Trabajo de Fin de Grado

Grado en Ingenieria de Organización Industrial

Desarrollo de soporte informático para optimización de los intervalos de las tareas de mantenimiento. Implementación de modelos cuantitativos basados en cadenas de Markov

Autor: Marcel Eduardo Martínez Rodríguez

Tutor: Dr. Antonio Sánchez Herguedas

Dpto. de Organización Industrial y Gestión de Empresas

Escuela Técnica Superior de Ingeniería

Sevilla, 2021

Trabajo de Fin de Grado
Ingenieria de Organización Industrial

Desarrollo de soporte informático para optimización de los intervalos de las tareas de mantenimiento. Implementación de modelos cuantitativos basados en cadenas de Markov

Autor:
Marcel Eduardo Martínez Rodriguez

Tutor:
Dr. Antonio Sánchez Herguedas
Profesor Contratado Doctor

Dpto. de Organización Industrial y Gestion de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Carrera: Desarrollo de soporte informático para optimización de los intervalos de las tareas de mantenimiento. Implementación de modelos cuantitativos basados en cadenas de Markov

Autor: Marcel Eduardo Martínez Rodríguez

Tutor: Dr. Antonio Sánchez Herguedas

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

Quiero agradecer a mi familia por apoyarme y enseñarme a siempre trabajar duro por mis metas. A Cristina León por siempre estar a mi lado e inspirarme a crecer.

Agradezco enormemente al Dr. Antonio Sánchez Herguedas por su valiosa ayuda y guía en la elaboración de este proyecto.

A mis grandes amigos que, aunque lejos, siempre están conmigo para ayudarme a seguir adelante.

Marcel Eduardo Martínez Rodríguez

Estudiante de la Universidad de Sevilla

Sevilla, 2021

Resumen

Esta memoria presenta el proceso de desarrollo de un soporte informático que permite la determinación de las estrategias de mantenimiento más adecuadas para un sistema de equipos, basando su funcionamiento en el análisis de la evolución del sistema a través del tiempo mediante procesos estocásticos. Para determinar esto, se emplea el uso de modelos cuatitativos basados en cadenas de Markov, aplicables a cualquier sistema de equipos homogéneos que cumplan con determinadas características básicas. El soporte informático pretende ser una potente herramienta para la gestión del mantenimiento y la toma de desiciónes permitiendo al usuario observar también todos los escenarios alternativos dentro del espacio de soluciones encontradas. Permite a su vez, obténer un análisis de la fiabilidad del sistema mediante la estimación de funciones de probabilidad de fallo, a partir del tiempo de operación entre fallos e intervenciones por mantenimiento.

En el desarrollo de esta memoria se estudia la versatilidad de la herramienta en casos donde se tenga que plantear, para determinados análisis, versiones alternativas modificando la escala de sus datos ingresados y comparando la variación de los resultados para demostrar la precisión en los resultados a un caso lo más próximo a la realidad.

Abstract

This report presents the process of developing a computer support that allows the determination of the most suitable maintenance strategies for a system of equipment, basing its operation on the analysis of the evolution of the system through time using stochastic processes. To determine this, the use of quantitative models based on Markov chains is used, applicable to any system of homogeneous equipment that meets certain basic characteristics. The computer support aims to be a powerful tool for the management of maintenance and decision-making, allowing the user to also observe all the alternative scenarios within the space of solutions found. In turn, it allows the user to obtain an analysis of the reliability of the system from the estimation of probability of failure functions based on the operating time between failures and maintenance interventions.

In the development of this report, the versatility of the tool is also studied for cases where alternative versions have to be proposed for certain analyzes by modifying the scale of the data entered and comparing the variation of the results to demonstrate the precision of the results in a case. the closest to reality.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En el presente trabajo de fin de carrera se abarcará el proceso de investigación y desarrollo de un sistema informático destinado a la mejora de los procesos de gestión del mantenimiento de un equipo o parque de equipos homogéneos como respuesta a la necesidad de implementación de herramientas más modernas con la llegada de la Industria 4.0.

Este programa permitirá al equipo técnico de mantenimiento tener una referencia específica sobre la programación del mantenimiento, inspecciones y costes asociados a esta estrategia, y estudiar la evolución del sistema, el número de averías que el programa pronostica para el equipo en cuestión.

1.1 Justificacion de la investigacion

En cualquier área dentro de un sistema industrial el empleo de técnicas convencionales o "instintivas" de gestión muchas veces no suelen ser totalmente efectivas para éste, ya sea por ser de gran tamaño, complejidad o estar expuesto a condiciones que afectan su naturaleza de tal manera que puede llegar a ser necesario el abarcar más características dentro de los estudios del sistema o valerse de herramientas cuantitativas más potentes. Como consecuencia, se pueden dar situaciones como pérdidas monetarias por la realización de mantenimiento antes de tiempo a ciertos equipos, o alto número de fallos en el sistema por falta de mantenimiento.

La herramienta a desarrollar en esta investigación pretende servir de apoyo al sistema de gestión del mantenimiento de cualquier industria, ayudando a reducir los costes variados que una política de mantenimiento mal diseñada puede ocasionar, empleando procesos estocásticos de cálculo para modelar el estado del sistema en cada instante del tiempo.

1.2 Objetivo de la investigación

Definir un sistema informatico que permita saber cuál es la política de mantenimiento óptima para un equipo o parque de equipos homogéneos, minimizando los costes a partir de ciertas variables de entrada.

1.2.1 Objetivos especificos de la investigación

- Implementar el uso de modelos cuatitativos basados en cadenas de Markov que permitan aproximar la evolución del un sistema a lo largo de distintos períodos temporales.
- Utilizacion del lenguaje de programacion Python para la contruccion del soporte informatico.

1.3 Sumario

La memória de esta investigación está conformada por 6 capítulos y un anexo abarcando en cada uno el siguiente contenido:

- Capítulo 1: la introducción de la investigación donde se realiza una breve explicación de la justificación y objetivos del proyecto, así como presentar el trabajo que se realizará.
- Capítulo 2: éste capítulo comprende toda la base teórica que sustenta la investigación y desarrollo del proyecto, se explican los principios básicos de la función de la gestión del mantenimiento industrial, se presentan los modelos matematicos que se emplean para el funcionamiento de la herramienta a desarrollar. Finalizando con una breve explicación del lenguaje de programación Python utilizado en el desarrollo del programa.
- Capítulo 3: se realiza un análisis del problema que se propone resolver la investigación y el desarrollo de la propuesta de solución.
- Capítulo 4: se presenta un caso práctico para poner en uso la herramienta desarrollada y poder poner a prueba sus prestaciónes mediante resultados experimentales del caso propuesto.
- Capítulo 5: las conclusiones del trabajo a partir de el desarrollo, resultados experimentales y aprendizaje en la realización de este proyecto.
- Capítulo 6: reflexión de mejoras que podrían aplicarse al resultado de este proyecto en trabajos futuros.

2 MARCO TEÓRICO

La herramienta a desarrollar tiene como objetivo en la obtención de la política óptima de mantenimiento a aplicar a un sistema, a partir de la fiabilidad de los equipos que lo conforman y los costes asociados a las operaciones de mantenimiento que incurran. Fundamenta su funcionamiento en el uso de modelos cuantitativos[1] que proyectan la evolución del sistema a travez del tiempo y calcula los costes que representarían en cada caso. En este capitulo se hará una breve introducción al mantenimiento y fiabilidad industrial, y explicará resumidamente el funcionamiento de los modelos usados en el sistema informatico desarrollado.

2.1 Introducción al mantenimiento industrial

Se define como el área encargada de la aplicación de estrategias técnicas para la conservación operativa de la maquinaria e instalaciones de una planta industrial, permitiendo que éstas estén disponibles el mayor tiempo posible. En una empresa el departamento de gestión del mantenimiento debe emplear un uso óptimo de los recursos que dispone y proyectar bien la implementación de las políticas de mantenimiento para minimizar los costes y cumplir con los objetivos.

Los planes de mantenimiento se idean mediante métodos de gestión, que estudian y valoran los parámetros de cada equipo y de cómo participan en el sistema productivo, así como también su impacto en él si éste llegara a fallar. Esto permite organizar las órdenes de trabajo, los recursos y la información documentada que participa y que es generada durante operaciones de mantenimiento para conseguir maximizar la productividad de cada elemento.

2.1.1 Objetivos del mantenimiento

La función de mantenimiento tiene como objetivos el permitir que cualquier elemento que forme parte del sistema productivo pueda realizar su función requerida el mayor tiempo posible y minimizar el deterioro de los equipos e instalaciones de la manera más económica para el sistema al que se implementa.

2.1.2 Influencia de los fallos en un sistema

Dependiendo de la naturaleza el fallo que ocurre y cuándo éste sucede tiene consecuencias relevantes sobre el sistema, se puede determinar mediante un análisis cualitativo o cuantitativo de la criticidad de un fallo para un equipo en concreto para el sistema productivo. El equipo técnico encargado de la gestión del sistema y con conocimiento de las características del equipo debe determinar las consecuencias de un fallo del equipo para todo lo que comprende el sistema productivo de manera escalar y la probabilidad de que éste ocurra, estos dos valores determinarán la criticidad de un equipo y el nivel de atención que debe tener.

2.1.3 Influencia económica del mantenimiento en una organización

Para una organización las operaciones de mantenimiento son vitales y tienen participación directa en el sistema, su implicación radica en que son responsables de la disponibilidad de los elementos principales y auxiliares de una cadena productiva o de servicios, las fallas o averías de los equipos puede afectar la economía de la organización a través de pérdidas de capacidad productiva, accidentes laborales, contaminación ambiental

fabricación de productos defectuosos entre otros.

Las actividades del departamento de mantenimiento están relacionadas con casi todas las partes de una empresa:

•Planificación operacional: casi todas las unidades de la empresa están influenciadas por las operaciones de trabajo de mantenimiento.

•Gestión de recursos: personal, herramientas, software, adquisiciones de materiales y disposición de las instalaciones.

Desde tiempos recientes las organizaciones emplean el uso de soportes informáticos para mejorar sus sistemas de gestión del mantenimiento, están implantados con el objetivo de almacenar datos, aprovechar el poder de cálculo de los ordenadores y mejorar las comunicaciones.

2.2 Caracteristicas de un sistema de gestion del mantenimiento

Para la implementación de un sistema de gestión del mantenimiento se deben analizar varios factores, principalmente porque en la mayoría de los casos no interesará realizar el mismo esfuerzo de mantenimiento a todos los componentes de una máquina sino centrarse en algunas partes en particular de ella, por eso, cuando se analiza la aplicación del mantenimiento preventivo a un equipo es necesario definir el **nivel de intervención** al cual se actúa para cada componente del elemento seleccionado. El nivel de intervención se elige teniendo en cuenta la capacidad que tiene la empresa de actuar en el equipo.

2.2.1 Planificación del mantenimiento y fases del ciclo de gestión

Las acciones de mantenimiento en una empresa tienen impacto directo en sus actividades, para la integración de las políticas de mantenimiento en el plan de acción de una empresa los encargados del departamento de mantenimiento deben tener información completa del rendimiento del sistema mediante estudios exhaustivos y poder planificar la programación de los recursos y el trabajo.

Para esto existe un estándar muy recomendado para la planificación del mantenimiento, definido en la **ISO 55000:2014 de Gestión de Activos**, este ciclo de gestión consta de 8 fases las cuales son:

•Fase 1: definición de objetivos, líneas de acción estratégicas y responsabilidades de mantenimiento.

•Fase 2: jerarquización de los equipos de acuerdo con la importancia de su función.

•Fase 3: análisis de los puntos débiles en equipos de alto impacto.

•Fase 4: diseño de rutinas de mantenimiento y asignación de recursos necesarios.

•Fase 5: programación del mantenimiento y estudio de la utilización de recursos.

•Fase 6: evaluación y control de la ejecución del mantenimiento.

•Fase 7: análisis del ciclo de vida y posible renovación de los equipos.

•Fase 8: implantación de un proceso de mejora continua y de adopción de nuevas tecnologías.

El objetivo de esta investigación se centra en la relación entre la **Fase 4** del ciclo estándar de gestión del mantenimiento y el impacto económico, enfocándose en minimizar los costes de las actividades de mantenimiento mediante la contribución a una eficiente programación de las actividades y asignación de los recursos.

2.3 Tipos de mantenimiento

Actualmente, el mantenimiento se cataloga comúnmente según las circunstancias en las que éste se emplea, permitiendo una gestión sistemática de los procedimientos y su organización definiéndolos como mantenimiento correctivo, preventivo predeterminado y preventivo bajo condición donde se incluye el predictivo.

2.3.1 Mantenimiento correctivo

Son las operaciones de mantenimiento que se limitan a reparar los elementos que han fallado durante su tiempo de funcionamiento, o que han sido dañados e inhabilitados como consecuencia de un fallo secundario.

Suele ser muy útil en industrias con poca carga productiva, donde las averías no están directamente relacionadas con el desgaste.

2.3.2 Mantenimiento preventivo

Cuando no es factible permitir que algún elemento de un equipo funcione hasta fallar, por ser perjudicial para los operarios, el sistema productivo o es económicamente muy costoso, se pueden emplear tareas técnicas que permiten devolver el elemento a un estado similar al que tenía cuando inició su vida operacional, que se realizan de forma periódica para minimizar la probabilidad de que éste falle al funcionar.

Elegir cuándo realizar estas tareas corresponde a los encargados de gestión del mantenimiento, donde analizando los diferentes aspectos del equipo en cuestión definen una agenda de ordenes de trabajo a realizar para el elemento, conscientemente con los recursos que se disponen. Estos normalmente vienen apoyados de una inspección preliminar de los equipos para determinar su estado antes de proceder a realizar las tareas técnicas sobre él.

En las funciones del mantenimiento preventivo existen distintos criterios de aplicación como pueden resaltar el mantenimiento predeterminado y el basado en la condición.

- Mantenimiento preventivo predeterminado: se realiza tomando en cuenta parámetros de los equipos como las horas de funcionamiento, kilometraje, etc. Con el objetivo de designar un cronograma de ordenes de trabajo sobre distintos elementos, se emplea un estudio de las características intrínsecas del equipo en cuestión con lo que se estimará un valor idóneo del instante que se programará estas órdenes.
- Mantenimiento preventivo basado en la condición: consiste en la monitorización en tiempo real del estado de los equipos, la información para estos modelos de gestión se recolecta a través del uso de sensores, a través la medición de cierto parámetro físico o químico mediante termografías, análisis de vibraciones, ultrasonidos, análisis de humos de combustión, el mantenimiento se realizaría cuando alguno de los parámetros medidos alcance ciertos valores inaceptables.
    - Mantenimiento predictivo: analizando las mediciones de parámetros físicos de un elemento en tiempo real se programan trabajos de mantenimiento en el futuro.

El criterio de aplicación que se estudia en esta investigación es el mantenimiento preventivo predeterminado, ya que se empleará una herramienta cuantitativa para determinación de una política de mantenimiento mediante datos de fiabilidad del funcionamiento del equipo, un número de equipos homogéneos al que se determinará su política de mantenimiento y un horizonte temporal para el que se realiza el estudio.

2.4 Expresión matemática para la Fiabilidad de sistemas

El mantenimiento y la fiabilidad de sistemas están estrechamente relacionados, la fiabilidad es la determinación científica de la probabilidad de que un bien o conjunto de bienes desarrollen su función requerida durante un determinado periodo de tiempo bajo condiciones particulares.[2]

La determinación matemática de la duración de vida de un bien o componente implican la utilización de funciones de probabilidad que, modeladas correctamente, reflejan de manera razonable el tiempo de duración de vida del bien como una variable aleatoria T que se expresa su función de distribución acumulada como:

$$F(t) = P(T \leq t) \quad (2\text{-}1)$$

Indicando la probabilidad de que el elemento sobreviva hasta el tiempo t y falle en éste.

La **función de fiabilidad R(t)** indicará la probabilidad de que un equipo sobreviva más de t periodos de tiempo.

$$R(t) = 1 - F(t) \quad (2\text{-}2)$$

2.4.1 Modelo de Markov o cadena de Markov

En fiabilidad para estudiar la probabilidad de transición de un periodo temporal a otro, fallando el elemento o no, se emplea una evaluación de probabilidad condicionada donde el valor de la probabilidad de un suceso depende exclusivamente del suceso anterior (en este caso, que el elemento haya sobrevivido hasta un tiempo t). Esta propiedad se conoce como "propiedad de Markov", la cual es la base de todo el modelo markoviano con el que más adelante se construye una **matriz de transición** a partir de la siguiente expresión:

$$P(t < T \leq t+1 \mid T > t) = \frac{F(t+1) - F(t)}{R(t)} \quad (2\text{-}3)$$

Representando la probabilidad de que un elemento funcionando hasta un tiempo t falle en t +1.

2.4.2 Modelado de la duracion del tiempo de funcionamiento de equipos a partir de funciones de distribucion de probabilidad

Usualmente se puede determinar la duración de los elementos a través de varios métodos, entre ellos el uso de funciones de distribución de probabilidad conocidas que definiendo correctamente sus parámetros puede aproximarse bastante al comportamiento esperado de la tasa de fallos real. Las más usadas actualmente para modelar la fiabilidad son la exponencial y la Weibull.

2.4.2.1 Función exponencial

La función exponencial es más adecuada cuando se trata con un equipo con tasa de fallos constante. Con parámetro de función λ su función de distribución acumulada viene dada por:

$$F(x) = 1 - e^{-\lambda x} \quad (2\text{-}4)$$

Siendo constante la tasa de fallos se puede obtener de la expresión **1/λ** que es igual a la media o esperanza del fenómeno de aparición de fallos.

2.4.2.2 Función Weibull

Particularmente utilizada para fallos que son "variables" crecen o decrecen en el tiempo, usualmente sirve muy bien para modelar la tasa de fallos por envejecimiento del equipo.

$$F(t) = 1 - e^{-\left(\frac{t-k}{\beta}\right)^{\alpha}} \quad (2\text{-}5)$$

Donde **α** representa el factor de forma de la curva, si éste es menor que 1 la tasa de fallos decrece en el tiempo, si es mayor a 1 la tasa de fallos crece en el tiempo y si es igual a 1 es constante.[3]

El factor de escala de la distribución **β**, cuanto mayor es el parámetro de la escala, más amplia es la distribución. El factor de escala representa el percentil 63.2 en todos los datos.

Y **k** el factor de localización (o desplazamiento a lo largo del eje t), coincidiendo con la función Weibull de 2 parámetros cuando éste es igual a 0.

2.5 Modelos cuantitativos

El modelado matemático para la optimización de un sistema clásicamente consiste en la expresión de las especificaciones, recursos y comportamiento de un sistema de elementos que interaccionan entre sí, mediante la relación entre ecuaciones matemáticas que siguen un criterio de optimización. Las **actividades de decisión** serán variables que definen el comportamiento del sistema modelado y cuyo valor se determinará al resolver el modelo siguiendo el criterio de optimización.

La metodología cuantitativa en la que se basa el programa para la determinación de la política más eficiente de mantenimiento para un determinado sistema a estudiar se consultó de trabajos de la *Escuela de Ingenieros Industriales de la Universidad Politécnica de Madrid*[1] conformado por cuatro modelos:

- **Política de mantenimiento correctivo:** consistirá en la realización de reparaciones a los equipos a medida que estos vayan fallando, cuando se alcance el régimen permanente el sistema logra estabilizarse consiguiendo obtener el coste total de esta política multiplicando el número de averías que se pronosticarán por el coste de las reparaciones C_1.
- **Política de mantenimiento preventivo total**: consistirá en la aplicación de mantenimiento preventivo a todos los equipos que conforman el sistema después de un determinado tiempo que se denominará T_2 y con coste C_2. El coste total de esta política se determinará de la suma entre los costes asociados a mantenimiento preventivo y las reparaciones realizadas en cada ciclo por equipos que se han averiado.
- **Política de mantenimiento preventivo en función del tiempo de funcionamiento sin fallos**: se definirá T_3 como el número de periodos de funcionamiento ininterrumpido, cuando determinados equipos lleguen a este tiempo se le aplicará mantenimiento preventivo. El coste total también será la suma entre los costes asociados al mantenimiento preventivo realizado y las reparaciones de los equipos que se hayan averiado.
- **Política de mantenimiento preventivo en función del tiempo de funcionamiento sin fallos y después de salir como en "mal estado" en una inspección previa**: es igual que el anterior con la diferencia de que todos los equipos que se le realizarán mantenimiento por estar en su período de funcionamiento ininterrumpido mayor o igual a T_3, pasarán por una inspección que los calificará como "buen estado" o "mal estado", realizando actividades de mantenimiento sólo si el equipo fue clasificado como "malo". En este caso se incluirá el coste de las inspecciones realizadas y la clasificación del equipo dependerá de la "calidad" de la inspección que se determinará mediante datos proporcionados por el usuario.

2.5.1 Descripcion cuantitativa del fenómeno de aparición de fallos en función del tiempo

En el estudio del fenómeno de aparición de fallos se tomarán en cuenta los siguientes criterios:

- Se tomará el origen del tiempo el instante de puesta en marcha del equipo.
- La unidad de tiempo se llamará como "período elemental".
- En la transición entre instantes sucesivos ha transcurrido un "período elemental".
- Se definen 3 funciones de probabilidad que corresponden a la evolución de sistema a lo largo del tiempo:
    - F(T): probabilidad de que el equipo funcione sin fallo hasta el instante T-1 y falle en el T.
    - S(T): probabilidad de que el equipo no llegue a fallar hasta el instante T.
    - P(T): probabilidad de que uno de los equipos que no llegue a fallar hasta el instante T-1 falle en el T.

De las funciones se deducen las siguientes relaciones:

$$F(T) = S(T-1) - S(T) \tag{2-6}$$

$$P(T) = \frac{F(T)}{S(T-1)} \tag{2-7}$$

2.5.2 Descripcion cuantitativa de la evolucion de un sistema mediante modelos cuatitativos

2.5.2.1 Hipótesis básicas para los modelos

El objetivo de optimización del mantenimiento será propuesto para un equipo en particular o parque de equipos homogéneos.

1. El tiempo de parada para realizar una reparación, inspección o mantenimiento preventivo se considerará despreciable por comparación de la longitud de dos intervalos de tiempo sucesivos de paradas.
2. La efectividad del equipo al realizar su función requerida será siempre buena, independientemente del número de periodos sin fallos que éste tenga o si está en su primer periodo de funcionamiento después de aplicarse sobre él una operación de mantenimiento.
3. Después de realizarse una operación de mantenimiento sobre un equipo éste quedará como en su primer periodo de funcionamiento al iniciar su vida operativa.
4. Se desprecia la probabilidad de que un equipo sufra dos o más fallos en un mismo periodo elemental, por lo que estos casos no son admitidos dentro del modelo.

2.5.2.2 Descripción del estado de un sistema en un instante T

Considerando un sistema integrado por N equipos homogéneos *Ni(T)* representará la cantidad de equipos que estén en su número i de periodos de funcionamiento ininterrumpido medido en el instante T desde la puesta en marcha del sistema completo. Se puede caracterizar de esto que:

$$\sum_{i=1}^{T1} Ni(T) = N$$
$$\forall i / i > T_1 : N_i = 0 \tag{2-8}$$

Por lo tanto, el vector de estado de un sistema *E(T)* estará conformado por:

$$E(T) = [N_1(T), N_2(T), \dots, N_i(T), \dots, N_{T1}(T)] \tag{2-9}$$

Dividiendo el vector por N para obtener su versión normalizada lo tendremos como:

$$e(T) = [n_1(T), n_2(T), \dots, n_i(T), \dots, n_{T1}(T)] \quad (2\text{-}10)$$

2.5.2.3 Política de mantenimiento correctivo

Consiste en dejar que el equipo funcione hasta el fallo, sin realizarle ningún tipo de mantenimiento preventivo, y una vez esto ocurra proceder con la reparación o sustitución de los elementos afectados. Para el caso de los modelos de esta investigación solo se analizará las características económicas de las interacciones de los fallos de un equipo en el sistema.

Tales características como sean el coste de la mano de obra, las herramientas utilizadas, las piezas de recambio y las perdidas productivas ocasionadas por el fallo ya sea que éste estuviera previsto o no, se tomará la suma de estos costes medios como C_1. El resto de las variables que puedan influir en el valor de esta constante ya están definidas en las hipótesis básicas para los modelos.

2.5.2.3.1 Estudio de la evolución de un sistema en la transición de un período a otro. Matriz de transición

Si un equipo se encuentra en el instante T en su periodo i de funcionamiento ininterrumpido, en el instante T+1 podrá ocurrir en una de las dos situaciones siguientes:

- En su primer período de funcionamiento sin fallos sufre una avería, esto con probabilidad:

$$p_{i,1} = P_{T=i}(T) \quad (2\text{-}11)$$

- En su i+1 período de funcionamiento sin fallos y no se ha averiado, esto con probabilidad:

$$p_{i,i+1} = 1 - P_{T=i}(T) \quad (2\text{-}12)$$

Para en el modelo estudiar la transición de un estado del sistema a otro se valdrá del vector de estado y la matriz de transición, teniendo el vector de estados en un determinado instante T se podrá conocer el estado del sistema en el instante siguiente T+1 a partir de multiplicar *E[T]* por la matriz de transición $p_{i,j}$.

$$p = \begin{bmatrix} p_{1,1} & p_{1,2} & 0 & \dots & 0 \\ p_{2,1} & 0 & p_{2,3} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ p_{T1-1,1} & 0 & 0 & \dots & p_{T1-1,T1} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2\text{-}13)$$

De esta matriz se puede observar:

- $p_{T1,\,1} = 1$. Porque está definido que todo equipo que llegue a su período T1 de funcionamiento ininterrumpido fallará.
- $\forall i: \ \sum_j p_{i,j} = 1$

La relación entre dos estados sucesivos del sistema quedaría como:

$$E(T+1) = E(T) * p_{ij} \quad (2\text{-}14)$$

Si se tiene el estado inicial del sistema *E(0)* se puede obtener la evolución de éste al instante siguiente:

$$E(1) = E(0) * p_{ij}$$

$$E(2) = E(1) * p_{ij} = E(0) * (p_{ij})^2$$

De aquí se deduce la ecuación que, conociendo el estado inicial del sistema se podrá conocer el estado de éste en cualquier T.

$$E(T) = E(0) * (p_{ij})^T$$

$$E(T + K) = E(T) * pij^K \quad (2\text{-}15)$$

2.5.2.3.2 Estudio del modelo en régimen permanente

El sistema que inicia en un estado *E(0)*, se deja evolucionar durante un número muy grande de periodos éste tenderá al régimen permanente, que poseerá una propiedad muy interesante:

$$Ep = \lim_{T \to \infty} (E(0) * pij^T) \quad (2\text{-}16)$$

$$Ep = Ep * pij \quad (2\text{-}17)$$

Buscamos el valor de *Ep* en la ecuación matricial:

$$Ep * (p_{ij} - I) = 0 \quad \text{o} \quad ep * (p_{ij} - I) = 0 \quad (2\text{-}18)$$

Que equivale a un sistema de T_1 ecuaciones lineales homogéneas y le agregamos la ecuación:

$$\sum_i np_i = 1 \quad (2\text{-}19)$$

Siendo $np_i = \lim_{T \to \infty} ni(T)$

Este sistema de ecuaciones se resolverá asignando a cualquier componente de *ep* un valor arbitrario como $np_1 = 1 = n'p_1$, y al resolverlo se obtiene el vector:

$$e'p = [n'p_1, n'p_2, \dots, n'p_{T1}] \quad (2\text{-}20)$$

Que será la solución del sistema de ecuaciones homogéneas y para cualquier otro vector ke´p, siendo k una constante arbitraria. Si se hace $k = \frac{1}{\sum_i n'pi}$ se obtendrá:

$$np_i = n'p_i * k \quad (2\text{-}21)$$

2.5.2.3.3 Estudio económico del modelo en régimen permanente

Como se sabe el np_1 representará el valor porcentual de equipos que se averiaron en un período anterior y que en el instante corriente se encuentran en su primer período de funcionamiento, lo que indicará el número de reparaciones que se realizaron en total para el sistema que multiplicándolo por el valor de C_1 se obtiene el coste total de la política de mantenimiento correctivo.

$$C_1 * N * np_1 = coste\ total \quad (2\text{-}22)$$

2.5.2.4 Políticas de mantenimiento preventivo

Para todos los costes que comprenden la actividad de un mantenimiento preventivo se corresponde al coste

unitario C_2 de cada equipo que se le aplique la actividad, se asume que por lo general C_2 siempre será mucho menor que C_1 y por conveniencia para el modelo.

2.5.2.4.1 Mantenimiento preventivo total

Esencialmente consiste en aplicarle a todo el sistema operaciones de mantenimiento preventivo sin importar el tiempo de funcionamiento ininterrumpido que tengan. T_2 corresponde al número de períodos elementales entre dos acciones de mantenimiento preventivo, también denominado "ciclo".

En este caso se plantea el vector *Ef* que corresponderá a el vector de estados del sistema inmediatamente anterior a la aplicación de mantenimiento preventivo sobre el sistema, por lo que cuando se le aplique mantenimiento preventivo el instante posterior estará determinado por el vector *E(1)*. Por lo que en este modelo se plantea que se realice mantenimiento sobre el sistema cada T_2 períodos devolviendo al sistema siempre al estado:

$$E(1) = [N, 0, 0, 0, \dots, 0]$$

Y se define una matriz de mantenimiento M, de T_1 filas y columnas, que multiplica al vector de estados para que este vuelva a su estado *E(1)*.

$$M = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2\text{-}23)$$

2.5.2.4.1.1 Estudio del modelo en régimen permanente

En un número muy grande de períodos de evolución del sistema se estabiliza, e independientemente del valor de *Ef*, este siempre se tendrá a *E(1)*.

$$E(1) = Ef * M \quad (2\text{-}24)$$

$$E(2) = E(1) * p_{ij}$$

$$E(3) = E(2) * p_{ij} = E(1) * (p_{ij})^2$$

$$E'(T2 + 1) = E(T2 + 1) * M = E(1) \quad (2\text{-}25)$$

2.5.2.4.1.2 Estudio económico del modelo en régimen permanente

Se sabe que a pesar de realizárse mantenimiento preventivo periódicamente a todos los equipos del sistema no implica que no puedan ocurrir fallos, ya que aún existe probabilidad según se planteó en la matriz de transición de que alguno pueda fallar. Se plantea entonces el número de averías, y por tanto reparaciones, por ciclos como:

$$Rc = N * \sum_{T=2}^{T2+1} n1(T) \quad (2\text{-}26)$$

Rc representa la suma de todas las averías ocurridas en cada instante T desde el inicio del ciclo hasta su fin, donde ya en $T_2 + 1$ correspondería entonces a realizarse mantenimiento preventivo a todos los elementos del sistema. Siendo n_1 el valor porcentual de equipos que en un determinado instante T empiezan su primer período de funcionamiento sin fallos, por lo que en el período T-1 estos han fallado y se les aplicó mantenimiento

correctivo. Teniendo entonces el coste de mantenimiento correctivo de este modelo: $K_{Rc} = C_1 * Rc$.

Y su coste por períodos:

$$K_{Rp} = \frac{(C_1 * R_c)}{T_2} \quad (2\text{-}27)$$

Los costes de mantenimiento preventivo, por otra parte, en cada ciclo serían el número de equipos del sistema por el coste:

$$K_{Mc} = C_2 * N.$$

Y su coste por períodos:

$$K_{Mp} = \frac{(C_2 * N)}{T_2} \quad (2\text{-}28)$$

La formulación para el coste medio total por período de este modelo de mantenimiento preventivo sería:

$$K_{Tp} = K_{Mp} + K_{Rp} \quad (2\text{-}29)$$

2.5.2.4.2 Mantenimiento preventivo periódico en función del tiempo de funcionamiento sin fallos

Se define T_3 como el número de períodos de funcionamiento ininterrumpido sin fallos, y T_2 seguirá siendo la duración del ciclo entre dos acciones sucesivas de mantenimiento preventivo.

El instante inicial *Ef* del modelo se obtendrá de multiplicar dicho vector del estado inicial por la matriz M, que ahora tiene la forma:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (2\text{-}30)$$

Con las propiedades:

- $\forall i / i \leq T_3 : M_{i,i} = 1$; ya que no se actuará sobre grupos de equipos que lleven T_3 o menos períodos de funcionamiento después del mantenimiento preventivo o correctivo.
- $\forall i \ / \ i > T_3 : M_{i,1} = 1$; ya todos los equipos al llegar a su tiempo T_3 de funcionamiento sin fallos serán enviados a mantenimiento preventivo.
- $\forall i : \sum_j M_{ij} = 1$

2.5.2.4.2.1 Estudio del modelo en régimen permanente

Para este caso conviene recordar lo propuesto en el modelo anterior, solo que en este caso después de evolucionar un numero de períodos muy largo el sistema no se estabiliza de manera inmediata como en el anterior, en este caso el régimen permanente sería de la forma:

$$E(1) = Ef * M$$

$$E(2) = E(1) * p_{ij}$$

....

$$E(T_2 + 1) = Ef * M * (p_{ij})^{T2} \quad (2\text{-}31)$$

Y la "matriz de cambio" será:

$$C = M * (p_{ij})^{T2} \quad (2\text{-}32)$$

Siendo el sistema matricial:

$$Ef * (C - I) = 0 \quad (2\text{-}33)$$

El objetivo será buscar el valor del vector *Ef*, que se resolverá de la misma manera que en el modelo de política de mantenimiento correctivo.

2.5.2.4.2.2 Estudio económico del modelo en régimen permanente

Tomando en cuenta para la presencia de los fallos dentro de los ciclos de mantenimiento preventivo se utiliza Rc para medir el valor acumulado de las averías que en cada instante T comprendido desde T igual a 0 hasta la siguiente operación de mantenimiento preventivo han ocurrido.

$$Rc = N * \sum_{T=2}^{T2+1} n1(T) \quad (2\text{-}34)$$

Los costes por ciclo $K_{Rc} = C_1 * Rc$ (2-35)

Y por período $K_{Rp} = (C_1 * Rc) / T_2$ (2-36)

De lo estudiado en el punto anterior se determina entonces el número de operaciones de mantenimiento preventivo por ciclo basado en un tiempo T_3 de períodos de funcionamiento ininterrumpido sin fallos Mc corresponde a la ecuación:

$$Mc = N * \sum_{i=T3+1}^{T1} nfi = N * \sum_{i=T3+1}^{T1} ni(T_2 + 1) \quad (2\text{-}37)$$

Con costes por ciclo:

$$K_{Mc} = C_2 * Mc \quad (2\text{-}38)$$

Y por período:

$$K_{Mp} = \frac{(C_2 * Mc)}{T_2} \quad (2\text{-}39)$$

En la ecuación de Mc, n_{fi} representa la probabilidad de que en el vector de estado Ef un equipo se encuentre en su período i de funcionamiento ininterrumpido desde la última vez que fue sometido a alguna operación de mantenimiento.

Las variables de decisión del modelo en este caso serán principalmente T_2 y T_3, y el coste total medio de la

política de mantenimiento se determinará con la ecuación de K_{Tp}.

$$K_{Tp} = K_{Mp} + K_{Rp} \quad (2\text{-}40)$$

2.5.2.4.3 Mantenimiento preventivo periódico basado en el tiempo de funcionamiento sin fallos y el resultado de una inspección previa

Puede pasar en los modelos de mantenimiento preventivo anteriores se aplique una operación de mantenimiento sobre algún equipo que podría seguir funcionando sin fallos, contribuyendo a elevar los costes y haciendo ligeramente menos eficiente estas estrategias en determinados sistemas.

Se plantea para evitar la inversión de capital en mantenimiento de equipos que aparentemente podían seguir funcionando sin fallos el uso de una inspección, que determinará de manera cualitativa si el equipo está en condiciones para continuar operando sin pasar por mantenimiento calificándolo como "bueno", aun así este haya cumplido el número de períodos necesarios para tener que pasar por una operación de mantenimiento preventivo, o si se considera que en efecto sí debe pasar por una operación de mantenimiento llegado el número de períodos de funcionamiento necesario calificándolo como "malo".

Este modelo por tanto dependerá fuertemente de la precisión de la inspección que se les realizarán a los equipos.

2.5.2.4.3.1 Análisis de la calidad del proceso de inspección

Se definirán entonces dos parámetros para introducir y manipular la calidad de las inspecciones realizadas. Se define $\mathbf{p_1}$ como la probabilidad de que un equipo realmente bueno sea declarado como **malo** y $\mathbf{p_2}$ como la probabilidad de que un equipo realmente malo sea declarado como **bueno**.

Con el empleo ahora de una inspección previa al mantenimiento se define una nueva matriz M que actuará de manera que se incluyan en las operaciones de mantenimiento preventivo aquellos equipos que la inspección haya clasificado como malos y excluyendo de esto a los que se hayan clasificado como buenos, pero aun así teniendo en cuenta en la matriz que los equipos que estén en su T_1 período fallarán y por tanto se someterán a mantenimiento correctivo.

Apoyándose en lo anterior se definen las siguientes propiedades de la matriz M:

- $\forall j \,/\, j \neq 1$: $M_{T1,1} = 1 \quad Y \quad M_{T1,j} = 0$
- $\forall ij \,/\, i \neq j \; Y \; i \leq T_3$: $M_{i,i} = 1 \quad Y \quad M_{ij} = 0$; ya que el modelo no plantea realizarle la inspección a los equipos que lleven igual o menos de T_3 períodos de funcionamiento ininterrumpido.
- Para el resto de las posiciones se deberá cumplir que:

$$M_{i,1} = p_{i,1} * (1 - p_2) + p_{i,i+1} * p_1 \quad (2\text{-}41)$$

$$M_{i,i} = p_{i,1} * p_2 + p_{i,i+1} * (1 - p_1) \quad (2\text{-}42)$$

Similar al modelo anterior, la matriz M será la que a partir de un estado inicial Ef que representará el momento inmediatamente anterior a la operación de mantenimiento de este modelo se obtiene el vector *E(1)* siendo el estado inicial del siguiente ciclo.

$$E(1) = Ef * M$$

2.5.2.4.3.2 Influencia de la política de mantenimiento selectivo sobre la evolución del sistema

Para modelar la influencia de un mantenimiento selectivo basado en las variables p_1 y p_2, la nueva matriz de transición W permitirá tener en cuenta que los equipos que debían pasar por mantenimiento preventivo pero que fueron calificados como "buenos" en una inspección deberá reflejar una tendencia de fallar menor que la media de los equipos normalmente. Por otra parte, aquellos teniendo también en cuenta que los equipos que aún están en su periodo $i \leq T_3$ evolucionarán como siempre.

$$\forall ij \,/\, i \leq T_3 \colon W_{ij} = p_{ij}$$

Para el resto de las posiciones de la matriz se cumplirá:

$$W_{i,1} = \frac{p_{i,1} * p_2}{p_{i,1} * p_2 + p_{i,i+1} * (1 - p_1)} \quad (2\text{-}43)$$

$$W_{i,i+1} = \frac{p_{i,i+1} * (1 - p_1)}{p_{i,1} * p_2 + p_{i,i+1} * (1 - p_1)} \quad (2\text{-}44)$$

Y el resto de los elementos de las filas i serán nulos.

Por lo que ahora para estudiar la transición de un instante a otro regulando el comportamiento de los elementos que fueron declarados como "buenos" en las inspecciones realizadas en el ciclo anterior se hará de la siguiente forma:

$$E(1) = Ef * M$$
$$E(2) = E(1) * W = Ef * M * W$$
$$E(3) = E(2) * P = Ef * M * W * p_{ij}$$
$$\ldots\ldots\ldots$$
$$E(T) = E(T-1) * p_{ij} = Ef * M * W * (p_{ij})^{T2} \quad (2\text{-}45)$$

La ecuación anterior representará el régimen permanente siguiendo las ecuaciones de transición anteriores.

El vector que definirá el estado anterior a la próxima inspección será:

$$E(T_2 + 1) = Ef * M * W * pij^{T2-1} = Ef \quad (2\text{-}46)$$

Siendo C = M * W * $(pij)^{T2-1}$ la matriz de cambio.

Y para obtener los valores del vector Ef se resuelve el sistema de ecuaciones siguiente:

$$Ef * (C - I) = 0$$

Que debe resolverse igual que en los modelos anteriores.

2.5.2.4.3.3 Estudio económico del modelo en régimen permanente

La ecuación del coste medio de esta política de mantenimiento se determina por el número de inspecciones por ciclo, el número de operaciones de mantenimiento preventivo realizadas y el número de averías total expresadas a continuación.

Número de equipos inspeccionados durante el ciclo:

$$Ic = N * \sum_{i=T3+1}^{T1-1} ni(T_2 + 1) \quad (2\text{-}47)$$

Con coste de las inspecciones por ciclo de:

$$K_{Ic} = C_3 * Ic \quad (2\text{-}48)$$

Y coste por período:

$$K_{Ip} = \frac{C_3 * Ic}{T_2} \quad (2\text{-}49)$$

Para el cálculo del número total de equipos que se les realiza operaciones de mantenimiento preventivo se obtendrán del número de equipos que aparecen en su primer período de funcionamiento sin fallos en un instante T menos los que se les acaba de aplicar mantenimiento correctivo.

$$Mc = [n_1(1) - n_1(T_2 + 1)] * N \quad (2\text{-}50)$$

Por lo tanto, los costes empleados en mantenimiento preventivo de esta política por ciclo serán:

$$K_{Mc} = C_2 * Mc \quad (2\text{-}51)$$

Y coste por período:

$$K_{Mp} = \frac{C_2 * Mc}{T_2} \quad (2\text{-}52)$$

Y finalmente el número de averías del sistema y por tanto operaciones de mantenimiento correctivo por ciclo:

$$Rc = N * \sum_{T=2}^{T_2+1} n_1(T) \quad (2\text{-}53)$$

El coste asociado a estas operaciones por ciclo será:

$$K_{Rc} = C_1 * Rc \quad (2\text{-}54)$$

Y coste por período:

$$K_{Rp} = \frac{C_1 * Rc}{T_2} \quad (2\text{-}55)$$

Con variables de decisión también T_2 y T_3 el coste medio de la política de mantenimiento será:

$$K_{Tp} = K_{Ip} + K_{Mp} + K_{Rp} \quad (2\text{-}56)$$

2.6 Obtención analítica de los parámetros de forma y escala de una F.D. Weibull a partir del tiempo de operacion entre fallos y mantenimiento preventivo

A continuacion se presenta un método[5] compuesto por varias etapas para el cálculo de los parámetros de una función de distribución de probabilidad Weibull, a partir de una tabla de datos con multiples medidas de tiempos de operacion entre fallos y mantenimiento preventivo de un equipo en cuestión.

El proceso está compuesto por varias etapas empezando por calcular, mediante el estimador estadistico Rango de Mediana (2-59), la Funcion Observada de distribucion de probabilidad *Fi*, ésta es una estimación teórica de la función de distribucion Weibull a partir de los datos de fallos; seguidamente se realiza el cálculo de la ecuación de regresión de la función Weibull transformada logaritmicamente.

2.6.1 Cálculo de la Función Observada mediante el estimador estadístico Rango de Mediana

Se empieza por ordenar los datos de la tabla de datos de forma creciente según el tiempo de operación como se muestra en la Tabla 2-1 con valores extraidos de un ejemplo en una investigación de referencia[5], la tabla debe tener en sus columnas el número de la medida, las horas de operación y el motivo de parada (siendo F en caso de fallo, y R en caso de mantenimiento preventivo).

Cuando se han ordenado los datos se determina un valor de "nuevo numero de la medida" utilizando las ecuaciones (2-57) y (2-58) exclusivamente para las mediciones de paros por fallo.

$$incremento = \frac{N + 1 - (nuevo\ numero\ de\ medida\ del\ elemento\ anteior\ que\ falló)}{N + 1 - (numero\ de\ elementos\ por\ debajo)} \qquad (2\text{-}57)$$

Donde N es el número total de medidas.

$$Nuevo\ número\ de\ la\ medida = incremento + número\ de\ la\ medida \qquad (2\text{-}58)$$

Tabla 2-1. Medidas ordenadas por horas de operación en orden creciente.

Numero de medida	Horas de operación	Motivo del paro	Incremento	Nuevo numero de la medida	*Fi*
81	45	R			
13	84	R			
9	84	R			
109	103	R			
33	121	R			
29	176	R			
71	190	F	1.05172	1.05172	0.006192
112	199	R			
92	217	R			
16	259	R			
12	276	F	1.07989	2.1316	0.015087
51	296	F	1.07989	3.2115	0.023982
…	…	…	…	…	

Para el cálculo de la función observada (*Fi)* se utiliza la ecuación de Rango de Mediana (2-58) donde i es el nuevo número de la medida calculado en la Tabla anterior para las medidas de fallos, este estimador (2-58) da valores que son suficientemente exactos para el estudio de fiabilidad en cuestión.

$$Fi = \frac{i - 0{,}3}{N + 0{,}4} \tag{2-59}$$

2.6.2 Método de Mínimos cuadrados

Para determiner los parámetros de forma y escala asociados a la función Weibull se realiza una doble transformacion logarítmica de ésta.

$$F(t) = 1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{2-60}$$

$$\frac{1}{1 - F(t)} = e^{\left(\frac{t}{\beta}\right)^{\alpha}}$$

$$\ln\left(\frac{1}{1 - F(t)}\right) = \ln\left(e^{\left(\frac{t}{\beta}\right)^{\alpha}}\right)$$

$$\ln\left(\frac{1}{1 - F(t)}\right) = \left(\frac{t}{\beta}\right)^{\alpha}$$

$$\ln\left[ln\left(\frac{1}{1 - F(t)}\right)\right] = \alpha * \ln\left(\frac{t}{\beta}\right)$$

$$\ln\left[ln\left(\frac{1}{1 - F(t)}\right)\right] = \alpha * \ln(t) - \alpha * \ln(\beta) \tag{2-61}$$

La ecuación (2-61) es una representación de la recta de regresión lineal $Y = mx - b$, donde:

$$Y = \ln\left[\ln\left(\frac{1}{1-F(t)}\right)\right] \tag{2-62}$$

$$x = \ln(t) \tag{2-63}$$

$$b = \alpha * \ln(\beta)$$

De los datos de la Tabla (2-2) usando el tiempo de operación como **t** y la función de distribucion observada

como **F(t)** se determinan los valores de x e Y de la tabla de valores para calcular la pendiente (α) y el intercepto (b) de la recta de regresión.

Tabla 2-2. Valores de la Función de distribución observada y valores para regresión lineal.

Horas de operación (t)	Función de distribución observada (F(t))	ln(t)	ln[ln(1/1-F(t))]
190	0.00619	5.2470	-5.0813
276	0.01509	5.6204	-4.1862
296	0.02398	5.6903	-3.7183
…	…	…	…

Para el ejemplo de los datos obtenidos en la Tabla 2-2[5] el valor de la pendiente calculada es igual a **2.358** que, como se mencionó anteriormente, es el valor del parámetro de forma de la función Weibull, para determinar el parámetro de la escala se utiliza la expresión (2-64), dando como resultado **1318.27**.

$$\beta = e^{-\frac{b}{\alpha}} \qquad (2\text{-}64)$$

2.7 El lenguaje de programación Python

Python es un lenguaje de programación interpretado, multiparadigma y multiplataforma que fue creado alrededor del año 1989 en el *Centro para las Matemáticas y la Informática* en los Países Bajos. Posee un gran rango de funcionalidades y recursos de calculo que permitirán el desarrollo de versátiles herramientas para ingeniería y tiene un amplio conjunto de librerías de uso libre a disposición creadas por la comunidad Open Source.

Tiene la ventaja que podrá ejecutarse el sistema informático desarrollado en este lenguaje en cualquier computador que tenga instalado su interprete (que puede obtenerse gratuitamente desde su pagina web oficial[5]) y dependencias necesarias.

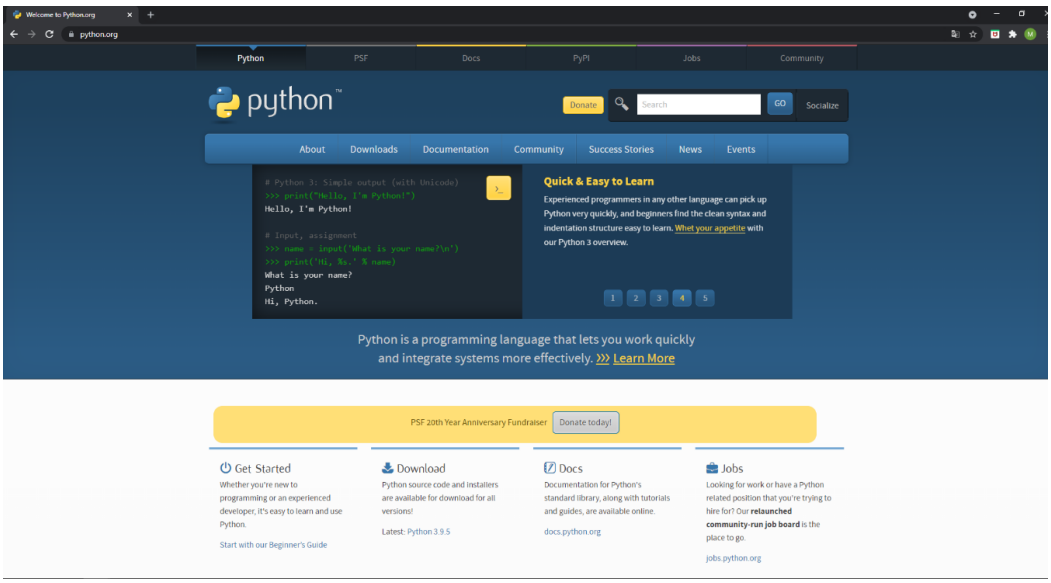

Figura 2-1. Lenguaje de programación Python. Fuente: Python.org[5]

2.7.1 Librerías de Desarrollo

Con vistas a las necesidades que presenta el sistema a desarrollar en este proyecto, entre las herramientas disponibles para el leguaje Python, se usará la librería de **Numpy** [7] para aprovechar sus facilidades en el manejo de matrices y vectores de gran tamaño, **MatplotLib** [8] para la creación de graficas de los valores que obtendremos de los modelos y **Tkinter** para el desarrollo de la interfaz de interacción con el usuario. Todas estas herramientas son de uso libre junto con el lenguaje de programación.

2.8 Arquitectura de un sistema informatico

En la programación y diseño de este algoritmo será conveniente estructurarlo de manera **modular** o en subfunciones, lo que facilitará su manejo, versatilidad y detección de errores. En el sistema informático a realizar las unidades funcionales se desarrollarán en módulos por separado que se irán conectando con la estructura central del algoritmo, esto en diseño estructurado se le llama acoplamiento normal que consiste en un módulo principal llamará a uno secundario y estos tan sólo intercambiarán datos, parámetros de entradas y salidas.

2.8.1 Ventajas de la Descomposición Modular

A simple vista consiste en descomponer el problema en tareas más simples. Cada tarea representa un problema en si con su respectiva codificación independiente. Cada subproblema se puede descomponer hasta obtener el nivel de complejidad deseado de los subproblemas, su correspondiente resolución se denomina refinamiento por pasos.

Se escoge esta arquitectura de software por las siguientes razones:

- Claridad.
- Reducción de costos/tiempo.
- Facilidad de detección de errores.
- Reutilización.

Los pasos de planificación de la descomposición modular serán identificar los módulos, describir cada módulo y describir las relaciones entre los módulos[6], contenido que será desarrollado en el Capítulo 3 de esta memoria.

3 DESARROLLO

En éste capítulo se presenta y analiza el problema objetivo de la investigación. Se muestra la metodología, herramientas y proceso de resolución empleados junto con pequeñas demostraciónes del resultado final utilizando un caso de ejemplo.

3.1 Análisis del problema

Como se destacó en el capitulo anterior, los cuatro modelos cosultados del trabajo del Dr. J. R. Figuera[1] utilizan 4 parámetros que son los costes de reparación, mantenimiento e inspecciones; el número de equipos que conforman el sistema, la precisión de las inspecciones y una matriz de probabilidades de transición. La matriz de transición se obtiene de una función de distribución de probabilidad acumulada (en la mayoría de los casos una FDP Weibull) generada a partir de un análisis estadístico del fenómeno de fallos del sistema estudiado.

Con el objetivo de mejorar las prestaciones del programa, entre sus características se tiene la posibilidad de generar la matriz de transición a partir de una función exponencial o Weibull según su caso mediante sus parámetros ingresados por el usuario en la interfaz; o también si el usuario dispone de una base de datos que contenga mediciones de los tiempos de operación entre de fallos y mantenimiento preventivo en un formato especificado en las instrucciones del programa.

3.1.1 Elementos del problema

Para realizar el análisis de los elementos, el problema se dividirá en subproblemas que corresponden a cada característica del sistema informático.

Se analizará el problema mediate los elementos que generan la interfaz con el usuario (viendo estos como el "cuerpo principal" de programa) y los que realizan las operaciones matemáticas para los cálculos que éste solicite en secciones independientes (estos como componentes secundarios que interaccionan con el "cuerpo principal").

3.1.1.1 Interfaz gráfica

El programa en sí está conformado por 3 pantallas de interacción con el usuario, cada una con elementos distintos para sus diferentes propósitos siendo la "*pantalla inicial"* la pantalla de partida al iniciarse el programa por primera vez, a partir de esta se accederá progresivamente a la *"pantalla de nuevo análisis"* y la *"pantalla de resultados"*. El código que conforma la interfaz gráfica será considerado como el "cuerpo" del programa, ya que desde él se controlarán también las llamadas a las funciones de acceso a datos, cálculos y graficación de valores.

Por conveniencia en el bloque del análisis de interfaz gráfica se describen y analizarán individualmente cada una de las pantallas.

3.1.1.1.1 Pantalla inicial

La pantalla de interacción con el usuario desde donde comenzará el programa se dispone a dirigir al usuario en 2 direcciones, si desea iniciar un análisis nuevo o si desea cargar los resultados de un análisis anterior ya realizado. Para esto estará compuesta por 3 botones dispuestos en el centro de la pantalla, el botón con nombre

"nuevo análisis", el botón "cargar análisis" y un botón de "ayuda", este utilmo desplegará una ventana emergente con información sobre el programa, una breve descripción de éste e indicaciones para continuar.

"Cargar análisis" abre una pestaña para examinar un documento antiguo, generado por el propio programa cada vez que éste realice un análisis, en formato ".csv" (Valores Separados por Coma), con los datos de un análisis previo que al seleccionarlo y cargarlo en el programa se abrirá directamente en la *pantalla de resultados*.

Esta pantalla tendrá también centrado el nombre del programa a modo de presentación.

3.1.1.1.2 Pantalla de nuevo análisis

Si desde la *pantalla inicial* se hace clic en el botón de *nuevo análisis* se dirigirá a la *pantalla de nuevo análisis*, que contendrá los campos respectivos para introducir todos los datos que necesitan los modelos en las unidades de cálculo del programa para retornar los resultados que se esperan. Estos campos dispuestos de forma que resulten intuitiva para el usuario serán el nombre del equipo, coste de reparación, coste de mantenimiento, coste de inspección, probabilidad de en una inspección clasificar un equipo malo como bueno, probabilidad de en una inspección clasificar un equipo bueno como malo, valor de T_1, y dos botones, uno para cargar al programa la matriz de probabilidad, también las opciones para calcularla, y finalmente el botón de *calcular*.

La matriz de probabilidad p_{ij} se carga en el programa desde un archivo ".csv" que contendrá los valores de ese equipo en particular.

3.1.1.1.3 Pantalla de resultados

Después de ser calculados los resultados de los modelos con los valores suministrados en la pantalla anterior se muestran los resultados óptimos para cada política de mantenimiento, donde el usuario elegirá cual será mejor basado en los resultados que se muestran en los campos de cada uno mostrados de manera seccionada.

- Política correctiva: se mostrarán los campos donde se muestre el coste total de esa política y el número de averías que se estimó.
- Política preventiva total: se mostrará en sus campos correspondientes el coste total de esa política, el número de averías que se estimó en el modelo y el valor del T_2 óptimo y un botón para el despliegue de una gráfica en una ventana emergente que muestre los valores de T_2 vs el Coste total y otro para mostrar el numero de averías estimado según los valores de cada T_2.
- Política basada en un numero T_3 de períodos de funcionamiento ininterrumpido: en esta sección se mostrará el coste total de esa política, el número de averías que se estimó, el valor de los T_2 y T_3 óptimo y un botón que desplegará en una ventana emergente una gráfica en 3D mostrando los valores de T_2 vs T_3 vs Coste total y el numero de averías estimado según los valores de las variables T_2 y T_3.
- Política basada en un numero T_3 de períodos de funcionamiento ininterrumpido con inspecciones: en esta última sección se mostrarán también el coste total de esta política, los valores de los T_2 y T_3 óptimos y finalmente un botón para desplegar en una ventana emergente una gráfica en 3D mostrando los valores de T_2 vs T_3 vs Coste total y el numero de averías estimadas según los valores de T_2 y T_3.

3.1.1.2 Módulos de calculo

Todas las unidades de cálculo según sea su caso recibirán los costes (C_1, C_2, C_3), T_1, la matriz pij y las probabilidades asociadas a la calidad de las inspecciones p_1 y p_2.

Las cuatro funciones de mantenimiento por separadas retornarán al programa principal en cada caso los valores siguientes:

- Correctivo: coste total, número de averías.
- Preventivo total: coste total, número de averías, T_2, vector averías, vector de costes.
- Preventivo sin inspección previa: coste total, número de averías, T_2, T_3, vector de averías, vector de costes.

- Preventivo con inspección previa: coste total, número de averías, T_2, T_3, vector de averías, vector de costes.

Adicionalmente, en esta sección se tienen las unidades que realizan los cálculos de la matriz de transición de 3 formas, mediante una F.D.A Exponencial, F.D.A Weibull o un método analítico para el cálculo aproximado de una función Weibull.

3.2 Desarrollo de la propuesta de solución

El soporte informático se desarrolló empleando como herramienta principal el editor de código *Visual Studio Code (en un ordenador con sistema operativo Windows 10 de 64bits)* y el lenguaje de programación Python (Version: 3.9.4), adicionalmente otras librerías de desarrollo que dan soporte al lenguaje y que son necesarias para la ejecución:

- Numpy (Versión: 1.20.2) [7]
- Matplotlib (Version: 3.4.2) [8]
- Tkinter (Incluido por defecto en las librerias internas del lenguaje Python) [5]

Lo primero que se debe hacer después de incluir el interprete de Python en el ordenador que se ejecutará el programa es incluir las librerías utilizando el sistema de gestion de paquetes de Python[9] incluido en el interprete de Python al instalarlo, en la **ventana de commandos** de Windows se ejecuta el commando *pip install* y el nombre de los modulos mencionados anteriormente.

3.2.1 Pantalla inicial

Cuando inicia la ejecución del programa se empieza por mostrar al usuario la primera pantalla, donde están dispuestas las opciones de iniciar un nuevo análisis, que te dirigira a la pantalla de *Nuevo análisis*, cargar un análisis realizado anteriormente, que te dirigirá directamente a la pantalla de *Resultados.*

Figura 3-1. Pantalla inicial de Manther 4.0

Al seleccionar la opción de “Ayuda” el usuario encontrará un mensaje de presentación del programa, así como el formato de los archivos que admite en su opción de cargar los valores de un análisis ya realizado anteriormente, puesto que el programa al realizar un análisis nuevo creará un archivo que almacena los datos ingresados para cargarlos directamente desde la pantalla inicial sin tener que volver a insertarlos en la pantalla de *Nuevo análisis*.

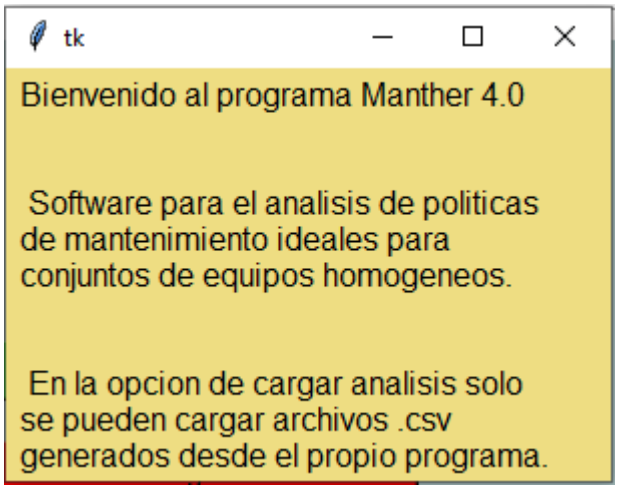

Figura 3-2 .Ventana de ayuda en pantalla inicial

3.2.2 Pantalla de Nuevo analisis

Cuando se elige la opción de iniciar un nuevo analisis desde la *pantalla inicial* el programa redirige al usuario a la pantalla de *Nuevo analisis*, desde aqui podrá ingresar en los campos correspondientes los datos, opciones de ayuda y calculo de la matriz de transicion.

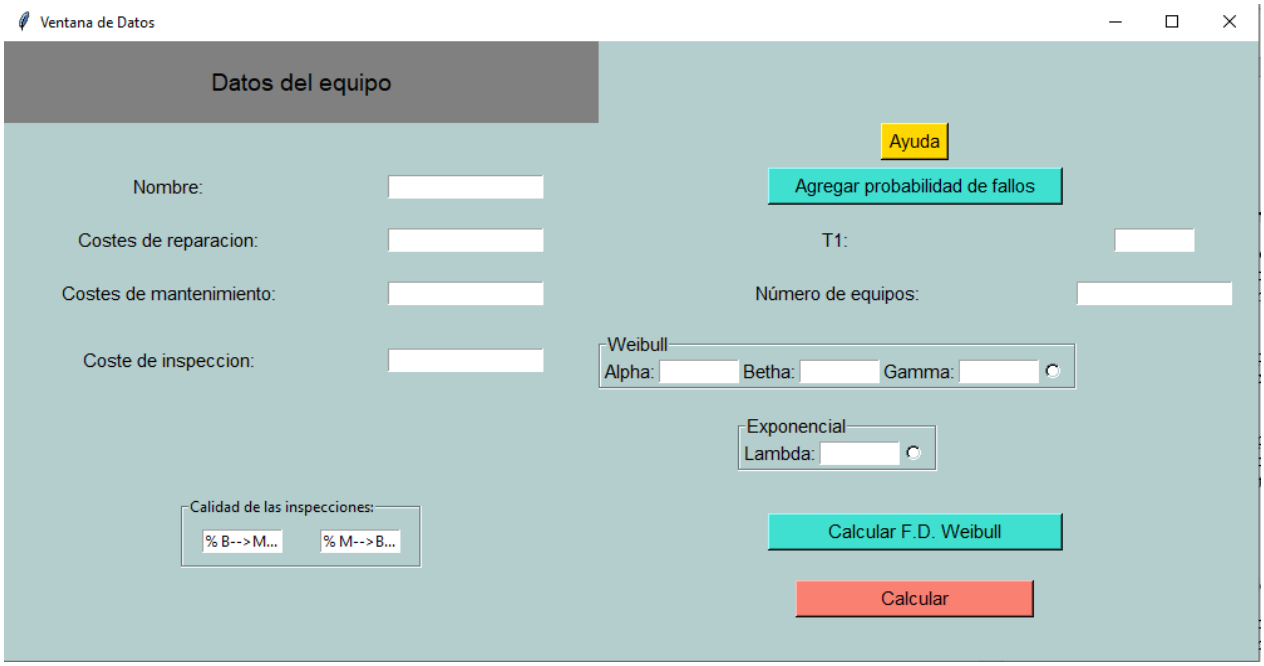

Figura 3-3. Pantalla de Nuevo análisis.

3.2.2.1 Capos de datos

En estos campos el usuario ingresa los datos asociados al sistema que va a analizarse según su caso:

- Nombre: que corresponderá al nombre que tendrá el archivo generado después de realizarse el análisis, con el cual el usuario podrá acceder nuevamente a los resultados obtenidos desde la opción de cargar análisis.
- Los costes de reparación y mantenimieno: deben incluir todos los gastos asociados a la operacion de reparacion (piezas de recambio, mano de obra, tiempo productivo perdido).
- Costes de las inspecciones y sus parámetros que medirán la efectividad de estas.
- T1: períodos continuos de funcionamiento ininterrumpidos a partir del cuál la probabilidad de fallo del equipo será del 100%.
- Número de equipos homogéneos que conforman el sistema estudiado.

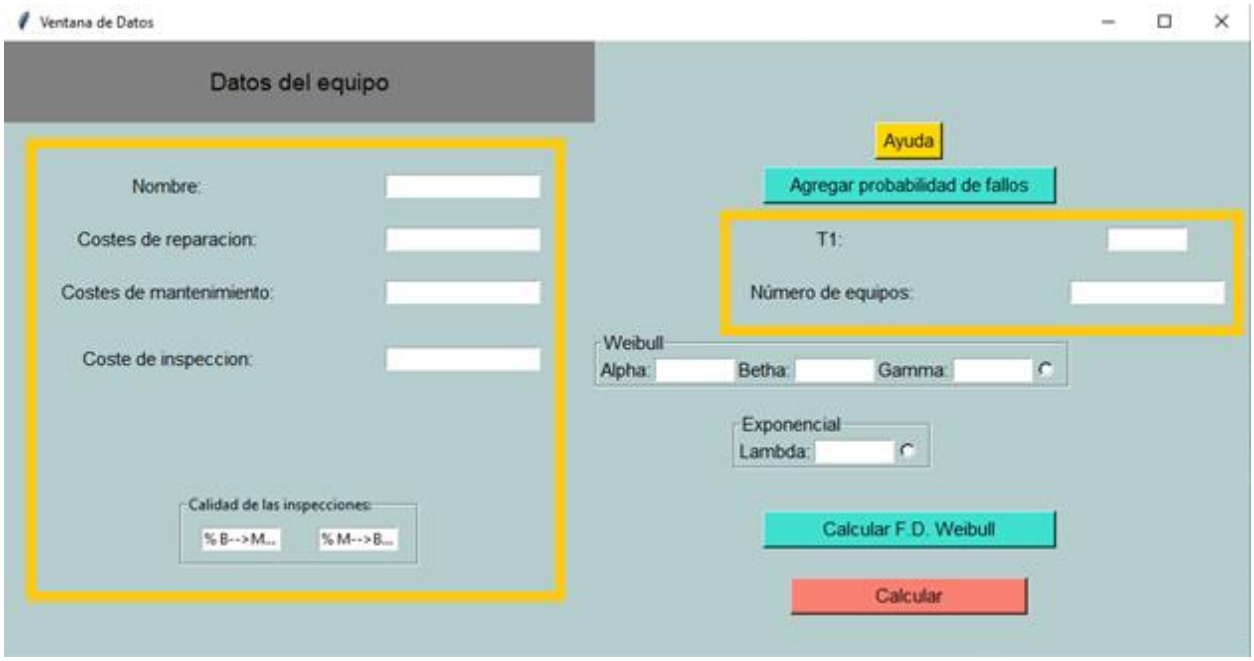

Figura 3-4. Campos de introduccion de datos del sistema a analizar.

3.2.2.2 Matriz de transición

El usuario dispone de un botón para ingresar los datos de la matriz de transición, cuando hace click sobre el botón de **Agregar probabilidad de fallos** se desplegará una ventana para examinar los archivos del disco local del ordenador, se carga un unico archivo que debe estar en formato .csv, con sus valores dispuestos como en el ejemplo del cuadro de "Ayuda" en la Figura 3-8.

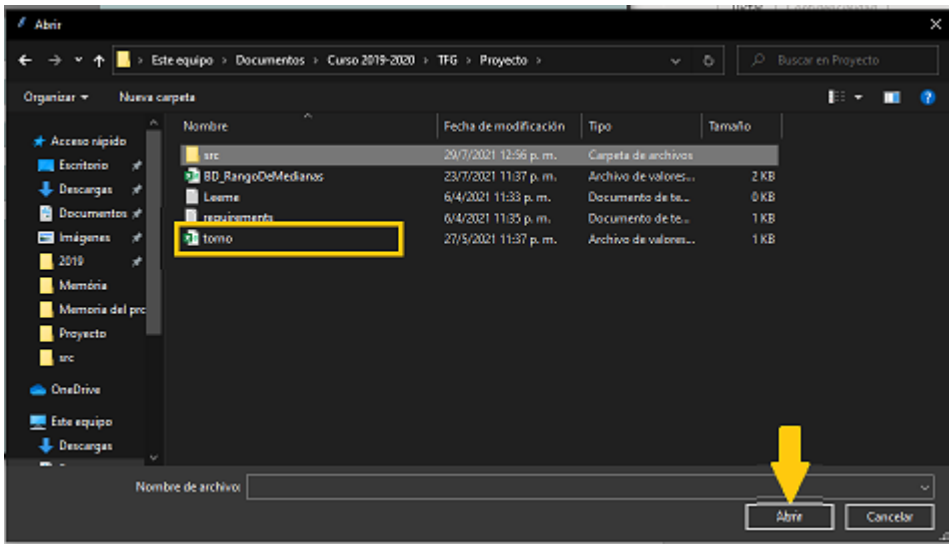

Figura 3-5. Ventana para agregar el archivo de probabilidad de fallos.

En otro caso, el usuario puede calcularla mediante 3 metodos distintos, a travez de 2 funciones de probabilidad, o por el metodo de Rango de Medianas.

3.2.2.2.1 Cálculo de matriz de transición a partir de funciones de distribución de probabilidad

El usuario indica utilizando los elementos de la pantalla “botones radiales” si la matriz de probabilidades será generada por el programa utilizando alguna de las funciones de distribución de probabilidad disponibles, a través del ingreso de sus parámetros en la que se vaya a utilizar.

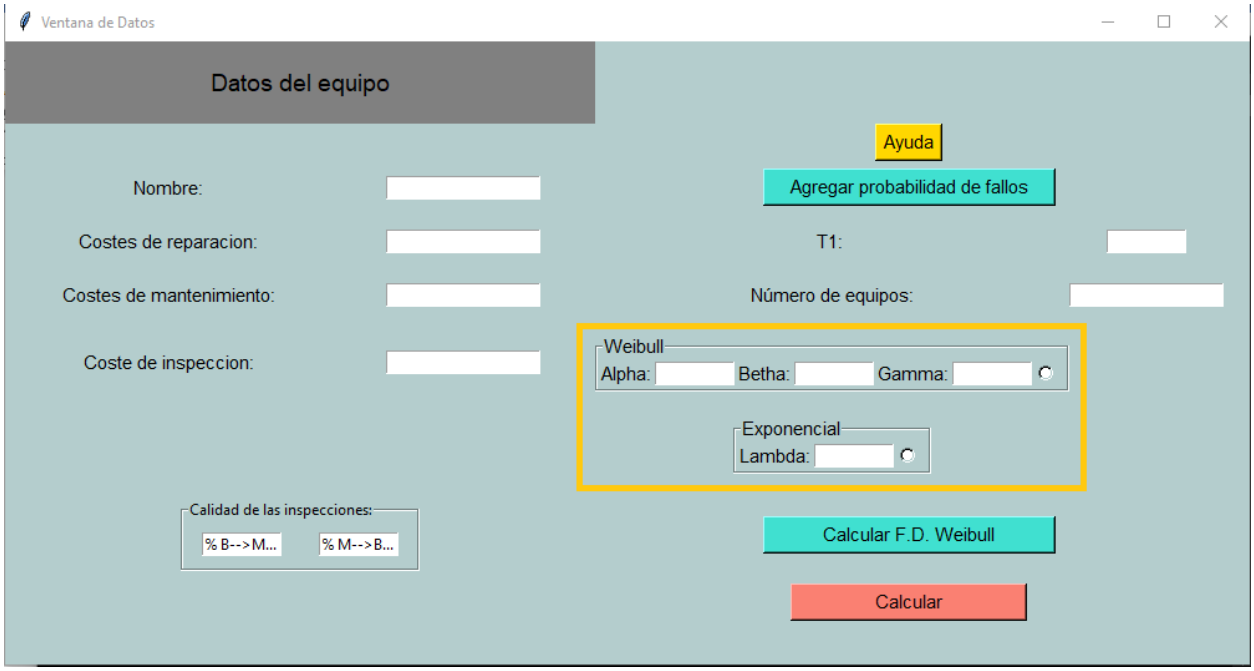

Figura 3-6. Campos para el calculo de la matriz de probabilidades utilizando una F.D.P Weibull o Exponencial.

3.2.2.2.2 Cálculo de matriz de transicion utilizando los tiempos de operacion entre fallos y mantenimiento de un equipo

Si el usuario dispone de una base de datos con distintas medidas del tiempo de operación entre fallos y mantenimieto del equipo puede generar la matriz de transición cargando dicha base de datos al clicar en el botón de "Calcular F.D. Weibull", el archivo a cargar deberá estar en formato .csv, y dispuesto como en el ejemplo mostrado en la Ventana de ayuda de esta pantalla (Figura3-8). El programa utilizará el metodo de **Rango de Medianas** para calcular de manera aproximada los valores de los parametros Alfa y Beta (forma y escala respectivamente) de una función de distribución Weibull utilizando las medidas de la base de datos cargada.

Al ser efectivo el cálculo de los parámetros de la función se desplegará un mensaje para el usuario mostrando los resultados, donde podrá ingresarlos en la sección de cálculo de la matriz de transición a partir de una función de distribución de probabilidad Weibull.

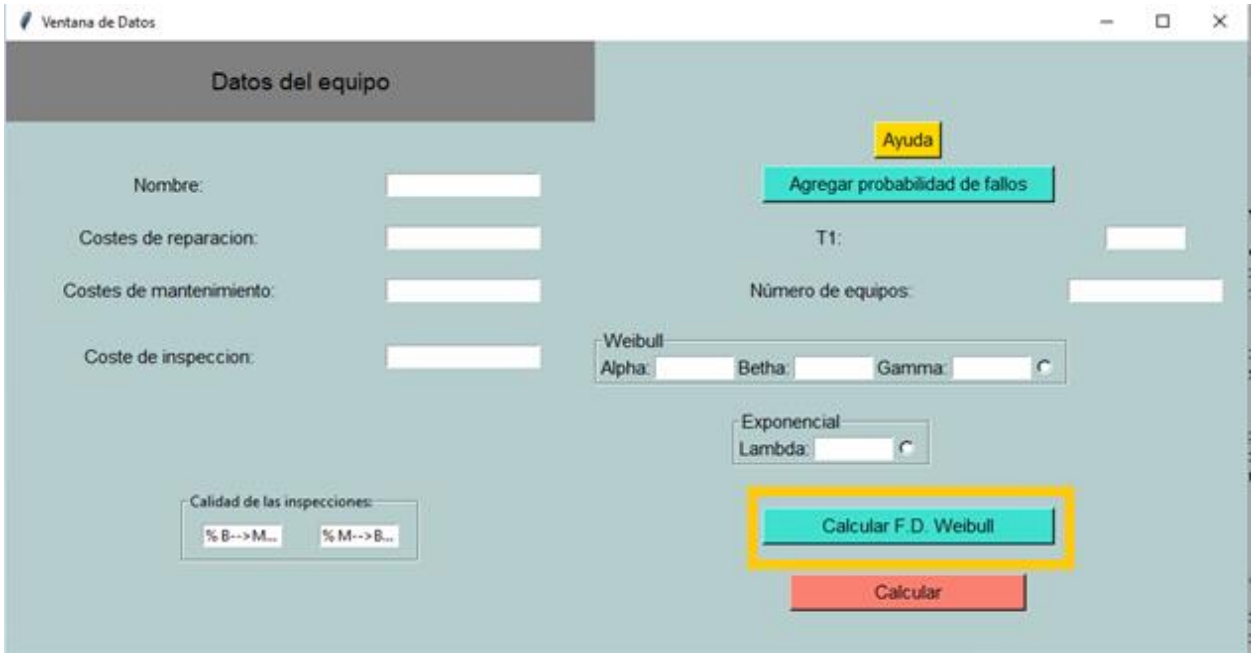

Figura 3-7. Cálculo de matriz de transición a partir de tiempo de operación entre fallos y mantenimiento.

Si el usuario clica en el botón "Ayuda" se desplegará una ventana emergente con información para el usuario de una explicación más profunda del ingreso de los datos y del manejo de esta ventana del programa.

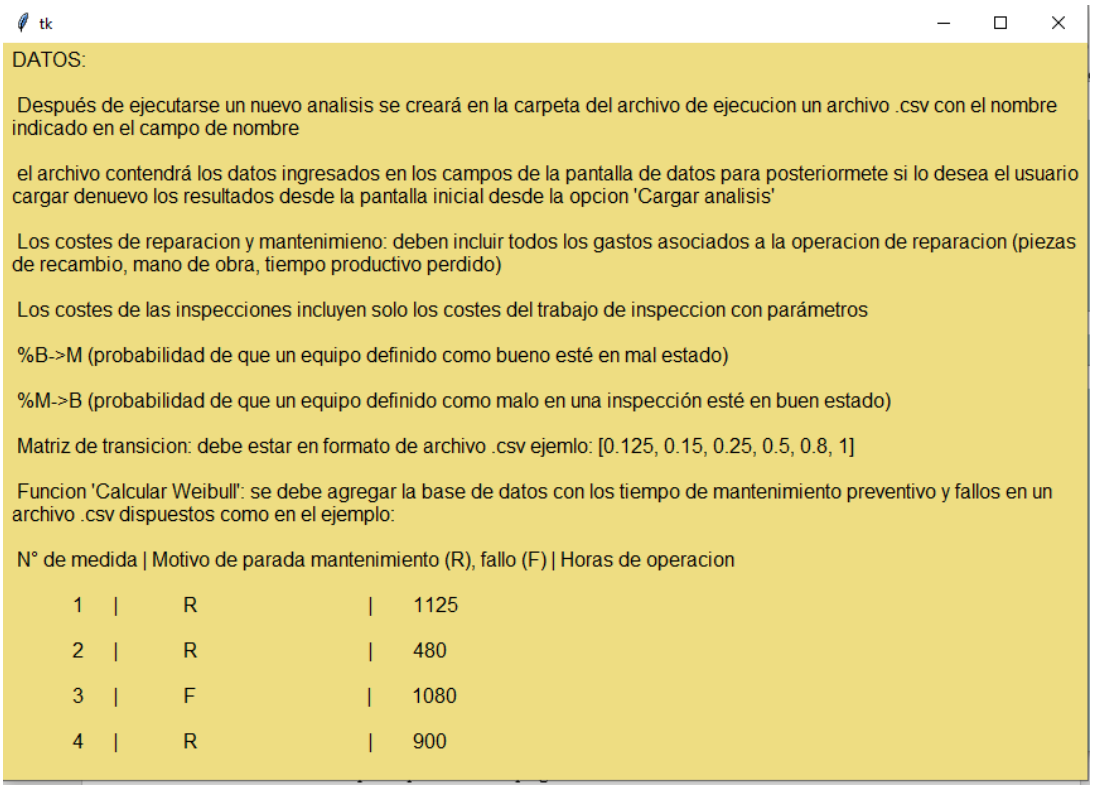

Figura 3-8. Ventana emergente de ayuda para el usuario, desplegada al clicar en el botón "Ayuda".

3.2.3 Pantalla de resultados

Cuando estén correctamente ingresados los campos de datos necesarios para llevar a cabo el análisis de los modelos y hacer click en el botón de "Calcular" se dirigirá al usuario a la *pantalla de resultados* donde se mostrará la solucion óptima que encontró el programa al resolver los modelos.

Las figuras mostradas en esta sección muestran los resultados de un análisis de ejemplo realizado a un "torno", con los datos de costes de mantenimiento, reparación, inspecciones, número de equipos y probabilidad de fallos obtenidos de otra investigación [10], los cuáles son los siguientes:

- Coste de reparación: 1000u.m
- Coste de mantenimiento preventivo: 300u.m
- Coste de inspecciones: 100u.m
- Calidad de las inspecciones:
    - Equipo "bueno" declarado como "malo" (%B→M): 0.5
    - Equipo "malo" delcarado como "Bueno" (%M→B): 0.5
- Probabilidad fallos: [0.025, 0.05, 0.055, 0.06, 0.065, 0.07, 0.085, 0.1, 0.125, 0.15, 0.175, 0.2, 0.35, 0.6, 1]
- T1: 15
- Número de equipos: 100

Figura 3-9. Pantalla de resultados

La pantalla destaca una sección separada para los resultados de cada modelo, en el caso de la política correctiva el coste total y el número de averías esperadas, y en el caso de las políticas preventivas se muestran el coste total de el caso óptimo, el número de averías esperadas para ese caso y el valor de las variables T2 y T3, también se tiene la posibilidad de generar una grafica que muestre los valores del costes y averías para cada valor de T2 y T3 clicando en su correspondiente botón.

Los valores óptimos mostrados de cada política son siguiendo el criterio de optimizacion del sistema planteado en los modelos, que es minimizar los costes independientemente del número de averías que presente el sistema.

3.2.3.1 Gráficas de averías en cada T

Para los casos de mantenimiento preventivo el usuario puede desplegar utilizando los botones corresponientes una gráfica en cada política para observar el número de fallos de equipos esperados para cada valor explorado de las variables tanto para T2 en el caso de preventivo total, como T2 y T3 en los otros dos modelos preventivos. El usuario es capaz también de poder explorar los valores de las curvas y superficies graficadas usando el mouse, que al moverlo a lo largo de la grafica puede verse las coordenadas del punto en el espacio sobre el cuál el esté situado.

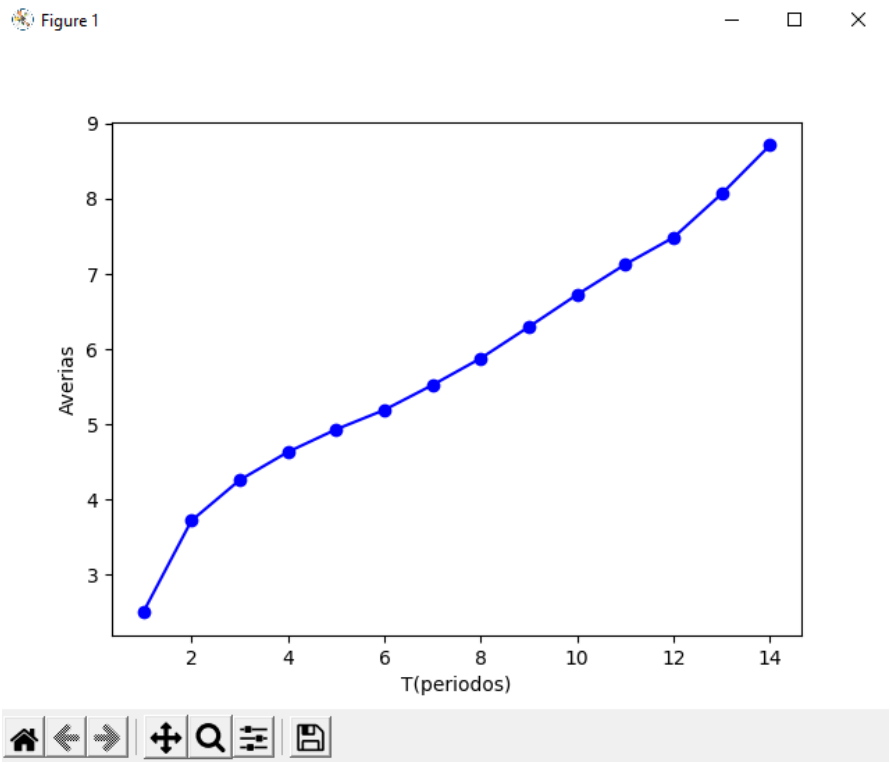

Figura 3-10. Gráfica de Averias vs T2 en mantenimiento preventivo total

La ventana de la grafica dispone de 7 botones (4 de ellos solo para uso de las graficas en 3D) con las funciones:

- se usa para volver a la vista inicial (solo para uso de grafica 3D).
- en una grafica 3D los botones flecha se usan para cambiar entre perspectivas anteriores tras rotar la grafica.
- habilita la opcion de ampliar o alejar la grafica al mover el mouse manteniendo presionado el boton derecho.
- realizar un acercamiento en un area seleccionada de la grafica.
- acceder al menu de configuración de manipulación de la gráfica mostrada.
- guardar la perspectiva actual de la gráfica como una imagen en formato PNG.

3.2.3.2 Grafica de costes en cada T

Al hacer click el botón de Costes en cada T según sea la política desplegará una gráfica que mostrará los costes totales del modelo en cada valor explorado de las variables T2 y T3.

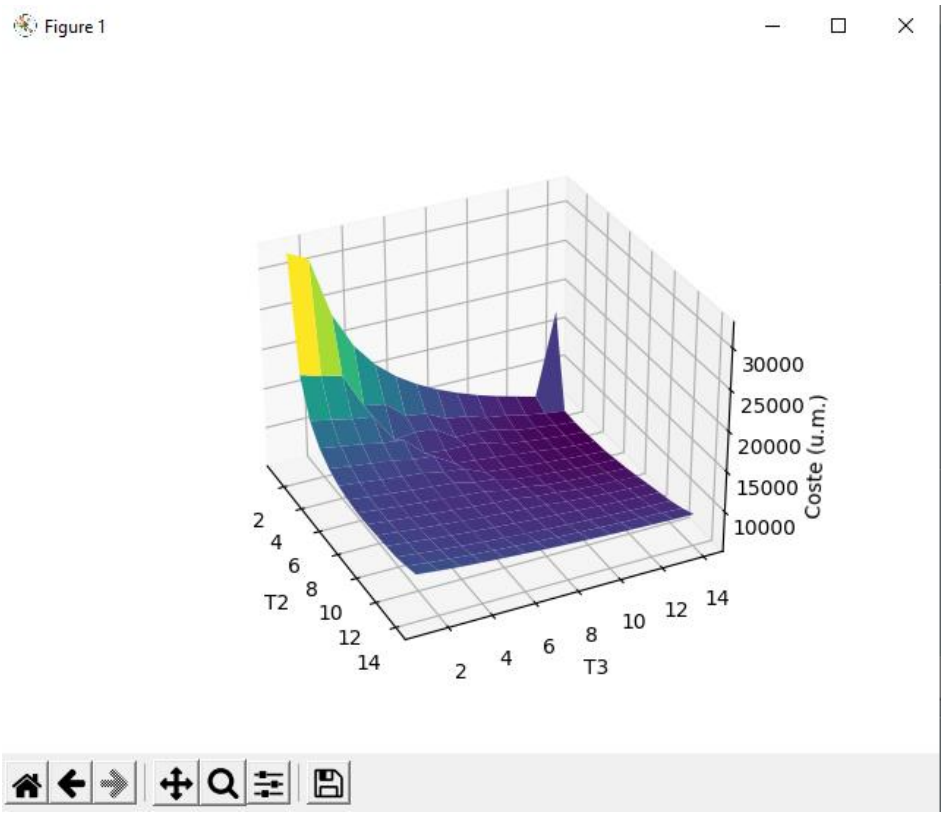

Figura 3-11. Gráfica de T2 y T3 vs Coste en mantenimiento prevenvtivo basado en tiempo ininterrumpido de funcionamiento sin fallos

3.2.4 Cargar un análisis realizado anteriormente

Según se explicó anteriormente, tras presionar el botón "Calcular" en la *pantalla nuevo análisis* el programa creará un nuevo fichero en la misma carpeta donde está el archivo fuente del programa con extension .csv, con nombre igual al nombre ingresado en su respectivo cámpo de la *pantalla de nuevo análisis*, donde almacenará los datos introducidos por el usuario permitiendo poder cargar y ejecutar de nuevo ese análisis desde la pantalla inicial.

En el archivo creado ordenará los datos según el formato de Valores Separador por Comas para que sean cargados correctamente, proceso que se explicará de forma más detallada en la siguiente sección.

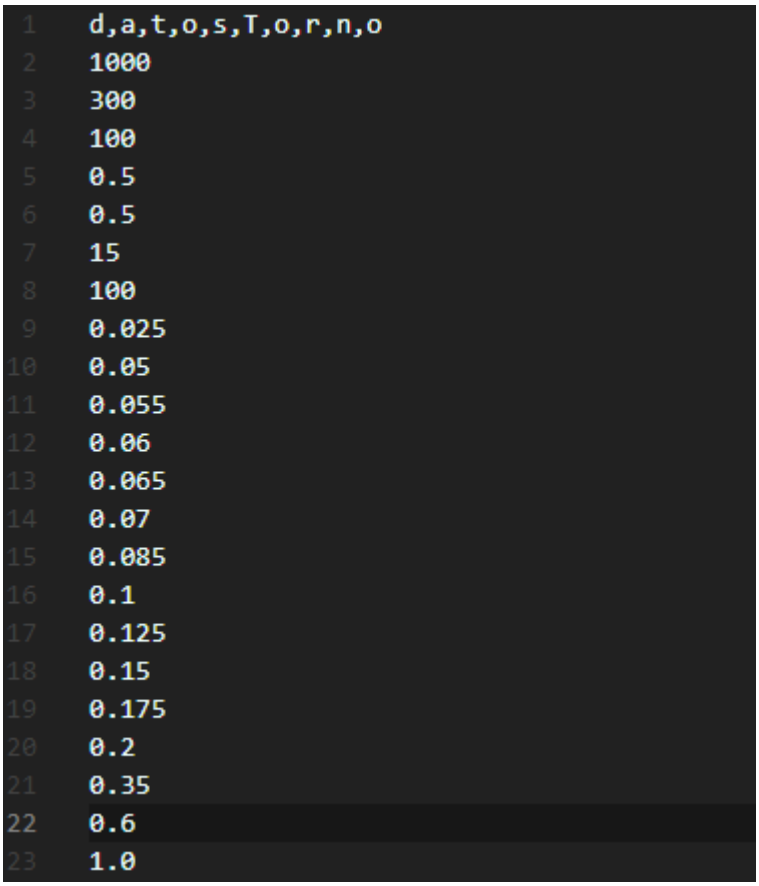

Figura 3-12. Ejemplo de archivo de datos creado por Manther tras ejecutar un análisis.

3.3 Codificación en Python de la solución desarrollada

En el siguiente epígrafe se explica de manera estructurada el código de programación en Python del desarrollo del programa, explicando a detalle las operaciones de ejecución e interacción entre cada elemento que lo conforma.

El software está descompuesto en diferentes módulos que representan cada funcionalidad del sistema desde interacción con el usuario hasta funciones de cálculo, estos utimos son llamados desde los módulos que generan la interaccion con el usuario y son considerados como el "cuerpo principal" del programa, en sí son 10 archivos que conforman el sistema informático desarrollado:

- Interfaz gráfica:
  - main.py
  - pDatos.py
  - pResultados.py
- Funcionalidades de cálculo:
  - cCorrectivo.py
  - cPreventivo1.py
  - cPreventivo2.py
  - cPreventivo3.py
  - cExponencial.py

- cWeibull.py
- cRangoMedianas.py

3.3.1 Desarrollo de la pantalla inicial

La interfaz de usuario se desarrolló como una instancia de la clase Tkinter de la librería de desarrollo con el mimso nombre, la ventana esta formada por una raiz que es la instancia de la clase y sobre la cual se le va a agregar un Frame sobre el cual se insertarán los elementos de interaccion con el usuario, widgets y figuras. Lo primero en el script del código es importar los archivos y librerías que se incluirán en los procesos del desarrollo:

```
1. import tkinter as tk
2. from tkinter import messagebox
3. from tkinter import font
4. from tkinter import filedialog
5.
6. from numpy import genfromtxt, reshape
7. import pDatos
8. import pResultados
```

main.py

Adicional a la inclusión de funcionalidades de las librerias de Desarrollo mencionadas antes, se incluyen también los scripts de *pDatos* y *pResultados* que serán necesarios en caso de llamar a las funciones para redirigir al usuario a la pantalla de datos o al cargar un análisis ya realizado dirigirlo directamente a la panalla de resultados.

La instancia de la aplicación se realiza en la "función main" del script, se crea la raíz de la ventana como una instancia de la clase Tkinter **Tk**, se configura las dimensiones y color de fondo de la ventana. Se creal el Frame que se agregará a la raíz de la ventana usando una clase de nombre MainWindow desarrollada en la investigación que heredará además funcionalidades de la clase **Frame** de la librería de Tkinter. Finalmente se pone la función *mainloop* para que la ventana se mantenga abierta.

```
1. if __name__ == "__main__":
2.     #instancia de app
3.     root = tk.Tk()
4.     root.geometry('700x500')
5.     root.configure(background = 'LightCyan3') #color de fondo
6.     app = MainWindow(root)
7.     app.mainloop()
```

main.py

3.3.1.1 Definicion de la clase MainWindow

La función de construcción de las ventanas gráficas se han encapsulado en una clase cada una, para tener una mejor estructura del código y poder hacerlo mas reutilizable. Especificamente el diseño del Frame de la ventana gráfica y todos sus elementos estan definidos en una clase MainWindow.

El método constructor de objeto instanciado de la clase *__init__*, que será la ventana grafica de la pantalla inicial y sus elementos, dispone de 4 elementos, de los cuales 3 son elementos botones de la clase **Button** de Tkinter que llaman cada uno a un método de la clase y un Label para mostrar a modo de presentacion el nombre del programa.

```
1. class MainWindow(tk.Frame):
2.
3.     #constructor de clase
4.     def __init__(self, parent):
5.         super().__init__(parent)
6.         self.parent = parent
7.         self.parent.title("Pagina inicial")
8.         fuente = font=('Arial', 11)
9.
10.         botonNuevoAnalisis = tk.Button(self.parent, text = "Nuevo anali
   sis", font=fuente, bg = "lime green",width = 25 ,command = self.pantall
   aDatos)
11.         botonNuevoAnalisis.place(x=250, y=150)
12.
13.         botonCargarAnalisis = tk.Button(self.parent, text="Cargar", fon
   t=fuente, bg="red", width=25, command = self.cargarAnalisis)
14.          botonCargarAnalisis.place(x=250, y=200)
15.
16.         labelTitulo = tk.Label(self.parent, font=('Arial', 26), text =
   'Manther 4.0', bg='LightCyan3')
17.          labelTitulo.place(x=275, y=50)
18.
19.         botonAyuda = tk.Button(self.parent, text = "Ayuda", font=fuente
   , bg = "salmon",width = 25 ,command = self.ayudaMessageBox)
20.         botonAyuda.place(x=250, y=250)
```

main.py

Cada botón de la ventana llama a una función definido dentro de la clase.

- Función pantallaDatos: del script desarrollado en el módulo *pDatos* se llama a su clase VentanaDatos para crear una instancia de ésta y dirigir la vista del usuario a su frame, esta clase para instanciarse al ser creada a partir de una clase **Frame** heredada debe recibir como parámetro el root o raíz de la ventana llamado 'parent' en la clase.

```
1. def pantallaDatos(self):
2.         #llamar a la pantalla de introduccion de datos
3.         pDatos.VentanaDatos(self.parent)
```

main.py

- Función cargarAnalisis: será el responsable de examinar en los archivos locales del ordenador del usuario, usando la función de Numpy *genfromtxt* que lee un archivo .csv y lo convierte en un array, al seleccionar el indicado cargar dicho array que contendrá los datos ordenados de un análisis pasado generado por el programa. Llama al módulo *pResultados* para hacer una instancia de la clase VentanaResultados que recibe cada dato obtenido de archivo .csv cargado y procede a dirigir la vista del usuario a ese frame.

```
1. def cargarAnalisis(self):
2.     #cargar datos de un analisis ya realizado y recalcularlo
3.     csvfile = genfromtxt(filedialog.askopenfilename(), delimiter='')
4.     pResultados.VentanaResultados(self.parent, csvfile[0], csvfile[1],
   csvfile[2],csvfile[3], csvfile[4], csvfile[5], csvfile[6], csvfile[7],
   csvfile[8:].reshape(int(csvfile[6]), 1))
5.     self.destroy()
```

main.py

- Función ayudaMessageBox: esta función llamada cuando se pulsa el botón "Ayuda" crea una instancia (root) de la clase **Tk** para desplegar una ventana emergente que utilizando la función *Message* de la librería Tkinter muestra el mensaje deseado con la configuración de fuente y fondo indicada. Al ser una ventana (root) nuevo éste debe tener su propio "mainloop".

```
1. def ayudaMessageBox(self):
2.      master = tk.Tk()
3.      mensaje = "Bienvenido al programa Manther 4.0 \t\n"\
4.          "\t\n Software para el analisis de politicas de mantenimiento
   ideales para conjuntos de equipos homogeneos.\t\n"\
5.          "\t\n En la opcion de cargar analisis solo se pueden cargar ar
   chivos .csv generados del propio programa."
6.      msj = tk.Message(master, text=mensaje)
7.      msj.config(font=('Arial', 12), bg='light goldenrod')
8.      msj.pack()
9.      tk.mainloop()
```

main.py

3.3.2 Desarrollo de la pantalla para ingresar los datos

Esta intefaz gráfica está conformada por numerosos elementos y funcionalidades para el ingreso de datos para los modelos, así como la llamada a varios de los módulos de cálculo del programa. Las librerías de desarrollo y módulos del programa utilizados en este script son:

```
1. import tkinter as tk
2. from tkinter import filedialog
3. from tkinter import font
4. import csv
5. from numpy import dtype, float64, genfromtxt
6. import pResultados
7. import cWeibull
8. import cExponencial
9. import cRangoMedianas
```

pDatos.py

3.3.2.1 Definicion de la clase VentanaDatos

En la creación de una nueva ventana, en este caso la ventana de datos proveniente de una ventana "padre" esta solo puede ser instanciada para el mismo root usando la clase de Tkinter **Toplevel**, la clase que definirá los elementos de la ventana de datos se creará por herencia de esta clase, a la cual debe pasarsele el parámetro root, que dentro de las clases encapsuladas se llamó "parent".

Igual que antes en el método constructor de la clase *__init__* se definió las dimensiones y color de fondo de la ventana así como variables que utiliza y widgets. La función *withdraw* ocultará la ventana anterior.

```
1. class VentanaDatos(tk.Toplevel):
2.     def __init__(self, parent):
3.         super().__init__(parent)
4.         self.parent = parent
5.         self.title("Ventana de Datos")
6.         self.geometry('1000x500')
7.         fondo = 'LightCyan3'
8.         self.configure(bg=fondo)
9.         fuente = font = ('Arial', 11)
10.          self.parent.withdraw()
```

pDatos.py

- Declaración de las variables que utilizará la clase:

```
1.         #variables del modelo
2.         self.nombre = tk.StringVar(self)
3.         self.costeRep = tk.StringVar(self)
4.         self.costeMant = tk.StringVar(self)
5.         self.costeInsp = tk.StringVar(self)
6.         self.prob = tk.StringVar(self)
7.         self.insp_B_M = tk.StringVar(self)
8.         self.insp_M_B = tk.StringVar(self)
9.         self.T1 = tk.StringVar(self)
10.          self.N = tk.StringVar(self)
11.          self.alpha = tk.StringVar(self)
12.          self.betha = tk.StringVar(self)
13.          self.k = tk.StringVar(self)
14.          self.L = tk.StringVar(self)
15.          self.opcion = tk.IntVar(self)
```

pDatos.py

- Labels de texto y elementos para ingresar los datos: se emplea la clase **Label** para instanciar los espacios de texto, y la clase **Entry** para los espacios para ingresar los datos.

```
1. #Labels y Entradas
2. Label = tk.Label(self, font=('Arial', 14), bg = 'gray', text = 'Datos d
   el equipo', padx = 15, pady = 20, width = 40)
3. Label.grid(row = 0, column = 0, columnspan = 2)
4.
5. LabelNombre = tk.Label(self, font=fuente, text = 'Nombre:', bg=fondo)
6. LabelNombre.grid(row = 2, column = 0, pady = 10)
7. nombre = tk.Entry(self, textvariable = self.nombre)
8. nombre.grid(row = 2, column = 1, pady = 10)
9.
10.  LabelReparacion = tk.Label(self, font=fuente, text = 'Costes de repara
   cion:', bg=fondo)
11.  LabelReparacion.grid(row = 4, column = 0, pady = 10)
12.  C1 = tk.Entry(self, textvariable = self.costeRep)
13.  C1.grid(row = 4, column = 1, pady = 10)
14.
```

```
15.  LabelMantenimiento = tk.Label(self, font=fuente, text = 'Costes de man
   tenimiento:', bg=fondo)
16.  LabelMantenimiento.grid(row = 6, column = 0, pady = 10)
17.  C2 = tk.Entry(self, textvariable = self.costeMant)
18.  C2.grid(row = 6, column = 1, pady = 10)
19.
20.  LabelInspeccion = tk.Label(self, font=fuente, text = 'Coste de inspecc
   ion:', bg=fondo)
21.  LabelInspeccion.grid(row = 8, column = 0)
22.  C3 = tk.Entry(self, textvariable = self.costeInsp)
23.  C3.grid(row = 8, column = 1, pady = 10)
24.
25.
26.  lf = tk.LabelFrame(self, text = 'Calidad de las inspecciones:', bg=fon
   do)
27.  lf.grid(row = 10, column = 0, columnspan = 2, pady = 10)
28.  self.insp_B_M.set("% B-->M...")
29.  self.insp_M_B.set("% M-->B...")
30.  B_M = tk.Entry(lf, textvariable = self.insp_B_M, width=10)
31.  B_M.grid(row = 0, column = 0, padx = 15, pady = 10)
32.  M_B = tk.Entry(lf, textvariable = self.insp_M_B, width=10)
33.  M_B.grid(row = 0, column = 1, padx = 15, pady = 10)
34.  LabelT1 = tk.Label(self, font=fuente, text = 'T1: ', bg=fondo)
35.  LabelT1.grid(row = 4, column = 4)
36.  T1 = tk.Entry(self, textvariable = self.T1, width=10)
37.  T1.grid(row = 4, column = 5)
38.
39.  LabelN = tk.Label(self, font=fuente, text = 'Número de equipos:', bg=f
   ondo)
40.  LabelN.grid(row = 6, column = 4)
41.  N = tk.Entry(self, textvariable = self.N)
42.  N.grid(row = 6, column = 5)
```

pDatos.py

- Espacios para ingresar parámetros de Exponencial o Weibull: en esta sección el usuario si desea generar la matriz de transición utilizando una función de probabilidad exponencial o Weibull, ingresando sus parámetros conocidos, seleccionando con los botones radiales la opción que desea usar. Para esto se usa el elemento *Radiobutton* de Tkinter asignando a la variable de la clase "opcion" el valor de 1 en caso de seleccionar la opción de la funcion Weibull y 2 en caso de seleccionar la opción de calcularla por una Exponencial.

```
1.  lf_weibull = tk.LabelFrame(self, text='Weibull', font=fuente, bg=fondo)
2.  lf_weibull.grid(row=8, column=4, pady=10)
3.  Alpha_Label = tk.Label(lf_weibull, font=fuente, text='Alpha:', bg=fondo
   )
4.  Alpha_Label.grid(row=0, column=0)
5.  A = tk.Entry(lf_weibull, textvariable= self.alpha, width=10)
6.  A.grid(row=0, column=1)
7.  Betha_Label = tk.Label(lf_weibull, font=fuente, text='Betha:', bg=fondo
   )
8.  Betha_Label.grid(row=0, column=2)
9.  B = tk.Entry(lf_weibull, textvariable= self.betha, width=10)
10.  B.grid(row=0, column=3)
11.  K_Label = tk.Label(lf_weibull, font=fuente, text='Gamma:', bg=fondo)
12.  K_Label.grid(row=0, column=4)
13.  k = tk.Entry(lf_weibull, textvariable= self.k, width=10)
14.  k.grid(row=0, column= 5)
```

```
15.  botonWeibull = tk.Radiobutton(lf_weibull, variable= self.opcion, value
   =1, bg=fondo)
16.  botonWeibull.grid(row=0, column=6)
17.
18.  lf_exponencial = tk.LabelFrame(self, text='Exponencial',font=fuente, b
   g=fondo)
19.  lf_exponencial.grid(row=9, column=4, pady=10)
20.  L_Label = tk.Label(lf_exponencial, font=fuente, text='Lambda:', bg=fon
   do)
21.  L_Label.grid(row=0, column=0)
22.  L = tk.Entry(lf_exponencial, textvariable= self.L, width=10)
23.  L.grid(row=0, column=1)
24.  botonExponencial = tk.Radiobutton(lf_exponencial, variable= self.opcio
   n, value=2, bg=fondo)
25.  botonExponencial.grid(row=0, column=2)
```

pDatos.py

- Botones de Calcular, Agregar probabilidad de fallos, Ayuda y Calcular F.D. Weibull: se utiliza en estos casos la clase de Tkinter **Button** que en cada caso ejecutará el metodo indicado en su atributo "command".

```
1. botonAyuda = tk.Button(self, font=fuente, text='Ayuda', bg='gold',comma
   nd= self.ayudaMessageBox)
2. botonAyuda.grid(row = 1, column = 4, columnspan=2)
3.
4. botonProb = tk.Button(self, font=fuente, text = 'Agregar probabilidad d
   e fallos', width = 25, command = self.incluirProb, bg='turquoise')
5. botonProb.grid(row = 2, column = 4, columnspan=2)
6.
7. botonCalWeibull = tk.Button(self, font=fuente, text='Calcular F.D. Weib
   ull', width=25, command= self.RangoDeMedianas, bg='turquoise')
8. botonCalWeibull.grid(row = 10, column = 4, columnspan=2)
9.
10.  botonCalcular = tk.Button(self, font=fuente, text = 'Calcular',width =
    20, command = self.calcular, bg = 'salmon')
11.  botonCalcular.grid(row = 11, column = 4, columnspan=2)
```

pDatos.py

- Función Calcular: este método de la clase **VentanaDatos** está compuesto por varios procesos, primero chequear si el usuario ha seleccionado la opción de calcular la matriz de transición usando una función de probabilidad Exponencial o Weibull para entonces proceder a llamar a sus funciones de cálculo en su caso. Seguidamente procede a llamar al módulo *pResultados* para crear una instancia de su clase **VentanaResultados** pasandole los parámetros que ésta requiere, que son los datos ingresados por el usuario y la matriz de probabilidades de transición.

  Finalmente se procede a crear el archivo .csv almacenando en el orden mostrado en la muestra de código de abajo los valores ingresados por el usuario y por último en forma de array la matriz de transición.

```
1.   def calcular(self):
2.
3.           if(self.opcion.get()==1):
4.               self.calWeibull()
5.
6.           if(self.opcion.get()==2):
```

```
            self.calExponencial()

        pResultados.VentanaResultados(self.parent, self.nombre.get(), s
elf.costeRep.get(), self.costeMant.get(),
                  self.costeInsp.get(),
                  self.insp_B_M.get(),
                  self.insp_M_B.get(),
                  self.T1.get(),
                  self.N.get(), self.prob)

        with open(self.nombre.get()+'.csv', 'w') as file:
            writer = csv.writer(file)
            writer.writerow(self.nombre.get())
            writer.writerow([self.costeRep.get()])
            writer.writerow([self.costeMant.get()])
            writer.writerow([self.costeInsp.get()])
            writer.writerow([self.insp_B_M.get()])
            writer.writerow([self.insp_M_B.get()])
            writer.writerow([self.T1.get()])
            writer.writerow([self.N.get()])
            for i in range(0, int(self.T1.get())):
                writer.writerow([self.prob.item(i)])

        self.destroy()
```

pDatos.py

- Función incluirProb: cuando el usuario desee ingresar la matriz de probabilidades de un archivo .csv en un formato igual al mostrado en el epígrafe 3.2.3 en probabilidad de fallos, esta función desplegará una ventana para examinar los archivos locales del ordenador del usuario usando la función de Numpy *genfromtxt* que convierte ese archivo .csv en un array de Numpy y lo asigna a la variable "prob" de la clase **VentanaDatos**.

```
def incluirProb(self):
      cvsfile = genfromtxt(filedialog.askopenfilename(), delimiter=',')
        self.prob = cvsfile
```

pDatos.py

- Función calWeibull y calExponencial: ambos métodos llaman a los módulos cWeibull y cExponencial en su caso donde, se llama a su función *calcula* y pasandoles los parámetros que requieren, estos métodos retornan la matriz de transición como un array unidimensional de Numpy y lo asignan a la variable de la clase **VentanaDatos** "prob".

```
def calWeibull(self):
    self.prob = cWeibull.calcula(self.alpha.get(), self.betha.get(), sel
f.k.get(), self.T1.get())

def calExponencial(self):
    self.prob = cExponencial.calcula(self.L.get(), self.T1.get())
```

pDatos.py

- Función RangoDeMedianas: este método utiliza la función de Numpy *genfromtxt* para buscar en los

archivos locales la base de datos en el formato solicitado para realizar el cálculo de los parámetros de la función Weibull llamando a la función *calcular* del módulo cRangoMedianas, éste retornará los parámetros de forma y escala de una función de probabilidad Weibull, para después llamar a la función *weubullMessageBox* y desplegar en una ventana nueva los valores calculados.

```
1. def RangoDeMedianas(self):
2.    cvsfile = genfromtxt(filedialog.askopenfilename(), delimiter=',', dt
   ype='str')
3.    self.alpha, self.betha = cRangoMedianas.calcular(cvsfile)
4.    self.weibullMessageBox(self.alpha, self.betha)
```

pDatos.py

- Función weibullMessageBox: despliega, similar a la función ayudaMessageBox en el epígrafe 3.3.1.1 una ventana nueva donde mostrará al usuario los valores de forma y escala de la función Weibull calculados en Rango de Medianas.

```
1.     def weibullMessageBox(self, alpha, betha):
2.         master = tk.Tk()
3.         mensaje = "Parametros de FD Weibull calculados con exito:\t\n"\
4.             "\t\n Alpha: "+str(alpha)+"\t\n"\
5.             "\t\n Betha: "+str(betha)+"\t\n"
6.         msj = tk.Message(master, text=mensaje)
7.         msj.config(font=('Arial', 12), bg='light goldenrod')
8.         msj.pack()
9.         tk.mainloop()
```

pDatos.py

- Función ayudaMessageBox: despliega una ventana con información necesaria para el usuario, desarrollada similarmente que en la función weibullMessageBox.

```
1. def ayudaMessageBox(self):
2.     master = tk.Tk()
3.     mensajeAyuda = "DATOS: \t\n"\
4.         "\t\n Después de ejecutarse un nuevo analisis se creará en la c
   arpeta del archivo de ejecucion un archivo .csv con el nombre indicado
   en el campo de nombre\t\n"\
5.         "\t\n el archivo contendrá los datos ingresados en los campos d
   e la pantalla de datos para posteriormete si lo desea el usuario cargar
    denuevo los resultados desde la pantalla inicial desde la opcion 'Carg
   ar analisis'\t\n"\
6.         "\t\n Los costes de reparacion y mantenimieno: deben incluir to
   dos los gastos asociados a la operacion de reparacion (piezas de recamb
   io, mano de obra, tiempo productivo perdido)\t\n"\
7.         "\t\n Los costes de las inspecciones incluyen solo los costes d
   el trabajo de inspeccion con parámetros\t\n"\
8.         "\t\n %B>M (probabilidad de que un equipo definido como bueno e
   sté en mal estado)\t\n"\
9.        "\t\n %M>B (probabilidad de que un equipo definido como malo en
   una inspección esté en buen estado)\t\n"\
10.          "\t\n Matriz de transicion: debe estar en formato de archivo .
   csv ejemlo: [0.125, 0.15, 0.25, 0.5, 0.8, 1] \t\n"\
```

```
11.          "\t\n Funcion 'Calcular Weibull': se debe agregar la base de d
   atos con los tiempo de mantenimiento preventivo y fallos en un archivo
   .csv dispuestos como en el ejemplo: \t\n"\
12.          "\t\n N° de medida | Motivo de parada mantenimiento (R), fallo
    (F) | Horas de operacion \t\n"\
13.          "\t\n            1      |            R    |        1125
     \t\n"\
14.          "\t\n            2      |            R    |        480
     \t\n"\
15.          "\t\n            3      |            F    |        1080
     \t\n"\
16.          "\t\n            4      |            R    |        900
     \n"
17.
18.      msj = tk.Message(master, text=mensajeAyuda)
19.      msj.config(font=('Arial', 12), bg='light goldenrod')
20.      msj.pack()
21.      tk.mainloop()
```

pDatos.py

3.3.3 Desarrollo de la pantalla de resultados

Esta es la Ventana donde el usuario podrá observar los resultados de la resolución de los modelos, así como poder analizar y comparár otras soluciones posibles para los casos de mantenimiento preventivo. El funcionamiento de el script usado para ésta ventana a parte de funcionalidades de interacción con el usuario son el llamar a los módulos de cálculo de los modelo cuantitativos pasandoles los datos insertados por el usuarios en la ventana de datos por lo que se incluyen en el script los módulos que contienen estas funciones de cálculo, dispone a su vez de botones para el despliegue de gráficas en los modelos de mantenimiento preventivo.

Para el desarrollo y ejecución de esta parte del programa se incluyen las librerías de desarrollo y módulos:

```
1. import tkinter as tk
2. from tkinter import ttk
3. from tkinter import font
4. from tkinter.constants import HORIZONTAL
5. import matplotlib.pyplot as plt
6. from mpl_toolkits.mplot3d import Axes3D
7.
8. import cCorrectivo
9. import cPreventivo1
10.  import cPreventivo2
11.  import cPreventivo3
```

pResultados.py

3.3.3.1 Definicion de la clase VentanaResultados

Similar al proceso de instanciar un nuevo Frame utilizando la clase de Tkinter **Toplevel** explicado anteriormente en el epigrafe 3.3.2.1, con la diferencia de que ésta instancia recibe todos los datos de costes, probabilidades ingresados por el usuario en la ventana anterior.

```
1. class VentanaResultados(tk.Toplevel):
2.     def __init__(self, parent, nombre, C1, C2, C3, B_M, M_B, T1, N, p):
3.         super().__init__(parent)
4.         self.parent = parent
```

```
5.          self.title("Ventana de Resultados")
6.          self.geometry('1050x700')
7.          fondo = 'LightCyan3'
8.          self.configure(bg=fondo)
9.          fuente = font = ('Arial', 11)
10.
11.           self.parent.withdraw()
```

pResultados.py

- Declaración de las variables que utiliza la clase:

```
1.          self.C1 = int(C1)
2.          self.C2 = int(C2)
3.          self.C3 = int(C3)
4.          self.B_M = float(B_M)
5.          self.M_B = float(M_B)
6.          self.T1 = int(T1)
7.          self.N = int(N)
8.          self.p = p
```

pResultados.py

- Ejecución de los métodos de clase correspondientes a los 4 modelos cuantitativos: se llaman a los métodos de clase encargados de llamar a los módulos de cálculo encargados de resolver los modelos, pasandoles como parámetros los datos necesarios en cada modelo, y estos métodos retornan y almacenan en distintas variables los resultados de los cálculos para mostrarlos más adelante en sus correspondientes elementos Label de la interfaz gráfica.

```
1. costeCorr, averiasCorr = self.calcularCor(self.C1, self.T1, self.N, sel
   f.p)
2.
3. costePrev1, T2Prev1, averiasPrev1, graf_averias_Prev1, graf_TiempoCoste
   _Prev1 = self.calcularPre(self.C1, self.C2, self.T1, self.N, self.p)
4.
5. costePrev2, T2Prev2, T3Prev2, averiasPrev2, graf_TiempoCoste_Prev2, gra
   f_averias_Prev2 = self.calcularPre2(self.C1, self.C2, self.T1, self.N,
   self.p)
6.
7. costePrev3, T2Prev3, T3Prev3, averiasPrev3, graf_TiempoCoste_Prev3, gra
   f_averias_Prev3 = self.calcularPre3(self.C1, self.C2, self.C3, self.T1,
    self.N, self.p, self.B_M, self.M_B)
8.
```

pResultados.py

- Sección de resultados política de mantenimiento correctivo: en esta sección se mostrará el resultado del modelo de política de mantenimiento correctivo utilizando dos elementos Label de la clase de Tkinter **Label**. Seguido se crea un elemento separador para dividir el espacio entre el área donde se muestran los resultados de políticas distintas, utilizando la clase de Tkinter **Separator**.

```
1. Label = tk.Label(self, bg = 'gray', text = 'Resultados', font=('Arial',
    14), padx = 15, pady = 20, width=40)
2. Label.grid(row = 0, column = 0, columnspan=1)
3.
4. labelFCorrectivo = tk.Label(self, text = 'Correctivo', bg='gold', font=
   ('Arial', 12))
5. labelFCorrectivo.grid(row=2, column=0)
6. labelCorrCost = tk.Label(self, font=fuente, text='Coste total: '+ str(c
   osteCorr), bg=fondo)
7. labelCorrCost.grid(row=3, column=1, pady=10)
8. labelCorrAveria = tk.Label(self, font=fuente, text='N° Averias: '+ str(
   averiasCorr), bg=fondo)
9. labelCorrAveria.grid(row=4, column=1, pady=10)
10.
11.  s = ttk.Separator(self, orient=HORIZONTAL)
12.  s.grid(row=5, column=0, columnspan=99, sticky=(tk.W, tk.E))
```

pResultados.py

- Secciónes de resultados políticas de mantenimiento preventivo: similar al caso anterior se emplean elementos de la clase **Label** de Tkinter para mostrar los resultados de la resolución del modelo para la política en cuestión y adicionalmente se agregan dos elementos de tipo **Button** que al clicar sobre ellos llamarán a los métodos de la clase VentanaResultados *graficaAverias_prev* y *graficaTC_prev*. De igual manera cada sección individual correspondiente a una política de mantenimiento se separa utilizando un elemento separador de la clase **Separator** de Tkinter.

```
1. LabelFPrev1 = tk.Label(self, text='Preventivo Total',bg='gold', font=('
   Arial', 12))
2. LabelFPrev1.grid(row=6, column=0)
3. labelPrev1Cost = tk.Label(self, font=fuente, text='Coste total: '+ str(
   costePrev1), bg=fondo)
4. labelPrev1Cost.grid(row=7, column=1, pady=10)
5. labelPrev1Averia = tk.Label(self, font=fuente, text='N° Averias: '+ str
   (averiasPrev1), bg=fondo)
6. labelPrev1Averia.grid(row=8, column=1, pady=10)
7. LabelPrev1_T2= tk.Label(self, font=fuente, text='T2:'+ str(T2Prev1), bg
   =fondo)
8. LabelPrev1_T2.grid(row=7, column=0, pady=10)
9.
10.          #boton de la grafica
11.  botonGraficaAverias_prev1 = tk.Button(self, font = fuente, text='Grafi
   ca Averias en cada T', command= lambda: self.graficaAverias_prev1(graf_
   averias_Prev1))
12.  botonGraficaAverias_prev1.grid(row=7, column=2, pady=10)
13.
14.  botonGraficaTC_prev1 = tk.Button(self, font=fuente, text='Grafica de T
   2 vs Coste', command= lambda: self.graficaTC_prev1(graf_TiempoCoste_Pre
   v1))
15.  botonGraficaTC_prev1.grid(row=8, column=2, pady=10)
16.
17.  s1 = ttk.Separator(self, orient=HORIZONTAL)
18.  s1.grid(row=9, column = 0, columnspan=99, sticky=(tk.W, tk.E))
19.
20.  LabelFPrev2 = tk.Label(self, text='Preventivo basado en tiempo de func
   ionamiento sin fallos', bg='gold', font=('Arial', 12))
21.  LabelFPrev2.grid(row=10,  column=0)
22.
23.  labelPrev2Cost = tk.Label(self, font=fuente, text='Coste total: '+ str
   (costePrev2), bg=fondo)
```

```
24.  labelPrev2Cost.grid(row=11, column=0, pady=10)
25.
26.  labelPrev2Averia = tk.Label(self, font=fuente, text='N° Averias: '+ st
   r(averiasPrev2), bg=fondo)
27.  labelPrev2Averia.grid(row=11, column=1, pady=10)
28.
29.  LabelPrev2_T2= tk.Label(self, font=fuente, text='T2:'+ str(T2Prev2), b
   g=fondo)
30.  LabelPrev2_T2.grid(row=12, column=1, pady=10)
31.
32.  LabelPrev2_T3= tk.Label(self, font=fuente, text='T3:'+ str(T3Prev2), b
   g=fondo)
33.  LabelPrev2_T3.grid(row=13, column=1, pady=10)
34.
35.          #boton de la grafica
36.  botonGraficaAverias_prev2 = tk.Button(self, font=fuente, text='Grafica
    Averias en cada T', command= lambda: self.graficaAverias_prev2(graf_av
   erias_Prev2))
37.  botonGraficaAverias_prev2.grid(row=12, column=2, pady=10)
38.  botonGraficaTC_prev2 = tk.Button(self, font=fuente, text='Grafica de T
   2 vs T3 vs Coste', command= lambda: self.graficaTC_prev2(graf_TiempoCos
   te_Prev2))
39.  botonGraficaTC_prev2.grid(row=13, column=2, pady=10)
40.
41.  s2 = ttk.Separator(self, orient=HORIZONTAL)
42.  s2.grid(row=14, column=0, columnspan=99, sticky=(tk.W, tk.E))
```

pResultados.py

```
1. LabelFPrev3 = tk.Label(self, text='Preventivo basado en tiempo de func
   + Insp', bg='gold', font=('Arial', 12))
2. LabelFPrev3.grid(row=15,  column=0)
3.
4. labelPrev3Cost = tk.Label(self, font=fuente, text='Coste total: '+ str(
   costePrev3), bg=fondo)
5. labelPrev3Cost.grid(row=16, column=0, pady=10)
6.
7. labelPrev3Averia = tk.Label(self, font=fuente, text='N° Averias: '+ str
   (averiasPrev3), bg=fondo)
8. labelPrev3Averia.grid(row=16, column=1, pady=10)
9.
10.  LabelPrev3_T2= tk.Label(self, font=fuente, text='T2:'+ str(T2Prev3), b
   g=fondo)
11.  LabelPrev3_T2.grid(row=17, column=1, pady=10)
12.  LabelPrev3_T3= tk.Label(self, font=fuente, text='T3:'+ str(T3Prev3), b
   g=fondo)
13.  LabelPrev3_T3.grid(row=18, column=1, pady=10)
14.
15.          #boton de la grafica
16.  botonGraficaAverias_prev3 = tk.Button(self, font=fuente, text='Grafica
    Averias en cada T', command= lambda: self.graficaAverias_prev3(graf_av
   erias_Prev3))
17.  botonGraficaAverias_prev3.grid(row=17, column=2, pady=10)
18.  botonGraficaTC_prev3 = tk.Button(self, font=fuente, text='Grafica de T
   2 vs T3 vs Coste', command= lambda: self.graficaTC_prev3(graf_TiempoCos
   te_Prev3))
19.  botonGraficaTC_prev3.grid(row=18, column=2, pady=10)
```

pResultados.py

- Elemento botón para salir:

```
1. exit_button = tk.Button(self, font=fuente, text='Salir', width=15, comm
   and= lambda: self.salir())
2. exit_button.grid(row=19, column=3)
```

pResultados.py

- Funciones para el llamado de los módulos de cálculo de modelos cuantitativos: se desarrollaron 4 métodos de la clase que al ejecutarse al instanciarse la ventana de la clase **VentanaResultados** se les pasa por parámetros los datos que requieren los modelos para resolverse y retornan los resultados para mostrarse en pantalla.

```
1. def calcularCor(self, C1, T1, N, p):
2.     #llamar cCorrectivo
3.     coste, Np = cCorrectivo.calcular(C1, T1, N, p)
4.     return coste, Np
5.
6. def calcularPre(self, C1, C2, T1, N, p):
7.     #llamar cPreventivo
8.    coste, T2, averias, graf_Averias, graf_TiempoCoste = cPreventivo1.ca
   lcular(C1, C2, T1, N, p)
9.     return coste, T2, averias, graf_Averias, graf_TiempoCoste
10.
11.  def calcularPre2(self, C1, C2, T1, N, p):
12.      #llamar cPreventivo2
13.      costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias = cPrev
   entivo2.calcular(C1, C2, T1, N, p)
14.      return costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias
15.
16.  def calcularPre3(self, C1, C2, C3, T1, N, p, B_M, M_B):
17.      #llamar cPreventivo3
18.      costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias = cPrev
   entivo3.calcular(C1, C2, C3, T1, N, p, B_M, M_B)
19.      return costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias
```

pResultados.py

- Función de gráficacion en 2D para política preventiva total: los módulos de cálculo encargados de resolver los 4 modelos cuantitativos al ser llamados en el código de la ventana de resultados, adicionalmente a retornar el resultado óptimo de cada modelo restornán una tabla en forma de array bidimensional (la variable "g") que contiene todas las soluciones exploradas con el objetivo de realizar una graficación de los valores de las variables contra el valor del criterio objetivo, que en este caso es el coste.

```
1.     def graficaTC_prev1(self, g):
2.         fig, ax = plt.subplots()
3.         ax.plot(g[1:, 0], g[1:, 1], color='blue', marker='o')
4.         plt.xlabel('T(periodos)')
5.         plt.ylabel('Coste (u.m.)')
6.         plt.show()
7.
8.     def graficaAverias_prev1(self, g):
9.         fig, ax = plt.subplots()
10.          ax.plot(g[1:, 0], g[1:, 1], color='blue', marker='o')
11.          plt.xlabel('T(periodos)')
```

```
12.          plt.ylabel('Averias')
13.          plt.show()
```

pResultados.py

Para esta funcionalidad se utilizó la librería de desarrollo Matplotlib[7] que permite desplegar en una nueva ventana utilizando su función *plot* pasandole por parámetros los valores de los puntos en los ejes "x" e "y".

- Funciones de graficación en 3D para políticas de mantenimiento preventivo basado en tiempo de funcionamiento sin fallos con y sin inspección previa: basado en la lógica del caso anterior se emplea el uso esta vez de un sistema de coordenadas tridimensional para graficar la relación del criterio objetivo con las dos variables temporales de los modelos. Se utiliza la clase de Matplotlib **Axes3D** y su función *plot_surface* para pasarle por parámetros los valores a graficar en la variable "g" obtenida de la ejecución de los módulos de cálculo de las políticas de mantenimiento basado en tiempo de funcionamiento sin fallos.

```
1.    def graficaTC_prev2(self, g):
2.         fig = plt.figure()
3.         ax = fig.add_subplot(projection='3d')
4.         Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:, 2
   ], cmap='viridis')
5.         ax.set_xlabel('T2')
6.         ax.set_ylabel('T3')
7.         ax.set_zlabel('Coste (u.m.)')
8.         plt.show()
9.
10.      def graficaAverias_prev2(self, g):
11.          fig = plt.figure()
12.          ax = fig.add_subplot(projection='3d')
13.          Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
   2], cmap='viridis')
14.          ax.set_xlabel('T2')
15.          ax.set_ylabel('T3')
16.          ax.set_zlabel('Averias')
17.          plt.show()
18.
19.      def graficaTC_prev3(self, g):
20.          fig = plt.figure()
21.          ax = fig.add_subplot(projection='3d')
22.          Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
   2], cmap='viridis')
23.          ax.set_xlabel('T2')
24.          ax.set_ylabel('T3')
25.          ax.set_zlabel('Coste (u.m.)')
26.          plt.show()
27.
28.      def graficaAverias_prev3(self, g):
29.          fig = plt.figure()
30.          ax = fig.add_subplot(projection='3d')
31.          Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
   2], cmap='viridis')
32.          ax.set_xlabel('T2')
33.          ax.set_ylabel('T3')
34.          ax.set_zlabel('Averias')
35.          plt.show()
```

pResultados.py

- Botón para cerrar la ejecución del programa: llamada por el elemento botón “exit_button” cuando se clica sobre él.

```
1.      def salir(self):
2.          self.destroy()
```

pResultados.py

3.3.4 Desarrollo de módulo de cálculo para modelo de mantenimiento correctivo

El código empleado para la resolución de este modelo emplea el uso de funcionalidades de la librería de desarrollo Numpy que se importa al principio de éste archivo, el modelo requiere de la declaracion de 7 variables, de las cuales 2 de ellas son los vectores “_ep” y “_epi” que representa el vector de estado del sistema normalizado y un vector auxiliar respectivamente, éste ultimo almacena los valores de resolucion del sistema de ecuacioes (2-18) planteado en el Capítulo 2. Otras dos variables declaradas son las matrices “_P” y “_Q” de dimensiones T1xT1 siendo “_P” la matriz de transición del sistema y “_Q” la matriz de cambio.

```
1. import numpy as np
2.
3. def calcular(C1, T1, N, p):
4.      #declarar el vector ep, epi= ep' y la matriz P y Q
5.      _ep = np.zeros(T1)
6.      _epi = np.ones(T1)
7.      _Q = np.zeros((T1, T1))
8.      _P = np.zeros((T1, T1))
9.      ratio = 0
10.       Np = 0
11.       coste = 0
```

cCorrectivo.py

Como se observa en la declaración de la función *calcular* ésta recibe por parámetros el coste de las reparaciones “C1”, el tiempo máximo de operación de un equipo “T1”, el numero de equipos del sistema “N”, y el vector con las probabilidades de fallo del equipo analizado para cada periodo, que son asignados a la matriz “_P”.

Para resolver el modelo ésta matriz se debe transformar en la matriz de cambio “_Q” restando a la matriz “_P” la matriz identidad.

```
1.      for i in range(0, T1-1):
2.          _P[i, 0] = p[i]
3.          _P[i, i+1] = 1- p[i]
4.      _P[T1 -1, 0] = 1
5.      #se transforma la matriz de prob p en q=p-I
6.      _Q = _P
7.      for i in range(0, T1):
8.          _Q[i, i] -= 1
```

cCorrectivo.py

El sistema de ecuaciones que corresponde a la matriz de cambio, más una restricción adicional (2-19), se plantea este sistema para resolverlo utilizando transformaciones de Gauss-Jordan dejando una matriz triangular superior y obteniendo los resultados de las variables del vector (2-20), de modo que la matriz “_Q” debe transponerse, y se declara un nuevo vector columna “b” representando el termino constante de las ecuaciones, los resultados de

la resolución se irán almacenando en el vector "_epi".

```
1.  #Resolucion del sistema aplicando transformaciones de
2. Gauss-Jordan para encontrar _ep
3.      _Q = _Q.T
4.      b = np.zeros(T1)
5.      b = b.reshape(T1, 1)
6.
7.      n = len(b)
8.
9.      for k in range(0, n):
10.          for i in range(k+1, n):
11.              ratio = _Q[i, k]/_Q[k, k]
12.              for j in range(k, n):
13.                  _Q[i, j] -= ratio*_Q[k, j]
14.              b[i] = b[i] - ratio*b[k]
15.
16.       b[n-1] = 1
17.
18.       _epi[n-1] = 1
19.
20.       for i in range(n-2, -1, -1):
21.          sum_j = 0
22.          for j in range(i+1, n):
23.              sum_j += _Q[i, j]*_epi[j]
24.          _epi[i] = (b[i] - sum_j)/_Q[i ,i]
```

cCorrectivo.py

Se procede a obtener el vector planteado en (2-21), que es el vector "_ep", el vector de estado del sistema en régimen permanente, el modelo se resuelve multiplicando la primera componente de este vector por el número de equipos que conforman el sistema y el coste de cada reparación tras pasado un número muy alto de períodos, obteniendo el valor de costes en régimen permanente.

Los valores al final son retornados a la salida de la función usando la palabra reservada de Python *return*, devolviendo el coste total y el numero de averías.

```
1.      suma = 0
2.      for i in range(0, n):
3.          suma = _epi[i] + suma
4.      _ep = _epi*(1/suma)
5.      #calculo de las reparaciones/averias totales
6.      Np = N*_ep[0]
7.      coste = Np*C1
8.      return coste, Np
```

cCorrectivo.py

3.3.5 Desarrollo del módulo de cálculo para el modelo de mantenimiento preventivo total

Se emplea el uso de la librería Numpy en éste archivo por lo que el primer paso es importarla, ésta función recibe por parámetros el coste de las reparaciones, el coste de las operaciones de mantenimiento, el número maximo de períodos de funcionamiento de un equipo, el número de equipos, y el vector de probabilidades de fallo.

El código emplea el uso de las variables que almacenán los valores de los costes totales y averías de cada solución del modelo y almacenán en unas matrices bidimensionales de Numpy ("graf_Averias" y "graf_TiempoCoste")

los resultados para la definición de las gráficas, y una variable auxiliar "costeMin" almacena la solución óptima.

Se declaran 3 arrays de Numpy, uno unidimensional "_e" que representa el vector de estado del sistema normalizado, "_P" que representa la matriz de transición y "_M" que representa la matriz de mantenimiento del modelo.

```
1. import numpy as np
2.
3. def calcular(C1, C2, T1, N, p):
4.     #declaracion de matriz de mantenimiento y transicion, vectores
5.     _M = np.zeros((T1, T1))
6.     _P = np.zeros((T1, T1))
7.     _e = np.zeros(T1)
8.     _e[0] = 1
9.
10.      graf_Averias = np.zeros((T1, 2))
11.      graf_TiempoCoste = np.zeros((T1, 2))
12.      costeCorr = 0
13.      costePrev = 0
14.      costeTotal = 0
15.      nAverias = 0
16.      costeMin = 0
```

cPreventivo1.py

Se procede a llenar el array que representará a la matriz de mantenimiento (2-23) y la matriz de probabilidad de transición con los valores del vector p.

```
1.     for i in range(0, T1):
2.         _M[i, 0] = 1
3.
4.     for i in range(0, T1 -1):
5.         _P[i, 0] = p[i]
6.         _P[i, i+1] = 1 - p[i]
7.     _P[T1 -1, 0] = 1
```

cPreventivo1.py

En la resolución de este modelo se explorarán todos los valores que puede tomar la variable "T2" y se tomará como mayor valor el que haga mínimo el valor de la variable "costeMin", utilizando un bucle se evalúan las expresiones (2-26) y (2-28) del modelo preventivo total y se utiliza la variable auxiliar "costeTotal" para verificar el coste mínimo obtenido para un determinado valor de "T2", para ésto inicialmente la variable "costeMin" se declara con un valor muy alto. El bucle al final de cáda ciclo "reinicia" el vector de estado multiplicandolo por la matriz "_M".

Simultaneamente la matriz bidimensional de Numpy "graf_TiempoCoste" irá almacenando todas las soluciones encontradas para generar la proyección grafica desde la pantalla de resultados su el usuario lo desea y la matriz "graf_Averías" se obtiene de estimar todas las averías que se acumulan para cada valor de "T2" siendo éstos el número de equipos que aparecen en la posicion 0 del vector de estado al pasar cada ciclo. Finalmente, la función retorna los resultados del modelo y las matrices con los valores para las graficas llamadas desde la pantallas de resultados.

```
1. #calculo del coste de reparaciones
2.     costeMin = C1*T1*10000
3.     T2 = 0
```

```
4.      nAverias = 0
5.      for i in range(1, T1):
6.          suma = 0
7.
8.          for k in range(1, i+1):
9.              _e = _e@_P
10.               suma = _e.item(0) + suma
11.           costeCorr = (C1*N*suma)/i #coste medio por periodo
12.           costePrev = (N*C2)/i
13.           graf_Averias[i, 0] = i
14.           graf_Averias[i, 1] = N*suma/i
15.
16.           #coste total medio por periodo
17.           costeTotal = costeCorr+costePrev
18.           graf_TiempoCoste[i, 0] = i
19.           graf_TiempoCoste[i, 1] = costeTotal
20.
21.       if(costeTotal < costeMin):
22.               costeMin = costeTotal
23.               T2 = i
24.                   nAverias = N*suma/i
25.       _e = _e@_M
26.       return costeMin, T2, nAverias, graf_Averias, graf_TiempoCoste
```

cPreventivo1.py

3.3.6 Desarrollo del módulo de cálculo para el modelo de mantenimiento preventivo basado en el tiempo de funcionamiento ininterrumpido sin fallos

En éste modulo de cálculo al emplearse funcionalidades de manejo de vectores y matrices se incluye al igual que en los anteriores la librería de desarrollo Numpy, se aprovechará la funcionalidad *matrix_power* de esta librería para el cálculo de la matriz de cambio "_C" correspondiente a la expresión (2-32). La función de cálculo de éste modulo recibe 5 parámetros, que al igual que en el epígrafe anterior corresponden a los costes por operación de mantenimiento y reparación, el número máximo de períodos de funcionamiento de un equipo, el número de equipos y las probabilidades de transición.

Seguidamente se declaran las variables que se utilizán en esta función, la matriz de trasición, las matrices bidimencionales que almacenarán todas las soluciones para las gráficas, costes y número de averías.

```
1. import numpy as np
2. def calcular(C1, C2, T1, N, p):
3.       #declaracion de matriz de mantenimiento y transicion, vectores
4.       _P = np.zeros((T1, T1))
5.       graf_Averias = np.zeros((T1, T1, 3))
6.       graf_TiempoCoste = np.zeros((T1, T1, 3))
7.       costeCorr = 0
8.       costePrev = 0
9.       costeTotal = 0
10.        costeMin = 0
11.        nAverias = 0
12.        Averias = 0
```

cPreventivo2.py

Se prepara la matriz de probabilidades de transición "_P" y se declaran las variables "T2" y "T3". Similar al caso anterior se le asigna a la variable "costeMin" un valor muy alto, puesto que éste será el que almacenará y

comprobará el criterio objetivo en cada solución explorada.

```
1. for i in range(0, T1 -1):
2.          _P[i, 0] = p[i]
3.          _P[i, i+1] = 1 - p[i]
4.      _P[T1 -1, 0] = 1
5.      costeMin = C1*T1*1000
6.      T2 = 0
7.      T3 = 0 #periodos de funcionamiento ininterrumpido
```

cPreventivo2.py

Se utilizan dos bucles *for* anidados que evaluarán el modelo en cada posible valor a tomar por las variables "T2" con el iterador "i" y "T3" con el iterador "j", que comprende desde 1 hasta el valor T1. Al inicio de cada exploración de solución se reinician los valores de los vectores de estado del sistema "_epi", "_ep" y "_e". Todos los procesos explicados acontinuación están contenidos dentro de estos bucles, hasta el uso de la función *return*.

```
1.      for i in range(1, T1):
2.          for j in range(1, T1):
3.              #vector de estado del sistema
4.              _epi = np.ones(T1)
5.              _ep = np.zeros(T1)
6.              _e = np.zeros(T1)
7.              _e[0] = 1
```

cPreventivo2.py

La matriz de mantenimiento al depender de los valores de "T2" y "T3" se declará en función de estos valores al inicio de cada "exploración", según el planteamiento matemático del modelo en el punto 2.5.2.4.2 del Capítulo 2. Se procede a plantear la matriz de cambio "_C" como se ve en las ecuaciones (2-32) y a plantear el sistema de ecuaciones (2-33). Para determinar el valor de la matriz de cambio se multiplica la matriz "_M" por la matriz "_P" elevada a la potencia del valor de "T2" explorado, se utiliza la función *matrix_power* de la libreria Numpy.

```
1.              #matriz de mantenimiento
2.              _M = np.zeros((T1, T1))
3.              for k in range(1, T1):
4.                  if(k<=j):
5.                      _M[k, k] = 1
6.                  else:
7.                      _M[k, 0] = 1
8.
9.              _C = np.zeros((T1, T1))
10.               _C = _M@(np.linalg.matrix_power(_P, i))
11.
12.               for l in range(0, T1):
13.                   _C[l,l] -= 1
```

cPreventivo2.py

Se plantea el sistema de ecuaciones (2-33) a resolver utilizando la matriz de cambio "_C" y un vector columna "b" representando éste último al vector columna con los términos independientes de las ecuaciones.

Se traspone la matriz de cambio y se resuelve el sistema aplicando transformaciones de Gauss-Jordan para tener una matriz triangular superior ampliada, y se va obteniendo el resultado arbitrario del vector de estado de abajo hacia arriba y se asignan a "_epi", para calcular finalmente el vector de estado en regimen permanente "_ep".

```
#Resolucion del sistema aplicando transformaciones de
Gauss-Jordan para encontrar _ep
            _C = _C.T
            b = np.zeros(T1)
            b = b.reshape(T1, 1)
            ratio = 0

            n = len(b)

              for k in range(0, n):
                  for ii in range(k+1, n):
                      ratio = _C[ii, k]/_C[k, k]
                      for jj in range(k, n):
                          _C[ii, jj] -= ratio*_C[k, jj]
                      b[ii] = b[ii] - ratio*b[k]

              b[n-1] = 1
              _epi[n-1] = 1

              for ii in range(n-2, -1, -1):
                  sum_j = 0
                  for jj in range(ii+1, n):
                      sum_j += _C[ii, jj]*_epi[jj]
                  _epi[ii] = (b[ii] - sum_j)/_C[ii ,ii]

              #vector de estado
              suma = 0
              for kk in range(0, n):
                  suma = _epi[kk] + suma
              _ep = _epi*(1/suma)
```

cPreventivo2.py

El cálculo de los costes por mantenimiento preventivo y reparaciónes medios por períodos se realiza evaluando la ecuación (2-39) con la suma acumulada de las componentes del vector de estado "_ep" desde el valor de "T3" explorado hasta el valor de "T1" según la expresión (2-37). Se calcula el número de operaciones de mantenimiento correctivo sumando el número de equipos que aparecerán en la primera componente del vector de estado puesto que si aparecen en ésta componente significa que han fallado en el período anterior, se suman todas las averías hasta el valor de "T2".

Los valores de averías y coste total de cada solución explorada se almacenan en las matrices "graf_TiempoCoste" y "graf_Averias", en cada fila, la primera componente será el valor de la variable "T2", la segunda "T3" y la tercera el coste total obtenido de la evaluación del modelo en esos resultados que serán retornadas por la función del módulo de cálculo al cuerpo del programa principal al igual que las soluciones óptimas del modelo.

```
            #calculo del coste de mantenimiento preventivo medio por pe
  riodo
            suma = 0
            for ll in range(j, T1):
                suma = _ep.item(ll) + suma
            costePrev = (C2*N*suma)/i

            #calculo del coste de mantenimiento correctivo medio por pe
  riodo
            suma = 0
            for kk in range(1, i+1):
                  _e = _e@_P
```

```
                suma = _e.item(0) + suma
            nAverias = suma*N
            costeCorr = (C1*N*suma)/i

            costeTotal = costePrev+costeCorr
            graf_TiempoCoste[i, j, 0] = i
            graf_TiempoCoste[i, j, 1] = j
            graf_TiempoCoste[i, j, 2] = costeTotal

            graf_Averias[i, j, 0] = i
            graf_Averias[i, j, 1] = j
            graf_Averias[i, j, 2] = nAverias/i

            if(costeTotal <  costeMin):
                costeMin = costeTotal
                T2 = i
                T3 = j
                Averias = nAverias/i

    return costeMin, T2, T3, Averias, graf_TiempoCoste, graf_Averias
```

cPreventivo2.py

3.3.7 Desarrollo de del módulo de cálculo para para el modelo de mantenimiento basado en el tiempo de funcionamiento ininterrumpido sin fallos y el resultado de una inspección

El desarrollo de este módulo empieza por incluir la libreria de desarrollo Numpy, la función de cálculo de éste modelo recibe por parámetros los valores de coste de mantenimiento, reparaciones, coste de inspecciones, número máximo de períodos de funcionamiento de un equipo, el número de equipos que conforma el sistema, las probabilidades de fallo y la calidad de los procesos de inspección.

Se procede a declarar las variables y vectores a utilizar en el desarrollo de la función, a diferencia del modelo anterior se emplea adicionalmente una variable que almacenará el coste de las inspecciones y la matriz "_W" planteada en el punto 2.5.2.4.3.2 del Capitulo 2 como solución a la influencia del mantenimiento selectivo en el modelo.

```
import numpy as np

def calcular(C1, C2, C3, T1, N, p, B_M, M_B):
    _P = np.zeros((T1, T1))
    _W = np.zeros((T1, T1))

    graf_Averias = np.zeros((T1, T1, 3))
    graf_TiempoCoste = np.zeros((T1, T1, 3))
    costeCorr = 0
      costePrev = 0
      costeInsp = 0
      costeTotal = 0
      costeMin = 0
      nAverias = 0
      Averias = 0
```

cPreventivo3.py

Se crea la matriz de probabilidad de transición asignando los valores del parámetro "p", se declaran las variables "costeMin", "T2" y "T3" como en el módulo *cPreventivo2.py*.

```
1.      for i in range(0, T1 -1):
2.          _P[i, 0] = p[i]
3.          _P[i, i+1] = 1 - p[i]
4.      _P[T1 -1, 0] = 1
5.      costeMin = C1*T1*1000
6.      T2 = 0
7.      T3 = 0 #periodos de funcionamiento ininterrumpido
```

cPreventivo3.py

Para la exploración de las soluciones se utilizan dos bucles *for* anidados como en el caso anterior, se declaran los vectores de estado a utilizar "_epi", "_ep" y "_e", en éste caso la matriz de mantenimiento al depender también del valor de las variables "T2" y "T3" se declara según el planteamiento del punto 2.5.2.4.3.1 del Capitulo 2 dentro del bucle y se reinicia al iniciar éste de nuevo.

Para determinar la matriz "_W" ésta se calcula según los valores y condiciones impuestos en el punto 2.5.2.4.3.2 del Capítulo 2, empleando un condicional para en el caso en que la fila de la matriz sea menor que "T3" se le deseará asignar el valor de la matriz de transición en esa posicion, y en caso contrario se aplican las expresiones (2-43) y (2-44).

```
1. for i in range(1, T1):
2.      for j in range(1, T1):
3.
4.          _epi = np.ones(T1)
5.          _ep = np.zeros(T1)
6.          _e = np.zeros(T1)
7.          _e[0] = 1
8.
9.          #matriz de mantenimiento
10.           _M = np.zeros((T1, T1))
11.
12.           for k in range(0, T1 -2):
13.               if(k<=j):
14.                   _M[k, k] = 1
15.               else:
16.                   _M[k, 0] = (_P[k, 0]*(1-M_B)) + (_P[k, k+1]*B_M)
17.                   _M[k, k] = (_P[k, 0]*M_B) + (_P[k, k+1]*(1-B_M))
18.           _M[T1-1, 0] = 1
19.
20.           for tt in range(0, T1-1):
21.               for ss in range(0, T1-1):
22.                   if(tt <= j):
23.                       _W[tt, ss] = _P[tt, ss]
24.                   else:
25.                       _W [tt, 0] = (_P[tt, 0]*M_B)/((_P[tt, 0]*M_B) + (_
    P[tt, tt+1]*(B_M)))
26.                       _W[tt, tt+1] = (_P[tt, tt+1]*(1-
    B_M)/((_P[tt, 0]*M_B) + (_P[tt, tt+1]*(1-B_M))))
```

cPreventivo3.py

Se calcula la matriz de cambio "_C" según la expresión (2-46), aquí es necesario emplear la función de la librería Numpy *matrix_power*. Realizando el planteamiento (2-33) se traspone la matriz "_C" y declarando el vector columna "b" aplica el método de Gauss-Jordan para resolver el sistema similar al caso del módulo anterior para almacenar los resultados en el vector auxiliar "_epi" y obtener el vector de estado del sistema en régimen permanente "_ep".

```
1.              _C = np.zeros(T1)
2.              _C = _M@_W@(np.linalg.matrix_power(_P, i-1))
3.
4.              for l in range(0, T1):
5.                  _C[l, l] -=1
6.              #Resolucion del sistema aplicando transformaciones de Gauss
   -Jordan para encontrar _ep
7.
8.              _C = _C.T
9.              b = np.zeros(T1)
10.               b = b.reshape(T1, 1)
11.               ratio = 0
12.
13.               n = len(b)
14.
15.               for k in range(0, n):
16.                   for ii in range(k+1, n):
17.                       ratio = _C[ii, k]/_C[k, k]
18.                       for jj in range(k, n):
19.                           _C[ii, jj] -= ratio*_C[k, jj]
20.                       b[ii] = b[ii] - ratio*b[k]
21.
22.               b[n-1] = 1
23.               _epi[n-1] = 1
24.
25.               for ii in range(n-2, -1, -1):
26.                   sum_j = 0
27.                   for jj in range(ii+1, n):
28.                       sum_j += _C[ii, jj]*_epi[jj]
29.                   _epi[ii] = (b[ii] - sum_j)/_C[ii ,ii]
30.
31.               #vector de estado
32.               suma = 0
33.               for kk in range(0, n):
34.                   suma = _epi[kk] + suma
35.
36.               _ep = _epi*(1/suma)
```

cPreventivo3.py

En el cálculo de los costes de este modelo la estimación de averías se obtiene con un procedimiento igual que el modelo anterior sumando los equipos que aparecen en su primer período de funcionamiento initerrumpido hasta llegar “T2”.

Las inspecciones realizadas se obtienen de sumar todos los equipos que en el vector de estado del sistema en régimen permanente aparecen en su período “T3” o mayor de funcionamiento ininterrumpido sin fallos y ésta se multiplica por el coste de las inspecciones.

El cálculo del coste de mantenimiento preventivo se obtiene de la diferencia entre el número de equipos que fueron reparados menos el número de equipos totales que aparecen como nuevos en el período siguiente. En algunos casos ésta diferencia puede llegar a resultar en valores de signo negativo, por lo que la operación debe pasarse por la función *abs* de la librería interna del interprete de Python[5] que retorna el valor absoluto de la operación realizada.

Se suman los costes totales y se compara con la solución de coste mínimo encontrada. Se guardan las soluciones de cada iteración del bucle en las matrices “graf_TiempoCoste” y “graf_Averias”, finalmente se emplea la palabra reservada de Python *return* para retornar al programa principal los resultados de la resolución del modelo y las matrices con los valores de las soluciones para realizar las gráficas.

```
#calculo del coste de mantenimiento correctivo medio por periodo
            suma = 0
            for kk in range(1, i+1):
                _e = _e@_P
                suma = _e.item(0) + suma
            nAverias = suma*N
            costeCorr = (C1*N*suma)/i

      #calculo del coste de las inspecciones medio durante cada periodo
             suma = 0
             for ll in range(j, T1-1):
                 suma = _ep.item(ll) + suma
             costeInsp = (C3*N*suma)/i
    #calculo del coste de mantenimiento preventivo durante cada periodo
             Mc = 0
             Mc = abs((nAverias/N) - _ep.item(0))
             costePrev = (C2*N*Mc)/i

             costeTotal = costePrev+costeCorr+costeInsp
             graf_TiempoCoste[i, j, 0] = i
             graf_TiempoCoste[i, j, 1] = j
             graf_TiempoCoste[i, j, 2] = costeTotal

             graf_Averias[i, j, 0] = i
             graf_Averias[i, j, 1] = j
             graf_Averias[i, j, 2] = nAverias/i

             if(costeTotal < costeMin):
                 costeMin = costeTotal
                 T2 = i
                 T3 = j
                 Averias = nAverias/i

     return costeMin, T2, T3, Averias, graf_TiempoCoste, graf_Averias
```

cPreventivo3.py

3.3.8 Desarrollo del módulo de cálculo para la obtención de la matriz de transición a partir de una función de distribución de probabilidad Exponencial

Se utilizan las funciones de manejo de vectores de la libreria Numpy y la función *exp* que calcula la exponecial de los elementos introducidos. La función *calcula* del módulo recibirá por parámetros el valor de lambda en la variable "k" y el largo del vector de probabilidades de fallo que explorará la función de cálculo, evaluando desde 1 hasta "T1-1" la expression (2-4).

```
import numpy as np

def calcula(k, T1):

    k = float(k)
    T1 = int(T1)

    p = np.zeros(T1)

   for i in range(1, T1-1):
       p[i] = 1 - np.exp(-(k*i))
   return p
```

cExponencial.py

3.3.9 Desarrollo del módulo de cálculo para la obtención de la matriz de transición a partir de una función de distribución de probabilidad Weibull de tres parámetros

Similar al epígrafe anterior se evaluarán los valores desde 1 hasta "T1-1" en la expresión (2-5), siendo los parámetros de la función "alpha" el factor de forma de la curva, "betha" el factor de escala y "k" el desplazamiento horizontal de la curva.

```
1. import numpy as np
2.
3. def calcula(alpha, betha, k, T1):
4.
5.     alpha = float(alpha)
6.     betha = float(betha)
7.     k = float(k)
8.     T1 = int(T1)
9.     p = np.zeros(T1)
10.
11.      for i in range(1, T1-1):
12.          #llenar el array p con los valores obtenidos de la FDA Weibull
13.          p[i] = 1 - np.exp(-((i-k)/betha)**alpha)
14.
15.      return p
```

cWeibull.py

3.3.10 Desarrollo de módulo de cálculo para la obtención de la matriz de transición a partir de una función de distribución de probabilidad Weibull, calculada empleando el estimador estadístico Rango de Mediana y la función observada

Este módulo tiene como objetivo calcular los valores del parámetro de forma y escala de la función Weibull como se planteó en el punto 2.6 del Capitulo 2. La función recibe por parámetros de entrada una matriz bidimensional de Numpy de nombre "tabla", estando en el formáto solicitado, se procede a cambiar los valores alfanuméricos de la tabla solo por valores numéricos del tipo 'int32'. Como se quiere tener solo valores numéricos en la segunda columna de la tabla, se cambian los caracteres 'R' por el valor 1 y los caracteres 'F' por el valor 2.

Se emplea la función de Numpy *argsort* para ordenar las filas de la tabla según la tercera columna (tiempo de operación) en orden creciente.

```
1. import numpy as np
2.
3. def calcular(tabla):
4.     alpha = 0
5.     betha = 0
6.     # ordenar los datos de la tabla en orden ascendente según las horas
    de operacion
7.     for i in range(0, len(tabla)):
8.         if(tabla[i, 1]=='R'):
9.             tabla[i, 1] = 1
10.          else:
11.              tabla[i, 1] = 2
12.      tabla = tabla.astype('int32')
13.      tabla = tabla[tabla[:, 2].argsort()]
```

cRangoMedianas.py

Para obtener el nuevo número de medida utilizando las expresiones (2-57) y (2-58) se declara un vector "incremento" y una matriz bidimensional de 2 columnas, denominada "NuevoOrden", que almacena en sus posiciones el nuevo número de medida y el tiempo de operación de esa medida, el resto de posiciones estarán vacías. Se crea un nuevo vector "tiempo_op" de longitud igual al número de posiciones distintas de 0 en el vector "NuevoOrden", con el objetivo de tener un vector que tenga las medidas ordenadas de forma similar al planteamiento de la **Tabla 2-2**.

```
1.      #calculo de indices de muestras de fallo
2.      N = len(tabla)
3.
4.      incremento = np.zeros(N)
5.      NuevoOrden = np.zeros((N,2))
6.      ElementoQueFallo = 0
7.
8.      for i in range(0, N):
9.          if(tabla[i, 1]==2):
10.             incremento[i] = (N+1-ElementoQueFallo)/(N+1-i)
11.             NuevoOrden[i,1] = incremento.item(i) + ElementoQueFallo
12.             ElementoQueFallo = NuevoOrden[i, 1]
13.             NuevoOrden[i,0] = tabla.item(i, 2)
14.
15.       tiempo_op = np.zeros((np.count_nonzero(NuevoOrden[:,0]), 2))
16.       i = 0
17.       for j in range(0, len(NuevoOrden)):
18.           if(NuevoOrden.item(j, 0)!=0):
19.               tiempo_op[i, 0] = NuevoOrden[j, 0] #asigna el tiempo de op
    eracion
20.               tiempo_op[i, 1] = NuevoOrden[j, 1] #asigna el nuevo i
21.               i = i+1
```

cRangoMedianas.py

Se calcula los valores de la función de observación mediante el estimador (2-59) y se almacenan en el vector "F". El último proceso de este método consiste en obtener la recta de regresión lineal de la transformación logarítmica de la función Weibull, se declaran dos vectores "X" e "Y" de longitud igual al del vector "tiempo_op". Emplenado un bucle *for* se evalua el valor de cada posición dentro del vector "tiempo_op" dentro de la expresión (2-62) y evalua cada posición del vector "F" en la expresión (2-63). Aquí se emplea el uso de la función de la libreria Numpy *log*, esta función retorna el logaritmo natural de la expresión que se le ingresa.

```
1.      #funcion de observacion
2.      F = np.zeros(len(tiempo_op))
3.      for i in range(0, len(tiempo_op)):
4.          F[i] = (tiempo_op[i, 1] -0.3)/(N+0.4)
5.
6.      # ln(t) y ln(ln(1/1-F))
7.
8.      Y = np.zeros(len(tiempo_op))
9.      X = np.zeros(len(tiempo_op))
10.
11.       for i in range(0, len(tiempo_op)):
12.           X[i] = np.log(tiempo_op[i, 0])
13.           Y[i] = np.log(np.log(1/(1-F[i])))
```

cRangoMedianas.py

Para concluir el proceso se calcula la recta mediante dos bucles *for* y los valores obtenidos dentro de los vectores "X" e "Y", con lo que se obtiene la pendiente almacenada en la variable "m" y el intercepto almacenado en la variable "b". Como se explica en el punto 2.6.2 en el Capítulo 2 la pendiente coincide con el valor del parámetro de forma de la curva, que en este módulo tiene el nombre de "alpha" y para calcular el parámetro de la escala se emplea la ecuación (2-64) utilizando el intercepto y la pendiente antes calculada, el parámetro de la escala se asigna a la variable de nombre "betha" y se retornan a la salida de la función utilizando la palabra reservada de Python *return*.

```
1.      #regresion lineal y = mx + b
2.      m = 0
3.      b = 0
4.      suma1 = 0
5.      suma2 = 0
6.
7.      for i in range(0, len(X)):
8.          suma1 = (X.item(i)- np.mean(X))*(Y.item(i)- np.mean(Y)) + suma1
9.      for j in range(0, len(X)):
10.           suma2 = ((X.item(j)- np.mean(X))**2) + suma2
11.
12.       m = suma1/suma2
13.
14.       b = np.mean(Y) - m*np.mean(X)
15.
16.       alpha= m
17.
18.       betha = np.exp(-(b/m))
19.
20.       return alpha, betha
```

cRangoMedianas.py

4 CASO PRÁCTICO

A continuación se emplea el uso de la herramienta desarrollada en un análisis práctico, con el objetivo de medir sus prestaciones y efectividad que puede proporcionar al usuario, realizando varios experimentos en función de un parámetro y comparár los resultados.

Se emplea la herramienta para analizar un caso real de una investigación de referencia[4], donde dispone de una serie de muestras de tiempo de operación entre fallos y mantenimiento preventivo de las juntas tóricas situadas en las conexiones finales del tubo de refrigeración que unen los dos escapes de un motor diesel alternativo de 12 cilindros en forma de V. Éstos elementos estan expuestos a altas temperaturas gracias a los gases de combustión que circulan por los escapes del motor, lo que contribuye a su deteríoro.

El sistema inicialmente tenia implementada una rutina de mantenimiento que consistía en reemplazar las juntas tóricas tras éstas acumular 4000 horas de operación con el motor encendido, el sistéma demostró ser ineficiente contra los fallos de las juntas puesto que se detectáron al menos 2 fallas en cada uno de los equipos antes de realizarse el primer mantenimiento preventivo. El motivo de realizar el estudio sobre este sistema es observar la política de mantenimiento que sugiera empleando los cuatro modelos cuantitativos que utiliza, también realizar otros dos analisis variando la escala numérica de las variables para contrastar los resultados.

4.1 Datos de partida

Según los datos de la investigación de referencia[4] en materia de costes por las intervenciónes, el coste asociado al fallo de un equipo es de 4230€, principalmente por motivos de péridas por no disponibilidad, el costo de mantenimiento correctivo es de 95€/h, operación que se indica tiene una duración media de 8 horas y el costo de la pieza de recambio es de 620€. En el caso de mantenimiento preventivo el costo de realizar la orden y pasar el equipo a estado de indisponible por mantenimiento es de 1€, el costo de la intervención es de 82€/h, operación que tiene una duración media de 7 horas y el costo de la pieza de recambio es de 620€. En el caso de las inspecciones, puesto que no se realiza ninguna interveción sobre el elemento que extienda su fiabilidad se toma como valor de referencia el coste del mantenimiento preventivo por hora (82€) pero se excluye el valor de las herramientas y piezas de recambio usadas (620€) y se asume una duración media de las inspecciones de una hora.

Para conocer el valor del número de períodos maximos que puede operar un equipo sin fallar y las probabilidades de fallo se recurrirá a la funcionalidad del programa para el cálculo de los parámetros de una función de distribucion Weibull. Se sabe que de este sistema se dispone de una base de datos con un total de 121 muestras de tiempos de operación entre fallos y mantenimiento preventivo que, almacenadas en un archivo con extension ".csv" y dispuestos según el ejemplo mostrado en la Figura 3-8, se cargarán en el programa mediante el botón "Calcular F.D. Weibull". Al terminar el cálculo, Manther desplegará un mensaje en una nueva ventana flotante indicando que los parámetros fueron calculados con exito, donde "Alpha" corresponde al parámetro de forma y "Betha" al parámetro de escala de la función.

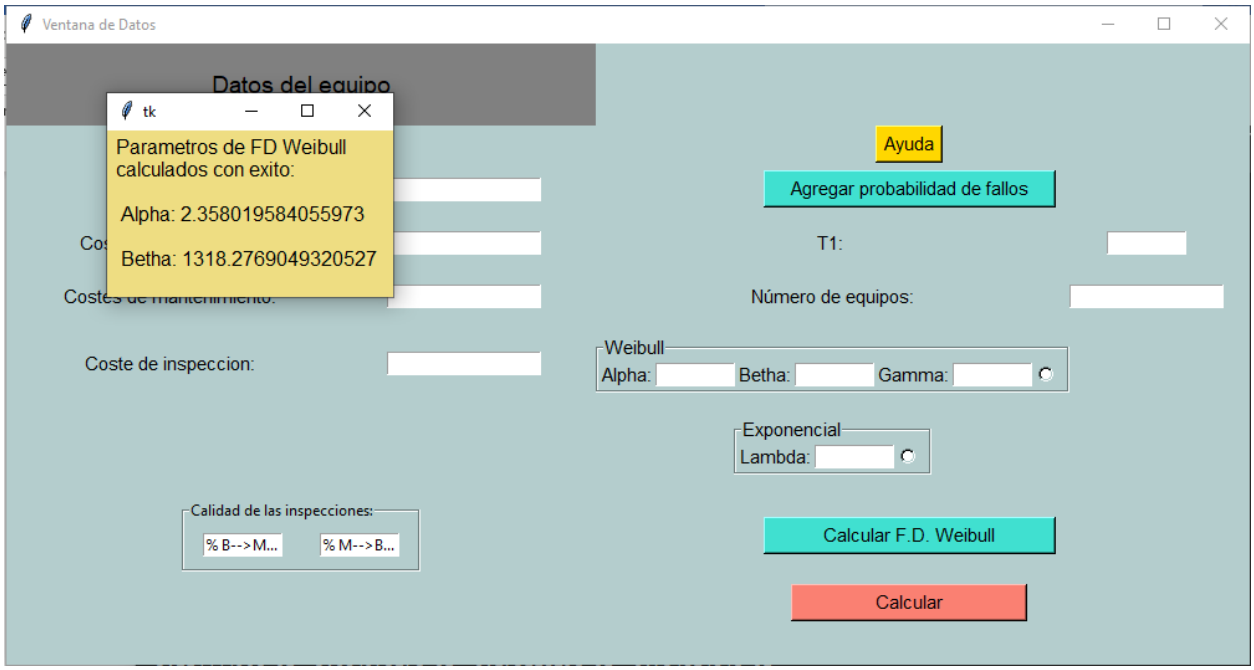

Figura 4-1. Ventana emergente con resultados del cálculo de parámetros de una F.D. Weibull.

Con los valores calculados se utiliza la funcionalidad de cálculo de las probabilidades de fallo, marcando el checkbox de la funciónWeibull e insertando los valores de los parámetros ya conocidos "Alpha" y "Betha" en sus correspondientes casillas, en el parámetro del desplazamiento en el eje horizontal se coloca igual a 0, por ser una curva Weibull de 2 parámetros la que se utilizará.

Para conocer el valor del tiempo máximo que puede operar un equipo sin fallar (T1) se utiliza el parámetro de la escala de la curva calculada. Se sabe que el valor del parámetro de escala ("Betha") representa el percentil 63.2 de todos los datos, se puede determinar un valor para T1 donde la probabilidad de fallo de un equipo en ese período de funcionamiento sea muy alta (alrededor del 95%). El valor que se obtiene es de 2085 horas, quedando los datos de partida del problema para ingresar en los modelos:

- **Costes**:
  - Reparaciones: 4320 + 8*95 + 620= 5700€
  - Mantenimiento preventivo: 1 + 7*82 + 620 = 1195€
  - Inspecciones: 82€
- **Número de equipos**: 32
- **Tiempo máximo de funcionamiento sin fallo**: 2085 h
- **Calidad de las inspecciones**:
  - Probabilidad de catalogar un equipo "en mal estado" como "en buen estado": 50%
  - Probabilidad de catalogar un equipo "en buen estado" como "en mal estado": 50%
- **Parámetro de F.D. Weibull**:
  - Forma: 2.358
  - Escala: 1318.277

4.2 Planteamiento de experimentos

Para el caso práctico se observa que la unidad de medida obliga al experimento a explorar un número muy

grande de soluciones, ya que si se estudia determinando el número de horas que se debe esperar hasta realizarse mantenimiento los modelos deben explorar un total de 4.347.225 soluciones que puede implicar un tiempo de cálculo y necesidad de recursos muy grande para el ordenador que se utilice para el estudio. Lo que nos interesaría es utilizar una unidad de medida diferente, reduciendo el tamaño de la escala original de los períodos temporales, buscando el valor que sea más conveniente para tener los resultados mas ajustados a la realidad, pero sin tener que usar un valor de T1 tan grande, por lo que se ajustará el tamaño de la escala a grupos de 50, 80, y 100 horas, se resolverá el problema para cada escala y se compararán los resultados.

Otro condicional de gran importancia para el planteamiento de la escala es para el cumplimiento de las hipótesis planteadas en el epígrafe 2.5.2.1 en el Capítulo 2, éstas condicionan la utilización de los modelos cuantitativos planteados en determinados sistemas, los valores de los grupos para escalar las variables son los escogidos para que el tiempo de una intervención de mantenimiento o reparación sea muy pequeño en comparación al tamaño de un período temporal elemental (grupo).

En los siguientes epígrafes se plantean por separado cada caso experimental en función de la escala del tamaño de los períodos temporales.

4.2.1 Aplicación de la herramienta al caso práctico utilizando como período temporal grupos de 50 horas

Se utitilizan los mismos valores de costes, número de equipos y calidad de los procesos de inspección descritos anteriormente. Según los resultados obtenidos del cálculo de la función de distribución de probabilidad Weibull el número máximo de horas que puede operar un equipo sin fallar (T_1) es de 2085 horas, escalando ésta medida a grupos de 50 horas se tienen 41.7 períodos temporales que, a efectos de utilización del programa desarrollado, se aproximará al valor de **42**.

Para determinar la matriz de probabilidades de fallo debe cambiarse el valor del parámetro de la escala de la curva Weibull a la misma escala de grupos de 50 horas, el valor del parámetro de escala cálculado de la curva Weibull es 1318.27 horas, que llevandolo a la escala deseada se tiene un valor de **26.35**.

Ingresando los datos antes mencionados en los campos correspondientes en la *pantalla de nuevo análisis* se obtienen los siguientes resultados:

Figura 4-2. Resultados de la resolución de los cuatro modelos siguiendo el criterio objetivo, planteamiento de escala en grupos de 50 horas.

4.2.1.1 Política de mantenimiento correctivo

Se obtuvo un número de averías acumuladas en el sistema al llegar al regimen permanente de 2, aunque se ve que el sistema tiene una tendencia ligeramente baja a fallar los costes de reparación son altos y pueden ser económicamente poco eficiente para la organización, a diferencia de implementar rutinas de mantenimiento que extiendan la vida util de los elementos.

4.2.1.2 Política de mantenimiento preventivo total

Se obtiene una solución siguiendo los modelos donde el sistema no se permita depender mucho del mantenimiento correctivo, aún sabiendo que éste puede tener una tendencia baja a fallar y puede permitirse intervalos entre mantenimiento preventivo mayores, el programa ha determinando que para este sistema el mejor intervalo de períodos para realizar mantenimiento a todos los equipos minimizando los costes es 7, que al llevarlo al valor de la escala original siguiere realizar mantenimiento preventive total cada **350 horas**.

En la Figura 4-3 se observa que a partir del valor 8 de T2 los costes tienden a elevarse hasta por encima del coste total de la política de mantenimiento correctivo unicamente.

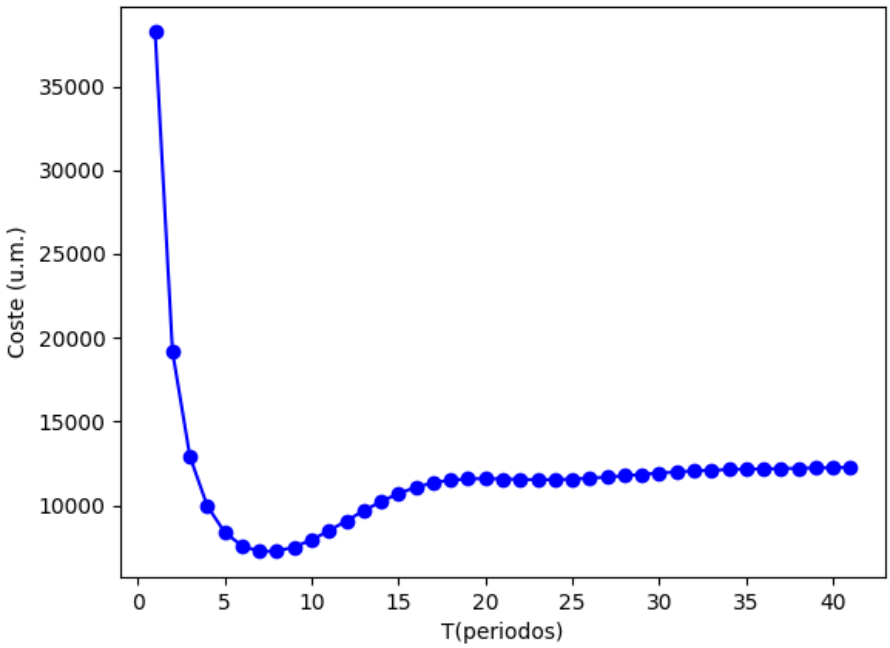

Figura 4-3. Gráfica de costes totales en función de T2.

En la Figura 4-4 se observa como a medida que se realiza mantenimiento preventivo total en períodos más largos el número de averías crece ya que se esta dejando más tiempo para realizarse las intervenciones.

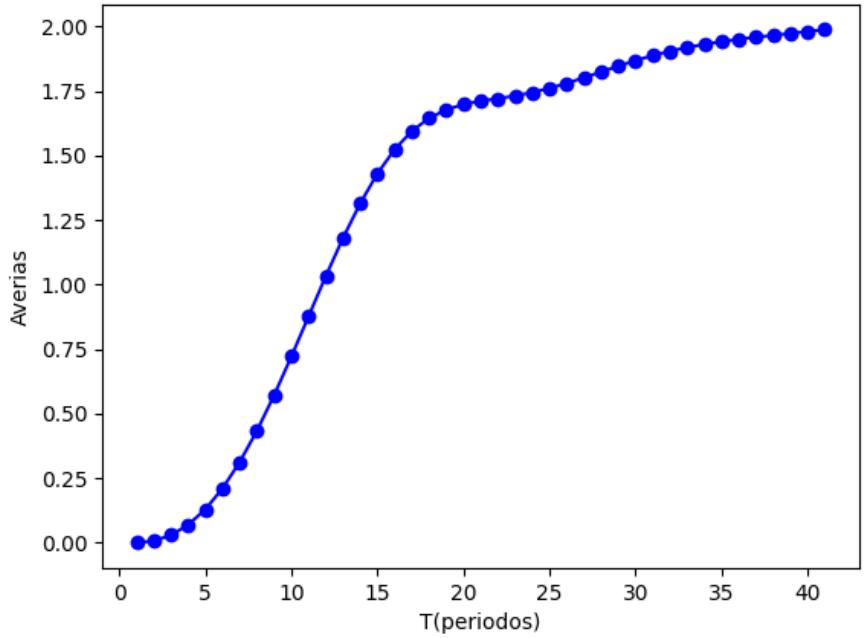

Figura 4-4. Gráfica de averías en función de T2.

4.2.1.3 Política de mantenimiento preventivo en función del número de períodos de funcionamiento sin fallos

El modelo anterior demostró que a pesar de tener el sistema una tendencia a fallos baja es conveniente plantear una estrategia donde las operaciónes de mantenimiento correctivo llevada a cabo por los fallos sean pocas, aún así los costes de mantenimiento preventivo en intervalos muy cortos genera altos gastos por lo que una manera de equilibrar esto y aprovechando que la tendencia de fallos es baja puede ser una política de mantenimiento preventivo selectivo, como se propone en éste modelo y el siguiente.

Se observa en la solución de esta política que los costes bajan considerablemente, así como las averías que presenta el sistema, proponiendo que se realice mantenimiento cada 3 períodos (T_2=**150 horas**) pero solo a los equipos que estén en su período número 41 de funcionamiento ininterrumpido (T_3=**2050 horas**).

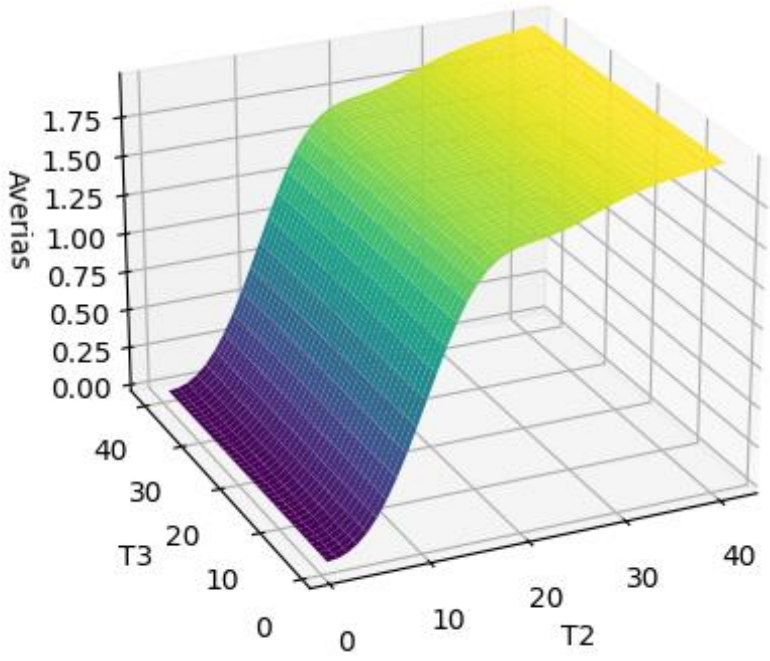

Figura 4-5. Grafica de averías en función de T2 y T3.

En la gráfica de la Figura 4-4 se observa que los costes por períodos tienden a ser muy bajos cuanto más cerca del 1 está el valor de la variable T2 y crece el valor de la variable T3, aunque ya para el valor de T2 = 1 hay un pico en los costes puesto que aunque se reducirian totalmente los costes de mantenimiento correctivo el mantenimiento prventivo se tendría que llevar a cabo en períodos muy cercanos.

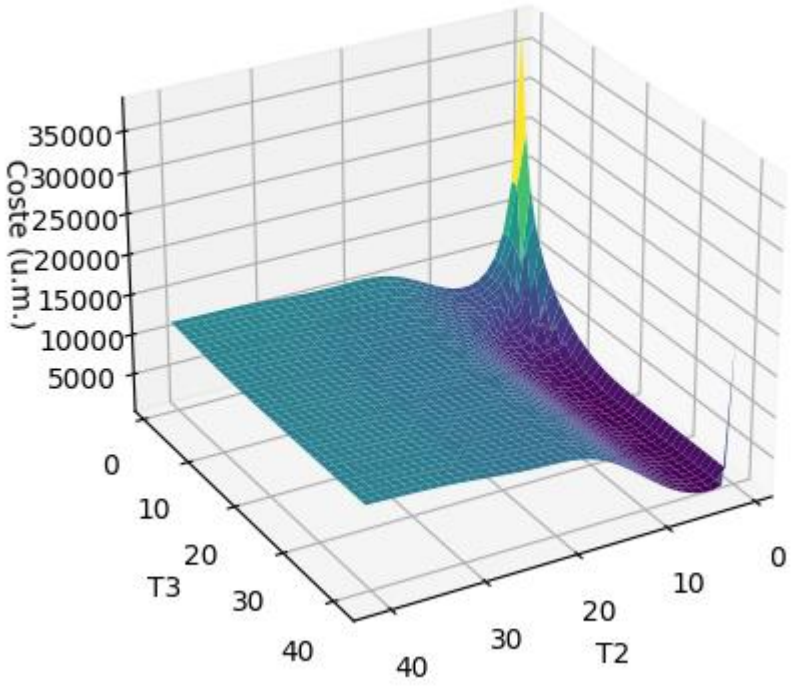

Figura 4-6. Gráfica de costes totales en función de T2 y T3.

4.2.1.4 Política de mantenimiento preventivo en función del número de períodos de funcionamiento sin fallos y una inspección previa

Se observa que Manther determina para este experimento como mejor política el caso de mantenimiento preventivo en función del tiempo de funcionamiento sin fallos más una inspección previa, se obtiene el valor de las variables en $\mathbf{T_2}$ **=4** y $\mathbf{T_3}$**=41,** que llevandolos a la escala original serán **200 horas** y **2050 horas** respectivamente. Con una muy pequeña diferencia seguiría el caso de preventivo sin inspección previa, pero teniendo éste un número mucho menor de averías acumuladas en el sistema en comparación al resto de modelos.

En la figura mostrada acontinuación se observan las dos gráficas de las 1.764 soluciónes exploradas en el modelo de la política óptima para este sistema. El mantenimiento selectivo influenciado por las inspecciónes demuestra ser efectivo para los casos donde se plantee realizarse mantenimiento preventivo en intervalos cortos inspeccionando todos los equipos que tengan un número de períodos de funcionamiento sin fallos corto pero se detecte que no sea necesario aplicarles mantenimiento por estar en buen estado, como no ocurría en el modelo anterior.

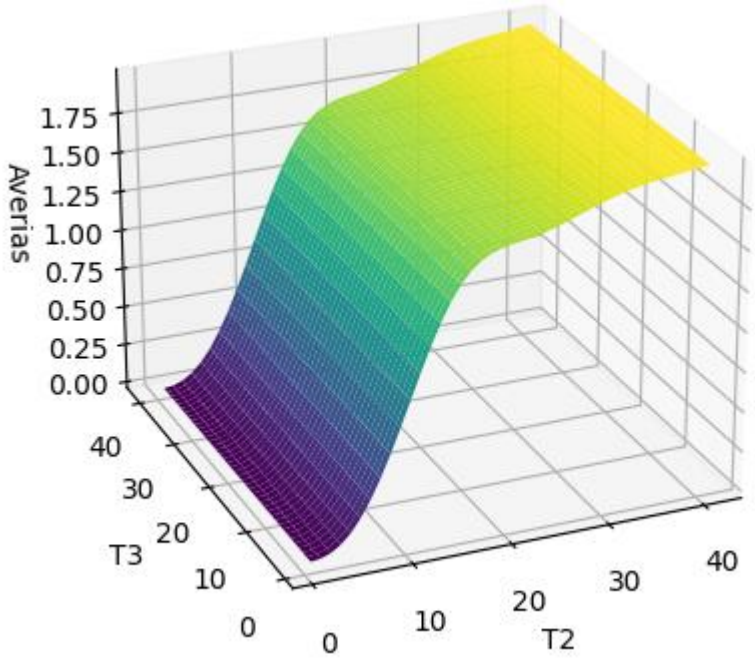

Figura 4-7. Gráfica de averías en función de T2 y T3.

La superficie que representa el número de averías en relación a las variables T2 y T3 principalmente en este modelo es siempre creciente, con un ligero cambio de pendiente cerca del valor de T2=20.

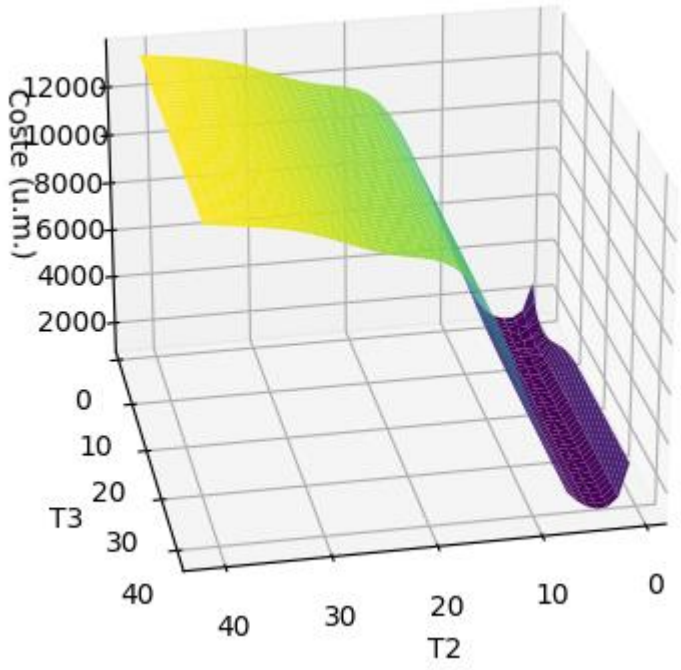

Figura 4-8. Gráfica de costes totales en función de T2 y T3.

4.3 Análisis de resultados

Se plantearon 3 análisis del sistema utilizando la herrramienta desarrollada para el caso práctico definido al comienzo de éste capítulo, en cada uno se utilizó una escala para los períodos temporales distinta teniendo en cuenta que se desea tener una escala lo más cercana a la original por lo que este planteamiento puede ser interesante para medir cuánto pueden varíar los resultados para cada escala y saber el nivel de confianza que se puede tener en los resultados de tener que aplicar la misma metodología para otro caso real.

En esta sección se pondrá en contraste los resultados de la política óptima de cada experimento realizado, para observar la variación entre la elección de la escala de los períodos temporales elementales para cada modelo. Con ésto se medirá el nivel de fiabilidad de los resultados.

Se presentan en los siguientes epígrafes las soluciónes de los otros experimentos.

4.3.1 Aplicación de la herramienta al caso práctico utilizando como período temporal grupos de 80 horas

En este caso tomando de punto de partida como en el caso anterior el número máximo de horas que puede operar un equipo sin fallar (T_1) y escalandolo a grupos de 80 horas se tiene 26.06 períodos que se aproximará a un valor de **27** períodos temporales.

El parámetro de escala de la curva Weibull utilizada para determinar la matriz de probabilidades de fallo se transforma a la escala de este estudio con un valor igual a **16.471**.

Figura 4-9. Resultados de la resolución de los cuatro modelos siguiendo el criterio objetivo, planteamiento de escala en grupos de 80 horas.

Al igual que en el experimento a escala de grupos de 50 horas el mejor caso de menor coste es el mantenimiento preventivo en función del número de períodos de funcionamiento sin fallos y aplicando una inspección previa.

4.3.2 Aplicación de la herramienta al caso práctico utilizando como período temporal grupos de 100 horas

Los valores se determinan siguiendo la misma logica que en los casos anteriores, por lo que se determina para el número máximo de períodos temporales que puede operar un equipo sin fallar (T_1) 20.85 períodos que se aproximará a **21** períodos a efectos prácticos.

El parámetro de escala de la curva Weibull en la escala necesaria para este experimento sería igual a **13.1827**.

Figura 4-10. Resultados de la resolución de los cuatro modelos siguiendo el criterio objetivo, planteamiento de escala en grupos de 100 horas

4.3.3 Comparación de resultados

De los resultados que se observaron, es de esperarse que a medida que la escala es más cercana a la escala real pueden encontrarse soluciones óptimas donde el coste total sea menor, pero a pesar de las variaciónes en los valores de costes totales de cada política y número de averías esperadas por período las soluciónes óptimas se siguien encontrando en sectores cercanos que pueden apreciarse y compararse con las gráficas de los valores de costes y averías en función de T2 y T3.

Especialmente el caso de la política óptima para éste se observa que la variable que más cambió fue T3, pero justo para el espacio de soluciones exploradas de esta política destaca una hendidura (véase Figura 4-11) donde la variación de el coste total dependiendo de T3 es muy pequeña a diferencia de los otros modelos de mantenimiento preventivo, ésto independientemente de el experimento, lo que explicaría el grán cambio en el valor de T3.

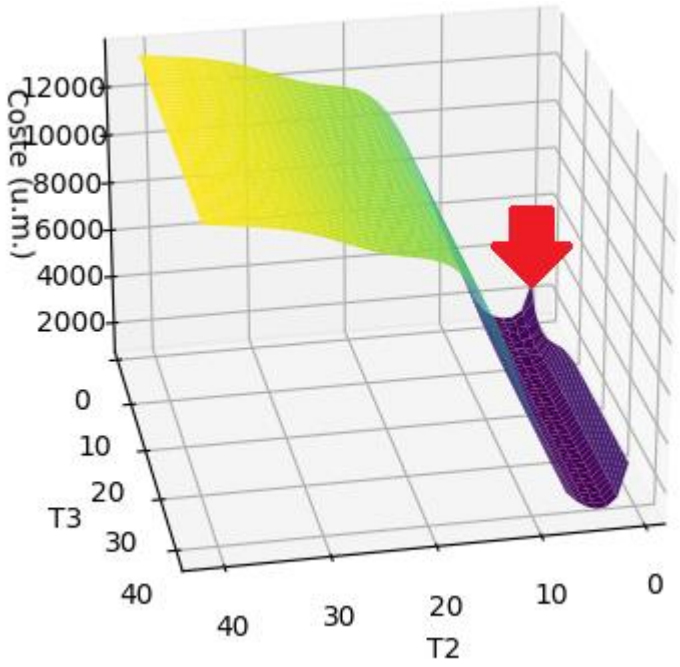

Figura 4-11. Gráfica de costes totales en función de T2 y T3. Hendidura formada en el espacio de soluciones de T2=0 hasta T2=10 para todos los valores de T3

En las siguientes tablas se observan los resultados de cada política para cada experimento llevado al valor de escala real en **horas**.

Tabla 4-1. Soluciones óptimas para cada política del experimento a escala de grupos de 50 horas en escala original.

Política de mantenimiento (escala de grupos)	T_2	T_3	Coste total	Averías (por período, casos preventivos)
Correctiva	-	-	13199.99	2.3157
Preventiva Total	350	-	7245.47	0.006254
Preventivo basado en tiempo de func.	150	2050	1007.29	0.0005826
Preventivo basado en tiempo de func. + Insp	200	2050	1003.67	0.001384

Al ser el valor más cercano a la escala del caso prácito original se usarán estas soluciónes como valores de comparación con los resultados de los otros dos experimentos mostrados en las tablas 4-2 y 4-3.

Tabla 4-2. Soluciones óptimas para cada política del experimento a escala de grupos de 80 horas en escala original.

Política de mantenimiento (escala de grupos)	T_2	T_3	Coste total	Averías (por período, casos preventivos)
Correctiva	-	-	17844.27	3.1305
Preventiva Total	400	-	9836.58	0.004798
Preventivo basado en tiempo de func.	240	2080	1604.37	0.001099
Preventivo basado en tiempo de func. + Insp	160	240	1569.577	0.0002701

Tabla 4-3. Soluciones óptimas para cada política del experimento a escala de grupos de 100 horas en escala original.

Política de mantenimiento (escala de grupos)	T_2	T_3	Coste total	Averías (por período, casos preventivos)
Correctiva	-	-	20515.96	3.599
Preventiva Total	500	-	11271.92	0.006327
Preventivo basado en tiempo de func.	200	2000	2048.58	0.0003653
Preventivo basado en tiempo de func. + Insp	200	300	1891.50	0.0003653

Comparando los resultados en ambos casos, los tres experimentos llevados a la escala original estan aproximadamente en rangos cercanos de valores. Cuanto más pequeño es el grupo de horas, el programa encuentra una solución de coste total mejor, ya que al tener un grupo de horas que representan una solución óptima y al subdividir ese grupo en valores individuales (si el modelo lo permite) se puede encontrar una solución mejor en alguno de esos valores.

En el análisis comparativo de las soluciones, debe tomarse en cuenta que se cambió la escala de las variables de desición y por lo tanto el valor de T1, en cada nueva escala el nuevo valor de resultaba en un número no entero, valor no permitido para T1 según las hipótesis de los modelos, por lo que una solución a este problema fue redondear ese valor al entero superior. Esta operación necesaria puede considerarse también como un factor a tomar en cuenta para evaluar cuánto fue la variación de los resultados al caso real.

5 CONCLUSIÓNES

En el presente proyecto de investigación se abarcó el proceso de desarrollo de una herramienta para el apoyo de toma de desiciónes, programación de recursos y en estudio de fiabilidad de sistemas empleando el uso de cuatro modelos cuantitativos definidos en un trabajo de investigación de la Universidad Politécnica de Madrid[1], con el objetivo de determinar cúal es la mejor política de mantenimiento, siguiendo como criterio objetivo minimizar los costes totales en la rutina elegida para el sistema. Se provee adicionalmente de funcionalidades para observar soluciones alternativas para los modelos variando el valor de las actividades de decisión del modelo y estudiar la fiabilidad del sistema basándose en su histórico de fallos.

Tras el desarrollo del sistema informático y emplear su uso en un caso práctico real se obtuvieron las siguientes conclusiónes:

- La herramienta emplea de manera efectiva y precisa el uso de los modelos Markovianos descritos en el Capítulo 2, ofreciendo resultados coherentes que fueron contrastados razonadamente en un ejemplo en el Capítulo 3 con otra investigación que empleó su uso para un determinado caso de ejemplo[10].
- Permite al usuario observar también todos los otros valores que puede tomar el coste total y el número de averías para cada valor de las variables de decisión del sistema, que será información adicional para la toma de desiciónes con el conocimiento de distintos escenarios.
- En el caso práctico presentado en el Capítulo 4 se mostró cuánto puede varíar la solución si se cambiase la escala de medida de las variables de decisión del sistema, para analizar la confianza en los resultados obtenidos de la herramienta en casos similares.
- Se ha diseñado la funcionalidad que permite generar una matriz de probabilidades de transición a partir de una función de distribución de probabilidad Exponencial o de una Weibull a partir de el ingreso de los parámetros de las curvas en los campos correspondientes en la pantalla de datos del análisis.
- A partir del estimador estadístico Rango de Mediana y el método de mínimos cuadrados el programa desarrollado permite al usuario tener una noción de la fiabilidad de su sistema calculando los valores de los parámetros de una curva de probabilidad Weibull, usando como datos de referencia el tiempo de operación entre fallos e intervenciones por mantenimiento de un equipo.
- Se empleó de manera exitosa el uso de el lenguaje de programación Python para el diseño de una interfaz gráfica amigable e intuitiva, y el uso de sus herramientas de desarrollo para análisis y resolución de problemas matemáticos complejos que resultaron clave en este proyecto.

6 TRABAJO FUTURO

En los casos de uso para aplicaciones de la herramienta, los más comunes serán con seguridad aquellos donde el usuario solamente disponga de una base de datos con el historico de fallos y mantenimiento preventivo de los equipos, por lo que optará por calcular la matriz de transmisión necesaria para los analisis del programa utilizando el historial de fallos, con lo que se obtiene una matriz de transición generada a partir de una función de distribución de probabilidad Weibull de dos parámetros, una forma de complementar el programa es mejorando la precisión de los análisis para estos casos calculando la matriz a partir de una curva Weibull de 3 parámetros que, visto en la investigación de referencia [4], es ligeramente más precisa que la de 2 parámetros.

También la opción que permita estudiar cómo cambiaría la solución final si se varian ligeramente los costes de reparación, mantenimiento e inspección.

ANEXOS

A.1 Código fuente de Manther 4.0

main.py

```
1. import tkinter as tk
2. from tkinter import font
3. from tkinter import filedialog
4.
5. from numpy import genfromtxt, reshape
6. import pDatos
7. import pResultados
8.
9. class MainWindow(tk.Frame):
10.
11.      #constructor de clase
12.      def __init__(self, parent):
13.          super().__init__(parent)
14.          self.parent = parent
15.          self.parent.title("Pagina inicial")
16.          fuente = font=('Arial', 11)
17.
18.          botonNuevoAnalisis = tk.Button(self.parent, text = "Nuevo anal
   isis", font=fuente, bg = "lime green",width = 25 ,command = self.pantal
   laDatos)
19.          botonNuevoAnalisis.place(x=250, y=150)
20.
21.          botonCargarAnalisis = tk.Button(self.parent, text="Cargar", fo
   nt=fuente, bg="red", width=25, command = self.cargarAnalisis)
22.          botonCargarAnalisis.place(x=250, y=200)
23.
24.          labelTitulo = tk.Label(self.parent, font=('Arial', 26), text =
    'Manther 4.0', bg='LightCyan3')
25.          labelTitulo.place(x=275, y=50)
26.
27.          botonAyuda = tk.Button(self.parent, text = "Ayuda", font=fuent
   e, bg = "salmon",width = 25 ,command = self.ayudaMessageBox)
28.          botonAyuda.place(x=250, y=250)
29.
30.      def pantallaDatos(self):
31.          #llamar a la pantalla de introduccion de datos
32.          pDatos.VentanaDatos(self.parent)
33.
34.      def cargarAnalisis(self):
35.          #cargar datos de un analisis ya realizado y recalcularlo
36.          csvfile = genfromtxt(filedialog.askopenfilename(), delimiter='
   ')
37.          pResultados.VentanaResultados(self.parent, csvfile[0], csvfile
   [1], csvfile[2],
38.          csvfile[3], csvfile[4], csvfile[5], csvfile[6], csvfile[7], cs
   vfile[8:].reshape(int(csvfile[6]), 1))
```

```
39.          self.destroy()
40.
41.
42.      def ayudaMessageBox(self):
43.          master = tk.Tk()
44.          mensaje = "Bienvenido al programa Manther 4.0 \t\n"\
45.              "\t\n Software para el analisis de politicas de mantenimie
   nto ideales para conjuntos de equipos homogeneos.\t\n"\
46.              "\t\n En la opcion de cargar analisis solo se pueden carga
   r archivos .csv generados del propio programa."
47.          msj = tk.Message(master, text=mensaje)
48.          msj.config(font=('Arial', 12), bg='light goldenrod')
49.          msj.pack()
50.          tk.mainloop()
51.
52.  if __name__ == "__main__":
53.      #instancia de app
54.      root = tk.Tk()
55.      root.geometry('700x500')
56.      root.configure(background = 'LightCyan3') #color de fondo
57.      app = MainWindow(root)
58.      app.mainloop()
```

main.py

pDatos.py

```
1. import tkinter as tk
2. from tkinter import filedialog
3. from tkinter import font
4. import csv
5. from numpy import dtype, float64, genfromtxt
6. import pResultados
7. import cWeibull
8. import cExponencial
9. import cRangoMedianas
10.
11.  class VentanaDatos(tk.Toplevel):
12.      def __init__(self, parent):
13.          super().__init__(parent)
14.          self.parent = parent
15.          self.title("Ventana de Datos")
16.          self.geometry('1000x500')
17.          fondo = 'LightCyan3'
18.          self.configure(bg=fondo)
19.          fuente = font = ('Arial', 11)
20.          self.parent.withdraw()
21.
22.          #variables del modelo
23.          self.nombre = tk.StringVar(self)
24.          self.costeRep = tk.StringVar(self)
25.          self.costeMant = tk.StringVar(self)
26.          self.costeInsp = tk.StringVar(self)
27.          self.prob = tk.StringVar(self)
28.          self.insp_B_M = tk.StringVar(self)
29.          self.insp_M_B = tk.StringVar(self)
30.          self.T1 = tk.StringVar(self)
31.          self.N = tk.StringVar(self)
32.          self.alpha = tk.StringVar(self)
```

```
33.          self.betha = tk.StringVar(self)
34.          self.k = tk.StringVar(self)
35.          self.L = tk.StringVar(self)
36.          self.opcion = tk.IntVar(self)
37.
38.          #Labels y Entradas
39.          Label = tk.Label(self, font=('Arial', 14), bg = 'gray', text =
    'Datos del equipo', padx = 15, pady = 20, width = 40)
40.          Label.grid(row = 0, column = 0, columnspan = 2)
41.
42.
43.          LabelNombre = tk.Label(self, font=fuente, text = 'Nombre:', bg
   =fondo)
44.          LabelNombre.grid(row = 2, column = 0, pady = 10)
45.          nombre = tk.Entry(self, textvariable = self.nombre)
46.          nombre.grid(row = 2, column = 1, pady = 10)
47.
48.          LabelReparacion = tk.Label(self, font=fuente, text = 'Costes d
   e reparacion:', bg=fondo)
49.          LabelReparacion.grid(row = 4, column = 0, pady = 10)
50.          C1 = tk.Entry(self, textvariable = self.costeRep)
51.          C1.grid(row = 4, column = 1, pady = 10)
52.
53.          LabelMantenimiento = tk.Label(self, font=fuente, text = 'Coste
   s de mantenimiento:', bg=fondo)
54.          LabelMantenimiento.grid(row = 6, column = 0, pady = 10)
55.          C2 = tk.Entry(self, textvariable = self.costeMant)
56.          C2.grid(row = 6, column = 1, pady = 10)
57.
58.          LabelInspeccion = tk.Label(self, font=fuente, text = 'Coste de
    inspeccion:', bg=fondo)
59.          LabelInspeccion.grid(row = 8, column = 0)
60.          C3 = tk.Entry(self, textvariable = self.costeInsp)
61.          C3.grid(row = 8, column = 1, pady = 10)
62.
63.
64.          lf = tk.LabelFrame(self, text = 'Calidad de las inspecciones:'
   , bg=fondo)
65.          lf.grid(row = 10, column = 0, columnspan = 2, pady = 10)
66.          self.insp_B_M.set("% B-->M...")
67.          self.insp_M_B.set("% M-->B...")
68.          B_M = tk.Entry(lf, textvariable = self.insp_B_M, width=10)
69.          B_M.grid(row = 0, column = 0, padx = 15, pady = 10)
70.          M_B = tk.Entry(lf, textvariable = self.insp_M_B, width=10)
71.          M_B.grid(row = 0, column = 1, padx = 15, pady = 10)
72.
73.          botonAyuda = tk.Button(self, font=fuente, text='Ayuda', bg='go
   ld', command= self.ayudaMessageBox)
74.          botonAyuda.grid(row = 1, column = 4, columnspan=2)
75.
76.          botonProb = tk.Button(self, font=fuente, text = 'Agregar proba
   bilidad de fallos', width = 25, command = self.incluirProb, bg='turquoi
   se')
77.          botonProb.grid(row = 2, column = 4, columnspan=2)
78.
79.          LabelT1 = tk.Label(self, font=fuente, text = 'T1: ', bg=fondo)
80.          LabelT1.grid(row = 4, column = 4)
81.          T1 = tk.Entry(self, textvariable = self.T1, width=10)
82.          T1.grid(row = 4, column = 5)
83.
84.          LabelN = tk.Label(self, font=fuente, text = 'Número de equipos
   :', bg=fondo)
```

```
85.          LabelN.grid(row = 6, column = 4)
86.          N = tk.Entry(self, textvariable = self.N)
87.          N.grid(row = 6, column = 5)
88.
89.          lf_weibull = tk.LabelFrame(self, text='Weibull', font=fuente,
   bg=fondo)
90.          lf_weibull.grid(row=8, column=4, pady=10)
91.          Alpha_Label = tk.Label(lf_weibull, font=fuente, text='Alpha:',
    bg=fondo)
92.          Alpha_Label.grid(row=0, column=0)
93.          A = tk.Entry(lf_weibull, textvariable= self.alpha, width=10)
94.          A.grid(row=0, column=1)
95.          Betha_Label = tk.Label(lf_weibull, font=fuente, text='Betha:',
    bg=fondo)
96.          Betha_Label.grid(row=0, column=2)
97.          B = tk.Entry(lf_weibull, textvariable= self.betha, width=10)
98.          B.grid(row=0, column=3)
99.          K_Label = tk.Label(lf_weibull, font=fuente, text='Gamma:', bg=
   fondo)
100.         K_Label.grid(row=0, column=4)
101.         k = tk.Entry(lf_weibull, textvariable= self.k, width=10)
102.         k.grid(row=0, column= 5)
103.         botonWeibull = tk.Radiobutton(lf_weibull, variable= self.opcio
   n, value=1, bg=fondo)
104.         botonWeibull.grid(row=0, column=6)
105.
106.         lf_exponencial = tk.LabelFrame(self, text='Exponencial',font=f
   uente, bg=fondo)
107.         lf_exponencial.grid(row=9, column=4, pady=10)
108.         L_Label = tk.Label(lf_exponencial, font=fuente, text='Lambda:'
   , bg=fondo)
109.         L_Label.grid(row=0, column=0)
110.         L = tk.Entry(lf_exponencial, textvariable= self.L, width=10)
111.         L.grid(row=0, column=1)
112.         botonExponencial = tk.Radiobutton(lf_exponencial, variable= se
   lf.opcion, value=2, bg=fondo)
113.         botonExponencial.grid(row=0, column=2)
114.
115.         botonCalWeibull = tk.Button(self, font=fuente, text='Calcular
   F.D. Weibull', width=25, command= self.RangoDeMedianas, bg='turquoise')
116.         botonCalWeibull.grid(row = 10, column = 4, columnspan=2)
117.
118.         botonCalcular = tk.Button(self, font=fuente, text = 'Calcular'
   ,width = 20, command = self.calcular, bg = 'salmon')
119.         botonCalcular.grid(row = 11, column = 4, columnspan=2)
120.
121.
122.     def calcular(self):
123.
124.         if(self.opcion.get()==1):
125.             self.calWeibull()
126.
127.         if(self.opcion.get()==2):
128.             self.calExponencial()
129.
130.         pResultados.VentanaResultados(self.parent, self.nombre.get(),
   self.costeRep.get(), self.costeMant.get(),
131.                 self.costeInsp.get(),
132.                 self.insp_B_M.get(),
133.                 self.insp_M_B.get(),
134.                 self.T1.get(),
135.                 self.N.get(), self.prob)
```

```
136.
137.         with open(self.nombre.get()+'.csv', 'w') as file:
138.             writer = csv.writer(file)
139.             writer.writerow(self.nombre.get())
140.             writer.writerow([self.costeRep.get()])
141.             writer.writerow([self.costeMant.get()])
142.             writer.writerow([self.costeInsp.get()])
143.             writer.writerow([self.insp_B_M.get()])
144.             writer.writerow([self.insp_M_B.get()])
145.             writer.writerow([self.T1.get()])
146.             writer.writerow([self.N.get()])
147.             for i in range(0, int(self.T1.get())):
148.                 writer.writerow([self.prob.item(i)])
149.
150.
151.         self.destroy()
152.
153.     def incluirProb(self):
154.         cvsfile = genfromtxt(filedialog.askopenfilename(), delimiter=
    ,')
155.        self.prob = cvsfile
156.
157.     def calWeibull(self):
158.
159.         self.prob = cWeibull.calcula(self.alpha.get(), self.betha.get
    (), self.k.get(), self.T1.get())
160.
161.     def calExponencial(self):
162.
163.         self.prob = cExponencial.calcula(self.L.get(), self.T1.get())
164.
165.     def RangoDeMedianas(self):
166.         cvsfile = genfromtxt(filedialog.askopenfilename(), delimiter=
    ,', dtype='str')
167.
168.         self.alpha, self.betha = cRangoMedianas.calcular(cvsfile)
169.         self.weibullMessageBox(self.alpha, self.betha)
170.
171.     def weibullMessageBox(self, alpha, betha):
172.         master = tk.Tk()
173.         mensaje = "Parametros de FD Weibull calculados con exito:\t\n"
    \
174.             "\t\n Alpha: "+str(alpha)+"\t\n"\
175.             "\t\n Betha: "+str(betha)+"\t\n"
176.         msj = tk.Message(master, text=mensaje)
177.         msj.config(font=('Arial', 12), bg='light goldenrod')
178.         msj.pack()
179.         tk.mainloop()
180.
181.     def ayudaMessageBox(self):
182.         master = tk.Tk()
183.         mensajeAyuda = "DATOS: \t\n"\
184.             "\t\n Después de ejecutarse un nuevo analisis se creará en
     la carpeta del archivo de ejecucion un archivo .csv con el nombre indi
    cado en el campo de nombre\t\n"\
185.             "\t\n el archivo contendrá los datos ingresados en los cam
    pos de la pantalla de datos para posteriormete si lo desea el usuario c
    argar denuevo los resultados desde la pantalla inicial desde la opcion
    'Cargar analisis'\t\n"\
186.             "\t\n Los costes de reparacion y mantenimieno: deben inclu
    ir todos los gastos asociados a la operacion de reparacion (piezas de r
    ecambio, mano de obra, tiempo productivo perdido)\t\n"\
```

```
187.             "\t\n Los costes de las inspecciones incluyen solo los cos
   tes del trabajo de inspeccion con parámetros\t\n"\
188.             "\t\n %B-
   >M (probabilidad de que un equipo definido como bueno esté en mal estad
   o)\t\n"\
189.             "\t\n %M-
   >B (probabilidad de que un equipo definido como malo en una inspección
   esté en buen estado)\t\n"\
190.             "\t\n Matriz de transicion: debe estar en formato de archi
   vo .csv ejemlo: [0.125, 0.15, 0.25, 0.5, 0.8, 1] \t\n"\
191.             "\t\n Funcion 'Calcular Weibull': se debe agregar la base
   de datos con los tiempo de mantenimiento preventivo y fallos en un arch
   ivo .csv dispuestos como en el ejemplo: \t\n"\
192.             "\t\n N° de medida | Motivo de parada mantenimiento (R), f
   allo (F) | Horas de operacion \t\n"\
193.             "\t\n             1      |             R
                   |        1125         \t\n"\
194.             "\t\n             2      |             R
                   |        480           \t\n"\
195.             "\t\n             3      |             F
                   |        1080         \t\n"\
196.             "\t\n             4      |             R
                   |        900          \n"
197.
198.         msj = tk.Message(master, text=mensajeAyuda)
199.         msj.config(font=('Arial', 12), bg='light goldenrod')
200.         msj.pack()
201.         tk.mainloop()
```

pDatos.py

pResultados.py

```
1. import tkinter as tk
2. from tkinter import ttk
3. from tkinter import font
4. from tkinter.constants import HORIZONTAL
5. import matplotlib.pyplot as plt
6. from mpl_toolkits.mplot3d import Axes3D
7.
8. import cCorrectivo
9. import cPreventivo1
10.  import cPreventivo2
11.  import cPreventivo3
12.
13.  class VentanaResultados(tk.Toplevel):
14.      def __init__(self, parent, nombre, C1, C2, C3, B_M, M_B, T1, N, p)
   :
15.          super().__init__(parent)
16.          self.parent = parent
17.          self.title("Ventana de Resultados")
18.          self.geometry('1050x700')
19.          fondo = 'LightCyan3'
20.          self.configure(bg=fondo)
21.          fuente = font = ('Arial', 11)
22.
23.          self.parent.withdraw()
24.
25.          self.C1 = int(C1)
```

```
26.          self.C2 = int(C2)
27.          self.C3 = int(C3)
28.          self.B_M = float(B_M)
29.          self.M_B = float(M_B)
30.          self.T1 = int(T1)
31.          self.N = int(N)
32.          self.p = p
33.
34.          costeCorr, averiasCorr = self.calcularCor(self.C1, self.T1, se
   lf.N, self.p)
35.          costePrev1, T2Prev1, averiasPrev1, graf_averias_Prev1, graf_Ti
   empoCoste_Prev1 = self.calcularPre(self.C1, self.C2, self.T1, self.N, s
   elf.p)
36.          costePrev2, T2Prev2, T3Prev2, averiasPrev2, graf_TiempoCoste_P
   rev2, graf_averias_Prev2 = self.calcularPre2(self.C1, self.C2, self.T1,
    self.N, self.p)
37.          costePrev3, T2Prev3, T3Prev3, averiasPrev3, graf_TiempoCoste_P
   rev3, graf_averias_Prev3 = self.calcularPre3(self.C1, self.C2, self.C3,
    self.T1, self.N, self.p, self.B_M, self.M_B)
38.
39.          Label = tk.Label(self, bg = 'gray', text = 'Resultados', font=
   ('Arial', 14), padx = 15, pady = 20, width=40)
40.          Label.grid(row = 0, column = 0, columnspan=1)
41.
42.
43.          labelFCorrectivo = tk.Label(self, text = 'Correctivo', bg='gol
   d', font=('Arial', 12))
44.          labelFCorrectivo.grid(row=2, column=0)
45.          labelCorrCost = tk.Label(self, font=fuente, text='Coste total:
    '+ str(costeCorr), bg=fondo)
46.          labelCorrCost.grid(row=3, column=1, pady=10)
47.          labelCorrAveria = tk.Label(self, font=fuente, text='N° Averias
   : '+ str(averiasCorr), bg=fondo)
48.          labelCorrAveria.grid(row=4, column=1, pady=10)
49.
50.          s = ttk.Separator(self, orient=HORIZONTAL)
51.          s.grid(row=5, column=0, columnspan=99, sticky=(tk.W, tk.E))
52.
53.
54.          LabelFPrev1 = tk.Label(self, text='Preventivo Total',bg='gold'
   , font=('Arial', 12))
55.          LabelFPrev1.grid(row=6, column=0)
56.          labelPrev1Cost = tk.Label(self, font=fuente, text='Coste total
   : '+ str(costePrev1), bg=fondo)
57.          labelPrev1Cost.grid(row=7, column=1, pady=10)
58.          labelPrev1Averia = tk.Label(self, font=fuente, text='N° Averia
   s: '+ str(averiasPrev1), bg=fondo)
59.          labelPrev1Averia.grid(row=8, column=1, pady=10)
60.          LabelPrev1_T2 = tk.Label(self, font=fuente, text='T2: '+ str(T
   2Prev1), bg=fondo)
61.          LabelPrev1_T2.grid(row=7, column=0, pady=10)
62.
63.          #boton de la grafica
64.          botonGraficaAverias_prev1 = tk.Button(self, font = fuente, tex
   t='Grafica Averias en cada T', command = lambda: self.graficaAverias_pr
   ev1(graf_averias_Prev1))
65.          botonGraficaAverias_prev1.grid(row=7, column=2, pady=10)
66.
67.          botonGraficaTC_prev1 = tk.Button(self, font=fuente, text='Graf
   ica de T2 vs Coste', command= lambda: self.graficaTC_prev1(graf_TiempoC
   oste_Prev1))
68.          botonGraficaTC_prev1.grid(row=8, column=2, pady=10)
```

```
69.
70.         s1 = ttk.Separator(self, orient=HORIZONTAL)
71.         s1.grid(row=9, column = 0, columnspan=99, sticky=(tk.W, tk.E))
72.
73.         LabelFPrev2 = tk.Label(self, text='Preventivo basado en tiempo
    de funcionamiento sin fallos', bg='gold', font=('Arial', 12))
74.         LabelFPrev2.grid(row=10,  column=0)
75.         labelPrev2Cost = tk.Label(self, font=fuente, text='Coste total
   : '+ str(costePrev2), bg=fondo)
76.         labelPrev2Cost.grid(row=11, column=0, pady=10)
77.         labelPrev2Averia = tk.Label(self, font=fuente, text='N° Averia
   s: '+ str(averiasPrev2), bg=fondo)
78.         labelPrev2Averia.grid(row=11, column=1, pady=10)
79.         LabelPrev2_T2 = tk.Label(self, font=fuente, text='T2: '+ str(T
   2Prev2), bg=fondo)
80.         LabelPrev2_T2.grid(row=12, column=1, pady=10)
81.         LabelPrev2_T3 = tk.Label(self, font=fuente, text='T3: '+ str(T
   3Prev2), bg=fondo)
82.         LabelPrev2_T3.grid(row=13, column=1, pady=10)
83.         #boton de la grafica
84.         botonGraficaAverias_prev2 = tk.Button(self, font=fuente, text=
   'Grafica Averias en cada T', command= lambda: self.graficaAverias_prev2
   (graf_averias_Prev2))
85.         botonGraficaAverias_prev2.grid(row=12, column=2, pady=10)
86.
87.         botonGraficaTC_prev2 = tk.Button(self, font=fuente, text='Graf
   ica de T2 vs T3 vs Coste', command= lambda: self.graficaTC_prev2(graf_T
   iempoCoste_Prev2))
88.         botonGraficaTC_prev2.grid(row=13, column=2, pady=10)
89.
90.         s2 = ttk.Separator(self, orient=HORIZONTAL)
91.         s2.grid(row=14, column=0, columnspan=99, sticky=(tk.W, tk.E))
92.
93.
94.         LabelFPrev3 = tk.Label(self, text='Preventivo basado en tiempo
    de func + Insp', bg='gold', font=('Arial', 12))
95.         LabelFPrev3.grid(row=15,  column=0)
96.         labelPrev3Cost = tk.Label(self, font=fuente, text='Coste total
   : '+ str(costePrev3), bg=fondo)
97.         labelPrev3Cost.grid(row=16, column=0, pady=10)
98.         labelPrev3Averia = tk.Label(self, font=fuente, text='N° Averia
   s: '+ str(averiasPrev3), bg=fondo)
99.         labelPrev3Averia.grid(row=16, column=1, pady=10)
100.        LabelPrev3_T2 = tk.Label(self, font=fuente, text='T2: '+ str(T
   2Prev3), bg=fondo)
101.        LabelPrev3_T2.grid(row=17, column=1, pady=10)
102.        LabelPrev3_T3 = tk.Label(self, font=fuente, text='T3: '+ str(T
   3Prev3), bg=fondo)
103.        LabelPrev3_T3.grid(row=18, column=1, pady=10)
104.        #boton de la grafica
105.        botonGraficaAverias_prev3 = tk.Button(self, font=fuente, text=
   'Grafica Averias en cada T', command= lambda: self.graficaAverias_prev3
   (graf_averias_Prev3))
106.        botonGraficaAverias_prev3.grid(row=17, column=2, pady=10)
107.
108.        botonGraficaTC_prev3 = tk.Button(self, font=fuente, text='Graf
   ica de T2 vs T3 vs Coste', command= lambda: self.graficaTC_prev3(graf_T
   iempoCoste_Prev3))
109.        botonGraficaTC_prev3.grid(row=18, column=2, pady=10)
110.
111.        # exit button to end process and destroy
```

```
112.         exit_button = tk.Button(self, font=fuente, text='Salir', width
   =15, command= lambda: self.salir())
113.         exit_button.grid(row=19, column=3)
114.
115.     def calcularCor(self, C1, T1, N, p):
116.         #llamar cCorrectivo
117.
118.         coste, Np = cCorrectivo.calcular(C1, T1, N, p)
119.         return coste, Np
120.
121.     def calcularPre(self, C1, C2, T1, N, p):
122.         #llamar cPreventivo
123.
124.         coste, T2, averias, graf_Averias, graf_TiempoCoste = cPreventi
   vo1.calcular(C1, C2, T1, N, p)
125.         return coste, T2, averias, graf_Averias, graf_TiempoCoste
126.
127.
128.     def calcularPre2(self, C1, C2, T1, N, p):
129.         #llamar cPreventivo2
130.         costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias = c
   Preventivo2.calcular(C1, C2, T1, N, p)
131.         return costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Aver
   ias
132.
133.     def calcularPre3(self, C1, C2, C3, T1, N, p, B_M, M_B):
134.         #llamar cPreventivo3
135.         costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Averias = c
   Preventivo3.calcular(C1, C2, C3, T1, N, p, B_M, M_B)
136.         return costeMin, T2, T3, nAverias, graf_TiempoCoste, graf_Aver
   ias
137.
138.     def graficaTC_prev1(self, g):
139.         fig, ax = plt.subplots()
140.         ax.plot(g[1:, 0], g[1:, 1], color='blue', marker='o')
141.         plt.xlabel('T(periodos)')
142.         plt.ylabel('Coste (u.m.)')
143.         plt.show()
144.
145.     def graficaAverias_prev1(self, g):
146.         fig, ax = plt.subplots()
147.         ax.plot(g[1:, 0], g[1:, 1], color='blue', marker='o')
148.         plt.xlabel('T(periodos)')
149.         plt.ylabel('Averias')
150.         plt.show()
151.
152.     def graficaTC_prev2(self, g):
153.         fig = plt.figure()
154.         ax = fig.add_subplot(projection='3d')
155.         Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
   2], cmap='viridis')
156.         ax.set_xlabel('T2')
157.         ax.set_ylabel('T3')
158.         ax.set_zlabel('Coste (u.m.)')
159.         plt.show()
160.
161.     def graficaAverias_prev2(self, g):
162.         fig = plt.figure()
163.         ax = fig.add_subplot(projection='3d')
164.         Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
   2], cmap='viridis')
165.         ax.set_xlabel('T2')
```

```
166.         ax.set_ylabel('T3')
167.         ax.set_zlabel('Averias')
168.         plt.show()
169.
170.     def graficaTC_prev3(self, g):
171.         fig = plt.figure()
172.         ax = fig.add_subplot(projection='3d')
173.         Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
    2], cmap='viridis')
174.         ax.set_xlabel('T2')
175.         ax.set_ylabel('T3')
176.         ax.set_zlabel('Coste (u.m.)')
177.         plt.show()
178.
179.     def graficaAverias_prev3(self, g):
180.         fig = plt.figure()
181.         ax = fig.add_subplot(projection='3d')
182.         Axes3D.plot_surface(ax, g[1:, 1:, 0], g[1:, 1:, 1], g[1:, 1:,
    2], cmap='viridis')
183.         ax.set_xlabel('T2')
184.         ax.set_ylabel('T3')
185.         ax.set_zlabel('Averias')
186.         plt.show()
187.
188.     def salir(self):
189.         self.destroy()
```

pResultados.py

cCorrectivo.py

```
1. import numpy as np
2.
3. def calcular(C1, T1, N, p):
4.     #declarar el vector ep, epi= ep' y la matriz P y Q
5.     _ep = np.zeros(T1)
6.     _epi = np.ones(T1)
7.     _Q = np.zeros((T1, T1))
8.     _P = np.zeros((T1, T1))
9.     ratio = 0
10.      Np = 0
11.      coste = 0
12.
13.
14.      for i in range(0, T1-1):
15.          _P[i, 0] = p[i]
16.          _P[i, i+1] = 1- p[i]
17.      _P[T1 -1, 0] = 1
18.      #se transforma la matriz de prob p en q=p-I
19.      _Q = _P
20.      for i in range(0, T1):
21.          _Q[i, i] -= 1
22.
23.      #Resolucion del sistema aplicando transformaciones de Gauss-
    Jordan para encontrar _ep
24.      _Q = _Q.T
25.      b = np.zeros(T1)
26.      b = b.reshape(T1, 1)
27.
```

```
28.      n = len(b)
29.
30.      for k in range(0, n):
31.          for i in range(k+1, n):
32.              ratio = _Q[i, k]/_Q[k, k]
33.              for j in range(k, n):
34.                  _Q[i, j] -= ratio*_Q[k, j]
35.              b[i] = b[i] - ratio*b[k]
36.
37.      b[n-1] = 1
38.
39.      _epi[n-1] = 1
40.
41.      for i in range(n-2, -1, -1):
42.          sum_j = 0
43.          for j in range(i+1, n):
44.              sum_j += _Q[i, j]*_epi[j]
45.          _epi[i] = (b[i] - sum_j)/_Q[i ,i]
46.
47.      suma = 0
48.      for i in range(0, n):
49.          suma = _epi[i] + suma
50.
51.      _ep = _epi*(1/suma)
52.
53.      #calculo de las reparaciones/averias totales
54.      Np = N*_ep[0]
55.      coste = Np*C1
56.
57.      return coste, Np
```

cCorrectivo.py

cPreventivo1.py

```
1. import numpy as np
2.
3. def calcular(C1, C2, T1, N, p):
4.      #declaracion de matriz de mantenimiento y transicion, vectores
5.      _M = np.zeros((T1, T1))
6.      _P = np.zeros((T1, T1))
7.      _e = np.zeros(T1)
8.      _e[0] = 1
9.
10.       graf_Averias = np.zeros((T1, 2))
11.       graf_TiempoCoste = np.zeros((T1, 2))
12.       costeCorr = 0
13.       costePrev = 0
14.       costeTotal = 0
15.       nAverias = 0
16.       costeMin = 0
17.
18.       for i in range(0, T1):
19.           _M[i, 0] = 1
20.
21.       for i in range(0, T1 -1):
22.           _P[i, 0] = p[i]
23.           _P[i, i+1] = 1 - p[i]
24.       _P[T1 -1, 0] = 1
```

```
25.      #calculo del coste de reparaciones
26.      costeMin = C1*T1*10000
27.      T2 = 0
28.      nAverias = 0
29.      for i in range(1, T1):
30.          suma = 0
31.
32.          for k in range(1, i+1):
33.              _e = _e@_P
34.              suma = _e.item(0) + suma
35.          costeCorr = (C1*N*suma)/i #coste medio por periodo
36.          costePrev = (N*C2)/i
37.          graf_Averias[i, 0] = i
38.          graf_Averias[i, 1] = N*suma/i
39.
40.          #coste total medio por periodo
41.          costeTotal = costeCorr+costePrev
42.          graf_TiempoCoste[i, 0] = i
43.          graf_TiempoCoste[i, 1] = costeTotal
44.          if(costeTotal < costeMin):
45.              costeMin = costeTotal
46.              T2 = i
47.              nAverias = N*suma/i
48.          _e = _e@_M
49.
50.
51.      return costeMin, T2, nAverias, graf_Averias, graf_TiempoCoste
```

pPreventivo1.py

pPreventivo2.py

```
1. import numpy as np
2.
3. def calcular(C1, C2, T1, N, p):
4.       #declaracion de matriz de mantenimiento y transicion, vectores
5.      _P = np.zeros((T1, T1))
6.
7.      graf_Averias = np.zeros((T1, T1, 3))
8.      graf_TiempoCoste = np.zeros((T1, T1, 3))
9.      costeCorr = 0
10.       costePrev = 0
11.       costeTotal = 0
12.       costeMin = 0
13.       nAverias = 0
14.       Averias = 0
15.
16.
17.       for i in range(0, T1 -1):
18.           _P[i, 0] = p[i]
19.           _P[i, i+1] = 1 - p[i]
20.       _P[T1 -1, 0] = 1
21.       costeMin = C1*T1*1000
22.       T2 = 0
23.       T3 = 0 #periodos de funcionamiento ininterrumpido
24.
25.       for i in range(1, T1):
26.           for j in range(1, T1):
27.               #vector de estado del sistema
```

```
28.             _epi = np.ones(T1)
29.             _ep = np.zeros(T1)
30.             _e = np.zeros(T1)
31.             _e[0] = 1
32.
33.             #matriz de mantenimiento
34.             _M = np.zeros((T1, T1))
35.             for k in range(1, T1):
36.                 if(k<=j):
37.                     _M[k, k] = 1
38.                 else:
39.                     _M[k, 0] = 1
40.
41.             _C = np.zeros((T1, T1))
42.             _C = _M@(np.linalg.matrix_power(_P, i))
43.
44.             for l in range(0, T1):
45.                 _C[l,l] -= 1
46.
47.             #Resolucion del sistema aplicando transformaciones de Gaus
    s-Jordan para encontrar _ep
48.             _C = _C.T
49.             b = np.zeros(T1)
50.             b = b.reshape(T1, 1)
51.             ratio = 0
52.
53.             n = len(b)
54.
55.             for k in range(0, n):
56.                 for ii in range(k+1, n):
57.                     ratio = _C[ii, k]/_C[k, k]
58.                     for jj in range(k, n):
59.                         _C[ii, jj] -= ratio*_C[k, jj]
60.                     b[ii] = b[ii] - ratio*b[k]
61.
62.             b[n-1] = 1
63.
64.             _epi[n-1] = 1
65.
66.             for ii in range(n-2, -1, -1):
67.                 sum_j = 0
68.                 for jj in range(ii+1, n):
69.                     sum_j += _C[ii, jj]*_epi[jj]
70.                 _epi[ii] = (b[ii] - sum_j)/_C[ii ,ii]
71.
72.             #vector de estado
73.             suma = 0
74.             for kk in range(0, n):
75.                 suma = _epi[kk] + suma
76.
77.             _ep = _epi*(1/suma)
78.
79.         #calculo del coste de mantenimiento preventivo medio por periodo
80.             suma = 0
81.             for ll in range(j, T1):
82.                 suma = _ep.item(ll) + suma
83.
84.             costePrev = (C2*N*suma)/i
85.
86.         #calculo del coste de mantenimiento correctivo medio por periodo
87.
88.             suma = 0
```

```
89.                 for kk in range(1, i+1):
90.                     _e = _e@_P
91.                     suma = _e.item(0) + suma
92.                 nAverias = suma*N
93.                 costeCorr = (C1*N*suma)/i
94.
95.                 costeTotal = costePrev+costeCorr
96.                 graf_TiempoCoste[i, j, 0] = i
97.                 graf_TiempoCoste[i, j, 1] = j
98.                 graf_TiempoCoste[i, j, 2] = costeTotal
99.
100.                graf_Averias[i, j, 0] = i
101.                graf_Averias[i, j, 1] = j
102.                graf_Averias[i, j, 2] = nAverias/i
103.
104.                if(costeTotal <  costeMin):
105.                    costeMin = costeTotal
106.                    T2 = i
107.                    T3 = j
108.                    Averias = nAverias/i
109.
110.    return costeMin, T2, T3, Averias, graf_TiempoCoste, graf_Averias
```

cPreventivo2.py

cPreventivo3.py

```
1. import numpy as np
2.
3. def calcular(C1, C2, C3, T1, N, p, B_M, M_B):
4.     _P = np.zeros((T1, T1))
5.     _W = np.zeros((T1, T1))
6.
7.     graf_Averias = np.zeros((T1, T1, 3))
8.     graf_TiempoCoste = np.zeros((T1, T1, 3))
9.     costeCorr = 0
10.      costePrev = 0
11.      costeInsp = 0
12.      costeTotal = 0
13.      costeMin = 0
14.      nAverias = 0
15.      Averias = 0
16.
17.      for i in range(0, T1 -1):
18.          _P[i, 0] = p[i]
19.          _P[i, i+1] = 1 - p[i]
20.      _P[T1 -1, 0] = 1
21.      costeMin = C1*T1*1000
22.      T2 = 0
23.      T3 = 0 #periodos de funcionamiento ininterrumpido
24.
25.      for i in range(1, T1):
26.          for j in range(1, T1):
27.
28.              _epi = np.ones(T1)
29.              _ep = np.zeros(T1)
30.              _e = np.zeros(T1)
31.              _e[0] = 1
32.
```

```
33.             #matriz de mantenimiento
34.             _M = np.zeros((T1, T1))
35.
36.             for k in range(0, T1 -2):
37.                 if(k<=j):
38.                     _M[k, k] = 1
39.                 else:
40.                     _M[k, 0] = (_P[k, 0]*(1-M_B)) + (_P[k, k+1]*B_M)
41.                     _M[k, k] = (_P[k, 0]*M_B) + (_P[k, k+1]*(1-B_M))
42.             _M[T1-1, 0] = 1
43.             #matriz W
44.
45.             for tt in range(0, T1-1):
46.                 for ss in range(0, T1-1):
47.                     if(tt <= j):
48.                         _W[tt, ss] = _P[tt, ss]
49.                     else:
50.                         _W [tt, 0] = (_P[tt, 0]*M_B)/((_P[tt, 0]*M_B)
   + (_P[tt, tt+1]*(B_M)))
51.                         _W[tt, tt+1] = (_P[tt, tt+1]*(1-
   B_M)/((_P[tt, 0]*M_B) + (_P[tt, tt+1]*(1-B_M))))
52.
53.             _C = np.zeros(T1)
54.             _C = _M@_W@(np.linalg.matrix_power(_P, i-1))
55.
56.             for l in range(0, T1):
57.                 _C[l, l] -=1
58.
59.            #Resolucion del sistema aplicando transformaciones de Gauss
   -Jordan para encontrar _ep
60.             _C = _C.T
61.             b = np.zeros(T1)
62.             b = b.reshape(T1, 1)
63.             ratio = 0
64.
65.             n = len(b)
66.
67.             for k in range(0, n):
68.                 for ii in range(k+1, n):
69.                     ratio = _C[ii, k]/_C[k, k]
70.                     for jj in range(k, n):
71.                         _C[ii, jj] -= ratio*_C[k, jj]
72.                     b[ii] = b[ii] - ratio*b[k]
73.
74.             b[n-1] = 1
75.
76.             _epi[n-1] = 1
77.
78.             for ii in range(n-2, -1, -1):
79.                 sum_j = 0
80.                 for jj in range(ii+1, n):
81.                     sum_j += _C[ii, jj]*_epi[jj]
82.                 _epi[ii] = (b[ii] - sum_j)/_C[ii ,ii]
83.
84.             #vector de estado
85.             suma = 0
86.             for kk in range(0, n):
87.                 suma = _epi[kk] + suma
88.
89.             _ep = _epi*(1/suma)
90.
91.       #calculo del coste de mantenimiento correctivo medio por periodo
```

```
92.
93.                 suma = 0
94.                 for kk in range(1, i+1):
95.                     _e = _e@_P
96.                     suma = _e.item(0) + suma
97.                 nAverias = suma*N
98.                 costeCorr = (C1*N*suma)/i
99.
100.     #calculo del coste de las inspecciones medio durante cada periodo
101.
102.                 suma = 0
103.                 for ll in range(j, T1-1):
104.                     suma = _ep.item(ll) + suma
105.
106.                 costeInsp = (C3*N*suma)/i
107.
108.    #calculo del coste de mantenimiento preventivo durante cada periodo
109.
110.                 Mc = 0
111.                 Mc = abs((nAverias/N) - _ep.item(0))
112.                 costePrev = (C2*N*Mc)/i
113.
114.                 costeTotal = costePrev+costeCorr+costeInsp
115.                 graf_TiempoCoste[i, j, 0] = i
116.                 graf_TiempoCoste[i, j, 1] = j
117.                 graf_TiempoCoste[i, j, 2] = costeTotal
118.
119.                 graf_Averias[i, j, 0] = i
120.                 graf_Averias[i, j, 1] = j
121.                 graf_Averias[i, j, 2] = nAverias/i
122.
123.                 if(costeTotal < costeMin):
124.                     costeMin = costeTotal
125.                     T2 = i
126.                     T3 = j
127.                     Averias = nAverias/i
128.
129.     return costeMin, T2, T3, Averias, graf_TiempoCoste, graf_Averias
```

cPreventivo3.py

cExponencial.py

```
1. import numpy as np
2.
3. def calcula(k, T1):
4.
5.     k = float(k)
6.     T1 = int(T1)
7.
8.     p = np.zeros(T1)
9.
10.
11.      for i in range(1, T1-1):
12.          p[i] = 1 - np.exp(-(k*i))
13.      return p
```

cExponencial.py

cWeibull.py

```
1. import numpy as np
2.
3. def calcula(alpha, betha, k, T1):
4.
5.     alpha = float(alpha)
6.     betha = float(betha)
7.     k = float(k)
8.     T1 = int(T1)
9.
10.      p = np.zeros(T1)
11.
12.      for i in range(1, T1-1):
13.          #llenar el array p con los valores obtenidos de la FDA Weibull
14.          p[i] = 1 - np.exp(-((i-k)/betha)**alpha)
15.
16.      return p
```

cWeibull.py

cRangoMedianas.py

```
1. import numpy as np
2.
3. def calcular(tabla):
4.     alpha = 0
5.     betha = 0
6.     # ordenar los datos de la tabla en orden ascendente según las horas
    de operacion
7.     for i in range(0, len(tabla)):
8.         if(tabla[i, 1]=='R'):
9.             tabla[i, 1] = 1
10.          else:
11.              tabla[i, 1] = 2
12.      tabla = tabla.astype('int32')
13.      tabla = tabla[tabla[:, 2].argsort()]
14.
15.      #calculo de indices de muestras de fallo
16.      N = len(tabla)
17.
18.      incremento = np.zeros(N)
19.      NuevoOrden = np.zeros((N,2))
20.      ElementoQueFallo = 0
21.
22.      for i in range(0, N):
23.          if(tabla[i, 1]==2):
24.              incremento[i] = (N+1-ElementoQueFallo)/(N+1-i)
25.              NuevoOrden[i,1] = incremento.item(i) + ElementoQueFallo
26.              ElementoQueFallo = NuevoOrden[i, 1]
27.              NuevoOrden[i,0] = tabla.item(i, 2)
28.
29.      tiempo_op = np.zeros((np.count_nonzero(NuevoOrden[:,0]), 2))
30.      i = 0
31.      for j in range(0, len(NuevoOrden)):
```

```
32.           if(NuevoOrden.item(j, 0)!=0):
33.               tiempo_op[i, 0] = NuevoOrden[j, 0] #asigna el tiempo de op
   eracion
34.               tiempo_op[i, 1] = NuevoOrden[j, 1] #asigna el nuevo i
35.               i = i+1
36.
37.       #funcion de observacion
38.       F = np.zeros(len(tiempo_op))
39.       for i in range(0, len(tiempo_op)):
40.           F[i] = (tiempo_op[i, 1] -0.3)/(N+0.4)
41.
42.       # ln(t) y ln(ln(1/1-F))
43.
44.       Y = np.zeros(len(tiempo_op))
45.       X = np.zeros(len(tiempo_op))
46.
47.       for i in range(0, len(tiempo_op)):
48.           X[i] = np.log(tiempo_op[i, 0])
49.           Y[i] = np.log(np.log(1/(1-F[i])))
50.       print(X)
51.       print(Y)
52.
53.       #regresion lineal y = mx + b
54.       m = 0
55.       b = 0
56.       suma1 = 0
57.       suma2 = 0
58.
59.       for i in range(0, len(X)):
60.           suma1 = (X.item(i)- np.mean(X))*(Y.item(i)- np.mean(Y)) + suma
   1
61.       for j in range(0, len(X)):
62.           suma2 = ((X.item(j)- np.mean(X))**2) + suma2
63.
64.       m = suma1/suma2
65.
66.       b = np.mean(Y) - m*np.mean(X)
67.
68.       alpha= m
69.
70.       betha = np.exp(-(b/m))
71.
72.       return alpha, betha
```

cRangoMedianas.py

A.2 Proceso para compilación de archivos y dependencias del software Manther 4.0 desarrollado en archivo único ejecutable

El sistema desarrollado es un script que es ejecutado por el interprete de Python, su ejecución rudimentaria se hace mediante la ventana de comandos de Windows, abriéndola en la carpeta donde se encuentra el archivo main de la aplicación y se ejecuta desde ella. Puede ser mas practico disponer mejor de un archivo ejecutable (.exe) que pudiera iniciar el programa al clicar sobre él, por lo que en esta sección se expondrá el proceso de compilación de todos los archivos que componen el programa en un único ejecutable utilizando la herramienta de desarrollo del interprete de Python "auto-py-to-exe".

Lo primero que debe hacerse es instalar mediante el sistema gestor de paquetes de Python *pip*[9] la herramienta "auto-py-to-exe", desde la ventana de comandos se utiliza el comando *pip install auto-py-to-exe* para realizar la instalación en el interprete de Python del ordenador. Finalizado el proceso de instalación se ejecuta desde la ventana de comandos la herramienta, se mostrará una interfaz gráfica como al de la Figura A1-0-1.

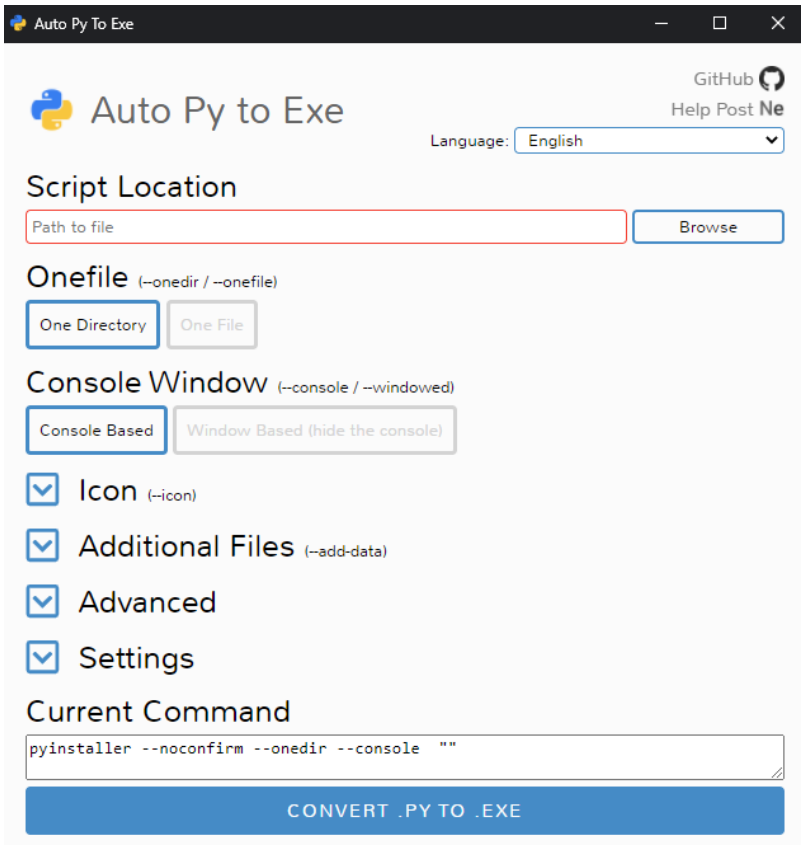

Figura A1- 0-1. Ventana de inicio de la herramienta "auto-py-to-exe".

El primer campo a llenar será la localización del script "main.py", utilizando el campo **Script Location**; como el programa utiliza más de un archivo .py en el campo de **Onefile** se selecciona la opción de "One Directory" y en el campo de **Console Window** la opción de "Window Based (hide the console)" ya que en el despliegue de la aplicación no será necesario que esté abierta la ventana de comandos.

En la sección de **Additional Files** se agregarán utilizando la opción de "Add Files" el resto de los archivos .py que componen las funcionalidades (módulos de cálculo y despliegue de intefaces graficas) de Manther 4.0 como se observa en la Figura A1-0-2 y se procede a generar el archivo ejecutable haciendo click en **Convert .py to .exe**. En la sección de **Settings** se ingresa en el campo de "Output" la carpeta de destino donde se creará el archivo.

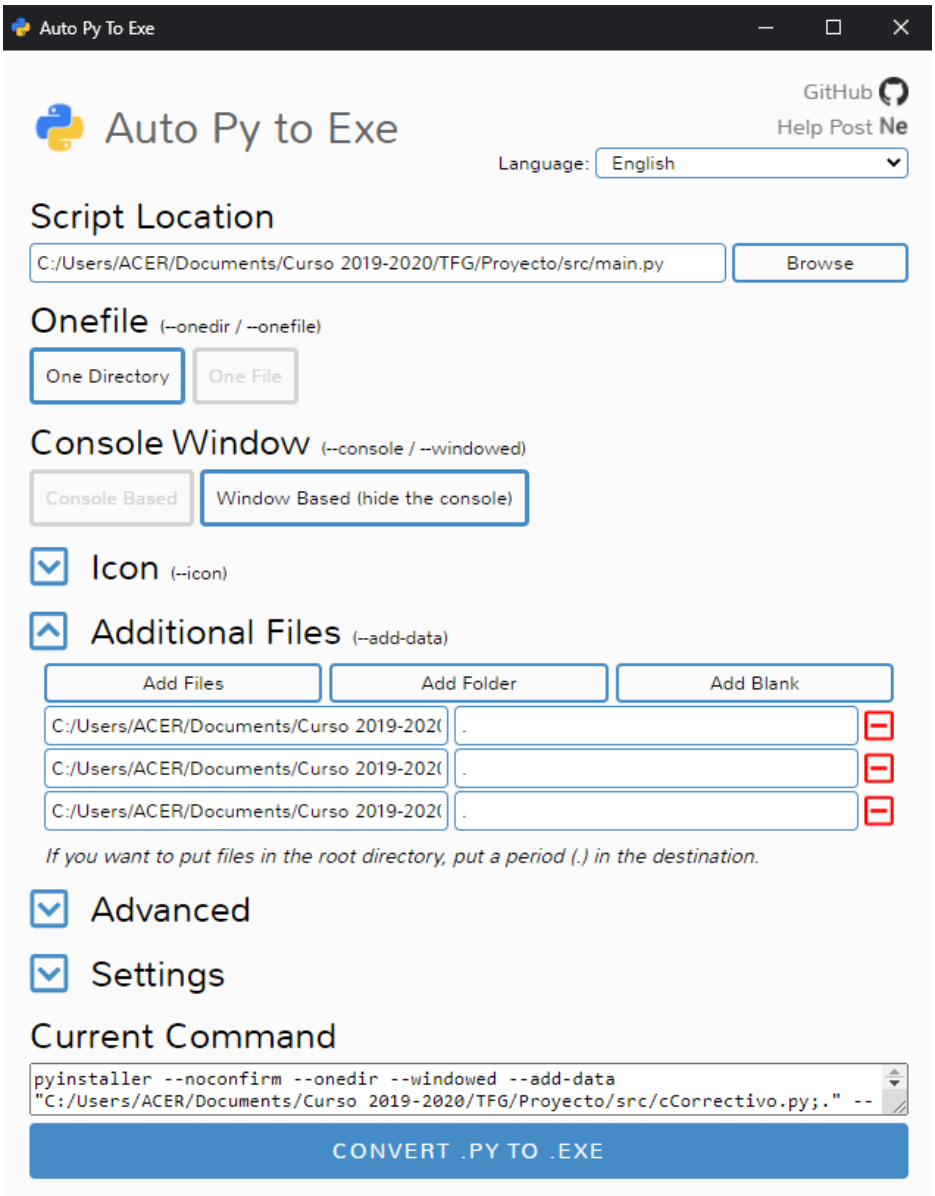

Figura A1- 0-2. Ejemplo de formulario de "auto-py-to-exe" con datos ingresados.

Finalizado el proceso de conversión en la carpeta especificada en **Settings** estará el archivo ejecutable.

A.3 Presentación de gráficas de costes y averías en función de T2 y T3 para experimentos del caso práctico a escala de 80 y 100 horas

A continuación, se adjuntan las gráficas de soluciones exploradas para cada modelo para los casos de utilización de período temporal de grupos de 80 horas.

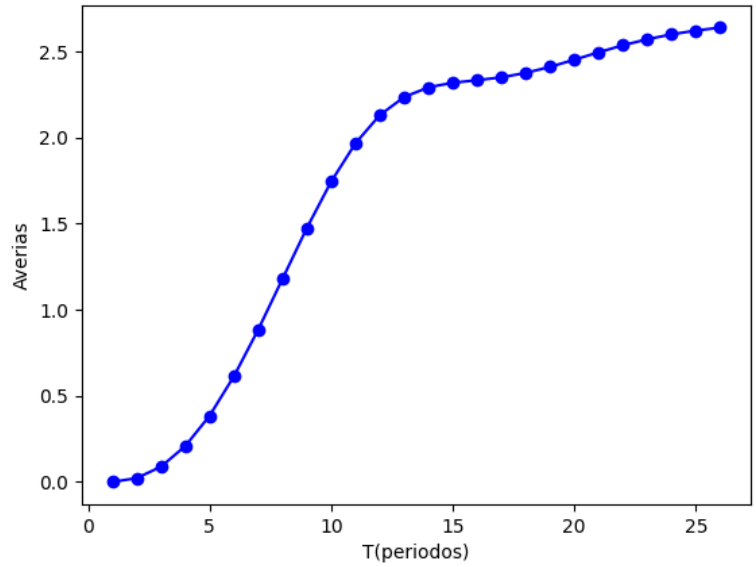

Figura A2-0-3. Gráfica de averías en funcion de T2, política de mantenimiento preventivo total, período temporal de grupos de 80 horas

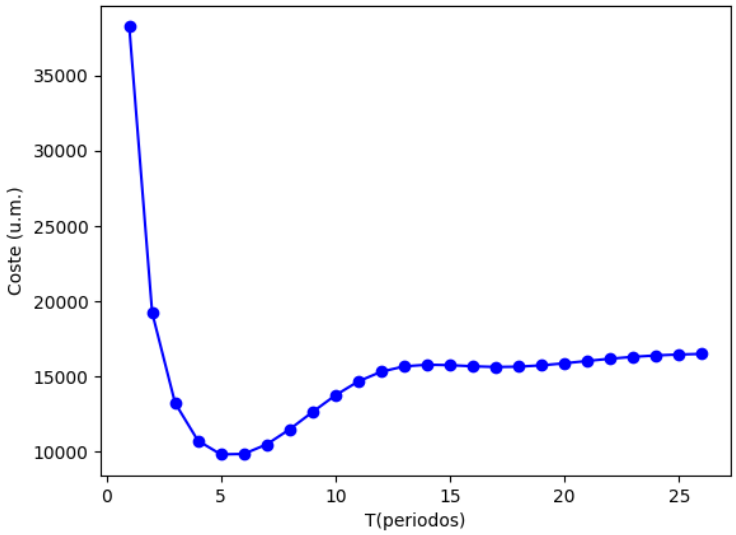

Figura A2-0-4. Gráfica de costes totales en función de T2, política de mantenimiento preventivo total, período temporal de grupos de 80 horas

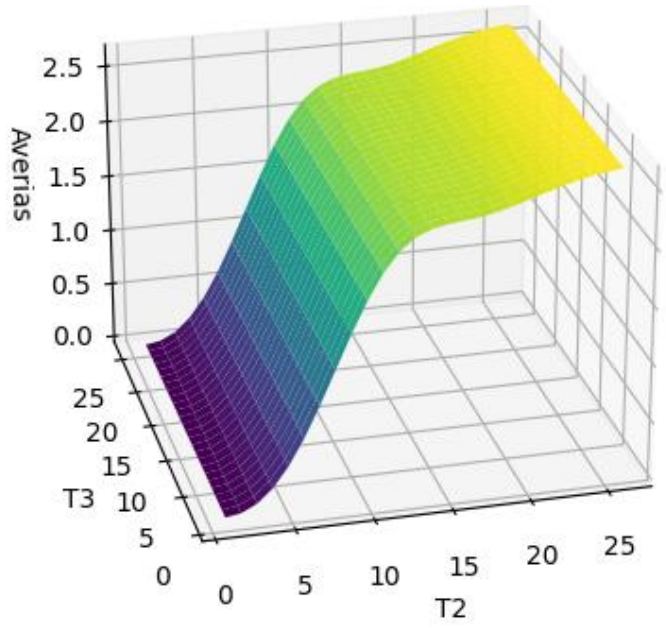

Figura A2-0-5. Gráfica de averías en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos, período temporal de grupos de 80 horas

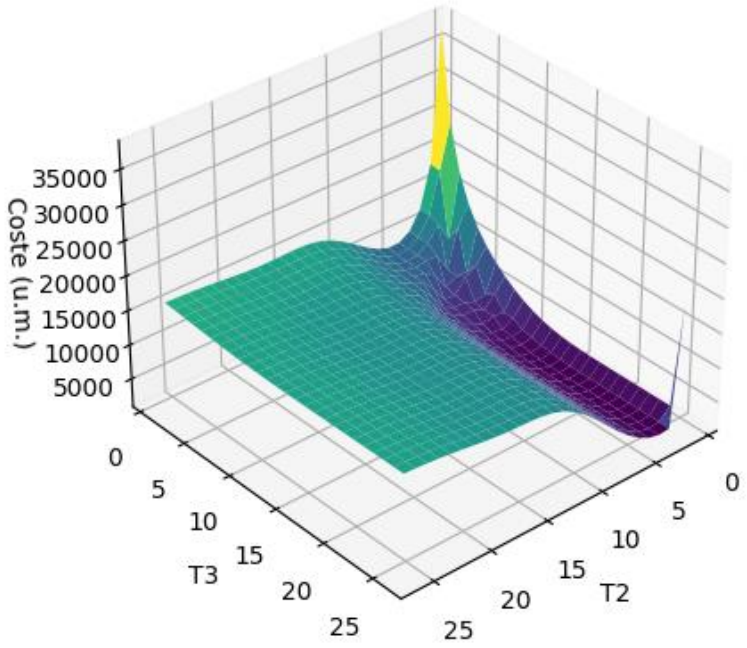

Figura A2-0-6. Gráfica de costes totales en función de T2 y T3, política de mantenimiento preventivo basado en tiempo de funcionamiento sin fallos, período temporal de grupos de 80 horas

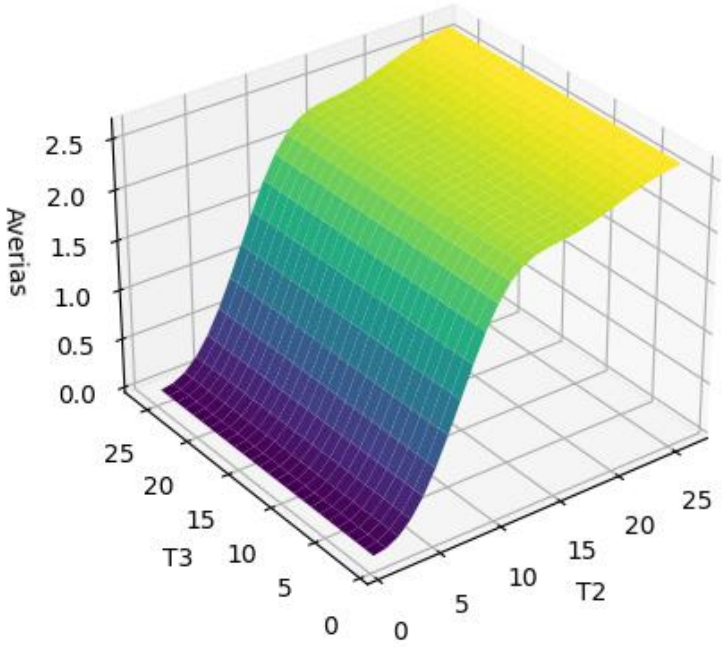

Figura A2-0-7. Gráfica de averías en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos y una inspección previa, período temporal de grupos de 80 horas

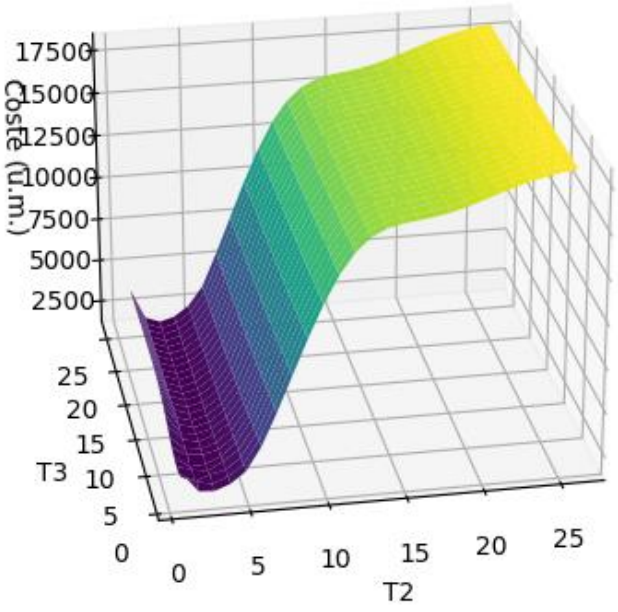

Figura A2-0-8. Gráfica de costes totales en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos mas una inspección previa, período temporal de grupos de 80 horas

En las siguientes gráficas se muestran los resultados explorados por Manther para encontrar la solución para el caso planteado utilizando como período temporal elemental grupos de 100 horas.

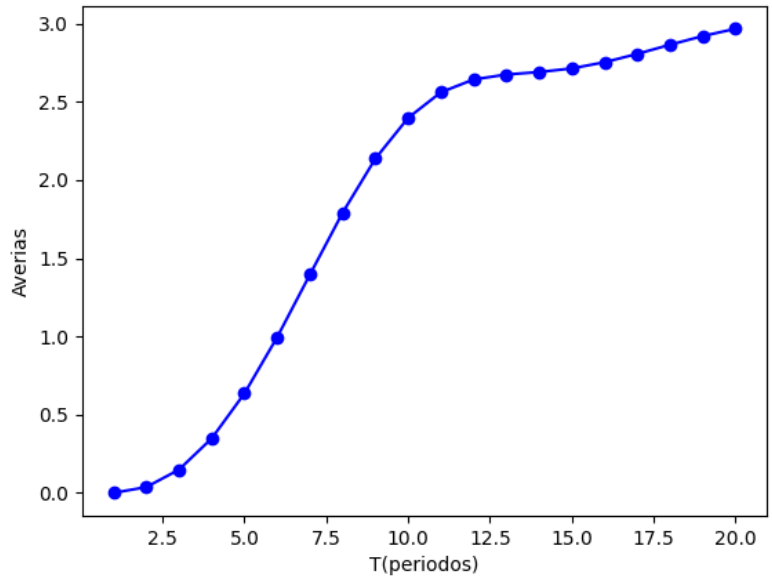

Figura A2-0-9. Gráfica de averías en función de T2, política de mantenimiento preventivo total, período temporal de grupos de 100 horas

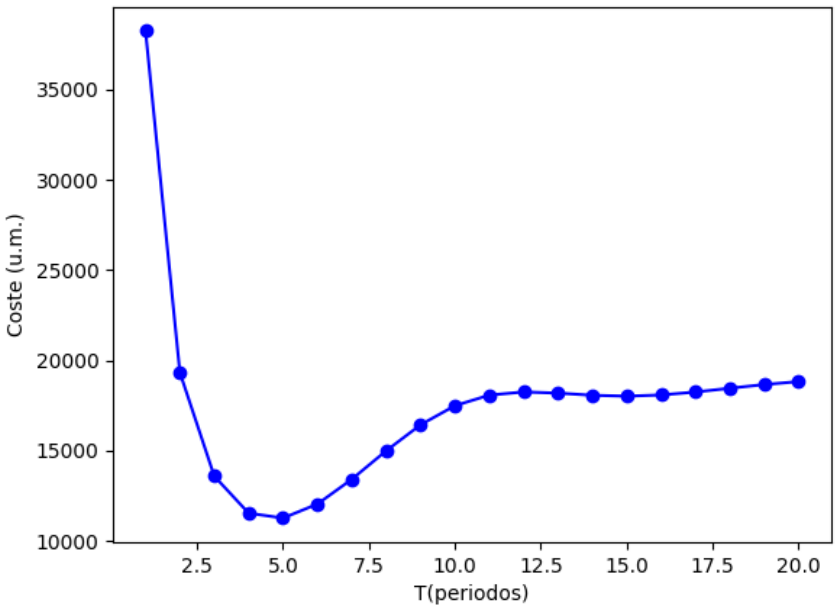

Figura A2-0-10. Gráfica de costes totales en función de T2, política de mantenimiento preventivo total, período temporal de grupos de 100 horas

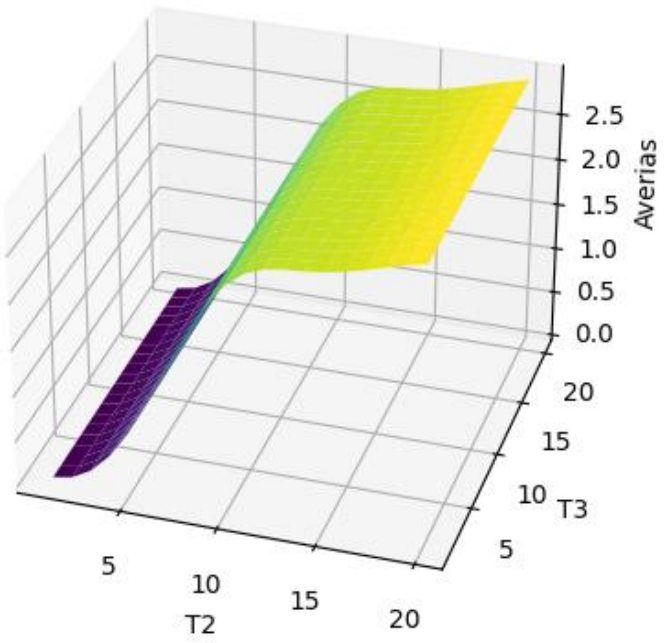

Figura A2-0-11. Gráfica de averías en función de T2 y T3, política de mantenimiento preventivo basando en el tiempo de funcionamiento sin fallos, período temporal de grupos de 100 horas

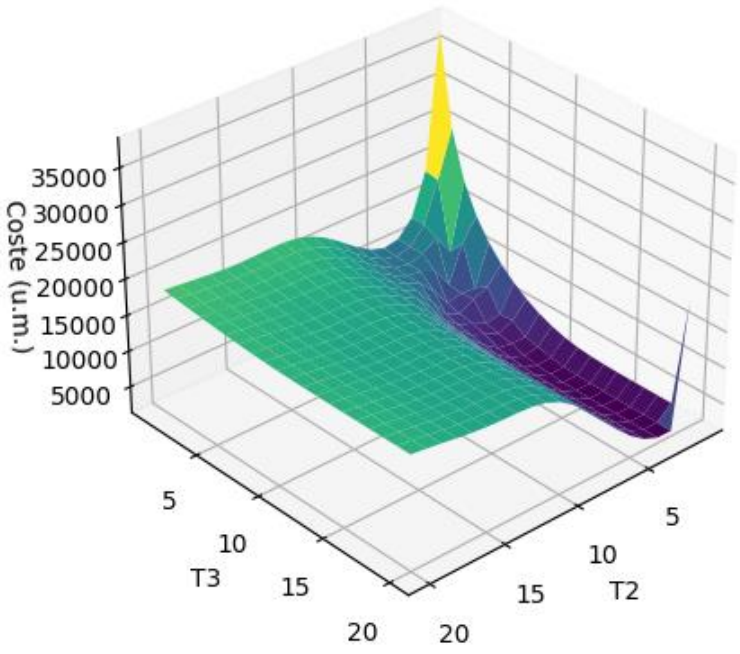

Figura A2-0-12. Gráfica de costes totales en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos, período temporal de grupos de 100 horas

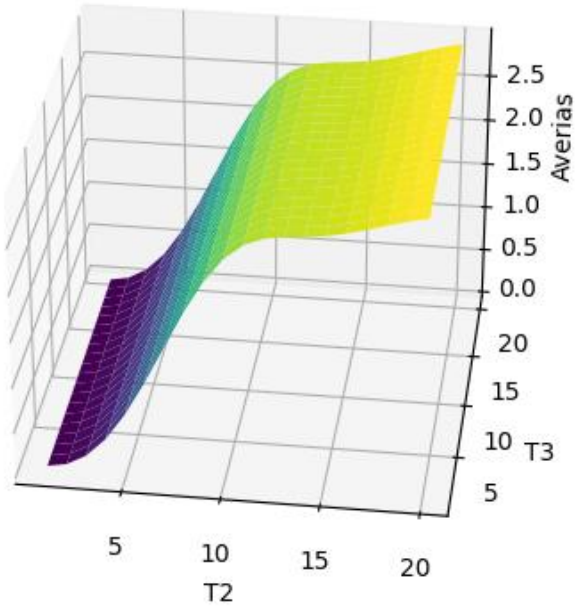

Figura A2-0-13. Gráfica de averías en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos mas una inspección previa, período temporal de grupos de 100 horas

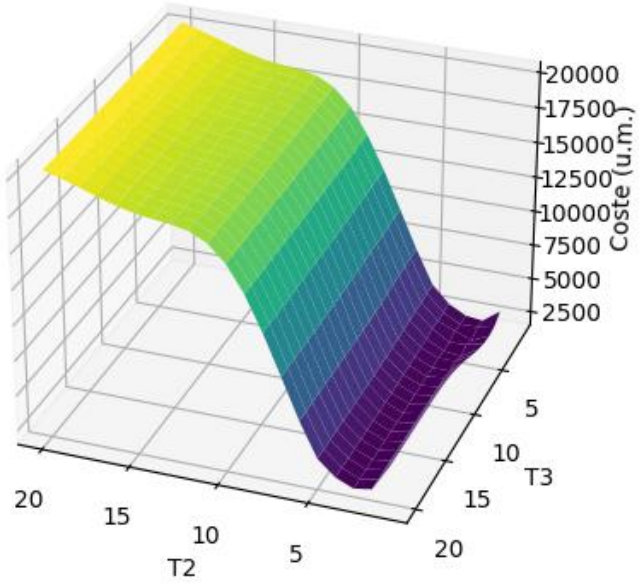

Figura A2-0-14. Gráfica de costes totales en función de T2 y T3, política de mantenimiento preventivo basado en el tiempo de funcionamiento sin fallos mas una inspección previa, período temporal de grupos de 100 horas

REFERENCIAS

[1] J. R. Figuera Figuera, «Mantenimiento de Equipos Industriales. Modelos Cuantitativos», Madrid, Publ. Interna ESII (Politecnica de Madrid), 1981

[2] Prieto García, Carlos: «*Fiabilidad, Mantenibilidad y Mantenimiento*»; Universidad de Sevilla, Área de Ingeniería Mecánica, Sevilla, 2008

[3] Papoulis, Pillai, «Probability, Random Variables, and Stochastic Processes», 4ta Edición.

[4] A. Sánchez-Herguedas, A. Mena-Nieto, F. Rodriguo-Muñoz, « A new analytical method to optimise the preventive maintenance interval by using a semi-Markov process and z-transform with an application to marine diesel engines », vol 207, pp. 4-7, 2021, 107394, ISSN 0951-8320, https://doi.org/10.1016/j.ress.2020.107394

[5] Python Org., «https://python.org» Documentación Python 3.9.6 (Acceso: 23/02/2021). https://docs.python.org/3/

[6] Ingeniería de software, «Arquitectura de software: Descomposición modular» (Acceso 8/4/2021) https://ittgweb.wordpress.com/2016/05/29/descomposicion-modular/

[7]Numpy Org., «https://numpy.org» Numpy v1.21 Manual (Acceso : 6/03/2021) https://numpy.org/doc/stable/

[8] https://matplotlib.org (Acceso : 6/03/2021)

[9] Python Org., « https://pypi.org/project/pip/ » pip documentation v21.2.3 (Acceso: 6/03/2021) https://pip.pypa.io/en/stable/getting-started/

[10] A. Sanchez-Herguedaz, «Optimización del Mantenimiento», ETSII, 1994