UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE SIMULADOR INTEGRADO DE MICROREDES INTELIGENTES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA

SEBASTIÁN EDUARDO FEHLANDT MUÑOZ

PROFESOR GUÍA:
LORENZO REYES CHAMORRO

MIEMBROS DE LA COMISIÓN:
RODRIGO PALMA BEHNKE
NELSON MORALES

SANTIAGO DE CHILE
DICIEMBRE 2012

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: SEBASTIÁN EDUARDO FEHLANDT MUÑOZ
FECHA: DICIEMBRE 2012
PROF. GUÍA: LORENZO REYES CHAMORRO

DESARROLLO DE SIMULADOR INTEGRADO DE MICROREDES INTELIGENTES

En el diseño y operación de redes eléctricas donde existen unidades de generación con recursos intermitentes, es necesario evaluar con anterioridad su funcionamiento en distintos escenarios, para lo que resultan bastante útiles los programas de simulación, de manera de realizar una correcta planificación, diseño y evaluación económica. En la actualidad es común encontrar programas enfocados en redes eléctricas de gran envergadura, donde se omiten ciertas consideraciones de pequeña escala, como el desbalance entre fases y la característica predominantemente resistiva de las líneas de distribución, típicas de una microred. Por lo anterior, surge la necesidad de una herramienta capaz de abordar las características particulares de sistemas eléctricos pequeños con generación distribuida.

El objetivo de este trabajo de título es disponer de una herramienta que incorpore un algoritmo de flujo de potencia trifásico, que permita simular la operación de una microred desbalanceada. Además la aplicación debe incorporar la modelación de las distintas unidades de generación y almacenamiento, y la incorporación del recursos energético solar, mediante un despacho económico que optimice el uso de los recursos.

Se implementa un algoritmo de flujo de potencia trifásico basado en el desacople de las tres mallas de secuencia, mediante compensaciones por inyecciones de corriente, lo que reduce la complejidad del problema matemático. El algoritmo se implementa como una extensión del proyecto MATPOWER que permite calcular flujos de potencia y flujos de potencia óptimos en MATLAB. El algoritmo se valida mediante la comparación con un caso de estudio internacional, obteniéndose un error de 0.0005[p.u.] para los voltajes de fase y 0.811 % para las potencias inyectadas, con un tiempo de ejecución de 0.1[s].

Se incorpora un despacho de unidades que permite dar las consignas de operación a las unidades de generación y almacenamiento de una microred, para un horizonte de simulación determinado. De esta forma, en conjunto con algoritmo de flujo de potencia, es posible simular la operación de la microred en un horizonte de simulación dado.

La herramienta es probada con un caso de estudio basado en la microred real de Huatacondo, desarrollada por el Centro de Energía de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, obteniendo resultados que reflejan su operación real. Obteniéndose una herramienta que constituye un aporte al análisis, estudio y desarrollo de microredes con generación distribuida.

Dedicado a la más grande, mi madre, Edith “Taty” Muñoz Canales.

Agradecimientos

Quiero agradecer a mis compañeros y amigos de la vida por la comprensión en mis meses de desaparición, a mis compañeros de la Universidad por el apoyo y las palabras de aliento. A la gente del Centro de Energía por el apoyo y sugerencias, Fernando, Bernardo, Carlos, Sebastián, Paola, a los profesores Rodrigo Palma, Guillermo Jiménez, y especialmente a mi profesor guía Lorenzo Reyes y al profesor Marcelo Matus por todo el apoyo y todas las dudas resueltas.

Muchas gracias a mi familia por todo el apoyo que me han brindado durante toda mi vida. A mi tío Miguel, mi hermano Christian, mi padre Alberto, a mi hermana Jocelyn, que durante todos estos años de Universidad me ayudó con una mesada para que no pasara pellejerías. A mi polola Silvana por todo el apoyo, compañía, comprensión y sobretodo paciencia. Pero por sobretodo a mi madre Taty, por haber hecho siempre hasta lo imposible para que yo pueda recibir la mejor educación posible y para que nunca me faltara nada. Que hoy esté acá se lo debo a muchas personas, pero más que nadie a ella.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Este trabajo consiste en el desarrollo de una herramienta computacional capaz de simular la operación de microrredes inteligentes de diversas topologías, ante distintas situaciones de demanda, disponibilidad de recursos, entre otros.

El trabajo a realizar contempla el desarrollo del núcleo del simulador y la incorporación de algunos modelos ya existentes de componentes para microrredes. El diseño debe ser modular y absolutamente extensible, de manera de poder añadir distintos componentes y tecnologías sin mayor dificultad.

En ese sentido, cabe aclarar que este trabajo corresponde a una parte del desarrollo conjunto de un equipo, en el que otros miembros de éste han colaborado con el modelamiento matemático de distintos componentes de microrredes, como por ejemplo, los sistemas fotovoltaicos, los sistemas de almacenamiento, la predicción de recursos, entre otros.

1.1. Motivación

En el diseño y operación de redes eléctricas donde existen unidades de generación con recursos intermitentes como energía solar o eólica, es necesario evaluar con anterioridad su funcionamiento en los distintos escenarios posibles. En la actualidad, existen distintos programas computacionales que permiten cumplir con este objetivo mediante la incorporación de herramientas de simulación de distinta índole.

La incorporación de estas fuentes de generación en un sistema eléctrico se puede realizar mediante dos enfoques: la creación de plantas que agrupen varias unidades que se conectan al sistema como un sólo generador, o la incorporación de estas unidades en redes de distribución. El segundo enfoque suele denominarse generación distribuida, y la red en la que se incorpora la generación distribuida suele denominarse microrred.

Los programas de simulación tradicionales están enfocados a redes eléctricas de gran envergadura, donde los desbalances son despreciables, lo que simplifica el modelo eléctrico y los cálculos asociados a éste de forma considerable. Sin embargo, esta simplificación no es posible en el contexto de microrredes ya que al tratarse de redes de distribución, los desbalances son frecuentes y no despreciables. Adicionalmente, estos programas están enfocados al análisis técnico y económico de sistemas eléctricos convencionales donde preponderan las fuentes de generación térmicas e hidráulicas, por lo que no abordan el problema de la disponibilidad de recursos intermitentes, ni la predicción de ésta.

Por otro lado, existen programas que permiten planificar y diseñar microrredes con estas características, pero no permiten simular su operación con el mismo nivel de detalle que los programas mencionados anteriormente.

Para estudiar la incorporación de fuentes de generación distribuida con recursos intermitentes en redes de distribución es necesario abordar ambos enfoques, el de disponibilidad de recursos y su impacto en la operación, y la operación eléctrica en detalle.

Por último, el Centro de Energía de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile ha estado desarrollando modelos y herramientas computacionales orientados a microrredes y generación distribuida. Es por esto que se hace necesario incorporar todos estos desarrollos en una herramienta capaz de abordar los alcances descritos anteriormente, con el fin de constituir una herramienta que permitirá no sólo simular la operación de una microrred, sino sentar las bases para el diseño, desarrollo, ampliación y análisis crítico de éstas, así como el impacto de los distintos medios de generación distribuida en una microrred.

1.2. Alcances

La herramienta computacional es desarrollada con el objetivo de simular el despacho de unidades y la operación en régimen estacionario de una microrred, con el fin de estudiar el comportamiento de ésta ante determinadas situaciones.

Para esto la herramienta incorpora distintos modelos de componentes de microrredes, tales como:

- Modelo estático de componentes típicos de redes eléctricas: barras, transformadores, líneas de transmisión, cargas, condensadores, reactores, compensadores de voltaje y conversores AC/DC y DC/DC.
- Modelo estático de bomba de agua como carga controlada.
- Modelo de generador diesel en régimen permanente.
- Modelación de integración de recurso solar y eólico.
- Modelación de planta fotovoltaica con seguimiento solar en régimen permanente.
- Modelación estática de sistema de almacenamiento en base a baterías de plomo ácido, incluyendo el SoC y SoH de las baterías.

De esta forma, en base a estos modelos y las características topológicas de la red y sus consumos, el programa incluye las siguientes herramientas:

- Solución del predespacho y despacho económico para varios periodos en un horizonte de tiempo determinado.
- Solución del flujo de potencia trifásico de la red, pudiendo ser un flujo de potencia individual, o en base a los resultados del despacho.
- Visualización gráfica de resultados, para todo el horizonte de simulación.

Cabe mencionar que la herramienta está orientada al análisis y estudio, no al monitoreo o control en tiempo real, por lo que la velocidad no es un requerimiento crítico, pero sí deseable.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general del trabajo a realizar consiste en facilitar el estudio y diseño de microrredes inteligentes, con el objetivo de incorporar tecnologías ERNC en sistemas de distribución, a través de una herramienta que permita simular la operación de una microrred en un horizonte dado.

Además, debido a que el objetivo de este trabajo es ser extendido a futuro, es menester que su diseño estructural sea modular, intuitivo, extensible y bien documentado.

1.3.2. Objetivos específicos

- Facilitar la utilización de los modelos matemáticos existentes de algunos componentes de una microrred, mediante su implementación en una herramienta computacional.
- Simplificar la interacción entre el usuario y la herramienta, mediante el desarrollo de una interfaz gráfica amigable.
- Disponer de una herramienta de despacho de unidades que determine las consignas de operación de las unidades de generación y almacenamiento.
- Permitir el cálculo o simulación de las variables eléctricas de una microrred durante su operación, mediante la implementación de un flujo de potencia trifásico que permita simular la operación desbalanceada de una microrred.

1.4. Estructura de la memoria

La estructura utilizada para exponer el trabajo realizado es la siguiente:

- **Capítulo 2. Antecedentes:** Se entregan las bases y se explican los conceptos necesarios para la comprensión y contextualización del trabajo.
- **Capítulo 3. Estado del arte de los algoritmos de flujo de potencia trifásico:** Se realiza una revisión de algoritmos de flujo de potencia trifásico con el fin de comprender las distintas técnicas o enfoques utilizados, identificando las ventajas y desventajas de cada uno de ellos. Este capítulo, en conjunto con el anterior, entregan los conceptos necesarios para comprender el trabajo realizado en los capítulos siguientes.
- **Capítulo 4. Modelo computacional de la herramienta:** Considerando los objetivos y alcances planteados, se desarrolla el modelo computacional de la herramienta. Este modelo está concebido bajo el paradigma de la programación orientada a objetos, por lo que se describe utilizando diagramas de clase UML (*Unified Modelling Language*) que permiten explicar de forma gráfica la relación entre los distintos objetos que componen el programa y sus características.

- **Capítulo 5. Desarrollo del flujo de potencia trifásico y validación:** En base al modelo desarrollado en el capítulo anterior se desarrolla e implementa un algoritmo de flujo de potencia trifásico. En este capítulo se describen los modelos utilizados por el algoritmo para representar los distintos componentes de la red, el algoritmo final, su implementación computacional y su validación mediante la comparación con un caso de estudio internacional.
- **Capítulo 6. Implementación computacional de la herramienta:** Considerando el modelo computacional y el flujo de potencia trifásico desarrollado, se describe la implementación computacional de la herramienta. Específicamente, se describe el sistema de archivos y paquetes, el desarrollo de la interfaz gráfica, la incorporación del despacho de unidades, y la incorporación del flujo de potencia trifásico.
- **Capítulo 7. Utilización y visualización de un caso de estudio:** Con la herramienta terminada, se describe la utilización del programa mediante la ejecución de un caso de ejemplo, basado en la operación real en una microrred. Además se muestran y analizan los resultados obtenidos.
- **Capítulo 8. Conclusiones:** Finalmente se enumeran las conclusiones del trabajo realizado y se proponen trabajos a realizar en el futuro.

Capítulo 2

Antecedentes

2.1. Microrredes

A continuación, se explican los conceptos esenciales relacionados con microrredes.

2.1.1. Orígenes: Generación distribuida

Actualmente se ha producido una nueva tendencia en la generación de energía eléctrica, la que consiste en la incorporación de pequeñas unidades de generación a nivel local en redes de distribución [1]. De esta forma, vista desde el sistema interconectado, la red de distribución puede ser considerada tanto un consumo como un generador, dependiendo de la existencia o no de excedentes de energía que se inyecten a la red [2].

Esta forma de generación posee distintos dispositivos, llamados Recursos de Energía Distribuidos (DER: Distributed Energy Resources), entre los que destaca la incorporación de distintos DERs basados en ERNC [3], que por sus tamaños y complejidades técnicas particulares se dificulta su inserción en sistemas de potencia a gran escala. En este sentido la generación distribuida no sólo permite la incorporación de estas tecnologías en los sistemas eléctricos, sino que permite la interconexión de varias de estas unidades entre sí, lo que acompañado de una matriz diversa de tecnologías y energéticos primarios permite minimizar el riesgo de desabastecimiento [4].

Evidentemente la generación distribuida implica una serie de desafíos entre los que se encuentran la necesidad de mejorar el sistema de protecciones [1], consideraciones en la medición y tarificación a los clientes y DERs del sistema de distribución, control de voltaje y frecuencia debido a las posibles intermitencias en la disponibilidad de recursos para algunos DERs particulares [2].

2.1.2. Definición de microrred

Hasta el momento no existe una definición definitiva de microrred, sin embargo se puede definir como un sistema de distribución eléctrico en el que pueden coexistir cargas o consumos, cargas controlables, sistemas de almacenamiento y unidades de generación distribuidos, pudiendo estar conectado o no a la red externa [4].

Dentro de los DER se consideran [3]:

- **Generadores Distribuidos (Distributed Generation DG):** Corresponden a las fuentes de generación de la microrred, sean estas renovables o no renovables.
- **Acumuladores Distribuidos (Distributed Storage DS):** Corresponden a todos los dispositivos de acumulación de energía, ya sean electroquímicos, de presión, gravitacionales, etc., tales como bancos de baterías, volantes de inercia, entre otros.
- **Cargas Controladas:** Dado que las microrredes corresponden a sistemas a nivel de distribución, son bastante desbalanceadas. Esto, sumado al bajo tamaño del sistema hacen que la variación de carga en este sea bastante importante. Por lo que un sistema de control de demanda o carga permite optimizar el funcionamiento de la microrred [4].

De esta forma una microrred debería ser capaz de [2]:

- Controlar localmente la generación y consumo distribuidos, reemplazando el despacho global del sistema eléctrico del país o región.
- Desconectarse de la red principal cuando se produzcan perturbaciones u otras condiciones anormales.
- Operar tanto en isla como conectada a la red.

Por otro lado, el despacho de la microrred se denomina centralizado si posee una unidad de control central que proporciona las consignas de potencia, voltaje y frecuencia a cada unidad, enfoque utilizado, por ejemplo, en [5]. En cambio si la operación es determinada por los mismos controladores de cada unidad y su interacción y comunicación entre sí, se dice que el despacho es descentralizado, enfoque utilizado en [2].

2.1.3. Arquitectura de una microrred

A continuación se presenta un esquema de microrred inserta en una sistema de distribución, que incorpora DGs, DSs, cargas locales, un switch de interconección al sistema de distribución, controladores para los DERs (Microsource Controller MC) y un sistema de control central (CC).

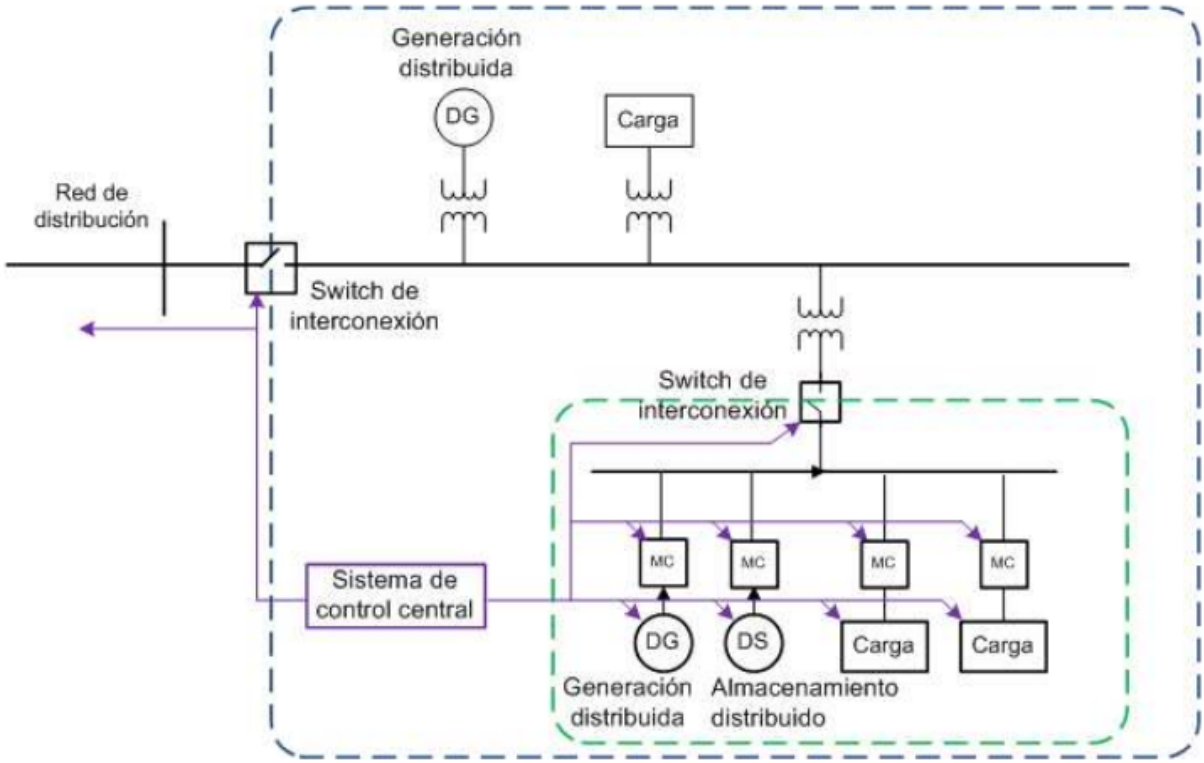

Figura 2.1: Arquitectura de una microrred. Fuente: [3].

Los componentes mencionados anteriormente presentan las siguientes funciones principales:

- Los MC son los controladores de los DER, controlan el funcionamiento y operación de éstos en base a las órdenes enviadas por el CC [1].

- El switch de interconección permite sincronizar y conectar la microrred con la red de distribución externa [1].
- El CC permite controlar el funcionamiento global de la microrred, mediante dos funciones primordiales:
  - Gestionar la energía: es decir despachar las unidades en base a criterios de optimización, generalmente minimizar los costos globales. De esta forma el CC entrega las consignas de potencias activa y reactiva a los MC de cada unidad [1].
  - Proteger el sistema de forma coordinada: es decir, adaptar el funcionamiento de la microrred ante contigencias, tales como fallas, salida intempestuosa de unidades o consumos, entre otras [3].
- Otra función del CC es el control sobre el switch de interconección, determinando su funcionamiento en isla o conectado a la red [3].
- Adicionalmente la microrred necesita de un sistema de comunicaciones seguro y confiable que permita comunicar el CC con los MC y el switch de interconección, asegurando una correcta coordinación entre éstos. Típicamente se utiliza un sistema SCADA, debido a su estabilidad y compatibilidad con distintos dispositivos [3].

2.2. Redes inteligentes

El concepto de red inteligente o *smart grid* aparece con la necesidad de robustecer los sistemas eléctricos. Aprovechando los crecientes avances en tecnología y comunicaciones es posible automatizar procesos relativos a la operación y protección del sistema, optimizando la utilización de sus recursos, previendo fallas y automatizando la toma de decisiones [4].

La definición de red inteligente es un poco difusa aún, por ejemplo la visión estadounidense de las redes inteligentes incorpora políticas de seguridad nacional, mercados competitivos, tecnologías de información, superconductores. almacenamiento de energía, equipos de electrónica de potencia, regulaciones medioambientales, entre otros [6]. Para la Unión Europea en cambio las redes inteligentes aportan accesibilidad, confiabilidad, flexibilidad y diversificación de la matriz energética, con su consecuente reducción de costos y riesgo. Por otro lado, la Agencia Internacional de Energía declara las redes inteligentes como esenciales para el desarrollo económico, seguridad del sistema eléctrico e inserción de energías renovables, que permitirán mitigar el cambio climático y otros efectos medioambientales [7].

En [8] se describen tres tipos principales de elementos que componen una red inteligente: Módulos de Gestión Activa de la Red, Dispositivos FACTS y Sistemas de Almacenamiento de Energía.

2.2.1. Módulos de gestión activa de la red

Entre los módulos de gestión activa de la red (Active network management modules) se encuentran los siguientes:

- **Medición inteligente (Smart metering):** Consiste en la medición en tiempo real de los consumos, mediante un sistema de comunicación con el control central de la red. De esta forma es posible aplicar políticas o estrategias de control de demanda más específicas, identificar fallas o interrupciones de suministro de manera más focalizada y rápida, e identificar robos de energía (mediante bypasseo del medidor o alguna otra forma).
- **Control de demanda (Demand response):** En conjunto con equipos de medición inteligente, permite una comunicación bidireccional con el control central de la red, permitiendo implementar estrategias de control de demanda, con el fin de desplazar los peaks de demanda de los clientes de forma que no se produzca el fenómeno de horas de punta. Esto permite disminuir la demanda punta del sistema que es la que determina la capacidad que deben tener las instalaciones, las proyecciones en la ampliación de la generación, en definitiva, mejorar la eficiencia del sistema.

- **Monitoreo activo térmico (Active thermal rating monitoring, ATRM):** Actualmente los flujos de potencia son medidos a nivel de transmisión. La incorporación de estos dispositivos permite medir los flujos en tramos a nivel de distribución, lo que permite identificar congestiones y modificar automáticamente la topología, con el fin de minimizar las pérdidas e inconvenientes que esto pueda causar.

- **Gestión de contingencias:** La incorporación de sensores inteligentes en las redes, asociados a un sistema de georeferenciación de la red, permite la identificación de contingencias y los lugares exactos de su ocurrencia, lo que permite acelerar la operación de los equipos de mantención en terreno.

- **Monitoreo activo de nivel de falla (Active fault level monitoring):** Permite el monitoreo dinámico y en tiempo real de los niveles de falla en los distintos puntos de la red, lo que permite la modificación de la topología de ésta con el fin de disminuirlos en caso de exceder el máximo nivel que soportan las protecciones instaladas.

2.2.2. Dispositivos FACTS

Los FACTS o *"Flexible AC Transmission Systems"* son dispositivos que pueden ser usados para optimizar redes de distribución y transmisión. Estos permiten realizar control de potencia, voltaje y factor de potencia, disminuir las corrientes de falla, reducir pérdidas en lineas de transmisión, entre otros. Dentro de estos dispositivos destacan:

- **Limitadores de corriente de falla (Fault-Current limiters, FCL):** Utilizan superconductores de alta temperatura, permiten disminuir las corrientes de falla en sistemas de transmisión y distribución sin modificar la impedancia de la red.

- **Dispositivos de control de tensión y soporte para barras:** Existen distintos tipos de dispositivos de control de tensión, tales como los clásicos reactores y condensadores controlados por tiristores (TCR y TSC) que retiran o inyectan potencia reactiva en las barras. Pero además existen dispositivos de control automático de voltaje para las barras, los Static Var Compensators SVC) y los Static Shynchronoys Compensators (STATCOM), que permiten controlar las variaciones de voltaje en tiempo real. La principal diferencia entre estos dos es que la máxima potencia reactiva de los SVC depende del voltaje, mientras que en los STATCOM es independiente del voltaje.

- **Control de la impedancia de la red:** El *Unified Power Flow Control* (*UPFC*) permite incrementar la capacidad de una línea de transmisión o bus de voltaje, controlando la magnitud y ángulo de los voltajes en ambos extremos de ésta.

2.2.3. Sistemas de almacenamiento de energía y vehículos eléctricos

El almacenamiento de energía permite contribuir a solucionar ciertos problemas de las redes tales como partida en negro del sistema, seguimiento de carga, disminución de la demanda punta, integración de energías renovables, entre otros.

Además los vehículos eléctricos constituirán cargas móviles, correspondientes al proceso de carga de sus baterías, que se podrán conectar a la red en distintos puntos. Sin embargo, siguiendo con el enfoque de red inteligente, los vehículos pueden acomodarse con el fin de utilizar su capacidad de almacenamiento de energía también.

2.3. Análisis técnico comparativo de lenguajes de programación

Con el fin de elegir el lenguaje de programación adecuado para el desarrollo de la herramienta se realiza una comparación de distintos lenguajes, esta comparación incluye tanto aspectos cualitativos como cuantitativos, tales como la velocidad de ejecución, uso de memoria, etc.

2.3.1. Aspectos cualitativos

A continuación se describen los aspectos cualitativos considerados:

- **Utilización de clases y objetos:** Existe un paradigma dentro de la programación que es la programación orientada a objetos, en la cuál todas las variables y datos se modelan como objetos con atributos y funciones definidas a través de una clase. Esto permite un modelamiento y desarrollo modular y más abstracto. Algunos lenguajes han sido desarrollados bajo esta filosofía, mientras que otros, sin aferrarse completamente al paradigma, sí incorporan clases y objetos, dejando las variables más elementales como tipos primitivos, este es el caso de Java.

- **Simplicidad:** Se dice que un lenguaje es simple cuando presenta una sintaxis simple que permite entender el código sin mayores complicaciones. Por otro lado la simplicidad de un lenguaje también tiene relación con la forma de tratar las variables o datos, por ejemplo en C los string se almacenan directamente en memoria, guardando una referencia a ese sector de memoria en una variable llamada puntero, mientras que en Java se tratan como un objeto, lo que hace que su utilización sea mucho más simple en comparación a C.

- **Seguridad:** Principalmente se refiere a la ausencia de álgebra de punteros. Como se explica en el punto anterior, los punteros señalan información almacenada en un sector específico de la memoria, se denomina álgebra de punteros a la propiedad de algunos lenguajes (como C) de acceder y modificar sectores intermedios de memoria al operar matemáticamente el puntero (por ejemplo sumándole un cierto número, lo que hace que se desplace una cierta cantidad de bits en la memoria). Esto puede producir errores graves y difíciles de detectar, por lo que en general los lenguajes modernos no lo permiten.

- **Portabilidad:** La portabilidad es la posibilidad de ejecutar un programa, ya compilado, en distintas plataformas, ya sea distintos procesadores (32 y 64 bits), o distintos S.O (Unix, Windows, etc).

- **Concurrencia:** Es la posibilidad de realizar programación concurrente o en paralelo, es decir, separar el programa en etapas que realizan labores distintas en paralelo, lo que es más útil con varios procesadores. Esto permite, por ejemplo, dividir un problema matemático con estrategias de cálculo en paralelo o que un mismo programa pueda realizar tareas independientes en paralelo. Existen 2 tipos principales de paralelismo:

    - **Threads:** También conocidos como “Hilos” o “Procesos Livianos”, son simplemente segmentos de código que se ejecutan en paralelo, comparten las variables y memoria utilizadas ya que son parte del mismo proceso, lo que causa que un problema en la ejecución de un thread, como por ejemplo una excepción que termina el programa, afectará a los demás threads.
    - **Procesos pesados:** Se llaman de esta forma porque son un clon del proceso padre completo, es decir clonan las variables existentes en el padre, por lo que estas son independientes entre uno y otro. Esto permite mayor independencia entre los procesos, lo que además brinda mayor robustez al programa, con respecto a los threads, pero a su vez exige más recursos y mecanismos más complejos de intercambio de información entre ellos. [9]

- **Tipos de datos:** Actualmente existen dos tipos de datos:
  - **Tipos estáticos:** Las variables son de un tipo definido que no puede cambiar, es decir, una vez que una variable se define con un determinado tipo no podrá almacenar una variable de un tipo distinto. Las verificaciones de tipos se realizan en tiempo de compilación.
  - **Tipos dinámicos:** Los tipos de variables se pueden cambiar, es decir, a lo largo del programa a una misma variable se le pueden asignar distintos tipos de datos. Las verificaciones de tipos se realizan en tiempo de ejecución, lo que exige pruebas más exhaustivas. [10]
- **Garbage collection:** Con los lenguajes antiguos el programador debe liberar la memoria que se deja de utilizar. El *Garbage Collector* se encarga de esto de forma automática durante la ejecución, lo que simplifica la tarea del programador. Cuando se determina que un dato deja de ser utilizado, y no volverá a ser utilizado, su memoria es liberada.

2.3.2. Aspectos cuantitativos

A continuación se describen los aspectos cuantitativos considerados:

- **Velocidad:** Consiste en el tiempo que toma un programa escrito en un determinado lenguaje, en realizar una tarea determinada. También existe el tiempo de compilación, que consiste el tiempo que tarda un programa en compilar. Sin embargo esto es menos importante ya que lo que determina la velocidad del programa es el tiempo de ejecución.
- **Uso de memoria:** Consiste en el uso de memoria RAM que hace el programa durante su ejecución. También se puede medir el uso de CPU, pero se considera menos importante, ya que además depende del sistema utilizado.

2.3.3. Lenguajes a comparar

Los lenguajes a comparar son los siguientes:

- C
- C++

- Go (Lenguaje desarrollado por Google)
- Java
- C#
- Python
- Matlab (no es un lenguaje en sí, sino una plataforma de desarrollo)

2.3.3.1. Lenguaje C

Es un lenguaje creado entre 1970 y 1972 por Brian Kernighan y Dennis Ritchie para escribir el código del sistema operativo UNIX. Es un lenguaje que conjuga la abstracción de los lenguajes de alto nivel con la eficiencia del lenguaje máquina. [11]

Sus características se resumen en:

- **Utilización de Clases y Objetos:** No posee.
- **Simplicidad:** Baja, la sintaxis es compleja y suele ser difícil entender código escrito por otra persona.
- **Seguridad:** Baja, sí posee álgebra de punteros.
- **Portabilidad:** Baja, interactúa directamente con las señales e interfaces del sistema
- **Concurrencia:** threads y procesos pesados (fork)
- **Tipos de Datos:** Estáticos
- **Garbage Collection:** No posee.
- **Velocidad:** Alta
- **Uso de Memoria:** Bajo

2.3.3.2. Lenguaje C++

Es una extensión de C al paradigma de Programación Orientada a Objetos, se dice que es un “Lenguaje Multiparadigma”, ya que pese a que los incorpora, no es un lenguage netamente orientado a objetos. [12]

Sus características se resumen en:

- **Utilización de Clases y Objetos:** Sí.
- **Simplicidad:** Baja, su sintaxis es bastante similar a la de C.
- **Seguridad:** Baja, sí posee álgebra de punteros.
- **Portabilidad:** Baja, idem C.
- **Concurrencia:** threads y procesos pesados (fork).
- **Tipos de Datos:** Estáticos.
- **Garbage Collection:** No.
- **Velocidad:** Alta.
- **Uso de Memoria:** Bajo.

2.3.3.3. Lenguaje Go

Es un lenguaje libre desarrollado por Google y anunciado oficialmente el año 2009. Orientado a la programación de software de sistema. Sus objetivos son: eficiencia, seguridad, velocidad, buen soporte para concurrencia y simplicidad. [13]

Sus características se resumen en:

- **Utilización de Clases y Objetos:** No pero permite crear estructuras de datos y asignarles funciones específicas.
- **Simplicidad:** La sintexis es bastante simple, orientada a la comodidad del programador, con el fin de maximizar su rendimiento.
- **Seguridad:** Seguro, no posee álgebra de punteros.
- **Portabilidad:** Sí.
- **Concurrencia:** No posee ni threads ni procesos pesados, sino que posee las denominadas “goroutines” que poseen un operador especial de comunicación, lo que facilita el intercambio de información entre ellas.
- **Tipos de Datos:** Estáticos.
- **Garbage Collection:** Sí.
- **Velocidad:** Alta, pero menor a C y C++.

- **Uso de Memoria:** Bajo.

Entre sus particularidades presenta números complejos como tipos de datos primitivos.

2.3.3.4. Lenguaje Java

Java es un lenguaje orientado a objetos desarrollado en los años 90 por Sun Microsystems. Sus objetivos son ser un lenguaje completamente portable, seguro y simple.

Sus características se resumen en:

- **Utilización de Clases y Objetos:** Sí.
- **Simplicidad:** Más simple que C y C++ pero menos que Go.
- **Seguridad:** Seguro, no posee álgebra de punteros.
- **Portabilidad:** Sí, a través de la máquina virtual de Java o Java Virtual Machine (JVM). El código java pasa por un compilador que genera un bytecode, análogo a los códigos de bajo nivel, el cuál es interpretado y ejecutado por la JVM. Luego pasa por un compilador JIT (Just in Time) y se obtiene el código de bajo nivel. De esta forma la JVM es el nexo entre el sistema de bajo nivel y el programa. Tal como se muestra en la siguiente imagen:

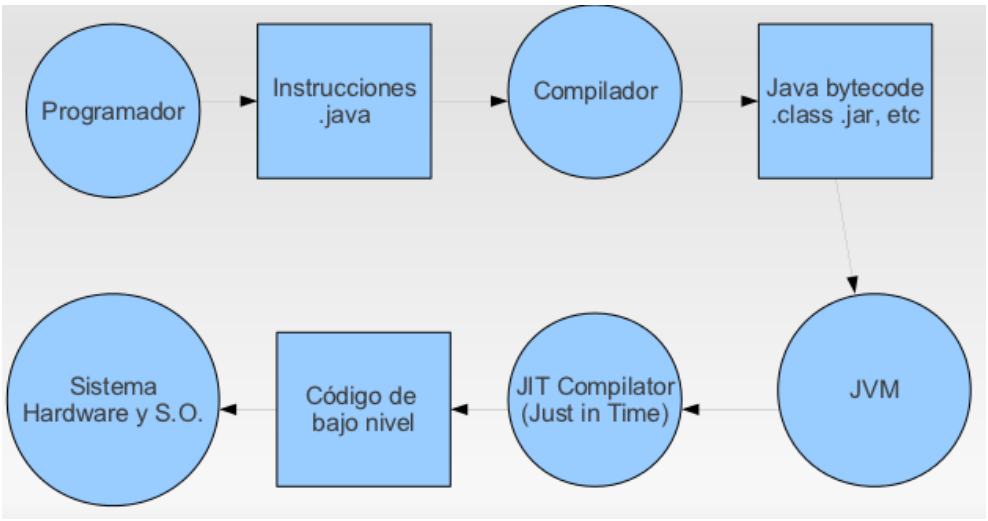

Figura 2.2: Esquema de compilación Java. Elaboración propia.

- **Concurrencia:** Posee threads.

- **Tipos de Datos:** Estáticos.
- **Garbage Collection:** Sí, realizada por la JVM

2.3.3.5. Lenguaje C#

C# es un lenguaje orientado a objetos desarrollado por Microsoft en 2001, está basado en Java, por lo que posee la mayoría de sus características salvo la portabilidad. Forma parte de la plataforma .NET, que puede ser añadida a Windows.

Sus características se resumen en:

- **Utilización de Clases y Objetos:** Sí.
- **Simplicidad:** Similar a Java, su sintaxis es bastante similar, sin embargo, se puede considerar un poco más simple ya que permite sobrecargar operadores.
- **Seguridad:** Al ser bastante similar a Java se puede decir que son igualmente seguros.
- **Portabilidad:** Solo entre plataformas con algún sistema que soporte .NET, que es una plataforma desarrollada por Microsoft con el fin de integrar todos sus lenguajes en una plataforma independiente del hardware, no portable a otros sistemas operativos. Su funcionamiento es similar al de Java pero integrando los demás lenguajes de Microsoft, primero el código es compilado, siguiendo un Common Language Specification (CLS), y se obtiene un Common Intermediate Language (CIL), el cuál es procesado por el Common Language Runtime (CLR), luego es compilado por un compilador JIT y llevado a código de bajo nivel. Como se muestra a continuación:

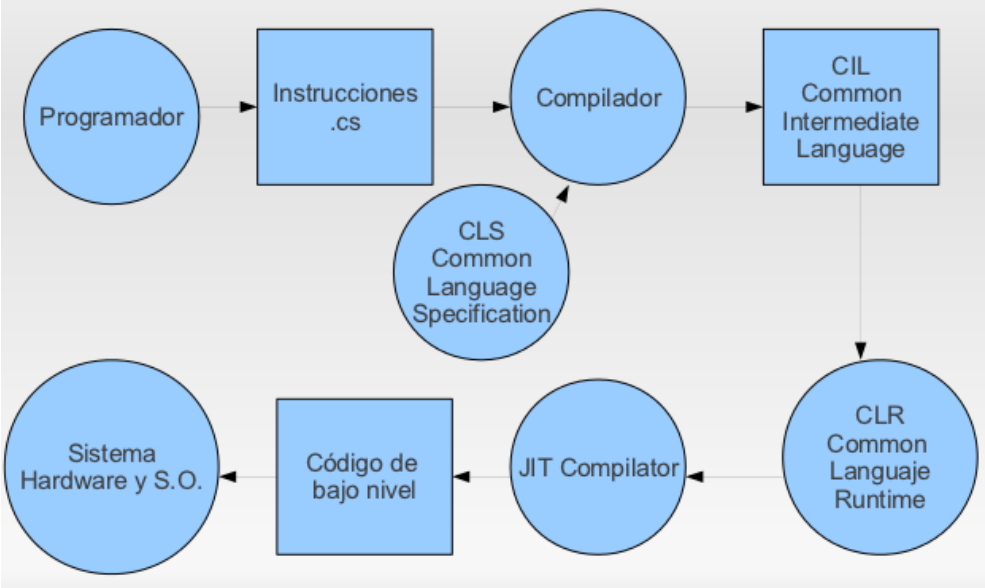

Figura 2.3: Esquema de compilación C#. Elaboración propia.

- **Concurrencia:** Threads
- **Tipos de Datos:** Estáticos
- **Garbage Collection:** Sí, realizada por el CLR.

2.3.3.6. Lenguaje Python

Python fue creado en 1991 por Guido Van Rossum, actualmente es administrado por la Python Software Foundation y posee licencia de fowtware libre. Es un lenguaje de alto nivel cuyo sintaxis es muy limpia y favorece un código legible. [14]

Sus características se resumen en:

- **Utilización de Clases y Objetos:** Sí posee.
- **Simplicidad:** Más simple que Java.
- **Seguridad:** Similar a Java, no posee álgebra de punteros.
- **Portabilidad:** Sí.
- **Concurrencia:** Threads y procesos pesados.
- **Tipos de Datos:** Dinámicos.
- **Garbage Collection:** Sí.

2.3.3.7. Lenguaje M (Matlab)

Matlab no es un lenguaje de programación propiamente tal, sino es un programa orientado a la resolución de problemas matemáticos. Posee un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Matlab permite manipular matrices y números complejos directamente, representar resultados mediante gráficos y funciones rápidamente, entre otros. Además incluye un editor de interfaces de usuario *GUI* llamado GUIDE y una herramienta de simulación multidominio, Simulink. Su principal desventaja es que no posee *garbage collector*, sin embargo provee una función para ejecutar un proceso de *garbage collection* [15].

Por otro lado, existen alternativas libres a Matlab tales como GNU Octave, en adelante Octave, que incorpora las mismas funcionalidades que Matlab. Octave está licenciado bajo la licencia *GPL* y puede descargarse en `http://www.gnu.org/software/octave/`.

La características del lenguaje M se resumen en:

- **Utilización de Clases y Objetos:** Sí
- **Simplicidad:** Similar a Python.
- **Seguridad:** Similar a Python.
- **Portabilidad:** Sí, dependiendo de la disponibilidad de una versión de Matlab para el sistema a utilizar.
- **Concurrencia:** Threads
- **Tipos de Datos:** Dinámicos
- **Garbage Collection:** No posee.

2.3.4. Comparación de velocidad

En el sitio web `http://alioth.debian.org/` [16] se encuentra un proyecto consistente en una serie de pruebas a distintos lenguajes de programación, llamada *"The Computer Language Benchmarks Game"* `http://shootout.alioth.debian.org/`. Las pruebas consisten en la medición de recursos consumidos, y tiempo de ejecución de programas que realizan una misma tarea pero escritos en distintos lenguajes.

El sitio dispone de resultados para un conjunto de 14 tareas, siendo posible comparar los resultados obtenidos entre dos lenguajes analizando su velocidad, uso de memoria y uso de código. Además es posible obtener gráficos de comparación entre dos o más lenguajes a elección, por tarea o para el conjunto de tareas. Las pruebas se realizan en 4 plataformas distintas:

- x86 Ubuntu™ Intel® Q6600® one core.
- x64 Ubuntu™ Intel® Q6600® one core.
- x86 Ubuntu™ Intel® Q6600® quad-core.
- x64 Ubuntu™ Intel® Q6600® quad-core.

A continuación, se presentan los resultados obtenidos para el conjunto de tareas comparando los lenguajes mencionados anteriormente, exceptuando el lenguaje M, ya que no se encuentran resultados para éste en el sitio. Los gráficos muestran el tiempo utilizado normalizado al programa más rápido, es decir: Tiempo utilizado / Tiempo utilizado por el programa más rápido.

Cabe señalar que cada cuadro representa el 50 % central de los resultados y la barra central representa la mediana.

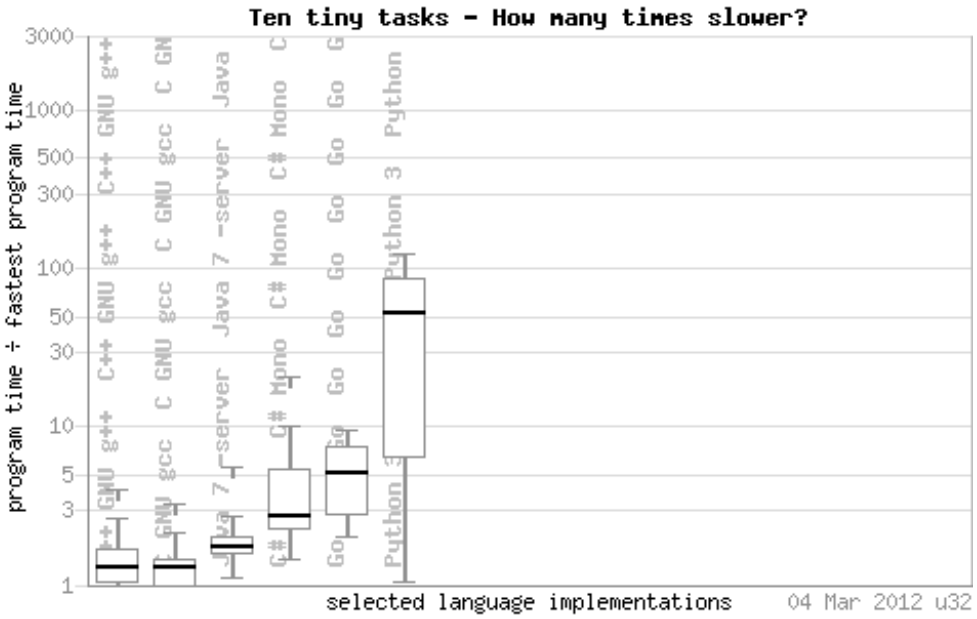

Figura 2.4: Comparación de velocidad de ejecución x86 Ubuntu™ Intel® Q6600® one core. Fuente: [16].

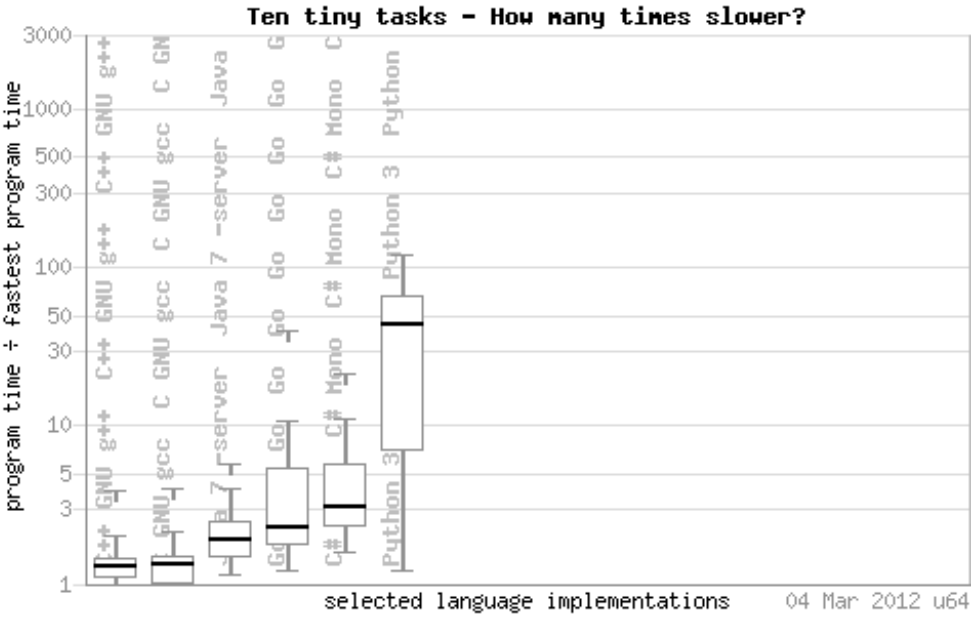

Figura 2.5: Comparación de velocidad de ejecución x64 Ubuntu™ Intel® Q6600® one core. Fuente: [16].

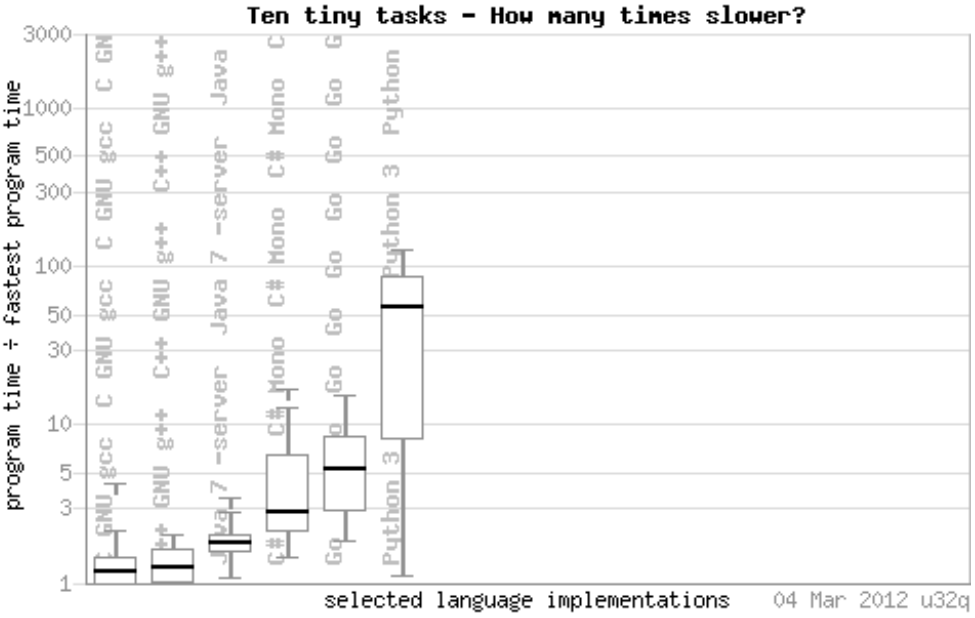

Figura 2.6: Comparación de velocidad de ejecución x86 Ubuntu™ Intel® Q6600® quad-core. Fuente: [16].

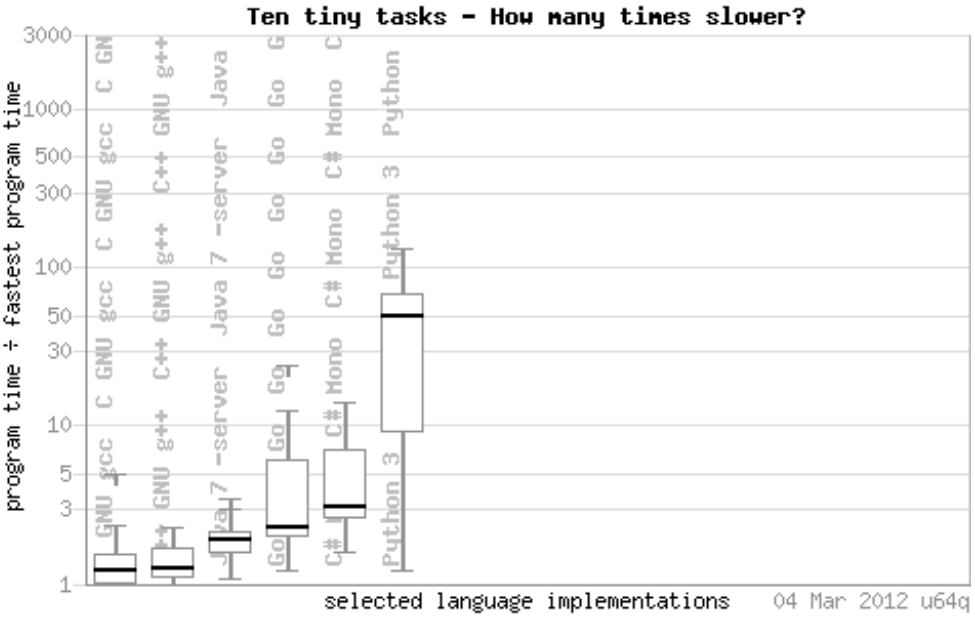

Figura 2.7: Comparación de velocidad de ejecución x64 Ubuntu™ Intel® Q6600® quad-core. Fuente: [16].

2.3.5. Resumen comparativo

Finalmente, en la siguiente tabla se resumen los resultados de la comparación, tanto en los aspectos cualitativos como los resultados cuantitativos obtenidos de [16]. Para los criterios de simplicidad, velocidad y uso de memoria los números en la tabla representan un orden de mayor a menor según orden ascendente.

Tabla 2.1: Resumen comparativo, lenguajes de programación. Elaboración propia.

Lenguaje	**C**	**C++**	**Go**	**Java**	**C#**	**Python**	**Matlab**
Objetos	No	Sí	No	Sí	Sí	Sí	Sí
Simplicidad	6	6	5	4	3	1	2
Álgebra de Punteros	Sí	Sí	No	No	No	No	No
Portabilidad	No	No	Sí	Sí	No	Sí	Sí
Concurrencia	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Tipos de Datos	Est.	Est.	Est.	Est.	Est.	Din.	Din.
Garbage Collector	No	No	Sí	Sí	Sí	Sí	No
Velocidad	1	1	3	2	3	4	-
Uso de Memoria	1	2	3	6	5	4	-

Al observar el cuadro comparativo se aprecia que los lenguajes más rápidos y que utilizan menos recursos, en este caso C y C++, son los de sintaxis más compleja y menor seguridad. Estos lenguajes son recomendables para aplicaciones en que la velocidad es un requerimiento fuerte o donde se requiera comunicación con sistemas de bajo nivel.

Por el contrario, los lenguajes más simples son los de tipos dinámicos y más lentos, en este caso Python y M, lo que es lógico debido a que los tipos dinámicos simplifican la sintaxis. Sin embargo, su desventaja es que realizan la comprobación de tipos en tiempo de ejecución, por lo que el código debe pasar una prueba menos antes de compilar, lo que puede dejar pasar errores que en un lenguaje de tipo estático se detectarían en tiempo de compilación.

Estos lenguajes son recomendables para aplicaciones simples en que la velocidad no sea un requerimiento fuerte y donde la estructura del programa no sea muy compleja, o para el desarrollo de módulos que realicen labores determinadas dentro de un proyecto mayor. Por ejemplo para la implementación de herramientas de cálculo resulta bastante útil utilizar el lenguaje M, ya sea mediante Matlab u Octave, debido a la facilidad para realizar cálculos matemáticos y la gran variedad de funciones matemáticas que incorpora.

Por otro lado, dentro de los lenguajes de tipo estático y de mayor simplicidad y seguridad se encuentran Java, C# y Go. Go no posee modelamiento de clases y objetos, además es un lenguaje relativamente nuevo por lo que no hay mucha experiencia en su utilización. C# y Java en cambio son lenguajes con basta utilización en desarrollos de distinta índole, sus características son bastante similares, teniéndose como principal diferencia que C# carece de portabilidad, mientras que Java fue concebido con este objetivo.

Debido a los requerimientos de la herramienta, las ventajas de los tipos estáticos para aplicaciones complejas, a que la velocidad no es un requerimiento crítico, y a que es deseable que la herramienta sea portable se elije Java como lenguaje principal. Así, el desarrollo de la herramienta y el modelo de los dispositivos eléctricos se realiza en un lenguaje que posee las ventajas de los tipos estáticos y una sintaxis simple. No obstante, esto no implica que no se puedan utilizar otros lenguajes para la incorporación de ciertos módulos o herramientas.

Capítulo 3

Estado del arte de los algoritmos de flujo de potencia trifásico

Uno de los principales aportes del simulador integrado es el cálculo de flujos de potencia trifásicos. Existen distintos métodos para calcular flujos de potencia balanceados, donde el problema se reduce a un equivalente monofásico. Entre éstos destaca el método de Newton Raphson, por su rapidez y convergencia. Sin embargo, cuando la topología de las redes y/o las cargas son desbalanceadas, estos métodos pierden validez.

En esta sección se presenta una breve revisión bibliográfica de los principales enfoques con los que se aborda este problema, los que se pueden agrupar en tres [17]:

1. Los métodos que consideran la estructura primordialmente radial de las redes de distribución.
2. Los que se basan en métodos iterativos clásicos para sistemas balanceados, principalmente Newton Raphson y Newton Raphson Desacoplado Rápido.
3. Los que utilizan componentes de secuencia.

Cabe mencionar que los nombres de los métodos expuestos a continuación no son oficiales y solo se han puesto como referencia. Asimismo, no se exponen los detalles de los cálculos realizados, salvo que tenga importancia para la comprensión del algoritmo.

3.1. Criterios de Comparación

Para realizar la comparación se definen a continuación algunos criterios que permiten comparar los métodos. Pese a que en los documentos estudiados se presentan resultados tales como tiempo de ejecución y cantidades de iteraciones, no son comparables ya que los casos de estudio, los lenguajes de programación y las plataformas donde fueron realizadas las pruebas son distintos. Para poder realizar una comparación fidedigna de estos algoritmos debieran ser programados todos en el mismo lenguaje, y probados en la misma plataforma, utilizando los mismos casos de estudio y criterios de convergencia, lo que está fuera del alcance y los objetivos de este trabajo.

3.1.1. Limitaciones en topología de la red

Los sistemas eléctricos suelen clasificarse en tres tipos dependiendo de su topología [18]:

1. **Topología radial:** son redes en que desde una fuente o subestación salen uno o más alimentadores que pueden ramificarse pero nunca se conectan a sí mismos en otro punto ni a otros alimentadores, formando una estructura similar a un árbol. Esta topología es la más económica pero la que ofrece menor seguridad, ya que ante una falla en un alimentador, todas las cargas “aguas abajo” de ésta quedarán desabastecidas. Por lo general las redes de distribución poseen topologías radiales.
2. **Topología en anillo:** son similares a las topologías radiales salvo que poseen algunos *loops*, bucles o anillos que permiten aumentar la seguridad, ya que si se produce una falla en un bucle, éste se rompe en ese punto, quedando ambos extremos alimentados.
3. **Topología enmallada:** son redes en que todos las ramas son anillos, por lo que se asemejan a una malla. Esta topología ofrece una seguridad mucho mayor pero a su vez eleva considerablemente el costo. Los sistemas de transmisión en Europa y Estados Unidos tienen topología enmallada.

Algunos algoritmos ([19], [20]) se basan en la suposición de redes primordialmente radiales, pudiendo o no incorporar algunos anillos. En el caso más general el algoritmo será capaz de trabajar indistintamente con redes de topología radial, anillo o enmallada.

3.1.2. Desbalances Topológicos

Una red puede ser desbalanceada debido a dos factores principalmente:

1. **Desbalance de cargas y/o generación:** corresponde al caso en que sólo las cargas y/o los generadores presentan desbalances en sus respectivos consumos e inyecciones, estando la red topológicamente balanceada. Si bien los generadores trifásicos por construcción son balanceados, existe la posibilidad de que en una microrred existan unidades de generación monofásicas, o incluso unidades conectadas mediante inversores que admiten desbalances en su inyección.
2. **Desbalance completo:** corresponde al caso en que toda la red puede estar desbalanceada, es decir, tanto los consumos e inyecciones como las impedancias y topología de la red.

Un algoritmo de flujo de potencia trifásico puede soportar desbalances de cargas y/o generación pero no en la topología, o bien soportar ambos.

3.1.3. Ramas shunt

Con el fin de simplificar cálculos es frecuente reducir una red eléctrica a un modelo que solo posea ramas serie, sin embargo la resolución computacional de flujos de potencia por lo general permite incorporar ramas *shunt* al modelo. La capacidad de un algoritmo de incorporar estas ramas al modelo es algo a tener en cuenta ya que podría limitar la generalidad del algoritmo.

3.1.4. Implementación del método de Newton Raphson

El método de Newton Raphson es un método iterativo de resolución de flujos de potencia basado en el método numérico de Newton. A groso modo el método de Newton Raphson calcula los voltajes y sus ángulos en las barras del sistema a partir de las inyecciones potencia activa en las barras PV y PQ, las inyecciones de potencia activa en las barras PQ, las consignas de voltaje de las barras PV y un voltaje de partida para las demás barras típicamente $V = 1 < 0°$. Para esto resuelve el siguiente sistema [18]:

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} J \end{bmatrix} \begin{bmatrix} \Delta \theta \\ \Delta V \end{bmatrix} \tag{3.1}$$

Donde J corresponde a la matriz Jacobiana del problema, la que se calcula en base a la matriz de admitancia de la red y al estado de ésta en la iteración anterior. Esta ecuación describe la relación entre los errores de voltaje y ángulo entre iteraciones consecutivas con el error de potencias. De esta forma para calcular los voltajes de la iteración k se utiliza la siguiente ecuación:

$$\begin{bmatrix} \Delta\theta^{k+1} \\ \Delta V^{k+1} \end{bmatrix} = \begin{bmatrix} \Delta\theta^{k} \\ \Delta V^{k} \end{bmatrix} + \begin{bmatrix} J^{k} \end{bmatrix}^{-1} \begin{bmatrix} \Delta P^{k} \\ \Delta Q^{k} \end{bmatrix} \tag{3.2}$$

El método termina o converge cuando se logra que el error entre dos iteraciones consecutivas es menor que un cierto umbral.

El método de Newton Raphson ha sido extendido a otra aplicaciones tales como flujo de potencia óptimo y estimación de estado, por lo que un método que no esté basado en Newton Raphson puede ser más difícil de extender a estas aplicaciones [20].

3.1.5. Desempeño ante redes acopladas

Las líneas de un sistema de transmisión, al tener un alto voltaje, suelen tener una resistencia mucho menor a la reactancia de las mismas, lo que permite despreciarlas, eliminando la relación entre el voltaje y la potencia activa y la del ángulo de éste con la potencia reactiva. Con esta simplificación en el modelo es posible simplificar el algoritmo y optimizarlo para este tipo de redes, de hecho existe una variante del método de Newton Raphson conocida como Newton Raphson Desacoplado [18]. Como se trata de una simplificación estos métodos pierden validez en redes que no presentan esta característica.

3.1.6. Tamaño del problema, en la utilización de Newton Rhpason

Cuando el algoritmo resuelve el problema utilizando el método de Newton Raphson, es posible medir el tamaño del problema en relación al tamaño de la matriz a invertir, es decir, del Jacobiano. Este tamaño se suele medir en proporción a la cantidad de barras N del sistema, por ejemplo, $6N \times 6N$.

3.2. Métodos basados en estructura radial de redes de distribución

3.2.1. Método radial Back/Forward con inyecciones de corriente

En [19] se presenta un método basado en la estructura primordialmente radial de las redes de distribución, pero que sin embargo es capaz de incorporar los efectos de algunos loops vía inyecciones de corriente.

Para esto primero se rompen los loops dejando una estructura netamente radial, donde se clasifican los nodos en capas o layers alejándose de la barra slack, correspondiente a la subestación.

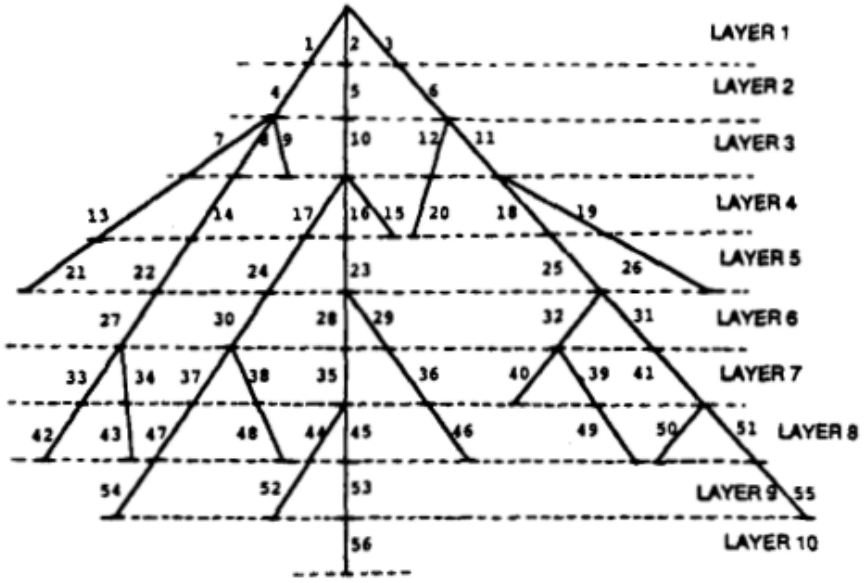

Figura 3.1: Clasificación de nodos, estructura radial. Fuente: [19].

Los loops rotos son reemplazados por inyecciones de corriente, de igual magnitud y distinto signo, en ambos lados de los puntos de ruptura o breakpoints, tal como se observa a continuación:

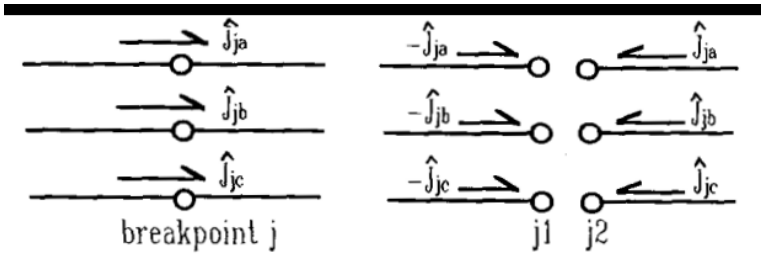

Figura 3.2: Uso de breakpoints, método radial Back/Forward con inyecciones de corriente. Fuente: [19].

Además se calcula una matriz de sensibilidad para las barras PV, con el fin de corregir el voltaje y mantenerlo en el valor determinado para la barra.

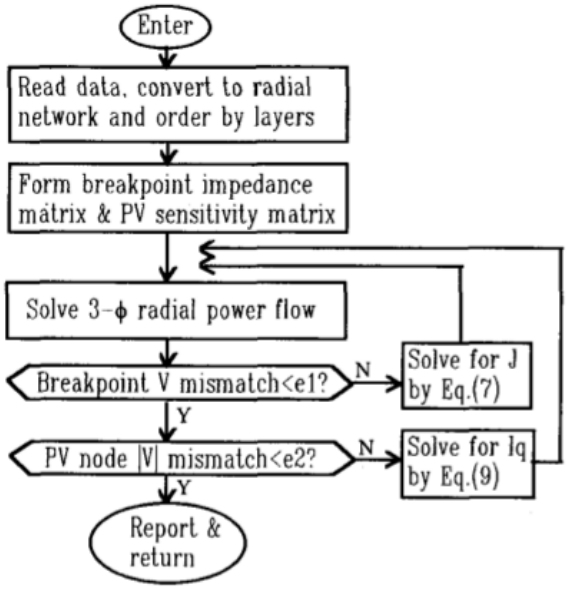

Figura 3.3: Algoritmo método radial Back/Forward con inyecciones de corriente. Fuente: [19].

Como se puede observar en la figura 3.3, el proceso se resume en:

1. En caso que la red no sea netamente radial, se rompen los loops en uno de sus extremos considerando un breakpoint con inyecciones de corrientes de fase determinadas, quedando una estructura de árbol. De esta forma se clasifican las ramas en capas alejándose desde la barra slack.

2. Luego se calculan las matrices de impedancia de ramas y de breakpoints, como se trata de redes desbalanceadas se consideran las 3 fases teniéndose para cada sección de línea o transformador una matriz de 3×3. Se calcula la matriz de sensibilidad para las barras PV y se inicializan los voltajes.

3. Se procede con el flujo de potencia trifásico:

   a) Se calculan las corrientes nodales trifásicas en base a las inyecciones de potencia constante y elementos shunt (Y_i) en cada barra i mediante la ecuación:

$$\begin{bmatrix} I_{ia} \\ I_{ib} \\ I_{ic} \end{bmatrix}^k = \begin{bmatrix} (S_{ia}/V_{ia}^{k-1})^* \\ (S_{ib}/V_{ib}^{k-1})^* \\ (S_{ic}/V_{ic}^{k-1})^* \end{bmatrix} - \begin{bmatrix} Y_{ia}^* & & \\ & Y_{ib}^* & \\ & & Y_{ic}^* \end{bmatrix} \begin{bmatrix} V_{ia} \\ V_{ib} \\ V_{ic} \end{bmatrix}^{k-1} \tag{3.3}$$

   b) Ciclo completo reverso para la suma de corrientes J_l en la línea l, partiendo desde la última capa hacia la primera, donde M es el conjunto de secciones de línea conectadas a la barra j.

$$\begin{bmatrix} J_{la} \\ J_{lb} \\ J_{lc} \end{bmatrix}^k = - \begin{bmatrix} I_{ja} \\ I_{jb} \\ I_{jc} \end{bmatrix}^k + \sum_{m \in M} \begin{bmatrix} J_{ma} \\ J_{mb} \\ J_{mc} \end{bmatrix}^k \tag{3.4}$$

   c) Ciclo completo hacia adelante para actualizar los voltajes nodales, partiendo desde la barra slack hacia la última capa:

$$\begin{bmatrix} V_{ja} \\ V_{jb} \\ V_{jc} \end{bmatrix}^k = \begin{bmatrix} V_{ia} \\ V_{ib} \\ V_{ic} \end{bmatrix}^k - \begin{bmatrix} z_{aa,l} & z_{ab,l} & z_{ac,l} \\ z_{ab,l} & z_{bb,l} & z_{bc,l} \\ z_{ac,l} & z_{bc,l} & z_{cc,l} \end{bmatrix} \begin{bmatrix} J_{la} \\ J_{lb} \\ J_{lc} \end{bmatrix}^k \tag{3.5}$$

4. Se verifica la convergencia de los voltajes en los breakpoints. Si para todos los breakpoints hay convergencia se pasa al paso siguiente, de lo contrario se vuelve al paso 3 previo cálculo de las inyecciones de corriente en éstos mediante:

$$\left[Z_B\right] \left[\hat{J}_B\right]^{(\mu)} = \left[\hat{V}_B\right]^{(\mu)} \tag{3.6}$$

5. Se comprueba la convergencia de voltaje en las barras PV, si hay convergencia el método ha terminado, de lo contrario:

   a) Se calculan las diferencias de voltaje en las barras PV

   b) En base a éstas, similarmente al caso anterior, se calculan las corrientes reactivas requeridas para eliminar estas diferencias de voltaje mediante la aproximación lineal:

$$\left[Z_V\right] \left[I_q\right]^{(\gamma)} = \left[\Delta V\right]^{(\gamma)} \tag{3.7}$$

c) En base a estas corrientes y los voltajes actualizados de barra se calculan las inyecciones de potencia reactiva requeridas.

d) Se verifica que la potencia reactiva requerida no exceda los límites de la barra, si excede uno se fija a ese valor y la barra se convierte en PQ.

Adicionalmente en el documento se describe la inclusión de reguladores de tensión, cargas alimentadas entre fases, carga distribuida, entre otras cosas que no están dentro del alcance de esta comparación.

3.2.2. Método Newton Raphson modificado para redes de distribución

Por otro lado en [20] se presenta un método basado en Newton Raphson que utiliza la estructura radial de los sistemas de distribución para simplificar los cálculos, principalmente en la factorización UDU^T de la matriz Jacobiana. Para esto hace dos suposiciones importantes:

- La diferencia de voltaje entre dos nodos consecutivos es pequeña
- No hay ramas shunt

La primera es medianamente razonable en una red de distribución pero la segunda deja de serlo si se incorporan reactores o condensadores shunt, cargas de impedancia constante y modelos Π completos de líneas de transmisión y/o transformadores. Esto sin embargo puede ser corregido modificando las inyecciones de potencia en los nodos correspondientes con los voltajes iniciales y sus actualizaciones en cada iteración.

El método utiliza además el supuesto de red radial para la factorización UDU^T del Jacobiano, lo que permite reescribir la ecuación 3.1 de la siguiente manera:

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix} = \begin{bmatrix}A_{n-1} & \\ & A_{n-1}\end{bmatrix}\begin{bmatrix}D_B & -D_G\\ D_G & D_B\end{bmatrix}\begin{bmatrix}A_{n-1}^T & \\ & A_{n-1}^T\end{bmatrix}\begin{bmatrix}\Delta\theta\\ \Delta V/V\end{bmatrix} \tag{3.8}$$

Donde D_B y D_G son matrices diagonales y A_{n-1} es una matriz triangular superior que caracteriza la red de distribución, ordenada de manera similar a como se hace en [19], como se puede ver en la figura 3.4

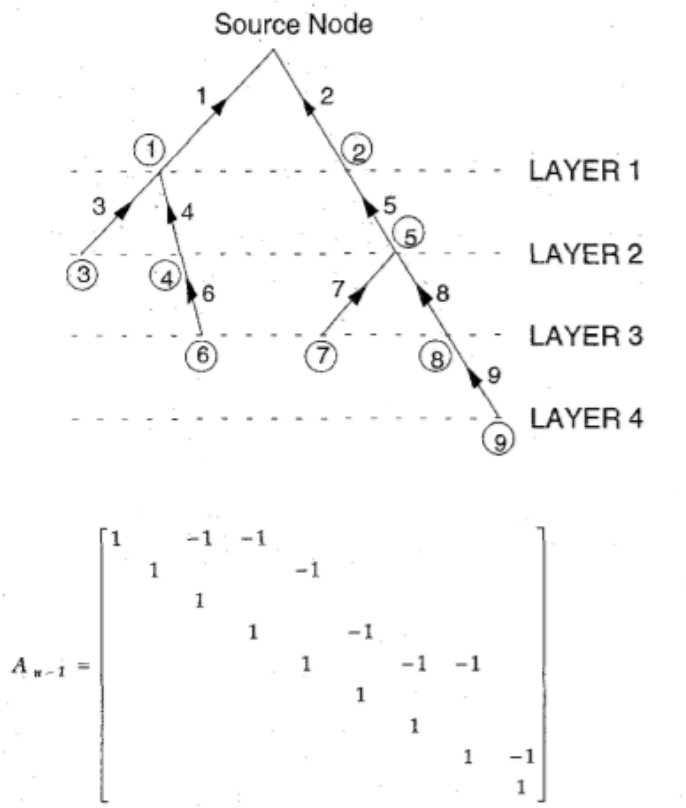

Figura 3.4: Relación entre matriz jacobiana y red de distribución, método Newton Raphson Modificado para redes de distribución. Fuente: [20].

Entonces es posible definir:

$$E = \Delta\theta + j\Delta V/V \tag{3.9}$$

$$S = \Delta P + j\Delta Q \tag{3.10}$$

$$W = D_B + jD_G \tag{3.11}$$

De esta forma, se transforma la ecuación 3.1 en:

$$A_{n-1}S_L = S \tag{3.12}$$

$$WA_{n-1}^T E = SL \tag{3.13}$$

Donde 3.12 es el barrido hacia atrás (backward) y 3.13 es el barrido hacia adelante (forward). De esta forma el algoritmo de este método es el siguiente:

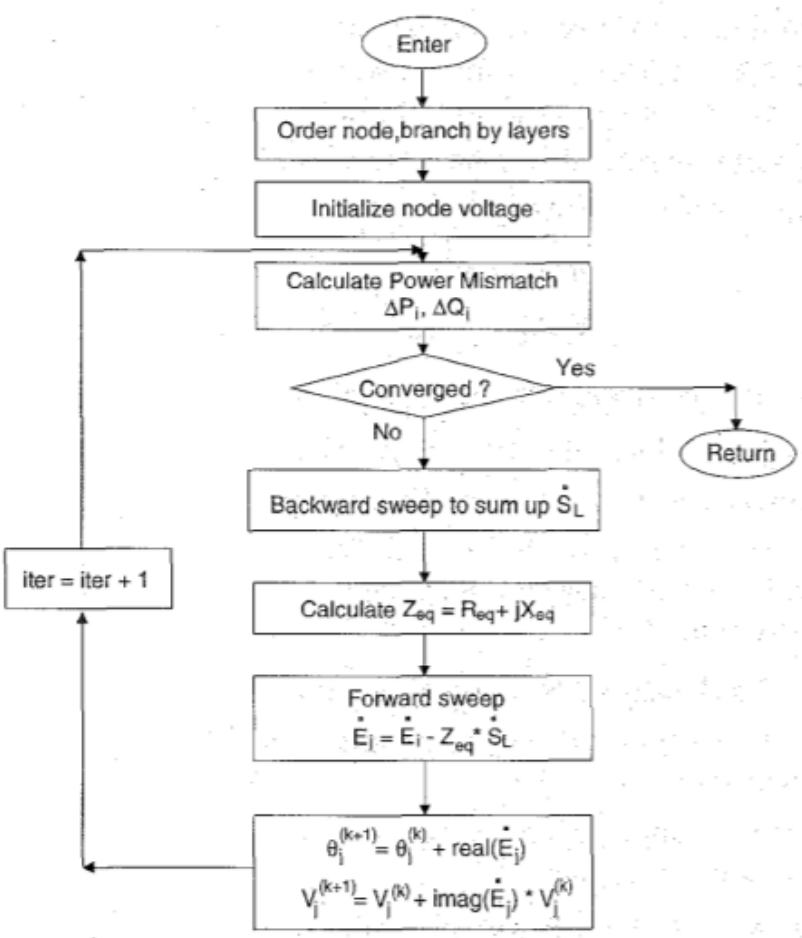

Figura 3.5: Algoritmo método Newton Raphson Modificado para redes de distribución. Fuente: [20].

La principal desventaja de estos métodos es que requieren que la red sea mayoritariamente radial, lo que es bastante razonable para un red de distribución pero significa una pérdida de generalidad al introducir el concepto de microrredes.

3.3. Métodos basados en el algoritmo de Newton Raphson

3.3.1. Método de inyecciones de corriente

Este método es bastante similar al de Newton Raphson, con la diferencia que en vez de utilizar potencias activas y reactivas para calcular los módulos y ángulos de los voltajes, utiliza las partes reales e imaginarias de las inyecciones de corriente para calcular las partes reales e imaginarias de los voltajes. Para esto en [21] se propone el siguiente algoritmo:

1. Inicializar los voltajes
2. Calcular las inyecciones de potencias activas y reactivas en cada barra k, para cada fase s:

$$(P_k^{calc})^s = V_{rk}^s (I_{rk}^{calc})^s + V_{mk}^s (I_{mk}^{calc})^s \tag{3.14}$$
$$(Q_k^{calc})^s = V_{rk}^s (I_{rk}^{calc})^s - V_{mk}^s (I_{mk}^{calc})^s \tag{3.15}$$

   Donde r es la parte real y m la imaginaria y además:

$$(I_{rk}^{calc})^s = \sum_{i=1}^{n} \sum_{t\in\{a,b,c\}} (G_{ki}^{st} V_{ri}^t - B_{ki}^{st} V_{mi}^t) \tag{3.16}$$
$$(I_{mk}^{calc})^s = \sum_{i=1}^{n} \sum_{t\in\{a,b,c\}} (G_{ki}^{st} V_{mi}^t - B_{ki}^{st} V_{ri}^t) \tag{3.17}$$

3. Calcular los incrementos de potencia, donde sp representa las potencias especificadas por los datos del sistema, y $calc$ las calculadas:

$$\Delta P^s = (P_k^{sp})^s - (P_k^{calc})^s \tag{3.18}$$
$$\Delta Q^s = (Q_k^{sp})^s - (Q_k^{calc})^s \tag{3.19}$$

4. Verificar la convergencia, es decir si los incrementos de potencia son todos menores que un cierto umbral la convergencia es alcanzada y el algoritmo se termina, de lo contrario se continua.

5. Calcular los incrementos de corriente:

$$\Delta I^s_{rk} = \frac{V^s_{rk}\Delta P^s_k + V^s_{mk}\Delta Q^s_k}{(V^s_{rk})^2 + (V^s_{mk})^2} \tag{3.20}$$

$$\Delta I^s_{mk} = \frac{V^s_{mk}\Delta P^s_k + V^s_{rk}\Delta Q^s_k}{(V^s_{rk})^2 + (V^s_{mk})^2} \tag{3.21}$$

6. Calcular la matriz Jacobiana, este cálculo es más engorroso por lo que no se expondrá en este trabajo ya que escapa a los objetivos del mismo.

7. Calcular los incrementos de voltaje resolviendo el siguiente sistema:

$$\begin{bmatrix} \Delta I^{abc}_{m_1} \\ \Delta I^{abc}_{r_1} \\ \hline \vdots \\ \hline \Delta I^{abc}_{m_i} \\ \Delta I^{abc}_{r_i} \\ \hline \vdots \\ \hline (\mathbf{\Delta I^{\bullet}_{m_k}})^{\mathbf{abc}} \\ (\mathbf{\Delta I^{\bullet}_{r_k}})^{\mathbf{abc}} \\ \hline \vdots \\ \hline \Delta I^{abc}_{m_l} \\ \Delta I^{abc}_{r_l} \\ \hline \vdots \end{bmatrix} = \left[\begin{array}{c|c|c|c|c|c|c|c} (Y^{\bullet}_{11})^{abc} & \cdots & Y^{abc}_{1i} & \cdots & (\mathbf{Y^{\bullet\bullet}_{1k}})^{\mathbf{abc}} & \cdots & Y^{abc}_{1l} & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline Y^{abc}_{i1} & \cdots & (Y^{\bullet}_{ii})^{abc} & \cdots & (\mathbf{Y^{\bullet\bullet}_{ik}})^{\mathbf{abc}} & \cdots & Y^{abc}_{il} & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline Y^{abc}_{k1} & \cdots & Y^{abc}_{ki} & \cdots & (\mathbf{Y^{\bullet\bullet}_{kk}})^{\mathbf{abc}} & \cdots & Y^{abc}_{kl} & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline Y^{abc}_{l1} & \cdots & Y^{abc}_{li} & \cdots & (\mathbf{Y^{\bullet\bullet}_{lk}})^{\mathbf{abc}} & \cdots & (Y^{\bullet}_{ll})^{abc} & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}\right] \begin{bmatrix} \Delta V^{abc}_{r_1} \\ \Delta V^{abc}_{m_1} \\ \hline \vdots \\ \hline \Delta V^{abc}_{r_i} \\ \Delta V^{abc}_{m_i} \\ \hline \vdots \\ \hline \mathbf{\Delta V^{abc}_{m_k}} \\ \mathbf{\Delta Q^{abc}_{k}} \\ \hline \vdots \\ \hline \Delta V^{abc}_{r_l} \\ \Delta V^{abc}_{m_l} \\ \hline \vdots \end{bmatrix}$$

Figura 3.6: Sistema de ecuaciones, método de inyecciones de corriente. Fuente: [21].

8. Actualizar los voltajes y volver al paso 2:

$$(V^{abc}_{rmk})^{h+1} = (V^{abc}_{rmk})^h + (\Delta V^{abc}_{rmk})^h \tag{3.22}$$

La principal desventaja de éste método es la dimensión del problema $6n \times 6n$, además la Matriz Jacobiana debe ser recalculada e invertida en cada iteración, sin embargo, en [21] se presentan resultados bastante satisfactorios tanto actualizando, como no actualizando el Jacobiano.

Una ventaja de éste método es que, pese a lo anterior, el Jacobiano tiene varios términos idénticos a los de la matriz de admitancia, por lo que no deben ser recalculados. En efecto a continuación se muestra un ejemplo de un sistema pequeño y los términos de su matriz Jacobiana:

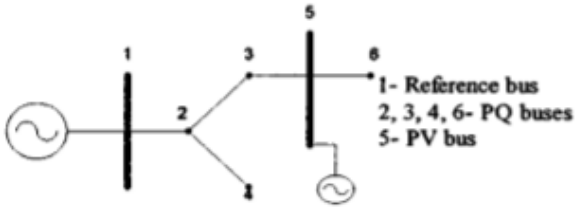

Figura 3.7: Sistema de ejemplo, método de inyecciones de corriente. Fuente: [21].

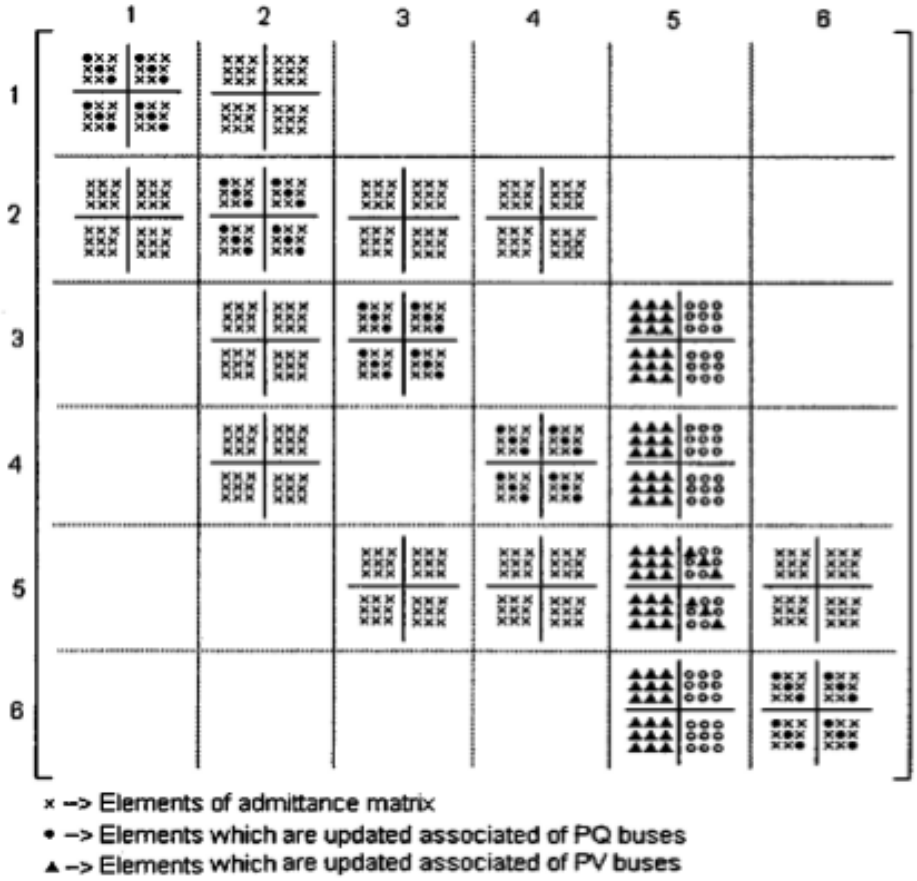

Figura 3.8: Términos de la matriz jacobiana, método de inyecciones de corriente. Fuente: [21].

3.3.2. Método de Newton Raphson trifásico

En [22] se presenta un modelo y algoritmo completos para el cálculo de flujos de potencia desbalanceados, basados en Newton Raphson pero aplicado a las 3 fases simultáneamente. La modelación de los componentes es la siguiente:

3.3.2.1. Modelo de componentes

A. Cargas El algoritmo puede modelar:

- **Cargas de potencia constante**
- **Corriente constante:** transformando estos valores en potencias equivalentes usando los correspondientes voltajes actualizados.
- **Impedancia constante:** Pueden ser modeladas conectadas en delta, estrella con neutro flotante o estrella con neutro aterrizado, ya sea directamente o a través de una impedancia. Se modelan como una matriz de impedancia de (3×3) que es agregada a la matriz de admitancia global del sistema, en el bloque diagonal correspondiente a la barra de la carga.

B. Generadores En estado estacionario estos son modelados como generadores síncronos con excitación balanceada:

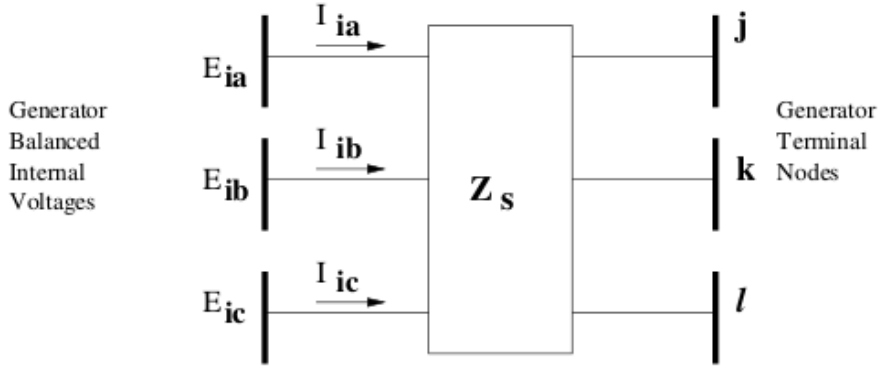

Figura 3.9: Modelo de generador distribuido, método de Newton Raphson trifásico. Fuente: [22].

Donde, a diferencia de los voltajes en bornes (i, j, k) las f.e.m. internas del generador (E_{ia}, E_{ib}, E_{ic}) al estar balanceadas tienen la misma magnitud y están desfasadas entre sí en 120°, por lo tanto solo E_{ia}, V_i, V_j y V_k son incógnitas.

C. Transformadores Son modelados, despreciando corrientes de excitación y considerando posibles derivaciones, como una matriz de admitancia de (3×3).

D. Condensadores y reactores shunt Son modelados de la misma forma que las cargas de impedancia constante mencionadas anteriormente.

E. Implementación de restricciones

F. Taps y límites de voltaje Los reguladores de voltaje son modelados como transformadores con derivaciones, las que son ajustadas en cada iteración en la resolución del sistema de ecuaciones.

G. Límites de reactivos Dado que en la experiencia para flujos de potencia monofásicos, la verificación de límites de reactivos debe ser realizada una vez que la solución haya alcanzado cierto grado de convergencia, se define un cierto umbral de convergencia para los incrementos de potencia. Es decir, se revisan los límites en cada iteración solo si el algoritmo ha alcanzado este umbral de convergencia, en caso que una barra PV exceda algún límite, la barra es transformada a una barra PQ con una potencia reactiva especificada igual al límite excedido.

H. Solución del problema Básicamente se resuelve el problema utilizando el método de Newton Raphson, si se tienen $N+1$ barras, de las cuales N_g son generadores y una es la barra slack, el sistema a resolver es de orden $3N + N_g$, dado el modelo de los generadores.

De esta forma el Jacobiano tiene la siguiente estructura:

$$J = \begin{array}{c} \\ \Delta P_L \\ \Delta P_G \\ \Delta Q_L \\ \Delta V_G \end{array} \begin{array}{c} \begin{array}{cccc} \Delta\theta_L & \Delta\theta_G & \Delta V_L & \Delta V_G \end{array} \\ \begin{bmatrix} J_{P_L\theta_L} & J_{P_L\theta_G} & J_{P_LV_L} & J_{P_LV_G} \\ J_{P_G\theta_L} & J_{P_G\theta_G} & J_{P_GV_L} & J_{P_GV_G} \\ J_{Q_L\theta_L} & J_{Q_L\theta_G} & J_{Q_LV_L} & J_{Q_LV_G} \\ J_{V_G\theta_L} & 0 & J_{V_GV_L} & 0 \end{bmatrix} \end{array}$$

Figura 3.10: Estructura matriz jacobiana, método de Newton Raphson trifásico. Fuente: [22].

Donde el subíndice G representa a las barras internas de los generadores (PV) y L a las barras del sistema (PQ), incluyendo a los bornes de los generadores (PQ con inyecciones de potencia nulas).

Como se puede observar en el Jacobiano, el sistema a resolver corresponde a:

$$\begin{bmatrix} \Delta P_L \\ \Delta P_G \\ \Delta Q_L \\ \Delta V_G \end{bmatrix} = \Big[J\Big] \begin{bmatrix} \Delta \theta_L \\ \Delta \theta_G \\ \Delta V_L \\ \Delta V_G \end{bmatrix} \tag{3.23}$$

Para esto se utiliza el método de Newton Raphson, los voltajes son inicializados en 1 pu con desfases de 0, -120, 120°en todas las barras, y los taps en su valor por defecto, usualmente igual a 1.

3.3.2.2. Algoritmo de solución

El algoritmo de solución se puede visualizar en el siguiente diagrama de flujo:

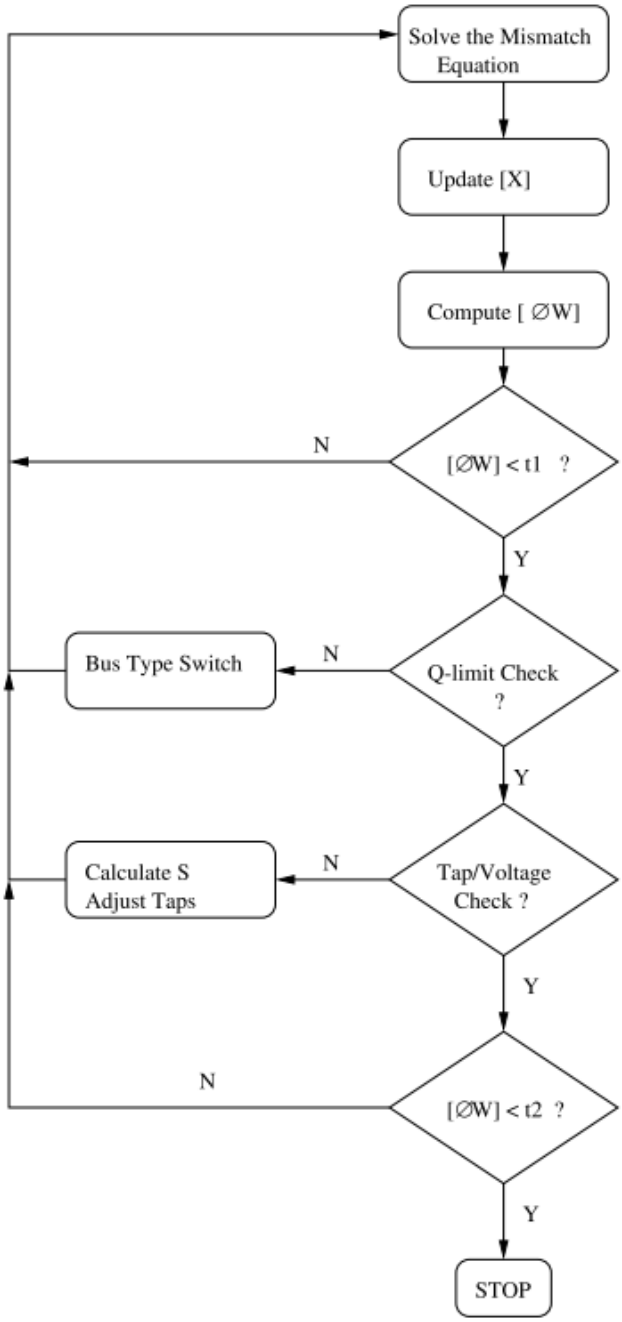

Figura 3.11: Algoritmo de solución, método de Newton Raphson trifásico. Fuente: [22].

3.4. Métodos basados en componentes de secuencia

3.4.1. Método basado en componentes de secuencia

En [17] se presenta una algoritmo basado en componentes de secuencia y los modelos utilizados, la ventaja de este método es que el problema se reduce de una matriz Jacobiana de $(6N \times 6N)$ a una de $(2N \times 2N)$ y dos de $(N \times N)$, lo que resulta en un 70 % menos de tiempo de factorización.

3.4.1.1. Modelos de componentes

A. Generador Se modelan de la siguiente manera:

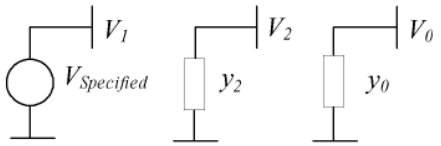

Figura 3.12: Modelo de generador en componentes de secuencia, método basado en componentes de secuencia. Fuente: [17].

B. Transformador Se modelan mediante una matriz de admitancia de (6×6):

$$Y_{012} = \begin{bmatrix} T & \\ & T \end{bmatrix}^{-1} Y_{abc} \begin{bmatrix} T & \\ & T \end{bmatrix} \tag{3.24}$$

Donde, T corresponde a la matriz de Fortescue.

C. Líneas Los modelos de las líneas balanceadas son desacoplados, al igual que en componentes de fase, por lo que su matriz de admitancia es simétrica. Lamentablemente cuando aparecen desbalances los modelos se acoplan, lo que no deja de ocurrir en componentes de secuencia, y por lo tanto se pierde esta simetría. Sin embargo es posible transformar un modelo acoplado en 3 modelos desacoplados mediante inyecciones de corriente, tal como se muestra a continuación:

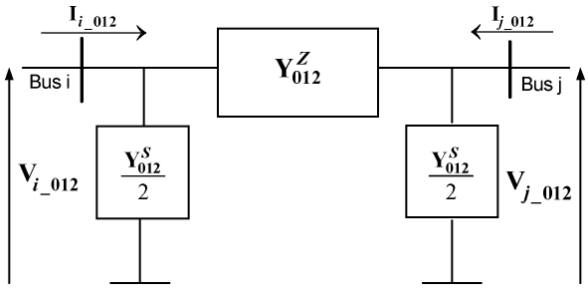

Figura 3.13: Modelo de línea balanceada en componentes de secuencia, método basado en componentes de secuencia. Fuente: [17].

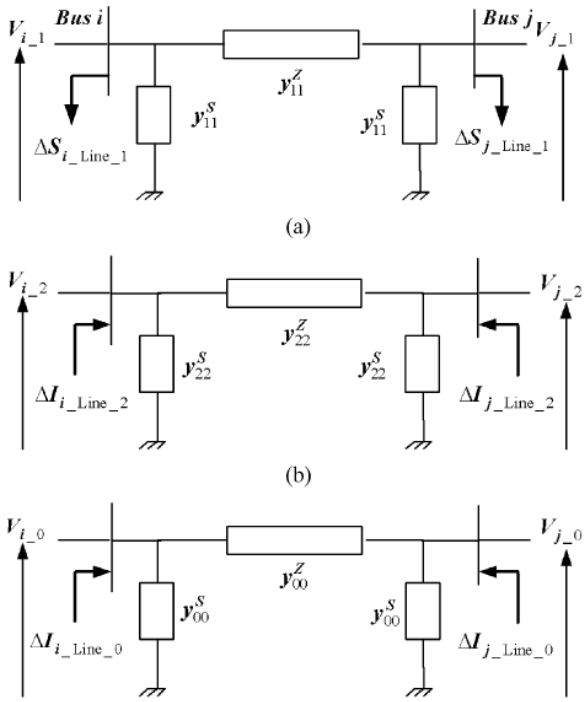

Figura 3.14: Modelo de línea desbalanceada en componentes de secuencia, método basado en componentes de secuencia. Fuente: [17].

De esta forma es posible modelar la línea mediante una matriz de admitancia simétrica que incorpore el acoplamiento mediante inyecciones de corriente $\Delta I_{i\_Line\_012}$ y potencia $\Delta S_{i\_Line\_012}$

3.4.1.2. Solución

A. Especificación de barras Para resolver el algoritmo se deben especificar las barras slack, PV y PQ.

- **Barra Slack:** se conoce su voltaje de secuencia positiva, en ángulo y magnitud.

- **Barras PV:** corresponden a las barras de los generadores, se conocen las magnitudes de los voltajes de secuencia positiva y la potencia activa total generada.
- **Barras PQ:** corresponden a las barras de carga, se conocen las potencias en cada fase. Con estas potencias se calculan las corrientes de fase y se inicia el algoritmo.

B. Algoritmo de solución Una vez calculadas las corrientes de fase iniciales, se transforman en corrientes de secuencia, las que, son sumadas a las inyecciones de corriente que representan el acoplamiento de las lineas desbalanceadas.

Luego para la secuencia positiva las corrientes son transformadas en potencias, con lo que se corre una iteración del flujo de potencia mediante Newton Raphson, obteniéndose los voltajes de secuencia positiva. Paralelamente, se calculan los voltajes de secuencias negativa y cero.

Finalmente, con estos voltajes, se actualizan los voltajes de fase y se repite el proceso. El diagrama de flujo de este procedimiento se puede observar en la figura 3.15

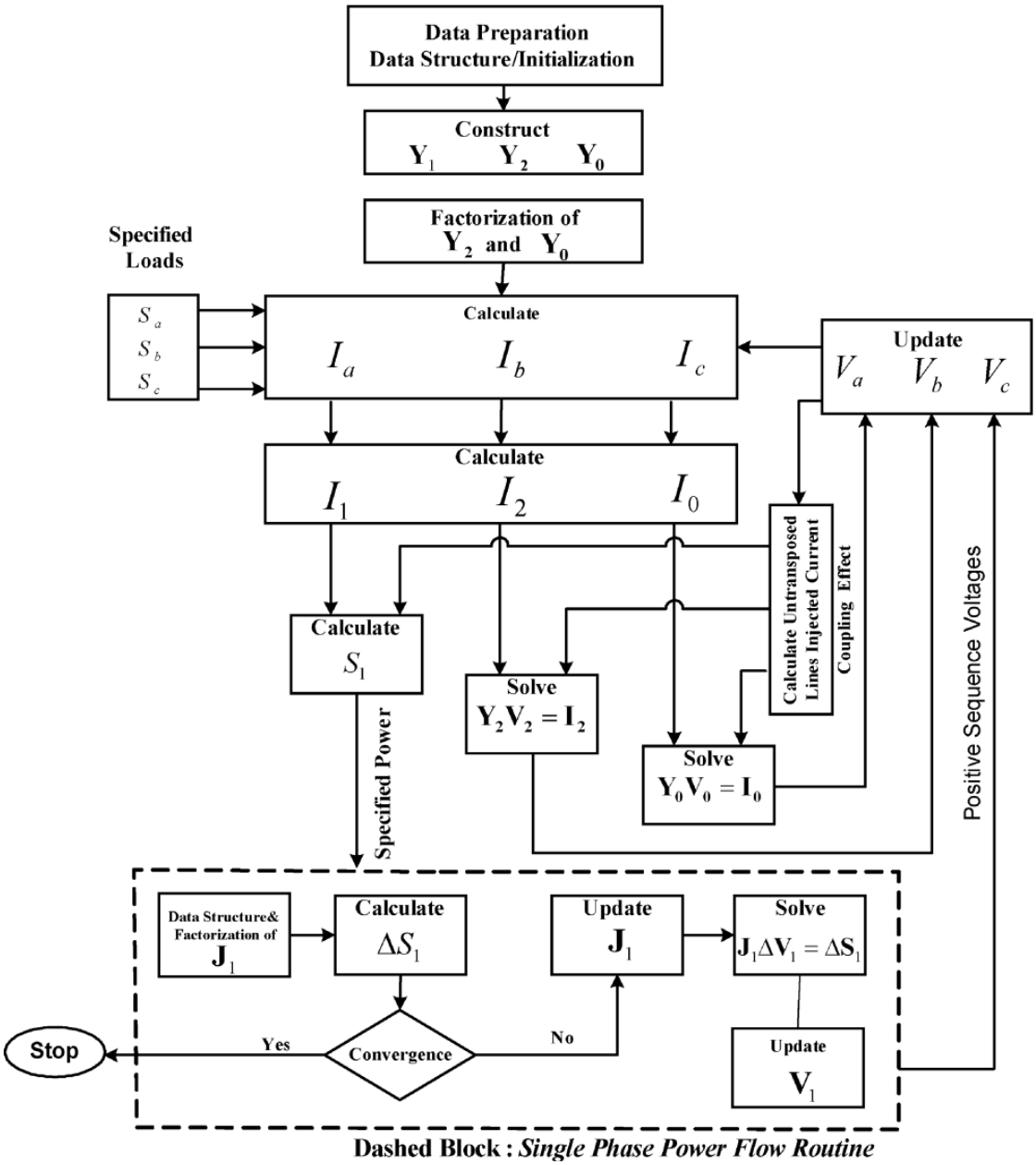

Figura 3.15: Algoritmo solución, método basado en componentes de secuencia. Fuente: [17].

3.4.2. Método basado en componentes de secuencia con unidades DER acopladas a VSC

Finalmente en [23] se presenta un método que incorpora la modelación de unidades DER conectadas electrónicamente a la microrred mediante VSC (*Voltage Source Converter*)

3.4.2.1. Modelo de unidades DER acopladas a VSC

En la figura 3.16 se puede observar un diagrama esquemático de un DER acoplado a su correspondiente VSC, de 3 o 4 conductores.

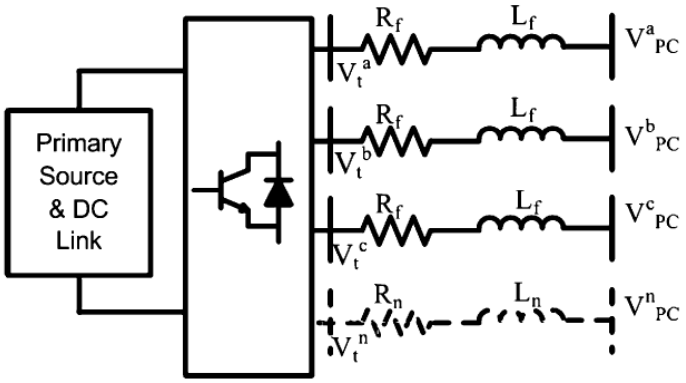

Figura 3.16: Diagrama esquemático unidad DER acoplada a un VSC, método basado en componentes de secuencia con unidades DER acopladas a VSC. Fuente: [23].

El modelo no incorpora la unidad de DER ya que asume que los controladores de los VSC controlan completamente el voltaje en el bus DC en condiciones estacionarias. Además solo considera la frecuencia fundamental, descartando armónicos producidos por posibles desbalances.

El objetivo principal de los controladores de los VSC es controlar voltaje y frecuencia (barra PV) o entregar consignas fijas de potencia (barra PQ), pero adicionalmente es posible ordenarles entregar corrientes de secuencia negativa y cero para corregir desbalances o detectar operación en isla.

A. Modelo de secuencia positiva Cuando la unidad está operando en modo PV (Figura 3.17(b)) su secuencia positiva se representa como una fuente de voltaje ideal con un voltaje igual al especificado y una potencia activa igual a un tercio de la especificada.

$$|V_{PC}^{1}| = V_{sp} \tag{3.25}$$

$$P^{1} = \frac{P_{sp}}{3} \tag{3.26}$$

Mientras que en modo PQ (Figura 3.17(b)) se modela como una fuente de potencia constante cuyas potencias especificadas, activa y reactiva, son:

$$P^{1} = \frac{P_{sp}}{3} \tag{3.27}$$

$$Q^{1} = \frac{Q_{sp}}{3} \tag{3.28}$$

B. Modelo de secuencias negativa y cero La unidad es modelada como una fuente de corriente en paralelo a admitancias ficticias. Cabe mencionar que cuando se trata de un VSC de 3 conductores, al no haber conexión de neutro, no pueden haber corrientes de secuencia cero, por lo que el modelo de la figura 3.17(d) no existe.

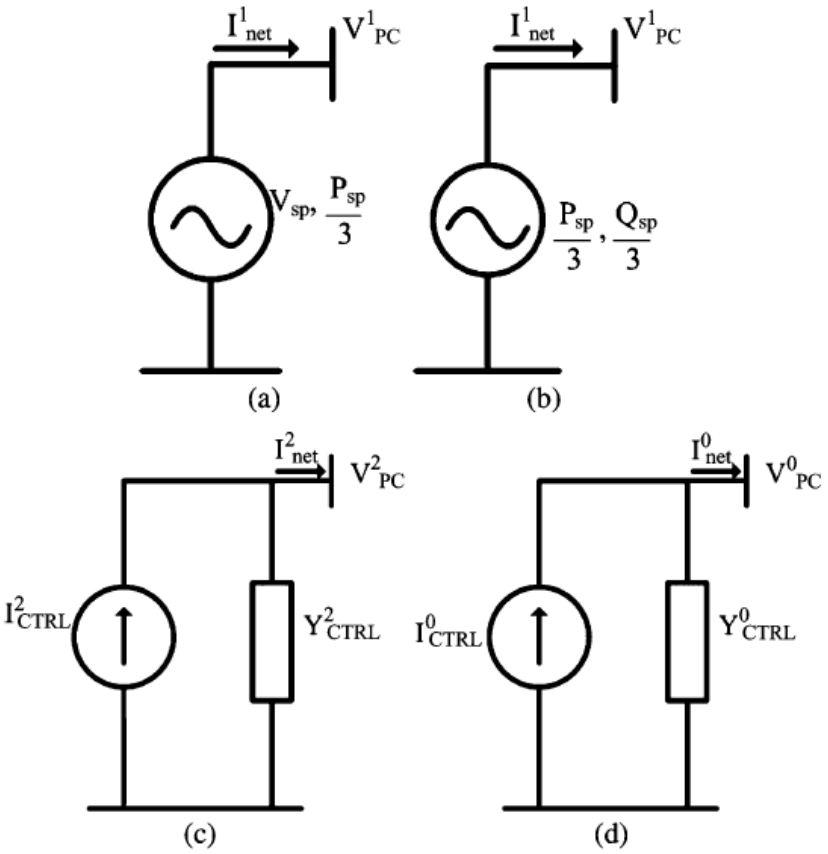

Figura 3.17: Modelo unidades DER acopladas a VSC. (a) Modelo de Secuencia Positiva para Modo PV, (b) Modelo de secuencia positiva para modo PQ, (c) Modelo de secuencia negativa, (d) Modelo de secuencia cero. Método basado en componentes de secuencia con unidades DER acopladas a VSC. Fuente: [23].

3.4.2.2. Algoritmo de solución

El algoritmo para la resolución del problema se divide en 2 etapas, primero un SFPS (*Three-phase sequence-component frame power-flow solver*) que resuelve el flujo de potencia. Al finalizar cada iteración del SFPS se ejecuta secuencialmente una evaluación de los límites de operación de los DER. De esta forma el algoritmo corresponde a:

1. Construir las matrices de admitancia en componentes de secuencia, añadiendo los términos correspondientes a las admitancias de los modelos de secuencia negativa y cero de los VSC.
2. Inicializar los voltajes de secuencia positiva dependiendo del tipo de barra. Los de secuencia cero y negativa pueden ser inicializados como nulos.
3. Calcular las potencias y corrientes especificadas para cada barra.

4. Resolver una iteración de flujo de potencia monofásico, para los voltajes de secuencia positiva, mediante Newton Raphson.

5. Calcular los voltajes de secuencia negativa y cero resolviendo:

$$I_{sp}^{0,2} = Y_{BUS}^{0,2} V_{BUS}^{0,2} \tag{3.29}$$

6. Para cada barra calcular los voltajes de fase utilizando la matriz de Fortescue.

7. Evaluar los límites de los VSC:

   a) Calcular los parámetros de los VSC

   b) Verificar límites de corrientes de fase, si alguna lo excede, se recalculan las potencias activas (y reactiva para barras PQ) especificadas

   c) Verificar límites de reactivos para barras PV, si se excede un límite se recalcula el voltaje especificado.

   d) Verificar límites de índices de modulación de fase, si se excede un límite se setea al máximo permitido, se realizan las correcciones necesarias y se recalcula la potencia activa y voltaje o reactiva según sea barra PV o PQ.

8. Evaluar el criterio de finalización, en este caso se utiliza convergencia de voltajes de fase. Si se cumple el criterio se finaliza, de lo contrario se vuelve al paso 3.

Los detalles de los cálculos mencionados anteriormente no se mencionarán en esta revisión, sin embargo cabe destacar que, a diferencia del algoritmo de Newton Raphson clásico, al superarse un límite de reactivos en una barra PV, ésta no se transforma en barra PQ, sino que se modifica su voltaje de forma que sus reactivos estén dentro de los límites. Esto simplifica el algoritmo ya que se evita reestructurar el problema al modificar una barra, y además representa mejor el funcionamiento real del controlador de un VSC.

Finalmente el algoritmo se resume en la Figura 3.18

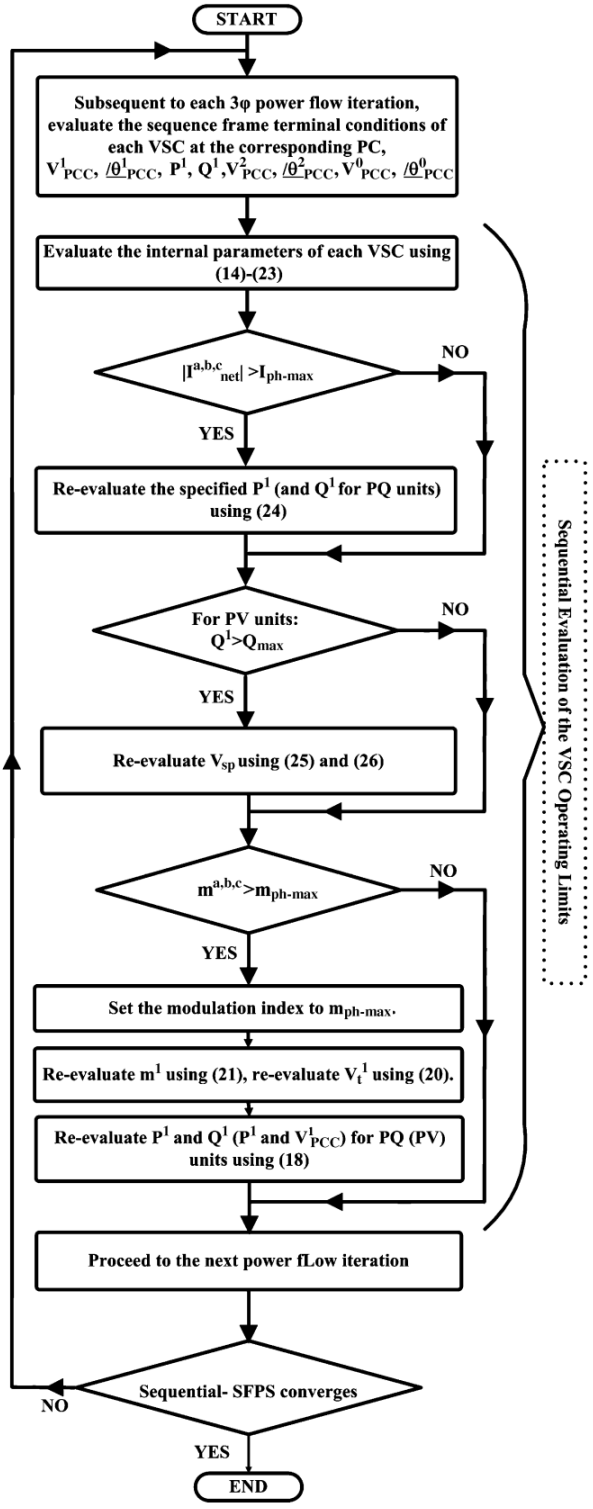

Figura 3.18: Algoritmo de solución, método basado en componentes de secuencia con unidades DER acopladas a VSC. Fuente: [23].

3.5. Resumen comparativo

Los métodos presentados anteriormente son sólo algunos ejemplos de muchos algoritmos más con similitudes entre sí. Sin embargo hay características distintivas que permitieron clasificarlos en los tres grupos presentados anteriormente. [1]

Tabla 3.1: Resumen comparativo, algoritmos de flujo de potencia desbalanceado. Elaboración propia.

Método	**Topología**	**Desbalance**	**Ramas Shunt**	**Newton Rhpason**	**Redes Acopladas**	**Tamaño del Problema**
Back/Forward	Radial o Anillo	Completo	Sí	No	Sí	-
NR Distribución	Radial o Anillo	Ninguno*	No*	Sí	Sí	$(N \times N)$*
Inyecciones Corriente	Cualquiera	Completo	Sí	Sí	Sí	$(6N \times 6N)$
NR Trifásico	Cualquiera	Completo	Sí	Sí	Sí	$(6N \times 6N)$
Comp. Secuencia	Cualquiera	Completo	Sí	Sí	Sí	1 de $(2N \times 2N)$ y 2 de $(N \times N)$
Comp. Sec con VCS	Cualquiera	Completo	Sí	Sí	Sí	1 de $(2N \times 2N)$ y 2 de $(N \times N)$

Dado que el simulador debe ser lo más general posible, es deseable que no tenga restricciones frente a la topología, por lo que los dos primeros métodos no son los más adecuados. Además el segundo método no fue implementado ni probado con desbalances ni ramas shunt, por lo que, pese a que en el documento correspondiente se describe cómo implementar estas características, no es recomendable su utilización para este contexto. Por otra parte el primer método no se trata de una implementación del método de Newton Raphson, lo que dificultaría su extensión a estimación de estados y flujo de potencia óptimo, entre otras aplicaciones [20].

Se observa en la tabla 3.1 que los cuatro últimos métodos son bastante similares, teniendo como principal diferencia que los métodos en componentes de secuencia tienen la ventaja de corresponder a problemas de menor dimensión que los de componentes de fase [17], tal como se observa en la tabla. Por otro lado, bajo éste enfoque es posible aprovechar el uso de corrientes de secuencia negativa y cero para calcular y corregir desbalances del sistema [23].

[1] "Las características marcadas como (*) no están contempladas en la implementación original y las pruebas realizadas, sin embargo, el trabajo indica cómo podrían implementarse en el algoritmo"

Capítulo 4

Modelo computacional de la herramienta

Para presentar el diseño de clases de la herramienta, por motivos de visualización y conceptualización, se divide el modelo en tres subdiagramas, presentando el modelo de los componentes eléctricos de la red, el modelo de los datos e información y el modelo de la interfaz gráfica. Cabe mencionar que los modelos expuestos a continuación no incorporan todas las variables ni métodos de las clases, limitándose a mostrar lo justo y necesario para ejemplificar y explicar el modelo del sistema. Los detalles de implementación se especifican en los capítulos siguientes.

4.1. Simbología de diagramas de clase

Para explicar la simbología utilizada en estos diagramas de clase se utilizará el ejemplo de la Figura 4.1.

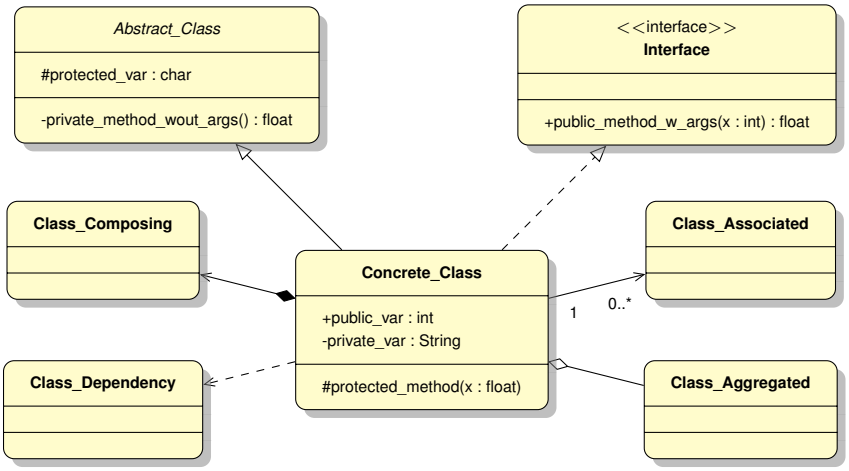

Figura 4.1: Simbología, diagrama de clases. Elaboración propia.

- **Clases:** `Concrete_Class` corresponde a una clase concreta, `Abstract_Class` a una clase abstracta que define una variable y un método abstractos. `Interface` es una interfaz que define un método abstracto. Las variables de instancia y métodos se clasifican en:
  - *Public*: accesibles por cualquier clase, se simboliza con +
  - *Private*: accesibles sólo por la misma clase, se simboliza con -
  - *Protected*: accesibles sólo por la misma clase y sus subclases, se simboliza con #
- **Extensión de Clases:** La flecha con punta sin relleno que une `Concrete_Class` con `Abstract_Class` indica que la primera extiende a la segunda, es decir, hereda sus variables y métodos. No todos los métodos de una clase abstracta serán abstractos, pero si lo son, la clase hijo debe implementarlos, ya que no poseen implementación en la clase padre.
- **Implementación de Interfaces:** La flecha punteada y con punta sin relleno indica que `Concrete_Class` implementa la interfaz `Interface`, por lo que debe implementar el método que ésta define.

- **Asociación de Clases:** la flecha con punta abierta indica que `Concrete_Class` está asociada a `Class_Associated`, es decir, están relacionadas y colaboran entre sí. Los números en ambos extremos indican la multiplicidad de esta relación. En el ejemplo un objeto `Concrete_Class` se relaciona con 0 o varios (0..*) objetos `Class_Associated`, mientras que un `Class_Associated` se relaciona con sólo un `Concrete_Class`. La dirección de la flecha indica la navegabilidad de la relación, en este caso es posible acceder desde `Concrete_Class` a su `Class_Associated`, pero no al revés. Si esta flecha no existe, es decir, es sólo una línea, la navegabilidad es en ambos sentidos.

- **Composición de Clases:** la flecha con un rombo con relleno en un extremo indica una relación de composición entre `Concrete_Class` y `Class_Composing`. Esta relación indica que la primera posee a la segunda dentro de sus variables de instancia, por lo que si se destruye un objeto `Concrete_Class`, se destruirá su `Class_Composing` asociado. Se dice que es una agregación por valor. Esta relación también puede tener multiplicidades.

- **Agregación de Clases:** la flecha con un rombo sin relleno en un extremo indica una relación de agregación entre `Concrete_Class` y `Class_Aggregated`. Esta relación es similar a la composición, pero con la salvedad de que los tiempos de vida de ambos objetos no están relacionados. Se dice que es una agregación por referencia. Esta relación también puede tener navegabilidad y multiplicidades.

- **Instanciación o Dependencia de Clases:** la flecha abierta punteada indica que `Concrete_Class` depende de `Class_Dependency`, ya que ésta implementa a `Concrete_Class`. Esta relación suele darse entre aplicación e interfaces gráficas. Esta relación también puede tener navegabilidad y multiplicidades.

.

4.2. Modelo eléctrico

En las figuras 4.2 y 4.3 se puede observar el modelo eléctrico de la herramienta. Éste contiene los componentes de la red eléctrica y la interacción entre ellos.

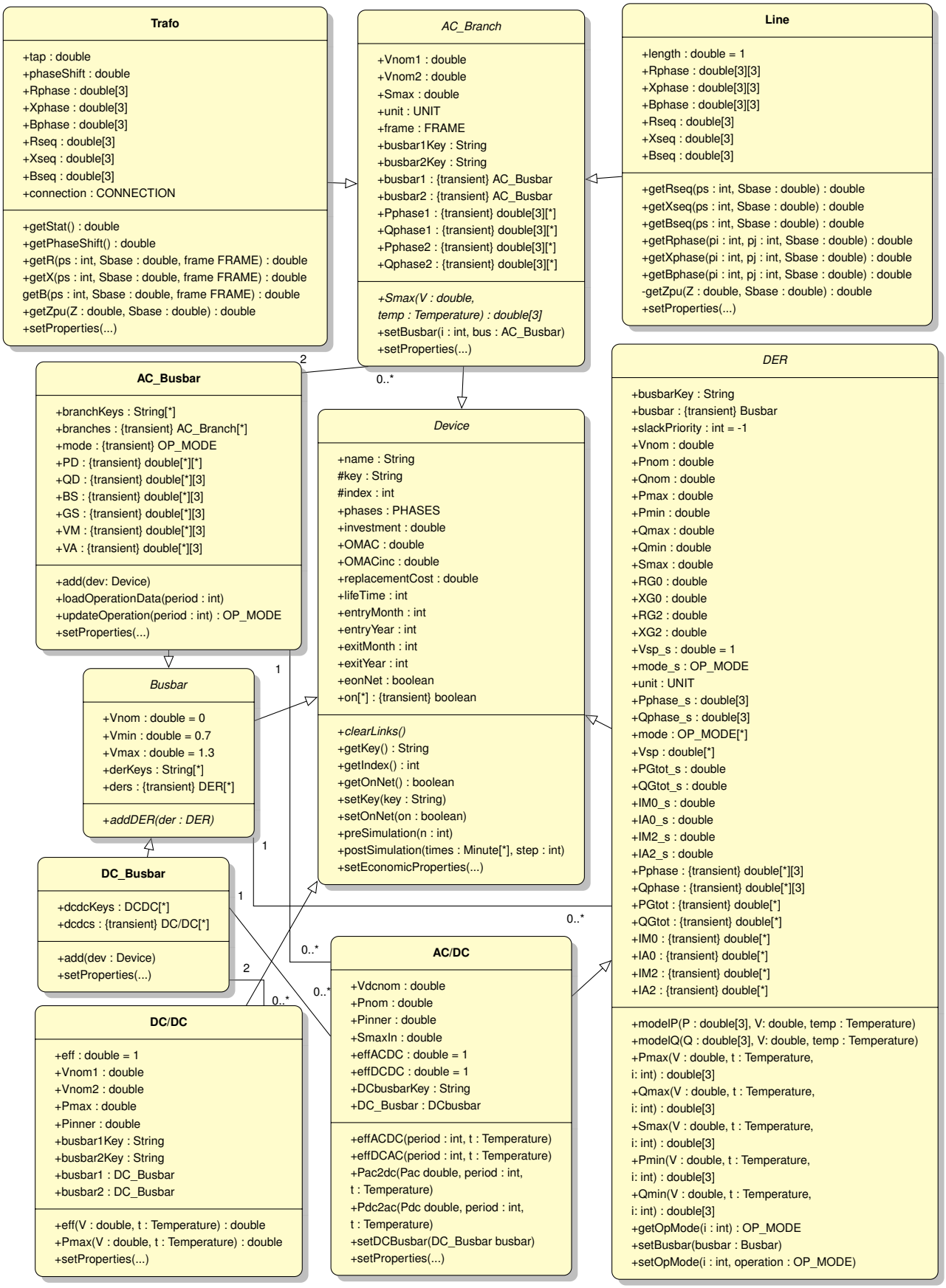

Figura 4.2: Diagrama de clases, componentes eléctricos. Elaboración propia.

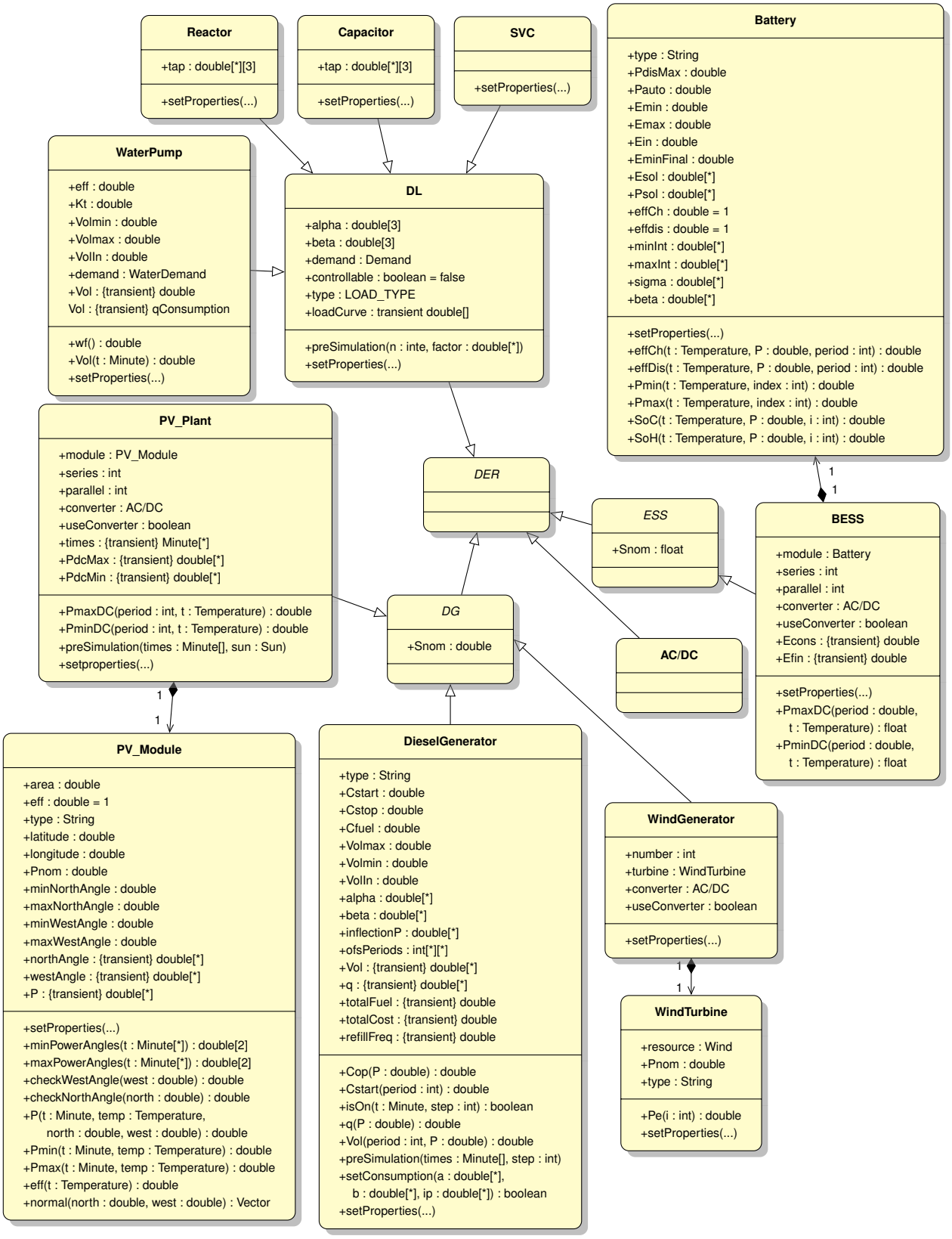

Figura 4.3: Diagrama de clases, dispositivos DER. Elaboración propia.

Para comprender este modelo hay que considerar dos puntos de vista distintos, el de despacho de unidades, y el de flujos de potencia. Asimismo, el modelo eléctrico se puede dividir en 3 tipos de elementos: las barras de conexión (`Busbar`), los elementos que se conectan a las barras (`DER`), y los elementos que conectan barras entre sí.

De la clase `Busbar` se extienden las clases `AC_Busbar` y `DC_Busbar` que permiten conectar componentes AC y DC respectivamente. De la misma manera, de la clase `AC_Branch` se extienden los elementos que representan las ramas de la red AC, tales como líneas y transformadores. Estos elementos sólo pueden conectarse entre dos barras, así como también es posible conectar las barras DC entre sí mediante conversores DC/DC.

Por otro lado de la clase `DER` se extienden las unidades de consumo (`DL`), generación (`DG`), almacenamiento (`ESS`) y sus respectivos derivados, los que pueden ser unidades AC o DC dependiendo de la tecnología. Las unidades DER de corriente continua se pueden conectar a la red de dos maneras: conectándolas directamente a una barra AC mediante un conversor AC/DC interno asociado mediante el parámetro `converter`; o conectándolas a una barra DC, la que a su vez puede tener más dispositivos conectados conformando una subred o grupo de unidades DC, el cual se conecta a la red AC mediante un conversor AC/DC.

Así, desde el punto de vista del despacho de unidades, cada unidad de almacenamiento o generación es considerada un `DER`, sin importar si se trata de una unidad AC o DC. Sin embargo, desde el punto de vista de flujos de potencia, todo grupo de unidades DC agrupadas en una barra es tratado como una sola unidad `DER`, por lo que la clase `AC/DC` también es una subclase de `DER`. De esta forma, todo grupo de elementos DC va a estar relacionado con una barra AC mediante su generalización a una clase `DER`, pero a su vez, desde el punto de vista del despacho uninodal de unidades, los costos y restricciones de cada unidad se consideran de forma independiente.

Adicionalmente, todas las clases provienen de la clase abstracta `Device` que define un identificador (`key`) y un nombre (`name`) de tipo `String`, y un identificador para la conexión de fases del equipo, cuyo tipo es explicado en la figura **??**. Además `Device` define todas las variables económicas, comunes para todos los componentes de la microrred.

Todos los dispositivos tienen además un método denominado `setProperties(..)` cuyos valores de entrada dependen del dispositivo, y se utilizan para modificar las propiedades de estos objetos mediante una sola instrucción. Las variables y aspectos principales del modelo, y las distintas clases que lo conforman, se explican a continuación:

4.2.1. Busbar

La clase abstracta `Busbar` representa la barras o nodos de conexión de los componentes. `Busbar` posee como variables de instancia su voltaje nominal, mínimo y máximo (`Vnom`, `Vmin` y `Vmax`), atemperatur (`temp`), y un arreglo `derKeys` que almacena los identificadores o `key` de los objetos `DER` conectados. Adicionalmente posee en arreglo `ders` que almacena las referencias directas a las unidades DER conectadas. Sin embargo, esta variable es temporal y sólo se utiliza para las simulaciones, es decir, no quedan guardadas en el objeto de forma permanente, lo que se simboliza con la característica `{transient}`.

`AC_Busbar`, que extiende a `Busbar`, posee como parámetros el tipo de barra PV, PQ o *slack* en la variable `mode`, el tipo de esta variable se puede observar en el diagrama de la figura 4.4. Además posee arreglos análogos a los de `Busbar`, pero que en vez de almacenar los identificadores y las referencias a las unidades DER, almacena los datos de las ramas conectadas. `AC_Busbar` almacena las potencias demandadas y los voltajes obtenidos de los procesos de simulación, en variables temporales (`PD`, `QD`, `GS`, `Bs`, `VM`, `VA`), consistentes en arreglos con los resultados para cada periodo de simulación.

`DC_Busbar` por su parte se utiliza para crear barras DC que permiten unir componentes DC y conectarlos mediante un mismo conversor `AC/DC` a la red alterna. Al igual que `AC_Busbar` posee arreglos que permiten almacenar los datos de los objetos `DC/DC` conectados. Estos objetos `DC/DC`, poseen por su parte los voltajes nominales en cada extremo (`Vnom1` y `Vnom2`), así como los identificadores y referencias temporales directas a las barras DC a las que están conectados.

Además se tiene la clase `AC/DC` que define un conversor de voltaje AC/DC uni o bidireccional, como esta clase se extiende de DER, los detalles de su modelo se explican en la sección 4.2.3.

4.2.2. AC_Branch

La clase `AC_Branch` define las ramas de la red, es decir, los elementos con dos puntos de conexión. Se modela mediante la potencia aparente máxima (`Smax`), los voltajes nominales en cada extremo (`Vnom1` y `Vnom2`) y los identificadores y referencias temporales directas a las barras de conexión (`busbar1Key`, `busbar2Key`, `busbar1` y `busbar2`), además de las variables definidas en `Device`. Las variables `unit` y `frame` se utilizan para identificar las unidades de medida en las que están almacenadas las impedancias de la rama y si están referidas a componentes de fase o de secuencia, respectivamente. Nuevamente, los detalles del tipo de estas variables se pueden obervar en la sección 4.2.3.

De esta clase se extienden:

- **`Trafo`:** representa un transformador o banco de transformadores, posee como parámetros adicionales las derivaciones o *taps* (`tap`), el ángulo de desfase (`phaseShift`), el tipo de conexión (`connection`), y arreglos con los valores de resistencia, reactancia y susceptancia para las tres fases y las tres secuencias.
- **`Line`:** representa una línea de transmisión, posee como variables de instancia el largo de la línea (`lenght`) y, al igual que `Trafo`, arreglos con los valores de resistencia, reactancia y susceptancia por unidad de longitud para cada secuencia. Para las componentes de fase, en cambio, utiliza una matriz de (3×3), permitiendo modelar el acople entre fases.

4.2.3. DER

La clase `DER` incluye a todos los elementos con una única conexión a barra, es decir, las cargas (`DL`), unidades de generación (`DG`) y unidades de almacenamiento (`ESS`). La clase `DER` posee como variables de instancia el `key` de la barra de conexión y una referencia temporal directa a ella, el voltaje nominal, las potencias activas y reactivas nominales, máximas y mínimas, la potencia aparente máxima, y las resistencias y reactancias de secuencia negativa y cero, utilizadas para el modelo de unidades activas.

Adicionalmente, la clase `DER` almacena los datos de operación en dos grupos de variables temporales. Un primer grupo de variables individuales que almacena los datos de la operación que son configurados manualmente por el usuario, los que son utilizados en el flujo de potencia trifásico estático, es decir, sin considerar varios periodos de simulación. Y un segundo grupo de arreglos de variables, donde se almacenan los datos de operación para simulaciones que abarquen diversos periodos de simulación. Las variables de estos grupos poseen los mismos nombres con la salvedad de que aquellas cuyos nombres terminan con "`_s`" corresponden a las variables individuales, mientras que las otras a los arreglos.

Los datos almacenados temporalmente por estas variables corresponden a:

- Los modos de operación de la unidad: *slack*, PV, PQ, con inyecciones especificadas por fase o trifásicas totales.
- Voltajes especificados para unidades que regulen tensión.
- Potencias activas y reactivas por fase, que se utilizan en caso de que la unidad opere con inyecciones especificadas por fase.
- Potencias activa y reactiva trifásicas totales, utilizadas cuando la unidad opera con especificaciones de potencia trifásicas totales.
- Corrientes de secuencia negativa y cero, especificadas como magnitudes y ángulos. En una unidad generadora estas inyecciones de corriente por lo general son nulas, ya que las unidades trifásicas son por construcción balanceadas. Sin embargo, algunas unidades electrónicas como los inversores, pueden inyectar corrientes de secuencia negativa y/o cero con el fin de ejecutar una acción de control determinada, tal como se explica en 3.4.2.1.

Cabe mencionar que algunas de estas variables serán utilizadas sólo por alguna de las subclases de DER, dependiendo primordialmente si se trata de una unidad de generación o no. Tomando esto en consideración, como se observa en la figura 4.2, de esta clase derivan:

4.2.3.1. ESS:

Representa a las unidades de almacenamiento en general, posee como parámetro la potencia aparente nominal de la unidad (`Snom`).. De esta clase se extiende `BESS`, que corresponde a un sistema de almacenamiento por baterías, `BESS` tiene como parámetros el número de módulos en serie y paralelo (`series` y `parallel` respectivamente), una referencia al converso AC/DC utilizado (`converter`), una variable booleana que indica si se utiliza o si la unidad se conecta directamente a una barra DC (`useConverter`), y una referencia al módulo (`module`), que corresponde a un objeto del tipo `Battery`. Adicionalmente, almacena en las variables temporales `Econs` y `Efin` la energía total consumida y la energía remanente en el banco de baterías, luego de una simulación.

`Battery` corresponde al modelo de batería, que dentro de sus parámetros posee: `type` que indica el tipo de baterías utilizadas, potencia máxima de carga de las baterías (`PdisMax`), potencia de auto-descarga (`Pauto`), restricciones de energía mínima y máxima aceptable para la batería (`Emin` y `Emax`), energía inicial `Ein`, energía mínima final (`EminFinal`) y eficiencias de cargas y descarga (`effCh` y `effDis`). Al final de una simulación, se almacenan en las variables temporales `Psol` y `Esol` la potencia y energía entregada en cada periodo.

Adicionalmente, para definir la curva de carga de la batería se utilizan los parámetros: `beta`, `sigma`, `minInt` y `maxInt`, de acuerdo a lo descrito en [5].

4.2.3.2. DL:

Las cargas distribuidas se modelan de la siguiente manera:

$$P = P_0(V/V_0)^{\alpha} \tag{4.1}$$

$$Q = Q_0(V/V_0)^{\beta} \tag{4.2}$$

De esta forma si $\alpha = \beta = 0$ la carga es de potencia constante, si $\alpha = \beta = 1$ de corriente constante y si $\alpha = \beta = 2$ es de impedancia constante.

De esta forma, la clase DL es modelada mediante sus potencias activa y reactiva nominales (Pnom y Qnom), y los respectivos α yβ (alpha y beta). Además posee un objeto del tipo Demand que modela la característica de demanda y una variable binaria controllable que indica si la carga es controlable o no. De todas formas, la implementación de esta característica se encuentra fuera de los alcances de este trabajo, por lo que estas variables se incorporan con el fin de dar una guía para su implementación futura.

De DL además se extienden las clases Reactor y Capacitor, que modelan condensadores y reactancias *shunt* conectadas a una barra con el fin de regular tensión. Estas clases sólo poseen un arreglo tap, que indica los *taps* o derivaciones (discretas) en que están operando en cada periodo, al igual que la controlabilidad de las cargas, esta característica se incorpora en el modelo para su futura implementación, pues ésta se encuentra fuera de los alcances de este trabajo.

También se extiende de DL la clase SVC, que modela un regulador de tensión SVC, el que se modela como una unidad de generación con nula, o muy baja, potencia activa.

Finalmente se extiende Water_Pump que modela una bomba de agua. Esta clase posee como variables de instancia la eficiencia de la bomba (eff), un coeficiente constante (Kt), el volumen inicial y sus valores máximo y mínimo aceptables (Vol, Volmin y Volmax), y su característica de demanda, correspondiente a un objeto Water_Demand. Sus métodos son wf() que permite calcular el flujo de entrada de agua al estanque y P() que permite calcular la potencia consumida por la bomba. El modelo de esta clase se obtiene del modelo desarrollado en [5].

4.2.3.3. DG:

Al igual que ESS, esta clase posee como parámetro la potencia aparente nominal de la unidad (Snom) y no posee métodos. De ella se extienden: PV_Plant, Wind_Generator y Diesel_Generator.

- **`Diesel_Generator`:** posee como parámetros el tipo de combustible (`type`), costos de partida y parada (`Cstart` y `Cstop`), volumen del estanque de combustible y sus máximo y mínimo (`Vol`, `Volmax`, `Volmin`) y costo de combustible (`Cfuel`). Su curva de rendimiento, de acuerdo al modelo definido en [5], se define mediante los parámetros `alpha`, `beta`, `maxInt`, `minInt`, y al final de una simulación almacena en las variables temporales `totalFuel`, `totalCost` y `refillFreq`, el consumo total de combustible, el costo total de operación, y la frecuencia de rellenado del estanque de combustible, de forma que la unidad pueda operar sin problemas.

- **`Wind_Generator`:** posee el número de unidades conectadas (`number`), y referencias a la turbina eólica de tipo `Wind_Turbine` (`turbine`) y al conversor `AC/DC` asociado (`converter`). `Wind_Generator` funciona de la misma manera que `BESS`, es decir, puede ser asociado a una barra AC asociándose un conversor cuyo modelo será incorporado internamente, o asociándose una barra DC para conformar un grupo DC.

    - Wind_Turbine: define la turbina eólica, posee un recurso eólico asociado (`resource`) de tipo `Wind`. Y como método posee el cálculo de la potencia eléctrica generada, dado el viento `Pe()`.

- **`PV_Plant`:** modela a la planta fotovoltaica, se representa por el número de módulos en serie y en paralelo (`series` y `parallel`), el tipo de módulo (`module`) y el conversor asociado (`converter`). Esta clase también funciona como como `BESS` y como `Wind_Generator`, dependiendo si se conecta a una barra DC o AC.

    - **`PV_Module`:** posee el área del módulo (`area`), su eficiencia (`eff`), la latitud y longitud geográficas del lugar donde está instalado (`latitude` y `longitude`), los ángulos máximos y mínimos, norte y oeste (`minNorthAngle`, `maxNorthAngle`, `minWestAngle`, `maxWestAngle`), y la potencia nominal del panel (`Pnom`). Todos los parámetros y métodos de esta clase siguen el modelo desarrollado en [3].

4.3. Modelo de recursos energéticos y clases auxiliares

El modelo de datos y recursos energéticos de la herramienta se presenta a continuación:

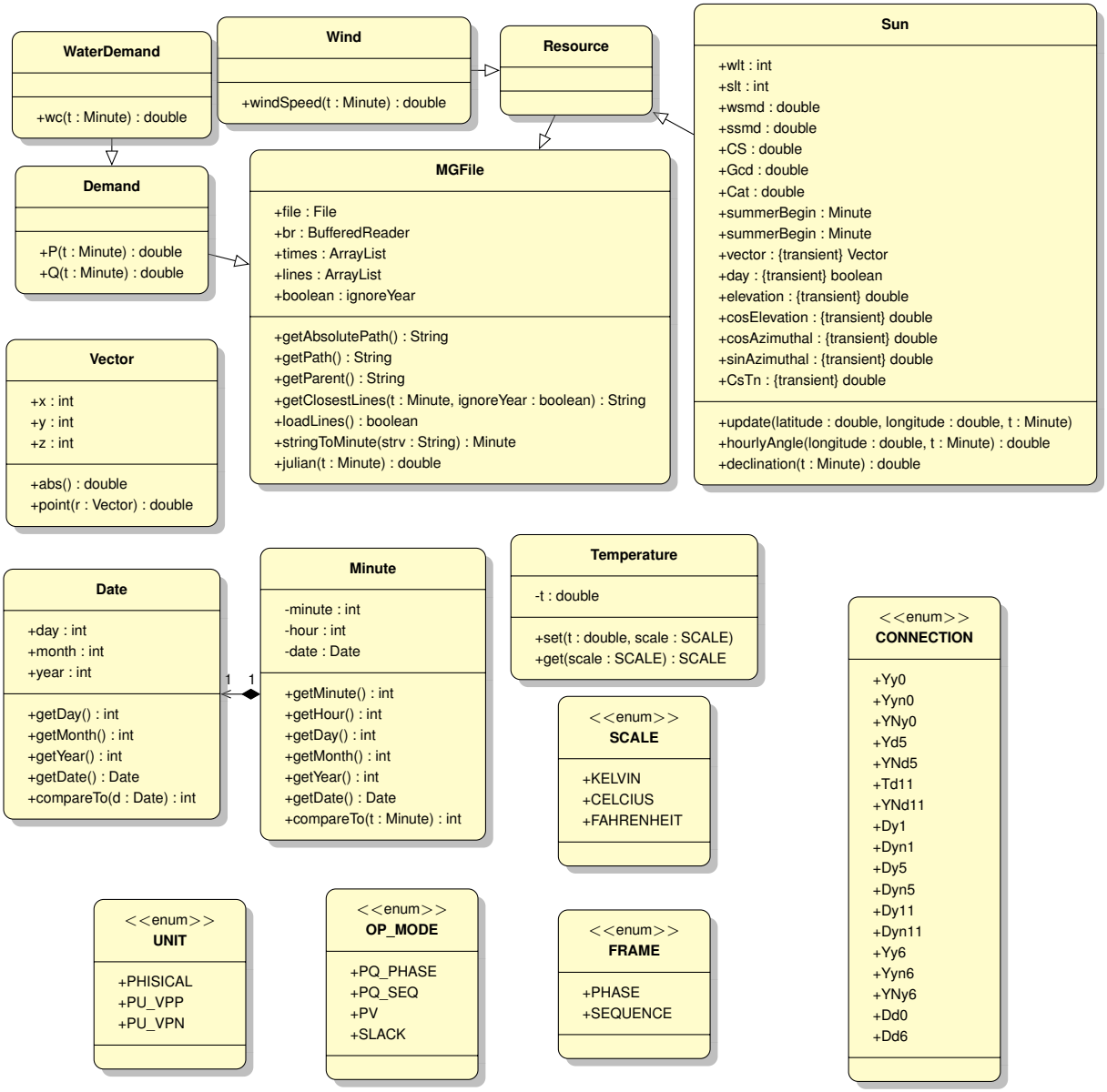

Figura 4.4: Diagrama de clases, recursos energéticos y clases auxiliares. Elaboración propia.

En la figura 4.4 es posible identificar las siguientes clases:

- **Resource:** El modelo de datos posee una clase abstracta `Resource` que modela un recurso energético, de la cuál se extienden las clases `Sun` y `Wind`, que modelan el sol y el viento respectivamente. Estas clases se asocian a los módulos PV y las turbinas eólicas respectivamente. Esta clase extiende a la clase MGFile.

- **MGFile:** encapsula un archivo, permitiendo ejecutar ciertas operaciones sobre él, lo que facilita su utilización dentro del contexto de la herramienta. Estas operaciones consisten en cargar el archivo y buscar datos en él. La implementación de estas operaciones escapa a los alcances de este trabajo, por lo que no son implementadas, dejándose propuestas como trabajo futuro.

- **Wind:** La clase `Resource`, posee un método `windSpeed(t : Minute)` que recibe un objeto `Minute` y calcula la velocidad del viento para ese periodo o instante de tiempo determinado. Este cálculo corresponde a la búsqueda del instante de tiempo dentro de un archivo con predicciones de potencia eólica, esta búsqueda está implementada dentro de la clase `MGFile`.

- **Sun:** Por su parte `Sun` posee como variables de instancia la latitud y longitud geográfica del lugar (`latitude` y `longitude`), las horaas locales de invierno y verano (`wlt` y `slt`), la hora a la que ocurre el mediodía solar en el lugar en invierno y verano (`wsmd` y `ssmd`), la constante solar (`CS`), la radiación directa de día claro(`Gcd`), la constante de atenuación o índice de claridad (`Cat`), y las fechas de inicio del horario de invierno y verano del lugar (`winterBegin` y `summerBegin`). Todo de acuerdo al modelo desarrollado en [3].

- **Vector:** Para poder describir vectores y operar con ellos se define la clase `Vector`, que posee como variables las componentes en los ejes `x`, `y`, `z` (`x`, `y`, `z`), y como métodos `point(r : vector)` que retorna el producto punto con el vector `r` recibido como parámetro, y `abs()` que retorna el módulo o norma del vector.

- **Minute:** La clase `Minute` describe un instante de tiempo medido en minutos. Esta clase además posee un objeto `Date` asociado que define la fecha.

- **Date:** Almacena el día, mes y año (`day`, `month` y `year`).

- **Temperature:** Modela la temperatura mediante una variable privada `t`, sólo accesible mediante los métodos `set(t : double, sclae : SCALE)` que permiten definir una temperatura dado un parámetro `t` que indica la temperatura y un parámetro `scale` que indica la escala respectiva. Y el método `get(scale : SCALE)`, que retorna la temperatura en la escala indicada como parámetro.

- **Demand:** Esta clase describe la curva de demanda de una carga determinada. Al igual que la clase `Wind` su implementación escapa a los alcances de este trabajo y se deja propuesta como trabajo futuro.
- **Water_Demand:** Análogamente a la clase anterior, `Water_Demand` define el consumo de agua. Esta clase define el método `wc(t : Time)` que retorna el caudal de agua consumido en un periodo de tiempo determinado, tal como se modela en [5].

Además de estas clases, la figura 4.4 presenta distintos tipos de datos auxiliares, conocidos como enumeraciones o *enums*. Estos tipos de datos se utilizan para realizar comparaciones, no para almacenar valores que tengan un significado propio. Los *enums* utilizados en el modelo eléctrico y de recursos en esta herramienta son:

- **UNIT:** Indica las unidades de medida utilizadas para valores de impedancia y admitancia, pudiendo ser unidades físicas (`PHISICAL`), en p.u. con referencia al voltaje fase-fase (`PU_VPP`), o p.u. con referencia al voltaje fase-neutro (`PU_VPN`).
- **OP_MODE:** Indica el modo de operación de una unidad DER, pudiendo ser `SLACK`, `PV`, PQ con inyecciones definidas en componentes de fase `PQ_PHASE` o PQ con inyecciones definidas en componentes de secuencia `PQ_SEQ`.
- **FRAME:** Indica si una rama está definida en componentes de fase o de secuencia, con los valores `PHASE` y `SEQUENCE`, respectivamente.

  SCALE: Indica una escala de temperatura `CELCIUS`, `KELVIN` y `FAHRENHEIT`.

  CONNECTION: Indica el tipo de conexión de los enrollados de un transformador trifásico.

4.4. Modelo de la visualización gráfica de la microrred

El modelo de la visualización gráfica de los componentes de la microrred se describe a continuación:

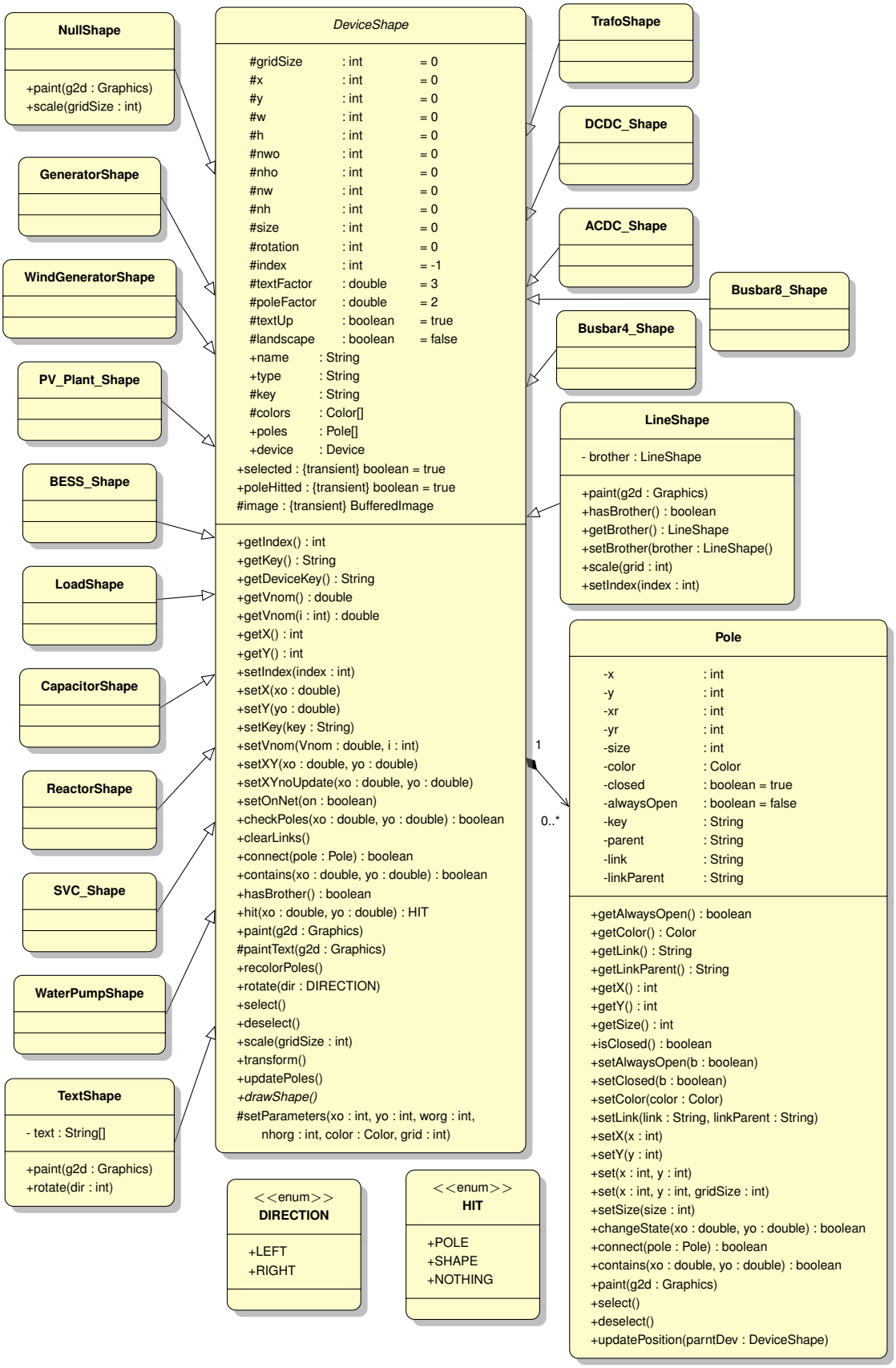

Figura 4.5: Diagrama de clases, modelo de la visualización gráfica de la microrred. Elaboración propia.

Como se puede observar el diagrama muestra dos grandes clases, `DeviceShape` y `Poles`, la primera consiste en el modelo general de las figuras, de la que se extienden las respectivas implementaciones para cada componente en particular, mientras que la segunda corresponde a los puntos de conexión de dichas figuras. A continuación se explican estas dos clases, además de las clases auxiliares utilizadas.

4.4.1. DeviceShape

Es una clase abstracta que corresponde al modelo general de las figuras utilizadas para representar gráficamente a los componentes de la microrred. Sin embargo, pese a ser una clase abstracta, esta clase define variables y métodos concretos, los que sientan las bases de la relación con los objetos `Device` mencionados en 4.2.

A grandes rasgos los objetos `DeviceShape` se identifican mediante un identificador de tipo `String` denominado `key`, y un índice `index`. El objeto posee una serie de parámetros físicos como sus dimensiones, posición en el panel de dibujo, entre otros, los que permiten dibujar la figura. La figura se dibuja en la variable interna `image` mediante el método `drawImage()`, luego esta imagen se dibuja en el panel de dibujo global de la microrred mediante el método `paint()`.

`DeviceShape` posee además dentro de sus variables un arreglo de objetos `Pole` que corresponden a los terminales de conexión del componente eléctrico que representa. Todas las operaciones de selección, escalamiento, rotación, movimiento, etc. se definen en esta clase. De esta forma, las subclases de `DeviceShape`, en su mayoría, slo modifican el método `drawImage()`, de acuerdo a su propia representación gráfica.

El detalle de los métodos y variables de esta clase se encuentra en el anexo A. Las subclases de `DeviceShape` son las siguientes:

- **GeneratorShape:** corresponde a la implementación de la figura de un generador genérico, se utilizará para el generador diesel.
- **WindGeneratorShape:** corresponde a la figura de un generador eólico.
- **PV_Plant_Shape:** corresponde a la figura de una planta fotovoltaica.
- **BESS_Shape:** representa un BESS.
- **LoadShape:** representa una carga.
- **CapacitorShape:** representa un condensador.

- **ReactorShape:** representa un reactor.
- **SVC_Shape:** representa un SVC.
- **WaterPumpShape:** representa una bomba de agua.
- **TrafoShape:** corresponde a la figura de un transformador, se dibuja utilizando 2 colores, uno para cada tensión.
- **DCDC_Shape:** representa un conversor DC/DC.
- **ACDC_Shape:** representa un conversor AC/DC, se dibuja utilizando 2 colores, uno para el lado de corriente alterna y otro para el de continua.
- **Busbar4_Shape:** corresponde a la figura de una barra con 4 puntos de conexión, además de dar una implementación concreta al método `+drawShape()`, `Busbar4_Shape` sobrescribe el método `#paintText(g2d : Graphics)` para insertar los textos de forma apropiada alrededor a la figura.
- **Busbar8_Shape:** análogo a `Busbar4_Shape`, salvo que representa una barra con 8 puntos de conexión.
- **LineShape:** para representar una línea se utilizan dos objetos `LineShape`, uno representando a cada extremo de ésta, para así poder mover cada extremo de forma independiente. Para esto la clase cuenta con una variable de instancia adicional `brother` que referencia al otro extremo de la línea, es decir al `key` de otro objeto `LineShape`.
- **NullShape:** permite insertar una figura nula o vacía, esto se utiliza para anular una figura sin la necesidad de preocuparse de considerar que la figura es nula y por tanto no se puede invocar sus métodos. Para esto sobrescribe los métodos `scale()` y `paint()` dándoles una implementación nula.
- **TextShape:** permite insertar texto, para esto incluye una variable de instancia adicional `text` que es un arreglo de objetos `String`, cada elemento del arreglo almacena una línea del texto insertado. Para su implementación, la clase sobrescribe los métodos `paint()` y `rotate()`, dándole al segundo una implementación nula.

4.4.2. Pole

Representa a los puntos de conexión de las figuras, dos figuras estarán conectadas cuando dos de sus poles estén en la misma posición. Si se presiona con el ratón sobre un polo de conexión éste cambiará su estado “abierto” o “cerrado”, si el polo está abierto se dibuja como un cuadrado sin relleno, en este caso equivale a que no hubiera conexión eléctrica.

Como sólo es posible conectar componentes a una barra, es decir, no es posible conectar ramas sucesivamente, ni un objeto DER a una rama directamente, los polos de los objetos `BusbarShape` están siempre gráficamente abiertos. Sin embargo, internamente se considera como abierto, por lo que el estado de la conexión depende netamente del polo de la unidad conectada.

Un objeto `Pole` se identifica mediante un parámetro de tipo `String` denominado `key`, e identifica a la figura a la que pertenece en el parámetro `parent`. Además almacenará el identificador del polo al que está conectado en `link`, y el código del padre de éste en `linkParent`.

4.5. Modelo global de la microrred

En la figura 4.6 se puede observar el modelo global de la microrred, es decir, cómo se relacionan los objetos gráficos con los que modelan el comportamiento eléctrico de las unidades.

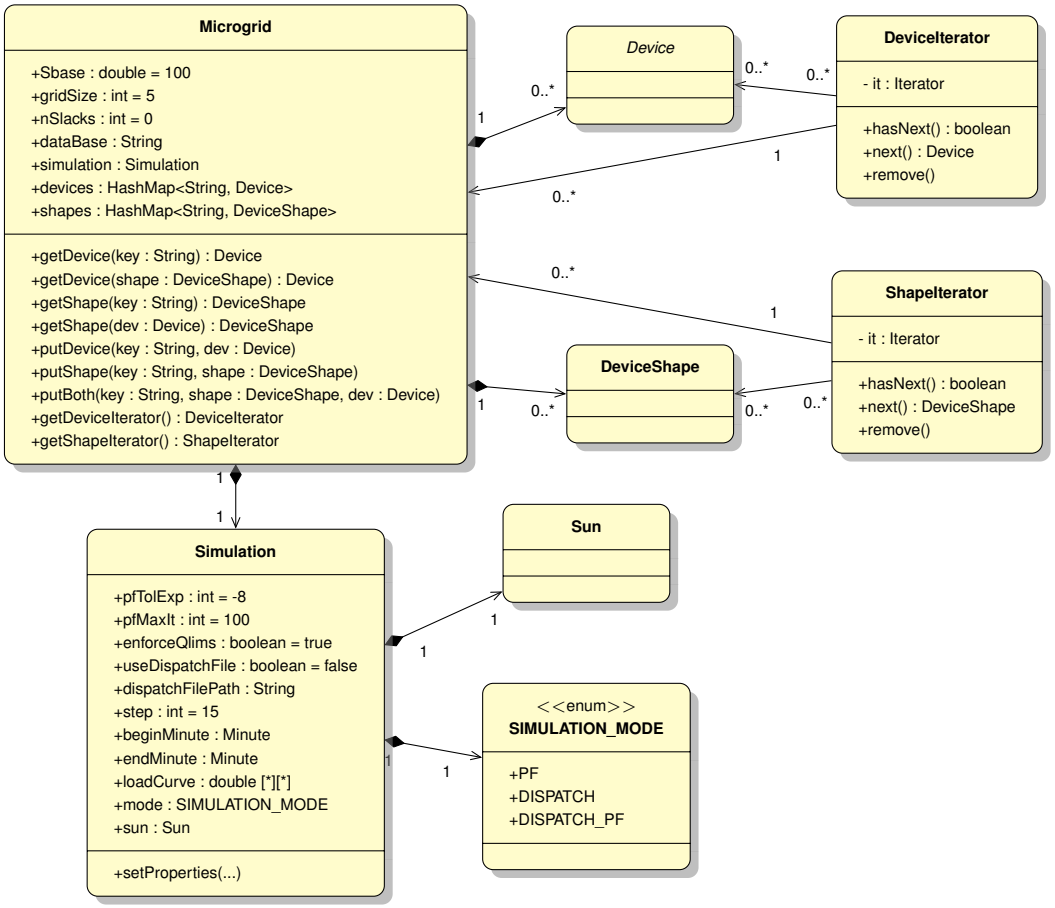

Figura 4.6: Diagrama de clases, modelo global de la microrred. Elaboración propia.

4.5.1. Microgrid

Contiene a todos los objetos `Device` y sus correspondientes figuras `DeviceShape`. Para esto utiliza objetos de tipo `HashMap` que almacenan pares (`String`, `Device`) y (`String`, `DeviceShape`), y se referencian con las variables `devices` y `shapes`, respectivamente. En estos pares, los `String` corresponden al `key` o identificador del objeto asociado, de esta forma, teniendo el mismo key en el objeto gráfico y en el objeto eléctrico, es posible acceder de uno al otro a través del objeto `Microgrid`.

Para realizar esto, la clase `Microgrid` define distintos métodos de búsqueda (`get`) e inserción (`put`) para ambos tipos de objetos. Adicionalmente define métodos que retornan iteradores (`DeviceIterator` y `ShapeIterator`) para ambas estructuras de datos, lo que permite recorrer completamente cualquiera de estas dos listas.

Adicionalmente, la clase `Microgrid` define otras variables que almacenan: la potencia base del sistema (`Sbase`), el tamaño de la grilla de dibujo (lo que permite almacenar el nivel de *zoom*) (`gridSize`), la cantidad de unidades *slack* (`nSlacks`), el nombre de la base de datos utilizada para el despacho económico (esto se explica con más detalle en la sección 6.4) (`dataBase`), y la configuración de la simulación (`simulation`), que corresponde a un objeto del tipo `Simulation`.

4.5.2. Simulation

Almacena la configuración de la simulación mediante las siguientes variables:

- **`pfTolExp`:** corresponde al exponente de la tolerancia utilizada para el flujo de potencia, considerando base 10. Por ejemplo, si la tolerancia es de 10^{-8}, `pfTolExp` tendrá asignado el valor `-8`.
- **`pfMaxIt`:** máximas iteraciones para el flujo de potencia.
- **`enforceQlims`:** corresponde a una variable binaria que indica si se deben verificar o no, los límites de reactivos en la ejecución del flujo de potencia.
- **`useDispatchFile`:** corresponde a una variable binaria que indica si se carga manualmente un archivo de despacho o si se ejecuta el módulo de despacho internamente.
- **`dispatchFilePath`:** corresponde a la ruta completa del archivo de despacho mencionado en el punto anterior.
- **`step`:** paso de tiempo, en minutos, entre dos periodos de simulación consecutivos.
- **`beginMinute`:** corresponde al periodo inicial de simulación.
- **`endMinute`:** corresponde al periodo final de simulación.
- **`loadCurve`:** corresponde a un arreglo que contiene la curva de variación de carga, utilizada para darle variabilidad a las cargas para el horizonte de simulación.
- **`sun`:** es un objeto `Sun`, y corresponde a la representación del sol en la microrred, se utiliza para modelar el recurso solar utilizado por las plantas fotovoltaicas.
- **`mode`:** corresponde al modo de simulación, es una variable de tipo `SIMULATION_MODE`, definida a continuación.

4.5.3. SIMULATION_MODE

Corresponde a un *enum* que indica el tipo de simulación a realizar dentro de tres opciones, estas son: flujo de potencia trifásico individual (`PF`), despacho económico para un horizonte de simulación determinado (`DISPATCH`) o despacho económico seguido de un flujo de potencia para cada uno de los periodos de simulación (`DISPATCH_PF`).

4.5.4. ShapeIterator

Implementa un iterador utilizado para recorrer la estructura que contiene las figuras. Para esto la clase encapsula un `Iterator` de Java mediante la variable de instancia `it`, y define los métodos:

- `+hasNext() : boolean` que indica si el elemento actualmente referenciado por el iterador posee un sucesor en la lista.
- `+next() : DeviceShape` que permite obtener la figura referenciada actualmente por el iterador y luego hacer que éste pase a la siguiente.
- `+remove()` que remueve de la lista al elemento referenciado por el iterador.

4.5.5. DeviceIterator

Completamente análogo al anterior, salvo que implementa un iterador utilizado para recorrer la estructura que contiene los dispositivos.

4.6. Esquema general de la herramienta

4.6.1. Simbología utilizada en los diagramas de flujo

Para presentar el esquema general de la herramienta se utiliza un diagrama de flujo que indica los procesos realizados por ésta. Estos diagramas utilizan la siguiente simbología:

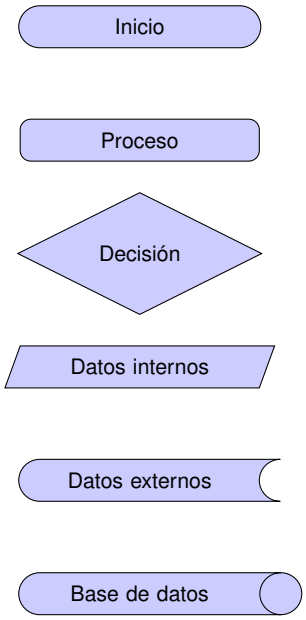

Figura 4.7: Simbología utilizada en los diagramas de flujo. Elaboración propia.

4.6.2. Diagrama de flujo general de la herramienta

A continuación se presenta el esquema general de la herramienta:

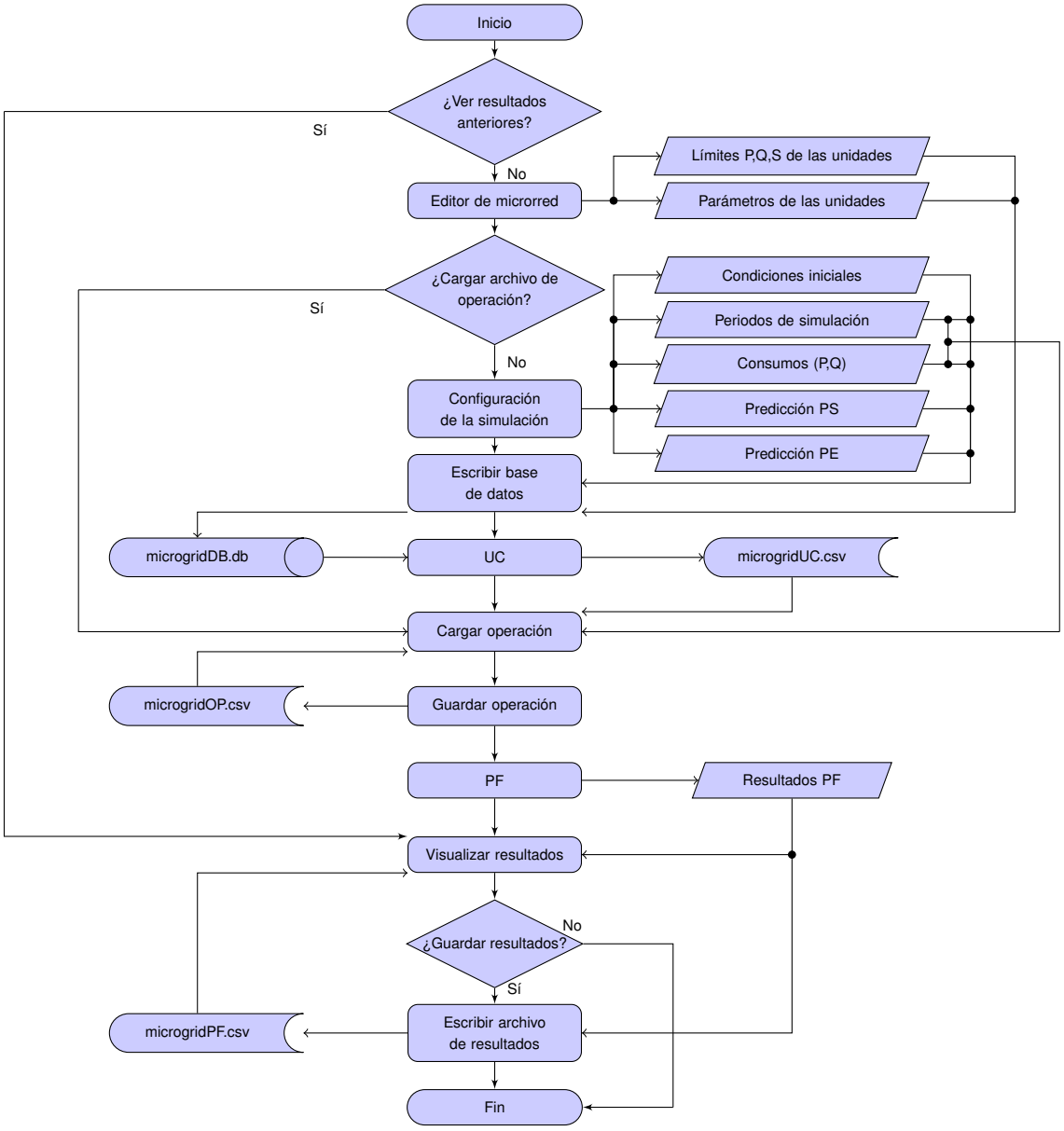

Figura 4.8: Diagrama de flujo general de la herramienta. Elaboración propia.

Como se observa en la figura 4.8, en el editor de microrred se configuran los parámetros de las unidades, la información de los consumos, las condiciones de operación y las restricciones del problema de optimización que entrega el despacho. Luego a partir de estos datos, se seleccionan los que requiere el módulo de despacho para su funcionamiento y se escriben en una base de datos (`microgridDb.db`).

Luego, se ejecuta el módulo de despacho, el cual leerá el contenido de la base de datos y entregará un archivo de salida (`microgridUC.csv`) que contiene el despacho de unidades. Este archivo luego es leído, se recupera su información y se carga nuevamente en la estructura interna del programa. Esta información, en conjunto con el resto de la información relativa a la operación, puede ser guardada en un archivo de despacho (`microgridOP.csv`) que no sólo contiene las consignas de operación de las unidades, sino además la información de los consumos de forma individual, incluyendo los datos de potencia reactiva.

Una vez recuperada la información, y con el despacho completamente listo, se ejecuta el módulo de flujo de potencia trifásico, ejecutando un flujo de potencia para cada periodo de simulación. Los resultados de estos flujos de potencia son leídos por la herramienta y son desplegados de forma gráfica en la ventana de visualización de resultados, teniendo la posibilidad de almacenarse en un archivo.

Adicionalmente, desde la interfaz gráfica de la herramienta es posible omitir ciertos pasos cargando archivos previamente guardados que contengan los resultados correspondientes. En efecto, en vez de ejecutar el módulo de despacho es posible cargar directamente un archivo `microgridUC.csv` con el resultado del despacho, o también cargar un archivo `microgridOP.csv` con la información completa de despacho, es decir, incluyendo reactivos y los datos individuales de carga. También es posible no ejecutar ninguna simulación y simplemente visualizar un archivo de resultados obtenido previamente.

Cabe mencionar que, si bien éste corresponde al modelo global de la herramienta, no todas estas funcionalidades son implementadas en este trabajo, ya que escapan a los alcances planteados. Por lo tanto, este modelo se presenta con la finalidad de ser la guía para el desarrollo futuro de esta herramienta.

Más detalles sobre la implementación de este modelo, y sobre la ejecución de los módulos de despacho y flujo de potencia trifásico, se encuentran en el capítulo 6.

Capítulo 5

Desarrollo del flujo de potencia trifásico y validación

El flujo de potencia trifásico desarrollado se basa en [17] y [23], y resuelve de manera independiente los voltajes de secuencias positiva, negativa y cero, utilizando el algoritmo de Newton Raphson para la primera y resolviendo un sistema de ecuaciones nodales del estilo: $\vec{I} = [Y] \cdot \vec{V}$ para las otras.

Como algunos datos del flujo, por ejemplo las potencias consumidas por las cargas, están especificados por fase, y su transformación a componentes de secuencia depende de los voltajes de fase y/o secuencia, es necesario relacionar las tensiones de las tres secuencias entre sí para obtener los voltajes de fase en cada iteración. De esta manera en cada iteración del flujo de potencia trifásico se corre sólo una iteración del algoritmo de Newton Raphson para la secuencia positiva, se resuelven las ecuaciones nodales para las otras secuencias y finalmente se calculan los voltajes de fase a partir de estos 3 resultados. El detalle del algoritmo se puede visualizar en la sección 5.3.

5.1. Modelo de componentes

A continuación se explican los modelos utilizados para los distintos componentes de una microrred. Cabe mencionar que todos los elementos de impedancia constante (líneas, cargas de impedancia constante, reactancias de generadores y transformadores, incluyendo sus taps) son incluidos en las matrices de admitancia en componentes de secuencia del sistema.

5.1.1. Cargas

Las cargas del sistema se modelan como consumos por fase, pudiendo ser de potencia constante o de impedancia constante, en caso de ser consumos de potencia constante, éstos se caracterizan por un consumo de potencia activa y uno de potencia reactiva en cada fase. Si se trata de una carga de impedancia constante, es decir una rama *shunt*, ésta se caracteriza por las partes real (conductancia) e imaginaria (susceptancia) de una carga conectada entre la barra y tierra.

Adicionalmente es posible modelar cargas activas, tales como máquinas, para lo que se utiliza el modelo de máquina sincrónica explicado en 5.1.3, pero con inyecciones de potencia negativas.

Este modelo, si bien no abarca todas las posibilidades del modelo descrito en 4.2.3.2, incorpora los modelos más comunes. La incorporación completa de este modelo de cargas se deja propuesta como trabajo futuro.

5.1.2. Ramas

El modelo general de las ramas consiste en representar la rama mediante una matriz de admitancia nodal. Si se denomina la barra de origen o *"from bus"* con la letra f, y la barra de destino o *"to bus"* con la letra t, el modelo de bipuerta, es decir, las ecuaciones que describen el comportamiento de la rama en ambos extremos se pueden escribir matricialmente como:

$$\begin{bmatrix} I_f \\ I_t \end{bmatrix} = \begin{bmatrix} y_{ff} & y_{ft} \\ y_{tf} & y_{tt} \end{bmatrix} \cdot \begin{bmatrix} V_f \\ V_t \end{bmatrix} \tag{5.1}$$

Si el sistema posee nb barras y nl ramas, es posible construir vectores de $(nl \times 1)$ $\vec{Y_{ff}}$, $\vec{Y_{ft}}$, $\vec{Y_{tf}}$ e $\vec{Y_{tt}}$, que contengan los elementos y_{ff}, y_{ft}, y_{tf} e y_{tt} de cada rama. Luego estos vectores $\vec{Y_{jk}}$ de $(nl \times 1)$ se transforman en matrices diagonales $[Y_{jk}]$ de $(nl \times nl)$ con los valores del vector en su diagonal. Luego se construyen matrices de conexión C_f y C_t de orden $(nl \times nb)$ que permiten conectar las matrices anteriores, de forma de obtener matrices con las admitancias vistas desde la barra de origen y desde la barra de destino de la siguiente manera:

$$[Y_f] = [Y_{ff}]C_f + [Y_{ft}]C_t \tag{5.2}$$

$$[Y_t] = [Y_{tf}]C_f + [Y_{tt}]C_t \tag{5.3}$$

Donde $[Y_f]$ e $[Y_t]$ son matrices de $(nl \times nb)$ que relacionan las corrientes inyectadas en cada extremo de las nl ramas con los voltajes en las nb barras mediante:

$$\vec{I_f} = [Y_f]\vec{V} \tag{5.4}$$

$$\vec{I_t} = [Y_t]\vec{V} \tag{5.5}$$

Cabe mencionar que el elemento (i, j) de la matriz C_f y el elemento (i, k) de la matriz C_t serán iguales a 1 para cada rama i que conecte la barra de origen (f) j con la barra de destino (t) k. Todos los demás elementos serán iguales a cero.

Finalmente la matriz de admitancia nodal del sistema se calcula a partir de las matrices $[Y_f]$ e $[Y_t]$ mediante:

$$[Y] = C_f^T[Y_f] + C_t^T[Y_{ft}] \tag{5.6}$$

A continuación se presentan los modelos de líneas y transformadores, una vez obtenidos los valores de y_{ff}, y_{ft}, y_{tf} e y_{tt}, la construcción de las matrices de admitancia nodales se realiza siguiendo el mismo procedimiento anterior pero para cada secuencia.

5.1.2.1. Líneas

El modelo Π de una línea balanceada se caracteriza por una impedancia serie z^z y una admitancia *shunt* y^s que se divide en ambos extremos de la línea. Es decir, en el modelo de rama descrito anteriormente se tienen los siguientes valores para la matriz de admitancia nodal de la rama:

$$y_{ff} = y_{tt} = 1/z^z + y^s/2 \tag{5.7}$$

$$y_{ft} = y_{tf} = -1/z^z \tag{5.8}$$

En el caso de una línea desbalanceada es posible replicar este modelo en cada fase y/o secuencia, permitiendo modelar líneas desbalanceadas. Sin embargo, los efectos de una pobre transposición en la línea no son incorporados ya que éstos incorporan acoples entre fases [24], por esta razón la forma general de modelar una línea desbalanceada en componentes de fase es mediante una matriz de impedancia serie y una matriz de admitancia *shunt* [24]:

$$[Z^z_{abc}] = \begin{bmatrix} z^z_{aa} & z^z_{ab} & z^z_{ac} \\ z^z_{ba} & z^z_{bb} & z^z_{bc} \\ z^z_{ca} & z^z_{cb} & z^z_{cc} \end{bmatrix}, [Y^s_{abc}] = \begin{bmatrix} y^s_{aa} & y^s_{ab} & y^s_{ac} \\ y^s_{ba} & y^s_{bb} & y^s_{bc} \\ y^s_{ca} & y^s_{cb} & y^s_{cc} \end{bmatrix} \tag{5.9}$$

Estas matrices de impedancia y admitancia en componentes de fase son siempre simétricas, los componentes diagonales dan cuenta de los parámetros por fase mientras que los elementos fuera de la diagonal corresponden al acople entre fases. Si la línea está correctamente transpuesta[1] estos acoples se eliminan, quedando una matriz diagonal. Luego para obtener las matrices en componentes de secuencia se transforman las matrices en componentes de fase de la siguiente manera:

$$[Z^z_{012}] = [T]^{-1} \cdot [Z^z_{abc}] \cdot [T] = \begin{bmatrix} z^z_{00} & z^z_{01} & z^z_{02} \\ z^z_{10} & z^z_{11} & z^z_{12} \\ z^z_{20} & z^z_{21} & z^z_{22} \end{bmatrix} \tag{5.10}$$

$$[Y^s_{012}] = [T]^{-1} \cdot [Y^s_{abc}] \cdot [T] = \begin{bmatrix} y^s_{00} & y^s_{01} & y^s_{02} \\ y^s_{10} & y^s_{11} & y^s_{12} \\ y^s_{20} & y^s_{21} & y^s_{22} \end{bmatrix} \tag{5.11}$$

Donde $[T]$ corresponde a la matriz de Fortescue:

[1] Se dice que una línea está correctamente transpuesta cuando ésta presenta transposiciones entre sus fases, de tal forma que se eliminan los acoples entre ellas y sus impedancias pueden considerarse iguales.

$$[T] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1\angle -120^\circ & 1\angle 120^\circ \\ 1 & 1\angle 120^\circ & 1\angle -120^\circ \end{bmatrix} \tag{5.12}$$

Cabe mencionar que si las matrices en componentes de fase son diagonales, con todos sus elementos iguales, sus análogos en componentes de secuencia serán también diagonales, es decir se eliminan los acoples entre secuencias, sin embargo, si hay al menos un elemento distinto en la diagonal, esto no se cumplirá. Finalmente, la matriz de impedancia serie se transforma en una matriz de admitancia serie:

$$[Y^z_{012}] = [Z^z_{012}]^{-1} = \begin{bmatrix} y^z_{00} & y^z_{01} & y^z_{02} \\ y^z_{10} & y^z_{11} & y^z_{12} \\ y^z_{20} & y^z_{21} & y^z_{22} \end{bmatrix} \tag{5.13}$$

De esta manera el modelo Π de la línea en componentes de secuencia queda de la siguiente manera:

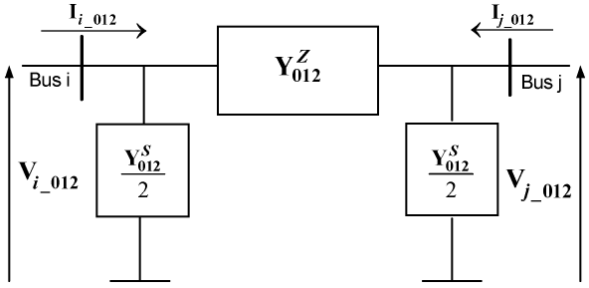

Figura 5.1: Modelo de línea desbalanceada en componentes de secuencia. Fuente: [17].

Para representar el sistema completo, si las líneas fueran balanceadas y correctamente transpuestas, bastaría construir una matriz de admitancia del sistema para cada secuencia, utilizando los elementos diagonales de la matriz, ya que los no diagonales serían nulos. De esta forma los sistemas a resolver serían de la forma:

$$\vec{I} = \vec{I_G} - \vec{I_D} = [Y] \cdot \vec{V} \tag{5.14}$$

Sin embargo esto no es posible debido a la existencia de elementos no diagonales en las matrices. Para abordar esto es posible construir matrices de admitancia sistémicas que relacionen las corrientes de una secuencia con los voltajes de otra, por ejemplo: $\vec{I_0} = [Y_{01}] \cdot \vec{V_1}$. De esta forma los sistemas a resolver serían:

$$\vec{I_0} = [Y_0]\vec{V_0} + [Y_{01}]\vec{V_1} + [Y_{02}]\vec{V_2} \tag{5.15}$$
$$\vec{I_1} = [Y_1]\vec{V_1} + [Y_{10}]\vec{V_0} + [Y_{12}]\vec{V_2} \tag{5.16}$$
$$\vec{I_2} = [Y_2]\vec{V_2} + [Y_{20}]\vec{V_0} + [Y_{21}]\vec{V_1} \tag{5.17}$$

Donde las matrices $[Y_1]$, $[Y_2]$, $[Y_2]$ representan al sistema en cada secuencia, mientras que $[Y_{01}]$, $[Y_{02}]$, $[Y_{10}]$, $[Y_{12}]$, $[Y_{20}]$, $[Y_{21}]$ dan cuenta del acople entre secuencias. Evidentemente este sistema es difícil de resolver computacionalmente por lo que, para simplificar el algoritmo, se simplifica el sistema incorporando las corrientes de acople al vector del lado izquierdo de la ecuación, es decir, de las ecuaciones anteriores se llega a [25]:

$$\vec{I_0} - ([Y_{01}]\vec{V_1} + [Y_{02}]\vec{V_2}) = [Y_0]\vec{V_0} \tag{5.18}$$
$$\vec{I_1} - ([Y_{10}]\vec{V_0} + [Y_{12}]\vec{V_2}) = [Y_1]\vec{V_1} \tag{5.19}$$
$$\vec{I_2} - ([Y_{20}]\vec{V_0} + [Y_{21}]\vec{V_1}) = [Y_2]\vec{V_2} \tag{5.20}$$

Luego estos sistemas se resuelven iterativamente, calculando las inyecciones de corrientes de acople utilizando los voltajes de una iteración, para con ellas calcular los nuevos voltajes. El detalle de este procedimiento se especifica en 5.3.

5.1.2.2. Transformadores

El modelo de un transformador trifásico corresponde a un transformador ideal con razón de transformación $a : 1$ y ángulo de desfase α, seguido de una impedancia serie. Los parámetros del transformador suelen entregarse por secuencia, pero de lo contrario, se puede aplicar un procedimiento análogo al de la línea. De esta forma, el modelo del transformador en componentes de secuencia corresponde a:

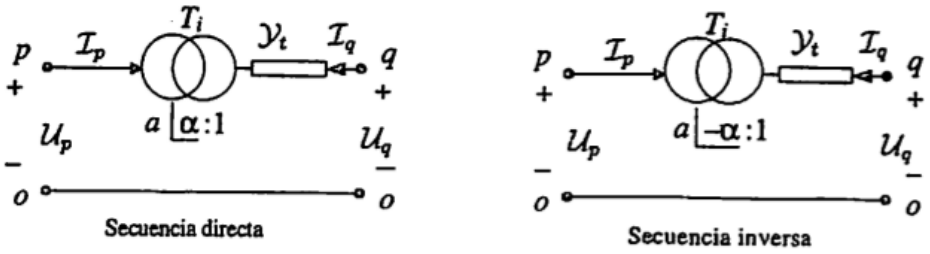

Figura 5.2: Modelo de transformador trifásico, secuencias positiva o directa y negativa o inversa. Fuente: [24].

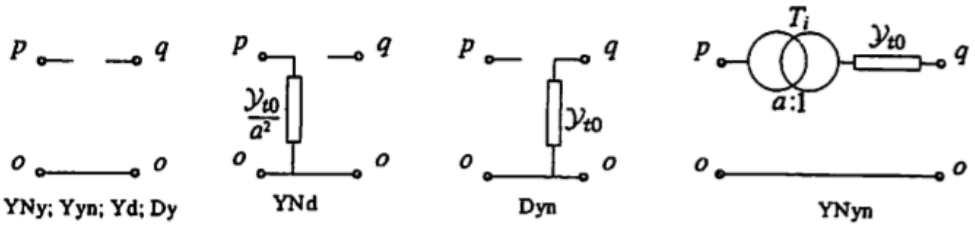

Figura 5.3: Modelo de transformador trifásico, secuencia cero u homopolar. Fuente: [24].

Así los valores de las admitancias y_{ff}, y_{ft}, y_{tf} e y_{tt} se resumen en la siguiente tabla [24], [17]:

Tabla 5.1: Transformador trifásico, admitancias de secuencia del modelo de bipuerta. Elaboración propia.

Admitancia	Secuencia Positiva	Secuencia Negativa	Secuencia Cero			
			YNyn	YNd	Dyn	Otros
Y_{ff}	Y_{t1}/a^2	Y_{t2}/a^2	Y_{t0}/a^2	Y_{t0}/a^2	0	0
Y_{tt}	Y_{t1}	Y_{t2}	Y_{t0}	0	Y_{t0}	0
Y_{ft}	$-\frac{Y_{t1}\angle\alpha}{a}$	$-\frac{Y_{t2}\angle-\alpha}{a}$	Y_{t0}/a	0	0	0
Y_{tf}	$-\frac{Y_{t1}\angle-\alpha}{a}$	$-\frac{Y_{t2}\angle\alpha}{a}$	Y_{t0}/a	0	0	0

De acuerdo al modelo general de rama desbalanceada y con acoples entre secuencias, es posible calcular las inyecciones de corriente en cada extremo de ella de la siguiente manera:

$$\vec{I_{f0}} = [Y_{f0}]\vec{V_0} + [Y_{f01}]\vec{V_1} + [Y_{f02}]\vec{V_2} \tag{5.21}$$
$$\vec{I_{f1}} = [Y_{f1}]\vec{V_1} + [Y_{f10}]\vec{V_0} + [Y_{f12}]\vec{V_2} \tag{5.22}$$
$$\vec{I_{f2}} = [Y_{f2}]\vec{V_2} + [Y_{f20}]\vec{V_0} + [Y_{f21}]\vec{V_1} \tag{5.23}$$
$$\vec{I_{t0}} = [Y_{t0}]\vec{V_0} + [Y_{t01}]\vec{V_1} + [Y_{t02}]\vec{V_2} \tag{5.24}$$
$$\vec{I_{t1}} = [Y_{t1}]\vec{V_1} + [Y_{t10}]\vec{V_0} + [Y_{t12}]\vec{V_2} \tag{5.25}$$
$$\vec{I_{t2}} = [Y_{t2}]\vec{V_2} + [Y_{t20}]\vec{V_0} + [Y_{t21}]\vec{V_1} \tag{5.26}$$

Y luego pueden ser transformadas a inyecciones de potencia, por ejemplo para la secuencia m:

$$\vec{S}_{fm} = [\vec{V}_{fm}]^T(\vec{I}_{fm})^* \tag{5.27}$$
$$\vec{S}_{tm} = [\vec{V}_{tm}]^T(\vec{I}_{tm})^* \tag{5.28}$$

5.1.3. Unidades de generación

Las máquinas sincrónicas se modelan en componentes de fase como reactancias detrás de fuentes de tensión, correspondientes a las FEM o voltajes internos por fase de la máquina. En componentes de secuencia sólo la secuencia positiva posee una fuente de tensión ya que los voltajes internos son balanceados [26], por lo que el modelo en secuencias negativa y cero se reduce a una reactancia *shunt*, como se muestra en la figura 5.4:

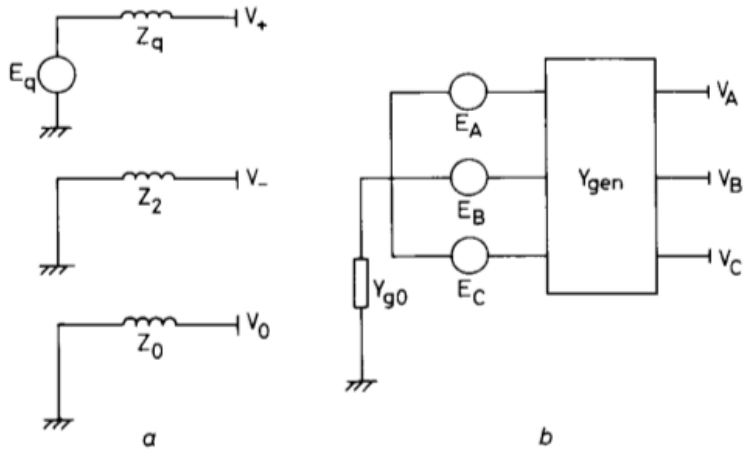

Figura 5.4: Generadores sincrónicos, a) Modelo en componentes de secuencia, b) Modelo en componentes de fase. Fuente: [26].

De esta forma, dependiendo del desbalance del sistema, aparecerán voltajes de secuencias negativa y cero en la barra del generador pese a que éste posea sólo una fuente de tensión en secuencia positiva, permitiendo que el generador opere desbalanceadamente. Con este modelo, asumiendo que la función de control de la máquina (PQ, PV, *slack*) está referida a la secuencia positiva y en la barra de conexión, no en su FEM interna, es posible eliminar la reactancia de secuencia positiva, dejando sólo una fuente de tensión en dicha secuencia [27].

Por otro lado, dadas las posibilidades de la electrónica de potencia, a las unidades conectadas mediante un inversor, es posible asignarle funciones de control adicionales mediante la inyección de ciertas corrientes de secuencia negativa y/o cero, por ejemplo, para contrarrestar el desbalance del sistema o para detectar la operación en isla del sistema [23]. Para incorporar esto al modelo, se incluyen fuentes de corriente en paralelo a las admitancias *shunt* de secuencias negativa y cero, como se muestra en la figura 5.5.

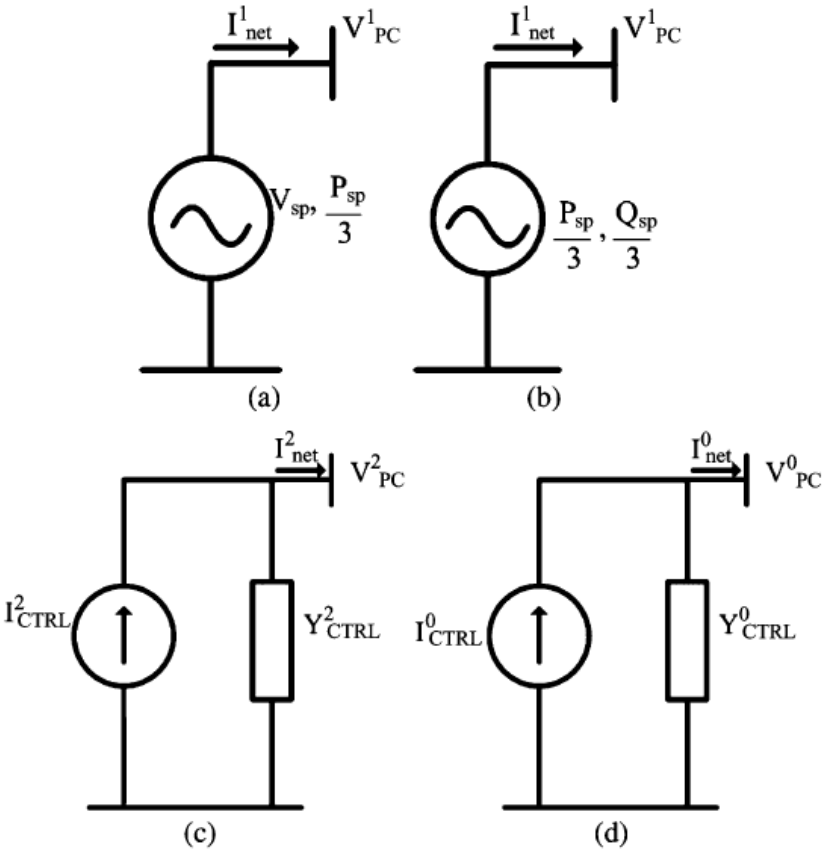

Figura 5.5: Modelo general de unidades de generación. (a) Modelo de secuencia positiva para modo PV, (b) Modelo de secuencia positiva para modo PQ, (c) Modelo de secuencia negativa, (d) Modelo de secuencia cero. Fuente: [23].

De esta forma, el modelo de generador sincrónico simple se obtiene asignando corrientes I^2_{CTRL} e I^0_{CTRL} nulas. Además, si la unidad está conectada con el neutro asilado de tierra, el modelo de secuencia cero desaparece, es decir $I^0_{CTRL} = Y^0_{CTRL} = 0$.

Las inyecciones en las secuencias negativa y cero corresponden al voltaje multiplicado por el conjugado de la corriente de control menos lo consumido por la admitancia de secuencia *shunt*:

$$S^{02} = V^{02} \cdot (I^{02}_{CTRL} - jY^{02}_{CTRL}V^{02})^* \tag{5.29}$$

$$P^{02} = \Re(S^{02}) \tag{5.30}$$

$$Q^{02} = \Im(S^{02}) \tag{5.31}$$

En cambio para la secuencia positiva se debe hacer un balance nodal, es decir, en cada barra la suma de las potencias inyectadas por los generadores S_{G1} debe ser igual a la suma de la potencia consumida por las cargas y la inyectada a las ramas. Luego, matricialmente:

$$\vec{S}_{G1} = (\vec{P}_{D1} + j\vec{Q}_{D1}) + [Y_1] \cdot \vec{V}_1 + [Y_{10}] \cdot \vec{V}_0 + [Y_{12}] \cdot \vec{V}_2 \tag{5.32}$$

Después, excluyendo las funciones de control adicionales proporcionadas por las secuencias negativa y cero, las unidades pueden clasificarse de la siguiente forma:

5.1.3.1. Unidades PQ

Se especifican las potencias activa y reactiva de secuencia positiva, estos valores corresponden a un tercio de las potencias trifásicas totales especificadas.

$$P_{sp1} = P_{sp3\phi}/3 \tag{5.33}$$

$$Q_{sp1} = Q_{sp3\phi}/3 \tag{5.34}$$

La magnitud y el ángulo del voltaje de secuencia positiva son resultado del algoritmo.

5.1.3.2. Unidades PV

Se especifica la magnitud del voltaje y la potencia activa de secuencia positiva, al igual que el caso anterior, la potencia activa corresponde a un tercio de la potencia activa trifásica total especificada.

$$|V_{sp1}| = |V_{sp}| \tag{5.35}$$

$$P_{sp1} = P_{sp3\phi}/3 \tag{5.36}$$

El ángulo de voltaje de secuencia positiva es resultado del flujo de potencia, la potencia reactiva se calcula a partir de la parte imaginaria de la ecuación 5.32.

$$\vec{Q}_{G1} = \Im(\vec{S}_{G1}) = \vec{Q}_{D1} + \Im([Y_1] \cdot \vec{V_1} + [Y_{10}] \cdot \vec{V_0} + [Y_{12}] \cdot \vec{V_2}) \tag{5.37}$$

Cabe mencionar que este vector $\vec{Q_{G1}}$ contiene la potencia reactiva requerida en cada barra para mantener la tensión especificada, por lo que para asignarla al(los) generador(es) conectado(s) a la barra primero hay que restar las inyecciones de potencia reactiva de los generadores PQ. Luego, si hay más de una unidad PQ (o incluso la *slack*) conectada a la barra, esta potencia reactiva se puede dividir de distintas formas entre los generadores, por ejemplo en partes iguales, en proporción a su capacidad reactiva, etc. De esta forma la potencia reactiva total que deben inyectar las unidades PV y/o *slack* corresponden a:

$$\vec{Q}_{G1_{PV}} = \vec{Q}_{D1} - \vec{Q}_{G1_{PQ}} + \Im([Y_1] \cdot \vec{V_1} + [Y_{10}] \cdot \vec{V_0} + [Y_{12}] \cdot \vec{V_2}) \tag{5.38}$$

5.1.3.3. Unidades *slack*

Se especifica la magnitud y ángulo de voltaje de secuencia positiva.

$$|V_{sp1}| = |V_{sp}| \tag{5.39}$$

$$\angle V_{sp1} = \angle V_{sp} \tag{5.40}$$

La potencia reactiva que debe inyectar el generador *slack* se calcula de la misma manera que para las unidades PV, mediante la ecuación 5.38. Para la potencia activa en cambio se debe tomar la parte real de la ecuación 5.32 y restar las inyecciones de potencia activa correspondientes a todas las demás unidades conectadas a esa barra, ya sean PV o PQ.

$$\vec{P}_{G1_{slack}} = \vec{P}_{D1} - \vec{P}_{G1_{PQ,PV}} + \Re([Y_1] \cdot \vec{V_1} + [Y_{10}] \cdot \vec{V_0} + [Y_{12}] \cdot \vec{V_2}) \tag{5.41}$$

5.2. Implementación

5.2.1. Introducción a MATPOWER

De acuerdo a las ventajas del lenguaje M, expuestas en 2.3.3.7, se utiliza éste para el desarrollo del flujo de potencia trifásico. Lo que permite desarrollar una herramienta de flujo de potencia trifásico con un código simple, lo que facilita su comprensión y edición. Para esto se utiliza el proyecto MATPOWER.

MATPOWER es un proyecto de *software* libre desarrollado por el *Power Systems Engineering Research Center*(*PSERC*) [28], consistente en un paquete de funciones para MATLAB. Este paquete de funciones permite resolver flujos de potencia (*PF*) y flujos de potencia óptimos (*OPF*). La última versión estable de MATPOWER al momento de redactar este documento, y la utilizada en el desarrollo, es la versión 4.1, licenciada bajo licencia *GPL* y puede descargarse de urlhttp://www.pserc.cornell.edu/matpower/. Para utilizar MATPOWER 4.1 se requiere de MATLAB versión 6.5 o superior, o GNU Octave versión 3.2 o superior. En este desarrollo se utiliza GNU Octave 3.2.4.

Originalmente en MATPOWER para resolver un flujo de potencia o flujo de potencia óptimo se guardan los datos en una estructura de datos de MATLAB o *casedata*, denotado con la variable `mpc`. Esta estructura típicamente se define en un archivo o *casefile* que puede ser un *M-file* que retorne la estructura `mpc` o un archivo de extensión `.mat` (*MAT-file*) que defina la variable `mpc` cuando sea cargado [29]. Los campos de esta estructura son `baseMVA` que indica la potencia base en [MVA] `bus`, `gen`, codebranch que son matrices que contienen todos los datos de las barras, generadores y ramas respectivamente; y opcionalmente `gencost` y `area` que contienen los datos de costos de los generadores y las áreas en las que se divide el sistema, sólo se utilizan en el *OPF*. Además `mpc` incluye el campo `version` que contiene el número 1 o 2, indicando si corresponde a la versión 1 o 2 de los casos de MATPOWER.

Para correr un flujo de potencia se utiliza la función `runpf`, almacenada en el archivo `runpf.m`, básicamente la función recibe una estructura *mpc* o el nombre un *casefile* que la contenga, muestra un resumen de los resultados obtenidos en pantalla y retorna la estructura *mpc* actualizada con los resultados obtenidos por el algoritmo. Adicionalmente, la función puede recibir un vector con opciones de MATPOWER, el nombre de un archivo donde guardar el resumen de resultados visualizado, y el nombre de un archivo (*M-file* o *MAT-file*) en el que se escribirá la estructura retornada. Así para correr por ejemplo un flujo de potencia para el caso IEEE de 9 barras [28] incluido en la librería, con corrección de límites de reactivos, almacenar su resultado en el archivo `solvedcase9.m` y guardar el resumen de resultados en el archivo `result9.txt`, basta con escribir en al línea de comandos de Octave (después de `octave:1>`).

```
octave:1> mpc = runpf('case9', mpoption('ENFORCE_Q_LIMS', 1), 'result9.txt', 'solvedcase9');
```

Los detalles del funcionamiento de MATPOWER se encuentran en la documentación oficial incluida con la librería. Adicionalmente cualquier función o *M-file* posee una ayuda que se puede consultar anteponiendo `help`, por ejemplo para revisar los detalles de la función `mpoption` basta ingresar:

```
octave:2> help mpoption
```

5.2.2. Estructura de datos

Para resolver el flujo trifásico se extiende la estructura de datos a las tres fases, se crean algunas funciones adicionales y se crean otras a partir de funciones incluidas en MATPOWER, con las modificaciones pertinentes. A continuación se presenta un resumen con el contenido de la estructura de datos utilizada, cada columna de las matrices corresponde a un parámetro distinto, mientras que cada fila corresponde a un elemento. Si bien hay bastantes elementos que no se utilizan para el flujo de potencia trifásico, cabe destacar que ésta corresponde a una extensión de las estructuras originales de MATPOWER, por lo que estos elementos se mantienen en las estructuras para futuras extensiones a *OPF*. No obstante, no se mencionan los elementos introducidos como resultado del *OPF*.

Tabla 5.2: Columnas del campo bus de la estructura de datos. Elaboración propia.

Parámetro	Columna	Descripción
BUS_I	1	Nombre de la barra, el nombre debe ser un número
BUS_TYPE	2	Tipo de barra (1 = PQ, 2 = PV, 3 = *slack*, 4 = aislada)
PDa	3	Potencia activa consumida en la fase A en [MW]
PDb	4	Potencia activa consumida en la fase B en [MW]
PDc	5	Potencia activa consumida en la fase C en [MW]
QDa	6	Potencia reactiva consumida en la fase A en [MVAr]
QDb	7	Potencia reactiva consumida en la fase B [MVAr]
QDc	8	Potencia reactiva consumida en la fase C [MVAr]
GSa	9	Conductancia shunt en la fase A (MW consumidos con V = 1 p.u.)
GSb	10	Conductancia shunt en la fase B (MW consumidos con V = 1 p.u.)
GSc	11	Conductancia shunt en la fase C (MW consumidos con V = 1 p.u.)
BSa	12	Susceptancia shunt en la fase A (MVAr consumidos con V = 1 p.u.)
BSb	13	Susceptancia shunt en la fase B (MVAr consumidos con V = 1 p.u.)
BSc	14	Susceptancia shunt en la fase C (MVAr consumidos con V = 1 p.u.)
BUS_AREA	15	Área a la que pertenece la barra (número de 1 a 100)
VMa	16	Magnitud de voltaje en la fase A en p.u.
VMb	17	Magnitud de voltaje en la fase B en p.u.
VMc	18	Magnitud de voltaje en la fase C en p.u.
VAa	19	Ángulo de voltaje en la fase A en grados
VAb	20	Ángulo de voltaje en la fase B en grados
VAc	21	Ángulo de voltaje en la fase C en grados
BASE_KV	22	Base de voltaje en [kV]
ZONE	23	Zona de pérdidas (número de 1 a 999)
VMAX	24	Máxima magnitud de voltaje en [kV]
VMIN	25	Mínima magnitud de voltaje en [kV]

Tabla 5.3: Columnas del campo `gen` de la estructura de datos. Elaboración propia.

Parámetro	Columna	Descripción
GEN_BUS	1	Barra a la que está conectado el generador
PG1	2	Potencia activa de secuencia positiva
QG1	3	Potencia reactiva de secuencia positiva
IM0	4	Magnitud de corriente de secuencia cero
IA0	5	Ángulo de corriente de secuencia cero
IM2	6	Magnitud de corriente de secuencia negativa
IA2	7	Ángulo de corriente de secuencia negativa
QMAX	8	Máxima potencia reactiva trifásica total en [MVAr]
QMIN	9	Mínima potencia reactiva trifásica total en [MVAr]
VG	10	Magnitud de voltaje de secuencia positiva para unidades PV, en p.u.
MBASE	11	Potencia base de la máquina en [MVA], por defecto es igual a `baseMVA`
GEN_STATUS	12	Status (0 - fuera de servicio, 1 = PQ, 2 = PV, 3 = *slack*, -1 = PQ con inyecciones especificadas por fase)
PMAX	13	Máxima potencia activa trifásica total en [MW]
PMIN	14	Mínima potencia activa trifásica total en [MW]
PC1	15	Parte más baja de la curva de capacidad PQ, en [MW]
PC2	16	Parte más alta de la curva de capacidad PQ, en [MW]
QC1MIN	17	Mínima potencia reactiva en PC1, en [MVAr]
QC1MAX	18	Máxima potencia reactiva en PC1, en [MVAr]
QC2MIN	19	Mínima potencia reactiva en PC2, en [MVAr]
QC2MAX	20	Máxima potencia reactiva en PC2, en [MVAr]
RAMP_AGC	21	Pendiente para seguimiento de carga/AGC en [MW/min]
RAMP_10	22	Pendiente para reservas de 10 minutos en [MW]
RAMP_30	23	Pendiente para reservas de 30 minutos en [MW]
RAMP_Q	24	Pendiente para potencia reactiva (escala de 2 segundos) [MVAr/min]
APF	25	Factor de participación de área
RG0	26	Resistencia interna de secuencia cero
XG0	27	Reactancia interna de secuencia cero
RG2	28	Resistencia interna de secuencia negativa
XG2	29	Reactancia interna de secuencia negativa
PGa	30	Potencia activa en la fase A
PGb	31	Potencia activa en la fase B
PGc	32	Potencia activa en la fase C
QGa	33	Potencia reactiva en la fase A
QGb	34	Potencia reactiva en la fase B
QGc	35	Potencia reactiva en la fase C

Cabe mencionar que los parámetros de resistencia, reactancia y susceptancia están en p.u. con `baseMVA` como potencia base y voltaje fase-neutro (V_{fn}) como base de voltaje, de esta forma si se tienen los parámetros en p.u. referidos a voltaje fase-fase (V_{ff}) se debe realizar la transformación:

$$Z_{p.u.V_{ff}} = Z_{p.u.V_{fn}} \cdot \frac{Z_{baseV_{fn}}}{Z_{baseV_{ff}}} = Z_{p.u.V_{fn}} \cdot \frac{(V_{fn}^2/S_{base})}{(V_{ff}^2/S_{base})}$$

Cuando se corra el flujo de potencia, los resultados para las potencias generadas de todos los generadores serán actualizados tanto en los campos de secuencia positiva, como en los campos por fase. Las corrientes de secuencia negativa y cero son corrientes de control, es decir, son un dato, no un resultado del algoritmo, por lo que no se modifican, el resto del desbalance en la inyección del generador lo dan las admitancias de secuencia negativa y cero, como se explica en 5.1.3.

Tabla 5.4: Columnas del campo `branch` de la estructura de datos. Elaboración propia.

Parámetro	Columna	Descripción
F_BUS	1	Barra de origen (*"from bus"*) de la rama
T_BUS	2	Barra de destino (*"to bus"*) de la rama
BR_Ra0	3	Resistencia serie de la fase A o secuencia cero, en p.u.
BR_Rb1	4	Resistencia serie de la fase B o secuencia positiva, en p.u.
BR_Rc2	5	Resistencia serie de la fase C o secuencia negativa, en p.u.
BR_Xa0	6	Reactancia serie de la fase A o secuencia cero, en p.u.
BR_Xb1	7	Reactancia serie de la fase B o secuencia positiva, en p.u.
BR_Xc2	8	Reactancia serie de la fase C o secuencia negativa, en p.u.
BR_Ba0	9	Susceptancia *shunt* de la fase A o secuencia cero, en p.u.
BR_Bb1	10	Susceptancia *shunt* de la fase B o secuencia positiva, en p.u.
BR_Bc2	11	Susceptancia *shunt* de la fase C o secuencia negativa, en p.u.
BR_Rab	12	Resistencia serie de acople entre fases A y B, en p.u.
BR_Rbc	13	Resistencia serie de acople entre fases B y C, en p.u.
BR_Rca	14	Resistencia serie de acople entre fases C y A, en p.u.
BR_Xab	15	Reactancia serie de acople entre fases A y B, en p.u.
BR_Xbc	16	Reactancia serie de acople entre fases B y C, en p.u.
BR_Xca	17	Reactancia serie de acople entre fases C y A, en p.u.
BR_Bab	18	Susceptancia *shunt* de acople entre fases A y B, en p.u.
BR_Bbc	19	Susceptancia *shunt* de acople entre fases B y C, en p.u.
BR_Bca	20	Susceptancia *shunt* de acople entre fases C y A, en p.u.
RATE_A	21	Potencia máxima de largo plazo en [MVA]
RATE_B	22	Potencia máxima de corto plazo en [MVA]
RATE_C	23	Potencia máxima de emergencia en [MVA]
TAP	24	Derivación del transformador, o razón de vueltas fuera de la nominal
SHIFT	25	Ángulo de desfase del transformador
BR_STATUS	26	Status y conexión del transformador (0 = Fuera de servicio, 1 = Línea, 2 = YNyn, 3 = YNd, 4 = Dyn, 5 = Otros. Los valores positivos consideran los datos en fase mientras que los negativos en secuencia.)
ANGMAX	27	Máxima diferencia angular entre las barras de origen y destino
ANGMIN	28	Mínima diferencia angular entre las barras de origen y destino
PFa	29	Potencia activa inyectada en desde la barra de origen en la fase A, [MW]
PFb	30	Potencia activa inyectada en desde la barra de origen en la fase B, [MW]
PFc	31	Potencia activa inyectada en desde la barra de origen en la fase C, [MW]
QFa	32	Potencia reactiva inyectada en desde la barra de origen en la fase A, [MW]
QFb	33	Potencia reactiva inyectada en desde la barra de origen en la fase B, [MW]
QFc	34	Potencia reactiva inyectada en desde la barra de origen en la fase C, [MW]
PTa	35	Potencia activa inyectada en desde la barra de destino en la fase A, [MW]
PTb	36	Potencia activa inyectada en desde la barra de destino en la fase B, [MW]
PTc	37	Potencia activa inyectada en desde la barra de destino en la fase C, [MW]
QTa	38	Potencia reactiva inyectada en desde la barra de destino en la fase A, [MW]
QTb	39	Potencia reactiva inyectada en desde la barra de destino en la fase B, [MW]
QTc	40	Potencia reactiva inyectada en desde la barra de destino en la fase C, [MW]

Los campos `PFa - QTc` son resultado del flujo de potencia, por lo tanto no es necesario incluirlas en un caso de entrada. Finalmente cabe mencionar que dentro de las extensiones realizadas para implementar el flujo trifásico, se incorpora la posibilidad de almacenar la estructura de resultados en un archivo .csv, que contenga la información en tablas.

5.3. Algoritmo

El algoritmo global del flujo de potencia se resume en la figura 5.6. Cabe mencionar que dado que MATPOWER ofrece dos formas de corregir los límites de reactivos cuando son violados esto se mantiene en el algoritmo desarrollado, así si a la opción `ENFORCE_Q_LIMS` se le da el valor:

- 0, no se corrigen límites de reactivos.
- 1, se corrigen todos los límites de reactivos.
- 2, se corrige sólo la diferencia más grande.

En consideración a lo anterior, el algoritmo se resume de la siguiente manera. Algunos de estos pasos son realizados íntegramente por una función, la que se indica entre paréntesis:

1. Se lee el caso de estudio en una estructura de datos, en adelante `mpc`. (`loadcase`)
2. Se obtienen listas con los distintos tipos de barras, PV, PQ y SL (*slack*). (`bustypes3`)
3. Se obtienen los voltajes iniciales en componentes de fase y de secuencia (`makeVbus3`). Los voltajes iniciales son entregados en el caso de estudio en componentes de fase, estos son transformados internamente a componentes de secuencia mediante la función `phase2seq`.
4. Se construyen las matrices de admitancia $[Y_{bus}]$, $[Y_f]$ e $[Y_t]$ en componentes de secuencia, en base a lo explicado en 5.1.2. (`makeYbus3`)
5. Para facilitar la resolución de las ecuaciones matriciales se factorizan las matrices de admitancia de secuencia cero y negativa, utilizando factorización LDU. (`lu`)
6. Se inicializan los voltajes de fase y de secuencia utilizando los voltajes iniciales obtenidos del caso de estudio.

7. Calcular los vectores de inyecciones en barras (`makeSbus3`):

   - $\vec{S}_a$, $\vec{S}_b$, $\vec{S}_c$: contienen las inyecciones de potencia aparente debidas a las cargas y generadores con especificaciones en componentes de fase.
   - $\vec{S}_1$: contiene las inyecciones de potencia aparente de secuencia positiva, debidas a los generadores con especificaciones en componentes de secuencia.
   - $\vec{I}_{G0}$, $\vec{I}_{G2}$: contienen las inyecciones de corriente de secuencia negativa y cero, debidas a los generadores con especificaciones en componentes de secuencia.

8. Con estos vectores, en conjunto con las inyecciones de corriente para la compensación de líneas desbalanceadas descritas en 5.1.2.1, se calculan los vectores de corriente en componentes de secuencia: $\vec{I}_0$, $\vec{I}_1$, $\vec{I}_2$. (`getIseq`)

9. Se transforma el vector de corrientes de secuencia positiva en un vector de potencia: $\vec{S}_1 = [\vec{S}_1]^T \cdot (\vec{I}_1)^*$.

10. Se calculan los voltajes de secuencia positiva mediante una iteración del algoritmo de Newton Raphson monofásico. (`newtonpf`)

11. Se calculan los voltajes de secuencia cero mediante la resolución de la ecuación: $\vec{I}_0 = [Y_0] \cdot \vec{V}_0$.

12. Se calculan los voltajes de secuencia negativa mediante la resolución de la ecuación: $\vec{I}_2 = [Y_2] \cdot \vec{V}_2$.

13. Luego se transforman los voltajes de secuencia en voltajes de fase mediante la transformación de fortescue: $\vec{V}_{abc} = [T] \cdot \vec{V}_{012}$. (`seq2phase`)

14. Si el método de Newton Raphson utilizado para la secuencia positiva no ha convergido se vuelve al paso 8, de lo contrario se continúa:

15. Se calculan las inyecciones en las ramas y las inyecciones de los generadores.

16. Se actualizan los datos en la estructura `mpc`.

17. Si no está configurado para revisar límites de reactivos se termina el algoritmo, de lo contrario:

18. Se verifican los límites de reactivos, si no se ha excedido ningún límite el algoritmo finaliza, de lo contrario:

19. Se transforman las unidades excedidas, y sus barras, a PQ.

20. Se obtienen nuevamente las listas de barras PV, PQ y SL. (`bustypes`).

21. Se verifica si cambió la barra *slack*, si no hubo cambios se vuelve al paso 6, de lo contrario:

22. Se transforman la primera barra PV del listado, y su primer generador PV en *slack*. Luego se vuelve al paso 6.

En la figura 5.6 se observa el diagrama de flujo del algoritmo desarrollado, mientras que en la figura 5.7 se presenta el diagrama correspondiente a la iteración del método de Newton Raphson.

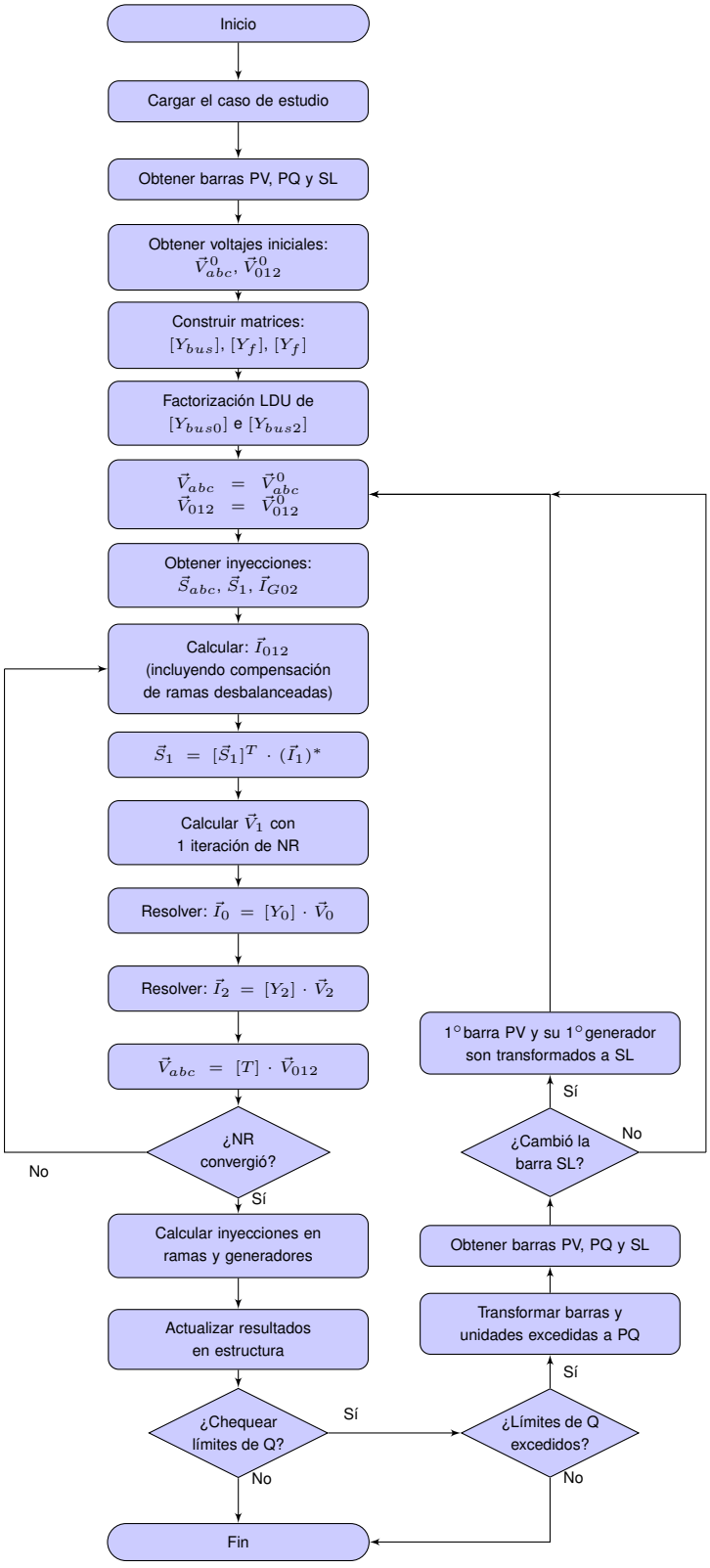

Figura 5.6: Flujo de potencia trifásico, diagrama de flujo. Elaboración propia.

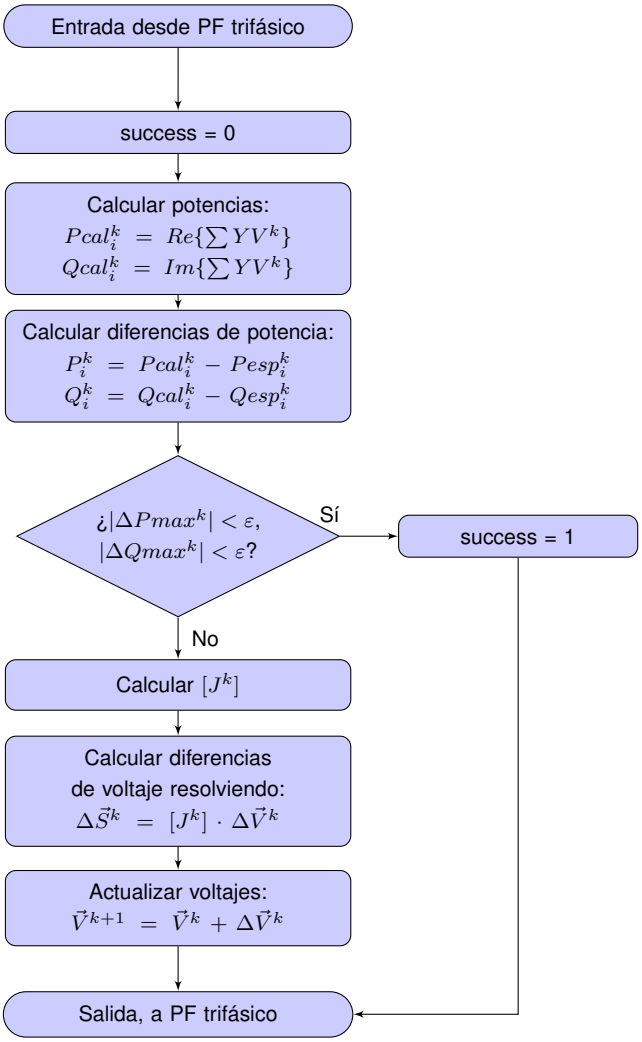

Figura 5.7: Diagrama de flujo, iteración de Newton Raphson para la secuencia positiva. Fuente: [18].

Como se puede observar, el algoritmo expuesto en la figura 5.6 está fuertemente basado en el desarrollado en [17], descrito en la figura 3.15. Teniendo como principal diferencia la incorporación de la corrección de límites de reactivos. En efecto, la parte iterativa del algoritmo presentado en dicho trabajo, es decir, exceptuando la lectura de datos e inicialización, está contenido entre los pasos 8 y 14 del algoritmo desarrollado.

5.4. Validación

La validación se separa en dos tipos de casos, balanceados y desbalanceados. Además se realiza en un computador portátil de 6GB de memoria RAM y procesador de 4 núcleos, con el sistema operativo Ubuntu 12.04 de 64 bits. El algoritmo es ejecutado en GNU Octave 3.2.4.

5.4.1. Casos balanceados

Para validar el flujo de potencia primero se comprueba su correcto funcionamiento para casos balanceados, para esto se utilizan los casos de estudio IEEE de 9 a 300 barras incluidos en MATPOWER [28]. Para esto se utiliza la función de prueba `balanced_test`, que retorna para cada columna de cada estructura la máxima diferencia entre sus elementos. De estos casos de pruebas el más grande es el IEEE de 300 barras, para correr este caso de prueba con el algoritmo trifásico se ingresa en la línea de comandos de Octave la siguiente instrucción:

```
octave:3> mpc = runpf3(loadcase3bal('case300'), mpoption('ENFORCE_Q_LIMS',1));
================        THREE-PHASE POWER FLOW  ====================

MATPOWER Version 4.1, 14-Dec-2011 -- AC Power Flow (Newton)
Three-Phase power flow converged in 5 iterations
Gen 2 at upper Q limit, converting to PQ bus
Gen 3 at upper Q limit, converting to PQ bus
Gen 22 at upper Q limit, converting to PQ bus
Gen 23 at upper Q limit, converting to PQ bus
Gen 24 at upper Q limit, converting to PQ bus
Gen 40 at upper Q limit, converting to PQ bus
Gen 48 at upper Q limit, converting to PQ bus
Gen 56 at upper Q limit, converting to PQ bus
Gen 57 at upper Q limit, converting to PQ bus
Gen 60 at upper Q limit, converting to PQ bus
Gen 65 at upper Q limit, converting to PQ bus
Bus 8 is new slack bus
Three-Phase power flow converged in 5 iterations
Gen 1 at upper Q limit, converting to PQ bus
Gen 4 at upper Q limit, converting to PQ bus
Gen 5 at upper Q limit, converting to PQ bus
Gen 13 at upper Q limit, converting to PQ bus
Gen 17 at upper Q limit, converting to PQ bus
Gen 51 at upper Q limit, converting to PQ bus
Gen 55 at upper Q limit, converting to PQ bus
Gen 58 at upper Q limit, converting to PQ bus
Gen 61 at upper Q limit, converting to PQ bus
Bus 69 is new slack bus
```

```
Three-Phase power flow converged in 5 iterations
Gen 12 at upper Q limit, converting to PQ bus
Three-Phase power flow converged in 5 iterations

Converged in 1.04 seconds
```

El resumen de los resultados obtenidos no se presenta debido a la extensión de éste, no obstante, se puede apreciar que en este caso de pruebas muchos generadores alcanzan sus límites de reactivos, por lo que son transformados a PQ y el flujo se vuelve a correr. Esto ocurre varias veces incluyendo cambios en la barra *slack*, la que es reemplazada por la primera barra PV disponible de acuerdo a lo expuesto en 5.3.

Luego para correr el test para este caso se ejecuta la siguiente instrucción, los resultados se presentan en las tablas 5.5, 5.6 y 5.7.

```
octave:4> balanced_test('case300');
```

Tabla 5.5: *Test* balanceado diferencias en el campo `bus`. Elaboración propia.

Parámetro	Columna	Máxima diferencia [p.u.]
VMa	16	$3.8858e-14$
VMb	17	$1.0281e-13$
VMc	18	$9.9920e-14$
VAa	19	$2.7018e-12$
VAb	20	$4.0643e-12$
VAc	21	$4.5333e-12$

Tabla 5.6: *Test* balanceado diferencias en el campo `gen`. Elaboración propia.

Parámetro	Columna	Máxima diferencia [p.u.]
PG1	2	$1.0829e-11$
QG1	3	$1.8339e-11$
PGa	30	$1.2562e-11$
PGb	31	$7.3783e-11$
PGc	32	$7.8103e-11$
QGa	33	$2.8820e-11$
QGb	34	$3.5811e-11$
QGc	35	$6.2926e-11$

Tabla 5.7: *Test* balanceado errores relativos en el campo `branch`. Elaboración propia.

Parámetro	Columna	Máximo error relativo (%)
PFa	29	$3.2924e-09$
PFb	30	$3.3600e-09$
PFc	31	$9.0450e-10$
QFa	32	$3.7190e-09$
QFb	33	$8.6162e-09$
QFc	34	$8.5432e-09$
PTa	35	$1.0731e-09$
PTb	36	$1.0875e-09$
PTc	37	$3.8475e-10$
QTa	38	$1.7858e-08$
QTb	39	$4.9095e-08$
QTc	40	$4.0773e-08$

Para las potencias se presentan los máximos errores relativos obtenidos, mientras que para los voltajes se utilizan simplemente las máximas diferencias en p.u. ya que eso permite dimensionar de mejor manera el impacto de estos errores en una red física. Sin embargo, como algunas potencias inyectadas por los generadores son nulas, los errores relativos de éstas se indefinen, por lo que para los generadores se presentan las máximas diferencias, en vez de los máximos errores relativos.

Como se puede observar, las diferencias en los voltajes son menores que $10^{-11}[p.u.]$, los errores relativos en las inyecciones en las ramas son inferiores al $10^{-7}[\,\%]$, y las diferencias en las potencias generadas son inferiores a $10^{-10}[p.u.]$. De esta forma, el algoritmo implementado logra muy buenos resultados para el caso de estudio, lo que valida su desarrollo para casos balanceados.

Adicionalmente, debido a que debe corregir límites de reactivos en 3 oportunidades, el algoritmo converge 4 veces en 5 iteraciones cada una, para una tolerancia de $10^{-8}[p.u]$, con un tiempo total de ejecución de 1.04 segundos. Lo que, considerando el tamaño del sistema y la cantidad de correcciones de límites de reactivos que realiza, es un tiempo bastante bajo.

5.4.2. Caso desbalanceado

Para realizar la validación de casos desbalanceados se utiliza el caso de pruebas utilizado en [25]. El que consiste en un sistema de 5 barras, 2 generadores, 2 transformadores y 3 líneas desbalanceadas y mal transpuestas, de las cuáles 1 es un doble circuito. El sistema se puede apreciar en la figura 5.8 mientras que sus parámetros se pueden ver de la tabla 5.9 a la 5.13. Todos los valores están en p.u. con una base de 33.33[MVA], salvo las potencias, que se encuentran en [p.u]

5.4.2.1. Caso de estudio

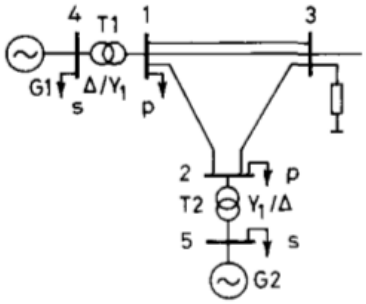

Figura 5.8: Diagrama unilineal de la red. Fuente: [25].

Tabla 5.8: Matriz de susceptancia *shunt* en la barra 3. Fuente: [25].

	a	*b*	*c*
a	*j*2.25	*j*0.0	*j*0.0
b	*j*0.0	*j*2.25	*j*0.0
c	*j*0.0	*j*0.0	*j*2.25

Tabla 5.9: Datos de los generadores. Fuente: [25].

	Sequence reactances			Power P_g^{abc}	Voltage V^+
	X^0	X^+	X^-		
G1	0.02	0.001	0.004	700.0	1.050
G2	0.02	0.001	0.004	slack	1.045

Tabla 5.10: Datos de los transformadores. Fuente: [25].

	Leakage impedance	Taps		Connection	
		p	s	p	s
T1	0.0016 + *j*0.015	0.025	0.01	Y_1/Δ	
T2	0.0016 + *j*0.015	0.025	0.00	Y_1/Δ	

Tabla 5.11: Matrices de impedancia serie y admitancia *shunt* de líneas 1-2 y 2-3. Fuente: [25].

	a	*b*	*c*		*a*	*b*	*c*
a	0.0066 + *j*0.056	0.0017 + *j*0.027	0.0012 + *j*0.021	*a*	0.0 + *j*0.15	0.0 − *j*0.03	0.0 − *j*0.01
b		0.0045 + *j*0.047	0.0014 + *j*0.022	*b*		0.0 + *j*0.25	0.0 − *j*0.02
c			0.0062 + *j*0.061	*c*			0.0 + *j*0.14

Tabla 5.12: Matriz de impedancia serie línea de doble circuito 1-3. Fuente: [25].

	a	*b*	*c*	*a*	*b*	*c*
a	0.008 + *j*0.055	0.004 + *j*0.016	0.004 + *j*0.011	0.002 + *j*0.013	0.002 + *j*0.011	0.001 + *j*0.02
b		0.007 + *j*0.054	0.004 + *j*0.016	0.002 + *j*0.013	0.002 + *j*0.014	0.002 + *j*0.02
c			0.008 + *j*0.055	0.001 + *j*0.013	0.002 + *j*0.014	0.002 + *j*0.02
a				0.008 + *j*0.054	0.004 + *j*0.016	0.003 + *j*0.02
b					0.007 + *j*0.054	0.004 + *j*0.02
c						0.008 + *j*0.055

Tabla 5.13: Matriz de admitancia *shunt* línea de doble circuito 1-3. Fuente: [25].

	a	*b*	*c*	*a*	*b*	*c*
a	0.0 + *j*0.1484	0.0 − *j*0.031	0.0 − *j*0.025	0.0 − *j*0.021	0.0 − *j*0.027	0.0 − *j*0.020
b		0.0 + *j*0.1495	0.0 − *j*0.030	0.0 − *j*0.015	0.0 − *j*0.018	0.0 − *j*0.016
c			0.0 + *j*0.1150	0.0 − *j*0.017	0.0 − *j*0.016	0.0 − *j*0.014
a				0.0 + *j*0.1488	0.0 − *j*0.031	0.0 − *j*0.025
b					0.0 + *j*0.1495	0.0 − *j*0.031
c						0.0 + *j*0.150

Para transformar las matrices de la línea de doble circuito en una matriz de admitancia simple de (3×3) se analiza el sistema:

$$\begin{bmatrix} I_a \\ I_b \\ I_c \\ I'_a \\ I'_b \\ I'_c \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} y_{aa} & y_{ab} & y_{ac} \\ y_{ba} & y_{bb} & y_{bc} \\ y_{ca} & y_{cb} & y_{cc} \end{bmatrix} & \begin{bmatrix} y_{aa'} & y_{ab'} & y_{ac'} \\ y_{ba'} & y_{bb'} & y_{bc'} \\ y_{ca'} & y_{cb'} & y_{cc'} \end{bmatrix} \\ \begin{bmatrix} y_{a'a} & y_{a'b} & y_{a'c} \\ y_{b'a} & y_{b'b} & y_{b'c} \\ y_{c'a} & y_{c'b} & y_{c'c} \end{bmatrix} & \begin{bmatrix} y_{a'a'} & y_{a'b'} & y_{a'c'} \\ y_{b'a'} & y_{b'b'} & y_{b'c'} \\ y_{c'a'} & y_{c'b'} & y_{c'c'} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} V_a \\ V_b \\ V_c \\ V'_a \\ V'_b \\ V'_c \end{bmatrix} \tag{5.42}$$

Por ejemplo para la fase A, las corrientes I_a e $I_{a'}$ serían:

$$I_a = y_{aa}V_a + y_{ab}V_b + y_{ac}V_c + y_{aa'}V'_a + y_{ab'}V'_b + y_{ac'}V'_c \tag{5.43}$$

$$I'_a = y_{a'a}V_a + y_{a'b}V_b + y_{a'c}V_c + y_{a'a'}V'_a + y_{a'b'}V'_b + y_{a'c'}V'_c \tag{5.44}$$

Pero como $V_a = V'_a$ se tiene:

$$I_a = (y_{aa} + y_{aa'})V_a + (y_{ab} + y_{ab'})V_b + (y_{ac} + y_{ac'})V_c \tag{5.45}$$

$$I'_a = (y_{a'a} + y_{a'a'})V_a + (y_{a'b} + y_{a'b'})V_b + (y_{a'c} + y_{a'c'})V_c \tag{5.46}$$

Y como la corriente total en la fase A corresponde a $\hat{I}_a = I_a + I_{a'}$, se tiene:

$$\begin{aligned}\hat{I_a} = I_a + I_{a'} &= (y_{aa} + y_{aa'} + y_{a'a} + y_{a'a'})V_a \\ &+(y_{ab} + y_{ab'} + y_{a'b} + y_{a'b'})V_b \\ &+(y_{ac} + y_{ac'} + y_{a'c} + y_{a'c'})V_c \\ &= \hat{y_{aa}}V_a + \hat{y_{ab}}V_b + \hat{y_{ac}}V_c \end{aligned} \tag{5.47}$$

Donde

$$\hat{y_{aa}} = y_{aa} + y_{aa'} + y_{a'a} + y_{a'a'}$$
$$\hat{y_{ab}} = y_{ab} + y_{ab'} + y_{a'b} + y_{a'b'}$$
$$\hat{y_{ac}} = y_{ac} + y_{ac'} + y_{a'c} + y_{a'c'}$$

Por lo tanto, extendiendo esta deducción a las demás fases, se puede deducir que si se suman las 4 submatrices de la ecuación 5.42 se obtiene una matriz de admitancia equivalente $[Y_{eq}]$ que cumple la ecuación $\vec{I} = [Y]\vec{V}$. Así, denominando con los subíndices L y L' a ambos circuitos de la línea y obteniendo la matriz de admitancia serie $[Y^z] = [Z]^{-1}$, las matrices de admitancia *shunt* e impedancia serie equivalentes son:

$$[Y_{eq}] = [Y^s_{LL}] + [Y^s_{LL'}] + [Y^s_{L'L}] + [Y^s_{L'L'}] \tag{5.48}$$
$$[Z_{eq}] = ([Y^z_{LL}]^{-1} + [Y^z_{LL'}]^{-1} + [Y^z_{L'L}]^{-1} + [Y^z_{L'L'}]^{-1})^{-1} \tag{5.49}$$

5.4.2.2. Resultados

Los resultados obtenidos por el algoritmo desarrollado se comparan con los presentados en [25], esta comparación se muestra en las tablas 5.14 a 5.18.

Tabla 5.14: Resultados caso desbalanceado, voltajes de barra. Elaboración propia.

	Resultados obtenidos en [25]						**Resultados obtenidos por el algoritmo**					
	Fase A		Fase B		Fase C		Fase A		Fase B		Fase C	
	V[p.u.]	ang[°]	V[p.u.]	ang[°]	V[p.u.]	ang[°]	V[p.u.]	ang[°]	V[p.u.]	ang[°]	V[p.u.]	ang[°]
1	1,0478	30.75	1.0531	-89.58	1.0433	150.98	1.0480	30.848	1.0533	-89.479	1.0429	150.989
2	1,0494	28.42	1,0529	-91.67	1.0479	148.41	1.0495	28.527	1.0530	-91.572	1.0476	148.513
3	1,0308	26.17	1,0466	-93.94	1.0263	146.27	1.0311	26.265	1.0469	-93.822	1.0253	146.368
4	1.0496	6.08	1,0508	-113.96	1.0495	126.00	1.0496	6.186	1.0509	-113.861	1.0495	126.103
5	1.0447	-0.10	1.0453	-120.10	1.0450	119.87	1.0447	0.000	1.0453	-120.001	1.0450	119.967

Tabla 5.15: Caso desbalanceado, diferencias en los voltajes de barra. Elaboración propia.

	Diferencias en p.u.					
	Fase A		Fase B		Fase C	
	V[p.u.]	ang[°]	V[p.u.]	ang[°]	V[p.u.]	ang[°]
1	-0.0002	-0.098	-0.0002	-0.101	0.0004	-0.009
2	-1e-04	-0.107	-1e-04	-0.098	0.0003	-0.103
3	-0.0003	-0.095	-0.0003	-0.118	0.001	-0.098
4	0	-0.106	-1e-04	-0.099	0	-0.103
5	0	-0.1	0	-0.099	0	-0.097

Tabla 5.16: Resultados caso desbalanceado, potencia generada. Elaboración propia.

#	**Barra**	**Tipo**	**Resultados [25]**		**Resultados algoritmo**		**Errores relativos**	
			P[MW]	Q[MVAr]	P[MW]	Q[MVAr]	P[%]	Q[%]
1	4	PV	700.00	92.93	700.000	94.261	0	-1.4323
2	5	*slack*	215.03	133.15	215.036	133.397	-0.0028	-0.1855

Tabla 5.17: Resultados caso desbalanceado, inyecciones en ramas. Elaboración propia.

			Resultados obtenidos en [25]				Resultados obtenidos por el algoritmo			
			Origen		Destino		Origen		Destino	
			P[MW]	Q[MVar]	P[MW]	Q[MVar]	P[MW]	Q[MVar]	P[MW]	Q[MVar]
1	2	a	47.388	-20.447	-47.356	16.159	47.428	-20.394	-47.395	16.104
		b	58.442	-6.409	-58.026	-1.624	58.412	-6.350	-57.995	-1.685
		c	39.093	-7.219	-38.716	3.274	39.072	-7.310	-38.697	3.370
1	3	a	152.953	17.180	-152.414	-17.364	152.952	17.211	-152.438	-17.372
		b	151.631	-10.826	-149.595	8.992	151.630	-10.835	-149.613	8.961
		c	153.742	12.351	-152.033	-12.936	153.751	13.606	-151.997	-12.724
1	4	a	-220.342	-6.734	216.568	29.233	-220.380	-6.817	216.258	29.870
		b	-230.075	7.234	224.163	18.651	-230.042	7.185	224.161	18.741
		c	-212.835	-15.134	229.273	30.044	-212.823	-16.297	229.582	30.649
2	3	a	47.769	-1.305	-47.585	-2.944	47.737	-1.346	-47.561	-2.897
		b	60.569	9.231	-60.405	-16.852	60.549	9.134	-60.388	-16.777
		c	38.473	14.275	-37.967	-18.069	38.521	14.671	-38.002	-18.435
2	5	a	-65.916	-40.355	65.039	44.111	-65.842	-40.259	64.855	44.359
		b	-68.045	-33.110	65.135	38.649	-68.054	-32.948	65.085	38.491
		c	-65.258	-43.050	69.860	41.393	-65.325	-43.541	70.096	41.548
3*		a	0	-79.691	-	-	0	-79.730	-	-
		b	0	-82.139	-	-	0	-82.184	-	-
		c	0	-78.995	-	-	0	-78.840	-	-

Tabla 5.18: Errores relativos caso desbalanceado, inyecciones en ramas. Elaboración propia.

			Errores relativos (%)			
			Origen		Destino	
			P[%]	Q[%]	P[%]	Q[%]
1	2	a	-0.0844	0.2592	-0.0824	0.3404
		b	0.0513	0.9206	0.0534	-3.7562
		c	0.0537	-1.2606	0.0491	-2.9322
1	3	a	0.0007	-0.1804	-0.0157	-0.0461
		b	0.0007	-0.0831	-0.0120	0.3448
		c	-0.0059	-10.1611	0.0237	1.6388
1	4	a	-0.0172	-1.2326	0.1431	-2.1790
		b	0.0143	0.6774	0.0009	-0.4825
		c	0.0056	-7.6847	-0.1348	-2.0137
2	3	a	0.0670	-3.1418	0.0504	1.5965
		b	0.0330	1.0508	0.0281	0.4451
		c	-0.1248	-2.7741	-0.0922	-2.0256
2	5	a	0.1123	0.2379	0.2829	-0.5622
		b	-0.0132	0.4893	0.0768	0.4088
		c	-0.1027	-1.1405	-0.3378	-0.3745
3*		a	0.0000	-0.0005	-	-
		b	0.0000	-0.0005	-	-
		c	0.0000	0.0020	-	-

Como se puede observar en las tablas 5.14 y 5.15, las magnitudes de voltajes presentan diferencias bastante bajas, con un máximo de 0.0004 [p.u.]„ es decir, de un 0.04 %. Mientras que en los ángulos se aprecian diferencias mayores, debido a que el algoritmo desarrollado, una vez que converge el flujo, reordena los ángulos tomando como referencia el ángulo de la fase “A” de la barra *slack*. Lo que se evidencia al notar que los errores de los ángulos son todos similares, de aproximadamente unos -0.1°.

De la tabla 5.16 se observa que las potencias activas presentan un error considerablemente menor que las potencias reactivas. En el caso del generador 2 el error es de un 0.0028 %, mientras que para el generador 1 el error es cero, lo que es bastante lógico pues se trata de un generador PV, y por consiguiente su potencia activa trifásica es un dato de entrada al algoritmo. En el caso de las potencias activas los errores son de 1.4323 y 0.1855 % para los generadores 1 y 2 respectivamente.

Por otro lado, en las inyecciones en las ramas es posible ver diferencias mayores. El peor caso corresponde a la inyección de potencia reactiva en la fase “C” de la línea de doble circuito 1-3, desde la barra 1, con un error de 10.16 %. Sin embargo, si se consideran las potencias aparentes el error es bastante menor, en efecto, denominando S_1 y S_2 a las potencias aparentes correspondientes los resultados del caso de estudio y del algoritmo respectivamente, se tiene que el error sería:

$$S_1 = (154.2373 \angle 4.593^\circ)$$
$$S_2 = (154.3519 \angle 5.0575^\circ)$$
$$\varepsilon = \frac{S_1 - S_2}{S_1} \cdot 100 = 0.811\,\%$$

Por lo tanto, considerando los resultados obtenidos se puede afirmar que el algoritmo presenta un buen desempeño para el caso de estudio internacional, lo que valida su funcionamiento para casos desbalanceados. Obteniéndose un error de 0.0005 [p.u.] para las magnitudes de los voltajes de fase y un error relativo de 1.5 % aproximadamente para las inyecciones de potencia. Los resultados fueron obtenidos en 9 iteraciones con una tolerancia de $10^{-10}[p.u]$ para los errores de potencia, con un tiempo de ejecución de 0.07 segundos.

Capítulo 6

Implementación computacional de la herramienta

6.1. Lenguaje de programación, entorno de desarrollo y clasificación de archivos

De acuerdo a lo evaluado en 2.3.3, se escoge el lenguaje Java como el lenguaje principal de la herramienta, debido al buen balance entre velocidad, uso de recursos y simplicidad, y además la portabilidad. Para simplificar la programación y desarrollo de la herramienta se utiliza el entorno de desarrollo integrado ("Integrated Development Environment", IDE) *DeepEdit* que permite el diseño de interfaces gráficas mediante la inserción de componentes con el *mouse*. En el anexo C se presenta más información de Netbeans y su utilización para la implementación de la herramienta.

6.2. Interfaz gráfica

6.2.1. Editor de microrred

En la figura 6.1 se muestra la interfaz gráfica desarrollada para la edición de las microrredes.

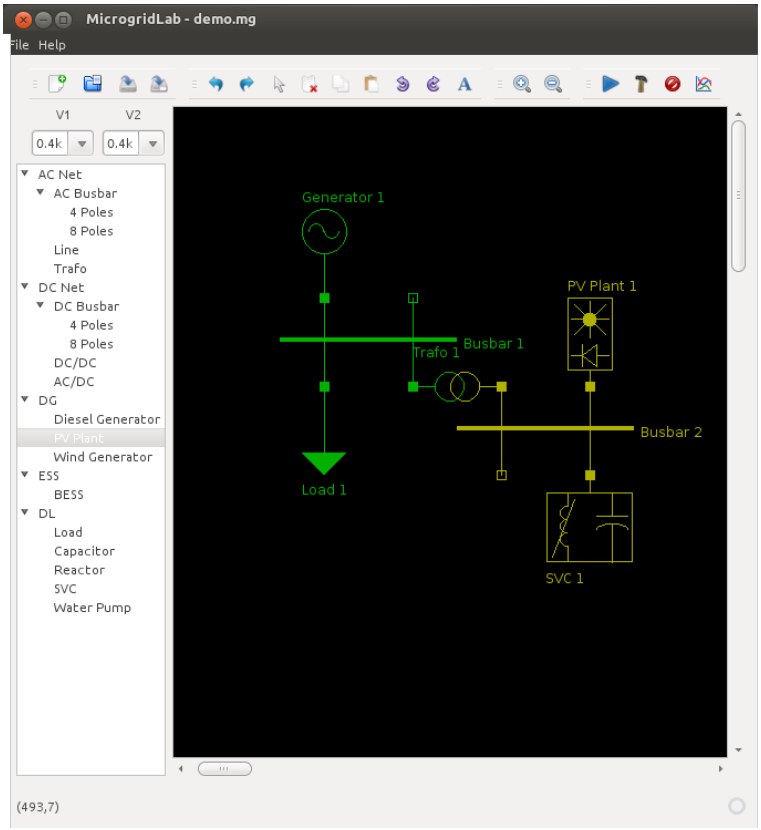

Figura 6.1: Interfaz gráfica, Editor de microrred. Elaboración propia.

En el panel superior se encuentran las herramientas disponibles, las que permiten abrir, editar y guardar cambios de una microrred, aplicar distintos niveles de *zoom* o acercamiento visual, configurar y ejecutar las simulaciones, entre otros. Todas estas funcionalidades poseen un comando asociado vía teclado, lo que facilita la manipulación de la interfaz, los que pueden observarse al posar el cursor sobre el botón correspondiente durante un corto periodo de tiempo. Los detalles de estas herramientas se encuentran en el anexo D.

6.2.1.1. Almacenamiento de la microrred en archivos

El almacenamiento de la microrred en un archivo se realiza mediante la librería XStream, que posee métodos simples y efectivos para guardar un objeto en formato `.xml`, y para obtener un objeto de un archivo `.xml`. De esta forma la microrred es almacenada en formato `XML` pero en un archivo de extensión customizada `.mgl`. De acuerdo a lo señalado en 4.5, todos los datos de la microrred se almacenan y se trabajan desde el objeto `Microgrid`, luego, para guardar la red, se guarda este objeto en un archivo `.mgl`.

La ventaja de almacenar los objetos en XML es que este formato es independiente del lenguaje, lo que permite cargar los objetos en otro programa escrito en otro lenguaje.

6.2.1.2. Edición de los componentes de la microrred

Para editar los parámetros de un componente basta presionar con el botón derecho del ratón sobre éste. Todos los componentes tienen una ventana con varias pestañas para editar distintos tipos de parámetros. Particularmente, todas estas ventanas poseen la misma pestaña *"Economic Properties"* donde se configuran sus parámetros económicos, los cuales se describen en 4.2. El detalle de estas ventanas de edición de parámetros se encuentra en el anexo D.

Por otra parte, es posible notar que las uniones de los componentes se realizan mediante unos pequeños cuadrados, estos cuadrados corresponden a los polos de conexión y si se presiona con el ratón sobre ellos pueden abrirse y cerrarse, lo que eléctricamente equivale a abrir o cerrar un interruptor, dejando la unidad fuera de servicio. En el caso de una rama (transformador, línea o DC/DC) es importante abrir ambos extremos para dejar la unidad fuera de servicio, de lo contrario puede producirse un error en la simulación.

6.2.1.3. Configuración de la simulación

En la ventana de configuración de simulación existen tres opciones o modos de simulación:

- ***Power flow*:** corre un flujo de potencia simple.
- ***Dispatch*:** realiza un despacho económico de la red para un horizonte de simulación determinado.
- ***Dispatch + Power flow*:** realiza un despacho económico de la red para un horizonte de simulación determinado y con esos resultados corre un flujo de potencia para cada periodo.

Cada uno de estos modos posee una pestaña de configuración propia. El recurso solar y la curva de variación diaria de carga se realizan en la pestaña de despacho. El detalle de esta configuración se encuentra en el anexo D.

6.2.2. Visualización de resultados

Para la visualización de resultados se utiliza la librería *JFreeChart* que permite insertar paneles gráficos y manejar sus datos de manera bastante simple, además ofrece funcionalidades como acercamiento o *zoom*, almacenar los gráficos como imágenes, imprimir los gráficos, entre otros.

A continuación se muestra la ventana de visualización de resultados, en ella es posible seleccionar los datos a graficar mediante los *check box* del lado derecho. Las unidades para las que se muestran los datos indicados en los *check box* corresponden a las unidades seleccionadas en el editor, para seleccionar múltiples unidades se puede arrastrar un rectángulo presionando el botón izquierdo del ratón o seleccionando individualmente mientras se mantiene presionada la tecla `Ctrl`. El gráfico se actualiza al modificar el estado de alguno de los *check box*, al presionar el botón dedicado para eso, o al apretar la tecla F5.

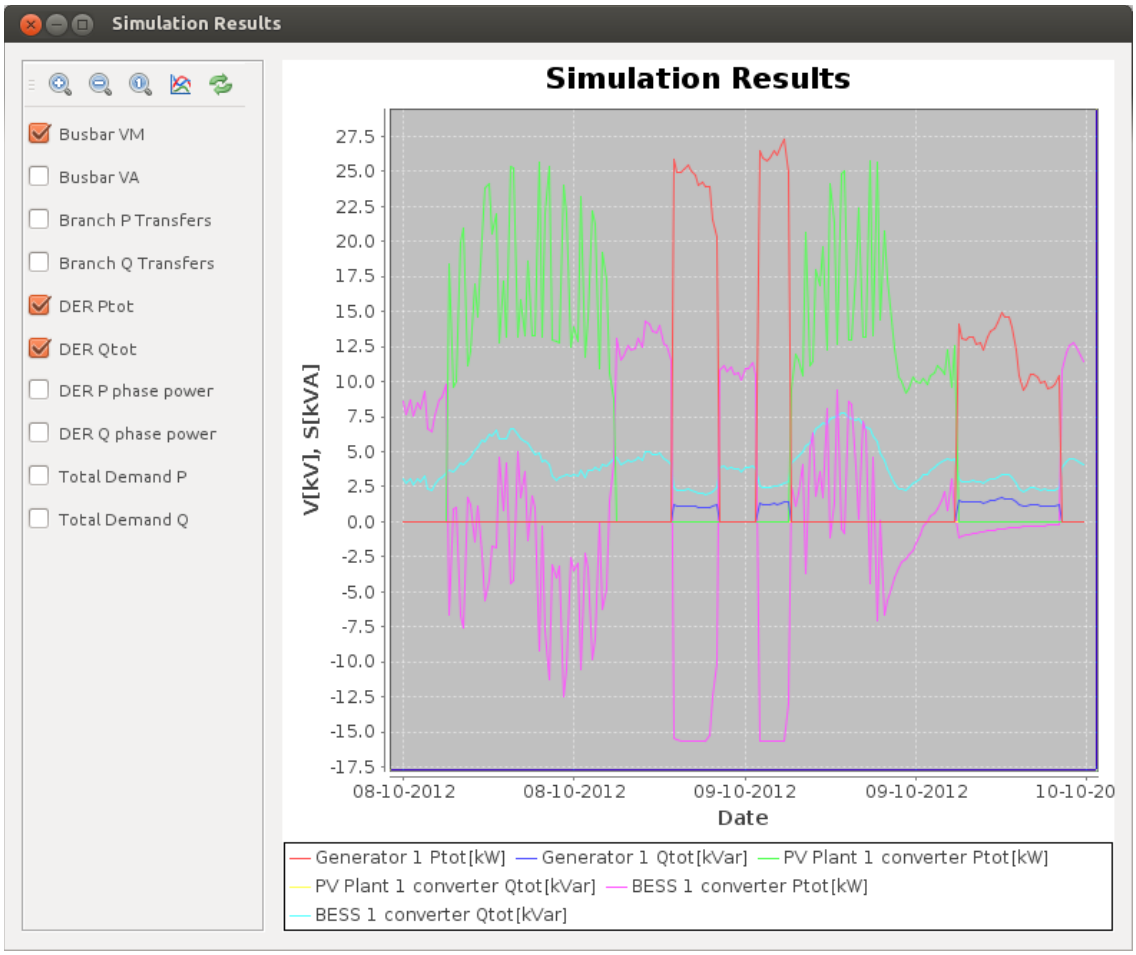

Figura 6.2: Interfaz gráfica, Visualización de resultados. Elaboración propia.

6.3. Módulo de flujo de potencia

Para ejecutar el flujo de potencia desarrollado en el capítulo 5 se escriben dos *script*, uno para sistemas Unix y otro para sistemas *Windows*, llamados `runpf3` y `runpf3.bat`, respectivamente. Estos *script* ejecutan *Octave* y dentro de éste ejecuta la función `runpf3.m` que corre el flujo de potencia. Además Reciben como parámetros la ruta del archivo de entrada al flujo de potencia y los parámetros `ENFORCE_Q_LIMS`, `PF_TOL` y `PF_MAX_IT`, descritos en 5.

Adicionalmente, los *script*, dentro de los parámetros que entregan a la función `runpf3.m`, entregan el archivo de salida donde debe almacenarse el caso final resuelto. El archivo entregado se denomina `save0.csv`, y por tratarse de un archivo de extensión `.csv`, los datos son almacenados como tablas, lo que facilita su posterior lectura por parte de la interfaz principal.

Cuando se ejecuta el flujo de potencia, el programa principal escribe un caso de estudio con la entrada para el flujo de potencia, de acuerdo a lo explicado en 5.2.2, en un archivo `mgcase0.m`. Luego, dependiendo del sistema operativo utilizado, el programa principal crea un proceso paralelo en el que ejecuta el *script* correspondiente. Estos *scripts* redireccionan la salida de Octave a un archivo `result0.txt`, el cual luego es leído por el programa principal y mostrado visualmente al usuario mediante una ventana dedicada para este propósito. Finalmente el programa principal lee los datos almacenados en el archivo de salida `save0.cs` y carga los datos leídos en los objetos.

Cabe mencionar que los archivos corresponden al caso de un flujo de potencia individual. Luego, si se ejecuta una simulación múltiple, es decir, para varios periodos de simulación, estos archivos tendrán los mismos nombres, pero en vez de contener el número “0” contendrán el número del periodo de simulación correspondiente.

6.4. Módulo de despacho de unidades

El módulo de despacho de unidades a incorporar corresponde a una herramienta de predespacho de unidades desarrollada en [30]. Esta herramienta permite incorporar diversas unidades de generación con distintas características y costos, y entrega como resultado un despacho de mínimo costo para un horizonte de simulación determinado. Para su funcionamiento requiere de una base de datos que incorpora toda la información del sistema, las unidades de generación y sus parámetros, condiciones iniciales, además de la demanda y otros datos por período.

Este trabajo ha sido extendido en [5] a una herramienta de gestión activa de una microrred, que incorpora un predespacho y despacho de unidades con horizonte deslizante. Es decir, realiza un despacho para periodos de 15 minutos durante 2 días, el que es corregido cada 15 minutos incorporando los datos de la operación real de la microrred. En este trabajo se configura manualmente la base de datos correspondiente a la microrred, luego sólo se modifican los datos que dependen de la operación, y que a su vez constituyen los datos de entrada, éstos son:

6.4.1. Datos de entrada

- **Condiciones iniciales:** corresponden a los volúmenes iniciales de los estanques de agua y diesel, la energía inicial de los bancos de batería, y el estado encendido o apagado inicial de los generadores diesel.
- **Restricciones adicionales:** tales como energía mínima y máxima de los bancos de batería, potencias mínimas y máximas en general, volúmenes mínimos y máximos de los estanques de diesel y agua, volúmenes mínimos finales de estos estanques, entre otros.
- **Predicción de disponibilidad de recursos:** predicción de recurso solar y eólico.
- **Costos de operación:** principalmente, asociados a las unidades diesel.
- **Predicciones de demanda:** predicción de demanda eléctrica y de agua.

6.4.2. Datos de salida

Los datos de salida se almacenan en un archivo de salida `solutionUC.csv`, y corresponden a una tabla con las consignas de potencia para cada unidad, entre otros datos, donde las unidades se identifican mediante un nombre. Para poder incorporar correctamente estos datos, estos nombres deben coincidir con los `key` definidos en los objetos `Device` del programa principal. La lectura de este archivo se realiza mediante una librería de Java llamada OpenCSV, que facilita la lectura y escritura de archivos `.csv`.

La escritura en la base de datos y la ejecución de la herramienta de despacho quedan fuera de los alcances de este trabajo, quedando propuesta su implementación como trabajo futuro. Sin embargo, la lectura de los archivos de salida del despacho es incorporada. Esto se traduce en que, desde la ventana de configuración de la simulación, es posible seleccionar directamente un archivo de despacho o ejecutar un nuevo despacho.

En caso de seleccionar un archivo, los potencias de los consumos ingresados por el usuario en la microrred son escalados a la demanda total por periodo, indicada en el archivo de despacho, con el fin de mantener el balance de potencia en la operación. Cabe mencionar que esta funcionalidad permite no sólo incorporar el despacho de unidades al flujo de potencia trifásico, sino además utilizar datos de operación real, dependiendo del contenido del archivo de despacho seleccionado.

Capítulo 7

Utilización y visualización de un caso de estudio

7.1. Descripción del caso de estudio

A continuación se analiza un caso de estudio basado en una microrred real, correspondiente a la microrred de Huatacondo desarrollada por Centro de Energía de la Universidad de Chile. Para esto se utiliza un modelo simplificado y balanceado de esta red, obtenido en [31]. Este modelo contempla dos plantas fotovoltaicas, un generador eólico, un generador diesel y un banco de baterías. En la figura 7.1 y la tabla 7.1 se puede observar el diagrama unilineal de la red, y su matriz de admitancia en p.u., con una potencia base de 25[kVA].

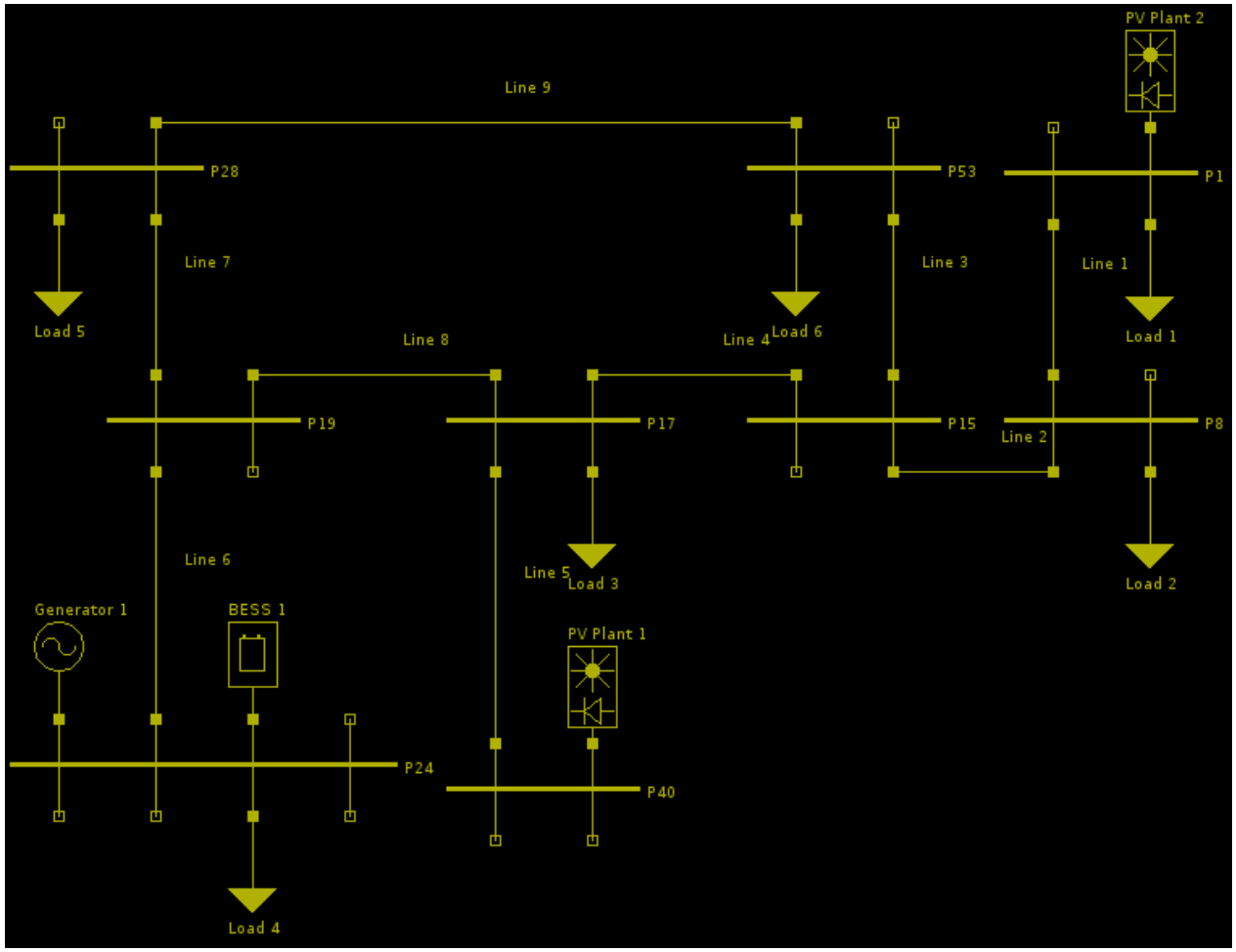

Figura 7.1: Red Huatacondina simplificada, diagrama unilineal. Elaboración propia.

Tabla 7.1: Red Huatacondina simplificada, matriz de admitancia. Fuente: [31].

		P1	P8	P15	P17	P19	P24	P28	P40	P53
		1	2	3	4	5	6	7	8	9
P1	1	21.61-2.11j	-21.61+2.11j	0	0	0	0	0	0	0
P8	2	-21.61+2.11j	394.49-48.72j	-372.88+46.61j	0	0	0	0	0	0
P15	3	0	-372.88+46.61j	755.2-89.03j	-200.1+25.01j	0	0	0	0	-182.22+17.41j
P17	4	0	0	-200.1+25.01j	390.03-47.78j	-156.96+19.62j	0	0	-32.97+3.15j	0
P19	5	0	0	0	-156.96+19.62j	370.65-54.73j	-129.74+25.58j	-83.95+9.53j	0	0
P24	6	0	0	0	0	-129.74+25.58j	129.74-25.58j	0	0	0
P28	7	0	0	0	0	-83.95+9.53j	0	119.44-12.92j	0	-35.49+3.39j
P40	8	0	0	0	-32.97+3.15j	0	0	0	32.97-3.15j	0
P53	9	0	0	-182.22+17.41j	0	0	0	-35.49+3.39j	0	217.71-20.8j

Al observar esta matriz de admitancia se puede notar que, los elementos de la diagonal corresponden exactamente al negativo de la suma de los elementos fuera de la diagonal, lo que quiere decir que no se considera la susceptancia *shunt* de las líneas. De esta forma, a partir de la matriz de admitancia, se obtienen los siguientes parámetros para las líneas balanceadas:

Tabla 7.2: Red Huatacondina simplificada, parámetros de líneas balanceadas. Elaboración propia.

Línea	Origen	Destino	R [p.u.]	X [p.u.]
L12	P1	P8	0.0458379	0.0044756
L23	P8	P15	0.002641	0.0003301
L34	P15	P17	0.004921	0.000615
L39	P15	P53	0.005438	0.0005196
L45	P17	P19	0.006273	0.0007841
L48	P17	P40	0.03006	0.002872
L56	P19	P24	0.007419	0.001463
L57	P19	P28	0.01176	0.001335
L79	P28	P53	0.02792	0.002667

Para introducir un desbalance en la línea se desbalancean de manera intencional estos parámetros, los que se resumen a continuación.

Tabla 7.3: Red Huatacondina simplificada, parámetros de líneas des balanceadas. Elaboración propia.

			Fase A		Fase B		Fase C	
Línea	Origen	Destino	R [p.u.]	X [p.u.]	R [p.u.]	X [p.u.]	R [p.u.]	X [p.u.]
L12	P1	P8	0.047	0.0044	0.043	0.0045	0.045	0.0047
L23	P8	P15	0.0026	0.00033	0.0028	0.00035	0.0023	0.00029
L34	P15	P17	0.0049	0.00062	0.005	0.00061	0.0047	0.00064
L39	P15	P53	0.0054	0.00052	0.0051	0.00049	0.0057	0.0005
L45	P17	P19	0.006	0.00078	0.0064	0.00077	0.0062	0.0008
L48	P17	P40	0.02	0.0029	0.04	0.0028	0.03	0.00291
L56	P19	P24	0.00742	0.001463	0.00741	0.001453	0.0075	0.001503
L57	P19	P28	0.012	0.001336	0.014	0.001339	0.013	0.001335
L79	P28	P53	0.0279	0.0026	0.02799	0.0027	0.027	0.0025

Adicionalmente, en base a los flujos medidos y estimados en [31] se estiman los consumos en las barras:

Tabla 7.4: Red Huatacondina simplificada, cargas balanceadas por barra. Elaboración propia.

		Fase A		Fase B		Fase C		Total	
Barra		P [kW]	Q [kVAr]	P [kW]	Q [kVAr]	P [kW]	Q [kVAr]	P [kW]	Q [kVAr]
P1	1	0	0	0	0	0	0	0	0
P8	2	0.297667	0.234	0.297667	0.234	0.297667	0.234	0.893	0.702
P15	3	0	0	0	0	0	0	0	0
P17	4	1.773	0.154	1.773	0.154	1.773	0.154	5.319	0.462
P19	5	0	0	0	0	0	0	0	0
P24	6	0.107667	0.164	0.107667	0.164	0.107667	0.164	0.323	0.492
P28	7	0.630667	0.308	0.630667	0.308	0.630667	0.308	1.892	0.924
P40	8	0	0	0	0	0	0	0	0
P53	9	-0.17567	0.154	-0.17567	0.154	-0.17567	0.154	-0.527	0.462

Para apreciar el efecto del desbalance de cargas, éstas se desbalancean ligeramente, manteniendo la potencia trifásica total:

Tabla 7.5: Red Huatacondina simplificada, cargas desbalanceadas por barra. Elaboración propia.

		Fase A		Fase B		Fase C		Total	
Barra		P [kW]	Q [kVAr]	P [kW]	Q [kVAr]	P [kW]	Q [kVAr]	P [kW]	Q [kVAr]
P1	1	0	0	0	0	0	0	0	0
P8	2	0.15	0.2	0.2	0.1	0.543	0.402	0.893	0.702
P15	3	0	0	0	0	0	0	0	0
P17	4	1.9	0.3	2	0.009	1.419	0.153	5.319	0.462
P19	5	0	0	0	0	0	0	0	0
P24	6	0.05	0.2	0.2	0.15	0.073	0.142	0.323	0.492
P28	7	0.492	0.1	0.4	0.6	1	0.224	1.892	0.924
P40	8	0	0	0	0	0	0	0	0
P53	9	0	0.154	0	0.154	0	0.154	-0.527	0.462

Cabe mencionar que los parámetros de las líneas presentan una relación resistencia-reactancia baja, lo que impide el desacople entre tensión y potencia activa, lo que dificulta la convergencia del método iterativo. Por lo tanto este caso de estudio permite además analizar el desempeño del algoritmo frente a esta característica.

A continuación se explican los datos utilizados para las unidades generadoras distinguiendo dos casos: simulación individual, es decir, un flujo de potencia simple, y simulación múltiple, es decir, múltiples flujos de potencia con datos obtenidos a partir de un despacho de unidades.

7.1.1. Simulación individual

Como se indica en [31] los resultados utilizados corresponden a un momento en que el generador diesel de la red se encontraba apagado, el generador eólico está fuera de servicio y la planta fotovoltaica de menor tamaño no se encuentra inyectando potencia. Por lo que solo se dejan como unidades generadoras las planta fotovoltaicas y el banco de baterías con su inversor.

En base a los flujos estimados en [31], se observa que el inversor se encuentra consumiendo aproximadamente unos 10[kW], es decir cargando las baterías, y entregando unos 4.5[kVAr] para regular tensión. Por otro lado la planta fotovoltaica de mayor tamaño entrega unos 19[kW] y consume unos 1.4[kVAr], mientras que la otra no inyecta potencia activa ni reactiva. En efecto esto flujos estimados se pueden observar en:

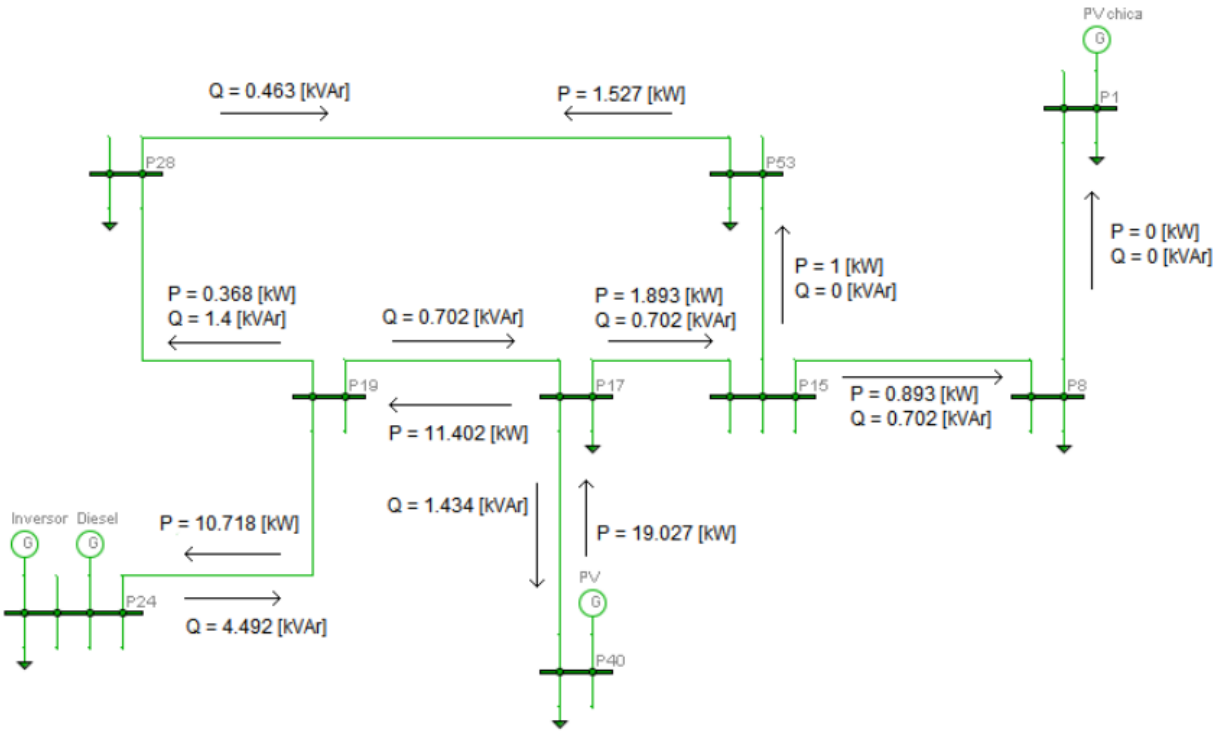

Figura 7.2: Red Huatacondina simplificada, estimación de flujos de potencia realizada en [31]. Fuente: [31].

Considerando que las plantas fotovoltaicas inyectan todo lo que pueden, y el inversor carga las baterías con la diferencia entre la generación y la demanda, se define al inversor como unidad *slack*, con un voltaje especificado de 1[p.u], que corresponde al voltaje medido en su barra según [31]. Las plantas fotovoltaicas se definen como unidades PQ, con las potencias mencionadas anteriormente como especificaciones de operación.

7.1.2. Simulación múltiple

Para obtener los datos de entrada al flujo de potencia trifásico para varios periodos se utiliza un archivo de despacho con datos de operación reales de la red de Huatacondo. Como se explica en 6.4, se utiliza la demanda total por periodo indicada en ese archivo, para escalar los consumos utilizados en la sección anterior. Para esta simulación además se elimina la planta fotovoltaica PV Plant 2, ya que el archivo de despacho utilizado solo contempla una planta fotovoltaica.

El archivo de despacho contempla la operación de 2 días, subdivididos en intervalos de 15 minutos.

7.2. Resultados Obtenidos

Considerando lo anterior se simulan cuatro casos:

1. La operación de la microrred con carga y líneas balanceadas.
2. La operación de la microrred con carga desbalanceada y líneas balanceadas.
3. La operación de la microrred con carga balanceada y líneas desbalanceadas.
4. La operación de la microrred con carga y líneas desbalanceadas.

7.2.1. Resultados simulación individual

Los resultados obtenidos se muestran como texto monoespaciado, tal como son mostrados por el programa. Sin embargo, solo se muestran los resultados para los voltajes en cada barra y la generación.

```
================================================================================
|     Generator Data                                                           |
================================================================================
     Total Three-Phase Generation
 Gen   Bus   Status     Pg        Qg
  #     #              (kW)     (kVAr)
----  -----  ------  --------  --------
  1      1      1      0.000    -0.000
  2      6      3    -10.110     4.488
  3      8      1     19.025    -1.395
                     --------  --------
            Total:      8.92      3.09

     Phase Generation Details
 Gen   Bus   Status     PGa        QGa     PGb        QGb     PGc       QGc
  #     #              (kW)     (kVAr)     (kW)     (kVAr)    (kW)    (kVAr)
----  -----  ------  --------  --------  --------  --------  --------  --------
  1      1      1     -0.000     0.000     0.000    -0.000     0.000     0.000
  2      6      3     -3.370     1.496    -3.370     1.496    -3.370     1.496
  3      8      1      6.342    -0.465     6.342    -0.465     6.342    -0.465
                     --------  --------  --------  --------  --------  --------
            Total:     2.972     1.031     2.972     1.031     2.972     1.031

================================================================================
|     Bus Data                                                                 |
================================================================================
 Bus      Voltage A        Voltage B            Voltage C
  #   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)
----- ------- --------  -------- --------  -------- --------
    1 1.0048    0.170    1.0048 -119.830    1.0048  120.170
    2 1.0048    0.170    1.0048 -119.830    1.0048  120.170
    3 1.0049    0.167    1.0049 -119.833    1.0049  120.167
    4 1.0054    0.161    1.0054 -119.839    1.0054  120.161
    5 1.0029    0.103    1.0029 -119.897    1.0029  120.103
    6 1.0000    0.000    1.0000 -120.000    1.0000  120.000*
    7 1.0027    0.135    1.0027 -119.865    1.0027  120.135
    8 1.0275    0.375    1.0275 -119.625    1.0275  120.375
    9 1.0046    0.166    1.0046 -119.834    1.0046  120.166
```

Figura 7.3: Red Huatacondina simplificada, resultados con carga y líneas balanceadas. Elaboración propia

```
================================================================================
|     Generator Data                                                           |
================================================================================
     Total Three-Phase Generation
 Gen   Bus   Status     Pg        Qg
  #     #              (kW)     (kVAr)
----  -----  ------  --------  --------
  1      1      1      0.000    -0.001
  2      6      3    -10.109     4.489
  3      8      1     19.025    -1.396
                     --------  --------
            Total:      8.92      3.09

     Phase Generation Details
 Gen   Bus   Status     PGa        QGa     PGb        QGb     PGc       QGc
  #     #              (kW)     (kVAr)     (kW)     (kVAr)    (kW)    (kVAr)
----  -----  ------  --------  --------  --------  --------  --------  --------
  1      1      1     -0.073    -0.020    -0.002    -0.003     0.075     0.023
  2      6      3     -3.449     1.459    -3.377     1.498    -3.283     1.533
  3      8      1      6.273    -0.468     6.342    -0.465     6.411    -0.463
                     --------  --------  --------  --------  --------  --------
            Total:     2.751     0.971     2.962     1.030     3.202     1.093

================================================================================
|     Bus Data                                                                 |
================================================================================
 Bus      Voltage A        Voltage B            Voltage C
  #   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)
----- ------- --------  -------- --------  -------- --------
    1 1.0055    0.171    1.0053 -120.013    1.0036  119.809
    2 1.0059    0.167    1.0054 -120.013    1.0032  119.814
    3 1.0060    0.164    1.0054 -120.015    1.0034  119.808
    4 1.0064    0.160    1.0059 -120.020    1.0041  119.800
    5 1.0039    0.099    1.0033 -120.071    1.0014  119.737
    6 1.0010    0.000    1.0004 -120.175    0.9986  119.632*
    7 1.0039    0.120    1.0034 -120.021    1.0008  119.764
    8 1.0283    0.373    1.0279 -119.806    1.0264  120.016
    9 1.0056    0.161    1.0051 -120.011    1.0030  119.806
```

Figura 7.4: Red Huatacondina simplificada, resultados con carga desbalanceada y líneas balanceadas. Elaboración propia

```
================================================================================
|     Generator Data                                                           |
================================================================================
     Total Three-Phase Generation
 Gen   Bus   Status     Pg        Qg
  #     #              (kW)     (kVAr)
----  -----  ------  --------  --------
  1      1      1     -0.000    -0.000
  2      6      3    -10.111     4.490
  3      8      1     19.025    -1.396
                     --------  --------
            Total:      8.91      3.09

     Phase Generation Details
 Gen   Bus   Status     PGa        QGa     PGb        QGb     PGc       QGc
  #     #              (kW)     (kVAr)     (kW)     (kVAr)    (kW)    (kVAr)
----  -----  ------  --------  --------  --------  --------  --------  --------
  1      1      1     -0.017    -0.055     0.011     0.058     0.007    -0.003
  2      6      3     -3.392     1.440    -3.356     1.554    -3.364     1.496
  3      8      1      6.335    -0.353     6.363    -0.581     6.328    -0.462
                     --------  --------  --------  --------  --------  --------
            Total:     2.925     1.031     3.018     1.031     2.971     1.031

================================================================================
|     Bus Data                                                                 |
================================================================================
 Bus      Voltage A        Voltage B            Voltage C
  #   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)
----- ------- --------  -------- --------  -------- --------
    1 1.0072    0.183    1.0024 -119.885    1.0048  120.151
    2 1.0073    0.166    1.0023 -119.869    1.0048  120.150
    3 1.0074    0.161    1.0024 -119.871    1.0049  120.147
    4 1.0079    0.154    1.0029 -119.876    1.0054  120.141
    5 1.0054    0.099    1.0003 -119.938    1.0029  120.083
    6 1.0026    0.000    0.9975 -120.043    1.0000  119.982*
    7 1.0053    0.132    1.0001 -119.900    1.0027  120.119
    8 1.0227    0.323    1.0323 -119.603    1.0274  120.356
    9 1.0071    0.161    1.0020 -119.871    1.0045  120.147
```

Figura 7.5: Red Huatacondina simplificada, resultados con carga balanceada y líneas desbalanceadas. Elaboración propia

```
================================================================================
|     Generator Data                                                           |
================================================================================
     Total Three-Phase Generation
 Gen   Bus   Status     Pg        Qg
  #     #              (kW)     (kVAr)
----  -----  ------  --------  --------
  1      1      1     -0.000    -0.001
  2      6      3    -10.110     4.491
  3      8      1     19.025    -1.396
                     --------  --------
            Total:      8.92      3.09

     Phase Generation Details
 Gen   Bus   Status     PGa        QGa     PGb        QGb     PGc       QGc
  #     #              (kW)     (kVAr)     (kW)     (kVAr)    (kW)    (kVAr)
----  -----  ------  --------  --------  --------  --------  --------  --------
  1      1      1     -0.090    -0.075     0.009     0.055     0.081     0.019
  2      6      3     -3.471     1.403    -3.362     1.556    -3.276     1.532
  3      8      1      6.266    -0.357     6.362    -0.581     6.397    -0.459
                     --------  --------  --------  --------  --------  --------
            Total:     2.706     0.971     3.009     1.030     3.201     1.093

================================================================================
|     Bus Data                                                                 |
================================================================================
 Bus      Voltage A        Voltage B            Voltage C
  #   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)   Mag(pu) Ang(deg)
----- ------- --------  -------- --------  -------- --------
    1 1.0078    0.184    1.0029 -120.067    1.0037  119.790
    2 1.0083    0.163    1.0028 -120.051    1.0032  119.794
    3 1.0084    0.158    1.0029 -120.052    1.0034  119.789
    4 1.0089    0.153    1.0033 -120.056    1.0040  119.781
    5 1.0065    0.096    1.0008 -120.110    1.0014  119.718
    6 1.0035    0.000    0.9978 -120.218    0.9986  119.615*
    7 1.0065    0.117    1.0009 -120.054    1.0008  119.748
    8 1.0235    0.321    1.0327 -119.784    1.0263  119.997
    9 1.0081    0.157    1.0026 -120.047    1.0029  119.787
```

Figura 7.6: Red Huatacondina simplificada, resultados con carga desbalanceada y líneas desbalanceadas. Elaboración propia

7.2.2. Resultados simulación múltiple

Como la comparación de los distintos desbalances se puede observar con los resultados anteriores, para la simulación múltiple se muestran simplemente los resultados correspondientes al caso balanceado y al caso en que están desbalanceadas las líneas y las cargas. Para esto se analizan las potencias de las unidades generadoras y banco de baterías, y las magnitudes de voltaje de las barras en que se encuentran conectados. Es importante mencionar, que las potencias mostradas en el caso balanceado corresponden a las potencias trifásicas totales, mientras que en el caso desbalanceado son las potencias por fase.

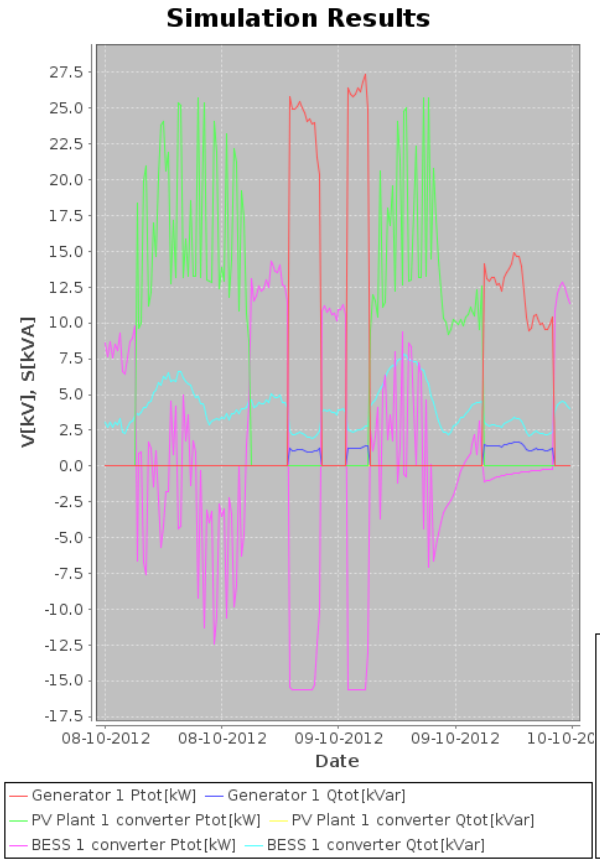

Figura 7.7: Resultados para múltiples periodos en red balanceada, inyecciones de potencia. Elaboración propia.

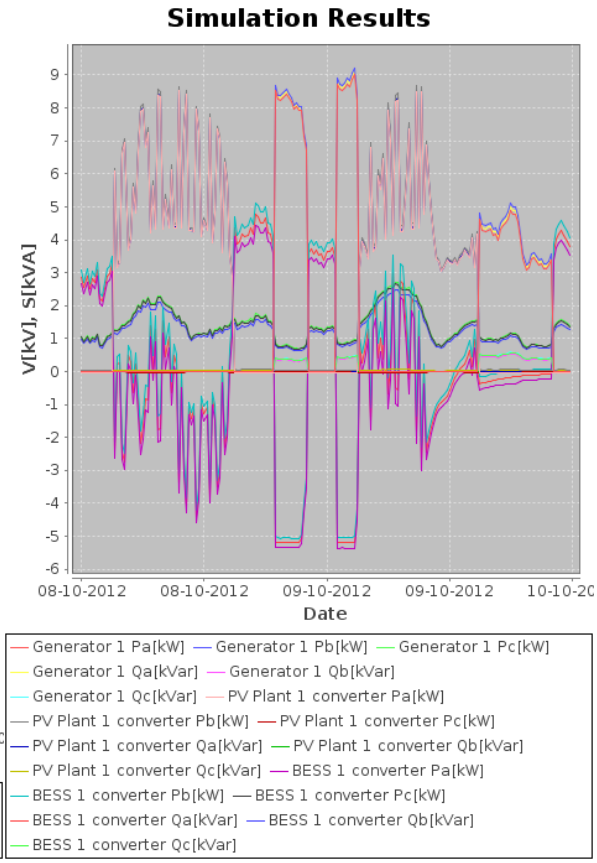

Figura 7.8: Resultados para múltiples periodos en red desbalanceada, inyecciones de potencia. Elaboración propia.

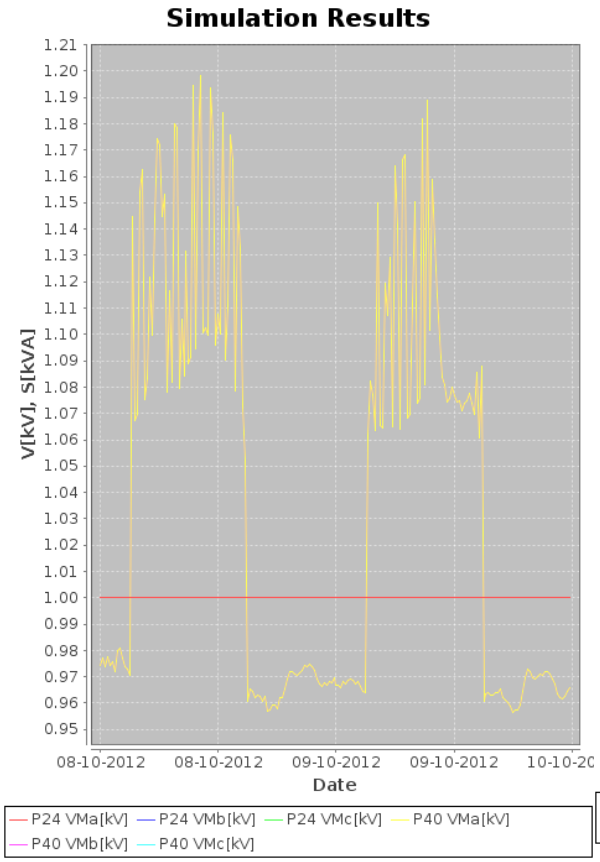

Figura 7.9: Resultados para múltiples periodos en red balanceada, voltajes en barras P24 y P40. Elaboración propia.

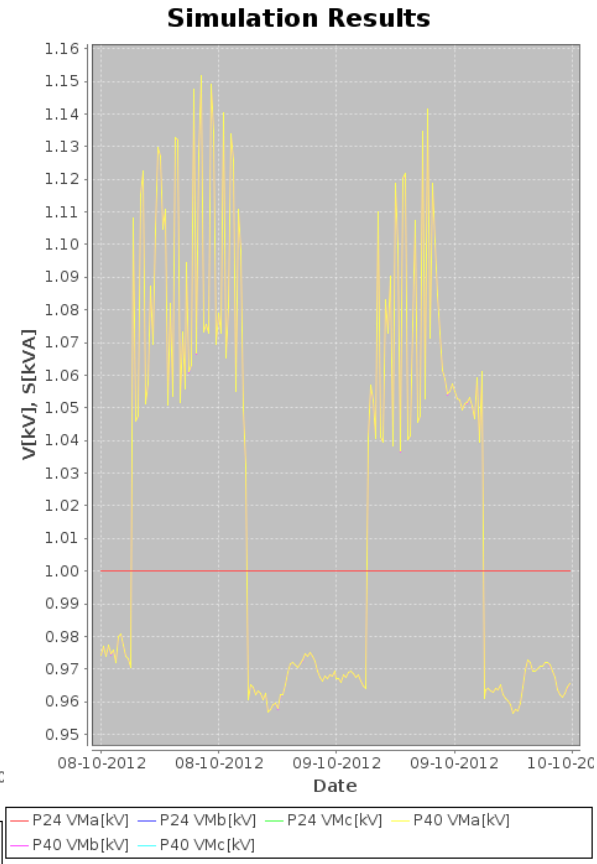

Figura 7.10: Resultados para múltiples periodos en red desbalanceada, voltajes en barras P24 y P40 . Elaboración propia.

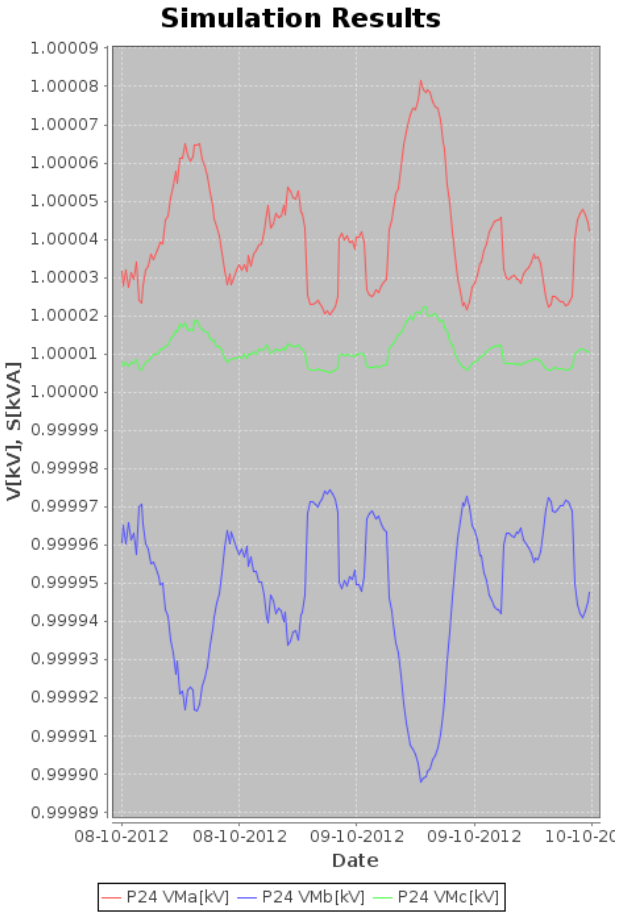

Figura 7.11: Resultados para múltiples periodos en red desbalanceada, voltajes en barra P24 . Elaboración propia.

7.3. Análisis de resultados

7.3.1. Análisisde resultados, simulación individual

Como se observa en la figura 7.3 los voltajes son perfectamente balanceados y los generadores inyectan lo mismo en cada fase, lo que es lógico debido a que se trata de un caso completamente balanceado. Se aprecia que las inyecciones del inversor corresponden a las esperadas, efectivamente el inversor posee una potencia activa negativa, lo que quiere decir que está cargando el banco de baterías, y una potencia reactiva positiva, es decir está manteniendo la tensión.

Las potencias de las demás unidades generadoras, que corresponden a ambas plantas fotovoltaicas, corresponden a los valores esperados, lo que es lógico ya que se trata de unidades PQ.

En las figuras 7.4, 7.5 y 7.6 es posible observar el efecto de los desbalances, es de notar que aparecen desbalances en los voltajes del orden de 0.01[p.u.] lo que es bastante bajo, sobretodo considerando que el voltaje nominal de la red es de 380[V]. Adicionalmente se observa que los generadores mantienen sus potencias trifásicas totales pero mediante inyecciones desbalanceadas, lo que permite suplir los desbalances en la demanda.

Es especialmente notable el caso del generador 1, que corresponde a la planta fotovoltaica menor, dicha planta, como fue asignado, entrega potencias trifásicas totales nulas. Sin embargo, producto de los desbalances, en algunas fases consume potencia mientras que en otras entrega, manteniendo una potencia total inyectada nula. Esto se explica con el modelo de las unidades, al existir reactancias de secuencias negativa y cero, pese a que no haya inyección de potencia, el desbalance en los voltajes producirá circulación de corrientes de secuencia en dichas reactancias. Estas corrientes de secuencia se traducen en corrientes, y por ende potencias de fase, no nulas, pero cuya suma sí es nula.

Cabe mencionar que en una microrred real una unidad que no esté inyectando potencia estará desconectada, por lo que el efecto descrito en el párrafo anterior no se apreciaría. Sin embargo, para efectos de análisis es interesante mantener este resultado ya que permite comprender mejor la influencia de los enrollados de secuencia negativa y cero.

En los demás casos se sigue apreciando que el inversor consume potencia activa y entrega potencia reactiva, manteniendo las magnitudes esperadas aproximadas, lo que valida el algoritmo frente a unidades *slack* que pueden consumir potencia, es decir, el algoritmo, frente a excesos de generación, puede efectivamente calcular cuánto consume el inversor, y en base a esto y su eficiencia, calcular cuánto se cargan las baterías.

Finalmente se tiene que la relación resistencia-reactancia no perjudica mayormente el desempeño del algoritmo ya que logra converger en 10 iteraciones con desbalance tanto de cargas como de líneas.

7.3.2. Análisis de resultados, simulación múltiple

Al observar las figuras 7.7 y 7.8 se observa que ambas tienen la misma forma, salvo que en el caso balanceado, como se grafican las potencias trifásicas totales, las magnitudes son el triple de las observadas en el gráfico del caso desbalanceado. Además en el caso desbalanceado es posible apreciar un leve desbalance en las fases.

Al analizar las curvas de potencia activa es posible observar cómo durante el día aparece la generación fotovoltaica, por lo que disminuye la descarga de las baterías, llegando incluso a cargarse en algunos periodos. Esta curva presenta irregularidades, lo que indica que corresponde a un día parcialmente nublado. Esto es más evidente en el segundo día, donde dicha curva posee una disminución considerable al final de la tarde. También se observa que el generador diesel, cuando entra en servicio, lo hace entregando bastante potencia, lo que genera excesos de generación, aprovechados por el banco de baterías que comienza a cargarse.

Por otro lado al observar las figuras 7.10 y 7.11 se observa que las curvas son bastante similares, más aún, en el caso desbalanceado no se aprecian desbalances. Sin embargo, al graficar solo los voltajes de la barra P24 se observa el desbalance entre las fases. Esto resulta lógico ya que la barra P40 no posee consumos, solo un generador fotovoltaico, por lo que el efecto del desbalance es menor en esa barra, mientras que la barra P24 posee una carga desbalanceada conectada directamente. De todas formas el desbalance entre las fases de la barra P24 es bastante bajo, pero el bajo nivel de variabilidad de la tensión en el tiempo hace que sea visible, contrariamente a lo que ocurre en la barra 40.

También se puede notar que el aumento de tensión en las barra P24 coincide con los periodos en que la planta fotovoltaica inyecta potencia a la red, y ambas curvas poseen una forma bastante similar, lo que coincide con lo esperado.

Capítulo 8

Conclusiones

En este trabajo se entrega una visión general del funcionamiento y operación de microrredes con unidades de generación distribuida, la modelación de éstas y la incorporación de recursos energéticos intermitentes, analizando las ventajas del enfoque de generación distribuida.

Como resultado se obtiene una aplicación que permite construir, configurar, guardar y simular la operación de distintas microrredes. Adicionalmente, se le otorga bastante énfasis al modelo computacional del programa, llegando a un diseño modular que facilita la extensión del programa y la implementación de nuevos componentes.

En efecto, el modelo computacional define las clases principales que modelan los distintos componentes de una microrred, por lo que para agregar nuevos componentes basta crear objetos que extiendan a los objetos ya definidos, lo que permite uniformar el modelo y facilitar la integración de nuevos componentes a la herramienta. Además el diseño permite abstraer a los nuevos diseños casi completamente del diseño gráfico, ya que los objetos que representan el comportamiento eléctrico de los componentes de la microrred, y los que los representan gráficamente son independientes, pudiendo reutilizarse alguno de los diseños ya existentes. Incluso la creación de nuevo objetos gráficos se simplifica al definirse casi todas las propiedades y métodos de éstos en una clase padre de la que se extienden.

La principal funcionalidad de la herramienta es la capacidad de simular flujos de potencia trifásicos en redes desbalanceadas, lo que permite analizar el nivel de desbalance en el sistema, y cómo las unidades de generación pueden corregir esto mediante distintas funciones de control. En este trabajo se explica en detalle el modelo utilizado, que se basa en el esquema de componentes de secuencia, lo que permite asignar las mencionadas funciones de control adicionales a las unidades, tales como inyecciones de ciertas corrientes de secuencia.

El algoritmo desarrollado es rápido, con un tiempo de cálculo comprobado de 0.1[s] para un caso desbalanceado de 5 barras y 1.04[s] para un caso balanceado de 300 barras. Además permite incorporar tanto unidades con restricciones en componentes de fase, como unidades con restricciones en componentes de secuencia, lo que hace al algoritmo bastante versátil, ya que permite incorporar unidades que regulan tensión de secuencia positiva y unidades con inyecciones determinadas en cada fase indistintamente. El algoritmo fue comparado con los casos de estudio balanceados IEEE de 9 a 300 barras, obteniéndose en el peor caso un error de voltaje $10^{-10}[p.u]$ y un error relativo de potencia de $10^{-7}[\,\%]$. También fue comparado con un caso de estudio desbalanceado de 5 barras obtenido de una publicación internacional, obteniéndose nuevamente resultados satisfactorios, con un error de 0.0005[p.u] para los voltajes y un error relativo de 0.811[%] en inyecciones de potencia.

Por otro lado, al estar desarrollado el flujo de potencia como una extensión del proyecto MATPOWER, no se limita su uso a sólo esta aplicación, sino que es posible incorporarlo en otros proyectos e incluso compartirlo bajo una licencia de software libre, con el fin de contribuir al desarrollo docente y generar más posibilidades de desarrollo del algoritmo en conjunto con otras instituciones docentes a nivel internacional.

Se logra incorporar la lectura de un despacho de unidades, que permite dar las consignas de operación a las unidades, para varios periodos en un horizonte de simulación dado. De esta forma es posible simular la operación de una microrred en un horizonte de simulación y no sólo en un instante determinado.

Estas funcionalidades se engloban en una interfaz de usuario simple e intuitiva, que permite manejar los datos de la microrred, visualizar los resultados del flujo de potencia y guardar los datos de la microrred, incluida la configuración de la simulación.

Adicionalmente, se analiza un caso de estudio basado en una microrred real, correspondiente al proyecto de Huatacondo desarrollado por Centro de Energía de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Con este caso de estudio se puede analizar el compartimiento de la herramienta frente a una red con parámetros reales, es decir de baja tensión y con una relación resistencia-reactancia baja, lo que afecta la convergencia del algoritmo. Como resultado de este análisis se obtiene que el algoritmo pudo converger sin problemas hacia una solución coherente y que no sólo reafirma el funcionamiento del algoritmo frente a una red desbalanceada, sino frente a una microrred, con las desventajas mencionadas anteriormente. Por otro lado, se observa que el algoritmo permite discernir cuando hay excedentes de generación, de esta forma, si se tiene un banco de baterías como unidad *slack* el algoritmo entrega como resultado la potencia que este consume, con lo que es posible obtener la potencia de carga de las baterías, entre otros datos.

8.1. Trabajos Futuros

Como trabajo futuro se propone lo siguiente:

- Incorporar el despacho económico de las unidades y la posibilidad de correr múltiples flujos de potencia a partir de ellos.
- Analizar los beneficios de un algoritmo de flujo de potencia óptimo trifásico e implementarlo, o bien incorporar el que ya trae implementado el proyecto MATPOWER.
- Modificar la interfaz de forma de, con el mismo algoritmo, poder resolver análisis de fallas.
- Mejorar la visualización gráfica de los resultados de acuerdo a las necesidades de los usuarios en el Centro de Energía de la Universidad de Chile.
- Implementar las funcionalidades “Deshacer”, “Rehacer”, “Copiar”, “Pegar” en el editor de microrredes.
- Implementar el monitoreo en tiempo real de una microrred a través de internet, en conjunto con la herramienta de estimación de estados.
- Incorporar modelos de otros componentes a la microrred, tales como:
    - Control de demanda

    - Vehículos eléctricos
    - Generadores micro y mini hidráulicos.

- Crear instaladores para el programa para distintos sistemas operativos, que incorporen todos los componentes adicionales al programa principal, las librerías requeridas, etc.

Glosario

Active Network Management Modules Módulos de Gestión Activa de la Red.

ATRM *"Active Thermal Rating Monitoring"*, Monitoreo Activo Térmico.

BESS *"Battery Energy Storage System"*, Sistema de almacenamiento de Energía en base a Baterías.

CC Control Central.

CIL *"Common Intermediate Language"*, Lenguaje Común Intermedio.

CLR *"Common Language Runtime"*.

CLS *"Common Language Specification"*, Especificación Común de Lenguajes.

DER *"Distributed Energy Resources"*, Recursos de Energía Distribuidos.

DG *"Distributed Generation"*, Generación Distribuida.

DS *"Distributed Storage"*, Acumuladores Distribuidos.

ERNC Energías Renovables No Convencionales.

ESS *"Electric Storage System"*, Sistema de Almacenamiento de Energía.

FACTS *"Flexible AC Transmision Systems"*, Sistemas Flexibles de Transmisión en Corriente Alterna.

GUI *"Graphical User Interface"*, Interfaz Gráfica para el Usuario.

IDE *"Integrated Development Environment"*, Entorno de Desarrollo Integrado.

JIT *"Just in Time Compilator"*, Compilador de bajo nivel de Java.

JVM *"Java Virtual Machine"*, Máquina Virtual de Java.

MC *"Microsource Controller"*, Microcontrolador.

OPF *"Optimal Power Flow"*, Flujo de Potencia Óptimo.

PF *"Power Flow"*, Flujo de Potencia.

SCADA *"Supervisory Control and Data Acquisition"*, Sistema de Supervisión, Control y Adquisición de Datos.

SFPS *"Three-phase Sequence-component Frame Power-flow Solver"*.

smartgrid Red Inteligente.

SO Sistema Operativo.

STATCOM *"Static Shynchronoys Compensators"*.

SVC *"Static Var Compensators"*.

UML *"Unified Modelling Language"*, Lenguaje Unificado de Modelación.

UPFC *"Unified Power Flow Control"*, Control Unificado de Flujos de Potencia.

VSC *"Voltage-Sourced Converter"*.

Bibliografía

[1] S.Chowdhury, S. Chowdhury, and P. Crossley, *Microgrids and Active Distribution Networks*, ser. Renewable Energy Series. London, United Kingdom: The Institution of Engineering and Technology, 2009, vol. 6, ch. 1 and 4.

[2] R. H. Lasseter, "Microgrids and distributed generation," *Energy Engineering*, 2007.

[3] B. Severino, "Modelación de generador fotovoltaico y banco de baterías de plomo Ácido como elementos de una microrred," Memoria (Título Ingeniero Civil Electricista), Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, Chile, Agosto 2011.

[4] C. Alvial-Palavicino, L. Reyes, G. Jiménez-Estévez, and R. Palma-Behnke, "Development of smart microgrids at community level: current challenges and barriers, and a novel methodological framework," *CE-FCFM*, 2012.

[5] F. Lanas, "Planteamiento de optimización del sistema de coordinación gevi," Memoria (Título Ingeniero Civil Electricista), Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, Chile, Abril 2011.

[6] O. o. E. T. United States Department of Energy and Distribution, "Grid 2030," in *Grid 2030 - A National Vision for Electricity's Second 100 years*, July 2003.

[7] I. E. Agency, "Technology roadmap: Smart grids," in *Technology Roadmap: Smart Grids*, Paris, 2011.

[8] V. Hamidi, K. Smith, and R. Wilson, "Smart grid technology review within the transmission and distribution sector," in *Innovative Smart Grid Technologies Conference Europe (ISGT Europe), 2010 IEEE PES*, Oct. 2010, pp. 1 –8.

[9] J. M. Piquer. (2011) Cc3301 - programación de software de sistemas. [Online]. Available: https://wiki.dcc.uchile.cl/cc3301/start

[10] M. Manzanedo, F. García, and I. Cruzado. (1999) Guía de iniciación al lenguaje java. [Online]. Available: http://zarza.usal.es/~fgarcia/doc/tuto2/I_4.htm

[11] J. M. Santos. Introducción al lenguaje c. [Online]. Available: http://sopa.dis.ulpgc.es/so/cpp/intro_c/

[12] cplusplus.com. [Online]. Available: http://www.cplusplus.com/info/description/

[13] The go programming language. [Online]. Available: http://golang.org/

[14] S. F. Python. Python programming language - official website. [Online]. Available: http://python.org/

[15] I. The MathWorks. Mathworks. [Online]. Available: http://www.mathworks.com/products/matlab/

[16] Debian.org. The computer language benchmarks game. [Online]. Available: http://shootout.alioth.debian.org/

[17] M. Abdel-Akher, K. Nor, and A. Rashid, "Improved three-phase power-flow methods using sequence components," *Power Systems, IEEE Transactions on*, vol. 20, no. 3, pp. 1389 – 1397, aug. 2005.

[18] W. Brokering, R. Palma, and L. Vargas, *Ñom Lüfke (El Rayo Domado) o Los Sistemas Eléctricos de Potencia*. Santiago, Chile: Pearson Prentice Hall, 2008.

[19] C. Cheng and D. Shirmohammadi, "A three-phase power flow method for real-time distribution system analysis," *Power Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 671 –679, may 1995.

[20] F. Zhang and C. Cheng, "A modified newton method for radial distribution system power flow analysis," *Power Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 389 –397, feb 1997.

[21] P. Garcia, J. Pereira, J. Carneiro, S., V. da Costa, and N. Martins, "Three-phase power flow calculations using the current injection method," *Power Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 508 –514, may 2000.

[22] A. Abur, H. Singh, H. Liu, and W. Klingensmith, "Three phase power flow for distribution systems with dispersed generation," *14th PSCC, Sevilla*, June 2002.

[23] M. Kamh and R. Iravani, "A unified three-phase power-flow analysis model for electronically coupled distributed energy resources," *Power Delivery, IEEE Transactions on*, vol. 26, no. 2, pp. 899 –909, april 2011.

[24] A. G. Expósito, *Análisis y operación de sistemas de energía eléctrica*. España: McGraw Hill, 2002, ch. 12.

[25] K. Lo and C. Zhang, “Decomposed three-phase power flow solution using the sequence component frame,” *Generation, Transmission and Distribution, IEE Proceedings C*, vol. 140, no. 3, pp. 181 –188, may 1993.

[26] B.-K. Chen, M.-S. Chen, R. Shoults, and C.-C. Liang, “Hybrid three phase load flow,” *Generation, Transmission and Distribution, IEE Proceedings C*, vol. 137, no. 3, pp. 177 –185, may 1990.

[27] X.-P. Zhang, “Fast three phase load flow methods,” *Power Systems, IEEE Transactions on*, vol. 11, no. 3, pp. 1547 –1554, aug 1996.

[28] R. D. Zimmerman, C. E. Murillo, and D. Gan. (2011) Matpower, a matlab power system simulation package. [Online]. Available: http://www.pserc.cornell.edu/matpower/m

[29] R. D. Zimmerman and C. E. Murillo-Sánchez, *Matpower 4.1 User’s manual*. Power Systems Engineering Research Center (Pserc), December 2011.

[30] C. Benavides, “Herramienta computacional para modelo de predespacho económico de carga,” Memoria (Título Ingeniero Civil Electricista), Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, Chile, Septiembre 2008.

[31] N. Lopez, “Desarrollo de un modelo de estimación de estado para la red eléctrica de huatacondo,” Memoria (Título Ingeniero Civil Electricista), Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, Chile, Noviembre 2011.

Anexo A

Detalle de la clase DeviceShape

A.1. Variables

- **gridSize:** corresponde al tamaño de la grilla, este valor es el mismo para todas las figuras de la imagen, y en base a este valor se construye y ubica la imagen en el panel.
- **x, y:** corresponden a las coordenadas `x` e `y` de la figura, estas posiciones se aproximan siempre al múltiplo de `gridSize` más cercano.
- **size:** corresponde al tamaño en pixeles de la figura, físicamente la figura es cuadrada, lo que facilita su rotación.
- **w, h:** corresponde al ancho y la altura de las figuras respectivamente, estas dimensiones son solo lógicas y se utilizan para definir los límites en que se considera que está la figura, lo que permite una correcta selección de ella.
- **nw, nh:** corresponden a la relación entre el ancho y la altura y el `gridSize`, respectivamente. Estos valores permiten redefinir las dimensiones de la figura cada vez que esta es escalada.
- **nwo, nho:** Al aplicar una rotación el ancho y la altura se intercambian entre sí, lo mismo ocurre con `nw` y `nh`, por lo que se tiene además a `nwo` y `nho` para almacenar las relaciones originales, es decir, sin aplicar rotación.
- **rotation:** almacena el ángulo de rotación en grados de la figura, pudiendo tomar los valores 0, 90, 180 y 270 °.

- **index:** permite asociar un índice entero a la figura con el fin de reconocerla en una lista.
- **textFactor:** corresponde a la relación entre el tamaño de la grilla y el tamaño que tendrán los textos, puede ser un valor no entero.
- **poleFactor:** corresponde a la relación entre el tamaño de la grilla y el tamaño que tendrán los puntos de conexión o `poles`, puede ser un valor no entero.
- **textUp:** variable booleana que indica si el texto debe ir en la parte superior de la figura o a un costado, como en caso de las barras.
- **landcape:** variable booleana que indica si la figura tiene una orientación horizontal o no, también se utiliza para la visualización de los textos.
- **landcape:** variable booleana que indica si la figura está seleccionada.
- **image:** variable de tipo `BufferedReader` que almacena el dibujo de la figura, posteriormente esta imagen es dibujada en el panel.
- **colors:** corresponde a un arreglo de objetos `Color` de Java que almacena los colores utilizados en al figura. Los colores se diferencian según corriente continua o alterna, distinguiendo 3 niveles de tensión alterna: 0.4, 1 y 3[kV]. En la mayoría de las figuras solo se utiliza un color, almacenado en la primera posición del arreglo, pero algunas como `ACDC_Shape` y `TrafoShape`, que conectan sistemas de distintas tensiones, utilizan más de un color.
- **name:** es una variable de tipo `String`, que almacena el nombre del dispositivo.
- **type:** es una variable de tipo `String`, que almacena el tipo, o subclase, del dispositivo. Es decir `type` puede contener: “TrafoShape” “LineShape”, etc.
- **poles:** corresponde a un arreglo de objetos `Pole`, estos objetos corresponden a los puntos de conexión entre un componente y otro y se utilizan para reconocer la interconexión de ellos.
- **device:** almacena el dispositivo del modelo eléctrico asociado.

A.2. Métodos

- `#setParameters(xo : int, yo : int, worg : int, nhorg : int, color : Color, grid : int)` permite definir la mayoría de los parámetros del objeto en una sola instrucción, este método, definido en la clase padre abstracta `DeviceShape` es llamado en los constructores de las clases derivadas, de esta forma el desarrollo de estas no requiere considerar esta labor.
- `+getX(): int` y `+getY(): int` permiten consultar las coordenadas `x` e `y` de la figura desde un entorno externo a ella.
- `+setX(xo: int)` y `+getY(yo: int)` permiten ajustar las coordenadas `x` e `y` de la figura desde un entorno externo a ella.
- `+contains(xo : float, yo : float) : boolean` es una función booleana que retorna un valor verdadero si la figura contiene el puento definido por las coordenadas `(xo,yo)`.
- `+hasBrother() : boolean` es una función booleana que retorna un valor verdadero si la figura posee un hermano, por defecto el retorno de este método siempre es falso, salvo en la caso de `LineShape`, que se explica más adelante.
- `+paint(g2d : Graphics)` este método pinta la figura en el entorno gráfico `g2d` de tipo `Graphics` recibido como parámetro, de esta forma, el `JPaintPanel` llama a este método de todas sus figuras entregándoles su propio entorno gráfico para que éstas se dibujen en él.
- `#paintText(g2d : Graphics)` es un método auxiliar llamado por el método `+paint(g2d : Graphics)` para escribir los textos en el entorno gráfico `g2d` de tipo `Graphics` dado como parámetro.
- `+rotate(dir : int)` este método rota la figura 90°en la dirección indicada en el parámetro `dir`.
- `+select()` permite seleccionar la figura, lo que además resalta sus colores para identificarla como seleccionada.
- `+deselect()` permite deseleccionar la figura, lo que regresa sus colores a su estado normal.
- `+getIndex() : int` permite consultar el índice asociado a la figura desde un contexto externo, además este índice será agregado al nombre del objeto, por ejemplo: “Busbar 2”.

- `+setIndex(index : int)` permite ajustar el índice asociado a la figura desde un contexto externo.
- `+scale(gridSize : int)` permite escalar la figura, para esto se modifica el tamaño de la grilla reemplazándolo por el valor del parámetro `gridSize`.
- `+drawShape()` finalmente la clase declara el método abstracto drawShape() que se utiliza para dibujar la figura, este método debe ser implementado por cada figura derivada de `DeviceShape` y es llamado por el método `+paint(g2d : Graphics)`.

Anexo B

Listado de funciones utilizadas en el flujo de potencia trifásico

A continuación se presenta un listado con las funciones más importantes y una breve descripción de lo que hacen:

Tabla B.1: Listado de funciones principales. Elaboración Propia

Función	**Descripción**
`[mpc1r, mpc3r] = balanced_test(casefile, mpopt)`	Corre un caso balanceado con el algoritmo original de MATPOWER y con el algoritmo desarrollado y compara sus resultados mostrando en pantalla las mayores diferencias en cada columna de cada campo.
`[ref, pv, pq] = bustypes3(bus, gen)`	Retorna vectores que contienen los números de las barras *slack*, PV y PQ
`out = convertZY(Z, Ys)`	Recibe una matriz de impedancia serie `Z` y una matriz de admitancia *shunt* `Ys` correspondientes a una rama, retorna en `out` un vector fila que contiene los parámetros necesarios para construir la matriz `branch`.
`define_constants3`	Es un *script* que define en el entorno de trabajo las variables retornadas por `idx_bus3`, `idx_gen3` e `idx_brch3`.
`f = fortescue`	Retorna la matriz de fortescue.
`[I0, I1, I2] = getIseq(SGa, SGb, SGc, SG1, IG0, IG2, Va, Vb, Vc, V0, V1, V2, Ybus)`	Retorna las corrientes especificadas por secuencia, es decir las corrientes inyectadas por los generadores menos la suma de las cargas y las inyecciones de corriente de acople debido a líneas desbalanceadas, de acuerdo a las ecuaciones 5.18, 5.19 y 5.20.

Tabla B.1: Listado de funciones principales. Elaboración Propia

Función	Descripción
`idx_bus3`, `idx_gen3`, `idx_brch3`	Retornan los índices de la columna a la que corresponde cada parámetro en las matrices `bus`, `gen`, `brch` respectivamente.
`[baseMVA, bus, gen, branch, areas, gencost, info] = loadcase(casefile)`	Es una función original de MATPOWER, retorna la estructura de datos almacenada en el archivo cuyo nombres es entregado en `casefile`.
`[baseMVA, bus, gen, branch, areas, gencost, info] = loadcase3bal(casefile, genstatus)`	Es una función análoga a la anterior pero permite cargar un caso monofásico como un caso trifásico balanceado.
`[SGa, SGb, SGc, SG1, IG0, IG2] = makeSbus3(baseMVA, bus, gen)`	Para los generadores con datos por secuencia retorna las inyecciones de potencia de secuencia positiva y corrientes de secuencias cero y negativa, para los que tienen datos por fase retorna las potencias en componentes de fase.
`[Va, Vb, Vc, V0, V1, V2] = makeVbus3(bus, gen)`	Retorna los voltajes por barra en cada fase y en cada secuencia
`[Ybus, Yf, Yt] = makeYbus3(baseMVA, bus, gen, branch)`	Retorna las estructuras $[Y_{bus}]$, $[Y_f]$ e $[Y_t]$, que contienen las correspondientes matrices para cada secuencia y combinación de ellas.
`[Ygen0, Ygen2] = makeYgen(baseMVA, gen)`	Retorna vectores con las admitancias de secuencia negativa y cero de los generadores.
`[options, names] = mpoption(varargin)`	Es una función original de MATPOWER que construye y retorna un vector de opciones de MATPOWER, si se llama sin argumentos retorna las opciones por defecto.
`[options, names] = mpoption3(varargin)`	Análogo a `mpoption` pero con las opciones por defecto para *PF* trifásicos.
`[V, converged, i] = newtonpf(Ybus, Sbus, V0, ref, pv, pq, mpopt)`	Corre el algoritmo de Newton Raphson monofásico, es una función original de MATPOWER.
`[bus, gen, branch] = pfsoln3(baseMVA, bus0, gen0, branch0, Ybus, Yf, Yt, V0, V1, V2, ref, pv, pq)`	Actualiza las matrices de datos de acuerdo a los resultados obtenidos del *PF*.
`[V0, V1, V2] = phase2seq( Va, Vb, Vc )`	Cada elemento i de los vectores en componentes de secuencia corresponde a la transformación de los i elementos de los vectores en componentes de fase. Esta función aplica para voltajes y corrientes.
`[lossA, lossB, lossC] = printpf3(baseMVA, bus, gen, branch, f, success, et, fd, mpopt)`	Imprime en el *file descriptor* `fd` el resumen de resultados del flujo de potencia trifásico.

Tabla B.1: Listado de funciones principales. Elaboración Propia

Función	Descripción
`[MVAbase, bus, gen, branch, success, et] = runpf3(casedata, mpopt, fname, solvedcase)`	Corre el flujo de potencia trifásico.
`fname_out = savecase3(fname, varargin)`	Guarda en el archivo indicado en `fname` el caso de MATPOWER entregado a través de los demás argumentos variables (`varargin`).
`[Va, Vb, Vc] = seq2phase( V0, V1, V2 )`	Cada elemento i de los vectores en componentes de fase corresponde a la transformación de los i elementos de los vectores en componentes de secuencia. Esta función aplica para voltajes y corrientes.
`[m, d] = to_pd(c, N)`	Retorna vectores con la magnitud y ángulo de un vector de números complejos.
`[c] = to_rd(m,d)`	Retorna un vector de números complejos a partir de vectores con sus magnitudes y ángulos.

Anexo C

Uso de Netbeans para la implementación de la herramienta

C.1. Interfaz gráfica de Netbeans

La versión de Netbeans IDE utilizada es la 7.0.1, licenciada bajo la *Common Development and Distribution License (CDDL)* y la *GNU General Public License version 2 with Classpath exception*. Es importante mencionar que esta versión de Netbeans es la última en soportar la edición de interfaces gráficas basadas en el *Single Application Framework* de Java, como se explica en **??**, la herramienta desarrollada se basa en este *framework*, por lo que para ser modificado mediante el diseñador de interfaz gráfica, se debe cuidar que la versión de Netbeans sea ésta o una inferior.

La interfaz de Netbeans es bastante modificable de modo que el usuario puede ajustarla de la manera que más le acomode, de todas formas la interfaz por defecto posee 6 ventanas principales, tal como se observa en la figura **??**.

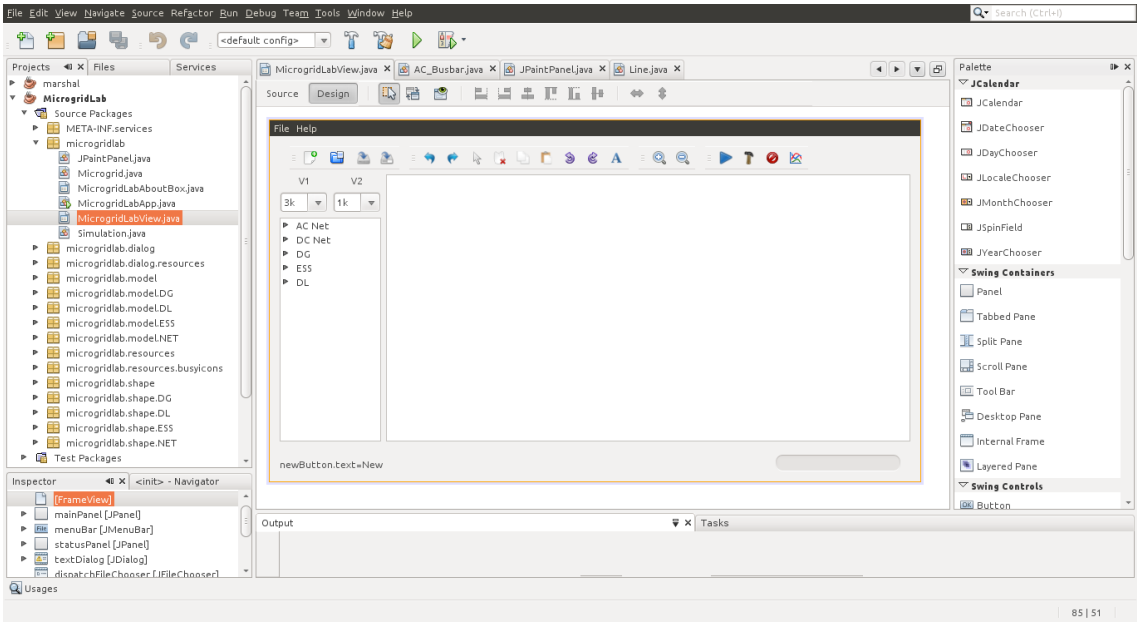

Figura C.1: Netbeans IDE 7.0.1. Elaboración propia.

- **Visualizador de proyectos, archivos y servicios:** esta ventana se ubica en la esquina superior izquierda y posee 3 pestañas:
  - *Projects*, donde se visualizan los proyectos y su jerarquía de paquetes y archivos.
  - *Files*, donde se visualizan los proyectos y la ubicación de sus archivos en el sistema.
  - *Services*, donde se visualizan los servicios o recursos de ejecución tales como servidores, bases de datos, etc.
- **Visualización de archivos:** en esta ventana, ubicada al centro, se edita el código de los archivos, es posible mantener abiertos varios archivos en pestañas.
- **Navigator:** al seleccionar un archivo que contiene el código de una o más clases, en esta ventana se muestra un resumen de la clase, es decir una lista con las clases definidas, sus variables de instancia, constructores y métodos. Esta ventana se ubica en al esquina inferior izquierda.
- **Inspector:** si se está trabajando en un proyecto con interfaz gráfica, al seleccionar el archivo que que define la interfaz, aparece la ventana *Inspector*, en la que se muestra un árbol jerárquico con los componentes de la interfaz. Cuando esto ocurre esta ventana aparece como una pestaña adicional junto a *Navigator*.

- **Palette:** al igual que en el caso anterior, si se selecciona el archivo que que define la interfaz es posible seleccionar entre las opciones *Source* y *Design*. En el primer caso se visualiza el código y en el segundo el editor del diseño gráfico de la interfaz, además en este caso aparece a la derecha la ventana *Palette*, en la que se muestra un menú para insertar componente en la interfaz.
- **Output:** en la parte inferior aparecen los mensajes correspondientes a la ejecución del programa, si hay más de una ejecución en curso éstas se muestran en pestañas.

C.2. Netbeans Projects

Para crear un nuevo proyecto en Netbeans se debe seleccionar la categoría "Java" y luego el tipo de proyecto, dentro de los que hay varios tipos. Para crear una aplicación simple basta seleccionar *Java Application*, pero para crear un proyecto con interfaz gráfica, utilizando las bondades de Netbeans, se debe seleccionar la opción *Java Desktop Application*, que utiliza el *Single Application Framework* de Java.

Al seleccionar un nuevo *Java Desktop Application* o al abrir un proyecto de este tipo previamente existente se pueden observar cuatro carpetas: *Source Packages*, *Test Packages*, *Libraries* y *Test Libraries*. Las tres últimas son administradas automáticamente por Netbeans, mientras que en la primera es donde el usuario debe poner sus archivos. Dentro de *Source Packages* se encuentran cuatro paquetes o *packages*: *META-INF.services*, *<proyecto>*, *<proyecto>.resources* y *<proyecto>.resources.busyicons*, donde *<proyecto>* corresponde al nombre del proyecto. *META-INF.services* contiene el nombre de la clase principal del programa, mientras que en *<proyecto>.resources.busyicons* se almacenan íconos. Estos paquetes vienen así por defecto en Netbeans y no es necesario editarlos.

Por otro lado en *<proyecto>* se almacenan los archivos con el código del programa, las clases, interfaz gráfica, etc, mientras que en *<proyecto>.resources* se encuentran archivos que definen las propiedades del programa, es decir los textos de los botones y otros componentes. No es necesario modificar estos archivos ya que todos estos cambios se pueden realizar directamente desde el editor de interfaz gráfica, sin embargo es útil utilizar este paquete para almacenar otro tipo de recursos necesarios para el programa como íconos u otros archivos.

Todo esto puede observarse en la figura **??**.

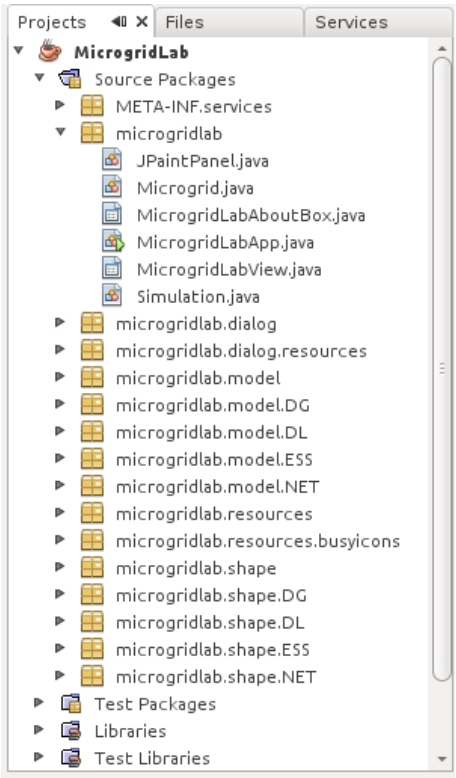

Figura C.2: Netbeans IDE 7.0.1, visualización de proyectos. Elaboración propia.

Como se puede observar en la figura, dentro del *package* principal (**`microgridlab`**) hay tres archivos que se componen con el nombre del proyecto:

- **MicrogridLabAboutBox.java**: contiene el código y diseño de la ventana *"About"* o "Acerca de" del programa, esta clase define un objeto que extiende la clase `JDialog` de Java.
- **MicrogridLabView.java**: contiene el código y diseño de la interfaz gráfica del programa, esta interfaz es un objeto de tipo `MicrogridLabView`. Esta clase extiende `FrameView` y es parcialmente modificable, ya que el código de declaración de los componentes de la interfaz está protegido, al igual que la declaración de los métodos definidos mediante el editor de interfaz gráfica, quedando pendiente su implementación al usuario. Adicionalmente el constructor de la clase llama al método `initComponents()` que es el encargado de crear la interfaz gráfica, este método también está protegido pero el resto del código del constructor no lo está.
- **MicrogridLabApp.java**: consiste en la clase principal, contiene el método `main()` que crea un objeto `MicrogridLabView`. Esta clase define un objeto que extiende la clase `SingleApplication` de Java.

Adicionalmente pueden obervarse otros *package*, los puntos en los nombres de paquetes, físicamente, corresponden a un subnivel o subcarpeta. Por ejemplo, según lo que se aprecia en la figura C.2 el *package* **microgrid.dialog** corresponde a una subcarpeta llamada `dialog` que se encuentra dentro de la carpeta **microgridlab**. Con esto en consideración, se clasifican los *package* restantes de la siguiente manera:

- **microgridlab.dialog**: contiene el código y diseño de las ventanas utilizadas para asignar los parámetros a las unidades, para cada componente se utiliza una ventana particular que corresponde a un objeto que extiende a la clase `JDialog` de Java. Así mismo el subpaquete **microgridlab.resources** es un paquete auxiliar donde se almacenan todas las propiedades de dichos objetos.
- **microgridlab.model**: contiene las clases correspondientes al modelo de datos descrito en 4.3. Así mismo, **microgridlab.model.NET**, contiene las clases correspondientes al modelo eléctrico descrito en la figura 4.2, mientras que **microgridlab.model.DG**, **microgridlab.model.DL**, y **microgridlab.model.ESS** contienen a las subclases de *DER*, descritas en la figura 4.3.
- **microgridlab.shape**: y sus subpaquetes, contienen las clases correspondientes al modelo de figuras descrito en 4.5.

Anexo D

Manual de utilización de la herramienta

D.1. Interfaz gráfica

D.1.1. Editor de microrred

En la figura 6.1 se muestra la interfaz gráfica desarrollada para la edición de las microrredes.

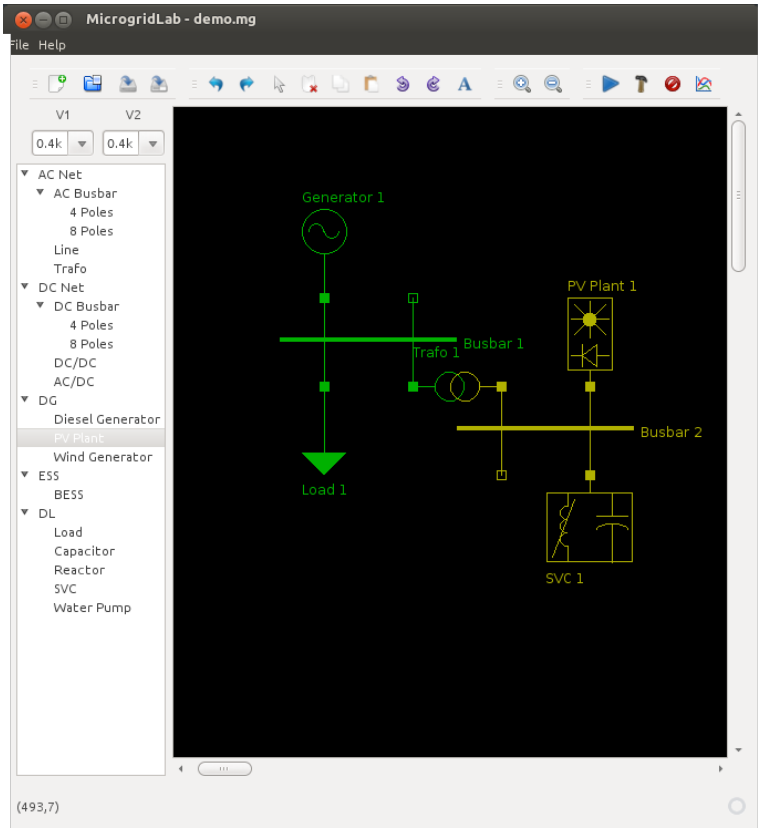

Figura D.1: Interfaz gráfica, Editor de microrred. Elaboración propia.

En el panel superior se encuentran las herramientas disponibles estas son:

- ***New***: crea una nueva microrred.
- ***Open***: permite buscar y abrir un archivo microrred.
- ***Save***: guarda la microrred en el archivo que corresponde, si es una nueva microrred abre una ventana de diálogo para buscar el archivo de destino.
- ***Save As***: lo mismo que el anterior pero siempre abre la ventana de diálogo para buscar el archivo.
- ***Undo***: "Deshacer", permite deshacer cambios realizados. No se encuentra implementado actualmente, se propone como trabajo futuro.
- ***Redo***: "Rehacer", permite rehacer cambios que se hayan "deshecho" con la función *Undo*. No se encuentra implementado actualmente, se propone como trabajo futuro.
- ***Select***: permite seleccionar y mover componentes de la microrred.
- ***Delete***: permite borrar componentes de la microrred.

- ***Copy***: permite copiar componentes de la microrred. No se encuentra implementado actualmente, se propone como trabajo futuro.
- ***Paste***: permite pegar componentes previamente copiados de la microrred. No se encuentra implementado actualmente, se propone como trabajo futuro.
- ***Rotate Counter Clockwise***: permite rotar los componentes seleccionados en sentido antihorario.
- ***Rotate Clockwise***: permite rotar los componentes seleccionados en sentido horario.
- ***Insert text***: permite insertar texto.
- ***Zoom In***: permite realizar *zoom in* o acercamiento visual a la microrred.
- ***Zoom Out***: permite realizar *zoom out* o alejamiento visual de la microrred.
- ***Run Simulation***: permite correr una simulación de acuerdo a la configuración de simulación guardada.
- ***Configure Simulation***: abre una ventana que permite configurar la simulación a realizar.
- ***Stop Simulation***: permite detener la simulación en curso. Actualmente no implementado, se proponer como trabajo futuro.
- ***Switch to Chart Window***: permite pasar rápidamente a la ventana con los resultados de la simulación.

Las conexiones de los componentes se realizan mediante unos pequeños cuadrados, éstos cuadrados corresponden a los polos de conexión y si se presiona sobre ellos con el ratón pueden abrirse y cerrarse, lo que eléctricamente equivale a abrir o cerrar un interruptor, dejando la unidad fuera de servicio. Es una rama (transformador, línea o DC/DC) es importante abrir ambos extremos para dejar la unidad fuera de servicio, de lo contrario puede producirse un error en la simulación.

D.1.1.1. Edición de parámetros de unidades

Para editar los parámetros de una microrred basta presionar el botón derecho del ratón sobre el componente a editar. Como cada componente posee una ventana personalizada, para no aburrir al lector, se mostrará algunas de las ventanas a modo de ejemplo.

Todos los componentes tienen una ventana con varias pestañas para editar distintos tipos de parámetros, particularmente todas estas ventanas poseen la misma pestaña *“Economic Properties”* donde se configuran los parámetros económicos del componente, el detalle de estos parámetros se encuentra en 4.2.

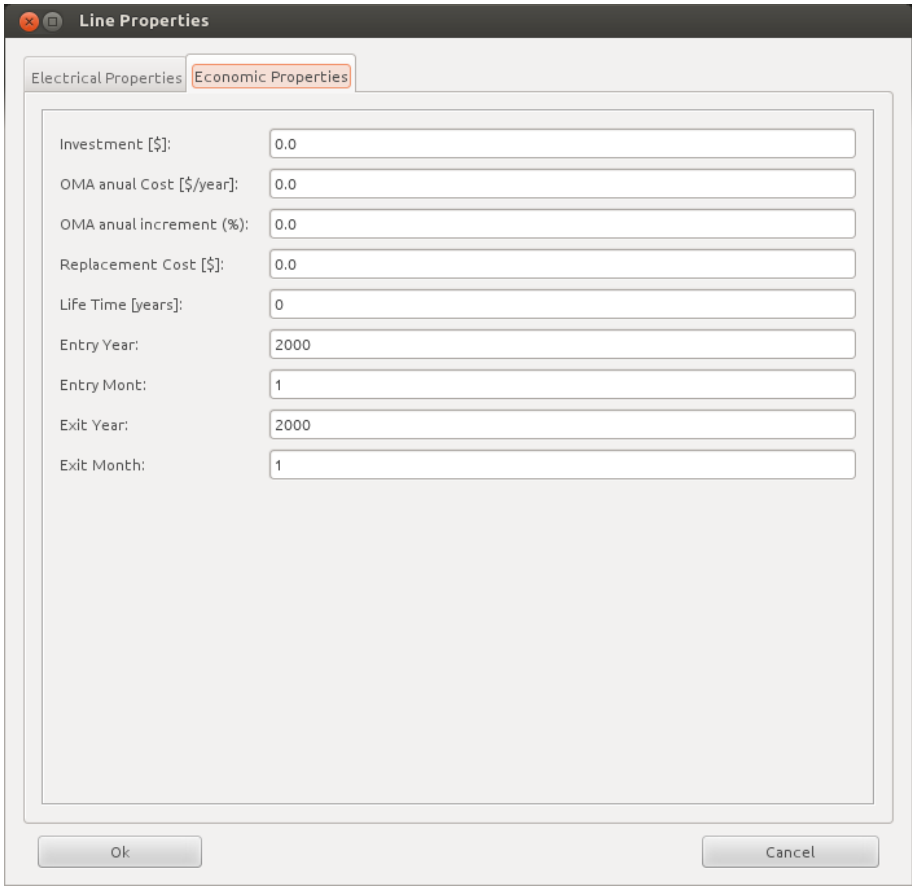

Figura D.2: Interfaz gráfica, Configuración de propiedades económicas. Elaboración propia.

A modo de ejemplo, en la figura **??** se presenta la ventana de configuración de las unidades *BESS*, donde en el panel izquierdo se configuran los parámetros del banco de baterías, mientras que en el panel derecho se configuran los parámetros de los módulos, incluída la curva de carga la que se describe mediante la tabla en la parte inferior de dicho panel, de la misma forma que en [5].

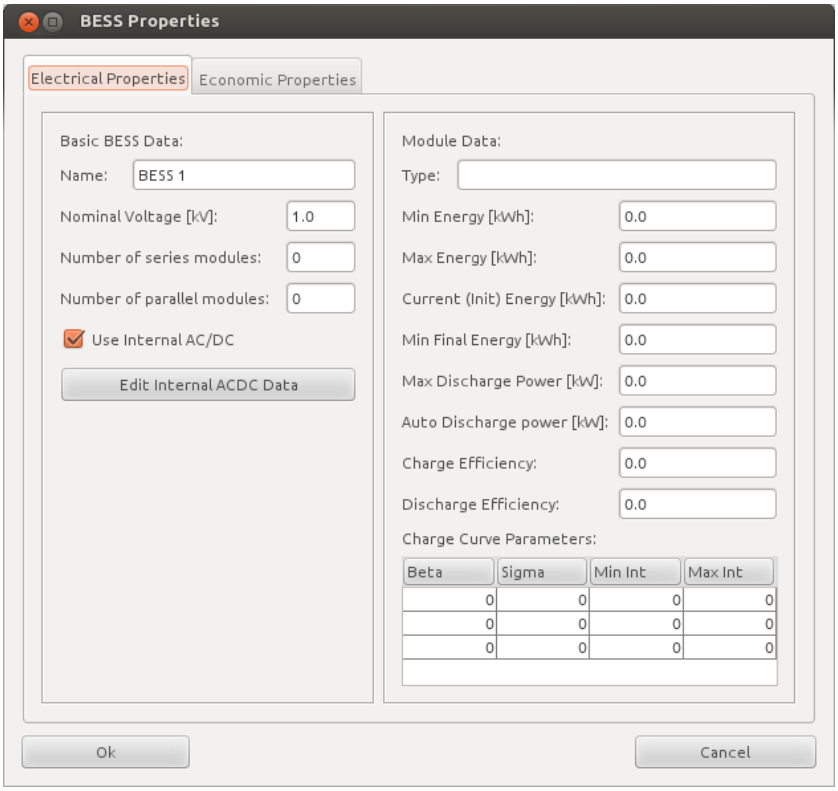

Figura D.3: Interfaz gráfica, Configuración de propiedades económicas. Elaboración propia.

El botón *Edit Internal ACDC Data* abre un cuadro de diálogo que permite configurar la unidad *AC/DC* interna del *BESS*, tal como se muestra en la figura fig:capImp:acdcGUI. Los campos para escribir los datos de voltaje están desactivados por tratarse de una unidad *AC/DC* asociada a una unidad de generación DC, por lo que toma como voltajes nominales el voltaje nominal de la unidad que la posee.

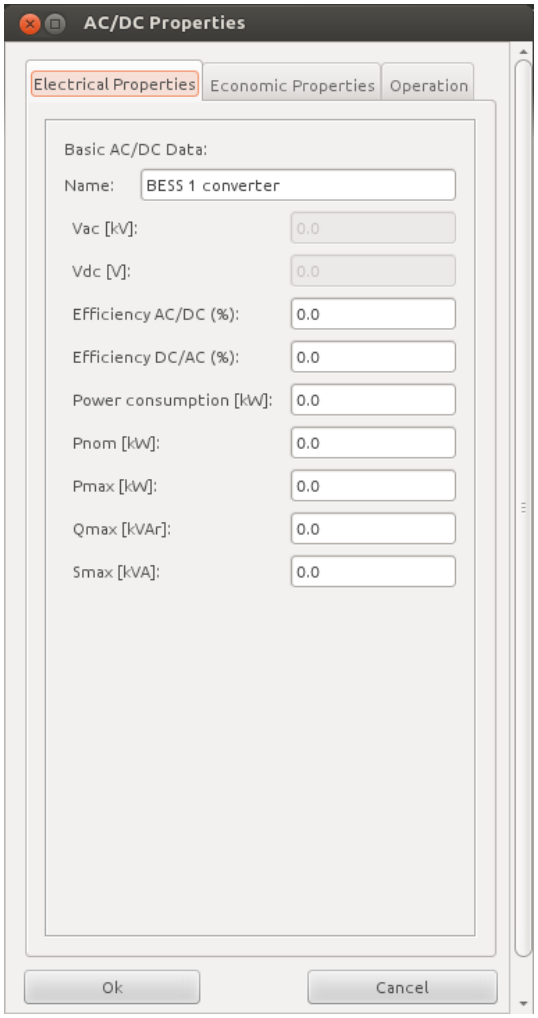

Figura D.4: Interfaz gráfica, Configuración de propiedades económicas. Elaboración propia.

D.1.1.2. Configuración de la simulación

En la ventana de configuración de simulación existen tres opciones o modos de simulación:

- ***Power flow*:** corre un flujo de potencia simple.
- ***Dispatch*:** realiza un despacho económico de la red para un horizonte de simulación determinado.
- ***Dispatch + Power flow*:** realiza un despacho económico de la red para un horizonte de simulación determinado y con esos resultados corre un flujo de potencia para cada periodo.

Cada uno de estos modos posee una pestaña de configuración:

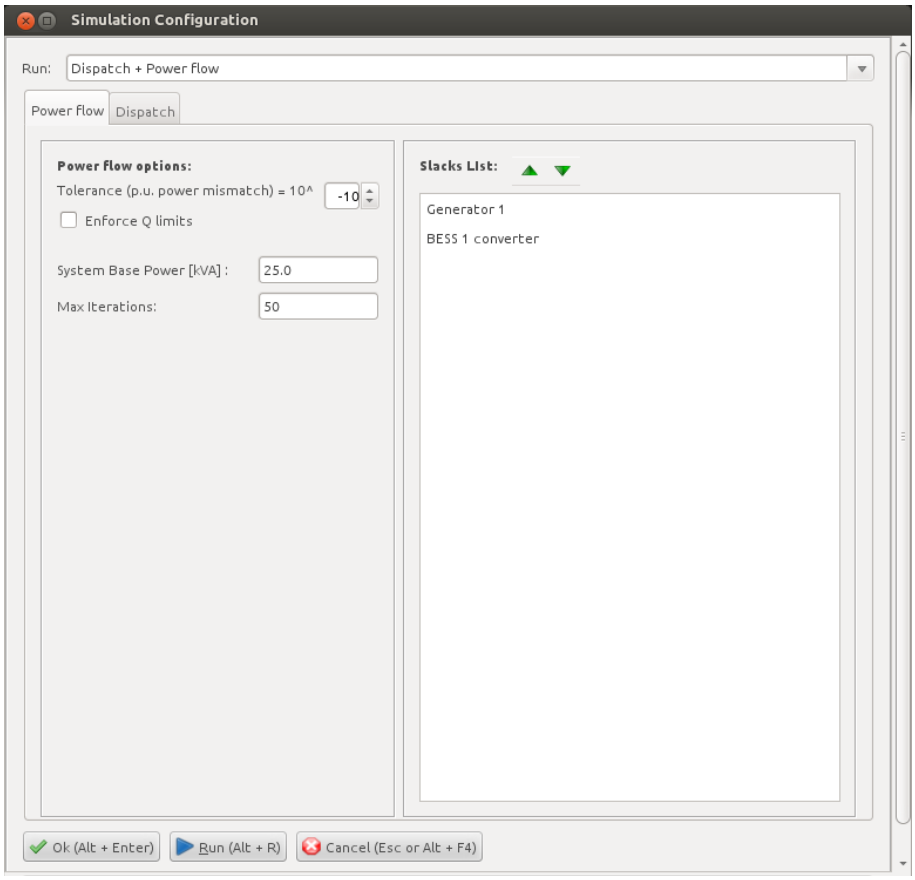

Figura D.5: Interfaz gráfica, Configuración del flujo de potencia. Elaboración propia.

Para el flujo de potencia se puede configurar la tolerancia utilizada para revisar la convergencia del método numérico, que corresponde al error máximo aceptable de potencia en p.u., y si se corrigen límites de reactivos de las unidades de generación.

Adicionalmente, en el panel del lado derecho se configura el orden de prioridad de las unidades *slack*, de esta forma, en caso de haber más de una, la primera unidad operará como *slack*, dejando a las demás como unidades PV. En los periodos en que la unidad *slack* esté fuera de servicio la unidad que sigue en esta lista tomará su lugar durante esos periodos.

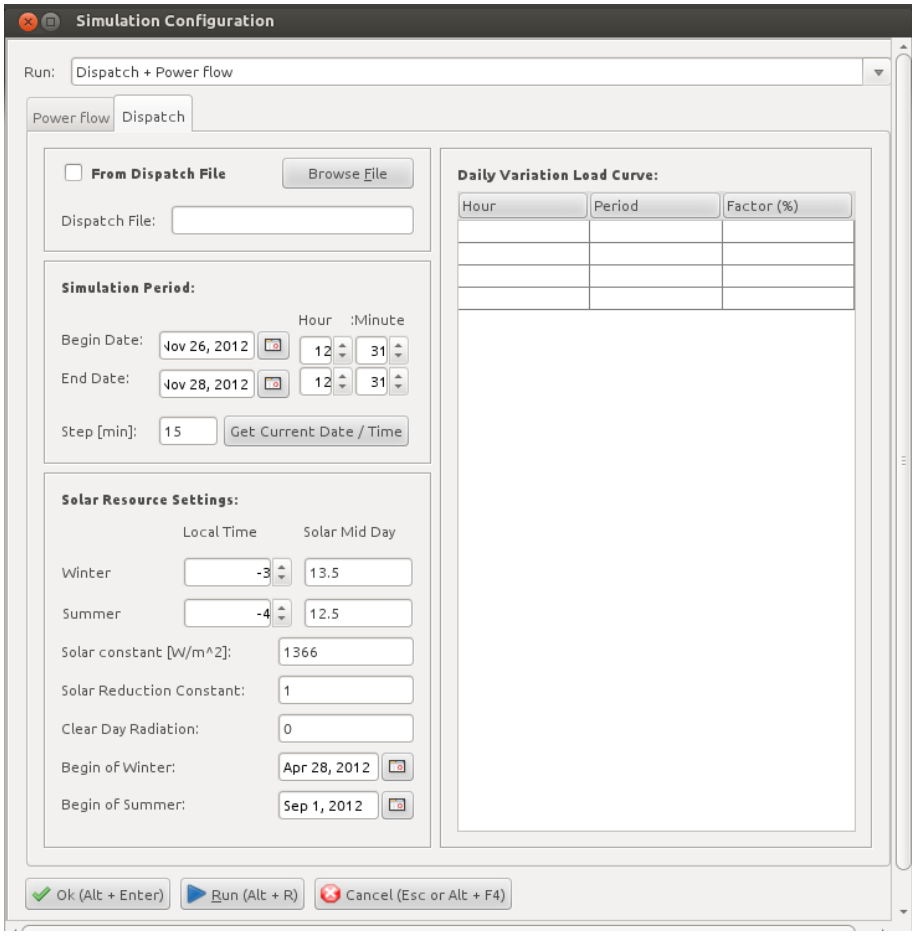

Figura D.6: Interfaz gráfica, Configuración del despacho económico. Elaboración propia.

Para el despacho económico se aprecian cuatro subpaneles:

- ***From Dispatch File:*** En el primero es posible cargar el despacho desde un archivo previamente almacenado.
- ***Simulation Period:*** En el segundo se configura el horizonte de simulación a partir de una fecha y hora de inicio y una fecha y hora de término, por defecto considera para el comienzo la fecha y hora actuales con una extensión de dos días completos. También se define el intervalo de tiempo entre cada despacho, por defecto son 15 minutos.
- ***Solar Resource Settings:*** En el tercero se configuran los datos correspondientes al recurso solar, los que corresponden a las horas locales y el medio día solar de invierno y verano, la constante solar, la constante de reducción solar y la radiación de día claro. Además de las fechas de comienzo de los horarios de invierno y verano.

- ***Daily Variation Load Curve:*** Finalmente, en el panel del lado derecho, se configura la curva de variación de carga diaria. Esta curva de variación de carga diaria corresponde a valores en porcentaje para cada periodo de la simulación, esta curva se aplicará a las potencias de fase de cada carga, con el fin de dar variabilidad a la demanda del sistema en el tiempo. Esto sin embargo impondrá que todas las cargas tendrán la misma variación temporal, por lo que se deja propuesto como trabajo a futuro implementar una forma de asignar curvas distintas a cada carga.

D.1.2. Visualización de resultados

La ventana de visualización de resultados ofrece funcionalidades como *zoom*, almacenar los gráficos como imágenes, imprimir los gráficos, entre otras, todas accesibles presionando el botón derecho del ratón sobre el gráfico. Es posible seleccionar los datos a graficar mediante los *check box* del lado derecho. Las unidades para las que se muestran los datos indicados en los *check box* corresponden a las unidades seleccionadas en el editor, para seleccionar múltiples unidades se puede arrastrar un rectángulo presionando el botón izquierdo del ratón o seleccionando individualmente mientras se mantiene presionada la tecla `Ctrl`. El gráfico se actualiza al modificar el estado de alguno de los *check box*, al presionar el botón dedicado para eso, o al apretar la tecla F5.

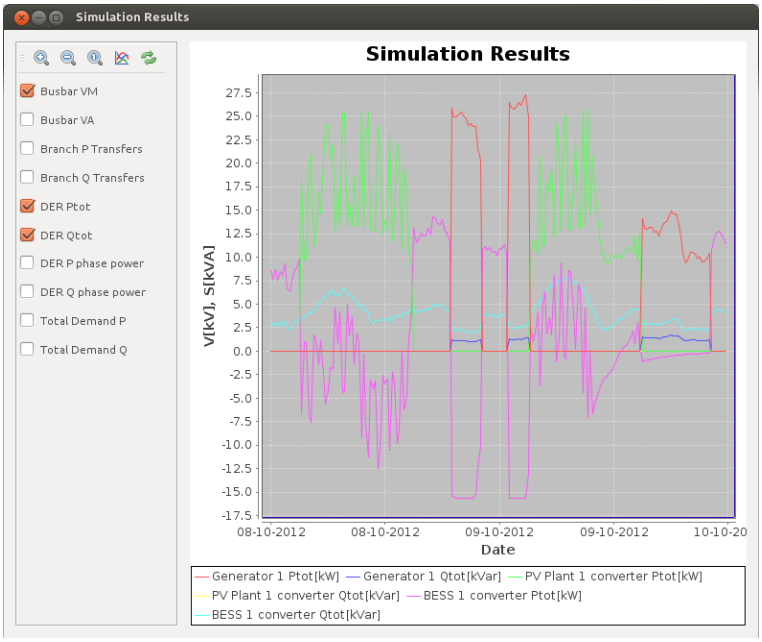

Figura D.7: Interfaz gráfica, Visualización de resultados. Elaboración propia.