FACULTAD DE MATEMÁTICAS
GRADO EN ESTADÍSTICA

UNIVERSIDAD DE SEVILLA
FACULTAD DE MATEMÁTICAS

TRABAJO FIN DE GRADO

Problema de optimización mediante software

Alumno: D. Gonzalo Manuel Roa Muñoz

Dirigido por: D. Antonio Beato Moreno

Junio 2020

Agradecimientos

Con este proyecto cierro un ciclo de mi vida que me planteé como un desafío personal fuera de la edad óptima, pero que me ha servido para satisfacer la necesidad de completar unas carencias de formación que llevaba tiempo echando en falta en mi etapa profesional como docente.

He disfrutado aprendiendo de todas las áreas de conocimiento que he estudiado, aunque me ha costado un esfuerzo que sólo mi familia más cercana conoce verdaderamente. Por eso quiero agradecerles haber sido mi principal soporte apoyándome y ayudándome en aquellos momentos que más lo he necesitado, dándome la fuerza necesaria para continuar luchando en lo que creo y nunca rendirme. Gracias, también, a todos mis amigos, a mis compañeros y al equipo directivo del colegio Altair, donde estoy como docente, por demostrarme su aprecio y permitirme que pudiera compaginar trabajo y estudio. Por último, y no menos importante, quiero agradecer el trabajo de revisión de este proyecto a D. Antonio Beato Moreno, por ayudarme a mejorarlo y darme las pautas necesarias para la culminación de dicho proyecto con éxito.

Gonzalo Manuel Roa Muñoz

Resumen-abstract

Resumen

El estudio y el desarrollo en la Investigación Operativa han permitido un gran avance en el mundo empresarial, ya que su aplicación facilita la toma de decisiones, considerando la disponibilidad limitada de recursos.

Este desarrollo ha tenido un gran auge gracias a los avances tecnológicos, ya que permite llevar a cabo su resolución más rápida y segura. Además de procesar una gran cantidad de datos, de forma que de otra manera no fuese posible, o fuese muy tedioso su desarrollo y la obtención de la solución.

En este Trabajo Fin de Grado (TFG) pretendo analizar distintos problemas de optimización matemática con la utilización de distintos software aunque me centrare más en el paquete R.

Abstract

The study and development in Operations Research have allowed a great advance in the business world, since its application facilitates the decision making, considering the limited availability of resources.

This development has had a great boom thanks to the technological advances, since it allows us to carry out its resolution faster and more secure. In addition, to process a large amount of data, otherwise it would not be possible, or it would be very tedious to get and obtain the solution.

In this Final Degree Project (TFG) I intend to analyze different mathematical optimization problems with the use of different sodtware, although I will focus more on the R Package.

Introducción

El estudio de la optimización la programación matemática es muy interesante actualmente, debido a su gran versatilidad para la aplicación en el mundo empresarial. El estudio de esta rama ha sido posible a lo largo del grado, en diferentes asignaturas, si bien en algunas, con mayor profundidad.

No obstante, si consideramos objetivamente, el mundo tecnológico actual en el que nos encontramos, el desarrollo de este ámbito se lleva en gran medida a través del uso de herramientas informáticas. El conocimiento teórico de esta disciplina permite tener los fundamentos básicos necesarios para un desarrollo e investigación de mayor profundidad. Esta investigación puede verse facilitada con el uso de software informáticos, además de poder avanzar a una gran velocidad en su desarrollo y aplicación.

Además de contar con muchos tipos de programas informáticos desde hace años, el incesante aumento del volumen de software hasta la actualidad, nos abre la posibilidad de elegir entre una enorme variedad de herramientas informáticas, que nos ayudan a la puesta en práctica de Investigación Operativa. Muchas de ellas se desarrollan para un uso en concreto, de forma que sean específicos y abarquen dicho campo en su totalidad, mientras que otras se desenvuelven de forma genérica para poder englobar diferentes ámbitos. Considerando esta gran variedad de software, así como la incesante evolución de ambas clases de programas, el uso de un conjunto de las herramientas informáticas, nos puede llevar a la aplicación práctica completa de esta rama, sopesando las desventajas o errores que puedan esconderse en ellas, con el uso de otras, y así poder complementarse mutuamente.

De forma más concreta, a lo largo de su estudio en la Universidad, se ha desarrollado su aplicación con el uso de hojas de cálculo Excel. Han sido de gran utilidad, además de permitir que profundicemos en el aprendizaje de este software, son muy usadas actualmente en el mundo laboral. Por todo ello, el mayor estudio e investigación de las herramientas actuales puede ser muy interesante, ya que con el conocimiento básico de algunas de ellas, podremos desarrollar muchos otros tipos de problemas. Este puede ser el caso del software informático R (The R Project for Statistical Computing), el cual es muy usado en la actualidad, no solo en esta materia, sino en una gran variedad ámbitos, estudios, investigaciones etc. que podremos comprender si estudiamos los cimientos de esta herramienta.

Este proyecto se basará en el estudio e investigación del software informáticos, que permitan aplicar y facilitar la resolución de diferentes problemas de optimización.

Contenido

Capítulo 1

Fundamentos de Optimización

1.1 Conceptos básicos

La optimización de funciones (tanto de una cómo de varias variables) es uno de los problemas más clásicos a estudiar y resolver. En parte este interés se debe a la modelización de fenómenos reales mediante funciones y a la necesidad de conocer cómo se podría obtener el mejor comportamiento del fenómeno en base a los factores de los que depende éste. Por ejemplo, cuando una empresa estudia los niveles de coste de su producción en base a los distintos factores que intervienen en dicha producción (e. g. salarios, materia prima, impuestos…), está interesada en poder tomar una decisión sobre las imputaciones presupuestarias que hará a los distintos factores con el fin de obtener el menor coste posible. Igualmente, podríamos plantear el problema con niveles de beneficio y ahora se buscaría obtener las imputaciones para el mayor nivel de beneficio posible.

En esta memoria nos centraremos en problemas definidos con funciones reales definidas sobre $\mathbb{R}^n$. El fenómeno que se quiere estudiar (en los ejemplos anteriores el coste o el beneficio de la empresa) se modeliza mediante una función real de varias variables $\boldsymbol{f}: \mathfrak{R}^{\boldsymbol{n}} \rightarrow \mathfrak{R}$ en la que las variables de la función representan los distintos factores que afectan al fenómeno y la expresión algebraica de $\boldsymbol{f}$ se corresponde con la relación que existe (o se deduce empíricamente) entre dichos factores. Por tanto, si lo que queremos es el mayor (o el menor) valor que alcanza el fenómeno estudiado nos limitamos a optimizar la función.

Al ser uno de los problemas clásicos del Análisis Matemático, la optimización es uno de los contenidos habituales de la Investigación Operativa (I.O.).
Resulta imposible analizar el origen de la programación matemática de forma aislada de la investigación operativa, disciplina científica que estudia la toma de mejores decisiones mediante la aplicación de métodos analíticos avanzados. Por lo general, las aplicaciones de la investigación operativa se resuelven acudiendo a algún tipo de problemas de programación matemática, por lo que esta última es considerada cómo la piedra angular

de la investigación operativa.
La programación matemática esta presente día a día como por ejemplo las compañías aéreas planifican sus vuelos para minimizar los costes o maximizar los beneficios. Las líneas de autobuses planifican su ruta para abarcar el máximo de paradas recorriendo los menos kilómetros posibles etc.

Optimizar significa buscar la mejor manera de realizar una actividad, y en términos matemáticos, hallar el máximo o mínimo de una cierta función, definida en algún dominio. Para saber un poco mas sobre la historia de la programación matemática podríamos irnos a la referencia [1]

Para ver los elementos fundamentales de un problema de optimización ponemos el siguiente ejemplo, que aunque simple, nos va a dar una idea clara de los elementos

$$\begin{aligned} &\textit{Maximizar} && x \cdot y \cdot z \\ &\textit{sujeto a} && 2 \cdot (x \cdot y + y \cdot z + z \cdot x) = A \end{aligned}$$

$$x, y, z \geq 0$$

En este ejemplo se distinguen tres elementos fundamentales: las variables del problema, una función de esas variables y un conjunto de relaciones que deben cumplir las variables del problema. Estos elementos se repetirán en todos los problemas de optimización y se definen formalmente a continuación:

1.- *Variables de decisión:* El primer elemento clave en la formulación de problemas de optimización es la selección de las variables independientes que sean adecuadas para caracterizar los posibles diseños candidatos y las condiciones de funcionamiento del sistema. Como variables independientes se suelen elegir aquellas que tienen un impacto significativo sobre la función objetivo.

Representaremos las variables independientes se representarán mediante vectores columna de $\Re^n$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

o vectores fila

$$x^T = (x_1 \dots x_n)$$

Aunque para los casos $n = 1, 2$ y 3 se emplearán las notaciones usuales de $x, (x, y)$ y (x, y, z) respectivamente.

2.- *Restricciones:* Una vez determinadas las variables independientes, el siguiente paso es establecer, mediante ecuaciones o inecuaciones las relaciones existentes entre las variables de decisión. Estas relaciones son debidas, entre otras razones, a limitaciones en el sistema, a leyes naturales o a limitaciones tecnológicas y son las llamadas restricciones del sistema. Podemos distinguir dos tipos de restricciones:

(a) *Restricciones de igualdad:* Son ecuaciones entre las variables de la forma

$$h(x) = h(x_1 \dots x_n) = 0$$

donde $h: A \subseteq \Re^n \to \Re$ es una función real de variables reales definida sobre un conjunto A de números reales.

(b) *Restricciones de desigualdad:* Son inecuaciones entre las variables de la forma

$$g(x) = g(x_1 \dots x_n) \leq 0$$

donde $g: A \subseteq \Re^n \to \Re$ es una función real de variables reales definida sobre un conjunto A de números reales.

3.- *Función objetivo:* Finalmente, el último ingrediente de un problema de optimización es la función objetivo, también llamado índice de rendimiento o criterio de elección. Este es el elemento utilizado para decidir los valores adecuados de las variables de decisión que resuelven el problema de optimización. La función objetivo permite determinar los mejores valores para las variables de decisión.

Independientemente del criterio seleccionado, dentro del contexto de la optimización matemática el adjetivo "mejor" siempre indica los valores de las variables de decisión que producen el mínimo o máximo valor (según el criterio utilizado) de la función objetivo elegida.

Algunos de estos criterios pueden ser por ejemplo de tipo económico (coste total, beneficio), de tipo tecnológico (energía mínima, máxima capacidad de carga, máxima tasa de producción) o de tipo temporal (tiempo de producción mínimo) entre otros.

Con la introducción de estos tres elementos, el objetivo de los problemas de optimización matemática está claro: Un problema de optimización consiste en la búsqueda de valores para unas determinadas variables (*variables de decisión*) de forma que, cumpliendo un conjunto de requisitos representados mediante ecuaciones y/o inecuaciones algebráicas (*restricciones*) que limitarán la elección de los valores de las variables de decisión, proporcionan el mejor valor posible para una función (*función objetivo*) que es utilizada

para medir el rendimiento del sistema que se estudia. Como se ha comentado previamente, con el mejor valor se quiere indicar el mayor o el menor valor posible para la función objetivo. En resumen, buscamos valores que cumplan unas condiciones y minimicen o maximicen una función que caracteriza el sistema.

El planteamiento abstracto general para resolver problemas de este tipo es el siguiente

$$\begin{array}{ll} \text{Optimizar} & f(x_1 \dots x_n) \\ \text{Sujeto a} & \textit{Restricciones} \end{array}$$

donde el empleo del término Optimizar incluirá a ambos objetivos tanto de *Minimización* como de *Maximización*. No obstante y en relación a este aspecto, la mayoría de los planteamientos pueden hacerse con uno sólo de los objetivos, por ejemplo el de minimización, ya que un problema con objetivo de maximización se puede transformar en otro equivalente con objetivo de minimización multiplicando para ello la función objetivo por (-1). El mínimo de la función $f(x) = x^2 + 1$ se alcanza en el punto $x^* = 0$. En este punto también se alcanza el máximo de la función opuesta $g(x) = -f(x) = -x^2 - 1$, notar que aunque el punto buscado en ambos casos es el mismo, los valores que cada función toma en dicho punto son justamente uno el opuesto del otro:

$$f(x^*) = f(0) = 1$$
$$g(x^*) = g(0) = -1$$

1.2 Definiciones

En esta sección se dan las definiciones elementales relacionadas con la teoría de la optimización matemática con el objetivo de que el lector se familiarice con el lenguaje matemático utilizado.

Definición 1.1 (PPNL) *Se define el problema de programación no lineal (*PPNL*) al expresado como:*

$$\textit{(PPNL)} \quad \begin{cases} Optimizar & f(x_1, \dots, x_n) & & \\ sujeto\ a & h_i(x_1, \dots, x_n) = 0 & i = 1, \dots, m & \textit{(1.1)} \\ & g_j(x_1, \dots, x_n) \leq 0 & j = 1, \dots, p & \\ & (x_1, \dots, x_n) \in A \subseteq \Re & & \end{cases}$$

Donde $f, h_i, g_j: A \to \Re$, o en notación vectorial como:

(PPNL)

$$\begin{cases} Optimizar & f(x) \\ sujeto\ a & h_i(x) = 0 \\ & g_j(x) \leq 0 \\ & x = (x_1, \dots, x_n) \in A \subseteq \Re \end{cases}$$

Donde ahora $h: A \to \Re^m$ y $g: A \to \Re^p$ son funciones vectoriales.

Las funciones implicadas en la definición del problema fundamental no tienen por qué tener ninguna propiedad en particular, pero en nuestro caso vamos a introducir hipótesis adicionales que ayudarán a simplificar el problema; por ejemplo, supondremos de forma general que las funciones f, h_i y g_j son continuas y en la mayoría de los casos tendrán derivadas primeras y segundas también continuas.

La resolución del problema de optimización *PPNL* consistirá en primer lugar, en buscar valores para las variables de decisión x_i que cumplan las ecuaciones e inecuaciones que forman el sistema de las restricciones y en segundo lugar encontrar de entre estos valores aquel o aquellos que proporcionen el mayor (si el objetivo es maximizar) o menor (si el objetivo es minimizar) valor para la función real $f(x_1, \dots, x_n)$.

Definición 1.2 *Se definen algunos casos particulares del problema general de optimización 1.1.*

1. Problemas sin restricciones: *En este tipo de problemas no hay restricciones de ningún tipo, es decir m = p = 0. La expresión general para estos problemas es*

$$(PSR) \begin{cases} Optimizar\ f(x_1, \dots, x_n) \\ sujeto\ a\ (x_1, \dots, x_n) \in A \end{cases} \qquad (1.2)$$

Las únicas limitaciones vienen dadas por el conjunto A de $\Re^n$ donde esté definida la función $f(x_1, \dots, x_n)$.

2. Problemas de Lagrange o problemas sólo con restricciones de igualdad: *Son problemas de optimización con restricciones donde solamente existen restricciones de igualdad, por tanto* $m \neq 0$ *y* $p = 0$*. Son problemas de la forma:*

$$(PRI) \quad \begin{cases} Optimizar\ f(x_1, \dots, x_n) \\ sujeto\ a\ h_i(x_1, \dots, x_n) \qquad i = 1, \dots, m \\ (x_1, \dots, x_n) \in A \end{cases} \qquad (1.3)$$

No hay restricciones dadas por inecuaciones, sólo por ecuaciones.

3. Problemas unidimensionales o univariantes: *Este es un caso particular de los problemas sin restricciones en los que solamente hay una variable, es decir para* $n = 1, m = 0\ y\ p = 0$*. El problema se expresa como:*

$$(P1D) \quad \begin{cases} Optimizar\ f(x) \\ sujeto\ a \quad x \in I \subseteq \Re \end{cases} \qquad (1.4)$$

donde I *es, en la mayoría de las ocasiones, un intervalo.*

Definición 1.3 (Solución factible) Diremos que $\overline{x} = (\overline{x}_1, \dots, \overline{x}_n) \in A \subseteq \Re$ es una solución factible del problema PPNL (ecuación 1.1) si cumple todas sus restricciones, es decir:

$$\overline{x}\ solucion\ factible \iff \begin{cases} h_i(\overline{x}_1, \dots, \overline{x}_n) = 0 \quad i = 1, \dots, m \\ g_j(\overline{x}_1, \dots, \overline{x}_n) \leq 0 \quad j = 1, \dots, p \end{cases}$$

Definición 1.4 (Conjunto factible) se define región o conjunto factible Ω del problema PPNL al conjunto de todas sus soluciones factibles.

$$\Omega = \{\overline{x} \in A \subseteq \Re : \overline{x}\ es\ una\ solucion\ factible\}$$

Observación 1.3 *Con estas definiciones se puede decir que el objetivo al intentar resolver el problema de optimización* PPNL *es encontrar la "mejor" de todas las soluciones factibles.*

Definición 1.5 (Mínimo global) Diremos que $x^* = (x_1^*, \dots, x_n^*) \in \Omega \subseteq \Re^n$ es un

mínimo global del problema PPNL o que $f(x)$ tiene un mínimo global sobre Ω, el conjunto factible de PPNL, si

$$\forall \overline{x} \in \Omega;\ \ \overline{x} \neq x^* \Rightarrow f(x^*) \leq f(\overline{x})$$

El punto x^* *será* mínimo global estricto *si la desigualdad es estricta*

$$\forall \overline{x} \in \Omega;\ \ \overline{x} \neq x^* \Rightarrow f(x^*) < f(\overline{x})$$

Esta definición implica que no hay en Ω un punto en el que la función tome un valor menor que el que toma en x^*.

Definición 1.6 (Máximo global) Diremos que $x^* = (x_1^*, \dots, x_n^*) \in \Omega \subseteq \Re^n$es un máximo global del problema PPNL o que $f(x)$ tiene un máximo global sobre Ω, el conjunto factible de PPNL, si

$$\forall \overline{x} \in \Omega;\ \ \overline{x} \neq x^* \Rightarrow f(\overline{x}) \geq f(x^*)$$

El punto x^* *será* máximo global estricto *si la desigualdad es estricta*

$$\forall \overline{x} \in \Omega;\ \ \overline{x} \neq x^* \Rightarrow f(\overline{x}) > f(x^*)$$

Esta definición implica que no hay en Ω un punto en el que la función tome un valor mayor que el que toma en x^*.

Observación 1.4 *Los máximos y mínimos globales de un problema de optimización se denominan* extremos globales.

Definición 1.7 (Solución óptima) *Diremos que* $x^* \in \Omega \subseteq \Re^n$ *es una* solución óptima *del problema* PPNL *o que* $f(x)$ *tiene un óptimo en* x^* *sobre el conjunto factible* Ω *si ocurre alguna de estas dos situaciones*

1. x^* *es un mínimo global del problema* PPNL *y el objetivo del problema es minimizar.*
2. x^* *es un máximo global del problema* PPNL *y el objetivo del problema es maximizar.*

Definición 1.8 (Valor óptimo) Si $x^* \in \Omega \subseteq \Re^n$ es una solución óptima del problema PPNL, entonces se define el valor óptimo como el valor de la función objeto en la solución óptima, es decir, si x^* es una solución óptima del problema PPNL, entonces $f(x^*)$ es el valor óptimo.

Resolver un problema de optimización es encontrar, si existe, sus soluciones óptimas, es decir, los extremos globales de la función objetivo sobre el conjunto factible. Desde el punto de vista práctico y computacional en algunas ocasiones bastará con obtener los llamados extremos locales que se definen a continuación.

Definición 1.9 (Mínimo local) Consideremos el problema de optimización PPNL y sea Ω su conjunto factible. Diremos que $x^* \in \Omega \subseteq \Re^n$ es un mínimo local o relativo de $f(x)$ en Ω si y sólo si

$$\exists \varepsilon > 0 \; tal \; que \; \forall \overline{x} \in \Omega; \;\; \overline{x} \neq x^*; \;\; \|\overline{x} - x^*\| < \varepsilon \Rightarrow f(x^*) \leq f(\overline{x})$$

El punto x^* *será un* mínimo local estricto *de* $f(x)$ *en* Ω *si la desigualdad es estricta*

$$\exists \varepsilon > 0 \; tal \; que \; \forall \overline{x} \in \Omega; \;\; \overline{x} \neq x^*; \;\; \|\overline{x} - x^*\| < \varepsilon \Rightarrow f(x^*) < f(\overline{x})$$

Definición 1.10 (Máximo local) Consideremos el problema general de optimización PPNL y sea Ω su conjunto factible. Diremos que $x^* \in \Omega \subseteq \Re^n$es un máximo local o relativo de $f(x)$ en Ω si y sólo si

$$\exists \varepsilon > 0 \; tal \; que \; \forall \overline{x} \in \Omega; \;\; \overline{x} \neq x^*; \;\; \|\overline{x} - x^*\| < \varepsilon \Rightarrow f(x^*) \geq f(\overline{x})$$

El punto x^* *será un* máximo local estricto *de* $f(x)$ *en* Ω *si la desigualdad es estricta*

$$\exists \varepsilon > 0 \; tal \; que \; \forall \overline{x} \in \Omega; \;\; \overline{x} \neq x^*; \;\; \|\overline{x} - x^*\| < \varepsilon \Rightarrow f(x^*) > f(\overline{x})$$

Observación 1.5 *Los máximos y mínimos locales de los problemas de optimización también se denominan* extremos locales o relativos.

A diferencia de los extremos globales que afectan a todo el conjunto factible Ω, los extremos locales afectan a cierto entorno a su alrededor.

La teoría inicial asociada a la optimización está orientada a la obtención de condiciones necesarias y suficientes para que un punto sea óptimo. Por otra parte, también es interesante conocer no sólo si un punto es o no óptimo desde el punto de vista teórico, sino también cómo encontrar esos óptimos desde el punto de vista práctico. Teniendo esto en cuenta, al considerar problemas de optimización se plantean dos cuestiones:

1. ¿Cómo podemos determinar si un punto x^* es o no la solución óptima de un problema de optimización? ¿Qué condiciones deberían cumplir las funciones $f(x), h_i(x)\ y\ g_j(x)$ para que un problema *PPNL* tenga solución? ¿Qué condiciones debe cumplir el punto x^*?
2. Si $\overline{x}$ no es punto óptimo, entonces ¿cómo podemos encontrar una solución óptima x^* utilizando la información de la función en $\overline{x}$?

Mientras que con la primera cuestión se trata de determinar condiciones necesarias y/o suficientes para que un punto sea o no una solución óptima, en la segunda de las cuestiones se consideran los métodos numéricos adecuados para conseguir encontrar esas soluciones óptimas.

Un resultado básico utilizado para conocer si un problema de optimización tiene solución es el *teorema de Weierstrass*, que recordamos a continuación dentro del contexto de la optimización matemática.

Teorema 1.1 (Teorema de Weierstrass) Sea $f(x)$ una función continua definida sobre un conjunto compacto (cerrado y acotado) $K \subseteq \Re^n$. Entonces el problema de optimización

$$\begin{cases} Optimizar & f(x) \\ sujeto\ a & x \in K \end{cases}$$

Tiene al menos una solución para ambos objetivos de minimización y maximización, es decir

$$\exists\, x^*_{min}, x^*_{max} \in K: \begin{cases} f(x^*_{min}) = \min_{x\in K} f(x) \\ f(x^*_{max}) = \max_{x\in K} f(x) \end{cases}$$

Este es un resultado importante a tener en cuenta en la resolución de problemas de optimización, sin embargo el Teorema no nos proporciona un métodos para la localización de las soluciones, solamente de su existencia en determinadas condiciones. Desde el punto de vista de las aplicaciones, lo interesante es caracterizar los puntos solución y diseñar un método efectivo para su cálculo.
Finalmente, apuntar que un problema de optimización puede tener solución única como en el siguiente planteamiento

$$\left.\begin{array}{l}\min \quad x^2+y^2 \\ x+y=3\end{array}\right\}$$

$$x=y=\frac{3}{2}$$

No tener ninguna solución, como en el problema

$$Optimizar \quad \frac{1}{x}$$
$$sujeto\ a \quad x \in (0,1)$$

O tener más de una solución, como en el problema

$$Optimizar \quad sen(x)$$
$$sujeto\ a \quad x \in \Re$$

En el que hay incluso infinitas soluciones óptimas, tanto de mínimo $(\frac{3\pi}{2}+2k\pi, k \in \mathbb{Z})$ como de máximo $(\frac{\pi}{2}+2k\pi, k \in \mathbb{Z})$

1.3 Convexidad

1.3.1 Conjuntos convexos

El concepto de convexidad es de gran importancia en el estudio de los problemas de optimización desde el punto de vista de la aplicación práctica, puesto que en algunos casos, bajo condiciones de convexidad, se puede garantizar que un extremo local de un problema es realmente un extremo global y por tanto la solución óptima del problema buscada.

Se describen en esta sección algunos conceptos básicos de convexidad útiles para el desarrollo de la programación matemática y aunque es posible definirlos en el ámbito de cualquier espacio topológico, en lo sucesivo consideraremos el espacio vectorial $\Re^n$.

Definición 1.11 (Segmento lineal) Dados dos puntos $x, y \in \Re^n$ el segmento lineal cerrado que une *x* con *y* es el conjunto definida por

$$[x, y] = \{\lambda x + (1 - \lambda)y \in \Re^n : 0 \leq \lambda \leq 1\}$$

Del mismo modo, el segmento lineal abierto que une *x* con *y* es el conjunto definida por

$$(x, y) = \{\lambda x + (1 - \lambda)y \in \Re^n : 0 < \lambda < 1\}$$

Definición 1.12 (Conjunto convexo) *Sea* $\Omega \subset \Re^n$ *entonces*

$$\Omega \; es \; convexo \Leftrightarrow \forall x, y \in \Omega \Longrightarrow [x, y] \subseteq \Omega$$

Esta definición se interpreta de forma que un conjunto será convexo si el segment lineal cerrado que une cualquier par de puntos del conjunto está contenido en dicho conjunto.

La figura 1.2 representa algunos conjuntos convexos y otros no convexos de $\Re^2$.

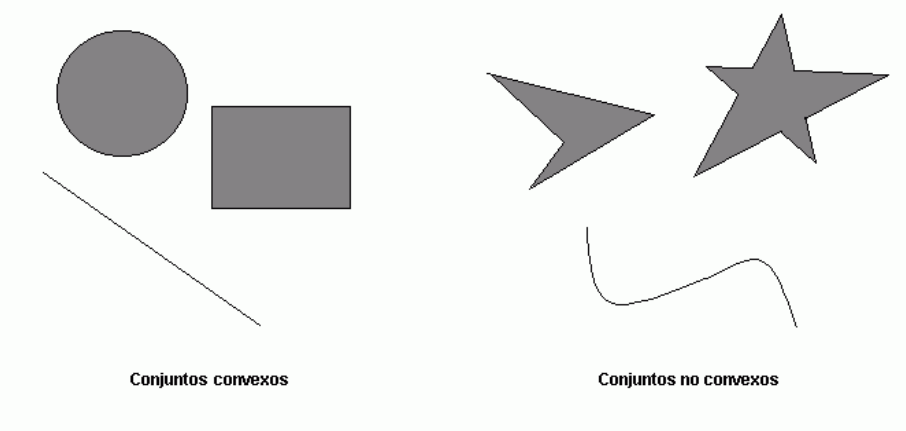


Figura 1.2: Convexidad en R^2

Por convenio el conjunto vacío ∅ es un conjunto convexo.
Uno de los tipos más importantes de conjunto convexo es el hiperplano, que definimos a continuación.

Definición 1.13 (Hiperplano) Sea $a \in \Re^n \; con \; a \neq 0 \; y \; b \in \Re$. Un hiperplano *H* en $\Re^n$ en un conjunto definido como

$$H = \{x \in \Re^n : a^t x = b\}$$

El vector *a* es el llamado vector normal al hiperplano. La expresión $a^t x$ es el producto escalar de ambos vectores

$$a^t x = a_1 x_1 + \cdots + a_n x_n$$

Definición 1.14 (Semiespacios) Sea $a \in \Re^n \; con \; a \neq 0 \; y \; b \in \Re$. Sea *H* un hiperplano construido a partir de *a* y *b* entonces definimos los semiespacios cerrados positivos y negativos asociados a *H*, respectivamente a los conjuntos

$$H_+ = \{x \in \Re^n : a^t x \geq b\}$$
$$H_- = \{x \in \Re^n : a^t x \leq b\}$$

Y semiespacios abiertos positives y negativos asociados a H a los conjuntos definidos como

$$H_+ = \{x \in \Re^n : a^t x > b\}$$
$$H_- = \{x \in \Re^n : a^t x < b\}$$

El siguiente lema es una consecuencia inmediata de la definición de convexidad y establece que la intersección de dos conjuntos convexos es convexa y que la suma algebraica de dos conjuntos convexos también es convexa.

Lema 1.1 Sean $\Omega_1, \Omega_2 \subseteq \Re^n$ dos conjuntos convexos, entonces

1. $\Omega_1 \cap \Omega_2$ es convexo.
2. $\Omega_1 + \Omega_2 = \{x + y \in \Re^n : x \in \Omega_1, y \in \Omega_2\}$ es convexo.
3. $\Omega_1 - \Omega_2 = \{x - y \in \Re^n : x \in \Omega_1, y \in \Omega_2\}$ es convexo.

Ejemplo. El conjunto definido como $S = \{x \in \Re^n : x \text{ es solución óptima de } P\}$ siendo *P* el problema

$$P = \begin{cases} Minimizar\ c^t x \\ sujeto\ a \quad Ax = b \\ \qquad\qquad x \geq 0 \end{cases}$$

Es un conjunto convexo

Solución: Para demostrar la convexidad de S utilizaremos la definición. Distinguimos tres casos:

1. Caso I: El problema *P* no tiene solución, en este caso *S* es el conjunto vacío ∅, que por definición es un conjunto convexo.

2. Caso II: El problema *P* tiene solución única: $S = \{x^*\}$. En este caso el único segmento lineal que se puede construir es $\lambda x^* + (1 - \lambda)x^* = x^* \in S$

   (en este caso incluso independientemente de si λ está en el intervalo [0, 1] o no).

3. Caso III: El problema *P* tiene más de una solución. En este caso *S* tiene más de un elemento y podemos suponer que existe $x^1\ y\ x^2 \in S$ que por ser soluciones del problema *P* serán puntos factibles y por tanto cumplirán las restricciones del problema.

$$Ax^1 = b$$
$$Ax^2 = b$$
$$x^1, x^2 \geq 0$$

Como además son mínimos de *P* ; si llamamos f^* al valor óptimo se tiene

$$f^* = c^t x^1 = c^t x^2$$

Ahora hay que comprobar que los elementos del segmento lineal que une x^1 con x^2 también están en S es decir son soluciones de P. para ello tomamos $\lambda \in [0,1]$ y consideramos el punto x^3 definido por

$$x^3 = \lambda x^1 + (1-\lambda)x^2$$

Comprobamos su factibilidad operando directamente. Por una parte x^3 es factible ya que por una parte cumple el sistema de ecuaciones

$$Ax^3 = A(\lambda x^1 + (1-\lambda)x^2) = \lambda A x^1 + (1-\lambda)Ax^2 = \lambda b + (1-\lambda)b = b$$

Y por otra parte, como $\lambda \in [0,1]$

$$0 \leq \lambda \leq 1 \Longrightarrow \begin{cases} \lambda \geq 0 \\ 1-\lambda \geq 0 \end{cases}$$

De donde al ser $x^1, x^2 \geq 0$ se obtiene

$$x^3 = \lambda x^1 + (1-\lambda)x^2 \geq 0$$

Ya que todos los sumandos son positivos. De forma x^3 es factible porque cumple las restricciones del problema.

Si ahora evaluamos la función objetivo en x^3

$$f(x^3) = c^t x^3 = c^t(\lambda x^1 + (1-\lambda)x^2) = \lambda c^t x^1 + (1-\lambda)c^t x^2 = \lambda f^* + (1-\lambda)f^* = f^*$$

Y por tanto se deduce que x^3 también es solución del problema *P*.

Definición 1.15 (Punto extremo) Sea $\Omega \subseteq \Re^n$ un conjunto convexo no vacío. Se dice que el punto $x \in \Omega$ es un vértice o punto extremo de $\Omega \Leftrightarrow$

$$Si\ x = \lambda x^1 + (1-\lambda)x^2 para\ algun\ \lambda \in [0,1] y\ x^1, x^2 \in \Omega \Longrightarrow x^1 = x^2 = x$$

Por ejemplo el conjunto convexo

$$\Omega = \{x = (x_1, x_2) \in \Re^2 \,/\, 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2\}$$

Tiene 4 puntos extremos dados por $P_1 = (0,0)$, $P_2 = (0,2)$, $P_3 = (2,0)\ y\ P_4 = (2,2)$ (ver figura 1.3).

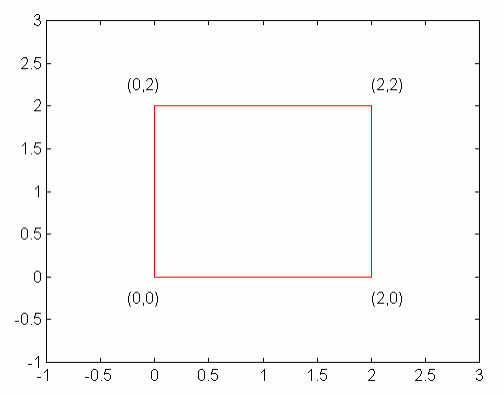


Figura 1.3: Puntos extremos

El conjunto de los puntos extremos de un conjunto convexo puede ser vacío, por ejemplo una bola abierta de $\Re^n$, contiene una cantidad finita de elementos, como en la figura 1.3 o tener una cantidad infinita de elementos, como una bola cerrada de $\Re^n$.

1.3.2 Funciones convexas

En esta sección se proporciona la definición de función convexa y se presentan alguna de sus propiedades más importantes, sobre todo aquellas que pueden utilizarse para resolver problemas de optimización.

Definición 1.16 (Función convexa) Diremos que la función $f: \Omega \subseteq \Re^n \longrightarrow \Re$, con Ω un conjunto convexo no vacío, es convexa sobre $\Omega \Leftrightarrow$

$$\forall x, y \in \Omega, \lambda \in [0,1] \Longrightarrow f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

Se dice que f es estrictamente convexa $\Leftrightarrow$

$$\forall x, y \in \Omega, \lambda \in [0,1] \Longrightarrow f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda) f(y)$$

Observación 1.7 Si definimos $\mu = 1 - \lambda$, entonces la definición puede ponerse como

$$f \text{ es convexa } \Leftrightarrow f(\lambda x + \mu y) \leq \lambda f(x) + \mu f(y), \qquad \forall \lambda, \mu \geq 0 \ \text{ con } \lambda + \mu = 1$$

Y del mismo modo para funciones estrictamente convexa.

Definición 1.17 (Función cóncava) Diremos que la función $f: \Omega \subseteq \Re^n \longrightarrow \Re$, con Ω un conjunto convexo no vacío, es cóncava sobre $\Omega \Leftrightarrow g = -f$ es convexa.
Esta definición equivale a decir que

$$f \text{ concava } \Leftrightarrow \forall x, y \in \Omega, \lambda \in [0,1] \Longrightarrow f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

Definición 1.18 Se dice que f es estrictamente cóncava $\Leftrightarrow g = -f$ es estrictamente convexa.

Proposición 1.3 Si $f_1(x) y\, f_2(x)$ son dos funciones convexas definidas sobre un conjunto convexo no vacío $\Omega \subseteq \Re^n \Longrightarrow f(x) = (f_1 + f_2)(x)$ es convexa sobre Ω.

Proposición 1.4 Si $f(x)$ es convexa sobre $\Omega \subseteq \Re^n$, siendo Ω un conjunto convexo no vacío, entonces $\forall \alpha > 0, la\ función\ (\alpha f)(x) definida\ por\ (\alpha f)(x) = \alpha f(x) es\ convexa\ sobre\ \Omega$.

Observación 1.8 Si $\alpha < 0$, entonces la función αf sería cóncava sobre Ω.

La caracterización de funciones convexas mediante su definición es, en general, muy difícil de aplicar en la práctica. Para comprobar si una función es o no convexa es necesario encontrar otras caracterizaciones más sencillas de aplicar. Los siguientes

resultados proporcional esas caracterizaciones para funciones convexas diferenciables en términos del gradiente y del Hessiano.

Proposición 1.5 (Caracterización de primer orden para funciones convexas) Sea

$f: \Omega \subseteq \Re^n \longrightarrow \Re, con\ \Omega\ un\ conjunto\ convexo\ y\ f(x) \in C^1(\Omega)$, es decir una función derivable en Ω con derivadas parciales continuas, entonces

$$f(x) es\ convexa\ sobre\ \Omega \Leftrightarrow f(y) \geq f(x) + \nabla f(x)(y-x)\ \ \forall x, y\ \in \Omega$$

$$f(x) es\ estrictamente\ convexa\ sobre\ \Omega \Leftrightarrow f(y) > f(x) + \nabla f(x)(y-x)\ \ \forall x, y\ \in \Omega$$

Proposición 1.6 (Caracterización de Segundo orden para funciones convexas)

Sea $f: \Omega \subseteq \Re^n \longrightarrow \Re, con\ \Omega\ un\ conjunto\ abierto\ convexo\ y\ f(x) \in C^2(\Omega)$, entonces

$f\ es\ convexa\ en\ \Omega \Leftrightarrow Hf(x) es\ semidefinida\ positiva\ \forall x \in \Omega$ siendo

$Hf(x) = \left[\left(\frac{\delta^2 f}{\delta x_i \delta x_j}(x)\right)_{i,j=1}^{n}\right]$ la matriz hessiana asociada a $f(x)$

1.3.3 Optimización de funciones convexas

En esta sección consideraremos el problema de minimizar y maximizar una función convexa sobre un conjunto convexo.
Debido a la correspondencia entre funciones cóncavas y convexas todos los resultados se presentan de forma equivalente para ambos tipos de funciones, por ello en cada teorema y entre corchetes se muestra el resultado alternativo.

Teorema 1.8 Sea $f: \Omega \subseteq \Re^n \longrightarrow \Re, con\ \Omega\ un\ conjunto\ convexo\ y\ f(x) \in C^1(\Omega)$ una función convexa [cóncava]. Supongamos que existe un $x^* \in \Omega$ que cumple la siguiente propiedad

$$\forall y \in \Omega \Longrightarrow \nabla f(x^*)^t (y - x^*) \geq 0 \quad [\nabla f(x^*)^t (y - x^*) \leq 0]$$

Entonces x^* es un punto de mínimo [máximo] global de f en Ω.

Proposición 1.7 Si Ω es un subconjunto no vacío, convexo y compacto de $\Re^n$ cuyo conjunto de puntos extremos es finito y si $f: \Omega \subseteq \Re^n \longrightarrow \Re$ es una función convexa [cóncava] sobre $\Omega \Longrightarrow f(x)$ posee en Ω un máximo [mínimo] global y se encuentra en uno de sus puntos extremos.

Teorema 1.9 Sea $f: \Omega \subseteq \Re^n \longrightarrow \Re$ convexa [cóncava] en Ω convexo y compacto $\Longrightarrow$ Si $f(x)$ tiene un máximo [mínimo] en Ω entonces lo alcanza en un punto extremo de Ω.

Lema 1.2 Si x^*, y^* son dos puntos de mínimo [máximo] global de una función convexa [cóncava] $f(x), f: \Omega \subseteq \Re^n \longrightarrow \Re$ con Ω convexo $\Longrightarrow \forall \lambda \in [0,1]$ los puntos definidos como $z^*(\lambda) = \lambda x^* + (1-\lambda) y^*$ también son mínimos [máximos] globales de $f(x)$.

Lema 1.3 Sea $f: \Omega \subseteq \Re^n \longrightarrow \Re$ con Ω un conjunto convexo no vacío y $f(x)$ convexa [cóncava]. Sea x^* un mínimo [máximo] local de f. Entonces, si x^* es un mínimo [máximo] local estricto de $f(x)$ o si $f(x)$ es estrictamente convexa [cóncava] sobre Ω, entonces x^* es el único mínimo [máximo] global de $f(x)$ en Ω.

Capítulo 2

Tipos de problemas de Optimización

Optimización hace referencia a la acción y efecto de optimizar. En términos generales, se refiere a la capacidad de hacer o resolver alguna cosa de la manera más eficiente posible y, en el mejor de los casos, utilizando la menor cantidad de recursos.
En las últimas décadas, el término optimización se ha vinculado al mundo de la informática. Sin embargo, es un concepto que también se utiliza en las matemáticas, en la gestión de procesos y la economía.

La optimización matemática es la elección del mejor elemento, dentro de un grupo más amplio de elementos disponibles. Estos problemas, que implican el uso de fórmulas para calcular valores óptimos, son llamados problemas de optimización, y forman parte de las matemáticas aplicadas.

La optimización matemática tiene varios subcampos, entre los que destacan:

- La optimización combinatoria, encargada de estudiar los problemas en los que el conjunto de soluciones puede ser reducido a uno, o puede ser discreto (divisible un número finito de veces)
- Optimización dimensional infinita: estudia los problemas cuyas soluciones se encuentran en un subconjunto de espacio de dimensión infinita (como por ejemplo, las funciones).
- Heurísticas y Metaheurísticas: se encargan de hacer suposiciones sobre un problema de optimización.

Otros subcampos son programación lineal, no lineal, cónica, de cono de segundo orden, geométrica, con enteros, semidefinida, cuadrática, fraccionaria, dinámica, entre otros.

2.1 Programación no restringida

Problemas sin restricciones, es decir, el problema se reduce a $\max f(x)$.

Si $f(x)$ es diferenciables, la condición necesaria para que $x = x^*$ sea óptima es

$$\frac{\partial f}{\partial x_j}|_{x=x^*} = 0, \forall j = 1,2, \dots, n.$$

La condición suficiente es que $f(x)$ sea cóncava (si el problema fuese de minimizar entonces la condición suficiente es que $f(x)$ sea convexa).

2.2 Programación restringida

2.2.1 Condiciones KKT

La clase de problema más general es la optimización no lineal o la programación no lineal (PNL).
Este es el problema donde al menos un $f_i, i = 1, \dots, m$ en la ecuación no es lineal. PNL no requiere que sea convexo, lo que dificulta en general obtener una solución global confiable.
Contrariamente al caso convexo, en una configuración no convexa, la mayoría de los algoritmos de optimización solo encuentran los extremos de f_0 en la vecindad del valor inicial (óptimo local).
Cuando un problema de PNL tiene sólo una o dos variables, se puede representar en forma gráfica. Si las funciones no son lineales, dibujaremos curvas en lugar de rectas, por lo que función objetivo y región factible dejarán de tener el aspecto que adquieren en la PL. La solución no tiene porqué estar en un vértice de la región factible, ni siquiera tiene porqué encontrarse en la frontera de esta. Por lo que desaparece ahora la gran simplificación que se utiliza en PL, que permite limitar la búsqueda de solución óptima a las soluciones en los vértices. Además en PNL un máximo local no es necesariamente un máximo global y en general, los algoritmos de PNL no pueden distinguir cuando se encuentra en un óptimo local o en uno global. Por tanto, es crucial conocer las condiciones en las que se garantiza que un máximo local es un máximo global en la región factible. Recuerde que en Análisis Matemático, cuando se maximiza una función ordinaria (doblemente diferenciable) de una sola variable f(x) sin restricciones, esta garantía está dada cuando

$$\forall x, \frac{d^2 f}{dx^2} \leq 0,$$

es decir, cuando la función es cóncava hacia abajo, o simplemente cóncava.

Si un problema de PNL no tiene restricciones, el hecho de que la función objetivo sea cóncava garantiza que un máximo local es un máximo global (de igual manera, una función objetivo convexa asegura que un mínimo local es un mínimo global). Si existen restricciones, se necesita una condicion mas para dar esta garantia, a saber, que la región factible sea un conjunto convexo. Por esta razón, los conjuntos convexos tienen un papel fundamental en la programación no lineal.

En la siguiente tabla se reflejan las condiciones de optimalidad para problemas de PNL restringidos o no

Tabla: Condiciones de optimalidad

Restricciones	Condiciones necesarias	Condiciones suficientes
No	$\nabla f(x) = 0$	$f(x)$ cóncava
$x_j \geq 0$	$\nabla f(x) = 0$ & $\nabla \mathrm{f(x)} \leq 0$ si $\mathrm{x_j} = 0$	$f(x)$ cóncava
General	Condiciones KKT	$f(x)$ cóncava y $g_i(x)$ convexas

Las condiciones suficientes es añadida a que se cumple la necesaria.

El trabajo de Karush [32] y posteriormente Kuhn y Tucker [33] constituyeron el gran avance en el desarrollo de las condiciones de optimalidad.

Teorema:

- Condiciones necesarias

Sean $f(x), g_1(x), \dots, g_m(x)$ funciones diferenciables que satisfacen ciertas condiciones de regularidad. Entonces $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ puede ser una solución óptima para el problema de PNL, solo si existen m números $u_1, u_2, \dots, u_m$ que satisfagan las siguientes condiciones de Karush-Kuhn-Tucker (KKT):

1. $\frac{\partial f(x^*)}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i(x^*)}{\partial x_j} \leq 0, j = 1,2, \dots, n$
2. $x_j^* \left(\frac{\partial f(x^*)}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i(x^*)}{\partial x_j} \right) = 0, j = 1,2, \dots, n$
3. $g_i(x^*) - b_i \leq 0, i = 1,2, \dots, m$
4. $u_i(g_i(x^*) - b_i) = 0, i = 1,2, \dots, m$
5. $x_j^* \geq 0, j = 1,2, \dots, n$
6. $u_i \geq 0, i = 1,2, \dots, m$

- Condiciones suficientes

Sea $f(x)$ cóncava y $g_1(x), g_2(x), \dots, g_m(x)$ convexas (es decir, se trata de un problema de programación convexa), donde todas estas funciones satisfacen las condiciones de regularidad y x^* satisface las condiciones KKT. Entonces, x^* es una solución óptima.

2.2.2 Programación lineal

La programación lineal es el campo de la programación matemática dedicado a maximizar o minimizar (optimizar) una función lineal, denominada función objetivo, de tal forma que las variables de dicha función estén sujetas a una serie de restricciones expresadas mediante un sistema de ecuaciones o inecuaciones también lineales.

El problema de la resolución de un sistema lineal de inecuaciones se remonta, al menos, a Joseph Fourier, después de quien nace el método de eliminación de Fourier-Motzkin. La programación lineal se plantea como un modelo matemático desarrollado durante la Segunda Guerra Mundial para planificar los gastos y los retornos, a fin de reducir los costos al ejército y aumentar las pérdidas del enemigo. Se mantuvo en secreto hasta 1947. En la posguerra, muchas industrias lo usaron en su planificación diaria.

Los fundadores de la técnica son George Dantzig, quien publicó el algoritmo simplex, en 1947, John von Neumann, que desarrolló la teoría de la dualidad en el mismo año, y Leonid Kantoróvich, un matemático de origen ruso, que utiliza técnicas similares en la economía antes de Dantzig y ganó el premio Nobel en economía en 1975. En 1979, otro matemático ruso, Leonid Khachiyan, diseñó el llamado Algoritmo del elipsoide, a través del cual demostró que el problema de la programación lineal es resoluble de manera eficiente, es decir, en tiempo polinomial. Más tarde, en 1984, Narendra Karmarkar introduce un nuevo método del punto interior para resolver problemas de programación lineal, lo que constituiría un enorme avance en los principios teóricos y prácticos en el área.

El ejemplo original de Dantzig de la búsqueda de la mejor asignación de 70 personas a 70 puestos de trabajo es un ejemplo de la utilidad de la programación lineal. La potencia de computación necesaria para examinar todas las permutaciones a fin de seleccionar la mejor asignación es inmensa (factorial de 70, 70!) ; el número de posibles configuraciones excede al número de partículas en el universo. Sin embargo, toma sólo un momento encontrar la solución óptima mediante el planteamiento del problema como una programación lineal y la aplicación del algoritmo simplex. La teoría de la programación lineal reduce drásticamente el número de posibles soluciones factibles que deben ser revisadas.

Asi pues un problema de programación lineal se puede expresar de la siguiente forma:

$$optimizar\ a_0^t \cdot x$$
$$sujeto\ a\ A \cdot x \leq b$$

Donde x es el vector de variables objetivas que debe optimizarse. Los coeficientes de la función objetivo está representada por $a_0 \in \Re^n$. $A \in \Re^{m\ x\ n}$ es una matriz de coeficientes representando las restricciones del problema lineal. $Ax \leq b$ podría escribirse como

$a_i^t \cdot x \leq b_i, \quad i = 1, \ldots, m$ (donde a_i *se* refiere a la *i*-ésima fila del coeficiente de la matriz A). Todos los problemas lineales son convexos y generalmente se resuelven mediante métodos de punto interior o simplex. . Para más información sobre el origen y las propiedades matemáticas de estos métodos remito al lector al libro de Nocedal y Wright (2006).

2.2.3 Programación cuadrática

La programación cuadrática (QP) es el nombre que se le da a un procedimiento que minimiza una función cuadrática de n variables sujeta a m restricciones lineales de igualdad o desigualdad. Un programa cuadrático es la forma más simple de problema no lineal con restricciones de desigualdad. La importancia de la programación cuadrática es debida a que un gran número de problemas aparecen de forma natural como cuadráticos (optimización por mínimos cuadrados, con restricciones lineales), pero además es importante porque aparece como un subproblema frecuentemente para resolver problemas no lineales más complicados. Las técnicas propuestas para solucionar los problemas cuadráticos tienen mucha similitud con la programación lineal.
Específicamente cada desigualdad debe ser satisfecha como igualdad. El problema se reduce entonces a una búsqueda de vértices exactamente igual que se hacía en programación lineal.
Un problema cuadrático es una generalización del problema lineal estándar, dónde la función objetivo contiene una parte cuadrática además del término lineal. La parte cuadrática está representada por una matriz $Q_0 \in \Re^{n\,x\,n}$. Por lo tanto, los problemas cuadráticos se pueden expresar en el siguiente manera:

$$optimizar \ \frac{1}{2} x^t \cdot Q_0 \cdot x + a_0^t \cdot x$$
$$sujeto\ a\ A \cdot x \leq b$$

Dado que las restricciones son lineales y presumiblemente independientes la cualificación de las restricciones se satisface siempre, así pues, las condiciones de Karush-KuhnTucker son también condiciones suficientes para obtener un extremo, que será además un mínimo global si Q es definida positiva. Si Q no es definida positiva el problema podría no estar acotado o llevar a mínimos locales.

La Programación Cuadrática juega un papel muy relevante en la teoría de optimización lineal y no lineal pues guarda una relación muy estrecha con la Programación Lineal y es un paso intermedio esencial para resolver eficazmente problemas generales de Programación No Lineal.

A diferencia de los problemas lineales, no todos los cuadráticos son convexos. Un problema cuadrático es convexo si y solo si Q_0es semidefinida positiva ($z^t \cdot Q_0 \cdot z \geq 0\ para\ todo\ z \in \Re^n$ no nulo).

2.2.4 Programación cónica

La programación cónica se refiere a una clase de problemas diseñados para modelar problemas de optimización convexos.
Un problema de programación cónica se puede expresar genéricamente de la siguiente forma:

$$\begin{aligned} &\text{minimize } a_0^t \cdot x \\ &sujeto\ a\ A \cdot x + s = b \\ &s \in \varkappa \end{aligned}$$

También llamado problema primal de la programación cónica, donde $x \in \Re^n$ es el vector de variables de decisión, $a_0 \in \Re^n, b \in \Re^m, A \in \Re^{n\ x\ m}$ son datos conocidos referentes al problema y $\varkappa \subset \Re^n$ es un cono convexo, cerrado con un interior no vacío. Los conos más relevantes que satisfacen estas propiedades son:

- Octante positivo.
- Cono de Lorentz o de segundo orden.
- Cono positivo semidefinido.

La forma estándar de CP como se da en la ecuación minimiza un objetivo lineal sobre un convexo cono (*b* - *Ax* = *s* ∈ K). Como Nemirovski (2006) señala que la representación de CP en este formulario tiene dos ventajas principales.

Primero, esta formulación tiene fuertes habilidades unificadoras, lo que significa solo algunos conos permiten modelar muchos tipos diferentes de OP. Además, las no linealidades ya no están representados por objetivos generales no lineales y funciones de restricción sino vectores y matrices que permiten a los algoritmos utilizar la estructura presente en los OP convexos.

En segundo lugar, la convexidad está incorporada en la definición de CP. Al mismo tiempo, teóricamente,cualquier OP convexo puede reformularse en la forma dada en la ecuación. Por lo tanto no lineal las funciones objetivas se expresan en forma de epígrafe (véase, por ejemplo, Boyd y Vandenberghe 2004):

$$\begin{aligned} &minimize\ t \\ &sujeto\ a\ f_0(x) \leq t \\ &f_i(x) \leq b_i \end{aligned}$$

Prácticamente el número de problemas cónicos que se pueden resolver está limitado por el número de conos soportado por un solver de optimización dado. Los solucionadores de última generación distinguen entre hasta ocho tipos diferentes de conos. Siguiendo las definiciones en Diamond y Boyd (2015) y O'Donoghue *y col.* (2016), un cono convexo K suele ser un producto cartesiano de simples conos convexos de los siguientes tipos.

2.2.5 Programación entera mixta

Un modelo de programación entera es aquel que contiene restricciones y una función objetivo idénticas a la formuladas en programación lineal, la única diferencia en que una o mas variables de decisión deben tomar valor entero en la solución final.

Existen tres tipos de modelos por programación entera:

A) PURA: Son modelos similares a los de programación entera.

B) BINARIA: Estos modelos lineales, las variables sólo toman valores 0 y 1, son usadas para uso probabilístico. Donde 0 se rechaza la opción y 1 se acepta la opción.

C) MIXTA: En estos tipos de modelos, integra las variables puras y las mixtas.

Los tipos de restricciones más usadas en la Programación Entera Mixta son:

1) Excluyentes: Solo sirve para elegir una alternativa de varias posibles.

2) Pre-requisito: Cuando necesitas realizar una acción antes de proceder con la siguiente.

3) Incluyente: Dicha restricción se da para cuando realizas una acción "A" entonces debes hacer la acción "B".

4) Costo Fijo: Cuando se nombra un costo fijo, es sinónimo de uso de variable mixta.

2.2.5.1 Problemas de redes

Los problemas de redes surgen en una gran variedad de situaciones. Las redes de transporte, eléctricas y de comunicaciones predominan en la vida diaria. La representación de redes se utiliza ampliamente en áreas tan diversas como producción, distribución, planeación de proyectos, localización de instalaciones, administración de recursos y planeación financiera, para nombrar sólo unos ejemplos. De hecho, una representación de redes proporciona un panorama general tan poderoso y una ayuda conceptual para visualizar las relaciones entre los componentes del sistema, que se usa casi en todas las

áreas científicas, sociales y económicas.

Uno de los mayores desarrollos recientes en investigación operativa (IO) ha sido el rápido avance tanto en la metodología como en la aplicación de los modelos de optimización de redes. La aparición de algunos algoritmos ha tenido un impacto importante, al igual que las ideas de ciencias de la computación acerca de estructuras de datos y la manipulación eficiente de los mismos. En consecuencia, ahora se dispone de algoritmos y paquetes de computadora y se usan en forma rutinaria para resolver problemas muy grandes que no se habrían podido manejar hace dos o tres décadas.

2.2.5.2 Problemas del viajante

El problema del viajante (*TSP* por sus siglas en inglés (Travelling Salesman Problem)), responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?

2.2.5.3 Problemas de ruta de vehículos

Los problemas de rutas de vehículos son una generalización de problema del viajero.

El problema del viajero (TSP) es uno de los más famosos y estudiados. En este problema, un viajante de comercio tiene que visitar un conjunto de clientes o ciudades, y regresar después al punto de partida. Antes de salir, deberá decidir en qué orden recorrer dichas ciudades para minimizar la distancia total recorrida. Señalar que tiene que pasar una sola vez por cada ciudad.

2.2.6 Otros problemas de optimización

2.2.6.1 Problemas Multiobjetivos

En la vida real, existen numerosas situaciones y problemas que son reconocidos como problemas multiobjetivo, es decir, no poseen un único criterio medible por el cual pueda declararse que una solución sea completamente satisfactoria. Dicho de otra forma, este tipo de problemas contiene múltiples criterios que han de satisfacerse o que han de ser tenidos en cuenta. A menudo dichos criterios entran en conflicto unos con otros y no existe una única solución que simultáneamente satisfaga a todos. Por tanto, la solución que se pretenda obtener debe estar en concordancia con las preferencias del decisor.
Los problemas de optimización de rutas de vehículos (VRP) se modelan normalmente

como un problema de optimización con un único objetivo, que minimiza el coste de llevar alguna mercancía a unos destinos, al tiempo que se satisfacen ciertas restricciones. No obstante, en la vida real se pueden tener en cuenta otros factores, como son: la minimización de distancias, minimizar tiempos de espera, buscar un equilibrio en las cargas de trabajo (tiempo, distancia,…)

En la optimización multiobjetivo, cada una de las funciones objetivo representa una aptitud o un costo a optimizar, las cuales dependen de n parámetros. Sin pérdida de generalidad. En particular, la r-ésima aptitud está dada por una función $f_r: \Re^n \to \Re$ y la función evaluadora del problema a optimizar está compuesta por dichas funciones de aptitud. Suponiendo que se deben optimizar m aptitudes, la función a optimizar es de la forma $F: \Re^n \to \Re^m$ y el problema general está dado por:

$$\max_x \ F(x) = (f_1(x), f_2(x), \dots, f_m(x))$$
$$sujeto\ a\ P_i \qquad con\ i = 1, 2, 3, \dots, k$$

donde $x \in \Re^n$ y la i-ésima restricción está dada por la proposición P_i. En la búsqueda de la solución óptima del problema general, se trabaja sobre un espacio de posibles soluciones en $\Re^n$, donde todas satisfacen un criterio multiobjetivo de optimalidad. En otras palabras, el objetivo es optimizar de manera simultánea todas las funciones objetivo, puesto que en la mayoría de los problemas no es posible encontrar un solo valor óptimo para todas ellas, ya que el óptimo de una función no necesariamente lo es para las demás (Zitzler et al., 2003).

Para ampliar los problemas multiobjetivos pueden dirigirse a apartado [27] de la Bibliografia.

2.2.6.2 Optimización FUZZY

Desde que Zadeh [28] introdujo el concepto de número difuso y Chang y Zadeh [29] propusieron la noción de aplicación difusa, se realizan numerosos estudios en las condiciones de optimalidad para los problemas difusos, problemas de programación matemática en los que la función objetivo es difusa.
El **agrupamiento difuso** (en inglés, ***fuzzy clustering***) es una clase de algoritmos de agrupamiento donde cada elemento tiene un grado de pertenencia difuso a los grupos.
Este tipo de algoritmos surge de la necesidad de resolver una deficiencia del agrupamiento exclusivo, que considera que cada elemento se puede agrupar inequívocamente con los elementos de su *cluster* y que, por lo tanto, no se asemeja al resto de los elementos. Tras la introducción de la lógica difusa por Zadeh en 1965 surgió una solución para este problema, caracterizando la similitud de cada elemento a cada uno de los grupos [30]. Esto se logra representando la similitud entre un elemento y un grupo por una función, llamada función de pertenencia, que toma valores entre cero y uno. Los valores cercanos a uno indican una mayor similitud, mientras que los cercanos a cero indican una menor similitud. Por lo tanto, el problema del agrupamiento difuso se reduce a encontrar una caracterización de este tipo que sea óptima.

Los algoritmos de agrupamiento difuso se han aplicado ampliamente en diferentes áreas como el procesamiento de imágenes, sistemas de ingeniería, estimación de parámetros, entre otras [31].

2.2.6.3 Optimización global

Optimización Global es una rama de la matemática aplicada y el análisis numérico que se ocupa de la optimización de una función o un conjunto de funciones de acuerdo a diferentes criterios.

Un modelo en su forma estándar es la minimización de una función real f en el espacio de los parámetros $\vec{x} \in P$, o en el subespacio definido por las restricciones D. (El problema de maximizar la función $g(x)$ es equivalente a minimizar una función $f(x) = (-1) \cdot g(x)$).

En muchos problemas de optimización no lineal, la función objetivo tiene un gran número de mínimos y máximos locales, encontrar un óptimo local es una tarea sencilla, usando métodos clásicos de optimización local. En estos casos encontrar un mínimo o un máximo global es una tarea de mayor complejidad, pues los métodos analíticos no se pueden utilizar en la gran mayoría de los casos y las estrategias numéricas traen consigo un número grande de problemas.

Algunos de los casos en los que se puede usar un enfoque de optimización global son:

- Predicción de la estructura de las proteínas (minimizar la energía/función de energía libre).
- Filogenética computacional (por ejemplo, minimizar el número de transformaciones de caracteres del árbol).
- Problema del viajante y el diseños de circuitos eléctricos (minimizar la longitud del camino).
- Ingeniería química (por ejemplo, analizar la energía libre de Gibbs).
- Verificación de seguridad, ingeniería de seguridad (por ejemplo, de estructuras mecánicas, edificios).
- Análisis del caso peor.
- Problemas matemáticos (por ejemplo, la conjetura de Kepler).
- El punto de partida de numerosas simulaciones dinámicas moleculares que consisten en una optimización inicial de la energía del sistema a ser simulado.
- Calibración de los modelos de propagación de radio y otros modelos en la ciencias y en la ingeniería.
- Ajustes de curvas como el análisis mínimo cuadrático y otras generalizaciones, como el ajuste de los datos de modelos experimentales, en la química, física, medicina, astronomía y otros.

Capítulo 3

Software

3.1 Visión general

En este capítulo se hace revisión de las herramientas más importantes de optimización convexa que podemos encontrar. Se estudian las caracteristicas de cada programa de forma que podemos conocer cual es más adecuado para el problema que necesitemos resolver, en función de diferentes propiedades como su complejidad o facilidad de uso, condiciones comerciales de distribución, y la variedad de problemas y formatos de datos que admita.
Los principales apartados que encontraremos corresponden a los tipos de programas que existen, que pueden ser de modelado o de resolución. Veremos la diferencia entre ambos.
También he reservado un espacio para herramientas de optimización global orientadas a problemas no necesariamente convexos.
Al final del capítulo se incluye un ejemplo de problema cuadrático que nos permite comparar el procedimiento de utilización de varias herramientas, y que también sieve para visualizar de forma intuitiva la estructura de los problemas que estamos tratando.
En estos momentos es posible para un usuario adquirir una gran variedad de aplicaciones que resuelven problemas de optimización, pero estos pueden ser de una naturaleza muy diversa, por lo que cada herramienta se centra en la resolución de un tipo o de un grupo más o menos amplio de problemas. Por ejemplo, si buscamos herramientas de **optimización lineal**, la cantidad de programas que encontramos en el mercado es innumerable; incluso en las hojas de cálculo de los paquetes de ofimática más populares se pueden resolver.

Cuando los problemas son **convexos**, las técnicas disponibles de resolución permiten una altísima eficiencia en cuanto a tiempo de convergencia, dimensiones de las variables, y cantidad de funciones de restricción que podemos indicar.

Podemos distinguir entre los programas específicos de resolución (***solvers***), que suelen estar especializados en algún tipo de problema (lineales, cuadráticos, cónicos, etc.); y por otro lado los sistemas de modelado, que se encargan de automatizar las transformaciones necesarias para convertir un problema en su formato estándar.

Los ***solvers*** corresponden a un nivel bajo de programación, en el sentido de que son más complicados para el usuario, y están enfocados a aplicar un determinado algoritmo para un tipo de problema en concreto. Normalmente forman parte del “núcleo” de los

programas de modelado, que a su vez, pueden utilizar una gran cantidad de *solvers* para resolver en cada caso un conjunto variado de problemas.

Los sistemas de modelado proporcionan un nivel alto de programación. No están especializados en un tipo de problema concreto, porque para eso cuentan con varios *solvers* una vez que han transformado el problema genérico para el que se utilizan.

3.2 Herramientas de resolución

La cantidad de herramientas disponibles en el mercado para resolución de problemas de optimización es muy grande. En esta sección se analizan algunas de ellas agrupándolas por el tipo de problemas que resuelven. Veremos que la selección por el tipo de problemas abarcado no es un criterio exacto, porque muchos *solvers* pueden tratar más de un tipo de problemas. Por ejemplo, las herramientas de resolución cuadrática resuelven también problemas lineales, ya que estos se pueden ver como un caso particular de los anteriores. En definitiva, para asignar cada herramienta a un tipo de problema me he basado en la clasificación que de ellas hacen los programas de modelado que las incluyen.

3.2.1 Problemas Lineales

Este grupo de programas es el más extenso. En este apartado presentamos o introducimos *lp_solve*, *GLPK*6, *BPMPD* y *MINOS*, aunque en el entorno académico puede resultar más accesible utilizar en *Matlab* el comando linprog del complemento *Matlab Optimization Toolbox* o incluso los *solvers* de *Microsoft Excel* y *OpenOffice*.

3.2.1.1 lp_solve

Es un programa de resolución de problemas lineales que utiliza el método símplex y el de ramificar y acotar (*Branch-and-Bound*) para problemas de variables enteras y conjuntos especiales ordenados. La información sobre esta herramienta procede de [10].

El tamaño de modelo admitido no tiene límite, y la entrada de datos y especificaciones se hace básicamente de tres formas:

- Por medio de las bibliotecas de interfaz de programación.
- Con ficheros de entrada.
- A través del entorno integrado de desarrollo.

Para la entrada por archivo acepta formatos estándar *lp* o *mps*. Se puede enlazar como biblioteca desde C, Visual Basic, .NET, Delphi, Excel, Java,. . .

También se puede utilizar desde *AMPL*10, *Matlab*, *O-Matrix*, *Scilab*, y *Octave*.

Se distribuye como *software* libre con licencia LGPL.

3.2.1.2 GLPK

Es una biblioteca GNU para resolver problemas lineales utilizando métodos de punto interior, que también puede trabajar con problemas de programación lineal entero mixto (MIP). Ofrece interfaces para *Matlab* y para el lenguaje de modelado *AMPL*. Está compuesto por un conjunto de rutinas escritas en ANSI11 C, y el paquete incluye componentes que implementan los siguientes algoritmos:

• Símplex dual y primal.
• Punto interior primal-dual.
• Ramificación y corte.

También se incluye el interfaz de programación o *API* junto con el *solver*. La información de esta herramienta procede de [11].

GLPK es parte del proyecto GNU, y se ofrece bajo la licencia pública general GNU-GPL.

3.2.1.3 BPMPD

Se trata de una implementación del método de punto interior primal-dual escrita en C. Es una herramienta para programas lineales, aunque en la última versión se ha añadido también soporte para programas cuadráticos.

La información sobre *BPMPD* procede de [12] y de la dirección *web* en la que esta cita se encuentra.

La entrada es por archivos *mps* a los que ya nos hemos referido antes.

Su eficiencia se basa en un tratamiento previo muy depurado de los datos antes del algoritmo:

• Se eliminan filas y columnas vacías, y filas o columnas con un solo elemento.
• Se eliminan columnas libres con un solo elemento.
• Se quitan restricciones redundantes, tanto en el problema primal como en el dual.
• Se sustituyen variables duplicadas.

- Se quitan cotas redundantes para las variables.
- Se eliminan variables libres.
- Se construye una matriz de restricciones dispersa.

Este programa fue desarrollado por Csaba Mészáros en el MTA SZTAKI (Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézete) de Hungría (Instituto de investigación de informática y automática) y su autor permite descargarlo libremente como ejecutable para *Windows* o como *DLL* (Dynamic Link Library).

3.2.1.4 MINOS

Es un paquete escrito en Fortran que sirve para resolver problemas de optimización a gran escala. Además de LP también resuelve problemas no lineales, aunque es especialmente eficiente cuando se trata de restricciones dispersas y lineales.

Utiliza una implementación dispersa del método símplex para problemas lineales. Para funciones no lineales utiliza un método de gradiente reducido, con aproximaciones casi-Newton para el hessiano reducido. Si se incluyen restricciones no lineales, se utiliza un algoritmo de lagrangiano proyectado relacionado con el método de Robinson. Este método resuelve una secuencia de subproblemas en los que las restricciones se linealizan y el objetivo es un lagrangiano aumentado que incluye las funciones lineales y no lineales. Con este método la convergencia aproximada es rápida.

La información de esta herramienta procede de [13] y de la *web* en la que se encuentra esta referencia.

La licencia de este programa es comercial. El propietario es *Stanford Business Software, Inc.*, y ofrece precios para clientes particulares o empresas y para organismos académicos en formato individual, para departamento o para corporaciones completas. También hay una versión de evaluación para tres meses, pero no es gratuita.

3.2.2 Problemas Cuadráticos

Para resolver problemas cuadráticos de grandes dimensiones podemos utilizar las herramientas que se estudian en este apartado, *CPLEX*, *Gurobi*, *Xpress-Optimizer*, y *OOQP*14, y muchas otras que no he abordado para poner un límite razonable de tamaño en esta revisión. También se pueden resolver problemas cuadráticos utilizando programas orientados a problemas cónicos, ya que son un caso particular de estos. Estas herramientas sirven también para resolver problemas lineales, ya que los LPs son un subconjunto de los problemas cuadráticos. Por otro lado, el complemento *Matlab Optimization Toolbox* nos permite utilizar el comando quadprog para resolver problemas cuadráticos con restricciones lineales.

3.2.2.1 CPLEX

IBM ILOG CPLEX es una herramienta de optimización ampliamente utilizada en la industria para estudios de alternativas, cuellos de botella o inconsistencias en las tomas de decisiones, así como para desarrollar planes y horarios que se adapten al curso de las operaciones. Se utilizan en diseño de líneas aéreas, planes de producción, gestión de carteras, o planificación temporal de tareas.

Los problemas que resuelve *CPLEX Optimizer* pueden ser **lineales, cuadráticos, con variables enteras, con restricciones cuadráticas, y también cónicos de segundo orden**. Esto último hace que podría haber puesto esta herramienta en el siguiente apartado, pero en muchos programas se hace referencia a *CPLEX* como un *solver* para programación cuadrática, probablemente por su capacidad en versiones más antiguas. También indica en sus especificaciones que se puede utilizar para problemas con restricciones, o de viabilidad. Los datos sobre esta herramienta proceden de [14].

Todos los algoritmos vienen integrados con métodos de procesamiento previo para reducir el tamaño de los problemas y con ello el tiempo de resolución sin necesidad de intervención por parte del usuario.

Los algoritmos símplex primal y símplex dual resuelven problemas lineales y cuadráticos con restricciones lineales, y en *CPLEX* se incluye una versión creada específicamente para problemas de redes.

El algoritmo de punto interior con barrera resuelve problemas lineales, y cuadráticos con restricciones cuadráticas. Las soluciones por este método también se pueden adaptar a los algoritmos símplex para reinicios rápidos y análisis de sensibilidad. Los reinicios rápidos son una técnica relacionada con el análisis de sensibilidad y que sirve para ver cómo se comporta nuestra solución al modificar algunos parámetros del problema sin tener que volver a ejecutar la optimización completamente.

Para los problemas con variables enteras se utiliza el algoritmo de ramificación y corte (*Branch-and-Cut*), donde el usuario puede modificar las estrategias heurísticas y los planos de corte, definiendo si es más importante encontrar una solución óptima o determinar rápidamente una buena solución viable. El usuario también puede pedir que el optimizador devuelva múltiples soluciones para comparar los efectos de cada preferencia.

También se incluyen versiones en paralelo que aprovechan los sistemas con CPUs múltiples para resolver problemas de dificultad extrema. También se pueden utilizar optimizadores concurrentes con diferentes algoritmos para comparar sus velocidades.

CPLEX Optimizer es un componente del paquete de modelado *CPLEX Optimization Studio*, pero también tiene conectores en otros programas como *AIMSS*, *AMPL*, *GAMS*16, *MPL, R* y *Microsoft Solver Foundation*. Existe también un conector para *Microsoft Excel*. Para incluir este *solver* en una aplicación se proporcionan bibliotecas de componentes para C, C++, .NET, Java y Python. Estas bibliotecas incluyen rutinas para definir, resolver,

analizar y crear informes de los problemas de optimización y sus soluciones, y también permiten a los desarrolladores depurar sus aplicaciones Además, el optimizador interactivo es una utilidad de línea de comandos que permite al usuario leer y escribir ficheros de programación matemática y ajustar el funcionamiento del optimizador a los requerimientos de cada problema específico.

CPLEX Optimizer se puede utilizar con una orientación matricial para representar los modelos matemáticos, y también con un sistema de alto nivel que utiliza objetos de modelado para ayudar al desarrollador a aprovechar las aproximaciones orientadas a objetos.

IBM ILOG CPLEX Optimizer se vende con una licencia comercial para una gran variedad de plataformas. Es posible conseguir una versión de evaluación para 90 días, que viene limitada en tamaño.

3.2.2.2 Gurobi

Este optimizador sirve para programas lineales, cuadráticos y sus versiones de variable entera (MILP, MIQP). Su diseño se planteó para explotar los procesadores modernos de núcleo múltiple.

Para resolver LPs y QPs incluye implementaciones de alto rendimiento de los métodos símplex primal y dual, y un sistema de barrera paralelo.

Para los modelos MILP y MIQP incorpora métodos de plano cortante y heurísticas.
En todos los casos se aprovechan técnicas de preprocesado para simplificar los modelos y reducir el tiempo de resolución.

Este programa está escrito en C y se puede acceder a él desde varios lenguajes: además de un interfaz interactivo con Python y con C orientado a matrices, también se incluyen adaptaciones para C++, Java y .NET. Las adaptaciones requieren muy poca carga de memoria o de procesador, por lo que funcionan de forma muy eficiente.

Se pueden comprar licencias en diferentes escenarios: Para sistemas simples, para usuarios en red con licencias flotantes, y para incluir *Gurobi* como parte de otro producto.

También hay una versión de evaluación gratuita, que está limitada a 500 variables y 500 restricciones. Además hay licencias gratuitas para uso académico que no tienen restricciones de tamaño.

Otra variante que se ofrece es la “nube” de *Gurobi*, que se puede utilizar por horas por medio de la *nube elástica de computación de Amazon* (*Amazon Elastic Computing Cloud - EC2-*).

3.2.2.3 Xpress-Optimizer

La herramienta de optimización cuadrática de *Xpress* se llama *Xpress-Optimizer*, aunque con la evolución de versiones, en este momento también existe un paquete de modelado (*Xpress-Mosel*) y motores de resolución para abarcar más tipos de problemas (*Xpress-SLP*).

Este optimizador trata las matrices dispersas de forma eficiente comprimiendo datos para resolver problemas de grandes dimensiones que se encuentran fácilmente en la industria. Los algoritmos que incluye permiten resolver problemas lineales, cuadráticos y sus variantes con variables enteras.

Se puede utilizar desde la línea de comandos de distintos sistemas operativos, o como biblioteca enlazable desde C, C++, Java, Fortran, Visual Basic y .NET. Es compatible con los formatos estándar de ficheros de entrada *lp* y *mps*. También se puede utilizar desde un entorno visual de desarrollo que se llama *Xpress-IVE*.

Los productos de optimización *Xpress* pertenecen a la empresa *FICO*, y se distribuyen con licencia comercial.

Es posible obtener una versión de evaluación durante 30 días sin limitaciones en capacidad ni funcionalidades, incluyendo soporte técnico y documentación.

Esta opción está disponible para empresas. También hay una versión gratuita para estudiantes, limitada a 400 restricciones (filas), 800 variables (columnas), 5000 coeficientes de matrices (elementos) y 400 variables globales enteras y binarias.

3.2.2.4 OOQP

Es un paquete programado en C++ orientado a objetos, que se basa en un método de punto interior de tipo primal-dual. Sirve para resolver problemas cuadráticos convexos. Contiene código que se puede utilizar en aplicaciones externas a este paquete, para resolver una gran variedad de problemas cuadráticos, que pueden ser dispersos, o problemas de regresión de Huber, y QPs con restricciones de acotación.

También se puede utilizar para diseñar otros *solvers* para otras clases de QPs estructurados. Su diseño permite la sustitución de los módulos de álgebra lineal, por lo que se pueden probar diferentes paquetes estándar de álgebra lineal.

Existe un interfaz para *Matlab*.

Los derechos de copia están a nombre de E. Michael Gertz, que permite la libre copia, uso, redistribución y modificaciones para trabajos derivados, si estos cambios se

documentan adecuadamente. El autor solicita que se incluya una referencia a los derechos de copia como comentario en el código fuente que utilice este *software*.

3.2.3 Problemas Cónicos

Este apartado contiene la descripción de cuatro herramientas de resolución de problemas cónicos: *Mosek*, *SeDuMi*, *SDPT3* y *PENSDP*.

3.2.3.1 Mosek

De acuerdo con [15], *Mosek* es un programa de optimización diseñado para resolver problemas matemáticos a gran escala. Incluye rutinas de resolución para distintos tipos de problemas: lineales, cuadráticos, cónicos, convexos en general y problemas con mezcla de enteros.

Sus características más destacables son:

- El tamaño de los problemas está limitado únicamente por la memoria disponible.
- Un optimizador de punto interior con identificación de base.
- Optimizadores símplex primal y dual para programación lineal.
- Procesamiento previo para reducir el tamaño del problema antes de la optimización.
- Incluye un optimizador especial para problemas de redes y estructuras de flujos.
- Para problemas con mezcla de variables enteras se implementa un algoritmo de ramificar, acotar y cortar.

Incluye interfaces para lenguajes como C/C++, .NET, Java y Python, además de un complemento para *Matlab*. El optimizador de punto interior es capaz de explotar múltiples CPUs o núcleos. También dispone de un optimizador concurrente para resolver un mismo problema con diferentes optimizadores simultáneamente. La entrada de datos se puede hacer con ficheros en formatos estándar como *mps*, *lp* y *xml*. Incluye herramientas para el diagnóstico y reparación de la inviabilidad. También se puede hacer análisis de sensibilidad en problemas lineales. Existen conectores para utilizar *Mosek* desde muchos lenguajes de programación como C, C++, Java, *Matlab*, .NET y Python. También se puede utilizar desde herramientas de modelado como *AIMMS*, *COIN-OR*, *GAMS*, *Microsoft Solver Foundation* y *OptimJ*.

Mosek está sujeto a una licencia comercial en la que se especifica el número de copias que se pueden utilizar simultáneamente y las características permitidas. También se pueden utilizar licencias flotantes para varios usuarios en forma de testigo que se devuelve al servidor después de utilizarlo. Se pueden conseguir licencias de prueba en dos modalidades:

• Licencia de prueba para 30 días.
• Licencia libre para uso académico, que está limitada a 90 días y contiene todas las características del producto.

3.2.3.2 SeDuMi

Es un paquete que se utiliza para resolver problemas de optimización sobre conos simétricos. Esto incluye problemas lineales, cuadráticos, semidefinidos, cónicos, y cualquier combinación de los anteriores. La información sobre esta herramienta la he obtenido de [16], y según se indica en esta cita, está inspirado en una técnica conocida como empotrado auto-dual publicada en [26], que consiste en optimizar sobre conos homogéneos autoduales (cono cuyo dual es él mismo). Esto permite resolver algunos problemas en una sola fase, que lleva a la solución óptima o a una certificación de inviabilidad.

Sus características más destacables son las siguientes:

• Permite utilizar valores complejos.
• Genera soluciones duales en problemas no viables.
• Aprovecha las matrices dispersas para acelerar la ejecución.
• Gestiona las columnas densas de forma separada, y con ello favorece la dispersión.
• Puede importar problemas lineales en formato *mps* y problemas semidefinidos en formato *sdpa*.

SeDuMi se distribuye de forma gratuita bajo los términos de código libre GPL.

3.2.3.3 SDPT3

Esta herramienta está diseñada para resolver problemas cónicos en los que el cono de las restricciones puede ser el producto de conos semidefinidos, conos de segundo orden, ortantes no negativos, o espacios euclídeos. La función objetivo es la suma de funciones lineales y términos de barrera logarítmica asociados con los conos de las restricciones. La información de esta herramienta procede del manual [17].

El código está escrito en *Matlab*, aunque también contiene rutinas escritas en C. Su funcionamiento se basa en un algoritmo de seguimiento de camino primal-dual no viable. Esto es una variante del algoritmo primaldual en el que se utilizan dos versiones del paso de Newton. Una de ellas está relacionada con este mismo sistema de ecuaciones, y se conoce como dirección NT (*Nesterov-Todd*), y la otra intenta linealizar una versión simétrica del mismo (llamada dirección HKM (*Helmberg-Rendl-Vanderbei-Wolkowick (1996), Kojima-Shindoh-Hara (1997), Monteiro (1995)*)).

Sus características más destacables son:

• Permite utilizar variables libres.
• Se pueden resolver problemas de maximización de determinantes.
• Puede resolver SDPs con datos complejos.
• Se basa en un modelo de 3 parámetros con restricciones en conos homogéneos autoduales, de forma semejante a *SeDuMi*.

SDPT3 se distribuye bajo los términos de licencia pública general de GNU GPL 2.0, por lo que no es compatible para aplicaciones comerciales, aunque en este último caso las empresas se pueden poner en contacto con los autores para personalizar alternativas.

3.2.3.4 PENSDP

Es un programa para resolver problemas de tipo semidefinido, que tienen una función de objetivo lineal, y desigualdades matriciales lineales como restricciones.

Está orientado a problemas SDP a gran escala con matrices densas o dispersas.

Las principales características son:

• Ahorro de utilización de memoria por liberación del hessiano.
• Solución iterativa para el sistema de Newton.
• Criterios de parada basados en la medida de error *DIMACS* (21*Center for Discrete Mathematics and Theoretical Computer Science*), que da como resultado unos cálculos más fiables.
• Modo híbrido para la solución del sistema de Newton.
• Orientado a problemas a gran escala.
• Tratamiento eficiente de distintos patrones de dispersión en los datos del problema.
• Detección de no viabilidad.

Se puede utilizar como programa autónomo, con ficheros de entrada en formato de *SDPA* (*SemiDefinite Programming Algorithm*), y también se puede utilizar desde *Matlab*, como función desde las herramientas de modelado *TOMLAB* o *YALIMP*. Otra opción es utilizarlo como rutina que se puede llamar desde los lenguajes de programación C o Fortran. En este último caso los datos se comunican en forma de parámetros de dichas rutinas.

PENOPT se distribuye bajo una licencia comercial, que se puede comprar en diferentes formatos, dependiendo de su utilización para una empresa, un departamento, o un usuario único. También hay otras modalidades para uso académico, a nivel de universidad, departamento o usuario único, con precios más reducidos.

Por otro lado, también existe una licencia gratuita para desarrolladores, orientada a usuarios en entorno académico, y está limitada a tres meses, pudiendo renovarse cuando se necesite. Esta licencia se ofrece a investigadores que desarrollen y prueben *software*

de optimización y utilicen *PENOPT* para pruebas, o también si lo necesitan para desarrollar *software* en aplicaciones que requieran resolver problemas de optimización.

Además, existe una licencia de evaluación para usuarios comerciales, que expira en un mes y no se puede renovar. Esta licencia no es gratuita, aunque su precio se descuenta si finalmente se compra una licencia definitiva antes de un plazo determinado.

3.2.4 Sistemas de Modelado

Los sistemas de modelado proporcionan un lenguaje comprensible que permite al usuario expresar su problema centrando su esfuerzo en definirlo correctamente, dejando en manos del programa la conversión a otro formato compatible con las herramientas de resolución que forman parte del núcleo del sistema de modelado. Todas estas transformaciones del modelo descrito por el usuario producen otro problema o conjunto de problemas en un formato estándar y su mecanismo interno consiste básicamente en técnicas como pueden ser los cambios de variables, eliminar o añadir restricciones, utilizar variables de holgura, y otros recursos que podemos ampliar en [18]. Después de todas estas transformaciones, el sistema de modelado se encarga de elegir el *solver* más adecuado para el tipo de problema tratado. A partir de ahí, interpreta el resultado, indicando si es o no viable, y nos devuelve la solución óptima si es posible, adaptándola de nuevo al formato original.

En esta sección se describen las cuatro herramientas sobre las que he encontrado mayor número de referencias en la bibliografía que he consultado sobre optimización convexa: *CVX*, *AMPL*, *GAMS* y *YALIMP*. Si bien este criterio de selección no es demasiado riguroso, ya que quedan sin tratar herramientas de gran calado en el mercado como *AIMMS*, *TOMLAB*, *LINGO*, etc., lo que persigue es presentar una muestra significativa de las características de este tipo de programas, aunque limitada para no extender excesivamente el tamaño de este TFG.

3.2.4.1 CVX

CVX se utiliza integrada como complemento de *Matlab*. El trabajo con esta herramienta se basa en lo que llaman *programación convexa disciplinada*, que consiste en un conjunto de reglas que aseguran que expresamos nuestro problema de forma convexa.

El programa aporta una biblioteca de funciones convexas que sirven de base para que el usuario construya las funciones y conjuntos del problema aplicando operaciones que conserven la convexidad.

Las restricciones y funciones objetivo que se construyen con estas reglas se transforman automáticamente en forma canónica, y a partir de ahí se resuelve con los programas de resolución.

También se pueden resolver problemas geométricos, ya que *CVX* los transforma a un formato convexo, y una vez resuelto se devuelve el resultado en su formato original.

La información sobre *CVX* de este apartado procede del manual [19].

Los programas de resolución que utiliza son *SDPT3* y *SeDuMi*.

CVX se distribuye como *software* libre bajo los términos de GNU-GPL. Los derechos de copia están a nombre de Michael Grant y Stephen Boyd.

3.2.4.2 AMPL

AMPL es un lenguaje de programación para modelado algebraico en problemas de optimización lineales y no lineales, con variables discretas y continuas, y fue desarrollado en los laboratorios *Bell*.

Su utilización está orientada al desarrollo de prototipos y para trabajar en producción repetitiva. La información de esta herramienta procede de [20].

Soporta la definición de conjuntos y operaciones de conjuntos. También se caracteriza por un tipo de sintaxis muy general y natural para describir expresiones aritméticas, lógicas y condicionales, así como para sumatorios y otros operadores de iteración.

Incluye características de programación no lineal como la posibilidad de especificar valores iniciales para los problemas primal y dual, funciones definidas por el usuario, diferenciación automática, y eliminación automática de variables definidas.

Se puede utilizar para problemas de redes, para lo que incluye indicaciones específicas para declarar nodos y arcos, y una sintaxis especial para funciones lineales por tramos.

Se opera por medio de un entorno de comandos interactivo con opciones para procesos en segundo plano. Hay comandos que permiten visualizar cualquier componente o expresión del modelo, a través de la pantalla o escribiéndolo en un fichero que utiliza un formato automático o según las preferencias del usuario.

En *AMPL* los datos del problema están separados del modelo, por lo que éste contiene un sistema abstracto de variables, objetivos y restricciones que representa la forma general del problema a resolver, mientras que los datos se recopilan después de haber definido el modelo, creando un caso específico del problema. De esta forma, los modelos se mantienen de forma concisa incluso cuando los conjuntos y las tablas de datos aumentan.

Los modelos pueden incorporar muchos tipos de condiciones para validar los datos, y se incluyen interfaces con las herramientas de resolución.

El entorno de *AMPL* permite la conexión con una gran cantidad de herramientas de resolución, y en muchos casos este *software* de enlace no supone un coste adicional.

• Para programación lineal

- Variable continua: *BPMPD*, *Mosek*, y *LOQO*, entre otros.
- Variable entera: *MINTO*, *LAMPS*, *lp_solve*, etc.
- Redes: *CPLEX*, *OSL*

• Para programación no lineal

- Cuadrática: *CPLEX*, *Mosek*, *OSL*
- Convexa: *Mosek*, *SOPT*
- Continua: *CONOPT*, *KNITRO*, *LANCELOT*, etc.
- Entera: *MINLP*

AMPL está sujeto a una licencia comercial, que se puede comprar para máquinas individuales o en forma de licencias flotantes, y se ofrecen para una gran cantidad de plataformas. El modo flotante se puede utilizar cuando el número de usuarios es mayor que el de licencias, y se consigue con un servidor de autorizaciones en una red.

Se puede descargar una versión de evaluación en la que podemos utilizar hasta 300 variables y 300 restricciones y objetivos.

Nos podemos encontrar una versión académica cuyas características son:

- Los tamaños de los problemas no están restringidos.
- Se incluyen solucionadores comerciales de alta calidad y con todas las funciones.
- AMPL y los solucionadores se proporcionan en un único archivo comprimido para cada plataforma, que puede distribuir libremente a los estudiantes que desean una instalación en sus propias computadoras.
- Los archivos AMPL y solucionadores distribuidos se activan durante la duración del curso y dejan de funcionar una vez que finaliza el curso.
- Solo se debe enviar un breve formulario de solicitud para cada oferta de cursos.

Esta opción se usa tanto para enseñar la optimización como para incluir la optimización en la enseñanza de otras materias. Los temas de las clases de optimización que usaban AMPL para los cursos incluyen:

- Programación lineal.
- Optimización lineal y no lineal.

- Optimización combinatoria.
- Investigación de operaciones deterministas.
- Optimización de red.
- Control óptimo.

Una variedad de materias utilizó AMPL para introducir la optimización en la enseñanza. Los siguientes son una muestra representativa:

- Sistemas de fabricación.
- La investigación de operaciones.
- Gestión de la cadena de suministro.
- Operaciones de transporte público.
- Econometría.
- Finanzas.
- Ingeniería Ambiental.
- Ingeniería Química.
- Redes de comunicaciones.
- Planificación.

También se incluye la posibilidad de comprar a la misma empresa algunas de las herramientas de resolución como *Gurobi*, *CONOPT*, *KNITRO*, *SNOPT* y *MINOS*.

3.2.4.3 GAMS

Se trata de un sistema de modelado para programación matemática y optimización que consiste en un compilador de lenguaje y una serie de herramientas de resolución de alto rendimiento. La información de esta herramienta procede de [21].

Este sistema es especialmente útil cuando el problema a resolver es de grandes dimensiones, y que pueden requerir muchas revisiones para establecer un modelo adecuado. Se puede encontrar en versiones para ordenadores personales y para grandes servidores.

GAMS facilita la descripción del problema haciendo que el modelado se haga de forma compacta y natural, permitiendo que el usuario pueda cambiar fácil y rápidamente la

formulación, o la selección de uno u otro *solver*, o incluso se puede pasar de un modelo lineal a no lineal sin mayor problema.

El usuario se libera de tareas específicas del ordenador o servidor en que se instala, como por ejemplo calcular direcciones, asignaciones de almacenamiento, enlazado de subrutinas, o control de flujo. *GAMS* aumenta el tiempo disponible para conceptualizar y ejecutar el modelo, y analizar el resultado.

Este sistema requiere especificaciones concisas y exactas sobre las entidades y sus relaciones. El lenguaje *GAMS* es formalmente similar a otros lenguajes de programación comunes, por lo que es familiar para cualquiera con experiencia en programación.

Los datos se introducen en forma de lista y de tabla. Los modelos se describen con sentencias algebraicas concisas fáciles de leer tanto para los usuarios como para las máquinas. Se pueden describir conjuntos completos de restricciones relacionadas en una sola orden, para que después *GAMS* genere automáticamente cada ecuación de restricción, permitiendo que el usuario añada excepciones cuando la generalidad no sea aplicable. Las órdenes se pueden reutilizar sin tener que cambiar el álgebra en otros problemas semejantes.

El usuario puede programar fácilmente un modelo para que se resuelva condiferentes valores en algún elemento, y a continuación generar un listado de soluciones para cada caso.

Los modelos se desarrollan y documentan simultáneamente porque el usuario puede incluir textos explicativos como parte de la definición de cualquier símbolo o ecuación.

GAMS incluye una gran cantidad de herramientas de resolución, como *CPLEX*, *Gurobi*, *Mosek*, entre otras. Estos *solvers* permiten resolver problemas de tipo LP, MIP, MIQCP (*Mixed Integer Quadratically Constrained Program*), NLP (*Non-Linear Program*), MINLP, CNS (*Constrained Non-Linear System*), MCP (*Mixed Complementarity Problem)* MPEC (*Mathematical Program with Equilibrium Constraints*).

GAMS se compra con licencia comercial para empresas, y hay precios diferentes para uso académico. Se puede obtener para un solo usuario o para múltiples máquinas, y el precio depende de la arquitectura y de los *solvers* que se deseen. Las licencias de estos últimos se tienen que comprar con el paquete *GAMS*.

Existe también una versión de demostración limitada a 300 restricciones y variables, con 2000 elementos distintos de cero, y con 50 variables discretas.

En el caso de utilizar un *solver* de optimización global (no convexo) el número de variables y restricciones está limitado a 10.

3.2.4.4 YALIMP

YALIMP se utiliza como un complemento (*toolbox*) gratuito para *Matlab*, y sirve para modelar problemas de optimización convexos y no convexos.

El lenguaje es consistente con la sintaxis de *Matlab*, por lo que es muy fácil de aprender para usuarios familiarizados con este entorno. Implementa una gran cantidad de recursos de modelado, permitiendo que el usuario se concentre en el modelo a alto nivel, mientras que *YALIMP* se ocupa del modelado a bajo nivel para obtener modelos eficientes y numéricamente satisfactorios.

Soporta todos los tipos de problemas que hemos visto, lineales, cuadráticos, cónicos de orden 2, semidefinidos, geométricos, y otros más como por ejemplo los problemas cónicos con mezcla de variables enteras.

Una de las ideas centrales de *YALIMP* es concentrarse en el lenguaje y en los algoritmos de alto nivel, dejando los verdaderos cálculos de optimización para los *solvers* externos. Sin embargo, también implementa algoritmos internos de optimización global no convexa, programación con mezcla de enteros, programación multiparamétrica, programación de suma de cuadrados y optimización robusta. Estos algoritmos se basan típicamente en el lenguaje de bajo nivel disponible en *YALIMP*, y resuelven subproblemas utilizando herramientas externas de resolución.

• Para programación lineal:
- Con licencia libre: *CDD*, *GLPK*, *lp_solve*, *QSOPT*
- Con licencia comercial o libre en versiones académicas: *BINTPROG*, *CPLEX*, *Gurobi*, linprog (de *Matlab Optimization Toolbox*), *Mosek*

• Para programación cuadrática:
- Con licencia libre: *BPMPD*, *CLP*, *OOQP*, *QPC*
- Con licencia comercial o libre en versiones académicas: *CPLEX*, *Mosek*, *NAG*, quadprog (de *Matlab Optimization Toolbox*), *Xpress*

• Para programación cónica de segundo orden:
- Con licencia libre: *SDPT3*, *SeDuMi*
- Con licencia comercial o libre en versiones académicas: *CPLEX*, *Mosek*

• Para programación semidefinida:
- Con licencia libre: *CSDP*, *DSDP*, *SDPA*, *SDPLR*, *SDPT3*, *SeDuMi*
- Con licencia comercial o libre en versiones académicas: *LMILAB*, *PENBMI*, *PENSDP*

• Para programación no lineal en general y otros *solvers*: fmincon (de *Matlab Optimization Toolbox*), *GPPOSY*, *IPOPT*, *KYPD*, *LMIRANK*, *MPT*, *PENNON*, *SNOPT*, *VSDP*

La versión actual de *YALIMP* es libre de cargo y se distribuye de forma abierta.

Esta disponibilidad implica que no cubre ningún servicio de garantía.

3.2.4.5 Entorno R

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico [22] y [23].

R nació como una reimplementación de software libre del lenguaje S, adicionado con soporte para alcance estático. Se trata de uno de los lenguajes de programación más utilizados en investigación científica, siendo además muy popular en los campos de aprendizaje automático (machine learning), minería de datos, investigación biomédica, bioinformática y matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo y representación gráfica.

R es parte del sistema GNU y se distribuye bajo la licencia GNU GPL. Está disponible para los sistemas operativos Windows, Macintosh, Unix y GNU/Linux.

3.3 Algunos ejemplos de modelado

3.3.1 MPS

MPS (Sistema de Programación Matemática) es un formato de archivo para la presentación y archivo de programación lineal (LP) y de programación entera mixta.

El formato fue nombrado después de uno de los primeros IBM producto LP y se ha convertido en un estándar de facto ASCII medio entre la mayoría de los solucionadores comerciales del LP. Esencialmente todos los solucionadores comerciales del LP aceptan este formato, y también es aceptada por el código abierto COIN-O sistema. Sin embargo, con la aceptación de lenguajes de modelado algebraicas el uso de MPS ha disminuido. Por ejemplo, de acuerdo con las NEOS estadísticas del servidor en enero de 2011 a menos de 1% de las presentaciones eran en forma de MPS en comparación con 59,4% de AMPL y 29,7% de GAMS presentaciones.

Aquí hay un pequeño modelo de ejemplo escritos en formato de MPS:

```
NAME          TESTPROB
ROWS
 N  COST
```

```
 L   LIM1
 G   LIM2
 E   MYEQN
COLUMNS
    XONE      COST                 1   LIM1                 1
    XONE      LIM2                 1
    YTWO      COST                 4   LIM1                 1
    YTWO      MYEQN               -1
    ZTHREE    COST                 9   LIM2                 1
    ZTHREE    MYEQN                1
RHS
    RHS1      LIM1                 5   LIM2                10
    RHS1      MYEQN                7
BOUNDS
 UP BND1      XONE                 4
 LO BND1      YTWO                -1
 UP BND1      YTWO                 1
ENDATA
```

A modo de comparación, aquí es el mismo modelo escrito en un formato orientado a ecuación:

```
Optimize
 COST:    XONE + 4*YTWO + 9*ZTHREE
Subject To
 LIM1:    XONE + YTWO           <= 5
 LIM2:    XONE         + ZTHREE >= 10
 MYEQN:         - YTWO + ZTHREE  = 7
Bounds
       XONE <= 4
 -1 <= YTWO <= 1
End
```

El BOUNDS inferior de XONE es o bien 0 ó $(-\infty)$, dependiendo de la aplicación, ya que no se especifica. Curiosamente, no hay nada en formato MPS especifica la dirección de la optimización, y no hay una dirección estándar "por defecto"; algunos solucionadores LP maximizarán, si no se indica lo contrario, otros minimizarán, y otros darán más prioridad a la seguridad y no tienen ningún defecto y requieren una selección, en alguna parte, de un programa de control o mediante un parámetro. Si el modelo está formulado para la

minimización y el solucionador requiere maximización (o viceversa), es fácil de convertir entre los dos mediante la negación de todos los coeficientes de la función objetivo. El valor óptimo de la función objetivo será entonces el negativo del valor óptimo original, pero los valores de las variables mismas será correcta. Algunos programas de apoyo a la especificación de la minimización / maximización dentro del archivo de MPS.

```
OBJSENSE
 MAX
```

El registro de nombre puede tener cualquier valor, a partir de la columna 15.

La sección Filas define los nombres de todas las restricciones; entradas en la columna 2 o 3 son E por la igualdad (=) filas, L por menos que (<filas =), G para mayores-que (> =) filas, y N para las filas no limitante. El orden de las filas mencionadas en esta sección no es importante, excepto para el mercado de filas no coactivo N, el primero de los cuales sería interpretado como la función objetivo.

La sección de columnas contiene las entradas de la A-matriz. Todas las entradas para una columna dada deben ser colocados consecutivamente, aunque dentro de una columna el orden de las entradas (filas) es irrelevante. Las filas que no se mencionan para una columna se implicaba tener un coeficiente de cero.

La sección RHS permite que uno o más vectores de lado de mano derecha para ser definidos; rara vez hay más de uno. En el ejemplo anterior, el nombre del vector RHS es RHS1, y tiene valores no cero en todos los 3 de las filas de restricción del problema. Filas no mencionados en un vector de RHS se supone que tiene una mano del lado derecho de cero.

La sección LÍMITES opcional especifica límites inferior y superior en variables individuales, si no se les da por filas en la matriz. Todos los límites que tienen un nombre que se da en la columna 5 se toman juntos como un conjunto. Variables no mencionados en un conjunto límites dados son llevados a ser no negativo (límite inferior de cero, no límite superior). A unida de tipo UP significa un límite superior se aplica a la variable. A unida de tipo LO significa un límite inferior se aplica. Un tipo consolidado de FX ("fijo") significa que la variable tiene límites superior e inferior iguales a un único valor. Un tipo consolidado de FR ("libre") significa que la variable no tiene ni menor ni límites superiores y así puede adoptar valores negativos. Una variante de que es MI gratis negativo, dando una cota superior de 0 pero no hay límite inferior. Tipo Bound PL es para un libre positivo de cero a más infinito, pero como este es el valor predeterminado normal, que rara vez se utiliza. También hay tipos consolidados para uso en MIP modelos - BV para binario, siendo 0 o 1. interfaz de usuario para número entero superior y LI para número entero inferior. SC significa semi-continuo, e indica que la variable puede ser cero, pero si no debe ser igual a por lo menos el valor dado.

Otra sección opcional llamado RANGOS especifica dobles desigualdades, de una manera poco intuitivo que no se describe aquí. Maneras de marcar las variables enteras son

también más allá del alcance de este artículo (marcador de palabras clave y posiblemente SOS están involucrados). La última carta debe ser ENDATA (nótese la ortografía impar).

Unos pocos casos especiales de la norma MPS no se manejan constantemente por las implementaciones. En la sección de los límites, si se da una variable de un arrastre de fuerza límite superior pero no cota inferior, su límite inferior puede por defecto a cero o a menos infinito (también, si se da el límite superior como cero, el límite inferior puede ser cero o negativo infinito). Si no superior ha unido especificado una variable de número entero, su límite superior puede por defecto a uno más que a más infinito.

MPS tiene muchas limitaciones. No se especifica la dirección de optimización que se maneja de manera diferente por resolver. Los campos numéricos tienen 12 caracteres por lo tanto, limitan el ancho de precisión. La representación no es ni fácil ni para la interpretación humana compacto (aunque la información para la columna reservas / fila, que a menudo es beneficioso para LP solucionador reproducibilidad comportamiento). Una de las alternativas a la MPS que no tienen sus limitaciones y es apoyado por la mayoría de los solucionadores es el formato de archivo nl .

Muchos productos LP incluyen extensiones al formato de MPS. El formato libre MPS permite nombres largos y datos más precisos, permitiendo campos para superar las columnas definidas por el estándar original, y aplicar espacios en blanco como separadores en vez de posiciones de las columnas fijas (tenga en cuenta que esto hace que algunos archivos de MPS que incluyen espacios en blanco como parte de nombres que ya no son válidos). Algunas extensiones incluyen la adición de nuevos tipos de datos en el fichero de MPS (por ejemplo, para incluir secciones sentido objetivo, los requisitos de integralidad, los datos de segundo grado o avanzados construcciones de modelado MIP). También hay un formato de archivo comprimido CBPD. SMPS es una extensión especializado, diseñado para representar de programación estocásticos casos de problemas, en uso, especialmente en entornos de investigación.

A pesar de que algunas extensiones no están estandarizados, el formato sigue siendo de uso general.

3.3.2 AMPL

La gran potencia del lenguaje AMPL está en separar el modelo en sí por un lado y por otro los datos particulares del problema concreto.

Una compañia fabrica tres productos, P1, P2 y P3, que precisan para su elaboración dos materias primas, M1 y M2. Las disponibilidades semanales de estas materias son 25 y 30 unidades, respectivamente. El benefício neto que proporciona cada unidad de producto, así como las unidades de materia prima que necesita para su elaboración, vienen dados en la siguiente tabla:

	P1	P2	P3
M1	1	2	2
M2	2	1	3
Beneficio (u.m.)	2	6	3

Este problema de forma general sería de la siguiente forma:

$$\max z = \sum_{j=1}^{n} c_j x_j$$
$$s.a. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \forall i = 1, \dots, m$$
$$x_j \geq 0, j = 1, \dots, n$$

Esto en AMPL se escribiria de la forma siguiente en dos ficheros. En el fichero del modelo aparecería:

```
// MODELO: EJEMPLO1.MOD
// FABRICACION DE n PRODUCTOS CON m MATERIAS PRIMAS
// PARAMETROS DEL MODELO
param n >=0, integer;
param m>=0, integer;
//CONJUNTOS DE INDICES
set PRODUCTOS := 1 . . n;
set MPRIMAS := 1 . . m;
// VARIABLES DE DECISION Y RESTRICCIONES NO NEGATIVIDAD
var x {j  in  PRODUCTOS} >= 0;
// MAS PARAMETROS DEL MODELO
param c {i in PRODUCTOS};
param b {j in MPRIMAS};
param a {(i, j) in {MPRIMAS,PRODUCTOS}};
//FUNCION OBJETIVOS DEL MODELO
maximize z : sum {j in PRODUCTOS} c[j]*x[j];
// RESTRICCIONES DEL MODELO
subject  to  restricción {i  in  MPRIMAS} :
        sum {j  in  PRODUCTOS}  a[i,j]*x[j]  <=  b[i] ;
```

Y el fichero de datos para nuestro ejemplo sería:

```
// DATOS: EJEMPLO1.DAT
Param n := 3;
Param n := 2;
Param c :=
        1       2
        2       6
        3       3;
Param a := 1  2  3 :=
        1               1  2  2
        2               2  1  3;
Param b :=
        1       26
        2       30;
```

3.3.3 GAMS

Veamos a continuación un caso típico de un problema de optimización lineal clásico y cómo este problema se codifica en el lenguaje GAMS.

Sean i fábricas de envadado y j mercados de consumo. Cada fábrica tiene una capacidad máxima de producción de a_i cajas y cada mercado demanda una cantidad b_j de cajas (se supone que la capacidad de producción total de las fábricas es superior a la demanda total para que el problema sea factible). El coste de transporte entre cada fábrica i y cada mercado j por cada caja es c_{ij}. Se desea satisfacer la demanda de cada mercado al mínimo coste. Las variables de decisión del problema serán las cajas transportadas entre cada fábrica i y cada mercado j, x_{ij}.

Las ecuaciones que deben satisfacerse son:

Límite de capacidad máxima de producción de cada fabrica:

$\sum_j x_{ij} \leq a_i$ para cada fábrica i

Satisfacción de la demanda de cada mercado:

$\sum_i x_{ij} \geq b_j$ para cada mercado j

La función objetivo será la minimización de los costes totales de transporte:

$$\sum_i \sum_j c_{ij} x_{ij}$$

Ésta es la forma algebraica de representación de este problema de optimización. La codificación en lenguaje GAMS aparece a continuación:

```
$TITLE MODELO DE TRANSPORTE

SETS
        I  fábrica de envasado      / VIGO, ALGECIRAS /
        J  mercados de consume      / MADRID, BARCELONA, VALENCIA /

PARAMETROS
        A(i)  capacidad de producción de la fábrica i  [cajas]
            / VIGO              350
              ALGECIRAS         700 /

        B(j)     demanda del mercado j [cajas]
            /    MADRID 400
                 BARCELONA      450
                 VALENCIA       150 /

TABLE  C(i,j)  coste unitario transporte entre  i y j  [miles de euros por caja]
                 MADRID BARCELONA              VALENCIA
VIGO               0.06              0.12          0.09
ALGECIRAS          0.05              0.15          0.11

VARIABLES
        X(i,j)   cajas transportadas entre fábrica i y mercado j  [cajas]
        CT                coste de  transporte  [miles de euros]

POSITIVE VARIABLE  X

EQUATIONS
        COSTE             coste total de transporte              [miles de euros]
        CAPACIDAD(i)      capacidad máxima de caja fábrica  i  [cajas]
        DEMANDA(j)        satisfacción demanda de cada mercado j  [cajas]

COSTE  ..                 CT = E = SUM[(i,j), C(i,j)*X(i,j)] ;

CAPACIDAD(I) ..           SUM[j, X(i,j)] = L = A(i) ;

DEMANDA(j) ..  SUM[i, X(i,j)] = G = B(j) ;

MODEL  TRANSPORTE  /  COSTE, CAPACIDAD, DEMANDA  /

SOLVE  TRANSPORTE USING LP MINIMIZING CT
```

El resultado de la ejecución del modelo de transporte se presenta a continuación:

```
C o m p i l a t i o n
COMPILATION TIME =        0.000 SECONDS       0.7 Mb WIN-19-115

Equation Listing        SOLVE TRANSPORTE USING LP FROM LINE 39
---- COSTE      =E=     coste total de transporte  [miles de euros]
COSTE . .  - 0.06*X (VIGO, MADRID) - 0.12*X(VIGO, BARCELONA) - 0.09*X(VIGO, VALENCIA)
         - 0.5 * X(ALGECIRAS, MADRID)  - 0.15* X(ALGECIRAS, BARCELONA)
         - 0.11 * X(ALGECIRAS, VALENCIA)  + CT =E=   0 ;   (LHS = 0 )
---- CAPACIDAD  =L=  capacidad máxima de cada fábrica i    [cajas]
CAPACIDAD (VIGO) . .   X(VIGO, MADRID) + X(VIGO, BARCELONA) + X(VIGO, VALENCIA) =L= 350 ;
         (LHS = 0)
CAPACIDAD (ALGECIRAS) . .  X(ALGECIRAS, MADRID)  +  X(ALGECIRAS, BARCELONA)
         +  X(ALGECIRAS, VALENCIA)  =L=  700  ;  (LHS = 0)
---- DEMANDA  =G=  satisfacción demanda de cada mercado j  [cajas]
DEMANDA (MADRID) . .  X(VIGO, MADRID)  +  X(ALGECIRAS, MADRID)  =G=  400  ;
         (LHS = 0,  INFES = 400 ***)
DEMANDA (BARCELONA) . .  X(VIGO, BARCELONA)  +  X(ALGECIRAS, BARCELONA)  =G=  450  ;
         (LHS = 0,  INFES = 450 ***)
DEMANDA (VALENCIA) . .  X(VIGO, VALENCIA)  +  X(ALGECIRAS, VALENCIA)  =G=  150  ;
         (LHS = 0,  INFES =150  ***)
Column Listing          SOLVE TRANSPORTE USING LP FROM LINE 39
---- X  cajas transportadas entre fábrica i y mercado j  [cajas]
X(VIGO, MADRID)
                        (.LO,  .L,  .UP  =  0,  0,  +INF)
                -0.06   COSTE
                1       CAPACIDAD (VIGO)
                1       DEMANDA (MADRID)
X(VIGO, BARCELONA)
                        (.LO,  .L,  .UP  =  0,  0,  +INF)
                -0.12   COSTE
                1       CAPACIDAD (VIGO)
                1       DEMANDA (VIGO)
X(VIGO, VALENCIA)
                        (.LO,  .L,  .UP  =  0,  0,  +INF)
                -0.09   COSTE
                1       CCAPACIDAD (VIGO)
                1       DEMANDA (VALENCIA)
REMAINING 3 ENTRIES SKIPPED
---- CT  coste de transporte   [miles de euros]
CT
                        (.LO,  .L,  .UP  =  -INF,  0,  +INF)
                1       COSTE
Model Statistics          SOLVE TRANSPORTE USING LP FROM LINE 39
MODEL STATISTICS
BLOCKS OF EQUATIONS     3       SINGLE EQUATIONS        6
BLOCKS OF VARIABLES     2       SINGLE VARIABLES        7
NON ZERO ELEMENTS       19
GENERATION TIME         =       0.140 SECONDS     1.4 Mb      WIN-19-115
EXECUTION TIME          =       0.140 SECONDS     1.4 Mb      WIN-19-115

                        S O L V E       S U M M A R Y
        MODEL           TRANSPORTE          OBJECTIVE    CT
        TYPE            LP                  DIRECTION    MINIMIZE
        SOLVER          CPLEX               FROM LINE    39
```

```
**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE

 RESOURCE USAGE, LIMIT      0.401         1000.000
 ITERATION COUNT, LIMIT     5             10000

GAMS/Cplex    Mar 5, 2020 WIN.CP.CP. 19.3 016.014.038.WAT For Cplex 6.6
Cplex 6.6.1, GAMS Link 16, Using a GAMS/Cplex demo license installed at runtime.

Optimal solution found.

Objective :           93.500000
                           LOWER       LEVEL       UPPER       MARGINAL
---- EQU COSTE                                                    1.000
       COSTE       coste total de transporte   [miles de euros]
---- EQU CAPACIDAD     capacidad máxima de cada fábrica i    [cajas]
                           LOWER       LEVEL       UPPER       MARGINAL

VIGO                       -INF       350.000     350.000      -0.030
ALGECIRAS                  -INF       650.000     700.000

---- EQU DEMANDA       satisfacción demanda de cada mercado j  [cajas]
                           LOWER       LEVEL       UPPER       MARGINAL

MADRID                  400.000     400.000        +INF         0.050
BARCELONA               450.000     450.000        +INF         0.150
VALENCIA                150.000     150.000        +INF         0.110
---- VAR X             cajas transportadas entre fábrica i y mercado j  [cajas]
                           LOWER       LEVEL       UPPER       MARGINAL

VIGO          .MADRID         .           .          +INF         0.040
VIGO          .BARCELONA      .       350.000        +INF           .
VIGO          .VALENCIA       .           .          +INF         0.010
ALGECIRAS     .MADRID         .       400.000        +INF           .
ALGECIRAS     .BARCELONA      .       100.000        +INF           .
ALGECIRAS     .VALENCIA       .       150.000        +INF           .
                           LOWER       LEVEL       UPPER       MARGINAL
---- VAR CT                -INF       03.500         +INF

       CT     coste de transporte   [miles de euros]
**** REPORT SUMMARY :        0     NONOPT
                             0  INFEASIBLE
                             0  UNBOUNDED
EXECUTION TIME       =            0.030 SECONDS      0.7 Mb       WIN-19-115

**** FILE SUMMARY

INPUT       D:\TR.GMS
OUTPUT      D:\TR.LST
```

Capítulo 4
Ejemplos de uso de software de optimización

4.1 Problemas de optimización lineal

Veamos cómo se pueden resolver problemas de optimización lineal con R a través de algunos ejemplos sencillos. La mayor parte de las funciones necesarias pueden encontrarse en el paquete lp_solve [25] que a su vez esta basado en el proyecto de código abierto lp_Solve. Otro paquete alternativo que también se puede usar es linprog.

4.1.1 El paquete lpSolve

Un problema en forma canónica

Como ejemplo vamos a usar el siguiente problema:

> Un pastelero dispone de 150 kg de harina, 22 kg de azúcar y 27.5 kg de mantequilla para elaborar dos tipos de pasteles (*A* y *B*). Cada caja de pasteles de tipo *A* requiere 3 kg de harina, 1 kg de azúcar y 1 kg de mantequilla y su venta le reporta un beneficio de 20 euros. Cada caja de pasteles de tipo *B* requiere 6 kg de harina, 0.5 kg de azúcar y 1 kg de mantequilla y su venta le reporta un beneficio de 30 euros.¿Cuántas cajas de cada tipo debe elaborar el pastelero de manera que se maximicen sus ganancias? (Se supone en principio que también puede elaborar cajas incompletas, es decir, que no se trata de un problema de programación entera.)

La forma canónica de un problema de optimización lineal es $\max c^T x \quad s.a.\ Ax \leq b, x \geq 0$. La función canónica del problema que vamos a resolver como ejemplo es:

$$\begin{aligned} &\max 20x_1 + 30x_2 \\ &s.a. \quad 3x_1 + 6x_2 \leq 150 \\ &\qquad x_1 + 0.5x_2 \leq 22 \\ &\qquad x_1 + x_2 \leq 27.5 \\ &\qquad x_1 \geq 0, x_2 \geq 0 \end{aligned}$$

Para resolverlo basta llamar a la función lp y pasarle como argumentos de forma ordenada los parámetros que definen el problema. El orden de los parámetros es el siguiente:

1. "min" o "max" en función de si el problema es de minimización o maximización respectivamente.
2. El vector de coeficientes de la función objetivo. En el ejemplo $c = (20, 30)$.
3. La matriz de coeficientes de las variables en las ecuaciones quew definen las restricciones. En el ejemplo,

$$A = \begin{pmatrix} 3 & 6 \\ 1 & 0.5 \\ 1 & 1 \end{pmatrix}$$

4. Un vector que determine la dirección de las restricciones $\leq, \geq o =$. En el ejemplo las desigualdades son del tipo $\leq$.
5. El vector con los términos independientes de las restricciones. En el ejemplo $b = (150, 22, 22.5)$.

Por defecto se supone que las variables de decisión no pueden ser negativas.

El siguiente código define todos los parámetros del problema y se los pasa como argumentos a la función lp en el orden que hemos indicado:

```
Library(lp_solve)

#Parametros del problema

coef <- c(20, 30)
A <- matriz(c(3, 1, 1, 6, 0.5, 1), ncol=2)
b <- c(150, 22, 27.5)
dir <- rep('<=', 3)

# Solucion

solucion <- lp('max', coef, A, dir, b)
```

el objeto solución que hemos generado al ejecutar la última línea es una lista con diversos elementos entre los cuales los más importantes son objval, el valor objetivo óptimo, y solution, las coordenadas de la solución factible óptimo del problema:

```
solución$objval

## [1] 775

Solución$solution

## [1] 5.0  22.5
```

Esto significa que la producción óptima del pastelero es de 5 cajas de A y 22.5 cajas de B, con lo que tendrá un beneficio máximo de 775 euros.

Programación entera

Supongamos ahora que el pastelero solo puede fabricar un número entero de cajas de cada tipo, es decir, que las variables x_1 y x_2 solo pueden tomar valores enteros. La forma de indicar esta restricción adicional es añadiendo en la función lp el argument **all.int=TRUE** (también existe la posibilidad de restringir únicamente un subconjunto de las variables):

```
solución <- lp('max', coef, A, dir, b, all.int=TRUE)
solución$objval

## [1] 770

solución$solution

## [1] 4 23
```

En este caso, la producción óptima del pastelero es de 4 cajas de A y 23 cajas de B con un beneficio de 770 euros.

Problema de transporte

El paquete lp_solve incluye también la función **lp.transport** para resolver el problema de transporte. Como ejemplo, vamos a resolver el problema de tramsporte definido por los siguientes datos:

3	4	6	8	9	30
2	2	4	5	5	80
2	2	2	3	3	10
3	3	2	4	2	60
10	50	20	80	20	

Las filas corresponden a los lugares de origen y las columnas a los lugares de destino. Los valores de la matriz de costes de transporte desde cada origen a cada destino. El último elemento de cada fila es la oferta de producto en cada origen. El último elemento de cada columna es la demanda total en cada destino.

A continuación definimos todos los parámetros del problema y los pasamos como argument de **lp.transport** en el orden adecuado:

```
cost <- matrix(c(3, 2, 2, 3, 4, 2, 2, 3, 6, 4, 2, 2, 8, 5, 3, 4, 9, 5, 3, 2), ncol=5)
direction <- 'min'
row.signs <- rep('=', 4)
row.rhs <- c(30, 80, 10, 60)
col.signs <- rep('=', 5)
col.rhs <- c(10, 50, 20, 80, 20)
lp.transport(cost, direction, row.signs, row.rhs, col.signs, col.rhs)

## Success: the objective function is 610

lp.transport(cost, direction, row.sings, row.rhs, col.signs, col.rhs)$solution

##      [,1] [,2] [,3] [,4] [,5]
## [1,]   10   20    0    0    0
## [2,]    0   30    0   50    0
## [3,]    0    0    0   10    0
## [4,]    0    0   20   20   20
```

La solución óptima del problema es transporte 10 desde el origen 1 hasta el destino 1, 20 desde el origen 1 hasta el destino 2, 30 desde el origen 2 hasta el destino 2, etc. esta solución óptima conllevan un coste de 610.

Problema de asignación

Para resolver el problema de asignación el paquete lp_solve incluye la función **lp.assign**. como ejemplo, vamos a resolver el problema de asignación definido por la matriz:

$$\begin{pmatrix} 15 & 10 & 9 \\ 9 & 15 & 10 \\ 10 & 12 & 8 \end{pmatrix}$$

Las filas representan a los trabajadores y las columnas a las tareas. Cada elemento de la matriz corresponde el coste de asignar la tarea I al trabajador j.

A continuación definimos la matriz de costes, que es el único argument de **lp.assign**:

```
cost <- matriz(c(15, 9, 10, 10, 15, 12, 9, 10, 8), nrow=3)
lp.assign(cost)

## Success: the objective function is 27

lp.assign(cost)$solution

##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    1    0    0
## [3,]    0    0    1
```

La solución óptima es asignar la tarea 1 al trabajador 2, la tarea 2 al trabajador 1 y la tarea 3 al trabajador 3, todo con un coste de 27.

4.1.2 El paquete linprog

Una alternativa a la función lp de lp_solve es la función solveLP del paquete **linprog**. Su funcionamiento es similar, aunque el rango de problemas que puede resolver es menor. Por ejemplo, no trata adecuadamente problemas en forma estándar con restricciones de igualdad.

El orden por defecto en el que hay que pasar los argumentos es diferente al de lp:

1. Coeficientes de la función objetivo c.
2. Términos independientes de las restricciones b.
3. Matriz de coeficientes de las variables en las restricciones A.
4. Determinar si queremos maximizar o minimizar (maximum=TRUE o maximum=FALSE).
5. Vector que determina la dirección de las restricciones: $\leq, \geq o =$.

```
library(linprog)
# Parametros del problema
coef <- c(20, 30)
A <- matrix(c(3, 1, 1, 6, 0.5, 1), ncol=2)
b <- c(150, 22, 27.5)
dir <- rep('<=', 3)
```

```
# Solucion
solución <- solveLP(coef, b, A, maximum=TRUE, dir)
summary(solución)

##
##
## Result of Linear Programming / Linear Optimization
##
## Solution
##       opt
## 1     5.0
## 2    22.5
```

La función solveLP tiene la ventaja sobre lp de dar más información sobre los pasos intermedios del algoritmos simplex utilizado para resolver el problema. Para el máximo nivel de detalle se usa la opción **verbose=4**.

```
solveLP(coef, b, A, maximum=TRUE, dir, verbose=4)

##       [1]    @initial Tableau"
##                 1         2  S  1  S  2  S  3      P0
## 1               3       6.0     1     0     0   150.0
## 2               1       0.5     0     1     0    22.0
## 3               1       1.0     0     0     1    27.5
## Z-C           -20    -30.0     0     0     0     0.0
##
## Pivot Column: 2 ( 2 )
## Pivot Row: 1 ( 1 )
##
##                 1   2       S  1   S   2   S  3         P0
## 2            0.50   1   0.16667       0       0       25.0
## 2            0.75   0  -0.08333       1       0        9.5
## 3            0.50   0  -0.16667       0       1        2.5
## Z-C         -5.00   0   5.00000       0       0     750.0
##
## Pivot Column: 1 ( 1 )
## Pivot Row: 3 ( 3 )
##
##      1   2       S  1   S   2   S  3           P0
```

```
## 2    0  1   0.3333       0   -1.0        22.50
## 2    0  0   0.1667       1   -1.5         5.75
## 1    1  0  -0.3333       0    2.0         5.00
## Z-C 0  0    3.3333       0   10.0       775.00

##
##
## Result of Linear Programming / Linear Optimization
##
## Objective function (Maximum): 775
##
## Iterations in phase 1: 0
## Iterations in phase 2: 2
## Solution
##       opt
## 1     5.0
## 2    22.5
##
## Basic Variables
##       opt
## 1      5.0
## 2     22.5
## S  2   5.75
##
## Constraints
##      actual  dir     bvec    free       dual    dual.reg
## 1    150.0   <=     150.0    0.00    3.33333     34.50
## 2    16.25   <=      22.0    5.75    0.00000      5.75
## 3    27.50   <=      27.5    0.00   10.00000      2.50
##
## All Variables (including slack variables)
##
##          opt  cvec      min.c      max.c         marg       marg.reg
## 1       5.00    20         15   30.00000           NA             NA
## 2      22.50    30         20   40.00000           NA             NA
## S  1    0.00     0       -Inf    3.33333     -3.33333           34.5
## S  2    5.75     0        -20    6.66667      0.00000             NA
## S  3    0.00     0       -Inf   10.00000    -10.00000            2.5
```

4.2 Problemas de optimización cuadrática

Tras una previa investigación teórica concluimos que una de las principales diferencias de la programación no lineal con respecto a la programación lineal, es que el punto óptimo no tiene por qué encontrarse en los puntos extremos de nuestra región factible, es más, puede encontrarse en el interior de nuestra región mientras que en el caso de la programación lineal nunca se encontrará en el interior.

Resolución con R:

Para su aplicación en R usaremos la función ***constrOptim*** para minimizar una función sujeta a restricciones de desigualdad.

Función:

```
constrOptim(theta, f, grad, ui, ci)
```

*Argumentos de la función **constrOptim:***

Theta: valor de partida numérico (vector) (de longitud p): debe estar en la región factible.

f: función para minimizar.
grad: gradiente de f (una función también), o NULL.
ui: matriz de restricción (k x p).
ci: vector de restricción de longitud k.

*Salidas de la función **constrOptim:***

$par: valor de las variables de decisión que optimiza la función objetivo.
$value: valor que alcanza la función objetivo con el valor obtenido para las variables de decisión.

Aplicación práctica

Empezamos aplicándolo con el uso de R® mediante la resolución de un ejemplo, de forma que se pueda ver en la práctica.

$$Minimizar\ Coste = 20.000 - 440x_1 - 300x_2 + 20x_1^2 + 12x_2^2 + x_1x_2$$
$$s.a. \qquad x_1 + x_2 = 100$$

Para llevar a cabo la resolución de este problema en R®, comenzamos introduciendo los datos que nos ofrece el problema, es decir, vamos a definir la función que queremos minimizar, pero en este caso utilizando el comando ***function*** para definir el objeto, donde indicaremos el nombre de las variables, x_1 y x_2, y a continuación entre corchetes debemos de introducir la función objetivo, $20.000 - 440x_1 - 300x_2 + 20x_1^2 + 12x_2^2 + x_1x_2$, teniendo en cuenta la adecuada escritura en R.

```
PNL<- function(x1,x2){(20000-440*x1-300*x2+20*x1^2+12*x2^2+x1*x2)}
```

Para la introducción de las restricciones se usará una matriz y para las disponibilidades se define el objeto mediante un vector.

```
matriz <- Matrix(c(1,1), nrow=1, byrow=TRUE)
vectorb <- c(100)
```

Una vez que introducimos los objetos podemos pasar a su resolución, en este caso, utilizaremos el paquete llamado ***constrOptim.***
En este paquete se pueden definir varios argumentos, nos centramos en aquellos de nuestro interés para la resolución del ejercicio.

```
constroptim (c(100,100),
        function (x) {PNL (x[1], x[2])},
        grad=NULL, ui=matriz, ci-vectorb}
```

Directamente al ejecutar el comando mediante “run” o usando “ctrl+r” nos mostrará todos los resultados, si queremos en concreto el valor de nuestras variables podremos fijarnos en **$par** donde observamos que x_1= 26’5537 y x_2=73’4463. Para el valor que tomará la función objetivo se mostrará en **$value** que en este caso será de 1.029.060 unidades.

```
$par                          $value

[1]  26.55377    73.44623     [1]  1029060
```

4.3 Problemas de optimización cónica

$$\begin{array}{ll} maximizar & x+y \\ sujeto\ a & \sqrt{x^2+y^2} \leq \sqrt{2} \\ & x,y \geq 0 \end{array}$$

```
socpl <- OP (objective = L_objective (c(1, 1), names = c("x", "y")),
          constraints = C_condtraint (rbind (c(0, 0), c(-1, 0), c(0, -1)),
                                      cones = K_soc ( 3 ),
                                      rhs = c (sqrt ( 2 ), 0, 0)),
          Maximum = TRUE)
(sol <- ROI_solve (socp1))
## Optimal solution found.
## The objective value is: 2.000000e+00

Solution (sol)
##  x  y

##  1  1
```

$$\begin{array}{ll} minimizar & x_1+x_2-x_3 \\ sujeto\ a & x_1 \cdot \begin{pmatrix} 10 & 3 \\ 3 & 10 \end{pmatrix} + x_2 \cdot \begin{pmatrix} 6 & -4 \\ -4 & 10 \end{pmatrix} + x_3 \begin{pmatrix} 8 & 1 \\ 1 & 6 \end{pmatrix} \leq \begin{pmatrix} 16 & -13 \\ -13 & 60 \end{pmatrix} \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

```
A1 <- rbind (c(10, 3), c(3, 10))
A2 <- rbind (c(6, -4),  c(-4, 10))
A3 <- rbind (c(8, 1), c(1, 6))
A4 <- rbind (c(16, -13), c(-13, 60))
psd <- OP (objective = L_objective (c(1, 1, -1)),
          constraints = C_constraint (L = vech (A1, A2, A3),
                                      cones = K_psd (3),
                                      rhs = vech (A4)))
(sol <- ROI_solve (psd))
## Optimal solution found.

## The objective value is: -1.486461e+00

solution (sol)
## [1]   9.453073e-14   -1.781861e-10    1.486461e+00

as.matrix (solution (sol, "psd") [ [1] ] )
##                  [,1]                  [,2]

##   [1,] 0.11050023      0.031337483

##   [2,] 0.03133748      0.008887202
```

$$minimizar\ x_1 - x_2 + x_3$$

$$sujeto\ a\ \ x_1 \cdot \begin{pmatrix} -7 & -11 \\ -11 & 3 \end{pmatrix} + x_2 \cdot \begin{pmatrix} 7 & -18 \\ -18 & 8 \end{pmatrix} + x_3 \cdot \begin{pmatrix} -2 & -8 \\ -8 & 1 \end{pmatrix} \leq \begin{pmatrix} 33 & -9 \\ -9 & 26 \end{pmatrix}$$

$$x_1 \cdot \begin{pmatrix} -21 & -11 & 0 \\ -11 & 10 & 8 \\ 0 & 8 & 5 \end{pmatrix} + x_2 \cdot \begin{pmatrix} 0 & 10 & 16 \\ 10 & -10 & -10 \\ 16 & -10 & 3 \end{pmatrix} + x_3 \cdot \begin{pmatrix} -5 & 2 & -17 \\ 2 & -6 & 8 \\ -17 & 8 & 6 \end{pmatrix} \leq \begin{pmatrix} 14 & 9 & 40 \\ 9 & 91 & 10 \\ 40 & 10 & 15 \end{pmatrix}$$

$$x_1, x_2, x_3 \in \Re$$

```
obj <- c(1, -1, 1)
A1 <- matrix ( c (-7, -11, -11,  3), 2)
A2 <- matrix ( c ( 7, -18, -18,  8), 2)
A3 <- matrix ( c (-2,  -8,   -8,  1), 2)
A4 <- matrix ( c (33, -9,   -9, 26), 2)
B1 <- matrix ( c (-22, -11,    0, -11,  10,   8,   0,    8,  5), 3)
B2 <- matrix ( c (   0,  10,   16,  10, -10, -10, 16, -10, 3), 3)
B3 <- matrix ( c (  -5,    2, -17,    2,  -6,   8, -17,   8,  6), 3)
rhs <- c (vech (A4), vech (B4))

psd <- OP (objective = obj,
                    constraints = C_constraint (L = rbind (vech (A1, A2, A3), vech (B1, B2, B3)),
                                cones = K_psd ( c (3, 6)), rhs = rhs),
                    bounds = V_bound (li = 1 : 3, lb = rep (-Inf, 3)))

(sol <- ROI_solve (psd, solver = "scs"))
## Optimal solution found.

## The objective value is: -3.153545e+00

solution (sol)
##       [1]       -0.3677513       1.8983332       -0.8874604

lapply (solution (sol, type = "psd"), as.matrix)
## $`5`

##                          [,1]                        [,2]
## [1,]          0.003961386            -0.004339146

## [2,]          -0.004339146           0.004752929

## $`6`

##                              [,1]                        [,2]                        [,3]
## [1,]          0.055803081           -0.0024106685          0.024214084

## [2,]          -0.002410669           0.0001041398          -0.001046038

## [3,]          0.024214084           -0.0010460378          0.010506980
```

4.4 Problemas de optimización entero mixto

Resolución con R:

R cuenta con una función específica, que se encuentra en el paquete ***lpSolve*** para la resolución de este tipo de problemas, en concreto, lp.transport, la cual se basa en valores enteros y además su objetivo será minimizar nuestro problema.

Función:

```
lp.transport(cost.mat, direction="min",
        row.signs, row.rhs, col.signs,
        col.rhs, integers = 1)
```

*Argumentos de la función **lp.transport:***

cost.mat: matriz de costes; cada elemento es el coste de transportar un elemento de la fuente i al destino j.
direction: indicar si se trata de minimizar o maximizar "min" o "max".
row.signs: vector de caracteres para indicar la dirección de las restricciones de fila: cada carácter debe ser alguno de los siguientes "<", "≤", "=", ">" o "≥".
row.rhs: vector de valores numéricos para los lados derechos de las restricciones.
col.signs: vector de cadenas de caracteres dando la dirección de las restricciones de columna: cada carácter debe ser alguno de "<", "≤", "=", ">" o "≥".
col.rhs: vector de valores para los lados derechos de las restricciones de la columna.
Integers: vector de enteros. Su longitud será el número de variables enteras. Predeterminado: todas las variables son enteras. Si se indica en NULL no se tendrá en cuenta de ser ninguna variable entera.

*Salidas de la función **lp.transport:***

$objval: valor que alcanza la función objetivo.
$solution: cantidad transportada desde cada origen a cada destino.

*Aplicación práctica de la función **lp.transport:***

Planteamos el desarrollo de un ejercicio práctico que se basa en la necesidad de transportar los motores fabricados en Amsterdam (A), Amberes (B) y El Havre (C) a los puntos donde se demandan, en concreto, en Alemania (1), Francia(2), Bélgica (3), Países Bajos (4). Las cantidades ofertadas se recogen en la Figura 4.1, las demandas en la Figura 4.2 y para entender la relación entre la cantidad demandada y ofertada con respecto a los costes no basaremos en la Figura 4.3 que recoge los datos de forma clara y esquemática.

Figura 4.1: Cantidad oferta por ciudad

PLANTA	CANTIDADES DE MOTORES
1	400
2	900
3	200
4	500

Figura 4.2: Cantidad demandada por ciudad

Puerto	Motores disponibles
A	500
B	700
C	800

Figura 4.3: Demanda, oferta y coste

DESTINO	ORIGEN				DEMANDA
	1	2	3	4	
A	120	130	41	62	500
B	61	40	100	110	700
C	102.50	90	122	42	800
OFERTA	400	900	200	500	

En primer lugar debemos de introducir los objetos en R como hasta ahora, usando una matriz para indicar el coste del transporte y vectores para las disponibilidades de la oferta, de la demanda y de la dirección. En este último caso en particular, debemos de pensar que buscamos satisfacer la demanda, por tanto, tenemos que cumplir la disponibilidad indicada o superar este valor, por ello su dirección será "≥". Mientras que para la oferta es lógico pensar que no podemos transportar más cantidad de la fabricada por ello se indica "≤".

```
require(lpSolve)
F.cost <- matrix(c(120,130,41,62,61,40,100,110,102.5,90,122,42), nrow = 3, byrow = TRUE)
F.row <- c(500,700,800)
Dir.row <- c("<=", "<=", "<=")
F.col <- c(400,900,200,500)
Dir.col <- c(">=", ">=", ">=", ">=")
```

Para usar la función ***lp.transport*** necesitamos definir los nuevos argumentos llamados ***row.signs, row.rhs, col.signs,*** y ***col.rhs***, definidos anteriormente.

```
Trans <- lp.transport (cost.mat = F.cost, direction = "min", row.signs = Dir.row, row.rhs = F.row,
                       col.signs = Dir.col, col.rhs = F.col)
```

Podemos obtener de R el coste total del transporte, así como la cantidad que se transporte a cada ciudad y desde que punto es transportada, mostrándose en una tabla fácilmente de interpretar que nos confecciona el software (Figura 4.4), de esta forma observamos, por ejemplo, que se transportan desde Amsterdam a Alemania 300 unidades y 100 unidades de Bélgica a Alemania, satisfaciendo la demanda total de Alemania de 400 unidades.

Figura 4.4: Solución transporte en R

```
> Trans$objval
[1]  121450
> Trans$solution
     [,1] [,2] [,3] [,4]
[1,]  300    0  200    0
[2,]    0  700    0    0
[3,]  100  200    0  500
```

Capítulo 5

Optimización en el entorno R

5.1 Introducción a R

R es un entorno de programación, análisis estadístico y software gráfico derivado del lenguaje de programación S (Becker, Chambers y Wilks, 1988; Chambers, 1998; Chambers y Hastie, 1992; Venables y Ripley, 2000). La primera versión de R se desarrolló en el Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda) por **R**oss Ihaka y **R**obert Gentleman (Ihaka y Gentleman, 1996). Esta, junto a la fonética de R "our" –nuestro–, que enlaza R con el software libre, son las razones del nombre R. R se difundió rápidamente y la expansión es hoy irrefrenable. Desde su creación, R se alimenta y crece con los trabajos de investigadores provenientes de prácticamente todas las ramas del conocimiento. Las aportaciones continuas y desinteresadas de funciones y paquetes de propósito general o específico perfilan a R como un entorno dinámico formado por una comunidad activa y adherida a la filosofía del software libre.

R, en tanto en cuanto software libre, se asienta dentro del proyecto GNU *General Public Licence.* (Licencia Pública General, GNU). Se trata de una licencia creada por *Free Software Foundation* (Fundación para el software libre) organización fundada por Richard Matthew Stallman (rms) en el año 1985. El principal propósito de la licencia GNU es declarar la libertad del uso, modificación y distribución del software y protegerlo de intentos de privatización que puedan de algún modo restringir su uso (el contenido de la licencia puede consultarse en el sitio http://www.gnu.org/copyleft/gpl. html). Dentro de esta licencia se distribuyen un sinfín de programas, muchos de los cuales son versiones libres del software privativo generalista más utilizado. De entre ellos tal vez los más extendidos sean la suite ofimática *OpenOffice*, el navegador *Mozilla*, los artículos de *wikipedia,* el sistema operativo *GNU/Linux*, o el editor de textos *Emacs*.

R integra multitud de paquetes cuya continua incorporación al entorno R incrementan su capacidad y versatilidad. Si bien definiremos más adelante que son los paquetes, podríamos equipararlos con los módulos que ofrece el software privativo y comercial para el análisis de datos. R dispone de funciones básicas relacionadas con los análisis descriptivos de datos, y de los modelos más complejos y actuales concernientes con los últimos avances en el campo de la estadística, la psicometría, la econometría o el análisis de datos en áreas como la psicología, economía, sociología, estadística, biología, enfermería, farmacia, medicina o informática.

Aparte de las capacidades de análisis estadístico, R es un potentísimo generador de gráficos. Permite componer un gráfico simple, definir figuras extremadamente complejas e incluso crear animaciones (Maindonald y Braun, 2007; Murrel, 2005; Sarkar, 2008).
Entre otras características dispone de:

- Almacenamiento y manipulación efectiva de datos.
- Operadores para cálculo sobre variables indexadas (Arrays), en particular matrices.
- Una amplia, coherente e integrada colección de herramientas para análisis de datos.
- Posibilidades gráficas para análisis de datos, que funcionan directamente sobre pantalla o impresora.
- Un lenguaje de programación bien desarrollado, simple y efectivo, que incluye condicionales, ciclos, funciones recursivas y posibilidad de entradas y salidas. (Debe destacarse que muchas de las funciones suministradas con el sistema están escritas en el lenguaje R).

El término “entorno” lo caracteriza como un sistema completamente diseñado y coherente, antes que como una agregación incremental de herramientas muy específicas e inflexibles, como ocurre frecuentemente con otros programas de análisis de datos.

R es en gran parte un vehículo para el desarrollo de nuevos métodos de análisis interactivo de datos. Como tal es muy dinámico y las diferentes versiones no siempre son totalmente compatibles con las anteriores. Algunos usuarios prefieren los cambios debido a los nuevos métodos y tecnología que los acompañan, a otros sin embargo les molesta ya que algún código anterior deja de funcionar. Aunque R puede entenderse como un lenguaje de programación, los programas escritos en R deben considerarse esencialmente efímeros.

5.2 Paquetes de optimización en R

R tiene paquetes (unos cuantos miles en la actualidad) que extienden sus funciones básicas. Al abrir R se cargan automáticamente una serie de paquetes básicos.

La lista completa de los paquetes oficiales puede consultarse en CRAN (https://cran.r-project.org/)((Comprehensive R Archve Network) es el repositorio oficial de paquetes de R, el lugar donde se publican las nuevas versiones del programa, etc. Contiene la lista completa de paquetes oficiales). Además de los oficiales, existen paquetes que pueden instalarse desde lugares como, por ejemplo, Github.

No es sencillo encontrar el paquete que puede ser útil para un determinado fin. Las vistas de CRAN, descripciones de paquetes usados en un determinado ámbito y mantenidas por un experto en la materia, pueden ser un buen punto de partida.

La instalación de paquetes puede realizarse o bien desde la consola:

```
install.packages("nombre_paquete")
```

o bien a través de los menús de RStudio (bajo Tools). Una vez instalados los paquetes es

necesario cargarlos para que las funciones que contienen estén disponibles en la sesión:

library(nombre_paquete)

5.3 ROI

El paquete R Optimization Infrastructure (ROI) proporciona una infraestructura extensible para modelar problemas de optimización lineal, cuadrática, cónica y no lineal general de manera consistente.

```
library(ROI)
```

Además, la infraestructura administra muchos solucionadores, reformulaciones, colecciones de problemas y funciones diferentes para leer y escribir problemas de optimización en varios formatos.

El **ROI** proporciona las capacidades de modelado y administra los complementos, los complementos agregan los solucionadores al **ROI**.

```
Sys.setenv(ROI_LOAD_PLUGINS = "FALSE")
library(ROI)
library(ROI.plugin.glpk)
library(ROI.plugin.qpoases)
library(ROI.plugin.ecos)
library(ROI.plugin.scs)
library(ROI.plugin.alabama)
library(ROI.plugin.lpsolve)
library(ROI.plugin.gurobi)
```

Problema de optimización

```
str(OP)
## function (objective, constraints, types, bounds, maximum = FALSE)
```

Objetivo

Objetivo lineal

```
str(L_objective)
## function (L, names = NULL)
```

$$1\,x + 2\,y + 3\,z$$

```
lo <- L_objective(c(1, 2, 3), c("x", "y", "z"))
```

Objetivo cuadrático

```
str(Q_objective)
## function (Q, L = NULL, names = NULL)
```

$$\frac{1}{2}(x_1^2 + x_2^2 + x_3^2) + 1x_1 + 2x_2 + 3x_3$$

```
qo <- Q_objective(diag(3), c(1, 2, 3), c("x_1", "x_2", "x_3"))
```

Objetivo funcional

```
str(F_objective)
## function (F, n, G = NULL, H = NULL, names = NULL)
```

$$x_1^2 + x_2^2$$

```
fo <- F_objective(F = function(x) sum(x^2), n = 2,
                  G = function(x) 2*x, names = c("x_1", "x_2"))
```

Restricciones

Restricciones lineales

```
str(L_constraint)
## function (L, dir, rhs, names = NULL)
```

$$\begin{aligned} 3x + 4y + 1z &\leq 90 \\ 1x + 0y + 2z &\geq 55 \\ 1x + 1y + 0z &= 2 \end{aligned}$$

```
lc <- L_constraint(L = rbind(c(3, 4, 1), c(1, 0, 2), c(1, 1, 0)),
                   dir = c("<=", ">=", "=="), rhs = c(90, 5, 2),
                   names = c("x", "y", "z"))
```

Restricciones cuadráticas

```
str(Q_constraint)
## function (Q, L, dir, rhs, names = NULL)
```

$$\frac{1}{2}(x^2 + y^2) + 1x + 2y \leq 3$$

```
qc1 <- Q_constraint(Q = diag(2), L = 1:2, dir = "<=",
                    rhs = 3, names = c("x", "y"))
```

$$\begin{aligned} x^2 + y^2 + 3x + 1y &\leq 3 \\ x + y &\leq 4 \\ \frac{1}{2}(3x^2 + 3y^2 + 2xy) + 2x &\leq 9 \end{aligned}$$

```
qc2 <- Q_constraint(Q = list(diag(2, 2), NULL, matrix(c(3, 1, 1, 3), 2)),
                    L = rbind(c(3, 1), c(1, 1), c(2, 5)),
                    dir = c("<=", "<=", "<="),
                    rhs = c(3, 4, 9), names = c("x", "y"))
```

Restricciones funcionales

```
str(F_constraint)
## function (F, dir, rhs, J = NULL, names = NULL)
```

$$\begin{aligned} x^2 &\leq 2 \\ y^2 &\leq 4 \end{aligned}$$

```
fc1 <- F_constraint(F = function(x) x^2, dir = c("<=", "<="), rhs = c(2, 4),
                    J = function(x) diag(x = 2, nrow = 2) * x,
                    names = c("x", "y"))
```

o equivalente

```
fc2 <- F_constraint(F = list(function(x) x[1]^2, function(x) x[2]^2),
                    dir = c("<=", "<="), rhs = c(2, 4),
                    J = list(function(x) rbind(c(2, 0) * x),
                             function(x) rbind(c(0, 2) * x)),
                    names = c("x", "y"))
```

Bounds

Una variable Bounds es un tipo especial de restricción utilizada para restringir una variable objetivo entre un límite inferior y superior real. Por lo tanto, las variables Bounds a menudo también se denominan "box bounds" o "box contraints". Aunque las variables bounds podrían modelarse fácilmente como restricciones lineales, muchos GPS solo admiten variables bounds. Además, la mayoría de los solucionadores que admite cualquier tipo de restricción, permite especificar límites de variables directamente. Por lo tanto, es razonable. pero también es conveniente considerarlos por separado.
Típicamente, las implementaciones de algoritmos de optimización diferencian entre cinco tipos de observación. límites de la variable objetivo: libre $(-\infty, \infty)$, superior $(-\infty, ub]$, inferior $[lb, \infty)$, doble acotada $[lb, ub]$, y límites fijos. En ROI, las variables bounds se representan como una lista con dos elementos: superior e inferior, donde solo los valores no predeterminados se almacenan en un formato simple y disperso. En esto solo los índices de formato disperso y los valores no predeterminados se almacenan. Para los bounds inferiores el valor predeterminado es cero y para los bounds superiores el valor predeterminado es infinito. Por lo tanto, para OPs donde se requieren todas las variables para tomar valores en el intervalo $[0, \infty)$ no tiene que haber bounds especificado. El bounds inferior y superior predeterminados se pueden cambiar con los argumentos ld y ud.
Los bounds superior y/o inferior se especifican proporcionando el índice *i* de la variable correspondientes capaz (argumentos li, ui) y su límite inferior (lb) o superior (ub), respectivamente. Por lo tanto, se construyen restricciones de caja $-\infty \leq x_1 \leq 4$, $0 \leq x_2 \leq 100$, $2 \leq x_3 \leq \infty$ y $0 \leq x_4 \leq \infty$ en ROI de la siguiente manera:

```
R> V_bound (li = 1: 4, ui = 1: 4, lb = c (-Inf, 0, 2, 0),
+ ub = c (4, 100, Inf, Inf))
```

variables bounds de ROI:

2 bounds de variables no estándar inferiores y 2 superiores no estándar.

Si se proporcionan todos los valores superiores e inferiores (no se omiten los valores predeterminados), los índices pueden ser excluido:

```
R> V_bound (lb = c (-Inf, 0, 2, 0), ub = c (4, 100, Inf, Inf))
```

variables bounds de ROI:

2 bounds de variables no estándar inferiores y 2 superiores no estándar.

Si se omiten los valores predeterminados, se debe proporcionar el número de variables objetivas.

```
R> V_bound (li = c (1L, 3L), ui = c (1L, 2L), lb = c (-Inf, 2), ub = c (4, 100),
+ nobj = 4L)
```

variables bounds de ROI:

2 bounds de variables no estándar inferiores y 2 superiores no estándar.

Tenga en cuenta que la caja restringe $0 \leq x_3 \leq 20$ y $-20 \leq x_i \leq 20$ para todo $i \in I = \{1,2,4\}$. Este límite de variable se puede construir mediante el siguiente código de ROI.

R> V_bound (li = 3, lb = 0, ld = -20, ud = 20, nobj = 4L)

variables bounds de ROI:

3 variables bounds no estándar inferiores y 4 superiores no estándar.

ROI distingue entre dos tipos diferentes de límites:

1) Sin límite (No_bound)
2) Limites variables V_bound

Por defecto, los límites de las variables se establecen en $0 \leq x_i \leq \infty$ para todo $i = 1, \dots N$.

```
str(V_bound)
## function (li, ui, lb, ub, nobj, ld = 0, ud = Inf, names = NULL)
```

$$-3 \leq x_1 \leq 3, \quad -\infty \leq x_2 \leq 7, \quad -9 \leq x_3 \leq \infty$$

```
vb <- V_bound(li = 1:3, ui = 1:3, lb = c(-3, -Inf, -9), ub = c(3, 7, Inf))
```

La infraestructura de optimización R está estructurada en el ROI del paquete y su acompañamiento de extensiones (complementos y modelos). El ROI del paquete proporciona todas las clases y métodos necesarios y administra las extensiones (es decir, carga automáticamente los complementos y administra los metadatos sobre los complementos). Los paquetes de extensión agregan solucionadores de optimización, funciones de lectura/escritura y recursos adicionales (p. ej., colecciones de modelos). Las extensiones de plug-in juegan un papel especial, por lo tanto, todos los complementos se cargan automáticamente cuando se carga el ROI. Cuando se carga un complemento proporciona datos sobre sus capacidades. Estos datos se almacenan en una base de datos en memoria e incluye información sobre a qué problemas se aplica el complemento, qué formatos puede leer/escribir y los argumentos de control disponibles del solver y cómo el solver específico de los argumentos de control se relacionan con los argumentos comúnmente utilizados.
Este mecanismo hace posible que ROI conozca todos los complementos instalados, sin necesidad de cambiar ROI cuando se agrega un nuevo complemento. Para hacer posible la carga automática los complementos deben seguir la convención de nombres

ROI.plugin. <nombre>, donde <nombre> es normalmente el nombre de un solver de optimización (por ejemplo, ROI.plugin.glpk (Theußl 2017)). Los prefijo ROI.models (por ejemplo, ROI.models.netlib (Schwendinger 2016)) se utiliza para paquetes de datos con OPs predefinidos.

Después de formular un OP se puede resolver llamando a la función ROI_solve (x, solucionador, control, ...). Esta función toma un objeto R de la clase 'OP' que contiene la formulación del OP, el nombre del solucionador que se utilizará y una lista para resolver parámetros específicos del solver como argumentos. El solver y los argumentos de control son
opcionales, si no se proporciona un argumento de solución, el ROI elegirá una solución de resolución automáticamente. Alternativamente, los parámetros específicos del solver se pueden especificar a través de argumentos de puntos
El problema indicado se puede resolver con el siguiente código de ROI:
R> (lp_sol <- ROI_solve (lp, solver = "glpk"))

Los status code del solver se utilizan para informar al usuario sobre el estado de salida del solver. A pesar de el uso común de los status code en los solver de optimización no existe un estándar ampliamente utilizado.

Sin embargo, creemos que es deseable proporcionar status code unificados. Los status code utilizados en ROI_solve son simples y consistentes con la práctica común, devuelve 0 en caso de éxito (si se encontró una "solución" que cumple los requisitos específicos del solver) 1 de lo contrario. Para la optimización de solvers especializados en resolver LPs convexos, QPs y CPs se puede suponer que se encontró una solución, si el código de estado es 0. Desafortunadamente, lo mismo no es cierto para los no convexos QPs y GPS, aquí los códigos de estado son menos informativos. Algunos paquetes como optimx y Alabama verifique las condiciones de Karush-Kuhn-Tucker (KKT) para verificar que la solución encontrada cumpla con los requisitos criterios de un mínimo local.

La función principal de un paquete de infraestructura de optimización general debe tomar al menos tres argumentos:

problema que representa un objeto que contiene la descripción del problema de optimización (OP) correspondiente,

solucionador especificando el solucionador que se utilizará (por ejemplo, "glpk", "nlminb", "scs"),

control que contiene una lista de argumentos de control adicionales para el solucionador correspondiente.

El solucionador de argumentos y el control se entienden fácilmente, ya que desde el solucionador disponible espectro solo tenemos que elegir aquellos que sean capaces de manejar el OP correspondiente y (opcionalmente) proporcione los parámetros de control apropiados. Sin embargo, al construir el objeto del problema, de manera general e intuitiva, es una tarea desafiante que lleva a varios problemas de diseño.

Parece natural instanciar OP basados en una función objetivo, una o varias restricciones, tipos y límites de las variables objetivas, así como la dirección de optimización (si se busca un mínimo o un máximo).

R> biblioteca ("ROI")

la función objetivo depende principalmente de su forma funcional. Si la función objetivo es lineal (L) entonces es una práctica común suministrar solo un vector de coeficientes $a_0 \in \Re^n$. Para funciones objetivos cuadraticas (Q) de la forma $\frac{1}{2}x^T Q_0 x + a_0^T x$ *la* mayoría de los solvers toman un vector $a_0 \in \Re^n$ y una matriz $Q_0 \in \Re^{nxn}$ como entrada. Las funciones objetivos no lineales se representan como una función R que toma el vector de variables objetivas como argumento y devuelve el valor objetivo.

Los tipos de funciones objetivos y los constructores correspondientes implementados en **ROI** son: F, Q, L.

F La forma más general de una función objetivo se crea con el F_objetivo (F, n, G, H, nombres) constructor simplemente suministrando F, una función R que representa $f_0(x)$, y *n* la longitud de *x*. Opcionalmente, la información sobre el gradiente y el hessiano puede ser proporcionado a través de los argumentos G y H. Si no se proporciona un gradiente, se calculará numéricamente si es necesario. El argumento de nombres opcionales se propaga a la solución objeto para que la solución sea más legible.

Q Las funciones objetivas que representan una forma cuadrática como se describe anteriormente se pueden crear fácilmente con el constructor Q_objetivo (Q, L, nombres) tomando Q, la parte cuadrática Q_0, y opcionalmente L, la parte lineal a_0, como argumentos. El argumento de los nombres es nuevamente opcional.

L Si el objetivo a optimizar es una función lineal, entonces uno debe usar el L_objetivo (L, nombres) constructor que suministra L (los coeficientes de las variables objetivas) como un vector numérico. El argumento de los nombres es nuevamente opcional.

Los tres constructores devuelven un objeto heredado de la clase 'objetivo'.

Para modelar las clases de problemas nos bastaría con cuatro tipos de restricciones. Por lo tanto, los argumentos con el mismo nombre tienen la misma funcionalidad independientemente del tipo de restricción.

ROI distingue entre 5 tipos diferentes de restricciones: F, C, Q, L, Sin restricciones.

F La forma más general de restricciones puede expresar cualquier restricción representable por una función R. Se crean a través de F_constraint (F, dir, rhs, J, nombres). Aquí F es ya sea una función o una lista de funciones, dir es un vector de caracteres que da la dirección de la restricción y rhs es un vector numérico que da el lado derecho de la restricción.

Los argumentos opcionales J y nombres se pueden usar para proporcionar el jacobiano y la variable nombres de las restricciones.

C Las restricciones cónicas se construyen mediante la función C_constraint (L, conos, rhs, nombres), donde L puede ser un vector numérico de longitud *n* o una matriz de dimension $m \times n$.
Una restricción cónica puede estar compuesta por varios conos, donde cada tipo de cono puede ocurrir varias veces.
Actualmente el ROI implementa constructors para los conos K_zero, K_lin, K_soc, K_psd, K_expp, K_expd, K_powp y K_powd. A combinar diferentes conos se puede utilizar la combinación genérica $c( )$.

Q Las restricciones cuadráticas definidas en la ecuación 5 se pueden crear fácilmente con el constructor Q_constraint (Q, L, dir, rhs, nombres). Las restricciones cuadráticas Q se dan como un lista de longitud *m* donde las entradas son de $n \times n$ matrices o NULL.

L Las restricciones lineales se construyen mediante la función L_constraint (L, dir, rhs, names).

Todos los constructores devuelven un objeto que hereda de la clase 'restricción'. Ya que en muchas situaciones es deseable optimizar una función objetivo dada sujeta a restricciones compuestas de diferentes tipos, el ROI puede combinar multiples restricciones en una sola restricción usando el generic funciones c ( ) o rbind ( ). Dado que las matrices L y Q pueden volverse enormes pero típicamente son escasas usamos el formato de matriz de triplete simple del slam (Hornik, Meyer y Buchta 2016) paquete para almacenarlos internamente. En el formato de matriz de triplete simple (a veces referido como formato de lista de coordenadas) solo los índices de fila, los índices de columna y los valores de los elementos distintos de cero se almacenan en una lista. Elegimos el paquete slam no solo por eficiencia razones, sino también debido al hecho de que muchas API de solucionador exigen dicho formato.

Como es una práctica común en los solucionadores de enteros mixtos distingue entre tipos de variables continuo, entero y binario.Codificamos la opción variable con los siguientes caracteres:

"C" para continuo, "I" para entero y "B" para binario. Por defecto se asumen todas las variables de tipo continuo.

La infraestructura de optimización R está estructurada en el ROI del paquete y su acompañamiento extensiones (plug-ins y models). El paquete ROI proporciona todas las clases y métodos necesarios y administra las extensiones (es decir, carga automáticamente los complementos y administra los metadatos sobre los complementos). Los paquetes de extensión agregan solucionadores de optimización, funciones de lectura / escritura y recursos adicionales (p. ej., colecciones de modelos). Las extensiones de plug-in juegan un papel especial, por lo tanto, todos los complementos se cargan automáticamente cuando se carga el ROI. Cuando se carga un complemento

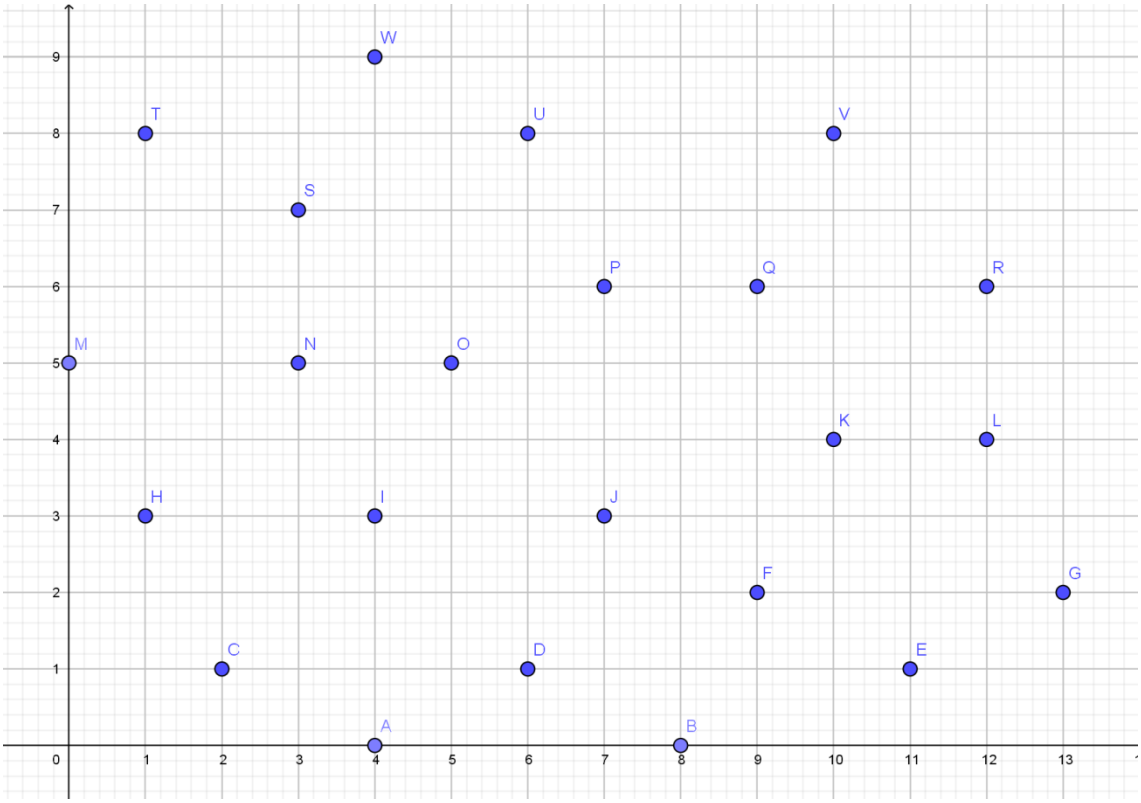
proporciona datos sobre sus capacidades. Estos datos se almacenan en una base de datos en memoria y incluye información sobre a qué problemas se aplica el complemento, qué formatos puede lectura / escritura y los argumentos de control disponibles del solucionador y cómo el solucionador específico
Los argumentos de control se relacionan con los argumentos comúnmente utilizados.

Este mecanismo hace posible que el ROI conozca todos los complementos instalados, sin necesidad de cambiar el ROI cuando se agrega un nuevo complemento. Para hacer posible la carga automática los complementos deben seguir la convención de nombres ROI.plugin. <nombre>, donde <nombre> es normalmente el nombre de un solucionador de optimización (por ejemplo, ROI.plugin.glpk (Theußl 2017)). Los prefijo ROI.models (por ejemplo, ROI.models.netlib (Schwendinger 2016)) se utiliza para paquetes de datos con OP predefinidos.

Capítulo 6

Caso práctico

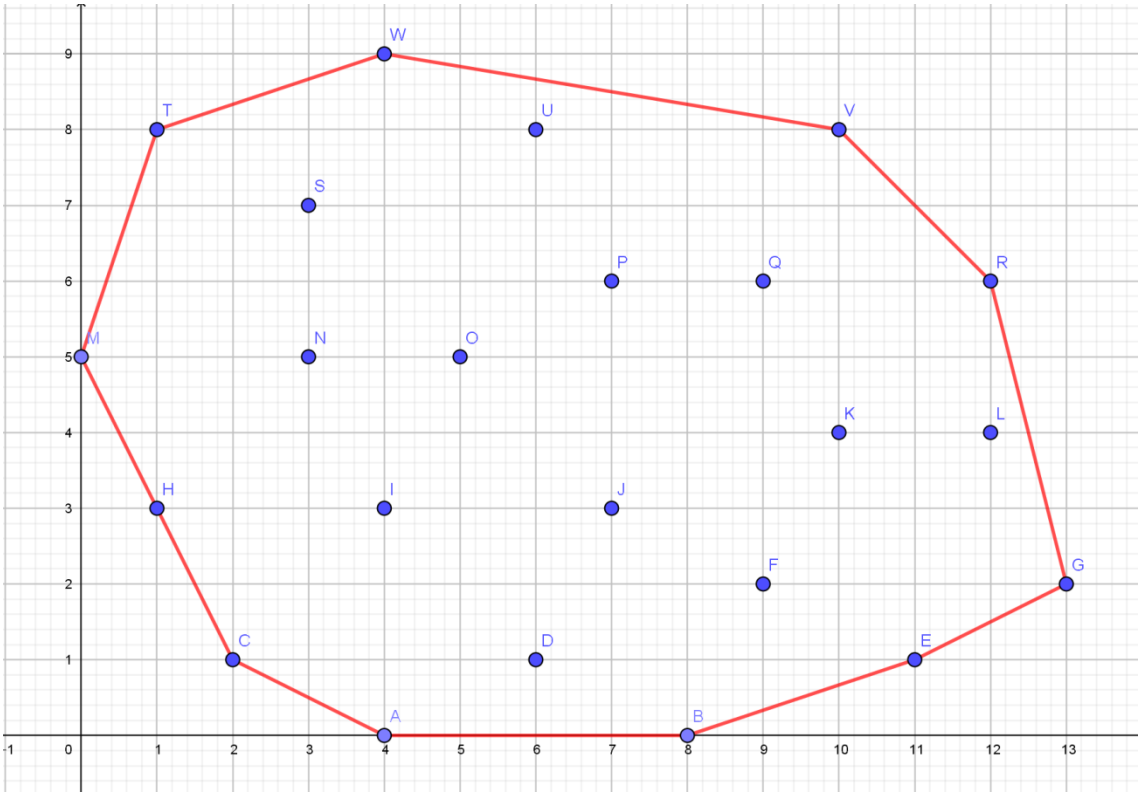
En este capítulo pretendemos resolver un problema de optimización.
Para ello vamos a considerar una ciudad ficticia considerablemente grande en la que pretendemos instalar un servicio de metro que una las distintas zonas o barrios de la ciudad.
Consideremos los puntos siguientes como paradas de nuestra ciudad ficticia:



La idea es tener 5 líneas de trenes y en cada línea tener una para cada sentido con lo que sería un total de 10 trenes.
Para calcular la línea 5 (línea circular) utilizamos el algoritmo marcha de Jarvis.
La marcha de Jarvis consiste en buscar el punto mas a la izquierda y a partir de él giramos (en sentido contrario a las agujas del reloj) una semirecta vertical hasta encontrar el siguiente punto de la envolvente.

Repetimos el proceso hasta volver al punto de partida.

En nuestro caso el punto mas a la izquierda es la parada M



De esta forma la línea circular estaría compuesta por las paradas A, B, E, G, R, V, W, T, M, H y C.
Para las otras 4 líneas de trenes vamos a utilizar K-means (algoritmo de agrupamiento) con el cual dividiremos las paradas en 4 grupos y a las paradas de cada grupo calcularemos la ruta óptima.
K-means es un método de agrupamiento (Clustering), es un algoritmo no supervisado por lo que no tiene una variable dependiente. Lo que trata de buscar en las observaciones son grupos con características similares, las observaciones en cada grupo han de ser similares pero diferentes a los otros grupos, es lo que entendiriamos por maximizar la variación inter-cluster y minimizar la intra-cluster.
Como ventajas es que es más rápido y el almacenamiento es económico (solo necesita guardar los k centroides).
Como desventaja es que hay que ir probando el número de clusters y presenta debilidades frente a outliers.
Elegiremos una parada común a dos grupos colindantes para facilitar el transbordo de un grupo a otro grupo que mas adelante explicaremos como cogemos este parada.

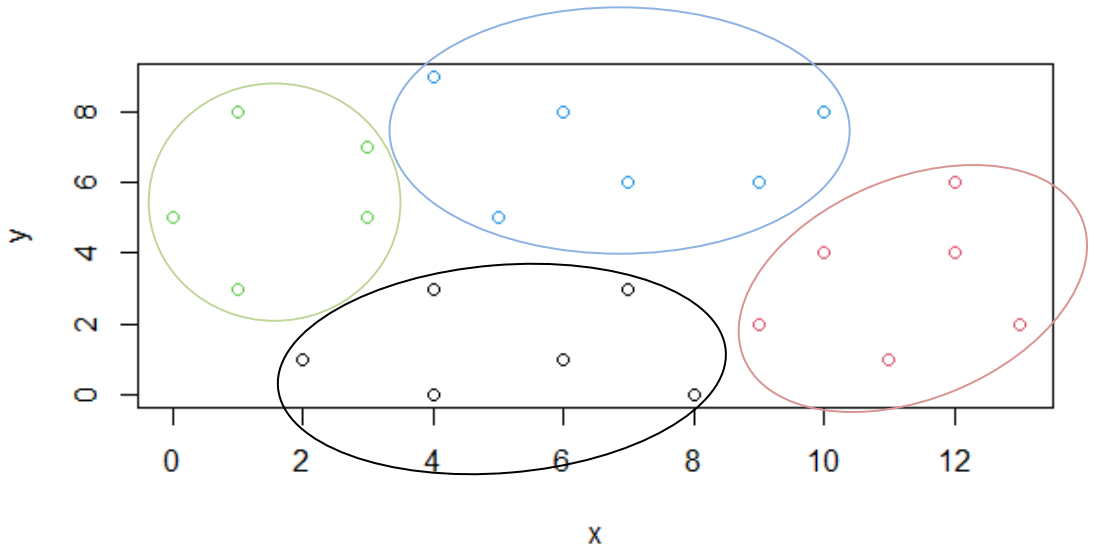
Las coordenadas de cada parada se refleja en la tabla siguiente:

PARADA	X	Y
A	4	0
B	8	0
C	2	1
D	6	1
E	11	1
F	9	2
G	13	2
H	1	3
I	4	3
J	7	3
K	10	4
L	12	4
M	0	5
N	3	5
O	5	5
P	7	6
Q	9	6
R	12	6
S	3	7
T	1	8
U	6	8
V	10	8
W	4	9

```
> x=c(4,8,2,6,11,9,13,1,4,7,10,12,0,3,5,7,9,12,3,1,6,10,4)  #coordenada x
> y=c(0,0,1,1,1,2,2,3,3,3,4,4,5,5,5,6,6,6,7,8,8,8,9)        #coordenada y
>
paradas=c("A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T"
,"U","V","W")     #nombre de las paradas
> xy.data=data.frame(x,y,row.names=paradas) #asignar una tabla a los nombres con sus
coordenadas
> print(xy.data)  #imprimir los datos en forma de tabla
   x y
A  4 0
B  8 0
C  2 1
D  6 1
E 11 1
F  9 2
G 13 2
H  1 3
I  4 3
J  7 3
```

```
K 10 4
L 12 4
M  0 5
N  3 5
O  5 5
P  7 6
Q  9 6
R 12 6
S  3 7
T  1 8
U  6 8
V 10 8
W  4 9
> plot(xy.data)    #dibuja los puntos de las paradas
> text(xy.data,labels=paradas,pos=4)  #asigna a cada punto su nombre
> xy.m1=dist(xy.data,"euclidean",TRUE,TRUE)    #calcula la distancia auclidea entre cada
punto y los restantes
> print(xy.m1)     #imprime una tabla con las distancias
          A         B         C         D         E         F         G         H
A  0.000000  4.000000  2.236068  2.236068  7.071068  5.385165  9.219544  4.242641
B  4.000000  0.000000  6.082763  2.236068  3.162278  2.236068  5.385165  7.615773
C  2.236068  6.082763  0.000000  4.000000  9.000000  7.071068 11.045361  2.236068
D  2.236068  2.236068  4.000000  0.000000  5.000000  3.162278  7.071068  5.385165
E  7.071068  3.162278  9.000000  5.000000  0.000000  2.236068  2.236068 10.198039
F  5.385165  2.236068  7.071068  3.162278  2.236068  0.000000  4.000000  8.062258
G  9.219544  5.385165 11.045361  7.071068  2.236068  4.000000  0.000000 12.041595
H  4.242641  7.615773  2.236068  5.385165 10.198039  8.062258 12.041595  0.000000
I  3.000000  5.000000  2.828427  2.828427  7.280110  5.099020  9.055385  3.000000
J  4.242641  3.162278  5.385165  2.236068  4.472136  2.236068  6.082763  6.000000
K  7.211103  4.472136  8.544004  5.000000  3.162278  2.236068  3.605551  9.055385
L  8.944272  5.656854 10.440307  6.708204  3.162278  3.605551  2.236068 11.045361
M  6.403124  9.433981  4.472136  7.211103 11.704700  9.486833 13.341664  2.236068
N  5.099020  7.071068  4.123106  5.000000  8.944272  6.708204 10.440307  2.828427
O  5.099020  5.830952  5.000000  4.123106  7.211103  5.000000  8.544004  4.472136
P  6.708204  6.082763  7.071068  5.099020  6.403124  4.472136  7.211103  6.708204
Q  7.810250  6.082763  8.602325  5.830952  5.385165  4.000000  5.656854  8.544004
R 10.000000  7.211103 11.180340  7.810250  5.099020  5.000000  4.123106 11.401754
S  7.071068  8.602325  6.082763  6.708204 10.000000  7.810250 11.180340  4.472136
T  8.544004 10.630146  7.071068  8.602325 12.206556 10.000000 13.416408  5.000000
U  8.246211  8.246211  8.062258  7.000000  8.602325  6.708204  9.219544  7.071068
V 10.000000  8.246211 10.630146  8.062258  7.071068  6.082763  6.708204 10.295630
W  9.000000  9.848858  8.246211  8.246211 10.630146  8.602325 11.401754  6.708204
          I         J         K         L         M         N         O         P
A  3.000000  4.242641  7.211103  8.944272  6.403124  5.099020  5.099020  6.708204
B  5.000000  3.162278  4.472136  5.656854  9.433981  7.071068  5.830952  6.082763
C  2.828427  5.385165  8.544004 10.440307  4.472136  4.123106  5.000000  7.071068
D  2.828427  2.236068  5.000000  6.708204  7.211103  5.000000  4.123106  5.099020
E  7.280110  4.472136  3.162278  3.162278 11.704700  8.944272  7.211103  6.403124
F  5.099020  2.236068  2.236068  3.605551  9.486833  6.708204  5.000000  4.472136
G  9.055385  6.082763  3.605551  2.236068 13.341664 10.440307  8.544004  7.211103
H  3.000000  6.000000  9.055385 11.045361  2.236068  2.828427  4.472136  6.708204
I  0.000000  3.000000  6.082763  8.062258  4.472136  2.236068  2.236068  4.242641
J  3.000000  0.000000  3.162278  5.099020  7.280110  4.472136  2.828427  3.000000
K  6.082763  3.162278  0.000000  2.000000 10.049876  7.071068  5.099020  3.605551
L  8.062258  5.099020  2.000000  0.000000 12.041595  9.055385  7.071068  5.385165
M  4.472136  7.280110 10.049876 12.041595  0.000000  3.000000  5.000000  7.071068
N  2.236068  4.472136  7.071068  9.055385  3.000000  0.000000  2.000000  4.123106
O  2.236068  2.828427  5.099020  7.071068  5.000000  2.000000  0.000000  2.236068
P  4.242641  3.000000  3.605551  5.385165  7.071068  4.123106  2.236068  0.000000
Q  5.830952  3.605551  2.236068  3.605551  9.055385  6.082763  4.123106  2.000000
R  8.544004  5.830952  2.828427  2.000000 12.041595  9.055385  7.071068  5.000000
S  4.123106  5.656854  7.615773  9.486833  3.605551  2.000000  2.828427  4.123106
T  5.830952  7.810250  9.848858 11.704700  3.162278  3.605551  5.000000  6.324555
U  5.385165  5.099020  5.656854  7.211103  6.708204  4.242641  3.162278  2.236068
V  7.810250  5.830952  4.000000  4.472136 10.440307  7.615773  5.830952  3.605551
W  6.000000  6.708204  7.810250  9.433981  5.656854  4.123106  4.123106  4.242641
          Q         R         S         T         U         V         W
A  7.810250 10.000000  7.071068  8.544004  8.246211 10.000000  9.000000
B  6.082763  7.211103  8.602325 10.630146  8.246211  8.246211  9.848858
C  8.602325 11.180340  6.082763  7.071068  8.062258 10.630146  8.246211
D  5.830952  7.810250  6.708204  8.602325  7.000000  8.062258  8.246211
E  5.385165  5.099020 10.000000 12.206556  8.602325  7.071068 10.630146
```

```
F  4.000000  5.000000  7.810250 10.000000  6.708204  6.082763  8.602325
G  5.656854  4.123106 11.180340 13.416408  9.219544  6.708204 11.401754
H  8.544004 11.401754  4.472136  5.000000  7.071068 10.295630  6.708204
I  5.830952  8.544004  4.123106  5.830952  5.385165  7.810250  6.000000
J  3.605551  5.830952  5.656854  7.810250  5.099020  5.830952  6.708204
K  2.236068  2.828427  7.615773  9.848858  5.656854  4.000000  7.810250
L  3.605551  2.000000  9.486833 11.704700  7.211103  4.472136  9.433981
M  9.055385 12.041595  3.605551  3.162278  6.708204 10.440307  5.656854
N  6.082763  9.055385  2.000000  3.605551  4.242641  7.615773  4.123106
O  4.123106  7.071068  2.828427  5.000000  3.162278  5.830952  4.123106
P  2.000000  5.000000  4.123106  6.324555  2.236068  3.605551  4.242641
Q  0.000000  3.000000  6.082763  8.246211  3.605551  2.236068  5.830952
R  3.000000  0.000000  9.055385 11.180340  6.324555  2.828427  8.544004
S  6.082763  9.055385  0.000000  2.236068  3.162278  7.071068  2.236068
T  8.246211 11.180340  2.236068  0.000000  5.000000  9.000000  3.162278
U  3.605551  6.324555  3.162278  5.000000  0.000000  4.000000  2.236068
V  2.236068  2.828427  7.071068  9.000000  4.000000  0.000000  6.082763
W  5.830952  8.544004  2.236068  3.162278  2.236068  6.082763  0.000000
> kmeans.res=kmeans(xy.data,centers=4)     #agrupa los datos en 4 clusters
> plot(xy.data,col=kmeans.res$cluster)     #dibuja los cluster cada uno de un color
```



Según lo hemos definido han salido 4 clusters compuesto por las paradas siguientes:

- Cluster 1: A, B, C, D, I y J.
- Cluster 2: E, F, G, K, L y R.
- Cluster 3: P, Q, U y V.
- Cluster 4: H, M, N, O, S, T y W.

Una vez los 4 clusters voy a buscar una parada comun a los cuatro clusters, esta parada será centrada a todos los clusters.

```
> x=c(2.375,7.2,11.6,6.8)
> y=c(4.625,7.4,3.4,1.2)
> clus=c("C1","C2","C3","C4")
> xy.clus=data.frame(x,y,row.names = clus)
> print(xy.clus)
```

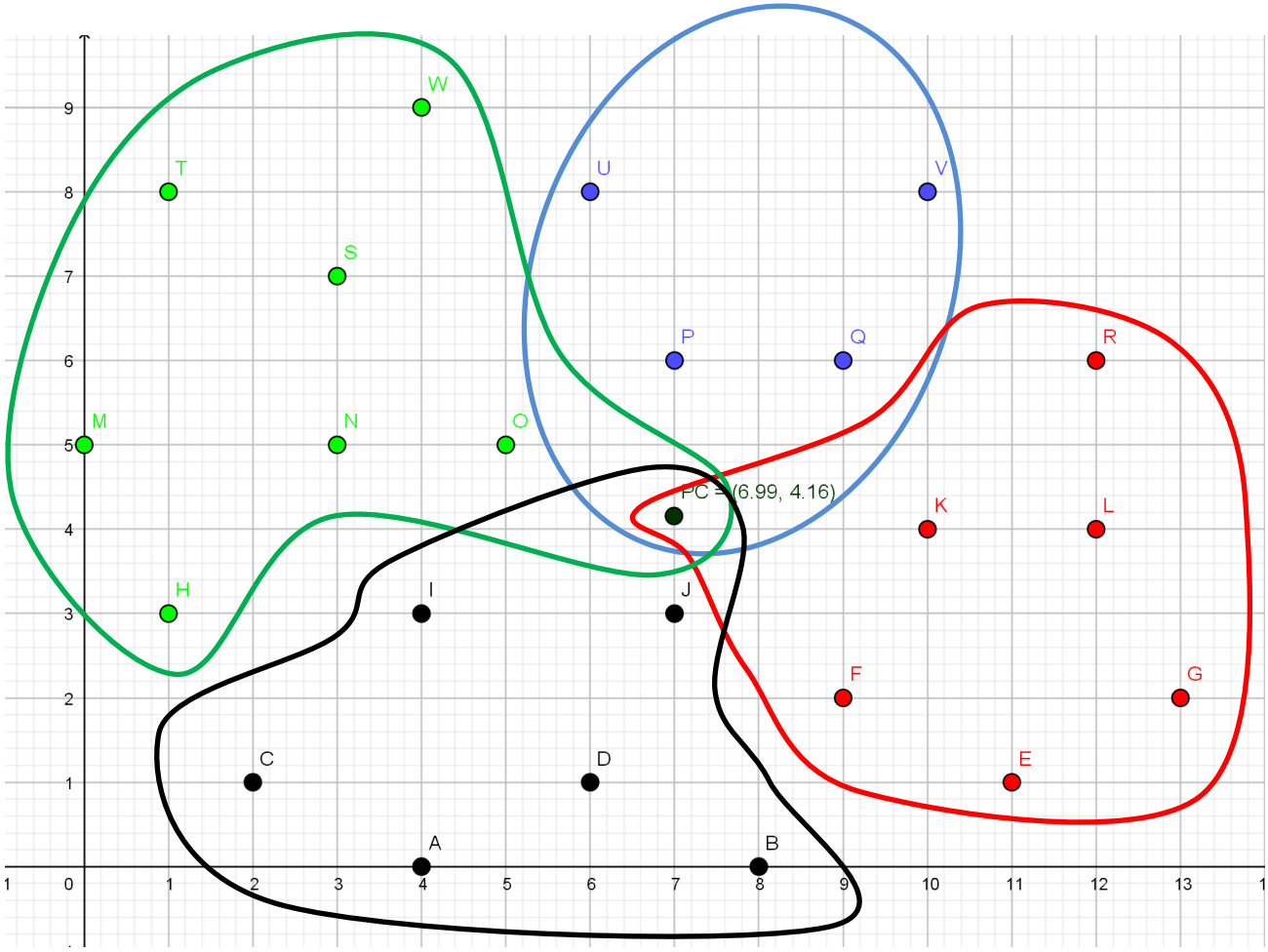
```
        x     y
C1  2.375 4.625
C2  7.200 7.400
C3 11.600 3.400
C4  6.800 1.200
> plot(xy.clus)
> text(xy.clus,labels=clus,pos=4)
> xy.m2=dist(xy.clus,"euclidean",TRUE,TRUE)
> print(xy.m2)
         C1       C2       C3       C4
C1 0.000000 5.566080 9.305979 5.595646
C2 5.566080 0.000000 5.946427 6.212890
C3 9.305979 5.946427 0.000000 5.280152
C4 5.595646 6.212890 5.280152 0.000000
> kmeans.clus<-kmeans(xy.clus,centers=1)
> print(kmeans.clus)
K-means clustering with 1 clusters of sizes 4

Cluster means:
        x       y
1 6.99375 4.15625

Clustering vector:
C1 C2 C3 C4
 1  1  1  1

Within cluster sum of squares by cluster:
[1] 62.68344
 (between_SS / total_SS =   0.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

Con lo que la media de los cuatro clusters nos da otro punto de coordenadas (6.99375,4.15625) que será la parada central (PC) por donde pasaran todos los trenes.

De esta forma las paradas de cada cluster serán:

- Cluster 1: A, B, C, D, I, J y PC.
- Cluster 2: E, F, G, PC, K, L y R.
- Cluster 3: PC, P, Q, U y V.
- Cluster 4: H, M, N, O, S, T, W y PC.

Gráficamente quedaría como se indica a continuación:



Ahora que tenemos los 4 clusters vamos a buscar la ruta más corta dentro de cada clusters.

Para el cluster 1 (C1) que esta formado por las paradas A, B, C, D, I, J y PC:

```
x=c(4,8,2,6,4,7,6.99375)
> y=c(0,0,1,1,3,3,4.15625)
> paradas=c("A","B","C","D","I","J","PC")
> xy.data=data.frame(x,y,row.names=paradas)
> print(xy.data)
```

```
          x       y
A  4.00000 0.00000
B  8.00000 0.0000x0
C  2.00000 1.00000
D  6.00000 1.00000
I  4.00000 3.00000
J  7.00000 3.00000
PC 6.99375 4.15625
> plot(xy.data)
> text(xy.data,labels=paradas,pos=4)
> xy.m1=dist(xy.data,"euclidean",TRUE,TRUE)
> print(xy.m1)
           A        B        C        D        I        J        PC
A  0.000000 4.000000 2.236068 2.236068 3.000000 4.242641 5.122202
B  4.000000 0.000000 6.082763 2.236068 5.000000 3.162278 4.276325
C  2.236068 6.082763 0.000000 4.000000 2.828427 5.385165 5.907576
D  2.236068 2.236068 4.000000 0.000000 2.828427 2.236068 3.308996
I  3.000000 5.000000 2.828427 2.828427 0.000000 3.000000 3.209276
J  4.242641 3.162278 5.385165 2.236068 3.000000 0.000000 1.156267
PC 5.122202 4.276325 5.907576 3.308996 3.209276 1.156267 0.000000
```

Vamos a representar por la variable x_{ij} como la arista que va de i a j. dicha variable tomara el valor 1 si pertenece a la ruta óptima, 0 en otro caso.

Así la función a minimizar es:

$$\begin{aligned} mín\ Z = {} & 4x_{AB} + 2{,}236068x_{AC} + 2{,}236068x_{AD} + 3x_{AI} + 4.242641x_{AJ} + 5{,}122202x_{APC} + 4x_{BA} \\ & + 6{,}082763x_{BC} + 2{,}236068x_{BD} + 5x_{BI} + 3{,}162278x_{BJ} + 4{,}276325x_{BPC} \\ & + 2{,}236068x_{CA} + 6{,}082763x_{CB} + 4x_{CD} + 2{,}828427x_{CI} + 5{,}385163x_{CJ} \\ & + 5{,}907576x_{CPC} + 2{,}236068x_{DA} + 2{,}236068x_{DB} + 4x_{DC} + 2{,}828427x_{DI} \\ & + 2{,}236068x_{DJ} + 3{,}308996x_{DPC} + 3x_{IA} + 5x_{IB} + 2{,}828427x_{IC} + 2{,}828427x_{ID} \\ & + 3x_{IJ} + 3{,}209276x_{IPC} + 4{,}242641x_{JA} + 3{,}162278x_{JB} + 5{,}385165x_{JC} \\ & + 2{,}236068x_{JD} + 3x_{JI} + 1{,}156267x_{JPC} + 5{,}122202x_{PCA} + 4{,}276325x_{PCB} \\ & + 5{,}907576x_{PCC} + 3{,}308996x_{PCD} + 3{,}209276x_{PCI} + 1{,}156267x_{PCJ} \end{aligned}$$

Y esta sujeta a las restricciones que dividiremos en tres grupos: restricciones de llegada, restricciones de salida y ruptura de subcircuitos.

Restricciones de llegada:

$$\begin{cases} x_{BA} + x_{CA} + x_{DA} + x_{IA} + x_{JA} + x_{PCA} = 1 \\ x_{AB} + x_{CB} + x_{DB} + x_{IB} + x_{JB} + x_{PCB} = 1 \\ x_{AC} + x_{BC} + x_{DC} + x_{IC} + x_{JC} + x_{PCC} = 1 \\ x_{AD} + x_{BD} + x_{CD} + x_{ID} + x_{JD} + x_{PCD} = 1 \\ x_{AI} + x_{BI} + x_{CI} + x_{DI} + x_{JI} + x_{PCI} = 1 \\ x_{AJ} + x_{BJ} + x_{CJ} + x_{DJ} + x_{IJ} + x_{PCJ} = 1 \\ x_{APC} + x_{BPC} + x_{CPC} + x_{DPC} + x_{IPC} + x_{JPC} = 1 \end{cases}$$

Restricciones de salida:

$$\begin{cases} x_{AB} + x_{AC} + x_{AD} + x_{AI} + x_{AJ} + x_{APC} = 1 \\ x_{BA} + x_{BC} + x_{BD} + x_{BI} + x_{BJ} + x_{BPC} = 1 \\ x_{CA} + x_{CB} + x_{CD} + x_{CI} + x_{CJ} + x_{CPC} = 1 \\ x_{DA} + x_{DB} + x_{DC} + x_{DI} + x_{DJ} + x_{DPC} = 1 \\ x_{IA} + x_{IB} + x_{IC} + x_{ID} + x_{IJ} + x_{IPC} = 1 \\ x_{JA} + x_{JB} + x_{JC} + x_{JD} + x_{JI} + x_{JPC} = 1 \\ x_{PCA} + x_{PCB} + x_{PCC} + x_{PCD} + x_{PCI} + x_{PCJ} = 1 \end{cases}$$

Restricciones de ruptura de subcircuitos: Como solo hay 7 paradas nos valdria con escribir rupturas de tamaño 2. Por ello obtenemos sólo 21 ecuaciones (6+5+4+3+2+1=21):

$$\begin{cases} x_{AB} + x_{BA} \leq 1 \\ x_{AC} + x_{CA} \leq 1 \\ x_{AD} + x_{DA} \leq 1 \\ x_{AI} + x_{IA} \leq 1 \\ x_{AJ} + x_{JA} \leq 1 \\ x_{APC} + x_{PCA} \leq 1 \\ x_{BC} + x_{CB} \leq 1 \\ x_{BD} + x_{DB} \leq 1 \\ x_{BI} + x_{IB} \leq 1 \\ x_{BJ} + x_{JB} \leq 1 \\ x_{BPC} + x_{PCB} \leq 1 \\ x_{CD} + x_{DC} \leq 1 \\ x_{CI} + x_{IC} \leq 1 \\ x_{CJ} + x_{JC} \leq 1 \\ x_{CPC} + x_{PCC} \leq 1 \\ x_{DI} + x_{ID} \leq 1 \\ x_{DJ} + x_{JD} \leq 1 \\ x_{DPC} + x_{PCD} \leq 1 \\ x_{IJ} + x_{JI} \leq 1 \\ x_{IPC} + x_{PCI} \leq 1 \\ x_{JPC} + x_{PCJ} \leq 1 \end{cases}$$

Para implementar el modelo en R y ejecutar las funciones de optimización, debemos representar los datos del modelo mediante matrices y vectores.

La función ***lp*** de ***lpSolver*** necesita una matriz formada por los coeficientes de las variables en las restricciones que Llamaremos ***const.coef*** y los vectores para los coeficientes de la función objetivo (***obj.coef***), las direcciones de las restricciones, ***strings***, y los valores b_m, ***b.coef***. Estos mismos elementos los utilizaremos para la función ***solveLP*** de ***linprog***.

Comenzamos el trabajo en R cargando los paquetes que vamos a emplear

```
> library(lpSolve)
> library(linprog)
```

y a continuación iremos creando los conjunto de valores. En primer lugar tenemos el vector de c_i, denominado ***obj.coef***,

```
> obj.coef<- c (4, 2.236068, 2.236068, 3, 4.242641, 5.122202, 4,
6.082763, 2.236068, 5, 3.162278, 4.276325, 2.236068, 6.082763, 4,
2.828427, 5.385163, 5.907576, 2.236068, 2.236068, 4, 2.828427, 2.236068,
3.308996, 3, 5, 2.828427, 2.828427, 3, 3.209276, 4.242641, 3.162278,
5.385165, 2.236068, 3, 1.156267, 5.122202, 4.276325, 5.907576, 3.308996,
3.209276, 1.156267)
```

Seguido de la matriz ***const.coef*** con los coeficientes de las restricciones, con tantas filas como número de restricciones y columnas como número de variables del modelo.

```
> a<- c(0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,
0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,
0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,
0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,
1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)
> const.coef<-matrix(a,nrow=35,byrow=T)
```

Y por último el vector ***strings*** para las desigualdades de las restricciones y ***b.coef*** para los valores b de las mismas

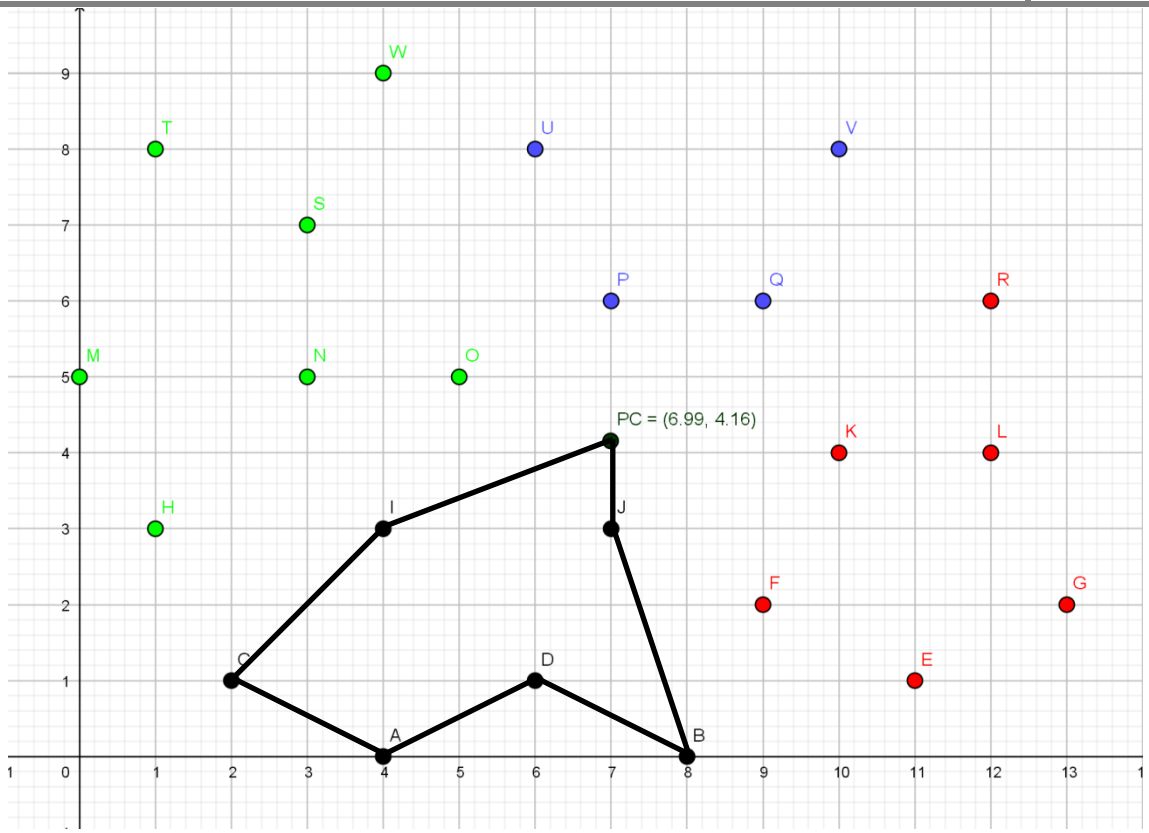
```
> strings<- c("=", "=", "=", "=", "=", "=", "=", "=", "=", "=", "=", "=",
"=", "=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=",
"<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=")
> b.coef<- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```

Para poder ejecurar lp y así poder ver si el modelo resulta válido o no.

```
> lpSolve1<- lp(direction="min", obj.coef, const.coef, strings, b.coef)
> lpSolve1$solution

[1] 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0

[41] 1 0
```

Con lo que me está diciendo que los caminos a coger son:

$$x_{AD}, x_{BJ}, x_{CA}, x_{DB}, x_{IC}, x_{JPC}, x_{PCI}$$



Para el cluster 2 (C2) que esta formado por las paradas E, F, G, PC, K, L y R:

```
> x=c(11,9,13,6.99375,10,12,12)
> y=c(1,2,2,4.15625,4,4,6)
> paradas=c("E","F","G","PC","K","L","R")
> xy.data=data.frame(x,y,row.names=paradas)
> print(xy.data)
          x       y
E  11.00000 1.00000
F   9.00000 2.00000
G  13.00000 2.00000
PC  6.99375 4.15625
K  10.00000 4.00000
L  12.00000 4.00000
R  12.00000 6.00000
> plot(xy.data)
> text(xy.data,labels=paradas,pos=4)
> xy.m1=dist(xy.data,"euclidean",TRUE,TRUE)
> print(xy.m1)
          E        F        G        PC        K        L        R
E  0.000000 2.236068 2.236068 5.100191 3.162278 3.162278 5.099020
F  2.236068 0.000000 4.000000 2.945242 2.236068 3.605551 5.000000
G  2.236068 4.000000 0.000000 6.381571 3.605551 2.236068 4.123106
PC 5.100191 2.945242 6.381571 0.000000 3.010308 5.008688 5.334975
K  3.162278 2.236068 3.605551 3.010308 0.000000 2.000000 2.828427
L  3.162278 3.605551 2.236068 5.008688 2.000000 0.000000 2.000000
R  5.099020 5.000000 4.123106 5.334975 2.828427 2.000000 0.000000
```

La función a minimizar seria:

$$\begin{aligned} mín\ Z = {} & 2{,}236068x_{EF} + 2{,}236068x_{EG} + 5{,}100191x_{EPC} + 3{,}162278x_{EK} + 3{,}162278x_{EL} \\ & + 5{,}099020X_{ER} + 2{,}236068X_{FE} + 4x_{FG} + 2{,}945242x_{FPC} + 2{,}236068x_{FK} \\ & + 3{,}605551x_{FL} + 5x_{FR} + 2{,}236068x_{GE} + 4x_{GF} + 6{,}381571x_{GPC} + 3{,}605551x_{GK} \\ & + 2{,}236068x_{GL} + 4{,}123106x_{GR} + 5{,}100191x_{PCE} + 2{,}945242x_{PCF} \\ & + 6{,}381571x_{PCG} + 3{,}010308x_{PCK} + 5{,}008688x_{PCL} + 5{,}334975x_{PCR} \\ & + 3{,}162278x_{KE} + 2{,}236068x_{KF} + 3{,}605551x_{KG} + 3{,}010308x_{KPC} + 2x_{KL} \\ & + 2{,}828427x_{KR} + 3{,}162278x_{LE} + 3{,}605551x_{LF} + 2{,}236068x_{LG} + 5{,}008688x_{LPC} \\ & + 2x_{LK} + 2x_{LR} + 5{,}099020x_{RE} + 5x_{RF} + 4{,}123106x_{RG} + 5{,}334975x_{RPC} \\ & + 2{,}828427x_{RK} + 2x_{RL} \end{aligned}$$

Restricciones de llegada:

$$\begin{cases} x_{FE} + x_{GE} + x_{PCE} + x_{KE} + x_{LE} + x_{RE} = 1 \\ x_{EF} + x_{GF} + x_{PCF} + x_{KF} + x_{LF} + x_{RF} = 1 \\ x_{EG} + x_{FG} + x_{PCG} + x_{KG} + x_{LG} + x_{RG} = 1 \\ x_{EPC} + x_{FPC} + x_{GPC} + x_{KPC} + x_{LPC} + x_{RPC} = 1 \\ x_{EK} + x_{FK} + x_{GK} + x_{PCK} + x_{LK} + x_{RK} = 1 \\ x_{EL} + x_{FL} + x_{GL} + x_{PCL} + x_{KL} + x_{RL} = 1 \\ x_{ER} + x_{FR} + x_{GR} + x_{PCR} + x_{KR} + x_{LR} = 1 \end{cases}$$

Restricciones de salida:

$$\begin{cases} x_{EF} + x_{EG} + x_{EPC} + x_{EK} + x_{EL} + x_{ER} = 1 \\ x_{FE} + x_{FG} + x_{FPC} + x_{FK} + x_{FL} + x_{FR} = 1 \\ x_{GE} + x_{GF} + x_{GPC} + x_{GK} + x_{GL} + x_{GR} = 1 \\ x_{PCE} + x_{PCF} + x_{PCG} + x_{PCK} + x_{PCL} + x_{PCR} = 1 \\ x_{KE} + x_{KF} + x_{KG} + x_{KPC} + x_{KL} + x_{KR} = 1 \\ x_{LE} + x_{LF} + x_{LG} + x_{LPC} + x_{LK} + x_{LR} = 1 \\ x_{RE} + x_{RF} + x_{RG} + x_{RPC} + x_{RK} + x_{RL} = 1 \end{cases}$$

Restricciones de ruptura:

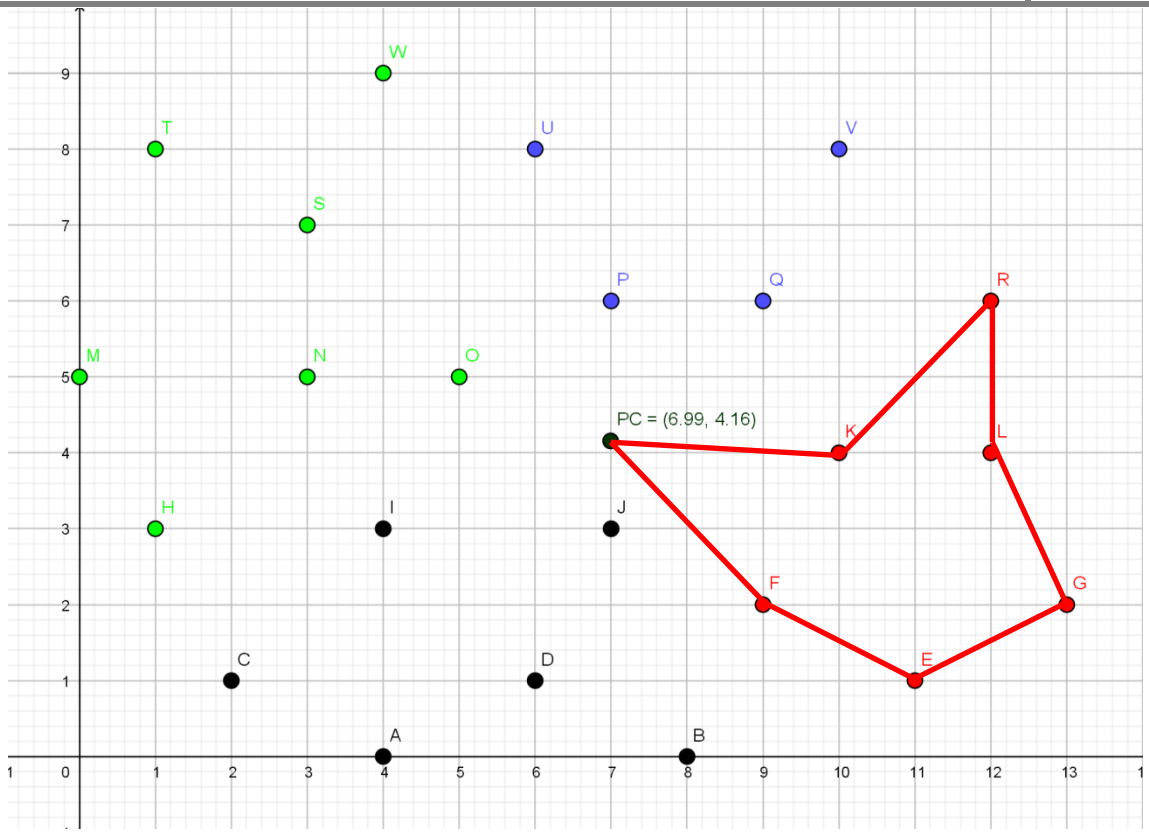
$$\begin{cases} x_{EF} + x_{FE} \leq 1 \\ x_{EG} + x_{GE} \leq 1 \\ x_{EPC} + x_{PCE} \leq 1 \\ x_{EK} + x_{KE} \leq 1 \\ x_{EL} + x_{LE} \leq 1 \\ x_{ER} + x_{RE} \leq 1 \\ x_{FG} + x_{GF} \leq 1 \\ x_{FPC} + x_{PCF} \leq 1 \\ x_{FK} + x_{KF} \leq 1 \\ x_{FL} + x_{LF} \leq 1 \\ x_{FR} + x_{RF} \leq 1 \\ x_{GPC} + x_{PCG} \leq 1 \\ x_{GK} + x_{KG} \leq 1 \\ x_{GL} + x_{LG} \leq 1 \\ x_{GR} + x_{RG} \leq 1 \\ x_{PCK} + x_{KPC} \leq 1 \\ x_{PCL} + x_{LPC} \leq 1 \\ x_{PCR} + x_{RPC} \leq 1 \\ x_{KL} + x_{LK} \leq 1 \\ x_{KR} + x_{RK} \leq 1 \\ x_{LR} + x_{RL} \leq 1 \end{cases}$$

```
> library(lpSolve)
> library(linprog)
> obj.coef<-c(2.236068,2.236068,5.100191,3.162278,3.162278,
5.099020,2.236068,4,2.945242,2.236068,3.605551,5,2.236068,4,
6.381571,3.605551,2.236068,4.123106,5.100191,2.945242,6.381571,
3.010308,5.008688,5.334975,3.162278,2.236068,3.605551,3.010308,
2,2.828427,3.162278,3.605551,2.236068,5.008688,2,2,5.099020,5,
4.123106,5.334975,2.828427,2)
> a<-c(0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,
0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0
,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,
0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0
,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0
,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,
0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1
,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1
,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1
,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
,0,0,0,0,0,1)
> const.coef<-matrix(a,nrow=35,byrow=T)
> strings<-c("=","=","=","=","=","=","=","=","=","=","=","=","=",
"=","=","<=","<=","<=","<=","<=","<=","<=","<=","<=","<=","<=",
"<=","<=","<=","<=","<=","<=","<=","<=","<=","<=")
> b.coef<- c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1)
> lpSolve1<- lp(direction="min",obj.coef,const.coef,strings, b.coef)
> lpSolve1$solution

[1] 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

[41] 1 0
```

Con lo que me está diciendo que los caminos a coger son:

$$x_{EG}, x_{FE}, x_{GL}, x_{PCF}, x_{KPC}, x_{LR}, x_{RK}$$



Para el cluster 3 (C3) que esta formado por las paradas PC, P, Q, U y V:

```
> x=c(6.99365,7,9,6,10)
> y=c(4.15625,6,6,8,8)
> paradas=c("PC","P","Q","U","V")
> xy.data=data.frame(x,y,row.names=paradas)
> print(xy.data)
          x       y
PC  6.99365 4.15625
P   7.00000 6.00000
Q   9.00000 6.00000
U   6.00000 8.00000
V  10.00000 8.00000
> plot(xy.data)
> text(xy.data,labels=paradas,pos=4)
> xy.m1=dist(xy.data,"euclidean",TRUE,TRUE)
> print(xy.m1)
          PC        P        Q        U        V
PC 0.000000 1.843761 2.724859 3.970108 4.879811
P  1.843761 0.000000 2.000000 2.236068 3.605551
Q  2.724859 2.000000 0.000000 3.605551 2.236068
U  3.970108 2.236068 3.605551 0.000000 4.000000
V  4.879811 3.605551 2.236068 4.000000 0.000000
```

La función a minimizar seria:

$$\begin{aligned} mín\ Z = {} & 1{,}843761x_{PCP} + 2{,}724859x_{PCQ} + 3{,}970108x_{PCU} + 4{,}879811x_{PCV} + 1{,}843761x_{PPC} \\ & + 2x_{PQ} + 2{,}236068x_{PU} + 3{,}605551x_{PU} + 2{,}724859x_{QPC} + 2x_{QP} + 3{,}605551x_{QU} \\ & + 2{,}236068x_{QV} + 3{,}970108x_{UPC} + 2{,}236068x_{UP} + 3{,}605551x_{UQ} + 4x_{UV} \\ & + 4{,}879811x_{VPC} + 3{,}605551x_{VP} + 2{,}236068x_{VQ} + 4x_{VU} \end{aligned}$$

Restricciones de llegada:

$$\begin{cases} x_{QP} + x_{UP} + x_{VP} + x_{PCP} = 1 \\ x_{PQ} + x_{UQ} + x_{VQ} + x_{PCQ} = 1 \\ x_{PU} + x_{QU} + x_{VU} + x_{PCU} = 1 \\ x_{PV} + x_{QV} + x_{UV} + x_{PCV} = 1 \\ x_{PPC} + x_{QPC} + x_{UPC} + x_{VPC} = 1 \end{cases}$$

Restricciones de salida:

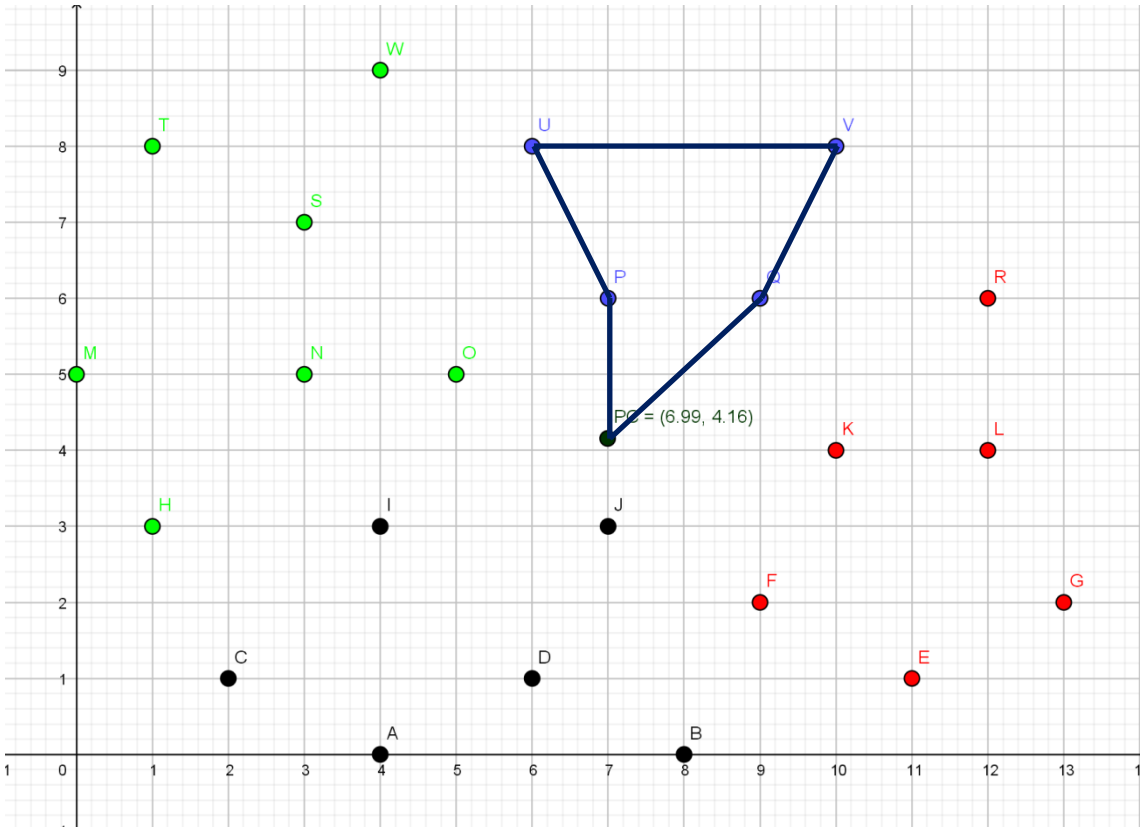
$$\begin{cases} x_{PQ} + x_{PU} + x_{PV} + x_{PPC} = 1 \\ x_{QP} + x_{QU} + x_{QV} + x_{QPC} = 1 \\ x_{UP} + x_{UQ} + x_{UV} + x_{UPC} = 1 \\ x_{VP} + x_{VQ} + x_{VU} + x_{VPC} = 1 \\ x_{PCP} + x_{PCQ} + x_{PCU} + x_{PCV} = 1 \end{cases}$$

Restricciones de ruptura:

$$\begin{cases} x_{PQ} + x_{QP} \leq 1 \\ x_{PU} + x_{UP} \leq 1 \\ x_{PV} + x_{VP} \leq 1 \\ x_{PPC} + x_{PCP} \leq 1 \\ x_{QU} + x_{UQ} \leq 1 \\ x_{QV} + x_{VQ} \leq 1 \\ x_{QPC} + x_{PCQ} \leq 1 \\ x_{UV} + x_{VU} \leq 1 \\ x_{UPC} + x_{PCU} \leq 1 \\ x_{VPC} + x_{PCV} \leq 1 \end{cases}$$

En este caso el camino seria:

$$x_{PPC}, x_{QV}, x_{UP}, x_{VU}, x_{PCQ}$$



Para el cluster 4 (C4) que esta formado por las paradas H, M, N, O, S, T, W y PC:

```
> x=c(1,0,3,5,3,1,4,6.99375)
> y=c(3,5,5,5,7,8,9,4.15625)
> paradas=c("H","M","N","O","S","T","W","PC")
> xy.data=data.frame(x,y,row.names=paradas)
> print(xy.data)
         x       y
H  1.00000 3.00000
M  0.00000 5.00000
N  3.00000 5.00000
O  5.00000 5.00000
S  3.00000 7.00000
T  1.00000 8.00000
W  4.00000 9.00000
PC 6.99375 4.15625
> plot(xy.data)
> text(xy.data,labels=paradas,pos=4)
> xy.m1=dist(xy.data,"euclidean",TRUE,TRUE)
> print(xy.m1)
         H        M        N        O        S        T        W       PC
H  0.000000 2.236068 2.828427 4.472136 4.472136 5.000000 6.708204 6.104257
M  2.236068 0.000000 3.000000 5.000000 3.605551 3.162278 5.656854 7.044463
N  2.828427 3.000000 0.000000 2.000000 2.000000 3.605551 4.123106 4.081906
O  4.472136 5.000000 2.000000 0.000000 2.828427 5.000000 4.123106 2.164937
```

```
S  4.472136 3.605551 2.000000 2.828427 0.000000 2.236068 2.236068 4.902750
T  5.000000 3.162278 3.605551 5.000000 2.236068 0.000000 3.162278 7.120355
W  6.708204 5.656854 4.123106 4.123106 2.236068 3.162278 0.000000 5.694247
PC 6.104257 7.044463 4.081906 2.164937 4.902750 7.120355 5.694247 0.000000
```

La función a minimizar será:

$$\begin{aligned} mín\ Z = {} & 2{,}236068x_{HM} + 2{,}828427x_{HN} + 4{,}472136x_{HO} + 4{,}472136x_{HS} + 5x_{HT} \\ & + 6{,}708204x_{HW} + 6{,}104257x_{HPC} + 2{,}236068x_{MH} + 3x_{MN} + 5x_{MO} \\ & + 3{,}605551x_{MS} + 3{,}162278x_{MT} + 5{,}656854x_{MW} + 7{,}044463x_{MPC} \\ & + 2{,}828427x_{NH} + 3x_{NM} + 2x_{NO} + 2x_{NS} + 3{,}605551x_{NT} + 4{,}123106x_{NW} \\ & + 4{,}081906x_{NPC} + 4{,}472136x_{OH} + 5x_{OM} + 2x_{ON} + 2{,}828427x_{OS} + 5x_{OT} \\ & + 4{,}123106x_{OW} + 2{,}164937x_{OPC} + 4{,}472136x_{SH} + 3{,}605551x_{SM} + 2x_{SN} \\ & + 2{,}828427x_{SO} + 2{,}236068x_{ST} + 2{,}236068x_{SW} + 4{,}902750x_{SPC} + 5x_{TH} \\ & + 3{,}162278x_{TM} + 3{,}605551x_{TN} + 5x_{TO} + 2{,}236068x_{TS} + 3{,}162278x_{TW} \\ & + 7{,}120355x_{TPC} + 6{,}708204x_{WH} + 5{,}656854x_{WM} + 4{,}123106x_{WN} \\ & + 4{,}123106x_{WO} + 2{,}236068x_{WS} + 3{,}162278x_{WT} + 5{,}694247x_{WPC} \\ & + 6{,}104257x_{PCH} + 7{,}044463x_{PCM} + 4{,}081906x_{PCN} + 2{,}164937x_{PCO} \\ & + 4{,}902750x_{PCS} + 7{,}120355x_{PCT} + 5{,}694247x_{PCW} \end{aligned}$$

Restricciones de llegada:

$$\begin{cases} x_{MH} + x_{NH} + x_{OH} + x_{SH} + x_{TH} + x_{WH} + x_{PCH} = 1 \\ x_{HM} + x_{NM} + x_{OM} + x_{SM} + x_{TM} + x_{WM} + x_{PCM} = 1 \\ x_{HN} + x_{MN} + x_{ON} + x_{SN} + x_{TN} + x_{WN} + x_{PCN} = 1 \\ x_{HO} + x_{MO} + x_{NO} + x_{SO} + x_{TO} + x_{WO} + x_{PCO} = 1 \\ x_{HS} + x_{MS} + x_{NS} + x_{OS} + x_{TS} + x_{WS} + x_{PCS} = 1 \\ x_{HT} + x_{MT} + x_{NT} + x_{OT} + x_{ST} + x_{WT} + x_{PCT} = 1 \\ x_{HW} + x_{MW} + x_{NW} + x_{OW} + x_{SW} + x_{TW} + x_{PCW} = 1 \\ x_{HPC} + x_{MPC} + x_{NPC} + x_{OPC} + x_{SPC} + x_{TPC} + x_{WPC} = 1 \end{cases}$$

Restricciones de salida:

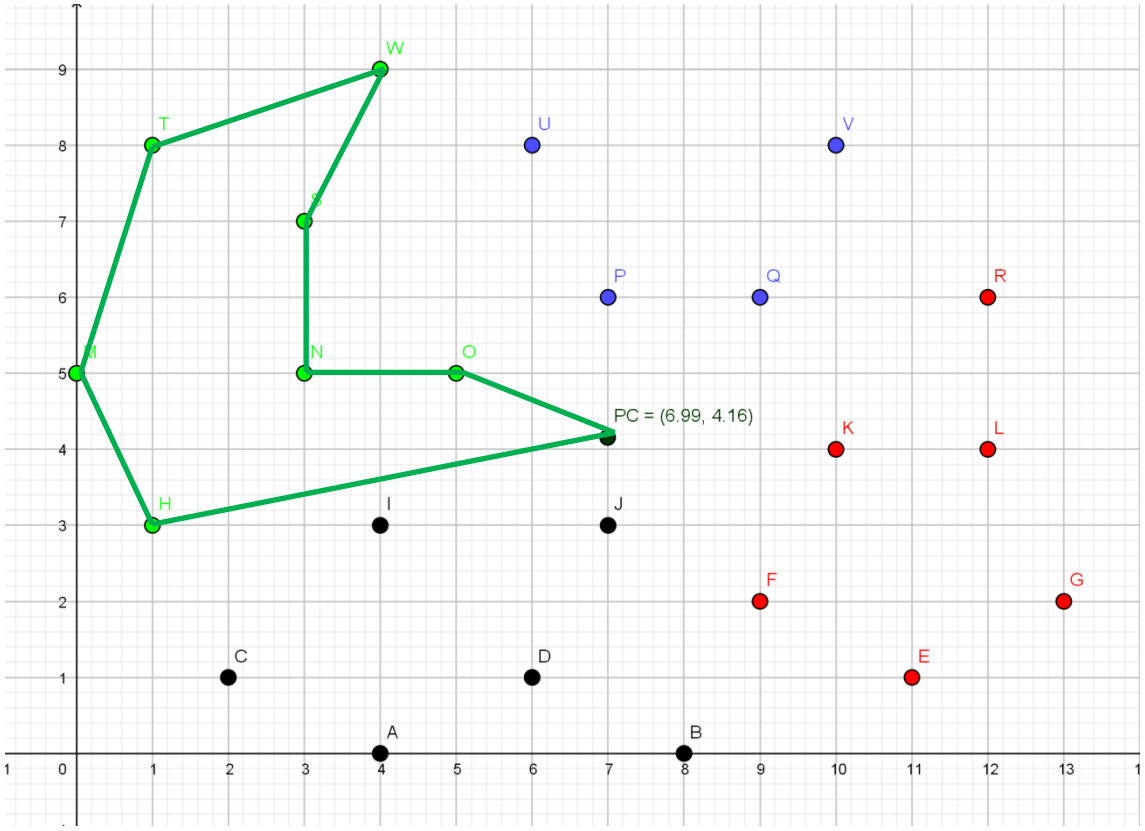
$$\begin{cases} x_{HM} + x_{HN} + x_{HO} + x_{HS} + x_{HT} + x_{HW} + x_{HPC} = 1 \\ x_{MH} + x_{MN} + x_{MO} + x_{MS} + x_{MT} + x_{MW} + x_{MPC} = 1 \\ x_{NH} + x_{NM} + x_{NO} + x_{NS} + x_{NT} + x_{NW} + x_{NPC} = 1 \\ x_{0H} + x_{OM} + x_{ON} + x_{OS} + x_{OT} + x_{OW} + x_{OPC} = 1 \\ x_{SH} + x_{SM} + x_{SN} + x_{SO} + x_{ST} + x_{SW} + x_{SPC} = 1 \\ x_{TH} + x_{TM} + x_{TN} + x_{TO} + x_{TS} + x_{TW} + x_{TPC} = 1 \\ x_{WH} + x_{WM} + x_{WN} + x_{WO} + x_{WS} + x_{WT} + x_{WPC} = 1 \\ x_{PCH} + x_{PCM} + x_{PCN} + x_{PCO} + x_{PCS} + x_{PCT} + x_{PCW} = 1 \end{cases}$$

Restricciones de ruptura:

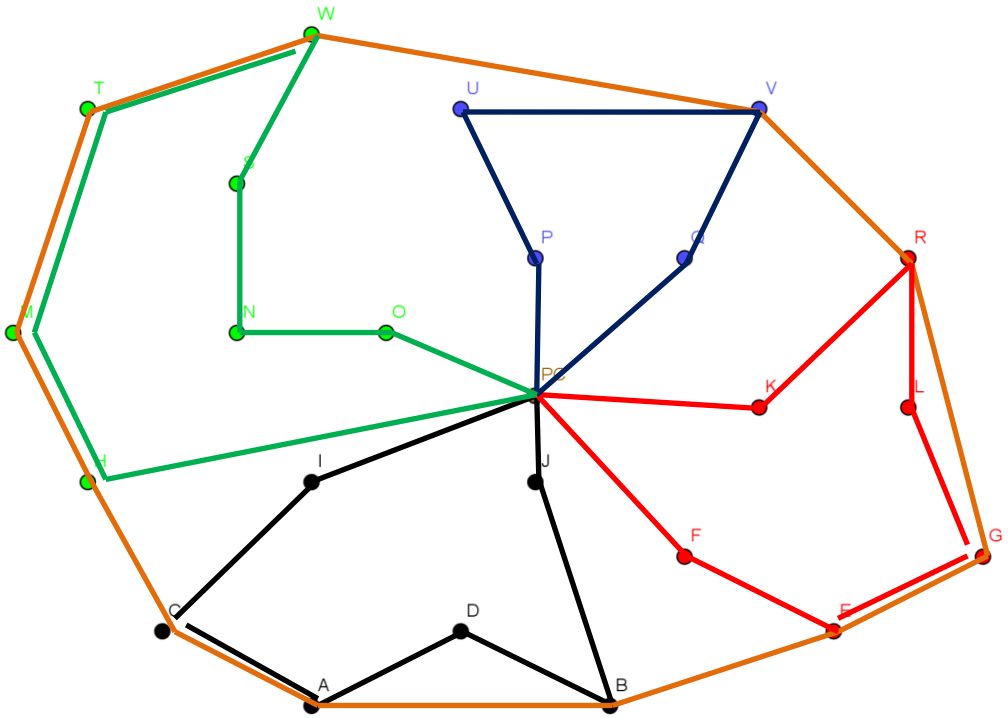
$$\begin{cases} x_{HM} + x_{MH} \leq 1 \\ x_{HN} + x_{NH} \leq 1 \\ x_{HO} + x_{OH} \leq 1 \\ x_{HS} + x_{SH} \leq 1 \\ x_{HT} + x_{TH} \leq 1 \\ x_{HW} + x_{WH} \leq 1 \\ x_{HPC} + x_{PCH} \leq 1 \\ x_{MN} + x_{NM} \leq 1 \\ x_{MO} + x_{OM} \leq 1 \\ x_{MS} + x_{SM} \leq 1 \\ x_{MT} + x_{TM} \leq 1 \\ x_{MW} + x_{WM} \leq 1 \\ x_{MPC} + x_{PCM} \leq 1 \\ x_{NO} + x_{ON} \leq 1 \\ x_{NS} + x_{SN} \leq 1 \\ x_{NT} + x_{TN} \leq 1 \\ x_{NW} + x_{WN} \leq 1 \\ x_{NPC} + x_{PCN} \leq 1 \\ x_{OS} + x_{SO} \leq 1 \\ x_{OT} + x_{TO} \leq 1 \\ x_{OW} + x_{WO} \leq 1 \\ x_{OPC} + x_{PCO} \leq 1 \\ x_{WPC} + x_{PCW} \leq 1 \end{cases}$$

Haciendo un estudio análogo a los anteriores nos sale como caminos:

$$x_{PCO}, x_{ON}, x_{NS}, x_{SW}, x_{WT}, x_{TM}, x_{MH}, x_{HPC}$$



Si ponemos las 5 rutas juntas quedaría de la siguiente forma:



CIRCULAR: A – B – E – G – R – V – W – T – M – H – C – A

RUTA 1: PC – I – C – A – D – B – J – PC

RUTA 2: PC – F – E – G – L – R – K – PC

RUTA 3: PC – Q – V – U – P – PC

RUTA 4: PC – O – N – S – W – T – M – H – PC

Bibliografía

[1] **Miguel A. Goberna.** Medio siglo de Programación Lineal. En: Jornadas de Educación Matemática de la Comunidad Valenciana 2000.

[2] **Eugene Schenkman.** The Weierstrass approximation theorem. American Mathematical Mobthly, pages 65-66. 1972

[3] **Goberna, M.A., Jornet, V., Puente, R.** Optimización lineal. Teoría, métodos y modelos. Addison Wesley, 2004.

[4] **Luenberger, D.** Programación Lineal y no Lineal. Addison Wesley, 1973.

[5] **Stefan TheuBl, Florian Schwendinger, Kurt Hornik.** ROI: An Extensible R Optimization Infrastructure. 2019.

[6] **Enrique Castillo, Antonio J. Conejo, Pablo Pedregal, Ricardo García y Natalia Alguacil.** Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia. 2002.

[7] **Enrique Baquela y Andrés Redchuk.** Optimización Matemática con R. Volumen I: Introducción al modelado y resolución de problemas. 2013

[8] **H. Calvete and P. Mateo.** Programación lineal, entera y meta. Colección Textos Docentes. Prensa Universitaria de Zaragoza, 1994.

[9] **J. Dennis and R. Schnable,** Numerical methods for unconstrained optimization ans nonlinear equations. Prentice-Hall, Inc., New Jersey, 1983.

[10] **M. Berkelaar, J. Dirks, K. Eikland, P. Notebaert, y J. Ebert,** *lp_solve Reference Guide*, Eindhoven University of Technology, 2011, Web, http://lpsolve.sourceforge.net/5.0/.

[11] **A. Makhorin,** *GLPK (GNU Linear Programming Kit)*, Department for Applied Informatics, Moscow Aviation Institute, 2011, Web, http://www.gnu.org/software/glpk/.

[12] **C. Mészáros,** *The BPMPD interior point solver for convex quadratic problems*, Tech. Report WP-98-8, Laboratory of Operations Research and Decision Systems, Budapest, Hungría, 1998, http://www.sztaki.hu/~meszaros/bpmpd/.

[13] **B. A. Murtagh y M. A. Saunders,** *MINOS 5.5 User Guide*, Tech. Report SOL 83-20R, Systems Optimization Laboratory, Dep. of Operations Research, Stanford University, dic 1983, http://www.sbsi-sol-optimize.com/manuals/Minos%20Manual.pdf.

[14] **IBM Corporation, Software Group,** *IBM ILOG CPLEX Optimizer, High Performance Mathematical Optimization Engines*, 2010, Web, http://public.dhe.ibm.com/common/ssi/ecm/en/wsd14044usen/WSD14044USEN.PDF.

[15] **MOSEK ApS,** *The MOSEK optimization toolbox for MATLAB manual. Version 6.0 (Revision 114)*, 2011, Web, http://www.mosek.com/fileadmin/products/6_0/tools/doc/html/toolbox/index.html.

[16] **J. F. Sturm,** *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Advances Optimization Lab, Mc-Master University, 1998, incluido en la descarga del *software*.

[17] **K. C. Toh, C. R. H. Tütüncü, y M. J. Todd,** *On the Implementation and Usage of SDPT3 - a Matlab Software Package for Semidefinite-Quadratic-Linear Programming, version 4.0*, jul 2006.

[18] **S. Boyd y L. Vandenberghe,** *Convex Optimization*, Cambridge University Press, mar 2004.

[19] **M. Grant y S. Boyd, *CVX,*** *Matlab Software for Disciplined Convex Programming, version 1.21*, feb 2011, disponible online, http://cvxr.com/cvx.

[20] AMPL Optimization LLC, *Key AMPL Features*, 2011, Web, http://www.ampl.com/key.html.

[21] GAMS Development Corporation, *An Introduction to GAMS*, 2011, Web, http://www.gams.com/docs/intro.html.

[22] **Becker, R.A., Chambers, J.M. y Wilks, A.R.** (1988). The new S language: A programming environment for data analysis and graphics. Pacific Grove, CA: Wadsworth.

[23] **Chambers, J.M.** (1998). Programming with data: a guide to the Language (2007). Software for data analysis: Programming with R. New York: Springer.

[24] **Venables, W. N., Smith, D. M. y the RDevelopment Core Team** (2007). *An Introduction to R.* Vienna, Austria: R Foundation for Statistical Computing.

[25] **Berkelaar, M.** *et al* (2014) *lpSolve: Interface to Lpsolve v. 5.5 to solve linear-integer programs*. R package version 5.6.10.

[26] **Y. Ye, M. J. Todd, y S. Mizuno,** *An O($\sqrt{nL}$)-iteration homogeneous and self-dual linear programming algorithm*, Mathematics of Operations Research **19** (1994), 53–67.

[27] **Zitzler, E., Laummans, M. & Thiele, L.** (May, 2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. En Computer Engineering and Networks Laboratory (TIK) Report.

[28] **L.A. Zadeh.** Fuzzy Sets, Inform. Control 8 (1965) 338-353

[29] **S.S.L. Chang, L.A. Zadeh.** On fuzzy mappings and control, IEEE Transactions on Systems, Man and Cybernetics 2 (1) (1972) 30-34.

[30] **Bezdek, James C.; Ehrlich, Robert; Full, William** (1984). FCM: The Fuzzy c-Means Clustering Algorithm. Computers & Geosciences 10 (2-3): 191-203.

[31] Yang, M. S. (1993). A Survey of Fuzzy Clustering. Mathl. Comput. Modelling 18 (11): 1-16. Archivado desde el original el 15 de diciembre de 2013. Consultado el 8 de marzo de 2020.

[32] W. Karush (1939). Minima of Functions of Several Variables with Inequalities as Side Constraints. M.Sc. Dissertation. Dept. of Mathematics, Univ. Of Chicago, Chicago, Illinois.

[33] H. W. Kuhn, Tucker, A. W., Proceedings of 2nd Berkeley Symposium, Nonlinear Programming, University of California Press, 1951, Berkeley

[34] Christos H. Papadimitriou y Kenneth Steiglitz. “Combinatorial Optimization: Algorithms and Complexity”. General Publishing Company (1998).