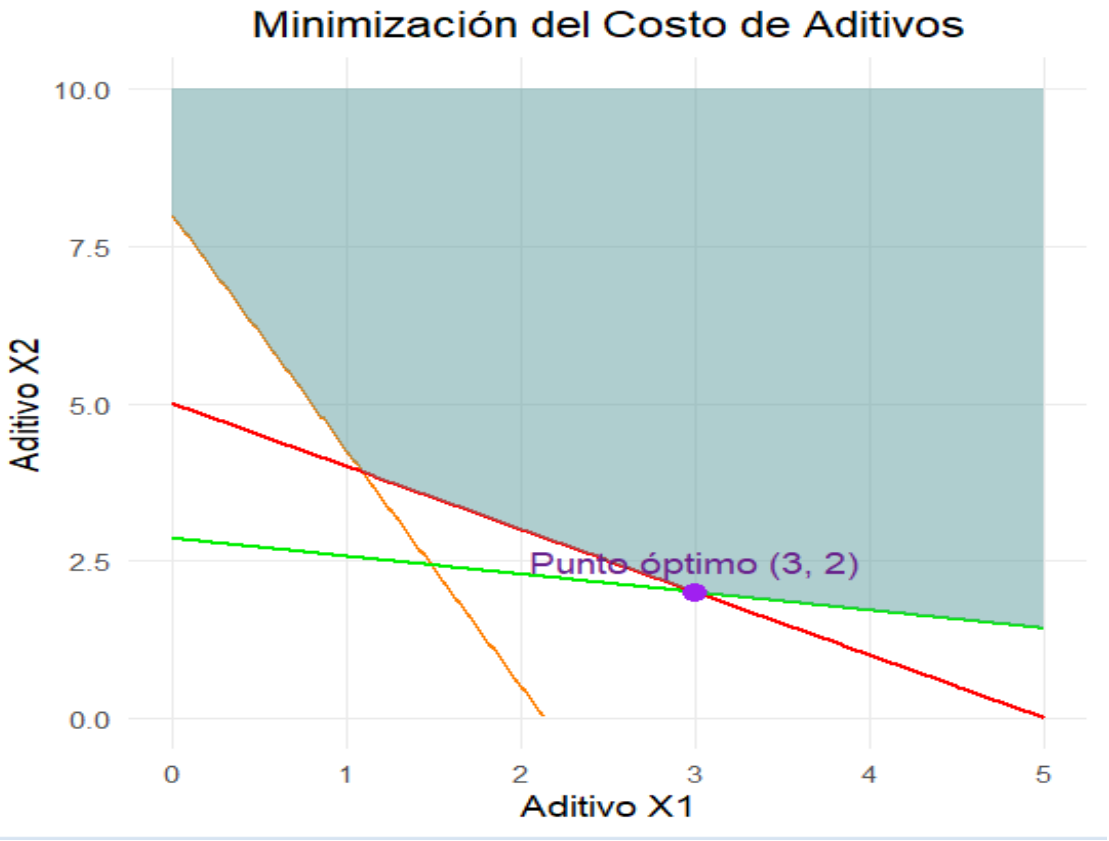
- **La región factible:** La región que se encuentra sombreada en azul en la gráfica representa todas las combinaciones de producción de tapices del tipo A y del tipo B que cumplen con todas las restricciones del problema, aquí es donde la empresa puede operar respetando todas las restricciones de recursos.
- **Líneas de restricción:** La línea roja representa una restricción importante que limita el espacio disponible en el área de producción y las líneas azules son restricciones impuestas al sistema de producción, la línea verde punteada representa el límite del tapiz del tipo B.
- **Punto óptimo:** El punto óptimo en la gráfica es (100, 200), esta combinación permite obtener el mayor beneficio posible dentro de las restricciones de producción.

Ejecución del (2.6. Ejercicio 4. R).

```
> # Mostrar los resultados
> cat("El punto óptimo es:\n")
El punto óptimo es:
> cat("X1 =", x_val, "\n")
X1 = 3
> cat("X2 =", y_val, "\n")
X2 = 2
> cat("Costo mínimo =", min_cost, "$\n")
Costo mínimo = 14 $
```

El objetivo del laboratorio es minimizar costos para eso el costo mínimo de producción debe de ser de 14 $, al utilizar exactamente 3 unidades de aditivo X1 y 2 unidades de aditivo X2, el costo total de los aditivos necesarios para el producto será el menor posible dentro de los límites de las restricciones ya que se está minimizando.

Solución Gráfica (2.6. Ejercicio 4. R).



El objetivo es minimizar el costo total al combinar dos aditivos, X1 y X2, que son necesarios para la producción de un medicamento o producto.

- **Región factible:** Como se observa en la gráfica esta área contiene todas las combinaciones de aditivos X1 y X2 que cumplen con todas las restricciones al mismo tiempo, al ser un problema de minimización se busca el punto dentro de esta región que minimice el costo total.
- **Líneas de restricciones:** Cada línea de color en la gráfica representa una restricción, estas restricciones son ecuaciones lineales que relacionan las cantidades de aditivo X1 y X2.
- **Punto óptimo:** El punto marcado en la gráfica como (3, 2) es la combinación de valores de aditivos X1 y X2 que minimiza el costo total y cumple con todas las restricciones, este punto no solo minimiza el costo, si no también garantiza que se cumplen todos los requisitos necesarios en el producto.

Método Simplex con tres variables.

Se muestran los resultados de la comparación de los programas Scilab, Python y R utilizando el método Simplex con tres variables.

Ejecución del (2.7.2. Ejercicio 5. Scilab).

```
"Tabla inicial:"

 1.    1.    1.    1.    0.    0.    100.
 2.    1.    3.    0.    1.    0.    80.
 1.    2.    1.    0.    0.    1.    60.
-3.   -2.   -4.    0.    0.    0.    0.

"Tabla intermedia:"

 0.3333333   0.6666667   0.   1.  -0.3333333   0.   73.333333
 0.6666667   0.3333333   1.   0.   0.3333333   0.   26.666667
 0.3333333   1.6666667   0.   0.  -0.3333333   1.   33.333333
-0.3333333  -0.6666667   0.   0.   1.3333333   0.   106.66667

"Tabla intermedia:"

 0.2   0.   0.   1.  -0.2  -0.4   60.
 0.6   0.   1.   0.   0.4  -0.2   20.
 0.2   1.   0.   0.  -0.2   0.6   20.
-0.2   0.   0.   0.   1.2   0.4   120.

"Tabla intermedia:"

 0.   0.  -0.3333333   1.  -0.3333333  -0.3333333   53.333333
 1.   0.   1.6666667   0.   0.6666667  -0.3333333   33.333333
 0.   1.  -0.3333333   0.  -0.3333333   0.6666667   13.333333
 0.   0.   0.3333333   0.   1.3333333   0.3333333   126.66667

"Tabla final:"

 0.   0.  -0.3333333   1.  -0.3333333  -0.3333333   53.333333
 1.   0.   1.6666667   0.   0.6666667  -0.3333333   33.333333
 0.   1.  -0.3333333   0.  -0.3333333   0.6666667   13.333333
 0.   0.   0.3333333   0.   1.3333333   0.3333333   126.66667

"Ganancia máxima: 126.66667"

"Cantidad de Producto A: 33.333333"

"Cantidad de Producto B: 13.333333"

"Cantidad de Producto C: 0"
```

Ejecución del (2.7.3. Ejercicio 5. Solución en Python).

```
Tabla inicial del simplex:
Coeficientes de las restricciones (A):
[[1 1 1]
 [2 1 3]
 [1 2 1]]
Recursos disponibles (b):
[100  80  60]
Función objetivo (c):
[-3 -2 -4]

Tabla inicial:
[[  1.   1.   1. 100.]
 [  2.   1.   3.  80.]
 [  1.   2.   1.  60.]
 [ -3.  -2.  -4.   0.]]

Tabla actualizada:
[[  0.33333333   0.66666667   0.          73.33333333]
 [  0.66666667   0.33333333   1.          26.66666667]
 [  0.33333333   1.66666667   0.          33.33333333]
 [ -0.33333333  -0.66666667   0.         106.66666667]]

Tabla actualizada:
[[  0.2   0.    0.   60. ]
 [  0.6   0.    1.   20. ]
...
 [  0.           1.          -0.33333333  13.33333333]
 [  0.           0.           0.33333333 126.66666667]]
Ganancia máxima: 126.66666666666666
Cantidad de productos A, B y C a producir: [33.33333333 13.33333333  0.        ]
```

La tabla inicial muestra las restricciones y la función objetivo. Las variables de holgura se añaden para convertir las restricciones en desigualdades. La función objetivo se escribe con los coeficientes negativos porque el Simplex minimiza por defecto. La primera tabla intermedia se identifica la columna pivote y se actualiza la tabla, la ganancia mejora a 106.67. La segunda tabla intermedia el proceso continúa con la mejora de las variables básicas, la ganancia alcanza 120. La tercera tabla intermedia se sigue actualizando la tabla, alcanzando una ganancia de 126.67. la tabla final la ganancia se estabiliza en 126.67, lo que indica que se ha alcanzado la solucion óptima. La solucion óptima es producir 33.33 unidades de A y 13.33 unidades de B, mientras que no es rentable el producto C. esto maximiza la ganancia total a 126.67 unidades monetarias.

Ejecución del (2.7.4. Ejercicio 5. Solución en R).

```
[1] "Tabla inicial del simplex:"
> print(initial_tableau)
                      b
[1,]  1  1  1 1 0 0 100
[2,]  2  1  3 0 1 0  80
[3,]  1  2  1 0 0 1  60
[4,] -3 -2 -4 0 0 0   0


> # Ejecutar el método simplex
> final_tableau <- simplex(initial_tableau)
[1] "Tabla después de iteración con pivote en columna 3 y fila 2 :"
                                                      b
[1,]  0.3333333  0.6666667 0 1 -0.3333333 0  73.33333
[2,]  0.6666667  0.3333333 1 0  0.3333333 0  26.66667
[3,]  0.3333333  1.6666667 0 0 -0.3333333 1  33.33333
[4,] -0.3333333 -0.6666667 0 0  1.3333333 0 106.66667
[1] "Tabla después de iteración con pivote en columna 2 y fila 3 :"
                            b
[1,]  0.2 0 0 1 -0.2 -0.4  60
[2,]  0.6 0 1 0  0.4 -0.2  20
[3,]  0.2 1 0 0 -0.2  0.6  20
[4,] -0.2 0 0 0  1.2  0.4 120
[1] "Tabla después de iteración con pivote en columna 1 y fila 2 :"
                                                      b
[1,] 0 0 -0.3333333 1 -0.3333333 -0.3333333  53.33333
[2,] 1 0  1.6666667 0  0.6666667 -0.3333333  33.33333
[3,] 0 1 -0.3333333 0 -0.3333333  0.6666667  13.33333
[4,] 0 0  0.3333333 0  1.3333333  0.3333333 126.66667
>
> # Mostrar los resultados finales
> print("Tabla final del simplex:")
[1] "Tabla final del simplex:"
> print(final_tableau)
                                                      b
[1,] 0 0 -0.3333333 1 -0.3333333 -0.3333333  53.33333
[2,] 1 0  1.6666667 0  0.6666667 -0.3333333  33.33333
[3,] 0 1 -0.3333333 0 -0.3333333  0.6666667  13.33333
[4,] 0 0  0.3333333 0  1.3333333  0.3333333 126.66667

> ganancia_maxima <- final_tableau[nrow(final_tableau), ncol(final_tableau)]
> cat("Ganancia máxima:", ganancia_maxima, "$\n")
Ganancia máxima: 126.6667 $
>
> # Calcular las cantidades de productos A, B y C
> num_variables <- length(c)
> solucion <- rep(0, num_variables)
> for (i in 1:num_variables) {
+   columna <- final_tableau[1:(nrow(final_tableau) - 1), i]
+   if (sum(columna == 1) == 1 && sum(columna == 0) == (nrow(final_tableau) - 2)) {
+     fila <- which(columna == 1)
+     solucion[i] <- final_tableau[fila, ncol(final_tableau)]
+   }
+ }
>
> cat("Cantidad de productos A, B y C a producir:", round(solucion, 4), "\n")
Cantidad de productos A, B y C a producir: 33.3333 13.3333 0
```

Aplicación del Método Gráfico.

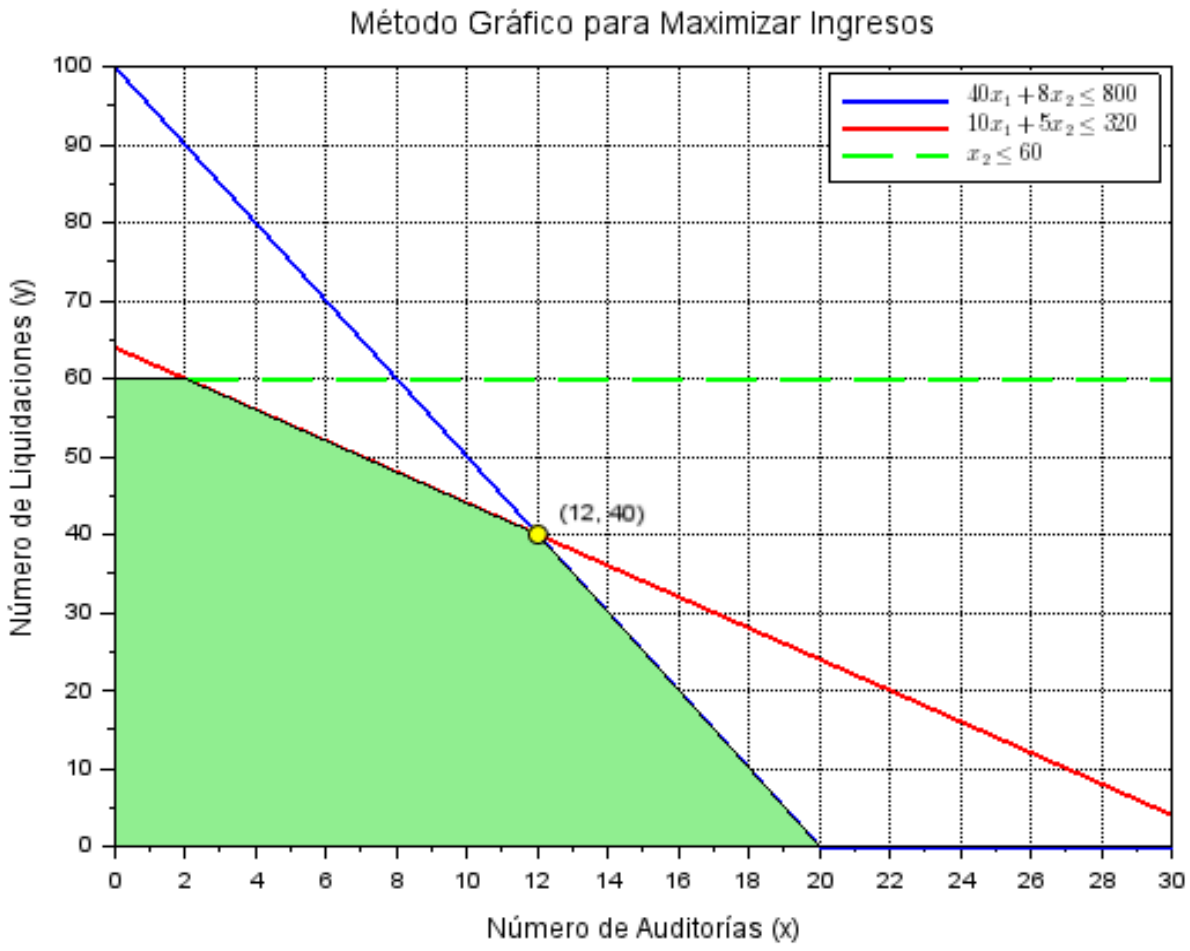
Se aplica el Método Gráfico en todos los programas para saber cuál es el más eficiente de los cuatro.

Ejecución del (2.8.2. Ejercicio 6. Solución en Scilab).

```
Vértice 1: Auditorías = 0, Liquidaciones = 0, Ingreso = $0
Vértice 2: Auditorías = 0, Liquidaciones = 60, Ingreso = $6000
Vértice 3: Auditorías = 12, Liquidaciones = 40, Ingreso = $7600
Vértice 4: Auditorías = 20, Liquidaciones = 0, Ingreso = $6000

Ingreso máximo se obtiene en el Vértice 3:
Auditorías = 12, Liquidaciones = 40, Ingreso = $7600
```

En este caso se necesitan 12 auditorías y 40 liquidaciones que es lo que permite alcanzar el ingreso máximo de $ 7600.

Solución Gráfica (2.8.2. Ejercicio 6. Solución Scilab).



Región factible: El área sombreada en color verde representa todas las combinaciones posibles de auditorías y liquidaciones que cumplen con las restricciones impuestas.

Líneas de restricciones: La línea azul representa el límite máximo de horas de trabajo directo disponibles, la línea roja representa el límite máximo de horas de revisión disponibles y la línea verde discontinua representa el máximo número de liquidaciones disponibles que es (60).

Punto óptimo: Este marcado en las coordenadas (12, 40) este punto representa la solución que maximiza los ingresos, cumpliendo con todas las restricciones.

Ejecución del (2.8.3. Ejercicio 6. Solución en Python).

```
...   Vértice 1: Auditorías = 0, Liquidaciones = 0, Ingreso = 0
      Vértice 2: Auditorías = 0, Liquidaciones = 60, Ingreso = 6000
      Vértice 3: Auditorías = 12, Liquidaciones = 40, Ingreso = 7600
      Vértice 4: Auditorías = 20, Liquidaciones = 0, Ingreso = 6000
```

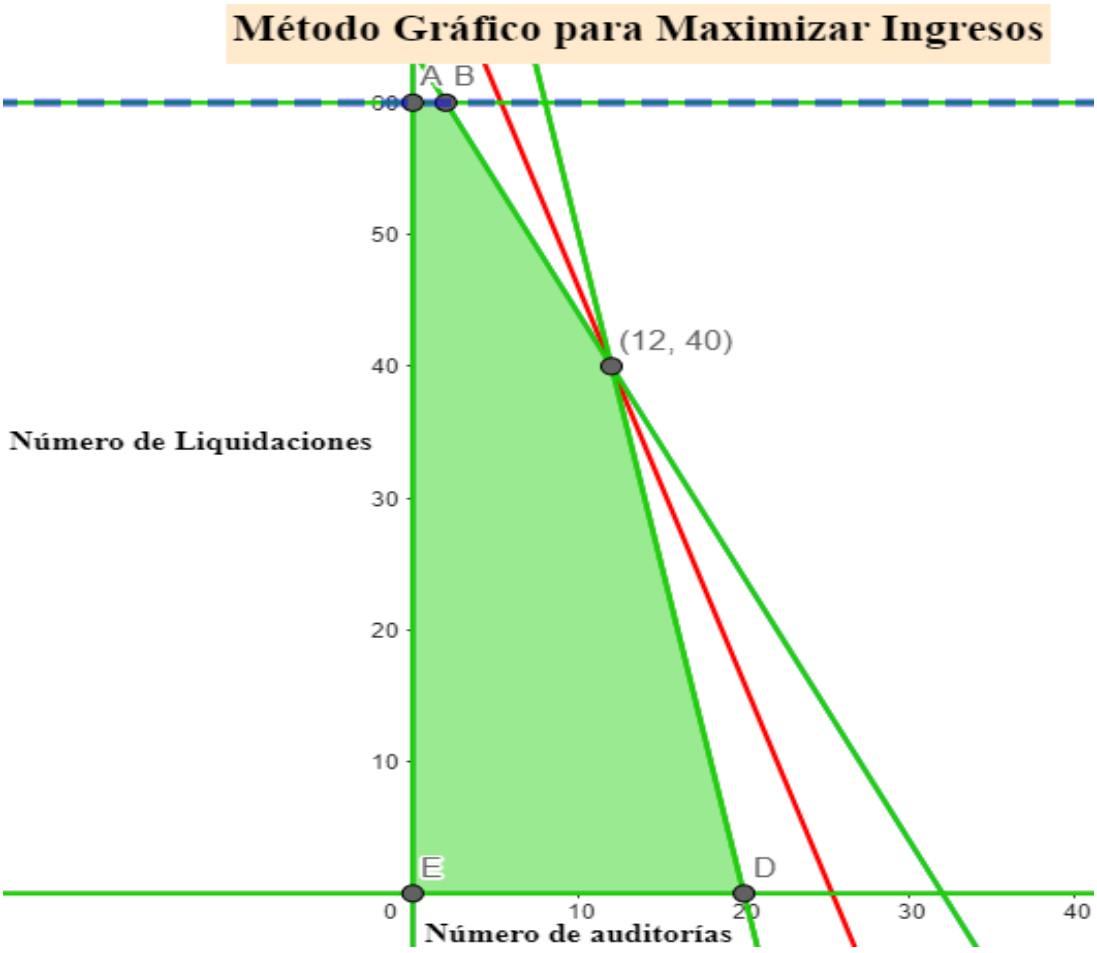
```
...
      Ingreso máximo se obtiene en el Vértice 3:
      Auditorías = 12, Liquidaciones = 40, Ingreso  = $ 7600
```

Solución Grafica (2.8.3. Ejercicio 6. Solución en Python).

Solución Gráfica (2.8.4. Ejercicio 6. Solución en GeoGebra).

C = Interseca(b, c)
= (12, 40)

g : 300 x + 100 y = 7600

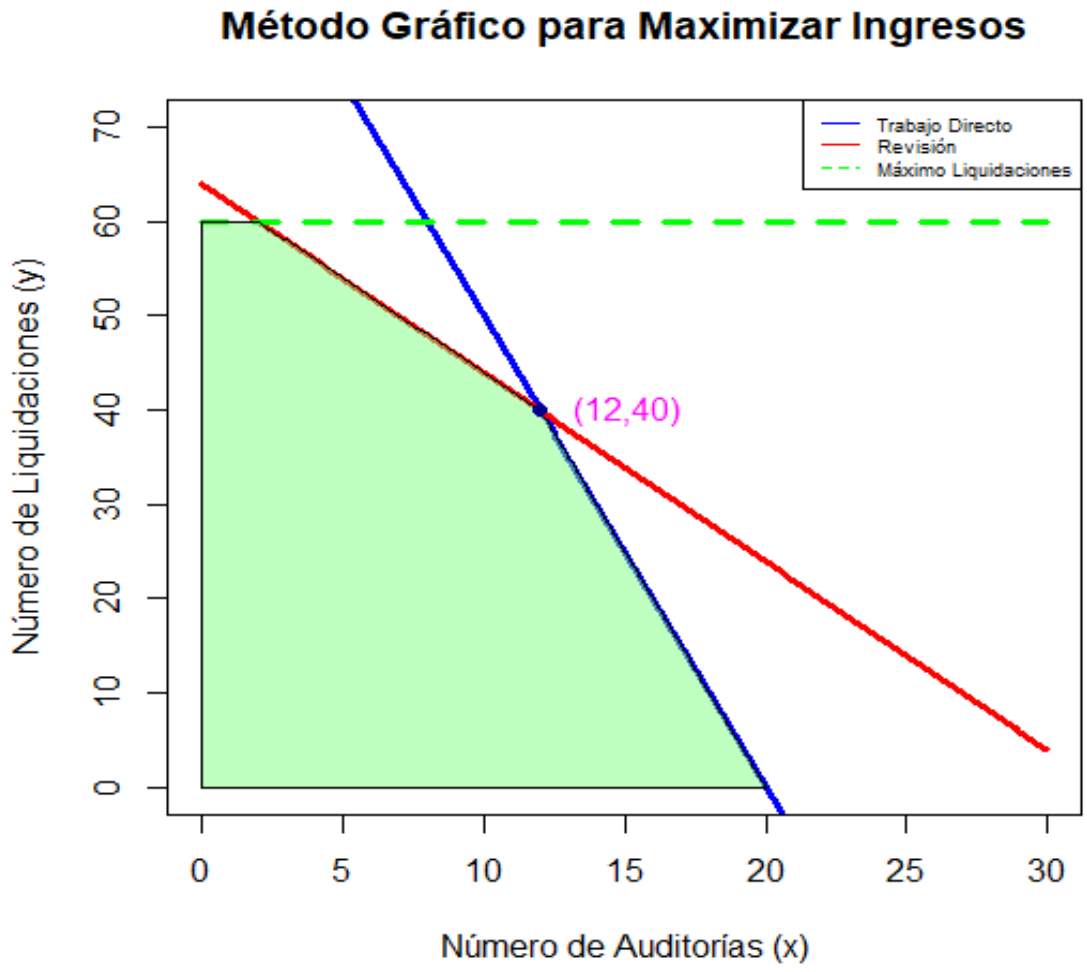


Ejecución del (2.8.5. Ejercicio 6. Solución en R).

```
Vértice 1: Auditorías = 0.00, Liquidaciones = 0.00, Ingreso = 0.00
Vértice 2: Auditorías = 0.00, Liquidaciones = 60.00, Ingreso = 6000.00
Vértice 3: Auditorías = 12.00, Liquidaciones = 40.00, Ingreso = 7600.00
Vértice 4: Auditorías = 20.00, Liquidaciones = 0.00, Ingreso = 6000.00
```

```
> cat("Número óptimo de auditorías (x):", round(x_opt, 2), "\n")
Número óptimo de auditorías (x): 12
> cat("Número óptimo de liquidaciones (y):", round(y_opt, 2), "\n")
Número óptimo de liquidaciones (y): 40
> cat("Ingreso total óptimo (Z):", max(Z), "$\n")  # Se muestra el ingreso máximo encontrado
Ingreso total óptimo (Z): 7600 $
```

Solución Gráfica (2.8.5. Ejercicio 6. Solución en R).



Capítulo 5

CONCLUSIONES

La aplicación de estos programas en programación lineal permite un enfoque variado, desde cálculos numéricos precisos hasta visualización y análisis gráfico, lo que facilita la toma de decisiones en entornos de producción, economía, ingeniería, salud y administración.

El uso conjunto de Scilab, Python, GeoGebra y R proporciona una base sólida para la solución de problemas de programación lineal, permitiendo abordar el problema desde diferentes ángulos. Scilab y Python son más adecuados para cálculos complejos y personalización de algoritmos, mientras que GeoGebra ofrece una comprensión visual intuitiva y R facilita el análisis de resultados y la interpretación de datos, a través de la implementación de estos programas, se puede no solo obtener soluciones óptimas para problemas de optimización, sino también explorar diferentes escenarios y realizar un análisis profundo de las restricciones y variables que afectan las decisiones. En conclusión, Python suele ser la mejor opción para la mayoría de los problemas de programación lineal porque combina capacidad de cálculo y flexibilidad con facilidad de uso, no obstante, si el enfoque es más visual y educativo GeoGebra es muy efectivo. Scilab es para cálculos detallados con dos variables y R para problemas con análisis estadístico integrado.

Capítulo 6

RECOMENDACIONES

- Este trabajo puede utilizarse de guía para estudiantes, profesores y futuros investigadores que estén interesados en la Programación lineal.

- Se sugiere a los estudiantes y docentes, que adopten herramientas como Scilab, Python, GeoGebra y R porque no solo facilitan la solución de problemas de programación lineal, sino que también generan un aprendizaje profundo y aplicable a situaciones reales.

- Es aconsejable que los estudiantes Investiguen métodos más avanzados para resolver problemas no solo lineales, sino también programación cuadrática o programación no lineal.

- Se propone a los estudiantes y docentes, explorar y desarrollar nuevos algoritmos de optimización que pueda ser implementados en estos programas para mejorar la eficiencia y la precisión.

REFERENCIAS BIBLIOGRÁFICAS

- Almanza, R. (2018). *Programación lineal con Python – Parte 1: Introducción*.https://medium.com/@ramcesc/aprendiendoprogramaci%C3%B3n-lineal-con-python-introducci%C3%B3n-8604fe2cbdeb
- Bahamonte, E. (2019) *Programación Lineal Scilab*. Santo Domingo-Ecuador.https://www.studocu.com/ec/document/universidad-de-las-fuerzas-armadas-de-ecuador/probabilidad-y-estadistica/programacion-lineal-scilab/5282789
- Bazaaraa, M.S, Jarvis, J.J & Sherali, H.D (2010) *Programación lineal y flujo en redes*. Cuarta edición
- Bazaraa, M.S, Jarvis, J.J. (1995). *Programación Lineal y Flujo en Redes.* Limusa Segunda Edición.
- Bieman, H, Bonini, C & Warren, H. (1999) *Análisis cuantitativo para los negocios.* 9° cd. Bogotá: Mc Graw Hill.
- Caro, A.A. & Sepúlveda, C. V. (2004). *Fundamentos de Scilab y Aplicaciones*.https://usuarios.fceia.unr.edu.ar/~ericah/Soft/fundamentos_app_scilab.pdf
- Carrillo Urbina, J.F, Rueda Baldanado, J.F & Zapata Zapata, M.R. (2005). *Uso del paquete Excel solver en la solución de problemas de programación lineal.*
- Fedossova, A, Buitrago Suescún, O. Y & Britto Agudelo, R. A. (2011). *Introducción a la Programación Lineal.* Primera edición. Bogotá, D. C.
- Ferrari, H. (2011). *Introducción al software para cálculo numérico Scilab.* https://www.educ.ar/recursos/14515/introduccion-al-software-para-calculo-numerico-scilab
- Frisoli, C. (2022). *Qué es Python: para que sirve y funciones (+ recursos)*. https://blog.hubspot.es/website/que-es-python
- Hernández Castelán, J. V. (2021). *Método Simplex en R.* https://bookdown.org/eljorgehdz/simplex/simplex.html

- Hiller, F & Lieberinan, G. (1993). *Introducción a la investigación de operaciones.* 5° cd. México: Mc Graw Hill.
- López Briega, R.E. (2017). *Problemas de Optimización con Python.* https://relopezbriega.github.io/blog/2017/01/18/problemas-de-optimizacion-con-python/#google_vignette
- Manco Chávez, J.A. (2020). *Aplicación del GeoGebra en la programación lineal*.https://www.researchgate.net/publication/344068235_Aplicacion_del_GeoGebra_en_la_programacion_lineal
- Mathur, K & Solow, D. (1996) *Investigación de Operaciones: El Arte de la Toma de Decisiones.* Prentice-Hall Hispanoamericana.
- Negrillo, N. (2024). Programación Lineal y GeoGebra. https://es.scribd.com/document/464838425/Programacion-Lineal-y-GeoGebra
- Nocedal, J & Wright, S.J. (2006) *Numerical Optimization.*
- Paz Santos, C. A & Minguillón Alfonso, J. (2019). Programación lineal en R. https://datascience.recursos.uoc.edu/es/programacion-lineal-en-r/
- Pykes, K. (2024). *Optimización en Python: Técnicas, Paquetes y Buenas Prácticas.* https://www.datacamp.com/es/tutorial/optimization-in-python
- Salazar López, B. (2019). *Método Simplex.* https://ingenieriaindustrialonline.com/investigacion-de-operaciones/metodo-simplex/
- Saldaña Acosta, R. (2017). *GeoGebra para la enseñanza de las matemáticas.*https://observatorio.tec.mx/edu-bits-blog/2017-6-6-geogebra-para-la-enseanza-de-las-matemticas/
- Shrijver, A. (1998) *Theory of Linear and integer programming.*
- Taha, H.A. (2004*) Investigación de Operaciones*. Séptima edición. Pearson Educación. México.
- Taha, H.A (2017). *Operations research: An introduction* (10th. ed). person.

ANEXOS

Códigos completos de los programas.

Scilab

```
// Definición de variables
profit_A = 3;  // Beneficio por unidad de A
resource1_A = 2; // Requerimiento de recurso 1 para A
resource2_A = 1; // Requerimiento de recurso 2 para A

profit_B = 4;  // Beneficio por unidad de B
resource1_B = 1;  // Requerimiento de recurso 1 para B
resource2_B = 3;  // Requerimiento de recurso 2 para B

available_resource1 = 100;  // Total de recurso 1 disponible
available_resource2 = 90;  // Total de recurso 2 disponible

// Definición del rango para las variables de decisión
x_A = 0:1:50;  // Cantidad de producto A (de 0 a 50)

// Cálculo de x_B (producto B) según las restricciones
x_B1 = (available_resource1 - resource1_A * x_A) / resource1_B; // Restricción para recurso 1
x_B2 = (available_resource2 - resource2_A * x_A) / resource2_B; // Restricción para recurso 2

// Limitar los valores de x_B a no negativos
x_B1(x_B1 < 0) = 0;
x_B2(x_B2 < 0) = 0;

// Graficar las restricciones con líneas más gruesas
clf; // Limpiar la figura
plot (x_A, x_B1, 'r', 'thickness', 4); // Restricción recurso 1 con grosor 3
plot (x_A, x_B2, 'b-', 'thickness', 4); // Restricción recurso 2 con grosor 3
xlabel ('Cantidad de producto x_A');
ylabel ('Cantidad de producto x_B');
title ('Método Gráfico para Maximizar Beneficios');
xgrid ();
legend ('Restricción Recurso 1', 'Restricción Recurso 2');

// Definir la región factible
fgx = [0, 42, 50, 0]; // Puntos en x
fgy = [30, 16, 0, 0]; // Puntos en y

// Definir el color dodgerblue1 (30, 144, 255 en RGB)
dodgerblue1 = color (30, 144, 255);

xfpoly (fgx, fgy, dodgerblue1); // Rellenar la región factible
```

```
// Resolver el sistema de ecuaciones para encontrar la intersección
A = [2, 1; 1, 3];
b = [100; 90];
solution = A \ b;
intersec_x_A = solution (1);
intersec_x_B = solution (2);

// Mostrar los resultados
disp ("Intersección de restricciones:");
disp ("x_A = " + string(intersec_x_A));
disp ("x_B = " + string(intersec_x_B));

// Calcular el beneficio total
beneficio_total = profit_A * intersec_x_A + profit_B * intersec_x_B;
disp ("Beneficio total = " + string(beneficio_total));

// Punto óptimo
plot (intersec_x_A, intersec_x_B, 'go', 'MarkerSize', 9); // Punto óptimo.
// Etiquetar el punto óptimo
xstring (intersec_x_A, intersec_x_B + 5, "(42, 16)", 2);
```

Python

```
import numpy as np
import matplotlib.pyplot as plt

# Definición de las restricciones
x = np.linspace (0, 50, 400)

# Horas de trabajo de acabado
y1 = 100 - 2*x

# Horas de trabajo de carpintería
y2 = 80 - x

# Limite de producción de soldados
y3 = np.full_like (x, 0) # para la restricción x <= 40

# Graficar las restricciones
plt.figure (figsize=(10, 6))
plt.plot (x, y1, label=r'$2x + y \leq 100$',color='blue')
plt.plot (x, y2, label=r'$x + y \leq 80$',color='red' )
plt.axvline (40, color='green', linestyle='--', label='Soldados(x <= 40)')

# Configurar límites del gráfico
plt.xlim (0, 50)
plt.ylim (0, 100)
plt.xlabel ('Número de Soldados (x)')
plt.ylabel ('Número de Trenes (y)')
plt.title ('Método Gráfico para Maximizar Ganancias')
```

```
plt.fill_between(x, 0, np.minimum (y1, y2), where=(x <= 40), color='gray',
alpha=0.5)

#identificar los puntos de intersección
x_inter = 20
y_inter = 60

# Vértices de la región factible
vertices = np.array ([[0, 0], [0, 80], [40, 0], [x_inter, y_inter]])

#Calcular la función objetivo de los vértices
Z = 3 * vertices [:, 0] + 2 * vertices[:, 1]

# Imprimir los resultados
for i, (vx, vy) in enumerate(vertices):
    print (f"vértice{i+1}: Soldados = {vx}, Trenes = {vy}, Ganancia =
{Z[i]}")

plt.plot (x_inter, y_inter, 'bo', label='Punto Óptimo')
plt.text (x_inter + 0.5, y_inter, f'({x_inter}, {y_inter})', fontsize=12,
color='magenta')
# Mostrar leyenda y Gráfico
plt.legend ()
plt.grid ()
plt.show ()

# solución
max_profit_index = np.argmax(Z)
print (f"\nGanancia máxima se obtiene en Vértice {max_profit_index+1}:")
print (f"Soldados = {vertices [max_profit_index, 0]}, Trenes =
{vertices[max_profit_index,1]}, Ganancia = {Z[max_profit_index]}")
```

R

```
# Cargar la librería ggplot2

library(ggplot2)


# Definir las funciones de las restricciones en términos de x (X1) y y (X2)

f1 <- function(x) {(32 - 15 * x) / 4}   # Vitamina A

f2 <- function(x) {(10 - 2 * x) / 2}    # Vitamina B

f3 <- function(x) {(40 - 4 * x) / 14}   # Vitamina C
```

```
# Crear valores de x para graficar
x_values <- seq (0, 5, length.out = 400)

# Calcular valores de y para cada x según las restricciones
y_f1 <- sapply (x_values, f1)
y_f2 <- sapply (x_values, f2)
y_f3 <- sapply (x_values, f3)

# Limitar valores negativos en las restricciones
y_f1[y_f1 < 0] <- NA
y_f2[y_f2 < 0] <- NA
y_f3[y_f3 < 0] <- NA

# Crear el gráfico con las restricciones
df <- data.frame (x = x_values, f1 = y_f1, f2 = y_f2, f3 = y_f3)

# Definir el punto óptimo
x_val <- 3
y_val <- 2
min_cost <- 2 * x_val + 4 * y_val

# Crear un dataframe para el punto óptimo
optimo <- data.frame (x_val = x_val, y_val = y_val)

# Graficar las restricciones y sombrear el área factible
ggplot (df, aes (x = x)) +
  geom_line (aes (y = f1), color = "darkorange1", linewidth = 1, linetype = "solid") +
  geom_line (aes (y = f2), color = "red", linewidth = 1, linetype = "solid") +
```

```
  geom_line (aes (y = f3), color = "green2", linewidth = 1, linetype = "solid") +


  # Sombreado de la región factible
  geom_ribbon (aes (ymin = pmax (f1, f2, f3, na.rm = TRUE), ymax = 10), fill =
"cadetblue", alpha = 0.5) +
  geom_point (data = optimo, aes (x = x_val, y = y_val), color = "purple", size = 4) +
# Punto óptimo
  annotate ("text", x = x_val, y = y_val + 0.5, label = "Punto óptimo (3, 2)", color =
"darkorchid4", size = 5, hjust = 0.5) +


  # Ajustes estéticos
  coord_cartesian (xlim = c (0, 5), ylim = c (0, 10)) +
  labs (x = "Aditivo X1", y = "Aditivo X2") +
  ggtitle ("Minimización del Costo de Aditivos") +
  theme_minimal (base_size = 14) +
  theme (plot.title = element_text(hjust = 0.5),
        panel.grid.minor = element_blank())


# Mostrar los resultados
cat ("El punto óptimo es:\n")
cat ("X1 =", x_val, "\n")
cat ("X2 =", y_val, "\n")
cat ("Costo mínimo =", min_cost, "$\n")
```

Método Simplex con tres variables.

Scilab.

```
// Definir los coeficientes de los productos A, B y C (para maximizar)
cost_A = 3; // Beneficio por unidad de A
cost_B = 2; // Beneficio por unidad de B
cost_C = 4; // Beneficio por unidad de C

// Definir la matriz de coeficientes de las restricciones
A = [1 1 1;  // Restricción 1: horas de trabajo
   2 1 3;  // Restricción 2: materia prima
   1 2 1]; // Restricción 3: espacio de almacenamiento
```

```
// Definir el vector de términos independientes
b = [100; 80; 60]; // Recursos disponibles

// Definir los coeficientes de la función objetivo (negados para maximizar)
c = [-cost_A; -cost_B; -cost_C; 0; 0; 0];

// Añadir variables de holgura a las restricciones
A = [A, eye (3, 3)]; // Añadir una matriz identidad de 3x3 para las variables de holgura

// Crear la tabla inicial del simplex
tableau = [A, b; c', 0];

// Mostrar la tabla inicial
disp ("Tabla inicial:");
disp(tableau);

// Función para encontrar la columna pivote (mínimo costo negativo)
function col_pivot=find_col_pivot(tableau)
  [min_val, col_pivot] = min (tableau ($, 1:$-1)); // Buscar la columna con el costo más negativo
endfunction

// Función para encontrar la fila pivote (mínima razón)
function row_pivot=find_row_pivot (tableau, col_pivot)
  ratios = tableau (1:$-1, $) ./ tableau(1:$-1, col_pivot); // Calcular las razones
  ratios (ratios <= 0) = %inf; // Ignorar ratios no positivas
  [min_val, row_pivot] = min(ratios); // Encontrar la fila con la razón mínima
endfunction

// Iterar hasta que no haya más columnas pivote negativas
while sum (tableau ($, 1:$-1) < 0) > 0
  col_pivot = find_col_pivot(tableau);  // Encontramos la columna pivote
  row_pivot = find_row_pivot (tableau, col_pivot); // Encontramos la fila pivote

  // Actualizar la tabla usando el elemento pivote
  pivot = tableau (row_pivot, col_pivot);
  tableau (row_pivot, :) = tableau (row_pivot, :) / pivot; // Normalizar fila pivote

  // Actualizar el resto de las filas
  for i = 1:size (tableau, 1)
    if i <> row_pivot then
      tableau (i, :) = tableau (i, :) - tableau (i, col_pivot) * tableau(row_pivot, :);
    end
  end

  // Mostrar la tabla después de cada iteración
  disp ("Tabla intermedia:");
  disp(tableau);
end

// Mostrar la tabla final
disp ("Tabla final:");
disp(tableau);

// Costo máximo (es el valor en la última celda)
max_cost = tableau ($,$);
disp ("Ganancia máxima: " + string(max_cost));

// Inicializar el vector de soluciones
x_values = zeros (3, 1);
```

```
// Identificar las columnas correspondientes a las variables básicas (A, B y C)
for i = 1:3
  columna = tableau (1:$-1, i);
  if sum (columna == 1) == 1 && sum (columna == 0) == (size (tableau, 1) - 2)
    fila = find (columna == 1);
    x_values(i) = tableau (fila, $);  // Extraemos el valor de la última columna (solución)
  end
end

// Mostrar las cantidades de productos A, B y C
disp ("Cantidad de Producto A: " + string (x_values (1)));
disp ("Cantidad de Producto B: " + string (x_values (2)));
disp ("Cantidad de Producto C: " + string (x_values (3)));
```

Python

```
import numpy as np
from scipy.optimize import linprog

# Definir la matriz de coeficientes de las restricciones
A = np.array ([
    [1, 1, 1],  # Restricción de horas de trabajo
    [2, 1, 3],  # Restricción de materia prima
    [1, 2, 1]   # Restricción de espacio de almacenamiento
])

# Definir el vector de términos independientes (recursos disponibles)
b = np.array([100, 80, 60])

# Definir la función objetivo (negativa para maximización)
c = np.array([-3, -2, -4])

# Resolver el problema usando el método HiGHS
res = linprog (c, A_ub=A, b_ub=b, method='highs')

# Mostrar la tabla inicial
print ("Tabla inicial del simplex:")
print ("Coeficientes de las restricciones (A):")
print(A)
print ("Recursos disponibles (b):")
print(b)
print ("Función objetivo (c):")
print (c)

# Función para encontrar la columna pivote
def encontrar_columna_pivote(tabla):
    return np.argmin (tabla[-1, :-1])
```

```python
# Función para encontrar la fila pivote
def encontrar_fila_pivote (tabla, col_pivote):
    filas = tabla [:-1, -1] / tabla[:-1, col_pivote]
    filas [filas <= 0] = np.inf
    return np.argmin (filas)

# Iterar hasta que no haya más columnas pivote negativas
def simplex(tabla):
    while np.any (tabla [-1, :-1] < 0):
        col_pivote = encontrar_columna_pivote(tabla)
        fila_pivote = encontrar_fila_pivote (tabla, col_pivote)
        pivote = tabla [fila_pivote, col_pivote]
        tabla [fila_pivote, :] = tabla[fila_pivote, :] / pivote
        for i in range(len(tabla)):
            if i != fila_pivote:
                tabla [i, :] -= tabla[i, col_pivote] * tabla[fila_pivote, :]
        print ("\nTabla actualizada:")
        print(tabla)
    return tabla

# Crear la tabla inicial del simplex
tabla_inicial = np.hstack ([A, b.reshape(-1, 1)]).astype(float)
tabla_inicial = np.vstack ([tabla_inicial, np.hstack([c, 0])])

print ("\nTabla inicial:")
print(tabla_inicial)

# Ejecutar el método simplex
tabla_final = simplex(tabla_inicial)

print ("\nTabla final:")
print(tabla_final)

# Mostrar los resultados
print ("Ganancia máxima:", -res.fun)
print ("Cantidad de productos A, B y C a producir:", res.x)
```

R

```
library(lpSolve)

# Definir la matriz de coeficientes de las restricciones

A <- matrix (c (1, 1, 1, # Restricción de horas de trabajo

               2, 1, 3, # Restricción de materia prima
```

```
            1, 2, 1), # Restricción de espacio de almacenamiento
          nrow = 3, byrow = TRUE)

# Definir el vector de términos independientes (recursos disponibles)
b <- c (100, 80, 60)

# Definir la función objetivo (positiva para maximización)
c <- c (3, 2, 4)

# Crear la tabla inicial del simplex
simplex_table <- function (A, b, c) {
  tableau <- cbind (A, diag(nrow(A)), b)
  tableau <- rbind (tableau, c (-c, rep (0, nrow (A) + 1)))
  return(tableau)
}

# Mostrar la tabla inicial
initial_tableau <- simplex_table (A, b, c)
print ("Tabla inicial del simplex:")
print(initial_tableau)

# Función para encontrar la columna pivote
pivot_column <- function(tableau) {
  return(which.min(tableau[nrow(tableau), 1:(ncol(tableau) - 1)]))
}

# Función para encontrar la fila pivote
pivot_row <- function (tableau, pivot_col) {
```

```
  ratios <- tableau[1:(nrow(tableau) - 1), ncol(tableau)] / tableau[1:(nrow(tableau) - 1), pivot_col]
  ratios [ratios <= 0] <- Inf
  return(which.min(ratios))
}

# Iterar hasta que no haya más columnas pivote negativas
simplex <- function(tableau) {
  while (any(tableau[nrow(tableau), 1:(ncol(tableau) - 1)] < 0)) {
    pivot_col <- pivot_column(tableau)
    pivot_row <- pivot_row (tableau, pivot_col)
    pivot_element <- tableau [pivot_row, pivot_col]

    tableau [pivot_row, ] <- tableau[pivot_row, ] / pivot_element
    for (i in 1:nrow (tableau)) {
      if (i != pivot_row) {
        tableau [i, ] <- tableau[i, ] - tableau[i, pivot_col] * tableau[pivot_row, ]
      }
    }

    print (paste ("Tabla después de iteración con pivote en columna", pivot_col, "y fila", pivot_row, ":"))
    print(tableau)
  }
  return(tableau)
}

# Ejecutar el método simplex
final_tableau <- simplex(initial_tableau)
```

```
# Mostrar los resultados finales
print ("Tabla final del simplex:")
print(final_tableau)
ganancia_maxima <- final_tableau[nrow(final_tableau), ncol(final_tableau)]
cat ("Ganancia máxima:", ganancia_maxima, "$\n")

# Calcular las cantidades de productos A, B y C
num_variables <- length(c)
solucion <- rep (0, num_variables)
for (i in 1:num_variables) {
  columna <- final_tableau[1:(nrow(final_tableau) - 1), i]
  if (sum (columna == 1) == 1 && sum (columna == 0) == (nrow(final_tableau) - 2))
{
    fila <- which (columna == 1)
    solucion[i] <- final_tableau [fila, ncol(final_tableau)]
  }
}

cat ("Cantidad de productos A, B y C a producir:", round (solucion, 4), "\n")
```

Método Gráfico.

Scilab

```
// Definición de variables
income_audit = 300;   // Ingreso por auditoría
income_settlement = 100; // Ingreso por liquidación

// Requerimientos de horas
hours_audit = 40;    // Horas requeridas para una auditoría
hours_settlement = 8; // Horas requeridas para una liquidación

// Disponibilidad de horas
available_hours_direct = 800;  // Total de horas de trabajo directo disponibles
available_hours_review = 320;  // Total de horas de revisión disponibles

// Límite de liquidaciones
max_settlements = 60; // Máximo de liquidaciones mensuales
```

```
// Definición del rango para las variables de decisión
x_audits = 0:0.1:30; // Cantidad de auditorías (de 0 a 30)

// Cálculo de y según las restricciones
y1 = (available_hours_direct - hours_audit * x_audits) / hours_settlement; // Restricción de horas de trabajo directo
y2 = (available_hours_review - 10 * x_audits) / 5; // Restricción de horas de revisión
y3 = max_settlements * ones(x_audits); // Restricción de máximo de 60 liquidaciones

// Limitar los valores negativos de y para representar solo la región factible
y1(y1 < 0) = 0;
y2(y2 < 0) = 0;

// Graficar las restricciones
clf; // Limpiar la figura
plot (x_audits, y1, 'b','thickness',2);
plot (x_audits, y2, 'r','thickness',2);
plot (x_audits, y3, 'g--','thickness',2);
xlabel ("Número de Auditorías (x)");
ylabel ("Número de Liquidaciones (y)");
title ("Método Gráfico para Maximizar Ingresos");
legend (["$40x_1 + 8x_2 \leq 800$", "$10x_1 + 5x_2 \leq 320$", "$x_2 \leq 60$"], "location", "northwest");
xgrid ();

// Rellenar el área factible
x_fill = [0, 0, x_audits (find (x_audits >= 0 & x_audits <= 30))];
y_fill = [0, 60, min(min(y1(find(x_audits >= 0 & x_audits <= 30)), y2(find(x_audits >= 0 & x_audits <= 30))),
y3(find(x_audits >= 0 & x_audits <= 30)))];
xfpoly (x_fill, y_fill, color (144, 238, 144));

// Calcular los vértices de la región factible y la función objetivo
vertices = [0, 0; 0, 60; 12, 40; 20, 0]; // Vértices de la región factible
Z = income_audit * vertices(:, 1) + income_settlement * vertices(:, 2); // Calcular Z = 300*x1 + 100*x2

// Mostrar los resultados en la consola
for i = 1:size (vertices, 1)
   mprintf ("Vértice %d: Auditorías = %d, Liquidaciones = %d, Ingreso = $%d\n", i, vertices (i, 1), vertices (i, 2),
Z(i));
end

// Encontrar el vértice con ingreso máximo
[max_profit, max_index] = max(Z);
optimal_audits = vertices (max_index, 1);
optimal_settlements = vertices (max_index, 2);

mprintf ("\nIngreso máximo se obtiene en el Vértice %d:\n", max_index);
mprintf ("Auditorías = %d, Liquidaciones = %d, Ingreso = $%d\n", optimal_audits, optimal_settlements,
max_profit);

// Marcar el punto óptimo en la gráfica
plot (optimal_audits, optimal_settlements, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'y');
xstring (optimal_audits + 0.5, optimal_settlements, sprintf ("(%d, %d)", optimal_audits, optimal_settlements), 1);
```

python

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Definición de las restricciones
x = np.linspace (0, 30, 400)

# Restricciones
y1 = (800 - 40 * x) / 8   # Restricción de trabajo directo
y2 = (320 - 10 * x) / 5   # Restricción de horas de revisión
y3 = np.full_like (x, 60)   # Restricción de máximo 60 liquidaciones

# Graficar las restricciones
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label=r'$40x_1 + 8x_2 \leq 800$', color='blue')
plt.plot(x, y2, label=r'$10x_1 + 5x_2 \leq 320$', color='red')
plt.plot(x, y3, label=r'$x_2 \leq 60$', color='green', linestyle='--')

# Configurar límites del gráfico
plt.xlim (0, 30)
plt.ylim (0, 70)
plt.xlabel ('Número de Auditorías ($x_1$)')
plt.ylabel ('Número de Liquidaciones ($x_2$)')
plt.title ('Método Gráfico para Maximizar Ingresos')

# Cambiar el color de la región factible a verde
plt.fill_between(x, 0, np.minimum(np.minimum(y1, y2), y3), where=(x <= 30),
color='lawngreen', alpha=0.3)

# Identificar los puntos de intersección
# Intersección entre las dos primeras restricciones (40x_1 + 8x_2 = 800 y
10x_1 + 5x_2 = 320)
x_opt = 12 # Auditorías óptimas
y_opt = 40 # Liquidaciones óptimas

# Vértices de la región factible
vertices = np.array([[0, 0], [0, 60], [12, 40], [20, 0]])

# Calcular la función objetivo de los vértices (Z = 300x_1 + 100x_2)
Z = 300 * vertices[:, 0] + 100 * vertices[:, 1]

# Imprimir los resultados
for i, (vx, vy) in enumerate(vertices):
    print (f"Vértice {i+1}: Auditorías = {vx}, Liquidaciones = {vy}, Ingreso
= {Z[i]}")

# Marcar el punto óptimo en el gráfico
plt.plot(x_opt, y_opt, 'bo', label='Punto Óptimo')
```

```
plt.text(x_opt + 0.5, y_opt, f'({x_opt}, {y_opt})', fontsize=12,
color='magenta')

# Mostrar leyenda y gráfico
plt.legend()
plt.grid()
plt.show()

# Solución: encontrar el máximo ingreso
max_profit_index = np.argmax(Z)
print (f"\nIngreso máximo se obtiene en el Vértice {max_profit_index+1}:")
print (f"Auditorías = {vertices [max_profit_index, 0]}, Liquidaciones =
{vertices[max_profit_index,1]}, Ingreso  = $ {Z[max_profit_index]}")
```

R

```
# Cargamos la librería necesaria
library(ggplot2)

# Definimos el rango de valores para las variables
x <- seq (0, 30, length.out = 400) # Número de auditorías

# Restricciones
y1 <- (800 - 40 * x) / 8   # Restricción de trabajo directo
y2 <- (320 - 10 * x) / 5   # Restricción de horas de revisión
y3 <- rep (60, length(x)) # Restricción de máximo 60 liquidaciones

# Graficar las restricciones
plot (x, y1, type = "l", col = "blue", ylim = c (0, 70), lwd = 3,
    xlab = "Número de Auditorías (x)", ylab = "Número de Liquidaciones (y)",
    main = "Método Gráfico para Maximizar Ingresos")

lines (x, y2, col = "red", lwd = 3)
lines (x, y3, col = "green", lty = 2, lwd = 3)
```

```
# Definir el rango para la región factible
x_fill <- seq (0, 20, length.out = 200) # Limitar hasta el máximo de auditorías

# Calcular los límites de la región factible
y_fill1 <- (800 - 40 * x_fill) / 8   # Trabajo directo
y_fill2 <- (320 - 10 * x_fill) / 5   # Revisión
y_fill3 <- rep (60, length(x_fill)) # Máximo 60 liquidaciones

# Limitar valores negativos para las restricciones
y_fill1[y_fill1 < 0] <- 0
y_fill2[y_fill2 < 0] <- 0

# Rellenar la región factible solo en el área válida
polygon (c (x_fill, rev(x_fill)),
        c (rep (0, length(x_fill)), rev(pmin(pmin(y_fill1, y_fill2), y_fill3))),
        col = rgb (0.5, 1, 0.5, 0.5))

# Identificar los puntos de intersección
# Solucionando 40x + 8y = 800 y 10x + 5y = 320
A <- matrix (c (40, 8, 10, 5), nrow = 2, byrow = TRUE)
b <- c (800, 320)
sol <- solve (A, b)

# Puntos óptimos
x_opt <- sol [1]
y_opt <- sol [2]
```

```
# Vértices de la región factible
vertices <- rbind (c (0, 0), c (0, 60), c (x_opt, y_opt), c(20, 0))


# Calcular la función objetivo de los vértices (Z = 300x + 100y)
Z <- 300 * vertices [, 1] + 100 * vertices [, 2]


# Imprimir los resultados
for (i in 1:nrow(vertices)) {
  cat (sprintf ("Vértice %d: Auditorías = %.2f, Liquidaciones = %.2f, Ingreso =
%.2f\n",
               i, vertices [i, 1], vertices [i, 2], Z[i]))
}
# Marcar el punto óptimo en el gráfico
points (x_opt, y_opt, col = "blue4", pch = 19)


# Etiquetar el punto óptimo al lado del punto
text (x_opt + 0.5, y_opt, labels = paste ("(", round (x_opt, 1), ",", round (y_opt, 1),
")", sep = ""),
     col = "magenta", pos = 4) # pos = 4 para que se posicione a la derecha
# Agregar la leyenda
legend ("topright", legend = c ("Trabajo Directo", "Revisión", "Máximo
Liquidaciones"),
       col = c ("blue", "red", "green"),
       lty = c (1, 1, 2),
       bty = "o", cex = 0.6)
# Mostrar los resultados finales
cat ("Número óptimo de auditorías (x):", round (x_opt, 2), "\n")
cat ("Número óptimo de liquidaciones (y):", round (y_opt, 2), "\n")
cat ("Ingreso total óptimo (Z):", max(Z), "$\n") # Se muestra el ingreso máximo
```