GRADO EN INGENIERÍA EN TECNOLOGÍAS DE INDUSTRIALES

TRABAJO FIN DE GRADO

MASSIVES RES INTEGRATION IN POWER SYSTEMS – LINKING PLANNING AND DYNAMIC SIMULATIONS

Autor: Ignacio Escartín Fernández de Landa

Director: Lukas Sigrist

Co-Director: Enrique Lobato Miguélez

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Massive RES integration in power systems – linking planning and dynamic simulations.

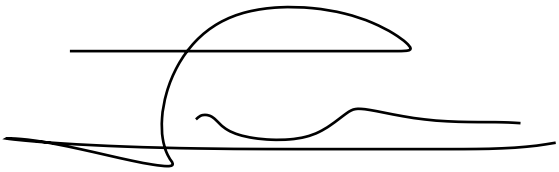
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Ignacio Escartín Fernández de Landa Fecha: 12/ 06/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Sigrist, Lukas Fecha: 12/ 06/ 2023

LOBATO MIGUELEZ ENRIQUE - 13154753H

Firmado digitalmente por LOBATO MIGUELEZ ENRIQUE - 13154753H
Fecha: 2023.06.12 08:28:19 +02'00'

Fdo.: Lobato Miguélez, Enrique Fecha: 12/ 06/ 2023

Agradecimientos

En agradecimiento, en primer lugar, gracias a Lukas Sigrist por su enorme ayuda y fuente de conocimiento el curso pasado durante mi primer año como colaborador del I.I.T, y por su enorme aportación este año como director de este proyecto. Agradecimientos también a Enrique Lobato por su ayuda y colaboración como director del proyecto.

INTEGRACIÓN MASIVA DE FUENTES DE GENERACIÓN RENOVABLE EN SISTEMAS ELÉCTRICOS – PLANIFICACIÓN DE ENLACES Y SIMULACIONES DINÁMICAS

Autor: Escartín Fernández de Landa, Ignacio.
Directores: Lukas Sigrist, Enrique Lobato Miguélez
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas (Instituto de Investigación Tecnológica)

RESUMEN DEL PROYECTO

El objetivo de este proyecto ha sido contribuir a la optimización de la operación de los sistemas eléctricos de forma más rápida y autónoma mediante el desarrollo de una serie de herramientas que permiten utilizar las capacidades de Python para automatizar análisis y procesos sobre sistemas eléctricos en PSSE.

Palabras clave: Python, PSSE, automatización, Script, flujo de cargas óptimo, análisis dinámicos, análisis de contingencias, generación renovable.

1. Introducción

Como consecuencia de recientes cambios en el funcionamiento de la sociedad, como el fuerte incremento de la demanda eléctrica, la proliferación de la generación y almacenamiento deslocalizados o la priorización del uso de fuentes de generación renovables, debido a iniciativas a favor del cuidado del medio ambiente, la estabilidad de los sistemas eléctricos se está viendo tensionada, y la gestión de la topología de la red se está volviendo cada vez más compleja. La generación renovable, que comprende tecnologías como las plantas fotovoltaicas o los parques eólicos, resulta en una generación intermitente y de bajo rendimiento que puede llevar a pérdidas de robustez en la red. Este cambio de paradigma está propiciando un endurecimiento de los procedimientos de control de los futuros sistemas eléctricos [1]

Países como España, los cuales han desarrollado una gran cantidad de proyectos de renovables en los últimos años, disponen de un gran potencial para la operación de la red, y para la descarbonización de nuestras economías. Sin embargo, es necesario desarrollar herramientas que permitan planificar y monitorizar de forma segura y precisa, en términos de operación, y en términos de nuevas inversiones, todos los elementos que se están conectando en nuestras redes eléctricas.

De esta manera, los operadores de red de diversos países, especialmente insulares, han desarrollado más regulaciones y requisitos de operación para empresas de generación, transporte y distribución eléctrica. Para asegurar el cumplimiento de dichas regulaciones, también conocidos como códigos de red, las empresas se encuentran destinando recursos y desarrollando cada vez más departamentos para la gestión óptima e inteligente de las redes, comúnmente conocido como *Smart Grids*.

2. Definición del Proyecto

Para dar respuesta al problema postulado en la sección anterior, este proyecto ha sido desarrollado en colaboración con el Instituto de Investigación Tecnológica (IIT) de la

Universidad Pontificia de Comillas. El proyecto se ha centrado en desarrollar una serie de algoritmos y ficheros de Pyhton que permitan automatizar la ejecución de análisis sobre sistemas eléctricos en PSSE. El caso de estudio para desarrollar la herramienta ha sido la isla de Tenerife, del cual se recibe el despacho económico por horas programado para cada día o semana, así como la demanda estimada, y se exportan, tras los análisis, datos sobre el funcionamiento óptimo de la red y su operación bajo contingencias.

3. Descripción de la herramienta

Como se puede observar en el diagrama de arquitectura de la *Figura 1*, la herramienta sigue el siguiente proceso para ejecutar un análisis sobre un sistema. Los procesos están interconectados entre sí mediante entradas y salidas de datos:

- Ficheros de Python que realizan un preparado de los casos de estudio del sistema a evaluar en PSSE (***Analysis_Part_1_Imports***) a través de la importación, el procesado y la clasificación de datos (despacho económico, demanda y generación renovable) de ficheros de texto y Excel (***Excel_Extraction***), y su posterior implementación en casos de estudio de PSSE, donde prepara la red para los análisis a realizar (***Python_To_PSSE***)
- Fichero de Python (***Analysis_Part_2_OPF***) encargado de ejecutar el flujo de cargas óptimo sobre cada uno de los casos horarios de estudio, obteniendo el punto óptimo de operación de la red
- Ficheros de Python que realizan posteriores análisis de fiabilidad sobre el sistema (***Analysis_Part_3_Tests***), y exportan los correspondientes resultados. Los análisis comprenden la obtención de resultados del flujo de cargas óptimo y la realización de simulaciones de contingencias sobre líneas, transformadores y generadores (***Static_Analysis_Library***), y la ejecución de simulaciones dinámicas de pérdidas de generación y regulación primaria, y cortocircuitos en nudos (***Main_Simulate_Trippins)***

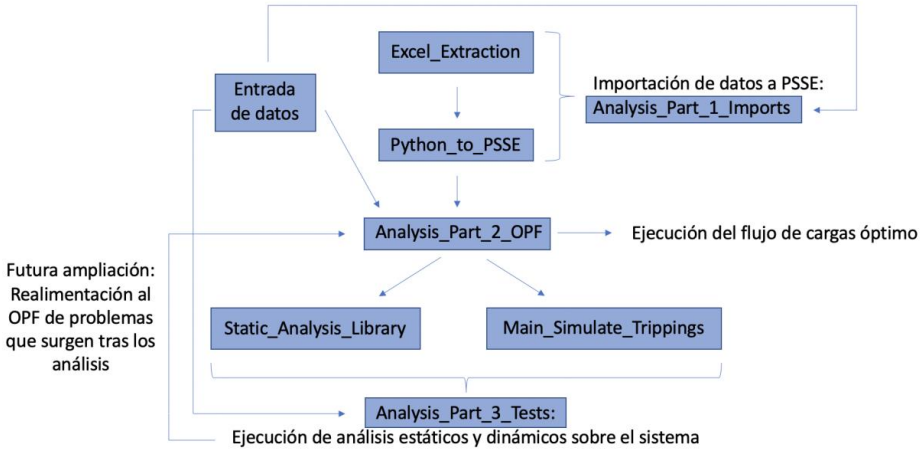


Figura 1: Diagrama de arquitectura de la herramienta

4. Resultados

Tras haber realizado todos los análisis pertinentes, se llega a un punto de funcionamiento óptimo del sistema eléctrico para cada programación horaria, durante un día o durante una semana. Ejemplos de los resultados son los siguientes, en este caso correspondientes a los

resultados del flujo de cargas óptimo en la isla de Tenerife en el primer caso de estudio horario, así como la gráfica de frecuencia de un análisis dinámico de pérdida de generación. Se puede observar que la carga de los flujos está debajo del máximo (100%), que las tensiones están dentro de los límites establecidos y que la frecuencia después de la pérdida de un generador varía como mucho 0.008 pu, es decir, alcanza un valor mínimo de 49.6 Hz:

BusFROM	BusTO	ID	LOAD%
7	16	1	30,803
7	16	2	30,738
7	32	1	24,277
7	46	1	33,541
7	46	2	33,496

Bus	Voltage(p.u)
7	1,0242
11	1,0045
16	1,0173
30	1,0154
31	1,0243

Tablas 1 y 2: Ejemplos de los resultados extraídos del flujo de cargas óptimo.

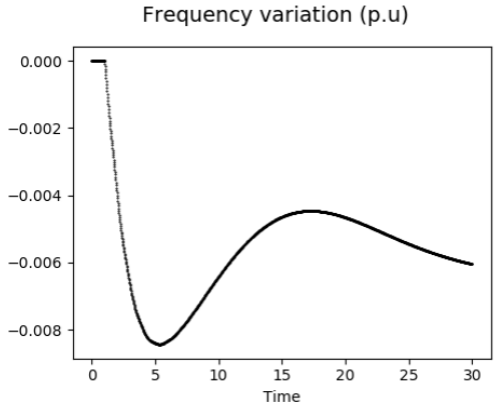


Figura 2: Variación dinámica de la frecuencia tras la desconexión de un genereador

5. Conclusiones

Los resultados obtenidos han demostrado la eficacia y conveniencia de la ejecución de los análisis de forma automática con la herramienta desarrollada, en contraposición con su ejecución a mano. Además, la flexibilidad aportada al usuario a la hora de la introducción de datos a los scripts de Python, le permiten realizar una gran variedad de análisis para topologías de red muy variopintas.

A modo de ampliación del trabajo desarrollado para un futuro próximo, los scripts se han acomodado, además, para la implementación de futuras retroalimentaciones de los datos obtenidos de los análisis dinámicos y de contingencias al flujo de cargas óptimo, como se puede observar en la *Figura 1,* de manera que la herramienta pueda, de forma independiente, corregir violaciones de límites detectadas tras los análisis mediante el cambio de comandos al flujo de cargas óptimo.

6. Referencias

[1] Oliver Smith et al.,The effect of renewable energy incorporation on power grid stability and resilience. Sci. Adv. 8, eabj6734 (2022). DOI:10.1126/sciadv.abj6734

MASSIVE RES INTEGRATION IN POWER SYSTEM – LINKING PLANNING AND DYNAMIC SIMULATIONS

Author: Escartín Fernández de Landa, Ignacio.
Supervisors: Sigrist, Lukas. Lobato Miguélez, Enrique
Collaborating Entity: ICAI – Universidad Pontificia Comillas (Instituto de Investigación Tecnológica)

ABSTRACT

The aim of this project has been to help optimize the operation of power systems in a faster and more autonomous way through the development of a series of tools that allow the user to leverage the capabilities of Python in order to automate analyses and processes on power systems in PSSE.

Keywords: Python, PSSE, automation, script, optimal power flow, Dynamic analysis, contingency analysis, renewable generation.

1. Introduction

As a consequence of changes in the mentality of our current societies, like the sharp increase in electricity demand, the proliferation of offshore generation and electrical energy storage, or the increasing prioritization of renewable energy generation sources due to initiatives to promote the welfare of our environment, the stability of our current power systems is being compromised and the topology of the grid is becoming increasingly complex. Renewable electricity generation, which involves technologies like photovoltaic plants or wind farms, yields an intermittent generation that has a low performance, something that can lead to robustness and resilience losses in the grid. This paradigm shift is forcing grid operators to tighten control procedures for future power systems [1].

Countries like Spain, that have developed a considerable amount of renewable energy projects in the past years, have a great potential to leverage in terms of grid operation and decarbonization of our economies. However, it is necessary to develop tools that allow these countries to plan and monitor in a safe and precise way, in terms of grid operation and of new investments, all the elements that are nowadays being connected to our electric grids.

Therefore, grid operators from different countries, especially insular, have developed more regulations and operation requirements for companies that perform activities related to generation, transmission, and distribution of electricity. To assure compliance with such regulations, also known as the Grid Code, companies are allocating resources and developing teams of people that can ensure the optimal and intelligent management of the power grid, commonly known as the Smart Grids field.

2. Project description

In order to provide solutions for the problem described above, this Project has been developed in partnership with the Institute of Technological Research (IIT) of the Comillas Pontifical University. The project has focused on the development of a series of algorithms and Python scripts that allow to automate the execution of analyses on power systems by using the PSSE software. The case study for the development of the analysis tool has been

the island of Tenerife, whose inputs are the hourly economic dispatch scheduled for each day or each week, as well as the estimated hourly demand. After the analyses, data regarding the optimal power flow of the system and regarding its operation under contingencies is exported.

3. Tool description

As it can be observed in the architecture diagram in *Figure 1,* the tool follows the following process in order to execute an analysis on a power system. The different processes are connected with each other through data inputs and outputs:

- Python files that set up the system study cases to be evaluated in PSSE (***Analysis_Part_1_Imports***) through the import, processing and classification of input data (economic dispatch, demand and renewable generation) from text and Excel files (***Excel_Extraction***), and its subsequent implementation in PSSE study cases, where the grid is set up for the analyses to conduct (***Python_To_PSSE***)
- A Python file (***Analysis_Part_2_OPF***) in charge of conducting the Optimal Power Flow in every hourly study case, after which the optimal operation point of the grid is obtained.
- Python files that conduct further reliability analyses of the system (***Analysis_Part_3_Tests***) and export their corresponding results. The analyses include the results of the optimal power Flow, and contingency analyses in lines, Transformers and generators (***Static_Analysis_Library***), as well as dynamic simulations of dispatch losses with primary regulation, and short circuits in nodes (***Main_Simulate_Trippins)***

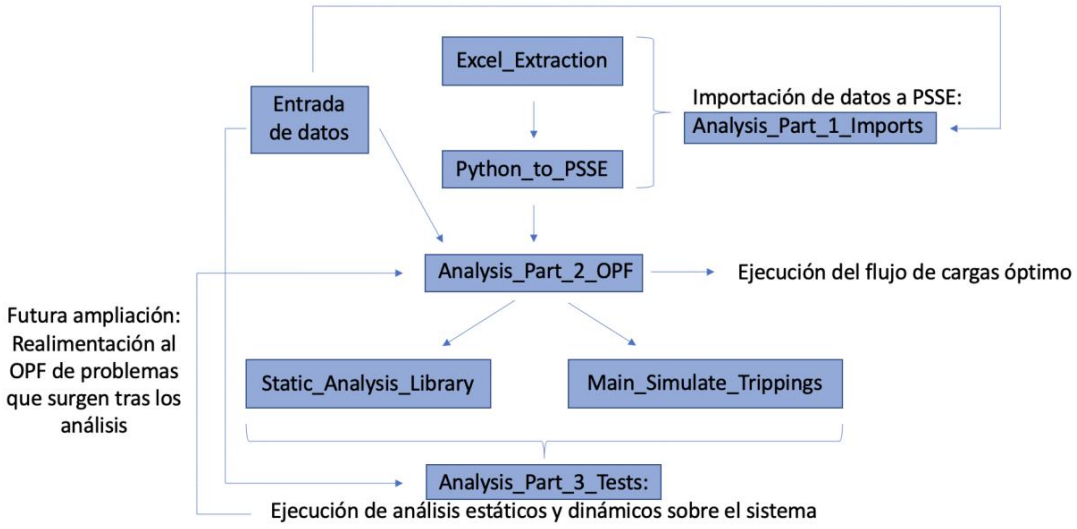


Figure 3: Tool architecture diagram

4. Results

After having conducted all the necessary analyses, the optimal operating point of the power system of Tenerife can be reached for every desired time frame (hourly in this case, during a day or during an entire week). Examples of the results are the following, in this case corresponding to the Optimal Power Flow of the Tenerife Island at the first provided hour of the first day, as well as part of the results of a generation dispatch dynamic analysis. As it

can be observed, branch flows are below their limit (100%), voltage limits are inside specified values, and that the system frequency after the loss of a generator varies, at most, around 0.008 p.u (a minimum value of 49,6 Hz).

BusFROM	BusTO	ID	LOAD%
7	16	1	30,803
7	16	2	30,738
7	32	1	24,277
7	46	1	33,541
7	46	2	33,496

Bus	Voltage(p.u)
7	1,0242
11	1,0045
16	1,0173
30	1,0154
31	1,0243

Tables 3 and 4: Examples of results extracted from the optimal power flow

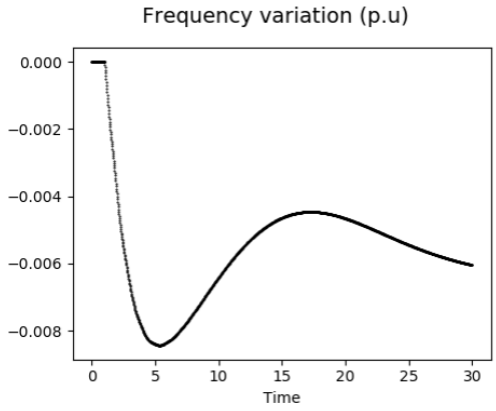


Figure 4: Dynamic frequency variation after a loss of generation dispatch

5. Conclusions

The obtained results have demonstrated the efficiency and convenience of the automatic execution of the analyses with the developed tool, as opposed to their execution by hand. Furthermore, the flexibility provided to the user when inputting the power system's data to the Python scripts, allows them to perform a wide variety of analyses for very diverse network topologies.

As an extension of the developed work for the near future, the scripts have been accommodated for the implementation on future feedback of the data obtained from the dynamic and contingency analyses to the Optimal Power Flow, as it can be observed in *Figure 3*, so that the developed tool can, autonomously, correct boundary violations detected after the analyses by switching commands to the Optimal Power Flow.

6. References

[1] Oliver Smith et al.,The effect of renewable energy incorporation on power grid stability and resilience. *Sci. Adv*. **8**, eabj6734 (2022). DOI:10.1126/sciadv.abj6734

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Tecnologías de generación como las fuentes de energía renovables y la generación de electricidad deslocalizada (cerca de edificios, hogares, empresas…) se están convirtiendo en unos de los avances más populares del siglo XXI. Desde una búsqueda de la independencia energética de los consumidores finales, mediante una relocalización de las fuentes de generación de electricidad a lugares próximos al consumidor y gestionados por ellos, hasta la búsqueda del cuidado de nuestro planeta mediante fuentes de generación de electricidad que no resulten perjudiciales para el medio ambiente, estas nuevas tecnologías presentan proyecciones y resultados muy provechosos para el consumidor final y, en general, para el conjunto de la humanidad, pero también necesitan de proyectos muy ambiciosos y cuya implementación presenta grandes retos.

Durante la mayor parte del siglo XX, la gestión de las redes eléctricas se ha llevado a cabo de forma centralizada por operadores de red. Las máquinas de generación utilizadas en el momento (centrales de carbón, de gas natural, ciclos combinados, centrales hidroeléctricas, centrales nucleares…), cuyo uso se podía programar con antelación, proporcionaban una generación de electricidad constante y robusta; la cual junto una demanda predecible, un sobredimensionamiento de la potencia instalada en los sistemas, la adecuada elección del despacho de potencia, y el uso de sistemas de monitorización de dichas centrales (SCADA), componían redes eléctricas estables y relativamente predecibles. El control de las redes eléctricas se resume en el control del perfil de tensiones y el control de la frecuencia como lazos de estabilidad del sistema, a parte de la prevención de sobrecargas en las líneas. Por otro lado, la monitorización de los dispositivos se realiza a través de protecciones y elementos de medida, que permiten realizar una estimación del estado actual del sistema.

Sin embargo, la aparición de tecnologías de generación renovable ha supuesto un completo cambio de paradigma a la hora de gestionar dichas redes eléctricas. Como es comúnmente

conocido a día de hoy, fuentes de generación renovable que tienen una importante presencia en los conjuntos de generación de las redes eléctricas, como la eólica o la solar fotovoltaica, presentan no sólo una generación de electricidad intermitente debido a sus intrínsecas características, sino también una ausencia de inercia que les impide absorber cambios en la frecuencia del sistema debidos a desequilibrios entre la generación y la demanda; lo que en última instancia resulta en un sistema eléctrico menos robusto, y que responde peor a desequilibrios o contingencias. Cabe mencionar que, aunque las turbinas eólicas disponen de máquinas rotativas, estas se encuentran mecánicamente desacopladas de la red. Además, la mencionada deslocalización de la generación de electricidad, en su mayoría compuesta por la instalación de sistemas como placas solares, diseñadas comúnmente para verter electricidad a la red cuando hay un exceso de generación, provocan que la gestión de las redes locales por parte las empresas distribuidoras (en el caso de España, empresas como Iberdrola o Endesa) sea más complicada puesto que, en la actualidad, la demanda también presenta una generación renovable, ergo intermitente, que en última instancia, dificulta también la gestión de la demanda.

El operador del sistema debe garantizar una operación segura de la red eléctrica. La seguridad se refiere al grado de riesgo de sobrevivir una perturbación. De esta manera, para ser seguro, el sistema de ser estable, refiriéndose como estabilidad a la continuidad de la operación intacta después de una perturbación. Según la naturaleza física del modo de inestabilidad, el tamaño de la perturbación y las dinámicas involucradas se puede diferenciar entre:

- Estabilidad de ángulo
- Estabilidad de frecuencia
- Estabilidad de tensión

La estabilidad de ángulo y de tensión se pueden además diferenciar entre estabilidad de gran y pequeña perturbación, según el tamaño de la falta aplicada. La estabilidad de frecuencia y de tensión se pueden diferenciar entre estabilidad de corta y larga duración, según el intervalo de tiempo escogido para realizar el análisis.

A la hora de realizar el análisis de seguridad, es necesario evaluar en el sistema, con modelos apropiados, su comportamiento estático y dinámico frente a perturbaciones. Los análisis estáticos incluyen los estudios del flujo de cargas y los análisis de contingencias. Los análisis dinámicos incluyen, entre otras, los cortocircuitos, y la pérdida de generación.

Por otro lado, el desacoplo mecánico de las turbinas eólicas y las placas solares debido a los convertidores de electrónica de potencia y los controles, también afecta a la respuesta de dicha generación, la cual es fundamentalmente distinta a la de un generador síncrono. Esto último implica que se deben utilizar modelos distintos para describir sus respuestas. Además, la rapidez de la respuesta de la generación renovable puede causar nuevos fenómenos de inestabilidad.

Los problemas descritos anteriormente han generado la necesidad de crear herramientas de análisis cada vez más complejos.

1.2 ESTADO DE LA CUESTIÓN

1.2.1 CÓDIGOS DE RED

A nivel nacional la gestión de la red eléctrica de la península ibérica (España y sus interconexiones con Portugal), Baleares, Canarias, Ceuta y Melilla, se realiza por parte de Red Eléctrica Española, una empresa pública de la cual el Estado español es en parte propietario. REE se constituyó en 1985 y fue la primera empresa del mundo dedicada en exclusividad al transporte y operación de sistemas eléctricos. Tras su salida a bolsa en 1999, la empresa comienza con la adquisición de todos los activos de transporte y gestión de la red eléctrica española. Con la creciente presencia de las energías renovables en el sistema eléctrico español, REE crea el CECRE (*Control Center of Renewable Energies*) en 2006, un centro para el control e integración de las energías renovables en el sistema eléctrico español. Además, en esta década se acometen proyectos para crear y mejorar las interconexiones eléctricas con Marruecos y Francia, así como con Baleares a inicios de la década de 2010. Posteriormente, durante esta década, REE se ha dedicado a la ampliación de sus funciones

como operador de la red española, y de la mejora y adaptación al contexto actual de la infraestructura eléctrica. [6]

En 2009, la Comisión Europea establece en el Reglamento del Parlamento Europeo (CE) 714/2009 las *Condiciones de acceso a la red para el comercio transfronterizo de electricidad*, los primeros "Códigos de Red" europeos, encargados de la armonización, integración y eficiencia del mercado eléctrico europeo. Posteriormente, entre 2015 y 2017, establece los *Códigos de red de conexión, Códigos de Red de Operación y los Códigos de Red de Mercado* [3]:

- Códigos de Red de Conexión: Establecen los requisitos para las instalaciones de generación eléctrica, demanda y HVDC (*High Voltage Direct Current*) que se conectan a la red:
    - Reglamento (UE) 2016/631 que establece un código de red sobre requisitos de conexión de generadores a la red eléctrica
    - Reglamento (UE) 2016/1388 por el que se establece un código de red para la conexión de la demanda
    - Reglamento (UE) 2016/1447 por el que se establece un código de red sobre requisitos de conexión a red de sistemas de HVDC y módulos de parque eléctrico conectados en CC (corriente continua)
- Códigos de Red de Operación: Establecen las reglas y procedimientos para la gestión de la red y operación del sistema en situaciones de emergencia y reposición del servicio
    - Reglamento (UE) 2017/1485 por el que se establecen directrices para la gestión de las redes de transporte de electricidad
    - Reglamento (UE) 2017/2196 por el que se establece una normativa para la gestión de emergencias y reposiciones del servicio
- Código de Red de Mercado: Establece las bases para la creación del mercado interno de electricidad (cálculo y asignación de capacidades) diario e intradiario y en el funcionamiento de los mercados del balance eléctrico.

- Reglamento (UE) 2015/1222: Establecimiento de directrices para la asignación de capacidad y la gestión de congestiones
- Reglamento (UE) 2016/1719: Directrices sobre la asignación de capacidad a largo plazo
- Reglamento (UE) 2017/2195: Directrices sobre el balance eléctrico

Posteriormente, y en cumplimiento con la normativa europea anteriormente mencionada, se llevó a cabo en España una implementación nacional de los Códigos de Red en España, mediante la aprobación de la siguiente normativa [2]:

- Real Decreto 647/2020, por el cual se regulan aspectos necesarios relacionados con la implementación de los códigos de red de determinadas instalaciones eléctricas.
- Orden TED/749/2020, por la cual se establecen los requisitos técnicos para la conexión a red para la implementación de los códigos de red de conexión
- Adicionalmente, para una mayor clarificación sobre las conexiones de módulos de generación eléctrica, se elaboraron posteriormente guías para la puesta en servicio de módulos de generación en el sistema eléctrico peninsular y no peninsular.

A modo de conclusión, los Códigos de Red establecen los requisitos de conexión y operación de módulos de generación eléctrica y de demanda, operación de las redes de transporte, y distribución del mercado eléctrico.

Red Eléctrica Española recoge en la misma sección, conocida como los Procedimientos de Operación (P.O), todas las publicaciones del Boletín Oficial del Estado que datan de 1998 a 2023 de carácter técnico e instrumental, necesarios para realizar una adecuada gestión técnica del sistema eléctrico peninsular y los sistemas eléctricos no peninsulares. Ejemplos de resoluciones de interés para este proyecto son las siguientes:

- Resolución del 30/07/1998: *P.O. 1.3 Tensiones admisibles nudos red.* Este documento se encarga de establecer perfiles de tensiones admisibles en distintos puntos de operación, como en condiciones normales o bajo contingencias.
- Resolución del 31/10/2002: *P.O. 6.1 Medidas de operación para garantizar la cobertura de la demanda en situaciones de alerta y emergencia*. Este documento se

encarga de definir requisitos de operación bajo contingencias por parte de los distintos elementos de los sistemas eléctricos.

- Resolución del 04/10/2006: *P.O. 12.3 Requisitos de respuesta frente a huecos de tensión de las instalaciones eólicas*. Este documento, como su propio nombre indica, se encarga de coordinar una respuesta por parte de las instalaciones eólicas a huecos bruscos de tensión.

Los procedimientos anteriormente mencionados fueron elaborados para sistemas peninsulares (la península ibérica). Para sistemas insulares como el de Tenerife (el sistema en el cual se está probando la herramienta a desarrollar por este proyecto) se definieron unos Procedimientos de Operación distintos debido a que, en general, los sistemas insulares suelen ser más débiles dado su menor tamaño y la falta de interconexiones con otros sistemas. De nuevo, ejemplos de dichas resoluciones son las siguientes:

- Resolución del 11/12/2019: *P.O. 1: Funcionamiento de los sistemas eléctricos no peninsulares.* Este documento se encarga de definir y recoger todos los criterios generales de seguridad y funcionamiento de sistemas eléctricos no insulares, como la operación en condiciones normales y bajo contingencias.
- Resolución del 11/12/2019: *P.O. 2.2: Cobertura de la demanda, programación de la generación y altas en el despacho económico*. Este documento se encarga de definir los criterios de seguridad y funcionamiento que deben aplicarse a la hora de garantizar una equidad entre generación y demanda.

Dentro de dichos documentos se pueden observar requisitos como la manutención del perfil de tensiones dentro de rango establecidos de todos los componentes de la red eléctrica, requisitos tanto estáticos como dinámicos de frecuencia de operación y requisitos de operación de las curvas P-Q y Q-V de los generadores, como se muestra en las siguientes imágenes:

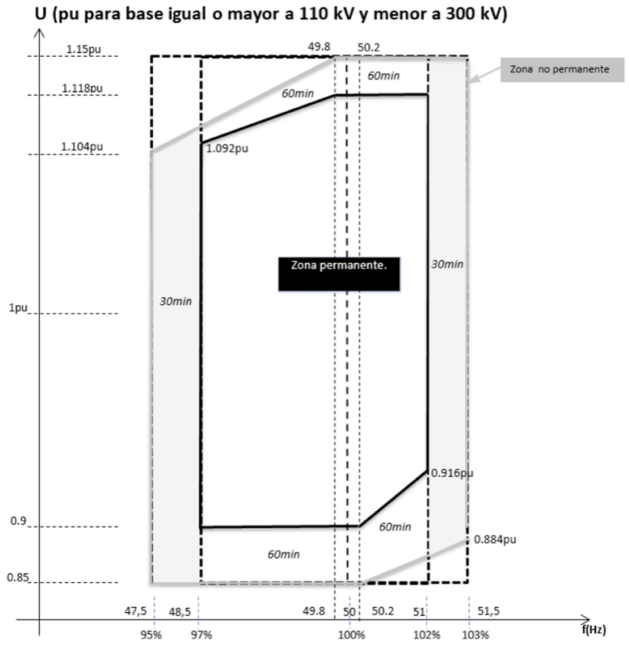


Figura 1 - Valores combinados de tensión-frecuencia que deberán ser capaces de soportar las instalaciones de distribución conectadas a la red de transporte, en el caso de que la tensión nominal del punto de conexión sea igual o mayor a 300 kV e igual o menor a 400 kV. [5]

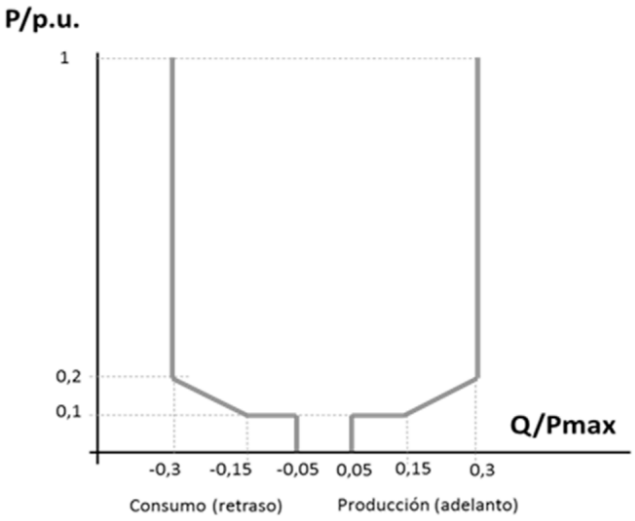


Figura 2 - Requisitos mínimos de operación P-Q de un módulo de generación para el caso en el que el parque eléctrico dispone de un cambiador de tomas en carga. [5]

1.2.2 HERRAMIENTAS

Hoy en día, se están utilizando por parte de las empresas de generación eléctrica softwares informáticos como PSSE, DIgSiLENT o PSCAD. Estos softwares, aparte de permitir un modelado preciso de módulos de generación eléctrica (desde centrales de ciclo combinado hasta parques eólicos y plantas fotovoltaicas), permiten simular la operación de la planta con mucho nivel de detalle .

PSSE es una potente herramienta informática que permite la ejecución de una gran variedad de análisis sobre redes eléctricas de generación, transporte y distribución, como la ejecución de análisis de contingencias, análisis dinámicos de perturbaciones, flujos de cargas y obtención de curvas de potencia, entre otras. Es uno de los principales softwares empleados en la industria, no sólo por su versatilidad, sino también por su gran capacidad de cómputo y precisión. Es usado por profesionales para garantizar un correcto cumplimiento de los códigos de red establecidos por el operador de mercado, como queda explicado en la *Sección 1.2.1*.

Algunos países insulares, como Irlanda, imponen por ley unos Códigos de Red muy estrictos para la operación de su red eléctrica y para la conexión de módulos de generación. El cumplimiento de dichos requisitos requiere análisis extensivos y constantes para un número muy amplio de situaciones. La realización de dichos análisis “a mano” es muy laboriosa y, en ocasiones, demasiado tedioso y repetitivo como para ser realizado por una persona. Esto ha creado la necesidad de automatizar la realización de dichos análisis, incorporando funcionalidades cada vez más complejas por parte de los propios softwares de análisis, y a través de métodos externos, como el uso de Python para ejecutar externamente scripts que realizan rutinas en dichos softwares (en concreto, en simuladores como PSSE o DIgSiLENT). De esta manera, programas como PSSE permiten la ejecución externa de scripts en lenguajes de programación como Python, o el uso de ficheros de código de uso específico para dichos programas, como IPLAN o IDEV. El uso de scripts externos permite automatizar la ejecución de tareas que resultarían muy largas en el caso de ser ejecutadas a mano.

1.3 DEFINICIÓN DEL TRABAJO

1.3.1 JUSTIFICACIÓN, MOTIVACIÓN Y OBJETIVOS DEL PROYECTO

La principal área de trabajo de este TFG nace de la, anteriormente mencionada, necesidad de desarrollar un programa que integre en uno y ejecute automáticamente todos los análisis necesarios de forma versátil y aplicable a sistemas muy variopintos

En este proyecto concreto, los estudios se particularizarán en un sistema insular, la isla de Tenerife. Los análisis que se integrarán son la ejecución del flujo de cargas óptimo (OPF), con un posterior análisis de contingencias N-1 (y N-2 en algún caso), y unos estudios dinámicos de desconexiones de componentes, cortocircuitos y contingencias. Posteriormente, los resultados de los últimos análisis realimentarían las consignas introducidas en el OPF, para poder ejecutar el análisis de nuevo. Esto permitirá realizar iteraciones del análisis que, en última instancia, llegarán a un punto de operación óptimo, y en el que las contingencias pueden ser mitigadas sin problemas. El fin último es la creación de una herramienta versátil que pueda adaptarse a distintos proyectos y sea fácil de usar por parte del usuario, que, a su vez, sea capaz de realizar análisis extensivos y detallados que se encuentren a la altura de los llevados a cabo por profesionales en empresas de operación eléctrica. La automatización de procesos de análisis de redes eléctricas libera considerablemente de carga de trabajo repetitiva a las personas encargadas de realizarlos.

A parte del principal objetivo descrito anteriormente, el crear una herramienta de análisis funcional que pueda ser utilizada por el Instituto de Investigación Tecnológica, otro importante objetivo de este trabajo es el énfasis en continuar investigando en el cumplimiento de los Códigos de Red y, sobre todo, el cumplimiento del código de red sobre la conexión de generadores nuevos, el cual se está convirtiendo en un factor importante a la hora de gestionar redes eléctricas, y de llevar una implementación segura de fuentes de generación renovable.

Otro evidente objetivo de la realización del trabajo es la búsqueda, como se describirá posteriormente, de una óptima operación de los sistemas de energía eléctrica, de manera que

se minimicen contingencias y se procure una operación más eficiente y barata. Dicha optimización en la operación de la red resultaría, en última instancia, en una operación menos costosa en términos de uso de materia prima para la construcción de infraestructura eléctrica y un uso de combustibles ligeramente menor para la generación de energía eléctrica.

Este proyecto, como se irá describiendo a lo largo de la memoria, se basa en el desarrollo de un conjunto de ficheros Python cuya ejecución permita llevar a cabo rutinas de análisis de redes en PSSE de forma automática y sin tener que acceder al propio software. A través de una entrada de datos a Python, se obtienen, ejecutando los ficheros, todos los resultados de los análisis pertinentes.

1.3.2 ALINEACIÓN CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE (ODS)

Puesto que el objetivo, en última instancia, de la realización de este proyecto es incentivar una implementación segura en los sistemas eléctricos actuales de fuentes de energía renovable, los objetivos de este proyecto están, en gran medida, en consonancia con los objetivos de la implementación de fuentes de energía renovables.

En general, las posibles contribuciones positivas que pueda realizar este proyecto son en forma de una reducción de las emisiones por quema de combustibles fósiles, tanto por un mayor uso de fuentes de energía renovables como por una reducción en las propias pérdidas de energía en transporte y distribución de la energía. Puesto que, como su propio nombre indica, los Objetivos de Desarrollo Sostenible persiguen un futuro próspero y comprometido con la sostenibilidad de nuestro planeta, una mejora en las emisiones de contaminantes al medio ambiente resulta indirectamente en la alineación con varios de los objetivos propuestos por la ONU.

De esta manera, el principal objetivo que persigue es una producción de energía limpia y no contaminante a través de dicho incentivo del uso de fuentes de energía renovables, lo que corresponde al objetivo número 7. Dicho objetivo deriva en otros relacionados, como la promoción de ciudades y comunidades sostenibles gracias a una generación de electricidad

renovable y deslocalizada (objetivo número 11) y la acción por el clima reduciendo la emisión de contaminantes por la quema de combustibles fósiles (objetivo número 13).

1.3.3 METODOLOGÍA DEL TRABAJO Y RECURSOS A EMPLEAR

El trabajo desarrollado ha consistido en la elaboración a lo largo del curso académico de una serie de scripts en Python que reciben y procesan una serie de archivos de PSSE y datos, realizan una serie de análisis mediante una ejecución única de la herramienta, y exportan una serie de resultados y estudios en archivos Excel y Word.

De esta forma, en la *Figura 3* se muestra un diagrama explicativo del proceso, en el cual se ejecuta un análisis completo sobre el sistema, y el cual contiene los correspondientes ficheros para cada paso, se puede observar en la siguiente figura:

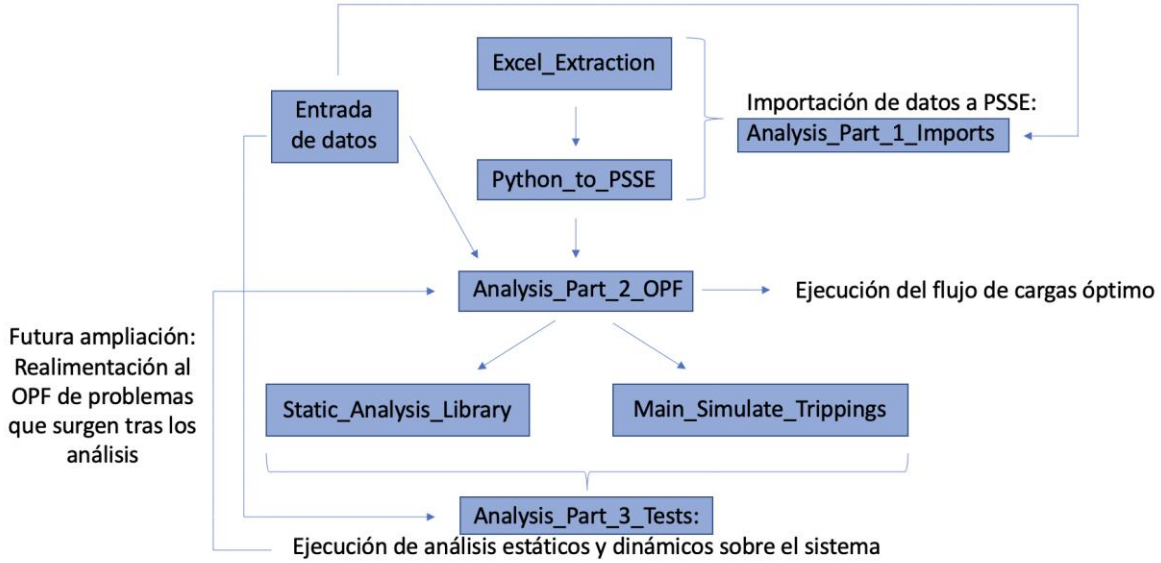


Figura 3 – Diagrama de arquitectura de la herramienta

El contenido de la herramienta es el siguiente:

- Archivos para aportar por el usuario (Entrada de datos):
  - Fichero (.sav) PSSE que contiene los datos del sistema eléctrico a analizar. Se puede interpretar como un caso base del sistema.

- Fichero (.dyr) PSSE que contiene características dinámicas del sistema a analizar (para la realización de análisis dinámicos)
- Ficheros de texto proveniente del despacho económico que contiene la generación convencional (desglosada por máquinas), generación renovable y demanda por horas del sistema eléctrico a analizar.
- Un documento Excel en el que figuren las equivalencias, en términos de nomenclatura de generación y demanda, entre los ficheros del departamento económico y el fichero PSSE.

- Ficheros de Python que realizarán las siguientes rutinas (en orden de ejecución):
  - *Analysis_Part_1_Imports, Analysis_Part_2_OPF, y Analysis_Part_3_Tests son los* ficheros principales desde los que se ejecutan el resto de los módulos secundarios de Python, que ejecutan todas las rutinas del análisis del sistema .
  - *Analysis_Part_1_Imports*:
    - *Excel_Extraction*: Módulo en el cual se realiza la importación a Python y clasificación de los datos aportados por el usuario, para su posterior uso
    - *Python_To_PSSE*: Módulo en el que se incluyen, ya ordenados, todos los datos importados de *Excel_Extraction* a ficheros de PSSE que se crean para su posterior análisis en *Analysis_Part_2_OPF*
  - *Analysis_Part_2_OPF*: Módulo en el que se ejecuta la herramienta OPF (Optimal Power Flow) de PSSE, la cual devuelve un sistema funcional con una operación óptima según unos parámetros establecidos.
  - *Analysis_Part_3_Tests:*
    - *Static_Analysis_Library*: Módulo en el que se realiza un análisis de contingencias del sistema (desconexiones de líneas, transformadores y generadores), y se exportan a Excel datos sobre las posibles violaciones de requisitos de operación que puedan ocurrir
    - *Main_Simulate_Trippings*: Módulo en el que se realizan análisis dinámicos (desconexiones de generadores y cortocircuitos en nudos) y se exportan a Excel datos sobre las posibles violaciones en régimen permanente de requisitos de operación que puedan ocurrir

- Archivos para exportar por la herramienta:
    - Archivos (.sav) de PSSE con una operación óptima del sistema, el cual puede anticipar y solucionar adecuadamente posibles contingencias.
    - Documentos Excel y Word que contienen los resultados de los análisis, tanto estáticos como dinámicos.

La compatibilidad entre PSSE y Python como lenguaje de programación para ejecutarlo, permite generar en Python algoritmos que ejecuten los programas sin la necesidad de tener abierto PSSE gracias a librerías preestablecidas, además de disponer PSSE de una herramienta que transforma automáticamente comandos dentro de la aplicación en ficheros de lenguajes de programación como Python conocida como la grabadora (*ANEXO 8: USO DE PSSE JUNTO CON PYTHON*). De esta manera, el uso activo de la aplicación de PSSE durante la elaboración del proyecto será escasa, y se utilizará primordialmente durante las fases de elaboración de los ficheros Python.

Capítulo 2. DESCRIPCIÓN DE FUNCIONALIDADES

Como se ha comentado anteriormente, el objetivo principal de este trabajo es la automatización del uso de PSSE a la hora de optimizar la operación de redes eléctricas. De esta manera, se han diseñado ficheros de Python que permiten ejecutar labores en PSSE de forma rápida y automática. Este capítulo se centra en explicar los fundamentos teóricos en los que se rigen los análisis a ejecutar en PSSE.

2.1 EJECUCIÓN DEL FLUJO DE CARGAS

El flujo de cargas en PSSE se puede ejecutar mediante el método Newton-Raphson (aunque haya otros disponibles), un método numérico iterativo que permite la resolución de ecuaciones diferenciales. PSSE calcula la matriz de admitancias de la red a evaluar, y plantea las ecuaciones de equilibrio de potencias en cada nudo, como se puede observar en las próximas imágenes:

$$\begin{bmatrix} \mathbf{y}_{11} & \cdots & \mathbf{y}_{1N} \\ \vdots & \ddots & \vdots \\ \mathbf{y}_{N1} & \cdots & \mathbf{y}_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_N \end{bmatrix} = \begin{bmatrix} \mathbf{i}_1 \\ \vdots \\ \mathbf{i}_N \end{bmatrix}$$

$$\mathbf{y}_{km} = g_{km} + jb_{km}$$

$$\mathbf{v}_k = v_k \angle \theta_k = v_k \, \mathrm{e}^{j\theta_k} = v_k \left(\cos\theta_k + j \sin\theta_k \right)$$

Figura 4 – Ecuaciones de nudos de un sistema, en la que se utiliza la matriz de admitancias Y [9]

$$\begin{aligned} \mathbf{s}_k &= p_k + jq_k = \left(p_{Gk} - p_{Dk}\right) + j\left(q_{Gk} - q_{Dk}\right) = \\ &= \mathbf{v}_k \mathbf{i}_k^* = \mathbf{v}_k \sum_{m=1}^{N} \mathbf{y}_{km}^* \mathbf{v}_m^* = \mathbf{y}_{kk}^* v_k^2 + \sum_{\substack{m=1 \\ m \neq k}}^{N} \mathbf{Y}_{km}^* \mathbf{v}_k \mathbf{v}_m^* \\ &= \left(g_{kk} - jb_{kk}\right) v_k^2 \\ &+ \sum_{\substack{m=1 \\ m \neq k}}^{N} \left(g_{km} - jb_{km}\right) v_k v_m \left[\cos\left(\theta_k - \theta_m\right) + j \sin\left(\theta_k - \theta_m\right)\right] \end{aligned}$$

Figura 5 – Balance de potencias en un nudo k del sistema [9]

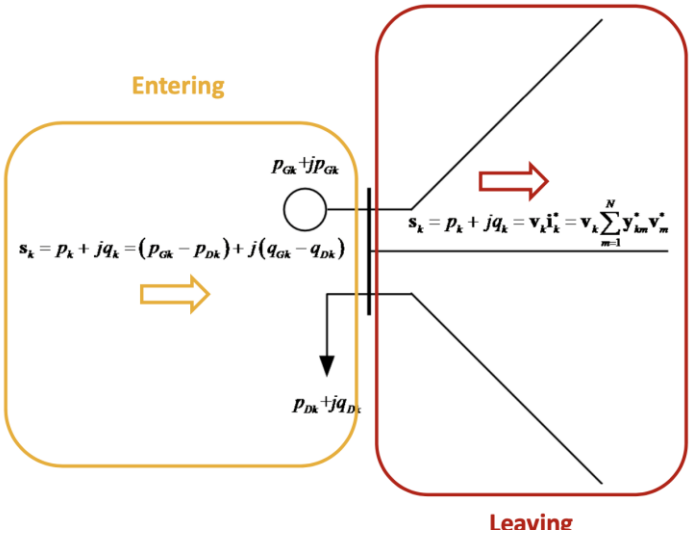


Figura 6 – Balance de potencias en un nudo k [9]

Una vez planteadas las ecuaciones de balance de potencias en cada nudo (en el cual, según el tipo que sea, 1, 2 o 3, se conoce el valor de su voltaje, su ángulo o sus potencias activa y reactiva), se aplica el método de Newton-Raphson. Puesto que, como con otros métodos numéricos, se busca hallar la solución a una ecuación compleja aproximándola a la recta tangente a una solución inicial dada; mediante suficientes iteraciones, se puede obtener un valor preciso de los parámetros de la red.

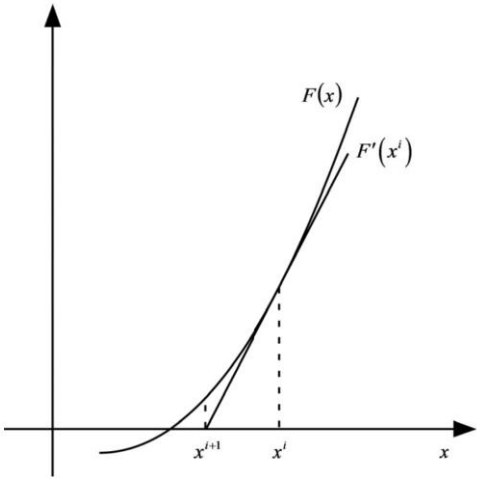


Figura 7 – Interpretación geométrica del método Newton-Raphson [9]

Además, PSSE permite implantar acciones como la modificación de tomas de transformadores, conexión de reactancias y condensadores o la regulación de la potencia intercambiada entre áreas.

2.2 EJECUCIÓN DE OPTIMAL POWER FLOW (OPF)

Para la determinación de unas condiciones iniciales óptimas de funcionamiento, el estado inicial del sistema se determinará mediante la ejecución del flujo de cargas óptimo (OPF) en PSSE. Comparado con el flujo de cargas, el OPF tiene el añadido de ser un flujo de cargas que se resuelve como un problema de optimización, en el cual se busca optimizar una función objetivo (por ejemplo, minimizar pérdidas de potencia), mientras que se procura que todas las variables (tensiones, flujos de potencia, generación, tomas de transformadores, susceptancia de *shunts*, intercambio entre áreas…) se encuentran dentro de unos límites establecidos. Las variables de decisión son típicamente la generación de potencia activa, las tensiones de consignas de los generadores o en su defecto la generación de potencia reactiva, las tomas de los transformadores, etc.

A continuación, se muestra una imagen del planteamiento del problema de optimización, en el cual f(x) es la función de la variable a optimizar, g(x) son las ecuaciones del balance de potencias, h(x) son los límites establecidos para cada variable y x es el vector de todas las posibles variables de decisión:

$$Min\ f(x)$$

$$sujeto\ a: g(x) = 0; h(x) \leq 0$$

$$x^T = [p_g^T\ \ q_g^T\ \ v^T\ \theta^T\ k^T\ \phi^T\ b^T]$$

Ecuación 1 - Planteamiento del OPF [9], donde pg y qg se refieren a la potencia generada, v a las tensiones, theta a los ángulos, k a tomas de transformadores, phi a desfases de transformadores, y b a la conexión de elementos shunt

Para otorgar más o menos importancia a ciertas restricciones (restricciones de tensión, de flujos por las líneas), PSSE permite aplicar sensibilidades a la aplicación de dichas restricciones en vez de poner límites estrictos, de manera que se priorice relajar ciertas restricciones en caso de que haya que garantizar la convergencia del caso.

Puesto que las restricciones aplicadas no son lineales, el problema es de grandes dimensiones, y que se generan inecuaciones al aplicar los límites, la resolución de función de optimización suele ser muy complicada. Para resolverlo, PSSE computa una función escalar Lagrangiana con la función a optimizar y las funciones a cumplir (*Ecuación 2*), donde μ y λ son los multiplicadores lagrangianos de las restricciones. Después, según el teorema de Karush-Kuhn-Tucker, se plantean las condiciones por las que se obtienen los valores óptimos de las variables (*Ecuación 3*), y se resuelven, como se ha visto en la *sección 2.1*, mediante el método iterativo de Newton-Raphson [9].

$$L(\mathbf{x},\lambda,\mu) = f(\mathbf{x}) + \lambda^T \cdot \mathbf{g}(\mathbf{x}) + \mu^T \cdot \mathbf{h}(\mathbf{x})$$

Ecuación 2 – Función lagrangiana computada durante la ejecución del OPF [9]

$$\left.\frac{\partial L(\mathbf{x},\lambda,\mu)}{\partial \mathbf{x}}\right|_{\mathbf{x}^*,\lambda^*,\mu^*} = \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} + \left(\left.\frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}^*}\right)^T \cdot \lambda^* + \left(\left.\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}^*}\right)^T \cdot \mu^* = \mathbf{0}$$

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$$

$$\text{diag}\{\mu^*\} \cdot \mathbf{h}(\mathbf{x}^*) = \mathbf{0}$$

$$\mu^* \geq \mathbf{0}$$

Ecuación 3 – Condiciones de optimalidad de Karush-Kuhn-Tacker [9]

De esta manera, la ejecución del OPF proporcionará una solución que optimizará las variables del interés para el usuario, a la vez que mantiene el perfil de tensiones, generación y flujos de potencia dentro de los límites. En este caso concreto, el OPF proporcionará las conexiones de tomas de transformadores, tensiones de consigna de los generadores y conexiones de elementos *shunt* necesarias para propiciar dichas condiciones óptimas. La

potencia activa despachada por cada generador está establecida por el despacho económico, por lo que no se modificará.

2.3 EJECUCIÓN DE ANÁLISIS ESTÁTICOS DE CONTINGENCIAS N-1 N-2

La ejecución de los análisis estáticos de contingencias N-1 (una única contingencia) y N-2 (dos contingencias a la vez) se ejecutan de manera sencilla; se provoca dicha contingencia, y se ejecuta un flujo de cargas como en la *Sección 2.1*, de manera que se obtienen los parámetros de la red en régimen permanente.

Para los casos a estudiar en este proyecto, se realizará el apagado individual de todos transformadores y las líneas del sistema, y se evaluarán sus consecuencias para el caso N-1. Para el caso N-2, se realizará el apagado sólo de líneas paralelas a modo de contingencias puesto que es común que líneas paralelas compartan apoyos, los cuales, al fallar, provocan la pérdida de suministro de ambas líneas.

2.4 EJECUCIÓN DE ANÁLISIS DINÁMICOS DE PÉRDIDAS DE GENERACIÓN

Los análisis dinámicos de pérdidas de generación se centrarán en el estudio dinámico de la regulación primaria del sistema tras el apagado individual de generadores. La pérdida de generación es uno de los eventos que pueden dar lugar a inestabilidades en frecuencia.

Los modelos dinámicos en PSSE permite incluir la regulación primaria para el análisis dinámica de pérdidas de generación. La regulación primaria de un sistema se realiza aplicando un control proporcional de lazo cerrado al diagrama de bloques equivalente del sistema. Como se puede observar de forma simplificada en la *Figura 8,* una variación en la generación de la potencia del sistema provoca un aumento de su frecuencia, y viceversa. En la figura, la variable *H* compone la suma de la inercia de todos los generadores del sistema

y *D,* la sensibilidad de la demanda, representa un valor experimental que cuantifica la variación de la demanda con respecto a la frecuencia.

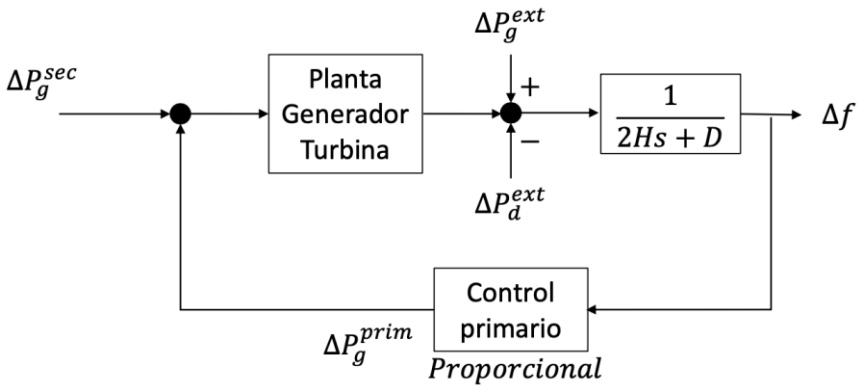


Figura 8 – Diagrama de bloques de control frecuencia-potencia de un sistema bajo regulación primaria de los generadores [10]

El control de regulación primaria (la cantidad que genera o deja de generar un generador en el evento de un desequilibrio de potencia activa, dando lugar a un desvío de la frecuencia de su valor nominal) de cada generador es proporcional a la variación de la frecuencia en régimen permanente, e inversamente proporcional a su estatismo, el cual suele estar relacionado con la potencia nominal del generador:

$$\Delta P_g^{prim} = -\frac{1}{R} * \Delta f$$

Estatismo

Figura 9 – Funcionamiento de la regulación primaria de un generador [10]

Antes de realizar los análisis dinámicos, e independientemente de la perturbación dinámica que se estudia, PSSE realiza una compleja serie de preparativos para acomodar el sistema para la ejecución del análisis dinámico, los cuales se pueden resumir en los siguientes pasos [11]:

- Modelado de los generadores como fuentes de corriente con su reactancia equivalente en paralelo

- Obtención de la matriz ampliada de admitancias del sistema en la que incluyen reactancias de generadores y demandas y
- Ordenación y factorización de dicha matriz de admitancias
- En cada paso del análisis, de duración establecida por el usuario, PSSE adopta una solución secuencial con un método de integración explícito:
    - Obtención de la potencia (corrientes) a inyectar por cada generador en función de la frecuencia
    - Resolución de las ecuaciones de red (I = Y*V) mediante el método de nudos.
    - Resolución de las ecuaciones diferenciales del sistema mediante el método de integración numérica explícito de Heun.

Tras el final del análisis, PSSE exporta un vector con todas las variables de interés y su valor en función del tiempo.

2.5 EJECUCIÓN DE ANÁLISIS DINÁMICOS DE CORTOCIRCUITOS EN NUDOS

El procedimiento para seguir a la hora de ejecutar análisis dinámicos de cortocircuitos en nudos de la red es muy similar al seguido a la hora de ejecutar los análisis de pérdidas de generación; se prepara el sistema para la ejecución del análisis dinámico, se aplica la falta, se graba el comportamiento del sistema y se exportan los datos.

En este caso, la falta se aplica solamente durante un breve período de tiempo (generalmente en torno a 100ms), periodo durante el cual la red se perturba considerablemente. PSSE permite la ejecución de cortocircuitos entre líneas o con la tierra, aparte de poder establecer la impedancia del cortocircuito y la fase en la que se aplica.

En los análisis de cortocircuitos a realizar en este proyecto, no se aplican protecciones o respuestas al cortocircuito. Simplemente, se aplica un cortocircuito durante un tiempo y se analiza el comportamiento de la red durante los segundos posteriores.

Capítulo 3. EJECUCIÓN DE LOS ANÁLISIS

Este capítulo se centra en describir en detalle cómo se ha desarrollado cada uno de los módulos de la herramienta Python, las entradas de datos que necesitan, y los resultados que exportan. A continuación, se muestra de nuevo un diagrama de arquitectura de la herramienta y, además, un diagrama que muestra cómo están estructurados los archivos del proyecto:

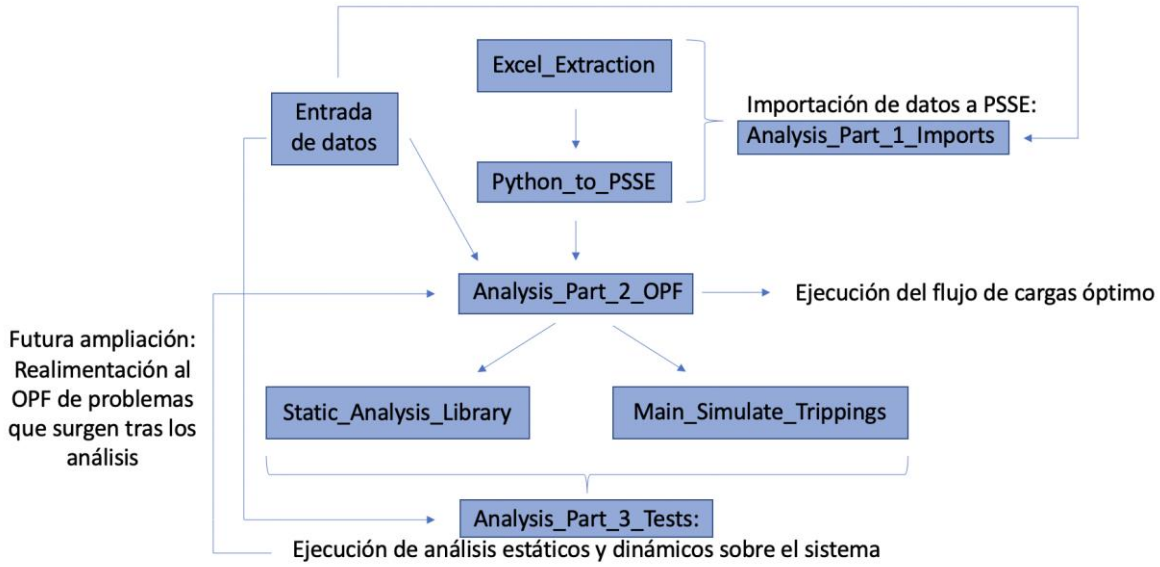


Figura 10 – Diagrama de arquitectura de la herramienta

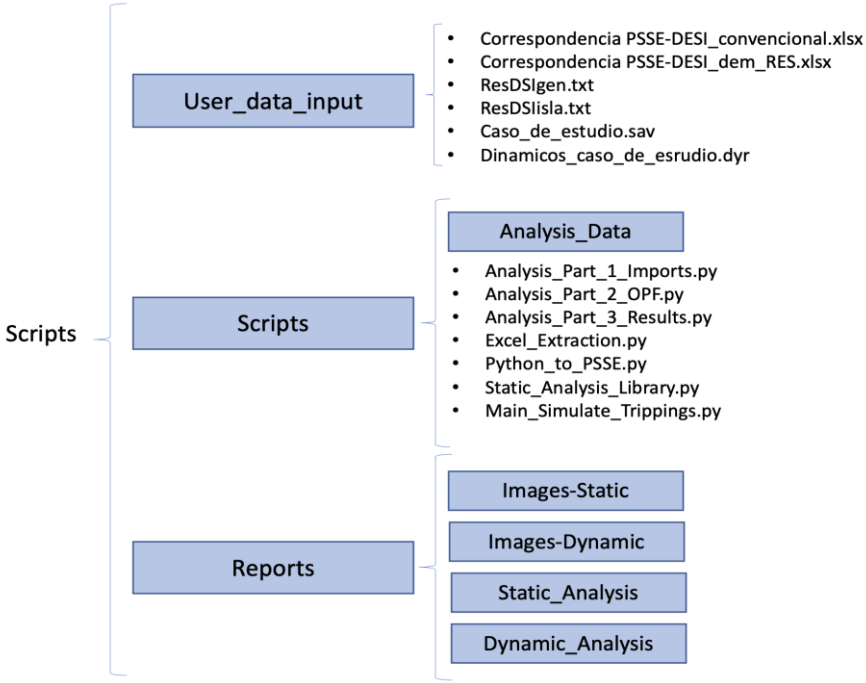


Figura 11 – Jerarquía de la carpeta del proyecto

3.1 ORGANIZACIÓN DE LOS FICHEROS Y DOCUMENTOS

Los módulos están diseñados para importar y exportar datos de un directorio con la misma configuración como el que se va a mostrar a continuación. Se han creado una serie de carpetas y subcarpetas que permiten a Python crear, acceder y ordenar todos los ficheros necesarios dentro de la carpeta del proyecto. Aunque el orden y la jerarquía del directorio es rígido, los nombres de las carpetas se pueden modificar, y en cada uno de los ficheros de Python existe un apartado donde el usuario puede definir las entradas y salidas de datos.

En el primer nivel, como se puede observar en la *Figura 11,* se definen tres carpetas: La carpeta *Reports*, en la que se exportan todos los resultados de los análisis, la carpeta *Scripts,* carpeta que contiene todos los scripts de Python, casos de estudio intermedios, y otros ficheros necesarios para los análisis que el usuario no necesita acceder o modificar. Por último, en la carpeta *User_data_input,* el usuario introduce todos los archivos necesarios para realizar el análisis:

Dentro de la carpeta *User_data_input* se encuentran los siguientes archivos:

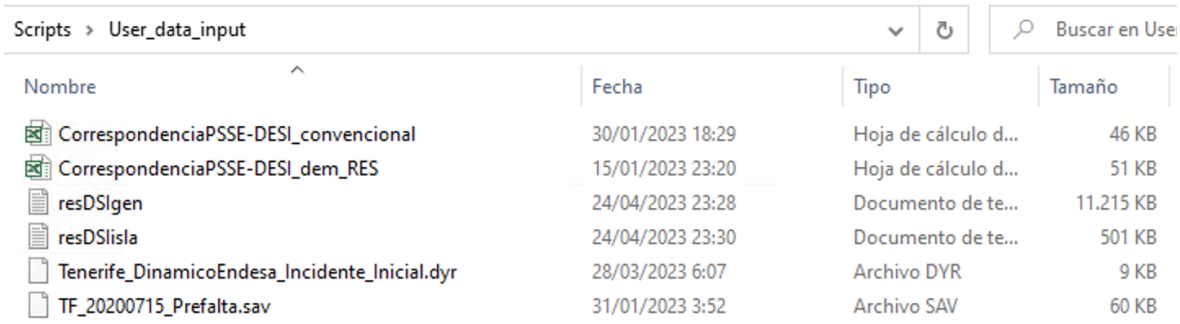
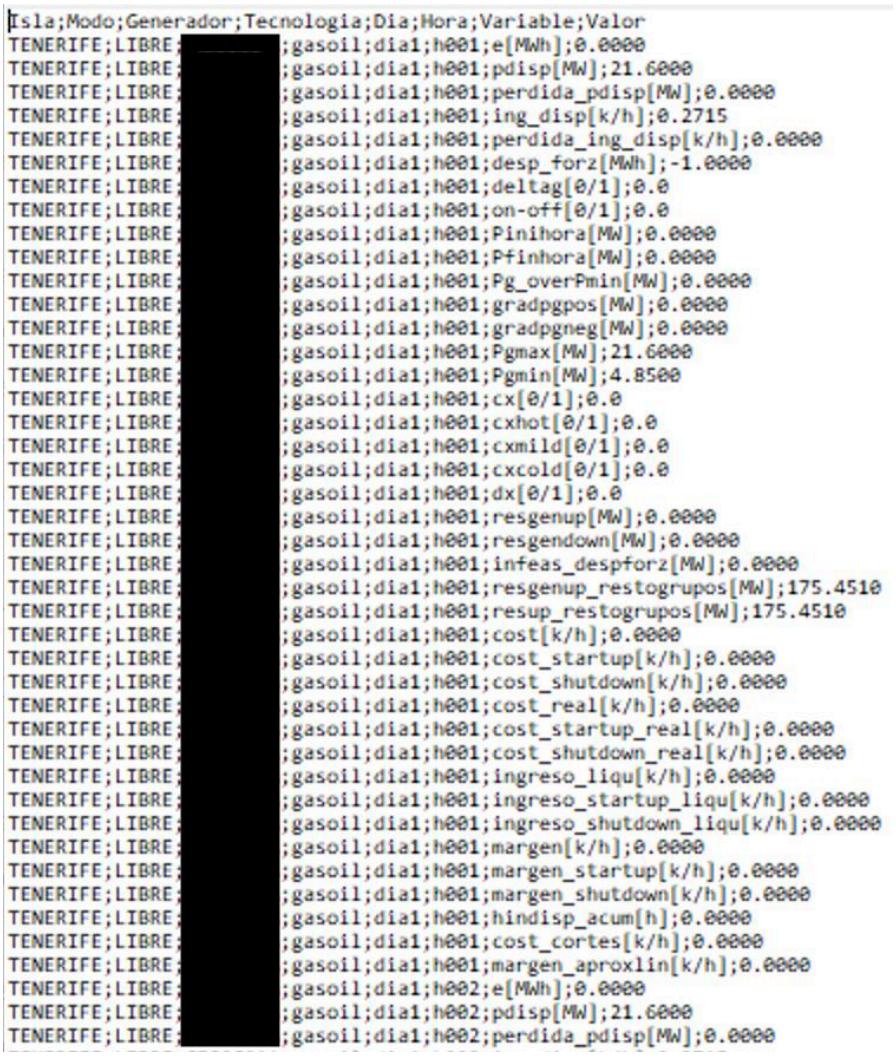


Figura 12 – Contenido de la carpeta User_data_input

Los ficheros .sav son los casos de estudio convencionales de PSSE que contienen la configuración de la red a estudiar, y los ficheros .dyr contienen todos los datos relacionados con el comportamiento dinámico del sistema, como datos y modelos relacionados con la regulación primaria de los generadores (estatismo, e inercia de los generadores), con los sistemas de excitación, la máquina síncrona, etc.

Los ficheros de texto *resDSIgen y resDSIisla* (*Figura 13* y *Figura 14,* respectivamente) contienen los datos de generación y demanda del caso a estudiar. En concreto, el fichero *resDSIisla* contiene, por horas durante un día o una semana, todos los datos relacionados con la demanda de Tenerife, como la demanda total, generación renovable eólica y fotovoltaica, y parámetros relacionados con el rendimiento económico. El fichero *resDSIgen* contiene, para generador y cada hora, datos relacionados con la generación actual, capacidad de generación, ingresos y costes, y otros parámetros relacionados con la regulación frecuencia-potencia como la reserva total disponible. Por motivos de una gestión adecuada de información sensible, el nombre del generador de la *Figura 13* se ha ocultado.



```
Isla;Modo;Generador;Tecnologia;Dia;Hora;Variable;Valor
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;e[MWh];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;pdisp[MW];21.6000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;perdida_pdisp[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;ing_disp[k/h];0.2715
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;perdida_ing_disp[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;desp_forz[MWh];-1.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;deltag[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;on-off[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;Pinihora[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;Pfinhora[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;Pg_overPmin[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;gradpgpos[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;gradpgneg[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;Pgmax[MW];21.6000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;Pgmin[MW];4.8500
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cx[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cxhot[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cxmild[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cxcold[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;dx[0/1];0.0
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;resgenup[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;resgendown[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;infeas_despforz[MW];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;resgenup_restogrupos[MW];175.4510
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;resup_restogrupos[MW];175.4510
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_startup[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_shutdown[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_real[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_startup_real[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_shutdown_real[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;ingreso_liqu[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;ingreso_startup_liqu[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;ingreso_shutdown_liqu[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;margen[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;margen_startup[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;margen_shutdown[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;hindisp_acum[h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;cost_cortes[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h001;margen_aproxlin[k/h];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h002;e[MWh];0.0000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h002;pdisp[MW];21.6000
TENERIFE;LIBRE;[redacted];gasoil;dia1;h002;perdida_pdisp[MW];0.0000
```

Figura 13 – Datos de generación de un generador para una hora de Tenerife

```
Isla;Modo;Dia;Hora;Variable;Valor
TENERIFE;LIBRE;dia1;h001;dem[MWh];311.8300
TENERIFE;LIBRE;dia1;h001;ess_charge[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;wind[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;wind_vertido[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;solar_vertido[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;solar[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;genresto[MWh];59.8810
TENERIFE;LIBRE;dia1;h001;e[MWh];251.9490
TENERIFE;LIBRE;dia1;h001;e_carbon[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;e_gas_natural[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;e_fueloil[MWh];176.4490
TENERIFE;LIBRE;dia1;h001;e_gasoil[MWh];75.5000
TENERIFE;LIBRE;dia1;h001;e_hvdc[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;pdisp[MW];1591.0000
TENERIFE;LIBRE;dia1;h001;ing_disp[k/h];18.1492
TENERIFE;LIBRE;dia1;h001;perdida_ing_disp[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;ess_discharge[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;charge_level[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;deltag[N];6.0
TENERIFE;LIBRE;dia1;h001;cx[N];0.0
TENERIFE;LIBRE;dia1;h001;dx[N];1.0
TENERIFE;LIBRE;dia1;h001;resgenup[MW];175.4510
TENERIFE;LIBRE;dia1;h001;resgenup_sinMayorGrupo[MW];44.4510
TENERIFE;LIBRE;dia1;h001;ressub_enlace[MW];0.0000
TENERIFE;LIBRE;dia1;h001;resbaj_enlace[MW];0.0000
TENERIFE;LIBRE;dia1;h001;resup[MW];175.4510
TENERIFE;LIBRE;dia1;h001;resup_sinMayorGrupo[MW];44.4510
TENERIFE;LIBRE;dia1;h001;resessup[MW];0.0000
TENERIFE;LIBRE;dia1;h001;resgendown[MW];71.9490
TENERIFE;LIBRE;dia1;h001;resdown[MW];71.9490
TENERIFE;LIBRE;dia1;h001;resdown/resup[pu];0.4101
TENERIFE;LIBRE;dia1;h001;resdown/resup_sinmayorgrupo[pu];1.6186
TENERIFE;LIBRE;dia1;h001;resessdown[MW];0.0000
TENERIFE;LIBRE;dia1;h001;MaxE[MWh];75.5000
TENERIFE;LIBRE;dia1;h001;MinResBajar[MWh];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Subir_generacion[MW];0.0000
TENERIFE;LIBRE;dia1;h001;infeas_dem_pos[MW];0.0000
TENERIFE;LIBRE;dia1;h001;infeas_dem_neg[MW];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Subir_importacion[MW];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Subir_renovables[MW];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Subir_Resmin[MW];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Subir[MW];0.0000
TENERIFE;LIBRE;dia1;h001;Infeas_Res_Bajar[MW];0.0000
TENERIFE;LIBRE;dia1;h001;coste[k/h];37.1239
TENERIFE;LIBRE;dia1;h001;coste_startup[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;coste_shutdown[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;coste_total[k/h];37.1239
TENERIFE;LIBRE;dia1;h001;coste_real[k/h];32.5935
TENERIFE;LIBRE;dia1;h001;coste_startup_real[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;coste_shutdown_real[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;coste_total_real[k/h];32.5935
TENERIFE;LIBRE;dia1;h001;ingreso_liqu[k/h];37.0831
TENERIFE;LIBRE;dia1;h001;ingreso_startup_liqu[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;ingreso_shutdown_liqu[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;ingreso_total_liqu[k/h];37.0831
TENERIFE;LIBRE;dia1;h001;margen[k/h];4.4896
TENERIFE;LIBRE;dia1;h001;margen_startup[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;margen_shutdown[k/h];0.0000
TENERIFE;LIBRE;dia1;h001;ing_disp[k/h];18.1492
TENERIFE;LIBRE;dia1;h001;margen_total[k/h];4.4896
TENERIFE;LIBRE;dia1;h002;dem[MWh];282.2200
TENERIFE;LIBRE;dia1;h002;ess_charge[MWh];0.0000
```

Figura 14 – Datos de demanda en Tenerife para una hora

Por último, los archivos Excel *CorrespondenciaPSSE-DESI_convencional* y *CorrespondenciaPSSE-DESI_dem_RES,* contienen las equivalencias de nombres y

parámetros de nudos y generadores entre el despacho económico contenido en los archivos de texto y la red configurada en PSSE, a utilizar por los ficheros Python.

Dentro de la carpeta *Scripts* (*Figura 15*), se encuentran todos los ficheros de Python que se van a utilizar, así como la carpeta *Analysis_Data,* en la que se almacenan ficheros relacionados con los análisis.

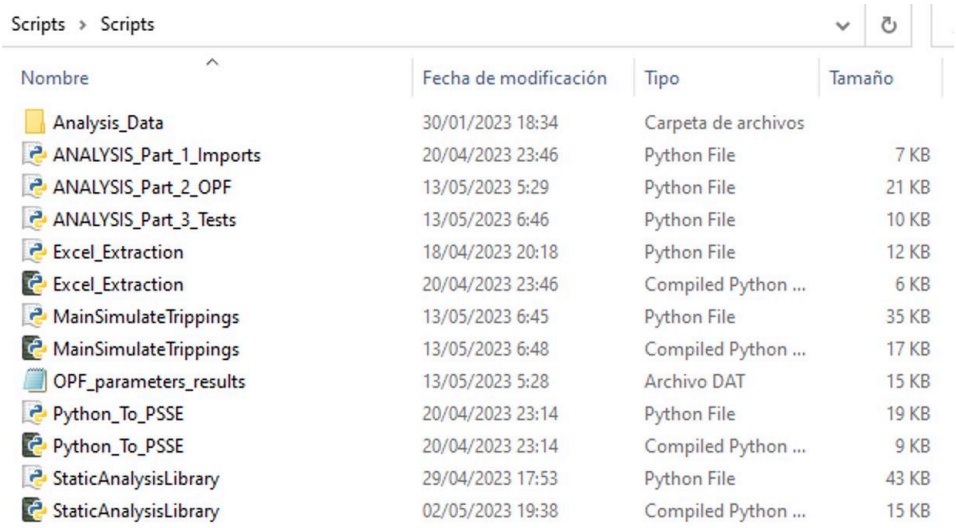


Figura 15 – Contenido de la carpeta Scripts

A su vez, dentro de la carpeta *Analysis_Data,* se encuentran otras tres carpetas: *Case_data* (*Figura 16*), en la que se almacenan los casos de PSSE .raw (equivalente al .sav) creados tras haber importado los datos de generación y demanda, y los archivos .sav que contienen los datos de la configuración de Tenerife obtenidos tras ejecutar el OPF, la carpeta *Dynamic_Files*, donde se generan y almacenan archivos, de poco interés para el usuario, relacionados con los análisis dinámicos, y la carpeta *Excel_Files*, en la cual se almacenan documentos Excel, de nuevo, de poco interés, que clasifican y almacenan todos los datos introducidos por el usuario.

Debido a la necesidad de ejecutar el OPF en PSSE 32 (Python 2.5), ha sido necesario la creación de archivos .raw compatibles con PSSE 32 (en vez de .sav) con los datos del sistema para ser introducidos en el OPF.

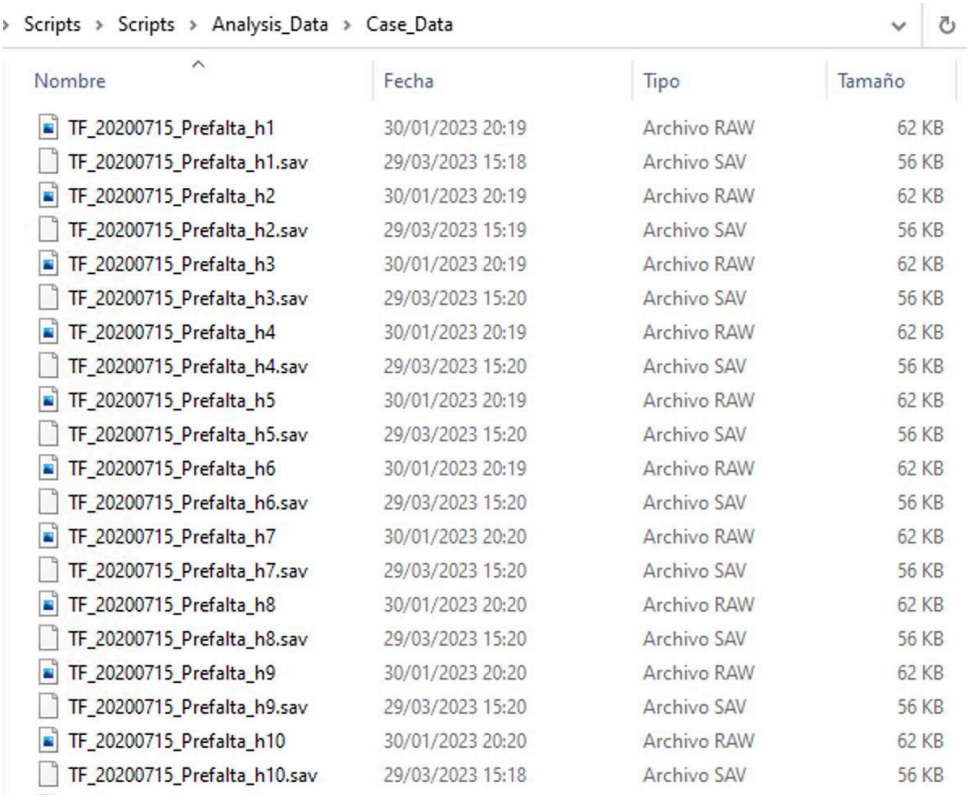


Figura 16 – Parte del contenido de la carpeta Case_Data

Por último, de cara a la organización de los datos a exportar, se asigna cada tipo de análisis (análisis estáticos, y análisis dinámicos) una carpeta en la que se almacena los archivos Excel y Word, y otro donde se almacenan imágenes generadas de interés.

Como se puede observar en la *Figura 17,* dentro del contenido de la carpeta de análisis estáticos se almacenan documentos Word y Excel con los resultados del OPF (*ResultsOPF)*, y documentos Word y Excel con los resultados de los análisis de contingencias N-1, N-2 (*ResultsN-1N-2_Fichero_y_hora_evaluados)*. Estos últimos sólo se generan en caso de que se detecte alguna violación de límites para alguno de los casos evaluados.

El contenido de estos archivos se explorará con más detalle en el *Capítulo 4.*

ResultsN-1N-2TF_20200715_Prefalta_h142	02/05/2023 23:33	Hoja de cálculo d...	139 KB
ResultsN-1N-2TF_20200715_Prefalta_h143	02/05/2023 23:42	Hoja de cálculo d...	135 KB
ResultsN-1N-2TF_20200715_Prefalta_h144	02/05/2023 23:42	Hoja de cálculo d...	13 KB
ResultsN-1N-2TF_20200715_Prefalta_h145	02/05/2023 23:43	Hoja de cálculo d...	9 KB
ResultsN-1N-2TF_20200715_Prefalta_h146	02/05/2023 23:47	Hoja de cálculo d...	57 KB
ResultsN-1N-2TF_20200715_Prefalta_h147	02/05/2023 23:51	Hoja de cálculo d...	63 KB
ResultsN-1N-2TF_20200715_Prefalta_h148	02/05/2023 23:56	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h149	03/05/2023 0:00	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h150	03/05/2023 0:05	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h151	03/05/2023 0:10	Hoja de cálculo d...	63 KB
ResultsN-1N-2TF_20200715_Prefalta_h152	03/05/2023 0:10	Hoja de cálculo d...	7 KB
ResultsN-1N-2TF_20200715_Prefalta_h153	03/05/2023 0:10	Hoja de cálculo d...	8 KB
ResultsN-1N-2TF_20200715_Prefalta_h154	03/05/2023 0:10	Hoja de cálculo d...	13 KB
ResultsN-1N-2TF_20200715_Prefalta_h155	03/05/2023 0:15	Hoja de cálculo d...	61 KB
ResultsN-1N-2TF_20200715_Prefalta_h156	03/05/2023 0:19	Hoja de cálculo d...	67 KB
ResultsN-1N-2TF_20200715_Prefalta_h157	03/05/2023 0:23	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h158	03/05/2023 0:25	Hoja de cálculo d...	45 KB
ResultsOPF	02/05/2023 20:44	Documento de Mi...	473 KB
ResultsOPF	02/05/2023 20:44	Hoja de cálculo d...	652 KB

Figura 17 – Parte de los resultados exportados para los análisis estáticos

En la *Figura 18,* se puede observar el contenido exportado de los análisis dinámicos para cada sistema y hora evaluados:

- Un fichero Word con los resultados dinámicos del análisis de pérdidas de generación (*DYNresultsGenTrip_Fichero_y_hora_evaluados)*
- Un fichero Word con los resultados dinámicos del análisis en nudos de generación de cortocircuitos en nudos selectos por el usuario (*DYNresultsBusSC_Fichero_y_hora_evaluados)*
- Un fichero Excel con los análisis estáticos de pérdidas de generación (*ResultsN-1_Gen_Fichero_y_hora_evaluados).* Estos resultados se exportan en esta sección, puesto que los análisis dinámicos de pérdidas de generación permiten la aplicación de la regulación primaria de generación durante el análisis, en contraposición con los análisis estáticos en los que simplemente se desconecta un generador, por lo que las conclusiones extraídas se asemejan más a la realidad.

De nuevo, el contenido de estos archivos se explorará con más detalle en el *Capítulo 4.*

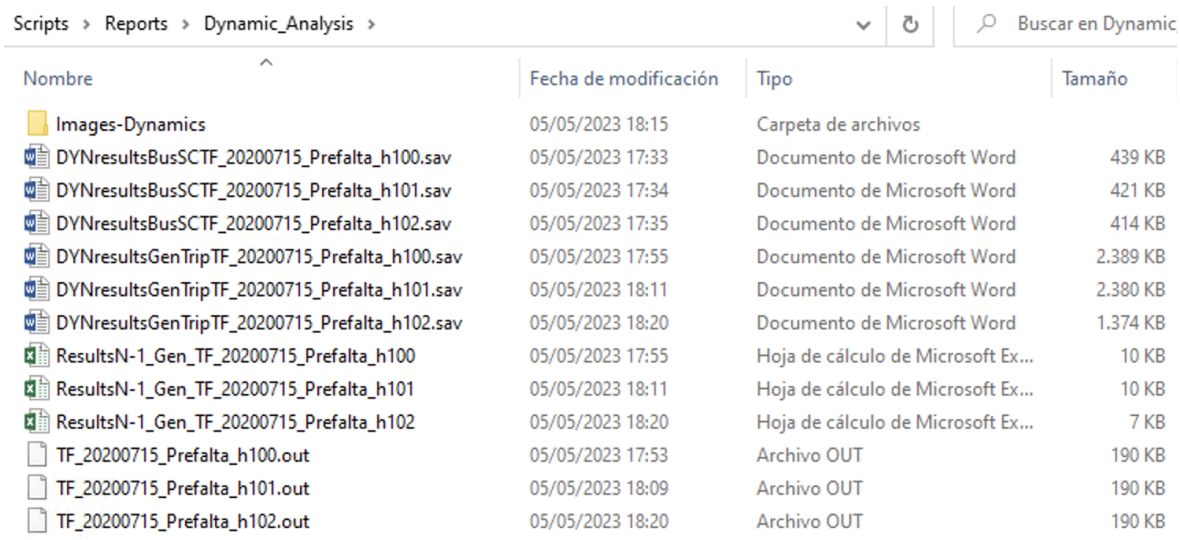


Figura 18 – Parte de los resultados exportados para los análisis estáticos

3.2 IMPORTACIÓN DE DATOS A PSSE

Esta sección se encarga de aclarar, en detalle, el funcionamiento de la primera de las tres partes de la herramienta desarrollada (*Analysis_Part1_Imports),* en la cual se importan a PSSE los datos proporcionados por el usuario y se crean los casos de estudio a evaluar.

A lo largo de esta sección, se va a ir mostrando el contenido del fichero anteriormente mencionado y de los sub-ficheros que va ejecutando a su vez, en los cuales se va resaltando en amarillo comandos de interés.

3.2.1 ANALYSIS_PART_1_IMPORTS

3.2.1.1 Analysis_Part_1_Imports – Ejecución de la función principal

```
var map;
# -*- coding: cp1252 -*-

# Program
# ----------------------------------------
def
RUN_ANALYSIS(str_aux_folder,str_generation_file,str_demand_file,str_excel_script_files,file_corre
spondence_gen,file_correspondence_dem,str_user_files,str_aux_case_folder,n_hours,slack_bus):

    #Imports all the necessary parameters to run the function
    import os,sys
    import matplotlib.pyplot as plt
```

```
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import collections
    #import dyntools

    #Imports the other scripts to be used as modules
    import Excel_Extraction
    import Python_To_PSSE

    #Creates the "Analysis_Data" folder if it does not exist.
    if not os.path.exists(str_aux_folder):
        os.makedirs(str_aux_folder)

    #Creates the "Excel_Files" folder if it does not exist.
    if not os.path.exists(str_excel_script_files):
        os.makedirs(str_excel_script_files)

    #Runs the function that generates the Generation dictionary
    Generators = Excel_Extraction.Gen_Excel_To_Dictionary(str_generation_file,
str_excel_script_files,file_correspondence_gen)
    #Runs the function that generates the Generation dictionary
    Demand =
Excel_Extraction.Dem_Excel_To_Dictionary(str_demand_file,str_excel_script_files,file_correspondence_gen)

    #Opens the PSSE-DESI CONVENTIONAL GENERATION Correspondence file
    xls_gen = pd.ExcelFile(file_correspondence_gen)
    datos_hoja_gen=xls_gen.parse('GeneracionConvencional')
    df_correspondencia_gen=datos_hoja_gen[["Nudo","Nombre","PGen (MW)","PMax (MW)","PMin
(MW)","QGen (Mvar)","QMax (Mvar)","QMin (Mvar)","Mbase (MVA)","Correspondencia
DESI","Comentario"]]

    #Opens the PSSE-DESI RES GENERATION/DEMAND Correspondence file
    xls_dem = pd.ExcelFile(file_correspondence_dem)
    datos_hoja_dem=xls_dem.parse('Demanda')
    datos_hoja_RES=xls_dem.parse('GeneracionRES')
    df_correspondencia_dem=datos_hoja_dem[["Nudo","Nombre","ID","Pload (MW)","% Demanda"]]
    df_correspondencia_RES=datos_hoja_RES[["Nudo","Nombre","ID","Pload (MW)","Tipo","% Demanda"]]

    #Creates the "Case_Data" folder if it does not exist.
    if not os.path.exists(str_aux_case_folder):
        os.makedirs(str_aux_case_folder)

    #Executes the function that introduces the data onto the Python files

Python_To_PSSE.Python_To_PSSE(n_hours,Generators,Demand,str_user_files,str_aux_case_folder,df_correspondencia_gen,df_correspondencia_dem,df_correspondencia_RES,slack_bus)
```

Esta primera sección se encarga, por orden, de realizar las siguientes rutinas:

- Incluir en la entrada de la función, variables introducidas por el usuario en el script.

- Importar todas las librerías y módulos necesarios para ejecutar la función, así como crear carpetas auxiliares en el caso de que no existan.
- Crear los diccionarios *Generators* y *Demand*, en los cuales se almacena toda la información de generación y demanda, ejecutando las funciones *Gen_Excel_To_Dictionary* y *Dem_Excel_To_Dictionary*, del fichero Python *Excel_Extraction.*
- Habiendo importado y abierto los ficheros Excel de correspondencia entre el despacho económico y el modelo PSSE, se ejecuta el fichero Python *Python_To_PSSE*

3.2.1.2 Analysis_Part1_Imports – Definición de parámetros por el usuario

Es importante remarcar, que los únicos ficheros Python que se van a ejecutar son, en orden, *ANEXO I: FICHERO PYTHON ANALYSIS_PART_1_IMPORTS, ANEXO 4: FICHERO PYTHON ANALYSIS_PART_2_OPF* y *ANEXO 5: FICHERO PYTHON ANALYSIS_PART_3_TESTS.* El resto de los ficheros Python se ejecutan internamente dentro de dichos ficheros principales.

En cada uno de los tres scripts principales (*Analysis_Part_1*, *Analysis_Part_2* y *Analysis_Part_3*), el usuario encontrará al final de cada uno, instrucciones para ejecutar cada uno de los ficheros y sus sub-módulos, así como parámetros y variables que debe definir el usuario para su correcto funcionamiento. Como se puede observar a continuación, para este primer fichero, el usuario debe de aportar datos como directorios de carpetas y archivos a utilizar, el bus *slack*, y el número de casos a evaluar (horas del día o de la semana):

```
# ----------------------------------------
# PARAMETERS TO BE DEFINED BY THE USER FOR THE EXCEL EXTRACTION FUNCTION AND FOR THE
PYTHON_TO_PSSE FUNCTION
# ----------------------------------------
    #THE INPUT FILES (.txt generation, correspondence file, .sav file) SHOULD BE INCLUDED ALL IN
THE SAME FOLDER
    #Introduce the .txtfile that contains all the generation data
    str_generation_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIgen.txt"
    #Introduce the .txtfile that contains all the demand data
    str_demand_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIisla.txt"
    #Introduce the folder where the this script´s auxiliary files are going to be located
    str_aux_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data"
    #Introduce the folder where the this script´s Excel auxiliary files are going to be located
```

```
    str_excel_script_files =
r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Excel_files"
    #Introduce the path of the file that has the correspondence between DESI and PSSE for
conventional generation (SAVED AS .XLS)
    file_correspondence_gen =
r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-DESI_convencional.xls"
    #Introduce the path of the file that has the correspondence between DESI and PSSE for demand
and RES generation (SAVED AS .XLS)
    file_correspondence_dem =
r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-DESI_dem_RES.xls"
    #Introduce the folder where the .sav files inputted by the user are located
    str_user_files = r"C:\Users\proyectista\Desktop\Scripts\User_data_input"
    #Introduce the path of the folder where the auxiliary case files are going to be created
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the number of the slack bus
    slack_bus = 284
    #Introduce the number of hours evaluated (Daily execution-24, Weekly execution-168)
    n_hours = 24

    ##    ####################################################
    ##    #FILE REQUIREMENTS FOR THE Excel_extraction and Python_To_PSSE:
    ##    #           1)The python scripts "Excel_Extraction" and "Python_To_PSSE" will be
located in the same folder as the ANALYSIS SCRIPT
    ##    #           2)The .sav file, PSSE-DESI correspondence files, .txt generation file and
.txt load file should all be in a same folder called "User_data_input"
    ##    #           3)A folder called "Excel_files" will contain the script, a xls with the
data and a .txt with the dictionary
    ##    #           4)The PSSE-DESI correspondence files must be a .xls file with the top left
cell ("Nudo 1") starting in A1, and the table must be alphabetically
    ##    #             ordered according to the "Correspondencia_DESI" column. It will have only
one sheet called "GeneracionConvencional"
    ##    #           5)The code is prepared to work with a generic case in which THE CODE NAMES
FOR REGULAR GENERATORS THE COMBINED CYCLES FOLLOW THE SAME NAME PATTERNS
    ##    #             AS TENERIFE AND THE PSSE-DESI CORRESPONDENCE FILE
    ##    #The output of this function are two dictionaries containing generation and load data
      ####################################################


    #Runs a complete analysis

RUN_ANALYSIS(str_aux_folder,str_generation_file,str_demand_file,str_excel_script_files,file_corre
spondence_gen,file_correspondence_dem,str_user_files,str_aux_case_folder,n_hours,slack_bus)
```

3.2.2 EXCEL_EXTRACTION

En esta sección, se describe en detalle el contenido del fichero Python *Excel_Extraction*, ejecutado en el interior del fichero *Analysis_Part_1_Imports,* y cuyo contenido completo se puede consultar en el *ANEXO 2: FICHERO PYTHON EXCEL_EXTRACTION.* Como se puede observar en el contenido de dicho sub-fichero , y en el de otros sub-ficheros, no existe ningún comando que se ejecute al ejecutar el propio fichero en el entorno de Python, sino que simplemente hay definidas funciones que se importan y ejecutan como módulos dentro de los ficheros principales.

Este fichero se encarga, por orden, de realizar las siguientes tareas:

- Para ambas funciones *Gen_Excel_To_Dictionary* y *Dem_Excel_To_Dictionary:*
  - Importar todos los módulos y librerías necesarios para la ejecución de las funciones.
  - Importar los ficheros de texto *ResDSIgen* (función *Gen*) y *ResDSIisla* (función *Dem*), y eliminar caracteres problemáticos que complican la importación de los datos a Excel.
  - Convertir los ficheros de texto anteriormente mencionados en ficheros Excel, para un manejo más fácil de los datos importados.
  - Creación de los diccionarios *Generators* y *Demand*, donde se guardan todos los datos relacionados con generación, generación renovable y demanda, los cuales se organizan en orden de llegada y poseen la siguiente jerarquía:
    - **Generators**: Nombre del generador → hora (1-168 en el caso de programación semanal) → variable → valor de la variable.
    - **Demand:** Hora (1-168 en el caso de programación semanal) → variable → valor de la variable.

3.2.3 PYTHON_TO_PSSE

En esta sección, se describe el detalle el contenido del fichero Python *Python_To_PSSE,* ejecutado en el interior del fichero *Analysis_Part_1_Imports,* y cuyo contenido completo se puede consultar en el *ANEXO 3: FICHERO PYTHON PYTHON_TO_PSSE*

Este fichero se encarga, por orden, de realizar las siguientes tareas:

- Importar todos los módulos y librerías necesarios para la ejecución de las funciones.
- Inicialización de PSSE
- Inicialización, para caso de estudio y para cada hora, de variantes del fichero .sav de los casos a estudiar.

- Apagado en el caso de PSSE de los generadores que se encuentran desconectados o sin despachar potencia en cada hora, según los datos del despacho económico (*ResDSIgen*).
- Encendido en el caso de PSSE de los generadores que se encuentran despachando potencia en cada hora, según los datos del despacho económico. El fichero también realiza el encendido del nudo asociado a cada generador, y de su correspondiente transformador MT/AT.
- Clasificado de los generadores del caso PSSE, según los datos del fichero de correspondencia entre el despacho económico y el modelo PSSE (*correspondenciaPSSE-DESI_convencional*), en generadores individuales o en generadores pertenecientes a ciclos combinados, para los cuales se calcula la potencia a despachar en el caso PSSE de la siguiente manera:
    - A los generadores individuales se les asigna directamente la generación obtenida del fichero del despacho económico (*ResDSIgen*).
    - A los ciclos combinados, puesto que en el despacho económico (*ResDSIgen*) vienen representados como un único generador con una única generación, pero en PSSE se le asigna a cada turbina un generador independiente, es necesario hacer el siguiente reparto de potencia en función de cuáles de las turbinas se encuentran en operación (dicha información es aportada por el despacho económico, y los ciclos combinados disponen de una turbina de gas y dos de vapor):
        - Cuando los ciclos combinados funcionan en modo simple (con una sola turbina en funcionamiento), se les asigna directamente la generación obtenida del fichero del despacho económico (*ResDSIgen*).
        - Cuando funciona en modo compuesto (turbina de gas + turbina de vapor, o 2 turbinas de gas + turbina de vapor), el reparto es el siguiente:

- Modo G+V (*Pmax* se refiere a la potencia instalada de cada turbina, *Pdesp* se refiere a la potencia obtenida del despacho económico):
  - $Pgen, gas = Pdesp * \frac{Pmax,g}{Pmax,g+0.5*Pmax,v}$
  - $Pgen, vapor = Pdesp * \frac{0.5*Pmax,v}{Pmax,g+0.5*Pmax,v}$
- Modo 2G+V (*Pmax* se refiere a la potencia instalada de cada turbina, *Pdesp* se refiere a la potencia obtenida del despacho económico):
  - $Pgen, gas\ (cada\ uno) = Pdesp * \frac{Pmax,g}{2*Pmax,g+Pmax,v}$
  - $Pgen, vapor = Pdesp * \frac{Pmax,v}{2*Pmax,g+Pmax,v}$

- Tanto para la generación renovable, como para la demanda, realiza las siguientes tareas:
  - Obtención del porcentaje que representan sobre el total la potencia activa y reactiva generada/consumida en cada caso de cada nudo de generación renovable o de demanda (los nudos de generación renovable están representados en PSSE como nudos de demanda negativa). Dichos datos se obtienen relacionando el fichero de correspondencia entre la generación renovable/demanda y el modelo PSSE (*correspondenciaPSSE-DESI_dem_res*) con la demanda y generación renovable totales obtenidas del despacho económico.
  - Escalado, para cada hora, de la potencia activa y reactiva de cada nudo, multiplicando la lectura total obtenida del fichero del despacho económico de demanda y generación renovable (*ResDSIisla*) por el porcentaje del total de potencia que representa cada nudo.
- Introducción de los datos calculados anteriormente de generación convencional, generación renovable y demanda en el modelo de PSSE.
- Para preparar la ejecución del flujo de cargas óptimo, se realizan las siguientes tareas en todos los elementos *shunt*:

- Si son elementos de potencia reactiva nominal fija, se convierten en elementos de potencia variable (por escalones) cuya máxima potencia es el valor de la potencia fija.
- Para nudos en los que hay elementos *shunt* de ambas potencia fija y potencia variable (de nuevo, por escalones), se eliminan los fijos y se añade su potencia a la potencia máxima de los variables.
- Se cambia el modo de dichos elementos al modo control de voltaje (ya seleccionado por la herramienta). Este modo representa la función de los elementos *shunt* en el OPF.

- Tras haber introducido todos los datos, y haber realizado los cambios pertinentes, se guarda el caso en un archivo .raw, listo para ser ejecutado por el OPF, en la versión 32 de PSSE.

3.3 EJECUCIÓN DEL FLUJO DE CARGAS ÓPTIMO (OPF)

Esta sección se encarga de aclarar, en detalle, el funcionamiento de la segunda de las tres partes de la herramienta desarrollada (*Analysis_Part_2_OPF),* en la cual se ejecuta el flujo de cargas óptimo (descrito con detalle en la *Sección 2.2*) sobre todos los casos de estudio creados tras la ejecución de los scripts descritos anteriormente, para la posterior ejecución de todos los análisis pertinentes. Este segundo paso se ejecuta a través de un único fichero Python, cuyo contenido se puede consultar con detalle en *ANEXO 4: FICHERO PYTHON ANALYSIS_PART_2_OPF*:

En concreto, el fichero se encarga de ejecutar la función *RunOPF*, encargada de realizar las siguientes tareas:

- Importar todos los módulos y librerías necesarios para la ejecución de las funciones.
- Inicialización de PSSE, en este caso la versión 32, compatible con Python 2.5, y la versión desde la cual se dispone de la herramienta para ejecutar el OPF.
- Apertura de los casos de estudio (semanal o diario por horas) y ejecución de un flujo de cargas para comprobar que los casos base creados converjan.

- Obtención de datos sobre todas las líneas, buses, elementos *shunt*, demandas, generadores y transformadores a evaluar en el sistema.
- Borrado de todos los posibles datos de OPF anteriores de los que pudiesen disponer los casos de estudio.
- Preparación de los elementos del sistema para una correcta ejecución del OPF:
    - Bloqueado de las tomas de los transformadores que no cumplan los siguientes requisitos, de manera que sólo se regularán las tomas de los transformadores que sí los cumplan:
        - Transformadores conectados a un nudo de generación (tipo 2)
        - Transformadores cuyo voltaje, tanto de alta como de baja, se encuentre dentro del rango de tensiones especificado por el usuario en la entrada de datos.
    - Definición de los límites de voltaje para los nudos, y los pesos que tendrán dichos límites en la función de optimización del OPF.
    - Definición de los límites para el ajuste de los elementos *shunt* para el control de voltaje, en bloques discretos que se pueden ir añadiendo o quitando automáticamente según sea necesario
    - Definición de límites sobre la carga que puede llevar cada línea, según lo especificado por el usuario en la entrada de datos (RATE A, RATE B, …)
    - Aunque actualmente no se aplica en los análisis, el fichero Python incluye la posibilidad de regular el uso de líneas con reactancia ajustable en el OPF, de manera que su uso se pueda incentivar o desincentivar en la función a optimizar.
    - De igual manera, se incluye la posibilidad de regular el uso de demandas con valor variable, de manera que su uso se pueda incentivar o desincentivar en la función a optimizar.
    - La modificación por el OPF de la generación de potencia activa despachada por los generadores no es necesaria, puesto que viene rígidamente especificada en el fichero proveniente del despacho económico. Asimismo, la generación de potencia reactiva por parte de los generadores tampoco es

necesario realizarla, puesto que esta vendrá mayoritariamente condicionada por los comandos de voltaje obtenidos tras ejecutar el flujo de cargas óptimo que minimice la variable a optimizar deseada, siempre manteniéndose dentro de los límites especificados en el propio caso base. De todas formas, se ha incluido en el fichero la capacidad de poder definir dicha regulación de potencia activa y reactiva, en el caso de que el fichero se quiera utilizar para fines distintos a los postulados en este proyecto de investigación.

- Definición de los parámetros a optimizar/minimizar en la ejecución del OPF (pérdidas de potencia, generación del nudo slack, conexiones de elementos *shunt*, …), así como otros parámetros relevantes al OPF (bloqueado de tomas de transformadores, imposición de límites de voltajes de emergencia en buses, fijado de elementos *shunt* variables)
- Ejecución del flujo de cargas óptimo y comprobación de convergencia del sistema. En caso de que no converja por debajo de las tolerancias especificadas, se refleja al final de la ejecución del script en una lista que, para cada caso, contiene un 1 si ha convergido caso y un 0 si no ha convergido.
- Creación del fichero PSSE (.sav) con los resultados del flujo de cargas óptimo a evaluar por los análisis estáticos y dinámicos.

A continuación, se muestra la parte del fichero Python donde el usuario debe de introducir los datos a evaluar, seguida de las instrucciones para el uso del script, y seguido de ejecución de la función principal *RunOPF*. En este caso, el usuario, aparte de directorios de archivos debe de incluir datos como límites de voltajes, límites de carga en las líneas, o los parámetros a minimizar/regular por el OPF:

```
    # ----------------------------------------
    # PARAMETERS TO BE DEFINED BY THE USER
    # ----------------------------------------

    #Introduce the KV range you want to evaluate. IN PART 2 INCLUDE ALL OF THE DIFFERENT KV IN THE
SYSTEM
    v_KVRANGE = [1,400]
    #Introduce the KV range for the transformer tap regulation
    WINDING_TAP_REGULATION_KV_RANGE = [15,400]
    #Introduce the load warning threshold as a percentage of the load limit
    loadwarning = 90
    #Introduce voltage upper and lower limits in p.u
    voltupperlimit = 1.05
```

```
   voltlowerlimit = 0.95
   #Introduce the rate to be used for load analysis of the branches (e.g: 'RATEA', 'RATEB')
   BRANCH_RATE = 'RATEA'
   #Introduce the PSSE version (32 for OPF)
   ISPSSEVERSION=32
   #Introduce the location of PSSE (PSSE32 in this case)
   PSSE_LOCATION = r"C:\Program Files (x86)\PTI\PSSE32\PSSBIN"
   #Introduce the path of the folder where the files to be evaluated are located
   strpathfolder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"

   #The OPF tolerances and control are the default suggested by PSSE
   #Introduce the parameters to minimize in the OPF (insert a 1 in the Parameters list if that
parameter should be minimized/regulated, a 0 if not).
      #1)Minimize active power slack generation
      #2)Minimize reactive power slack generation
      #3)Minimize active power losses
      #4)Minimize reactive power losses
      #5)Minimize adjustable branch reactances
      #6)Minimize adjustable bus shunts
      #7)Minimize adjustable bus loads
      #8)Minimize interface flows
      #9)Minimize reactive generation reserve
      #10)Regulate area interchange
      #11)Constrain interface flows
      #12)Use automatic scaling
      #13)Use dual variable convergence criteria
      #14)Fix transformer tap ratios
      #15)Fix transformer phase shift angles
      #16)Fix switched shunts
      #17)Round transformer tap ratios
      #18)Round switched shunt vars
      #19)Automatically adjust bus voltages for feasibility
      #20)Impose emergency bus voltage limits
      #21)Impose emergency branch flow limits
   Parameters = [1,0,1,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,1,1]

   ######################################################
   #FILE REQUIREMENTS FOR THIS SCRIPT:
   #           1)The python script will have to be located in the "Scripts" folder. THIS SCRIPT
IS EXECUTED AFTER THE ANALYSIS_PART_1 script
   #           2)The .raw file(s) generated in the ANALYSIS_PART_1 will need to be run in PSSE32
(the file can be saved in that version)
   #           3)The OPF results for each hour are exported onto a .sav file (PSSE32, can be
opened by 34) in the folder where the scripts are
   #           4)A file called OPF_Parameter_Results is generated. It can be ignored
   #The .sav files are generated on the same folder as the .raw files, and are ready to be
analyzed by ANALYSIS_PART_3
   ######################################################

   convergence =
RunOPF(strpathfolder,strfilesbs,r"OPFresults.dat",v_KVRANGE,WINDING_TAP_REGULATION_KV_RANGE,loadw
arning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,Parameters,BRANCH_RATE)
   print(convergence)
```

3.4 EJECUCIÓN DE ANÁLISIS

Esta sección se encarga de aclarar, en detalle, el funcionamiento de la tercera de las tres partes de la herramienta desarrollada (*Analysis_Part_3_TESTS),* en la cual se ejecutan los posteriores análisis estáticos y dinámicos de contingencias sobre todos los casos de estudio creados tras la ejecución del flujo de cargas óptimo OPF. Este tercer paso se ejecuta a través de un fichero Python principal, cuyo contenido se puede consultar con detalle en *ANEXO 5: FICHERO PYTHON ANALYSIS_PART_3_TESTS*. Dicho fichero Python llama, a su vez, a subsecuentes funciones que ejecutan los análisis que se describirán posteriormente en esta sección. El contenido del fichero *Analysis_Part_3_TESTS* se divide en la función principal a ejecutar, y en la entrada de datos que se debe definir por el usuario:

```
# ----------------------------------------
# Program
# ----------------------------------------
def
RUN_ANALYSIS(str_report_folder,str_report_folder_static,str_aux_case_folder,file_correspondence,str_user_files,n
_hours,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,BRANCH_RATE,strdynfile,FB
ASE,tfin_gen,tfin_bus,FUF1,str_aux_dyn_folder,dynamic_gen_parameters,str_report_folder_dynamic,dynamic_bus_param
eters,sc_buses,max_angle,min_angle,max_angle_ratio):
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    #from docx.text.parargaph import Paragraph
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import collections
    #import dyntools

    #Imports the other scripts
    import StaticAnalysisLibrary
    import MainSimulateTrippings

    #Opens the PSSE-DESI Correspondence file
    xls = pd.ExcelFile(file_correspondence)
    datos_hoja=xls.parse('GeneracionConvencional')
    df_correspondencia=datos_hoja[["Nudo","Nombre","PGen (MW)","PMax (MW)","PMin (MW)","QGen (Mvar)","QMax
(Mvar)","QMin (Mvar)","Mbase (MVA)","Correspondencia DESI","Comentario"]]

    #Creates the  reports folder if it does not exist.
    if not os.path.exists(str_report_folder):
        os.makedirs(str_report_folder)

    #Creates the static reports folder if it does not exist.
    if not os.path.exists(str_report_folder_static):
        os.makedirs(str_report_folder_static)

    #Runs the Static Analytic Library Analysis script (Contingency analysis)
    StaticAnalysisLibrary.RunLF(str_report_folder_static,str_aux_case_folder,v_KVRANGE, loadwarning,
voltupperlimit, voltlowerlimit, ISPSSEVERSION, PSSE_LOCATION,BRANCH_RATE)
```

```
    StaticAnalysisLibrary.RunStaticN_1N_2branches(str_report_folder_static,str_aux_case_folder,v_KVRANGE,
loadwarning, voltupperlimit, voltlowerlimit, ISPSSEVERSION, PSSE_LOCATION,BRANCH_RATE)

    #Creates the dynamic reports folder if it does not exist.
    if not os.path.exists(str_report_folder_dynamic):
        os.makedirs(str_report_folder_dynamic)

    #Runs the dynamics simulations (Generator Trippings and Short-Circuits)
    MainSimulateTrippings.RunDynamicGenTrippings(strdynfile,FBASE,tfin_gen,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_gen_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE,max_angle,min_angle,max_angle_ratio)
    MainSimulateTrippings.RunDynamicShortCircuits(strdynfile,FBASE,tfin_bus,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_bus_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE,sc_buses,max_angle,min_angle,max_angle_ratio)
```

Como se ha podido observar anteriormente, la función principal se encarga de importar todas las librerías necesarias para la ejecución de las funciones, importar los ficheros auxiliares para los análisis (*StaticAnalysisLibrary,* la cual se puede acceder en *ANEXO 6: FICHERO PYTHON STATIC_ANALYSIS_LIBRARY* y *MainSimulateTrippings*, la cual se puede acceder en *ANEXO 7: FICHERO PYTHON MAIN_SIMULATE_TRIPPINGS*), y ejecutar las siguientes funciones que se encuentran en su interior:

- ***StaticAnalysisLibrary****.RunLf*: Ejecución de un flujo de cargas que obtenga los resultados del flujo de cargas óptimo.
- ***StaticAnalysisLibrary****.RunStaticN-1N-2branches*: Ejecución de análisis estáticos de contingencias sobre líneas y transformadores.
- ***MainSimulateTrippings****.RunDynamicGenTrippings*: Ejecución de análisis dinámicos y en régimen permanente de pérdidas de generación bajo regulación primaria.
- ***MainSimulateTrippings****.RunDynamicShortCircuits*: Ejecución de análisis dinámicos de cortocircuitos en nudos selectos por el usuario.

Asimismo, en la otra mitad del fichero principal se proporciona la definición de parámetros que el usuario debe de realizar para la correcta ejecución de los análisis:

```
if __name__=="__main__":

   # The main runs the functions defined above.

    import os, sys

# ----------------------------------------
# PARAMETERS TO BE DEFINED BY THE USER FOR THE STATIC ANALYSIS FUNCTION
# ----------------------------------------
    #THE INPUT FILES (.txt generation, correspondence file, .sav file) SHOULD BE INCLUDED ALL IN THE SAME FOLDER
    #Introduce the path of the file that has the correspondence between DESI and PSSE (SAVED AS .XLS)
    file_correspondence = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-
DESI_convencional.xls"
```

```
    #Introduce the folder where the .sav files inputted by the user are located
    str_user_files = r"C:\Users\proyectista\Desktop\Scripts\User_data_input"
    #Introduce the path of the folder where the .sav files to be evaluated are localized
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the path of the folder of the "Reports" folder
    str_report_folder = r"C:\Users\proyectista\Desktop\Scripts\Reports"
    #Introduce the path of the folder where the reports are going to be generated
    str_report_folder_static = r"C:\Users\proyectista\Desktop\Scripts\Reports\Static_Analysis"
    #Introduce the number of hours evaluated (Daily execution-24, Weekly execution-168)
    n_hours = 24
    #Introduce the KV range you want to evaluate.
    #Since transformer loads are evaluated, please specify a range in which no transformers have a bus out of
range and a bus inside range, an error will occur
    #Generator trippings are evaluated in the contingency analysis, please make sure that all generators' buses
are included in the kV range
    v_KVRANGE = [1,500]
    #Introduce the load warning threshold as a percentage of the load limit
    loadwarning = 90
    #Introduce the rate to be used for load analysis of the branches (e.g: 'RATEA', 'RATEB',...)
    BRANCH_RATE = 'RATEA'
    #Introduce the voltage upper and lower limits in p.u
    voltupperlimit = 1.10
    voltlowerlimit = 0.95
    #Introduce the version of PSSE and where it is located on your computer
    ISPSSEVERSION=34
    PSSE_LOCATION = r"C:\Program Files (x86)\PTI\PSSE34\PSSPY27"

   ######################################################
   #FILE REQUIREMENTS FOR THIS SCRIPT:
   #           1)The python script and the .sav file(s) have to be placed at the same folder level
   #           2)A folder called "Images-Static" will be created at the same level as the report folder
   #           3)Due to excel sheet requirements, names of the .sav files MUST be kept short
   #The output Word and Excel files (for each hour one Word and Excel for each analysis (GenTrip, BusSC) will be
generated in a folder inside the "Reports" folder called "Static_Analysis"
   #If an excel file for one hour is NOT GENERATED, it means that there is NO VIOLATIONS for any contingency at
that hour
   ######################################################

    # ---------------------------------------
    # PARAMETERS TO BE DEFINED BY THE USER FOR THE DYNAMIC ANALYSIS FUNCTION
    # ---------------------------------------
    #Choose the path of the dynamic file (.dyr) to be evaluated
    strdynfile =
r"C:\Users\proyectista\Desktop\Scripts\User_data_input\Tenerife_DinamicoEndesa_Incidente_Inicial.dyr"
    #Introduce the path of the folder where the .sav files to be evaluated are localized
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the path of the folder where the auxiliary "Dynamic_Files" folder will be created
    str_aux_dyn_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Dynamic_Files"
    #Introduce the path of the folder where the reports are going to be generated
    str_report_folder_dynamic = r"C:\Users\proyectista\Desktop\Scripts\Reports\Dynamic_Analysis"
    #Choose base frequency (Hz)
    FBASE = 50.0
    #The tripping of the generator is done at t=1s, choose the time at which the simulation ends
    tfin_gen = 50.0
    #The short-circuit at the bus is generated at t=1s, choose the time at which the simulation ends
    tfin_bus = 10.0
    #Choose the frequency warning threshold for generator trippings
    FUF1 = 48.5
    #Choose the positive generator angle warning threshold for short circuits
    max_angle = 50
    #Choose the negative generator angle warning threshold for short circuits
    min_angle = -50
    #Choose the warning treshold for the relationship between the peak and steady-state value of the maximum and
minimum generation angles in the system
    #Max_angle_ratio = max or min_dynamic_angle / steady-state max or min angle
    max_angle_ratio = 1.5
    #Include in the following list the bus numbers in which the short circuits will be triggered
    sc_buses = [11,16,419]
    #Select the parameters to plot in the dynamic generator tripping analysis in the following list (1 yes, 0
no)
    #1)Plot Electrical Power for each generator
    #2)Plot Mechanical Power for each generator
```

```
    #3)System frequency. The data will be extracted from the speed of one of the generators. Recommended to
include it
    #4)Plot System totals (PELEC, PMECH, PACCL, PLOAD, PE-PL)
    dynamic_gen_parameters = [1,1,1,1]
    #Select the parameters to plot in the dynamic bus tripping analysis in the following list (1 yes, 0 no)
    #1)Plot Voltage angle for each generator
    #2)Plot Voltage (p.u) for each generator
    #3)System frequency. The data will be extracted from the speed of one of the generators. Recommended to
include it
    #4)MANDATORY FOR IT TO BE ACTIVATED. Evaluate machine angle statistics (Average angle, largest angle, bus
with largest angle, smallest angle, bus with smallest angle, angle spread)
    dynamic_bus_parameters = [1,1,1,1]
    #Please, keep in mind that if other different variables want to be exported, the function GetDynResults will
have to be slightly modified to accomodate the new variables
####################################################
    #FILE REQUIREMENTS FOR THIS SCRIPT:
    #           1)The .sav and .dyr files will need to be placed into the "User_data_input" folder
    #           2)A folder called "Images-Dynamics" will be created inside the "Reports" folder in order to
accomodate all the images
    #           3)A folder called "Dynamic_Files" will be created in the Scripts-->Analysis_Data folder in order
to accomodate some dynamics files
    #The output Word and Excel documents, among othe PSSE output files will be generated inside the "Reports"
folder in a folder called "Dynamic_Analysis"
    ####################################################

    #Runs a complete analysis

RUN_ANALYSIS(str_report_folder,str_report_folder_static,str_aux_case_folder,file_correspondence,str_user_files,n
_hours,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,BRANCH_RATE,strdynfile,FB
ASE,tfin_gen,tfin_bus,FUF1,str_aux_dyn_folder,dynamic_gen_parameters,str_report_folder_dynamic,dynamic_bus_param
eters,sc_buses)
```

El código presentado anteriormente se encarga de la definición de parámetros para la ejecución de cada uno de los módulos de análisis a ejecutar, como la introducción de directorios de ficheros, definición de límites o selección de parámetros a extraer de los análisis, así como la presentación de instrucciones a seguir por el usuario.

3.4.1 EJECUCIÓN DEL FLUJO DE CARGAS SOBRE EL SISTEMA OBTENIDO DEL FLUJO DE CARGAS ÓPTIMO (OPF)

Esta sección se encarga de aclarar, en detalle el contenido de la función *RunLF*, del fichero Python *StaticAnalysisLibrary,* accesible en *ANEXO 6: FICHERO PYTHON STATIC_ANALYSIS_LIBRARY*. Dicha función se encarga de extraer los resultados del flujo de cargas óptimo mediante los siguientes pasos:

- Importación de todas las librerías necesarias para la ejecución de la función como, por ejemplo, *pandas* para exportar datos a Excel, o *matplotlib* para la creación de gráficas.

- Definición de los parámetros del flujo de cargas a ejecutar. Puesto que el flujo de cargas óptimo ya se ha ejecutado en *Analysis_Part_2_Imports*, simplemente se quiere comprobar de nuevo la convergencia del sistema para extraer datos, y se bloquean las tomas de transformadores, la modificación de conexiones de elementos *shunt*, o las tensiones de consigna en los nudos.
- Apertura de PSSE
- Para cada uno de los casos de estudio de PSSE (por horas) obtenidos tras la ejecución del flujo de cargas óptimo, se ejecuta el flujo de cargas con los parámetros definidos anteriormente, y se comprueba su convergencia. En el caso de que el sistema converja, se puede continuar con el análisis.
- Extracción del voltaje resultante en cada nudo, y de la carga en cada línea. Creación de unas listas con los datos obtenidos para su posterior procesamiento y añadido en los documentos Excel y Word de salida.
- Creación de los documentos Excel y Word para exportar los resultados. En este caso, se exporta un único documento Word con los resultados de todas las horas evaluadas, y un único documento Excel, en el cual cada pestaña representa los resultados de cada hora evaluada.
- Ejecución de la función *GetLFResults,* la cual se describirá en la *Sección 3.4.2.1*, encargada de crear los *dataframes* (bases de datos) que se incluirán en los documentos de salida, así como la creación de figuras y gráficas como alternativas para la visualización de datos.

3.4.2 EJECUCIÓN DE ANÁLISIS ESTÁTICOS DE CONTINGENCIAS N-1 N-2

Esta sección se encarga de aclarar, en detalle el contenido de la función *RunStaticN-1N-2branches*, del fichero Python *StaticAnalysisLibrary,* accesible en *ANEXO 6: FICHERO PYTHON STATIC_ANALYSIS_LIBRARY*. Dicha función se encarga de ejecutar los análisis de contingencias N-1 (propiciado de una única contingencia en la red) y N-2 (propiciado de dos contingencias a la vez en la red), tras los cuales se extraen resultados en régimen permanente.

En el caso del análisis N-1, para cada caso de estudio evaluado, se realizan y evalúan las desconexiones individuales de todas las líneas y transformadores del sistema. En el caso del análisis N-2 y, aunque en el caso de Tenerife, debido a su baja resiliencia no se ha decidido aplicar, se realizan y evalúan las desconexiones de dos en dos de todas las líneas y transformadores paralelos (que empiezan y acaban en el mismo nudo) del sistema.

El proceso que sigue esta función es similar al seguido para la ejecución de la función *RunLF*:

- Importación de todas las librerías necesarias para la ejecución de la función.
- Definición de los parámetros del flujo de cargas a ejecutar tras la contingencia.
- Apertura de PSSE
- Para cada uno de los casos de estudio de PSSE (por horas) obtenidos tras la ejecución del flujo de cargas óptimo, se ejecuta un primer flujo de cargas previo a la contingencia con los parámetros definidos anteriormente, y se comprueba su convergencia. En el caso de que el sistema converja, se puede continuar con el análisis.
- Creación de listas con todos los buses y líneas del sistema para su posterior tratado.
- Para todos los casos a estudiar, ejecución de todas las posibles contingencias (apagado de una línea o de dos líneas), y comprobación de la convergencia del caso. En el caso de que el sistema no converja, ya sea por un colapso de la red, o por el apagado de líneas cuyo funcionamiento es esencial para no aislar nudos o zonas enteras, se mostrará un mensaje en el documento de resultados que lo refleje.
- Extracción del voltaje resultante en cada nudo, y de la carga en cada línea, para su posterior evaluación.
- En el análisis de contingencias sólo se exportarán datos sobre casos en los que ocurran voltajes fuera de rango en nudos, o sobrecargas en líneas. De esta manera el fichero se encarga de aislar y procesar únicamente dichos casos

- Ejecución de la función *GetLFResults,* encargada de crear los *dataframes* (bases de datos) de las contingencias que han causado violaciones de los límites de tensión o carga. Dichos *dataframes* se incluirán en los documentos de salida.
- Creación de los documentos Excel y Word para exportar los resultados. En este caso, se exporta un único documento Word con los resultados de todos los casos y todas las contingencias evaluadas (en el caso de que exista alguna violación de límites), y un documento Excel por cada caso evaluado, en el que cada pestaña representa los resultados de una contingencia aplicada tras la cual, de nuevo, se ha producido una violación de límites.

3.4.2.1 Función secundaria GetLFResults de apoyo a los análisis estáticos

Esta sección se encarga de aclarar el contenido de la función *GetLFResults,* del fichero Python *StaticAnalysisLibrary,* accesible en *ANEXO 6: FICHERO PYTHON STATIC_ANALYSIS_LIBRARY.* Dicha función sirve de apoyo para la obtención de los resultados del flujo de cargas óptimo, y de los análisis estáticos de contingencias.

Dicha función, mediante la entrada de datos provenientes tanto del flujo de cargas óptimo, como de los análisis de contingencias (voltajes en nudos y cargas en líneas) crea gráficas comparativas, por niveles de tensión, de los voltajes y cargas obtenidos, así como tablas con dichos datos, mediante la librería de Python *pandas*, que permiten su pegado rápido en Word y Excel. El contenido de dichos resultados se presentará con detalle en el *Capítulo 4.*

3.4.3 EJECUCIÓN DE ANÁLISIS DINÁMICOS DE PÉRDIDAS DE GENERACIÓN

Esta sección se encarga de aclarar, en detalle, el contenido de la función *RunDynamicGenTrippings,* del fichero Python *MainSimulateTrippings,* accesible en *ANEXO 7: FICHERO PYTHON MAIN_SIMULATE_TRIPPINGS.* Dicha función se encarga de realizar el análisis dinámico de pérdidas de generación en cada caso de estudio, tras el cual se extraen gráficas y datos sobre el comportamiento dinámico del sistema durante la actuación de la regulación primaria, así como la extracción de resultados en régimen permanente.

El proceso que sigue esta función es el siguiente:

- Importación de todos los ficheros necesarios para la ejecución de la función
- Definición de parámetros y creación de archivos necesarios para el funcionamiento interno del análisis dinámico
- Para cada caso de estudio evaluado (cada hora), se crean documentos Word y Excel donde se van a exportar los resultados, y se inicializa PSSE.
- Obtención de datos de los elementos del sistema, para su monitorización durante la ejecución del análisis dinámico.
- Para cada caso de estudio evaluado, ejecución de todos los posibles análisis de pérdidas de generación (de un generador cada vez, sólo sobre generadores activos):
    - Ejecución de un flujo de cargas inicial y comprobado de su convergencia. En el caso de que converja, se puede proceder con el análisis.
    - Resolución de las ecuaciones del sistema, como se comentó en la *sección 2.4*, para la creación de ficheros dinámicos auxiliares.
    - Desconexión del generador, y recopilado de simulaciones dinámicas sobre parámetros a elegir por el usuario, como las potencias mecánica y eléctrica de cada generador, o la frecuencia del sistema.
    - Ejecución de la función *GetDYNResults*, mediante la cual se extraen a un documento Word gráficas con los resultados de las simulaciones dinámicas tras la pérdida de cada generador. El contenido de dicha función se comenta con detalle en la *sección 3.4.4.1.* Se crea un documento Word por cada caso de estudio, en el que se incluyen los resultados de las simulaciones para todos los generadores evaluados.
    - Ejecución de la función auxiliar *GetStaticN_1GenTrippings* (perteneciente también al fichero *MainSimulateTrippings*), mediante la cual, de la misma manera en la que se extraían datos tras el flujo de cargas óptimo y los análisis de contingencias, se extraen a un documento Excel los datos en régimen permanente de voltajes en nudos y cargas en líneas tras la pérdida del

generador. Se crea un documento Excel caso de estudio, en el que cada pestaña corresponde a los resultados de la pérdida de un generador.

3.4.4 EJECUCIÓN DE ANÁLISIS DINÁMICOS DE CORTOCIRCUITOS EN NUDOS SELECTOS

Esta sección se encarga de aclarar, en detalle, el contenido de la función *RunDynamicShortCircuits,* del fichero Python *MainSimulateTrippings,* accesible en *ANEXO 7: FICHERO PYTHON MAIN_SIMULATE_TRIPPINGS.* Dicha función se encarga de realizar el análisis dinámico de breves cortocircuitos en nudo en cada caso de estudio, tras el cual se extraen gráficas y datos sobre el comportamiento dinámico del sistema.

De una manera similar al que sigue la función *RunDynamicGenTrippings*, el proceso a seguir por esta función es el siguiente:

- Importación de todos los ficheros necesarios para la ejecución de la función
- Definición de parámetros y creación de archivos necesarios para el funcionamiento interno del análisis dinámico
- Para cada caso de estudio evaluado (cada hora), se crean documentos Word donde se van a exportar los resultados, y se inicializa PSSE.
- Obtención de datos de los elementos del sistema, para su monitorización durante la ejecución del análisis dinámico.
- Para cada caso de estudio evaluado, ejecución de análisis de cortocircuitos en todos los nudos selectos por el usuario:
    - Ejecución de un flujo de cargas inicial y comprobado de su convergencia. En el caso de que converja, se puede proceder con el análisis.
    - Resolución de las ecuaciones del sistema, como se comentó en la *sección 2.5*, para la creación de ficheros dinámicos auxiliares.
    - Ejecución de un cortocircuito (fase-neutro en este caso) en el nudo durante 100ms, y recopilado de simulaciones dinámicos sobre parámetros a elegir por

el usuario, como voltajes y ángulos en cada nudo de generación, o la frecuencia del sistema.

- Ejecución de la función *GetDYNResults*, mediante la cual se extraen a un documento Word gráficas con los resultados de las simulaciones dinámicas tras el cortocircuito cuasi instantáneo en el nudo. Se crea un documento Word por cada caso de estudio, en el que se incluyen los resultados para todos los nudos seleccionados por el usuario.

3.4.4.1 Función secundaria GetDYNResults de apoyo a los análisis dinámicos

Esta sección se encarga de aclarar el contenido de la función *GetDYNResults,* del fichero Python *MainSimulateTrippings,* accesible en *ANEXO 7: FICHERO PYTHON MAIN_SIMULATE_TRIPPINGS* .Dicha función sirve de apoyo para la obtención de los resultados de los análisis dinámicos de pérdidas de generación y de cortocircuitos en nudos.

De esta manera, esta función recopila la variación temporal de cada parámetro del sistema estudiado, y lo plasma en gráficas, mediante la herramienta de Python *matplotlib*, que posteriormente se guardan en el documento Word en el que se extraen los resultados. El contenido de dichos resultados se presentará con detalle en el *Capítulo 4.*

Capítulo 4. OBTENCIÓN DE RESULTADOS

La finalidad de este capítulo es mostrar ejemplos de los resultados que exporta Python tras los análisis. La salida de datos está estructurada en resultados de análisis estáticos y dinámicos, dentro de los cuales se aportan los diversos ejemplos y formatos utilizados.

Es importante remarcar que el objetivo principal de este proyecto no es el análisis detallado sobre la isla de Tenerife, ni la resolución de sus problemas eléctricos, sino el desarrollo de la herramienta propuesta. En esta sección, se muestran resultados obtenidos del análisis sobre la isla para mostrar el funcionamiento de la herramienta, acompañado de breves comentarios y observaciones.

4.1 CASO DE ESTUDIO - TENERIFE

El caso estudiado, en el que se ha evaluado la efectividad del código, se trata de la isla de Tenerife, a través de un fichero PSSE que contiene toda su red de alta tensión. Al tratarse de una isla, Tenerife dispone de una red eléctrica más débil que la de un sistema como uno peninsular, más grande e interconectado con países vecinos. Como se ha podido observar, la desconexión de ciertos elementos del sistema es suficiente para propiciar problemas, hasta el punto en el que el caso presenta una alta estabilidad. Además, Tenerife es una isla en la que se está llevando a cabo una gran cantidad de inversiones en energías renovables, sobre todo solar fotovoltaica, que suponen una generación más dispersa y difícil de controlar.

La entrada de datos al algoritmo de Python se ha realizado a través de unos ficheros de texto provenientes del despacho económico, los cuales contienen todos los datos, por horas (durante un día o durante una semana), de la generación convencional, generación renovable y demanda de cada nudo, como se ha comentado en la *sección 3.2*.

El objetivo final es, para cada uno de los distintos despachos económicos durante el día, obtener el punto de operación óptimo de la red en términos de tensiones de consigna de

generadores, conexiones de elementos *shunt*, y posiciones de las tomas de los transformadores, así como exportar resultados del análisis de contingencias y de análisis dinámicos, para poder corregirlos.

4.2 RESULTADOS DE LOS ANÁLISIS ESTÁTICOS

ResultsN-1N-2TF_20200715_Prefalta_h142	02/05/2023 23:33	Hoja de cálculo d...	139 KB
ResultsN-1N-2TF_20200715_Prefalta_h143	02/05/2023 23:42	Hoja de cálculo d...	135 KB
ResultsN-1N-2TF_20200715_Prefalta_h144	02/05/2023 23:42	Hoja de cálculo d...	13 KB
ResultsN-1N-2TF_20200715_Prefalta_h145	02/05/2023 23:43	Hoja de cálculo d...	9 KB
ResultsN-1N-2TF_20200715_Prefalta_h146	02/05/2023 23:47	Hoja de cálculo d...	57 KB
ResultsN-1N-2TF_20200715_Prefalta_h147	02/05/2023 23:51	Hoja de cálculo d...	63 KB
ResultsN-1N-2TF_20200715_Prefalta_h148	02/05/2023 23:56	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h149	03/05/2023 0:00	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h150	03/05/2023 0:05	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h151	03/05/2023 0:10	Hoja de cálculo d...	63 KB
ResultsN-1N-2TF_20200715_Prefalta_h152	03/05/2023 0:10	Hoja de cálculo d...	7 KB
ResultsN-1N-2TF_20200715_Prefalta_h153	03/05/2023 0:10	Hoja de cálculo d...	8 KB
ResultsN-1N-2TF_20200715_Prefalta_h154	03/05/2023 0:10	Hoja de cálculo d...	13 KB
ResultsN-1N-2TF_20200715_Prefalta_h155	03/05/2023 0:15	Hoja de cálculo d...	61 KB
ResultsN-1N-2TF_20200715_Prefalta_h156	03/05/2023 0:19	Hoja de cálculo d...	67 KB
ResultsN-1N-2TF_20200715_Prefalta_h157	03/05/2023 0:23	Hoja de cálculo d...	64 KB
ResultsN-1N-2TF_20200715_Prefalta_h158	03/05/2023 0:25	Hoja de cálculo d...	45 KB
ResultsOPF	02/05/2023 20:44	Documento de Mi...	473 KB
ResultsOPF	02/05/2023 20:44	Hoja de cálculo d...	652 KB

Figura 19 – Parte del contenido de la carpeta Static_Analyisis en el que se observan los archivos de resultados del OPF y del análisis de contingencias de las horas 142 a 158

4.2.1 FLUJOS DE CARGAS ÓPTIMO

Los primeros datos de interés para el usuario son los obtenidos tras la ejecución del flujo de cargas óptimo (OPF), antes de ejecutar los análisis de contingencias pertinentes. Puesto que la finalidad del OPF, aparte de optimizar alguna variable de interés, es configurar la red para que su funcionamiento se encuentre dentro de límites, se espera que el perfil de tensiones y las cargas en cada línea se encuentren dentro de las consignas establecidas. Cabe destacar, que puede que existan casos en los que el OPF no pueda converger el caso con los requisitos exigidos, y necesitará relajar los límites de tensiones de algunos nudos.

Se exportan tanto a un documento Word (un único documento para todas las horas), como a un documento Excel (un único documento para todas las horas, una hoja por variable y hora) tablas con los voltajes en cada nudo, y las cargas en cada línea. Obsérvese a continuación en una tabla de resultados exportados, cómo el OPF realiza leves cambios al sistema para mantenerlo dentro del rango de estabilidad (carga menor al 100% en las líneas, y tensiones dentro del rango 0.95-1.1 p.u), mientras maximiza la función objetivo (minimizar pérdidas en este caso):

BusFROM	BusTO	RATE	LOAD-MVA	LOAD% OPF	LOAD% NO OPF
7	16	89.0	27.415	30.803	30,823
7	16	89.0	27.357	30.738	30,758
7	32	67.0	16.266	24.277	23,404
7	46	67.0	22.472	33.541	33,634
7	46	67.0	22.443	33.496	33,59
7	112	45.0	0.0	0.0	0,015
7	297	144.0	3.726	25.872	25,854
7	323	67.0	32.716	48.83	48,941
7	323	67.0	32.556	48.591	48,703
7	467	125.0	51.844	41.475	41,4
7	468	125.0	51.213	40.97	40,896
7	469	125.0	51.846	41.477	41,708
11	54	290.0	94.619	32.627	32,581
11	419	290.0	61.18	21.097	20,974
11	467	125.0	52.389	41.911	41,664
11	468	125.0	51.751	41.401	41,157
11	469	125.0	51.872	41.498	41,559
16	209	58.0	13.476	23.235	23,361

Tabla 1 – Cargas en algunas líneas del sistema tras la ejecución del OPF en la hora 100 del despacho económico

Bus	Voltage(p.u) OPF	Voltage(p.u) NO OPF

7	1,0242	1,0283
11	1,0045	1,0043
16	1,0173	1,0226
30	1,0154	1,0317
31	1,0243	1,029
32	1,0221	1,0267
37	1,044	1,0519
38	1,0565	1,073
39	1,0119	1,0194
44	1,06	1,0663
45	1,0655	1,0717
46	1,0145	1,0205
52	1,0316	1,0323
54	1,0122	1,0099
69	1,0306	1,0307
87	1,0443	1,0505
99	1,0433	1,0491
100	1,0429	1,0486
101	1,0151	1,0206
112	1,0242	1,0283
131	1,0989	1,0059

Tabla 2 – Tensión en algunos nudos del sistema tras la ejecución del OPF en la hora 100 del despacho económico

Los ficheros de Python desarrollados ofrecen la posibilidad de exportar los datos observados en la *Tabla 1* y la *Tabla 2* en gráficas ordenadas por niveles de tensión (generalmente MT y AT). Debido a la gran cantidad de información a tratar y de casos a evaluar durante el análisis, crear dichas gráficas consume una gran cantidad de tiempo y de memoria, por lo que, generalmente, dicha funcionalidad está desactivada. A continuación, se muestran ejemplos de las gráficas obtenidas de modo ilustrativo, las cuales indicarían también cuándo una medida se encuentra fuera del rango establecido. Obsérvese que las gráficas corresponden a otro caso de estudio, en el que **no se ha ejecutado el OPF**, de manera que se pueden observar violaciones de límites previas al OPF:

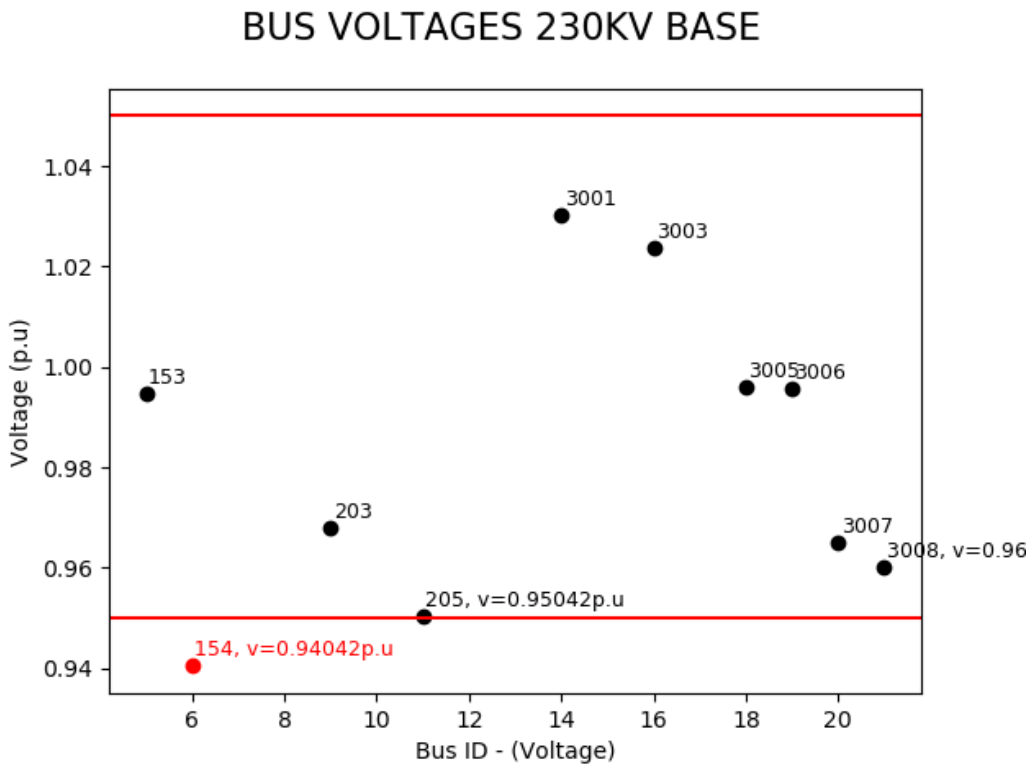
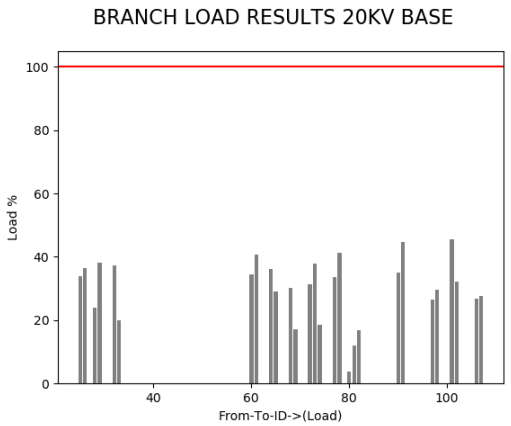


Figura 20 – Niveles de tensión obtenidos para los nudos de tensión nominal 230 kV tras la ejecución del ***flujo de cargas*** *(no OPF) en el archivo Estudio.sav proporcionado por PSSE*

Obsérvese que en la *Figura 20,* los niveles de tensión de los nudos únicamente se indican cuando se encuentran cerca de los límites establecidos, ya sea por dentro o por fuera.



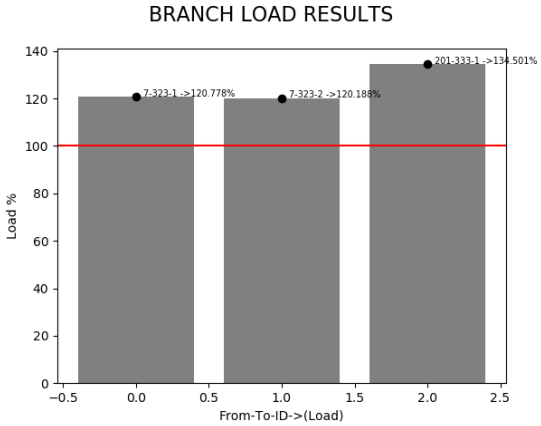


*Figura 21 y Figura 22 – Cargas en las líneas de 20kV de tensión nominal tras la ejecución del **flujo de cargas** (no OPF) en el archivo Estudio.sav proporcionado por PSSE (izquierda) y ejemplo de cómo se mostrarían en una gráfica sólo las líneas que presenten sobrecargas (derecha)*

De la misma manera, en la *Figura 21* se muestran las líneas en las que se produce una violación de los límites, junto con su valor actual, en el caso de que la carga de alguna línea superase el límite establecido.

4.2.2 ANÁLISIS DE CONTINGENCIAS N-1 N-2

El formato de los resultados obtenidos tras la ejecución de los análisis estáticos de contingencias es similar al utilizar al exportar los resultados del OPF. En este caso, se exportan únicamente datos de voltajes y cargas en líneas cuando se producen violaciones de límites en alguna contingencia. Se produce un único documento Word con todos los resultados de todas las contingencias posibles para todos los casos (horas), y un documento Excel por cada hora evaluada, en el que cada hoja son los resultados de tensiones o sobrecargas de las contingencias problemáticas.

En el caso de que el sistema no converja tras provocar una contingencia, o de que se trate de una línea o transformador que deje nudos o zonas enteras aislados de la red (situación en la que PSSE no puede ejecutar el flujo de cargas), se mostrará un mensaje en la pestaña correspondiente de que el caso no ha convergido. Estos últimos tipos de líneas serán, evidentemente, las mismas para todas las horas evaluadas.

Como se comentó en la *sección 3.4.2,* aunque no se ha utilizado en este proyecto por las características de Tenerife, el fichero permite realizar también el análisis de contingencias para líneas paralelas (N-2) a parte de para contingencias simples.

Por último, puesto que durante el análisis de contingencias también se ejecuta la función *GetLFResults (Sección 3.4.2.1)*, se pueden exportar gráficas de los resultados obtenidos de la misma manera de la que se muestra en la *Sección 4.2.1.*

A continuación, se muestran ejemplos de los resultados obtenidos tras los análisis. Es de esperar que se produzcan sobrecargas o tensiones demasiado bajas tras el apagado de líneas y transformaciones:

BusFROM	BusTO	ID	LOAD%
32	257	1	119,481
54	419	1	104,278
201	333	1	127,535

Tabla 3 – Sobrecargas de más del 100% del límite especificado para cada rama obtenidas en el sistema tras aplicar una contingencia simple en la línea 11-54-ID:1 en la hora 158 del despacho económico

Bus	Voltage(p.u)
11	0,948
39	0,9471
101	0,9494
370	0,9496
371	0,9494
467	0,9363
468	0,9363
469	0,9377

Tabla 4 – Tensiones fuera del rango 0.95 p.u – 1.10 p.u obtenidas en el sistema tras aplicar una contingencia simple en la línea 11-54-ID:1 en la hora 155 del despacho económico

Bus	Voltage(p.u)
466	1,1262

Tabla 5 - Tensiones fuera del rango 0.95 p.u – 1.10 p.u obtenidas en el sistema tras aplicar una contingencia simple en la línea 7-46-ID:2 en la hora 147 del despacho económico

Existen casos particulares como el de la *Tabla 5* - Tensiones fuera del rango 0.95 p.u – 1.10 p.u obtenidas en el sistema tras aplicar una contingencia simple en la línea 7-46-ID:2 en la hora 147 del despacho económicoen los cuales las propias consignas de tensión exportadas tras ejecutar el OPF se encuentran fuera de los límites establecidos debido a la necesidad de relajarlos para facilitar la convergencia del caso. Como ocurre en este caso concreto, suele deberse a un despacho económico erróneo, que provoca problemas de tensiones previos al OPF. Una solución simple para evitar este tipo de situaciones es endurecer (aplicando manualmente una penalización lineal en vez de un límite estricto) considerablemente la

penalización aplicada a la función de optimización tras relajar los límites de tensión de nudos, en detrimento de relajar la consigna de la función de optimización.

4.2.3 ANÁLISIS EN RÉGIMEN PERMANENTE DE PÉRDIDAS DE GENERACIÓN TRAS LA REGULACIÓN PRIMARIA

Como parte de los datos exportados tras los análisis dinámicos de pérdidas de generación, se exportan datos de tensiones fuera de rango y sobrecargas producidas tras la actuación de la regulación primaria. Se crea un documento Excel por cada hora del despacho económico evaluada, en la que cada pestaña corresponde con el apagado de un generador activo. Puesto que no se alcanza el régimen permanente completo en un periodo de tiempo razonable, se han utilizado los valores obtenidos al final del período de la simulación dinámica para poder exportar los resultados. Ejemplos de los resultados obtenidos son los siguientes. Es de esperar que se produzcan tensiones bajas tras el apagado de generadores, más acentuadas para máquinas de mayor potencia, y en mayor medida que cuando se realiza el apagado de líneas:

Bus	Voltage(p.u)
11	0,9456
467	0,9377
468	0,9377
469	0,9391

Tabla 6 – Tensiones fuera del rango 0.95 p.u – 1.10 p.u obtenidas en el sistema tras apagar el generador 134 en la hora 36 del despacho económico

En el caso observado en la *Tabla 6,* el generador despacha 21,12MW, el 5,87% del total de la generación del sistema, una cantidad modesta que produce un leve descenso en el perfil de tensiones del sistema.

Bus	Voltage(p.u)
7	0,928
11	0,9089
16	0,92
30	0,9265
31	0,9238

32	0,9265
37	0,9387
39	0,9141
46	0,9196
54	0,9197
69	0,9471
87	0,9461
99	0,9461
100	0,9398
101	0,9168
112	0,928
138	0,944
209	0,9181
257	0,9471
259	0,95
284	0,9227
293	0,9193
323	0,9195
326	0,9467
333	0,9284
341	0,9192
345	0,9196
370	0,9169
371	0,9168
419	0,9194
423	0,9471
455	0,9355
456	0,9354
467	0,9012
468	0,9012
469	0,9026
504	0,9463
505	0,9462
48001	0,92
49025	0,9284

Tabla 7 - Tensiones fuera del rango 0.95 p.u – 1.10 p.u obtenidas en el sistema tras apagar el generador 132 en la hora 36 del despacho económico

En el caso observado en la *Tabla 7,* el generador despacha 71,98 MW, el 19,9% del total de la generación del sistema, una cantidad considerable que provoca un brusco descenso en el

perfil de tensiones de la red, el cual se encontraba en condiciones aceptables antes del incidente.

4.3 RESULTADOS DE LOS ANÁLISIS DINÁMICOS

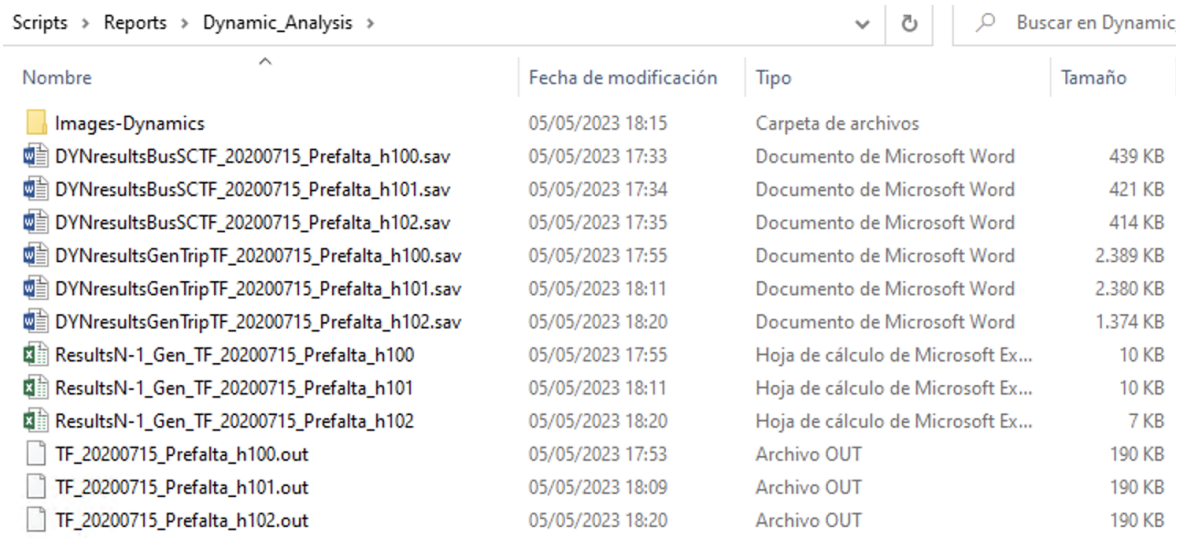


Figura 23 – Parte del contenido de la carpeta Dynamic_Analysis

4.3.1 ANÁLISIS DINÁMICOS DE PÉRDIDAS DE GENERACIÓN

Tras la ejecución de los análisis dinámicos de pérdidas de generación se exporta un documento Word con resultados para cada hora del despacho económico evaluada, en el cual se incluyen los resultados del apagado individual de cada generador. El usuario puede elegir entre exportar datos sobre la potencia eléctrica y mecánica de cada generador, la frecuencia del sistema u otro tipo de variables globales del sistema. A continuación, se muestran ejemplos de las gráficas obtenidas tras el apagado del generador 131 en la hora 36 del despacho:

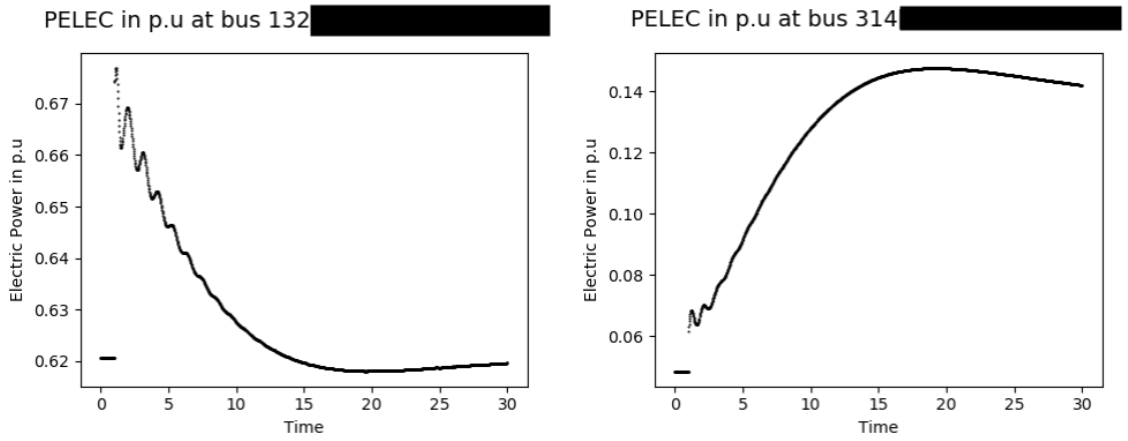


Figura 24 y Figura 25 - Evolución dinámica de la potencia eléctrica en p.u de los generadores 132 (izquierda) y 314 (derecha) (su nombre se ha ocultado) tras el apagado del generador 131 en la hora 36 del despacho económico

Obsérvese que existen discontinuidades en las gráficas obtenidas. Esto se debe a que el tiempo de muestreo de datos es mayor al paso de integración utilizado en el método de integración numérica (*Sección 2.4*). Esto permite reducir el tiempo de creación y exportación de las gráficas.

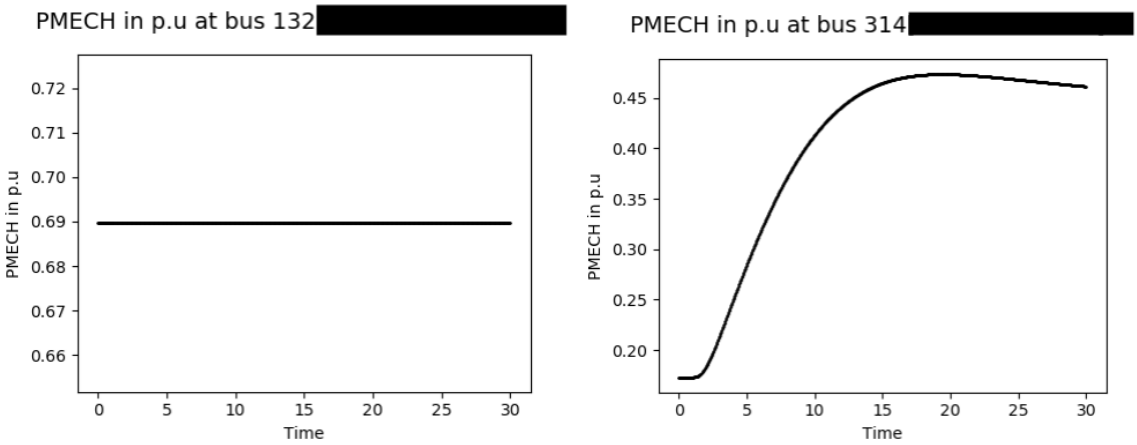


Figura 26 y Figura 27 - Evolución dinámica de la potencia mecánica en p.u de los generadores 132 (izquierda) y 314 (derecha) (su nombre se ha ocultado) tras el apagado del generador 131 en la hora 36 del despacho económico

Aunque ambas potencias eléctricicas y mecánicas de los generadores dependen de la frecuencia, tienen una dependencia distinta, y se comportan de distinta manera según los

modelos dinámicos de los distintos tipos de generadores. De esta manera, como se puede observar de la *Figura 24* a la *Figura 26,* las gráficas de potencia eléctrica y mecánica de los generadores se comportan de manera distinta.

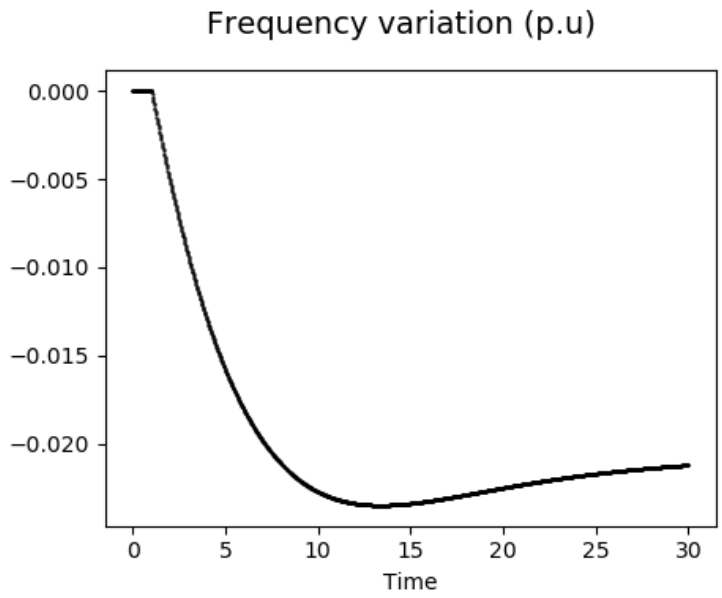


Figura 28 – Variación de la frecuencia del sistema en p.u tras el apagado del generador 131 en la hora 36 del despacho económico

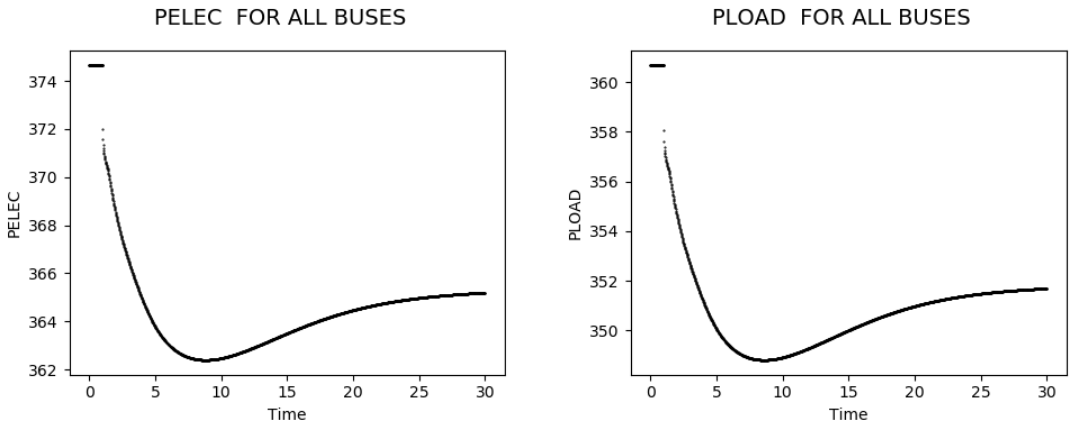


Figura 29 y Figura 30 – Variación de la potencia eléctrica y demanda del sistema tras el apagado del generador 131 en la hora 36 del despacho económico.

Obsérvese que la diferencia entre generación y demanda se debe a las pérdidas del sistema. En el caso concreto evaluado, la regulación primaria actúa con rapidez, y la frecuencia se

estabiliza en torno a 49 Hz, por lo que presenta una respuesta adecuada a la pérdida de generación (unos 30 MW, en torno al 10% del despacho del sistema).

De la misma manera de la que se ha podido observar en los resultados obtenidos en la *Sección 4.2.3,* por regla general desconexiones de generadores con una mayor potencia despachada producirán una mayor variación de la frecuencia y la potencia despachada de los otros generadores en régimen permanente, así como unas respuestas dinámicas mucho más acentuadas. Sin embargo, el sistema es muy complejo, y los resultados dependen mucho también de otras variables como la reserva de potencia en el resto de generadores, y en menor medida de otras como el perfil de tensiones o los flujos de reactiva. Obsérvese que, en general, los intervalos de alcance del régimen permanente total son mayores a los escogidos para exportar los datos (suficiente para observar estabilidad en el sistema) debido, de nuevo, a que se procura que el tiempo de generación de las gráficas no sea muy alto.

Para ahorrar tiempo de computación durante los análisis, y sólo extraer información detallada en situaciones problemáticas, el código está preparado para exportar todas las gráficas anteriormente vistas sobre las variables de interés sólo en los casos en los que la frecuencia del sistema desciende en cualquier momento por debajo de un nivel mínimo establecido. Para el resto de los casos en los que la frecuencia del sistema no desciende de forma problemática, simplemente se exporta al documento Word un mensaje en el que se muestra el valor máximo, mínimo y en régimen permanente de la frecuencia.

4.3.2 ANÁLISIS DINÁMICOS DE CORTOCIRCUITOS EN NUDOS SELECTOS

De la misma manera en la que se obtienen resultados para los análisis dinámicos de pérdidas de generación, tras la ejecución de los análisis dinámicos de cortocircuitos en nudos se exporta un documento Word con resultados para cada hora del despacho económico evaluada, en el cual se incluyen los resultados del cortocircuito individual en cada nudo. El usuario puede elegir entre exportar datos sobre el ángulo y el voltaje en cada nudo de generación, la variación de frecuencia del sistema, y estadística sobre el comportamiento general de los ángulos de los nudos de generación del sistema. A continuación, se muestran

ejemplos de las gráficas obtenidas tras el apagado del generador 419 en la hora 100 del despacho:

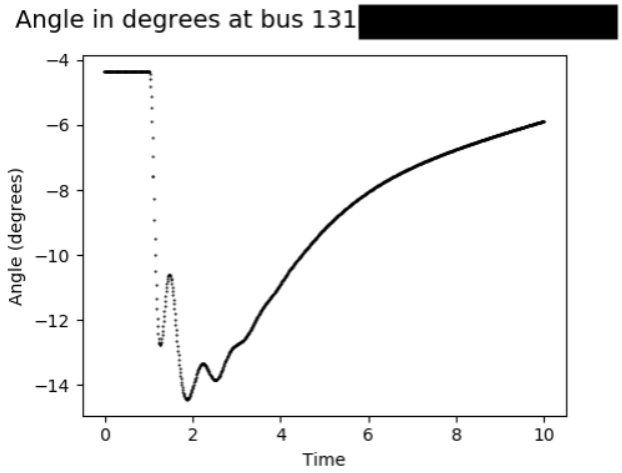


Figura 31 - Evolución dinámica del ángulo en grados del generador 131 (su nombre se ha ocultado) tras un cortocircuito fase-tierra en el nudo 411 en la hora 100 del despacho económico

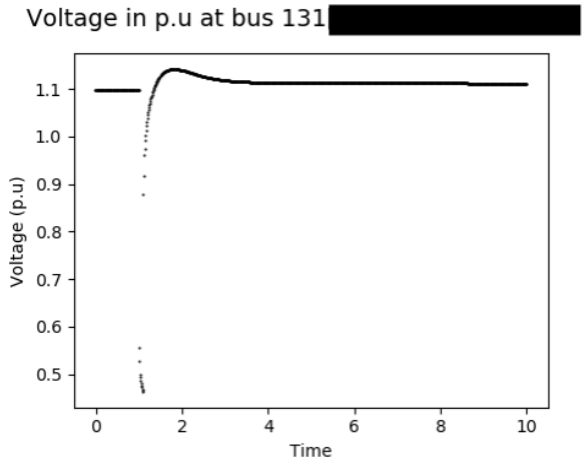


Figura 32 - Evolución dinámica del voltaje en p.u del generador 131 (su nombre se ha ocultado) tras un cortocircuito fase-tierra en el nudo 419 en la hora 100 del despacho económico

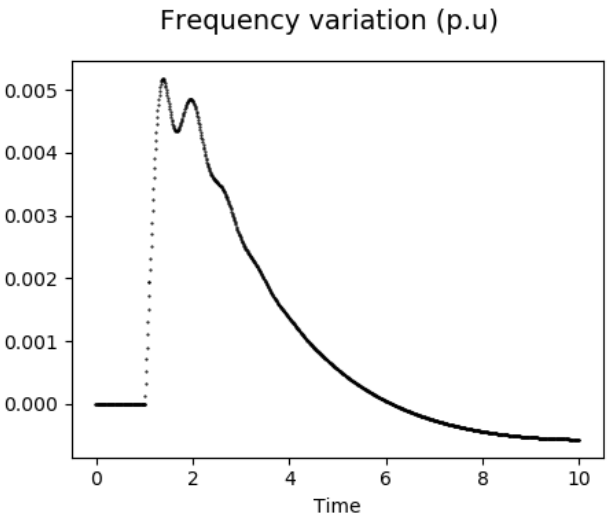


Figura 33 - Evolución dinámica de la frecuencia del sistema en p.u tras un cortocircuito fase-tierra en el nudo 419 en la hora 100 del despacho económico

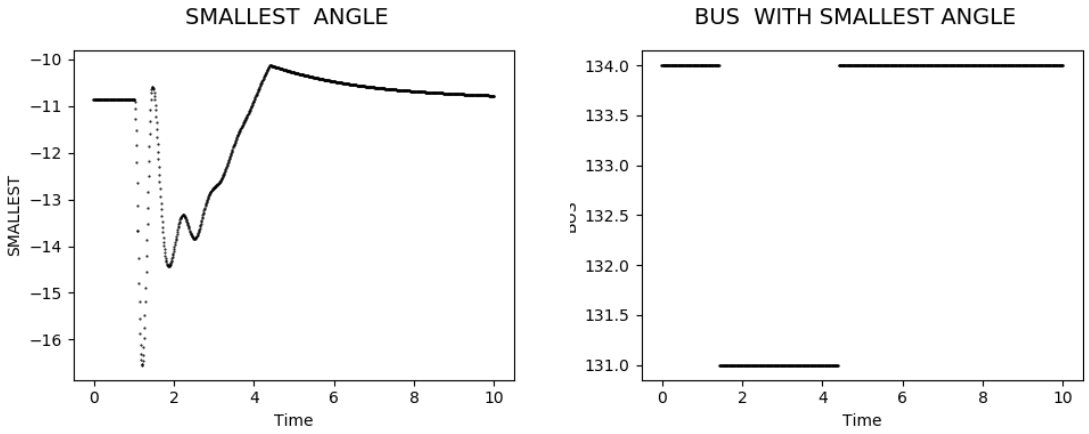


Figura 34 y Figura 35 - Evolución dinámica del ángulo más bajo de los nudos de generación (izquierda) y nudos en los que ocurre (derecha) tras un cortocircuito fase-tierra en el nudo 419 en la hora 100 del despacho económico

En general, y como se puede observar en el ejemplo aportado, tras un cortocircuito en un nudo, la frecuencia sufrirá una variación muy pequeña y una rápida recuperación hacia sus condiciones iniciales (*Figura 33*). De la misma manera, y a pesar de una gran caída del perfil de tensiones durante la duración de la falta, el perfil de tensiones se reestablece muy rápidamente hasta los valores iniciales (*Figura 32).*

Por otro lado, tras un cortocircuito los ángulos de los generadores sufren una gran variación (ya sea positiva o negativa) que puede resultar problemática durante los instantes posteriores a la falta. El valor de ángulos puede llegar a duplicarse o triplicarse (*Figura 31*), y en ocasiones su recuperación hasta las condiciones iniciales es lenta.

Para ahorrar tiempo de computación durante los análisis, y sólo extraer información detallada en situaciones problemáticas, el código está preparado para exportar todas las gráficas anteriormente vistas sobre las variables de interés sólo en los casos en los que los ángulos máximo o mínimo del sistema (con respecto al nudo *swing*) sobrepasan en cualquier momento los límites establecidos por el usuario, o si el ratio entre los valores pico de ángulo máximo o mínimo del sistema y su valor en régimen permanente es mayor que el establecido.

Para el resto de los casos en los que los ángulos del sistema no varían de forma problemática, simplemente se exporta al documento Word un mensaje en el que se muestra el valor máximo y en régimen permanente, y mínimo y en régimen permanente de dichos ángulos máximos y mínimos del sistema, respectivamente.

Capítulo 5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 IMPLEMENTACIÓN DE RESULTADOS – FUNCIONAMIENTO ÓPTIMO DE TENERIFE

Este proyecto nació de la necesidad de la necesidad de desarrollar una herramienta de análisis funcional, que integre en uno y ejecute automáticamente todos los análisis necesarios de forma versátil y aplicable a sistemas muy variopintos, que pudiese ser utilizada por el Instituto de Investigación Tecnológica.

Por otro lado, a parte del principal objetivo descrito anteriormente, el crear una herramienta de análisis funcional, otro importante objetivo de este trabajo ha sido el énfasis en continuar investigando en el cumplimiento de los Códigos de Red y, sobre todo, el cumplimiento del código de red sobre la conexión de generadores nuevos y de la implementación de fuentes de generación renovable en nuestras redes eléctricas.

A modo de conclusión general sobre el proyecto desarrollado, cabe destacar que se han cumplido las expectativas tanto para los objetivos propuesto para el inicio de año, como para los resultados obtenidos al realizar los análisis sobre la isla de Tenerife. La metodología desarrollada para la ejecución de los análisis ha funcionado correctamente, y tanto las funcionalidades de cada uno de los módulos, como la interconexión que tienen entre ellos se han escogido adecuadamente para realizar un análisis global y exhaustivo del sistema.

Respecto a los objetivos propuestos, se ha logrado desarrollar una serie de ficheros en Python que permite el procesamiento de datos sobre el despacho económico de la red por horas, la creación de un caso de estudio para cada hora, la ejecución del flujo de cargas óptimo en cada caso, y su posterior análisis a través de simulaciones estáticas y dinámicas. Todo este proceso se realiza automáticamente y con escasa intervención por parte de la persona que los utilice.

En todo momento, se ha procurado que los ficheros puedan ser utilizados modularmente (uno por separado, o todos en conjunto), y para que la adición de nuevas funcionalidades (funciones de Python que realicen nuevas rutinas en PSSE, en este caso) sea lo más sencilla posible. Además, se ha limitado la interacción que debe de realizar el usuario con los ficheros de Python a una entrada de datos relacionados con el sistema, explicada con instrucciones. De la misma manera, se ha procurado que los ficheros se puedan utilizar para sistemas con características muy distintas, no sólo para la isla de Tenerife.

Respecto a los resultados obtenidos al realizar los análisis en el sistema eléctrico de Tenerife, especialmente tras la gran penetración de fuentes de energía renovable que está ocurriendo, se han podido observar los resultados teorizados al inicio del proyecto. Al estar el sistema eléctrico de las islas Canarias aislado, presenta una robustez menor que sistemas eléctricos como el de la península ibérica.

Esto último se ha podido comprobar durante la extracción de resultados de los análisis, en los que frecuentemente se producen violaciones de límites (poco severos, generalmente) tras el apagado simple de líneas, o generadores, y ligeras desviaciones de frecuencia o de voltajes en los nudos de generación tras simulaciones de pérdidas de generación o de cortocircuitos en nudos. Incluso, se han podido observar algún caso concreto en el que el despacho económico provocaba un perfil de tensiones difícil de estabilizar por el OPF.

Sin embargo, y pese a la presencia considerable de generación renovable durante algunos instantes del día en la generación de Tenerife (hasta un 25-30%), se ha podido observar un comportamiento generalmente estable, en el que el flujo de cargas óptimo ha convergido para todos los casos del despacho económico semanal, y los problemas obtenidos tras la aplicación de faltas y contingencias han sido leves y corregibles.

5.2 CONTINUACIÓN DEL TRABAJO – POSIBLES APLICACIONES FUTURAS

Este proyecto ha sido concebido y desarrollado para facilitar el trabajo de personas que se encargan de realizar análisis sobre sistemas eléctricos para la detección de problemas y la implementación de mejoras. Consciente de que queda mucho contenido y muchas funcionalidades por añadir al proyecto desarrollado para que se pueda utilizar como modelo de análisis global de un sistema eléctrico, es importante definir las áreas de mejora, y posibles extensiones de los ficheros Python, sobre todo en el tratamiento de los datos obtenidos tras los análisis estáticos y dinámicos.

5.2.1 RETROALIMENTACIÓN AL OPF TRAS LOS ANÁLISIS ESTÁTICOS Y DINÁMICOS

Una de las posibles extensiones a implementar es la retroalimentación al flujo de cargas óptimo de los resultados de los análisis de contingencias en líneas, nudos y generadores, de manera que, en el caso de detectar alguna violación grave de los límites tras aplicar una contingencia en un caso, se produzcan modificaciones en las consignas del OPF para poder evitar esa posible contingencia al ejecutarlo de nuevo.

Por ejemplo, en el caso de detectar nudos con tensiones bajas (altos) tras aplicar una contingencia, se puede ordenar al OPF la conexión (desconexión) de bancos de condensadores localizados en zonas próximas a los nudos problemáticos, y cambiar su modo de operación de control de voltaje, (admitancia modificable durante el OPF) a bloqueado (admitancia constante durante el OPF)

Por otro lado, en el caso de detectar sobrecargas en líneas tras aplicar una contingencia, se pueden aplicar penalizaciones en el OPF al uso de ciertas líneas para regular su flujo, redistribuyéndolo a otras líneas menos problemáticas.

5.2.2 SIMULACIÓN DE CONTINGENCIAS EN CASCADA

Por último, y como tema de interés para las labores a realizar para proyectos del I.I.T, se pueden añadir funcionalidades a los ficheros Python para automatizar la simulación del comportamiento del sistema durante contingencias en cascada, es decir, durante contingencias que van produciéndose en "efecto dominó" conforme elementos del sistema van fallando o las protecciones van actuando para proteger a los equipos, algo común en la operación de los sistemas de potencia.

Un ejemplo directo de una simulación en cascada sería el apagado consecutivo de una línea en las que se produce una violación de los límites de carga tras una contingencia previa. Tras esta sucesión, se puede ejecutar de nuevo un análisis completo de la nueva situación para ver si la red se estabiliza o si, por el contrario, siguen surgiendo problemas en otros elementos de la red.

Capítulo 6. BIBLIOGRAFÍA

[1] AELEC (18 de julio de 2019). Norma técnica de supervisión de la conformidad de los módulos de generación de electricidad según el Reglamento UE 2016/631. *AELEC.* https://aelec.es/wp-content/uploads/2021/07/20210709-NTS-SEPE-v2.1.pdf

[2] Boletín Oficial del Estado (1 de agosto de 2022). Orden TED/749/2020, de 16 de julio, por la que se establecen los requisitos técnicos para la conexión a la red necesarios para la implementación de los códigos de red de conexión. *Boletín Oficial del Estado.*

[3] *Códigos de Red.* (30 de octubre de 2022). AELEC. https://aelec.es/codigos-de-red/

[4] *Objetivos de Desarrollo Sostenible.* (11 de enero de 2023). ONU. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[5] *Procedimientos de Operación.* (12 de enero de 2023). Red Eléctrica Española. https://www.ree.es/es/actividades/operacion-del-sistema-electrico/procedimientos-de-operacion

[6] *Red Eléctrica – Nuestra historia.* (30 de octubre de 2022). Red Eléctrica Española. https://www.ree.es/es/conocenos/ree-en-2-minutos/nuestra-historia

[7] Siemens Power Technologies International. (Julio de 2019). *Application Program Interface (API) PSSE 34.6.0*

[8] Siemens Power Technologies International. (2019). *Program Operation Manual PSSE 34.6.1*

[9] Rouco Rodríguez, L. (2021). *Electric Power Systems – Power Flow*

[10] García Amorós, O. (2022). *Sistemas de Energía Eléctrica – Tema 3: Control fP*

[11] Egido, I., Renedo, J., Sigrist, L. (2021). *Modelado de parques eólicos en PSSE para estudios estáticos*

[12] García Amorós, O. (2022). *Sistemas de Energía Eléctrica – Tema 2: Control Q-V*

ANEXO I: FICHERO PYTHON

ANALYSIS_PART_1_IMPORTS

```
# -*- coding: cp1252 -*-

# Program
# ---------------------------------------


def
RUN_ANALYSIS(str_aux_folder,str_generation_file,str_demand_file,str_excel_script_files,file_correspondence_gen,f
ile_correspondence_dem,str_user_files,str_aux_case_folder,n_hours,slack_bus):

    #Imports all the necessary parameters to run the function
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import collections
    #import dyntools

    #Imports the other scripts to be used as modules
    import Excel_Extraction
    import Python_To_PSSE

    #Creates the "Analysis_Data" folder if it does not exist.
    if not os.path.exists(str_aux_folder):
        os.makedirs(str_aux_folder)

    #Creates the "Excel_Files" folder if it does not exist.
    if not os.path.exists(str_excel_script_files):
        os.makedirs(str_excel_script_files)

    #Runs the function that generates the Generation dictionary
    Generators = Excel_Extraction.Gen_Excel_To_Dictionary(str_generation_file,
str_excel_script_files,file_correspondence_gen)
    #Runs the function that generates the Generation dictionary
    Demand =
Excel_Extraction.Dem_Excel_To_Dictionary(str_demand_file,str_excel_script_files,file_correspondence_gen)

    #Opens the PSSE-DESI CONVENTIONAL GENERATION Correspondence file
    xls_gen = pd.ExcelFile(file_correspondence_gen)
    datos_hoja_gen=xls_gen.parse('GeneracionConvencional')
    df_correspondencia_gen=datos_hoja_gen[["Nudo","Nombre","PGen (MW)","PMax (MW)","PMin (MW)","QGen
(Mvar)","QMax (Mvar)","QMin (Mvar)","Mbase (MVA)","Correspondencia DESI","Comentario"]]

    #Opens the PSSE-DESI RES GENERATION/DEMAND Correspondence file
    xls_dem = pd.ExcelFile(file_correspondence_dem)
    datos_hoja_dem=xls_dem.parse('Demanda')
    datos_hoja_RES=xls_dem.parse('GeneracionRES')
    df_correspondencia_dem=datos_hoja_dem[["Nudo","Nombre","ID","Pload (MW)","% Demanda"]]
    df_correspondencia_RES=datos_hoja_RES[["Nudo","Nombre","ID","Pload (MW)","Tipo","% Demanda"]]
```

```
    #Creates the "Case_Data" folder if it does not exist.
    if not os.path.exists(str_aux_case_folder):
        os.makedirs(str_aux_case_folder)

    #Executes the function that introduces the data onto the Python files

Python_To_PSSE.Python_To_PSSE(n_hours,Generators,Demand,str_user_files,str_aux_case_folder,df_correspondencia_ge
n,df_correspondencia_dem,df_correspondencia_RES,slack_bus)




if __name__=="__main__":

   # The main runs the functions defined above.

    import os, sys

# ---------------------------------------
# PARAMETERS TO BE DEFINED BY THE USER FOR THE EXCEL EXTRACTION FUNCTION AND FOR THE PYTHON_TO_PSSE FUNCTION
# ---------------------------------------
    #THE INPUT FILES (.txt generation, correspondence file, .sav file) SHOULD BE INCLUDED ALL IN THE SAME FOLDER
    #Introduce the .txtfile that contains all the generation data
    str_generation_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIgen.txt"
    #Introduce the .txtfile that contains all the demand data
    str_demand_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIisla.txt"
    #Introduce the folder where the this script´s auxiliary files are going to be located
    str_aux_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data"
    #Introduce the folder where the this script´s Excel auxiliary files are going to be located
    str_excel_script_files = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Excel_files"
    #Introduce the path of the file that has the correspondence between DESI and PSSE for conventional
generation (SAVED AS .XLS)
    file_correspondence_gen = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-
DESI_convencional.xls"
    #Introduce the path of the file that has the correspondence between DESI and PSSE for demand and RES
generation (SAVED AS .XLS)
    file_correspondence_dem = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-
DESI_dem_RES.xls"
    #Introduce the folder where the .sav files inputted by the user are located
    str_user_files = r"C:\Users\proyectista\Desktop\Scripts\User_data_input"
    #Introduce the path of the folder where the auxiliary case files are going to be created
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the number of the slack bus
    slack_bus = 284
    #Introduce the number of hours evaluated (Daily execution-24, Weekly execution-168)
    n_hours = 24


    ##    ####################################################
    ##    #FILE REQUIREMENTS FOR THE Excel_extraction and Python_To_PSSE:
    ##    #           1)The python scripts "Excel_Extraction" and "Python_To_PSSE" will be located in the same
folder as the ANALYSIS SCRIPT
    ##    #           2)The .sav file, PSSE-DESI correspondence files, .txt generation file and .txt load file
should all be in a same folder called "User_data_input"
    ##    #           3)A folder called "Excel_files" will contain the script, a xls with the data and a .txt
with the dictionary
    ##    #           4)The PSSE-DESI correspondence files must be a .xls file with the top left cell ("Nudo 1")
starting in A1, and the table must be alphabetically
    ##    #             ordered according to the "Correspondencia_DESI" column. It will have only one sheet
called "GeneracionConvencional"
    ##    #           5)The code is prepared to work with a generic case in which THE CODE NAMES FOR REGULAR
GENERATORS THE COMBINED CYCLES FOLLOW THE SAME NAME PATTERNS
    ##    #             AS TENERIFE AND THE PSSE-DESI CORRESPONDENCE FILE
    ##    #The output of this function are two dictionaries containing generation and load data
      ####################################################


    #Runs a complete analysis

RUN_ANALYSIS(str_aux_folder,str_generation_file,str_demand_file,str_excel_script_files,file_correspondence_gen,f
ile_correspondence_dem,str_user_files,str_aux_case_folder,n_hours,slack_bus)
```

ANEXO 2: FICHERO PYTHON

EXCEL_EXTRACTION

```
# -*- coding: cp1252 -*-
def Gen_Excel_To_Dictionary(str_generation_file, str_script_files,file_correspondence):

    #Imports necessary modules to run the function
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import csv
    import openpyxl
    import fileinput
    import openpyxl
    import xlsxwriter
    import numpy
    import xlrd, xlwt
    from openpyxl import load_workbook
    import json
    import collections

    #Removes all the euros in the text file
    search_text = "€"
    replace_text = ""
    with open(str_generation_file, 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(str_generation_file, 'w') as file:
        file.write(data)

    #Removes all the spaces in the file
    search_text = " "
    replace_text = ""
    with open(str_generation_file, 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(str_generation_file, 'w') as file:
        file.write(data)

    #Inputs all the data from the .txt file to an excel file
    read_file = pd.read_csv(str_generation_file,sep=';')
    read_file.to_csv(str_script_files + "\\" +"resDSIgen.csv", index=None)
    df=pd.read_csv(str_script_files + "\\" + "resDSIgen.csv",encoding='unicode_escape')
    df.to_excel(str_script_files + "\\" +"resDSIgen.xlsx", 'Generation')
    os.remove(str_script_files +"\\" + "resDSIgen.csv")

    #Extracts data from the Excel file
    filexls =str_script_files + "\\" + "resDSIgen.xlsx"
    workbook = openpyxl.load_workbook(filexls)
    sheet = workbook.active
    sheet.delete_cols(1)
    data = sheet.values
    columns = next(data)[0:]
```

```
    df_generators = pd.DataFrame(data, columns=columns)

    #Checks if a dictionary with the Generation information exists and imports it. If it does not exist, it
creates the Generation dictionary
##    dictionary_exists = os.path.isfile(str_script_files+ "\\" +"Generation_dictionary.txt")
##    if(dictionary_exists == True):
##        with open(str_script_files+ "\\" +"Generation_dictionary.txt") as f:
##            Generators = f.read()
##        Generators = json.loads(Generators)
##        dictionary_generation = 1
##        print(Generators)
##    else:
##        Generators = collections.OrderedDict()
##        dictionary_generation = 0

    #Creates an Ordered Dictionary that will store allthe conventional generation data
    Generators = collections.OrderedDict()
    dictionary_generation = 0
    if(dictionary_generation == 0):
        numero_variables = 0
        #Gets the number of variables evaluated for each generator
        for i in range(1,1000):
            if((df_generators.iloc[i-1,5] == df_generators.iloc[i,5])):
                numero_variables+=1
            if((df_generators.iloc[i-1,5] != df_generators.iloc[i,5])):
               break
        numero_variables+=1

       #Creates and fills the generation dictionary
        for gen in range(len(df_generators.iloc[:,2])):
            #Creates a first dictionary level for each generator
            if(df_generators.iloc[gen-1,2] != df_generators.iloc[gen,2]):
                Generators[str(df_generators.iloc[gen,2])]={}
                #print(str(df_generators.iloc[gen,2]))

                #Determines wether the execution for this generator is for 24 or 168 hours
                number_hours = 0
                while((gen+number_hours) != (len(df_generators.iloc[:,2])-
1)and(df_generators.iloc[gen+number_hours,2] == df_generators.iloc[gen+number_hours+1,2])):
                    number_hours+=1
                number_hours+=1
                number_hours = number_hours/numero_variables
                #Creates a secondary dictionary level for each hour evaluated (taking into account it can be 24-
hour or 168-hour execution)
                for hour in range(number_hours):
                    Generators[df_generators.iloc[gen,2]][hour+1] = {}
                    for variable in range(numero_variables):#Number of non-zero variables
                        #Creates a third dictionary level for each variable evaluated in each hour, and fills it
with the value
                        if((float(df_generators.iloc[gen+(hour*numero_variables)+variable,7]) != 0) and
((gen+(hour*numero_variables)+variable) != (len(df_generators.iloc[:,2])))):

Generators[str(df_generators.iloc[gen,2])][hour+1][str(df_generators.iloc[gen+(hour*numero_variables)+variable,6
])] = {}

Generators[str(df_generators.iloc[gen,2])][hour+1][str(df_generators.iloc[gen+(hour*numero_variables)+variable,6
])] = float(df_generators.iloc[gen+(hour*numero_variables)+variable,7])

    #Returns the Ordered Dictionary as the output of the function
    return Generators

##    #Saves the generation dictionary to an external file (if it is not saved yet) for a faster read later
##    if(dictionary_generation==0):
##        with open(str_script_files+ "\\" +"Generation_dictionary.txt", 'w') as convert_file:
##            convert_file.write(json.dumps(Generators))

def Dem_Excel_To_Dictionary(str_demand_file,str_script_files,file_correspondence):

    #Imports all the necessary modules to run the function
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
```

```
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import csv
    import openpyxl
    import fileinput
    import openpyxl
    import xlsxwriter
    import numpy
    import xlrd, xlwt
    from openpyxl import load_workbook
    import json
    import collections

    #Removes all the euros in the text file
    search_text = "€"
    replace_text = ""
    with open(str_demand_file, 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(str_demand_file, 'w') as file:
        file.write(data)

    #Removes all the ° in the file
    search_text = "°"
    replace_text = ""
    with open(str_demand_file, 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(str_demand_file, 'w') as file:
        file.write(data)

    #Removes all the spaces in the file
    search_text = " "
    replace_text = ""
    with open(str_demand_file, 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(str_demand_file, 'w') as file:
        file.write(data)

    #Inputs all the data from the .txt file to an excel file
    read_file = pd.read_csv(str_demand_file,sep=';')
    read_file.to_csv(str_script_files + "\\" +"resDSIisla.csv", index=None)
    df=pd.read_csv(str_script_files + "\\" + "resDSIisla.csv",encoding='unicode_escape')
    df.to_excel(str_script_files + "\\" +"resDSIisla.xlsx", 'Demand')
    os.remove(str_script_files +"\\" + "resDSIisla.csv")

    #Extracts data from the Excel file
    filexls =str_script_files + "\\" + "resDSIisla.xlsx"
    workbook = openpyxl.load_workbook(filexls)
    sheet = workbook.active
    sheet.delete_cols(1)
    data = sheet.values
    columns = next(data)[0:]
    df_demand = pd.DataFrame(data, columns=columns)

    #Checks if a dictionary with the Demand information exists and imports it. If it does not exist, it creates
the Demand dictionary
##    dictionary_exists = os.path.isfile(str_script_files+ "\\" +"Demand_dictionary.txt")
##    if(dictionary_exists == True):
##        with open(str_script_files+ "\\" +"Demand_dictionary.txt") as f:
##            Demand = f.read()
##        Demand = json.loads(Demand)
##        dictionary_demand = 1
##        print(Demand)
##    else:
##        Demand = collections.OrderedDict()
```

```
##        dictionary_demand = 0

    #Creates an ordered dictionary that will store all the information about demand and RES generation
    Demand = collections.OrderedDict()
    dictionary_demand = 0
    if(dictionary_demand == 0):
        numero_variables = 0
        #Gets the number of variables evaluated for each hour
        for i in range(1,1000):
            if((df_demand.iloc[i-1,3] == df_demand.iloc[i,3])):
                numero_variables+=1
            if((df_demand.iloc[i-1,3] != df_demand.iloc[i,3])):
               break
        numero_variables+=1

       #Creates and fills the demand dictionary
        i=0
        for hour in range(len(df_demand.iloc[:,2])):
            #Creates a first dictionary level for each hour
            if(df_demand.iloc[hour-1,3] != df_demand.iloc[hour,3]):
                i+=1
                Demand[i]={}
                #print(str(df_demand.iloc[hour,3]))

                #Creates a secondary dictionary level for each variable evaluated
                for variable in range(numero_variables):#Number of non-zero variables
                    #Creates a third dictionary level for each variable evaluated in each hour, and fills it
with the value
                    if((hour+variable) != (len(df_demand.iloc[:,2]))):
                        Demand[i][str(df_demand.iloc[hour+variable,4])] = {}
                        Demand[i][str(df_demand.iloc[hour+variable,4])] = float(df_demand.iloc[hour+variable,5])
                    variable+=1

    #Returns the demand ordered dictionary as the output of the function
    return Demand
##    #Saves the demand dictionary to an external file (if it is not saved yet) for a faster read later
##    if(dictionary_demand==0):
##        with open(str_script_files+ "\\" +"Demand_dictionary.txt", 'w') as convert_file:
##            convert_file.write(json.dumps(Demand))


if __name__=="__main__":

   # The main runs the functions defined above.

    import os, sys

##   # ---------------------------------------
##   # PARAMETERS TO BE DEFINED BY THE USER
##   # ---------------------------------------
##   #THE INPUT FILES (.txt generation, correspondence file, .sav file) SHOULD BE INCLUDED ALL IN THE SAME
FOLDER
##    #Introduce the .txt file that contains all the generation data
##    str_generation_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIgen.txt"
##    #Introduce the .txt file that contains all the demand data
##    str_demand_file = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\resDSIisla.txt"
##    #Introduce the folder where the this script´s auxiliary files are going to be located
##    str_script_files = r"C:\Users\proyectista\Desktop\Scripts\Excel_files"
##    #Introduce the path of the file that has the correspondence between DESI and PSSE (SAVED AS .XLS)
##    file_correspondence = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-
DESI_convencional.xls"

    #Runs the function that generates the Generation dictionary
    #Gen_Excel_To_Dictionary(str_generation_file, str_script_files,file_correspondence)
    #Runs the function that generates the Generation dictionary
    #Dem_Excel_To_Dictionary(str_demand_file,str_script_files,file_correspondence)
```

ANEXO 3: FICHERO PYTHON

PYTHON_TO_PSSE

```
# ----------
# Function
# ----------

#Defines the main function
def
Python_To_PSSE(n_hours,Generators,Demand,str_user_files,str_aux_case_folder,df_correspondencia_gen,df_correspond
encia_dem,df_correspondencia_RES,slack_bus):

    #Imports all the necessary modules to run the function
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    #from docx.text.parargaph import Paragraph
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import collections
    import psse34
    import psspy
    import redirect
    redirect.psse2py()

    #Defines some parameters to be usedby PSSE
    _i=psspy.getdefaultint()
    _f=psspy.getdefaultreal()
    _s=psspy.getdefaultchar()

    sid = -1
    ties=3
    flag=1
    entry=1

    # go through all .sav files in the User_input folder
    for file in os.listdir(str_user_files):
        if file.endswith(".sav"):
            psspy.psseinit(20000)
            strfilefdc = file
            strpathfilefdc = str_user_files + '\\' + strfilefdc
            #Creates and fills a .sav file for every hour
            for i in range (n_hours):
                psspy.case(strpathfilefdc)
                saved_file = str_aux_case_folder + '\\' + strfilefdc

            #Sets the generator out of service if it is not used
                for gen in range(len(df_correspondencia_gen.iloc[:,0])):
                    if((str(df_correspondencia_gen.iloc[gen,10]) == "No usado") or
(str(df_correspondencia_gen.iloc[gen,10]) == "Indisponible")):
                        in_service = 0
                        Pgen = 0
                        ID = r"""1"""
                        if (df_correspondencia_gen.iloc[gen-1,0] == df_correspondencia_gen.iloc[gen,0]):
                           ID = r"""2"""
```

```
psspy.machine_chng_2(df_correspondencia_gen.iloc[gen,0],ID,[in_service,_i,_i,_i,_i,_i],[Pgen,_f,_f,_f,_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f])

                #Checks the correspondenve file to evaluate each generator independently
                for gen in range(len(df_correspondencia_gen.iloc[:,9])):
                    if(type(df_correspondencia_gen.iloc[gen,9])!= float):
                        print(i)

                        #Checks if it is a Combined Cycle
                        #(Criteria to find combined cycle: if it has a "_" in the Generator name)
                        if(df_correspondencia_gen.iloc[gen,9].find("_")>0):
                            combined_cycle = True
                            gas1 = df_correspondencia_gen.iloc[gen,9]
                            gas2 = df_correspondencia_gen.iloc[gen+1,9]
                            steam = df_correspondencia_gen.iloc[gen+2,9]
                        else:
                            combined_cycle = False

                        #Obtains the generation data if it is not a Combined Cycle
                        if(combined_cycle == False):
                            if (Generators[df_correspondencia_gen.iloc[gen,9]][i+1].has_key("e[MWh]")):
                                in_service = 1
                                Pgen = Generators[df_correspondencia_gen.iloc[gen,9]][i+1]["e[MWh]"]
                            else:
                                in_service = 0
                                Pgen = 0

                            ID = r"""1"""
                            if (df_correspondencia_gen.iloc[gen-1,0] == df_correspondencia_gen.iloc[gen,0]):
                               ID = r"""2"""

psspy.machine_chng_2(df_correspondencia_gen.iloc[gen,0],ID,[in_service,_i,_i,_i,_i,_i],[Pgen,_f,_f,_f,_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f])

        ###############################################################################
                            #Code to switch on the corresponding bus and transformer in case that they are
inactive
                            if((Pgen>0)and(df_correspondencia_gen.iloc[gen,0]!=slack_bus)):
                                #Switches on the bus

psspy.bus_chng_4(df_correspondencia_gen.iloc[gen,0],0,[2,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f],_s)
                                #Switches on the transformer (assumes that the only branch connected to the bus
is the step-up transformer)
                                winding1 = df_correspondencia_gen.iloc[gen,0]
                                ierr = psspy.inibrn(winding1, 2)
                                ierr, winding2, ickt = psspy.nxtbrn(winding1)

psspy.two_winding_chng_6(winding1,winding2,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

psspy.two_winding_chng_6(winding2,winding1,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

        ###############################################################################

                        #Obtains the operation point for the Combined Cycle evaluated in the Correspondence file
                        if((combined_cycle == True)and(df_correspondencia_gen.iloc[gen,9][:-2] ==
df_correspondencia_gen.iloc[gen+2,9][:-1])):
                            g1_name = df_correspondencia_gen.iloc[gen,9]
                            g2_name = df_correspondencia_gen.iloc[gen+1,9]
                            g1_steam_name = df_correspondencia_gen.iloc[gen,9] + "V"
                            g2_steam_name = df_correspondencia_gen.iloc[gen+1,9] + "V"
                            g1_g2_steam_name = df_correspondencia_gen.iloc[gen,9] +
df_correspondencia_gen.iloc[gen+1,9][-2:]+"V"

                            if(Generators[g1_name][i+1].has_key("e[MWh]")): #Operating simple mode with first
gas turbine on
                                Pgen_gas1 = Generators[g1_name][i+1]["e[MWh]"]
                                Pgen_gas2 = 0
                                Pgen_steam = 0
                                in_service_gas1 = 1
                                in_service_gas2 = 0
```

```
                                in_service_steam = 0
                            elif(Generators[g2_name][i+1].has_key("e[MWh]")): #Operating simple mode with second
gas turbine on
                                Pgen_gas1 = 0
                                Pgen_gas2 = Generators[g2_name][i+1]["e[MWh]"]
                                Pgen_steam = 0
                                in_service_gas1 = 0
                                in_service_gas2 = 1
                                in_service_steam = 0
                            elif(Generators[g1_steam_name][i+1].has_key("e[MWh]")): #Operating compound mode
with first gas turbine and steam turbine on
                                Pgen_gas1 =
Generators[g1_steam_name][i+1]["e[MWh]"]*df_correspondencia_gen.iloc[gen,3]/(df_correspondencia_gen.iloc[gen,3]+
(0.5*df_correspondencia_gen.iloc[gen+2,3]))
                                Pgen_gas2 = 0
                                Pgen_steam =
Generators[g1_steam_name][i+1]["e[MWh]"]*(0.5*df_correspondencia_gen.iloc[gen+2,3])/(df_correspondencia_gen.iloc
[gen,3]+(0.5*df_correspondencia_gen.iloc[gen+2,3]))
                                in_service_gas1 = 1
                                in_service_gas2 = 0
                                in_service_steam = 1
                            elif(Generators[g2_steam_name][i+1].has_key("e[MWh]")): #Operating compound mode
with second gas turbine and steam turbine on
                                Pgen_gas1 = 0
                                Pgen_gas2 =
Generators[g2_steam_name][i+1]["e[MWh]"]*df_correspondencia_gen.iloc[gen+1,3]/(df_correspondencia_gen.iloc[gen+1
,3]+(0.5*df_correspondencia_gen.iloc[gen+2,3]))
                                Pgen_steam =
Generators[g2_steam_name][i+1]["e[MWh]"]*(0.5*df_correspondencia_gen.iloc[gen+2,3])/(df_correspondencia_gen.iloc
[gen+1,3]+(0.5*df_correspondencia_gen.iloc[gen+2,3]))
                                in_service_gas1 = 0
                                in_service_gas2 = 1
                                in_service_steam = 1
                            elif(Generators[g1_g2_steam_name][i+1].has_key("e[MWh]")): #Operating compound mode
with two gas turbines and steam turbine on
                                Pgen_gas1 =
Generators[g1_g2_steam_name][i+1]["e[MWh]"]*df_correspondencia_gen.iloc[gen,3]/(df_correspondencia_gen.iloc[gen,
3]+df_correspondencia_gen.iloc[gen+1,3]+df_correspondencia_gen.iloc[gen+2,3])
                                Pgen_gas2 =
Generators[g1_g2_steam_name][i+1]["e[MWh]"]*df_correspondencia_gen.iloc[gen+1,3]/(df_correspondencia_gen.iloc[ge
n,3]+df_correspondencia_gen.iloc[gen+1,3]+df_correspondencia_gen.iloc[gen+2,3])
                                Pgen_steam =
Generators[g1_g2_steam_name][i+1]["e[MWh]"]*df_correspondencia_gen.iloc[gen+2,3]/(df_correspondencia_gen.iloc[ge
n,3]+df_correspondencia_gen.iloc[gen+1,3]+df_correspondencia_gen.iloc[gen+2,3])
                                in_service_gas1 = 1
                                in_service_gas2 = 1
                                in_service_steam = 1
                            else:
                                Pgen_gas1 = 0
                                Pgen_gas2 = 0
                                Pgen_steam = 0
                                in_service_gas1 = 0
                                in_service_gas2 = 0
                                in_service_steam = 0

psspy.machine_chng_2(df_correspondencia_gen.iloc[gen,0],r"""1""",[in_service_gas1,_i,_i,_i,_i,_i],[Pgen_gas1,_f,
_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])

psspy.machine_chng_2(df_correspondencia_gen.iloc[gen+1,0],r"""1""",[in_service_gas2,_i,_i,_i,_i,_i],[Pgen_gas2,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])

psspy.machine_chng_2(df_correspondencia_gen.iloc[gen+2,0],r"""1""",[in_service_steam,_i,_i,_i,_i,_i],[Pgen_steam
,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])

            ########################################################################
                            #Code to switch on the corresponding gas turbine 1 bus and transformer in case that
they are inactive
                            if((Pgen_gas1>0)and(df_correspondencia_gen.iloc[gen,0]!=slack_bus)):
                                #Switches on the bus

psspy.bus_chng_4(df_correspondencia_gen.iloc[gen,0],0,[2,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f],_s)
                                #Switches on the transformer (assumes that the only branch connected to the bus
is the step-up transformer)
```

```
                                winding1 = df_correspondencia_gen.iloc[gen,0]
                                ierr = psspy.inibrn(winding1, 2)
                                ierr, winding2, ickt = psspy.nxtbrn(winding1)

psspy.two_winding_chng_6(winding1,winding2,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

psspy.two_winding_chng_6(winding2,winding1,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

                            #Code to switch on the corresponding gas turbine 1 bus and transformer in case that
they are inactive
                            if((Pgen_gas2>0)and(df_correspondencia_gen.iloc[gen+1,0]!=slack_bus)):
                                #Switches on the bus

psspy.bus_chng_4(df_correspondencia_gen.iloc[gen+1,0],0,[2,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f],_s)
                                #Switches on the transformer (assumes that the only branch connected to the bus
is the step-up transformer)
                                winding1 = df_correspondencia_gen.iloc[gen+1,0]
                                ierr = psspy.inibrn(winding1, 2)
                                ierr, winding2, ickt = psspy.nxtbrn(winding1)

psspy.two_winding_chng_6(winding1,winding2,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

psspy.two_winding_chng_6(winding2,winding1,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

                            #Code to switch on the corresponding steam turbine bus and transformer in case that
they are inactive
                            if((Pgen_steam>0)and(df_correspondencia_gen.iloc[gen+2,0]!=slack_bus)):
                                #Switches on the bus

psspy.bus_chng_4(df_correspondencia_gen.iloc[gen+2,0],0,[2,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f],_s)
                                #Switches on the transformer (assumes that the only branch connected to the bus
is the step-up transformer)
                                winding1 = df_correspondencia_gen.iloc[gen+2,0]
                                ierr = psspy.inibrn(winding1, 2)
                                ierr, winding2, ickt = psspy.nxtbrn(winding1)

psspy.two_winding_chng_6(winding1,winding2,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

psspy.two_winding_chng_6(winding2,winding1,r"""1""",[1,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_
f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"","")

        ########################################################################

                #Extracts data for total reactive power consumed by all demand and RES generation
                ierr, v_allloads = psspy.aloadint(sid, flag, 'NUMBER')
                v_allloads = v_allloads[0]

                #Sets demand proportional to the base case. Active power is changedby multiplying the total MW
demand by its fixed demand share percentage.
                #Reactive power is changed in percentage as much as the active changes
                demand_load=0
                for demand_load in range(len(df_correspondencia_dem.iloc[:,0])):
                    ierr, cmpval_dem = psspy.loddt2(df_correspondencia_dem.iloc[demand_load,0],
str(df_correspondencia_dem.iloc[demand_load,2]) ,"MVA", "ACT")
                    P_dem = df_correspondencia_dem.iloc[demand_load,4]*Demand[i+1]["dem[MWh]"]
                    Q_dem = cmpval_dem.imag*P_dem/cmpval_dem.real

psspy.load_chng_5(df_correspondencia_dem.iloc[demand_load,0],str(df_correspondencia_dem.iloc[demand_load,2]),[_i
,_i,_i,_i,_i,_i,_i],[P_dem,Q_dem,_f,_f,_f,_f,_f,_f])

                #Sets RES generation proportional to the base case. Active power is changedby multiplying the
total MW demand by its fixed demand share percentage.
                #Reactive power is changed in percentage as much as the active changes
                for RES_load in range(len(df_correspondencia_RES.iloc[:,0])):
                    ierr, cmpval_RES = psspy.loddt2(df_correspondencia_RES.iloc[RES_load,0],
str(df_correspondencia_RES.iloc[RES_load,2]) ,"MVA", "ACT")
```

```
                    P_RES = -
(df_correspondencia_RES.iloc[RES_load,5]*Demand[i+1]["wind[MWh]"]+df_correspondencia_RES.iloc[RES_load,5]*Demand
[i+1]["solar[MWh]"])
                    Q_RES = cmpval_RES.imag*P_RES/cmpval_RES.real

psspy.load_chng_5(df_correspondencia_RES.iloc[RES_load,0],str(df_correspondencia_RES.iloc[RES_load,2]),[_i,_i,_i
,_i,_i,_i,_i,_i],[P_RES,Q_RES,_f,_f,_f,_f,_f,_f])


                #Gets all switched shunts buses
                ierr, v_allswshunts = psspy.aswshint(sid, flag, 'NUMBER')
                v_allswshunts = v_allswshunts[0]

                #Gets all fixed shunts and turns them into switched shunts
                ierr, v_allfxshunts = psspy.afxshuntint(sid, flag, 'NUMBER')
                v_allfxshunts = v_allfxshunts[0]
                ierr, mvar = psspy.afxshuntreal(sid, flag,'SHUNTNOM')
                mvar = mvar[0]

                for q in range(len(v_allfxshunts)):
                #If a switched shunt already exists on the bus, adds the MVA of the fixed shunt to the fist
block of the switched shunt
                    if (v_allswshunts.count(v_allfxshunts[q])>0):

psspy.switched_shunt_chng_4(v_allfxshunts[q],[_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i,_i],[_f+mvar[q],_f,_f,_f,_f,_f
,_f,_f,_f,_f,_f+mvar[q],_f],"")
                #If a switched shunt does not exist on the bus, creates a new one
                    if (v_allswshunts.count(v_allfxshunts[q])==0):
                        psspy.purgshunt(v_allfxshunts[q],r"""1""")

psspy.switched_shunt_data_4(v_allfxshunts[q],[1,_i,_i,_i,_i,_i,_i,_i,0,v_allfxshunts[q],_i,_i,_i],[mvar[q],_f,_f
,_f,_f,_f,_f,_f,_f,_f,mvar[q],_f],"")

                #Gets all switched shunts buses after having included fixed shunt buses and changes their status
to voltage control
                ierr, v_allswshunts = psspy.aswshint(sid, flag, 'NUMBER')
                v_allswshunts = v_allswshunts[0]
                for p in range(len(v_allswshunts)):

psspy.switched_shunt_chng_4(v_allswshunts[p],[_i,_i,_i,_i,_i,_i,_i,_i,1,_i,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f
,_f,_f,_f],"")

                file_name = saved_file[:len(saved_file)-4] + "_h"+str(i+1)+".raw"
                if os.path.exists(file_name):
                    os.remove(file_name)
                #Saves the file into a SAV file in PSSE 32 for it to be used by the OPF
                psspy.writerawversion(r"""32""",0,file_name)

if __name__=="__main__":

    import os, sys

   # The main runs the functions defined above.

#Python_To_PSSE(n_hours,Generators,str_user_files,str_aux_case_folder,df_correspondencia_gen,df_correspondencia_
dem,df_correspondencia_RES)
```

ANEXO 4: FICHERO PYTHON

ANALYSIS_PART_2_OPF

```
# ----------
# Function
# ----------

def
RunOPF(strpathfolder,strfilesbs,strfiletxtout,v_KVRANGE,WINDING_TAP_REGULATION_KV_RANGE,loadwarning,voltupperlim
it,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,Parameters,BRANCH_RATE):

   #PSSE32 initialization and importsof the necessary modules to run the function
   sid = -1 # subsystem identifier
   v_LFoptions = [0,1,0,1,0,1,99,0] # tap, area interchange, phase shift, dc tap adjustment, switched shunt,
flat start, var limit, non-divergence
   import os,sys
   sys.path.append(PSSE_LOCATION)
   os.environ['PATH'] = os.environ['PATH'] + ';' + PSSE_LOCATION
   import psspy
   import redirect
   redirect.psse2py()
   _i=psspy.getdefaultint()
   _f=psspy.getdefaultreal()
   _s=psspy.getdefaultchar()

   # Set and opens subsystem file
   # --------------------------
   strpathfilesbs = strpathfolder + r"\\" + strfilesbs

   #Creates the list where convergence of the file is going to be reported
   convergence = []
   for file in os.listdir(strpathfolder): # go through all .raw files

      #Goes through all .raw files
      if file.endswith(".raw"):

         #Creates case and prepares psse
         strfilefdc = file
         strpathfilefdc = strpathfolder + '\\' + strfilefdc
         psspy.psseinit(20000)
         psspy.read(0,strpathfilefdc)

         # Solve load flow and prepare case
         SBASE = psspy.sysmva()
         ierr,TOLN = psspy.newton_tolerance()
         # check for islands and disconnect them
         isislands = 1
         ierr,busesisland = psspy.tree(1,0) # check for island
         while isislands:
            if busesisland>0:
               ierr,busesisland = psspy.tree(2,1) # disconnect island and check for other island
            else:
               isislands = 0
         psspy.fnsl(v_LFoptions)

         # check convergence of the load flow algorithm - it must converge to be useful
         mismatchmva = psspy.sysmsm()
         ierr, ibus, cmplxmismatch = psspy.maxmsm()
         print(cmplxmismatch)
         mismatchmva = abs(cmplxmismatch)
         isconverged = 1
         if mismatchmva>TOLN:
```

```
            isconverged = 0

        # set subsystem for the analysis of the results only
        psspy.bsysinit(sid)
        psspy.bsys(sid,1,[v_KVRANGE[0], v_KVRANGE[1]],0,[],0,[],0,[],0,[])
        ierr = psspy.bsysrcl(sid,strpathfilesbs)
        psspy.bsysdef(sid, 1)

        # gets all branches and buses data in the previously defined subsystem
        ties=3
        flag=1
        entry=1
        ierr, nallbrnchs = psspy.abrncount(sid, 1, ties, flag, entry)
        ierr, v_allbrnchsFROM = psspy.abrnint(sid, 1, ties, flag, entry, 'FROMNUMBER')
        v_allbrnchsFROM = v_allbrnchsFROM[0]
        ierr, v_allbrnchsTO = psspy.abrnint(sid, 1, ties, flag, entry, 'TONUMBER')
        v_allbrnchsTO = v_allbrnchsTO[0]
        ierr, v_allbrnchsID = psspy.abrnchar(sid, 1, ties, flag, entry, 'ID')
        v_allbrnchsID = v_allbrnchsID[0]
        ierr, nallbuses = psspy.abuscount(sid, flag)
        ierr, v_allbuses = psspy.abusint(sid, flag, 'NUMBER')
        v_allbuses = v_allbuses[0]
        ierr, v_allbuses_type = psspy.abusint(sid, flag, 'TYPE')
        v_allbuses_type = v_allbuses_type[0]


        #Gets all switched shunts buses
        ierr, v_allswshunts = psspy.aswshint(sid, flag, 'NUMBER')
        v_allswshunts = v_allswshunts[0]

        #Gets all loads buses
        ierr, v_allloads = psspy.aloadint(sid, flag, 'NUMBER')
        v_allloads = v_allloads[0]
        ierr, v_allloadsID = psspy.aloadchar(sid, flag, 'ID')
        v_allloadsID = v_allloadsID[0]

        #Gets all generator buses, generator IDs and generators power outputs
        ierr, v_allgens = psspy.amachint(sid, flag, 'NUMBER')
        v_allgens = v_allgens[0]
        ierr, v_allgensID = psspy.amachchar(sid, flag, 'ID')
        v_allgensID = v_allgensID[0]
        ierr, v_pmax = psspy.amachreal(sid, flag, 'PMAX')
        v_pmax = v_pmax[0]
        ierr, v_pmin = psspy.amachreal(sid, flag, 'PMIN')
        v_pmin = v_pmin[0]
        ierr, v_mva = psspy.amachreal(sid, flag, 'MBASE')
        v_mva = v_mva[0]
        ierr, v_qmin = psspy.amachreal(sid, flag, 'QMIN')
        v_qmin = v_qmin[0]

        #Gets all 2-winding transformets data (From Bus, To Bus, ID, control code, From KV, To KV)
        ierr, v_alltwowindingFROM = psspy.atrnint(sid, 1, ties, flag, entry, 'FROMNUMBER')
        v_alltwowindingFROM = v_alltwowindingFROM[0]
        ierr, v_alltwowindingTO = psspy.atrnint(sid, 1, ties, flag, entry, 'TONUMBER')
        v_alltwowindingTO = v_alltwowindingTO[0]
        ierr, v_alltwowindingID = psspy.atrnchar(sid, 1, ties, flag, entry, 'ID')
        v_alltwowindingID = v_alltwowindingID[0]
        ierr, v_alltwowindingCODE = psspy.atrnint(sid, 1, ties, flag, entry, 'CODE')
        v_alltwowindingCODE = v_alltwowindingCODE[0]
        ierr, v_allbusesKVbase = psspy.abusreal(sid, flag, 'BASE')
        v_allbusesKVbase = v_allbusesKVbase[0]

        #Locks transformer tap ratio if it does not meet the requirements (connected to a generation bus or
both buses inside the stated KV range)
        for w in range(len(v_alltwowindingFROM)):
            #GetsKV and bus type data from winding 1 and winding 2 buses
            frombus = v_alltwowindingFROM[w]
            tobus = v_alltwowindingTO[w]
            frombusindex = v_allbuses.index(frombus)
            tobusindex = v_allbuses.index(tobus)
            frombusKV = v_allbusesKVbase[frombusindex]
            tobusKV = v_allbusesKVbase[tobusindex]
            frombustype = v_allbuses_type[frombusindex]
```

```
            tobustype = v_allbuses_type[tobusindex]

            #Requirements
            bus_type_requirement = 0
            from_bus_is_gen_bus = 0
            to_bus_is_gen_bus = 0
            bus_KV_requirement = 0
            from_bus_inside_range = 0
            to_bus_inside_range = 0

            #Defines bus KV requirements
            if((frombusKV >= WINDING_TAP_REGULATION_KV_RANGE[0]) and (frombusKV <=
WINDING_TAP_REGULATION_KV_RANGE[1])):
               from_bus_inside_range = 1
            if((tobusKV >= WINDING_TAP_REGULATION_KV_RANGE[0]) and (tobusKV <=
WINDING_TAP_REGULATION_KV_RANGE[1])):
               to_bus_inside_range = 1
            if((from_bus_inside_range == 1)and(to_bus_inside_range == 1)):
               bus_KV_requirement = 1

            #Defines bus type requirements
            if((frombustype == 2) or (frombustype == 3)):
               from_bus_is_gen_bus = 1
            if((tobustype == 2) or (tobustype == 3)):
               to_bus_is_gen_bus = 1
            if((from_bus_is_gen_bus == 1)or(to_bus_is_gen_bus == 1)):
               bus_type_requirement = 1

            #0 for no control mode, (-code) for control mode (code) WITHOUT auto adjust.
            if(v_alltwowindingCODE[w] != 0):
               code = abs(v_alltwowindingCODE[w])
               if(bus_type_requirement == 1):

psspy.two_winding_data_3(v_alltwowindingFROM[w],v_alltwowindingTO[w],v_alltwowindingID[w],[_i,_i,_i,_i,_i,_i,_i,
_i,_i,_i,_i,code,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],_s)
               elif(bus_KV_requirement == 1):

psspy.two_winding_data_3(v_alltwowindingFROM[w],v_alltwowindingTO[w],v_alltwowindingID[w],[_i,_i,_i,_i,_i,_i,_i,
_i,_i,_i,_i,code,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],_s)
               if((bus_type_requirement == 0)and(bus_KV_requirement == 0)):

psspy.two_winding_data_3(v_alltwowindingFROM[w],v_alltwowindingTO[w],v_alltwowindingID[w],[_i,_i,_i,_i,_i,_i,_i,
_i,_i,_i,_i,-code,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],_s)

         #Obtains branch flows and their rates
         branchesflow = []
         rates = []
         for iline in range(0,nallbrnchs):
            ibus=v_allbrnchsFROM[iline]
            jbus=v_allbrnchsTO[iline]
            idckt=v_allbrnchsID[iline]
            ierr, RATE = psspy.brndat(ibus,jbus,idckt,BRANCH_RATE)
            ierr, APFLOW = psspy.brnmsc(ibus,jbus,idckt,'MVA')
            rates.append(RATE)
            branchesflow.append(APFLOW)

         # Purges previous OPF data, creates the rop file and data for the sav case file
         psspy.purge_all_opf_data()
         psspy.newopf()

         #Sets limits and penalties for bus voltages
         for i in range(len(v_allbuses)):
            bus_voltage_limit_type = 2
            bus_voltage_soft_limit_penalty = 100

psspy.opf_bus_indv(v_allbuses[i],[bus_voltage_limit_type,0],[voltupperlimit,voltlowerlimit,voltupperlimit,voltlo
werlimit,bus_voltage_soft_limit_penalty])


         #Sets limits for adjustable bus shunts controls
         for j in range(len(v_allswshunts)):
            ierr, block1 = psspy.aswshreal(sid, flag, 'BSTPBLOCK1')
            block1 = block1[0][j]
```

```
            ierr, block2 = psspy.aswshreal(sid, flag, 'BSTPBLOCK2')
            block2 = block2[0][j]
            ierr, block3 = psspy.aswshreal(sid, flag, 'BSTPBLOCK3')
            block3 = block3[0][j]
            ierr, initial_sus = psspy.aswshreal(sid, flag, 'BSWACT')
            initial_sus = initial_sus[0][j]
            #Calculates maximum susceptance by summing all the steps
            maxsus =block1+block2+block3
            minsus = 0
            cost = 10
            psspy.opf_adjvar_indv(v_allswshunts[j],r"""1""",[1,_i,0],[initial_sus, maxsus,minsus, cost])


         #Sets adjustable branch reactances. BY DEFAULT all branch reactances are modifyable and the default
parameters are the following:
         max_reactance_multiplier = 1
         min_reactance_multiplier = 1
         cost_scale_coefficient = 1 #$/p.u
         for k in range(len(v_allbrnchsFROM)):

psspy.opf_adjbrx_indv(v_allbrnchsFROM[k],v_allbrnchsTO[k],v_allbrnchsID[k],[1,_i,0],[_f,max_reactance_multiplier
, min_reactance_multiplier,cost_scale_coefficient])



         #Sets branch flows limits in different operating conditions
         for l in range(len(v_allbrnchsFROM)):
            psspy.opf_brflw_brn_indv(v_allbrnchsFROM[l],v_allbrnchsTO[l],v_allbrnchsID[l],r"""1""",[3,1],[
rates[l],_f,rates[l],_f,_f])


         #Sets the adjustable bus load values and table (by default there is only one table for bus loads)
         #Default load multipliers are set to one
         load_multiplier = 1
         max_load_multiplier = 1
         min_load_multiplier = 1
         load_cost_coefficient = 1
         for m in range(len(v_allloads)):
            psspy.opf_load_indv(v_allloads[m],v_allloadsID[m],1)
         psspy.opf_adjload_tbl(1,[1,_i,0],[load_multiplier, max_load_multiplier, min_load_multiplier, 1.0, 1.0,
1.0,load_cost_coefficient])


##          #Sets the generator dispatch and dispatch tables
##             #The following function classifies generators onto a table depending on ther maximum active power
capability (100 to 1500 MW in 100 MW steps for a total of 15 types)
##             #In each generator type a maximum dispatch will be assigned (max MW from table) and a 0.1
multiplier will be assigned for the minimum
##          dispatch_table = 0 #For when this function is not going to be used in OPF
##          min_gen_pwr_multiplier = 0.1
##          for n in range(len(v_allgens)):
##             psspy.opf_gendsp_indv(v_allgens[n],v_allgensID[n],round(v_pmax[n]/100)*dispatch_table,_f)
##
##          for p in range(15):
##             gen_cost_coefficient = 1
##
psspy.opf_apdsp_tbl(p+1,[3,_i,p+1],[(p+1)*100,(p+1)*100*min_gen_pwr_multiplier,gen_cost_coefficient])#The
default cost curve type is set to quadratic and the default cost scale coefficient is set to


##          #Sets the generator reactive capability spreadsheet (maximum reactive absorption will be calcualted
as Qmin/generatorMVAbase
##          default_lagging_pwr_factor = 0.95 #Inductive
##          default_leading_pwr_factor = 0.9 #Capacitive
##          default_stator_current_limit = 1
##          default_Xd = 1.76 #CHANGES TO BE MADE HERE, THE XD HAS TO BE CALCULATED FOR EACH ONE OF THE
GENERATORS
##          for q in range(len(v_allgens)):
##             max_qgen_absorption = v_qmin[q]/v_mva[q]
##             psspy.opf_gen_rcap_indv(v_allgens[q],v_allgensID[q],4,[default_Xd,default_stator_current_limit,
default_lagging_pwr_factor, default_leading_pwr_factor, max_qgen_absorption])
##             #Fixed Edf by default
```

```
        #Generator reserve spreadsheet not necessary / Power-Frequency dynamics not necessary for OPF

        #The OPF tolerances and control are the default suggested by PSSE
        if (Parameters[0] == 1):
           psspy.minimize_p_slack(1) #Minimize active power slack generation
        if (Parameters[1] == 1):
           psspy.minimize_q_slack(1) #Minimize reactive power slack generation
        if (Parameters[2] == 1):
           psspy.minimize_p_losses(1) #Minimize active power losses
        if (Parameters[3] == 1):
           psspy.minimize_q_losses(1) #Minimize reactive power losses
        if (Parameters[4] == 1):
           psspy.minimize_series_comp(1) #Minimize adjustable branch reactances
        if (Parameters[5] == 1):
           psspy.minimize_adj_bus_shunts(1) #Minimize adjustable bus shunts
        if (Parameters[6] == 1):
           psspy.minimize_load_adjustments(1) #Minimize adjustable bus loads
        if (Parameters[7] == 1):
           psspy.minimize_interface_flows(1) #Minimize interface flows
        if (Parameters[8] == 1):
           psspy.minimize_reactive_reserve(1) #Minimize reactive generation reserve
        if (Parameters[9] == 1):
           psspy.opf_regulate_area_int(1) #Regulate area interchange
        if (Parameters[10] == 1):
           psspy.constrain_interface_flows(1) #Constrain interface flows
        if (Parameters[11] == 1):
           psspy.apply_automatic_scaling(1) #Use automatic scaling
        if (Parameters[12] == 1):
           psspy.use_dual_criteria(1) #Use dual variable convergence criteria
        if (Parameters[13] == 1):
           psspy.opf_fix_tap_ratios(1) #Fix transformer tap ratios
        if (Parameters[14] == 1):
           psspy.opf_fix_phase_shifters(1) #Fix transformer phase shift angles
        if (Parameters[15] == 1):
           psspy.opf_fix_switched_shunts(1) #Fix switched shunts
        if (Parameters[16] == 1):
           psspy.opf_round_tap_ratios(1) #Round transformer tap ratios
        if (Parameters[17] == 1):
           psspy.opf_round_switched_shunts(1) #Round switched shunt vars
        if (Parameters[18] == 1):
           psspy.open_bus_voltage_limits(1) #Automatically adjust bus voltages for feasibility
        if (Parameters[19] == 1):
           psspy.use_emergency_volt_limits(1) #Impose emergency bus voltage limits
        if (Parameters[20] == 1):
           psspy.use_emergency_flow_limits(1) #Impose emergency branch flow limits

        #Sets output parameters
        psspy.solution_parameters_3([_i,50,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f]) #Set
the maximum iterations to 50
        psspy.opf_use_generator_vsched(1) #Starts the iterations with generator scheduled voltages
        psspy.produce_opf_log_file(1,"OPF_parameters_results")#Produces an OPF parameters output file
        psspy.add_details_to_opf_log(0)#Adds details to the OPF parameters file produced
        psspy.set_opf_report_subsystem(3,1)#Sets the reporting subsystem
        #psspy.write_opf_options_file() #Writes an OPT file with the OPF options selected


        #Runs the OPF
        psspy.nopf(0,1)

        #Checks for convergence and outputs .sav files with the optimal solution
        mismatchmva = psspy.sysmsm()
        ierr, ibus, cmplxmismatch = psspy.maxmsm()
        mismatchmva = abs(cmplxmismatch)
        isconverged = 1
        if mismatchmva>TOLN:
           isconverged = 0
           hour = strpathfilefdc[:len(strpathfilefdc)-4]
           convergence.append(hour[-4:]+":"+str(0))
        if isconverged == 1:
           hour = strpathfilefdc[:len(strpathfilefdc)-4]
           convergence.append(hour[-4:]+":"+str(1))
           #Produces output file with grid details and changes
           filename = strpathfolder + r"\\" +strfilefdc + strfiletxtout
```

```
            #psspy.report_output(2,strfilefdc + strfiletxtout,2)
         else:
            print("OPF CASE NOT CONVERGED, PLEASE CHECK THE OPF PARAMETERS")

         file_name = strpathfilefdc[:len(strpathfilefdc)-4] + '.sav'
         if os.path.exists(file_name):
            os.remove(file_name)
         psspy.save(strpathfilefdc[:len(strpathfilefdc)-4] + '.sav')

   return convergence

# ------------------------------
# Main - runs if .py is executed
# ------------------------------

if __name__=="__main__":

   import os, sys

   # ---------------------------------------
   # PARAMETERS TO BE DEFINED BY THE USER
   # ---------------------------------------

   #Introduce the KV range you want to evaluate. IN PART 2 INCLUDE ALL OF THE DIFFERENT KV IN THE SYSTEM
   v_KVRANGE = [1,400]
   #Introduce the KV range for the transformer tap regulation
   WINDING_TAP_REGULATION_KV_RANGE = [100,400]
   #Introduce the load warning threshold as a percentage of the load limit
   loadwarning = 90
   #Introduce voltage upper and lower limits in p.u
   voltupperlimit = 1.1
   voltlowerlimit = 0.9
   #Introduce the rate to be used for load analysis of the branches (e.g: 'RATEA', 'RATEB')
   BRANCH_RATE = 'RATEA'
   #Introduce the PSSE version (32 for OPF)
   ISPSSEVERSION=32
   #Introduce the location of PSSE (PSSE32 in this case)
   PSSE_LOCATION = r"C:\Program Files (x86)\PTI\PSSE32\PSSBIN"
   #Introduce the path of the folder where the files to be evaluated are located
   strpathfolder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
   strfilesbs = r"CasoPeninsula220400.sbsxml"

   #The OPF tolerances and control are the default suggested by PSSE
   #Introduce the parameters to minimize in the OPF (insert a 1 in the list if that parameter should be
minimized/regulated, a 0 if not).
      #1)Minimize active power slack generation
      #2)Minimize reactive power slack generation
      #3)Minimize active power losses
      #4)Minimize reactive power losses
      #5)Minimize adjustable branch reactances
      #6)Minimize adjustable bus shunts
      #7)Minimize adjustable bus loads
      #8)Minimize interface flows
      #9)Minimize reactive generation reserve
      #10)Regulate area interchange
      #11)Constrain interface flows
      #12)Use automatic scaling
      #13)Use dual variable convergence criteria
      #14)Fix transformer tap ratios
      #15)Fix transformer phase shift angles
      #16)Fix switched shunts
      #17)Round transformer tap ratios
      #18)Round switched shunt vars
      #19)Automatically adjust bus voltages for feasibility
      #20)Impose emergency bus voltage limits
      #21)Impose emergency branch flow limits
   Parameters = [0,0,1,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0]

   ####################################################
   #FILE REQUIREMENTS FOR THIS SCRIPT:
   #           1)The python script will have to be located in the "Scripts" folder. THIS SCRIPT IS EXECUTED
AFTER THE ANALYSIS_PART_1 script
```

```
   #           2)The .raw file(s) generated in the ANALYSIS_PART_1 will need to be run in PSSE32 (the file can
be saved in that version)
   #           3)The OPF results for each hour are exported onto a .sav file (PSSE32, can be opened by 34) in
the folder where the scripts are
   #           4)A file called OPF_Parameter_Results is generated. It can be ignored
   #The .sav files are generated on the same folder as the .raw files, and are ready to be analyzed by
ANALYSIS_PART_3
   ######################################################

   convergence =
RunOPF(strpathfolder,strfilesbs,r"OPFresults.dat",v_KVRANGE,WINDING_TAP_REGULATION_KV_RANGE,loadwarning,voltuppe
rlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,Parameters,BRANCH_RATE)
   print(convergence)
```

ANEXO 5: FICHERO PYTHON

ANALYSIS_PART_3_TESTS

```
# ----------
# -*- coding: cp1252 -*-
#Initialization parameters

# ----------------------------------------
# Program
# ----------------------------------------
def
RUN_ANALYSIS(str_report_folder,str_report_folder_static,str_aux_case_folder,file_correspondence,str_user_files,n
_hours,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,BRANCH_RATE,strdynfile,FB
ASE,tfin_gen,tfin_bus,FUF1,str_aux_dyn_folder,dynamic_gen_parameters,str_report_folder_dynamic,dynamic_bus_param
eters,sc_buses,max_angle,min_angle,max_angle_ratio):
    import os,sys
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    import matplotlib.pyplot as plt
    #from docx.text.parargaph import Paragraph
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import collections
    #import dyntools

    #Imports the other scripts
    import StaticAnalysisLibrary
    import MainSimulateTrippings

    #Opens the PSSE-DESI Correspondence file
    xls = pd.ExcelFile(file_correspondence)
    datos_hoja=xls.parse('GeneracionConvencional')
    df_correspondencia=datos_hoja[["Nudo","Nombre","PGen (MW)","PMax (MW)","PMin (MW)","QGen (Mvar)","QMax
(Mvar)","QMin (Mvar)","Mbase (MVA)","Correspondencia DESI","Comentario"]]

    #Creates the  reports folder if it does not exist.
    if not os.path.exists(str_report_folder):
        os.makedirs(str_report_folder)

    #Creates the static reports folder if it does not exist.
    if not os.path.exists(str_report_folder_static):
        os.makedirs(str_report_folder_static)

    #Runs the Static Analytic Library Analysis script (Contingency analysis)
    StaticAnalysisLibrary.RunLF(str_report_folder_static,str_aux_case_folder,v_KVRANGE, loadwarning,
voltupperlimit, voltlowerlimit, ISPSSEVERSION, PSSE_LOCATION,BRANCH_RATE)
    StaticAnalysisLibrary.RunStaticN_1N_2branches(str_report_folder_static,str_aux_case_folder,v_KVRANGE,
loadwarning, voltupperlimit, voltlowerlimit, ISPSSEVERSION, PSSE_LOCATION,BRANCH_RATE)

    #Creates the dynamic reports folder if it does not exist.
    if not os.path.exists(str_report_folder_dynamic):
        os.makedirs(str_report_folder_dynamic)

    #Runs the dynamics simulations (Generator Trippings and Short-Circuits)
```

```
    MainSimulateTrippings.RunDynamicGenTrippings(strdynfile,FBASE,tfin_gen,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_gen_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE,max_angle,min_angle,max_angle_ratio)
    MainSimulateTrippings.RunDynamicShortCircuits(strdynfile,FBASE,tfin_bus,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_bus_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE,sc_buses,max_angle,min_angle,max_angle_ratio)

if __name__=="__main__":

   # The main runs the functions defined above.

    import os, sys

# ---------------------------------------
# PARAMETERS TO BE DEFINED BY THE USER FOR THE STATIC ANALYSIS FUNCTION
# ---------------------------------------
    #THE INPUT FILES (.txt generation, correspondence file, .sav file) SHOULD BE INCLUDED ALL IN THE SAME FOLDER
    #Introduce the path of the file that has the correspondence between DESI and PSSE (SAVED AS .XLS)
    file_correspondence = r"C:\Users\proyectista\Desktop\Scripts\User_data_input\CorrespondenciaPSSE-
DESI_convencional.xls"
    #Introduce the folder where the .sav files inputted by the user are located
    str_user_files = r"C:\Users\proyectista\Desktop\Scripts\User_data_input"
    #Introduce the path of the folder where the .sav files to be evaluated are localized
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the path of the folder of the "Reports" folder
    str_report_folder = r"C:\Users\proyectista\Desktop\Scripts\Reports"
    #Introduce the path of the folder where the reports are going to be generated
    str_report_folder_static = r"C:\Users\proyectista\Desktop\Scripts\Reports\Static_Analysis"
    #Introduce the number of hours evaluated (Daily execution-24, Weekly execution-168)
    n_hours = 24
    #Introduce the KV range you want to evaluate.
    #Since transformer loads are evaluated, please specify a range in which no transformers have a bus out of
range and a bus inside range, an error will occur
    #Generator trippings are evaluated in the contingency analysis, please make sure that all generators' buses
are included in the kV range
    v_KVRANGE = [1,500]
    #Introduce the load warning threshold as a percentage of the load limit
    loadwarning = 90
    #Introduce the rate to be used for load analysis of the branches (e.g: 'RATEA', 'RATEB',...)
    BRANCH_RATE = 'RATEA'
    #Introduce the voltage upper and lower limits in p.u
    voltupperlimit = 1.12
    voltlowerlimit = 0.95
    #Introduce the version of PSSE and where it is located on your computer
    ISPSSEVERSION=34
    PSSE_LOCATION = r"C:\Program Files (x86)\PTI\PSSE34\PSSPY27"

   ######################################################
   #FILE REQUIREMENTS FOR THIS SCRIPT:
   #           1)The python script and the .sav file(s) have to be placed at the same folder level
   #           2)A folder called "Images-Static" will be created at the same level as the report folder
   #           3)Due to excel sheet requirements, names of the .sav files MUST be kept short
   #The output Word and Excel files (for each hour one Word and Excel for each analysis (GenTrip, BusSC) will be
generated in a folder inside the "Reports" folder called "Static_Analysis"
   #If an excel file for one hour is NOT GENERATED, it means that there is NO VIOLATIONS for any contingency at
that hour
   ######################################################

    # ---------------------------------------
    # PARAMETERS TO BE DEFINED BY THE USER FOR THE DYNAMIC ANALYSIS FUNCTION
    # ---------------------------------------
    #Choose the path of the dynamic file (.dyr) to be evaluated
    strdynfile =
r"C:\Users\proyectista\Desktop\Scripts\User_data_input\Tenerife_DinamicoEndesa_Incidente_Inicial.dyr"
    #Introduce the path of the folder where the .sav files to be evaluated are localized
    str_aux_case_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Case_Data"
    #Introduce the path of the folder where the auxiliary "Dynamic_Files" folder will be created
    str_aux_dyn_folder = r"C:\Users\proyectista\Desktop\Scripts\Scripts\Analysis_Data\Dynamic_Files"
    #Introduce the path of the folder where the reports are going to be generated
    str_report_folder_dynamic = r"C:\Users\proyectista\Desktop\Scripts\Reports\Dynamic_Analysis"
    #Choose base frequency (Hz)
    FBASE = 50.0
    #The tripping of the generator is done at t=1s, choose the time at which the simulation ends
```

```
    tfin_gen = 30.0
    #The short-circuit at the bus is generated at t=1s, choose the time at which the simulation ends
    tfin_bus = 10.0
    #Choose the frequency warning threshold for generator trippings
    FUF1 = 48.5
    #Choose the positive generator angle warning threshold for short circuits
    max_angle = 50
    #Choose the negative generator angle warning threshold for short circuits
    min_angle = -50
    #Choose the warning treshold for the relationship between the peak and steady-state value of the maximum and
minimum generation angles in the system
    #Max_angle_ratio = max or min_dynamic_angle / steady-state max or min angle
    max_angle_ratio = 1.5
    #Include in the following list the bus numbers in which the short circuits will be triggered
    sc_buses = [11,16,419]
    #Select the parameters to plot in the dynamic generator tripping analysis in the following list (1 yes, 0
no)
    #1)Plot Electrical Power for each generator
    #2)Plot Mechanical Power for each generator
    #3)System frequency. The data will be extracted from the speed of one of the generators. Recommended to
include it
    #4)Plot System totals (PELEC, PMECH, PACCL, PLOAD, PE-PL)
    dynamic_gen_parameters = [1,1,1,1]
    #Select the parameters to plot in the dynamic bus tripping analysis in the following list (1 yes, 0 no)
    #1)Plot Voltage angle for each generator
    #2)Plot Voltage (p.u) for each generator
    #3)System frequency. The data will be extracted from the speed of one of the generators. Recommended to
include it
    #4)MANDATORY FOR IT TO BE ACTIVATED. Evaluate machine angle statistics (Average angle, largest angle, bus
with largest angle, smallest angle, bus with smallest angle, angle spread)
    dynamic_bus_parameters = [1,1,1,1]
    #Please, keep in mind that if other different variables want to be exported, the function GetDynResults will
have to be slightly modified to accomodate the new variables
    ######################################################
    #FILE REQUIREMENTS FOR THIS SCRIPT:
    #           1)The .sav and .dyr files will need to be placed into the "User_data_input" folder
    #           2)A folder called "Images-Dynamics" will be created inside the "Reports" folder in order to
accomodate all the images
    #           3)A folder called "Dynamic_Files" will be created in the Scripts-->Analysis_Data folder in order
to accomodate some dynamics files
    #The output Word and Excel documents, among othe PSSE output files will be generated inside the "Reports"
folder in a folder called "Dynamic_Analysis"."Sheet1" in the N-1Gen Excel can be ignored
    ######################################################

    #Runs a complete analysis

RUN_ANALYSIS(str_report_folder,str_report_folder_static,str_aux_case_folder,file_correspondence,str_user_files,n
_hours,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PSSE_LOCATION,BRANCH_RATE,strdynfile,FB
ASE,tfin_gen,tfin_bus,FUF1,str_aux_dyn_folder,dynamic_gen_parameters,str_report_folder_dynamic,dynamic_bus_param
eters,sc_buses,max_angle,min_angle,max_angle_ratio)
```

ANEXO 6: FICHERO PYTHON

STATIC_ANALYSIS_LIBRARY

```
# ----------
# Functions
# ----------

def
RunLF(str_report_folder,str_aux_case_folder,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimit,ISPSSEVERSION,PS
SE_LOCATION,BRANCH_RATE):

   # This function runs load flows and
   # checks whether lines in a defined subsystem suffer
   # overloads.
   #
   # Inputs:   Path to folder where load flow files .sav
   #           Subsystem file .sbs
   #           File name of results output .dat
   #           v_KVRANGE e.g., [220, 400]
   #                 Loadwarning value
   # Output:   Word file with loads and voltages

   #Defines some parameters
   sid = 0 # subsystem identifier
   v_LFoptions = [0,1,0,1,0,0,99,0] # tap, area interchange, phase shift, dc tap adjustment, switched shunt,
flat start, var limit, non-divergence
   contingency = 0

   #Imports all the necessary functions to run the function
   import os,sys
   import matplotlib.pyplot as plt
   import pandas as pd
   import docx
   from docx import Document
   from docx.shared import Cm, Pt
   #from docx.text.parargaph import Paragraph
   from docx.styles.style import WD_STYLE_TYPE
   from docx.enum.table import WD_TABLE_ALIGNMENT
   from docx.enum.table import WD_TABLE_DIRECTION
   from docx.enum.text import WD_ALIGN_PARAGRAPH
   import csv
   import openpyxl
   import fileinput
   import xlsxwriter
   import numpy
   import xlrd, xlwt
   from openpyxl import load_workbook

   # Prepares PSS/e to be executed stand-alone
   # ----------------------------------------
   if ISPSSEVERSION==34:
      import psse34              # it sets new path for psspy
   else:
       sys.path.append(PSSE_LOCATION)
       os.environ['PATH'] = os.environ['PATH'] + ';' + PSSE_LOCATION
   import psspy
   import redirect
   redirect.psse2py()

   #Defines some parameters useful for PSSE usage
   _i=psspy.getdefaultint()
   _f=psspy.getdefaultreal()
```

```
_s=psspy.getdefaultchar()

word=0 #Used to correctly plot all the graphs onto the word document
excel=0 #Used for ocrrectly opening and closing excel sheets

#Creates the "Images-Static" folder if it does not exist
strpathresultimages = str_report_folder + r"\Images-Static"
if not os.path.exists(strpathresultimages):
   os.makedirs(strpathresultimages)

for file in os.listdir(str_aux_case_folder): # go through all .sav files

   if file.endswith(".sav"):
      strfilefdc = file

      strpathfilefdc = str_aux_case_folder + '\\' + strfilefdc

      psspy.psseinit(20000) # opens PSS/e and the case in the .sav file
      psspy.case(strpathfilefdc)
      # psspy.read(0,strpathfilefdc) # if .raw file

      # Solve load flow and prepare case
      # --------------------------------
      SBASE = psspy.sysmva()
      ierr,TOLN = psspy.newton_tolerance()

      # check for islands and disconnect them
      isislands = 1
      ierr,busesisland = psspy.tree(1,0) # check for island
      while isislands:
         if busesisland>0:
            ierr,busesisland = psspy.tree(2,1) # disconnect island and check for other island
         else:
            isislands = 0

      # psspy.fdns(v_LFoptions)
      psspy.fnsl(v_LFoptions)

      # get swing bus (only peninsula = first entry, second one is Balearic Islands)
      treeobj = psspy.treedat(20)
      l_swingbus = treeobj['swing_busnum']
      swingbus = l_swingbus[0]


      # check convergence of the load flow algorithm - it must converge to be useful
      mismatchmva = psspy.sysmsm()
      ierr, ibus, cmplxmismatch = psspy.maxmsm()
      mismatchmva = abs(cmplxmismatch)
      isconverged = 1
      if mismatchmva>TOLN:
         isconverged = 0

      # Analysis of the results
      # -----------------------
      # set subsystem for the analysis of the results only
      psspy.bsysinit(sid)
      psspy.bsys(sid,1,[v_KVRANGE[0], v_KVRANGE[1]],0,[],0,[],0,[],0,[])
      #ierr = psspy.bsysrcl(sid,strpathfilesbs)
      psspy.bsysdef(sid, 1)

      #Print an alphabetically sorted table of all buses in the system.
      psspy.alph(sid,0)

      # get all branches and buses in the previously defined subsystem
      ties=3
      branchflag=3 #Both transformers and branches
      busflag = 1 #Only in service buses
      entry=1
      ierr, nallbrnchs = psspy.abrncount(sid, 1, ties, branchflag, entry)
      ierr, v_allbrnchsFROM = psspy.abrnint(sid, 1, ties, branchflag, entry, 'FROMNUMBER')
      v_allbrnchsFROM = v_allbrnchsFROM[0]
      ierr, v_allbrnchsTO = psspy.abrnint(sid, 1, ties, branchflag, entry, 'TONUMBER')
      v_allbrnchsTO = v_allbrnchsTO[0]
```

```
        ierr, v_allbrnchsID = psspy.abrnchar(sid, 1, ties, branchflag, entry, 'ID')
        v_allbrnchsID = v_allbrnchsID[0]

        ierr, nallbuses = psspy.abuscount(sid, busflag)
        ierr, v_allbuses = psspy.abusint(sid, busflag, 'NUMBER')
        v_allbuses = v_allbuses[0]

        # Extract line flows and writes them into the file
        branchesflow = []
        rates = []
        for iline in range(0,nallbrnchs):
           ibus=v_allbrnchsFROM[iline]
           jbus=v_allbrnchsTO[iline]
           idckt=v_allbrnchsID[iline]
           ierr, RATE = psspy.brndat(ibus,jbus,idckt,BRANCH_RATE)
           ierr, APFLOW = psspy.brnmsc(ibus,jbus,idckt,'MVA')
           rates.append(RATE)
           branchesflow.append(APFLOW)

        # Extract bus voltage magnitudes and writes them into the file
        v_busvoltages = []
        busesKV = []
        for ibus in range(0,nallbuses):
           bus = v_allbuses[ibus]
           ierr, vbus = psspy.busdat(bus ,'PU')
           ierr, busKV = psspy.busdat(bus,'BASE')
           v_busvoltages.append(vbus)
           busesKV.append(busKV)
           loadlimits = 1

        #Create a sorted list of all different bus voltages evaluated on the case
        busesKVint = []
        for a in busesKV:
           busesKVint.append(int(a))
        busesKVint = list(dict.fromkeys(busesKVint))
        print(busesKVint)

        #Code to create the Word document
        if (word==0):
           document = Document()
           document.add_heading('LOAD FLOW RESULTS '+strfilefdc, 0)
        else:
           document.add_page_break()
           document.add_heading('LOAD FLOW RESULTS '+strfilefdc, 0)

        #Code to create the Excel document
        if (excel==0):
           excel_file_erase = str_report_folder + "\\" + "ResultsOPF.xlsx"
           if os.path.exists(excel_file_erase):
              os.remove(excel_file_erase)
           excel_file = str_report_folder + "\\" + "ResultsOPF.xlsx"
           writer = pd.ExcelWriter(excel_file, engine = 'xlsxwriter')
        else:
           print(" ")

        #Runs the function
        if(isconverged ==1):
        df_loads,df_voltages = GetLFResults(contingency, busesKV, busesKVint, document, voltupperlimit,
voltlowerlimit, loadwarning, strfilefdc, str_report_folder, v_allbrnchsFROM, rates, branchesflow, v_allbrnchsTO,
v_allbrnchsID, v_allbuses, v_busvoltages,excel_file,writer)
        else:
        document.add_paragraph('CASE NOT CONVERGED, PLEASE CHECK LOAD FLOW PARAMETERS')

        #Code to save the documents (deletes the doc document if it already exists and creates a new one)
        file_name_doc = str_report_folder+'\ResultsOPF.docx'
        if os.path.exists(file_name_doc):
           os.remove(file_name_doc)
        if(word==0):
           print(' ')
           #Saves the Word document
           document.save(file_name_doc)
           #Saves the Excel Document
           with pd.ExcelWriter(excel_file, engine='xlsxwriter') as writer:
```

```
                df_loads.to_excel(writer,sheet_name=strfilefdc[:len(strfilefdc)-4]+"MVA",index=False)
                df_voltages.to_excel(writer,sheet_name=strfilefdc[:len(strfilefdc)-4]+"V",index=False)
                writer.close()

            else:
               #Saves the Word document
               document.save(file_name_doc)
               #Saves the Excel document
               book = load_workbook(excel_file)
               with pd.ExcelWriter(excel_file, engine='openpyxl') as writer:
                  writer.book = book
                  writer.sheets = dict((ws.title, ws) for ws in book.worksheets)
                  df_loads.to_excel(writer,sheet_name=strfilefdc[:len(strfilefdc)-4]+"MVA",index=False)
                  df_voltages.to_excel(writer,sheet_name=strfilefdc[:len(strfilefdc)-4]+"V",index=False)
                  writer.save()
            word+=1
            excel+=1



def
RunStaticN_1N_2branches(str_report_folder,str_aux_case_folder,v_KVRANGE,loadwarning,voltupperlimit,voltlowerlimi
t,ISPSSEVERSION,PSSE_LOCATION,BRANCH_RATE):

   # This function applies N-1 and N-2 contingencies to branches and
   # checks whether lines in a defined subsystem suffer overloads and voltages in buses are out of range.

   #THIS FUNCTION ONLY DISPLAYS VOLTAGES AND FLOWS THAT ARE OUT OF RANGE, NOT ALL LOAD FLOW RESULTS

   #Defines some parameters
   sid=0
   v_LFoptions = [0,1,0,1,1,0,99,0] # tap, area interchange, phase shift, dc tap adjustment, switched shunt
enabled, flat start, var limit, non-divergence
   contingency = 1

   #Imports all the necessary functions to run the function
   import os,sys
   import matplotlib.pyplot as plt
   import pandas as pd
   import docx
   import random
   from docx import Document
   from docx.shared import Cm, Pt
   from docx.styles.style import WD_STYLE_TYPE
   from docx.enum.table import WD_TABLE_ALIGNMENT
   from docx.enum.table import WD_TABLE_DIRECTION
   from docx.enum.text import WD_ALIGN_PARAGRAPH
   import csv
   import openpyxl
   import fileinput
   import xlsxwriter
   import numpy
   import xlrd, xlwt
   from openpyxl import load_workbook

   #Imports and prepares PSSE
   if ISPSSEVERSION==34:
      import psse34              # it sets new path for psspy
   else:
       sys.path.append(PSSE_LOCATION)
       os.environ['PATH'] = os.environ['PATH'] + ';' + PSSE_LOCATION
   import psspy
   import redirect
   redirect.psse2py()
   import dyntools

   #Defines some parameters useful for durther PSSE usage
   _i=psspy.getdefaultint()
   _f=psspy.getdefaultreal()
   _s=psspy.getdefaultchar()

   word=0 #Used to correctly plot all the graphs onto the word document
   excel=0 #Used for ocrrectly opening and closing excel sheets
```

```
   #Creates the "Images-Static" folder if it does not exist
   strpathresultimages = str_report_folder + r"\Images-Static"
   if not os.path.exists(strpathresultimages):
      os.makedirs(strpathresultimages)

   for file in os.listdir(str_aux_case_folder): # go through all .sav files

      if file.endswith(".sav"):

         strfilefdc = file
         strpathfilefdc = str_aux_case_folder + '\\' + strfilefdc

         psspy.psseinit(20000)
         psspy.case(strpathfilefdc)


         # Solve load flow and prepare case
         # --------------------------------
         ierr,TOLN = psspy.newton_tolerance()
         SBASE = psspy.sysmva()

         # psspy.fdns(v_LFoptions)
         psspy.fnsl(v_LFoptions)

         # get swing bus (only peninsula = first entry, second one is Balearic Islands)
         treeobj = psspy.treedat(20)
         l_swingbus = treeobj['swing_busnum']
         swingbus = l_swingbus[0]


         # check convergence
         #Obtain mismatch
         mismatchmva = psspy.sysmsm()
         #Obtain bus mismatch at the bus with the largest MVA mismatch
         ierr, ibus, cmplxmismatch = psspy.maxmsm()
         mismatchmva = abs(cmplxmismatch)
         isconverged = 1
         if mismatchmva>TOLN:
            isconverged = 0

         # set subsystem
         psspy.bsysinit(sid)
         psspy.bsys(sid,1,[v_KVRANGE[0], v_KVRANGE[1]],0,[],0,[],0,[],0,[])
         #ierr = psspy.bsysrcl(sid,strpathfilesbs)
         psspy.bsysdef(sid, 1)


         #Print an alphabetically sorted table of all buses in the system.
         psspy.alph(sid,0)

         #Returns arrays of subsystem elements
         ties=3 #for both interior subsystem branches and tie branches.
         branchflag=3 #for only in-service transformer and non-transformer branches.
         busflag=1 #for only in-service buses.
         generatorflag=1 #for only in-service generators.
         entry=1 #for single entry (each branch once).
         #Return the number of array entries required to accommodate the data to be returned by the remaining
members of the branch data family.
         ierr, nallbrnchs = psspy.abrncount(sid, 1, ties, branchflag, entry)
         #Return an array of integer values for subsystem branches (from, to and ID)
         ierr, v_allbrnchsFROM = psspy.abrnint(sid, 1, ties, branchflag, entry, 'FROMNUMBER')
         v_allbrnchsFROM = v_allbrnchsFROM[0]
         ierr, v_allbrnchsTO = psspy.abrnint(sid, 1, ties, branchflag, entry, 'TONUMBER')
         v_allbrnchsTO = v_allbrnchsTO[0]
         ierr, v_allbrnchsID = psspy.abrnchar(sid, 1, ties, branchflag, entry, 'ID')
         v_allbrnchsID = v_allbrnchsID[0]

         #Number of buses in the subsystem SID that meet the flag requirements
         ierr, nallbuses = psspy.abuscount(sid, busflag)
         #Voltage of those buses
         ierr, v_allbuses = psspy.abusint(sid, busflag, 'NUMBER')
         v_allbuses = v_allbuses[0]
```

```
        #Returns an array of buses in which machines are located
        ierr, v_allgens = psspy.amachint(sid, generatorflag, 'NUMBER')
        v_allgens= v_allgens[0]
        ierr, v_allgensID = psspy.amachchar(sid, generatorflag, 'ID')
        v_allgensID = v_allgensID[0]

        #Code to create the Word document
        if (word==0):
              document = Document()
        else:
              document.add_page_break()

        #Code to create the Excel document
        if(excel==0):
           excel_file_erase = str_report_folder + "\\" + "ResultsN-1N-2" + strfilefdc[:len(strfilefdc)-4] +
".xlsx"
           if os.path.exists(excel_file_erase):
              os.remove(excel_file_erase)
           excel_file = str_report_folder + "\\" + "ResultsN-1N-2" + strfilefdc[:len(strfilefdc)-4] + ".xlsx"
           writer2 = pd.ExcelWriter(excel_file, engine = 'xlsxwriter')

        # N-1 Branch trippings
        for iline in range(0,nallbrnchs):
           #Creates vectors to export data onto word document
           cont_v_allbrnchsFROM = []
           cont_v_allbrnchsTO = []
           cont_v_allbrnchsID = []
           cont_rates = []
           cont_branchesflow = []
           cont_v_allbuses = []
           cont_v_busvoltages = []
           busesKV = []
           busesKVint = []

           #Gests contigency line info
           ibus=v_allbrnchsFROM[iline]
           jbus=v_allbrnchsTO[iline]
           idckt=v_allbrnchsID[iline]

           #Switches off branch, runs LF
           psspy.branch_data(ibus,jbus,idckt,[0,_i,_i,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
           psspy.fnsl([0,0,0,1,1,0,99,0])
           ierr, ibusmax, cmplxmismatch = psspy.maxmsm()
           mismatchmva = abs(cmplxmismatch)
           isconverged = 1
           if mismatchmva>TOLN:
              isconverged = 0

           #If it is converged, gets contigency data
           if isconverged:

              #Gets data from other lines
              v_remaininglines = range(0,iline)+range(iline+1,nallbrnchs)
              for ilineremain in v_remaininglines:
                 branchesflow = []
                 rates = []
                 ibusremain=v_allbrnchsFROM[ilineremain]
                 jbusremain=v_allbrnchsTO[ilineremain]
                 idcktremain=v_allbrnchsID[ilineremain]
                 ierr, RATE = psspy.brndat(ibusremain,jbusremain,idcktremain,BRANCH_RATE)
                 ierr, APFLOW = psspy.brnmsc(ibusremain,jbusremain,idcktremain,'MVA')
                 rates.append(RATE)
                 branchesflow.append(APFLOW)
                 ierr, Vi = psspy.busdat(ibusremain ,'PU')
                 ierr, Vj = psspy.busdat(jbusremain ,'PU')

                 #Checks if there are overloads
                 if ((APFLOW>RATE) and (RATE > 0)):
                    cont_v_allbrnchsFROM.append(v_allbrnchsFROM[ilineremain])
                    cont_v_allbrnchsTO.append(v_allbrnchsTO[ilineremain])
                    cont_v_allbrnchsID.append(v_allbrnchsID[ilineremain])
                    cont_rates.append(RATE)
```

```
                    cont_branchesflow.append(APFLOW)

                #Gets data from buses, checks if voltage is ok
                for ibuss in range(len(v_allbuses)):
                   bus = v_allbuses[ibuss]
                   ierr, vbus = psspy.busdat(bus ,'PU')
                   if((vbus>voltupperlimit) or (vbus<voltlowerlimit)):
                      cont_v_allbuses.append(v_allbuses[ibuss])
                      cont_v_busvoltages.append(vbus)

             #Switches branch on, opens case again without saving to restart OPF conditions
             psspy.branch_data(ibus,jbus,idckt,[1,_i,_i,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
             psspy.case(strpathfilefdc)

             #Exports data to word document if there is a contigency.
             if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0)):
                document.add_heading('N-1 Contingency results branch:'+str(v_allbrnchsFROM[iline])+'-
'+str(v_allbrnchsTO[iline])+'-'+str(v_allbrnchsID[iline])+' '+strfilefdc, 0)
                df_loads,df_voltages = GetLFResults(contingency, busesKV, busesKVint, document, voltupperlimit,
voltlowerlimit, loadwarning, strfilefdc, str_report_folder, cont_v_allbrnchsFROM, cont_rates, cont_branchesflow,
cont_v_allbrnchsTO, cont_v_allbrnchsID, cont_v_allbuses, cont_v_busvoltages,excel_file,writer2)
                document.add_page_break()

             if(isconverged == 0):
                document.add_heading('N-1 Contingency results branch:'+str(v_allbrnchsFROM[iline])+'-
'+str(v_allbrnchsTO[iline])+'-'+str(v_allbrnchsID[iline])+' '+strfilefdc, 0)
                document.add_paragraph('CASE NOT CONVERGED, PLEASE CHECK LOAD FLOW PARAMETERS')
                document.add_page_break()

             #Gives a message in the excel in the case that it does not converge
             if(isconverged == 0):
                df_voltages = pd.DataFrame(["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"],index = [1],columns =
["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"])
                df_loads = pd.DataFrame(["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"],index = [1],columns =
["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"])

             #Saves the dataframes onto the Excel document
             if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0) or (isconverged == 0)):
                if (excel==0):
                   #Saves the Excel Document
                   with pd.ExcelWriter(excel_file, engine='xlsxwriter') as writer2:
                      if(df_loads.empty == False):
                         df_loads.to_excel(writer2,sheet_name="N-1|"+str(ibus)+"-"+str(jbus)+"-
"+str(idckt)+"MVA",index=False)
                      if(df_voltages.empty == False):
                         df_voltages.to_excel(writer2,sheet_name="N-1|"+str(ibus)+"-"+str(jbus)+"-
"+str(idckt)+"V",index=False)
                      writer2.close()
                   excel+=1
                else:
                   #Saves the Excel document
                   book = load_workbook(excel_file)
                   with pd.ExcelWriter(excel_file, engine='openpyxl') as writer2:
                      writer2.book = book
                      writer2.sheets = dict((ws.title, ws) for ws in book.worksheets)
                      if(df_loads.empty == False):
                         df_loads.to_excel(writer2,sheet_name="N-1|"+str(ibus)+"-"+str(jbus)+"-
"+str(idckt)+"MVA",index=False)
                      if(df_voltages.empty == False):
                         df_voltages.to_excel(writer2,sheet_name="N-1|"+str(ibus)+"-"+str(jbus)+"-
"+str(idckt)+"V",index=False)
                      writer2.save()


          # N-2 parallel branches tripping
          m_brnchsFROMTON2 = []

          #Creates a list with all the parallel branches
          for iline in range(0,nallbrnchs):
             cont_v_allbrnchsFROM = []
             cont_v_allbrnchsTO = []
             cont_v_allbrnchsID = []
             cont_rates = []
```

```
            cont_branchesflow = []
            cont_v_allbuses = []
            cont_v_busvoltages = []
            busesKV = []
            busesKVint = []
            ibus=v_allbrnchsFROM[iline]
            jbus=v_allbrnchsTO[iline]
            idckt=v_allbrnchsID[iline]

            #Ensures that the N-2 contigency will only occur for parallel branches only and not for any
combination of two branches
            v_idxFROM = [i for i,x in enumerate(v_allbrnchsFROM) if x==v_allbrnchsFROM[iline]]
            for iline2 in v_idxFROM[1:]: # only 2 lines always - if more than 2 lines in parallel, get them in
the next loop
                if (iline2>iline) and (v_allbrnchsTO[iline2]==v_allbrnchsTO[iline]):
                    m_brnchsFROMTON2.append([ibus,jbus,idckt,v_allbrnchsID[iline2],iline,iline2])

##          #Runs the N-2 analysis
##          for iline2 in range(len(m_brnchsFROMTON2)):
##             cont_v_allbrnchsFROM = []
##             cont_v_allbrnchsTO = []
##             cont_v_allbrnchsID = []
##             cont_rates = []
##             cont_branchesflow = []
##             cont_v_allbuses = []
##             cont_v_busvoltages = []
##             busesKV = []
##             busesKVint = []
##             ibus=v_allbrnchsFROM[iline2]
##             jbus=v_allbrnchsTO[iline2]
##             idckt=v_allbrnchsID[iline2]
##
##            #Runs LF and checks that the case has converged
##             psspy.fnsl([0,0,0,1,1,0,99,0])
##             ierr, ibusmax, cmplxmismatch = psspy.maxmsm()
##             mismatchmva = abs(cmplxmismatch)
##             isconverged = 1
##             if mismatchmva>TOLN:
##                isconverged = 0
##
##             #Switches off branches
##
psspy.branch_data(m_brnchsFROMTON2[iline2][0],m_brnchsFROMTON2[iline2][1],m_brnchsFROMTON2[iline2][2],[0,_i,_i,_
i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
##
psspy.branch_data(m_brnchsFROMTON2[iline2][0],m_brnchsFROMTON2[iline2][1],m_brnchsFROMTON2[iline2][3],[0,_i,_i,_
i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
##
##             #Runs LF and checks that the case has converged
##             psspy.fnsl([0,0,0,1,1,0,99,0])
##             ierr, ibusmax, cmplxmismatch = psspy.maxmsm()
##             mismatchmva = abs(cmplxmismatch)
##             isconverged = 1
##             if mismatchmva>TOLN:
##                isconverged = 0
##
##             if isconverged:
##
##                #Checks if there is contigency in the remaining lines
##                v_remaininglines =
range(0,m_brnchsFROMTON2[iline2][4])+range(m_brnchsFROMTON2[iline2][4]+1,m_brnchsFROMTON2[iline2][5])+range(m_br
nchsFROMTON2[iline2][5]+1,nallbrnchs)
##                for ilineremain in v_remaininglines:
##
##                   #Gets data from other lines
##                   branchesflow = []
##                   rates = []
##                   ibusremain=v_allbrnchsFROM[ilineremain]
##                   jbusremain=v_allbrnchsTO[ilineremain]
##                   idcktremain=v_allbrnchsID[ilineremain]
##                   ierr, RATE = psspy.brndat(ibusremain,jbusremain,idcktremain,BRANCH_RATE)
##                   ierr, APFLOW = psspy.brnmsc(ibusremain,jbusremain,idcktremain,'MVA')
##                   rates.append(RATE)
```

```
##                   branchesflow.append(APFLOW)
##
##                   #Checks if there are overloads
##                   if ((APFLOW>RATE) and (RATE > 0)):
##                      cont_v_allbrnchsFROM.append(v_allbrnchsFROM[ilineremain])
##                      cont_v_allbrnchsTO.append(v_allbrnchsTO[ilineremain])
##                      cont_v_allbrnchsID.append(v_allbrnchsID[ilineremain])
##                      cont_rates.append(RATE)
##                      cont_branchesflow.append(APFLOW)
##
##               #Checks if there are voltages out of range in the buses
##                for ibuss in range(len(v_allbuses)):
##                   bus = v_allbuses[ibuss]
##                   ierr, vbus = psspy.busdat(bus ,'PU')
##                   if((vbus>voltupperlimit) or (vbus<voltlowerlimit)):
##                      cont_v_allbuses.append(v_allbuses[ibuss])
##                      cont_v_busvoltages.append(vbus)
##
##            #Switches on branches
##
psspy.branch_data(m_brnchsFROMTON2[iline2][0],m_brnchsFROMTON2[iline2][1],m_brnchsFROMTON2[iline2][2],[1,_i,_i,_
i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
##
psspy.branch_data(m_brnchsFROMTON2[iline2][0],m_brnchsFROMTON2[iline2][1],m_brnchsFROMTON2[iline2][3],[1,_i,_i,_
i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f])
##
##            #Exports data to word document if there is a contigency.
##            if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0)):
##               document.add_heading('N-2 Contingency results branches:'+str(m_brnchsFROMTON2[iline2][0])+'-
'+str(m_brnchsFROMTON2[iline2][1])+'-ID:1,2 '+strfilefdc, 0)
##               df_loads,df_voltages = GetLFResults(contingency, busesKV, busesKVint, document, voltupperlimit,
voltlowerlimit, loadwarning, strfilefdc, str_report_folder, cont_v_allbrnchsFROM, cont_rates, cont_branchesflow,
cont_v_allbrnchsTO, cont_v_allbrnchsID, cont_v_allbuses, cont_v_busvoltages,excel_file,writer2)
##               document.add_page_break()
##
##            #Prints an error onto the document if the case has not converged
##            if(isconverged == 0):
##               document.add_heading('N-2 Contingency results branches:'+str(m_brnchsFROMTON2[iline2][0])+'-
'+str(m_brnchsFROMTON2[iline2][1])+'-ID:1,2 '+strfilefdc, 0)
##               document.add_paragraph('CASE NOT CONVERGED, PLEASE CHECK LOAD FLOW PARAMETERS')
##               document.add_page_break()
##
##            #Gives a message in the excel in the case that it does not converge
##            if(isconverged == 0):
##               df_voltages = pd.DataFrame(["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"],index = [1],columns
= ["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"])
##               df_loads = pd.DataFrame(["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"],index = [1],columns =
["NO CONVERGENCE OR ESSENTIAL LINE DEACTIVATED"])
##
##            if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0) or (isconverged == 0)):
##               if(excel==0):
##                  #Saves the Excel Document
##                  with pd.ExcelWriter(excel_file, engine='xlsxwriter') as writer2:
##                     if(df_loads.empty == False):
##                        df_loads.to_excel(writer2,sheet_name="N-2|"+str(m_brnchsFROMTON2[iline2][0])+"-
"+str(m_brnchsFROMTON2[iline2][1])+"-
"+str(m_brnchsFROMTON2[iline2][2])+","+str(m_brnchsFROMTON2[iline2][3])+"MVA",index=False)
##                     if(df_voltages.empty == False):
##                        df_voltages.to_excel(writer2,sheet_name="N-2|"+str(m_brnchsFROMTON2[iline2][0])+"-
"+str(m_brnchsFROMTON2[iline2][1])+"-
"+str(m_brnchsFROMTON2[iline2][2])+","+str(m_brnchsFROMTON2[iline2][3])+"V",index=False)
##                     writer2.close()
##                  excel+=1
##               else:
##                  #Saves the Excel document
##                  book = load_workbook(excel_file)
##                  with pd.ExcelWriter(excel_file, engine='openpyxl') as writer2:
##                     writer2.book = book
##                     writer2.sheets = dict((ws.title, ws) for ws in book.worksheets)
##                     if(df_loads.empty == False):
##                        df_loads.to_excel(writer2,sheet_name="N-2|"+str(m_brnchsFROMTON2[iline2][0])+"-
"+str(m_brnchsFROMTON2[iline2][1])+"-
"+str(m_brnchsFROMTON2[iline2][2])+","+str(m_brnchsFROMTON2[iline2][3])+"MVA",index=False)
```

```
##                     if(df_voltages.empty == False):
##                        df_voltages.to_excel(writer2,sheet_name="N-2|"+str(m_brnchsFROMTON2[iline2][0])+"-
"+str(m_brnchsFROMTON2[iline2][1])+"-
"+str(m_brnchsFROMTON2[iline2][2])+","+str(m_brnchsFROMTON2[iline2][3])+"V",index=False)
##                     writer2.save()

         file_name_doc = str_report_folder+'\ResultsN-1N-2.docx'
         if os.path.exists(file_name_doc):
            os.remove(file_name_doc)
         if(word==0):
            #Saves the Word document
            document.save(file_name_doc)
         else:
            #Saves the Excel document
            document.save(file_name_doc)
         word+=1
         excel=0



def GetLFResults(contingency, busesKV, busesKVint, document, voltupperlimit, voltlowerlimit, loadwarning,
strfilefdc, str_report_folder, v_allbrnchsFROM, rates, branchesflow, v_allbrnchsTO, v_allbrnchsID, v_allbuses,
v_busvoltages,excel_file,writer):

         #Imports all the necessary functions to run the function
         import matplotlib.pyplot as plt
         import pandas as pd
         import docx
         from docx import Document
         from docx.shared import Cm, Pt
         from docx.shared import RGBColor
         #from docx.text.parargaph import Paragraph
         from docx.styles.style import WD_STYLE_TYPE
         from docx.enum.table import WD_TABLE_ALIGNMENT
         from docx.enum.table import WD_TABLE_DIRECTION
         from docx.enum.text import WD_ALIGN_PARAGRAPH
         import csv
         import fileinput
         import openpyxl # Agregado por mi
         import xlsxwriter
         import numpy
         import xlrd, xlwt
         from openpyxl import load_workbook

         #Graphs all the buses and their voltages, marking in red the ones that do noy meet the requirements
##         #Plots the branches flow graph
##         if(contingency == 0):
##            for r in range(len(busesKVint)):
##               plt.clf()
##               plt.close()
##               plt.figure()
##               plt.figure().clear()
##               for i in range (len(v_allbrnchsFROM)):
##                  idlist = v_allbuses.index(v_allbrnchsFROM[i])
##                  if(int(busesKV[idlist]) == busesKVint[r]):
##                     if(rates[i]!=0):
##                        if (branchesflow[i]>rates[i]):
##                           plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                           plt.text(i+.05, 100*branchesflow[i]/rates[i]+.1,str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i]+'->'+str(round(100*branchesflow[i]/rates[i],3))+'%', fontsize=7,
color = 'black')
##                           plt.bar(i, 100*branchesflow[i]/rates[i],color = 'r')
##                        if (branchesflow[i]>(rates[i]*0.9) and branchesflow[i]<rates[i]):
##                           plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                           plt.text(i+.05, 100*branchesflow[i]/rates[i]+.1,str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i]+'->'+str(round(100*branchesflow[i]/rates[i],3))+'%', fontsize=7,
color = 'black')
##                           plt.bar(i, 100*branchesflow[i]/rates[i],color = 'b')
##                        else:
##                           #plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                           #plt.text(i+.05, 100*branchesflow[i]/rates[i],str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i],fontsize = 7 ,color = 'black')
##                           plt.bar(i, 100*branchesflow[i]/rates[i],color = 'grey')
```

```
##
##               plt.ylabel("Load %")
##               plt.xlabel("From-To-ID->(Load)")
##               plt.suptitle('BRANCH LOAD RESULTS '+str(busesKVint[r])+'KV BASE', fontsize=16)
##               plt.axhline(y=100, color='r', linestyle='-')
##               plt.savefig(str_report_folder+'\Images-
Static'+'\Loads'+str(busesKVint[r])+'KV'+strfilefdc+'.png')
##         else:
##            plt.clf()
##            plt.close()
##            plt.figure()
##            plt.figure().clear()
##            for i in range (len(v_allbrnchsFROM)):
##               if(rates[i]!=0):
##                  if (branchesflow[i]>rates[i]):
##                     plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                     plt.text(i+.05, 100*branchesflow[i]/rates[i]+.1,str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i]+'->'+str(round(100*branchesflow[i]/rates[i],3))+'%', fontsize=7,
color = 'black')
##                     plt.bar(i, 100*branchesflow[i]/rates[i],color = 'r')
##                  if (branchesflow[i]>(rates[i]*0.9) and branchesflow[i]<rates[i]):
##                     plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                     plt.text(i+.05, 100*branchesflow[i]/rates[i]+.1,str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i]+'->'+str(round(100*branchesflow[i]/rates[i],3))+'%', fontsize=7,
color = 'black')
##                     plt.bar(i, 100*branchesflow[i]/rates[i],color = 'b')
##                  else:
##                     #plt.plot(i,100*branchesflow[i]/rates[i],'o',color ='black')
##                     #plt.text(i+.05, 100*branchesflow[i]/rates[i],str(v_allbrnchsFROM[i])+'-
'+str(v_allbrnchsTO[i])+'-'+v_allbrnchsID[i],fontsize = 7 ,color = 'black')
##                     plt.bar(i, 100*branchesflow[i]/rates[i],color = 'grey')
##
##            plt.ylabel("Load %")
##            plt.xlabel("From-To-ID->(Load)")
##            plt.suptitle('BRANCH LOAD RESULTS ', fontsize=16)
##            plt.axhline(y=100, color='r', linestyle='-')
##            plt.savefig(str_report_folder+'\Images-Static'+'\Loads'+strfilefdc+'.png')

         #Creates a table to add branch loads to the Word Document
         p = document.add_paragraph('Load results for each branch (transformers included): ')
         table = document.add_table(len(v_allbrnchsFROM)+1, 5)
         table.style = 'TableGrid'
         first_column_width = 5
         second_column_width = 10
         row = table.rows[0]
         row.cells[0].text = str("BusFROM")
         row.cells[1].text = str("BusTO")
         row.cells[2].text = str("RATE")
         row.cells[3].text = str("LOAD-MVA")
         row.cells[4].text = str("LOAD%")
         for i in range(len(v_allbrnchsFROM)):
             row = table.rows[i+1]
             row.cells[0].text = str(v_allbrnchsFROM[i])
             row.cells[1].text = str(v_allbrnchsTO[i])
             row.cells[2].text = str(rates[i])
             row.cells[3].text = str(round(branchesflow[i],3))
             if(rates[i]!=0):
                 if (branchesflow[i]>rates[i]):
                     row.cells[4].text = str(round(100*branchesflow[i]/rates[i],3))
                 else:
                     row.cells[4].text = str(round(100*branchesflow[i]/rates[i],3))

##         #Graphs all the branches and their load value
##         if(contingency == 0):
##            for s in range(len(busesKVint)):
##               document.add_picture(str_report_folder+'\Images-
Static'+'\Loads'+str(busesKVint[s])+'KV'+strfilefdc+'.png')
##         else:
##            if(len(v_allbrnchsFROM)!=0):
##               document.add_picture(str_report_folder+'\Images-Static'+'\Loads'+strfilefdc+'.png')
##         #document.add_page_break()

##         #Plots the bus voltages graph
```

```
##         if(contingency == 0):
##            for p in range(len(busesKVint)):
##               plt.clf()
##               plt.close()
##               plt.figure()
##               plt.figure().clear()
##
##               for i in range(len(v_allbuses)):
##                   if(int(busesKV[i]) == busesKVint[p]):
##                      if (v_busvoltages[i]>voltlowerlimit and v_busvoltages[i]<voltupperlimit):
##                         if (v_busvoltages[i]<(voltlowerlimit+0.01) or v_busvoltages[i]>(voltupperlimit-
0.01)):
##                            plt.plot(i+1, v_busvoltages[i],'o',color ='black')
##                            plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i])+',
v='+str(round(v_busvoltages[i],5))+'p.u', fontsize=9, color = 'black')
##
##                         else:
##                            plt.plot(i+1, v_busvoltages[i],'o',color ='black')
##                            plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i]), fontsize=9, color =
'black')
##                      else:
##                          plt.plot(i+1, v_busvoltages[i],'o',color ='red')
##                          plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i])+',
v='+str(round(v_busvoltages[i],5))+'p.u', fontsize=9, color = 'red')
##               plt.ylabel("Voltage (p.u)")
##               plt.xlabel("Bus ID - (Voltage)")
##               plt.suptitle('BUS VOLTAGES '+str(busesKVint[p])+'KV BASE', fontsize=16)
##               plt.axhline(y=voltlowerlimit, color='r', linestyle='-')
##               plt.axhline(y=voltupperlimit, color='r', linestyle='-')
##               plt.savefig(str_report_folder+'\Images-
Static'+'\Voltages'+str(busesKVint[p])+'KV'+strfilefdc+'.png')
##
##         else:
##            plt.clf()
##            plt.close()
##            plt.figure()
##            plt.figure().clear()
##            for i in range(len(v_allbuses)):
##               if (v_busvoltages[i]>voltlowerlimit and v_busvoltages[i]<voltupperlimit):
##                  if (v_busvoltages[i]<(voltlowerlimit+0.01) or v_busvoltages[i]>(voltupperlimit-0.01)):
##                        plt.plot(i+1, v_busvoltages[i],'o',color ='black')
##                        plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i])+',
v='+str(round(v_busvoltages[i],5))+'p.u', fontsize=9, color = 'black')
##
##                  else:
##                        plt.plot(i+1, v_busvoltages[i],'o',color ='black')
##                        plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i]), fontsize=9, color =
'black')
##               else:
##                  plt.plot(i+1, v_busvoltages[i],'o',color ='red')
##                  plt.text(i+1+.05, v_busvoltages[i]+.002, str(v_allbuses[i])+',
v='+str(round(v_busvoltages[i],5))+'p.u', fontsize=9, color = 'red')
##
##            plt.ylabel("Voltage (p.u)")
##            plt.xlabel("Bus ID - (Voltage)")
##            plt.suptitle('BUS VOLTAGES ', fontsize=16)
##            plt.axhline(y=voltlowerlimit, color='r', linestyle='-')
##            plt.axhline(y=voltupperlimit, color='r', linestyle='-')
##            plt.savefig(str_report_folder+'\Images-Static'+'\Voltages'+strfilefdc+'.png')

         #Create a table to add bus voltages to the Word document
         p = document.add_paragraph('Bus voltages for each bus: ')
         table = document.add_table(len(v_allbuses)+1, 2)
         table.style = 'TableGrid'
         first_column_width = 5
         second_column_width = 10
         row = table.rows[0]
         row.cells[0].text = str("Bus")
         row.cells[1].text = str("Voltage(p.u)")
         for i in range(len(v_allbuses)):
             row = table.rows[i+1]
             row.cells[0].text = str(v_allbuses[i])
             row.cells[1].text = str(round(v_busvoltages[i],4))
```

```
        #Creates a table to add branch loads to the Excel File
        loads_data = []
        index_list_loads = []
        for i in range(len(v_allbrnchsFROM)):
            index_list_loads.append(i)
            loads_data.append([])
            loads_data[i].append(int(v_allbrnchsFROM[i]))
            loads_data[i].append(int(v_allbrnchsTO[i]))
            loads_data[i].append(int(v_allbrnchsID[i]))
            if(rates[i]!=0):
                if (branchesflow[i]>rates[i]):
                    loads_data[i].append(float(round(100*branchesflow[i]/rates[i],3)))
                else:
                    loads_data[i].append(float(round(100*branchesflow[i]/rates[i],3)))

        #Creates a table to add branch loads to the Excel File
        voltages_data = []
        index_list_voltages = []

        for i in range(len(v_allbuses)):
            index_list_voltages.append(i)
            voltages_data.append([])
            voltages_data[i].append(int(v_allbuses[i]))
            voltages_data[i].append(float(round(v_busvoltages[i],4)))

        #Outputs dataframes for the Excel file
        df_voltages = pd.DataFrame(voltages_data,index = index_list_voltages,columns = ["Bus","Voltage(p.u)"])
        df_loads = pd.DataFrame(loads_data,index = index_list_loads,columns = ["BusFROM","BusTO","ID","LOAD%"])
        return df_loads, df_voltages

##         if(contingency == 0):
##            for q in range(len(busesKVint)):
##               document.add_picture(str_report_folder+'\Images-
Static'+'\Voltages'+str(busesKVint[q])+'KV'+strfilefdc+'.png')
##         else:
##            if(len(v_allbuses)!=0):
##               document.add_picture(str_report_folder+'\Images-Static'+'\Voltages'+strfilefdc+'.png')


# ------------------------------
# Main - runs if .py is executed
# ------------------------------

if __name__=="__main__":

   import os, sys

   #RunLF(str_report_folder,str_aux_case_folder,v_KVRANGE, loadwarning, voltupperlimit, voltlowerlimit,
ISPSSEVERSION, PSSE_LOCATION)

   #RunStaticN_1N_2branches(str_report_folder,str_aux_case_folder,v_KVRANGE, loadwarning, voltupperlimit,
voltlowerlimit, ISPSSEVERSION, PSSE_LOCATION)
```

ANEXO 7: FICHERO PYTHON

MAIN_SIMULATE_TRIPPINGS

```
# ----------
# Defined Functions
# ====================

def
RunDynamicGenTrippings(strdynfile,FBASE,tfin,FUF1,str_report_folder,str_aux_case_folder,str_aux_dyn_folder,dynam
ic_parameters,voltupperlimit,voltlowerlimit,BRANCH_RATE,max_angle,min_angle,max_angle_ratio):

    #Imports all the necessary modules to run the function
    import os,sys
    import psse34
    import psspy
    import redirect
    redirect.psse2py()
    import dyntools
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    from docx import Document
    from docx.shared import Cm, Pt
    #from docx.text.parargaph import Paragraph
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import csv
    import openpyxl
    import fileinput
    import openpyxl # Agregado por mi
    import xlsxwriter
    import numpy
    import xlrd, xlwt
    from openpyxl import load_workbook

    #Defines some parameters necessary to run the analysis
    pconsti = 100 # 100 para ajuste inicial - 90 para ajuste 2 # usually 100 here
    pconsty = 100 - pconsti    # 100 - pconsti # for wind power (100 here, 0 const current)
    qconsti = 0 # 0
    qconsty = 100 - qconsti    # 100
    TOLUFLS = 0.005 # tolerance for UFLS identification (1% of demand)
    strdll = [] # A list to be used fo the libraries

    _i=psspy.getdefaultint()
    _f=psspy.getdefaultreal()
    _s=psspy.getdefaultchar()

    #Creates the folder where the auxiliary dynamic files are going to be located
    if not os.path.exists(str_aux_dyn_folder):
             os.makedirs(str_aux_dyn_folder)
    #Creates some auxiliary files for the dynamic simulation
    strconec = str_aux_dyn_folder + r"\conec.flx"
    strconet = str_aux_dyn_folder + r"\conet.flx"
    strcompile = str_aux_dyn_folder + r"\compile.bat"
    for idll in range(len(strdll)):
            strdll[idll] = str_aux_case_folder + "\\" + strdll[idll]

    #Used for correctly opening and closing Word and Excel files
    word=0#For word files
    excel = 0#For Excel files
```

```
    #Goes through each file in the designated folder
    for file in os.listdir(str_aux_case_folder):
        if file.endswith(".sav"):

            # Prepare simulations
            # ===================

            # Path, load flow file, dynamic input file
            # ----------------------------------------
            strlffilename = file

            #Code to create or update the word document depending on wether it is the first case file or not
            excel = 0
            file_name = str_report_folder+"\DYNresultsGenTrip"+strlffilename+".docx"
            if os.path.exists(file_name):
                    os.remove(file_name)

            document = Document()
            document.add_heading('DYNAMICS RESULTS '+strlffilename, 0)

            #Creates or updates the excel document depending on wether it is the first case file or not

            excel_file_erase = str_report_folder + "\\" + "ResultsN-1_Gen_" + strlffilename[:len(strlffilename)-
4] + ".xlsx"
            if os.path.exists(excel_file_erase):
               os.remove(excel_file_erase)
            excel_file = str_report_folder + "\\" + "ResultsN-1_Gen_" + strlffilename[:len(strlffilename)-4] +
".xlsx"
            writer = pd.ExcelWriter(excel_file, engine = 'xlsxwriter')

            #Creates some auxiliary files for the dynamic simulation
            strconvlffile = str_aux_dyn_folder + '\\' + strlffilename[:len(strlffilename)-4] + '_conv.sav'
            strsnapfile = str_aux_dyn_folder + '\\' + strlffilename[:len(strlffilename)-4] + '.snp'
            strlffile = str_aux_case_folder + '\\' + strlffilename
            stroutputfile = str_report_folder + "\\" + strlffilename[:len(strlffilename)-4] + '.out'

            # Case file
            # ---------
            psspy.psseinit(2000)
            # psspy.lines_per_page_one_device(1,60)
            # psspy.progress_output(2,strreportfile,[0,0])
            psspy.case(strlffile)

            # Extract data and modify load flow for ESS
            # -----------------------------------------
            MVABASE = psspy.sysmva()

            ties=3 #for both interior subsystem branches and tie branches.
            entry=1 #for single entry (each branch once).
            sid = -1 # subsystem identifier --> All subsystems
            busflag = 1 #Only in service buses
            generatorflag=1 #Only in-service generators
            branchflag=3 #for only in-service transformer and non-transformer branches.

            #Gets the number of loads, to get the ampliada Y matrix
            ierr, nummachs = psspy.amachcount(sid, busflag) # only in service machines
            ierr, numloads = psspy.aloadcount(sid, busflag) # only in service loads

            #Gets load buses, buses count, and buses.
            ierr, v_loadbusnumbers = psspy.aloadint(sid, busflag, 'NUMBER')
            v_loadbusnumbers = v_loadbusnumbers[0]
            ierr, nbuses = psspy.abuscount(sid, busflag) # only in service buses
            ierr, v_allbuses = psspy.abusint(sid, busflag, 'NUMBER')
            v_allbuses = v_allbuses[0]

            #Returns an array of buses in which machines are located, used to evaluate the trippings
            ierr, v_genbusout = psspy.amachint(sid, generatorflag, 'NUMBER')
            v_genbusout= v_genbusout[0]
            ierr, v_genbusIDout = psspy.amachchar(sid, generatorflag, 'ID')
            v_genbusIDout = v_genbusIDout[0]
```

```
            #Return the number of array entries required to accommodate the data to be returned by the remaining
members of the branch data family.
            ierr, nallbrnchs = psspy.abrncount(sid, 1, ties, branchflag, entry)
            #Return an array of integer values for subsystem branches (from, to and ID)
            ierr, v_allbrnchsFROM = psspy.abrnint(sid, 1, ties, branchflag, entry, 'FROMNUMBER')
            v_allbrnchsFROM = v_allbrnchsFROM[0]
            ierr, v_allbrnchsTO = psspy.abrnint(sid, 1, ties, branchflag, entry, 'TONUMBER')
            v_allbrnchsTO = v_allbrnchsTO[0]
            ierr, v_allbrnchsID = psspy.abrnchar(sid, 1, ties, branchflag, entry, 'ID')
            v_allbrnchsID = v_allbrnchsID[0]

            #Creates the lists of the generators to trip, and the time at which it occurs
            v_tgenout = [1] #This list includes the time at which the trippings are evaluated
            v_tgenout.append(tfin)
            tstep = 0.01#This parameter sets the frequency time each sample is taken

            for igenout in range(len(v_genbusout)):
                if(word!=0):
                    #Starts the case over, without the tripping of the previous generator
                    psspy.case(strlffile)

                    # Extract data and modify load flow for ESS
                    # ----------------------------------------
                    MVABASE = psspy.sysmva()

                # Load flow
                #Runs an initial load flow, to further check the P-V generators curves are not on the unstable
side.
                psspy.fnsl([0,1,0,1,0,0,99,0])# tap, area interchange, phase shift, dc tap adjustment, switched
shunt, flat start, var limit, non-divergence

                # Convert generators and loads, resolve load flow & add dynamic data
                # ------------------------------------------------------------------

                #Defines the way generators are going to be represented (Norton equivalent)
                psspy.cong(0)

                #Defines the ways the loads are going to be represented (p=po, p=u*io, p=u^2/r)
                psspy.conl(0,1,1,[0,0],[ pconsti,pconsty,qconsti,qconsty])
                psspy.conl(0,1,2,[0,0],[ pconsti,pconsty,qconsti,qconsty])
                psspy.conl(0,1,3,[0,0],[ pconsti,pconsty,qconsti,qconsty])

                #Orders the matrix lines and columns to try to triangulate it
                psspy.ordr(0)

                #Factorises the Y matrix into two simpler ones
                psspy.fact()

                #Solves grid equations   Yamp*v=i   SI EL NUMERO DE ITERACIONES ES MAYOR QUE 1, PUEDE HABER
PROBLEMAS PARA LOS DINAMICOS
                psspy.tysl(0)

                #Saves the file with the parameters of the grid
                psspy.save(strconvlffile)

                #Anadir el fichero con parametros dinamicos
                psspy.dyre_new([1,1,1,1],strdynfile,strconec,strconet,strcompile)

                #Modifies dynamic simulation solution parameters
                psspy.dynamics_solution_param_2([_i,_i,_i,_i,_i,_i,_i,1],[0.7,_f,tstep,_f,_f,_f,_f,_f])
                psspy.change_channel_out_file(stroutputfile)

                # Set output, frequency dependence
                psspy.set_netfrq(1)

                #Disable dynamic simulation if there is a fatal data error
                psspy.set_disable_run(1)

                #Enable ZSORCE reconciliation
                psspy.set_zsorce_reconcile_flag(1)

                evaluate_pelec = 0
                evaluate_pmech = 0
```

```
evaluate_spd = 0
evaluate_all = 0
####Selects the data to be extracted from the file results
if(dynamic_parameters[0] == 1):
    ierr = psspy.chsb(0,1,[-1,-1,-1,1,2,0]) # Evaluates pelec
    evaluate_pelec = 1
if(dynamic_parameters[1] == 1):
    ierr = psspy.chsb(0,1,[-1,-1,-1,1,6,0]) # Evaluates pmech
    evaluate_pmech = 1
if(dynamic_parameters[2] == 1):
    ierr = psspy.chsb(0,1,[-1,-1,-1,1,7,0]) # Evaluates speed
    evaluate_spd = 1
if(dynamic_parameters[3] == 1):
    ierr = psspy.chsb(0,1,[-1,-1,-1,7,0,0]) # Evaluates all totals
    evaluate_all = 1

####Save dynamics to a snapshot file
psspy.snap([-1,-1,-1,-1,-1],strsnapfile)

# ierr = psspy.pssehalt_2()

print ("LF solved, case converted, dynamics added.")

# os.system(strcompile)
# os.system(strcload4file)

#Add a library to the list to be searched for library models
for idll in range(len(strdll)):
        psspy.addmodellibrary(strdll[idll])

print ("User models compiled and linked.")

# psspy.psseinit(2000)
# psspy.case(strlffile)

# Tripping computation
# ====================
#Select case to study
psspy.case(strconvlffile)

#Read dynamics snapshot file
psspy.rstr(strsnapfile)

#Initialise, specify channel output file
ierr = psspy.strt(0,stroutputfile)

# Consec. outages
# ---------------

#Run simulation with step times
psspy.run(0,v_tgenout[0],0,0,0)

print "Outage unit at bus: ", str(v_genbusout[igenout]), " and ID: ",str(v_genbusIDout[igenout])

#Turn off the selected generator
psspy.dist_machine_trip(v_genbusout[igenout],v_genbusIDout[igenout])

#Change channel output file
psspy.change_channel_out_file(stroutputfile)

#Run dynamics again with tripped generator
psspy.run(0, v_tgenout[1],0,2,0)

####### Results
# -------
#Saca los datos del output file en forma de excel
chnfobj = dyntools.CHNF(stroutputfile)

#Sorts the data of the simulation into title, ID and the data
short_title, chanid, chandata = chnfobj.get_data()

#Runs the dynamic analysis of the trippings
gen_tripping = 1 #This case if for generator trippings
```

```
                bus_sc = 0 #This case is not for bus short circuits
                GetDYNresults(document, word,chanid, chandata, strlffilename, FBASE,
v_tgenout,v_genbusIDout[igenout],v_genbusout[igenout],nummachs,str_report_folder,evaluate_pelec,evaluate_pmech,e
valuate_spd,evaluate_all,v_genbusout,FUF1,gen_tripping,bus_sc,max_angle,min_angle,max_angle_ratio)

                #Runs the Static Analysis
                df_voltages, df_loads, cont_v_allbrnchsFROM, cont_v_allbuses, isconverged =
GetStaticN_1GenTrippings(nallbrnchs,v_allbrnchsFROM,v_allbrnchsTO,v_allbrnchsID,v_allbuses,voltupperlimit,voltlo
werlimit,BRANCH_RATE)

                #Saves the word document after filling it out
                if(word==0):
                        document.save(file_name)
                        word+=1
                else:
                        document.save(file_name)

                #Saves the dataframes onto the Excel document
                if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0) or (isconverged == 0)):
                    if (excel==0):
                        #Saves the Excel Document
                        with pd.ExcelWriter(excel_file, engine='xlsxwriter') as writer:
                            if(df_loads.empty == False):

df_loads.to_excel(writer,sheet_name="Gen_"+str(v_genbusout[igenout])+"_ID_"+str(v_genbusIDout[igenout])+"MVA",in
dex=False)
                            if(df_voltages.empty == False):

df_voltages.to_excel(writer,sheet_name="Gen_"+str(v_genbusout[igenout])+"_ID_"+str(v_genbusIDout[igenout])+"V",i
ndex=False)
                            writer.close()
                        excel+=1
                    else:
                        #Saves the Excel document
                        book = load_workbook(excel_file)
                        with pd.ExcelWriter(excel_file, engine='openpyxl') as writer:
                            writer.book = book
                            writer.sheets = dict((ws.title, ws) for ws in book.worksheets)
                            if(df_loads.empty == False):

df_loads.to_excel(writer,sheet_name="Gen_"+str(v_genbusout[igenout])+"_ID_"+str(v_genbusIDout[igenout])+"MVA",in
dex=False)
                            if(df_voltages.empty == False):

df_voltages.to_excel(writer,sheet_name="Gen_"+str(v_genbusout[igenout])+"_ID_"+str(v_genbusIDout[igenout])+"V",i
ndex=False)
                            writer.save()

                #Closes the analysis for it to be started over again with the next generator
                psspy.close_powerflow()



def
GetStaticN_1GenTrippings(nallbrnchs,v_allbrnchsFROM,v_allbrnchsTO,v_allbrnchsID,v_allbuses,voltupperlimit,voltlo
werlimit,BRANCH_RATE):

    #Imports all the necessary functions to run the function
    import os,sys
    import psse34
    import psspy
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    import random
    from docx import Document
    from docx.shared import Cm, Pt
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import csv
    import openpyxl
```

```
    import fileinput
    import openpyxl # Agregado por mi
    import xlsxwriter
    import numpy
    import xlrd, xlwt
    from openpyxl import load_workbook

    #Creates vectors to export data onto word document
    cont_v_allbrnchsFROM = []
    cont_v_allbrnchsTO = []
    cont_v_allbrnchsID = []
    cont_rates = []
    cont_branchesflow = []
    cont_v_allbuses = []
    cont_v_busvoltages = []
    busesKV = []
    busesKVint = []

    #Runs LF and checks mismatch
    psspy.fnsl([0,1,0,1,0,0,99,0])# tap, area interchange, phase shift, dc tap adjustment, switched shunt, flat
start, var limit, non-divergence
    ierr,TOLN = psspy.newton_tolerance()
    ierr, ibusmax, cmplxmismatch = psspy.maxmsm()
    mismatchmva = abs(cmplxmismatch)
    isconverged = 1
    if mismatchmva>TOLN:
        isconverged = 0

    #If it is converged, gets contigency data
    #Gets data from other lines
    for iline in range(nallbrnchs):
        branchesflow = []
        rates = []
        ibus=v_allbrnchsFROM[iline]
        jbus=v_allbrnchsTO[iline]
        idckt=v_allbrnchsID[iline]
        ierr, RATE = psspy.brndat(ibus,jbus,idckt,BRANCH_RATE)
        ierr, APFLOW = psspy.brnmsc(ibus,jbus,idckt,'MVA')
        rates.append(RATE)
        branchesflow.append(APFLOW)

        #Checks if there are overloads
        if ((APFLOW>RATE) and (RATE > 0)):
            cont_v_allbrnchsFROM.append(v_allbrnchsFROM[iline])
            cont_v_allbrnchsTO.append(v_allbrnchsTO[iline])
            cont_v_allbrnchsID.append(v_allbrnchsID[iline])
            cont_rates.append(RATE)
            cont_branchesflow.append(APFLOW)

    #Gets data from buses, checks if voltage is ok
    for ibuss in range(len(v_allbuses)):
        bus = v_allbuses[ibuss]
        ierr, vbus = psspy.busdat(bus ,'PU')
        if((vbus>voltupperlimit) or (vbus<voltlowerlimit)):
            cont_v_allbuses.append(v_allbuses[ibuss])
            cont_v_busvoltages.append(vbus)

    #Exports data to word document if there is a contigency.
    if((len(cont_v_allbrnchsFROM)> 0) or (len(cont_v_allbuses)> 0) or (isconverged == 0)):
        #Creates a table to add branch loads to the Excel File
        loads_data = []
        index_list_loads = []
        for i in range(len(cont_v_allbrnchsFROM)):
            index_list_loads.append(i)
            loads_data.append([])
            loads_data[i].append(int(cont_v_allbrnchsFROM[i]))
            loads_data[i].append(int(cont_v_allbrnchsTO[i]))
            loads_data[i].append(int(cont_v_allbrnchsID[i]))
            if(cont_rates[i]!=0):
                if (cont_branchesflow[i]>cont_rates[i]):
                    loads_data[i].append(float(round(100*cont_branchesflow[i]/cont_rates[i],3)))
                else:
                    loads_data[i].append(float(round(100*cont_branchesflow[i]/cont_rates[i],3)))
```

```
        #Creates a table to add branch loads to the Excel File
        voltages_data = []
        index_list_voltages = []

        for i in range(len(cont_v_allbuses)):
            index_list_voltages.append(i)
            voltages_data.append([])
            voltages_data[i].append(int(cont_v_allbuses[i]))
            voltages_data[i].append(float(round(cont_v_busvoltages[i],4)))

        #Outputs dataframes for the Excel file
        df_voltages = pd.DataFrame(voltages_data,index = index_list_voltages,columns = ["Bus","Voltage(p.u)"])
        df_loads = pd.DataFrame(loads_data,index = index_list_loads,columns = ["BusFROM","BusTO","ID","LOAD%"])

    #Gives a message in the excel in the case that it does not converge
##    if(isconverged == 0):
##        df_voltages = pd.DataFrame(["NO CONVERGENCE"],index = [1],columns = ["NO CONVERGENCE"])
##        df_loads = pd.DataFrame(["NO CONVERGENCE"],index = [1],columns = ["NO CONVERGENCE"])

    return df_voltages, df_loads, cont_v_allbrnchsFROM, cont_v_allbuses, isconverged

def
RunDynamicShortCircuits(strdynfile,FBASE,tfin,FUF1,str_report_folder,str_aux_case_folder,str_aux_dyn_folder,dyna
mic_parameters,voltupperlimit,voltlowerlimit,BRANCH_RATE,sc_buses,max_angle,min_angle,max_angle_ratio):

    #Imports all the necessary modules to run the function
    import os,sys
    import psse34
    import psspy
    import redirect
    redirect.psse2py()
    import dyntools
    import matplotlib.pyplot as plt
    import pandas as pd
    import docx
    from docx import Document
    from docx.shared import Cm, Pt
    #from docx.text.parargaph import Paragraph
    from docx.styles.style import WD_STYLE_TYPE
    from docx.enum.table import WD_TABLE_ALIGNMENT
    from docx.enum.table import WD_TABLE_DIRECTION
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    import csv
    import openpyxl
    import fileinput
    import openpyxl # Agregado por mi
    import xlsxwriter
    import numpy
    import xlrd, xlwt
    from openpyxl import load_workbook

    #Defines some parameters necessary to run the analysis
    pconsti = 100 # 100 para ajuste inicial - 90 para ajuste 2 # usually 100 here
    pconsty = 100 - pconsti    # 100 - pconsti # for wind power (100 here, 0 const current)
    qconsti = 0 # 0
    qconsty = 100 - qconsti    # 100
    TOLUFLS = 0.005 # tolerance for UFLS identification (1% of demand)
    strdll = [] # A list to be used fo the libraries

    _i=psspy.getdefaultint()
    _f=psspy.getdefaultreal()
    _s=psspy.getdefaultchar()

    #Creates the folder where the auxiliary dynamic files are going to be located
    if not os.path.exists(str_aux_dyn_folder):
             os.makedirs(str_aux_dyn_folder)
    #Creates some auxiliary files for the dynamic simulation
    strconec = str_aux_dyn_folder + r"\conec.flx"
    strconet = str_aux_dyn_folder + r"\conet.flx"
    strcompile = str_aux_dyn_folder + r"\compile.bat"
    for idll in range(len(strdll)):
            strdll[idll] = str_aux_case_folder + "\\" + strdll[idll]
```

```
    #Used for correctly opening and closing Word and Excel files
    word=0#For word files

    #Goes through each file in the designated folder
    for file in os.listdir(str_aux_case_folder):
        if file.endswith(".sav"):

            # Prepare simulations
            # ===================

            # Path, load flow file, dynamic input file
            # ----------------------------------------
            strlffilename = file

            #Code to create or update the word document depending on whether it is the first case file or not
            file_name = str_report_folder+"\DYNresultsBusSC"+strlffilename+".docx"
            if os.path.exists(file_name):
                    os.remove(file_name)

            document = Document()
            document.add_heading('DYNAMICS RESULTS '+strlffilename, 0)

            #Creates some auxiliary files for the dynamic simulation
            strconvlffile = str_aux_dyn_folder + '\\' + strlffilename[:len(strlffilename)-4] + '_conv.sav'
            strsnapfile = str_aux_dyn_folder + '\\' + strlffilename[:len(strlffilename)-4] + '.snp'
            strlffile = str_aux_case_folder + '\\' + strlffilename
            stroutputfile = str_report_folder + "\\" + strlffilename[:len(strlffilename)-4] + '.out'

            # Case file
            # ---------
            psspy.psseinit(2000)
            # psspy.lines_per_page_one_device(1,60)
            # psspy.progress_output(2,strreportfile,[0,0])
            psspy.case(strlffile)

            # Extract data and modify load flow for ESS
            # -----------------------------------------
            MVABASE = psspy.sysmva()

            ties=3 #for both interior subsystem branches and tie branches.
            entry=1 #for single entry (each branch once).
            sid = -1 # subsystem identifier --> All subsystems
            busflag = 1 #Only in service buses
            generatorflag=1 #Only in-service generators
            branchflag=3 #for only in-service transformer and non-transformer branches.

            #Creates the lists of the buses in which to produce the Short Circuit, and the time at which it
occurs
            v_allbuses = sc_buses
            v_tgenout = [1] #This list includes the time at which the trippings are evaluated
            v_tgenout.append(tfin)
            tstep = 0.02#This parameter sets the frequency time each sample is taken
            ierr, nummachs = psspy.amachcount(sid, busflag) #Number of machines to be evaluated

            for sc_bus in range(len(v_allbuses)):

                os.remove(stroutputfile)
                stroutputfile = str_report_folder + "\\" + strlffilename[:len(strlffilename)-4] + '.out'

                #Applies "1" to the ID of the bus in order for the common function with generator trippings
(GetDynResults) to be used
                v_genbusIDout = "1"

                if(word!=0):
                    #Starts the case over, without the tripping of the previous bus
                    psspy.case(strlffile)

                    # Extract data and modify load flow for ESS
                    # -----------------------------------------
                    MVABASE = psspy.sysmva()

                # Load flow
```

```
                #Runs an initial load flow, to further check the P-V generators curves are not on the unstable
side.
                psspy.fnsl([0,0,0,1,1,0,99,0])# tap, area interchange, phase shift, dc tap adjustment, switched
shunt, flat start, var limit, non-divergence

                # Convert generators and loads, resolve load flow & add dynamic data
                # ---------------------------------------------------------------------

                #Defines the way generators are going to be represented (Norton equivalent)
                psspy.cong(0)

                #Defines the ways the loads are going to be represented (p=po, p=u*io, p=u^2/r)
                psspy.conl(0,1,1,[0,0],[ pconsti,pconsty,qconsti,qconsty])
                psspy.conl(0,1,2,[0,0],[ pconsti,pconsty,qconsti,qconsty])
                psspy.conl(0,1,3,[0,0],[ pconsti,pconsty,qconsti,qconsty])

                #Orders the matrix lines and columns to try to triangulate it
                psspy.ordr(0)

                #Factorises the Y matrix into two simpler ones
                psspy.fact()

                #Solves grid equations   Yamp*v=i   SI EL NUMERO DE ITERACIONES ES MAYOR QUE 1, PUEDE HABER
PROBLEMAS PARA LOS DINAMICOS
                psspy.tysl(0)

                #Saves the file with the parameters of the grid
                psspy.save(strconvlffile)

                #Anadir el fichero con parametros dinamicos
                psspy.dyre_new([1,1,1,1],strdynfile,strconec,strconet,strcompile)

                #Modifies dynamic simulation solution parameters
                psspy.dynamics_solution_param_2([_i,_i,_i,_i,_i,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f])
                psspy.delete_all_plot_channels()
                psspy.change_channel_out_file(stroutputfile)

                # Set output, frequency dependence
                psspy.set_netfrq(1)

                #Enables relative angle calculation
                psspy.set_relang(1,0,"")

                #Disable dynamic simulation if there is a fatal data error
                psspy.set_disable_run(1)

                #Enable ZSORCE reconciliation
                psspy.set_zsorce_reconcile_flag(1)

                evaluate_angle = 0
                evaluate_voltage = 0
                evaluate_speed = 0
                evaluate_angle_stats = 0

                ####Selects the data to be extracted from the file results
                if(dynamic_parameters[0] == 1):
                    ierr = psspy.chsb(0,1,[-1,-1,-1,1,1,0]) # Evaluates angle
                    evaluate_angle = 1
                if(dynamic_parameters[1] == 1):
                    ierr = psspy.chsb(0,1,[-1,-1,-1,1,4,0]) # Evaluates terminal voltage
                    evaluate_voltage = 1
                if(dynamic_parameters[2] == 1):
                    ierr = psspy.chsb(0,1,[-1,-1,-1,1,7,0]) # Evaluates speed
                    evaluate_speed = 1
                if(dynamic_parameters[3] == 1):
                    ierr = psspy.chsb(0,1,[-1,-1,-1,8,0,0]) # Evaluates machine angle statistics
                    evaluate_angle_stats = 1

                ####Save dynamics to a snapshot file
                psspy.snap([-1,-1,-1,-1,-1],strsnapfile)

                # Consec. outages
                # ---------------
```

```
                #Select case to study
                psspy.case(strconvlffile)

                #Read dynamics snapshot file
                psspy.rstr(strsnapfile)

                #Start output file
                ierr = psspy.strt(0,stroutputfile)

                #Run simulation with step times
                psspy.run(0,v_tgenout[0],0,0,0)

                print "Outage unit at bus: ", str(v_allbuses[sc_bus])

                #Turn off the selected bus
                #1)Bus
                #2)Fault admittance in MVA
                #3)Bus base voltage is used
                #4)Default values for the fault admittance
                psspy.dist_bus_fault(v_allbuses[sc_bus],1,0.0,[0.0,-0.2E+10])
                psspy.change_channel_out_file(stroutputfile)

                #Run dynamics again with tripped generator
                t_fault = 0.1
                psspy.run(0,v_tgenout[0]+t_fault,0,0,0)

                #Clears the fault and runs the dynamics again
                psspy.dist_clear_fault(1)
                #psspy.change_channel_out_file(stroutputfile)
                psspy.run(0, v_tgenout[1],0,0,0)

                ####### Results
                # -------
                #Saca los datos del output file en forma de excel
                chnfobj = dyntools.CHNF(stroutputfile)

                #Sorts the data of the simulation into title, ID and the data
                short_title, chanid, chandata = chnfobj.get_data()

                #Runs the dynamic analysis of the trippings
                gen_tripping = 0 #This case is not for generator trippings
                bus_sc = 1 #This case is for bus short circuits
                GetDYNresults(document, word,chanid, chandata, strlffilename, FBASE,
v_tgenout,v_genbusIDout,v_allbuses[sc_bus],nummachs,str_report_folder,evaluate_angle,evaluate_voltage,evaluate_s
peed,evaluate_angle_stats,v_allbuses,FUF1,gen_tripping,bus_sc,max_angle,min_angle,max_angle_ratio)

                #Saves the word document after filling it out
                if(word==0):
                        document.save(file_name)
                        word+=1
                else:
                        document.save(file_name)

                #Closes the analysis for it to be started over again with the next generator
                psspy.close_powerflow()


def GetDYNresults(document, word, chanid, chandata, strlffilename, FBASE, v_tgenout,trip_gen_ID,trip_gen,
nummachs,str_report_folder,evaluate_pelec,evaluate_pmech,evaluate_spd,evaluate_all,v_genbusout,FUF1,gen_tripping
,bus_sc,max_angle,min_angle,max_angle_ratio):

    #Imports all the necessary functions to run the function
    import os,sys
    import matplotlib.pyplot as plt
    import pandas
    import dyntools
    import csv
    import openpyxl
    import fileinput
    import xlsxwriter
    import numpy
    import xlrd, xlwt
```

```
    from openpyxl import load_workbook

    #Establishes the path where the images are going to be stored
    strpathresultimages = str_report_folder + r"\Images-Dynamics"

    #Creates the images folder
    if not os.path.exists(strpathresultimages):
         os.makedirs(strpathresultimages)

    #If exporting of the speed of the generators/frequency of the system is chosen, it modifies the chandata
dictionary so only the speed of one functional generator is shown
    if(trip_gen==v_genbusout[0]):
        for key in range
(nummachs*(evaluate_pelec+evaluate_pmech)+1,nummachs*(evaluate_pelec+evaluate_pmech+evaluate_spd)):
            chandata.pop(key,"None")
            chanid.pop(key,"None")
    else:
        for key in range
(nummachs*(evaluate_pelec+evaluate_pmech)+2,nummachs*(evaluate_pelec+evaluate_pmech+evaluate_spd)+1):
            chandata.pop(key,"None")
            chanid.pop(key,"None")

    #Obtains the variables and pops out the time keys from the dictionaties
    time_dict = chandata['time']
    time = chandata['time']
    chandata.pop('time',"None")
    chanid.pop('time',"None")

    if(gen_tripping == 1):
        #Prepares some frequency-related data for the Word document
        parrafo = ""
        if(evaluate_spd==1):
            if(trip_gen==v_genbusout[0]):
                v_df = chandata[nummachs*(evaluate_pelec+evaluate_pmech+evaluate_spd)] #Frequency variation
vector
            else:
                v_df = chandata[nummachs*(evaluate_pelec+evaluate_pmech)+1] #Frequency variation vector

            v_f = list(map(lambda x: x + 1, v_df)) # f = 1 + df FREQUENCY VECTOR
            fmin = round(min(v_f)*FBASE,3)    #Minimum frequency
            fmax = round(max(v_f)*FBASE,3)     #Maximum frequency
            fss = round(v_f[len(v_f)-1]*FBASE,3) #Stead state frequency (last value)
            parrafo = "The min freq is " +str(fmin)+"Hz, " +"the max freq is " +str(fmax)+"Hz, " + "the steady-
state freq is " +str(fss)+"Hz"

        #Writes the title of the figures with some key information about the analysis in them
        parrafo = str(trip_gen)+" ID:"+str(trip_gen_ID) + " at time 1s. "+parrafo
        p = document.add_paragraph('DYNAMICS RESULTS FOR TRIPPINGS AT BUS '+parrafo)

        if(fmin>=FUF1):
            p = document.add_paragraph('FREQUENCY IS MANTAINED ABOVE THRESHOLD AT ALL TIMES')

        if(fmin<FUF1):
            #Iterates to fill out the graphs
            for i in chandata:

                #Creates figure and deletes the previous one to free up memory
                plt.clf()
                plt.close()
                plt.figure().clear()
                plt.figure(figsize=(5, 3.75))

                #Lines of code for composing the title for the graph
                variable = []
                v=0
                while True:
                    if (chanid[i][v] == ' '):
                     break
                    else:
                     v+=1
                variable=chanid[i][0:v]
                bus = chanid[i][v:]
                #Divides the graphs on types depending on what they are evaluating
```

```
                if (variable == 'POWR'):
                    variable = 'PELEC in p.u at bus' +str(bus)
                    ylabel = 'Electric Power in p.u'
                elif (variable == 'PMEC'):
                    variable = 'PMECH in p.u at bus' +str(bus)
                    ylabel = 'PMECH in p.u'
                elif (variable == 'SPD'):
                    variable = 'Frequency variation (p.u)'
                    ylabel = 'Frequency (p.u)'
                else:
                    ylabel = variable
                    variable =  variable + ' '+str(bus)

                #Plots every parameter over time time on the units requested or provided by PSSE
                for k in range(len(time_dict)):
                    plt.plot(time_dict[k],chandata[i][k],'.',markersize=1,color ='black')
                    plt.ylabel(ylabel)
                    plt.xlabel("Time")
                    plt.suptitle(variable, fontsize=14)

                #Saves the figure on the Images folder
                plt.savefig(strpathresultimages+'\ '+strlffilename+chanid[i]+'.png')

                #Saves the image onto the word document
                document.add_picture(strpathresultimages+'\ '+strlffilename+chanid[i]+'.png')

    if(bus_sc == 1):
        #Prepares some frequency-related data for the Word document
        parrafo = ""
        v_d_max_angle = chandata[nummachs*(evaluate_pelec+evaluate_pmech+evaluate_spd)+2] #Vector that contains
data related to maximum angle in the system
        v_d_min_angle = chandata[nummachs*(evaluate_pelec+evaluate_pmech+evaluate_spd)+4] #Vector that contains
data related to minimum angle in the system

        #Maximum angle
        v_max_angle = list(map(lambda x: x + 1, v_d_max_angle)) # angle = 1 + d angle ANGLE VECTOR
        angle_max = round(max(v_max_angle),3)    #Maximum angle
        angle_max_ss = round(v_max_angle[len(v_max_angle)-1],3) #Steady state maximum angle
        parrafo = "The max system dynamic angle is " +str(angle_max)+" Degrees, the steady state max angle is  "
+str(angle_max_ss)+" Degrees. "

        #Minimum angle
        v_min_angle = list(map(lambda x: x + 1, v_d_min_angle)) # angle = 1 + d angle ANGLE VECTOR
        angle_min = round(min(v_min_angle),3)    #Minimum angle
        angle_min_ss = round(v_min_angle[len(v_min_angle)-1],3) #Steady state minimum angle
        parrafo = parrafo + "The min system dynamic angle is " +str(angle_min)+" Degrees, the steady state max
angle is  " +str(angle_min_ss)+" Degrees. "

        #Writes the title of the figures with some key information about the analysis in them
        parrafo = str(trip_gen)+" ID:"+str(trip_gen_ID) + " at time ls. "+parrafo
        p = document.add_paragraph('DYNAMICS RESULTS FOR SHORT CIRCUIT AT BUS '+parrafo)


if((angle_max>max_angle)or(abs(angle_max/angle_max_ss)>max_angle_ratio)or(angle_min<min_angle)or(abs(angle_min/a
ngle_min_ss)>max_angle_ratio)):
            #Iterates to fill out the graphs
            for i in chandata:

                #Creates figure and deletes the previous one to free up memory
                plt.clf()
                plt.close()
                plt.figure().clear()
                plt.figure(figsize=(5, 3.75))

                #Lines of code for composing the title for the graph
                variable = []
                v=0
                while True:
                    if (chanid[i][v] == ' '):
                     break
                    else:
                     v+=1
                variable=chanid[i][0:v]
```

```
                bus = chanid[i][v:]
                #Divides the graphs on types depending on what they are evaluating
                if (variable == 'SPD'):
                    variable = 'Frequency variation (p.u)'
                    ylabel = 'Frequency (p.u)'
                elif (variable == 'ETRM'):
                    variable = 'Voltage in p.u at bus' +str(bus)
                    ylabel = 'Voltage (p.u)'
                elif (variable == 'ANGL'):
                    variable = 'Angle in degrees at bus' +str(bus)
                    ylabel = 'Angle (degrees)'
                else:
                    ylabel = variable
                    variable =  variable + ' '+str(bus)

                #Plots every parameter over time time on the units requested or provided by PSSE
                for k in range(len(time_dict)):
                    plt.plot(time_dict[k],chandata[i][k],'.',markersize=1,color ='black')
                    plt.ylabel(ylabel)
                    plt.xlabel("Time")
                    plt.suptitle(variable, fontsize=14)

                #Saves the figure on the Images folder
                plt.savefig(strpathresultimages+'\ '+strlffilename+chanid[i]+'.png')

                #Saves the image onto the word document
                document.add_picture(strpathresultimages+'\ '+strlffilename+chanid[i]+'.png')

        else:
            p = document.add_paragraph('ALL ANGLES ARE MANTAINED INSIDE SPECIFIED THRESHOLDS AT ALL TIMES')

if __name__=="__main__":

##    # The main runs the functions defined above.
##    #
##    # User-defined variables:  Faulting generators/elements
##    #                          Fault time intervals
##    #                          Simulation intervals
##    #                          Parameters to be evaluated (Electric Power, Mechanical Power, Generator Speed
and All totals in this case)
##    # The program can be run by pressing F5 or double clicking the file
##
    import os, sys
#RunDynamicGenTrippings(strdynfile,FBASE,tfin_gen,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_gen_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE)
#RunDynamicShortCircuits(strdynfile,FBASE,tfin_bus,
FUF1,str_report_folder_dynamic,str_aux_case_folder,str_aux_dyn_folder,dynamic_bus_parameters,voltupperlimit,volt
lowerlimit,BRANCH_RATE,sc_buses)
```

ANEXO 8: USO DE PSSE JUNTO CON PYTHON

CONFIGURACIÓN DE UNA RED EN PSSE

Como en otros simuladores de sistemas de potencia, PSSE se configura y se utiliza por elementos que se unen e interactúan entre sí. A continuación, se muestra una imagen con los elementos que se utilizar convencionalmente para ejecutar análisis en PSSE, los cuales coinciden con los contenidos en el modelo PSSE de la isla de Tenerife:

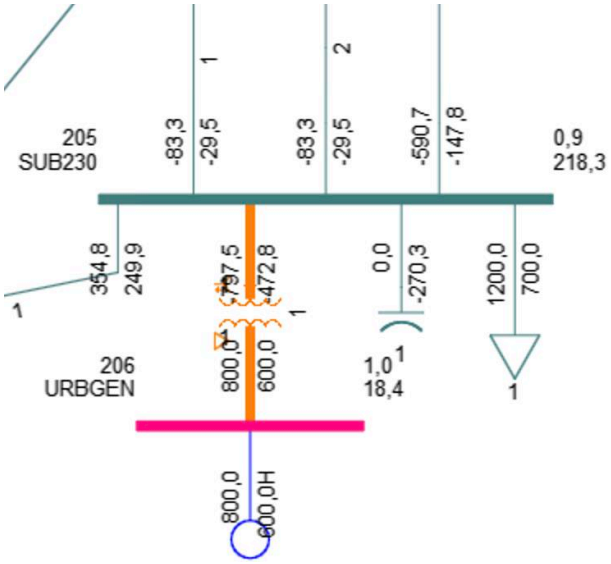


Figura 36 – Modelo simplificado de una red eléctrica en PSSE

La red anterior está compuesta por un generador, un nudo de generación (tipo 2, y en este caso tipo 3 porque es el nudo *swing*), un transformador de generación y un nudo de distribución (tipo 1) con una demanda y un condensador conectados y del cual salen también líneas de transporte. Estos son los elementos básicos con los que se trabajará y los cuales se buscará modificar en este proyecto.

USO DE PYTHON EN CONJUNTO CON PSSE. CREACIÓN DE SCRIPTS CON LA "GRABADORA" DE PYTHON

En la interfaz de PSSE, existe una herramienta que permite grabar en un script de Python una serie de ejecuciones a elegir por el usuario. Su funcionamiento es sencillo, una vez el usuario enciende dicha herramienta, todos los cambios que ejecute el usuario en PSSE (correr un flujo de cargas, encender una línea, añadir un nuevo generador…) quedarán grabados en un fichero Python.

Por ejemplo, a continuación, se muestra el código Python obtenido de la grabadora al apagar una línea y correr un flujo de cargas:

```
# ----------
psspy.branch_chng_3(32,257,r"""1""",[_i,_i,_i,_i,_i,_i],[_f,_f,_f,_f,_f,_f,_f,_f,
_f,_f,_f,_f],[_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f,_f],"")
psspy.fnsl([0,0,0,1,1,0,0,0])
```

La ejecución de scripts de Python se puede realizar incluso con la aplicación cerrada. A continuación, se muestra una rutina para arrancar PSSE y crear un nuevo caso de estudio, obtenido del módulo *Python_to_PSSE*, encargado de importar a PSSE los diccionarios generados por Python que contienen todos los datos del caso de estudio. Resaltados en amarillo quedan los comandos encargados de inicializar PSSE y crear el caso mencionado. En este caso particular, se está creando, para cada archivo .sav que se encuentre en el directorio especificado, un nuevo caso PSSE para cada hora del día:

```
# ----------
# Function
# ----------

#Defines the main function
def
Python_To_PSSE(n_hours,Generators,Demand,str_user_files,str_aux_case_folder,df_co
rrespondencia_gen,df_correspondencia_dem,df_correspondencia_RES,slack_bus):

    #Imports all the necessary modules to run the function
    import os,sys
    import matplotlib.pyplot as plt
```

```
import pandas as pd
import docx
import random
from docx import Document
from docx.shared import Cm, Pt
import matplotlib.pyplot as plt
#from docx.text.parargaph import Paragraph
from docx.styles.style import WD_STYLE_TYPE
from docx.enum.table import WD_TABLE_ALIGNMENT
from docx.enum.table import WD_TABLE_DIRECTION
from docx.enum.text import WD_ALIGN_PARAGRAPH
import collections
import psse34
import psspy
import redirect
redirect.psse2py()

#Defines some parameters to be usedby PSSE
_i=psspy.getdefaultint()
_f=psspy.getdefaultreal()
_s=psspy.getdefaultchar()

#Creates some parameters related to the subsystem
sid=-1
ties=3
flag=1
entry=1

# go through all .sav files in the User_input folder
for file in os.listdir(str_user_files):
    if file.endswith(".sav"):
        psspy.psseinit(20000)
        strfilefdc = file
        strpathfilefdc = str_user_files + '\\' + strfilefdc
        #Creates and fills a .sav file for every hour
        for i in range (n_hours):
            psspy.case(strpathfilefdc)
            saved_file = str_aux_case_folder + '\\' + strfilefdc
```

Como se puede observar, la mayoría de los comandos de Python para PSSE se ejecutan a través de la librería *psspy,* que a su vez contiene dentro de sí todas las funciones correspondientes a cada una de las tareas que se pueden ejecutar en PSSE.

Ejemplos de las tareas son, por ejemplo, *pssy.case(file_name)* → crea un nuevo caso de estudio en PSSE o *list_name = psspy.aswshint* → obtiene una lista con el número de nudo de todos los *shunts* variables del sistema.