



UNIVERSIDAD DE ALMERÍA

ESCUELA POLITÉCNICA SUPERIOR Y FACULTAD DE CIENCIAS EXPERIMENTALES

ESTUDIO DE ALGORITMOS  
MATEMÁTICOS IMPLEMENTADOS EN  
LIBRERÍAS DE CÓDIGO ABIERTO:  
MÉTODO DEL SIMPLEX.

TRABAJO FIN DE GRADO PRESENTADO POR  
ESTEFANÍA GÁLVEZ SÁNCHEZ

DIRECTOR Y CODIRECTORA  
JULIO BARÓN MARTÍNEZ Y MERCEDES MARTÍNEZ DURBÁN

2014

GRADO EN MATEMÁTICAS

DEPARTAMENTO DE INFORMÁTICA

---

# Agradecimientos

Este trabajo no habría sido posible sin el apoyo y el estímulo de mi compañera y amiga, Alba Cánovas Pérez, juntas desde el primer día del comienzo del grado hasta el último haciendo nuestro trabajo fin de grado de una misma rama que nos gusta a ambas. Julio Barón Martínez y Mercedes Martínez Durbán, mi tutor y cotutora respectivamente, también han sido generosamente serviciales y me han ayudado de numerosos modos, incluyendo el permitirme asistir a sus clases de programación para así adquirir los suficientes conocimientos de **Java** para poder desarrollar este trabajo.

No puedo terminar sin agradecer a mi familia, cuyo apoyo y amor he tenido a lo largo de estos cuatro años.

---

# Índice general

<b>Agradecimientos</b>	<b>2</b>
<b>1. Introducción y objetivos</b>	<b>5</b>
<b>2. Antecedentes Bibliográficos</b>	<b>6</b>
2.1. Definiciones y teoremas previos . . . . .	6
2.2. Método del Simplex . . . . .	10
2.2.1. Inicialización . . . . .	12
2.2.2. Prueba de optimalidad . . . . .	12
2.2.3. Selección de variable que entra en la base . . . . .	13
2.2.4. Selección de variable que abandona la base . . . . .	13
2.3. Método de las dos fases . . . . .	18
2.4. Instalación de la librería <i>Apache Commons Math</i> . . . . .	21
2.4.1. Introducción a Apache Commons . . . . .	22
2.4.2. Commons Math: The Apache Commons Mathematics Library . . . . .	22
<b>3. Resultados y Discusión</b>	<b>27</b>
3.1. Resultados . . . . .	27
3.1.1. Problema del Transporte . . . . .	27
3.1.2. Problema de la Dieta . . . . .	32
3.2. Discusión . . . . .	35
<b>4. Conclusiones</b>	<b>38</b>
<b>Bibliografía</b>	<b>39</b>
<b>Apéndices</b>	<b>40</b>
<b>A. Programación en Java y entorno Eclipse</b>	<b>41</b>
A.1. Programación en Java . . . . .	41
A.1.1. Origen del lenguaje de programación en Java . . . . .	41
A.1.2. Características de Java . . . . .	41
A.1.3. El entorno de desarrollo integrado Eclipse . . . . .	42
A.1.4. Subversion . . . . .	45

---

# Índice de figuras

2.1.	Diagrama de flujo del Algoritmo del Simplex . . . . .	16
2.2.	Descargas de librerías . . . . .	23
2.3.	Archivos .jar de la librería en carpeta lib . . . . .	24
2.4.	Propiedades del ProyectoJava . . . . .	24
2.5.	Incluyendo las fuentes . . . . .	25
2.6.	Incluyendo Javadoc . . . . .	26
3.1.	Diagrama del ejemplo del Problema del Transporte . . . . .	28
3.2.	Solución del ejemplo del Problema del Transporte . . . . .	31
3.3.	Solución del ejemplo del Problema de la Dieta . . . . .	34
3.4.	Paquete <code>org.apache.commons.math3.optim.linear</code> . . . . .	35
3.5.	Descripción del paquete <code>org.apache.commons.math3.optim.linear</code> . . . . .	36
3.6.	Diagrama de clases UML . . . . .	36
3.7.	Otros paquetes de Apache Commons Math Optim . . . . .	37
A.1.	Pantalla inicial de <i>Eclipse</i> . . . . .	42
A.2.	Perspectiva inicial orientada al desarrollo de proyectos Java2 SE. . . . .	43
A.3.	Perspectivas . . . . .	44
A.4.	Perspectiva Depurador . . . . .	44
A.5.	Nuevo Repositorio . . . . .	45

---

# Capítulo 1

## Introducción y objetivos

Se llama Programación Lineal al conjunto de técnicas matemáticas que pretenden resolver el siguiente tipo de problemas: optimizar (maximizar o minimizar) una función objetivo, función lineal de varias variables, sujeta a unas condiciones definidas por inecuaciones lineales.

En la actualidad uno de los problemas que más interesa resolver, a nivel práctico, es la optimización de recursos, tiempo, personas, rutas de viaje, asignación de tareas, dietas. Por ello, he elegido este tema para mi trabajo fin de grado.

En el año 1947 George Dantzig, físico y matemático estadounidense, presentó el algoritmo Simplex. A partir de él, se logró resolver problemas de optimización que llevaban más de un siglo sin solución. A su vez, el desarrollo de la computación digital facilitó su rápido desarrollo y aplicación empresarial en todo tipo de problemas.

La gran virtud del Método del Simplex es su sencillez, ya que solo trabaja con los coeficientes de la función objetivo y de las restricciones, además, gracias a su existencia, se pueden resolver problemas complejos. Este método conforma la base de la programación lineal.

En este trabajo, presentaremos en primer lugar una teoría básica con la que entenderemos el algoritmo del Método del Simplex. En segundo lugar, veremos cómo resolver problemas de optimización lineal mediante el Método Simplex, con programas basados en la librería de programación *Apache Commons Math*.

Así pues, nuestros objetivos pueden resumirse en los siguientes puntos:

1. Introducción a la resolución de problemas de programación lineal mediante el Método del Simplex.
2. Utilizar librerías de Java para resolver problemas de programación lineal mediante el Método del Simplex.
3. Comparar la resolución de dichos problemas con el programa Mathematica.
4. Estudiar cuál de estas dos alternativas es más adecuada para la resolución de este tipo de ejercicios.

---

# Capítulo 2

## Antecedentes Bibliográficos

Vamos a dedicar este capítulo a introducir resultados y definiciones para entender cómo funciona el Método del Simplex. Los conocimientos que no se explican y se dan por sabidos son muy básicos, de todos modos puede recurrirse a los siguientes libros y apuntes [1], [2], [3] y [4]. También veremos cómo se instalan las librerías que necesitamos en el entorno de desarrollo software **Eclipse** [5].

### 2.1. Definiciones y teoremas previos

#### Definición 1.

Un segmento lineal dados dos puntos  $\mathbf{x}_1, \mathbf{x}_2$  es el conjunto de puntos  $\mathbf{z}$  que verifica  $\mathbf{z} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$  con  $\lambda \in (0, 1)$ .

#### Definición 2.

Un conjunto  $C$  es convexo si para cualesquiera dos puntos  $\mathbf{x}_1, \mathbf{x}_2 \in C$  se verifica que  $\mathbf{z} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$  pertenece a  $C$  para todo  $\lambda \in (0, 1)$ .

#### Definición 3.

Dado un conjunto convexo  $C$ , un punto  $\mathbf{z} \in C$  se dice que es un *punto extremo* de  $C$  si  $\mathbf{z}$  no puede expresarse como combinación lineal convexa de ningún par de puntos distintos del conjunto. Es decir, si  $\mathbf{z}$  es un punto extremo de  $C$  y  $\mathbf{z} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$ , para un cierto  $\lambda \in (0, 1)$ , entonces  $\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{z}$ .

#### Definición 4.

Dado un conjunto convexo cerrado  $S \subseteq \mathbb{R}^n$ , un vector no nulo  $\mathbf{d} \in \mathbb{R}^n$  se llama *dirección* si  $\mathbf{x} + \lambda\mathbf{d} \in S, \forall \lambda \geq 0$  y  $\forall \mathbf{x} \in S$ .

*Observaciones:*

Dos direcciones  $\mathbf{d}_1$  y  $\mathbf{d}_2$  son distintas si  $\mathbf{d}_1 \neq \alpha \mathbf{d}_2, \forall \alpha \geq 0$ .

Una dirección  $\mathbf{d}$  se dice *extrema* si no puede ponerse como una combinación lineal positiva de dos direcciones distintas, esto es, si  $\mathbf{d}$  es una dirección extrema y  $\mathbf{d} = \lambda_1 \mathbf{d}_1 + \lambda_2 \mathbf{d}_2$  con  $\lambda_1, \lambda_2 > 0$  entonces  $\mathbf{d}_1 = \alpha \mathbf{d}_2$  para algún  $\alpha > 0$ .

**Teorema 1.**

Sea  $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  donde  $A$  es una matriz  $m \times n$  de rango  $m$  y  $\mathbf{b}$  es un  $m$  vector. Un punto  $\mathbf{x}$  es un punto extremo de  $S$  si, y sólo si,  $A$  puede descomponerse, tras reordenar sus columnas, en la forma  $A = [B, N]$  cumpliendo que:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} B^{-1}\mathbf{b} \\ \mathbf{0} \end{pmatrix}$$

Donde  $B$  es una  $m \times m$  matriz inversible cumpliendo  $B^{-1}\mathbf{b} \geq \mathbf{0}$ .

*Demostración.* Supongamos que  $A$  puede descomponerse en la forma  $A = [B, N]$  cumpliendo que  $\mathbf{x} = [B^{-1}\mathbf{b}, \mathbf{0}]'$  y  $B^{-1}\mathbf{b} \geq \mathbf{0}$ . Obviamente  $\mathbf{x} \in S$ , ahora supongamos que  $\mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$ , con  $\mathbf{x}_1$  y  $\mathbf{x}_2 \in S$  y  $\lambda \in (0, 1)$ , expresamos estos puntos en función de la factorización anterior

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} B^{-1}\mathbf{b} \\ \mathbf{0} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{x}_{1B} \\ \mathbf{x}_{1N} \end{pmatrix} + (1 - \lambda) \begin{pmatrix} \mathbf{x}_{2B} \\ \mathbf{x}_{2N} \end{pmatrix}$$

como  $\mathbf{x}_{1N}$  y  $\mathbf{x}_{2N}$  son mayores o iguales que cero y  $\lambda \in (0, 1)$  entonces  $\mathbf{x}_{1N} = \mathbf{x}_{2N} = \mathbf{0}$  y esto implica que  $\mathbf{x}_{1B} = \mathbf{x}_{2B} = B^{-1}\mathbf{b}$ , ya que  $A\mathbf{x}_1 = \mathbf{b} = A(\mathbf{x}_{1B}, \mathbf{x}_{1N}) = B\mathbf{x}_{1B} + N\mathbf{x}_{1N} = B\mathbf{x}_{1B} + \mathbf{0}$ . Por lo tanto  $\mathbf{x} = \mathbf{x}_1 = \mathbf{x}_2$ .

Supongamos ahora que  $\mathbf{x}$  es un punto extremo de  $S$ . Sin pérdida de generalidad supongamos que  $\mathbf{x} = (x_1, x_2, \dots, x_k, 0, 0, \dots, 0)'$  con  $x_j > 0, j = 1, \dots, k$ . Demostraremos primero que las columnas correspondientes  $A_1, A_2, \dots, A_k$  de  $A$  son linealmente independientes, supongamos para ello que no fuera así, entonces existirían  $\lambda_1, \lambda_2, \dots, \lambda_k$  escalares no todos nulos tales que  $\sum_{j=1}^k \lambda_j A_j = \mathbf{0}$ , sea  $\Delta = (\lambda_1, \lambda_2, \dots, \lambda_k, 0, \dots, 0)'$ . Construimos los siguientes dos vectores  $\mathbf{x}_1 = \mathbf{x} + \alpha \Delta$  y  $\mathbf{x}_2 = \mathbf{x} - \alpha \Delta$  donde  $\alpha$  se elige de manera que  $\mathbf{x}_1$  y  $\mathbf{x}_2$  se mantengan mayores o iguales que cero.

$$A\mathbf{x}_1 = \sum_{j=1}^k (x_j + \alpha \lambda_j) A_j = \sum_{j=1}^k x_j A_j + \alpha \sum_{j=1}^k \lambda_j A_j = \mathbf{b}$$

y similarmente  $A\mathbf{x}_2 = \mathbf{b}$ . Resumiendo  $\mathbf{x}_1, \mathbf{x}_2 \in S, \mathbf{x}_1 \neq \mathbf{x}_2$  ya que  $\alpha > 0$  y además  $\mathbf{x} = \frac{1}{2}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_2$  y esto contradice el hecho de que  $\mathbf{x}$  sea un punto extremo. Por lo tanto  $A_1, A_2, \dots, A_k$  son linealmente independientes y pueden elegirse  $m - k$  columnas del resto de  $n - k$  que sean linealmente independientes con estas  $k$  ya que el rango de  $A$  es  $m$ .

Por simplificar notación supongamos que éstas son  $A_{k+1}, A_{k+2}, \dots, A_m$ . Con esto podemos escribir  $A = [B, N]$  donde  $B = [A_1, \dots, A_k, A_{k+1}, A_{k+2}, \dots, A_m]$ , además  $B^{-1}\mathbf{b} =$

$(x_1, x_2, \dots, x_k, 0, \dots, 0)'$  y como  $(x_1, x_2, \dots, x_k) > 0$  entonces  $B^{-1}\mathbf{b} \geq \mathbf{0}$  y la demostración del teorema queda completa.  $\square$

**Corolario 1.** *El número de puntos extremos de  $S$  es finito.*

*Demostración.* Cualquier combinación lineal no negativa de  $m$  vectores linealmente de  $A$ , corresponde con un punto extremo de  $S$ , como solo podemos tomar un número finito de combinaciones de  $m$  vectores de entre los  $n$  de la matriz  $A$ , concluimos que el número de puntos extremos de  $S$  es finito, además es

$$C(n, m) = \binom{n}{m} = \frac{n!}{m!(n-m)!}$$

$\square$

**Teorema 2.**

Sea  $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  no vacío, donde  $A$  es una matriz  $m \times n$  de rango completo  $m$  y  $\mathbf{b}$  es un vector de dimensión  $m$ , entonces  $S$  tiene al menos un punto extremo.

**Teorema 3.**

Sea  $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  no vacío, donde  $A$  es una matriz  $m \times n$  de rango completo  $m$  y  $\mathbf{b}$  es un vector de dimensión  $m$ . Un vector  $\bar{\mathbf{d}}$  es una dirección extrema de  $S$  si, y sólo si,  $A$  puede descomponerse en  $[B, N]$  tal que  $B^{-1}A_j \leq \mathbf{0}$  para alguna columna  $A_j$  de  $N$  y  $\bar{\mathbf{d}}$  es un múltiplo positivo de  $\mathbf{d} = [-B^{-1}A_j, \mathbf{e}_j]$ , donde  $\mathbf{e}_j$  es un  $n - m$  vector de ceros excepto la posición  $j$  donde tiene un 1.

La demostración de los teoremas anteriores podemos encontrarlas en los apuntes de *Fundamentos matemáticos del método simplex* [6].

**Corolario 2.** *El número de direcciones extremas de  $S$  es finito.*

*Demostración.* Hay a lo más  $\binom{n}{m}$  formas de elegir  $B^{-1}$  y como hay  $n - m$  columnas en  $N$ , entonces  $(n - m) \binom{n}{m}$  es el número máximo de direcciones extremas.  $\square$

Veamos ahora un resultado importante que nos permite representar cualquier punto de un poliedro como sumas de puntos extremos y direcciones extremas.



**Teorema 4** (Teorema de representación).

Sea  $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  un poliedro no vacío donde  $A$  es una matriz  $m \times n$  de rango  $m$ . Sean  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  los puntos extremos de  $S$  y  $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_l$  las direcciones extremas de  $S$ . Entonces  $\mathbf{x} \in S$  si, y sólo si,

$$\mathbf{x} = \sum_{i=1}^k \lambda_i \mathbf{x}_i + \sum_{j=1}^l \mu_j \mathbf{d}_j \quad (2.1)$$

con  $\sum_{i=1}^k \lambda_i = 1, \lambda_i \geq 0 \quad \forall i = 1, \dots, k$  y  $\mu_j \geq 0 \quad \forall j = 1, \dots, l$ .

La demostración de este teorema podemos encontrarla en los apuntes de *Fundamentos matemáticos del método simplex* [6].

**Corolario 3.** Sea  $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  un poliedro no vacío donde  $A$  es una matriz  $m \times n$  de rango  $m$ . Entonces  $S$  tiene al menos una dirección extrema si, y sólo si, es no acotado.

*Demostración.* ( $\Rightarrow$ ) Supongamos que  $S$  tiene una dirección  $d$ , entonces es no acotado puesto que dado  $x \in S$  se tiene que  $x + \lambda d \in S, \forall \lambda \geq 0$  y por lo tanto  $\lim_{\lambda \rightarrow \infty} \|x + \lambda d\| = \infty$ . ( $\Leftarrow$ ) Supongamos que  $S$  es no acotado y que no posee direcciones. Entonces tampoco posee direcciones extremas y, por el teorema anterior, todo punto  $x \in S$  puede escribirse de la forma  $x = \sum_{i=1}^k \lambda_i x_i$ , para algunos  $\lambda_i \geq 0, i = 1, \dots, k, \sum_{i=1}^k \lambda_i = 1$ .

Por la desigualdad triangular

$$\|x\| = \left\| \sum_{i=1}^k \lambda_i x_i \right\| \leq \sum_{i=1}^k \lambda_i \|x_i\| \leq \sum_{i=1}^k \|x_i\| < \infty \quad \forall x \in S$$

lo que contradice que  $S$  sea no acotado. □

**Teorema 5** (Condiciones de optimalidad en programación lineal).

Considerar el P.P.L (Problema de Programación Lineal)  $\max Z = \mathbf{c}\mathbf{x}, s.a : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ , suponer que la región factible es no vacía, sean  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  los puntos extremos de  $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  y  $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_l$  sus direcciones extremas. Una condición necesaria y suficiente para que exista una solución óptima finita del problema es que  $\mathbf{c}\mathbf{d}^j \leq \mathbf{0}, j = 1, \dots, l$ . En este caso existe un punto extremo que es solución óptima del problema.

*Demostración.* Por el teorema de representación,  $A\mathbf{x} = \mathbf{b}$  y  $\mathbf{x} \geq \mathbf{0}$  es equivalente a que  $\mathbf{x}$  se pueda expresar como

$$\mathbf{x} = \sum_{i=1}^k \lambda_i \mathbf{x}_i + \sum_{j=1}^l \mu_j \mathbf{d}_j$$

con  $\sum_{i=1}^k \lambda_i = 1, \lambda_i \geq 0 \forall i = 1, \dots, k$  y  $\mu_j \geq 0, \forall j = 1, \dots, l$ .

Por tanto el P.P.L puede ponerse como:

$$\text{máx } Z = \mathbf{c} \left( \sum_{j=1}^k \lambda_j \mathbf{x}^j + \sum_{j=1}^l \mu_j \mathbf{d}^j \right) \quad (2.2)$$

$$s.a : \quad \sum_{j=1}^k \lambda_j = 1 \quad (2.3)$$

$$\lambda_j \geq 0, j = 1, \dots, k \quad (2.4)$$

$$\mu_j \geq 0, j = 1, \dots, l$$

Observamos que si  $\mathbf{cd}^j > 0$  para algún  $j$ , entonces  $\mu_j$  puede elegirse arbitrariamente grande, proporcionando una solución no acotada. Esto demuestra que una condición necesaria y suficiente para que haya una solución óptima finita es  $\mathbf{cd}^j \leq 0, j = 1, \dots, l$ .

Supongamos que  $\mathbf{cd}^j \leq 0, j = 1, \dots, l$  como queremos maximizar la función objetivo elegimos  $\mu_j = 0 \forall j$  con lo que el problema se reduce a maximizar  $c(\sum_{j=1}^k \lambda_j \mathbf{x}^j)$  sujeto a  $\sum_{j=1}^k \lambda_j = 1$  y  $\lambda_j \geq 0, j = 1, \dots, k$ . Está claro que la solución óptima de este problema es finita y se encuentra haciendo  $\lambda_i = 1$  y  $\lambda_j = 0$  para  $j \neq i$ , donde el índice  $i$  viene dado por  $\mathbf{cx}^i = \max_{i \leq j \leq k} \mathbf{cx}^j$ .

Observamos que  $\lambda_i = 1$  y  $\lambda_j = 0, j \neq i$  implica que la solución del problema se alcanza en el punto extremo  $i$ -ésimo, con lo que tenemos demostrado el teorema.  $\square$

## 2.2. Método del Simplex

El Método del Simplex es un procedimiento sistemático para resolver P.P.L. moviéndonos desde un punto extremo a otro con una mejora (o al menos no empeoramiento) de la función objetivo.

El algoritmo se irá moviendo por puntos extremos cumpliendo lo anterior hasta que se alcanza el punto extremo óptimo o hasta que se detecta una dirección extrema de no acotación (con  $\mathbf{cd} < 0$ ).

Consideraremos que el P.P.L. está escrito en la forma estándar cuando tengamos:  $\text{máx } Z = \mathbf{cx}$  s.a :  $\mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ . Suponiendo que al menos existe un punto verificando las restricciones y además el rango de  $\mathbf{A}$  es  $m$ .

A continuación veremos una serie de definiciones referidas a las soluciones posibles de un P.P.L.

### Definición 5.

**Solución factible** es cualquier punto  $\mathbf{x}$  verificando  $\mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ .

**Definición 6.**

**Solución básica, SB** a partir de ahora, es cualquier punto que verifica  $\mathbf{Ax} = \mathbf{b}$ , en el cual al menos  $n - m$  variables toman valor 0.

Observamos que por construcción los puntos extremos son **SB**. Una **SB** puede obtenerse sin más que fijar  $n - m$  variables a cero y resolviendo el sistema resultante.

**Definición 7.**

**Base** es la colección de variables con valor no obligatoriamente nulo en un cierto orden que forman una **SB**. Las variables de ésta se denominan **variables básicas**.

**Definición 8.**

**Solución básica factible, SBF** a partir de ahora, es aquella solución básica que verifica las condiciones de no negatividad. Es decir, una solución básica que además es solución factible.

**Definición 9.**

**SBF no degenerada** es una SBF que tiene exactamente  $m$  valores no nulos y es degenerada en caso contrario.

**Definición 10.**

**Solución óptima, SO** a partir de ahora, es una SBF en la cual se alcanza el óptimo de la función objetivo.

**Definición 11.**

Una SBF  $\mathbf{x}$  es **adyacente** a otra SBF  $\mathbf{y}$  si coinciden todas las variables de la base excepto una.

El Método del Simplex irá calculando SBF adyacentes de manera que la nueva SBF tenga mejor, o al menos no peor, valor de la función objetivo.

El algoritmo consta de las siguientes etapas:

- Inicialización.
- Prueba de optimalidad.
- Paso de una SBF a otra SBF:
  1. Selección de una variable para la nueva SBF.
  2. Selección de una variable de la antigua SBF para que abandone la base.
  3. Operación de cambio de base.

### 2.2.1. Inicialización

Para iniciar el algoritmo es necesario conocer una SBF que será siempre un punto extremo. Dicha SBF debe ser tal que su matriz  $\mathbf{B}$  asociada sea la matriz identidad. Cuando esto no ocurre de forma ‘natural’, nos vemos obligados a introducir variables artificiales para así obtener la matriz identidad. Llegados a ese punto usaremos el método de las dos fases, que explicaremos más adelante, para resolver nuestro problema.

### 2.2.2. Prueba de optimalidad

El siguiente paso es saber si la SBF actual que tenemos en esta primera iteración o en cualquier otra iteración es óptima.

Sea  $\bar{\mathbf{x}}$  un punto extremo y por lo tanto una SBF, podemos poner  $\bar{\mathbf{x}}' = (\bar{\mathbf{x}}_B, \bar{\mathbf{x}}_N)' = (B^{-1}\mathbf{b}, \mathbf{0})$ , con  $B^{-1}\mathbf{b} \geq \mathbf{0}$  y  $A = [B, N]$ . El valor de la función objetivo es  $\mathbf{c}\bar{\mathbf{x}} = \mathbf{c}_B\bar{\mathbf{x}}_B + \mathbf{c}_N\bar{\mathbf{x}}_N = \mathbf{c}_B B^{-1}\mathbf{b}$ .

Las variables correspondientes a la base vienen subindicadas como  $x_1, \dots, x_m$  y el conjunto de índices asociados se denomina  $I = 1, \dots, m$ .

Sea  $\mathbf{x}$  un punto factible cualquiera,  $\mathbf{x}$  puede factorizarse como  $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)'$  y como  $A\mathbf{x} = B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b}$  entonces  $\mathbf{x}_B = B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N$ . El valor de la función objetivo en el punto  $\mathbf{x}$  puede expresarse como  $\mathbf{c}\mathbf{x} = \mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N = \mathbf{c}_B(B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N) + \mathbf{c}_N\mathbf{x}_N$ , operando se obtiene  $\mathbf{c}\mathbf{x} = \mathbf{c}_B B^{-1}\mathbf{b} + (\mathbf{c}_N - \mathbf{c}_B B^{-1}N)\mathbf{x}_N$  pero  $B^{-1}\mathbf{b} = \bar{\mathbf{x}}_B$  luego  $\mathbf{c}\mathbf{x} = \mathbf{c}_B\bar{\mathbf{x}}_B + (\mathbf{c}_N - \mathbf{c}_B B^{-1}N)\mathbf{x}_N$ , es más  $\mathbf{c}\bar{\mathbf{x}} = \mathbf{c}_B\bar{\mathbf{x}}_B + \mathbf{c}_N\bar{\mathbf{x}}_N$ , pero  $\bar{\mathbf{x}}_N = \mathbf{0}$ , con lo que sustituyendo obtenemos:

$$\mathbf{c}\mathbf{x} = \mathbf{c}\bar{\mathbf{x}} + (\mathbf{c}_N - \mathbf{c}_B B^{-1}N)\mathbf{x}_N = \mathbf{c}\bar{\mathbf{x}} + \sum_{j \notin I} (c_j - \mathbf{c}_B B^{-1}A_j)x_j$$

Tener en cuenta las siguientes dos situaciones:

1. Como  $\mathbf{x}_N \geq \mathbf{0}$  para cualquier solución factible, si  $c_j - \mathbf{c}_B B^{-1}A_j \leq 0, \forall j \notin I$  entonces  $\sum_{j \notin I} (c_j - \mathbf{c}_B B^{-1}A_j)x_j \leq 0$  y por lo tanto  $\mathbf{c}\mathbf{x} = \mathbf{c}\bar{\mathbf{x}} + (\mathbf{c}_N - \mathbf{c}_B B^{-1}N)\mathbf{x}_N = \mathbf{c}\bar{\mathbf{x}} + \sum_{j \notin I} (c_j - \mathbf{c}_B B^{-1}A_j)x_j \leq \mathbf{c}\bar{\mathbf{x}}, \forall \mathbf{x}$  factible y  $\bar{\mathbf{x}}$  cumple que  $\mathbf{c}\bar{\mathbf{x}} \leq \mathbf{c}\mathbf{x}, \forall \mathbf{x}$  factible y por lo tanto es la solución óptima del problema.

2. Si por el contrario existen uno o más  $j$  tal que  $c_j - \mathbf{c}_B B^{-1} A_j > 0$  entonces si tomamos una solución en la que  $x_j > 0 (j \notin I)$  toma valor y el resto se quedan como están tenemos:

$$\mathbf{c}\mathbf{x} = \mathbf{c}\bar{\mathbf{x}} + (\mathbf{c}_N - \mathbf{c}_B B^{-1} N)\mathbf{x}_N = \mathbf{c}\bar{\mathbf{x}} + \sum_{j \notin I} (c_j - \mathbf{c}_B B^{-1} A_j)x_j > \mathbf{c}\bar{\mathbf{x}}$$

y por tanto la solución  $\mathbf{x}$  será mejor que  $\bar{\mathbf{x}}$ .

### 2.2.3. Selección de variable que entra en la base

Si estamos en esta situación es porque existen variables  $x_j$  para las que  $c_j - \mathbf{c}_B B^{-1} A_j > 0$ . Por el desarrollo anterior tenemos:

$$\mathbf{c}\mathbf{x} = \mathbf{c}\bar{\mathbf{x}} + \sum_{j \notin I} (c_j - \mathbf{c}_B B^{-1} A_j)x_j$$

Debemos elegir una de las variables no básicas para que entre en la base. Tomaremos aquella variable que mejore en mayor cantidad el valor de la función objetivo por unidad tomada, es decir aquella variable:

$$x_{j_s} : c_{j_s} - \mathbf{c}_B B^{-1} A_{j_s} = \max_{j=j_1, \dots, j_s} \{c_j - \mathbf{c}_B B^{-1} A_j\},$$

ya que a la vista de la expresión anterior por cada unidad de  $x_{j_s}$  la función objetivo aumentará en  $c_j - \mathbf{c}_B B^{-1} A_j$ . Esta regla es heurística, el hecho de seleccionar esta variable no garantiza que la mejora vaya a ser máxima ya que puede ser que la variable seleccionada tome finalmente un valor pequeño, mientras que otra variable  $c_j - \mathbf{c}_B B^{-1} A_j > 0$  y menor que la anterior finalmente pueda tomar un valor grande de manera que la mejora total sea superior.

#### Definición 12.

Dada una SBF con  $\bar{\mathbf{x}} = (\bar{\mathbf{x}}_B, \bar{\mathbf{x}}_N)' = (B^{-1}\mathbf{b}, \mathbf{0})'$  las cantidades  $c_j - \mathbf{c}_B B^{-1} A_j$  se denominan **costes marginales (reducidos, relativos)** asociados a las variables  $x_j$  y representan la cantidad en la que la FO varía por cada unidad que tome la variable  $x_j$ .

*Observación.* Los costes marginales de las variables básicas son cero siempre.

### 2.2.4. Selección de variable que abandona la base

Del proceso anterior supongamos que  $j_0$  es el índice de la variable no básica que queremos que tome valor, es decir, la variable que debe entrar en la nueva base. Construimos un nuevo punto de la forma  $\mathbf{x} = \bar{\mathbf{x}} + \lambda \mathbf{d}^{j_0}$  donde  $\mathbf{d}^{j_0} = \begin{pmatrix} -B^{-1} A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix}$  y  $\mathbf{e}^{j_0}$  es un  $n - m$  vector con un 1 en la posición  $j_0$ -ésima y ceros en el resto.

Vemos que  $A\mathbf{d}^{j_0} = [B, N] \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix} = -BB^{-1}A_{j_0} + A_{j_0} = \mathbf{0}$  y por lo tanto  $A\mathbf{x} = A(\bar{\mathbf{x}} + \lambda\mathbf{d}^{j_0}) = A\bar{\mathbf{x}} + \lambda A\mathbf{d}^{j_0} = \mathbf{b}, \forall \lambda$ . Como conclusión, cualquier punto  $\mathbf{x}$  de la forma anterior verifica  $A\mathbf{x} = \mathbf{b}$ , para que el punto sea factible, deberá verificar además que  $\mathbf{x} \geq \mathbf{0}$ , es decir que  $\mathbf{x} = \bar{\mathbf{x}} + \lambda \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{x}}_B \\ \mathbf{0} \end{pmatrix} + \lambda \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix} \geq \mathbf{0}$ .

Observamos que si  $\lambda > 0$  entonces la coordenada correspondiente a  $x_{j_0}$  es mayor que cero. Se plantean dos casos:

a) Si  $B^{-1}A_{j_0} \leq \mathbf{0}$  para todas sus coordenadas entonces  $-B^{-1}A_{j_0} \geq \mathbf{0}$  y  $\lambda(-B^{-1}A_{j_0} \geq \mathbf{0}, \forall \lambda \geq 0$ . Y por lo tanto, sea cual sea  $\lambda$  el punto  $\mathbf{x}$  es factible.

b) Si  $B^{-1}A_{j_0} \not\leq \mathbf{0}$  las cosas no quedan tan sencillas como antes,

$\mathbf{x} = \begin{pmatrix} B^{-1}\mathbf{b} + \lambda(-B^{-1}A_{j_0}) \\ \lambda\mathbf{e}^{j_0} \end{pmatrix}$ , y habrá que elegir  $\lambda$  de manera que ese vector  $\mathbf{x}$  se mantenga mayor o igual que cero.

Usamos la siguiente notación

$$B^{-1}\mathbf{b} = \bar{\mathbf{b}} = (\bar{b}_1, \dots, \bar{b}_m)'$$

$$B^{-1}A_{j_0} = Y_{j_0} = (y_{1j_0}, \dots, y_{mj_0})'$$

con esto

$$x = \begin{pmatrix} \bar{\mathbf{b}} - \lambda Y_{j_0} \\ \lambda \mathbf{e}^{j_0} \end{pmatrix} = \begin{pmatrix} \bar{b}_1 - \lambda y_{1j_0} \\ \dots \\ \bar{b}_m - \lambda y_{mj_0} \\ 0 \\ \dots \\ \lambda \\ \dots \\ 0 \end{pmatrix}$$

con algún  $y_{i_{j_0}} > 0$  por hipótesis. Para satisfacer que  $\mathbf{x} \geq \mathbf{0}$  bastará con que  $\bar{b}_i - \lambda y_{i_{j_0}} \geq 0$  para aquellos  $i$  en los que  $y_{i_{j_0}} > 0$ , o lo que es equivalente, que para estos  $i$  se cumpla que  $\lambda \leq \frac{\bar{b}_i}{y_{i_{j_0}}}$ , observamos que para los  $y_{i_{j_0}} \leq 0$  no hay problema, en conclusión, se debe cumplir que  $\lambda \leq \frac{\bar{b}_i}{y_{i_{j_0}}}, \forall i$  tal que  $y_{i_{j_0}} > 0$ .

Observamos también que si tomamos

$$\lambda = \min_{1 \leq i \leq m} \left\{ \frac{\bar{b}_i}{y_{i_{j_0}}} : y_{i_{j_0}} > 0 \right\}$$

se cumple la condición anterior. Es más, si el mínimo se alcanza en  $s$ , es decir

$$\lambda = \frac{\bar{b}_s}{y_{s_{j_0}}} = \min_{1 \leq i \leq m} \left\{ \frac{\bar{b}_i}{y_{i_{j_0}}} : y_{i_{j_0}} > 0 \right\},$$

entonces  $\bar{b}_s - \lambda y_{s j_0} = 0$  y  $\mathbf{x}$  tiene con valor las variables que tenía  $\bar{\mathbf{x}}$  menos  $x_s$  que pasa a tomar valor cero y en su lugar aparece la variable  $x_{j_0}$  que toma valor  $\lambda$ , el resto de variables toman valor  $\bar{b}_i - \frac{\bar{b}_s}{y_{s j_0}} y_{i j_0}$ .

Con esto acabamos de construir una SBF adyacente a la anterior en la que  $x_{j_0}$  toma valor como deseábamos.

Resumiendo, una vez detectada la nueva variable  $x_{j_0}$  que debe introducirse en la base, determinaremos la variable que sale de la base como aquella  $x_s$  en la que se alcanza el siguiente mínimo:

$$\lambda = \frac{\bar{b}_s}{y_{s j_0}} = \min_{1 \leq i \leq m} \left\{ \frac{\bar{b}_i}{y_{i j_0}} : y_{i j_0} > 0 \right\},$$

además el nuevo punto es  $\mathbf{x} = \bar{\mathbf{x}} + \lambda \mathbf{d}^{j_0}$ .

El valor de la función objetivo en  $\mathbf{x} = \bar{\mathbf{x}} + \lambda \mathbf{d}^{j_0}$  en cualquiera de los dos casos será:

$$\begin{aligned} \mathbf{c}\mathbf{x} &= \mathbf{c}\bar{\mathbf{x}} + \mathbf{c}(\lambda \mathbf{d}^{j_0}) = \mathbf{c}\bar{\mathbf{x}} + \lambda \mathbf{c} \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix} \\ &= \mathbf{c}\bar{\mathbf{x}} + \lambda(\mathbf{c}_B, \mathbf{c}_N) \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix} \\ &= \mathbf{c}\bar{\mathbf{x}} + \lambda(-\mathbf{c}_B B^{-1}A_{j_0} + c_{j_0}) \end{aligned}$$

Recordar que  $\lambda = x_{j_0}$ . Veamos como queda el valor de la función objetivo en los dos casos que hemos considerado anteriormente.

- a) En esta situación no había nungún problema, independientemente del valor de  $\lambda$  la solución era factible,  $c_{j_0} - \mathbf{c}_B B^{-1}A_{j_0} > 0$ , por lo tanto  $\mathbf{c}\bar{\mathbf{x}} + \lambda(-\mathbf{c}_B B^{-1}A_{j_0} + c_{j_0}) \rightarrow \infty$  cuando  $0 < \lambda \rightarrow \infty$  y el problema tiene solución no acotada.

Observamos que en este caso  $\mathbf{d}^{j_0} = \begin{pmatrix} -B^{-1}A_{j_0} \\ \mathbf{e}^{j_0} \end{pmatrix}$  es una dirección extrema cumpliendo que  $\mathbf{c}\mathbf{d}^{j_0} > 0$  ( $\mathbf{c}\mathbf{d}^{j_0} = c_{j_0} - \mathbf{c}_B B^{-1}A_{j_0}$ ), luego estamos en las condiciones del teorema de condiciones de optimalidad en P.P.L. que nos permite ratificar nuevamente que estamos ante una solución no acotada.

- b) En este otro caso el valor de la nueva SBF habrá mejorado en  $|\mathbf{c}\bar{\mathbf{x}} - \mathbf{c}\mathbf{x}| = \lambda|c_{j_0} - \mathbf{c}_B B^{-1}A_{j_0}|$  y como  $x_{j_0} = \lambda$  la función objetivo habrá mejorado en  $|-\mathbf{c}_B B^{-1}A_{j_0} + c_{j_0}|$  unidades por cada unidad de  $x_{j_0}$ .

El formato de tablas que utilizaremos para aplicar el Método del Simplex es:

			$c_1 \ c_2 \ \dots \ c_n$	
			$x_1 \ x_2 \ \dots \ x_n$	$\bar{b}$
$c_1$	$x_1$		$y_{11} \ \dots \ y_{n1}$	$\bar{b}_1$
$c_2$	$x_2$		$y_{12} \ \dots \ y_{n2}$	$\bar{b}_2$
$\vdots$	$\vdots$		$\vdots$	$\vdots$
$c_m$	$x_m$		$y_{1m} \ \dots \ y_{nm}$	$\bar{b}_m$
			$\mathbf{c} - \mathbf{c}_B B^{-1}A$	

Tras todo esto, podemos resumir el algoritmo que sigue el Método del Simplex como vemos en la figura 2.1:

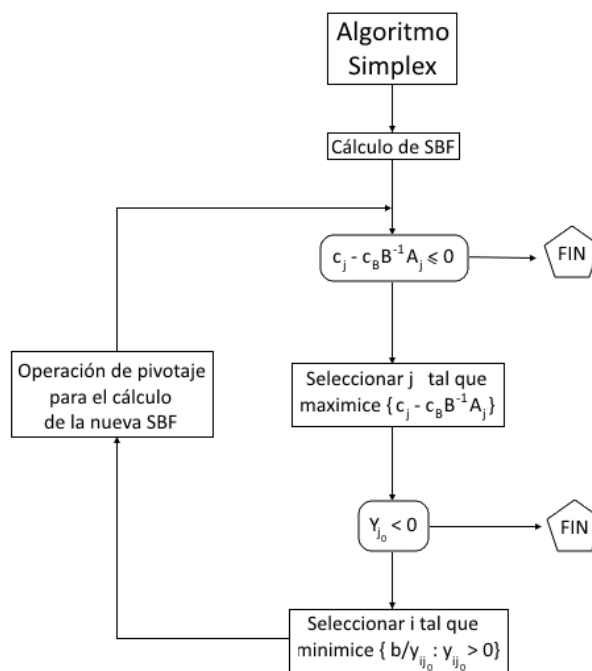


Figura 2.1: Diagrama de flujo del Algoritmo del Simplex

Vamos a ver un ejemplo en el que aplicaremos el Método del Simplex. Veremos cuál es la solución básica factible inicial, calcularemos cuál es la variable que debe entrar y cuál es la que debe salir de la base y veremos cómo pivotar. Haremos las iteraciones necesarias hasta llegar a una solución básica factible óptica y la interpretaremos.

**Ejemplo 1.**

Una empresa elabora cereales integrales y cereales especiales para dieta (cereales tipo A y tipo B, respectivamente, a partir de ahora). Sabemos que dispone para su elaboración de 150 g de trigo y 100 g de cebada diariamente. Además, el horario de fabricación es de 80 minutos al día. También sabemos que para hacer un paquete de cereales tipo A necesitan 5 g de trigo, 2 g de cebada y 4 minutos de trabajo, y para hacer un paquete de cereales tipo B necesitan 2 g de trigo, 3 g de cebada y 2 minutos de trabajo. Además, el precio por el que se venden es 12 € para cada paquete de cereales tipo A y 8 € para cada paquete de cereales tipo B. Vamos a plantear un problema que maximice el neto obtenido al vender los paquetes de cereales que se hacen diariamente.

**Solución.** Vamos a desglosar el problema en unos pocos pasos muy sencillos, uno de ellos será el método para pivotar.



- a) Planteamos el problema, siendo  $x_1$  el número de paquetes de cereales tipo A y  $x_2$  el número de paquetes de cereales tipo B:

$$\begin{aligned} \text{máx} \quad & Z = 12x_1 + 8x_2 \\ \text{s.a :} \quad & 5x_1 + 2x_2 \leq 150 \\ & 2x_1 + 3x_2 \leq 100 \\ & 4x_1 + 2x_2 \leq 80 \\ & x_1, x_2 \geq 0 \end{aligned}$$

- b) Introducimos variables de holgura  $x_3, x_4$  y  $x_5$  para convertir las desigualdades en igualdades:

$$\begin{aligned} \text{máx} \quad & Z = 12x_1 + 8x_2 + 0x_3 + 0x_4 + 0x_5 \\ \text{s.a :} \quad & 5x_1 + 2x_2 + x_3 = 150 \\ & 2x_1 + 3x_2 + x_4 = 100 \\ & 4x_1 + 2x_2 + x_5 = 80 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

- c) Hacemos la primera tabla del método. Las variables que vamos a elegir para formar nuestra primera solución básica factible serán aquellas que forman la matriz identidad:  $x_3, x_4, x_5$ :

		12	8	0	0	0	
		$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$\bar{b}$
0	$x_3$	5	2	1	0	0	150
0	$x_4$	2	3	0	1	0	100
0	$x_5$	4	2	0	0	1	80
		12	8	0	0	0	

- d) Como  $z_i > 0$  para algún  $i$ , esta solución no es óptima. Debemos construir una nueva SBF adyacente a la actual. Para ello, vamos a elegir la variable básica que debe entrar en la base, ésta será la que, con  $w_i > 0$ , haga máximo  $w_i$ . En este caso vemos que será  $\text{máx}\{12, 8\} = 12$ , por tanto la variable que debe entrar a la base es  $x_1$ .
- e) Elijamos ahora la variable que debe salir de la base, que será la que, con  $y_{1j} > 0$  (tomamos  $i = 1$  ya que hemos visto que la variable de entrada es  $x_1$ ) haga mínimo el cociente  $\frac{b_j}{y_{1j}}$ , es decir:  $\text{mín}\{\frac{150}{5}, \frac{100}{8}, \frac{80}{4}\} = \frac{80}{4}$ , luego abandona la base  $x_5$ .
- f) **Pivotamos:**

1. Comenzamos a escribir la nueva tabla con la nueva base  $x_3, x_4$  y  $x_1$ . Debemos hacer que el elemento  $y_{13} = 1$  (ya que es el elemento en el que intersecan la

columna de la variable que ha entrado  $x_1$  y la fila de la variable que salió), para ello basta con dividir la fila 3ª entre 4:

		12	8	0	0	0	$\bar{b}$
		$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
0	$x_3$						
0	$x_4$						
12	$x_1$	1	1/2	0	0	1/4	20

2. Ahora tenemos que hacer ceros todos los elementos que estén por encima y por debajo de nuestro pivote, mediante combinación lineal de nuestra fila pivote con el resto (en este caso solo tenemos elementos por encima):

		12	8	0	0	0	$\bar{b}$
		$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
0	$x_3$	0	-1/2	1	0	-5/4	50
0	$x_4$	0	2	0	1	-1/2	60
12	$x_1$	1	1/2	0	0	1/4	20
		0	2	0	0	-3	

- g) Como sigue habiendo algún  $z_i > 0$  la solución no es óptima. Elegimos la nueva variable que entra en la base:  $\max\{2\} = 2$ , entonces entra  $x_2$  y saldrá:  $\min\{\frac{60}{2}, \frac{20}{\frac{1}{2}}\} = \frac{60}{2}$ , sale  $x_4$ . Construimos la nueva tabla pivotando esta vez el elemento  $y_{22}$ :

		12	8	0	0	0	$\bar{b}$
		$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
0	$x_3$	0	0	1	1/4	-11/8	65
8	$x_2$	0	1	0	1/2	-1/4	30
12	$x_1$	1	0	0	-1/4	3/8	5
		0	0	0	-1	-5/2	

Como  $z_i \leq 0, \forall i = 1, 2, 3, 4, 5$  hemos llegado a la SBF óptima, la cual es:  $x_1 = 5, x_2 = 30, x_3 = 65, x_4 = x_5 = 0$ . Esto se traduce en que al día se deben fabricar 5 paquetes de cereales tipo A, 30 paquetes de cereales tipo B y además nos sobrarán 65 g de trigo al día, los cuales podremos invertirlos en otros productos o directamente reducir la cantidad de trigo que compramos. Y además, obtendremos un total de  $12 \times 5 + 8 \times 30 = 300$  € al día.

### 2.3. Método de las dos fases

Vamos a hacer una breve introducción a este método. Para un estudio más profundo se recomienda la lectura del libro [7].

Como el mismo nombre indica, este método consta de dos fases:

1. Fase 1: Definimos la función objetivo y las restricciones del problema y agregamos las variables artificiales necesarias para asegurar una solución básica de inicio. A continuación, se determina una solución básica de las ecuaciones resultantes, que *minimice* la suma de las variables artificiales. Si el valor mínimo de la suma es positivo, el problema no tiene solución factible y terminamos el proceso. En caso contrario, pasamos a la fase 2.
2. Fase 2: Utilizamos la solución factible de la fase 1 como inicio para el problema original.

Veamos un ejemplo práctico para que sea más claro, que se encuentra recogido en [7].

### Ejemplo 2.

$$\begin{array}{ll}
 \text{Minimizar} & z = 4x + y \\
 \text{sujeto a} & : \quad 3x + y = 3 \\
 & \quad 4x + 3y \geq 6 \\
 & \quad x + 2y \leq 4 \\
 & \quad x, y \geq 0
 \end{array}$$

**Solución.** En primer lugar vamos a escribir nuestro problema en forma estándar. Para ello debemos añadir dos variables de holgura positivas  $h_1$  y  $h_2$ :

$$\begin{array}{ll}
 \text{Minimizar} & z = 4x + y + 0h_1 + 0h_2 \\
 \text{sujeto a} & : \quad 3x + y = 3 \\
 & \quad 4x + 3y - h_1 = 6 \\
 & \quad x + 2y + h_2 = 4 \\
 & \quad x, y, h_1, h_2 \geq 0
 \end{array}$$

Observamos que no podemos encontrar una submatriz en la matriz de coeficientes  $A$  tal que sea la matriz identidad:  $A = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 4 & 3 & -1 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix}$ . Por tanto, nos vemos obligados a incluir las variables artificiales  $a_1$  y  $a_2$  positivas para así obtener una solución básica factible inicial. Al hacer esto, debemos penalizar en la función objetivo añadiendo las variables artificiales con un coeficiente  $M$  “muy grande” y positivo. El problema queda

de la siguiente forma:

$$\begin{aligned}
 \text{Minimizar} \quad & z = 4x + y + 0h_1 + 0h_2 + Ma_1 + Ma_2 \\
 \text{sujeto a} \quad & : 3x + y + a_1 = 3 \\
 & 4x + 3y - h_1 + a_2 = 6 \\
 & x + 2y + h_2 = 4 \\
 & x, y, h_1, h_2, a_1, a_2 \geq 0
 \end{aligned} \tag{2.5}$$

Obteniendo así la matriz de coeficientes  $A = \begin{pmatrix} 3 & 1 & 0 & 0 & 1 & 0 \\ 4 & 3 & -1 & 0 & 0 & 1 \\ 1 & 2 & 0 & 1 & 0 & 0 \end{pmatrix}$ . Como hemos tenido que recurrir al uso de variables artificiales, vamos a usar el método de las dos fases para resolver el ejemplo:

1. **Fase 1.** Creamos la función  $r = a_1 + a_2$  y junto a las restricciones de nuestro problema (2.5), planteamos el siguiente P.P.L:

$$\begin{aligned}
 \text{Minimizar} \quad & r = a_1 + a_2 \\
 \text{sujeto a} \quad & : 3x + y + a_1 = 3 \\
 & 4x + 3y - h_1 + a_2 = 6 \\
 & x + 2y + h_2 = 4 \\
 & x, y, h_1, h_2, a_1, a_2 \geq 0
 \end{aligned}$$

Tomamos  $a_1, a_2$  y  $h_2$  como las variables básicas iniciales y construimos la primera tabla de nuestro problema (como es un problema de minimizar, observar que minimizar  $x =$  maximizar  $-x$ ):

		0	0	0	0	-1	-1	$\bar{b}$
		$x$	$y$	$h_1$	$h_2$	$a_1$	$a_2$	
-1	$a_1$	3	1	0	0	1	0	3
-1	$a_2$	4	3	-1	0	0	1	6
0	$h_2$	1	2	0	1	0	0	4
		7	4	-1	0	0	0	

Procedemos resolviendo este problema, como vimos en el ejemplo 1. Por tanto, ahora entra en la base  $x$  y sale  $a_1$ . La tabla quedaría:

		0	0	0	0	-1	-1	$\bar{b}$
		$x$	$y$	$h_1$	$h_2$	$a_1$	$a_2$	
0	$x$	1	1/3	0	0	1/3	0	1
-1	$a_2$	0	5/3	-1	0	-4/3	1	2
0	$h_2$	0	5/3	0	1	-1/3	0	3
		0	5/3	-1	0	-7/3	0	

Aún tenemos  $z_i \geq 0$ , así que debemos hacer una nueva iteración del algoritmo. Ahora entrará en la base la variable  $y$  y saldrá la variable  $a_2$ :

		0	0	0	0	-1	-1	
		$x$	$y$	$h_1$	$h_2$	$a_1$	$a_2$	$\bar{b}$
0	$x$	1	0	1/5	0	3/5	-1/5	3/5
0	$y$	0	1	-3/5	0	-4/5	3/5	6/5
0	$h_2$	0	0	1	1	1	-1	1
		0	0	0	0	-1	-1	

Todos los coeficientes  $z_i \leq 0$  así que hemos llegado a una solución básica factible óptima, la cual es:  $x = 3/5, y = 6/5, h_2 = 1$ . Vemos que nuestra función  $z = 0+0 = 0$  no es positiva, así que podemos pasar a la fase 2. Llegados a este punto, como todas las variables artificiales que introdujimos han salido de la base, las eliminamos de la tabla y pasamos a la fase 2.

2. **Fase 2.** Una vez eliminadas las columnas correspondientes a las variables artificiales, tenemos la siguiente tabla:

		0	0	0	0	
		$x$	$y$	$h_1$	$h_2$	$\bar{b}$
0	$x$	1	0	1/5	0	3/5
0	$y$	0	1	-3/5	0	6/5
0	$h_2$	0	0	1	1	1
		0	0	0	0	

A partir de ella, creamos un nuevo problema con la función óptima que teníamos al principio,  $z = 4x + y$ , y las restricciones que se obtienen de esta tabla. Así tenemos el siguiente P.P.L:

$$\begin{aligned}
 \text{Minimizar} \quad & z = 4x + y \\
 \text{suje}to \quad & a : x + \frac{h_1}{5} = \frac{3}{5} \\
 & y - \frac{3h_1}{5} = \frac{6}{5} \\
 & h_1 + h_2 = 1 \\
 & x, y, h_1, h_2 \geq 0
 \end{aligned}$$

A partir de aquí se procedería normalmente aplicando el Método del Simplex, ya que en la matriz de coeficientes  $A$  ya podemos encontrar una submatriz identidad y no tenemos ninguna variable artificial.

## 2.4. Instalación de la librería *Apache Commons Math*

En esta sección vamos a introducir e instalar la librería *Apache Commons Math*, la cual usaremos para resolver problemas de ecuaciones diferenciales, centrándonos especialmente en los que no tienen solución analítica.

### 2.4.1. Introducción a Apache Commons

Commons es un proyecto de Apache centrado en todos los aspectos de componentes reutilizables de Java, es de código abierto. El proyecto Apache Commons está compuesto de tres partes:

1. The Commons Proper; un repositorio de componentes reutilizables de Java.
2. The Commons Sandbox; Un espacio de trabajo para el desarrollo de los componentes de Java.
3. The Commons Dormant; un repositorio de componentes que habitualmente se encuentran inactivos.

#### The Commons Proper

The Commons Proper está dedicada a un objetivo principal: creación y mantenimiento de componentes reutilizables de Java. Dicha aplicación funciona como lugar para la colaboración e intercambio, donde los colaboradores y creadores de este software de todas partes de la comunidad de Apache pueden trabajar juntos sobre proyectos.

#### The Commons Sandbox

Este proyecto también contiene un espacio de trabajo que es abierto a todo el conjunto de usuarios de Apache. Es un lugar para probar nuevas ideas y prepararse para la inclusión en la parte de The Commons Proper del proyecto o en otro proyecto de Apache. Los usuarios son libres de experimentar con los componentes desarrollados en The Commons Sandbox, pero los componentes de The Commons Sandbox no necesariamente serán mantenidos, en particular en su estado corriente.

#### The Commons Dormant

Está formado por los componentes de The Commons Sandbox que han sido considerados inactivos ya que han tenido poca actividad de desarrollo reciente. Si se desea usar cualquiera de estos componentes, se deben construir por uno mismo.

### 2.4.2. Commons Math: The Apache Commons Mathematics Library

Commons Math es una librería ligera e independiente, de componentes software que implementan algoritmos matemáticos y estadísticos, que aborda los problemas más comunes que no están disponibles en el lenguaje de programación Java o Commons Lang.

## Instalación librería

Para poder usar esta librería debemos seguir los siguientes pasos:

1. Descargar los archivos commons-math3-3.3-bin.zip y commons-math3-3.3-src.zip de la página web [8] (Ver Fig. 2.2).

The screenshot shows the Apache Commons Math website. At the top left is the Apache Commons logo with the URL <http://commons.apache.org/>. To the right is the Commons Math logo. Below the logos is a navigation bar with the text "Apache Commons Math™", "Last Published: 15 May 2014 | Version: 3.3", and links for "ApacheCon", "Apache", and "Commons".

The main content area is titled "Download Apache Commons Math". Underneath is a section "Using a Mirror" with a paragraph of text and a form for "Other mirrors:" with a dropdown menu showing "http://apache.rediris.es/" and a "Change" button.

Below that is a section "Apache Commons Math 3.3 (requires Java 1.5+)" followed by a "Binaries" section containing a table:

<a href="#">commons-math3-3.3-bin.tar.gz</a>	<a href="#">md5</a>	<a href="#">pgp</a>
<a href="#">commons-math3-3.3-bin.zip</a>	<a href="#">md5</a>	<a href="#">pgp</a>

Below the binaries table is a "Source" section containing another table:

<a href="#">commons-math3-3.3-src.tar.gz</a>	<a href="#">md5</a>	<a href="#">pgp</a>
<a href="#">commons-math3-3.3-src.zip</a>	<a href="#">md5</a>	<a href="#">pgp</a>

On the left side of the page is a navigation menu with sections "MATH" and "USER GUIDE". The "MATH" section includes links for Overview, Downloads, Latest API docs (development), and several Javadoc releases (3.3, 3.2, 3.1.1, 3.1, 3.0, 2.2). The "USER GUIDE" section includes links for Contents, Overview, Statistics, Data Generation, Linear Algebra, Numerical Analysis, Special Functions, Utilities, Complex Numbers, Distributions, Fractions, Transform Methods, 3D Geometry, Optimization, Ordinary Differential Equations, Genetic Algorithms, and Filters.

Figura 2.2: Descargas de librerías

2. Creamos una carpeta, que llamaremos lib, dentro de nuestro proyecto (ProyectoJava).

3. Descomprimos los archivos que descargamos, en ellos buscamos los archivos commons-math3-3.3-javadoc.jar, commons-math3-3.3-sources.jar y commons-math3-3.3.jar y los copiamos en nuestra carpeta lib. (Ver Fig. 2.3)

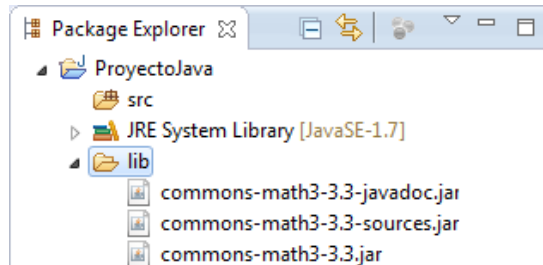


Figura 2.3: Archivos .jar de la librería en carpeta lib

4. Sobre nuestro proyecto ProyectoJava, hacemos click con el botón derecho y seleccionamos Properties → Java Build Path → Libraries  
Nos aparecerá una ventana como la de la Fig. 2.4:

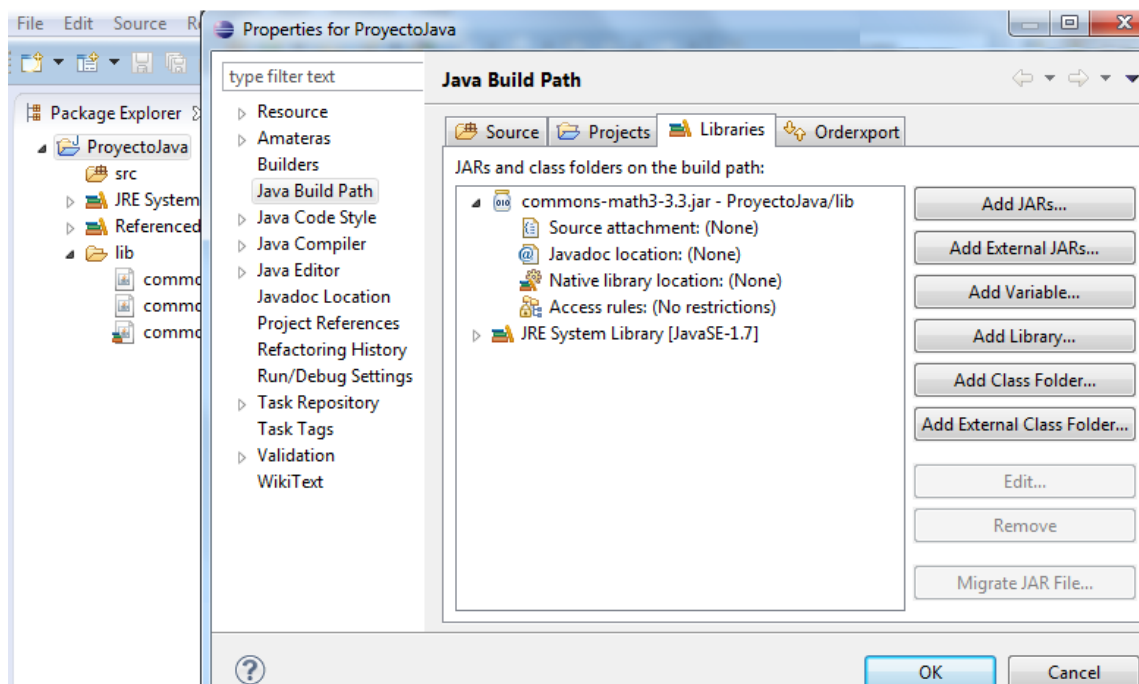


Figura 2.4: Propiedades del ProyectoJava



5. Seleccionamos **Source attachment:(None)** y editamos en la opción **Edit**, situada en la parte derecha. En la ventana abierta, marcamos **Workspace location** y al examinar (**Browse**) dentro de la carpeta **lib**, seleccionamos el archivo **commons-math3-3.3-sources.jar** (Ver 2.5).

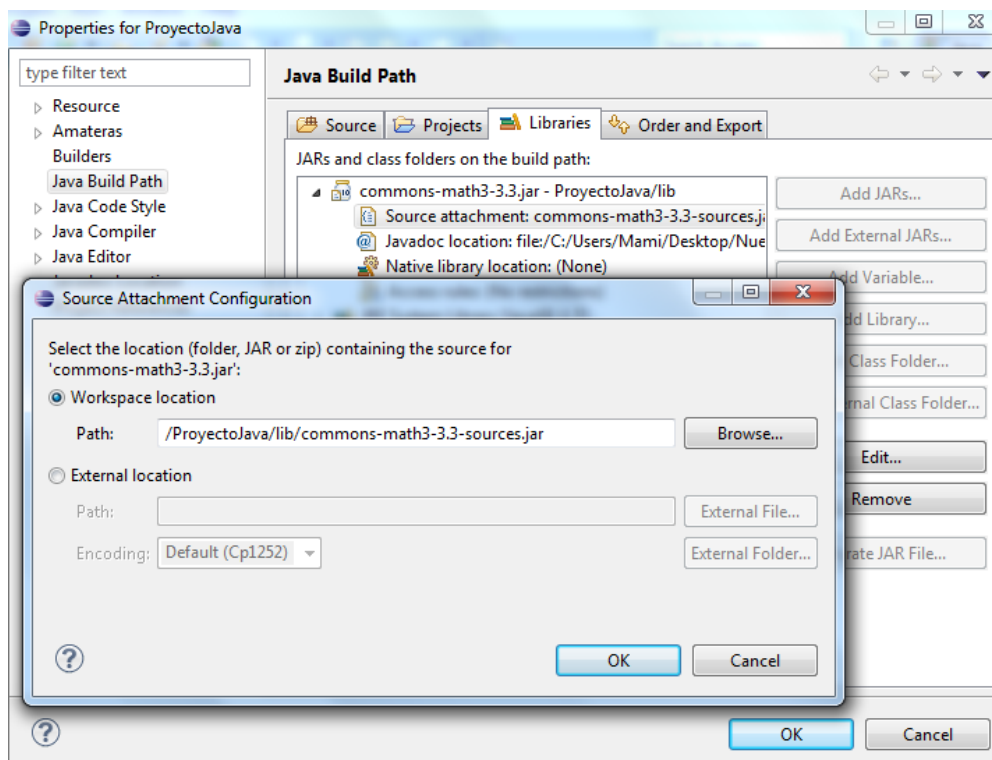


Figura 2.5: Incluyendo las fuentes

6. Seleccionamos **Javadoc location:(None)** y editamos igual que el paso anterior. En la ventana abierta, marcamos **Javadoc in archive** → **External file** y en la opción **Archive path** examinamos (**Browse**) y seleccionamos dentro de la carpeta **lib** el archivo **commons-math3-3.3-javadoc.jar** (Ver Fig 2.6).

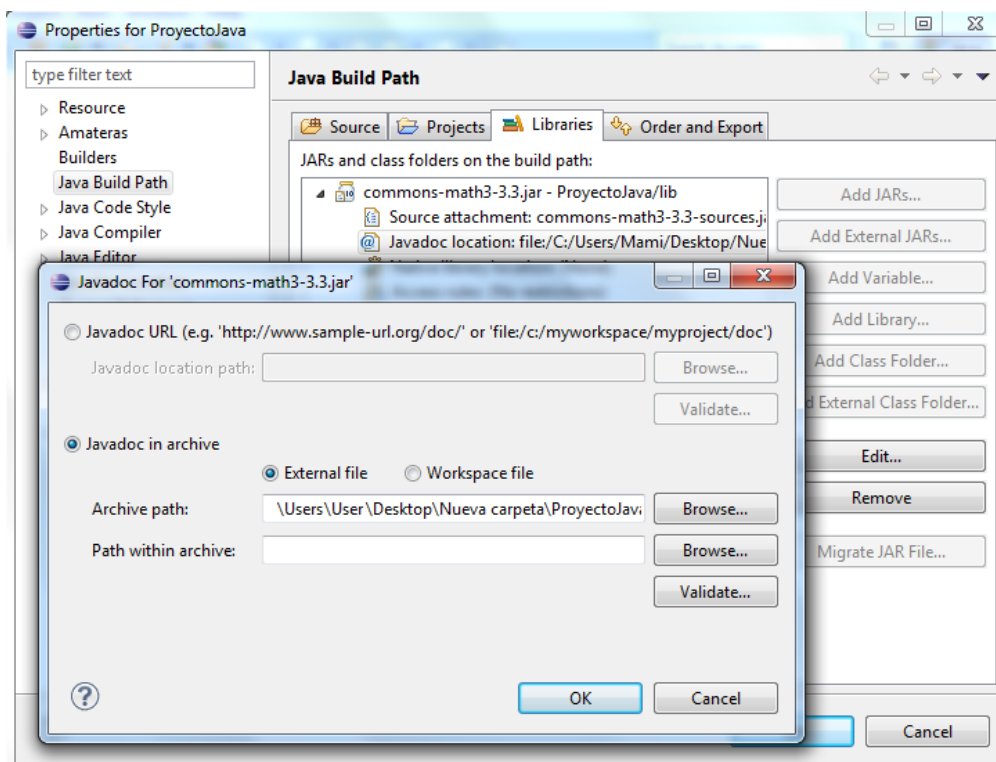


Figura 2.6: Incluyendo Javadoc

Así concluimos la instalación de esta librería.

---

## Capítulo 3

# Resultados y Discusión

Una vez estudiado el Método del Simplex, vemos que para problemas con muchas variables puede resultar tedioso y largo de resolver. Por ello vamos a recurrir a utilizar programas que nos resuelvan este tipo de P.P.L.

Como dijimos, nos hemos centrado en la librería **Apache Commons Math** ya que es una librería open source a la que se puede acceder, modificar y adaptar según las necesidades del problema. En concreto, hemos hecho uso del paquete *optim* implementado en la versión 3.2 de la librería. Gracias a este paquete hemos podido resolver ejercicios de programación lineal con unas pocas y sencillas líneas de código, en las cuales definimos nuestra función objetivo y las restricciones del problema, a continuación llamamos al constructor para que genere el problema y finalmente lo resolvemos y mostramos los resultados por consola.

### 3.1. Resultados

Hemos elegido dos ejemplos tipo para ver cómo funciona la librería: Problema del Transporte [9] y Problema de la Dieta [10].

#### 3.1.1. Problema del Transporte

Este tipo de problema es muy importante ya que en la actualidad el transporte es esencial y siempre se busca optimizar en tiempo y minimizar el gasto, ya sea en transporte de mercancías o en rutas de vuelo de un avión. Hemos elegido un problema en el que una empresa tiene que transportar desde sus tres almacenes, un producto, a tres de sus tiendas. Concretamente el enunciado es<sup>1</sup>:

---

<sup>1</sup>Los datos que utilizamos en el enunciado han sido obtenidos del libro [11]

**Ejemplo 3.**

Una empresa de hostelería tiene almacenados 120 kg de harina en sus almacenes de Sevilla, Huelva y Málaga, de modo que tiene 40 kg en el almacén de Sevilla, 50 kg en el de Huelva y 30 kg en el de Málaga. Han recibido los siguientes pedidos: una tienda de Córdoba requiere 25 kg de harina, una panadería en Huelva ha solicitado 35 kg y un supermercado de Cádiz necesita 60 kg. El coste del transporte entre los distintos almacenes y los destinos y el nombre asignado a cada recorrido lo vemos en el diagrama 3.1.

Y con todos estos datos construimos nuestra función objetivo y las restricciones:

$$\text{mín } 7a + 6b + 8c + 9d + 4e + 3f + 5g + 8h + 6i$$

Sujeto a:

$$\text{Oferta} = \begin{cases} a + b + c \leq 40 \\ d + e + f \leq 50 \\ g + h + i \leq 30 \end{cases} \quad \text{Demanda} = \begin{cases} a + d + g \geq 25 \\ b + e + h \geq 35 \\ c + f + i \geq 60 \end{cases}$$

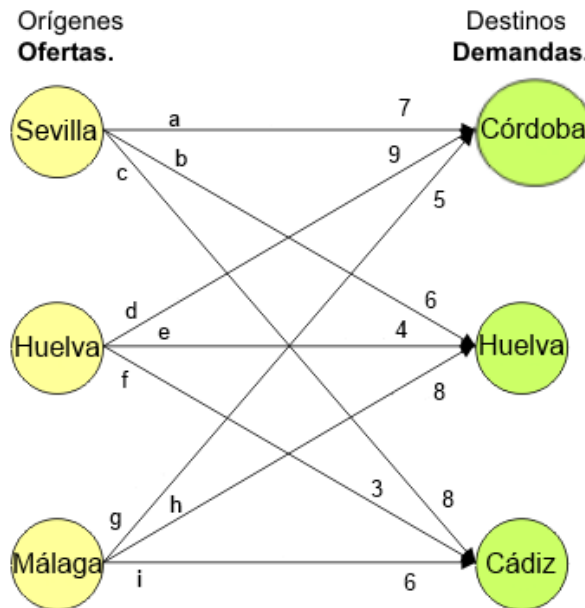


Figura 3.1: Diagrama del ejemplo del Problema del Transporte

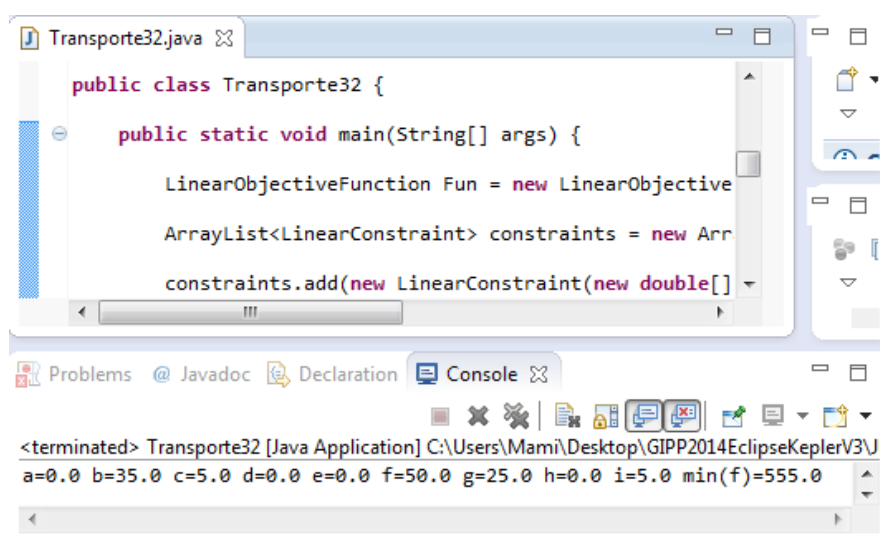
A continuación se muestra el código que hemos implementado en *java* para la resolución de este ejercicio:

```
1 package org.tfg.ejemplos;
2
3 import java.util.ArrayList;
4
5 import org.apache.commons.math3.optim.MaxIter;
6 import org.apache.commons.math3.optim.PointValuePair;
7 import org.apache.commons.math3.optim.linear.LinearConstraintSet;
8 import org.apache.commons.math3.optim.linear.LinearObjectiveFunction;
9 import org.apache.commons.math3.optim.linear.LinearConstraint;
10 import org.apache.commons.math3.optim.linear.NonNegativeConstraint;
11 import org.apache.commons.math3.optim.linear.Relationship;
12 import org.apache.commons.math3.optim.linear.SimplexSolver;
13 import org.apache.commons.math3.optim.nonlinear.scalar.GoalType;
14
15 /*
16  *Minimize: 7 a + 6 b + 8 c + 9 d + 4 e + 3 f + 5 g + 8 h + 6 i
17  *Sujeto a:
18  *
19  *Ofertas:
20  *a+b+c<=40
21  *d+e+f<=50
22  *g+h+i<=30
23  *
24  *Demandas:
25  *a+d+g>=25
26  *b+e+h>=35
27  *c+f+i>=60
28  *
29  */
30
31 public class Transporte32 {
32
33     public static void main(String[] args) {
34
35         LinearObjectiveFunction Fun = new
36             LinearObjectiveFunction(new double[] { 7, 6, 8, 9,
37                 4, 3, 5, 8, 6 }, 0);
38
39         ArrayList<LinearConstraint> constraints = new
40             ArrayList<LinearConstraint>();
41
42         constraints.add(new LinearConstraint(new double[] {
43             1, 1, 1, 0, 0, 0, 0, 0, 0 }, Relationship.LEQ, 40)
44             );
45         constraints.add(new LinearConstraint(new double[] {
46             0, 0, 0, 1, 1, 1, 0, 0, 0 }, Relationship.LEQ, 50)
```

```
41     );
42     constraints.add(new LinearConstraint(new double [] {
43         0, 0, 0, 0, 0, 0, 1, 1, 1 }, Relationship.LEQ, 30)
44     );
45     constraints.add(new LinearConstraint(new double [] {
46         1, 0, 0, 1, 0, 0, 1, 0, 0 }, Relationship.GEQ, 25)
47     );
48     constraints.add(new LinearConstraint(new double [] {
49         0, 1, 0, 0, 1, 0, 0, 1, 0 }, Relationship.GEQ, 35)
50     );
51     constraints.add(new LinearConstraint(new double [] {
52         0, 0, 1, 0, 0, 1, 0, 0, 1 }, Relationship.GEQ, 60)
53     );
54
55     SimplexSolver solver = new SimplexSolver();
56
57     PointValuePair solution = solver.optimize(new MaxIter
58         (10000), Fun, new LinearConstraintSet(constraints)
59         ,GoalType.MINIMIZE, new NonNegativeConstraint(true)
60     ));
61
62     // get the solution
63     double a = solution.getPoint()[0];
64     double b = solution.getPoint()[1];
65     double c = solution.getPoint()[2];
66     double d = solution.getPoint()[3];
67     double e = solution.getPoint()[4];
68     double f = solution.getPoint()[5];
69     double g = solution.getPoint()[6];
70     double h = solution.getPoint()[7];
71     double i = solution.getPoint()[8];
72
73     double min = solution.getValue();
74
75     System.out.println( "a=" +a+ " b=" +b+
76         " c=" +c+ " d=" +d+ " e=" +e+ " f=" +f+ "
77         g=" +g+ " h=" +h+ " i=" +i+ " min(f) = " +
78         min);
79 }
80 }
```

Los pasos que hemos implementado son:

1. Definir la función objetivo. Entre las llaves escribimos en orden los coeficientes de las variables de la función objetivo y el último número fuera del corchete, en este caso 0, es el término independiente.
2. Creamos las restricciones. De nuevo, entre llaves escribimos en orden los coeficientes de las variables de cada restricción y al final indicamos el tipo de relación que tienen (*LEQ* = *Lesser than or Equal*, *GEQ* = *Greater than or Equal*, *EQUAL*) con el término final.
3. Generamos el problema. Ahora hacemos uso de todo lo anterior, usamos el método de objeto `optimize` para resolver el problema al cual tenemos que indicarle:
  - a) Número máximo de iteraciones.
  - b) La función objetivo que hemos creado anteriormente.
  - c) Las restricciones que hemos generado.
  - d) Hacia dónde queremos dirigir el problema (minimizar o maximizar).
  - e) Si las variables definidas deben o no ser no negativas.
4. Finalmente hemos generado una salida por consola para comprobar que el problema se resuelve correctamente. La salida se muestra en la figura 3.2.



```
public class Transporte32 {  
    public static void main(String[] args) {  
        LinearObjectiveFunction Fun = new LinearObjective  
        ArrayList<LinearConstraint> constraints = new Arr  
        constraints.add(new LinearConstraint(new double[]
```

<terminated> Transporte32 [Java Application] C:\Users\Mami\Desktop\GIPP2014EclipseKeplerV3\J  
a=0.0 b=35.0 c=5.0 d=0.0 e=0.0 f=50.0 g=25.0 h=0.0 i=5.0 min(f)=555.0

Figura 3.2: Solución del ejemplo del Problema del Transporte

Utilizamos seguidamente el programa privativo *Mathematica*. Con el siguiente código implementado, comprobamos que el resultado coincide.

```

1 Minimize[{7 a+6 b+8 c+9 d+4 e+3 f+5 g+8 h+6 i, a>=0,b>=0,c>=0,d>=0,e>=0,f>=0,g
  >=0,h>=0,i>=0,a+b+c<=40,d+e+f<=50,g+h+i<=30,a+d+g>=25,b+e+h>=35,c+
  f+i>=60},{a,b,c,d,e,f,g,h,i}]
2 {555,{a->0,b->35,c->5,d->0,e->0,f->50,g->25,h->0,i->5}}
```

### 3.1.2. Problema de la Dieta

Este problema representa una de las primeras aplicaciones de la programación lineal. Comenzó a utilizarse en los hospitales para determinar la dieta más económica con la que alimentar a los pacientes a partir de unas especificaciones nutritivas mínimas. En la actualidad, también se aplica con éxito en el ámbito agrícola con la idea de encontrar la combinación óptima de alimentos que, logrando un aporte nutritivo mínimo, suponga el menor coste posible.

Vamos a ver un ejemplo del problema de la dieta, obtenido del libro *Optimizando recursos con programación lineal* [12].

#### Ejemplo 4.

Se quiere alimentar el ganado de una granja con la dieta más económica posible. Dicha dieta debe contener cuatro tipos de nutrientes identificados como A, B, C, y D. Estos componentes se encuentran en dos tipos de piensos M y N. La cantidad, en gramos, de cada componente por kilo de estos piensos viene dada en la tabla siguiente:

	A	B	C	D
M	100	-	100	200
N	-	100	200	100

La dieta diaria de un animal debe estar compuesta por al menos 0.4 Kg del componente A, 0.6 Kg del componente B, 2 Kg del componente C y 1.7 Kg del componente D. El compuesto M cuesta 0.2 €/Kg y el compuesto N 0.08 €/Kg. ¿Qué cantidades de piensos M y N se deben adquirir para que el gasto en comida sea el menor posible?

A continuación se muestra el código que hemos implementado en *Java* para resolver este ejemplo:

```

1 package org.tfg.ejemplos;
2
3 import java.util.ArrayList;
4
```



```
5 import org.apache.commons.math3.optim.MaxIter;
6 import org.apache.commons.math3.optim.PointValuePair;
7 import org.apache.commons.math3.optim.linear.LinearConstraintSet;
8 import org.apache.commons.math3.optim.linear.LinearObjectiveFunction;
9 import org.apache.commons.math3.optim.linear.LinearConstraint;
10 import org.apache.commons.math3.optim.linear.NonNegativeConstraint;
11 import org.apache.commons.math3.optim.linear.Relationship;
12 import org.apache.commons.math3.optim.linear.SimplexSolver;
13 import org.apache.commons.math3.optim.nonlinear.scalar.GoalType;
14
15 /*
16  * Se propone alimentar el ganado de una granja con la dieta mas
17  * economica posible. Dicha
18  * dieta debe contener cuatro tipos de nutrientes identificados como
19  * A, B, C, y D. Estos
20  * componentes se encuentran en dos tipos de piensos M y N.
21  * La dieta diaria de un animal debe estar compuesta por al menos 0.4
22  * Kg del componente A,
23  * 0.6Kg del componente B, 2Kg del componente C, y 1.7Kg del
24  * componente D. El compuesto M
25  * cuesta 0.2 euros el kilo y el compuesto N 0.08 euros el kilo. Que
26  * cantidades de piensos M y N se deben
27  * adquirir para que el gasto en comida sea el menor posible?
28  * x: cantidad de pienso M en Kg
29  * y: cantidad de pienso N en Kg
30  * Componente A:  $0.1x \geq 0.4$ 
31  * Componente B:  $0.1y \geq 0.6$ 
32  * Componente C:  $0.1x + 0.2y \geq 2$ 
33  * Componente D:  $0.2x + 0.1y \geq 1.7$ 
34  *  $x \geq 0$ 
35  *  $y \geq 0$ 
36  * La funcion objetivo es: Minimizar  $Z = 0.2x + 0.08y$ 
37  */
38
39 public class Dieta32 {
40
41     public static void main(String[] args) {
42
43         LinearObjectiveFunction f = new
44             LinearObjectiveFunction(new double[] { 0.2, 0.08
45             }, 0);
46
47         ArrayList<LinearConstraint> constraints = new
48             ArrayList<LinearConstraint>();
49
50         constraints.add(new LinearConstraint(new double[] {
51             0.1, 0 }, Relationship.GEQ, 0.4));
52         constraints.add(new LinearConstraint(new double[] {
```

```

44     0, 0.1 }, Relationship.GEQ, 0.6));
45     constraints.add(new LinearConstraint(new double [] {
46         0.1, 0.2 }, Relationship.GEQ, 2));
47     constraints.add(new LinearConstraint(new double [] {
48         0.2, 0.1 }, Relationship.GEQ, 1.7));
49
50     SimplexSolver solver = new SimplexSolver();
51
52     PointValuePair solution = solver.optimize(new MaxIter
53         (1000000), f, new LinearConstraintSet(constraints)
54         , GoalType.MINIMIZE, new NonNegativeConstraint(true
55         ));
56
57     // get the solution
58         double x = solution.getPoint()[0];
59         double y = solution.getPoint()[1];
60
61         double min = solution.getValue();
62
63         System.out.println("x=" + x + " y=" + y + "
64             min(f) = " + min);
65     }
66 }

```

Hemos seguido exactamente los mismos pasos que en el ejemplo anterior utilizando por tanto los mismos métodos de la librería para definir la función objetivo, las restricciones y mostrar la solución como muestra la figura 3.3.

The screenshot shows the Eclipse IDE with the following content:

```

Dieta32.java
* y: cantidad de pienso N en Kg
* Componente A: 0.1x >= 0.4
* Componente B: 0.1y >= 0.6
* Componente C: 0.1x + 0.2y >= 2
* Componente D: 0.2x + 0.1y >= 1.7
* x >= 0
* y >= 0
* La funcion objetivo es: Minimizar Z = 0.2x
*/

public class Dieta32 {

    public static void main(String[] args) {

```

The console output is:

```

<terminated> Dieta32 [Java Application] C:\Users\Mami\Desktop\GIPP2014EclipseKeplerV3\JDK
x= 4.000000000000001 y= 9.000000000000002 min(f)= 1.520000000000005

```

Figura 3.3: Solución del ejemplo del Problema de la Dieta

Igual que en ejemplo del transporte, comprobamos el resultado con el obtenido al resolver el problema de la dieta con *Mathematica* utilizando el siguiente código:

```

1 Minimize[{0.2x+0.08y, 0.1x>=0.4,0.1y>=0.6,0.1x+0.2y>=2,0.2x+0.1y
2 >=1.7},{x,y}]
{1.52,{x->4.,y->9.}}
```

## 3.2. Discusión

Como hemos visto, es muy sencillo implementar un problema de programación lineal con la librería *Apache Commons Math*. Las funciones son simples e intuitivas y esto ayuda a adaptar y resolver los problemas que se puedan plantear. Como cabía esperar, los resultados son los esperados utilizando tanto las librerías como el programa *Mathematica*.

En este trabajo sólo hemos visto una parte del paquete `optim` referida a la resolución de problemas con funciones y restricciones lineales (Ver 3.4).

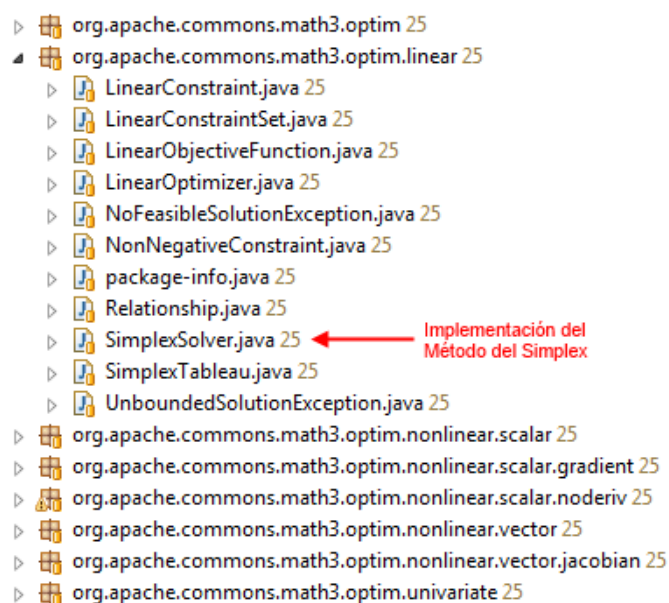


Figura 3.4: Paquete `org.apache.commons.math3.optim.linear`

Si miramos en la web oficial [13], encontramos una breve descripción de las clases que se implementan en el paquete que estamos usando (Ver 3.5).

### Package org.apache.commons.math3.optim.linear

Optimization algorithms for linear constrained problems.

Class Summary	
Class	Description
LinearConstraint	A linear constraint for a linear optimization problem.
LinearConstraintSet	Class that represents a set of <code>linear constraints</code> .
LinearObjectiveFunction	An objective function for a linear optimization problem.
LinearOptimizer	Base class for implementing linear optimizers.
NonNegativeConstraint	A constraint for a linear optimization problem indicating whether all variables must be restricted to non-negative values.
SimplexSolver	Solves a linear problem using the "Two-Phase Simplex" method.
SolutionCallback	A constraint for a linear optimization problem indicating whether all variables must be restricted to non-negative values.

Enum Summary	
Enum	Description
PivotSelectionRule	Pivot selection rule to the use for a Simplex solver.
Relationship	Types of relationships between two cells in a Solver <code>LinearConstraint</code> .

Exception Summary	
Exception	Description
NoFeasibleSolutionException	This class represents exceptions thrown by optimizers when no solution fulfills the constraints.
UnboundedSolutionException	This class represents exceptions thrown by optimizers when a solution escapes to infinity.

### Package org.apache.commons.math3.optim.linear Description

Optimization algorithms for linear constrained problems.

Figura 3.5: Descripción del paquete org.apache.commons.math3.optim.linear

En la figura 3.6 vemos un diagrama de clases UML (Unified Modeling Language) donde se muestran las clases que hemos usado y las relaciones que hay entre ellas.

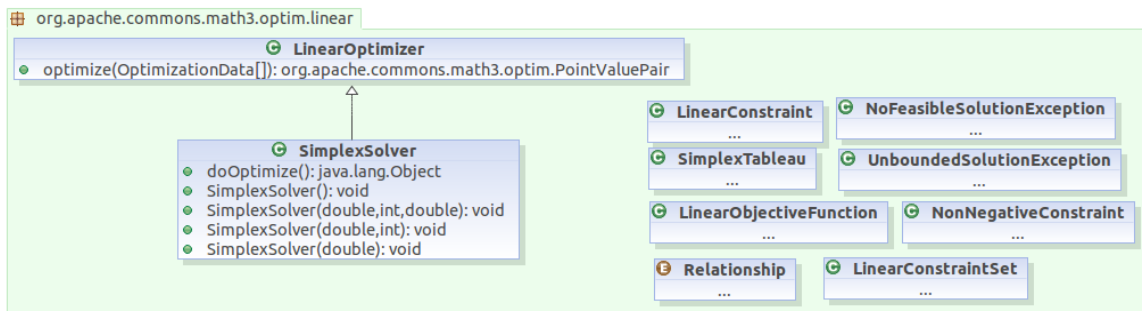


Figura 3.6: Diagrama de clases UML

Al realizar la búsqueda de información sobre la resolución de problemas de programación lineal haciendo uso de la librería *Apache Commons Math* y, concretamente, haciendo uso de la parte dedicada a problemas lineales del paquete `optim`, he podido comprobar que tiene métodos para resolver problemas no lineales y univariados (Ver 3.7).









- ▷  org.apache.commons.math3.optim 25
- ▷  org.apache.commons.math3.optim.linear 25
- ▷  org.apache.commons.math3.optim.nonlinear.scalar 25
- ▷  org.apache.commons.math3.optim.nonlinear.scalar.gradient 25
- ▷  org.apache.commons.math3.optim.nonlinear.scalar.noderiv 25
- ▷  org.apache.commons.math3.optim.nonlinear.vector 25
- ▷  org.apache.commons.math3.optim.nonlinear.vector.jacobian 25
- ▷  org.apache.commons.math3.optim.univariate 25

Figura 3.7: Otros paquetes de Apache Commons Math Optim

Todo lo anterior da una idea de hasta dónde podemos llegar con las librerías que hay implementadas hasta ahora, pero siempre podemos modificarlas, adaptarlas, mejorarlas o crear nuevas.

---

# Capítulo 4

## Conclusiones

A continuación se describen los puntos relevantes con que se puede concluir este trabajo:

1. La resolución de problemas de programación lineal es un tema importante y abordado desde hace muchos años. A partir de que Dantzig desarrollase el Método del Simplex se han podido resolver gran parte de estos problemas.
2. Debido a la dificultad que plantea el Método del Simplex para problemas con gran número de variables y restricciones, es bueno conocer y poder trabajar con programas que permitan implementar y resolver el problema de manera correcta.
3. El uso de librerías de código abierto nos permite adaptar y seleccionar el código existente para conseguir resolver el ejercicio que se nos plantee.
4. Una característica del paquete *optim* es que con cuatro sencillos pasos podemos implementar cualquier problema de programación lineal y obtener su solución.
5. *Mathematica* también es una herramienta potente para la resolución de ejercicios pero, debido a que no es de código abierto, no podemos acceder a sus funciones y modificarlas, por tanto debemos de adaptar siempre nuestro problema a sus condiciones.

---

# Bibliografía

- [1] P.M. Mateo y D. Lahoz. Programación lineal 1. 2009.
- [2] Rene Meziat. Introducción al análisis convexo y los fundamentos de la programación matemática. 2005.
- [3] Fundamentos de investigación de operaciones 1. método simplex. Agosto 2004.
- [4] Jose Luis de la Fuente O'Connor. *Técnicas de cálculo para Sistemas de Ecuaciones, Programación Lineal y Programación Entera*. Reverte s.a. edition.
- [5] Pagina web de eclipse. URL <http://www.eclipse.org/>.
- [6] Oscar Eduardo Gómez Rojas. Fundamentos matemáticos del método simplex. 2004.
- [7] Hamdy A. Taha. *Investigación de Operaciones*. Pearson edition, 2004. URL <http://vagosuatfis.files.wordpress.com/2012/07/thaja-investigacion-de-operaciones-by-k9.pdf>.
- [8] Link de descarga de la librería apache commons math. URL [http://commons.apache.org/proper/commons-math/download\\_math.cgi](http://commons.apache.org/proper/commons-math/download_math.cgi).
- [9] Activity analysis of production and allocation. 1951. URL <http://cowles.econ.yale.edu/P/cm/m13/>.
- [10] George B. Dantzig. The diet problem. 1990.
- [11] Ing. Carlos I. Careaga de la Garza. *Investigación de Operaciones: Programación Lineal y Aplicaciones*. Polilibros edition, 2004. URL [http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/P\\_terminados/Investigacion\\_de\\_Operaciones\\_Careaga/Common/izquierda.htm](http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/P_terminados/Investigacion_de_Operaciones_Careaga/Common/izquierda.htm).
- [12] Optimizando recursos con programación lineal. URL <http://www.phpsimplex.com/index.htm>.
- [13] Pagina oficial apache commons math. URL <http://commons.apache.org/proper/commons-math/javadocs/api-3.3/index.html>.

---

# Apéndices



---

# Apéndice A

## Programación en Java y entorno Eclipse

### A.1. Programación en Java

#### A.1.1. Origen del lenguaje de programación en Java

El lenguaje de programación Java tiene sus orígenes en un lenguaje de programación anterior, llamado *Oak* (roble en inglés), que nació de un proyecto interno en *Sun Microsystems* en el año 1991 llamado *Green project*.

*Oak* fue creado con el objetivo de ser el lenguaje de programación con el que programar dispositivos electrónicos domésticos, en particular aparatos de televisión inteligentes e interactivos, aunque esto nunca se materializó. De modo simultáneo, a principios de la década de los 90 surgió *Internet* y con ella, la aparición de los primeros *navegadores web*. Los líderes del *Green project* fueron conscientes de la importancia que iba a tener *Internet* y orientaron su lenguaje de programación *Oak* para que programas escritos en este lenguaje se pudiesen ejecutar dentro del navegador web *Mozilla*. Y éste fue el inicio de *Java*, así llamado porque se intentó registrar el nombre *Oak* y éste ya estaba registrado.

La ventaja de que un programa escrito en Java se pueda ejecutar en una gran cantidad de plataformas ha hecho de él un interesante lenguaje de programación por su « *universalidad* »

#### A.1.2. Características de Java

Java es un lenguaje de programación orientado a objetos y de propósito general que toma de otros lenguajes de programación algunas ideas fundamentales, en particular toma de *Smalltalk* el hecho de que los programas Java se ejecutan sobre una máquina virtual. Y del lenguaje de programación C++ toma su sintaxis. De entre las muchas características que Java posee destacamos:

- Java es multiplataforma.
- Java es seguro.
- Java tiene un amplio conjunto de bibliotecas estándar.

- Java incluye una biblioteca portable para la creación de interfaces gráficas de usuario (AWT en Java 1.0/1.1, JFC/Swing en Java 2 y Java FX).
- Java simplifica algunos aspectos a la hora de programar.

### A.1.3. El entorno de desarrollo integrado Eclipse

Un entorno integrado de desarrollo o IDE de sus siglas en inglés (Integrated Develop Enviroment) nos permite escribir código de un modo cómodo. La comodidad reside en que los entornos de desarrollo integrado son mucho más que un simple editor de textos.

Eclipse reúne todas las características comunes a los modernos IDE. Además posee un sistema de *plug-ins* con los que se pueden añadir nuevas funcionalidades. Por ejemplo, mediante un plug-in nos podemos conectar al sistema de control de versiones *Subversion*.

#### Descarga e instalación de Eclipse

Eclipse se puede descargar desde el sitio web <http://www.eclipse.org>. Existen versiones para las principales plataformas y sistemas operativos.

Una particularidad de Eclipse es que no necesita instalación. Una vez descargado el fichero comprimido, lo único que debemos hacer para empezar a utilizarlo es descomprimirlo en algún directorio y seguidamente ejecutar el archivo correspondiente. La Figura A.1 muestra la página de inicio de Eclipse. Los iconos que muestra esta pantalla son enlaces a sitios de información sobre Eclipse.

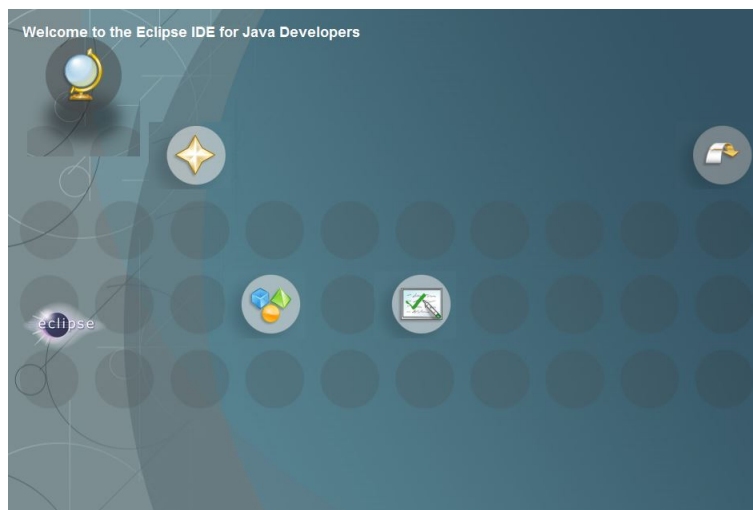


Figura A.1: Pantalla inicial de *Eclipse*

En la Figura A.2 vemos la perspectiva inicial de Eclipse y las diferentes partes de esta ventana principal:

1. Explorador de paquetes.
2. Editor de código.
3. Área compuesta por múltiples vistas que podremos seleccionar según nuestras necesidades. Entre ellas destacamos “Problems”, que mostrará mensajes de error en tiempo de compilación, “Console”, que será donde se muestren las salidas de ejecución.

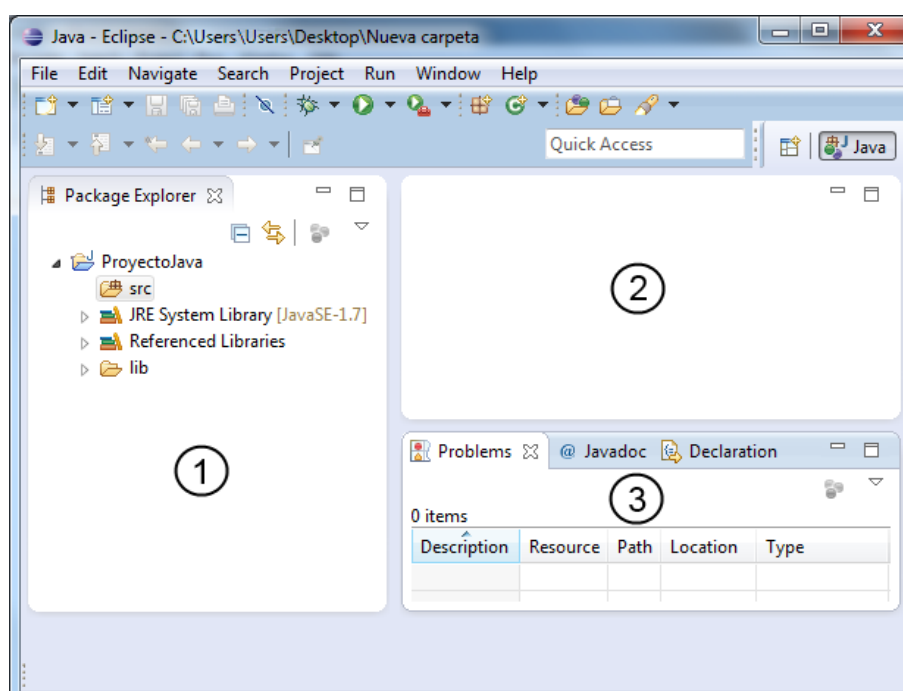


Figura A.2: Perspectiva inicial orientada al desarrollo de proyectos Java2 SE.

## Perspectivas

Una perspectiva de Eclipse es una agrupación de vistas y editores de manera que den apoyo a una actividad completa del proceso de desarrollo software. Los editores normalmente permiten realizar una tarea completa, las vistas proporcionan funciones de apoyo o auxiliares tales como mostrar información, requerir datos, etc. Las perspectivas pueden seleccionarse haciendo clic en los iconos de perspectiva en la esquina superior derecha (Ver A.3) o eligiendo **Window**→**Open Perspective** del menú.

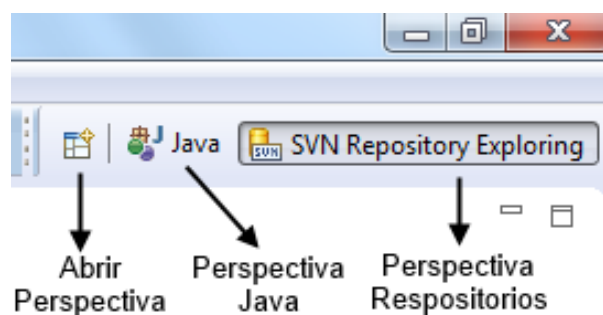


Figura A.3: Perspectivas

Nosotros hemos trabajado con las siguientes perspectivas:

- **Java:** esta perspectiva se centra en tareas de programación, mostrando paquetes, clases, métodos y atributos en sus vistas asociadas.
- **Debug:** esta perspectiva se abre cuando lanzamos una ejecución con el depurado (Ver A.4). Un depurador es una herramienta que permite seguir el programa línea a línea y ver cómo cambian los valores de las variables y expresiones que tengamos seleccionadas durante la ejecución. Permite, por tanto, controlar y analizar la ejecución del programa. Ayuda a localizar diferentes tipos de errores.

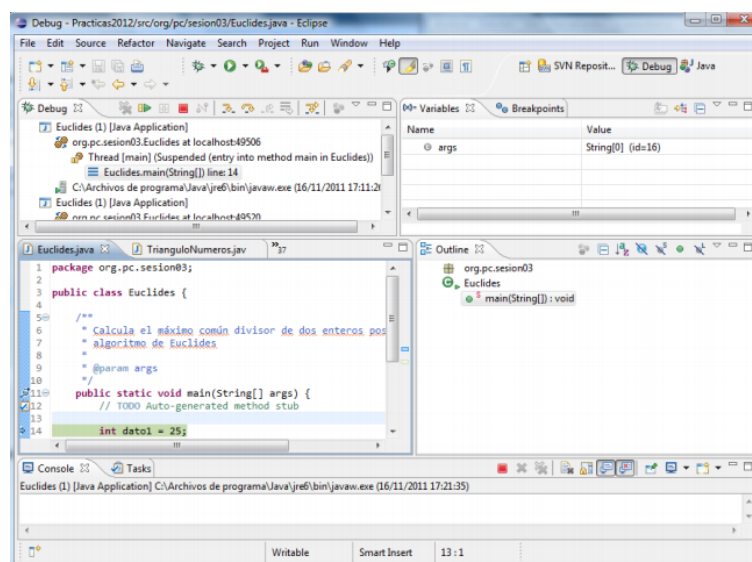


Figura A.4: Perspectiva Depurador

- **SVN:** esta perspectiva se centra en el trabajo con repositorios. Permitirá establecer conexiones con diferentes repositorios, acceder a los mismos, navegar por ellos, etc... En el siguiente apartado desarrollaremos esta perspectiva.

## A.1.4. Subversion

### ¿Qué es un sistema de control de versiones?

Un sistema de control de versiones es una herramienta software que, de manera automática, se encarga de facilitar la gestión de las versiones del código de un proyecto de manera centralizada.

### Principales características de *Subversion*

El concepto central en *Subversion* es el *Repositorio*. Por repositorio se entiende la última versión del proyecto que existe en el sistema de control de versiones.

El paradigma que Subversion utiliza es *Copia-Modificación-Fusión* (Copy-Modify-Merge en inglés). En este paradigma, cada uno de los miembros del equipo, cuando empieza a trabajar en el proyecto, hace una copia local del contenido del repositorio; modifica su copia local y finalmente fusiona sus modificaciones locales con el código del repositorio. Al finalizar esta fase, se dice que se ha creado una nueva versión del proyecto en el repositorio.

Una de las características principales de *Subversion* es que las actualizaciones en el repositorio son incrementales, sólo se actualizan los ficheros que se han modificado respecto a la versión anterior. Otra característica es relativa a la numeración de la versión del repositorio, cada vez que se realiza una modificación en el repositorio, se actualiza la versión de todos los ficheros existentes en el repositorio y no únicamente de los ficheros que se han modificado.

### Trabajar con Repositorios

Debemos abrir la perspectiva de **SVN Repository Exploring** y crear el repositorio con el icono de nuevo repositorio. Para hacer la conexión daremos la dirección, autenticación y clave necesaria para acceder al repositorio. (Ver A.5)

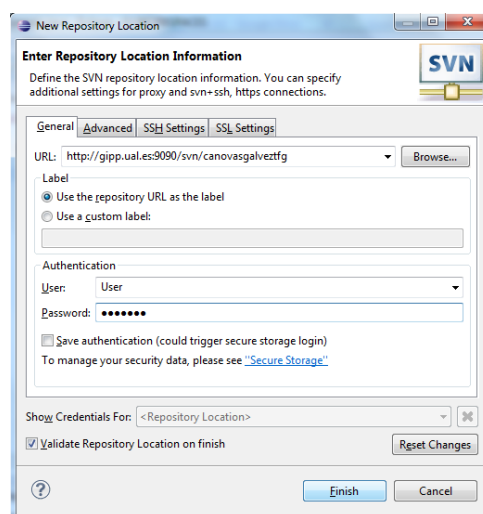


Figura A.5: Nuevo Repositorio

Ahora vamos a ver una serie de opciones que nos permite trabajar con los repositorios:

- **Check out:** Para traernos un proyecto Java desde la perspectiva del repositorio a nuestra perspectiva Java. En la perspectiva SVN: **Botón derecho sobre el proyecto** → **Check out**.
- **Commit:** Una vez trabajado en nuestro proyecto en la perspectiva Java, para subirlo al repositorio tenemos que hacer un Commit. En la perspectiva Java: **Botón derecho sobre el proyecto** → **Team** → **Commit**.
- **Update:** Actualiza el proyecto. Trae las modificaciones del repositorio al proyecto. En la perspectiva Java: **Botón derecho sobre el proyecto** → **Team** → **Update**.