Modelos matemáticos para la gestión de producción de productos perecederos

Sara Coscolluela López

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: F. Javier López Lorente
27 de noviembre de 2019

Prólogo

A lo largo de este proyecto, vamos a analizar y estudiar métodos de resolución de problemas de gestión de stocks, en particular, de productos perecederos como son los concentrados de plaquetas. En concreto, realizaremos el análisis de dos de dichos métodos y los emplearemos para resolver nuestro problema que se verá en el Capítulo 2.

Debido a que los modelos de aprovisionamiento no son estudiados a lo largo del Grado y a la dificultad que reside en los métodos de resolución para productos perecederos, ha sido necesario estudiar primero los modelos matemáticos de aprovisionamiento sobre productos no perecederos, que se puede ver en una asignatura del Máster Universitario de Modelización e Investigación Matemática, Estadística y Computación, ver [9].

Además, para entender los artículos que hemos analizado, eran necesarios algunos conceptos dados en asignaturas del Grado en Matemáticas como, Introducción a la probabilidad y la estadística, Análisis numérico, Cálculo de probabilidades, Estadística matemática, entre otros, pero sobretodo, Optimización estocástica, asignatura optativa del Grado.

En la actualidad existen una gran cantidad de artículos con la resolución de problemas de producción y gestión de productos no perecederos. Sin embargo, hay un menor número de artículos sobre productos perecederos y, más aún, cuando el tiempo de vida útil del producto supera los dos períodos, como será el problema que planteemos sobre concentrados de plaquetas (vida útil de cinco períodos).

Los métodos en los que profundizamos a lo largo del TFG no han sido utilizados hasta la fecha para resolver el problema particular de producción óptima de unidades de concentrados de plaquetas, de ahí la novedad de este trabajo.

Abstract

The present work deals with the mathematical optimization of production of perishable items and, more specifically, of blood platelet concentrates. Optimization of management and/on production of perishable items is a complicated problem and analytical solutions are only known in small and very particular cases.

Therefore, there is a number of approximate solutions which work well in practice but require a detailed analysis of the models for their implementation. We consider the optimization of the production of blood platelets in the Basque Center for Transfusions, see [6], and we are going to apply two different methods, one analytic [12] and another based on dynamic programming [8].

We remark that the application of these methods to our model is not straightforward and they need to be adaptaded for our situation (for instance, the lifetime of units in [6] is 2, while it is 5 in our case, consequently we have expressions more expressions). Once we have adapted the techniques to our model, we have written programs in C, run them and analyzed their optimal solutions under several settings.

Índice general

Prólogo III

Abstract V

1. Modelos matemáticos de aprovisionamiento 1
1.1. Objetivo, conceptos básicos y métodos . . . 1
1.2. Modelos estocásticos . . . 2
1.2.1. Modelo de revisión continua (r,q) . . . 3
1.2.2. Modelo de revisión periódica (R,S) . . . 3
1.3. Productos perecederos y revisión bibliográfica . . . 4

2. Planteamiento del problema y objetivo 7

3. Un modelo de inventario perecedero con tiempo de entrega del pedido positivo 9
3.1. Artículo de referencia . . . 9
3.1.1. Notación . . . 9
3.1.2. Problema de tiempo de vida de dos períodos . . . 10
3.2. Problema de producción de plaquetas (problema de tiempo de vida de cinco períodos) . 11
3.2.1. Resultados experimentales . . . 14

4. Programación dinámica 19
4.1. Introducción . . . 19
4.2. Problema estocástico de la producción de plaquetas . . . 21
4.2.1. Resultados experimentales . . . 22

Bibliografía 27

A. Ecuaciones, teoremas y otros 29

B. Código C 37
B.1. Programa para artículo Patuwo y Williams . . . 37
B.2. Programa para programación dinámica . . . 74

Capítulo 1

Modelos matemáticos de aprovisionamiento

En primer lugar, veremos el objetivo principal del aprovisionamiento de productos no perecederos, los conceptos básicos necesarios para entender el funcionamiento de los modelos matemáticos de aprovisionamiento y mencionaremos algunos de sus métodos.

En segundo lugar, nos centraremos en los modelos estocásticos, ya que nuestro problema planteado en este trabajo se basa en modelos aleatorios. En particular, prestaremos atención a los modelos de múltiples pedidos. Es necesario entender en qué consisten los principales modelos aleatorios de productos no perecederos para saber interpretarlos en el caso de perecederos.

1.1. Objetivo, conceptos básicos y métodos

El objetivo primordial del aprovisionamiento es organizar los pedidos de un producto satisfaciendo las demandas de los clientes y cumpliendo ciertos objetivos de coste y beneficio.

Para poder estudiar modelos matemáticos que gestionen el aprovisionamiento, debemos ver algunos conceptos básicos. Solo mencionaremos los más relevantes para nuestro objetivo, se pueden consultar otros en [10]:

- Demanda: cantidad de productos que hay que enviar a nuestros clientes. Puede ser conocida de antemano o no.
- Stock: almacén con existencias de ciertos productos.
- Modelo determinístico: si la demanda es conocida de antemano.
- Modelo aleatorio o estocástico: no conocemos con certeza la demanda a satisfacer, pero sí una estimación de ésta basada en predicciones mediante datos de temporadas anteriores o por los conocimientos que adquirimos del problema.
- Productos "fast movers": debemos satisfacer una gran demanda de pedidos pequeños.
- Existen diferentes formas sobre cómo realizar los pedidos al proveedor:
    1. Modelo de un único pedido.
    2. Modelos en los que debemos hacer los pedidos de manera continuada durante un largo periodo de tiempo (dos tipos: sistema de revisión periódica y de revisión continua, los veremos más adelante).
- Lead-time o tiempo de entrega, L: tiempo que transcurre entre que realizamos un pedido y lo recibimos. Puede ser:

    - Determinista, es conocido.
    - Aleatorio, no es conocido, necesitamos una estimación del lead-time.

- Ruptura del stock: sucede en el momento en que tenemos que servir una cantidad que no tenemos disponible en el almacén.
- Inventory position: es la situación actual del inventario. La cantidad fundamental a la hora de gestionar el aprovisionamiento y realizar pedidos.
- Reorder point o punto de pedido: en los modelos de revisión continua, es el valor tal que cuando el inventory position se sitúa por debajo suyo, se realiza un nuevo pedido.
- Ciclo: tiempo que transcurre entre dos pedidos.
- Nivel del servicio: porcentaje de ciclos en los que servimos todas las unidades a tiempo.
- Stock de seguridad: cantidad media esperada en stock cuando nos llega el pedido.
- Factor de seguridad: constante que relaciona el stock de seguridad con la desviación típica de la variable que representa la demanda durante el lead-time.

A lo largo de esta memoria, realizaremos abreviaciones como:

- u. para unidades de producto.
- u.m. para unidades monetarias.
- u.t. para unidades de tiempo.

En la siguiente sección, vamos a tratar los modelos matemáticos principales para resolver problemas estocásticos. También existen modelos matemáticos para problemas deterministas, pero este último tipo no es el objetivo de nuestro trabajo por lo que no profundizamos en ellos, pueden consultarse en [10].

1.2. Modelos estocásticos

Los modelos estocásticos o aleatorios son aquellos en los que ciertas cantidades no son conocidas exactamente y solo tenemos previsiones suyas (demanda, lead-time, ...). En estos modelos la ruptura del stock es de vital importancia ya que puede aparecer debido a que la demanda diste de las previsiones que nosotros habíamos realizado.

El principal problema de estos modelos reside en que es muy costoso o imposible garantizar con total seguridad que no se va a producir una ruptura del stock.

El análisis de los modelos estocásticos consiste en calcular el coste total del aprovisionamiento y el nivel de servicio, en función de las variables decisión. En nuestro caso, estas variables decisión, serán cuántas unidades de producto pedimos y cuándo las pedimos. Por tanto, dado una política de aprovisionamiento podemos calcular su coste y su nivel de servicio.

En los modelos de este tipo, las cantidades aleatorias (como la demanda) se modelan mediante variables aleatorias. Lo idóneo es conocer su distribución de probabilidad o su función de densidad, pero en ocasiones no es posible; por tanto, debemos tener estimaciones de su valor medio y su varianza. Esto es posible si se dispone de datos anteriores fiables para realizar la predicción y así, calcular la media y la varianza de dichos datos y utilizarlas como valores de la media y la varianza de la demanda futura.

Como el problema que estudiaremos en este trabajo es de múltiples pedidos, nos vamos a centrar en este tipo de modelos. Para la realización de los pedidos supondremos que son pequeños en relación con la demanda (fast-movers).

1.2.1. Modelo de revisión continua (r,q)

Para el modelo de revisión continua se usa la política (r,q), donde:

- r, punto de pedido.
- q, cantidad fija de pedido que se realiza cuando la posición del inventario llega a r.

En este modelo tenemos total control del stock en cada momento y podemos pedir unidades cuando sean necesarias.

Existe la posibilidad de ruptura del stock durante el lead-time, ya que cuando observamos que nuestro stock baja de r realizamos el pedido pero puede ocurrir que la demanda durante el lead-time sea mayor que r, por tanto no podremos satisfacerla. De esta manera, nos es interesante la variable X, que representa la demanda durante el lead-time. La cantidad media de demanda esperada durante el lead-time es μ_X, así que si realizamos el pedido cuando tenemos un nivel de stock de r, en media esperamos tener $r-\mu_X$ (stock de seguridad) en stock cuando nos llegue el pedido. Es habitual expresar el stock de seguridad como un factor de la desviación típica de X: $r-\mu_X = k\sigma_X$ (k factor de seguridad). Claramente, a mayor k, menor probabilidad de ruptura del stock pero mayor gasto de almacenamiento.

Realizamos un análisis general del coste. Veamos los diferentes tipos de costes que se ven involucrados:

- c u.m. por u. de producto (no influirá en nuestro problema).
- a u.m. fijo por realizar el pedido (no influirá en nuestro problema).
- h u.m. por u. y u.t. de almacenamiento.
- c_S u.m. fijo por cada unidad no servida a tiempo (escasez, demanda insatisfecha).

En nuestro problema tendremos además un coste por unidad de producto caducado, pues no podrán servirse a nuestros clientes los productos que estén obsoletos. El coste por ciclo viene dado por:

$$\text{coste/ciclo} = a + qc + \left(k\sigma_X + \frac{q}{2}\right)\frac{qh}{\mu_D} + c_S\int_r^\infty (x-r)f_X(x)\mathrm{d}x, \tag{1.1}$$

donde f_X es la función de densidad de la demanda durante el lead time y σ_X su desviación típica. Así, el coste en promedio por u.t. será:

$$\text{coste} = \frac{a\mu_D}{q} + \mu_D c + \left(k\sigma_X + \frac{q}{2}\right)h + \frac{c_S\mu_D}{q}\int_r^\infty (x-r)f_X(x)\mathrm{d}x. \tag{1.2}$$

Se puede consultar en detalle en el Capítulo 3, Subsección 3.2.1 de [10]. Además, en esa sección se indican diversas formas de cómo calcular r y q dependiendo de cuál sea nuestro objetivo (minimizar el coste, garantizar un nivel de servicio, ambas,...).

1.2.2. Modelo de revisión periódica (R,S)

El modelo de revisión periódica es el más interesante para nuestro proyecto. En este caso, o bien, tenemos acceso al nivel de stock sólo tras intervalos de R u.t., o bien, tenemos acceso pero sólo podemos realizar pedidos en intervalos de R u.t. Con la política (R,S), debemos notar que:

- R, es una cantidad que puede estar prefijada o no. Si no está prefijada la tenemos que determinar.
- S, es una cantidad fijada de forma que si, al revisar el stock, el nivel de inventario se encuentra en g, pediremos $S-g$.

De nuevo, nos interesa el coste del modelo. La variable aleatoria importante en este modelo es X_{L+R} se trata de la demanda que tenemos en $L+R$ u.t. El coste esperado por ciclo es:

$$coste/ciclo = a + \mu_D Rc + \left(S - E(X_{L+R}) + \frac{\mu_D R}{2}\right) Rh + c_S \int_S^\infty (x-S) f_X(x) \mathrm{d}x,$$

donde f_X es la función de densidad de la demanda en $L+R$ u.t. y $E(X_{L+R})$ su esperanza. Con ello, el coste esperado por u.t. será:

$$coste = \frac{a}{R} + \mu_D c + \left(S - E(X_{L+R}) \frac{\mu_D R}{2}\right) h + \frac{c_S}{R} \int_S^\infty (x-S) f_X(x) \mathrm{d}x. \tag{1.3}$$

Podemos calcular R y S para minimizar el coste globalmente (derivando (1.3) respecto de R y S e igualando a 0).

1.3. Productos perecederos y revisión bibliográfica

En la sección 1.2 hemos visto algunos de los modelos básicos de aprovisionamiento. Una característica común de ellos es que los productos son no perecederos y, por tanto, pueden mantenerse almacenados indefinidamente.

En cambio, cuando se trabaja con productos perecederos, los modelos matemáticos y su resolución se complica sustancialmente. Esto se debe a que, por una parte, la caducidad implica unos costes por las unidades que hay que desechar y, por otra, que a la hora de controlar el stock hay que tener en cuenta las posibles caducidades. Como norma general, en el caso de productos perecederos, las políticas óptimas no dependen exclusivamente del número de unidades en stock sino de su composición etaria.

A continuación, vamos a hacer un breve resumen de los métodos de resolución de este tipo de problemas.

En los años 70-80, en el comienzo del estudio de la gestión de productos perecederos, se intentó desarrollar una teoría similar a la de los productos no perecederos que, para entonces, ya estaba bien establecida y para la que se conocían soluciones óptimas en un gran número de modelos. Sin embargo, pronto se vieron las dificultades que aparecían y tan solo algunos casos particulares pudieron ser resueltos de manera analítica, como por ejemplo, los casos en que $m = 1, 2$, con m el tiempo de vida útil del producto.

Un posible método de resolver problemas de este tipo de productos, es el hallazgo de expresiones analíticas del tipo 1.3. Es decir, buscar expresiones cuya solución permita tomar una decisión óptima al problema, como en [11], [12], donde en ambos lo realizan para $m = 2$. Para $m \geq 3$, el análisis es mucho más complejo debido a que la variable estado es multidimensional. Así, el estado del sistema se trata de un vector que incluye el número de u. que hay en stock según la edad restante de dichas unidades. Suponiendo que las u. tienen una vida útil de m períodos, el vector de estado mencionado, es de la siguiente manera:

$$\mathbf{x} = (x_1, \ldots, x_{m-1})$$

donde x_i es la cantidad de stock disponible que caduca en i períodos, tal y como se indica en [11]. Aunque se tengan expresiones explícitas de la función objetivo, las expresiones a minimizar son muy complicadas y, en la mayoría de los casos, la resolución ha de realizarse de manera numérica.

Otro posible enfoque, es la resolución mediante simulación de eventos discretos como se ha hecho en [1, 3, 7, 8], entre otros. Esta simulación permite comparar diferentes políticas modelando el sistema e introduciéndolo en un software de simulación. Sin embargo, este método no permite obtener expresiones analíticas de la solución.

Por otro lado, también se puede resolver heurísticamente como se hace en [6, 11]. Los métodos heurísticos no intentan buscar la solución óptima sino una política razonable que sea útil en la práctica.

Por último, otro atractivo método, es el uso de la programación dinámica, como en [8], que nos halla la solución óptima del problema. Este procedimiento consta principalmente de un espacio de

estados, similar al que hemos mencionado en el método analítico, y una función de transferencia. Si bien este método proporciona soluciones óptimas, es de difícil implementación debido al gran número de estados del sistema. Por esta razón, en muchas ocasiones se realiza un reescalamiento del problema para simplificar el espacio de estados.

En este proyecto, en los Capítulos 3 y 4, veremos dos de los métodos mencionados, ya que resolveremos un problema particular de inventario perecedero tanto analíticamente como haciendo uso de la programación dinámica.

Capítulo 2

Planteamiento del problema y objetivo

Nuestro interés de estudio en este trabajo es la sangre. En el momento en que a un donante le extraen sangre comienza un proceso llamado *tipaje* (que identifica el grupo sanguíneo del donante) y, a continuación, un proceso denominado *centrifugado* que permite separar los componentes de la sangre en plaquetas, glóbulos rojos y plasma. Nos inclinamos al estudio de producción de plaquetas, debido a que tienen una vida útil mucho más corta con respecto a los glóbulos rojos y el plasma; 5 ó 7 días, 42 días y 3 años, respectivamente.

Nuestro problema a resolver trata de optimizar la cantidad diaria de concentrados de plaquetas a producir por un Banco de Sangre. En concreto, vamos a trabajar con el esquema de funcionamiento del Centro Vasco de Transfusión y Tejidos Humanos (CVTTH); ver [6]. La decisión a tomar sobre la cantidad diaria de u. que hay que producir depende de diversos factores.

En primer lugar, a primera hora de la mañana de un día, a la vista de las u. que hay en stock, debemos tomar una decisión, sobre cuántas unidades producir. Estas unidades se reciben después de un tiempo de entrega del pedido fijado en 1 período para todos los días, excepto los viernes que es de 3 períodos (los períodos en nuestro problema representarán los días). Es decir, las unidades que decidamos producir el viernes llegan al inventario el siguiente lunes. Por tanto, todas las u. producidas se colocan en stock a la mañana siguiente y llegan con 5 días de vida útil restantes, excepto los lunes que llegan con 3 días de vida útil restantes.

Es importante mencionar que no habrá que tomar una decisión para el fin de semana, ya que vamos a considerar que los sábados y domingos no se produce debido a la baja y nula producción, respectivamente, de estos dos días en la vida real.

En segundo lugar, para nuestro problema seguiremos una política de emisión FIFO (*First In First Out*), es decir, primero enviaremos las u. más antiguas. A pesar de que, obviamente, la demanda tiene una distribución discreta, podemos elegir una distribución continua y discretizarla como se hace en un gran número de artículos. La demanda diaria en el CVTTH se ajusta bien a una distribución normal (discretizada), como se menciona en [6]. Así, en este trabajo también emplearemos dicha distribución.Veamos en la siguiente tabla cuáles serán las medias (μ) y desviaciones típicas (σ) de las demandas para cada día de la semana:

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
$\mu = 27{,}75$	$\mu = 23{,}71$	$\mu = 24{,}57$	$\mu = 22{,}16$	$\mu = 29{,}39$	$\mu = 13{,}29$	$\mu = 11{,}82$
$\sigma = 6{,}85$	$\sigma = 5{,}65$	$\sigma = 7{,}86$	$\sigma = 6{,}90$	$\sigma = 7{,}81$	$\sigma = 4{,}89$	$\sigma = 4{,}38$

Cuadro 2.1: Tabla de medias y desviaciones típicas

Asimismo, supondremos que las demandas en distintos días son variables aleatorias independientes. Toda la demanda que no se pueda servir a tiempo (escasez de demanda) tendrá un coste, p, asociado a ella ya que necesitaremos pedir las unidades que nos falten a otro banco de sangre.

En tercer lugar, las unidades que al inicio del día le quedaban 1 día de vida restante y que hayan

sobrado al terminar el día, caducan y, por tanto, se desechan al finalizar dicho día. Las unidades descartadas también tendrán un coste asociado por unidad, θ.

En cuarto y último lugar, también tendremos un coste de almacenamiento, h. Este coste se genera por unidad que pasa la noche en el almacén (u. en stock). Es importante recalcar que las unidades que pedimos un día no tienen coste de almacenamiento ese día, ya que no llegan hasta el inicio de la mañana siguiente, por tanto, no han pasado la noche en el almacén.

Fijamos, para este problema de producción de plaquetas, todos los costes que hemos mencionado:

- $h = 2$ u.m.
- $p = 5000$ u.m.
- $\theta = 400$ u.m.

La finalidad en los capítulos siguientes, será encontrar la solución óptima del problema de producción de plaquetas. Para ello emplearemos dos métodos, resolución analítica (3) y resolución mediante programación dinámica(4).

Antes de resolverlo por el primer método, se ha necesitado realiza un intenso estudio de un artículo reciente sobre resolución analítica de productos perecederos con $m = 2$ períodos de vida útil del producto [12]. En el presente trabajo de fin de grado, hemos adaptado las técnicas de este artículo (desarrollado para un período corto de caducidad, $m = 2$) a nuestra situación, con $m = 5$. Debido a las dificultades de la resolución analítica de las expresiones que obtendremos, ha sido menester la creación de un programa en C (puede consultarse en Anexo B.1) que nos devuelve la decisión óptima de unidades a producir de concentrados de plaquetas para cada día de la semana.

El segundo método de resolución que vamos a aplicar es el de programación dinámica. Para este método también se ha necesitado crear un programa en C (Anexo B.2) que nos decidirá las unidades que debemos ordenar según el día de la semana.

Cabe destacar que, hasta la actualidad, no se ha resuelto este problema particular de producción de plaquetas mediante los métodos anteriores, y de ahí la novedad de este proyecto.

Capítulo 3

Un modelo de inventario perecedero con tiempo de entrega del pedido positivo

En este capítulo, explicaremos abreviadamente la resolución de un problema de inventario perecedero con tiempo de vida del producto de $m = 2$ períodos extraído del artículo [12]. Posteriormente, adaptaremos dicho artículo al problema planteado en el Capítulo 2.

3.1. Artículo de referencia

El artículo [12], trata las ecuaciones necesarias para determinar la política óptima de revisión periódica de un producto con una vida útil de dos períodos, sujeto a un tiempo de entrega del pedido positivo. En dicho artículo, se calculan las cantidades de pedido óptimas para tiempos de entrega de hasta cuatro períodos para diferentes niveles y distribuciones de la demanda.

Vamos a mencionar solamente las hipótesis que son distintas a nuestro problema planteado del Capítulo 2. El modelo de [12], supone que todos los pedidos se reciben después de un tiempo de entrega fijado de L períodos y todo el stock llega nuevo con m períodos de vida útil restantes.

3.1.1. Notación

Veamos algunas notaciones útiles para entender los resultados que aparecen en el artículo [12] y las ecuaciones.

Los costes serán los mencionados en el Capítulo 2, pero no se les asigna un valor concreto. Además, aparece un nuevo coste por unidad de producto producida. Es decir,

- c u.m. por u. de producto.
- h u.m. por u. y u.t. de almacenamiento, mantenimiento de inventario.
- p u.m. fijo por u. de demanda insatisfecha, escasez de inventario.
- θ u.m. por u. caducada.

La notación usada será:

- D_k demanda en el período k.
- $G_k(\cdot)$ función de distribución de D_k.
- $g_k(\cdot)$ función de densidad de D_k.
- S_k la escasez de inventario, demanda insatisfecha en el período k.

- X_k^r el inventario inicial con r períodos de vida restante en el período k. Como $m = 2$ y el inventario se compone por $X_k^1, \ldots, X_k^{m-1}$ (es decir, inventario con 1 período de vida restante,..., inventario con $m-1$ períodos de vida útil restante), entonces X_k^r será X_k^1.
- $F_k(\cdot)$ función de densidad del inventario inicial en el período k con un período de vida restante, es decir, de X_k^1.
- $f_k(\cdot)$ función de densidad de X_k^1.
- a_k exceso de demanda, $[D_k - X_k^1]^+$ con $g^+ = \max(0, g)$, en el período k.
- $q_k(\cdot)$ función de densidad de $[D_k - X_k^1]^+$ en el período k.
- y_k cantidad de pedido, unidades a producir, al inicio del período $k-L$ que llegan al inicio del período k, con L el tiempo de entrega del pedido .
- O_k^{k+m-1} cantidad de y_k que caducará al final del período $k+m-1$. Como $m = 2$, esto será O_k^{k+1}.

Una vez denotado lo que nos puede ser útil, veamos cuál es la resolución de este problema de producto perecedero con una vida útil de dos períodos ($m = 2$).

3.1.2. Problema de tiempo de vida de dos períodos

Con $m = 2$, el objetivo es determinar la decisión óptima, y_k, que minimiza la función de coste total esperado siguiente:

$$E[TC_k] = c \cdot y_k + h \cdot E[X_{k+1}^1] + p \cdot E[S_k] + \theta \cdot E[O_k^{k+1}] \tag{3.1}$$

donde c, h, p y θ son constantes definidas en 3.1.1; y_k son las unidades de pedido del período $k-L$ que llegan en el período k; X_{k+1} es el inventario inicial en el período $k+1$ con un período de vida restante, es decir, de las u. que llegan en el período k, y_k, las que duermen en el almacén la noche k llegan al período $k+1$ con un período de vida restante; S_k, representa la demanda insatisfecha del período k, unidades que no han podido ser servidas; y, por último, O_k^{k+1}, de las unidades que llegan en el período k cuántas caducan al final del período $k+1$, es decir, no se agotan en dicho período.

El inventario inicial en el período $k+1$ con un período de vida restante está dado por

$$X_{k+1}^1 = [y_k - (D_k - X_k^1)^+]^+$$

recordamos que $g^+ = \max\{0, g\}$. La escasez de inventario en el período k es

$$S_k = [D_k - y_k - X_k^1]^+,$$

y la cantidad de pedido en el período k que caducará al final del período $k+1$ está dado por

$$O_K^{k+1} = [y_k - D_{k+1} - (D_k - X_k^1)^+]^+.$$

Consideran que la función objetivo resultante es muy compleja, por lo que buscan presentar la ecuación de forma que sea más fácil de calcular. Para ello recurren a la función de densidad de X_k^1 para $k = 1, 2, \ldots$, es decir, $f_k(x_k^1)$, y la resuelven iterativamente a través del siguiente teorema.

Teorema 3.1.1. *La función de densidad para el inventario inicial, X_k^1 para $k = 1, 2, 3, \ldots$, está dada por*

- *Para* $t = 1$,

$$f_1(x_1^1) = \begin{cases} 1 - G_0(X_0^1 + y_0) & si\ x_1^1 = 0, \\ g_0(X_0^1 + y_0 - x_1^1) & si\ 0 \le x_1^1 \le X_0^1 + y_0, \\ 0 & en\ otro\ caso. \end{cases}$$

- *Para* $t = 2,3,4,\ldots,$

$$f_t(x_t^1) = \begin{cases} [1-G_{t-1}(y_{t-1})]\cdot f_{t-1}(0) + \int_{x_{t-1}^1=0}^{y_{t-2}} [1-G_{t-1}(y_{t-1}+x_{t-1}^1)]\cdot f_{t-1}(x_{t-1}^1)\mathrm{d}x_{t-1}^1 & \textit{si } x_t^1 = 0, \\ g_{t-1}(y_{t-1}-x_t^1)\cdot f_{t-1}(0) + \int_{x_{t-1}^1=0}^{y_{t-2}} [g_{t-1}(y_{t-1}+x_{t-1}^1-x_t 1)]\cdot f_{t-1}(x_{t-1}^1)\mathrm{d}x_{t-1}^1 & \textit{si } 0 \le x_t^1 \le y_{t-1}, \\ 0 & \textit{en otro caso.} \end{cases}$$

Es aquí donde se encuentra el primer inconveniente, ya que en el artículo no está demostrado el teorema y no es del todo correcto. Se puede consultar la modificación y demostración en Anexo A. Seguidamente, mediante esperanzas condicionadas aplican el teorema anterior para hallar la esperanza del inventario en el período k y la escasez de inventario. Para determinar el valor esperado de la caducidad, se requiere otro resultado (Teorema 3.2 de [12]), en el que no vamos a entrar en detalle por extensión del trabajo.

Una vez se tiene $E[X_{k+1}^1]$, $E[S_k]$ y $E[O_k^{k+1}]$ en términos de funciones de densidad, se sustituyen en 3.1, y así el coste total esperado en el período k está dado por

$$\begin{aligned} E[TC_k] = {} & c\cdot y_k \\ & + h\cdot\left(\int_{z_k=0}^{y_k}(y_k-z_k)g_k(z_k\mathrm{d}z_k)\cdot f_k(0) + \int_{x_k^1=0}^{y_{k-1}}\int_{z_k=x_k^1}^{x_k^1+y_k}(y_k+x_k^1-z_k)g_k(z_k)\mathrm{d}z_k\cdot f_k(x_k^1)\mathrm{d}x_k^1\right) \\ & + p\cdot\left(\int_{z_k=y_k}^{\infty}(z_k-y_k)g_k(z_k)\mathrm{d}z_k\cdot f_k(0) + \int_{x_k^1=0}^{y_{k-1}}\int_{z_k=y_k+x_k^1}^{\infty}(z_k-y_k-x_k^1)g_k(z_k)\mathrm{d}z_k\cdot f_k(x_k^1)\mathrm{d}x_k^1\right) \\ & + \theta\cdot\left(\int_{z_{k+1}=0}^{y_k}(y_k-z_{k+1})g_{k+1}(z_{k+1})\mathrm{d}z_{k+1}\cdot q_k(0)\right) \\ & + \theta\cdot\left(\int_{a_k=0}^{\infty}\int_{z_{k+1}=0}^{y_k-a_k}(y_k-a_k-z_{k+1})g_{k+1}(z_{k+1})\mathrm{d}z_{k+1}\cdot q_k(a_k)\mathrm{d}a_k\right) \end{aligned} \tag{3.2}$$

A continuación, realizan la primera derivada con respecto a y_k y la igualan a cero. Para ello, aplican la regla de Leibniz y después comprueban que efectivamente la función 3.1 es convexa.

Finalmente, para determinar la cantidad de pedido que minimiza la función coste de un solo período, la calculan numéricamente mediante el método simple de Newton-Raphson.

3.2. Problema de producción de plaquetas (problema de tiempo de vida de cinco períodos)

Para nuestro modelo, usaremos los costes definidos en Capítulo 2 y la notación de la sección anterior. Recordamos las características más relevantes, ya explicadas, algunas de ellas, en el Capítulo 2.

Denotaremos los días de la semana por sus iniciales, $L,M,\ldots,D$ de esta manera L representará el lunes, M el martes, así sucesivamente hasta D que representará el domingo. Por otro lado, el inventario inicial total en el período k consiste en la cantidad de artículos dependientes de la edad, es decir, $X_k^1,\ldots,X_k^{m-1}$, como los concentrados de plaquetas durarán $m = 5$ días, tendremos $X_k^1,\ldots,X_k^4$ (es decir, inventario con 1 período de vida útil restante, ..., inventario con 4 días de vida útil restante, respectivamente).

La cantidad de pedido y_k (realizado en el período $k-1$ para todos los días de la semana excepto para el lunes que será el pedido del viernes) será utilizada para satisfacer las demandas que comienzan en el período k y terminan en el período $k+4$, es decir, cantidad que pedimos el lunes y llega el martes (M), y_M, será útil hasta 5 días después desde que se piden, es decir, hasta el sábado. Las unidades de y_k que no se hayan utilizado al final del período $(k+1)+m-1$, es decir, $k+5$, caducarán.

Veamos entonces las funciones a optimizar utilizando de referencia la ecuación 3.1 pero adaptada a nuestro problema. Antes de ello, observaremos que el coste c que en el artículo [12] consideran por

tomar una decisión, es decir, por ordenar una cantidad de producto, no estará incluido en nuestras funciones (pues hay que satisfacer una demanda). Esto se debe a que si consideramos que la caducidad de una unidad significa su pérdida, no es sensato incluir el pago de la unidad dos veces (en compra y caducidad). De esta manera, escribiremos a continuación del día de la semana el coste que genera tomar esa decisión ese día, por ejemplo, para el lunes, se tendrá a su derecha el coste que genera tomar la decisión y_M. Así, las funciones objetivos son las siguientes:

1. Lunes:

$$E[TC_M] = h \cdot E[X_X^4 + X_J^3 + X_V^2 + X_S^1] + p \cdot E[S_M] + \theta \cdot E[O_M^S] \tag{3.3}$$

2. Martes:

$$E[TC_X] = h \cdot E[X_J^4 + X_V^3 + X_S^2 + X_D^1] + p \cdot E[S_X] + \theta \cdot E[O_X^D]$$

3. Miércoles:

$$E[TC_J] = h \cdot E[X_V^4 + X_S^3 + X_D^2 + X_L^1] + p \cdot E[S_J] + \theta \cdot E[O_J^L]$$

4. Jueves:

$$E[TC_V] = h \cdot E[X_S^4 + X_D^3 + X_L^2 + X_M^1] + p \cdot E[S_V + S_S + S_D] + \theta \cdot E[O_V^M] \tag{3.4}$$

5. Viernes:

$$E[TC_L] = h \cdot E[X_M^2 + X_X^1] + p \cdot E[S_L] + \theta \cdot E[O_L^X]$$

Notemos que la cantidad a pedir, y_k, variable de optimización, no aparece de manera explícita en la función objetivo, pero está incluida en las variables $X_k^r,\ S_k,\ O_k^{k+4}$.

Analicemos algunas de las funciones a optimizar anteriores. Comencemos con $E[TC_M]$ (3.3), en la que hay que decidir y_M, la cantidad de pedido que se hace el lunes. Esta decisión se toma a primera hora de la mañana del lunes y los datos de los que se dispone son X_L^1 (cantidad de stock en la mañana del lunes con un día de vida restante), X_L^2 (cantidad de stock al que le quedan 2 días de vida restantes en la mañana del lunes) e y_L (cantidad de pedido que se ordenó el viernes y acaba de llegar con 3 días de vida restantes). Como se observa en la función 3.3, las variables que aparecen no son directamente de las que disponemos sino funciones suyas. Debido a esto, tal como se hace en el artículo [12], hay que establecer unas fórmulas recursivas que relacionen $X_X^4, X_J^3, X_V^2, X_S^1, S_M$ y O_M^S con X_L^1, X_L^2 e y_L y, por supuesto, algunas cantidades aleatorias relacionadas con la demanda entre el lunes y el sábado. Veamos las relaciones existentes entre las variables X_k^r, S_k y O_k^{k+4}.

Como vamos a decidir sobre las u. a producir del lunes, el inventario inicial en el período $k+1$ actúa de forma diferente al resto de días de la semana. Esto se debe a que, como se ha dicho anteriormente, el fin de semana no se produce. De esta manera tenemos:

- Unidades en inventario en la etapa $k+1 = M$ con 1 día de vida restante:

$$X_M^1 = \begin{cases} X_L^2 & \text{si } X_L^1 \geq D_L, \\ X_L^2 - [D_L - X_L^1] & \text{si } X_L^1 < D_L < X_L^1 + X_L^2, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.5}$$

- Unidades en inventario en la etapa $k+1 = M$ con 2 días de vida restantes:

$$X_M^2 = \begin{cases} y_L & \text{si } X_L^1 + X_L^2 \geq D_L, \\ y_L - [D_L - (X_L^1 + X_L^2)] & \text{si } X_L^1 + X_L^2 < D_L < X_L^1 + X_L^2 + y_L, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.6}$$

Por otro lado, la escasez de inventario en el período $k+1$ está dado por:

$$S_{k+1} = \begin{cases} D_{k+1} - y_{k+1} - X_k & \text{si } D_{k+1} > y_{k+1} + X_k \\ 0 & \text{en otro caso} \end{cases} \tag{3.7}$$

donde $X_k = X_k^1 + X_K^2 + X_k^3 + X_k^4$.

Por último, hemos hallado la ecuación de caducidad que representa la cantidad de pedido del período k que llega en el período $k+1$ y que caducará al final del período $k+5$, y está dada por:

$$O_{k+1}^{k+5} = [X_{k+5}^1 - D_{k+5}]^+ \tag{3.8}$$

Para los demás días de la semana, las ecuaciones de recursividad son algo distintas y son ciertas solo si existen, ya que en algunos días de la semana no son posibles algunas de ellas. Esto se debe a que los sábados y domingos no hay producción. Por ejemplo, los martes no habrá unidades en inventario con 3 días de vida restantes, pues serían las correspondientes a las producidas el sábado (día de no producción), ni habrá u. de 4 días de vida restantes, ya que corresponderían con u. producidas el domingo (que no es posible). y están determinadas por Exceptuando los casos que correspondan a la explicación previa, el inventario inicial en el período $k+1$ con $1,2,3$, y 4 períodos de vida restante está dado por las siguientes ecuaciones.

- Unidades en inventario en la etapa $k+1$ con 1 día de vida restante:

$$X_{k+1}^1 = \begin{cases} X_k^2 & \text{si } X_k^1 \geq D_k, \\ X_k^2 - [D_k - X_k^1] & \text{si } X_k^1 < D_k < X_k^1 + X_K^2, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.9}$$

- Unidades en inventario en la etapa $k+1$ con 2 días de vida restantes:

$$X_{k+1}^2 = \begin{cases} X_k^3 & \text{si } X_k^1 + X_k^2 \geq D_k, \\ X_k^3 - [D_k - (X_k^1 + X_k^2)] & \text{si } X_k^1 + X_k^2 < D_k < X_k^1 + X_K^2 + X_k^3, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.10}$$

- Unidades en inventario en la etapa $k+1$ con 3 días de vida restantes.

$$X_{k+1}^3 = \begin{cases} X_k^4 & \text{si } X_k^1 + X_k^2 + X_k^3 \geq D_k, \\ X_k^4 - [D_k - (X_k^1 + X_k^2 + X_k^3)] & \text{si } X_k^1 + X_k^2 + X_k^3 < D_k < X_k^1 + X_K^2 + X_k^3 + X_k^4, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.11}$$

- Unidades en inventario en la etapa $k+1$ con 4 días de vida restantes:

$$X_{k+1}^4 = \begin{cases} y_k & \text{si } X_k^1 + X_k^2 + X_k^3 + X_k^4 \geq D_k, \\ y_k - [D_k - (X_k^1 + X_k^2 + X_k^3 + X_k^4)] & \text{si } X_k^1 + X_k^2 + X_k^3 + X_k^4 < D_k < X_k^1 + X_K^2 + X_k^3 + X_k^4 + y_k, \\ 0 & \text{en otro caso.} \end{cases} \tag{3.12}$$

Sin embargo, las ecuaciones de escasez y caducidad para el resto de días de la semana si que coinciden con 3.7 y 3.8.

Con lo anterior, las ecuaciones recursivas 3.5 y 3.6, se pueden expandir llegando a las siguientes ecuaciones en las que se observa la complejidad existente en hallar una solución analítica a nuestro problema.

- **Lunes:**

$$
\begin{aligned}
& y_M \qquad (3.13)\\
& X_X^4 = [y_M - [D_M - [(X_L^2 - [D_L - X_L^1]^+)^+ + (y_L - [D_L - (X_L^1 + X_L^2)]^+)^+]]^+]^+\\
& X_J^3 = [X_X^4 - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]^+\\
& \quad = [(y_M - [D_M - [(X_L^2 - [D_L - X_L^1]^+)^+ + (y_L - [D_L - (X_L^1 + X_L^2)]^+)^+]]^+)^+\\
& \qquad - [D_X - ([y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ - [D_M - [X_L^2 - [D_L - X_L^1]^+]^+]^+)]^+]^+\\
& X_V^2 = [[(y_M - [D_M - [(X_L^2 - [D_L - X_L^1]^+)^+ + (y_L - [D_L - (X_L^1 + X_L^2)]^+)^+]]^+)^+\\
& \qquad - [D_X - ([y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ - [D_M - [X_L^2 - [D_L - X_L^1]^+]^+]^+)]^+]^+ - D_J]^+\\
& X_S^1 = [([[(y_M - [D_M - [(X_L^2 - [D_L - X_L^1]^+)^+ + (y_L - [D_L - (X_L^1 + X_L^2)]^+)^+]]^+)^+\\
& \qquad - [D_X - ([y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ - [D_M - [X_L^2 - [D_L - X_L^1]^+]^+]^+)]^+]^+ - D_J)^+ - D_V]^+
\end{aligned}
$$

así:

$$
\begin{aligned}
O_M^S = & [[([(y_M - [D_M - [(X_L^2 - [D_L - X_L^1]^+)^+ + (y_L - [D_L - (X_L^1 + X_L^2)]^+)^+]]^+)^+\\
& - [D_X - ([y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ - [D_M - [X_L^2 - [D_L - X_L^1]^+]^+]^+)]^+]^+ - D_J)^+ - D_V]^+ - D_S]^+
\end{aligned}
$$

En efecto, se ve que para decidir y_M el coste 3.3 puede escribirse como función de X_L^1, X_L^2 e y_L. Las esperanzas que aparecen en este coste han de tomarse respecto a las variables $D_L, \ldots, D_S$ correspondientes a las demandas aleatorias de lunes a sábado.

Al comparar estas ecuaciones con las del artículo [12], vemos que en nuestro caso son mucho más complejas debido a los 5 días de vida (en vez de los 2 días en el artículo [12]). Optimizar de manera analítica la función anterior (calculando la demanda respecto de y_M) es inabordable, ya que en las expresiones de la esperanza aparecen integrales iteradas (hasta 6 veces) con límites de integración complicados, debido a las partes positivas, por lo que no se puede encontrar una solución analítica de $\frac{\mathrm{d}E[TC_M]}{\mathrm{d}y_M} = 0$. Esto era esperable, ya que este tipo de dificultad se ha encontrado en la literatura cuando se han buscado soluciones analíticas en el caso de $m \neq 1, 2$.

Para resolver el problema, dado que y_M es discreto, lo hemos realizado de manera directa, evaluando la función $E[TC_M]$ para cada valor de y_M y eligiendo el menor coste.

Este cálculo es muy costoso computacionalmente, porque las expresiones de la esperanza, al tratarse de relaciones no lineales, no pueden simplificarse. Por tanto para evaluar cada una de ellas, hemos recorrido todos los posibles valores de $D_L, \ldots, D_S$, calculando el valor de la función para esos valores y multiplicando por sus probabilidades.

No vamos a estudiar todos los casos de los costes esperados a detalle, como hemos hecho con 3.3, pero sí vamos a analizar alguna diferencia que se puede observar en 3.4 con respecto a las otras funciones objetivo. En este sentido, debemos destacar que en la función coste que genera la decisión óptima a tomar el jueves, dado por 3.4, aparece la esperanza de la escasez de demanda no solo del viernes, sino también de la del sábado y domingo. Este hecho se debe a que la decisión que se tome el jueves, nos debe cubrir las demandas de los tres días (viernes, sábado y domingo), ya que las u. que se deciden producir el viernes, no llegan al stock hasta el lunes siguiente, y el fin de semana (sábado y domingo) no hay producción.

Para el resto de días de la semana, el problema de optimización es análogo al que hemos realizado para el lunes pero con expresiones más complicadas, por lo que las añadimos en Anexo A debido a la extensión del trabajo.

3.2.1. Resultados experimentales

Una vez que tenemos todos los resultados anteriores, ya podemos crear un programa con las nociones del artículo pero adaptado a nuestro problema de producción de plaquetas.

De esta manera, el programa creado en C (puede verse el código en el Anexo B.1) nos muestra por pantalla el número de unidades que debemos producir de forma que minimicemos costes dependiendo

del día de la semana en que estemos y de las unidades existentes en el almacén. Para ello, ha sido necesario crear unos vectores con las probabilidades correspondientes (con media y desviación típica dadas en 2.1) dependiendo del día de la semana.

Al crear el programa, nos hemos encontrado con la necesidad de agilizar el proceso de computación, ya que si fuera llevado este programa a un Banco de Sangre, no tendría sentido que al inicio de la jornada laboral ejecutaran el programa y tuvieran que esperar demasiado tiempo para conocer las unidades que deben producir ese día.

Para ello, hemos recorrido las posibles demandas de 2 en 2. Además, se les ha establecido un mínimo $\mu - 2{,}5\sigma$, y un máximo $\mu + 4\sigma$, donde μ representa la media y σ la desviación típica.

Con respecto a las decisiones, finalmente también las hemos recorrido de 2 en 2; de esta manera tomar la decisión óptima tiene un tiempo de ejecución en torno a 5 minutos en un HP Pavilion 15 con CPU AMD A8-6410 APU. También hemos probado a recorrer las posibles decisiones de 1 en 1, que obviamente le costaba el doble tomar la decisión óptima. También le hemos puesto un límite de producción, como máximo se pueden producir 60 u. diarias.

Al ejecutar el programa, nos preguntará por pantalla qué día es y las unidades que tenemos según tengan 1, 2, 3 ó 4 días de vida restante. Además de pedirnos las unidades que llegan nuevas ese día al almacén.

Con todo ello, realizamos representaciones gráficas donde se puede observar cómo depende la decisión óptima a tomar según el stock existente y el día de la semana en el que estamos. En cada gráfica representaremos solo tres escenarios posibles para poder visualizarlas con mayor precisión. Además, en cada gráfica aparecen escenificadas distintas rectas en diversos colores. La diferencia de colores simboliza:

Día de la semana (unidades de plaquetas dependiendo de los días que les quedan de vida)

Por ejemplo, si:

Martes (1 día de vida)

se tiene que la recta de color rojo representa el día de la semana martes cuando a todas las unidades de plaquetas que tenemos en el almacén les quedan un día de vida. También puede darse el siguiente caso en que haya dos especificaciones en el mismo paréntesis. Ponemos un pequeño ejemplo, si:

Martes (1 y 2 días de vida)

entonces la recta de color azul representa el día de la semana martes si todas las u. tienen, equitativamente repartidas, uno y dos días de vida restantes.

Gráficas según día de la semana

En primer lugar, vemos la dependencia del stock total con la decisión óptima al comienzo del lunes.

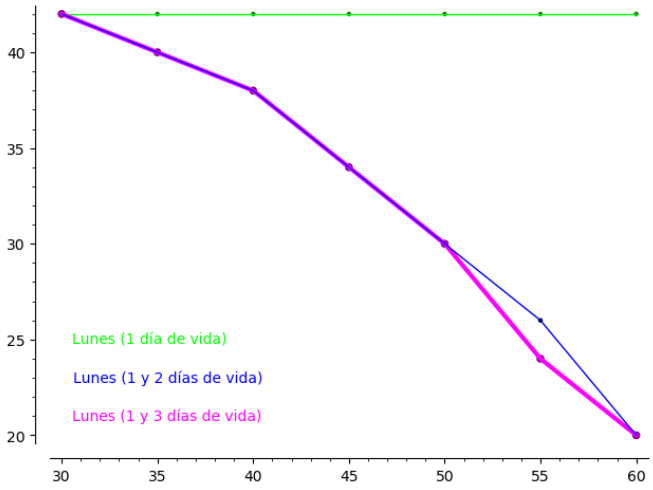

Figura 3.1: **Decisión óptima de los lunes en función de las u. en stock**

Eje x: suma stock total ‖ Eje y: unidades a producir.

Observamos que si todas las unidades que están en el almacén tienen solo 1 día de vida (ver recta verde), la decisión óptima siempre es la misma, producir 42 unidades. Esto se debe a que si todas las unidades tienen un día de vida restante, quiere decir que todas las unidades que no se hayan agotado hoy, caducan al finalizar el día, por tanto, tenemos 0 unidades en el almacén para cubrir todas las demandas posteriores a hoy.

Estas explicaciones son válidas para el resto de días de la semana como podemos observar en las gráficas siguientes. Aunque, por supuesto, adaptando para cada día el valor de la decisión óptima.

En segundo lugar, observamos la evolución para los martes.

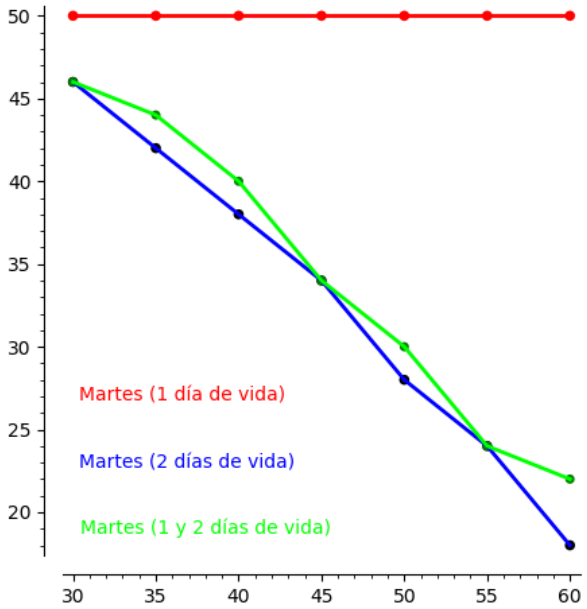

Figura 3.3: **Decisión óptima de los martes en función de las u. en stock**

Eje x: suma stock total ‖ Eje y: unidades a producir.

En tercer lugar, si el día de la semana es miércoles.

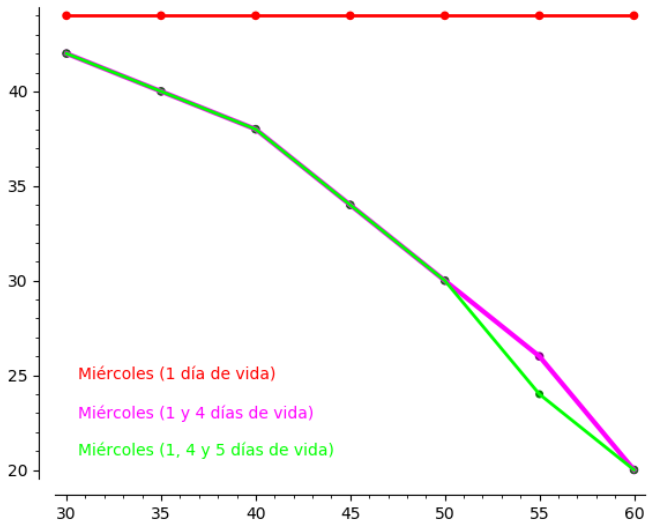

Figura 3.5: **Decisión óptima de los miércoles en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

En cuarto lugar, para los jueves.

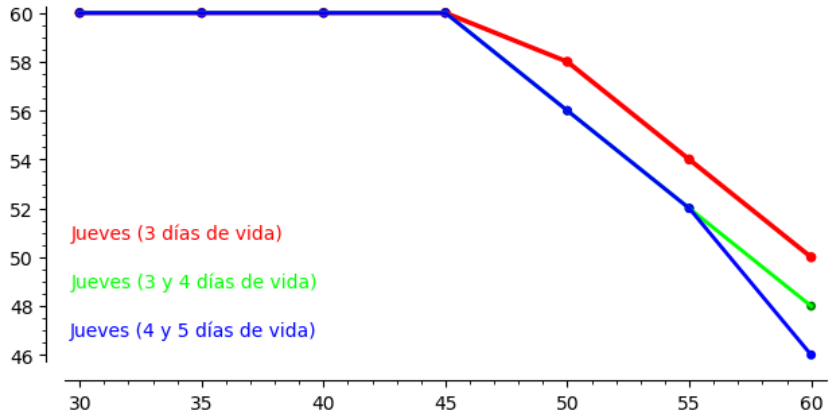

Figura 3.7: **Decisión óptima de los jueves en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

Notar que, cuando es jueves, no existe la línea recta como en el resto de días de la semana ya que no es posible que haya unidades con 1 día de vida restante los jueves, pues supondría que el sábado se ha tomado una decisión, y como hemos mencionado anteriormente, esto no es posible.

Por último, representamos la correspondiente al viernes. Para esta gráfica, tomamos un eje X cuyo máximo tenga mayor valor que las gráficas anteriores porque al principio de este día es esperable un mayor volumen de unidades almacenadas para hacer frente al fin de semana en el que no hay producción.

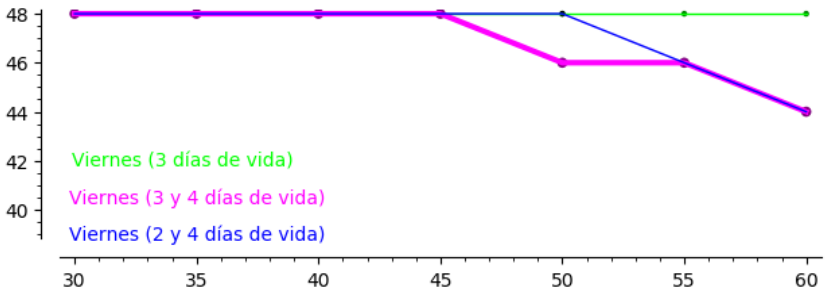

Figura 3.9: **Decisión óptima de los viernes en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

Observamos que, en este caso, la línea recta que en todos los días representa el caso en que las unidades de stock solo tienen un día de vida restante, los viernes no representa lo mismo. Aquí, la recta verde representa los casos en que todas u. de stock tienen 3 días de vida. Recordamos, que los viernes no es posible que haya u. con 1 día de vida restante, ya que los domingos no se produce.

Para concluir este capítulo, siguiendo el artículo [12], se puede observar que la distribución del stock no es muy importante, es decir, es irrelevante cuántos días de vida les quede a las u. de plaquetas que tenemos en el almacén. Excepto en casos triviales en los que las unidades de concentrados de plaquetas están todos distribuidos en 1 día de vida restante. En ese caso, se agota todo el primer día ya que lo que no se agota debido a la demanda, caduca y tenemos que empezar de nuevo en el almacén. Esto es ligeramente diferente el jueves, donde sí que parece que la composición por edades del stock debería influir en la cantidad a pedir.

Capítulo 4

Programación dinámica

La notación utilizada en este capítulo es una combinación de notaciones de [4], [5] y [12]. Todas ellas toman parte de la notación de [11].

4.1. Introducción

La programación dinámica es la colección de herramientas matemáticas que ofrece un procedimiento metódico para determinar la combinación de decisiones que o bien maximiza el beneficio total o bien minimiza el coste total, dependiendo de nuestros intereses, en problemas en los que ha de tomarse un conjunto de decisiones secuencialmente. A diferencia de la programación lineal u otros tipos de problemas de optimización, no existe un planteamiento matemático estándar del problema de programación dinámica. Ésta trata el problema como si estuviera formado por una sucesión de problemas, cada uno de los cuales tuviera solo una o pocas variables aunque puede contener el problema inicial una gran cantidad de variables interrelacionadas. El decisor trata de identificar una sucesión de decisiones, llamada *política*, que optimiza el comportamiento del sistema a lo largo de su horizonte finito de planificación. Los problemas para resolver mediante programación dinámica pueden ser deterministas o estocásticos, como es nuestro caso.

Es necesario definir algunos conceptos previos.

Definición 4.1.1. Una *etapa* es cada periodo o sección en la que se divide el problema, de forma que se ha de tomar una decisión en cada etapa.

Definición 4.1.2. Un *estado* es toda la información necesaria en cualquier etapa para poder tomar una decisión óptima.

Veamos las principales características así como el planteamiento para un problema de programación dinámica. Nos centramos solo en programación dinámica estocástica pues es la que nos es útil:

- El sistema evoluciona a lo largo de un cierto número de etapas.
- Cada etapa está ligada a un número de estados.
- La decisión tomada en cualquier etapa describe cómo se transforma el estado en la etapa actual en el estado en la siguiente etapa. En nuestro caso se trata de programación dinámica estocástica, por tanto, el estado actual y la decisión tomada solo determinan la distribución de probabilidad del estado en la etapa siguiente.
- Cuando se toma una decisión se obtiene un pago (*beneficio* o *costo*) que depende del estado en el que estaba el sistema, la decisión tomada y, en el caso estocástico, de la aleatoriedad.
- El propósito del problema es optimizar alguna función del estado inicial y de la decisiones siguientes.

- **Principio de optimalidad.** Dado el estado actual, la decisión óptima para cada una de las restantes etapas no depende de los estados en los que se ha estado previamente o de las decisiones tomadas anteriormente. Solo depende dicha decisión del estado actual.

La notación empleada no será la utilizada en [4], por comodidad con el estudio realizado del artículo [12] emplearemos la misma notación que se emplea en dicho artículo. Veámoslo:

- N, es el número de etapas. Indica el horizonte de planificación. Usaremos k como índice de las etapas.
- $X_k,\ k = 0,\ \ldots,\ N-1$, estado. Al comienzo del horizonte de planificación, el estado del sistema será X_0.
- $y_k,\ k = 0,\ \ldots,\ N-1$, denota la variable de decisión. Siendo el estado del sistema X_k, y_k se ha de seleccionar en la etapa $k-1$. El conjunto de decisiones del sistema, $Y_k(X_k)$, dependerá del estado del sistema.
- $D_k,\ k = 0,\ \ldots,\ N-1$, denota la aleatoriedad en la etapa k. Representa el parámetro aleatorio que afecta la transición entre estados y los pagos generados, en nuestro caso será la demanda.
- $X_{k+1} = f_k(X_k, y_k, D_k),\ k = 0,\ \ldots,\ N-1$, se llama ecuación de transferencia e indica cómo se transforma el estado actual X_k por efecto de la decisión y_k y de la componente aleatoria D_k en el estado en la etapa siguiente X_{k+1}. Como nuestra ecuación de transferencia será un vector que depende de varias funciones, la denominaremos *función de transferencia*.
- $g_k(X_k, y_k, D_k),\ k = 0,\ \ldots,\ N-1$, pago en una etapa. Este pago indica el beneficio o costo que el sistema genera en una etapa por estar en el estado actual X_k y tomar la decisión y_k, y por el efecto de la componente aleatoria D_k.
- $g_N(X_N)$, denota el pago terminal en el que se puede incurrir por finalizar el proceso.
- $J_k(X_k),\ k = 0,\ \ldots,\ N-1$ es el valor óptimo de la función objetivo para un problema con $N-k$ etapas que comienza en X_k en la etapa k.
- $J^*(X_0)$, denota el valor óptimo de la función objetivo que depende de las condiciones en las que estaba el sistema al comienzo del horizonte de planificación.
- $y_k^*(X_k) = y_k^*,\ k = 0,\ \ldots,\ N-1$, denota la decisión óptima que ha de tomarse en cada etapa, es decir, optimiza $J_k(X_k)$ para cada X_k y k. Con ello la *política óptima* será un plan de decisiones óptimas para cada uno de los estados X_k en los que puede estar el sistema en cada etapa. Es decir, la política óptima será: $y_0^*, y_1^*,\ \ldots,\ y_{N-1}^*$.

La función objetivo tiene como finalidad minimizar (o maximizar) el valor esperado de la suma de los pagos generados en cada etapa y por finalizar el proceso, es decir:

$$\min_{\{y_0,\cdots,y_{N-1}\}} E_{\{D_0,\cdots,D_{N-1}\}} \left\{ g_N(X_N) + \sum_{k=0}^{N-1} g_k(X_k, y_k, D_k) \right\}.$$

Por último, para poder resolver este problema se emplea el siguiente algoritmo:

1. Hacer $k = N$ y $J_N(X_N) = g_N(X_N)$.
2. Hacer $k = k-1$ y determinar, para cada estado,

$$J_k(X_k) = \min_{y_k \in Y(X_k)} E_{D_k} \left\{ g_k(X_k, y_k, D_k) + J_{k+1}[f_k(X_k, y_k, D_k)] \right\}.$$

3. Si $k = 0$, detener el algoritmo; en otro caso ir al paso anterior.

Resolviendo este algoritmo, se obtiene la solución a un problema estocástico de programación dinámica.

Definición 4.1.3. La relación entre $J_k(X_k)$ y J_{k+1} que aparece en el algoritmo

$$J_k(X_k) = \min_{y_k \in Y(X_k)} E_{D_k}\{g_k(X_k, y_k, D_k) + J_{k+1}[f_k(X_k, y_k, D_k)]\}.$$

se denomina *ecuación recursiva*.

Teorema 4.1.1. *El valor óptimo de la función objetivo se obtiene en el último paso del algoritmo, es decir:*

$$J^*(X_0) = J_0(X_0).$$

La demostración de este teorema puede encontrarse en [5].

4.2. Problema estocástico de la producción de plaquetas

El problema de producción de plaquetas ha sido explicado anteriormente en el Capítulo 2.

En primer lugar, como los estados de este problema dependerán de los días de vida restante que les quede a las plaquetas que estén en el stock, de la decisión del último día y de las unidades que falten por llegar (esto último afectará solo a sábados y domingos, el resto de días será cero), cada estado será un vector $\widetilde{X}_k$ con la siguiente estructura:

$$\widetilde{X}_k = [X_k^1, X_k^2, X_k^3, X_k^4, X_k^5, \bar{X}_k],$$

donde X_k^r representará las unidades de plaquetas en la etapa k con r días de vida restantes, $r = 1, \ldots, 4$, X_k^5 las unidades de pedido que llegan en la etapa k, es decir, y_k (variable decisión) y $\bar{X}_k$ son las unidades que faltan por llegar, será $\bar{X}_k = 0$ para todos los días de la semana excepto para el sábado y domingo, pues tomará el valor de la decisión óptima del viernes, ya que no llegan esas unidades al stock hasta el lunes. La necesidad de crear esa última componente del vector de estados, se debe a que, como bien hemos comentado anteriormente, en programación dinámica solo es necesaria la anterior etapa para calcular la actual; por tanto, si no se guardan las unidades que se deciden producir el viernes, no llegarían de ninguna manera al lunes.

Al igual que en la Sección 3.2, vamos a expresar las ecuaciones que hemos debido hallar para realizar los cálculos precisos mediante programación dinámica.

La función de transferencia necesaria para la programación dinámica está dada por las mismas ecuaciones que el inventario de la Sección 3.2 (ver 3.9, 3.10, 3.11 y 3.12). La escasez de inventario en el período k está dada por la ecuación

$$\widetilde{S}_k = \begin{cases} D_k - y_k - \dot{X}_k & \text{si } D_k > y_k + \dot{X}_k \\ 0 & \text{en otro caso} \end{cases} \tag{4.1}$$

donde $\dot{X}_k = X_k^1 + X_K^2 + X_k^3 + X_k^4$.

La función de caducidad en el período k queda determinada por

$$\widehat{O}_k = [X_k^1 - D_k]^+.$$

Vamos a pasar a definir la función coste. Al tratarse de programación dinámica, en cada etapa pondremos un coste dependiendo del estado en el que nos encontremos y de la decisión tomada. Como se verá, las funciones, g_k, de coste no coinciden, para cada día, con las del Capítulo 3, ya que los costes se incluyen en días distintos.

La función pago terminal consideraremos que es 0 para cualquier estado, es decir,

$$g_N(\widetilde{X}_k) = 0, \quad \forall\, \widetilde{X}_k.$$

Por otro lado, el pago en cada etapa viene dado por:

$$g_k(\widetilde{X}_k, y_k, D_k) = h \cdot \left(\sum_{r=1}^{4} X_k^r \right) + p \cdot \widetilde{S}_k + \theta \cdot \widehat{O}_k$$

la variable aleatoria que representa la demanda interviene implícitamente en cada una de las variables que aparecen en esta función pago.

Por último, el valor óptimo de la función objetivo obtenida en el último paso del algoritmo será:

$$J^*(\widetilde{X}_0) = J_0(\widetilde{X}_0)$$

donde

$$J_0(\widetilde{X}_0) = \min_{y_0} E_{D_0} \{ g_0(\widetilde{X}_0, y_0, D_0) + J_1[f_0(\widetilde{X}_0, y_0, D_0)] \}.$$

Podemos notar que hay importantes diferencias entre el planteamiento del problema en el Capítulo 3 y la programación dinámica en cuanto a la estructura del problema, ya que el planteamiento de la optimización y la función objetivo a resolver no son las mismas a pesar de que la forma en la que se relacionan las variables X_k^r, que definen la evolución del sistema, sí lo sea. En primer lugar, en el Capítulo 3, la optimización de cada día de la semana se realiza de manera independiente. Para cada día, de lunes a viernes, se elige la cantidad de pedido óptima en el sentido de minimizar los costes que ese pedido generará: almacenamiento de esas unidades hasta su entrega o caducidad, caducidad de esas unidades y ruptura del stock que podría haberse evitado con un pedido mayor ese día. En este sentido, la optimización del Capítulo 3 puede considerarse "miope", ya que no tiene en cuenta cómo puede afectar la decisión tomada hoy en el futuro. Es decir, puede suceder que una decisión me deje el stock en una situación muy cercana a su ruptura, que habrá que pagar en el futuro. El planteamiento en programación dinámica es distinto, ya que para calcular la cantidad de pedido de hoy, se realiza a un horizonte largo, por tanto, el coste considera las implicaciones que la decisión actual tendrá en el futuro. Con respecto a la función objetivo, los costes que involucran la decisión del día k no se incluyen en la función g_k, sino que el valor S_k tiene en cuenta la composición del stock del día k y es función de las decisiones anteriores. De esta manera, observamos que la programación dinámica tiene una visión global del sistema (hasta el horizonte de planificación), mientras que el artículo [12] no la tiene. Notemos que, aunque solo hay que tomar decisiones sobre el pedido de lunes a viernes, en programación dinámica tenemos que considerar todos los días de la semana como etapas (aunque no haya que tomar ninguna decisión los sábados y domingos, estos dos días si generan un coste). A la hora de implementar el algoritmo de programación dinámica nos encontramos con la particularidad de que no es estacionario, es decir, todas las etapas no tienen las mismas funciones ya que dependen del día de la semana; entre otras dificultades que veremos en la siguiente sección.

4.2.1. Resultados experimentales

Una vez que hemos explicado todo lo necesario, podemos crear el programa en C (ver Anexo B.2).

Los costes que utilizaremos para este programa, como hemos mencionado anteriormente, son los mismos que en el Capítulo 2.

Al intentar resolver numéricamente este problema de optimización de producción de plaquetas, nos hemos encontrado con una serie de problemas. En primer lugar, debíamos poner un límite de producción diaria, límite que hemos tomado como 60, es decir, a lo máximo al día podrán producirse 60 unidades. Por un lado, también se debe poner un tope de almacenamiento. Hemos escogido 120 unidades, como máximo, en el almacén (el doble de producción). Por tanto, si al hallar el nuevo estado mediante las funciones de transferencia en una etapa de programación dinámica, llegamos a un estado que la suma de sus componentes de stock supere esas 120 u., desecharemos las unidades más viejas hasta que la suma no supere el límite. Las unidades descartadas se sumarán en el coste como unidades caducadas. Por otro lado, a las demandas diarias se le ha puesto el mismo límite inferior y superior que en el programa creado en el anterior capítulo, es decir, límite inferior $\mu - 2{,}5\sigma$, y como límite superior $\mu + 4\sigma$, donde

μ representa la media y σ la desviación típica. En segundo lugar, la creación de estados, $\widetilde{X}_k$, supone un coste computacional muy elevado. Este problema, conocido como la "maldición de la dimensionalidad" en programación dinámica, ha estado también presente en otros estados de gestión de productos perecederos por programación dinámica, como [8, 7]. La forma de resolverlo en estos artículos es mediante "down-sizing", es decir, agrupando estados con características similares. En algunos trabajos considera la posibilidad de trabajar por lotes, por ejemplo de 5 en 5. En este TFG, se han agrupado los estados en función de si la componente estaba más o menos alejada de la decisión actual. Así, las siguientes elecciones no se han realizado arbitrariamente, sino que hemos considerado más lógico que eran relevante las unidades que perecen hoy (X_k^1), las que llegan hoy (y_k) y las que están por llegar ($\bar{X}_k$), debido a que son decisivas en nuestras ecuaciones de transferencia y en las de caducidad. De esta forma, la primera, quinta y sexta componente de cada vector que representa un estado irán recorriendo los posibles valores de uno en uno; la segunda y tercera componente irán de 5 en 5; y, la cuarta componente de 10 en 10. Con esto, el número de estados según el día de la semana (lunes, ..., domingo) son 8489, 48373, 26047, 5551, 59780, 2224670 y 517829.

A pesar de la agrupación de estados, el coste computacional es elevado. Para cada etapa, el coste se puede calcular aproximadamente de la siguiente manera. Recorremos todos los posibles estados, para cada uno de ellos se analiza cada posible decisión, y éstas implican estudiar todas las posibilidades de demanda. Un hecho que ralentiza la resolución del problema es que, en el momento a emplear la función de transferencia y obtener un estado nuevo, se necesita buscar el valor del coste $J(\widetilde{X}_{k+1})$ asociado a ese nuevo estado. Aunque los estados se guardan ordenados, no es inmediato encontrar la posición del estado nuevo para asignarle su coste J. Por ello, se ha necesitado usar el método de bisección (pues es más rápido que secuencialmente) en la matriz donde se encuentra el estado nuevo para hallar el coste J más rápidamente.

El tiempo de ejecución del programa en un HP Pavilion 15 con CPU AMD A8-6410 APU es de aproximadamente de 1 hora y 50 minutos, para la resolución en $N = 6$ etapas (una semana, pues C se inicia en cero). Debe notarse que, si bien el tiempo de resolución es mucho más largo que con el método del Capítulo 3, en este caso, con una única resolución se tiene la solución óptima para todos los posibles estados iniciales, al contrario que en el método del Capítulo 3, donde hay que ejecutar el programa para cada estado inicial de stock.

A continuación, veamos las gráficas que se han obtenido con las decisiones óptimas mediante programación dinámica. Al igual que hemos hecho en el Capítulo 3, las separaremos por días de la semana.

Gráficas según día de la semana

Veamos, en primer lugar, la dependencia del stock total con la decisión óptima al comienzo del lunes.

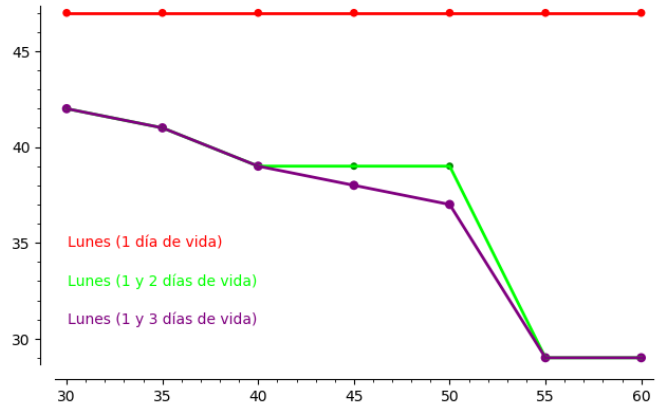

Figura 4.1: **Decisión óptima de los lunes en función de las u. en stock**

Eje x: suma stock total ‖ Eje y: unidades a producir.

Podemos observar que, si todas las unidades que están en el almacén tienen solo 1 día de vida (ver recta roja), la decisión óptima siempre es producir 60 unidades, ya que si todas las unidades tienen un día de vida restante, quiere decir que todas las unidades que no se hayan agotado hoy, caducan al finalizar el día, por tanto, tenemos 0 unidades en el almacén para cubrir todas las demandas posteriores a hoy.

En segundo lugar, observamos la evolución para los martes.

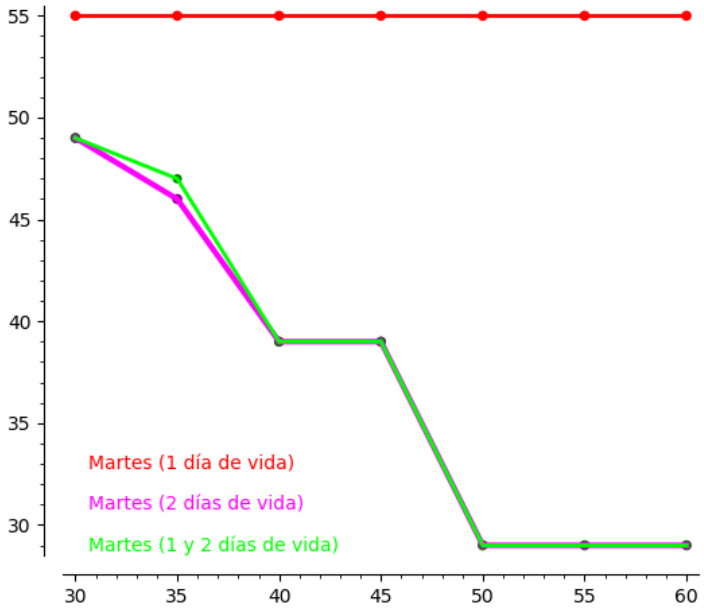

Figura 4.3: **Decisión óptima de los martes en función de las u. en stock**

Eje x: suma stock total ‖ Eje y: unidades a producir.

En tercer lugar, si el día de la semana es miércoles.

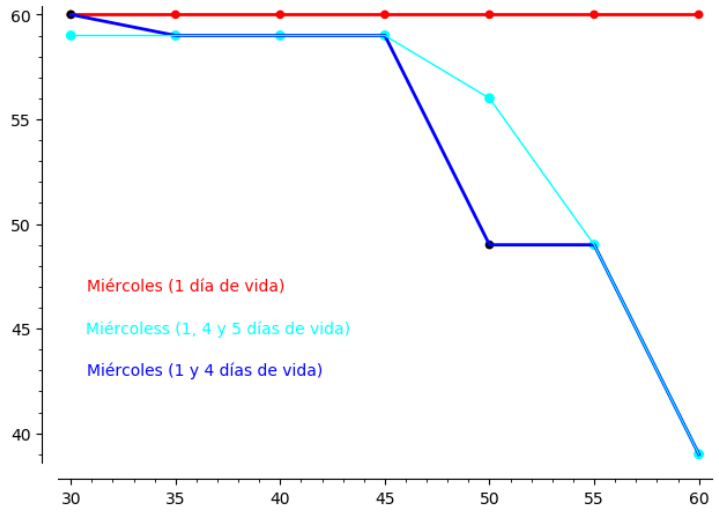

Figura 4.5: **Decisión óptima de los miércoles en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

En cuarto lugar, para los jueves.

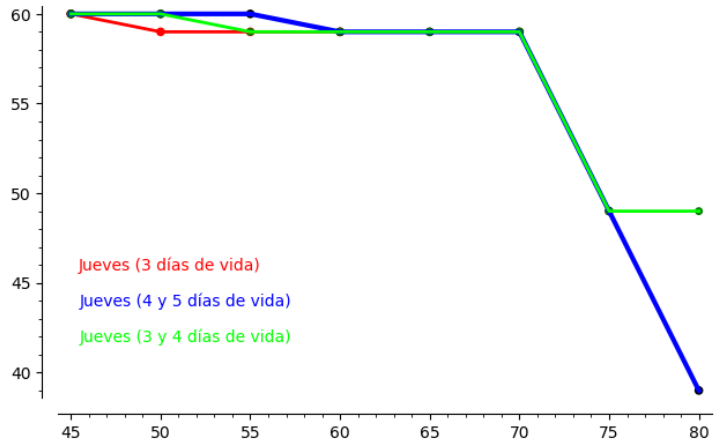

Figura 4.7: **Decisión óptima de los jueves en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

Debemos destacar que los jueves, a pesar de tener un elevado número de u. en stock, se decide producir una gran cantidad de concentrados de plaquetas. Se debe a que la ecuación de coste que genera la decisión que se toma el jueves afecta estrictamente al fin de semana, ya que dicha decisión debe satisfacer las demandas del viernes, sábado y domingo.

Por último, representamos la correspondiente al viernes.

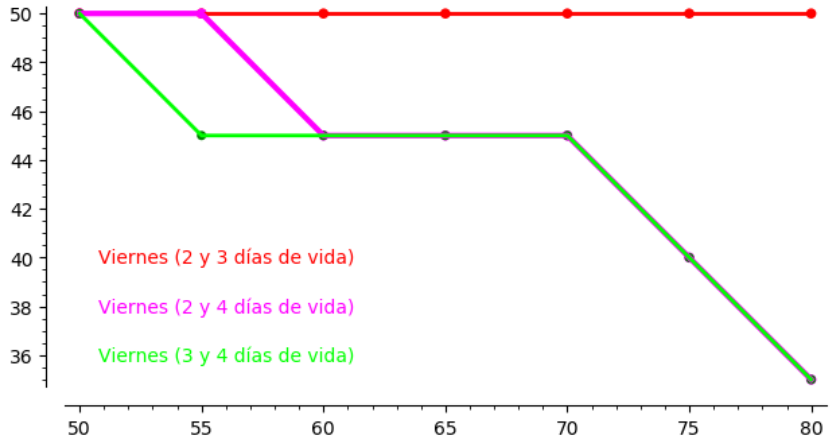

Figura 4.9: **Decisión óptima de los viernes en función de las u. en stock**

Eje x: suma stock total || Eje y: unidades a producir.

Para finalizar este capítulo y con ello este proyecto, podemos comparar los resultados obtenidos en este capítulo con el anterior. Por ello, debemos destacar que en programación dinámica es más importante la composición del stock que en el método del artículo [12], resuelto en el Capítulo 3. Por otro lado, cuando las unidades de stock están distribuidas equitativamente, hemos comprobado que la decisión óptima del método del artículo de Patuwo-Williams [12] es siempre mayor a la que se toma mediante programación dinámica. Esto seguramente se deba a que en programación dinámica, recordemos que resolvemos sobre un horizonte de planificación, por tanto, la situación en la que nos deja nuestra última decisión es aceptable si tuviéramos que seguir tomando decisiones al día siguiente. En cambio, en el artículo [12], recordamos que las decisiones son diarias y no tiene en cuenta como quede el estado del sistema en un futuro.

Bibliografía

[1] F. BAESLER, A. BASTÍAS, M. NEMETH Y C. MARTÍNEZ, *Analysis of inventory strategies for blood components in a regional blood center using process simulation*, Blood Management, 2014.

[2] D. P. BERTSEKAS, *Dynamic Programming. Deterministic and Stochastic Models*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1987.

[3] R. A. C. M. BROEKMEULEN Y K.H. VAN DONSELAAR, *A heuristic to manage perishable inventory with batch ordering, positive lead-times, and time-varying demand*, Computers and Operations Research 36, 2009, 3013-3018.

[4] H. I. CALVETE, *Optimización estocástica: Programación dinámica*, Departamento de Métodos Estadísticos, Universidad de Zaragoza, 2018.

[5] S. E. DREYFUS Y A. M. LAW, *The Art and Theory of Dynamic Programming*, Academic Press, San Diego, CA, 1977.

[6] C. EGUIZABAL, C. GORRIA, M. LEZAUN, F. J. LÓPEZ, J. MONGE, M. Á. PÉREZ VAQUERO Y M.A. VESGA, *Optimization of the management of platelet concentrate stocks in the Basque Country using mathematical simulation*, The International Journal of Transfusion Medicine, International Society of Blood Transfusion, Vox Sanguinis 110, 2016, 369-375.

[7] R. HAIJEMA, *A new class of stock-level dependent orddering policies for perishables with a short maximum shelf life*, Int. J. Production Economics 143, 2013, 434-439.

[8] R. HAIJEMA, J. VAN DER WAL Y N. M. VAN DIJK, *Blood platelet production: Optimization by dynamic programming and simulation*, Computers and Operations Research 34, 2007, 760-779.

[9] G. LABATA, *Optimización de la producción de concentrados de plaquetas en bancos de sangre*, Trabajo de fin de Máster en Modelización e Investiacion Matemática, Estadística y Computación, Universidad de Zaragoza, 2017.

[10] F. J. LÓPEZ LORENTE, *Modelos matemáticos de aprovisionamiento*, Asignatura Modelos de logística del Máster en Modelización e Investigación Matemática, Estadística y Computación.

[11] S. NAHMIAS, *Perishable Inventory Theory: A Review*, The University of Santa Clara, Santa Clara, California, 1982, 680-708.

[12] B. E. PATUWO Y C. L. WILLIAMS, *A perishable inventory model with positive order lead times*, European Journal of Operational Research 116, 1999, 352-373.

[13] B. D. SIVAZLIAN Y L. E. STANFEL, *Optimization Techniques in Operations Research*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1975.

[14] W. L. WINSTON, *Operations Research*, Thomsom Brooks/Cole, Belmont, CA, 4th edición, 2004.

Anexo A

Ecuaciones, teoremas y otros

Ecuaciones

En primer lugar, escribimos el desarrollo de las X_k^r que habíamos mencionado en 3.2. Veámoslo para cada día de la semana:

- Variables que influyen en el coste del lunes y unidades que pedimos el lunes, que llegan el martes y caducan el sábado:

$$
\begin{aligned}
& y_M \\
& X_X^4 = [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ \\
& X_J^3 = [X_X^4 - [D_X - X_X^1]^+]^+ \\
& X_V^2 = [X_J^3 - D_J]^+ \\
& X_S^1 = [X_V^2 - D_V]^+ \\
& O_M^S = [X_S^1 - D_S]^+
\end{aligned}
$$

- Variables involucradas en el coste del martes, y unidades que pedimos el martes, que llegan el miércoles y caducan el domingo:

$$
\begin{aligned}
& y_X \\
& X_J^4 = [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
& X_V^3 = [X_J^4 - [D_J - X_J^3]^+]^+ \\
& X_S^2 = [X_V^3 - [D_V - X_V^2]^+]^+ \\
& X_D^1 = [X_S^2 - [D_S - X_S^1]^+]^+ \\
& O_X^D = [X_D^1 - D_D]^+
\end{aligned}
$$

- Variables necesarias para hallar el coste del miércoles y unidades que pedimos el miércoles, que llegan el jueves y caducan el lunes:

$$
\begin{aligned}
& y_J \\
& X_V^4 = [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ \\
& X_S^3 = [X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ \\
& X_D^2 = [X_S^3 - [D_S - (X_S^1 + X_S^2)]^+]^+ \\
& X_L^1 = [X_D^2 - [D_D - X_D^1]^+]^+ \\
& O_J^L = [X_L^1 - D_L]^+
\end{aligned}
$$

- Variables que aparecen en el coste del jueves y unidades que pedimos el jueves, que llegan el viernes y caducan el martes:

$$
\begin{aligned}
& y_V \\
& X_S^4 = [y_V - [D_V - (X_V^2 + X_V^3 + X_V^4)]^+]^+ \\
& X_D^3 = [X_S^4 - [D_S - (X_S^1 + X_S^2 + X_S^3)]^+]^+ \\
& X_L^2 = [X_D^3 - [D_D - (X_D^1 + X_D^2)]^+]^+ \\
& X_M^1 = [X_L^2 - [D_L - X_L^1]^+]^+ \\
& O_V^M = [X_M^1 - D_M]^+
\end{aligned}
$$

- Variables que influyen en el coste del viernes, y unidades que pedimos el viernes, que llegan el lunes y caducan el miércoles:

$$
\begin{aligned}
& y_L \\
& X_M^2 = [y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ \\
& X_X^1 = [X_M^2 - [D_M - X_M^1]^+]^+ \\
& O_L^X = [X_X^1 - D_X]^+
\end{aligned}
$$

Estas ecuaciones han de expresarse en función del stock del día correspondiente en que pedimos, así como la variable que representa ese pedido y las demandas posibles, como se ha hecho en 3.13. De esta manera, expandiendo las ecuaciones recursivas se obtiene:

- **Martes:**

$$
\begin{aligned}
& y_X \\
& X_J^4 = [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
& \quad = [y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ + [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& X_V^3 = [[y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ + [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& \qquad - [D_J - (X_X^4 - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+)^+]^+]^+ \\
& \quad = [[y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ + [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& \qquad - [D_J - ([y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+)^+]^+]^+ \\
& X_S^2 = [[[y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ + [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& \qquad - [D_J - ([y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+)^+]^+]^+ \\
& \qquad - [D_V - [[y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]]^+]^+ \\
& X_D^1 = [[[y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ + [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& \qquad - [D_J - ([y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+)^+]^+]^+ \\
& \qquad - [D_V - [[y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]]^+ \\
& \qquad - [D_S - [([[y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]^+ \\
& \qquad - D_J)^+ - D_V]^+]^+]^+
\end{aligned}
$$

luego,

$$
\begin{aligned}
O_X^D = & ([[[y_X - [D_X - ([X_M^2 - [D_M - X_M^1]^+]^+ - [y_M - [D_M - (X_M^1 + X_M^2)]^+]^+)]^+]^+ \\
& - [D_J - ([y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+)^+]^+]^+ \\
& - [D_V - [[y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]]^+]^+ \\
& - [D_S - [([[y_M - [D_M - (X_M^1 + X_M^2)]^+]^+ - [D_X - [X_M^2 - [D_M - X_M^1]^+]^+]^+]^+ \\
& - D_J)^+ - D_V]^+]^+ - D_D)^+
\end{aligned}
$$

- **Miércoles:**

$$
\begin{aligned}
& y_J \\
X_V^4 &= [y_J - [D_J - ([X_X^4 - [D_X - X_X^1]^+]^+ + [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+)]^+]^+ \\
X_X^3 &= [[y_J - [D_J - ([X_X^4 - [D_X - X_X^1]^+]^+ + [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+)]^+]^+ \\
&\quad - [D_V - ([[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ + [[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
&\quad - [D_J - [X_x^4 - [D_X - X_X^1]^+]^+]^+]^+)]^+]^+ \\
X_D^2 &= [[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - (X_S^1 + X_S^2)]^+]^+ \\
&= [[[y_J - [D_J - ([X_x^4 - [D_X - X_X^1]^+]^+ + [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+)]^+]^+ \\
&\quad - [D_V - ([[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ + [[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
&\quad - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+)]^+]^+ - [D_S - ([[[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ - D_V]^+ \\
&\quad + [[[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+]^+ - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+ \\
&\quad - [D_V - [[X_X^4 - (D_X - X_X^1)^+]^+ - D_J]^+]^+]^+)]^+]^+ \\
X_L^1 &= [[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - (X_S^1 + X_S^2)]^+]^+ \\
&= [[[y_J - [D_J - ([X_x^4 - [D_X - X_X^1]^+]^+ + [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+)]^+]^+ \\
&\quad - [D_V - ([[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ + [[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
&\quad - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+)]^+]^+ - [D_S - ([[[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ - D_V]^+ \\
&\quad + [[[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+]^+ - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+ \\
&\quad - [D_V - [[X_X^4 - (D_X - X_X^1)^+]^+ - D_J]^+]^+]^+)]^+]^+ - [D_D - [[[[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+]^+ \\
&\quad - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+ - [D_V - [[X_X^4 - [D_X - X_X^1]^+]^+]^+ \\
&\quad - D_J]^+]^+]^+ - [D_S - [[[X_X^4 - (D_X - X_X^1)^+]^+ - D_J]^+ - D_V]^+]^+]^+]^+]^+
\end{aligned}
$$

entonces,

$$
\begin{aligned}
O_J^L &= [[[[[y_J - [D_J - ([X_x^4 - [D_X - X_X^1]^+]^+ + [y_X - [D_X - (X_X^1 + X_X^4)]^+]^+)]^+]^+ \\
&\quad - [D_V - ([[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ + [[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+ \\
&\quad - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+)]^+]^+ - [D_S - ([[[X_X^4 - [D_X - X_X^1]^+]^+ - D_J]^+ - D_V]^+ \\
&\quad + [[[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+]^+ - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+ \\
&\quad - [D_V - [[X_X^4 - (D_X - X_X^1)^+]^+ - D_J]^+]^+]^+)]^+]^+ - [D_D - [[[[y_X - [D_X - (X_X^1 + X_X^4)]^+]^+]^+ \\
&\quad - [D_J - [X_X^4 - [D_X - X_X^1]^+]^+]^+]^+ - [D_V - [[X_X^4 - [D_X - X_X^1]^+]^+]^+ \\
&\quad - D_J]^+]^+]^+ - [D_S - [[[X_X^4 - (D_X - X_X^1)^+]^+ - D_J]^+ - D_V]^+]^+]^+]^+]^+ - D_L]^+
\end{aligned}
$$

- **Jueves:**

$$
\begin{aligned}
& y_V \\
X_S^4 &= [y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
X_D^3 &= [[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+ + [X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+]^+ \\
&= [[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
X_L^2 &= [[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+ + [X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+]^+ \\
&= [[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
&\quad - [D_D - ([X_S^2 - [D_S - X_S^1]^+]^+ + [X_S^3 - [D_S - (X_S^1 + X_S^2)]^+]^+)]^+]^+ \\
&= [[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ \\
&\quad + [[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
&\quad - [D_D - ([[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+ + [[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ \\
&\quad - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+)]^+]^+ \\
&= [[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+)]^+]^+ \\
X_M^1 &= [[[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 \\
&\quad - D_J]^+]^+]^+)]^+]^+)]^+]^+ - [D_L - [X_D^2 - [D_D - X_D^1]^+]^+]^+]^+ \\
&= [[[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[X_S^3 - [D_S - (X_S^1 + X_S^2)]^+]^+ - [D_D - X_D^1]^+]^+]^+]^+
\end{aligned}
$$

$$
\begin{aligned}
&= [[[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+ \\
&\quad - [D_D - [X_S^1 - [D_S - X_S^1]^+]^+]^+]^+]^+ \\
&= [[[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+ \\
&\quad - [D_D - [[[X_J^4 - [D_J - X_J^3]^+]^+ - D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+]^+]^+
\end{aligned}
$$

así,

$$
\begin{aligned}
O_V^M = &[[[[[y_V - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+ + [y_J - [D_J - (X_J^3 + X_J^4)]^+]^+)]^+]^+ \\
&\quad - [D_S - [[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ + [[y_J \\
&\quad - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+]^+ \\
&\quad - [D_D - ([[[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ - D_V]^+]^+]^+ \\
&\quad + [[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[[[y_J - [D_J - (X_J^3 + X_J^4)]^+]^+]^+ - [D_V - ([X_J^3 - D_J]^+ + [X_J^4 - [D_J - X_J^3]^+]^+)]^+]^+ \\
&\quad - [D_S - ([[X_J^3 - D_J]^+ - D_V]^+ + [[X_J^4 - [D_J - X_J^3]^+]^+ - [D_V - [X_J^3 - D_J]^+]^+]^+)]^+]^+ \\
&\quad - [D_D - [[[X_J^4 - [D_J - X_J^3]^+]^+ - D_V - [X_J^3 - D_J]^+]^+]^+ - [D_S - [[X_J^3 - D_J]^+ \\
&\quad - D_V]^+]^+]^+]^+]^+ - D_M]^+
\end{aligned}
$$

- **Viernes:**

$$
\begin{aligned}
& y_L \\
X_M^2 &= [y_L - [D_L - (X_L^1 + X_L^2)]^+]^+ \\
&= [y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 \\
&\quad - [D_V - X_V^2]^+]^+)]^+]^+ - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+]^+)]^+]^+ \\
X_X^1 &= [X_M^2 - [D_M - X_M^1]^+]^+ \\
&= [[y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+ \\
&\quad - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+)]^+]^+ - [D_M - [X_L^2 - [D_L - X_L^1]^+]^+]^+]^+ \\
&= [[y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 \\
&\quad - [D_V - X_V^2]^+]^+)]^+]^+ - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+)]^+]^+ \\
&\quad - [D_M - [[X_D^3 - [D_D - (X_D^1 + X_D^2)]^+]^+ - [D_L - [X_D^2 - [D_D - X_D^1]^+]^+]^+]^+]^+ \\
&= [[y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 \\
&\quad - [D_V - X_V^2]^+]^+)]^+]^+ - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+)]^+]^+ \\
&\quad - [D_M - [[[X_S^4 - [D_S - (X_S^1 + X_S^2 + X_S^3)]^+]^+ - [D_D - ([X_S^2 - [D_S - X_S^1]^+]^+ + [X_S^3 - [D_S \\
&\quad - (X_S^1 + X_S^2)]^+]^+)]^+]^+ - [D_L - [[X_S^3 - [D_S - (X_S^1 + X_S^2)]^+]^+ - [D_D - [X_S^2 - [D_S - X_S^1]^+]^+]^+]^+]^+]^+ \\
&= [[y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 \\
&\quad - [D_V - X_V^2]^+]^+)]^+]^+ - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+)]^+]^+ \\
&\quad - [D_M - [[[[y_V - [D_V - (X_V^2 + X_V^3 + X_V^4)]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+ + [X_V^4 - [D_V \\
&\quad - (X_V^2 + X_V^3)]^+]^+)]^+]^+ - [D_D - ([[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+ \\
&\quad + [[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+ \\
&\quad - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+]^+]^+]^+
\end{aligned}
$$

se tiene,

$$
\begin{aligned}
O_L^X &= [[[y_L - [D_L - ([[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 \\
&\quad - [D_V - X_V^2]^+]^+)]^+]^+ - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+)]^+]^+ \\
&\quad - [D_M - [[[[y_V - [D_V - (X_V^2 + X_V^3 + X_V^4)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+ + [X_V^4 - [D_V \\
&\quad - (X_V^2 + X_V^3)]^+]^+)]^+]^+ - [D_D - ([[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+ \\
&\quad + [[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+)]^+]^+ \\
&\quad - [D_L - [[[X_V^4 - [D_V - (X_V^2 + X_V^3)]^+]^+]^+ - [D_S - ([X_V^2 - D_V]^+ + [X_V^3 - [D_V - X_V^2]^+]^+)]^+]^+ \\
&\quad - [D_D - [[X_V^3 - [D_V - X_V^2]^+]^+ - [D_S - [X_V^2 - D_V]^+]^+]^+]^+]^+]^+]^+ - D_X]^+
\end{aligned}
$$

Teoremas

En segundo lugar, vamos a modificar el primer teorema que aparece en el artículo [12], debido a que no es del todo correcto, por tanto, lo corregimos y demostramos.

Teorema A.0.1. *La función de densidad para el inventario inicial, X_t^1, para $t = 1,2,3,\ldots$, está dada por:*

- *Para* $t = 1$,

$$f_1(x_1^1) = \begin{cases} 1 - G_0(X_0^1 + y_0) & si\ x_1^1 = 0, \\ g_0(X_0^1 + y_0 - x_1^1) & si\ 0 \le x_1^1 \le y_0, \\ 0 & en\ otro\ caso. \end{cases} \tag{A.1}$$

- *Para* $t = 2,3,4,\ldots$,

$$f_t(x_t^1) = \begin{cases} [1 - G_{t-1}(y_{t-1})] \cdot f_{t-1}(0) + \int_{x_{t-1}^1=0}^{y_{t-2}} [1 - G_{t-1}(y_{t-1} + x_{t-1}^1)] \cdot f_{t-1}(x_{t-1}^1) \mathrm{d}x_{t-1}^1 & si\ x_t^1 = 0, \\ g_{t-1}(y_{t-1} - x_t^1) \cdot f_{t-1}(0) + \int_{x_{t-1}^1=0}^{y_{t-2}} [g_{t-1}(y_{t-1} + x_{t-1}^1 - x_t 1)] \cdot f_{t-1}(x_{t-1}^1) \mathrm{d}x_{t-1}^1 & si\ 0 \le x_t^1 \le y_{t-1}, \\ 0 & en\ otro\ caso. \end{cases} \tag{A.2}$$

Demostración. Para $t = 1$, x_1^1 está dado por:

$$x_1^1 = [y_0 - (D_0 - x_0^1)^+]^+,$$

donde $g(x)^+ = \text{máx}\{0,\ g(x)\}$.

Si $x_1^1 = 0$:

$$x_1^1 = 0 \iff (D_0 - X_0^1)^+ > y_0 \iff D_0 - X_0^1 > y_0 \iff D_0 > y_0 + X_0^1$$

Luego,

$$P(x_1^1) = P(D_0 > y_0 + X_0^1) = 1 - P(D_0 \le y_0 + X_0^1) = 1 - G_0(y_0 + X_0^1) \quad \text{si} \quad x_1^1 = 0.$$

La última igualdad, se da porque la función de distribución de la demanda, D_0, es $G_0(\cdot)$.

Si $X_1^1 > x_1^1$ y $x_1^1 \le y_0$:

$$\begin{aligned} P(X_1^1 > x_1^1) &= P((y_0 - (D_0 - X_0^1)^+)^+ > x_1^1) = P(y_0 - (D_0 - X_0^1)^+ > x_1^1) \\ &= P(y_0 - x_1^1 > (D_0 - X_0^1)^+) = P((D_0 - X_0^1)^+ < y_0 - x_1^1) = P(D_0 - X_0^1 < y_0 - x_1^1) \\ &= P(D_0 < y_0 + X_0^1 - x_1^1) = G_0(y_0 + X_0^1 - x_1^1). \end{aligned}$$

La última igualdad se da porque la función de distribución de la demanda, D_t, es $G_t(\cdot)$.

Por tanto, si $x_1^1 \le y_0$:

$$F(X_1^1) = P(X_1^1 \le x_1^1) = 1 - P(X_1^1 > x_1^1) = 1 - G_0(y_0 + X_0^1 - x_1^1).$$

Derivamos:

$$f_{X_1^1}(x_1^1) = \frac{\mathrm{d}F(X_1^1)}{\mathrm{d}x_1^1} = -g_0(y_0 + X_0^1 - x_1^1) \cdot (-1) = g_0(y_0 + X_0^1 - x_1^1) \quad \text{si} \quad 0 \le x_1^1 \le y_0.$$

En otro caso, es 0, ya que X_1^1 no puede tomar valores negativos.

Para $t = 2,3,\cdots$, x_t^1 está dado por

$$X_t^1 = (y_{t-1} - (D_{t-1} - X_{t-1}^1)^+)^+.$$

$$x_t^1 = 0 \iff (D_{t-1} - X_{t-1}^1)^+ > y_{t-1} \iff D_{t-1} - X_{t-1}^1 > y_{t-1} \iff D_{t-1} > y_{t-1} + X_{t-1}^1.$$

Así,

$$\begin{aligned} P(X_t^1 = 0) &= P(D_{t-1} > y_{t-1} + x_{t-1}^1) = 1 - P(D_{t-1} \leq y_{t-1} + x_{t-1}^1) \\ &= 1 - G_{t-1}(y_{t-1} + x_{t-1}^1). \end{aligned}$$

Luego, si $x_{t-1}^1 = 0$,

$$\begin{aligned} f_{X_t^1}(x_t^1) &= 1 - G_{t-1}(y_{t-1} + x_{t-1}^1) \\ &= [1 - G_{t-1}(y_{t-1} + 0)] \cdot f_{X_{t-1}^1}(x_{t-1}^1 = 0) + \int_{x_{t-1}^1=0}^{y_{t-2}} [1 - G_{t-1}(y_{t-1} + x_{t-1}^1)] \cdot f_{X_{t-1}^1}(x_{t-1}^1)\, \mathrm{d}x_{t-1}^1 \end{aligned}$$

Si $X_{t-1}^1 > x_{t-1}^1$ y $x_t^1 \leq y_{t-1}$, se tiene:

$$\begin{aligned} P(X_t^1 > x_t^1) &= P((y_{t-1} - (D_{t-1} - x_{t-1}^1)^+)^+ > x_t^1) = P((D_{t-1} - x_{t-1}^1)^+ < y_{t-1} - x_t^1) \\ &= P(D_{t-1} < y_{t-1} + x_{t-1}^1 - x_t^1) = G_{t-1}(y_{t-1} + x_{t-1} - x_t^1) \end{aligned}$$

Con esto, si $x_t^1 \leq y_{t-1}$:

$$\begin{aligned} F(X_t^1) &= P(X_t^1 \leq x_t^1) = 1 - P(X_t^1 > x_t^1) \\ &= 1 - G_{t-1}(y_{t-1} + x_{t-1}^1 - x_t^1) \end{aligned}$$

Derivamos, ya que nos interesa la función de densidad,

$$f_{X_t^1}(x_t^1) = \frac{\mathrm{d}F}{\mathrm{d}x_t^1}(X_t^1) = -g_{t-1}(y_{t-1} + x_{t-1}^1 - x_t^1) \cdot (-1) = g_{t-1}(y_{t-1} + x_{t-1}^1 - x_t^1)$$

Por tanto, depende del valor de x_{t-1}^1, si $0 < x_t^1 \leq y_{t-1}$:

$$\begin{aligned} f_{X_t^1}(x_t^1) &= g_{t-1}(y_{t-1} + x_{t-1}^1 - x_t^1) \\ &= [g_{t-1}(y_{t-1} + 0 - x_t^1)] \cdot f_{X_{t-1}^1}(x_{t-1}^1 = 0) + \int_{x_{t-1}^1=0}^{y_{t-2}} g_{t-1}(y_{t-1} + x_{t-1}^1 - x_t^1) \cdot f_{X_{t-1}^1}(x_{t-1}^1)\, \mathrm{d}x_{t-1}^1 \end{aligned}$$

En otro caso, es 0.

□

Anexo B

Código C

B.1. Programa para artículo Patuwo y Williams

```
// Este programa calcula la decision optima de unidades que hay que producir
// de plaquetas dependiendo del dia de la semana que es Lunes, Martes, ..., Viernes
// siguiendo las nociones del articulo de Patuwo y Williams.
// Recordatorio: Sabado y Domingo no se produce, por tanto, si es uno de esos dos
// dias, no llamaremos al programa

#include<stdio.h>
#include<stdlib.h>

// Numero de plaquetas como maximo que puedo producir en cada etapa, en las tablas
// que se generan y_k ira de 0, 1, ..., MAX_DECISION

#define MAX_DECISION 60

// Como maximo duraran las plaquetas 4 dias (4 representa: 0, 1, 2, 3)

#define DIAS 4

// DIASEMANA va de 0 a 6, donde 0 es L y 6 es D

#define DIASEMANA 7

//Definimos los costes aqui, asi en el programa solo aparecera c, h, p, theta.
// Se pueden cambiar los valores desde aqui.

#define c 0 // Coste producir
#define h 2 // Coste por permanecer una noche en inventario
#define p 5000 // Coste por demanda insatisfecha
#define theta 400 // Coste por caducidad



// PROGRAMA PRINCIPAL


main()
```

```
{
char diasemana;
int decision_ayer;
int stock[DIAS][DIASEMANA];

printf("Introduce el dia de hoy (EN MAYUSCULAS), siendo\n L, lunes;\n M, martes;\n
X, miercoles;\n J, jueves;\n V, viernes. \n");
scanf("%c", &diasemana);
int DS = pasaranumero(diasemana);

int decisionOptima;
decisionOptima = actualizar_stock(DS,diasemana,decision_ayer);

printf("Decision optima: Producir %i unidades de plaquetas \n", decisionOptima);
}


// ------------------------------------------------

// Veamos, a continuacion, todas las funciones necesarias para poder ejecutar
// el programa principal


// En el fichero ZZ.txt se encuentran guardadas las probabilidades de la N(0,1)

#define DIM 10001

void normalZ(double *v)
{
FILE *pleer;
double x;
int i=0;
pleer = fopen("ZZ.txt","r+");

fscanf(pleer, "%lf", &x);
while(feof(pleer)==0)
{
v[i] = x;
fscanf(pleer, "%lf", &x);
i++;
}

fclose(pleer);
}

// Leemos el fichero donde se encuentran las distribuciones de las demandas de cada
// dia de la semana (medias y desv.tip.) :

#define DIM1 14

void mediavar(double *w)
```

```
{
FILE *pleer;

double x;
int j=0;
pleer = fopen("mediavar3.txt","r+");

fscanf(pleer, "%lf", &x);
while(feof(pleer)==0)
{
w[j] = x;
fscanf(pleer, "%lf", &x);
j++;
}

fclose(pleer);
}

#define DIMP 200

// Guardamos en un vector que se llame PL las probabilidades de la demanda del LUNES

void probaL(double *PL, int *pmindL, int *pmaxdL)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dL, dLmax, mindL, maxdL;
int jmin;
jmin=-5000;

int k,m;
double mu = w[0], sigma = w[7];

mindL = (int)(mu - 2.5 * sigma);
maxdL = (int)(mu + 4 * sigma);

for(dL=0; dL<=DIMP; dL++)
{
k=(int)((((dL+0.5)-mu)/sigma)*1000);
m=(int)((((dL-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dLmax=dL;
break;
```

```
}

PL[dL]=v[k-jmin]-v[m-jmin];
}

*pmindL=mindL;
*pmaxdL=maxdL;

return;
}

// Guardamos en un vector que se llame PM las probabilidades de la demanda del MARTES

void probaM(double *PM, int *pmindM, int *pmaxdM)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dM,dMmax, mindM, maxdM;
int jmin;
jmin=-5000;
int k,m;

double mu = w[1], sigma = w[8];

mindM = (int)(mu - 2.5 * sigma);
maxdM = (int)(mu + 4 * sigma);

for(dM=0; dM<=DIMP; dM++)
{
k=(int)((((dM+0.5)-mu)/sigma)*1000);
m=(int)((((dM-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dMmax=dM;
break;
}

PM[dM]=v[k-jmin]-v[m-jmin];
}

*pmindM=mindM;
*pmaxdM=maxdM;

return;
```

```
}


// Guardamos en un vector que se llame PX las probabilidades de la demanda del MIERCOLES

void probaX(double *PX, int *pmindX, int *pmaxdX)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dX,dXmax, mindX, maxdX;
int jmin;
jmin=-5000;
int k,m;

double mu = w[2], sigma = w[9];

mindX = (int)(mu - 2.5 * sigma);
maxdX = (int)(mu + 4 * sigma);

for(dX=0; dX<=DIMP; dX++)
{
k=(int)((((dX+0.5)-mu)/sigma)*1000);
m=(int)((((dX-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dXmax=dX;
break;
}

PX[dX]=v[k-jmin]-v[m-jmin];
}

*pmindX=mindX;
*pmaxdX=maxdX;

return;
}

// Guardamos en un vector que se llame PJ las probabilidades de la demanda del JUEVES

void probaJ(double *PJ, int *pmindJ, int *pmaxdJ)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
```

```
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dJ,dJmax, mindJ, maxdJ;
int jmin;
jmin=-5000;
int k,m;

double mu = w[3], sigma = w[10];

mindJ = (int)(mu - 2.5 * sigma);
maxdJ = (int)(mu + 4 * sigma)+1;

for(dJ=0; dJ<=DIMP; dJ++)
{
k=(int)((((dJ+0.5)-mu)/sigma)*1000);
m=(int)((((dJ-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dJmax=dJ;
break;
}

PJ[dJ]=v[k-jmin]-v[m-jmin];
}

*pmindJ=mindJ;
*pmaxdJ=maxdJ;

return;
}


// Guardamos en un vector que se llame PV las probabilidades de la demanda del VIERNES

void probaV(double *PV, int *pmindV, int *pmaxdV)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dV,dVmax, mindV, maxdV;
int jmin;
```

```
jmin=-5000;
int k,m;

double mu = w[4], sigma = w[11];

mindV = (int)(mu - 2.5 * sigma);
maxdV = (int)(mu + 4 * sigma)+1;

for(dV=0; dV<=DIMP; dV++)
{
k=(int)((((dV+0.5)-mu)/sigma)*1000);
m=(int)((((dV-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dVmax=dV;
break;
}

PV[dV]=v[k-jmin]-v[m-jmin];
}

*pmindV=mindV;
*pmaxdV=maxdV;

return;
}

// Guardamos en un vector que se llame PS las probabilidades de la demanda del SABADO

void probaS(double *PS, int *pmindS, int *pmaxdS)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dS,dSmax, mindS, maxdS;
int jmin;
jmin=-5000;
int k,m;

double mu = w[5], sigma = w[12];

mindS = (int)(mu - 2.5 * sigma);
maxdS = (int)(mu + 4 * sigma)+1;

for(dS=0; dS<=DIMP; dS++)
```

```
{
k=(int)((((dS+0.5)-mu)/sigma)*1000);
m=(int)((((dS-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dSmax=dS;
break;
}

PS[dS]=v[k-jmin]-v[m-jmin];
}

*pmindS=mindS;
*pmaxdS=maxdS;

return;
}


// Guardamos en un vector que se llame PD las probabilidades de la demanda del DOMINGO

void probaD(double *PD, int *pmindD, int *pmaxdD)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dD,dDmax, mindD, maxdD;
int jmin;
jmin=-5000;
int k,m;

double mu = w[6], sigma = w[13];

mindD = (int)(mu - 2.5 * sigma);
maxdD = (int)(mu + 4 * sigma);

for(dD=0; dD<=DIMP; dD++)
{
k=(int)((((dD+0.5)-mu)/sigma)*1000);
m=(int)((((dD-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dDmax=dD;
break;
```

```
}

PD[dD]=v[k-jmin]-v[m-jmin];
}

*pmindD=mindD;
*pmaxdD=maxdD;

return;
}

// -------------------------------------------------------

// Pasamos letras de la semana a numeros y viceversa

int pasaranumero (char diasemana)
{
if (diasemana == 'L')
return 0;
else if (diasemana == 'M')
return 1;
else if (diasemana == 'X')
return 2;
else if (diasemana == 'J')
return 3;
else if (diasemana == 'V')
return 4;
else if (diasemana == 'S')
return 5;
else if (diasemana == 'D')
return 6;
}

char pasaradia (int numerosemana)
{
if (numerosemana == 0)
return 'L';
else if (numerosemana == 1)
return 'M';
else if (numerosemana == 2)
return 'X';
else if (numerosemana == 3)
return 'J';
else if (numerosemana == 4)
return 'V';
else if (numerosemana == 5)
return 'S';
else if (numerosemana == 6)
return 'D';
}
```

```
// ------------------------------------------------------

// Programa que nos realiza los calculos pertinentes para poder tomar una decision optima

int actualizar_stock(int DS,char diasemana, int decision_ayer)
{
int stock[DIAS][DIASEMANA];

// Pedimos lo necesario y conocido por el trabajador por pantalla

if(diasemana == 'L' || diasemana =='M' || diasemana =='X' || diasemana =='S'
|| diasemana =='D')
{
printf("Introduce unidades con 1 dia de vida restante: \n");
scanf("%i", &stock[0][DS]);
}

if(diasemana == 'L' || diasemana =='M' || diasemana =='V' || diasemana =='S'
|| diasemana =='D')
{
printf("Introduce unidades con 2 dias de vida restantes: \n");
scanf("%i", &stock[1][DS]);
}

if(diasemana =='J' || diasemana =='V' || diasemana =='S' || diasemana =='D')
{
printf("Introduce unidades con 3 dias de vida restantes: \n");
scanf("%i", &stock[2][DS]);
}

if(diasemana == 'X' || diasemana =='J' || diasemana =='V' || diasemana =='S')
{
printf("Introduce unidades con 4 dias de vida restantes: \n");
scanf("%i", &stock[3][DS]);
}


if(diasemana == 'L' || diasemana =='M' || diasemana =='X' || diasemana =='J'
|| diasemana =='V')
{
printf("Introduce unidades que llegan hoy: \n");
scanf("%i", &decision_ayer);
}



// Declaramos variables

int decision_hoy;
int i,nveces=1;
int dL, dM, dX, dJ, dV, dS, dD;
```

```
double minimoCoste;
int decisionOptima;


// Llamamos a las funciones donde estan los vectores de probabilidades de las demandas
// de cada dia de la semana

double PL[DIMP];
int maxdL, mindL;
probaL(PL,&mindL,&maxdL);

// ---------------

double PM[DIMP];
int maxdM, mindM;
probaM(PM,&mindM,&maxdM);

// ---------------

double PX[DIMP];
int maxdX, mindX;
probaX(PX,&mindX,&maxdX);

// ---------------

double PJ[DIMP];
int maxdJ, mindJ;
probaJ(PJ,&mindJ,&maxdJ);

// ---------------

double PV[DIMP];
int maxdV, mindV;
probaV(PV,&mindV,&maxdV);

// ---------------

double PS[DIMP];
int maxdS, mindS;
probaS(PS,&mindS,&maxdS);

// ---------------

double PD[DIMP];
int maxdD, mindD;
probaD(PD,&mindD,&maxdD);

// -----------------------------------------------------

// Distinguimos 5 casos (pues solo se toma decisiones de LUNES a VIERNES)
```

```
// LUNES (K= L, K+1= M)


if (diasemana == 'L')
{
// Bucle para las y_k
for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy = decision_hoy +2)
{
double Ecaducidad = 0;
double Eescasez = 0;
double Estock = 0;
double Ecoste = 0;

for (dL=mindL; dL <= maxdL; dL = dL + 2)
for (dM=mindM; dM <= maxdM; dM = dM + 2)
for (dX=mindX; dX <= maxdX; dX = dX + 2)
for (dJ=mindJ; dJ <= maxdJ; dJ = dJ + 2)
for (dV=mindV; dV <= maxdV; dV = dV + 2)
for (dS=mindS; dS <= maxdS; dS = dS + 2)
{
// Actualizamos stock:

// Veamos la ecuacion de transferencia para las u. de stock de 1 dia de vida
// restante (primer elemento del vector) es decir:
// X_M^1

if(stock[0][0]>=dL)
{
stock[0][1] = stock[1][0];
}
else if(stock[0][0] < dL && (dL < (stock[0][0] + stock[1][0])))
{
stock[0][1] = stock[1][0] - (dL - stock[0][0]);
}
else
{
stock[0][1]=0;
}

// Veamos la ecuacion de transferencia para las u. de stock de 2 dias de vida restante
// (segundo elemento del vector) es decir:
// X_M^2

if((stock[0][0] + stock[1][0]) >= dL)
{
stock[1][1] = decision_ayer;
}
else if((stock[0][0] + stock[1][0]) < dL && (dL < (stock[0][0] +
stock[1][0] + decision_ayer)))
{
```

```
stock[1][1] = decision_ayer - (dL - (stock[0][0] + stock[1][0]));
}
else
{
stock[1][1] = 0;
}

// El Lunes NO hay u. de stock de 3 ni 4 dias de vida restantes


// Veamos las ecuaciones necesarias para hallar la caducidad

//HOY ES LUNES

//X_X^4
if (decision_hoy > dM - (stock[0][1] + stock[1][1]))
{
if (dM > stock[0][1] + stock[1][1])
{
stock[3][2] = decision_hoy - (dM - (stock[0][1] + stock[1][1]));
}
else if (dM <= stock[0][1] + stock[1][1])
{
stock[3][2] = decision_hoy;
}
}
else
stock[3][2] = 0;

//X_X^1
if (stock[1][1] > dM - stock[0][1])
{
if (dM > stock[0][1])
{
stock[0][2] = stock[1][1] - (dM - stock[0][1]);
}
else if (dM <= stock[0][1])
{
stock[0][2] = stock[1][1];
}
}
else
stock[0][2] = 0;

//X_J^3
if (stock[3][2] > dX - stock[0][2])
{
if (dX > stock[0][2])
{
stock[2][3] = stock[3][2] - (dX - stock[0][2]);
}
```

```
else if (dX <= stock[0][2])
{
stock[2][3] = stock[3][2];
}
}
else
stock[2][3] = 0;

//X_V^2
if (stock[2][3] > dJ)
{
stock[1][4] = stock[2][3] - dJ;
}
else
stock[1][4] = 0;

//X_S^1
if (stock[1][4] > dV)
{
stock[0][5] = stock[1][4] - dV;
}
else
stock[0][5] = 0;


// COSTE stock:

double stockTOTAL;
stockTOTAL = (stock[3][2] + stock[2][3] + stock[1][4] + stock[0][5]) * PL[dL]
* PM[dM] * PX[dX] * PJ[dJ] * PV[dV] * PS[dS];

Estock = Estock + stockTOTAL; //esperanza stockTOTAL


//Unidades que pido el lunes, me llegan el martes y me caducan el sabado
//O_M^S
int caducidad_M_S;

if (stock[0][5] > dS)
{
caducidad_M_S = stock[0][5] - dS;
}
else
{
caducidad_M_S = 0;
}


// COSTE caducidad

double caducidad_MS;
```

```
caducidad_MS = caducidad_M_S * PL[dL] * PM[dM] * PX[dX] * PJ[dJ] * PV[dV] * PS[dS];

Ecaducidad = Ecaducidad + caducidad_MS; //esperanza caducidad


// Creamos la variable de escasez de demanda en la etapa k = L
int escasez_demanda_M;
if(dM > decision_hoy + stock[0][1] + stock[1][1])
{
escasez_demanda_M = dM - decision_hoy - (stock[0][1] + stock[1][1]);
}
else if(dM <= decision_hoy + stock[0][1] + stock[1][1])
{
escasez_demanda_M = 0;
}


// COSTE escasez

double escasez_demandaM;
escasez_demandaM = escasez_demanda_M * PL[dL] * PM[dM] * PX[dX] * PJ[dJ]
* PV[dV] * PS[dS];

Eescasez = Eescasez + escasez_demandaM; //esperanza caducidad
}


// COSTE TOTAL

Ecoste = c * decision_hoy + h * Estock + p * Eescasez + theta * Ecaducidad;

if(nveces==1)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
else if(Ecoste<minimoCoste)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
}
}


// ------------------------------------------------


// MARTES (K=M, K+1= X)
```

```
if (diasemana == 'M')
{
// Este bucle es para las y_k
for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy = decision_hoy + 2)
{
double Ecaducidad = 0;
double Eescasez = 0;
double Estock = 0;
double Ecoste = 0;

for (dM=mindM; dM <= maxdM; dM = dM +2)
for (dX=mindX; dX <= maxdX; dX = dX +2)
for (dJ=mindJ; dJ <= maxdJ; dJ = dJ +2)
for (dV=mindV; dV <= maxdV; dV = dV +2)
for (dS=mindS; dS <= maxdS; dS = dS +2)
for (dD=mindD; dD <= maxdD; dD = dD +2)
{
// Actualizamos stock:

// Veamos la ecuacion de transferencia para las u. de stock de 1 dia de vida
// restante (primer elemento del vector) es decir:
// X_X^1

if(stock[0][1]>=dM)
{
stock[0][2] = stock[1][1];
}
else if((stock[0][1] < dM) && (dM < (stock[0][1] + stock[1][1])))
{
stock[0][2] = stock[1][1] - (dM - stock[0][1]);
}
else
{
stock[0][2]=0;
}

// El Martes NO hay u. de stock de 3 ni 4 dias de vida restantes


// Veamos las ecuaciones necesarias para hallar la caducidad

// HOY ES MARTES, necesitamos las del Lunes y otras
// Las del Lunes

//X_X^4
if (decision_ayer > dM - (stock[0][1] + stock[1][1]))
{
if (dM > stock[0][1] + stock[1][1])
{
```

```
stock[3][2] = decision_ayer - (dM - (stock[0][1] + stock[1][1]));
}
else if (dM <= stock[0][1] + stock[1][1])
{
stock[3][2] = decision_ayer;
}
}
else
stock[3][2] = 0;

//X_J^3
if (stock[3][2] > dX - stock[0][2])
{
if (dX > stock[0][2])
{
stock[2][3] = stock[3][2] - (dX - stock[0][2]);
}
else if (dX <= stock[0][2])
{
stock[2][3] = stock[3][2];
}
}
else
stock[2][3] = 0;

//X_V^2
if (stock[2][3] > dJ)
{
stock[1][4] = stock[2][3] - dJ;
}
else
stock[1][4] = 0;

//X_S^1
if (stock[1][4] > dV)
{
stock[0][5] = stock[1][4] - dV;
}
else
stock[0][5] = 0;

// Las de hoy Martes

//X_J^4
if (decision_hoy > dX - (stock[0][2] + stock[3][2]))
{
if (dX > stock[0][2] + stock[3][2])
{
stock[3][3] = decision_hoy - (dX - (stock[0][2] + stock[3][2]));
}
else if (dX <= stock[0][2] + stock[3][2])
```

```
{
stock[3][3] = decision_hoy;
}
}
else
stock[3][3] = 0;

//X_V^3
if (stock[3][3] > dJ - stock[2][3])
{
if (dJ > stock[2][3])
{
stock[2][4] = stock[3][3] - (dJ - stock[2][3]);
}
else if (dJ <= stock[2][3])
{
stock[2][4] = stock[3][3];
}
}
else
stock[2][4] = 0;

//X_S^2
if (stock[2][4] > dV - stock[1][4])
{
if (dV > stock[1][4])
{
stock[1][5] = stock[2][4] - (dV - stock[1][4]);
}
else if (dV <= stock[1][4])
{
stock[1][5] = stock[2][4];
}
}
else
stock[1][5] = 0;

//X_D^1
if (stock[1][5] > dS - stock[0][5])
{
if (dS > stock[0][5])
{
stock[0][6] = stock[1][5] - (dS - stock[0][5]);
}
else if (dS <= stock[0][5])
{
stock[0][6] = stock[1][5];
}
}
else
stock[0][6] = 0;
```

```
// COSTE stock:

double stockTOTAL;
stockTOTAL = (stock[3][3] + stock[2][4] + stock[1][5] + stock[0][6]) * PM[dM] * PX[dX]
* PJ[dJ] * PV[dV] * PS[dS]* PD[dD];

Estock = Estock + stockTOTAL; //esperanza stockTOTAL


//Unidades que pido el Martes, me llegan el Miercoles y me caducan el Domingo
//O_M^S
int caducidad_X_D;

if (stock[0][6] > dD)
{
caducidad_X_D = stock[0][6] - dD;
}
else
{
caducidad_X_D = 0;
}


// COSTE caducidad

double caducidad_XD;
caducidad_XD = caducidad_X_D * PM[dM] * PX[dX] * PJ[dJ] * PV[dV] * PS[dS]* PD[dD];

Ecaducidad = Ecaducidad + caducidad_XD; //esperanza caducidad


// Creamos la variable de escasez de demanda en la etapa k = M
int escasez_demanda_X;
if(dX > decision_hoy + stock[0][2] + stock[3][2])
{
escasez_demanda_X = dX - decision_hoy - (stock[0][2] + stock[3][2]);
}
else if(dX <= decision_hoy + stock[0][2] + stock[3][2])
{
escasez_demanda_X = 0;
}


// COSTE escasez

double escasez_demandaX;
escasez_demandaX = escasez_demanda_X * PM[dM] * PX[dX] * PJ[dJ] * PV[dV] * PS[dS]* PD[dD];

Eescasez = Eescasez + escasez_demandaX; //esperanza escasez
```

```
}


// COSTE TOTAL

Ecoste = c * decision_hoy + h * Estock + p * Eescasez + theta * Ecaducidad;

if(nveces==1)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
else if(Ecoste<minimoCoste)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
}
}


// -------------------------------------------------


// MIERCOLES (K= X, K+1= J)


if (diasemana == 'X')
{
// Este bucle es para las y_k
for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy = decision_hoy + 2)
{
double Ecaducidad = 0;
double Eescasez = 0;
double Estock = 0;
double Ecoste = 0;

for (dX=mindX; dX <= maxdX; dX = dX + 2)
for (dJ=mindJ; dJ <= maxdJ; dJ = dJ + 2)
for (dV=mindV; dV <= maxdV; dV = dV + 2)
for (dS=mindS; dS <= maxdS; dS = dS + 2)
for (dD=mindD; dD <= maxdD; dD = dD + 2)
for (dL=mindL; dL <= maxdL; dL = dL + 2)
{
// Actualizamos stock:

// Veamos la ecuacion de transferencia para las u. de stock de 3 dias de vida
// restantes (tercer elemento del vector)
// X_J^3
```

```
if(stock[0][2] >= dX)
{
stock[2][3] = stock[3][2];
}
else if ((stock[0][2] < dX) && (dX < (stock[0][2] + stock[3][2])))
{
stock[2][3] = stock[3][2] - (dX - (stock[0][2]));
}
else
{
stock[2][3] = 0;
}

// Veamos la ecuacion de transferencia para las u. de stock de 4 dias de vida
// restantes (cuarto elemento del vector)
// X_J^4

if((stock[0][2] + stock[3][2]) >= dX)
{
stock[3][3] = decision_ayer;
}
else if (((stock[0][2] + stock[3][2]) < dX ) &&
(dX < (stock[0][2] + stock[3][2] + decision_ayer)))
{
stock[3][3] = decision_ayer - (dX - (stock[0][2] + stock[3][2]));
}
else
{
stock[3][3] = 0;
}

// El Miercoles NO hay u. de stock de 2 ni 3 dias de vida restantes, por tanto, el
// Jueves no habra u. de stock de 1 ni 2 dias de vida restantes


// Veamos las ecuaciones necesarias para hallar la caducidad

// HOY ES MIERCOLES, necesitamos las del Martes y otras
// Las del Martes necesarias

//X_J^4, ya esta escrita en ec. transferencia, justo antes

//X_V^3
if (stock[3][3] > dJ - stock[2][3])
{
if (dJ > stock[2][3])
{
stock[2][4] = stock[3][3] - (dJ - stock[2][3]);
}
else if (dJ <= stock[2][3])
```

```
{
stock[2][4] = stock[3][3];
}
}
else
stock[2][4] = 0;

//X_S^2, necesitamos aqui tambien X_V^2
//X_V^2
if (stock[2][3] > dJ)
{
stock[1][4] = stock[2][3] - dJ;
}
else
stock[1][4] = 0;
//X_S^2
if (stock[2][4] > dV - stock[1][4])
{
if (dV > stock[1][4])
{
stock[1][5] = stock[2][4] - (dV - stock[1][4]);
}
else if (dV <= stock[1][4])
{
stock[1][5] = stock[2][4];
}
}
else
stock[1][5] = 0;

//X_D^1, necesitamos X_S^1 para X_D^1
//X_S^1
if (stock[1][4] > dV)
{
stock[0][5] = stock[1][4] - dV;
}
else
stock[0][5] = 0;
//X_D^1
if (stock[1][5] > dS - stock[0][5])
{
if (dS > stock[0][5])
{
stock[0][6] = stock[1][5] - (dS - stock[0][5]);
}
else if (dS <= stock[0][5])
{
stock[0][6] = stock[1][5];
}
}
else
```

```
stock[0][6] = 0;

// Las de hoy Miercoles

//X_V^4
if (decision_hoy > dJ - (stock[2][3] + stock[3][3]))
{
if (dJ > stock[2][3] + stock[3][3])
{
stock[3][4] = decision_hoy - (dJ - (stock[2][3] + stock[3][3]));
}
else if (dJ <= stock[2][3] + stock[3][3])
{
stock[3][4] = decision_hoy;
}
}
else
stock[3][4] = 0;

//X_S^3
if (stock[3][4] > dV - (stock[1][4] + stock[2][4]))
{
if (dV > stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4] - (dV - (stock[1][4] + stock[2][4]));
}
else if (dV <= stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4];
}
}
else
stock[2][5] = 0;

//X_D^2
if (stock[2][5] > dS - (stock[0][5] + stock[1][5]))
{
if (dS > stock[0][5] + stock[1][5])
{
stock[1][6] = stock[2][5] - (dS - (stock[0][5] + stock[1][5]));
}
else if (dS <= stock[0][5] + stock[1][5])
{
stock[1][6] = stock[2][5];
}
}
else
stock[1][6] = 0;

//X_L^1, este Lunes es de la nueva semana
if (stock[1][6] > dD - stock[0][6])
```

```
{
if (dD > stock[0][6])
{
stock[0][0] = stock[1][6] - (dD - stock[0][6]);
}
else if (dD <= stock[0][6])
{
stock[0][0] = stock[1][6];
}
}
else
stock[0][0] = 0;


// COSTE stock:

double stockTOTAL;
stockTOTAL = (stock[3][4] + stock[2][5] + stock[1][6] + stock[0][0]) * PX[dX] * PJ[dJ]
* PV[dV] * PS[dS] * PD[dD] * PL[dL];

Estock = Estock + stockTOTAL; //esperanza stockTOTAL


//Unidades que pido el Miercoles, me llegan el Jueves y me caducan el Lunes
//O_J^L
int caducidad_J_L;

if (stock[0][0] > dL)
{
caducidad_J_L = stock[0][0] - dL;
}
else
{
caducidad_J_L = 0;
}


// COSTE caducidad

double caducidad_JL;
caducidad_JL = caducidad_J_L * PX[dX] * PJ[dJ] * PV[dV] * PS[dS] * PD[dD] * PL[dL];

Ecaducidad = Ecaducidad + caducidad_JL; //esperanza caducidad


// Creamos la variable de escasez de demanda en la etapa k = X
int escasez_demanda_J;

if(dJ > decision_hoy + stock[2][3] + stock[3][3])
{
escasez_demanda_J = dJ - decision_hoy - (stock[2][3] + stock[3][3]);
```

```
}
else if(dJ <= decision_hoy + stock[2][3] + stock[3][3])
{
escasez_demanda_J = 0;
}


// COSTE escasez

double escasez_demandaJ;
escasez_demandaJ = escasez_demanda_J * PX[dX] * PJ[dJ] * PV[dV] * PS[dS]
* PD[dD] * PL[dL];

Eescasez = Eescasez + escasez_demandaJ; //esperanza escasez
}


// COSTE TOTAL

Ecoste = c * decision_hoy + h * Estock + p * Eescasez + theta * Ecaducidad;

if(nveces==1)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
else if(Ecoste<minimoCoste)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
}
}


// ------------------------------------------------


// JUEVES (K= J, K+1= V)


if (diasemana == 'J')
{
// Este bucle es para las y_k
for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy = decision_hoy +2)
{
double Ecaducidad = 0;
double Eescasez = 0;
double Estock = 0;
```

```
double Ecoste = 0;

for (dJ=mindJ; dJ <= maxdJ; dJ = dJ + 2)
for (dV=mindV; dV<= maxdV; dV = dV + 2)
for (dS=mindS; dS <= maxdS; dS = dS + 2)
for (dD=mindD; dD <= maxdD; dD = dD + 2)
for (dL=mindL; dL <= maxdL; dL = dL + 2)
for (dM=mindM; dM <= maxdM; dM = dM + 2)
{
// Actualizamos stock:

// Veamos la ecuacion de transferencia para las u. de stock de 2 dias de vida
// restantes
//X_V^2
if (stock[2][3] > dJ)
{
stock[1][4] = stock[2][3] - dJ;
}
else
stock[1][4] = 0;

// Veamos la ecuacion de transferencia para las u. de stock de 3 dias de vida
// restantes
// X_V^3
if(stock[2][3] >= dJ)
{
stock[2][4] = stock[3][3];
}
else if ((stock[2][3] < dJ) && (dJ < (stock[2][3] + stock[3][3])))
{
stock[2][4] = stock[3][3] - (dJ - (stock[2][3]));
}
else
{
stock[2][4] = 0;
}

// Veamos la ecuacion de transferencia para las u. de stock de 4 dias de vida restantes
// (cuarto elemento del vector)
// X_V^4
if((stock[2][3] + stock[3][3]) >= dJ)
{
stock[3][4] = decision_ayer;
}
else if (((stock[2][3] + stock[3][3]) < dJ ) && (dJ < (stock[2][3] + stock[3][3]
+ decision_ayer)))
{
stock[3][4] = decision_ayer - (dJ - (stock[2][3] + stock[3][3]));
}
else
{
```

```
stock[3][4] = 0;
}

// El Jueves NO hay u. de stock de 2 ni 1 dias de vida restantes, por tanto, el
// Viernes no habra u. de stock de 1 dia de vida restante


// Veamos las ecuaciones necesarias para hallar la caducidad

// HOY ES JUEVES, necesitamos las del Miercoles y otras
// Las del Miercoles necesarias
// X_V^4, ya esta escrita en ec. transferencia, justo antes

// X_S^3
if (stock[3][4] > dV - (stock[1][4] + stock[2][4]))
{
if (dV > stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4] - (dV - (stock[1][4] + stock[2][4]));
}
else if (dV <= stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4];
}
}
else
stock[2][5] = 0;

// X_D^2, necesitamos X_S^1 y X_S^2 del L y M, respectivamente
//X_S^1
if (stock[1][4] > dV)
{
stock[0][5] = stock[1][4] - dV;
}
else
stock[0][5] = 0;
//X_S^2
if (stock[2][4] > dV - stock[1][4])
{
if (dV > stock[1][4])
{
stock[1][5] = stock[2][4] - (dV - stock[1][4]);
}
else if (dV <= stock[1][4])
{
stock[1][5] = stock[2][4];
}
}
else
stock[1][5] = 0;
```

```
//X_D^2
if (stock[2][5] > dS - (stock[0][5] + stock[1][5]))
{
if (dS > stock[0][5] + stock[1][5])
{
stock[1][6] = stock[2][5] - (dS - (stock[0][5] + stock[1][5]));
}
else if (dS <= stock[0][5] + stock[1][5])
{
stock[1][6] = stock[2][5];
}
}
else
stock[1][6] = 0;

// X_L^1, necesitamos X_D^1
//X_D^1
if (stock[1][5] > dS - stock[0][5])
{
if (dS > stock[0][5])
{
stock[0][6] = stock[1][5] - (dS - stock[0][5]);
}
else if (dS <= stock[0][5])
{
stock[0][6] = stock[1][5];
}
}
else
stock[0][6] = 0;

//X_L^1, este Lunes es de la nueva semana
if (stock[1][6] > dD - stock[0][6])
{
if (dD > stock[0][6])
{
stock[0][0] = stock[1][6] - (dD - stock[0][6]);
}
else if (dD <= stock[0][6])
{
stock[0][0] = stock[1][6];
}
}
else
stock[0][0] = 0;

// Las de hoy Jueves
// X_S^4
if (decision_hoy > dV - (stock[1][4] + stock[2][4] + stock[3][4]))
{
if (dV > stock[1][4] + stock[2][4] + stock[3][4])
```

```
{
stock[3][5] = decision_hoy - (dV - (stock[1][4] + stock[2][4] + stock[3][4]));
}
else if (dV <= stock[1][4] + stock[2][4] + stock[3][4])
{
stock[3][5] = decision_hoy;
}
}
else
stock[3][5] = 0;

//X_D^3
if (stock[3][5] > dS - (stock[0][5] + stock[1][5] + stock[2][5]))
{
if (dS > stock[0][5] + stock[1][5] + stock[2][5])
{
stock[2][6] = stock[3][5] - (dS - (stock[0][5] + stock[1][5] + stock[2][5]));
}
else if (dS <= stock[0][5] + stock[1][5] + stock[2][5])
{
stock[2][6] = stock[3][5];
}
}
else
stock[2][6] = 0;

//X_L^2, este LUNES es de la nueva semana
if (stock[2][6] > dD - (stock[0][6] + stock[1][6]))
{
if (dD > stock[0][6] + stock[1][6])
{
stock[1][0] = stock[2][6] - (dD - (stock[0][6] + stock[1][6]));
}
else if (dD <= stock[0][6] + stock[1][6])
{
stock[1][0] = stock[2][6];
}
}
else
stock[1][0] = 0;

//X_M^1, este MARTES es de la nueva semana
if (stock[1][0] > dL - stock[0][0])
{
if (dL > stock[0][0])
{
stock[0][1] = stock[1][0] - (dL - stock[0][0]);
}
else if (dL <= stock[0][0])
{
stock[0][1] = stock[1][0];
```

```
}
}
else
stock[0][1] = 0;


// COSTE stock:

double stockTOTAL;
stockTOTAL = (stock[3][5] + stock[2][6] + stock[1][0] + stock[0][1]) * PJ[dJ] * PV[dV]
* PS[dS] * PD[dD]* PL[dL] * PM[dM];

Estock = Estock + stockTOTAL; //esperanza stockTOTAL


//Unidades que pido el Jueves, me llegan el Viernes y me caducan el Martes
//O_V^M
int caducidad_V_M;

if (stock[0][1] > dM)
{
caducidad_V_M = stock[0][1] - dM;
}
else
{
caducidad_V_M = 0;
}


// COSTE caducidad

double caducidad_VM;
caducidad_VM = caducidad_V_M * PJ[dJ] * PV[dV] * PS[dS] * PD[dD]* PL[dL] * PM[dM];

Ecaducidad = Ecaducidad + caducidad_VM; //esperanza caducidad


// Creamos la variable de escasez de demanda en la etapa k = J
// En este caso, la decision que tomemos este dia nos influye en la
// escasez del Viernes, Sabado y Domingo

// Escasez Viernes
int escasez_demanda_V;
if(dV > decision_hoy + stock[1][4] + stock[2][4] + stock[3][4])
{
escasez_demanda_V = dV - decision_hoy - (stock[1][4] + stock[2][4] + stock[3][4]);
}
else if(dV <= decision_hoy + stock[1][4] + stock[2][4] + stock[3][4])
{
escasez_demanda_V = 0;
}
```

```
// Escasez Sabado
int escasez_demanda_S;
if(dS > stock[0][5] + stock[1][5] + stock[2][5] + stock[3][5])
{
escasez_demanda_S = dS - (stock[0][5] +  stock[1][5] + stock[2][5]
+ stock[3][5]);
}
else if(dS <= stock[0][5] + stock[1][5] + stock[2][5] + stock[3][5])
{
escasez_demanda_S = 0;
}

// Escasez Domingo
int escasez_demanda_D;
if(dD > stock[0][6] + stock[1][6] + stock[2][6])
{
escasez_demanda_D = dD - (stock[0][6] + stock[1][6] + stock[2][6]);
}
else if(dD <= stock[0][6] + stock[1][6] + stock[2][6])
{
escasez_demanda_D = 0;
}


// COSTE escasez

double escasez_demandaVSD;
escasez_demandaVSD = (escasez_demanda_V + escasez_demanda_S + escasez_demanda_D)
* PJ[dJ] * PV[dV] * PS[dS] * PD[dD]* PL[dL] * PM[dM];

Eescasez = Eescasez + escasez_demandaVSD; //esperanza escasez
}


// COSTE TOTAL

Ecoste = c * decision_hoy + h * Estock + p * Eescasez + theta * Ecaducidad;

if(nveces==1)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
else if(Ecoste<minimoCoste)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
```

```
}
}


// ------------------------------------------------


// VIERNES (K=V, K+1= S)


if (diasemana == 'V')
{
// Este bucle es para las y_k
for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy = decision_hoy + 2)
{
double Ecaducidad = 0;
double Eescasez = 0;
double Estock = 0;
double Ecoste = 0;

for (dV=mindV; dV <= maxdV; dV = dV +2)
for (dS=mindS; dS <= maxdS; dS = dS +2)
for (dD=mindD; dD <= maxdD; dD = dD +2)
for (dL=mindL; dL <= maxdL; dL = dL +2)
for (dM=mindM; dM <= maxdM; dM = dM +2)
for (dX=mindX; dX <= maxdX; dX = dX +2)
{
// Actualizamos stock:

// Veamos la ecuacion de transferencia para las u. de stock de 1 dia de vida restante
// X_S^1
if (stock[1][4] > dV)
{
stock[0][5] = stock[1][4] - dV;
}
else
stock[0][5] = 0;

// Veamos la ecuacion de transferencia para las u. de stock de 2 dias de vida restantes
//X_S^2
if(stock[1][4] >= dV)
{
stock[1][5] = stock[2][4];

}
else if ((stock[1][4] < dV) && (dV < (stock[1][4] + stock[2][4])))
{
stock[1][5] = stock[2][4] - (dV - (stock[1][4]));
}
else
{
```

```
stock[1][5] = 0;
}

// Veamos la ecuacion de transferencia para las u. de stock de 4 dias de vida restantes
// (cuarto elemento del vector)
// X_S^4
if((stock[1][4] + stock[2][4] + stock[3][4]) >= dV)
{
stock[3][5] = decision_ayer;
}
else if (((stock[1][4] + stock[2][4] + stock[3][4]) < dV ) && (dV < (stock[1][4]
+ stock[2][4] + stock[3][4] + decision_ayer)))
{
stock[3][5] = decision_ayer - (dV - (stock[1][4] + stock[2][4] + stock[3][4]));
}
else
{
stock[3][5] = 0;
}

// El Viernes no hay u. en stock de 1 dia de vida restante, pero esto no afecta
// al Sabado pues caducan

// Veamos las ecuaciones necesarias para hallar la caducidad

// HOY ES VIERNES, necesitamos las del Jueves y otras
// Las del Jueves necesarias y otras

// X_S^3
if (stock[3][4] > dV - (stock[1][4] + stock[2][4]))
{
if (dV > stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4] - (dV - (stock[1][4] + stock[2][4]));
}
else if (dV <= stock[1][4] + stock[2][4])
{
stock[2][5] = stock[3][4];
}
}
else
stock[2][5] = 0;

// X_D^2, necesitamos X_S^1 y X_S^2 del L y M, respectivamente
(ya estan en ec. transferencia)

//X_D^2
if (stock[2][5] > dS - (stock[0][5] + stock[1][5]))
{
if (dS > stock[0][5] + stock[1][5])
{
```

```
stock[1][6] = stock[2][5] - (dS - (stock[0][5] + stock[1][5]));
}
else if (dS <= stock[0][5] + stock[1][5])
{
stock[1][6] = stock[2][5];
}
}
else
stock[1][6] = 0;

// X_L^1, necesitamos X_D^1
//X_D^1
if (stock[1][5] > dS - stock[0][5])
{
if (dS > stock[0][5])
{
stock[0][6] = stock[1][5] - (dS - stock[0][5]);
}
else if (dS <= stock[0][5])
{
stock[0][6] = stock[1][5];
}
}
else
stock[0][6] = 0;
//X_L^1, este Lunes es de la nueva semana
if (stock[1][6] > dD - stock[0][6])
{
if (dD > stock[0][6])
{
stock[0][0] = stock[1][6] - (dD - stock[0][6]);
}
else if (dD <= stock[0][6])
{
stock[0][0] = stock[1][6];
}
}
else
stock[0][0] = 0;

//X_D^3
if (stock[3][5] > dS - (stock[0][5] + stock[1][5] + stock[2][5]))
{
if (dS > stock[0][5] + stock[1][5] + stock[2][5])
{
stock[2][6] = stock[3][5] - (dS - (stock[0][5] + stock[1][5] + stock[2][5]));
}
else if (dS <= stock[0][5] + stock[1][5] + stock[2][5])
{
stock[2][6] = stock[3][5];
}
```

```
}
else
stock[2][6] = 0;

//X_L^2, este LUNES es de la nueva semana
if (stock[2][6] > dD - (stock[0][6] + stock[1][6]))
{
if (dD > stock[0][6] + stock[1][6])
{
stock[1][0] = stock[2][6] - (dD - (stock[0][6] + stock[1][6]));
}
else if (dD <= stock[0][6] + stock[1][6])
{
stock[1][0] = stock[2][6];
}
}
else
stock[1][0] = 0;

//X_M^1, este MARTES es de la nueva semana
if (stock[1][0] > dL - stock[0][0])
{
if (dL > stock[0][0])
{
stock[0][1] = stock[1][0] - (dL - stock[0][0]);
}
else if (dL <= stock[0][0])
{
stock[0][1] = stock[1][0];
}
}
else
stock[0][1] = 0;


// Las de hoy Viernes

// X_M^2
if (decision_hoy > dL - (stock[0][0] + stock[1][0]))
{
if (dL > stock[0][0] + stock[1][0])
{
stock[1][1] = decision_hoy - (dL - (stock[0][0] + stock[1][0]));
}
else if (dL <= stock[0][0] + stock[1][0])
{
stock[1][1] = decision_hoy;
}
}
else
stock[1][1] = 0;
```

```
// X_X^1
if (stock[1][1] > dM - stock[0][1])
{
if (dM > stock[0][1])
{
stock[0][2] = stock[1][1] - (dM - stock[0][1]);
}
else if (dM <= stock[0][1])
{
stock[0][2] = stock[1][1];
}
}
else
stock[0][2] = 0;


// COSTE stock:

double stockTOTAL;
stockTOTAL = (stock[1][1] + stock[0][2]) * PV[dV] * PS[dS]* PD[dD]
* PL[dL]* PM[dM] * PX[dX];

Estock = Estock + stockTOTAL; //esperanza stockTOTAL


//Unidades que pido el Martes, me llegan el Miercoles y me caducan el Domingo
//O_M^S
int caducidad_L_X;

if (stock[0][2] > dX)
{
caducidad_L_X = stock[0][2] - dX;
}
else
{
caducidad_L_X = 0;
}


// COSTE caducidad

double caducidad_LX;
caducidad_LX = caducidad_L_X * PV[dV] * PS[dS]* PD[dD] * PL[dL]* PM[dM] * PX[dX];

Ecaducidad = Ecaducidad + caducidad_LX; //esperanza caducidad


// Creamos la variable de escasez de demanda en la etapa k = M
int escasez_demanda_L;
```

```
if(dL > decision_hoy + stock[0][0] + stock[1][0])
{
escasez_demanda_L = dL - decision_hoy - (stock[0][0] + stock[1][0]);
}
else if(dL <= decision_hoy + stock[0][0] + stock[1][0])
{
escasez_demanda_L = 0;
}


// COSTE escasez

double escasez_demandaL;
escasez_demandaL = escasez_demanda_L * PV[dV] * PS[dS] * PD[dD] * PL[dL]
* PM[dM] * PX[dX];

Eescasez = Eescasez + escasez_demandaL; //esperanza escasez
}


// COSTE TOTAL

Ecoste = c * decision_hoy + h * Estock + p * Eescasez + theta * Ecaducidad;

if(nveces==1)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
else if(Ecoste<minimoCoste)
{
minimoCoste = Ecoste;
decisionOptima = decision_hoy;
nveces++;
}
}
}


// ------------------------------------------------


printf("Ecoste = %lf \n", minimoCoste);

return decisionOptima;

}
```

B.2. Programa para programación dinámica

```
// Este programa calcula la decision optima de unidades que hay que producir de
// plaquetas dependiendo del dia de la semana (que es Lunes, Martes, ..., Domingo)
// mediante programacion dinamica

#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define MAX_DECISION 60 // numero de plaquetas como maximo que puedo producir en
// cada etapa ira de 0, 1, ..., MAX_DECISION
#define DIAS 6  // 6 representa los elementos del vector stock: 0, 1, 2, 3, 4, 5
#define DIASEMANA 7 // va de 0 a 6, donde 0 es L y 6 es D

// Definimos los costes aqui, asi en el programa solo aparecera c, h, p, theta.
// Se pueden cambiar los valores desde aqui
#define h 2
#define p 5000
#define theta 400

#define DIM 10001
#define DIM1 14
#define DIMP 200
#define NESTADOS 10000000

// Matrices, vectores y otros globales que necesitaremos
int X[NESTADOS][DIAS];
int Xfuturo[NESTADOS][DIAS];
double Jfuturo[NESTADOS];
double Jpresente[NESTADOS];
int stock[DIAS];
int stock_nuevo[DIAS];
int stocknuevo[DIAS];
int decOptimaEtapaPrimera[NESTADOS];
int i_selec;
int nestados,nestadosfuturo;

double PL[DIMP];
double PM[DIMP];
double PX[DIMP];
double PJ[DIMP];
double PV[DIMP];
double PS[DIMP];
double PD[DIMP];
double P[DIMP];

// Leemos el fichero donde se encunetran las distribuciones de una N(0,1)
void normalZ(double *v)
{
FILE *pleer;
```

```
double x;
int i=0;
pleer = fopen("ZZ.txt","r+");

fscanf(pleer, "%lf", &x);
while(feof(pleer)==0)
{
v[i] = x;
fscanf(pleer, "%lf", &x);
i++;
}

fclose(pleer);
}

// Leemos el fichero donde se encuentran las distribuciones de las demandas de
// cada dia de la semana (medias y desvtip):
void mediavar(double *w)
{
FILE *pleer;

double x;
int j=0;
pleer = fopen("mediavar3.txt","r+");

fscanf(pleer, "%lf", &x);
while(feof(pleer)==0)
{
w[j] = x;
fscanf(pleer, "%lf", &x);
j++;
}

fclose(pleer);
}


// Guardamos en un vector que se llame PL las probabilidades de la demanda del LUNES
void probaL(int *pmindL, int *pmaxdL)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dL, dLmax, mindL, maxdL;
     int jmin;
jmin=-5000;
```

```
int k,m;
double mu = w[0], sigma = w[7];

mindL = (int)(mu - 2.5 * sigma);
maxdL = (int)(mu + 4 * sigma);

for(dL=0; dL<=DIMP; dL++)
{
k=(int)((((dL+0.5)-mu)/sigma)*1000);
m=(int)((((dL-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dLmax=dL;
break;
}

PL[dL]=v[k-jmin]-v[m-jmin];

}

*pmindL=mindL;
*pmaxdL=maxdL;

return;
}

//  Guardamos en un vector que se llame PM las probabilidades de la demanda del MARTES
void probaM(int *pmindM, int *pmaxdM)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);


int dM,dMmax, mindM, maxdM;
int jmin;
jmin=-5000;
int k,m;

double mu = w[1], sigma = w[8];

mindM = (int)(mu - 2.5 * sigma);
maxdM = (int)(mu + 4 * sigma);

for(dM=0; dM<=DIMP; dM++)
```

```
{
k=(int)((((dM+0.5)-mu)/sigma)*1000);
m=(int)((((dM-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dMmax=dM;
break;
}

PM[dM]=v[k-jmin]-v[m-jmin];
}

*pmindM=mindM;
*pmaxdM=maxdM;

return;
}

//  Guardamos en un vector que se llame PX las probabilidades de la demanda del MIERCOLES
void probaX(int *pmindX, int *pmaxdX)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dX,dXmax, mindX, maxdX;
int jmin;
jmin=-5000;
int k,m;

double mu = w[2], sigma = w[9];

mindX = (int)(mu - 2.5 * sigma);
maxdX = (int)(mu + 4 * sigma);

for(dX=0; dX<=DIMP; dX++)
{
k=(int)((((dX+0.5)-mu)/sigma)*1000);
m=(int)((((dX-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dXmax=dX;
break;
}
```

```
PX[dX]=v[k-jmin]-v[m-jmin];
}

*pmindX=mindX;
*pmaxdX=maxdX;

return;
}
//  Guardamos en un vector que se llame PJ las probabilidades de la demanda del JUEVES
void probaJ(int *pmindJ, int *pmaxdJ)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);


int dJ,dJmax, mindJ, maxdJ;
int jmin;
jmin=-5000;
int k,m;

double mu = w[3], sigma = w[10];

mindJ = (int)(mu - 2.5 * sigma);
maxdJ = (int)(mu + 4 * sigma)+1;

for(dJ=0; dJ<=DIMP; dJ++)
{
k=(int)((((dJ+0.5)-mu)/sigma)*1000);
m=(int)((((dJ-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dJmax=dJ;
break;
}

PJ[dJ]=v[k-jmin]-v[m-jmin];
}

*pmindJ=mindJ;
*pmaxdJ=maxdJ;

return;
}
```

```
//  Guardamos en un vector que se llame PV las probabilidades de la demanda del VIERNES
void probaV(int *pmindV, int *pmaxdV)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);


int dV,dVmax, mindV, maxdV;
int jmin;
jmin=-5000;
int k,m;

double mu = w[4], sigma = w[11];

mindV = (int)(mu - 2.5 * sigma);
maxdV = (int)(mu + 4 * sigma)+1;

for(dV=0; dV<=DIMP; dV++)
{
k=(int)((((dV+0.5)-mu)/sigma)*1000);
m=(int)((((dV-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dVmax=dV;
break;
}

PV[dV]=v[k-jmin]-v[m-jmin];
}

*pmindV=mindV;
*pmaxdV=maxdV;

return;
}
//  Guardamos en un vector que se llame PS las probabilidades de la demanda del SABADO
void probaS(int *pmindS, int *pmaxdS)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);
```

```
int dS,dSmax, mindS, maxdS;
int jmin;
jmin=-5000;
int k,m;

double mu = w[5], sigma = w[12];

mindS = (int)(mu - 2.5 * sigma);
maxdS = (int)(mu + 4 * sigma)+1;

for(dS=0; dS<=DIMP; dS++)
{
k=(int)((((dS+0.5)-mu)/sigma)*1000);
m=(int)((((dS-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dSmax=dS;
break;
}

PS[dS]=v[k-jmin]-v[m-jmin];
}

*pmindS=mindS;
*pmaxdS=maxdS;

return;
}


//  Guardamos en un vector que se llame PD las probabilidades de la demanda del DOMINGO
void probaD(int *pmindD, int *pmaxdD)
{
//Llamamos al vector de la normal 0,1
double v[DIM];
normalZ(v);

//Llamamos al vector de media-varianza
double w[DIM1];
mediavar(w);

int dD,dDmax, mindD, maxdD;
int jmin;
jmin=-5000;
int k,m;

double mu = w[6], sigma = w[13];

mindD = (int)(mu - 2.5 * sigma);
```

```
maxdD = (int)(mu + 4 * sigma);


for(dD=0; dD<=DIMP; dD++)
{
k=(int)((((dD+0.5)-mu)/sigma)*1000);
m=(int)((((dD-0.5)-mu)/sigma)*1000);

if(k>=5000 || m>=5000)
{
dDmax=dD;
break;
}

PD[dD]=v[k-jmin]-v[m-jmin];
}

*pmindD=mindD;
*pmaxdD=maxdD;

return;
}

// -----------------------------------------

// Pasamos letras de la semana a numeros y viceversa
int pasaranumero (char diasemana)
{
if (diasemana == 'L')
return 0;
else if (diasemana == 'M')
return 1;
else if (diasemana == 'X')
return 2;
else if (diasemana == 'J')
return 3;
else if (diasemana == 'V')
return 4;
else if (diasemana == 'S')
return 5;
else if (diasemana == 'D')
return 6;
}

char pasaradia (int numerosemana)
{
if (numerosemana == 0)
return 'L';
else if (numerosemana == 1)
return 'M';
else if (numerosemana == 2)
```

```
return 'X';
else if (numerosemana == 3)
return 'J';
else if (numerosemana == 4)
return 'V';
else if (numerosemana == 5)
return 'S';
else if (numerosemana == 6)
return 'D';
}

// ---------------------------------------------
// Creamos un programa que nos halle el minimo de las componentes de un vector
double minimo(double v[])
{
int i;
double min = 1.0e20;
for(i=0; i<=sizeof(v); i++)
{
if(v[i]<min)
{
min = v[i];
}
}
return min;
}


// ------------------------------------------------------------------------


// Creamos un programa que nos halle la suma de todas las componentes de un vector
int suma(int v[], int dim)
{
int i;
int sum = 0;
for(i=0; i<dim; i++)
{
sum = sum + v[i];
}
return sum;
}

// ---------------------------------------------
// Funcion que me guarda el numero de etapas

void totaletapas(int *N)
{
printf("Introduce un entero que represente el numero de etapas: \nN =  ");
scanf("%i", N);
}
```

```
// ------------------------------------------------------------------------

// Funcion que me devuelve el dia de la semana de la ultima etapa siendo
// la ultima etapa N
void diaUltEtapa(int DS, int N, char *diasemana)
{
// Aqui guardamos el numero que corresponde con el dia de la semana de la ultima etapa
int DSnuevo, DSultetapa;
for(int j=0; j<N;j++)
{
if(DS!=6)
{
DSnuevo=DS+1;
DS=DSnuevo;
}
else
{
DSnuevo=0;
DS=DSnuevo;
}

if(j==N-1)
{
DSultetapa=DS;
}
}

*diasemana = pasaradia (DSultetapa);
}

// ------------------------------------------

// Funcion que me devuelve el dia de la semana anterior, es decir, si hoy es
// martes el anterior fue lunes
void anteriordia(char diasemanaViejo, int k, char *diasemana) // pasamos el diasemana de la e
{
int DS = pasaranumero(diasemanaViejo);
int DSnuevo;

if(DS!=0)
{
DSnuevo=DS-1;
DS=DSnuevo;
}
else
{
DSnuevo=6;
DS=DSnuevo;
}
```

```
*diasemana = pasaradia(DS);
}

// -------------------------------------------

// Creamos una funcion que nos genere todos los estados posibles
void estados(char diasemana)
{
int i, j0,j1,j2,j3,j4,j5;
int nreg1, nreg=0;
int ntot=0;
int sum;

if(diasemana=='L') //8489 ESTADOS
{
stock[3]=0;
stock[4]=0;
stock[5]=0;
for(j0=0; j0<=MAX_DECISION; j0+=1)
for(j1=0; j1<=MAX_DECISION; j1+=5)
for(j2=0; j2<=MAX_DECISION; j2+=5)
{
stock[0]=j0;
stock[1]=j1;
stock[2]=j2;

// total de pasadas
ntot++;

// Llamamos a la funcion suma

// Si la suma de unidades de plaquetas en almacen es mayor que 60, no creamos
// ese estado porque max.capac. es 60
sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]> 2*MAX_DECISION)
{
continue;
}

// Guardamos los vectores
for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}


else if(diasemana=='M') // 48373 ESTADOS
{
```

```
stock[2]=0;
stock[3]=0;
stock[5]=0;
for(j0=0; j0<=MAX_DECISION; j0+=1)
for(j1=0; j1<=MAX_DECISION; j1+=5)
for(j4=0; j4<=MAX_DECISION; j4+=1)
{
stock[0]=j0;
stock[1]=j1;
stock[4]=j4;

ntot++;

sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}


else if(diasemana=='X') // 26047 ESTADOS
{
stock[1]=0;
stock[2]=0;
stock[5]=0;
for(j0=0; j0<=MAX_DECISION; j0+=1)
for(j3=0; j3<=MAX_DECISION; j3+=10)
for(j4=0; j4<=MAX_DECISION; j4+=1)
{
stock[0]=j0;
stock[3]=j3;
stock[4]=j4;

ntot++;

sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];
```

```
nreg++;
}
}


else if(diasemana=='J') // 5551 ESTADOS
{
stock[0]=0;
stock[1]=0;
stock[5]=0;
for(j2=0; j2<=MAX_DECISION; j2+=5)
for(j3=0; j3<=MAX_DECISION; j3+=10)
for(j4=0; j4<=MAX_DECISION; j4+=1)
{
stock[2]=j2;
stock[3]=j3;
stock[4]=j4;

ntot++;

sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}


else if(diasemana=='V') // 59780 ESTADOS
{
stock[0]=0;
stock[5]=0;
for(j1=0; j1<=MAX_DECISION; j1+=5)
for(j2=0; j2<=MAX_DECISION; j2+=5)
for(j3=0; j3<=MAX_DECISION; j3+=10)
for(j4=0; j4<=MAX_DECISION; j4+=1)
{
stock[1]=j1;
stock[2]=j2;
stock[3]=j3;
stock[4]=j4;

ntot++;

sum=suma(stock, DIAS);
```

```
if(sum-stock[4]-stock[5]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}


else if(diasemana=='S') //  2224670 ESTADOS
{
stock[4]=0;
for(j0=0; j0<=MAX_DECISION; j0+=1)
for(j1=0; j1<=MAX_DECISION; j1+=5)
for(j2=0; j2<=MAX_DECISION; j2+=5)
for(j3=0; j3<=MAX_DECISION; j3+=10)
for(j5=0; j5<=MAX_DECISION; j5+=1)
{
stock[0]=j0;
stock[1]=j1;
stock[2]=j2;
stock[3]=j3;
stock[5]=j5;

ntot++;

sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}


else if(diasemana=='D') //  517829 ESTADOS
{
stock[3]=0;
stock[4]=0;
for(j0=0; j0<=MAX_DECISION; j0+=1)
for(j1=0; j1<=MAX_DECISION; j1+=5)
```

```
for(j2=0; j2<=MAX_DECISION; j2+=5)
for(j5=0; j5<=MAX_DECISION; j5+=1)
{
stock[0]=j0;
stock[1]=j1;
stock[2]=j2;
stock[5]=j5;

ntot++;

sum=suma(stock, DIAS);
if(sum-stock[5]-stock[4]>2*MAX_DECISION)
{
continue;
}

for(i=0;i<DIAS;i++)
X[nreg][i]=stock[i];

nreg++;
}
}
nestados = nreg;
}

// ----------------------------------------

// Esta funcion nos calcula el estado de la etapa k+1, dado el estado k
void transferencia (char diasemana, int decision_hoy, int demanda, int *resta)
{
double sum, i;

// Definimos las ecuaciones de transferencia
// X_{k+1}^1
if(stock[0] >= demanda)
{
stock_nuevo[0] = stock[1];
}
else if (stock[0] < demanda && demanda < stock[0] + stock[1])
{
stock_nuevo[0] = stock[1] - (demanda - stock[0]);
}
else
stock_nuevo[0] = 0;

// X_{k+1}^2
if(stock[0] + stock[1] >= demanda)
{
stock_nuevo[1] = stock[2];
}
else if (stock[0] + stock[1] < demanda && demanda < stock[0] + stock[1] + stock[2])
```

```
{
stock_nuevo[1] = stock[2] - (demanda - (stock[0] + stock[1]));
}
else
stock_nuevo[1] = 0;

// X_{k+1}^3
if(diasemana=='L' || diasemana=='M' || diasemana=='X' || diasemana=='J' ||
diasemana=='V' || diasemana=='S')
{
if(stock[0] + stock[1] + stock[2] >= demanda)
{
stock_nuevo[2] = stock[3];
}
else if (stock[0] + stock[1] + stock[2] < demanda && demanda < stock[0] + stock[1]
 + stock[2] + stock[3])
{
stock_nuevo[2] = stock[3] - (demanda - (stock[0] + stock[1] + stock[2]));
}
else
stock_nuevo[2] = 0;
}
else if (diasemana == 'D')
{
stock_nuevo[2] = stock[5];
}

// X_{k+1}^4
if(stock[0] + stock[1] + stock[2] + stock[3] >= demanda)
{
stock_nuevo[3] = stock[4];
}
else if (stock[0] + stock[1] + stock[2] + stock[3] < demanda && demanda < stock[0]
+ stock[1] + stock[2] + stock[3] + stock[4])
{
stock_nuevo[3] = stock[4] - (demanda - (stock[0] + stock[1] + stock[2] + stock[3]));
}
else
stock_nuevo[3] = 0;

// X_{k+1}^5 este elemento del vector corresponde a las unidades pedidas
// el dia k (decision optima)
if (diasemana == 'L' ||  diasemana == 'M' || diasemana == 'X' || diasemana == 'J')
{
stock_nuevo[4] = decision_hoy;
}
else if (diasemana == 'V' || diasemana == 'S' || diasemana == 'D')
{
stock_nuevo[4] = 0;
}
```

```
 // X_{k+1}^6, este elemento del vector stock corresponde a las unidades que faltan por llega
if(diasemana == 'L' ||  diasemana == 'M' || diasemana == 'X' || diasemana == 'J' || diasemana
{
stock_nuevo[5] = 0;
}
else if (diasemana == 'V')
{
stock_nuevo[5] = decision_hoy;
}
else if (diasemana == 'S')
{
stock_nuevo[5] = stock[5];
}


// Tenemos ahora el estado nuevo, pero debemos modificarlo un poco porque existe la posibilid
// un estado posible en la etapa k+1


stocknuevo[1]= 5 * round(stock_nuevo[1] /5);  // No hace falta round int/int = int
stocknuevo[2]= 5 * round(stock_nuevo[2] /5);
stocknuevo[3]= 10 * (stock_nuevo[3] /10);    // Redondeamos pues esa componente es multiplo d
stocknuevo[0]=stock_nuevo[0];
stocknuevo[4]=stock_nuevo[4];
stocknuevo[5]=stock_nuevo[5];


int resta1=0;

// Sumamos las primeras cuatro componentes del vector
int stocknuevoReal = stocknuevo[0]+stocknuevo[1]+stocknuevo[2]+stocknuevo[3];


if(stocknuevoReal > 2*MAX_DECISION)
{

resta1 = stocknuevoReal - 2*MAX_DECISION;

if(stocknuevo[0] > 0)
{
if(stocknuevo[0] >= resta1)
{
stocknuevo[0]= stocknuevo[0] - resta1;
}
else
{
stocknuevo[0]=0;
resta1 = resta1-stock_nuevo[0];

if(stocknuevo[1] >= resta1)
{
```

```
stocknuevo[1] = stocknuevo[1]-resta1;
}
else
{
stocknuevo[1]=0;
}
}
}
else if (stocknuevo[0] ==0 && stocknuevo[1] > 0)
{
if(stocknuevo[1] >= resta1)
{
stocknuevo[1]=stocknuevo[1]-resta1;
}
else
{
stocknuevo[1]=0;
}
}
}
*resta=resta1;
}

// ---------------------------------------------

//Funcion compara
int compara(int donde, int k, int decision_hoy)
{
int valor;
valor=0;
for(int j=0; j<DIAS && valor==0; j++)
{
if(Xfuturo[donde][j]>stocknuevo[j]) valor=1;
if(Xfuturo[donde][j]>stocknuevo[j]) valor=-1;

}

return valor;
}
// ----------------------------------------

// Programa que nos realiza los calculos pertinentes para poder tomar una decision optima (et

void dinamica(char diasemana, int k, int N)
{
int nreg, resta;

// Declaramos variables
int decision_hoy, demanda, inf, sup;
int i,nveces=1;
int decisionOptima;
```

```
// Llamamos a las funciones donde estan los vectores de probabilidades de las demandas de cad

int maxdL, mindL;

probaL(&mindL,&maxdL);

// ------------------------------

int maxdM, mindM;

probaM(&mindM,&maxdM);

// ------------------------------

int maxdX, mindX;

probaX(&mindX,&maxdX);

// ------------------------------

int maxdJ, mindJ;

probaJ(&mindJ,&maxdJ);

// ------------------------------

int maxdV, mindV;

probaV(&mindV,&maxdV);

// ------------------------------

int maxdS, mindS;

probaS(&mindS,&maxdS);

// ------------------------------

int maxdD, mindD;

probaD(&mindD,&maxdD);

// ------------------------------------------------------------

if(diasemana=='L')
{
inf=mindL;
sup=maxdL;
for(int i=mindL; i<=maxdL; i++)
P[i]=PL[i];
```

```
}
else if(diasemana=='M')
{
inf=mindM;
sup=maxdM;
for(int i=mindM; i<=maxdM; i++)
P[i]=PM[i];
}
else if(diasemana=='X')
{
inf=mindX;
sup=maxdX;
for(int i=mindX; i<=maxdX; i++)
P[i]=PX[i];
}
else if(diasemana=='J')
{
inf=mindJ;
sup=maxdJ;
for(int i=mindJ; i<=maxdJ; i++)
P[i]=PJ[i];
}
else if(diasemana=='V')
{
inf=mindV;
sup=maxdV;
for(int i=mindV; i<=maxdV; i++)
P[i]=PV[i];
}
else if(diasemana=='S')
{
inf=mindS;
sup=maxdS;
for(int i=mindS; i<=maxdS; i++)
P[i]=PS[i];
}
else if(diasemana=='D')
{
inf=mindD;
sup=maxdD;
for(int i=mindD; i<=maxdD; i++)
P[i]=PD[i];
}

// Llamada a la funcion estados()
estados(diasemana);

// me devuelve una decisionOptima y un coste J que depende del dia de la semana
// y del stock

// Definimos variables
```

```
double minimoCoste;
int stockTOTAL, l0, l1, contador, donde, resultado;
double costeAlmacen;
int escasez, caducidad;
double costeDecision, valorCoste;
double coste=0;
double costeEscasez;
double costeCaducidad;
double costeSobrantes;

// costeFuturo es el valor que hay en el vector Jfuturo en la posicion del
// estado nreg
double costeJFuturo;
// Recorremos todos los estados posibles

for(nreg=0; nreg<nestados; nreg++)
{
minimoCoste = 1.0e20;
for(i=0;i<DIAS;i++)
stock[i]=X[nreg][i];


// Stock total y coste de dicho stock

stockTOTAL = (stock[0] + stock[1] + stock[2] + stock[3]);
costeAlmacen = h * stockTOTAL;


for (decision_hoy=0; decision_hoy <= MAX_DECISION; decision_hoy ++)
{
coste=0;

for(demanda=inf; demanda<=sup; demanda++)
{
// Escasez y costeEscasez
if(demanda > stock[4] + stockTOTAL)
{
escasez = demanda - stock[4] - stockTOTAL;
}
else if (demanda<=stock[4] + stockTOTAL)
{
escasez = 0;
}

costeEscasez = p * escasez;


// Caducidad y costeCaducidad
if(stock[0] > demanda)
{
caducidad = stock[0] - demanda;
```

```
}
else
{
caducidad = 0;
}

costeCaducidad = theta * caducidad;

// Llamada a la funcion transferencia() me calcula el estado de la etapa k+1
transferencia (diasemana, decision_hoy, demanda, &resta);

costeSobrantes = theta * resta;


if(k!=N)
{
i_selec = -1;
// Seleccionamos la posicion del vector stocknuevo
// Metodo de biseccion
l0=0;
l1=nestadosfuturo-1;

for (int iteraciones=1; iteraciones<100 &&i_selec<0; iteraciones++)
{
donde = (l1-l0)/2 +l0;

int valor=0;
for(int j=0; j<DIAS && valor==0; j++)
{
if(Xfuturo[donde][j]>stocknuevo[j])
{
valor=1;
}


if(Xfuturo[donde][j]<stocknuevo[j])
{
valor=-1;
}
}

if(valor==0) i_selec=donde;
if(valor==1) l1=donde-1;
if(valor==-1) l0=donde+1;
if(iteraciones>100)
{
printf("error");
system("pause");
}
}
```

```
costeJFuturo = Jfuturo[i_selec];
}

else
{
costeJFuturo = 0;
}

valorCoste = costeDecision + costeAlmacen + costeEscasez + costeCaducidad
+ costeJFuturo + costeSobrantes;

coste = coste + valorCoste * P[demanda];

}


if(coste<minimoCoste)
{
decisionOptima = decision_hoy;
minimoCoste = coste;
}

}

Jpresente[nreg] = minimoCoste;

if(k==0)
{
decOptimaEtapaPrimera[nreg] = decisionOptima;
}
}

nestadosfuturo=nestados;
for(nreg=0; nreg< nestados; nreg++)
{
Jfuturo[nreg] = Jpresente[nreg];
for(i=0;i<DIAS;i++)
Xfuturo[nreg][i] = X[nreg][i];
}
}

// ------------------------------------------------------------------------


// Calculos primera etapa
void primeraEtapa(char diasemana)
{
// Pedimos lo necesario y conocido por el trabajador por pantalla solo para la primera etapa

if(diasemana == 'L' || diasemana =='M' || diasemana =='X' || diasemana =='S'
|| diasemana =='D')
```

```
{
printf("Introduce unidades con 1 dia de vida restante: \n");
scanf("%i", &stock[0]);
}
else if (diasemana=='J' || diasemana=='V')
{
stock[0]=0;
}

if(diasemana == 'L' || diasemana =='M' || diasemana =='V' || diasemana =='S'
|| diasemana =='D')
{
printf("Introduce unidades con 2 dias de vida restantes: \n");
scanf("%i", &stock[1]);
}
else if (diasemana=='X' || diasemana =='J')
{
stock[1]=0;
}

if(diasemana =='J' || diasemana =='V' || diasemana =='S' || diasemana =='D')
{
printf("Introduce unidades con 3 dias de vida restantes: \n");
scanf("%i", &stock[2]);
}
else if(diasemana== 'M' || diasemana=='X')
{
stock[2]=0;
}

if(diasemana == 'X' || diasemana =='J' || diasemana =='V' || diasemana =='S')
{
printf("Introduce unidades con 4 dias de vida restantes: \n");
scanf("%i", &stock[3]);
}
else if (diasemana=='L' || diasemana =='M' || diasemana=='D')
{
stock[3]=0;
}



if(diasemana == 'L')
{
printf("Introduce unidades que llegan hoy: \n");
scanf("%i", &stock[2]);
}

if(diasemana =='M' || diasemana =='X' || diasemana =='J' || diasemana =='V')
{
printf("Introduce unidades que llegan hoy: \n");
```

```
scanf("%i", &stock[4]);
}
else if(diasemana=='L' || diasemana=='S' || diasemana == 'D')
{
stock[4]=0;
}

if(diasemana =='S' || diasemana =='D')
{
printf("Introduce unidades que estan por llegar, pero no llegan hoy: \n");
scanf("%i", &stock[5]);
}
else if (diasemana =='L' || diasemana == 'M' || diasemana =='X' || diasemana =='J'
|| diasemana == 'V')
{
stock[5]=0;
}

// Tenemos ahora el estado nuevo, pero debemos modificarlo un poco porque
// existe la posibilidad de que no sea un estado posible en la etapa k+1

stock[1]= 5 * round(stock[1] /5);
stock[2]= 5 * round(stock[2] /5);
stock[3]= 10 * (stock[3] /10);

i_selec = -1;
// Seleccionamos la posicion del vector stocknuevo
// Metodo de biseccion
int l0=0;
int l1=nestadosfuturo-1;
int donde;

for (int iteraciones=1; iteraciones<100 &&i_selec<0; iteraciones++)
{
donde = (l1-l0)/2 +l0;
int valor=0;
for(int j=0; j<DIAS && valor==0; j++)
{
if(Xfuturo[donde][j]>stock[j])
{
valor=1;
}

if(Xfuturo[donde][j]<stock[j])
{
valor=-1;
}
}

if(valor==0) i_selec=donde;
if(valor==1) l1=donde-1;
```

```
if(valor==-1) l0=donde+1;
if(iteraciones>100)
{
printf("error");
system("pause");
}
}
}

// ------------------------------------------

// PROGRAMA PRINCIPAL

main()
{
char diasemana, diasemanaViejo;
int N;

printf("Introduce el dia de hoy (EN MAYUSCULAS), siendo\n L, lunes;\n M, martes;\n
 X, miercoles;\n J, jueves;\n V, viernes; \n S, sabado; NO SE PRODUCE HOY\n D, domingo: NO SE
scanf("%c", &diasemana);
int DS = pasaranumero(diasemana);

// Llamamos a la funcion que nos guarda el valor N
totaletapas(&N);


// Recorremos todas las etapas
for(int k = N; k >=0; k--)
{
printf("Etapa: %i \n", k);
if(k!=N)
{
//Llamamos a la funcion que nos halla el dia de la semana de ayer
anteriordia(diasemanaViejo, k, &diasemana);
printf("Dia semana etapa %i es %c \n", k, diasemana);
}
else
{
// Llamamos a la funcion que me dice que dia de la semana es la ultima etapa
diaUltEtapa(DS, N, &diasemana);
printf("Dia semana etapa N=%i : %c \n", k, diasemana);
}

// Llamamos a la funcion que me da el valor de una decision optima y de su coste asociado
dinamica(diasemana, k, N);

// Guardamos diasemana como diasemanaViejo
diasemanaViejo = diasemana;
```

```
if(k==0)
{
for(int z=0; z< 130;z++)
{
primeraEtapa(diasemana);
if(diasemana=='S' || diasemana =='D')
{
printf("NO SE PRODUCE HOY \n");
}
else
{
printf("La decision optima para el estado: %i %i %i %i %i %i \n es %i \n con coste
%lf \n", stock[0], stock[1], stock[2], stock[3], stock[4], stock[5],
 decOptimaEtapaPrimera[i_selec], Jfuturo[i_selec]);
}
}
}
}
}
```