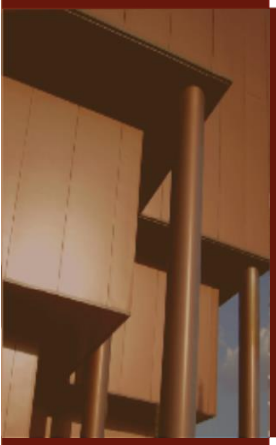
Trabajo de Fin de Grado

Grado en Ingeniería de las Tecnologías Industriales

Diseño e implementación de un sistema de optimización del inventario de piezas de repuesto

Autor: Ángel León López

Tutor: Pedro Luis González Rodríguez



Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Diseño e implementación de un sistema de optimización del inventario de piezas de repuesto

Autor:
Ángel León López

Tutor:
Pedro Luis González Rodríguez
Catedrático de Universidad

Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: Diseño e implementación de un sistema de optimización del inventario de piezas de repuesto

Autor: Ángel León López

Tutor: Pedro Luis González Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Dar las gracias a todo el mundo que ha estado detrás de mí apoyándome, ya no sólo en este trabajo, sino durante todo el tiempo que he estado cursando el grado.

Mención especial a familia y amigos, que me han ayudado en incontables ocasiones con ánimos y consejos que han hecho que nunca bajase los brazos.

Por supuesto, también quería agradecer a mi tutor Pedro Luis toda la ayuda proporcionada durante el desarrollo del trabajo, siempre ha sacado un tiempo para mí y, gracias a eso, he sido capaz de completar el proyecto.

Ángel León López

Sevilla, 2021

Resumen

En este proyecto, abordamos el diseño y desarrollo de un sistema para la optimización de los parámetros asociados al sistema de gestión de piezas de repuesto. Estas cuentan con unas particularidades que las hacen diferentes al resto y que debemos tener en cuenta. El trabajo consta de una completa memoria en la cual se detallan todos y cada uno de los puntos del sistema elaborado.

En primer lugar, se analiza la problemática existente en estos sistemas de gestión, a su vez, se definen las características que estos tienen debido al uso de piezas de repuesto. En segundo lugar, realizamos un análisis sobre cómo tratar el problema relativo a la optimización de los parámetros, exponiendo la metodología elegida para ello.

A continuación, se detalla el código desarrollado en Python. Tanto la interfaz gráfica como la optimización están realizadas en este lenguaje de programación. Además, explicamos cual es la función de cada una de las partes que conforman dicho código.

Por último, se realiza una comparación teórica-práctica mediante la generación de múltiples ejemplos. Esto nos ayuda a comprender el funcionamiento del sistema que se ha desarrollado.

Abstract

In this project, we will design and develop a parameter optimization system related to a spare parts management system. They have distinctive feature which makes them different and must be taken into account. The assignment is based on a long report in which all the system's details will be described.

At first, we analyse the main problems and define the spare parts management system characteristics we can find. Secondly, we explain how the problem related to the parameter's optimization has been solved, showing the applied methods.

Next, we disclose the code developed in Python. The graphic interface and the optimization code are written in this programming language. Moreover, we explain all the code parts functions.

Finally, we make a technical comparison between the theory and the practical. We use several examples to help us to understand how the developed system works.

Índice

Índice de figuras

Notación

D	Unidades demandas por período
Q	Tamaño del pedido
s	Punto de pedido
PA	Plazo de aprovisionamiento
v	Coste unitario de producto
ss	Stock de seguridad
r	Tasa aplicada a mantener el inventario
A	Coste de pedir
μ	Media
σ	Desviación típica
cv	Coeficiente de variación
z	Estadístico

1 JUSTIFICACIÓN Y OBJETIVO DEL PROYECTO

La necesidad del diseño y la creación de sistemas que gestionen el inventario eficiente y, sobre todo, óptimamente, es algo que cada vez es más evidente en un mercado cada vez más competitivo. Pero ¿qué pasaría si ese inventario estuviese compuesto por piezas de repuesto?, pues que esa necesidad sería todavía mayor.

Las piezas de repuesto son tratadas como si fueran un producto común dentro de los sistemas de gestión, y haciendo eso, se estaría cometiendo un gran error. La demanda de estas piezas cuenta con unas particularidades que hacen que en el diseño del sistema se deba tener en cuenta una serie de factores que nos harán actuar de diferente manera.

En la actualidad existen muchas empresas que se dedican a comercializar este tipo de piezas. En nuestro proyecto se diseñará un sistema que consiga adaptarse a esas características y pueda optimizar de la mejor manera el inventario de las piezas de repuesto.

1.1 Introducción y Justificación

La necesidad de estos sistemas de gestión de inventario es vital en cualquier tipo de entidad, ya sea en un ámbito interno o externo, es decir, para gestionar las piezas de repuesto que necesite en cada momento una empresa para el uso propio o, para controlar los niveles de inventario que debe tener una compañía que se dedique a la comercialización de este tipo de producto.

Existen multitud de empresas cuyo principal, o en ocasiones único, tipo de pieza que comercializan son las de repuesto. Debemos destacar que estas piezas no sólo se usarán como productos para la venta, es decir, tal y como se comentó al principio, internamente en la propia empresa también se requerirán del uso de estas como podría ser dentro de un sistema de gestión de mantenimiento impuesto en la compañía.

Este tipo de piezas, las cuales estudiaremos a lo largo de este trabajo, constan se una serie de características que las hacen diferentes al resto. Una de ellas es la difícil previsión de una demanda que es muy intermitente y desigual [1]. Otro punto que se debe destacar es la necesidad que tendrán los clientes por obtener un elemento de este tipo, es decir, no debemos olvidar que son piezas de repuesto, por lo que seguramente la persona que nos realiza un pedido necesitará que este se complete en el menor tiempo posible, por lo que cumplir con esto será fundamental para mantener el nivel de servicio requerido.

Según una investigación realizada en la Universidad de Brescia (Italia) [2], el negocio de las piezas de repuesto maneja cantidades de alrededor de 200 mil millones de dólares en todo el mundo. Por lo que la necesidad de un estudio y análisis, con un enfoque en este tipo de piezas, es fundamental para conseguir ser lo más eficiente posible en un mercado cada vez más competitivo.

Otro dato que hace resaltar la necesidad de este tipo de sistemas es que las investigaciones sobre la gestión de este tipo de piezas se han visto aumentadas considerablemente en la última década, llegándose a alcanzar cifras relacionadas con la venta de software de sistemas de gestión de piezas de repuesto las cuales rondaban los 100 millones de dólares anuales.

A pesar de todas las inversiones que se han venido realizando, todavía se cometen algunos errores vitales. Uno de los principales que tiene lugar en muchos de los sistemas de gestión del inventario es el de tratar a todas las piezas envueltas en este de la misma forma, sin distinguir a una de otra y, sin embargo, aplicando una gestión diferenciada podríamos obtener mejores resultados. Como ya se ha comentado previamente, las piezas de repuesto tienen unas características especiales que hacen que sea necesario la creación de un sistema enfocado únicamente en ellas.

Desde un punto de vista más personal, mi interés en el proyecto abordado radica en los siguientes puntos:

- Aplicación de los conocimientos adquiridos teóricamente durante el grado en un ámbito más práctico y fiel a la realidad.
- En el proyecto se hará uso del lenguaje de programación Python, que empecé a usar en la escuela y que, gracias a este trabajo, podré seguir aprendiendo y aumentando mis conocimientos sobre él.
- También, se necesitarán aplicar conceptos aprendidos sobre el diseño de sistemas productivos, así como algunos puntos vistos sobre la logística dentro de la empresa. Las asignaturas en las cuales se imparten este contenido son muy teóricas, y creo que el trabajo le da un enfoque práctico fundamental para llegar a comprender los conceptos a la perfección.
- Otro punto que me llama la atención y me animó a realizar el proyecto, es cómo la gestión de piezas de repuesto puede llegar a ser tan diferente si la comparamos con la de cualquier otro tipo de producto.
- Por último, añadir que la necesidad de diseñar y desarrollar una aplicación hace del proyecto algo todavía más atractivo, puesto que es algo nuevo para mí y creo que aprender la manera en la que se puede llegar a montar una interfaz de este tipo, es importante de cara al futuro.

1.2 Objetivo General

Nuestro principal objetivo en este proyecto es el diseño y la implementación de un sistema de optimización de un inventario compuesto por piezas de repuesto. Estos tipos de sistemas tienen una serie de características que los hacen diferentes a los demás, por lo que para el desarrollo del sistema se debe tener en cuenta una serie de condicionantes para adecuarlo a lo que se pretende lograr.

El problema principal que está presente en el diseño de nuestro sistema es la difícil predicción de la frecuencia con la cual se suelen realizar los pedidos, además de la cantidad demandada en cada uno de ellos, que en la mayoría de los casos no supera la unidad.

1.3 Objetivos Específicos

Para llegar a lograr el objetivo general necesitaremos ir paso por paso, e ir analizando cada una de las partes que forman estos tipos de sistemas. Para ello estableceremos una serie de metas que se deberán de ir desarrollando y alcanzando para conseguir comprender a la perfección estos tipos de sistemas, lo que nos llevará a ser capaces de diseñarlos e implementarlos. A continuación, expondremos los diferentes objetivos específicos que se han decidido establecer en nuestro proyecto.

1. Entender la problemática asociada a los sistemas de gestión de inventario, enfocándonos como no podía ser de otra manera, es los de piezas de repuesto. En definitiva, se realizará una profundización sobre el porqué es necesario un estudio sobre estos tipos de procedimientos.
2. Una vez comprendida la dificultad que estos tipos de problemas acarrean, tocará proponer una metodología para la optimización de todas las variables y todos los parámetros que intervendrán en el sistema de gestión de las piezas de repuesto.
3. Una vez que se ha establecido la forma en la que se abordará el problema, deberemos de implementarlo informáticamente haciendo uso del lenguaje de programación. Para ello será necesario diseñar y desarrollar una aplicación que sustente los requisitos establecidos.
4. Por último, se deberá realizar una comparación teórico-práctica para de esa forma, conseguir explicar el uso que tiene nuestro sistema y analizar las diferencias con las que nos encontraremos al cotejarlo con las estimaciones teóricas habituales.

1.4 Estructura del Documento

Para facilitar la lectura del documento, al inicio del mismo, se han establecido una serie de apartados que nos ayudarán a comprender de una mejor manera los contenidos y la estructura de este.

En primer lugar, nos encontraremos con un breve resumen para irnos introduciendo con qué nos encontraremos dentro del proyecto, escribiéndolo también en inglés para que más personas lo puedan leer. Si continuamos avanzando veremos un complejo índice donde podremos comprobar cada uno de los puntos en los que se divide nuestra memoria. Además, contaremos también con un índice de figuras para tener identificada a todas las ilustraciones en una sola lista. Otro punto que debemos destacar será el apartado de notaciones donde detallaremos brevemente el significado de cada una de las abreviaciones usadas a lo largo del trabajo.

Entrando en la parte del desarrollo del proyecto nos encontramos con 4 capítulos, que analizaremos a continuación:

- En el primer capítulo explicaremos cómo funcionan los sistemas de gestión de inventario y describiremos los tipos con los que nos podemos encontrar. También, hablaremos y profundizaremos en la problemática existente en el tipo de sistema que abordaremos en nuestro proyecto, es decir, los de piezas de repuesto. Esto último es uno de los puntos más importantes de este trabajo, puesto que la justificación del problema es fundamental para

entender por qué se ha llevado a cabo el proyecto.

- En el siguiente apartado realizaremos un análisis sobre cómo pensamos abordar el problema. Explicaremos la forma en la que pretendemos diseñar e implementar el sistema basado en las piezas de repuesto.
- Si en el capítulo anterior se explicó cómo se pretendía resolver el problema, en este desarrollaremos la forma en la que se debe aplicar la metodología propuesta. Se detallará como se ha implementado la optimización del sistema mediante una fase de análisis de las distribuciones de los datos históricos de entrada, para su posterior uso en un sistema de optimización basado en una simulación de Montecarlo.
- En el siguiente apartado realizaremos una comparación teórico-práctica para comprobar las diferencias con las estimaciones que se han venido haciendo en los últimos tiempos.
- Finalmente, en el último capítulo se completa la memoria con una conclusión sobre el proyecto realizado. En él, analizamos los objetivos establecidos al principio de este para posteriormente detallar que es lo que se ha cumplido y que es lo que no se ha sido capaz de completar.

Cuando se termine de desarrollar todos y cada uno de los puntos detallados previamente, completaremos el apartado de las referencias, donde encontraremos toda la bibliografía citada a lo largo de la memoria.

Para finalizar, añadimos en el anexo los códigos desarrollados para llevar a cabo la optimización de nuestro sistema y la creación e implementación de la aplicación que la sustenta.

2 SISTEMAS DE GESTIÓN DE INVENTARIO DE PIEZAS DE REPUESTO

En este primer punto, procederemos a explicar el concepto sobre el que se basa nuestro proyecto, y el porqué de la necesidad de su desarrollo y creación. Un buen sistema de gestión es básico para el devenir de cualquier entidad, por lo que, en primer lugar, deberemos conocer a fondo sus funciones.

2.1 Finalidad y Características del Sistema de Gestión

Toda empresa que se dedique a la producción o fabricación de algún tipo de producto, o incluso a la prestación de algún tipo de servicio, tienen un mismo objetivo común, satisfacer la demanda. Estas compañías, tratarán, basándose en una serie de criterios que ahora expondremos, de cumplir con las exigencias y los pedidos de sus clientes. Una buena gestión de estos proporcionará a la empresa resultados financieros positivos, además de posicionarse a través de una buena reputación, en un mercado cada vez más competitivo.

Antes de seguir profundizando en las particularidades de estos tipos de sistemas, debemos hablar del ámbito en los que se pueden aplicar. Estos, pueden ser usados con en el propósito indicado al principio de este apartado, es decir, para poder hacer frente de manera efectiva a la demanda exigida por la clientela, pero, también los podemos aplicar a un ámbito interno. Un ejemplo de ello sería en un sistema de gestión de mantenimiento impuesto dentro de la propia empresa, donde tendríamos unas necesidades de piezas de repuesto marcadas por la durabilidad de estas.

Desde el punto de vista de una entidad que se encarga de la venta y distribución de estas piezas, para la consecución de lo mencionado en el primer párrafo, necesitaremos centrarnos en una serie de normas:

- Maximizar satisfacción de los clientes
- Minimizar costes de producción

Mantener un buen grado de satisfacción en nuestros clientes es fundamental para el mantenimiento y crecimiento de nuestra empresa, por ello debemos de intentar maximizar todo lo que podamos esta cuestión, pero ¿hasta qué punto?

Existen diversas formas de conseguir ese nivel de agrado de nuestra clientela, pero en la mayoría de los casos conlleva un aumento de los costes. Un buen sistema de gestión del inventario debe tratar de minimizar estos, pero evidentemente, teniendo en cuenta el nivel de servicio que estamos proporcionando. En definitiva, debemos buscar un punto de equilibrio, que en la vida real es muy difícil de encontrar.

Para llegar a conseguir un ritmo de producción eficiente, necesitaremos estudiar y controlar una

serie de variables, donde algunas son difíciles de predecir. A continuación, analizaremos brevemente los puntos más significativos dentro de un sistema de gestión de inventario.

Empezaremos detallando los principales costes existentes dentro de una fábrica o almacén [3]:

- Costes de lanzamiento de pedidos: Tal y como su propio nombre indica, es el gasto asociado a realizar un encargo a nuestro proveedor. Está compuesto por tareas como, ajuste del set-up[1] de la maquinaria, costes administrativos relativos a la preparación de la orden, inspecciones… Mencionar también que, según los acuerdos con el proveedor, estos costes se pueden ver incrementados según el tamaño del lote, aunque en otras ocasiones, estos son independientes.

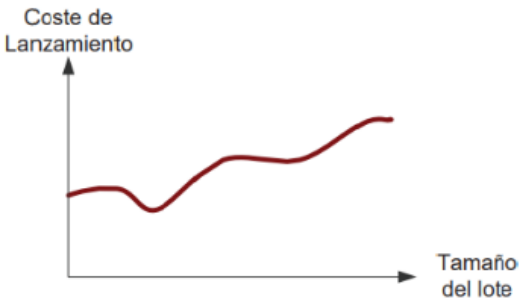


Figura 1: Coste de lanzamiento dependiente

Figura 2: Coste de lanzamiento independiente

En el caso más sencillo, es decir, conociendo la demanda en cada período, el coste de lanzamiento se calcularía de la siguiente forma**:**

$$C.Lanz = A \cdot \frac{D}{Q}$$

- Costes de mantenimiento del inventario: Tener un artículo guardado en el almacén durante un periodo de tiempo considerable, lo único que va a provocar es un aumento de los gastos de la empresa. Esto es precisamente lo que ocurrirá, si realizamos pocos pedidos, pero de lotes muy grandes, que tendremos que almacenar un gran número de productos, y eso no siempre es bueno. Al igual que se indicó en el punto anterior, bajo la misma suposición, el coste de mantenimiento viene regido por la siguiente expresión:

$$C.Mtto = \frac{Q}{2} \cdot v \cdot r$$

- Costes de ruptura de stock: Este tipo de coste es el que pretendemos evitar a toda costa. Podemos encontrarnos con 2 situaciones, en primer lugar, una de rotura, es decir, que no tengamos inventario disponible para surtir el pedido y perdamos la demanda. Por otra parte, puede ocurrir una situación de carencia, donde nos referiremos a los retrasos en los pedidos. Estos costes provocan, además de muchas otras situaciones, un considerable deterioro del prestigio de la empresa, que, por consiguiente, ensuciará el nivel de servicio.

[1] Preparación y ajustes necesarios asociados en este caso, a maquinaria.

En las fórmulas descritas para obtener cada uno de los costes, podemos observar una serie de constantes que pasaremos a definir a continuación.

- *A*, es simplemente el coste que conlleva hacer un pedido. Se mide en unidades monetarias, que en nuestro caso serán *euros*.
- *r*, es la tasa aplicada a mantener el inventario y se mide en *euros/euros*período*.
- *v*, es el coste unitario del producto y como es lógico de mide en *euros/unidad.*

Antes de hablar sobre la demanda y el problema que ocasiona, el cual es el principal motivo por el que hemos desarrollado este proyecto, definiremos las variables que queremos optimizar y determinar. Son dos:

Q, es el lote de aprovisionamiento. Consiste en la cantidad de producto que queremos recibir por parte de nuestro proveedor en un pedido. Es uno de los puntos más complicados de gestionar, ya que hay que pedir lo suficiente como para cubrir la demanda, pero no nos podremos pasar, puesto que lo que conseguiremos es aumentar los costes. Se profundizará en esto más adelante.

s, el punto de pedido. En nuestra fábrica o almacén, contamos con un cierto inventario, el cual va aumentando y disminuyendo continuamente según reposiciones y pedidos. El punto de pedido, nos indica el momento en el que debemos de realizar un nuevo encargo a nuestro proveedor de forma que no haya problemas para cumplir con nuestra demanda. Con un ejemplo se explicará mucho mejor. Si nosotros vendemos llantas, y hemos decidido establecer el punto de pedido en 2, significará que cuando a medida que vayamos atendiendo pedidos, y lleguemos a una situación en la que nos encontremos con 2 unidades en el inventario, será momento de realizar el pedido a nuestro proveedor.

2.2 Problemática en los Sistemas de Gestión

Llegados a este punto es conveniente explicar cuál es el principal problema de estos sistemas y cuáles son sus causantes.

Existen dos variables que van a marcar el "tempo" en nuestro sistema. Estas son, la demanda y el plazo de aprovisionamiento (*PA*). En todas las fábricas del mundo, desearían que ambas variables fueran completamente deterministas, es decir, que se pudiese conocer con exactitud cada cuanto tiempo nos van a pedir, la cantidad del lote y cuánto va a tardar nuestro proveedor en entregarnos la materia prima. Como hemos comentado, esto sería una situación ideal, y por lo tanto irreal.

Muchas empresas tratan de usar valores medios para intentar predecir la demanda, pero como es lógico, lo único que consiguen al emplear esta metodología es la generación de problemas. Centrándonos en nuestro particular caso de estudio, es decir, las piezas de repuesto, tendremos unas particularidades que comentaremos ahora y luego en el último punto del capítulo, profundizaremos.

Estos sistemas de gestión de inventario de piezas de repuesto tienen que enfrentarse a una demanda de la que podemos deducir algunas características. El intervalo de tiempo que transcurre entre que se pide una pieza de repuesto hasta que se vuelve a demandar otra, va a ser

generalmente más grande que el plazo de aprovisionamiento, siendo esta una peculiaridad fundamental para entender el tratamiento de este tipo de sistemas.

Es lógico pensar que la necesidad de este tipo de piezas no va a ser el mismo que el de cualquier otro producto, por lo que el tiempo entre pedidos será generalmente grande. Es importante destacar que esto no siempre será así, existiendo periodos de tiempo en donde puede que se pidan más de una pieza consecutivamente, y es ahí donde deberemos tener cuidado.

Además, no solo nos toparemos con esa particularidad, sino que también nos encontraremos con otra al enfocarnos sobre la cantidad requerida en cada pedido. Cuando se realiza una orden de piezas de repuesto, en la mayoría de los casos, se podrá comprobar que esta es de una unidad, o como mucho dos. Esto es así debido a que cuando un cliente necesita una pieza de este tipo, es porque ha tenido alguna avería y se le ha roto una pieza o ve que está a punto de ello. Es muy extraño que alguien demande muchas piezas de repuesto a la vez, ya que eso significaría que se le han estropeado varios elementos a la vez, y eso muy raramente ocurre.

Existen muchas empresas que invierten cada año grandes sumas de dinero en estudios y análisis para intentar comprender de mejor manera el comportamiento de la demanda de las piezas de repuesto. Según se indica en más de un artículo científico citado en algún momento en la memoria, se están realizando numerosas investigaciones relacionadas con la degradación de este tipo de piezas, ayudándoles de alguna manera a intentar predecir cada cuanto tiempo se requerirá de estas. Por ejemplo, el aumento de la temperatura incrementa la tasa de corrosión de los metales [4], debido principalmente a que estas consiguen acelerar la cinética de las reacciones, por lo tanto, sabremos que la demanda de piezas de repuesto metálicas será mayor en el periodo estival que en cualquier otro.

Si tanto los datos de demanda como los tiempos de aprovisionamiento se rigiesen por una distribución normal de media μ y de desviación típica σ, podríamos obtener una muy buena aproximación de lo que ocurrirá en un futuro [5], pero ¿Y si los datos no se correspondiesen con este tipo de distribución?

En el caso de que esto no fuera así y sea otra distribución la que se ajustase a nuestros datos, se optaría por hacer uso de una simulación de eventos discretos [6], que, de hecho, es la metodología que usaremos para llevar a cabo el proyecto.

Los sistemas con este tipo de eventos se caracterizan por unas variables que van a ir definiendo el estado de nuestro sistema en cada momento, estas, irán cambiando en un conjunto discreto de instantes de tiempo.

En nuestro sistema de gestión, estas variables a las que hacíamos referencia en el párrafo anterior son el tiempo entre demandas y el plazo de aprovisionamiento. Iremos avanzando en el tiempo simulando ambas variables según unos parámetros que veremos más adelante.

Antes de continuar, debemos introducir un concepto clave en los sistemas de gestión de inventario.

Tanto la demanda, como el plazo de aprovisionamiento son variables aleatorias, y por lo tanto nuestro inventario no se va a vaciar de forma uniforme. En esta situación de incertidumbre debemos de establecer una franja de seguridad para reducir los problemas de rotura de nuestro inventario, utilizaremos lo que se conoce como stock de seguridad (*ss*) [7].

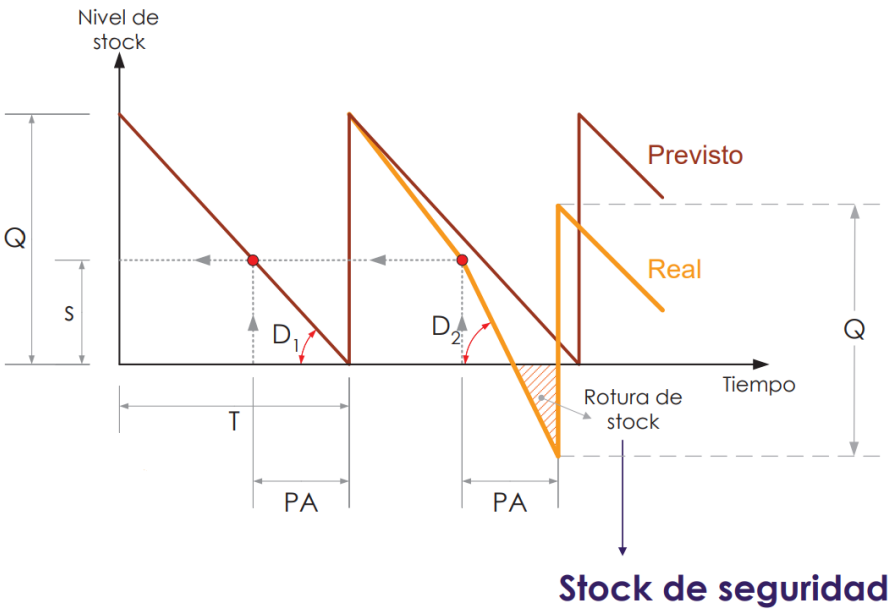


Figura 3: Explicación necesidad de stock de seguridad

En definitiva, el stock de seguridad no es más que añadir una concreta cantidad de producto a nuestro inventario para hacer frente a las desconocidas fluctuaciones de demanda y *PA*.

El stock de seguridad, como se verá más adelante, se deberá de establecer previamente, por lo que habrá que tener muy en cuenta la variabilidad de los tiempos que estamos analizando, porque de ello dependerá si se pone un mayor o menor nivel de seguridad.

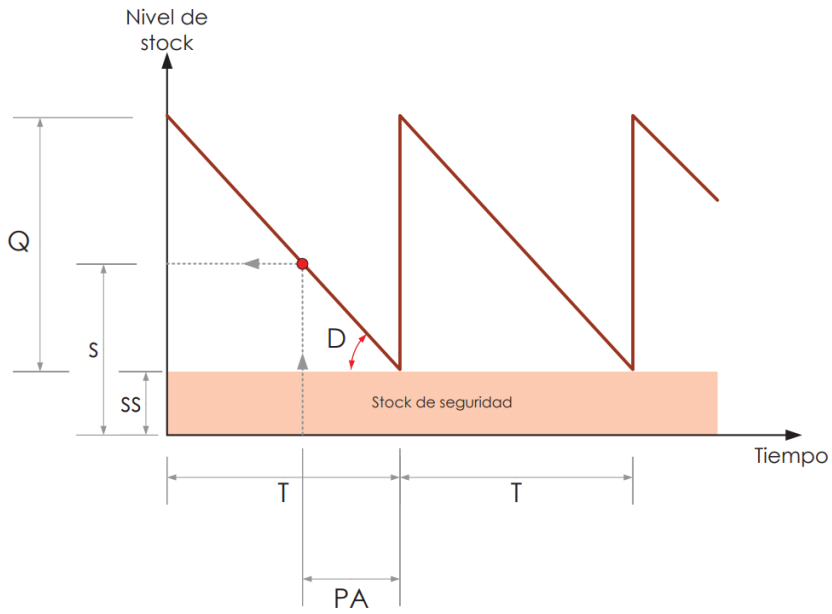


Figura 4: Añadido stock de seguridad

2.3 Tipo de Sistemas de Gestión de Inventario

Antes de analizar en profundidad el caso concreto sobre el cual vamos a basar el proyecto, expondremos los diferentes tipos de gestión de inventario con los que nos podemos encontrar [8].

DEMANDA DETERMINISTA Y ESTÁTICA

En este primer caso, tendremos una tasa de demanda conocida y constante. Además, nos podemos encontrar con diferentes sistemas con pequeñas variaciones entre sí, entre los que destacan:

- Sistema simple de revisión continua (*s, Q*)

  En este sistema, el aprovisionamiento será por lotes, es decir, si por ejemplo se realiza un pedido a nuestro proveedor de X unidades, cuando estas nos lleguen al almacén, lo harán todas al mismo tiempo. También cabe destacar que, si usamos este procedimiento, significará que el plazo de aprovisionamiento será nulo.

  La inexistencia de un tiempo de aprovisionamiento significará que tendremos un punto de pedido establecido en 0. En este caso cuando nos quedemos sin inventario en el almacén, pediremos y recibiremos instantáneamente los productos, por lo que no será necesario establecer ningún punto de pedido.

- Sistema de revisión continua (*s, Q*) con Tasa de Entrega

  Aquí nos encontramos ya con una pequeña diferencia, los artículos pedidos al proveedor se recibirán con una tasa constante *P*. En el punto anterior vimos que los productos ordenados, se recibían en un lote conjunto todos al mismo tiempo, ahora, estos artículos los iremos recibiendo poco a poco según el valor de la tasa de entrega.

  El plazo de aprovisionamiento sigue siendo nulo, por lo que el punto de pedido lo estableceremos también en 0.

- Sistema de revisión continua (*s, Q*) con Plazo de Aprovisionamiento

  En este caso volveremos a tener un aprovisionamiento por lotes, tal y como ocurría con el primer sistema. Sin embargo, ahora si se considerará que existe un determinado plazo de aprovisionamiento, es decir, que cuando realicemos un pedido este nos tardará en llegar un tiempo, y no instantáneamente como ocurría en los dos casos anteriores.

  En este tipo de sistemas, tendremos la necesidad de establecer un punto de pedido y lo haremos en función del valor del tiempo de aprovisionamiento.

- Sistema de revisión continua (*s, Q*) con Tasa de Entrega y Plazo de Aprovisionamiento

  Llegamos al último sistema de revisión continua, este, junto con el anterior, son los que más se asemejan a la realidad.

  En este caso existirá una tasa de entrega, es decir, no será por lotes, y, además existirá un determinado plazo de aprovisionamiento que regirá el tiempo que tendremos que esperar para que nos empiecen a llegar los artículos pedidos.

- Sistema simple de revisión periódica (*R, S*)

  Como su propio nombre indica, en este tipo de sistemas lo que haremos será establecer un período entre revisiones del stock, *R*. Examinaremos nuestro inventario en los intervalos de tiempos fijados, para realizar un pedido con la cantidad apropiada, es decir, que el tamaño de este variará según la demanda, que en nuestro caso tiene un valor constante.

  En este tipo de sistemas se adopta un aprovisionamiento por lotes y estos se recibirán instantáneamente, es decir, tendremos un *PA* nulo.

- Sistema simple de revisión periódica (*R, S*) con Plazo de Aprovisionamiento

  En último lugar, veremos otro sistema con revisión periódica, que a diferencia del visto anteriormente, si cuenta con un plazo de aprovisionamiento que deberemos de tener en cuenta.

DEMANDA PROBABILÍSTICA Y ESTÁTICA

En el caso anterior, todos los sistemas se regían por una demanda determinista, es decir, sabíamos cuánto nos iban a pedir. También teníamos una tasa de demanda estática, por lo que sabíamos cuando iban a ser esos pedidos.

Ahora, como podemos observar en el título, tendremos una demanda aleatoria o probabilística y una tasa de demanda igual que en el punto anterior, es decir, estática.

Si consideramos situaciones deterministas, que son prácticamente ideales, podremos calcular el nivel de inventario que debemos de tener con una absoluta precisión. En cambio, si la demanda es aleatoria, el stock ya no se vaciará de forma uniforme, por ello, para prevenir problemas con el inventario y evitar que se produzcan las roturas, se introducirá el ya conocido stock de seguridad.

Nos podemos encontrar con los siguientes tipos de sistemas:

- Sistema de revisión continua (*s, Q*) con Stock de Seguridad
- Sistema de revisión periódica (*R, S*) con Stock de Seguridad

Ambos son muy parecidos, diferenciándose únicamente en la forma en la que se revisan los niveles del inventario.

DEMANDA DETERMINISTA Y VARIABLE

Ahora tocará ver un tipo de problema, donde la demanda será determinista, pero la tasa de demanda no será estática, sino que será variable. Teóricamente podemos usar algunos métodos para la gestión del inventario, entre los que destacamos:

- Equilibrado de Costes
- Método de Mínimos Costes Medios

DEMANDA COMPLETAMENTE ESTOCÁSTICA

Este es el tipo de sistemas más común en la realidad. Las compañías que se dediquen a la venta de algún tipo de producto, en prácticamente la totalidad de las ocasiones, desconocen cuando le van a realizar un pedido y cuál es la cantidad asociada a este.

Por ello se realizan grandes inversiones en estudios estadísticos y probabilísticos para intentar de la mejor manera posible predecir estos factores.

2.4 Particularidades del Sistema Objeto de Estudio

Una vez analizada la problemática y los diversos sistemas de gestión del inventario con los que nos podemos encontrar, nos volveremos a centrar en nuestro caso de estudio, es decir, en las piezas de repuesto.

Como ya comentamos al principio del capítulo, la principal diferencia entre un sistema que gestione piezas de recambio y otro que lo haga con productos completos y más comunes, va a ser la cantidad de unidades que se nos requerirá en cada pedido. Por esto la demanda en nuestro sistema la tomaremos como la unidad, siendo entonces más interesante modelar el tiempo entre demandas que la demanda en sí. El plazo de aprovisionamiento será simplemente el tiempo que tarde nuestro proveedor en entregarnos el pedido realizado.

En la *figura 5*, podemos observar que es lo que ocurriría con la demanda de las piezas de repuesto a medida que se avanza en el tiempo. Como ya se ha comentado, debido a la naturaleza intrínseca de la demanda de este tipo de piezas, en raras ocasiones esta superará la unidad. Por esta razón usar en nuestro sistema el tiempo entre demandas es algo más lógico y, sobre todo, mucho mejor para la simulación y optimización de nuestro problema.

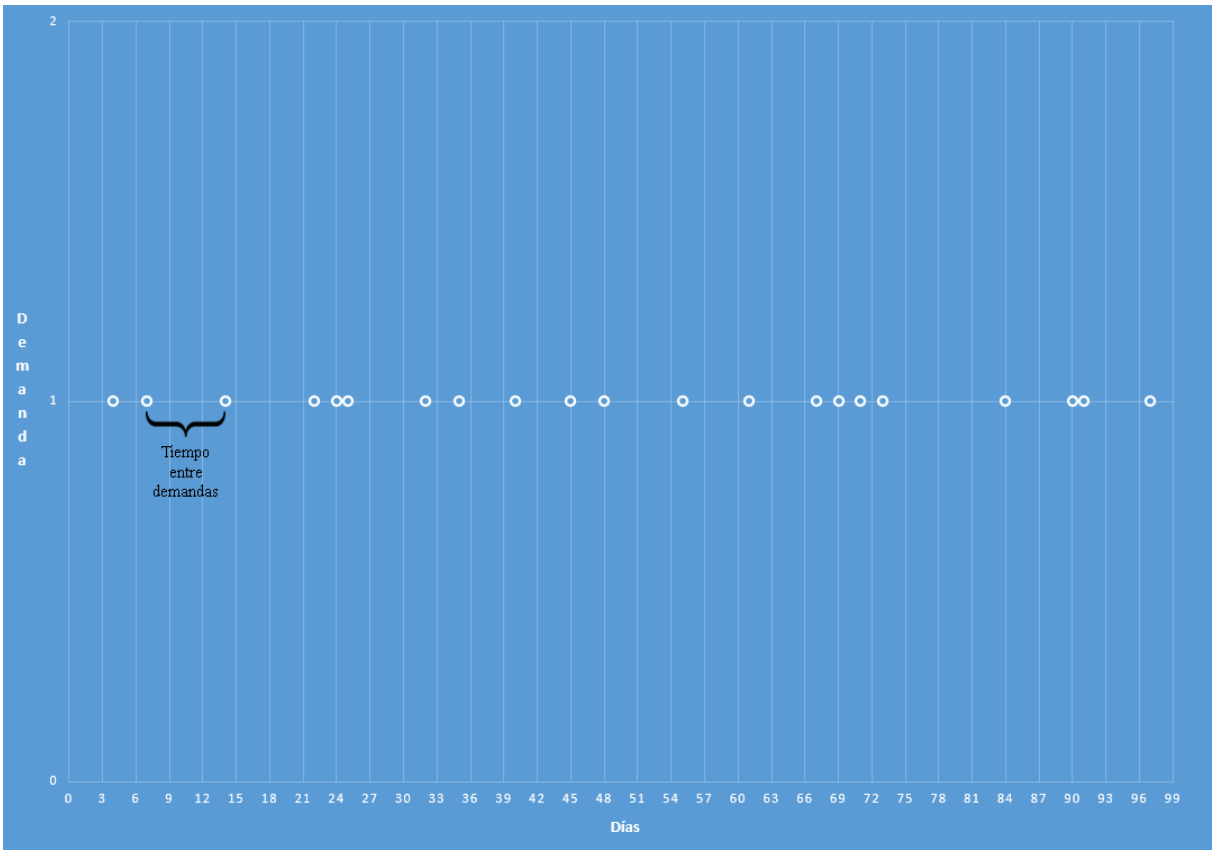


Figura 5: Variable a modelar en nuestro problema

Otro punto importante es la relación existente entre esos tiempos entre pedidos y el plazo de aprovisionamiento. En la mayoría de los productos del mercado la frecuencia con la cual se requerirán de estos provocará una situación en la que deberemos de pedir grandes cantidades a nuestro proveedor, ya que los tiempos de aprovisionamiento serán considerablemente superiores a los tiempos que transcurrirán entre un pedido y otro. En nuestro caso de estudio, es decir, las piezas de repuesto, nos encontraremos con una situación algo diferente a la que se acaba de detallar. La frecuencia con la que se requerirán estas piezas no será la misma que la que se puede llegar a ver en otras más comunes, debido principalmente a que la necesidad de una pieza para una sustitución suele ser algo extraordinario.

Debido a todo lo comentado anteriormente, en nuestro caso particular, los plazos de aprovisionamiento serán generalmente inferiores a los intervalos existentes entre una demanda y otra, perdiendo estos una importancia considerable dentro de nuestro sistema. La variabilidad con la que cuente el *PA* en nuestro problema será algo prácticamente irrelevante, por lo que no afectará demasiado al sistema a la hora de realizar la optimización. Esta, es otra de las características peculiares que podremos resaltar en nuestro proyecto.

También recordar, que estos tipos de sistemas son aplicables dentro de las propias compañías. Las que usen un gran número de maquinaria o cualquier otro tipo de herramienta que requiera de

alguna sustitución en algún momento, necesitan de un buen modelo para perder el menor tiempo posible en esas reparaciones o cambios. Como ya se ha comentado en más de una ocasión, las investigaciones sobre este tipo de sistemas no paran de aumentar.

En un estudio realizado por varias universidades en Francia [9] se propuso un nuevo modelo para la optimización conjunta del mantenimiento preventivo periódico y el problema de inventario de piezas de repuesto, obteniéndose resultados buenos y, sobre todo, eficientes.

Otro claro ejemplo lo encontramos en el artículo publicado por Willem van Jaarsveld, Twan Dollevoet y Rommert Dekker [10] en el cual se investiga sobre sistemas de control del inventario de piezas de repuesto para un taller de reparación de componentes de aeronaves. Para hacernos una idea de lo importantes que son estos sistemas, en el artículo se detalla que en los últimos años se han venido facturando en las reparaciones de estos componentes, alrededor de 9 mil millones de dólares. También se comenta que existe una evidente demora en los tiempos de reparación y recambios de estos componentes, siendo el principal causante de todo esto los deficientes métodos de optimización del inventario de piezas de repuesto.

Si comparamos nuestro problema a resolver con los diferentes tipos de sistemas vistos previamente, nos damos cuenta de que nosotros tendremos una demanda determinista, puesto que sabemos que en cada orden se nos requerirá una pieza. En cambio, tendremos una tasa de demanda variable, puesto que no sabremos cada cuanto tiempo nos van a pedir en el futuro.

Hay que mencionar que debido a las evidentes carencias con las que cuentan la mayoría de los métodos usados hoy en día en muchas empresas para el diseño de los sistemas que gestionen un inventario compuesto por piezas de repuesto, tal y como se aludió anteriormente, haremos uso de una simulación de eventos discretos. A medida que avancemos en esta memoria, iremos entendiendo mejor en que consiste esta metodología y como se ha aplicado a nuestro problema.

En toda empresa, uno de los mayores problemas existentes son los despilfarros, siendo la sobreproducción[2] el más importante de todos ellos. El factor causante de este inconveniente es el de una mala planificación y gestión, generada en la mayoría de los casos por una mala estimación de la demanda, que provoca que en algunas ocasiones se fabrique más de lo necesario, generando costes evitables tanto de lanzamiento como, sobre todo, de mantenimiento. Cabe resaltar que también se puede dar la situación contraria, es decir, que no tengamos suficiente inventario como para cubrir la demanda requerida y se produzcan las temidas roturas de stock.

Debido a todo lo visto en este capítulo, la creación de un buen modelo especializado en las piezas de repuestos es básico para toda compañía en la que se traten con ellas, ya que como se ha visto tienen unas particularidades que no se pueden obviar.

En el siguiente apartado, se analizará y explicará la forma en la que se ha abordado el problema para tratar de minimizar los problemas existentes en la actualidad.

[2] Exceso de producción de una cosa por encima de las necesidades dictadas por el mercado

3 SOLUCIÓN PROPUESTA

En este punto analizaremos cómo pensamos abordar este problema y definiremos una propuesta de metodología. Además, tal y como se adelantó en la parte anterior, en nuestro sistema vamos a tratar con piezas de repuestos, y esto conlleva unas consideraciones que debemos tener en cuenta a la hora de encontrar una solución viable.

3.1 Introducción

Una vez hemos detallado la problemática existente en estos tipos de problemas tendremos que analizar las posibles alternativas con las que contamos para resolverlos, viendo si estas son viables o no.

3.1.1 Resolución Analítica

En primer lugar, estudiaremos si es posible obtener una buena solución a nuestro problema mediante el uso de métodos analíticos. Si recordamos lo visto en el capítulo anterior, uno de los principales problemas que existían en muchos de los sistemas implantados hoy en día, era la mala predicción y, por tanto, los imprecisos resultados que se obtenían.

Debido a la naturaleza intrínseca con la que cuentan las piezas de repuesto, la demanda de estas tiene un comportamiento altamente estocástico. En la mayoría de propuestas basadas en una resolución analítica, se hace uso de aproximaciones y estimaciones muy irregulares tanto de la demanda como de los plazos de aprovisionamientos.

Con este tipo de resolución es imposible tener en consideración todos los detalles que forman parte de un sistema de gestión de inventario de piezas de repuesto, por todo esto, una de las pocas maneras de optimizarlo sería haciendo uso de valores medios y, suponiendo una gran cantidad de detalles que hacen de este método una básica y simple aproximación de lo que nos podemos llegar a encontrar.

El libro *"Simulation modeling and analysis"* [6] estudia el uso de la metodología analítica para la resolución de problemas. En él, se comenta que existen soluciones obtenidas haciendo uso de este método que son realmente simples, sin embargo, algunas son muy complejas y requieren de una gran capacidad de computación para obtenerlas.

Si la solución analítica propuesta para un modelo matemático es computacionalmente eficiente, lo lógico es estudiar el modelo usando esta metodología en vez de usar una simulación. Sin embargo, el sistema que queremos diseñar en este proyecto es bastante complejo, excluyendo cualquier posibilidad de obtener una solución analíticamente.

En un artículo en el que se analiza la previsión de la demanda de piezas de repuesto [11], se habla de un trabajo fundamental que escribió J.D.Croston sobre el pronóstico de la demanda intermitente. En él, en lugar de centrarse en la media en cada período, decidió dividir el estudio en dos partes, analizando el intervalo entre demandas y el tamaño de esta, por separado. El método

de Croston contaba con un problema que lo hacía inviable, este, era que en períodos donde no había demandas la previsión de esta no se reducía.

Debido a esto, múltiples autores han investigado y realizado ajustes sobre el método creado por Croston en 1972, consiguiendo notables mejorías. Aun así, estos métodos tal y como se explica en el artículo, no se pueden aplicar en un entorno práctico, puesto que en él se tratan miles de piezas de repuesto, algo que queda totalmente fuera de alcance.

En definitiva, para el tratamiento de unos datos tan complejos hace falta algo más de precisión y detalle que desafortunadamente, haciendo uso de esta metodología, nunca seremos capaces de lograr.

3.1.2 Resolución por Simulación

Debido a la imposibilidad de obtener una buena optimización del sistema haciendo uso de métodos analíticos, se deberá buscar otra manera a través de la cual se pueda conseguir el nivel buscado en este trabajo.

Después de una profunda investigación sobre posibles alternativas para intentar resolver nuestro problema, se llegó a la conclusión de que la mejor manera para ello era el uso de una simulación basada en eventos discretos. Lo que conseguiremos gracias a esta, será obtener un sistema donde se reproducirán los comportamientos de uno real, siguiendo un patrón basado en una serie de eventos e interacciones.

Como podremos deducir, los eventos mencionados previamente marcarán el *"tempo"* dentro de nuestra simulación. Estos, se corresponden con los tiempos entre demandas existentes y los plazos de aprovisionamiento con los que nos encontraremos cada vez que se realice un pedido.

Llegados a este punto nos podremos preguntar cómo vamos a ir obteniendo esos valores para ir avanzando en el tiempo dentro de la simulación. Para intentar predecir el comportamiento que tendrá la demanda de las piezas de repuesto en un futuro, no nos valdría con tomar valores medios u alguna aproximación básica, debido a la alta estocasticidad con la que estas cuentan.

Para llevar a cabo la simulación, haremos uso de los tiempos históricos de las 2 variables que rigen los eventos en nuestro sistema, es decir, tomaremos toda la información posible relativa al tiempo entre demandas y el *PA* que se han ido dando en momentos anteriores al actual. Analizaremos estos datos, realizando un complejo contraste de hipótesis para ver cuál es la distribución a la que mejor se ajustan los tiempos. Con esta información en nuestra mano podremos establecer una serie de parámetros dentro de nuestro sistema, que variarán en función de la distribución que se haya obtenido, además de poder completar la propia simulación.

Se ha elegido esta opción ya que, gracias a este tipo de simulación, seremos capaces de ir analizando paso por paso todo lo que ocurre alrededor y, sobre todo, dentro de nuestro sistema de gestión.

Además, para conseguir una buena optimización haciendo uso de esta metodología, necesitaremos de una serie de ajustes previos a la simulación.

3.2 Diagrama de Flujo del Proceso

Para facilitar la comprensión del proceso de resolución de nuestro problema, se ha desarrollado un diagrama de flujo del proceso. De esta manera se podrá ir comprobando detalladamente que es lo que se ha realizado en cada uno de los puntos de la optimización. En los siguientes apartados, se entrará en profundidad en cada uno de esos puntos.

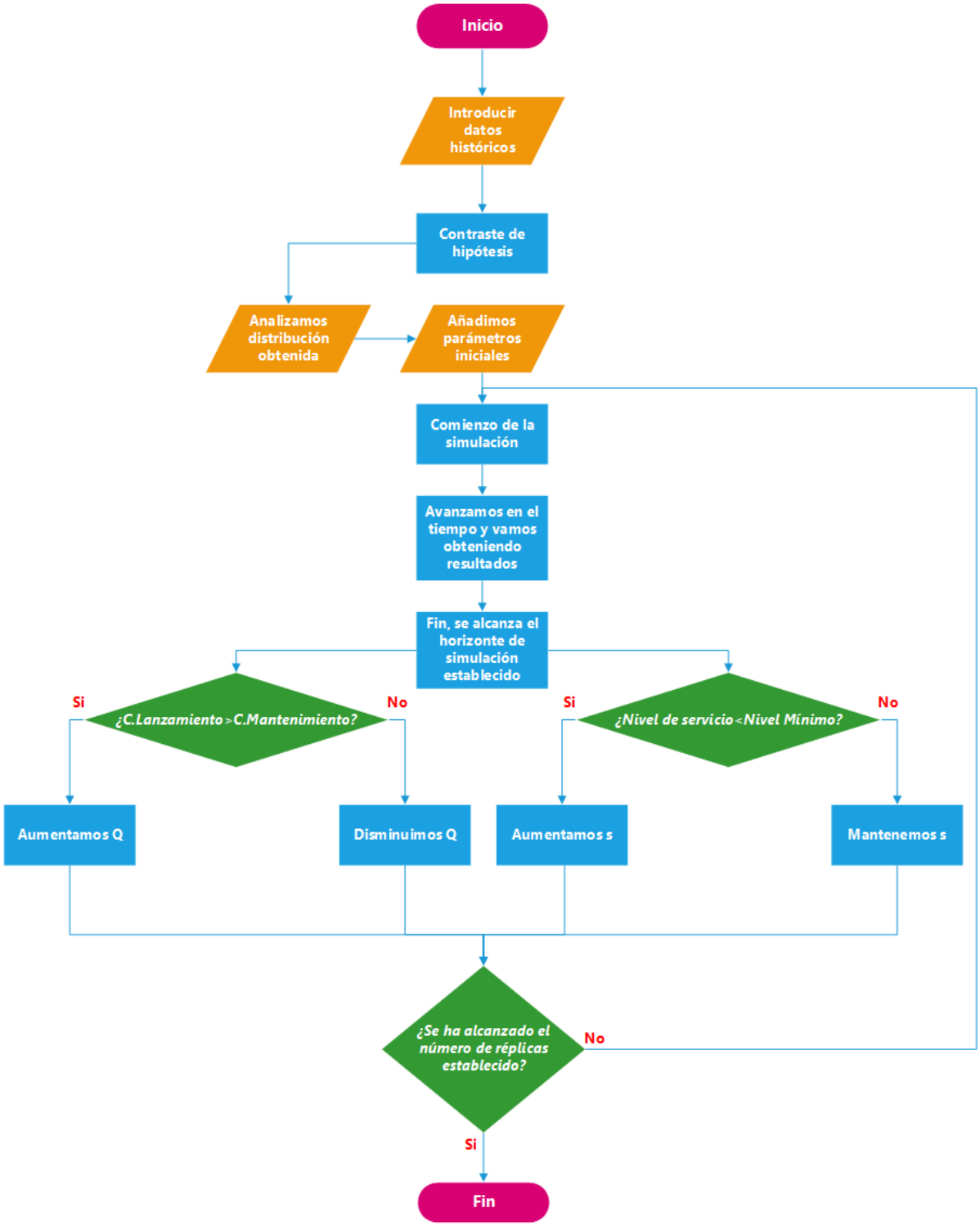


Figura 6: Diagrama de flujo del proceso

3.3 Proceso de Resolución

En los siguientes apartados iremos analizando cada uno de los puntos que componen la optimización de nuestro sistema, detallando en todos ellos el porqué de cada decisión tomada a lo largo del proceso.

3.3.1 Tiempos Históricos

Antes de entrar en detalle sobre cómo se ha realizado la simulación del problema, vamos a comentar en que nos hemos basado para obtener los datos tanto de la demanda como de los plazos de aprovisionamiento.

En nuestro sistema de gestión pensamos que debido a la estocasticidad de estos tipos de eventos, lo mejor y más preciso sería tomar los datos históricos de ambos. Al fin y al cabo, lo que queremos es tener en nuestra mano la mejor información sobre los tiempos de demanda y el *PA*, y que mejor que lo ocurrido en el pasado para predecir el futuro.

A continuación, explicaremos la forma en la que se ha conseguido pasar de una simple recopilación de los datos históricos relativos a las piezas de repuesto, a información crucial para completar la optimización.

Lo primero que haremos será recabar una lista con los tiempos entre pedidos y otra, con los plazos de aprovisionamiento asociados a cada una de las ordenes lanzadas a nuestro proveedor. A la hora de realizar la simulación, el usuario tendrá la opción de introducir estos tiempos en ella gracias a la aplicación desarrollada.

Antes de comenzar con la simulación, analizaremos los datos introducidos. Para ello, lo que haremos será identificar la distribución estadística que mejor se ajusta a nuestros valores de tiempos. Esto lo conseguiremos gracias a un contraste de hipótesis, estudiando la complicidad de nuestros datos con las siguientes distribuciones:

- Normal
- Exponencial
- Gamma
- Logística
- Log-normal
- Triangular
- Uniforme

Una vez completado el proceso detallado en el párrafo anterior, el usuario podrá comprobar previamente, cual de esas distribuciones es la que mejor se ajusta con los tiempos que ha introducido. A la hora de establecer el stock de seguridad, se deberá tener en cuenta la variabilidad de los datos, por lo que conocer con antelación esta información será algo fundamental.

3.3.2 Parámetros Iniciales

Una vez evaluados los tiempos y obtenida la distribución que mejor se ajusta a estos, podemos comenzar a introducir todos los parámetros necesarios para que se pueda llevar a cabo la optimización. Haremos una distinción entre los dos tipos de información que necesita nuestro sistema.

3.3.2.1 Parámetros de la Simulación

Para poder empezar con la simulación, el usuario deberá indicar previamente los valores necesarios para que esta se pueda llevar a cabo, son:

- Número de réplicas: Servirá para indicar al programa las veces que se repetirá el proceso de simulación.
- Horizonte de simulación: Cada una de las simulaciones durará el tiempo que indique el usuario previamente al programa.

Para establecer dichos parámetros aconsejamos que se usen los procedimientos y recomendaciones descritas acerca de la simulación en el libro *"Simulation modeling and análisis"* [6].

3.3.2.2 Parámetros Iniciales de la Optimización

A continuación, definiremos cuales son las variables y constantes que se deberán introducir previamente en nuestro programa para poder optimizar los parámetros del sistema.

1. Tipo de distribución: El usuario deberá introducir los datos históricos en la aplicación, para que esta posteriormente realice el contraste de hipótesis, identificando que tipo de distribución es la que mejor se ajusta a nuestros datos, para poder predecir los valores futuros.
2. Inventario inicial: Le indicaremos el número de piezas de repuesto que tendremos inicialmente a nuestra disposición en el almacén o fábrica.
3. Le pasaremos también los valores iniciales de nuestras variables a optimizar, es decir, la cantidad Q, y el punto de pedido, s, establecidos en un principio.
4. Otro dato muy importante que se deberá indicar es el del stock de seguridad (ss), este, no variará dentro de la optimización. El usuario establecerá el ss que considere oportuno, para que una vez que el proceso de optimización concluya, ver en función de los resultados obtenidos, si ha de aumentarlo o disminuirlo.
5. Otro dato importante que es necesario indicar es el nivel de servicio mínimo que debemos mantener en la empresa, por debajo del cual no querremos estar nunca.
6. Por último, tendremos que añadir 3 constantes fundamentales para el cálculo y control de los costes en nuestra organización. Estos son, el coste de lanzamiento (A), el precio unitario de cada producto (v), y la tasa aplicada a mantener el inventario (r).

3.3.3 Valores de salida

Una vez completada la simulación y optimización de nuestro sistema, obtendremos el valor de Q y el del punto de pedido que han sido calculados teniendo en cuenta la limitación del nivel de

servicio mínimo impuesto por el usuario previamente al inicio de todo el proceso.

En los próximos apartados veremos en profundidad como se han determinado el valor de cada uno de estos parámetros.

3.3.4 Comienzo de la Simulación

Una vez detallados todos los elementos necesarios para la simulación y optimización de nuestro problema, procederemos a analizar cómo se ha desarrollado.

Para llevar a cabo este proceso nos basaremos en la simulación de eventos discretos. En definitiva, lo que haremos será simular la petición de piezas de repuesto y la llegada por parte del proveedor, ajustando a su vez los parámetros del modelo de manera similar a la simulación de Montecarlo. Trataremos, a partir de los datos iniciales, de generar un escenario lo más parecido a la actualidad posible, consiguiendo así la predicción del comportamiento de algunas variables las cuales queremos optimizar.

Debido a que la demanda de piezas de repuesto suele tener una frecuencia menor que la que puede tener cualquier otro tipo de producto, nos encontraremos con un entorno donde el tiempo entre pedidos va a ser muy grande si lo comparamos con el plazo de aprovisionamiento. Por lo que el *PA*, pasará a tener bastante menos relevancia en nuestro sistema.

Gracias a una librería disponible en Python (simpy[3]), podremos recorrer todo el espacio de tiempo en el intervalo establecido como la duración de cada simulación (horizonte de simulación), es decir, seremos capaces de ir estudiando cual es la situación de nuestro inventario cada vez que se produzca un cambio en este.

Los "saltos de tiempo" en nuestra simulación, irán marcados por las peticiones de demanda y los tiempos de aprovisionamiento. Estos, serán generados a partir de los datos históricos y de la distribución obtenida gracias al contraste de hipótesis realizado en instantes anteriores.

Dentro de la simulación, cada vez que se reciba una petición de pieza, el sistema restará una unidad a nuestro inventario, tal y como ocurriría en la vida real. Además, cuando el nivel de inventario llegue al establecido en el punto de pedido, se lanzará una orden a nuestro proveedor, donde entrará en juego el *PA*.

A lo largo del proceso que acabamos de describir, se irán guardando en variables toda la información que nos será útil para comprender como ha sido el comportamiento de nuestro sistema en la simulación. Una vez alcanzado el horizonte de esta, tal y como podemos observar en el diagrama de flujo, tocará realizar un reajuste de las variables que estamos pretendiendo optimizar en nuestro sistema.

3.3.5 Cálculo Parámetro Q

El programa disminuirá en una unidad nuestro inventario cada vez que se produzca una petición de pieza. Iremos avanzando en el tiempo, según las demandas entrantes, hasta que lleguemos a un nivel de inventario preestablecido del que ya hemos hablado, es decir, el punto de pedido.

[3] Librería de Python usada para la simulación de eventos discretos

Una vez que nuestro nivel de stock llegue a esa situación, el programa lanzará una orden de aprovisionamiento a nuestro proveedor, cuyo tamaño será inicialmente el indicado al introducir el valor de *Q*, además, como es lógico, nuestros clientes seguirán lanzando órdenes de pedido, y es en esta situación donde debemos tener cuidado.

El tiempo entre pedidos es generalmente mayor que el plazo de aprovisionamiento, al ser piezas de repuesto, la frecuencia con la que estos se requieren no es demasiado alta, por lo que en la mayoría de los casos las piezas llegarán a nuestro almacén antes de que nos demanden más artículos. Pero esto no siempre es así, ya que en algunas ocasiones la llegada de nuestro pedido se puede demorar más de lo previsto y encontrarnos con una situación en la que no tengamos stock.

Por lo mencionado anteriormente, debemos ir ajustando *Q* y *s*, para evitar caer en este problema, pero con cuidado de no pasarnos, puesto que la sobreproducción es uno de los mayores generadores de despilfarros que nos podemos llegar a encontrar.

Para evitar tener unos costes excesivos y perder eficiencia en la empresa, deberemos encontrar el equilibrio entre los diferentes gastos. Con una ilustración, se verá algo mejor:

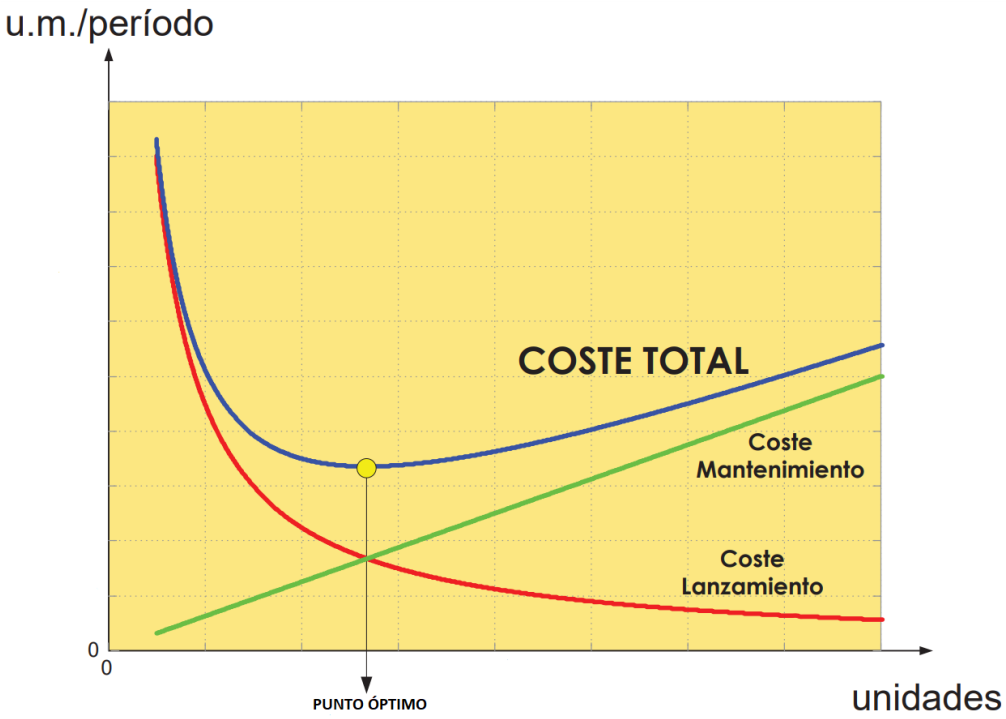


Figura 7: Equilibrado de costes

Como se puede corroborar con el gráfico, podemos tener diferentes situaciones en nuestro sistema.

Por un lado, si la cantidad que pedimos a nuestro proveedor es demasiado grande (valor de *Q* alto), nuestro coste de mantenimiento aumentará considerablemente, puesto que lo que conseguiremos es ampliar el número de unidades en nuestro inventario, aunque por otra parte, al realizar pedidos de mayor magnitud, tendremos que efectuar un menor número de lanzamientos,

puesto que tendremos unidades de sobra para cubrir la demanda, y por lo tanto, el coste de pedir bajará. Observando la gráfica vemos que esta situación no es la ideal.

Por otra parte, también puede ocurrir lo contrario, es decir, establecer una Q no muy alta para evitar despilfarros en el mantenimiento teniendo en todo momento pocas unidades en el inventario, lo que desafortunadamente provocará un aumento del número de lanzamientos, puesto que tendremos que cubrir la demanda que nos vaya llegando. Esto, como podemos observar, tampoco es un escenario en el cual nos gustaría estar.

En conclusión, el programa irá ajustando en cada simulación, un tamaño de pedido (Q), que haga que el coste de mantenimiento y el de lanzamiento sean prácticamente iguales, porque, como podemos observar en el gráfico, la igualdad de ambas variables nos indicará que nuestros costes totales van a ser lo más pequeño posible.

Una vez analizada la forma en la que se obtendrá Q, debemos de ver cómo obtener el punto de pedido.

3.3.6 Cálculo Punto de Pedido

Cada vez que alcancemos el horizonte de nuestra simulación establecido previamente, se analizarán los resultados derivados. Por un lado, comprobaremos los valores obtenidos a lo largo del período, tanto del coste de mantenimiento como del coste de lanzamiento, para poder establecer, como ya hemos comentado, el valor de Q, y, por otro lado, estudiaremos el nivel de servicio obtenido al final del intervalo preestablecido, para decidir qué hacer con el punto de pedido.

Cuando analizamos los costes en el primer apartado de la memoria, definimos las fórmulas que íbamos a usar para el cálculo de cada uno de ellos, sin los cuales sería imposible optimizar la cantidad a pedir a nuestro proveedor. Ahora, veremos cómo hemos obtenido el nivel de servicio al final del período de simulación.

A medida que avanza el tiempo, habrá ocasiones en las que se darán roturas de stock, las cuales anotaremos. Una vez conocido este dato y sabiendo el número total de pedidos que se han realizado a lo largo del horizonte de simulación, podremos calcular el porcentaje de pedidos que no han llegado a los clientes o simplemente se han retrasado. Si lo vemos desde el otro lado, es decir, 100 - % pedidos retrasados, obtendremos el nivel de servicio que hemos ofrecido a nuestra clientela.

Al introducir las variables iniciales en nuestra simulación, nos damos cuenta de que una de ellas es el nivel de servicio mínimo que aceptaríamos tener en la empresa. Esto es importante, puesto que el punto de pedido va a depender del nivel que indiquemos. Cada vez que se complete una simulación, compararemos el nivel de servicio obtenido con el mínimo que aceptaríamos, en caso de que sea inferior, lo que haremos es aumentar nuestro punto de pedido para intentar resolver el problema.

Cuando el programa vaya ajustando el parámetro s, y aumente su valor levemente, porque tal y como hemos explicado en más de una ocasión tendremos un número de roturas de stock relativamente bajo, lo que conseguiremos es pedir con algo más de margen, es decir, en el momento en el que se lance una nueva orden tendremos algunas unidades más en el inventario,

por lo que nuestro nivel de servicio aumentará. Es cierto que también aumentarán los costes, pero si queremos dar un buen servicio a nuestros clientes, habrá que sacrificar otras cosas.

3.4 Comentarios finales

En definitiva, tal y como se puede observar en el diagrama de flujo, una vez finalizada la primera simulación el programa habrá reajustado las variables *Q* y *s* en función de nuestras necesidades y, requisitos indicados a lo largo del programa. Llegados a este punto el programa comprobará si se han alcanzado el número de réplicas establecido al principio de la optimización por el usuario, en caso de que este valor no se haya alcanzado todavía, el sistema deberá realizar de nuevo todo el proceso de simulación, realizando un nuevo ajuste en los parámetros de este.

A lo largo del proceso, también se irán recogiendo algunos datos para mostrar al usuario los resultados de la optimización y facilitar así la comprensión del proceso realizado.

Otro punto que sería importante explicar y analizar, es la función del stock de seguridad dentro del sistema diseñado. El *ss* es la forma en la que se denomina al conjunto de piezas preparadas para hacer uso de ellas cuando nos quedemos sin inventario, es decir, es como un colchón de seguridad que se establece dentro del sistema para evitar la situación de quedarnos sin piezas y no poder hacer frente a los pedidos. En nuestro sistema, el stock de seguridad lo meteremos dentro del inventario existente en este. Cada vez que se haga uso del *ss*, lo anotaremos, para poder analizar al final del proceso si el valor escogido es el correcto o no.

El stock de seguridad tendrá una gran importancia dentro de nuestro sistema, debido a la intermitente demanda de las piezas de repuesto. Conocer la variabilidad que tendrán los tiempos entre demandas será fundamental para establecer el adecuado *ss*. Si los datos históricos introducidos cuentan con un coeficiente de variación alto, se deberá establecer un stock de seguridad elevado para que, ante períodos de mucha demanda, se pueda hacer frente a esta sin problema.

4 DESARROLLO DE LA APLICACIÓN

Ahora hablaremos sobre cómo se ha implementado y, sobre todo, cuáles han sido las herramientas usadas para llevar a cabo todo lo explicado en el apartado anterior. Se detallará cuales son cada una de las partes que conforman la optimización y cómo funcionan cada una de ellas. Los códigos desarrollados en todo nuestro proyecto estarán expuestos en el anexo del final de la memoria, junto con una explicación detallada de los elementos y las funciones usadas en cada uno de ellos.

4.1 Lenguaje Utilizado

Nuestro proyecto se ha basado en un único lenguaje de programación, este ha sido, Python. Sin embargo, para el diseño de la interfaz gráfica de nuestra aplicación hemos tenido que usar un software llamado QT Designer.

4.1.1 Python

Fue creado en el año 1991 por el informático neerlandés Guido van Rossum. Es uno de los lenguajes de programación más usados en el mundo. Destacamos su versatilidad, además de su relativa limpieza y legibilidad con la que cuenta su estructura, también, otro punto que es importante destacar, es que se puede ejecutar en diferentes sistemas operativos, simplemente eligiendo el intérprete correspondiente.

Se ha escogido este lenguaje de programación debido a las infinitas posibilidades de desarrollo que este puede llegar a ofrecer al usuario, además de que este posee una licencia de código abierto, lo que permite su utilización para diferentes objetivos sin tener la necesidad de pagar por su uso.

4.1.2 QT Designer

Para facilitar el uso de nuestro programa, hemos creado una interfaz, es decir, una aplicación en la que se puedan manejar y visualizar de una manera muy sencilla todos los datos y variables que intervienen en el proceso. Para ello hemos usado QT Designer.

Qt es un entorno de desarrollo integrado, creado por la empresa noruega Trolltech, que sirve principalmente para el diseño y creación de entornos gráficos. La elección de este software se debe principalmente a que soporta el lenguaje Python para su desarrollo, además de ser gratuito y de ofrecer una interfaz de diseño muy clara y evidente. Por último, se ha de destacar la gran capacidad con la que este software cuenta, siendo usado en diversas industrias como la automoción (el famoso fabricante Mercedes lo usa para sus cada vez más avanzados sistemas de control), aplicaciones médicas, interfaces de las televisiones inteligentes…

A continuación, se mostrará y explicará cuales son las funcionalidades con las que cuenta nuestra aplicación.

4.2 Diagrama de Casos de Uso

Antes de entrar en el análisis del diagrama, se ha de mencionar que la aplicación desarrollada para este proyecto no es profesional, ni demasiado compleja, es decir, que simplemente cumple la función de analizar y comprobar los resultados que se pueden llegar a obtener en diferentes sistemas. Tan sólo contará con un perfil de usuario, tal y como veremos en el diagrama.

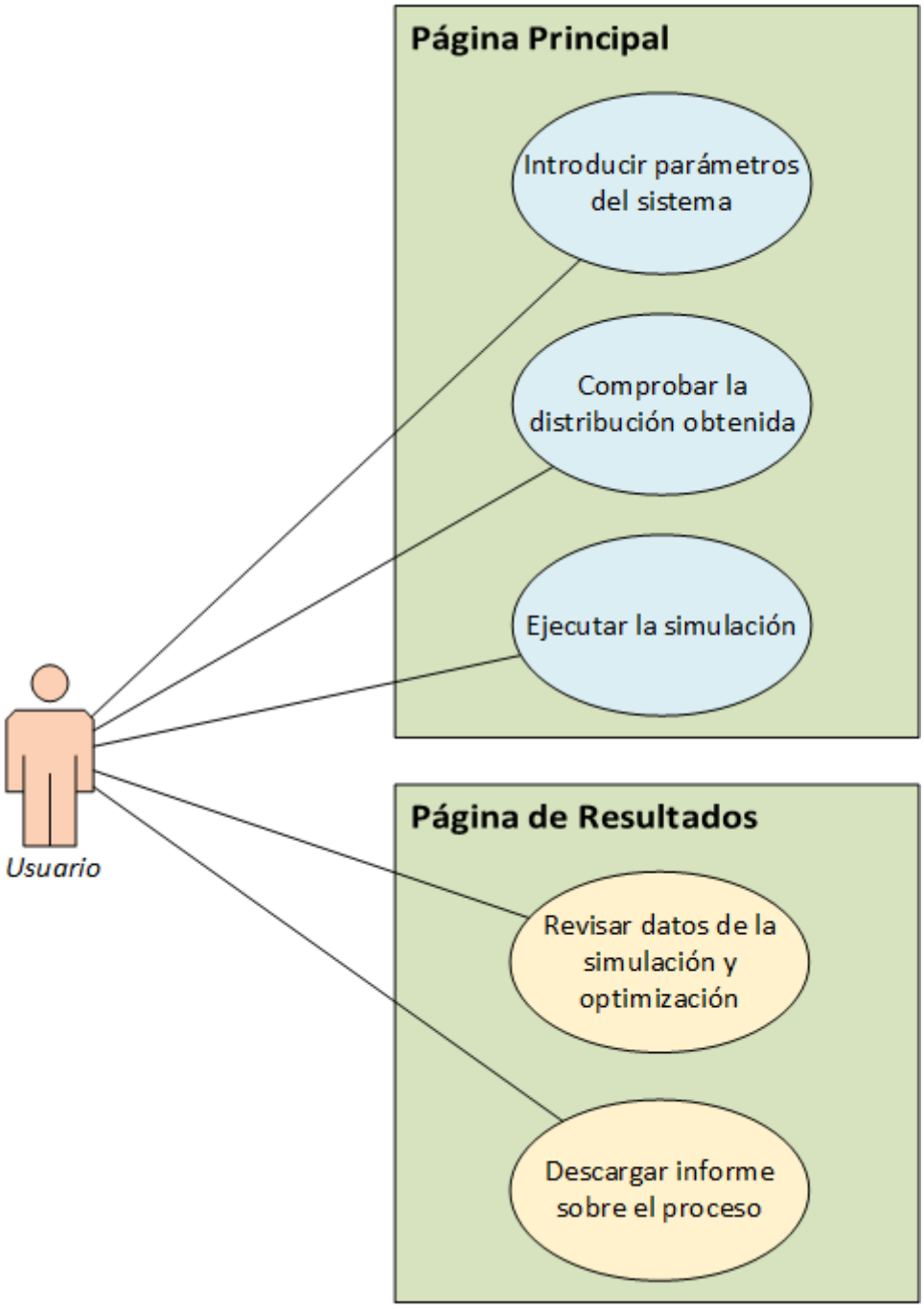


Figura 8: Diagrama de casos de uso

4.3 Diagrama de Actividades

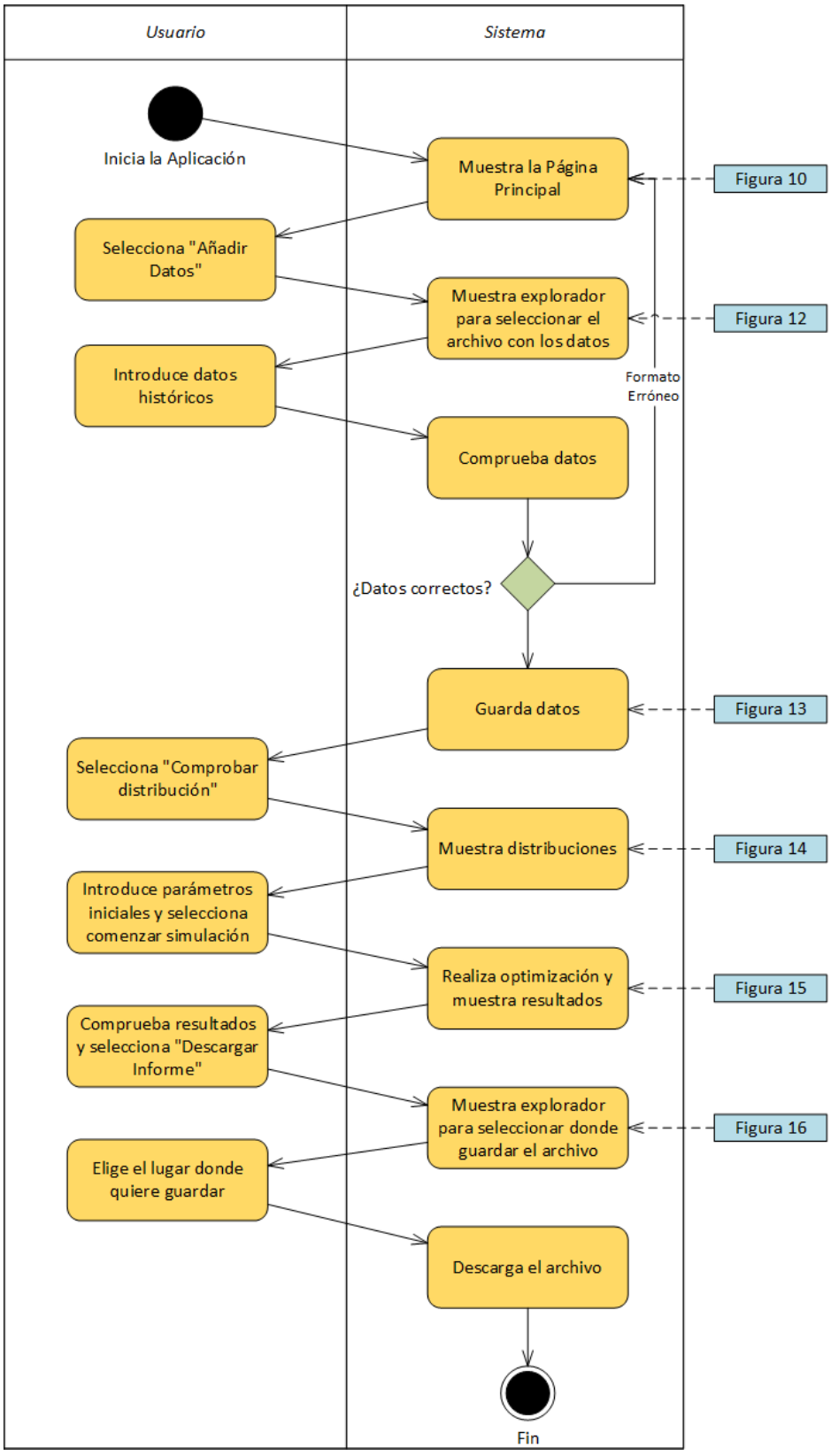


Figura 9: Diagrama de actividades de la aplicación

Una vez hemos conocido el diagrama de casos de uso y el de actividades de nuestra aplicación, analizaremos cada uno de los puntos que los conforman. Para ello nos ayudaremos de capturas tomadas de nuestra aplicación, puesto que de esa manera será mucho más sencillo explicar todas las funcionalidades con las que esta cuenta.

4.4 Desglose del Diagrama de Actividades

Al iniciar nuestra aplicación, lo primero con lo que nos encontramos es con la página principal de esta. Para aclarar los valores a introducir en cada una de las casillas, por si con la descripción en la aplicación no fuera suficiente, comentaremos brevemente cada una de ellas.

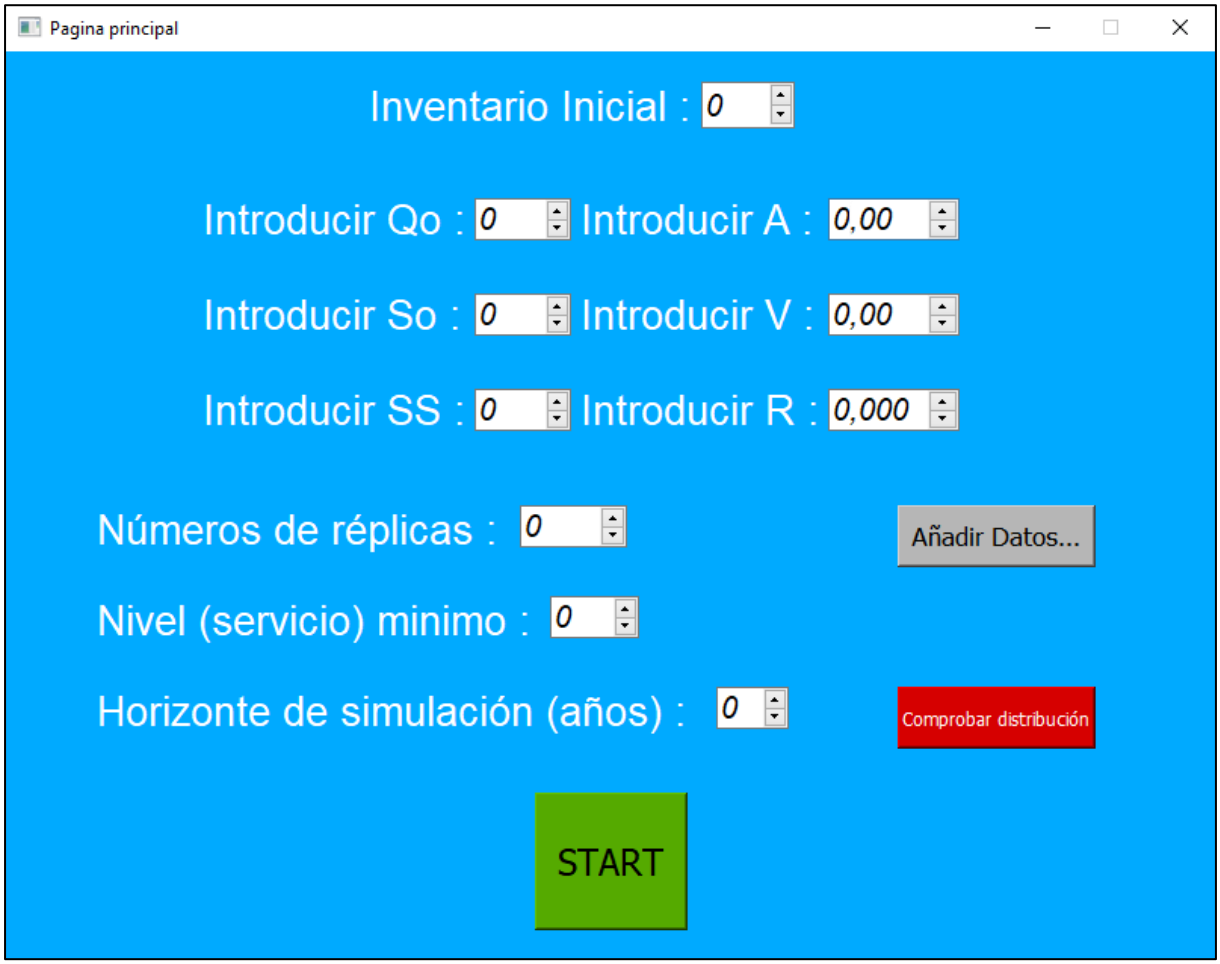


Figura 10: Página principal de la aplicación

En el primer casillero, indicaremos el inventario inicial con el que contamos.

Las 3 siguientes casillas de la columna de la izquierda hacen referencia a lo siguiente:

- Tamaño inicial del lote de aprovisionamiento
- Punto de pedido inicialmente establecido
- Stock de seguridad

En la columna derecha, indicaremos nuestros parámetros relativos a los costes:

- Coste de pedir
- Coste unitario del producto
- Tasa aplicada a mantener el inventario

Finalmente, los 3 casilleros restantes corresponden a variables relativas a la simulación que se pretende realizar, estos son:

- Número de simulaciones a realizar
- Nivel de servicio mínimo que se desea tener
- Intervalo de tiempo establecido en cada simulación

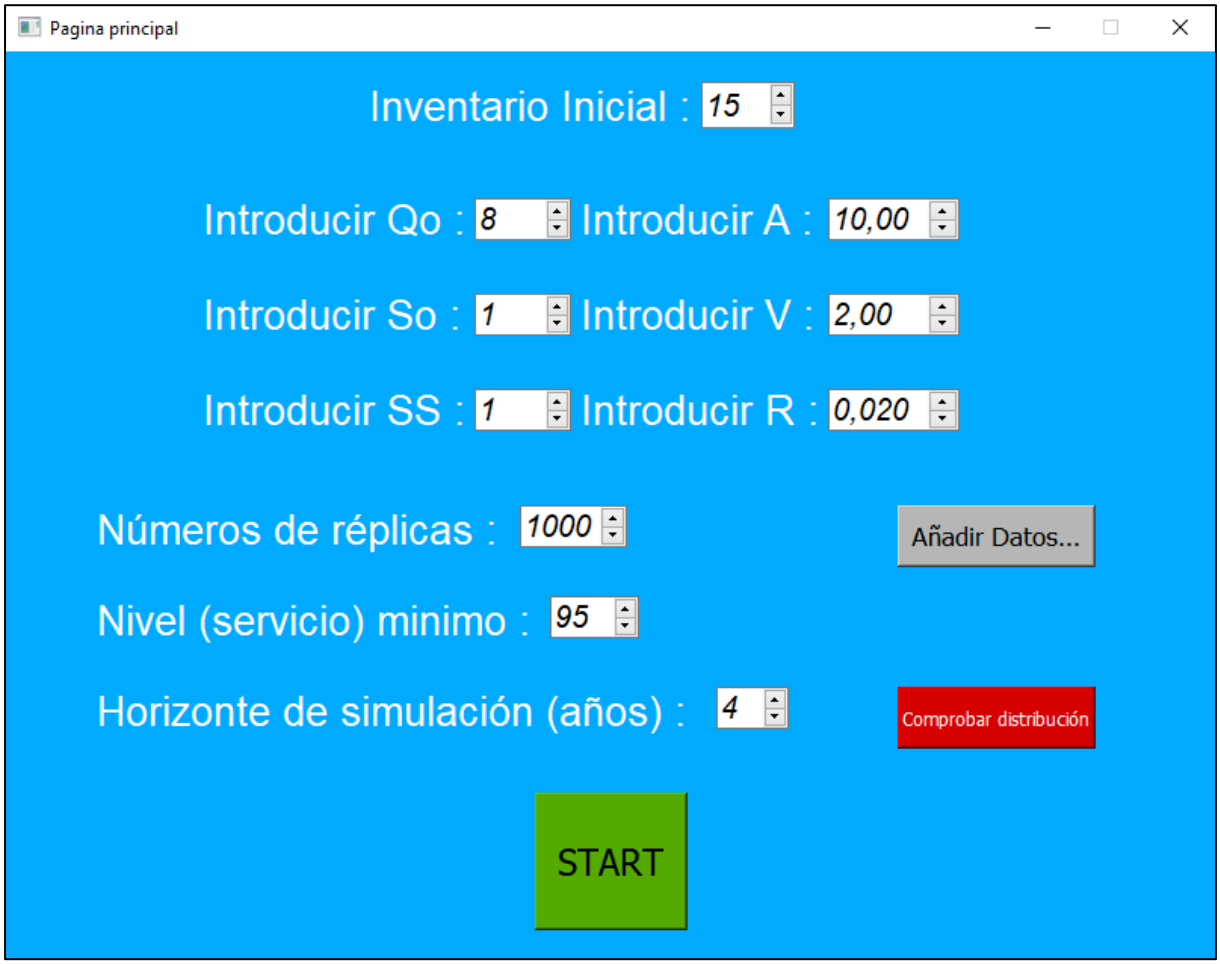


Figura 11: Página principal con datos de ejemplo

Una vez que se han completado todos los campos requeridos, tocará añadir los tiempos históricos que queramos analizar. Antes de continuar, me gustaría comentar que si el usuario intenta comprobar la distribución a la que se ajustan los datos sin haberlos introducido previamente, saldrá un mensaje de error, indicándole la forma correcta en la que actuar.

Cuando se presione sobre el botón *"Añadir Datos"*, se abrirá una ventana como esta:

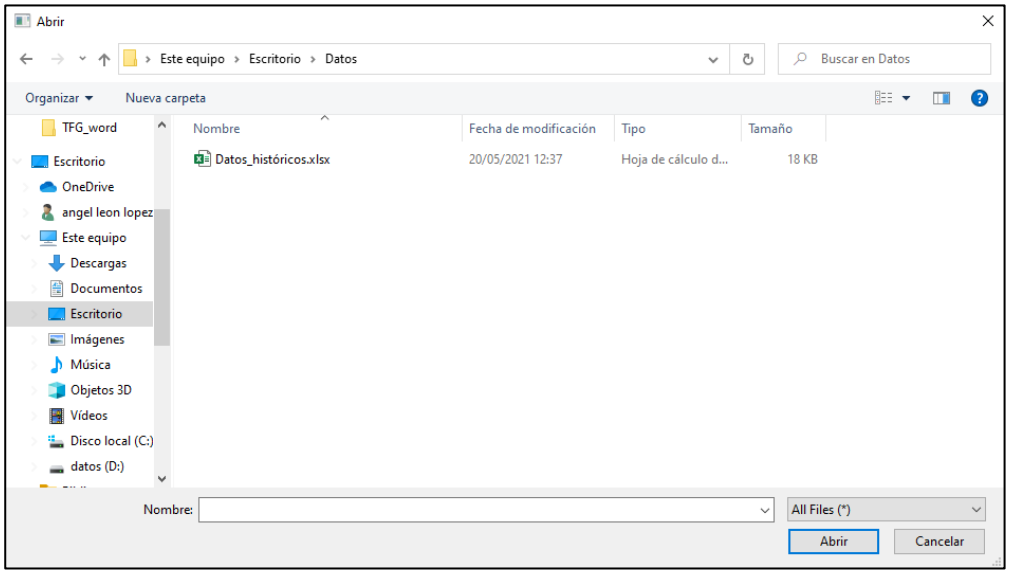


Figura 12: Explorador de archivos para cargar datos

En el explorador de archivos, deberemos seleccionar el archivo Excel, donde habremos introducido previamente nuestros datos. Para que el programa lea correctamente los valores, se tendrán que crear dos columnas, la primera de ellas bajo el nombre de *"Demanda"*, y la segunda *"PA"*. Debajo de cada columna se irán introduciendo fila por fila, los tiempos que queramos usar para la simulación.

Si el programa ha leído correctamente los datos, se nos indicará bajo un mensaje en el botón, dándonos también la posibilidad de cargar otros tiempos distintos. Se vería de la siguiente manera:

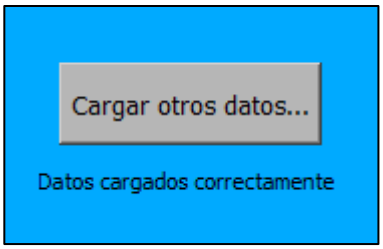


Figura 13: Mensaje confirmación de carga

Una vez añadido al sistema los datos históricos se da la posibilidad al usuario de conocer esta información antes de comenzar la optimización, simplemente pulsando el botón *"Comprobar distribución"*.

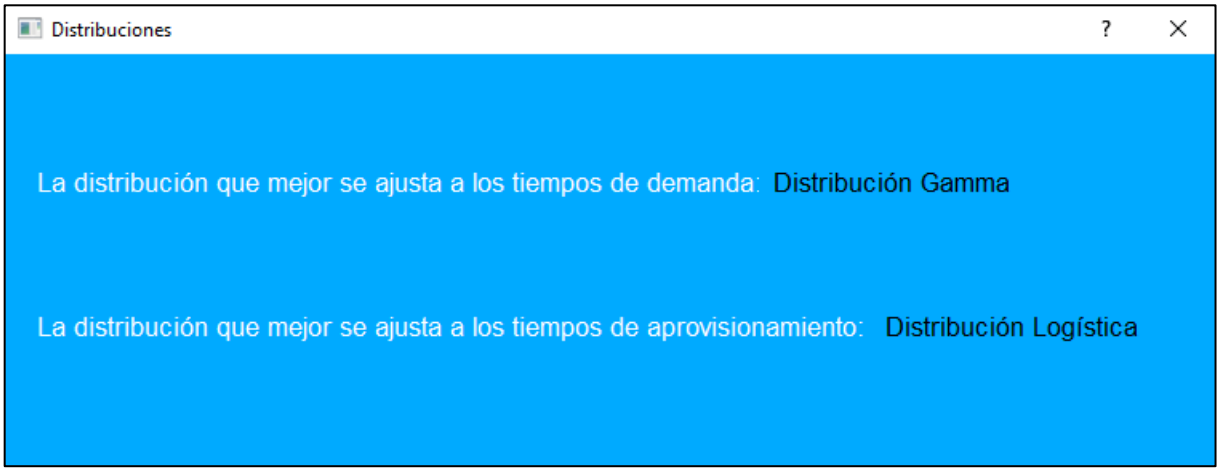


Figura 14: Ventana con las distribuciones

Cuando hayamos establecido todos los parámetros necesarios e introducido los datos relativos a los tiempos históricos, podremos iniciar el programa. Para ello simplemente habrá que pulsar el botón verde *"START"*.

El código que contiene la simulación, el cual se encuentra disponible en el anexo I de la memoria, se pondrá en marcha y dependiendo del número de simulaciones establecido y, sobre todo, de la capacidad de computación del ordenador, tardará más o menos.

Una vez que el dispositivo termine de realizar los cálculos, la aplicación nos mostrará los resultados en una nueva ventana.

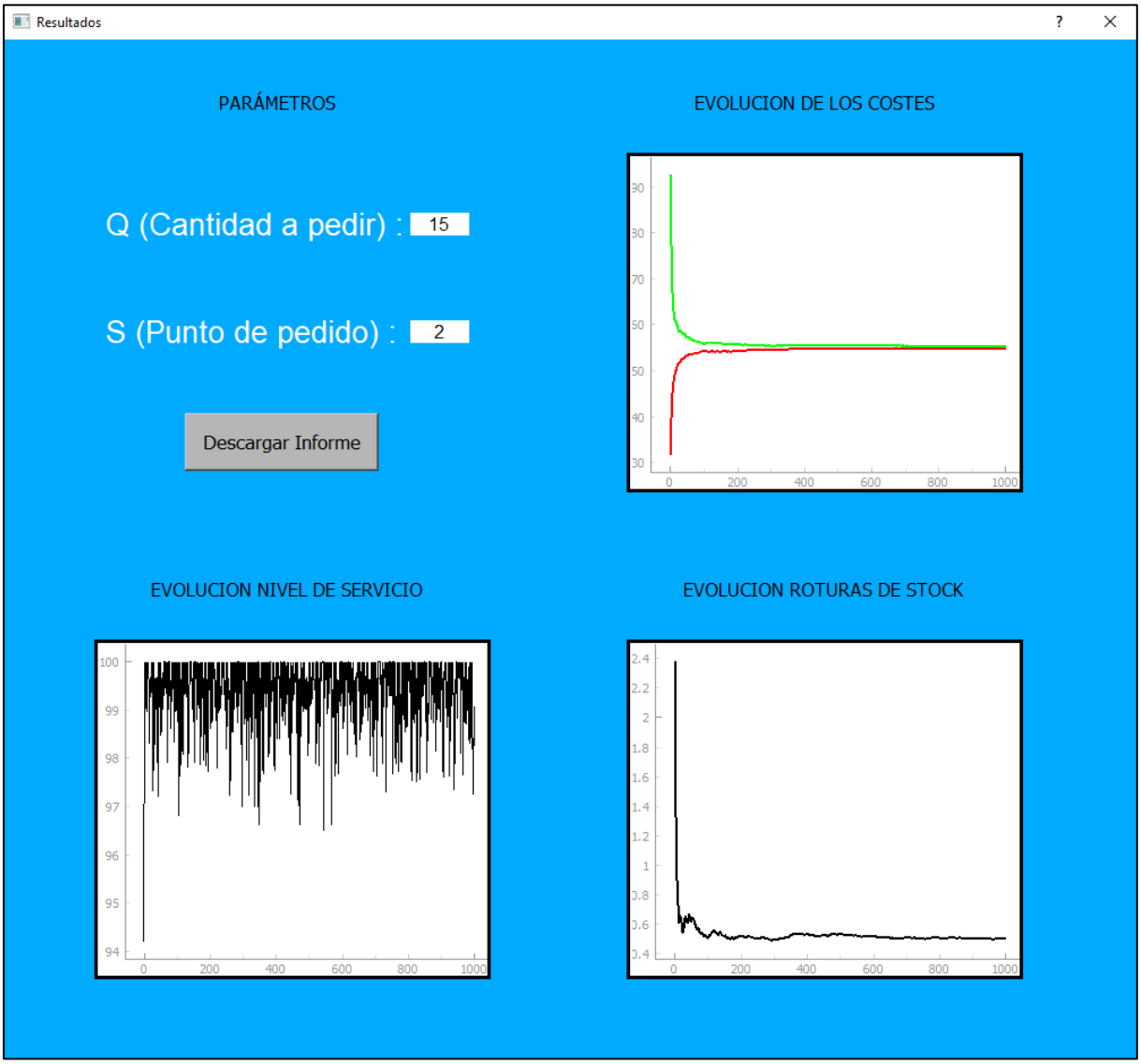


Figura 15: Ventana con los resultados obtenidos

Como podremos observar en la ilustración, se detallará la evolución del nivel de servicio, la de las roturas de stock y la de los costes, tanto el de mantenimiento como el de lanzamiento. Además, se añadirán los valores de *Q* y el punto de pedido ya optimizados.

En esta misma ventana se añadió la posibilidad de generar un informe sobre la simulación y la optimización que había tenido lugar. Para descargarlo, lo único que hay que hacer es pulsar el botón *"Descargar Informe"*. Inmediatamente después de clicar sobre este, se nos abrirá un explorador de archivos muy similar al expuesto más arriba, diferenciándose con él únicamente en que en este deberemos seleccionar la raíz en la cual queremos guardar el archivo de Word generado.

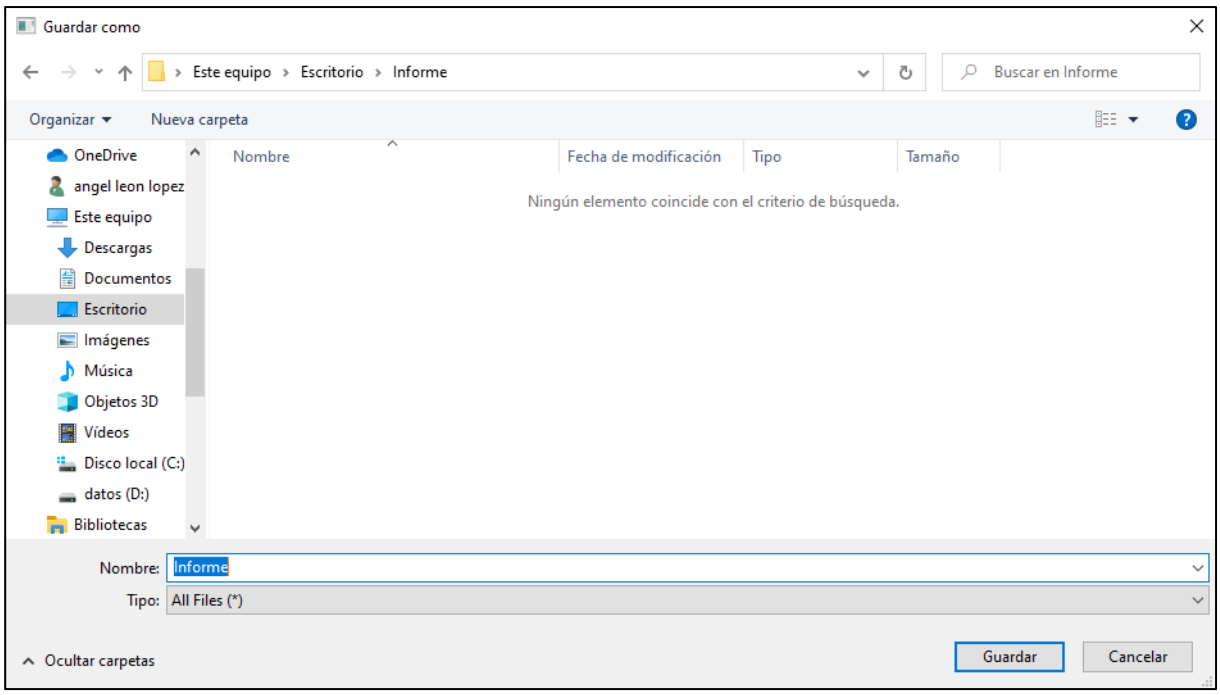


Figura 16: Explorador de archivos para guardar informe

Como pasaba a la hora de cargar los datos, la aplicación nos mostrará un mensaje de confirmación de descarga y, además se nos dará la opción de volver a obtener el documento.

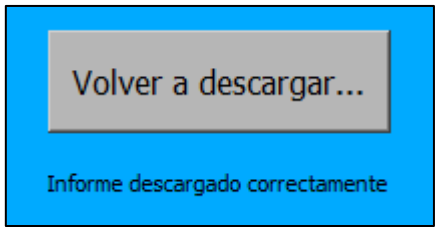


Figura 17: Volver a descargar

En el informe se detallarán más a fondo las gráficas que vimos en la ventana de resultados. Además de una serie de recomendaciones dirigidas hacia el usuario y el resultado del contraste de hipótesis realizado previamente. También contendrá una serie de tablas con más datos acerca de la simulación, tanto previos a esta, como posteriores.

4.5 Diseño de la Aplicación

Antes de continuar, expondremos brevemente la interfaz con la que se ha diseñado la aplicación, detallando los puntos que destacan en ella.

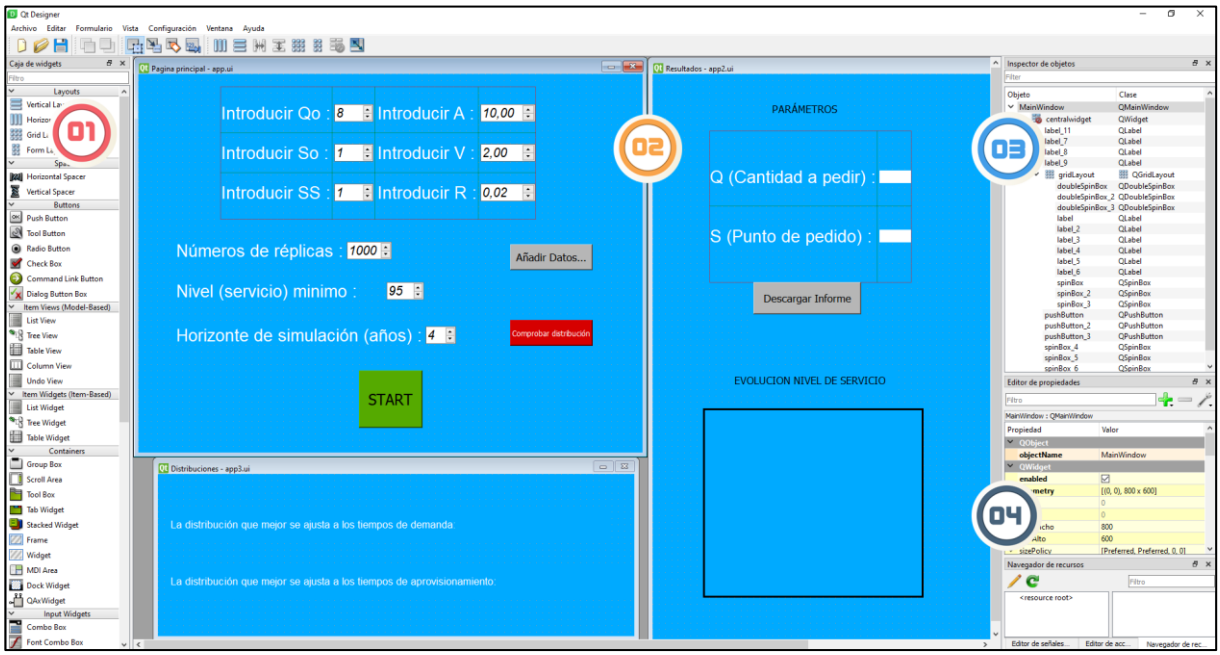


Figura 18: Qt Designer

Como se comentó previamente, el programa usado para el diseño de nuestra aplicación ha sido el Qt Designer. Gracias a la numeración de la imagen, iremos explicando cada una de las partes de las cuales se compone el programa.

1. En esta parte, se encuentran todas las herramientas y funcionalidades que podremos añadir a nuestra aplicación. Para ello simplemente habrá que arrastrarlas hasta nuestra área de trabajo. Aquí se encuentran los botones, las etiquetas, los casilleros…
2. Ahora analizaremos el área de trabajo donde elegiremos la distribución de cada uno de los elementos de las ventanas con las que cuenta nuestra aplicación. Además, podremos cambiar el tamaño y los estilos de la tipografía existentes en ella.
3. En este punto tendremos una tabla con los elementos añadidos a la ventana que tengamos seleccionada en cada instante. Es muy importante recordar los nombres establecidos para cada uno de estos, puesto que luego para dotarlos de funcionalidad los necesitaremos.
4. Finalmente, en esta parte seremos capaces de editar algunas de las características del elemento que tengamos seleccionado. Podremos establecer el tamaño de ventana que deseemos, el tipo de cursor que aparezca cuando estemos encima del elemento…

4.6 Librería Usada para el Desarrollo de la Aplicación

En este apartado detallaremos cuales han sido las librerías usadas en el código desarrollado para dotar de funcionalidad a nuestra aplicación. Se comentará brevemente las principales características de cada una de ellas, para comprender mejor el porqué de su inclusión en el código implementado.

- Sys: Este módulo proporciona funciones y variables que son usadas para manipular diferentes partes del entorno de ejecución de Python. Podremos acceder a parámetros y funciones específicos del sistema.
- PyQt5: De esta librería vamos a importar todos los módulos necesarios para poder desarrollar todas las funcionalidades que queremos que tenga nuestra interfaz. Está enfocada especialmente hacia el desarrollo de aplicaciones por lo que será una de las librerías más importantes que usemos.
- Copia: Este es el nombre del otro archivo Python, en el cual se profundizará más adelante, y que contiene todo el código de la simulación y optimización de nuestro sistema. Es necesario importarlo como si fuese una librería para poder llamarlo como función en nuestra aplicación.
- Pyqtgraph: Contiene todas las funciones gráficas que nos hará falta para la representación de los datos obtenidos en la simulación. Es una librería particular ya que se creó exclusivamente para el uso en aplicaciones creadas a través de QT.
- Pandas: Es una librería especializada en el manejo y análisis de estructuras de datos. Nos hará falta para importar los tiempos históricos de demanda y del plazo de aprovisionamiento.
- Docx: Gracias a esta, podremos generar un archivo .docx, editándolo y rellenándolo como se desee, haciendo uso exclusivo del lenguaje de programación Python. Con esta librería seremos capaces de programar que se genere un informe una vez completada la simulación.
- Matplotlib: Librería similar a Pyqtgraph. Con esta podremos introducir cualquier tipo de gráfico que se requiera en el informe que se pretende generar. Es una de las mejores en cuanto a lo que representación gráfica respecta. Cuenta con bastantes opciones de edición, lo que hace que podamos tener un sinfín de posibilidades de representaciones.
- Datetime: Nos servirá únicamente para indicar la fecha y la hora en la cual hemos realizado la simulación. Esta información irá expuesta en el informe al final de cada una de las páginas.
- Contrastep: Este es el nombre de otro archivo Python, el cual se explicará más adelante, y que contiene el código que identificará la distribución que mejor se ajusta a nuestros datos históricos. Es necesario importarlo para poder llamarlo como función en nuestra aplicación.
- Io: Nos servirá únicamente para poder introducir las gráficas generadas en el informe sin necesidad de tener que descargarlas previamente en nuestro dispositivo.

4.7 Librería Usada para el Desarrollo de la Simulación y el Contraste

En el caso de la librería usada para llevar a cabo la simulación, optimización y el contraste de hipótesis, fue la siguiente:

- Random: Como su propio nombre indica, con esta librería podremos generar números aleatorios, basándonos en la distribución estadística que corresponda.
- Simpy: Es la librería más importante que usaremos en nuestro proyecto. Simpy, puede generar simulaciones basadas en eventos discretos. Gracias a ella, seremos capaces de ir ajustando y calculando los valores de los parámetros que juegan un papel fundamental dentro de nuestra optimización.
- Statistics: Esta librería nos proporcionará una serie de funciones con las que podremos calcular una enorme variedad de parámetros estadísticos, tales como, la media, desviaciones típicas, coeficiente de variación de nuestros datos...
- Numpy: Es una de las librerías más usadas dentro del lenguaje de programación python. Nos dará soporte para la creación de vectores y matrices multidimensionales, además de, una diversa colección de funciones matemáticas las cuales podremos usar para facilitarnos algunas operaciones.
- Scipy: Módulo que contiene una gran variedad de distribuciones estadísticas. Esta será la librería que se usará para realizar el contraste de hipótesis y obtener así las distribuciones que mejor se ajustan a nuestros tiempos históricos.
- Math: Librería que nos proporcionará algunas funciones matemáticas que necesitaremos a lo largo del proceso de optimización.

4.8 Diagrama de Flujo de la Simulación y Optimización

En el capítulo en el cual se indicó la solución propuesta, se expuso un diagrama de flujo en el cual se podía ver cuál era la manera en la que se pretendía obtener los valores optimizados. El problema de aquel gráfico era que cuando se llegaba a la simulación, no se explicaba el proceso que estaba teniendo lugar en ese instante.

Por ello se ha decidido realizar un diagrama más amplio en el cual se explica más a fondo como es el proceso de simulación y optimización de los parámetros de nuestro sistema. Como podremos ver en el gráfico, habrá algunas partes copiadas del último diagrama para que no quede nada sin explicar.

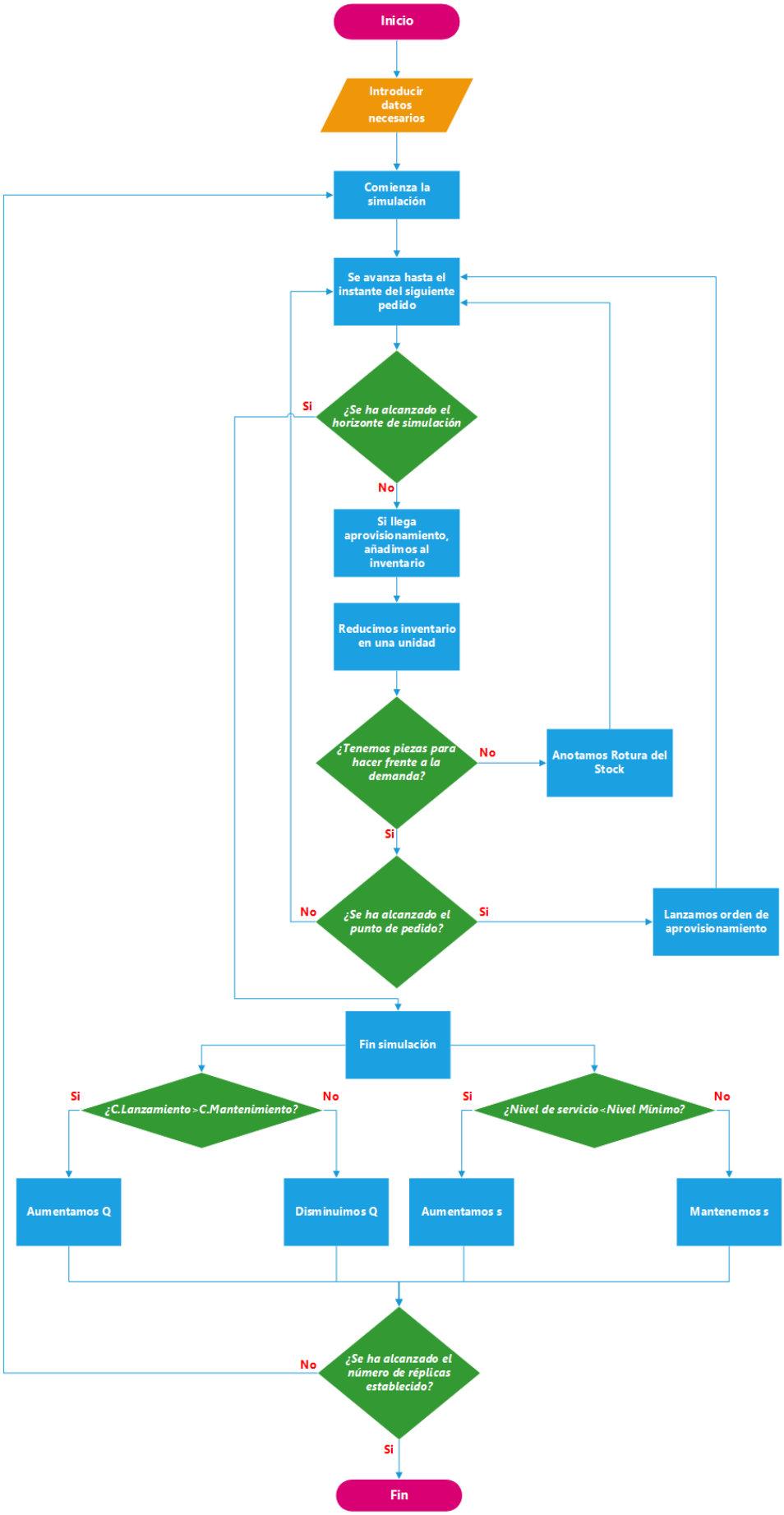


Figura 19: Diagrama de flujo completo

4.8.1 Análisis del Diagrama

En primer lugar, tendremos que introducir los datos necesarios. En este diagrama de flujo se ha resumido la parte en la cual el usuario ha de introducir los tiempos históricos y los parámetros más relevantes de nuestro sistema. En el capítulo donde se expuso la solución propuesta se puede ver esta parte con más claridad.

En este gráfico se ha querido dar más importancia y nivel de detalle al proceso de simulación del sistema. Cuando este comienza, según la distribución obtenida, el sistema generará un valor que nos marcará el tiempo que pasará hasta que se realice un nuevo pedido. En caso de que en ese intervalo se haya alcanzado el horizonte de simulación establecido, daremos por finalizada la simulación. Si todavía no hemos alcanzado ese punto, debemos continuar con esta.

En caso de que nos llegue algún pedido realizado con anterioridad a nuestro proveedor, este, se deberá de añadir al inventario existente. A continuación, reduciremos en una unidad nuestro stock de piezas de repuesto, que tal y como se argumentó previamente, siempre será 1. Llegados a este punto el programa se hará una pregunta, ¿hay suficientes piezas en nuestro inventario para hacer frente al pedido?

En caso negativo se anotará que hemos tenido una rotura de stock, y seguiremos con el ciclo hasta que el lote de aprovisionamiento ordenado previamente llegue a nuestro almacén. Si tenemos piezas de sobra, continuaremos con el proceso, donde tocará comprobar si hemos alcanzado el punto de pedido, puesto que de ser eso cierto, lanzaremos una nueva orden de aprovisionamiento para posteriormente, continuar con la simulación. Si no estamos en niveles del punto de pedido, se seguirá avanzando en el tiempo hasta que llegue un nuevo pedido.

Hay una condición en el diagrama que dará por terminada la simulación cuando se alcance el horizonte establecido antes de comenzar con esta. Cuando se llegue, comenzaremos con el proceso de optimización.

Esta parte es exactamente igual que la que se expuso en el capítulo de la solución propuesta. A lo largo de la simulación, se han ido calculando los costes de mantenimiento y de lanzamiento que se han tenido durante esta, además del nivel de servicio existente.

Como ya se analizó en ese apartado, buscamos que los costes sean lo más parecido posible puesto que de esa manera estaríamos alcanzando un punto óptimo en cuanto a este aspecto respecta. Se irá ajustando Q al final de cada simulación de manera que se vaya alcanzando la situación comentada.

Por otra parte, antes de comenzar con la simulación, el usuario estableció un nivel de servicio mínimo que desearía tener en la compañía, por lo que como pasaba con los costes, aquí también iremos realizando un ajuste, pero en este caso del punto de pedido.

Cuando terminemos todo el proceso de optimización de las variables, el programa comprobará si se ha alcanzado el número de réplicas establecido al principio de todo el proceso. En caso de que no hayamos terminado todavía, el programa volverá a realizar una nueva simulación, repitiendo todo el proceso de nuevo. En cambio, cuando se alcance este valor, el proceso terminará tal y como queda indicado en el diagrama.

4.9 Comentarios Finales

Como hemos podido comprobar a lo largo del capítulo, la simulación llevada a cabo tiene una serie de particularidades debido principalmente a que nuestro inventario está compuesto por piezas de repuesto, que hacen que debamos de tener en cuenta una serie de consideraciones durante el proceso.

5 CASO DE ESTUDIO

En este capítulo realizaremos una comparación teórica-práctica. Generaremos una serie de distribuciones con diferentes coeficientes de variación para posteriormente obtener una serie de valores los cuales cotejaremos con los diferentes resultados alcanzados en los demás cálculos.

Como ya sabemos, en el caso en el que tanto la demanda como los plazos de aprovisionamiento históricos se puedan considerar ambos como variables aleatorias normales, podremos obtener una pequeña estimación del valor que van a adquirir estas. De hecho, las pocas aproximaciones existentes para la obtención de estos parámetros son para unos datos que se rigen por una distribución normal. En el caso de que esto no sea así, se debe buscar otro procedimiento de resolución, siendo este otro de los principales motivos por el que se decidió llevar a cabo este proyecto.

Todas las expresiones usadas en este capítulo están sacadas del libro "*Administración de empresas para los grados de ingeniería"* [5] . Hay que añadir que todas las comparaciones se han realizado con un inventario inicial de 15 unidades.

5.1 Distribución Normal 1

Antes de comenzar con los cálculos, se expondrán los valores de los parámetros que definen a los datos históricos que usaremos. Estos son:

- Media: $\mu_D = \frac{1}{6{,}21}$ y $\mu_{PA} = 11{,}21$
- Coeficiente de variación: $cv_D = 0{,}53$ y $cv_{PA} = 0{,}29$

El primer valor corresponde con los tiempos entre pedidos y el segundo con los de aprovisionamiento. Las unidades en las que están obtenidos los valores medios son en días.

Los parámetros iniciales que se introducirán en nuestro sistema son:

- *A:* 10
- *v:* 2
- *r:* 0,02

5.1.1 Resultados Teóricos

Empezaremos con el cálculo de *Q*. Para ello usaremos la siguiente expresión:

$$Q = \sqrt{\frac{2 \cdot A \cdot D}{v \cdot r}}$$

Para obtener el valor de *D*, se ha tenido que realizar la inversa de la media obtenida a partir de nuestros tiempos históricos. Esto es debido a que estos datos son el tiempo medio que pasa entre

un pedido y otro, y en la formula, se deberá de introducir el número de piezas requeridos por día, que como nos podremos imaginar, será un número menor de la unidad.

El resultado obtenido, una vez aplicada una aproximación hacia su valor entero, ha sido:

$$Q = 9$$

A continuación, calcularemos el punto de pedido. Para ello, deberemos de definir previamente el porcentaje de ruptura (*Pr*) mínimo que se pretende tener en el sistema. Este, lo estableceremos en un 10%. Al igual que ocurría con *Q*, el punto de pedido también tiene una expresión a través de la cual podemos realizar la estimación:

$$s = \mu_{PA} \cdot \mu_D + z \cdot \sqrt{\mu_D{}^2 \, \sigma_{PA}{}^2 + \mu_{PA}{}^2 \, \sigma_D{}^2}$$

Analizando cada uno de los términos de la expresión anterior, podremos observar que hay uno de ellos, que es el estadístico *z*, cuyo valor deberemos obtener. Este, representa las veces que se aleja la desviación estándar del valor central para alcanzar el nivel de servicio que se ha establecido previamente (1-*Pr*).

Para una calidad de servicio del 90%, el valor del estadístico obtenido es *z* = 1,28155.

Con los valores medios de cada una de las variables, y con los coeficientes de variación, seremos capaces de sacar sus respectivas desviaciones, de manera que solo tendremos que sustituir los valores en la expresión para obtener el punto de pedido. Haciendo todo esto, el resultado, ajustado a su valor entero más próximo, es el siguiente:

$$s = 3$$

También podremos obtener, en función del del estadístico sacado a partir del porcentaje de ruptura establecido, el stock de seguridad. Para ello usaremos la siguiente expresión:

$$ss = z \cdot \sqrt{\mu_D{}^2 \, \sigma_{PA}{}^2 + \mu_{PA}{}^2 \, \sigma_D{}^2}$$

Sustituyendo y redondeando valores, tenemos que el stock de seguridad que deberíamos de establecer sería de:

$$ss = 1$$

5.1.2 Resultados Prácticos

Ahora, haciendo uso de los mismos tiempos históricos cuyas características acabamos de ver, trataremos de analizar los resultados que obtendremos haciendo uso de nuestra aplicación. Como ya hemos visto en capítulos anteriores, necesitaremos introducir una serie de datos iniciales, estos son:

- *Qo:* 7
- *So:* 1
- *ss:* 1
- *A:* 10
- *v:* 2
- *r:* 0,02

Haremos 500 simulaciones en donde en cada una de ellas tendremos un horizonte de 4 años. Estableceremos un nivel de servicio mínimo de 90%.

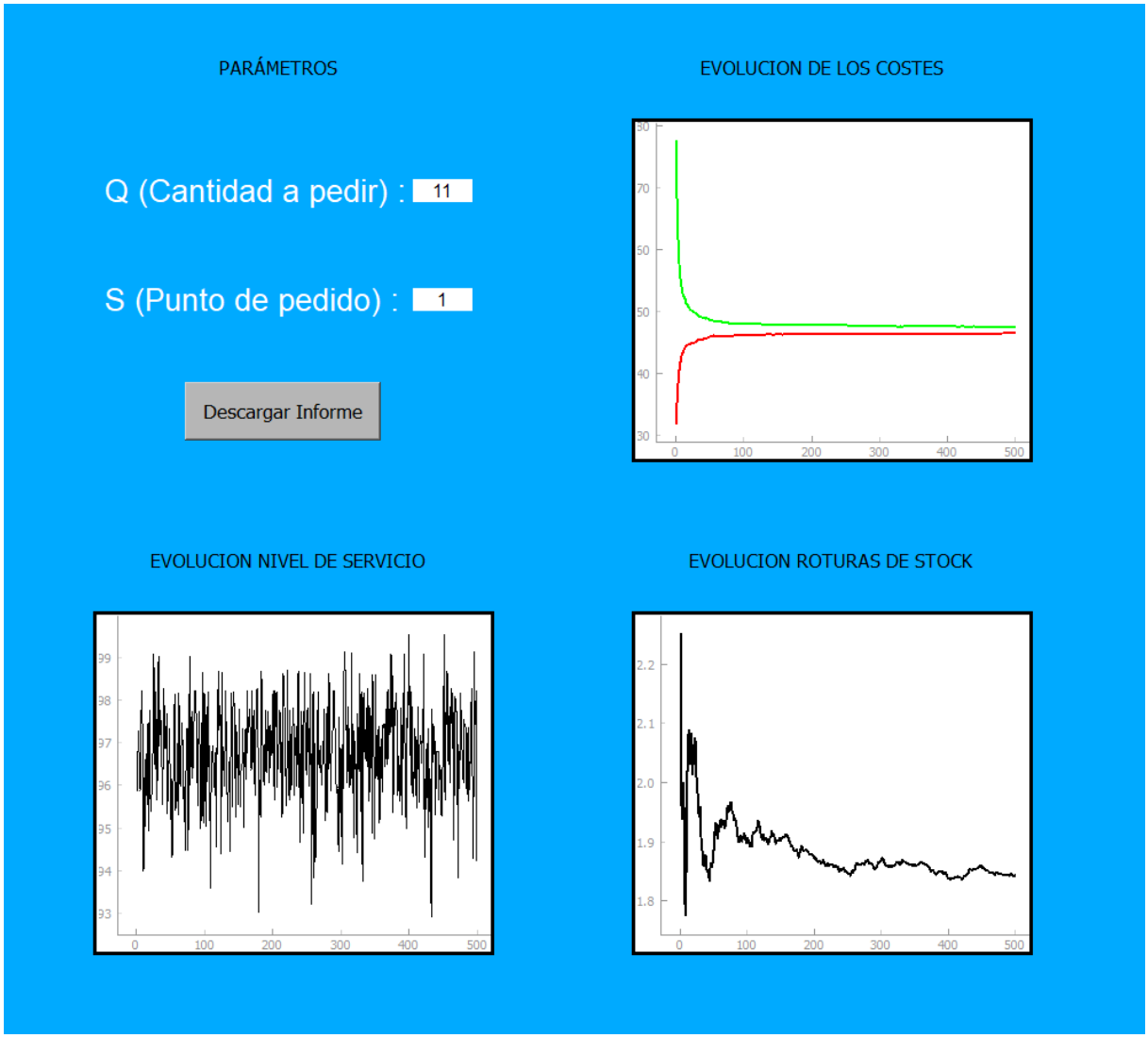


Figura 20: Resultados simulación 1

En esta ilustración, podemos observar los resultados de aplicar los parámetros anteriores en nuestra aplicación. Los datos más relevantes indicados en esta y en el informe generado son:

- *Q:* 11
- Punto de pedido: 1
- Nivel de servicio medio ofrecido: 96,77%

5.1.3 Comparación y Conclusión

Empezaremos analizando el valor de *Q*, el cual como podremos observar, es muy parecido en los dos ámbitos. En los cálculos hechos por la aplicación, se ha ido observando que, debido a la demanda, se han tenido que ir haciendo un buen número de pedidos a nuestro proveedor, elevándose así los costes de lanzamiento. La aplicación ha corregido esto, aumentando *Q* para equilibrar los costes y conseguir acercarnos al punto óptimo. Esto, es algo que nuestro modelo teórico no puede hacer, aunque aun así se ha obtenido una muy buena aproximación.

En el punto de pedido, vemos que la diferencia es algo mayor. Si recordamos el coeficiente de

variación que tenía la demanda, veremos que este es notablemente alto, lo que provocará una situación en la cual los pedidos serán difíciles de prevenir.

En los cálculos teóricos obtendremos un punto de pedido establecido con una relativa antelación, es decir, realizaremos un pedido a nuestro proveedor teniendo aún 3 unidades en el inventario, esto es debido principalmente a la variabilidad de los datos históricos y al considerable *PA* medio con el que estos cuentan. En cambio, nuestro programa obtiene un nivel de servicio por encima del deseado con un punto de pedido establecido en 1. Esta diferencia se debe principalmente al tiempo medio entre demandas de los datos históricos, el cual es demasiado grande, por lo que la aparición de rupturas de stock en nuestro sistema no será muy numerosa.

5.2 Distribución Normal 2

En este segundo caso analizaremos unos datos que cuentan con las siguientes características:

- Media: $\mu_D = \frac{1}{5{,}55}$ y $\mu_{PA} = 10{,}35$
- Coeficiente de variación: $cv_D = 0{,}35$ y $cv_{PA} = 0{,}62$

El primer valor corresponde con los tiempos entre pedidos y el segundo con los de aprovisionamiento. Las unidades en las que están obtenidos los valores medios son en días.

Los parámetros iniciales que se introducirán en nuestro sistema son:

- *A:* 10
- *v:* 2
- *r:* 0,02

5.2.1 Resultados Teóricos

Como en el ejemplo previo ya se detallaron y explicaron todas las expresiones que usaremos para calcular los valores de los parámetros, se expondrán directamente todos los resultados obtenidos a partir de los datos con las características mencionadas anteriormente. No sin antes añadir que se establecerá un porcentaje de ruptura del 5%. Los valores calculados han sido:

- *Q:* 10
- Punto de pedido: 4
- Stock de seguridad: 2

5.2.2 Resultados Prácticos

Tal y como hicimos en el primer ejemplo, expondremos los parámetros iniciales introducidos en nuestro programa, estos han sido:

- *Qo:* 2
- *So:* 1
- *ss:* 1
- *A:* 10
- *v:* 2
- *r:* 0,02

Haremos 500 simulaciones en donde en cada una de ellas tendremos un horizonte de 4 años. Estableceremos como nivel mínimo de servicio, al igual que pasaba con el apartado teórico, de 95%.

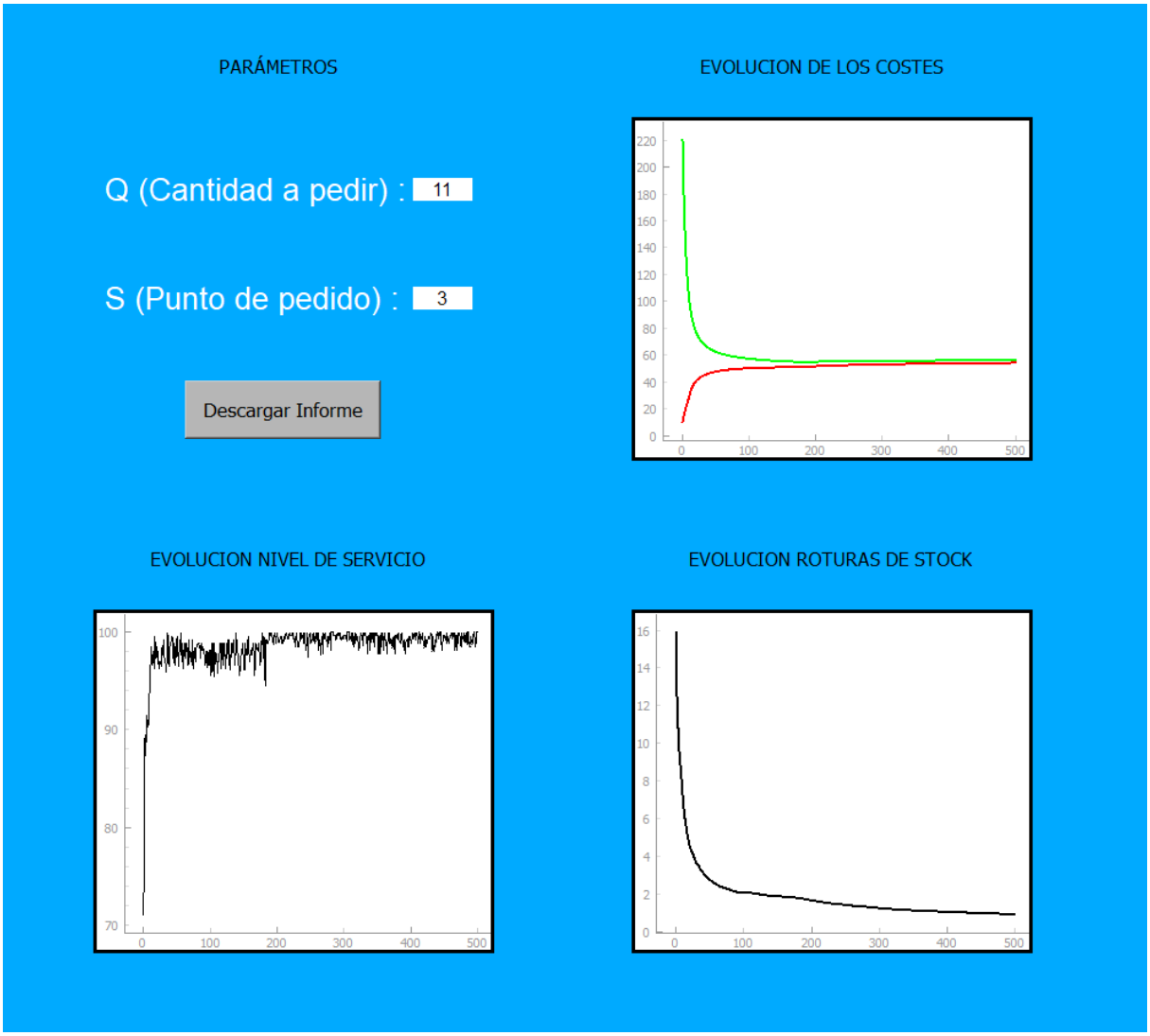


Figura 21: Resultados simulación 2

En esta ilustración, podemos observar los resultados de aplicar los parámetros anteriores en nuestra aplicación. Los datos más relevantes indicados en esta y en el informe generado son:

- *Q:* 11
- Punto de pedido: 3
- Nivel de servicio medio ofrecido: 98,41%

5.2.3 Comparación y Conclusión

Si tomamos los coeficientes de variación de los datos históricos usados en esta ocasión, observamos que el de los tiempos entre demandas tiene un valor bajo, en cambio el de los plazos de aprovisionamiento es considerablemente alto. Si recordamos el análisis hecho sobre los sistemas gestión del inventario de piezas de repuesto, en estos, el plazo de aprovisionamiento no tenía tanta importancia como los tiempos entre demandas, por lo que el hecho de que el *PA* tenga tanta variabilidad no dificultará demasiado la optimización de los parámetros del sistema.

Al exigir un nivel de servicio tan alto, los parámetros optimizados tendrán unos valores considerables, es decir, pediremos con relativa antelación y lo haremos en lotes grandes para poder reducir los costes. En la gráfica se puede observar claramente como se reducen las rupturas de stock en nuestro sistema gracias al aumento del punto de pedido.

5.3 Distribución Normal 3

En este ejemplo realizaremos un estudio con unos valores más parecidos a los que tendremos en un entorno en el que se traten piezas de repuesto. Hemos venido estableciendo un *PA* superior a los tiempos entre demandas, y si recordamos lo que se explicó en la memoria, esto no era generalmente así.

En este tercer caso analizaremos unos datos que cuentan con las siguientes características:

- Media: $\mu_D = \frac{1}{5,2}$ y $\mu_{PA} = 4,4$
- Coeficiente de variación: $cv_D = 0,65$ y $cv_{PA} = 0,22$

El primer valor corresponde con los tiempos entre pedidos y el segundo con los de aprovisionamiento. Las unidades en las que están obtenidos los valores medios son en días.

Los parámetros iniciales que se introducirán en nuestro sistema son:

- *A:* 10
- *v:* 2
- *r:* 0,02

5.3.1 Resultados Teóricos

Usaremos las mismas expresiones y el mismo porcentaje de ruptura (5%), por lo que el valor del estadístico será de *z* = 1,64485. Los resultados obtenidos:

- *Q:* 10
- Punto de pedido: 2
- Stock de seguridad: 1

5.3.2 Resultados Prácticos

Los parámetros detallados a continuación, son los que se han usado para completar la simulación y optimización:

- *Qo:* 2
- *So:* 1
- *ss:* 1
- *A:* 10
- *v:* 2
- *r:* 0,02

Haremos 500 simulaciones en donde en cada una de ellas tendremos un horizonte de 4 años. Estableceremos como nivel mínimo de servicio, al igual que pasaba con el apartado teórico, de 95%.

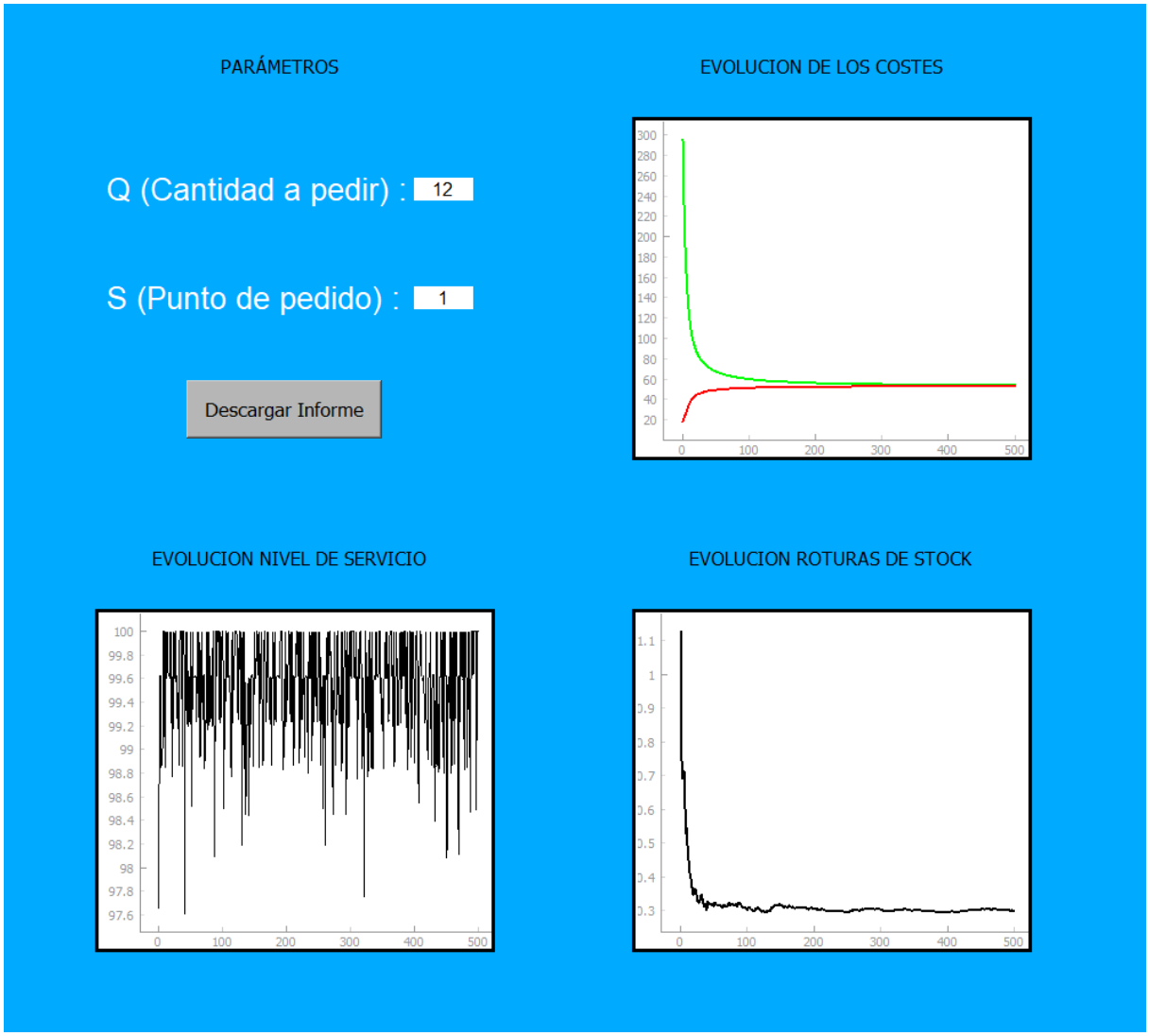


Figura 22: Resultados simulación 3

En esta ilustración, podemos observar los resultados de aplicar los parámetros anteriores en nuestra aplicación. Los datos más relevantes indicados en esta y en el informe generado son:

- *Q:* 12
- Punto de pedido: 1
- Nivel de servicio medio ofrecido: 99,43%

5.3.3 Comparación y Conclusión

En esta ocasión, hemos escogido unos datos históricos cuyas características se corresponden con el de un sistema en el que se gestionan piezas de repuesto. En este ejemplo, se puede observar fácilmente la diferencia entre un modelo especializado en este tipo de piezas y uno que trata productos generales.

Analizando los resultados detallados en el informe, nos damos cuenta de que el nivel de servicio obtenido en nuestro programa, con un punto de pedido establecido en 1, ha sido muy bueno. El modelo teórico, debido a la alta variabilidad del tiempo entre demandas, ajusta el valor de *s* en 2 unidades, siendo esto una optimización no muy eficiente, puesto que tal y como el programa corrobora, con una unidad es más que suficiente para cumplir con el nivel de servicio mínimo establecido antes de comenzar con el proceso.

5.4 Distribución Normal 4

En este cuarto y último ejemplo analizaremos unos datos que cuentan con las siguientes características:

- Media: $\mu_D = \frac{1}{5,2}$ y $\mu_{PA} = 3,4$
- Coeficiente de variación: $cv_D = 0,22$ y $cv_{PA} = 0,65$

El primer valor corresponde con los tiempos entre pedidos y el segundo con los de aprovisionamiento. Las unidades en las que están obtenidos los valores medios son en días.

Los parámetros iniciales que se introducirán en nuestro sistema son:

- *A:* 10
- *v:* 2
- *r:* 0,02

5.4.1 Resultados Teóricos

Usaremos las mismas expresiones, y el mismo porcentaje de ruptura, un 5%. Los resultados obtenidos:

- *Q:* 10
- Punto de pedido: 1

- Stock de seguridad: 1

5.4.2 Resultados Prácticos

Los parámetros detallados a continuación, son los que se han usado para completar la simulación y optimización:

- *Qo:* 2
- *So:* 1
- *ss:* 1
- *A:* 10
- *v:* 2
- *r:* 0,02

Haremos 500 simulaciones en donde en cada una de ellas tendremos un horizonte de 4 años. Estableceremos como nivel mínimo de servicio, al igual que pasaba con el apartado teórico, de 95%.

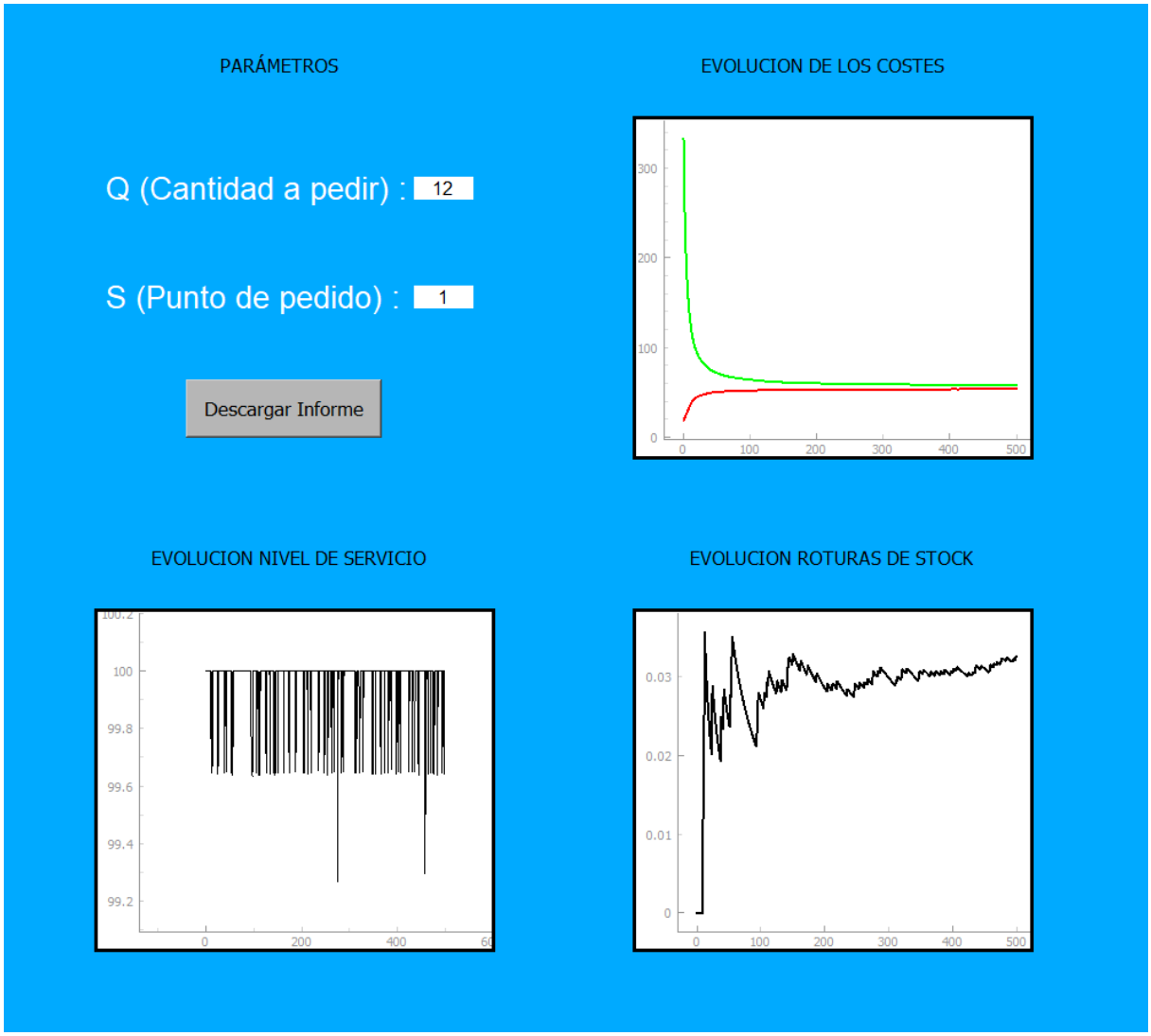


Figura 23: Resultados simulación 4

En esta ilustración, podemos observar los resultados de aplicar los parámetros anteriores en nuestra aplicación. Los datos más relevantes indicados en esta y en el informe generado son:

- *Q:* 12
- Punto de pedido: 1
- Nivel de servicio medio ofrecido: 99,95%

5.4.3 Comparación y Conclusión

En esta ocasión se han testeado unos datos con un plazo de aprovisionamiento que contaba con una variabilidad muy alta. Debido a la reducción, respecto del ejemplo anterior, del valor medio del *PA*, en el modelo teórico se ha obtenido un punto de pedido menor, concretamente de 1 unidad.

Si recordamos lo analizado en los primeros capítulos de la memoria, los tiempos de aprovisionamiento perdían gran importancia en los sistemas donde el inventario está compuesto en su totalidad por piezas de repuesto, cosa que queda de manifiesto observando el nivel de servicio obtenido durante la optimización. En esta, han aparecido contadas rupturas del stock, puesto que nuestro modelo está desarrollado para tratar con estas piezas tan especiales tal y como corroboran los resultados derivados.

6 CONCLUSIÓN

En este capítulo recordaremos los objetivos establecidos al principio del proyecto y analizaremos si hemos cumplido cada uno de ellos. En caso de que alguno no se haya alcanzado totalmente, indicaremos que es lo que nos ha faltado para conseguirlo.

1. El primer objetivo marcado fue el de entender la problemática asociada a los sistemas de gestión del inventario de piezas de repuesto. Después de efectuar una profunda investigación, analizando artículos y estudios realizados en torno a la demanda de las piezas de repuesto, se ha logrado comprender el complejo comportamiento que esta tiene y, por tanto, hemos podido desarrollar el problema, detallándolo completamente en uno de los capítulos de la memoria.
2. El siguiente punto consistía en buscar y desarrollar una metodología para lograr la optimización de los parámetros del sistema. Se han analizado diferentes formas de resolver el problema, descartando aquellas que resultaban inviables. Una vez definida la metodología se ha explicado en qué consistía, detallando como se iba a llevar a cabo la optimización y, destacando a su vez los puntos más relevantes del proceso.
3. Este objetivo ha sido el que más tiempo nos ha llevado completarlo, puesto que consistía en implementar informáticamente la optimización de nuestro sistema, así como la creación de la aplicación que la sustentase. Usando el lenguaje de programación Python, hemos conseguido desarrollar la metodología escogida previamente cumpliendo con todos los aspectos que esta debía tener. En cuanto a la aplicación, puede que nos haya quedado un poco básica, pero al ser una interfaz no profesional, cumple de sobra con su función.
4. Para finalizar con el trabajo debíamos realizar una serie de comparaciones teórico-práctica, para poder comprobar de esta forma que hay algunas situaciones en las que resolver el modelo analíticamente, no es una opción. Hemos completado 4 ejemplos en donde en cada uno de ellos variábamos algunas características de los datos históricos, para estudiar como reaccionaba el modelo desarrollado en distintas situaciones.

El diseño de sistemas para la gestión del inventario de piezas de repuesto es algo que hoy en día se continúa estudiando y analizando para intentar conseguir la mejor optimización posible. En el proyecto realizado, hacemos uso de los datos históricos para predecir los valores futuros, pero esta no es la única forma que podemos utilizar.

La implementación de estos sistemas se puede llevar a cabo de múltiples formas, por ello, se podrían añadir algunos factores que hagan aún más precisa la optimización. Las piezas de repuesto y, sobre todo, la demanda de estas, cuentan con muchas particularidades de las que nos podemos aprovechar, estudiando esas características para añadirla a modelos futuros, partiendo del conseguido en este proyecto.

REFERENCIAS

[1] L. R. Rodrigues y T. Yoneyama, «A spare parts inventory control model based on Prognostics and Health monitoring data under a fill rate constraint,» *Computers & Industrial Engineering,* vol. 148, 2020.

[2] A. Bacchetti y N. Saccani, «Spare parts classification and demand forecasting for stock control: Investigating the gap between research and practice,» *Omega,* vol. 40, nº 6, pp. 722-737, 2012.

[3] Grupo de Ingeniería de Organización de la Universidad de Sevilla, «Introducción a la Gestión de Inventarios,» Sevilla.

[4] E. F. Navarrete Cueva y O. C. Salgado Rodas, «Corrosión y degradación de los metales,» de *Determinación de la corrosividad atmosférica de las ciudades de Santo Domingo y Esmeraldas*, QUITO/ EPN/ 2007, 2007, pp. 1-4.

[5] Á. Arcos Vargas, M. Calle Suárez, P. L. González-R, J. M. León Blanco, M. Á. Muñoz Pérez y F. Núñez Hernández, «El stock bajo incertidumbre,» de *Administración de empresas para los grados de ingeniería*, Vivelibro, 2016, p. 64.

[6] A. M.Law y W. D. Kelton, Simulation modeling and analysis, McGraw-Hill, 1991.

[7] Grupo de Ingeniería de Organización de la Universidad de Sevilla, «Gestión de Inventarios, El stock de seguridad,» Sevilla.

[8] Grupo de Ingeniería de Organización de la Universidad de Sevilla, «Gestión de Inventarios, El lote económico».

[9] A. Mjirda, R. Benmansour, H. Allaoui y G. Goncalves, «On the joint optimization of the periodic preventive maintenance and the spare parts inventory problem,» *IFAC-PapersOnLine,* vol. 49, nº 12, pp. 881-885, 2016.

[10] W. v. Jaarsveld, T. Dollevoet y R. Dekker, «Improving spare parts inventory control at a repair shop,» *Omega,* vol. 57, nº B, pp. 217-229, 2015.

[11] S. V. d. Auweraer, R. N.Boute y A. A.Syntetos, «Forecasting spare part demand with installed base information: A review,» *International Journal of Forecasting,* vol. 35, nº 1, pp. 181-196, 2019.

ANEXO I

Código Desarrollado para la Ejecución de la Optimización

```
import random
import simpy
import statistics
import numpy as np
import scipy.stats
from scipy.stats import variation
import math
from math import pi
import contrastep
```

Para que la simulación y optimización realizada en este código, pudiera ser usada desde la aplicación, había que convertirla en función, y eso es lo que se puede observar en las siguientes líneas.

```
def
funcion(var_q,var_s,var_ss,var_a,var_v,var_r,var_ns,var_repl,var_sim
,datitos_dt,datitos_lt,var_I):
```

A continuación, crearemos la clase "SM", la cual contendrá prácticamente todo el proceso de simulación.

Comenzaremos, como no podría ser de otra forma, definiendo el método inicial, donde lo único que haremos será crear los atributos necesarios para llevar a cabo el proceso. Al lado de cada variable, añadiremos una pequeña descripción de cada una de ellas, para ayudar a conocer cuáles serán sus funciones en el programa.

```
    class SM:
      def __init__(self, dist_dt, dist_lt, env, I, Q, s, SS, a_dt,
      a_lt, cv_dt, cv_lt, A, v, r):

          self.dist_dt = dist_dt #Distribución asociada a la demanda
          self.dist_lt = dist_lt #Distribución asociada a los PA
          self.env = env #El entorno de la simulación
          self.LO=False #Señal para el lanzamiento de una orden
          self.txt_log="" #Informe
          self.I=I #Nivel de inventario
          self.Q=Q #Tamaño del lote pedido
```

```
        self.s=s #Punto de pedido
        self.SS=SS #Stock de seguridad
        self.a_dt=a_dt #Lista con los datos de la demanda
        self.a_lt=a_lt #Lista con los datos del PA
        self.cv_dt=cv_dt #Coeficiente de variación de la demanda
        self.cv_lt=cv_lt #Coeficiente de variación del PA
        self.A=A #Coste de pedir
        self.v=v #Coste unitario del producto
        self.r=r #Tasa aplicada a mantener el inventario
        self.pedidos=0 #Para calcular el número total de pedidos
        self.usos_ss=0 #Para calcular los usos del SS
        self.clanz=0 #Para calcular los costes de lanzamiento
        self.cmtto=0 #Para calcular los costes de mantenimiento

        #Para recoger más información
        self.num_st_out_t=[0] #Instantes de rotura de stock
        self.Inv_level_t=[0] #Instantes donde varía el inventario
        self.num_st_out_SS_t=[0] #Instantes donde se usa el SS
        self.num_st_out=[0] #Contador de roturas de stock
        self.num_st_out_SS=[0] #Contador de usos del SS
        self.Inv_level=[I] #Niveles de inventario en el proceso
        self.action=env.process(self.RUN('demand'))
```

Una vez descritas todas las variables que intervendrán en el proceso, procederemos a analizar cómo y cuándo se han usado cada una de ellas.

Empezaremos con el segundo método con el que nos encontraremos dentro de la clase "SM". Para poder ir avanzando en el tiempo en la simulación, vamos a necesitar predecir cada cuanto tiempo nos van a pedir, y en caso de que seamos nosotros los que necesitemos realizar la orden, en cuanto tiempo nos llegarán las provisiones al almacén. Esta es la función que realiza el método "contraste".

Una vez identificada la distribución que mejor se ajusta a los datos, lo cual veremos más adelante, nos encargaremos de pasar dicha información como argumento de entrada, junto con los datos históricos y el coeficiente de variación. Dentro del método, generaremos unos valores aleatorios, diferenciando primero, según la distribución introducida, para luego a partir de los datos antiguos obtener una predicción de los futuros.

```
    def contraste(self,dist,datos,cv):

        if dist=="norm":
            n=random.gauss(np.mean(datos),np.mean(datos)*cv)
            while n<0:
                n=random.gauss(np.mean(datos),np.mean(datos)*cv)
            res=n

        if dist=="expon":
            e=random.expovariate(1/np.mean(datos))
            while e<0:
                e=random.expovariate(1/np.mean(datos))
            res=e
```

```
if dist=="gamma":
    g=np.random.gamma(pow(np.mean(datos),2)/pow(np.mean(da
    tos)*cv,2),pow(np.mean(datos)*cv,2)/np.mean(datos))
    while g<0:
        g=np.random.gamma(pow(np.mean(datos),2)/pow(np.mea
        n(datos)*cv,2),pow(np.mean(datos)*cv,2)/np.mean(da
        tos))
    res=g

if dist=="logistic":
    l=np.random.logistic(np.mean(datos),math.sqrt((3*pow(n
    p.mean(datos)*cv,2))/pi**2))
    while l<0:
        l=np.random.logistic(np.mean(datos),math.sqrt((3*p
        ow(np.mean(datos)*cv,2))/pi**2))
    res=l

if dist=="lognorm":
    ln=np.random.lognormal(np.mean(np.log(datos)),np.mean(
    np.log(datos))*cv)
    while ln<0:
        ln=np.random.lognormal(np.mean(np.log(datos)),np.m
        ean(np.log(datos))*cv)
    res=ln

if dist=="triang":
    t=random.triangular(min(datos),max(datos),(min(datos)+
    max(datos))/2)
    while t<0:
        t=random.triangular(min(datos),max(datos),(min(dat
        os)+max(datos))/2)
    res=t

if dist=="uniform":
    u=random.uniform(min(datos),max(datos))
    while u<0:
        u=random.uniform(min(datos),max(datos))
    res=u

return res
```

En este punto, nos encontramos con una de las partes más importante de la simulación. Para analizarlo con más detenimiento, se hará como se hizo con las variables iniciales, realizaremos una breve descripción al lado de cada sentencia a lo largo del código.

Antes de entrar a detallar cada una de las líneas, cabe mencionar que, en esta parte es donde se va a llevar cabo el avance en el tiempo dentro del horizonte establecido, para comprobar si con los parámetros indicados en ese momento se produce rotura de stock o no.

```
def RUN(self, name):
    aux=0
     while True:
        demand_interval=self.contraste(self.dist_dt,self.a_d
        t,self.cv_dt) #Llamamos al método "contraste" para
        que nos devuelva el tiempo predicho hasta el próximo
        pedido
        PA=self.contraste(self.dist_lt,self.a_lt,self.cv_lt)
        #Llamamos al método "contraste" para que nos devuelva
        el plazo de aprovisionamiento predicho
        if name=="demand": #Si existe pedido
            self.I=self.I-1 #Reducimos el inventario
            self.pedidos+=1 #Aumentamos números de pedido

            if self.env.now!=0:
                self.cmtto+=((self.Inv_level[-
                1]/2)*self.v*self.r)*(self.env.now-aux) #Para
                calcular el coste de mantenimiento según el
                nivel de inventario actual
            aux=self.env.now

            self.txt_log+= str(self.env.now) + " " +
            str(name) + " I:" + str(self.I) + "\n"
            #Recolección de información en cadena de texto
            self.Inv_level+=[self.I] #Creamos lista según el
            nivel de inventario, nos será útil para la
            representación gráfica
            self.Inv_level_t+=[self.env.now] #Igual que
            arriba pero anotando los instantes en donde el
            inventario varía
            if self.I<self.SS: #Entramos en el uso del SS
                self.num_st_out_SS+=[self.num_st_out_SS[-
                1]+1] #Guardamos en lista las veces que se
                hace uso del SS
                self.usos_ss+=1
                self.num_st_out_SS_t+=[self.env.now]
                #Guardamos instantes donde hacemos uso del SS
            if self.I<0: #Situación de rotura de stock
                self.txt_log+=str(self.env.now) + "se ha
                producido una ruptura\n" #Recolección de
                información en cadena de texto
                self.num_st_out+=[self.num_st_out[-1]+1]
                #Guardamos en lista las veces que ocurre una
                ruptura de stock
                self.num_st_out_t+=[self.env.now] #Guardamos
                los instantes en los cuales se produce esa
                ruptura
            elif self.I<=self.s and self.LO==False: #Si se
            alcanza el punto de pedido, activamos la orden
            de lanzamiento
                 self.LO=True
                 self.env.process(self.RUN('launch_order'))
            yield self.env.timeout(demand_interval) #Vamos
            avanzando en el tiempo según el tiempo entre
            pedidos
```

```
            elif name=="launch_order": #Lanzamiento de un pedido
                 self.txt_log+='%2d: launched_order I %2d %g\n' %
                 (self.env.now, self.I, self.LO)
                 yield self.env.timeout(PA) #Ahora tendremos en
                 cuenta el plazo de aprovisionamiento
                 self.LO=False #Cuando llega el pedido ya no hace
                 falta pedir más
                 self.txt_log+='%2d: received_order %g\n' %
                 (self.env.now, self.LO)
                 if self.I<0:
                    self.I=0
                 self.I=self.Q+self.I #Cuando se recibe el
                 pedido, el inventario disponible es igual a lo
                 que hemos pedido al proveedor más lo que
                 teníamos

                 self.clanz+=self.A #Cálculo del coste de
                 lanzamiento del pedido

                 self.Inv_level+=[self.I] #Iremos actualizando de
                 nuevo el nivel de inventario
                 self.Inv_level_t+=[self.env.now] #Haremos
                 exactamente lo mismo con los instantes donde el
                 inventario varía
                 break
```

A continuación, declararemos algunas variables, la mayoría de ellas auxiliares, que nos harán falta más adelante.

```
a_st=[]
sd=[]
cv=[]
list_mean_a_st=[]
a_SS=[]
list_mean_a_SS=[]
data_dt = list()
data_lt = list()

eyex = list()
eyeylanz = list()
eyeymtto = list()
meanlanz = list()
meanmtto = list()
levelservise = list()
levelservise_mean = list()
```

En las próximas líneas estableceremos los parámetros de nuestra simulación. El número de réplicas, el horizonte de simulación y los datos históricos, han sido introducidos previamente por el usuario en la aplicación, y ahora es cuando los interpretaremos en el código. También crearemos un listado con las posibles distribuciones estadísticas a ajustar.

```
num_repl=var_repl
warm=50
sim_horizon=var_sim*365

i=0
j=0
data_dt=datitos_dt
data_lt=datitos_lt
```

Una vez que el usuario ha introducido los tiempos históricos, el programa se encarga se ver cuál es la distribución que mejor se ajusta a esos datos, y eso es lo que veremos a continuación.

Llamaremos a la función *"contrastep"*, para que, metiendo como argumentos de entrada los tiempos históricos, nos devuelva, a parte de la distribución obtenida, los valores medios y los coeficientes de variación de cada uno de los datos introducidos.

```
nombre_buen_dt,nombre_buen_lt,dist_dt,dist_lt,media_dedt,media_d
elt,cv_1,cv_2=contrastep.funcion_2(data_dt,data_lt)
```

Antes de entrar en el bucle donde iniciaremos la simulación, declararemos algunas variables, entre las que destacamos "valorq" y "valors". Estas adoptarán los valores iniciales del lote encargado a nuestro proveedor y del punto de pedido, los cuales serán previamente indicados por el usuario en la interfaz gráfica. Se añadirá al lado de cada variable una breve descripción con la función de cada una de ellas en el código.

```
refini=0 #Para calcular las roturas de stock en el proceso
resfiniped=0 #Para calcular el número total de pedidos

totalroturas=0 #Para calcular el número total de roturas de stock
totalusos_ss=0 #Para calcular las veces que se usó el SS
total=0 #Para calcular el nivel de servicio en cada simulación
valorq=var_q #Cantidad a pedir inicial
valors=var_s #Punto de pedido inicialmente establecido
```

En la siguiente parte de nuestro programa, inicializaremos la simulación y a su vez iremos optimizando los parámetros del sistema.

Una vez que entramos en el bucle, nos encontramos con una condición, la cual nos indica cuando se saldrá de este. En caso de que superemos el número de réplicas indicado por el usuario en la aplicación, el programa saldrá del bucle, es decir, realizaremos tantas simulaciones como se hayan indicado previamente.

Si seguimos avanzando veremos la llamada a la clase "SM". La información que se pasará como argumento de entrada es la siguiente:

- Tipo de distribución que mejor se ajusta a los datos históricos
- Cantidad del lote ordenado a nuestro proveedor
- Punto de pedido
- Stock de Seguridad establecido
- Tiempos históricos
- Coeficiente de variación
- Coste de pedir
- Coste unitario del producto
- Tasa aplicada a mantener el inventario

Antes de continuar nos gustaría destacar la última línea de este trozo de código. En él, se establece el horizonte temporal hasta el que va a poder avanzar la simulación. Como ya se ha comentado en varias ocasiones, este parámetro deberá estar concretado previamente por parte del usuario.

```
while True:
    if i>=num_repl: break
    i+=1
    env = simpy.Environment()
    sm=SM(dist_dt, dist_lt, env,
    I=var_I,Q=valorq,s=valors,SS=var_ss,a_dt=data_dt,a_lt=data_lt
    , cv_dt=cv_1, cv_lt=cv_2, A=var_a, v=var_v, r=var_r)
    env.run(until=sim_horizon)
```

Las líneas siguientes, servirán para la toma de datos para la posterior representación gráfica. Es importante conocer la evolución de los parámetros para cerciorarnos de que lo que ha ocurrido en la optimización ha estado siguiendo un buen curso.

Debemos de mencionar, que, en algunas gráficas, se ha optado por ir tomando la media en cada uno de los instantes del proceso. Esto se ha establecido así, debido a la diversidad de valores en cada una de las simulaciones, por lo que, de esta manera, se podrá observar fácilmente la tendencia de cada uno de los parámetros representados.

```
    eyex+=[i]
    lanz_anu=365* sm.clanz/sim_horizon #Obtención datos anuales
    eyeylanz+=[lanz_anu]
    meanlanz+=[statistics.mean(eyeylanz)]
    mtto_anu=365* sm.cmtto/sim_horizon #Obtención datos anuales
    eyeymtto+=[mtto_anu]
    meanmtto+=[statistics.mean(eyeymtto)]
```

Llegamos a un punto clave en la simulación, la optimización. En las próximas líneas de código, iremos ajustando algunos parámetros en función de los resultados obtenidos en la simulación.

En primer lugar, nos encontramos con el ajuste de la variable Q. Como se comentó al principio de la memoria, el punto óptimo en cuanto a los costes de lanzamiento y de mantenimiento, es cuando

ambos son iguales. Por ello usaremos la condición *"if"*, de manera que, según los valores de ambos costes, vayamos variando la Q, para obtener ese punto deseado.

Debemos tener en cuenta que realizaremos muchas simulaciones, y habrá casos excepcionales en cuanto a los costes, por ello, estableceremos un margen de 1.1 para que no exista ningún tipo de problema.

```
if sm.clanz>=(1.1*sm.cmtto):
    valorq+=1
if (1.1*sm.clanz)<sm.cmtto:
    valorq=valorq-1
```

Una vez optimizada la variable Q, deberemos de hacer lo propio con el nivel de servicio.

Las dos primeras líneas son para la obtención de resultados finales relativos a la optimización.

De la simulación, obtendremos unos valores los cuales son, el número de roturas de stock existentes en el periodo establecido y el número total de pedidos realizados en este. Gracias a estos datos, y una vez multiplicado por 100, obtendremos un valor que nos indicará cuantos productos no somos capaces de entregar cada 100 pedidos. Por lo tanto, el porcentaje restante nos indicará la calidad del servicio que estamos ofreciendo en ese momento.

Además del cálculo del nivel de servicio, tendremos otras líneas de código en las que en una iremos almacenado los datos relativos a dicho cálculo para poder ver posteriormente en un gráfico su evolución a lo largo del proceso y en la otra contaremos el número total de veces que hemos usado el stock de seguridad.

```
resfini=resfini+sm.num_st_out[-1]
resfiniped=resfiniped+sm.pedidos

total=100-(100*(sm.num_st_out[-1]/sm.pedidos))
levelservise+=[total]
levelservise_mean+=[statistics.mean(levelservise)]
totalusos_ss+=sm.usos_ss
```

El nivel de servicio mínimo será uno de los parámetros que el usuario deberá establecer antes de comenzar con la optimización. Por ello, en caso de que los valores obtenidos bajen de ese nivel establecido, pondremos un punto de pedido superior, de esta forma lo que conseguiremos es anticiparnos de una manera más holgada a cualquier demanda.

Como se ha comentado en más de una ocasión, el plazo de aprovisionamiento es relativamente pequeño si lo comparamos con el tiempo entre demandas, por ello, el nivel de servicio calculado será considerablemente alto durante todo el proceso.

```
if valors<valorq:
    if i>10:
```

```
            if total<=var_ns:
                valors+=1
```

A continuación, seguiremos recopilando información acerca de lo que sucede a lo largo del proceso de simulación. En el caso de que exista alguna rotura de stock, se calculará el valor promedio de estas anualmente. Para ello necesitaremos las veces que no hemos sido capaces de abastecer la demanda en el periodo establecido por el usuario al principio de la simulación.

También se contemplará el caso de que durante el horizonte de simulación concretado no se dé ninguna rotura de stock.

Debemos de mencionar que, en todo el proyecto, el número de días que tiene un año se ha establecido en 365.

```
if sm.num_st_out_t[-1]:
    totalroturas+=sm.num_st_out[-1]
    a_st_i=365* sm.num_st_out[-1]/sim_horizon

else:
    a_st_i=0

a_st+= [a_st_i]
```

Después de haber recopilado la información necesaria en cuanto a lo que roturas de stock respecta, toca hacer lo propio con las veces que se usará el stock de seguridad. Se realizará de la misma forma, primero comprobando si se ha hecho uso de este, para posteriormente, en caso de que así sea, realizar los cálculos pertinentes.

```
if sm.num_st_out_SS_t[-1]:
    a_SS_i=365* sm.num_st_out_SS[-1]/sim_horizon

else:
    a_SS_i=0

a_SS+= [a_SS_i]
```

En las próximas líneas continuaremos compilando datos de la simulación. Cabe destacar el motivo por el que se ha establecido que para entrar dentro de la instrucción “if”, se deberá de llevar 2 simulaciones ya completas. Esto es así, debido a que algunas de las funciones usadas en el interior de dicha instrucción, requieren de al menos 2 números para completar sus cálculos correspondientes.

```
if i>=2:
    mean_a_st=statistics.mean(a_st)
    list_mean_a_st+=[mean_a_st]
    mean_a_SS=statistics.mean(a_SS)
    list_mean_a_SS+=[statistics.mean(a_SS)]

    sd_i=statistics.stdev(a_st)
    sd+=[sd_i]
```

```
total_2=100-(100*(resfini/resfiniped))
print("\n La Q optimizada es: ",valorq)
print("\n La s optimizada es: ",valors)

list_mean_a_st+=[list_mean_a_st[-1]]
```

Llegamos al final del código de la simulación. Cabe recordar que para que los datos puedan ser usados en la aplicación, hemos tratado este programa como una función. Por lo tanto, en este punto toca devolver a nuestra interfaz gráfica los datos más relevantes, estos son:

- Valores optimizados tanto del punto de pedido como de Q
- El número de simulaciones que ha habido en la optimización
- Una lista con la información de la proyección del coste de lanzamiento
- Otra lista con la información de la proyección del coste de mantenimiento
- Lista con los datos de la evolución de las roturas de stock
- Lista con los datos sobre las veces que se ha hecho uso del nivel de servicio
- Información relativa al contraste de hipótesis completado

```
return
valorq,valors,sm.Inv_level_t,sm.Inv_level,eyex,meanlanz,meanmtto
,list_mean_a_st,levelservise,totalroturas,totalusos_ss,nombre_bu
en_dt,nombre_buen_lt,media_dedt,media_delt,cv_1,cv_2,total_2
```

ANEXO II

Código Desarrollado para Dotar de Funcionalidad a la Aplicación

```
import sys
from PyQt5.QtWidgets import  QApplication, QMainWindow, QMessageBox,
QPushButton, QDialog, QProgressDialog, QFileDialog
from PyQt5 import uic
import copia
import contrastep
import pyqtgraph as pg
import pandas as pd
import io
from docx import Document
from docx.shared import Inches
from docx.enum.text import WD_ALIGN_PARAGRAPH
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D
```

Empezaremos definiendo la clase "Ventana", que será nuestra pestaña principal. En el método inicial, cargaremos la interfaz gráfica diseñada en el QT Designer cuyo nombre es "app.ui". Más adelante nos encontramos con 3 líneas de código cuyo objetivo es que cuando se pulse un determinado botón, se dirija al método indicado entre paréntesis, para ver la funcionalidad asignada a cada uno de ellos. Para que quede más claro, la sintaxis sería de la siguiente forma: *self."nombre_del_boton".clicked.connect(self."nombre_metodo").*

Las siguientes líneas simplemente sirven para mantener fijo el tamaño de nuestra ventana, para que esta no se pueda agrandar ni achicar. También daremos valor a una variable, la cual será una auxiliar. Veremos muchas más de este tipo a lo largo de todo el código.

```
class Ventana(QMainWindow):

 def __init__(self):
  QMainWindow.__init__(self)
  uic.loadUi("app.ui", self)


  self.pushButton_2.clicked.connect(self.exam1)
  self.pushButton.clicked.connect(self.loginfunction)
  self.pushButton_3.clicked.connect(self.distribucion)
  self.pushButton.clicked.connect(self.abrirDialogo)
```

```
self.setMinimumSize(800,600)
self.setMaximumSize(800,600)

Ventana.datos_si=0
```

A continuación, se definirá un nuevo método, este simplemente se encargará de hacer aparecer un cuadro de aviso cuando se trate de cerrar la ventana principal. En él, se nos preguntará si queremos realmente salir de la aplicación, dándonos la opción de confirmar la acción o, por el contrario, retractarnos.

```
def closeEvent(self, event):
 resultado = QMessageBox.question(self, "Salir ...", "¿Seguro que
 quieres salir de la aplicación?", QMessageBox.Yes |
 QMessageBox.No)
 if resultado == QMessageBox.Yes: event.accept()
 else: event.ignore()
```

Antes de pulsar el botón START, habremos establecido todas las variables iniciales necesarias para llevar a cabo la simulación. Cuando lo pulsemos, uno de los métodos a los que se dirigirá, tal y como se vio al principio del código, es al de *"loginfunction"*. Lo primero con lo que nos encontraremos, será con una serie de líneas que cumplen simplemente con la función de guardar en variables, los datos introducidos por el usuario en las casillas de la aplicación. Como se puede observar algunas variables las trataremos como números enteros, y otros debemos de darle la opción de tener decimales.

Si continuamos avanzando, nos encontraremos con la llamada a nuestro código principal, es decir, el de la simulación. Se le meterán como argumentos de entrada, las variables iniciales indicadas por el usuario para que realice la optimización y nos devuelva a la aplicación los resultados obtenidos en esta.

```
def loginfunction(self):

 Ventana.q_value=int(self.spinBox.text())
 Ventana.s_value=int(self.spinBox_2.text())
 Ventana.ss_value=int(self.spinBox_3.text())
 Ventana.I_value=int(self.spinBox_7.text())
 Ventana.num_repl=int(self.spinBox_6.text())
 Ventana.hor_sim=int(self.spinBox_5.text())
 Ventana.A_value=self.doubleSpinBox_3.text()
 Ventana.A_value = float(Ventana.A_value.replace(",", "."))
 Ventana.V_value=self.doubleSpinBox_2.text()
 Ventana.V_value =float(Ventana.V_value.replace(",", "."))
 Ventana.R_value=self.doubleSpinBox.text()
 Ventana.R_value =float(Ventana.R_value.replace(",", "."))
 Ventana.NS_value=int(self.spinBox_4.text())


 Ventana.resq,Ventana.ress,Ventana.x,Ventana.y,Ventana.dosx,Ventana
 .lanzy,Ventana.mttoy,Ventana.meanstockout,Ventana.servisio,Ventana
```

```
.rots,Ventana.ususs,Ventana.resu_dt,Ventana.resu_lt,Ventana.med_dt
,Ventana.med_lt,Ventana.coe_dt,Ventana.coe_lt,Ventana.total_22=cop
ia.funcion(Ventana.q_value,Ventana.s_value,Ventana.ss_value,Ventan
a.A_value,Ventana.V_value,Ventana.R_value,Ventana.NS_value,Ventana
.num_repl,Ventana.hor_sim,Ventana.datito_dt,Ventana.datito_lt,Vent
ana.I_value)

self.dialogo = Dialogo()
```

Llegados a este punto, en *"abrirDialogo"*, lo que haremos será llamar a la clase "Dialogo", que como se verá más adelante, abrirá una nueva ventana en donde mostraremos todos los resultados obtenidos durante la optimización.

```
def abrirDialogo(self):
 self.dialogo.exec_()
```

En "exam1", al cual llegaremos pulsando el botón de carga de archivos que vimos cuando mostramos la aplicación, se nos redirigirá a *"open_dialog_box"*, donde desde el explorador de archivos abierto una vez pulsado el botón, podremos obtener la dirección en la que se encuentra el fichero Excel en el cual tenemos guardado todos los tiempos históricos relativos a nuestro problema. Almacenaremos la dirección en la variable "*path*".

```
def exam1(self):
 self.open_dialog_box()

def open_dialog_box(self):
 Ventana.datito_dt=list()
 Ventana.datito_lt=list()
 filename = QFileDialog.getOpenFileName()
 path = filename[0]
 print(path)
```

Una vez obtenida la ruta de nuestro archivo, obtendremos el contenido de este gracias a una función de la librería pandas. Luego lo que haremos, será guardar los tiempos de cada columna en una lista individualizada para poder incluirlos como argumentos de entrada cuando se llame a la función que realiza la optimización. En el Excel, cuando el usuario quiera introducir los datos históricos relativos a su compañía, deberá hacerlo por columnas, existiendo dos de ellas ya preestablecidas en el fichero, una tiene el nombre de "Demanda", la cual corresponde a los tiempos entre pedidos, y la otra se ha denominado "PA", que como su propio nombre indica, hace referencia al plazo de aprovisionamiento.

```
df = pd.read_excel(path)
Ventana.datito_dt = df['Demanda'].tolist
Ventana.datito_lt = df['PA'].tolist()
```

Las siguientes líneas de código, corresponden a arreglos visuales en la aplicación, es decir, cuando se cargue un archivo, saldrá un mensaje en la aplicación afirmando que estos se han

aceptado correctamente. Además, el texto adherido al botón cambiará, preguntando ahora qué si queremos cargar unos nuevos datos, sustituyéndolo por los que ya habíamos añadido.

También, tendremos un cambio de valor de la variable auxiliar previamente definida. En el caso de que esta valga 1, significará que se han introducido los datos en nuestra aplicación.

```
Ventana.datos_si=1

self.pushButton_2.setStyleSheet("background-color: rgb(182,
182, 182);""font: 75 10pt ")
self.pushButton_2.setText('Cargar otros datos...')
self.label_11.setText('Datos cargados correctamente')
```

Este método se activará en caso de que se pulse el botón establecido para conocer, previamente a la simulación, la distribución a la que mejor se ajustan nuestros datos. Gracias a la variable auxiliar definida bajo el nombre de *"Ventana.datos_si"*, podremos advertir mediante un mensaje de error, que no es posible conocer dicha información, sin previamente haber introducido los tiempos.

En caso de que estos si estén ya en nuestra aplicación, llamaremos a la clase *"diag_contraste"*, para mostrar por pantalla la información de la que ya hemos hablado.

```
def distribucion(self):
 if Ventana.datos_si==0:
    QMessageBox.critical(self, "Error", "Introduzca primero los
    datos que quiere analizar", QMessageBox.Ok)
 else:
     self.cont = diag_contraste()
     self.cont.exec_()
```

En esta clase tan sólo contaremos con un método, el inicial. En él estableceremos el título de nuestra ventana, además de cargar la interfaz gráfica diseñada previamente.

Por otra parte, se llamará a la función que contiene el código donde se obtendrán previamente el nombre de las 2 distribuciones, una para la demanda y la otra para el *PA*.

```
class diag_contraste(QDialog):

 def __init__(self):

  QDialog.__init__(self)
  self.setWindowTitle("Distribuciones")
  uic.loadUi("app3.ui",self)

  dist_res_dt,dist_res_lt,auxiliinut,auxilinut_2,medi_dt,medi_lt,coe
  f_dt,coef_lt=contrastep.funcion_2(Ventana.datito_dt,Ventana.datito
  _lt)

  self.label_3.setText(dist_res_dt)
```

```
self.label_4.setText(dist_res_lt)
```

Si subimos un poco más arriba en el código, veremos que, una vez terminada la optimización, se llama a la clase *"Dialogo"*. En el método inicial de esta, abriremos la segunda ventana creada en el QT Designer, que responde al nombre de *"app2.ui"*, para poder mostrar los datos obtenidos en la simulación. Como se puede observar, *"Resultados"* será el título que lleve la nueva pestaña. También, en este primer método, se llamará a las funciones restantes de la clase, cuyo contenido se analizará más adelante.

Por último, se ha de comentar que se ha dotado de funcionalidad al botón disponible para generar el informe, al clicar sobre este, el código de dirigirá al método *"informe"*, donde se detallarán todas las características de este.

```
class Dialogo(QDialog):

 def __init__(self):

  QDialog.__init__(self)
  self.setWindowTitle("Resultados")
  uic.loadUi("app2.ui",self)
  self.loaddata()
  self.plot()
  self.generar.clicked.connect(self.informe)
```

En el primer método con el que nos encontramos, *"loaddata"*, lo que se haremos es conseguir mostrar en la aplicación los datos numéricos obtenidos en la optimización. Previamente, tal y como vimos en el desglose de la interfaz de nuestra aplicación, deberemos de tener creadas una especie de etiquetas, las cuales editaremos a través de la siguiente función: *self."nombre_etiqueta".setText("valor_a_imprimir")*.

```
 def loaddata(self):

     #IMPRIMIR VALORES EN RESULTADOS

  variq=Ventana.resq
  varis=Ventana.ress
  ts_value=str(varis)
  tq_value = str(variq)
  self.lineEdit.setText(tq_value)
  self.lineEdit_2.setText(ts_value)
```

En las siguientes líneas de código, veremos cómo crear y mostrar en nuestra aplicación las gráficas, las cuales nos ayudarán a comprender mejor el funcionamiento y la evolución que ha tenido nuestro modelo durante la optimización.

Podemos observar cómo se han editado algunas de las características de las gráficas, entre estas podemos ver, el cambio de color del fondo, el cambio de color de las líneas del trazado, y el

ancho de estas últimas. En definitiva, cambios estéticos realizados en la interfaz.

```
def plot(self):

    #MOSTRAR GRAFICA EN RESULTADOS

 self.PlotWidget.setBackground('w')
 self.PlotWidget_2.setBackground('w')
 self.PlotWidget_3.setBackground('w')

 self.PlotWidget.plot(Ventana.dosx,Ventana.lanzy, pen=pg.mkPen('g',
 width=2))
 self.PlotWidget.plot(Ventana.dosx,Ventana.mttoy, pen=pg.mkPen('r',
 width=2))


 self.PlotWidget_2.plot(Ventana.dosx,Ventana.meanstockout,
 pen=pg.mkPen('k', width=2))


 self.PlotWidget_3.plot(Ventana.dosx,Ventana.servisio,
 pen=pg.mkPen('k', width=1))
```

A continuación, analizaremos el método en el cual se genera el informe con los resultados. En primer lugar, nos encontramos con sentencias cuya única función van a ser añadir texto, diferenciando entre títulos y párrafos más desarrollados.

```
def informe(self):

 self.document=Document()
 self.document.add_heading("INFORME OPTIMIZACIÓN",0)
 self.document.add_paragraph("A continuación, se mostrará un
 resumen del proceso de simulación y optimización, donde se
 incluirán todas las variables implicadas en el proceso.")
 self.document.add_heading('Valores Iniciales', level=1)
 paragraph_2=self.document.add_paragraph('')
 paragraph_2.paragraph_format.space_before=Inches(0.05)
 self.document.add_paragraph("Para que se puedan comprobar que los
 parámetros introducidos por el usuario son los correctos, se
 adjunta una tabla con cada uno de estos valores y una breve
 explicación de la función de cada uno de ellos.")
```

Generaremos una lista bajo el nombre de *"iniciales"*, donde guardaremos gran parte de la información que se pretende introducir en la simulación. También, se añadirá una pequeña descripción sobre cada una de las variables descritas.

```
 iniciales = (
   ('Q', Ventana.q_value, 'Cantidad a pedir inicial'),
   ('s', Ventana.s_value, 'Punto de pedido inicial'),
```

```
('ss', Ventana.ss_value,'Stock de seguridad'),
('Num_Repl', Ventana.num_repl, 'Número de réplicas'),
('Hor_Sim', Ventana.hor_sim, 'Horizonte de simulación'),
('A', Ventana.A_value, 'Coste de pedir'),
('v', Ventana.V_value, 'Coste unitario del producto'),
('r', Ventana.R_value, 'Tasa aplicada a mantener el
inventario'),
('NS', Ventana.NS_value, 'Nivel de servicio mínimo'),
('Io', Ventana.I_value, 'Inventario Inicial'))
```

Para imprimir en el documento la lista generada previamente, deberemos crear una tabla. Esto es lo que se llevará a cabo en las siguientes líneas, definiendo el estilo de estas y los títulos asociados a cada una de las columnas.

```
table = self.document.add_table(rows=1, cols=3)
table.style='Colorful List Accent 6'
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'Item'
hdr_cells[1].text = 'Valor'
hdr_cells[2].text = 'Desc'
```

Una vez creada la tabla, necesitaremos rellenarla. Para ello, realizaremos un bucle para ir fila por fila imprimiendo en ella toda la información recabada en la lista.

```
for it, valore, desc in iniciales:
 row_cells = table.add_row().cells
 row_cells[0].text = it
 row_cells[1].text = str(valore)
 row_cells[2].text = desc
```

En las siguientes líneas seguiremos añadiendo párrafos que faciliten la comprensión de los puntos expuestos en el informe.

```
paragraph=self.document.add_paragraph('')
paragraph.paragraph_format.space_before=Inches(0.05)
self.document.add_heading('Resultados Finales', level=1)
paragraph_3=self.document.add_paragraph('')
paragraph_3.paragraph_format.space_before=Inches(0.05)
self.document.add_paragraph("A continuación, podremos ver el
resultado de la optimización. Todos estos valores han sido
obtenidos a partir de los valores iniciales introducidos
previamente.")
```

Lo que veremos a continuación será exactamente lo mismo que vimos previamente. La única diferencia será que ahora mostraremos los resultados finales una vez completada la simulación. El procedimiento será el mismo, es decir, en primer lugar, se formará la lista con los valores

obtenidos, luego crearemos la tabla y, finalmente la rellenaremos con el contenido recabado en ella.

```
ns_medio_ajust = round(Ventana.total_22, 2)
finales = (
 ('Q', Ventana.resq, 'Cantidad a pedir óptima'),
 ('s', Ventana.ress, 'Punto de pedido óptimo'),
 ('Rot_Tot',Ventana.rots,'Roturas de stock en la simulación'),
 ('Uso_SS',Ventana.usoss,'Veces que se ha usado el SS durante la
 simulación'),
 ('NS_medio',ns_medio_ajust,'Valor medio del nivel de servicio
 obtenido durante la simulación'))


table_2 = self.document.add_table(rows=1, cols=3)
table_2.style='Colorful List Accent 6'
hdr_cells_2 = table_2.rows[0].cells
hdr_cells_2[0].text = 'Item'
hdr_cells_2[1].text = 'Valor'
hdr_cells_2[2].text = 'Desc'

for it, valore, desc in finales:
 row_cells_2 = table_2.add_row().cells
 row_cells_2[0].text = it
 row_cells_2[1].text = str(valore)
 row_cells_2[2].text = desc
```

En este parte dentro del método lo único que se consigue es poner la fecha en la que se ha generado el informe como pie de página del documento. Para ello, tal y como explicamos al principio del apartado, usaremos la librería *"Datetime"*.

```
now=datetime.now()
format = now.strftime('Fecha: %d/%m/%Y, Hora: %H:%M')
footer_section = self.document.sections[0]
footer = footer_section.footer
footer_text = footer.paragraphs[0]
footer_text.text = format
```

En la siguiente parte del código desarrollado, crearemos e introduciremos en el informe una serie de gráficas. Para ello, crearemos un buffer de memoria para guardar las gráficas. Los pasos a seguir para la creación de cada una de ellas serían:

1. Crear el espacio para poder guardar la gráfica
2. Definir y editar las características de esta
3. Guardar la imagen de la gráfica en la memoria
4. Añadir al documento y cerrar la memoria creada

Primera imagen:

```
self.document.add_heading('Gráficas e interpretaciones', level=1)

memfile = io.BytesIO()
plt.figure(0)
plt.plot(Ventana.dosx,Ventana.servisio, "g")
plt.title("EVOLUCIÓN NIVEL DE SERVICIO",
       fontdict={'size': 16})
plt.xlabel("Simulaciones", size = 14,)
plt.ylabel("Nivel de servicio (%)", size = 14)
plt.grid(True)
plt.savefig(memfile)
self.document.add_picture(memfile, width=Inches(5))
last_paragraph = self.document.paragraphs[-1]
last_paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER
self.document.add_paragraph("En la primera gráfica, podremos
observar la evolución del nivel de servicio. En las primeras
simulaciones el programa empezará a ajustar los parámetros,
siempre en función del nivel mínimo establecido por el usuario
previamente.")
self.document.add_paragraph("Como se ha comentado en más de una
ocasión, con unos tiempos entre demandas generalmente superiores
a los tiempos de aprovisionamiento, no tendremos que establecer
un punto de pedido muy alto para tener buenos niveles de
servicio. Como podremos observar, hay muchas simulaciones en las
que estos son del 100%.")
plt.clf()
memfile.close()
```

Segunda imagen:

```
memfile_2 = io.BytesIO()
plt.figure(1)
plt.plot(Ventana.dosx,Ventana.meanstockout, "g")
plt.title("EVOLUCIÓN ROTURAS DE STOCK",
       fontdict={'size': 16})
plt.xlabel("Simulaciones", size = 14,)
plt.ylabel("Media roturas de stock anuales", size = 14)
plt.grid(True)
plt.savefig(memfile_2)
self.document.add_picture(memfile_2, width=Inches(5))
last_paragraph = self.document.paragraphs[-1]
last_paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER
self.document.add_paragraph("En esta ocasión hemos representado
la media acumulada del número de roturas de stock anuales. Se ha
decidido realizar la representación de esta forma, porque así se
puede observar a la perfección como a medida que se van ajustando
los parámetros vamos reduciendo el número de roturas.")
self.document.add_paragraph("Como se comentó en la primera
gráfica, debido a los tiempos de demanda y del plazo de
aprovisionamiento, vamos a tener un número muy bajo de roturas
que el programa reducirá en no demasiadas simulaciones." )
plt.clf()
memfile_2.close()
```

Tercera imagen:

```
memfile_3 = io.BytesIO()
plt.figure(2)
line1,=plt.plot(Ventana.dosx,Ventana.lanzy, "g", label="C.Lanz")
line2,=plt.plot(Ventana.dosx,Ventana.mttoy, "r", label="C.Mtto")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=4)})
plt.title("EVOLUCIÓN DE LOS COSTES",
       fontdict={'size': 16})
plt.xlabel("Simulaciones", size = 14,)
plt.ylabel("Media costes anuales (u.m)", size = 14)
plt.grid(True)
plt.savefig(memfile_3)
self.document.add_picture(memfile_3, width=Inches(5))
last_paragraph = self.document.paragraphs[-1]
last_paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER

self.document.add_paragraph("Como en la representación anterior,
también se ha hecho uso de la media acumulada, pero esta vez
sobre los costes de lanzamiento y mantenimiento existentes cada
año.")
self.document.add_paragraph("Se puede observar, como se va
buscando el punto óptimo, es decir, donde ambos costes tienen un
valor prácticamente igual. A partir de ese punto, los costes
tenderán a estabilizarse en un valor, más o menos, constante." )
plt.clf()
memfile_3.close()
```

A continuación, generaremos una tabla en la que guardaremos toda la información relativa a los datos históricos y al contraste de hipótesis realizado sobre ellos.

```
paragraph_6=self.document.add_paragraph('')
paragraph_6.paragraph_format.space_before=Inches(2)
self.document.add_heading('Resultados Contraste de Hipótesis',
level=1)
paragraph_4=self.document.add_paragraph('')
paragraph_4.paragraph_format.space_before=Inches(0.05)


med_medio_ajust = round(Ventana.med_dt, 2)
melt_medio_ajust = round(Ventana.med_lt, 2)
coedt_medio_ajust = round(Ventana.coe_dt, 2)
coelt_medio_ajust = round(Ventana.coe_lt, 2)

contrastito = (
 ('Dist_dt', Ventana.resu_dt, 'Distribución ajustada a los
 tiempos entre demandas'),
 ('Dist_lt', Ventana.resu_lt, 'Distribución ajustada a los
 tiempos de aprovisionamiento'),
```

```
 ('μ_d', med_medio_ajust,'Media de los tiempos entre demandas
 (días)'),
 ('μ_pa', melt_medio_ajust, 'Media de los PA (días)'),
 ('cv_d', coedt_medio_ajust, 'Coeficiente de variación de los
 tiempos entre demandas'),
 ('cv_pa', coelt_medio_ajust, 'Coeficiente de variación de los
 tiempos de aprovisionamiento')
 )

table_3 = self.document.add_table(rows=1, cols=3)
table_3.style='Colorful List Accent 6'
hdr_cells_3 = table_3.rows[0].cells
hdr_cells_3[0].text = 'Item'
hdr_cells_3[1].text = 'Valor'
hdr_cells_3[2].text = 'Desc'


for it, valore, desc in contrastito:
 row_cells_3 = table_3.add_row().cells
 row_cells_3[0].text = it
 row_cells_3[1].text = str(valore)
 row_cells_3[2].text = desc
```

Para finalizar, realizaremos una serie de recomendaciones dentro del informe. Además, necesitaremos una dirección donde guardar el documento, y esto es lo que realiza el código siguiente, preguntando previamente al usuario el sitio donde desea hacerlo.

```
self.document.add_heading('RECOMENDACIONES', level=1)
self.document.add_paragraph("Obervando los resultados obtenidos
se pueden sacar algunas conlusiones. La primera de ellas, es que
podemos ser muy exigentes con el nivel de servicio, es decir,
podremos establecer un nivel mínimo considerablemente alto,
puesto que con un mínimo ajuste, los niveles rondarán la
perfección en muchos puntos de la simulación.")
self.document.add_paragraph("Otro punto importante, es establecer
inicialmente un punto de pedido lo más bajo posible, ya que
debido al problema que estamos abordando (piezas de repuesto),
este no va a ser muy alto, por lo que debemos intentar buscar
siempre colocarlo lo más cerca del 0 posible.")
direccio = QFileDialog.getSaveFileName()
raiz = direccio[0]
filepath = raiz + '.docx'
self.document.save(filepath)

self.generar.setStyleSheet("background-color: rgb(182, 182,
182);""font: 75 12pt ")
self.generar.setText('Volver a descargar...')
self.label_7.setText('Informe descargado correctamente')
```

Hemos visto todas y cada una de las partes que componen nuestra aplicación, pero no como inicializarla, y de esto es precisamente, de lo que se encargan estas últimas líneas.

```
app = QApplication(sys.argv)
Ventana = Ventana()
Ventana.show()
app.exec_()
```

Código Desarrollado para la Obtención de las Distribuciones

Tal y como se comentó cuando expusimos la aplicación, había un botón que, al pulsarlo nos devolvía cual era la mejor distribución para cada uno de los tiempos históricos introducidos previamente. Cada vez que se realice esa acción, el código de la aplicación, llamará al desarrollado en el archivo *"costrastep.py"*, el cual usaremos como función.

Empezaremos, como se ha hecho siempre, definiendo las librerías usadas en el código. En este caso solo ha sido necesario el uso de *"scipy"*, que si recordamos lo visto anteriormente, contenía múltiples herramientas estadísticas las cuales nos harán falta para el cálculo de las distribuciones pertinentes.

En la función nos encontramos con el ajuste de los tiempos entre pedido. Usaremos un bucle "*for*", para de esa manera, obtener el pValor correspondiente a cada una de las distribuciones indicadas previamente en la lista. Iremos buscando cuál de ellas tiene el pValor más alto, cosa que llevaremos a cabo gracias al "*if*" colocado en el interior del bucle.

```
import scipy.stats
from scipy.stats import variation
from scipy import mean


def funcion_2(tiempo_dt,tiempo_lt):

    l_distribuciones = ["norm", "expon", "gamma", "logistic",
    "lognorm", "triang", "uniform"]

     mini=0
     for distribucion in l_distribuciones:
          parametrosAutoAjustados =
          eval("scipy.stats."+distribucion+".fit(tiempo_dt)")
          estadistico_KS, pValor = scipy.stats.kstest(tiempo_dt,
          distribucion, args=parametrosAutoAjustados)
          if pValor>=mini:
              tiempo_fin_dt=distribucion
              mini=pValor

     if tiempo_fin_dt=="norm":
         bien="Distribución Normal"
     if tiempo_fin_dt=="expon":
         bien="Distribución Exponencial"
     if tiempo_fin_dt=="gamma":
         bien="Distribución Gamma"
     if tiempo_fin_dt=="logistic":
```

```
        bien="Distribución Logística"
    if tiempo_fin_dt=="lognorm":
        bien="Distribución Log-Normal"
    if tiempo_fin_dt=="triang":
        bien="Distribución Triangular"
    if tiempo_fin_dt=="uniform":
        bien="Distribución Uniforme"
```

Posteriormente obtendremos los coeficientes de variación asociados a cada uno de los tiempos introducidos, así como el valor medio de cada uno de ellos. Además, realizaremos la misma operación que antes para ver la distribución que corresponde al *PA*.

```
    cv_1=variation(tiempo_dt)
    cv_2=variation(tiempo_lt)

    a=mean(tiempo_dt)
    b=mean(tiempo_lt)

    l_distribuciones = ["norm", "expon", "gamma", "logistic",
    "lognorm", "triang", "uniform"]

    mini=0
    for distribucion in l_distribuciones:
        parametrosAutoAjustados =
        eval("scipy.stats."+distribucion+".fit(tiempo_lt)")
        estadistico_KS, pValor = scipy.stats.kstest(tiempo_lt,
        distribucion, args=parametrosAutoAjustados)
        if pValor>=mini:
            tiempo_fin_lt=distribucion
            mini=pValor

    if tiempo_fin_lt=="norm":
        bien="Distribución Normal"
    if tiempo_fin_lt=="expon":
        bien="Distribución Exponencial"
    if tiempo_fin_lt=="gamma":
        bien="Distribución Gamma"
    if tiempo_fin_lt=="logistic":
        bien="Distribución Logística"
    if tiempo_fin_lt=="lognorm":
        bien="Distribución Log-Normal"
    if tiempo_fin_lt=="triang":
        bien="Distribución Triangular"
    if tiempo_fin_lt=="uniform":
        bien="Distribución Uniforme"

    return bien,bien_2,tiempo_fin_dt,tiempo_fin_lt,a,b,cv_1,cv_2
```