

Manual de usuario

Modelo de gestión de energía de Metro



Nicolás Cifuentes O.
ncifuentes@gmail.com
2017



Contenido

Capítulo 1: Guía de instalación	3
1.1. Requerimientos.....	3
1.2. Instalación	4
Capítulo 2: Estructura de datos	13
2.1. Tablas para redes de tracción DC	13
2.2. Tablas para redes de suministro en media tensión AC	16
2.3. Tablas para escenarios de simulación	20
2.4. Tablas de resultados.....	21
Capítulo 3: Estructura algoritmo	24
3.1. Diagrama de flujo general.....	24
3.2. Estructura módulo RedLineas	25
3.3. Estructura módulo RedAC	28
Capítulo 4: Fundamentos teóricos.....	32
4.1. Representación nodal de línea de tracción DC.....	32
4.2. Solución numérica de red nodal.....	33



Capítulo 1: Guía de instalación

Este capítulo tiene por objetivo especificar los requerimientos y pasos de instalación de **Python**, así como de los módulos necesarios para ejecutar la herramienta de simulación para la gestión de energía de Metro en **Windows**.

1.1.Requerimientos

El modelo de gestión de energía de Metro consiste en dos módulos principales desarrollados en Python: *RedLineas.py* y *RedAC.py*. Ambos módulos tienen una programación orientada al objeto, dada su pertinencia a la hora de entregar flexibilidad para la representación de sistemas complejos como la red de Metro.

Adicionalmente, el modelo de gestión de energía de Metro tiene dos módulos secundarios de lectura y guardado de datos: *LeerDatos.py* y *GuardarDatos.py*. Estos módulos tienen únicamente funciones, las cuales permiten -en el primer caso- rescatar los datos de las topologías de las redes de tracción y de suministro de corriente de alterna, junto con los perfiles de consumo y generación de cada escenario de simulación, datos disponibles en una base de datos MySQL. La información es procesada y guardada en un diccionario Python para entregarla como entrada a los módulos *RedLineas.py* y *RedAC.py*. El segundo módulo se encarga de guardar los resultados de las simulaciones en una base de datos MySQL a partir de un diccionario python, que se genera como salida de los módulos *RedLineas.py* y *RedAC.py*.

Finalmente, para ejecutar los módulos del modelo de gestión de energía de Metro se necesitan los siguientes módulos adicionales:

- Numpy
- SciPy
- Pypower
- Matplotlib
- Pymysql

Para instalar estos módulos se necesita instalar *Python 3* junto con *pip*. A continuación se resumen los requerimientos para la instalación y ejecución del modelo de gestión de energía de Metro.

Proceso	Requerimiento
Instalación	<ul style="list-style-type: none"> Python 3.6.2 Pip
Ejecución	<ul style="list-style-type: none"> <i>RedLineas.py</i> <i>RedAC.py</i> Numpy 1.13.1+mkl Scipy 0.19.1 Pypower 5.1.2 Matplotlib 2.0.2
Guardado de datos	<ul style="list-style-type: none"> <i>LeerDatos.py</i> <i>GuardarDatos.py</i> PyMySQL 0.7.11

Tabla 1: Requerimientos modelo de gestión de energía de Metro.

1.2. Instalación

1.2.1. Python

En primer lugar es necesario instalar Python 3, a partir del instalador disponible en la página oficial <https://www.python.org/downloads/>. Se sugiere desbloquear el instalador, mediante la pestaña propiedades a través de click derecho, así como ejecutarlo en modo administrador como se muestra en las imágenes a continuación.

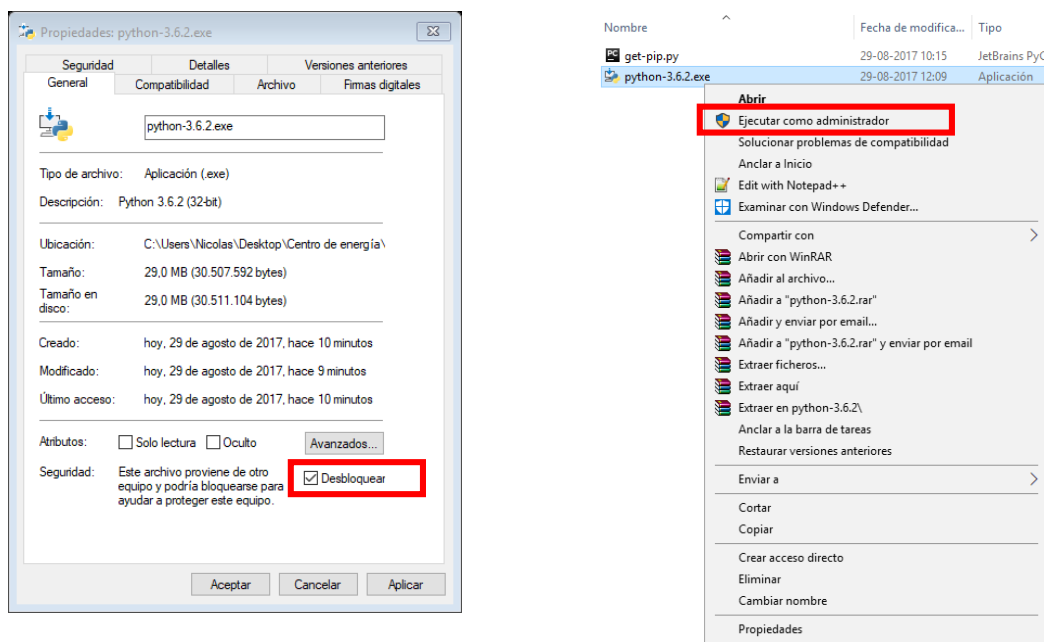


Figura 1: Instalación Python 3.6.2



Luego se deben seguir los pasos sugeridos en el asistente de instalación de Python **asegurándose de marcar que se agregue Python al PATH como se muestra a continuación.**

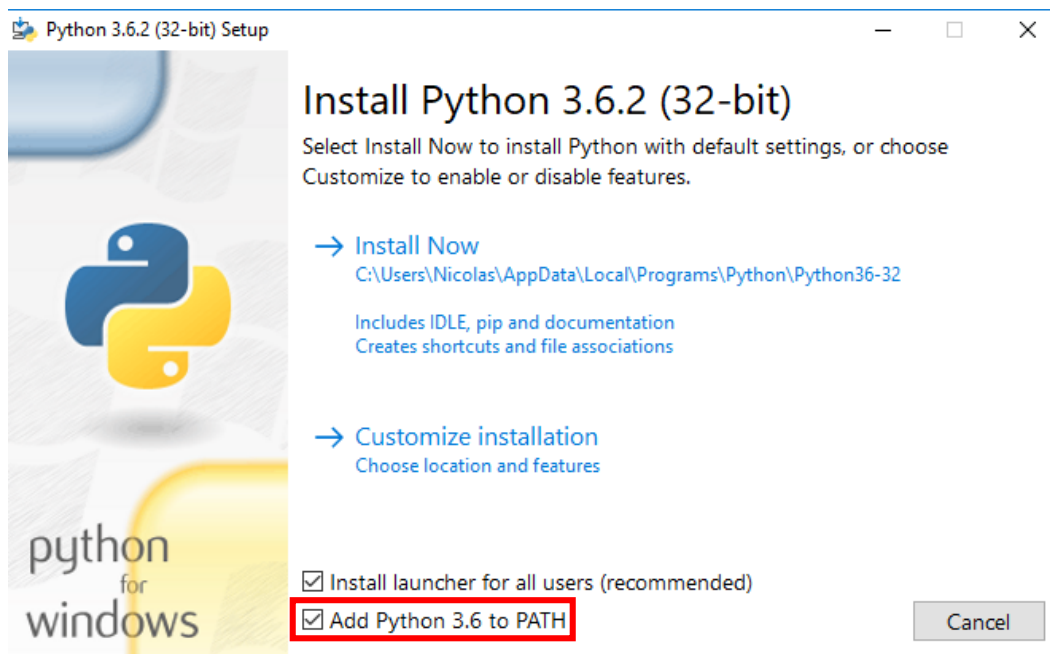


Figura 2: Agregar Python a PATH de Windows.

1.2.2. Pip

Luego de que la instalación de Python 3 se ha realizado con éxito, se debe instalar pip. Para esto se debe descargar el archivo `python get-pip.py` desde <https://bootstrap.pypa.io/get-pip.py>. Una vez descargado el archivo, se debe ejecutar la consola de comandos de Windows en **modo administrador**¹ como se muestra a continuación.

¹ Cuando se ejecuta la consola de comandos en modo administrador, esta se encuentra en el directorio "C:\WINDOWS\System32" en lugar de en el directorio del usuario respectivo "C:\Users\UsuarioX".

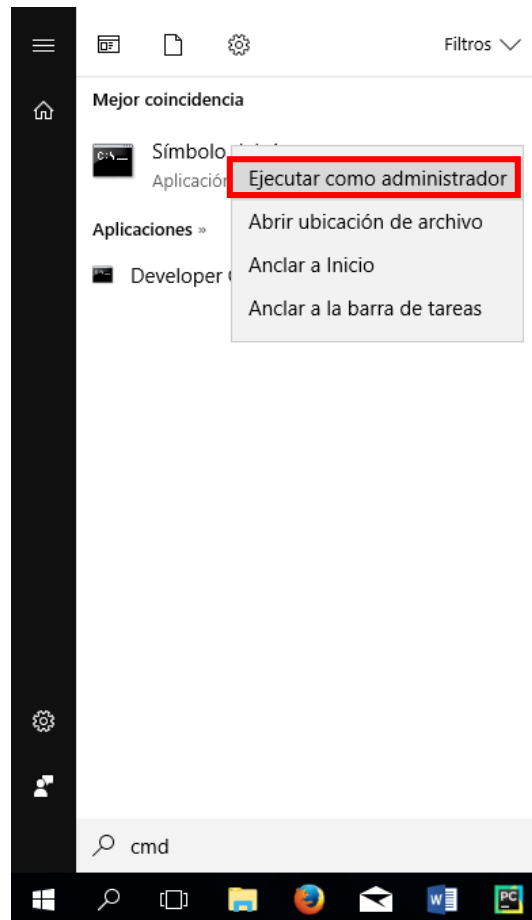


Figura 3: Instalación pip.

Una vez abierta la consola de comandos en modo administrador, es necesario ubicarse en el directorio donde se encuentra el archivo `python get-pip.py` usando el comando:

```
[REDACTED]
```

Una vez en el directorio del archivo `python get-pip.py`, se debe usar el siguiente comando:

```
[REDACTED]
```

Si se siguieron los pasos de las secciones 1.2.1 y 1.2.2 correctamente, pip debería quedar instalado y se ve el siguiente mensaje de instalación correcta en la consola de comandos de Windows.



CENTRO DE ENERGÍA

```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>python get-pip.py
Requirement already up-to-date: pip in c:\users\nicolas\appdata\local\programs\python\python36-32\lib\site-packages
Collecting wheel
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
    100% |#####| 71kB 305kB/s
Installing collected packages: wheel
Successfully installed wheel-0.29.0

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>
```

Figura 4: Instalación exitosa de pip en Windows.

Pip permite instalar de forma sencilla módulos de Python. A continuación se usa para instalar los módulos adicionales necesarios para ejecutar el modelo de gestión de energía de Metro.

1.2.3. Módulos adicionales

Para ejecutar el modelo de gestión de energía de Metro, es necesario instalar los módulos adicionales usando pip mediante los siguientes comandos:

1. Numpy 1.13.1+mkl:

Se sugiere instalar en primer lugar este módulo dado que varios otros dependen de este. Descargar el archivo Wheel para instalar el módulo de numpy usando pip desde: <http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy>. Se debe seleccionar la versión que corresponda con la de Python, ejemplo: Se tiene instalado “Python 3.6.2 32 bits”, en ese caso se selecciona el archivo Wheel: “numpy-1.13.1+mkl-cp36-cp36m-win32.whl”. Luego se debe ejecutar el siguiente comando estando en el directorio donde se encuentra el archivo wheel:





CENTRO DE ENERGÍA

```
Administrador: Símbolo del sistema

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energía\Instalación>pip install numpy
Collecting numpy
  Downloading numpy-1.13.1-cp36-none-win32.whl (6.8MB)
    100% |#####| 6.8MB 161kB/s
Installing collected packages: numpy
Successfully installed numpy-1.13.1

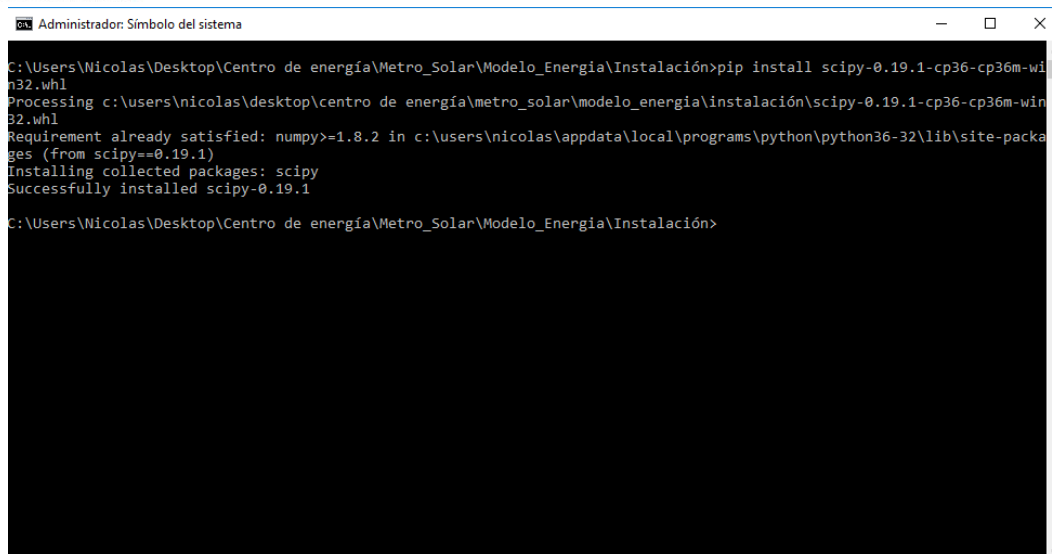
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energía\Instalación>
```

Figura 5: Instalación módulo numpy.

2. Scipy 0.19.1:

Antes de instalar este módulo se debe descargar Visual C++ dependiendo de la versión de Python que se use. En este caso, para Python 3.6 se necesita Visual C++ 2015. Luego se debe descargar el archivo Wheel para instalar el módulo de SciPy usando pip desde: <http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy>. Al igual que para el módulo de numpy, se debe seleccionar la versión que corresponda con la de Python, ejemplo: Se tiene instalado “Python 3.6.2 32 bits”, en ese caso se selecciona el archivo Wheel: “scipy-0.19.1-cp36-cp36m-win32.whl”. Luego se debe ejecutar el siguiente comando estando en el directorio donde se encuentra el archivo wheel:





```
Administrador: Símbolo del sistema

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>pip install scipy-0.19.1-cp36-cp36m-win32.whl
Processing c:\users\nicolas\desktop\centro de energía\metro_solar\modelo_energía\instalación\scipy-0.19.1-cp36-cp36m-win32.whl
Requirement already satisfied: numpy>=1.8.2 in c:\users\nicolas\appdata\local\programs\python\python36-32\lib\site-packages (from scipy==0.19.1)
Installing collected packages: scipy
Successfully installed scipy-0.19.1

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>
```

Figura 6: Instalación de módulo SciPy.

Para los siguientes módulos no es necesario descargar archivos, se instalan directamente desde la consola de comandos usando pip.

3. Pypower 5.1.2:



```
Administrador: Símbolo del sistema

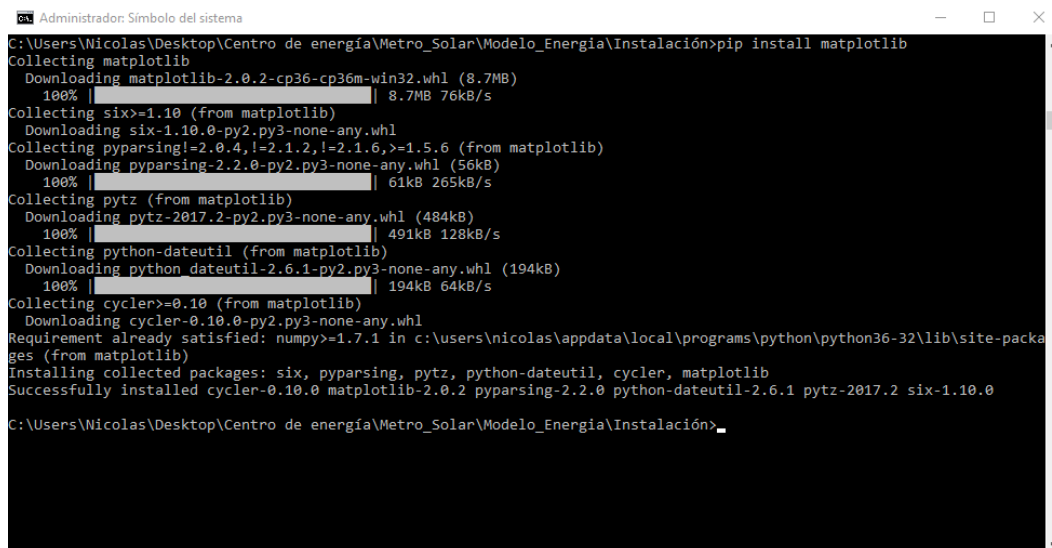
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>pip install PYPPOWER
Collecting PYPPOWER
  Downloading PYPPOWER-5.1.2.tar.gz (269kB)
    100% |#####| 276kB 60kB/s
Building wheels for collected packages: PYPPOWER
  Running setup.py bdist_wheel for PYPPOWER ... done
  Stored in directory: C:\Users\Nicolas\AppData\Local\pip\Cache\wheels\b0\de\c0\1c4a3aad66226c1f7c70a2b02d9b05ea61d04eb71134dbb03e
Successfully built PYPPOWER
Installing collected packages: PYPPOWER
Successfully installed PYPPOWER-5.1.2

C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>
```

Figura 7: Instalación módulo PYPOPWER.

4. Matplotlib 2.0.2:

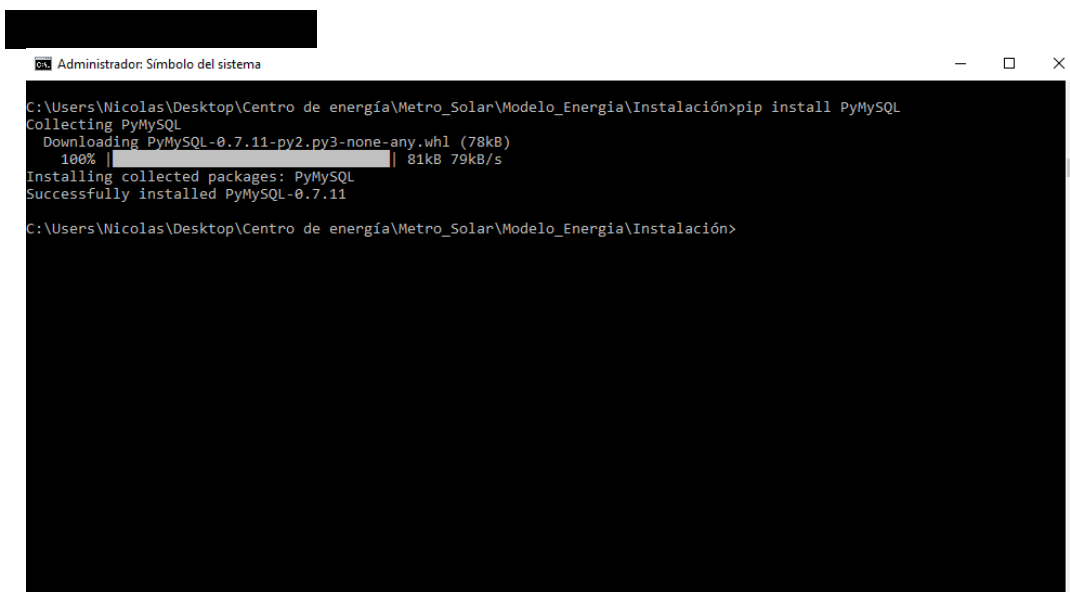




```
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-2.0.2-cp36-cp36m-win32.whl (8.7MB)
    100% |#####| 8.7MB 76kB/s
Collecting six>=1.10 (from matplotlib)
  Downloading six-1.10.0-py2.py3-none-any.whl
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=1.5.6 (from matplotlib)
  Downloading pyparsing-2.2.0-py2.py3-none-any.whl (56kB)
    100% |#####| 61kB 265kB/s
Collecting pytz (from matplotlib)
  Downloading pytz-2017.2-py2.py3-none-any.whl (484kB)
    100% |#####| 491kB 128kB/s
Collecting python-dateutil (from matplotlib)
  Downloading python_dateutil-2.6.1-py2.py3-none-any.whl (194kB)
    100% |#####| 194kB 64kB/s
Collecting cython>=0.10 (from matplotlib)
  Downloading cython-0.10.0-py2.py3-none-any.whl
Requirement already satisfied: numpy>=1.7.1 in c:\users\nicolas\appdata\local\programs\python\python36-32\lib\site-packages (from matplotlib)
Installing collected packages: six, pyparsing, pytz, python-dateutil, cython, matplotlib
Successfully installed cython-0.10.0 matplotlib-2.0.2 pyparsing-2.2.0 python-dateutil-2.6.1 pytz-2017.2 six-1.10.0
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>
```

Figura 8: Instalación módulo matplotlib.

5. PyMySQL 0.7.11:



```
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>pip install PyMySQL
Collecting PyMySQL
  Downloading PyMySQL-0.7.11-py2.py3-none-any.whl (78kB)
    100% |#####| 81kB 79kB/s
Installing collected packages: PyMySQL
Successfully installed PyMySQL-0.7.11
C:\Users\Nicolas\Desktop\Centro de energía\Metro_Solar\Modelo_Energia\Instalación>
```

Figura 9: Instalación módulo PyMySQL.

1.2.4. Otras alternativas de instalación

Otros medios para instalar los módulos necesarios para la ejecución del modelo de gestión de energía de Metro son:

1. Usar el gestor de módulos de algún IDE²:

² Por sus siglas en inglés “Integrated Development Environment”



Las IDE son aplicaciones especialmente diseñadas para el desarrollo de software. Existe una amplia gama de IDE, en este caso se recomienda usar PyCharm.

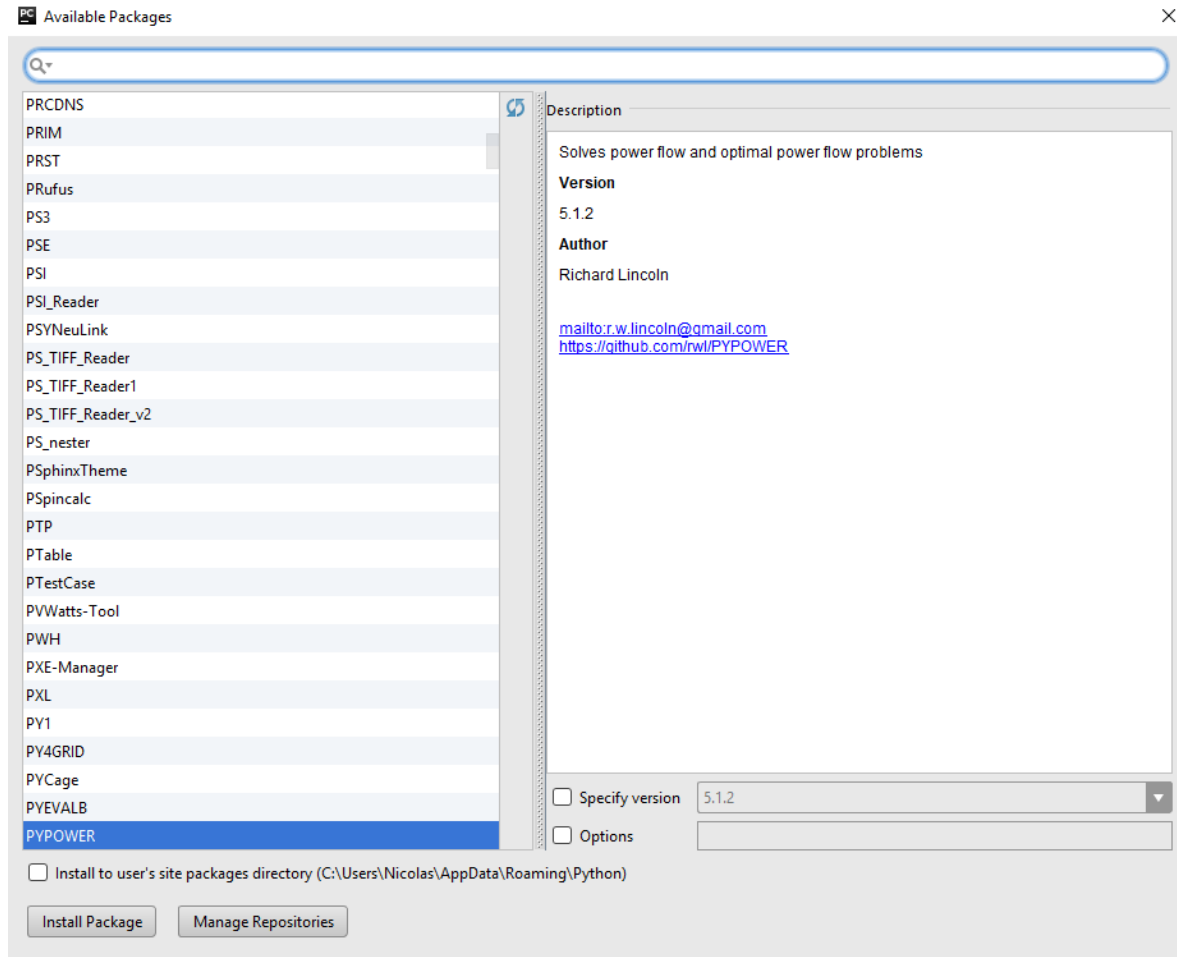


Figura 10: Gestor de módulos de PyCharm.

Al usar el gestor de módulos de Pycharm, este ejecuta las líneas de comando que se describieron en las secciones anteriores. Esto no siempre funciona y el programa muchas veces no puede resolver el problema por sí solo, como es el caso con el módulo de SciPy. Al intentar instalar SciPy usando pip con el comando “*pip install scipy*”, en Windows generalmente se va a generar un error dado que faltan librerías adicionales. Por esta razón en esta guía se sugiere usar un archivo Wheel que sí permite una correcta instalación de SciPy.

2. Descargar Anaconda:

Anaconda es una plataforma de Python que contiene por defecto una amplia gama de librerías para el desarrollo de software, entre ellas algunas como Numpy y SciPy. Adicionalmente Anaconda posee un medio de búsqueda de módulos para su instalación conocido como Conda. Si bien Anaconda simplifica



muchas tareas, entre ella la instalación de nuevos módulos, también posee algunas desventajas. La más significativa es que instala una distribución de Python de forma independiente y en su propio directorio. Esto puede traer complicaciones dado que otras aplicaciones que usan Python pueden no encontrar la distribución por encontrarse en un directorio diferente al usado por defecto.

Siguiendo los pasos descritos en las secciones anteriores se cumplen los requisitos necesarios para poder ejecutar el modelo de gestión de energía de Metro, para simular la operación de redes de tracción DC, redes de suministro en media tensión AC y un sistema conjunta de AC/DC.



Capítulo 2: Estructura de datos

El modelo de gestión de energía de Metro está diseñado para almacenar las características de las redes, los parámetros de simulación, los perfiles de generación y consumo de diferentes escenarios así como los resultados de las simulaciones bajo la estructura de tablas de datos. Estas tablas pueden encontrarse almacenadas en bases de datos como por ejemplo en un servidor MySQL, en archivos Excel o en formato csv. Este capítulo tiene por objetivo describir las estructuras de las diferentes tablas necesarias para realizar simulaciones.

En primer lugar se definen las tablas que contienen los atributos de todos los objetos que conforman una red DC, luego una red AC y finalmente aquellas necesarias para definir perfiles para realizar simulaciones.

2.1. Tablas para redes de tracción DC

2.1.1. Atributos líneas

Esta tabla almacena los nombres identificadores o ID de todas las líneas DC que se encuentran definidas en la base de datos, y que se desean simular. La tabla posee los siguientes campos:

Encabezado	Tipo	Unidad
Linea_ID	String	N/A
Vnom	Double	[V]

Tabla 2: Campos de datos tabla atributos líneas.

Adicionalmente se almacena el valor del voltaje operacional nominal de cada línea, el cual se usa como punto de partida para la resolución numérica de la operación de la línea como se define en secciones posteriores de este documento.

2.1.2. Atributos vías

Esta tabla almacena las IDs de las vías que conforman cada línea, así como sus propiedades físicas. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Via_ID	String	N/A
Linea_ID	String	N/A
Largo	Double	[m]
Resistividad	Double	[Ω /m]
Nrieles	Integer	N/A

Tabla 3: Campos de datos tabla atributos vías.



Cada línea puede estar definida por varias vías con trenes viajando en diferentes direcciones de forma simultánea. Dado que la potencia es suministrada a los motores mediante los rieles de cada vía, su largo, resistividad y número de rieles en paralelo define la matriz de admitancia del equivalente nodal de cada línea de tracción.

2.1.3. Lista elementos DC

En esta tabla se tiene un registro de todos los elementos que forman parte de cada línea DC. Entre estos elementos se encuentran trenes, SERs y módulos fotovoltaicos, cada cual identificado por un tipo que puede ser “Tren”, “SER” o “PVdc” respectivamente. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Elemento_ID	String	N/A
Linea_ID	String	N/A
Tipo	String	N/A
En_operacion	Boolean	N/A
Save	Boolean	N/A

Tabla 4: Campos de datos tabla lista elementos DC.

Para cada elemento se deben definir dos identificadores booleanos; uno para definir si se encuentra o no en operación, y otro para definir si se desean guardar los resultados de las simulaciones para ese elemento en particular. Dado que se pueden llegar a simular horizontes de tiempo largos, no se guardan por defecto todos los resultados de las simulaciones para todos los elementos.

2.1.4. Atributos trenes

Esta tabla almacena las IDs de los trenes de cada línea de tracción, la vía en la cual realizan su viaje y si se encuentran operacionales. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Tren_ID	String	N/A
Linea_ID	String	N/A
Via_ID	String	N/A
En_operacion	Boolean	N/A
Save	Boolean	N/A

Tabla 5: Campos de datos tabla atributos trenes.

Para cada tren se repiten los identificadores booleanos definidos en la sección 2.1.3.

2.1.5. Atributos SER

En esta tabla se identifican cada una de la subestaciones rectificadores que suministran la potencia necesaria para la operación de cada línea, a partir de su conexión con una red de suministro en media tensión AC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
SER_ID	String	N/A
Linea_ID	String	N/A
Via_ID	String	N/A
Term_ID	String	N/A
En_operacion	Boolean	N/A
PosVia	Double	[m]
Resistencia	Double	[Ω]
Vdc	Double	[V]
Vac	Double	[kV]
Save	Boolean	N/A

Tabla 6: Campos de datos tabla atributos SER.

Cada SER se representa como una fuente de voltaje que se conecta en una posición en cada vía que suministra respecto a su inicio, la cual se define por un voltaje en vacío denominado por Vdc, así como por una resistencia interna. La curva característica de una SER se muestra a continuación.

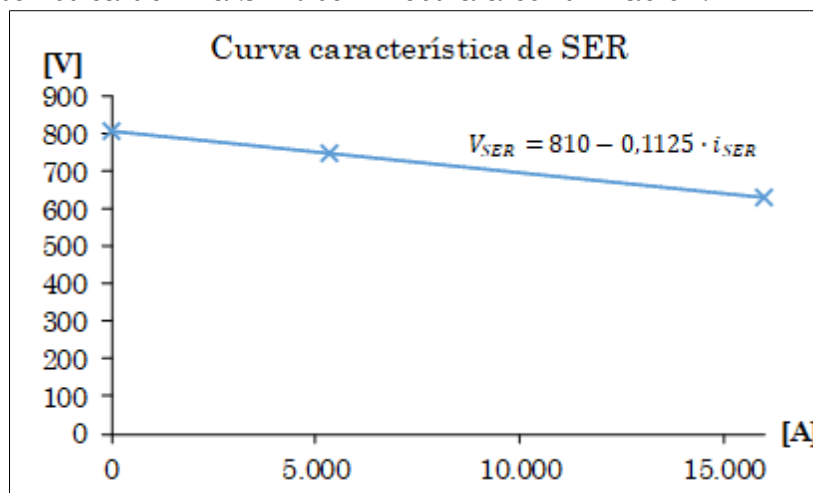


Figura 11: Curva característica SER.

Adicionalmente se define el identificador del terminal de conexión de cada SER a su red de suministro AC, así como su voltaje de operación nominal Vac en corriente alterna en el lado de baja tensión de su transformador de alimentación.

2.1.6. Atributos PVdc

En esta tabla se identifican cada uno de los módulos fotovoltaicos conectados en una línea de tracción DC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
PVdc_ID	String	N/A
Linea_ID	String	N/A
Via	String	N/A
PosVia	String	N/A
En_operacion	Boolean	N/A
PerfilPV	String	N/A
Save	Boolean	N/A

Tabla 7: Campos de datos tabla atributos PVDC.

Cada módulo fotovoltaico puede suministrar potencia a varias vías diferentes, para diferentes líneas. Las vías suministradas por cada módulo se especifican en el campo Via. El punto de conexión del módulo se especifica como su posición con respecto al inicio de cada vía. Cada módulo tiene un perfil de generación fotovoltaico, el cual se identifica mediante su ID en el campo PerfilPV. Cada perfil se define en la tabla *Perfiles pv* que se describirá en la sección sobre escenarios de simulación más adelante. También se especifica un identificador booleano para definir si se desean guardar resultados de simulación asociados con uno o varios módulos específicos.

2.2. Tablas para redes de suministro en media tensión AC

2.2.1. Atributos terminales

En esta tabla se identifican todos los terminales de conexión que forman una red de suministro en media tensión AC. Los terminales de conexión corresponden a los puntos de conexión de elementos como generadores (CDC o módulos fotovoltaicos) y consumos (SAF y SER). La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Term_ID	String	N/A
Red_ID	String	N/A
Vnom	Double	[kV]
Save	Boolean	N/A

Tabla 8: Campos de datos tabla atributos terminales.



Cada terminal se define como barra slack, PV o PQ para la solución del flujo de potencia con el módulo PYPOWER. Para cada terminal se define su voltaje operacional nominal que luego se usa como voltaje base en PYPOWER para resolver los flujos de potencia de la red. Finalmente se define un identificador booleano para especificar si se desean guardar los resultados de simulaciones asociados con uno o varios terminales de cada red AC.

2.2.2. Atributos trafos

En esta tabla se definen las características de cada transformador que conforma cada red de suministro en media tensión AC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Trafo_ID	String	N/A
Red_ID	String	N/A
En_operacion	Boolean	N/A
Snom	Double	[MVA]
HVTerm_ID	String	N/A
LVTerm_ID	String	N/A
Resistencia	Double	[p.u.]
Reactancia	Double	[p.u.]
Save	Boolean	N/A

Tabla 9: Campos de datos tabla atributos trafos.

En esta tabla se especifican los terminales de conexión para los lados de alta (HV) y baja (LV) tensión respectivamente. Adicionalmente se definen sus características físicas como potencia aparente nominal, resistencia y reactancia. Finalmente se define un identificador booleano para especificar si se desean guardar los resultados de simulaciones asociados con uno o varios transformadores de cada red AC.



2.2.3. Atributos SAF

En esta tabla se establecen las características operacionales de los de cada red de suministro en media tensión AC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
SAF_ID	String	N/A
Red_ID	String	N/A
Term_ID	String	N/A
En_operacion	Boolean	N/A
Vac	Double	[kV]
Snom	Double	[MVA]
Consumo_ID	String	N/A

Tabla 10: Campos de datos tabla atributos SAF.

Para cada consumo SAF se definen sus características operacionales tales como su terminal de conexión, su voltaje y potencia nominal. Cada SAF tiene un perfil de consumo de potencia activa y reactiva, los cuales se identifican mediante su ID en el campo Consumo_ID. Cada perfil se define en la tabla *Perfiles saf* que se describirá en la sección sobre escenarios de simulación más adelante.

2.2.4. Atributos CDC

En esta tabla se definen las características operacionales de cada centro de distribución de carga o CDC. Cada CDC representa el punto de conexión de la red de suministro en media tensión AC con la red de transmisión en alta tensión. Por esta razón cada CDC se define como el generador slack y es **único para cada red AC** para resolver los flujos de potencia usando el módulo de PYPOWER. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
CDC_ID	String	N/A
Red_ID	String	N/A
Term_ID	String	N/A
Vnom	Double	[kV]
Snom	Double	[MVA]
Save	Boolean	N/A

Tabla 11: Campos de datos tabla atributos CDC.

Para cada CDC se define su voltaje nominal en el punto de conexión con la red de transmisión, en este caso el lado de baja tensión del transformador de la subestación eléctrica respectiva. También se define la potencia nominal del CDC.



2.2.5. Atributos PVAC

En esta tabla se especifican los parámetros operacionales de los módulos fotovoltaicos conectados en las redes de suministro en media tensión AC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
PVac_ID	String	N/A
Red_ID	String	N/A
Term_ID	String	N/A
En_operacion	Boolean	N/A
Vnom	Double	[kV]
Snom	Double	[MVA]
Perfil_PV	String	N/A
cosphi	Double	N/A

Tabla 12: Campos de datos tabla atributos PVAC.

Para cada módulo fotovoltaico se definen sus características operacionales tales como su terminal de conexión en la red AC, su voltaje y potencia nominal así como también su factor de potencia. Al igual que para los módulos conectados en las redes de tracción DC, se define un perfil de generación fotovoltaico, el cual se identifica mediante su ID en el campo Perfil_PV.

2.2.6. Atributos cables

En esta tabla se definen las características de los cables de media tensión que conforman cada red AC. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Cable_ID	String	N/A
Red_ID	String	N/A
En_operacion	Boolean	N/A
Term_ID1	String	N/A
Term_ID2	String	N/A
Largo	Double	[m]
Snom	Double	[MVA]
Resistencia	Double	[p.u./m]
Reactancia	Double	[p.u./m]
Capacitancia	Double	[p.u./m]
Save	Boolean	N/A

Tabla 13: Campos de datos tabla atributos cables.



Para cada cable se definen los identificadores de los terminales que se interconectan. También se establecen sus características técnicas tales como su largo, resistencia, reactancia y capacitancia por unida de largo. Finalmente se define un identificador para el guardado de resultados de las simulaciones.

2.3. Tablas para escenarios de simulación

2.3.1. Bitácora de trenes

En esta tabla se definen los recorridos realizados por cada tren en operación en las diferentes vías y líneas. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Tren_ID	String	N/A
Linea_ID	String	N/A
Fecha	Datetime ³	N/A
Via	String	N/A
Posicion	Double	[m]
Velocidad	Double	[m/s]
Aceleracion	Double	[m/s ²]
Potencia	Double	[W]

Tabla 14: Campos de datos tabla bitácora trenes.

En esta tabla, para cada fecha se establecen las coordenadas de cada tren en su vía, así como su potencia con valor negativo si se consume durante la etapa de aceleración y de velocidad inercial, o bien positiva durante la etapa de frenado si es que el tren cuenta con frenos regenerativos.

2.3.2. Perfiles PV

En esta tabla se establecen diferentes perfiles de potencia fotovoltaica disponible para los escenarios de simulación. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Perfil_ID	String	N/A
Fecha	Datetime	N/A
P	Double	[W]

Tabla 15: Campos de datos tabla perfiles PV.

³ El formato del tipo datetime depende de cada base de datos, en este caso se usa “yyyy-mm-dd hh:mm:ss”



2.3.3. Perfiles SAF

De forma análoga a los perfiles fotovoltaicos, en esta tabla se definen diferentes perfiles de consumo de potencia activa y reactiva. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Consumol_ID	String	N/A
Fecha	Datetime	N/A
P	Double	[W]
Q	Double	[Var]

Tabla 16: Campos de datos tabla perfiles SAF.

2.4. Tablas de resultados

2.4.1. Resumen simulaciones DC

En esta tabla se guardan indicadores generales para cada línea de tracción DC luego de realizar simulaciones. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Sim_ID	String	N/A
E_SER	Double	[MWh]
E_Trenes	Double	[MWh]
E_Perdidas	Double	[MWh]
E_almacenable	Double	[MWh]

Tabla 17: Campos de datos tabla resumen simulaciones DC.

Entre los indicadores generales para las líneas de tracción se encuentra la energía total inyectada por los SER, la energía total consumida por los trenes, la energía total de pérdidas y la energía almacenable, la cual se define como los excedentes de potencia de toda la red DC.

2.4.2. Resultados elementos DC

En esta tabla se registran los resultados de las simulaciones para todos aquellos elementos DC con su identificador booleano *Save* definido como verdadero. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Sim_ID	String	N/A
Fecha	Datetime	N/A
Elemento_ID	String	N/A
Linea_ID	String	N/A
V	Double	[V]
P	Double	[W]

Tabla 18: Campos de datos tabla resultados elementos DC.

Para cada fecha o instante de tiempo simulado se guardan los valores de voltaje y potencia consumida o inyectada en el punto de conexión con la red de tracción DC.

2.4.3. Resultados terminales

En esta tabla se guardan los resultados de las simulaciones de los flujos de potencia, para todos aquellos terminales con su identificador booleano *Save* definido como verdadero. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Sim_ID	String	N/A
Fecha	Datetime	N/A
Term_ID	String	N/A
Red_ID	String	N/A
V	Double	[V]
delta	Double	[°]
P	Double	[MW]
Q	Double	[MVar]

Tabla 19: Campos de datos tabla resultados terminales.

2.4.4. Resultados branch

En esta tabla se guardan los resultados de las simulaciones de los flujos de potencia, para todos aquellos transformadores o cables con su identificador booleano *Save* definido como verdadero. En particular se guardan los flujos de entrada, las pérdidas de potencia y la cargabilidad del transformador o del cable respectivo para cada fecha o instante de tiempo simulado. La tabla posee los siguientes campos.

Encabezado	Tipo	Unidad
Sim_ID	String	N/A
Fecha	Datetime	N/A
Branch_ID	String	N/A
Red_ID	String	N/A
Pf	Double	[MW]
Qf	Double	[MVar]
Ploss	Double	[MW]
Qloss	Double	[MVar]
Loading	Double	[%]

Tabla 20: Campos de datos tabla resultados branch.

Capítulo 3: Estructura algoritmo

3.1. Diagrama de flujo general

De los cuatro módulos del modelo de gestión de energía de Metro, dos de ellos –*LeerDatos.py* y *GuardarDatos.py*– están orientados a comunicarse con la base de datos para leer y guardar atributos o resultados de simulaciones, mientras que los otros dos –*RedAC.py* y *RedLineas.py*– están destinados a modelar y simular las redes AC y DC respectivamente. A continuación se muestra un diagrama general de la ejecución secuencial del modelo de gestión de energía de Metro.

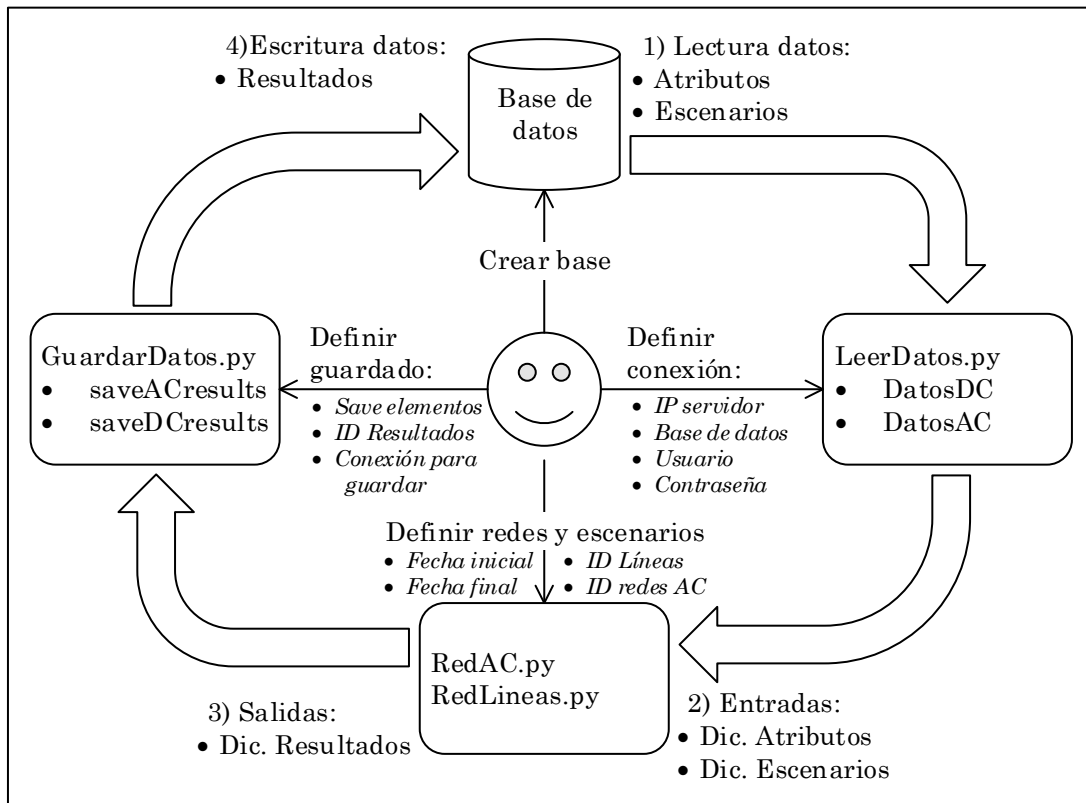


Figura 12: Ejecución secuencial general de modelo.

Los módulos de simulación se encuentran programados orientados al objeto, para brindar mayor flexibilidad y eficiencia para representar complejos sistemas tanto AC como DC. A continuación se muestra la estructura que siguen ambos módulos.

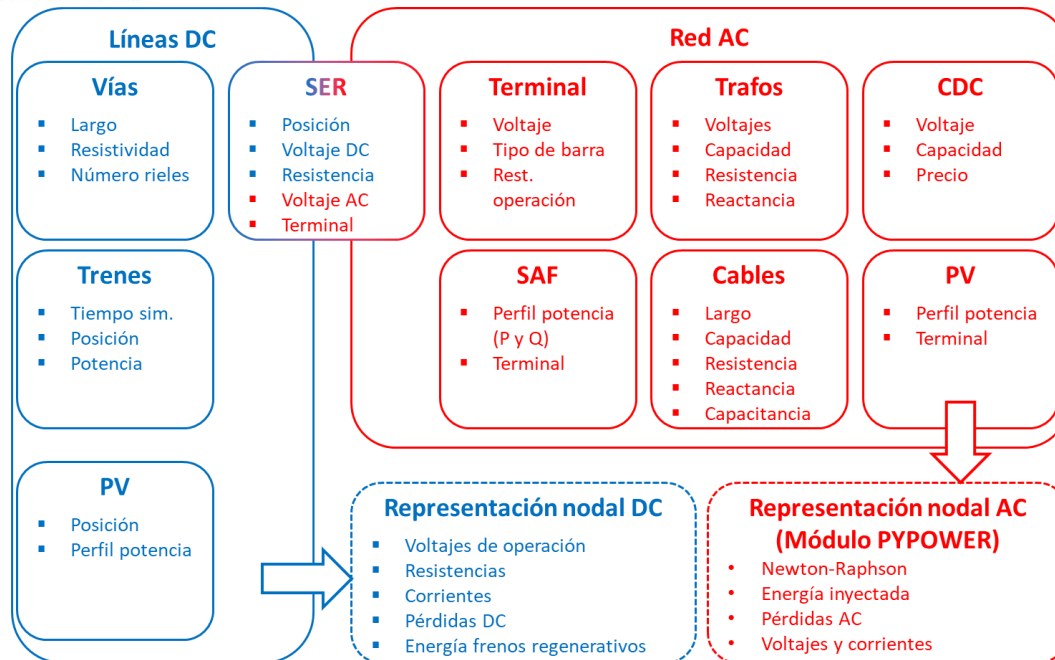


Figura 13: Estructura módulos LeerDatos.py y GuardarDatos.py.

3.2. Estructura módulo RedLineas

Como se puede ver en la Figura 13, cada objeto línea se compone de los objetos *Vías*, *SER*, *Trenes* y *PV*. A pesar de que cada uno de estos objetos se encuentra definido en su propia clase, y por lo tanto se pueden instanciar de forma independiente con sus respectivos atributos, la idea general del módulo es que estos objetos deben pertenecer a una línea. Por esta razón, la clase *Línea* tiene las funciones *Linea.addVia*, *Linea.addSER*, *Linea.addTren* y *Linea.addPV*. Estos métodos permiten instanciar cada objeto (el cual retornan) y adicionalmente agregarlos a su línea respectiva, para que queden definidos bajo la lógica y estructura necesarias para realizar simulaciones.

Adicionalmente, la clase *Línea* tiene las siguientes funciones.

3.2.1. *Linea.CargarDatos(Data)*

Esta función permite crear de forma rápida una línea DC completa instanciando todos sus sub elementos a través de las funciones mencionadas en la sección anterior, esto a partir de un diccionario Python *Data* que contiene toda la información necesaria, estructurada de una forma predeterminada como la que se muestra a continuación.

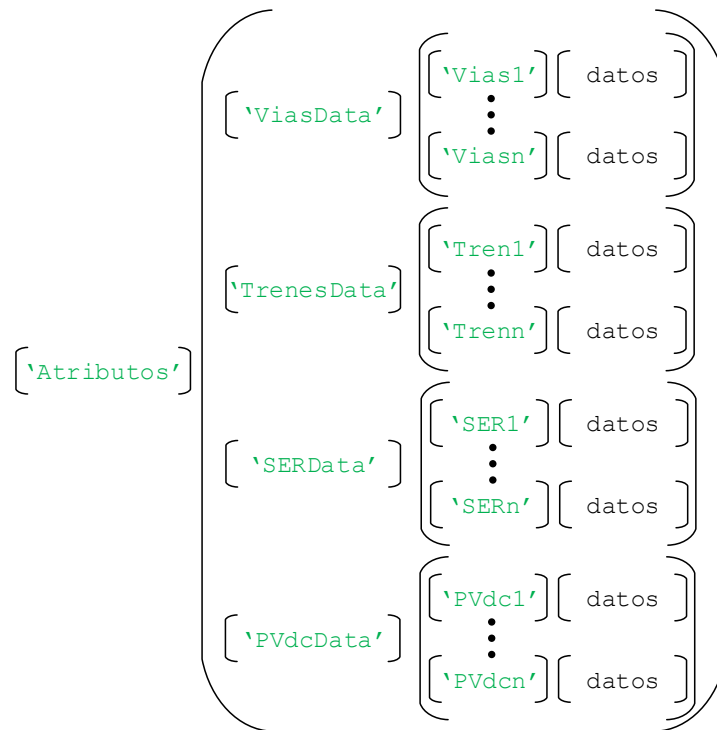


Figura 14: Diccionario de atributos para líneas de tracción DC.

3.2.2. *Linea.Reset*

Esta función permite borrar los resultados de simulaciones que se encuentran guardados en listas y diccionarios, tanto del objeto línea como de sus sub objetos. Esta función está orientada para realizar varias simulaciones consecutivas.

3.2.3. *Linea.DefinirSimulacion(SimData)*

De forma similar a la función *Linea.CargarDatos*, esta función permite cargar las bitácoras de los trenes, así como los perfiles de generación fotovoltaicos de los escenarios que se desean simular, para todos los objetos de la línea a partir de un diccionario Python *SimData*. La estructura de este diccionario se muestra a continuación.

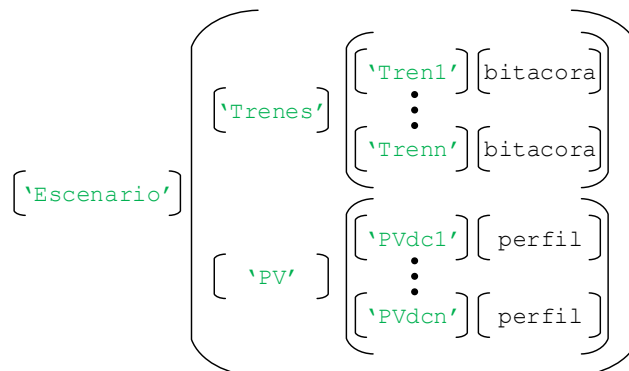


Figura 15: Diccionario de escenario para líneas de tracción DC.



3.2.4. *Linea.simular(t, *Niter, *tolerancia)*

Esta función permite resolver el estado de la red eléctrica de tracción DC en base a su representación nodal, así como de un método de resolución numérico que se detalla en una sección posterior de este documento. Para realizar la simulación se puede definir un número máximo de iteraciones así como una tolerancia de error numérico. Si estos argumentos no se especifican entonces se considera un máximo de 100 iteraciones y un error de 0,01 en los voltajes de la red por defecto.

3.2.5. *Linea.plotresults*

Esta función permite graficar los resultados obtenidos de las simulaciones usando el módulo de *matplotlib*.

3.2.6. *Linea.saveresults*

Esta función permite guardar los resultados obtenidos de las simulaciones en un diccionario Python *Results* que es un atributo de cada línea. En este diccionario se guardan resultados generales de las simulaciones tales como energía total inyectada por las SER, energía total consumida por los trenes, energía total de pérdidas y energía total almacenable. Además, se guardan los resultados de cada elemento de la red de tracción DC con su indicador *Save* en definido como verdadero. A continuación se muestra la estructura de este diccionario de resultados.

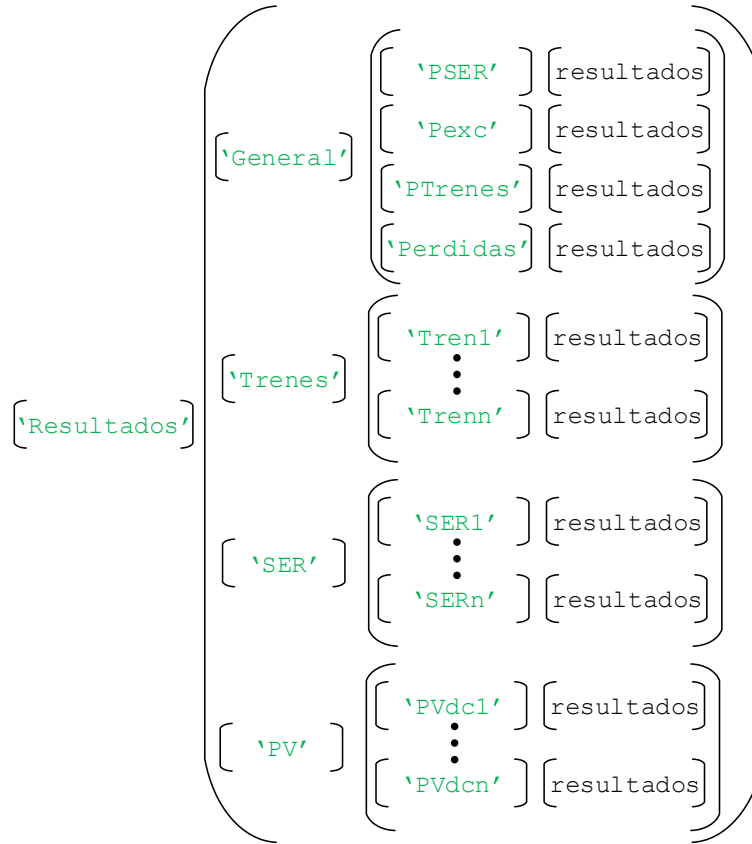


Figura 16: Estructura diccionario de resultados líneas de tracción DC.

3.3.Estructura módulo RedAC

De forma análoga al módulo *RedLineas.py*, cada objeto RedAC se compone de los objetos *Terminal*, *Trafo*, *CDC*, *SAF*, *Cabels* y *PV*. A pesar de que cada uno de estos objetos se encuentra definido en su propia clase, y por lo tanto se pueden instanciar de forma independiente con sus respectivos atributos, la idea general del módulo es que estos objetos deben pertenecer a una RedAC. Por esta razón, la clase *RedAC* tiene las funciones *RedAC.addVTerm*, *RedAC.addTrafo*, *RedAC.addCDC*, *RedAC.addSAF*, *RedAC.addCable* y *RedAC.addPVAC*. Estos métodos permiten instanciar cada objeto (el cual retornan) y adicionalmente agregarlos a su RedAC respectiva, para que queden definidos bajo la lógica y estructura necesarias para realizar simulaciones.

Adicionalmente, la clase RedAC tiene las siguientes funciones.

3.3.1. *RedAC.CargarDatos(Data)*

Equivalente al método descrito en la sección 3.2.1, salvo por la estructura del diccionario Python de datos de entrada que se muestra a continuación.

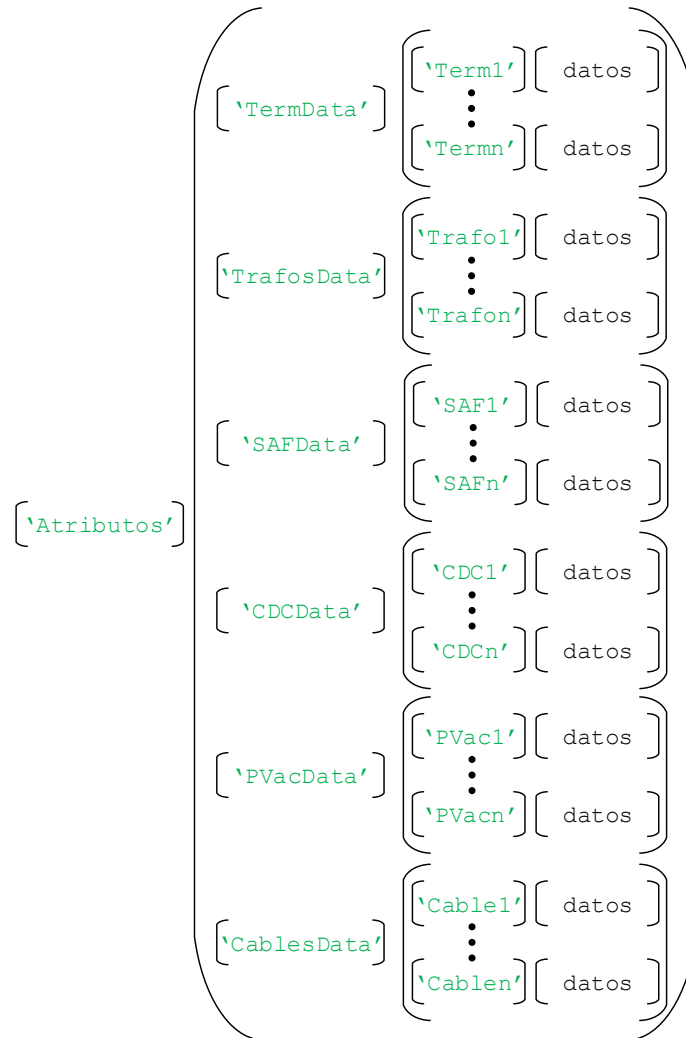


Figura 17: Diccionario de atributos para para red de suministro en media tensión AC.

3.3.2. *RedAC.Reset*

Equivalente al método descrito en la sección 3.2.2.

3.3.3. *RedAC.DefinirSimulacion(SimData)*

Equivalente al método descrito en la sección 3.2.3, salvo por la estructura del diccionario Python de datos de escenarios que se muestra a continuación.

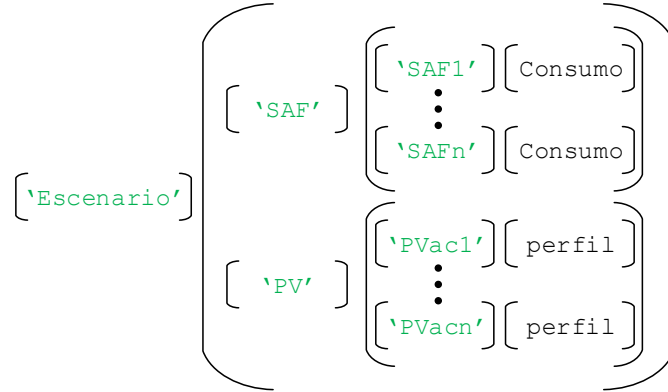


Figura 18: Diccionario de escenario para red de suministro en media tensión AC.

3.3.4. *RedAC.simular(Sbase)*

Esta función permite resolver el flujo de potencia de la red de suministro en media tensión AC usando el módulo de PYPOWER considerando una potencia base. Este método primero resuelve el estado de todas las líneas de tracción DC conectadas a la red de suministro en media tensión respectiva, usando el método *Linea.simular(t, *Niter, *tolerancia)* descrito en la sección 3.2.4. Una vez hecho esto, cada SER se representa como un consumo de potencia activa puro conectado en su terminal respectivo, y luego se procede a resolver el flujo de potencia de la red de suministro AC usando la función *RedAC.FlujosAC* que se describe en la siguiente sección. Este proceso se resuelve para todo el vector de tiempo definido en el escenario de simulación, que queda definido como un atributo de la *RedAC.t*. En este proceso se asume que el vector de tiempo de la RedAC tiene un muestreo igual o menor al de las líneas DC (*Linea.t*), dado que las bitácoras de los trenes pueden definirse con gran precisión. Por esta razón, si los vectores de tiempo de simulación son diferentes entre una RedAC y sus líneas de tracción DC, en ese caso los perfiles PV y los consumos SAF se interpolan para coincidir con el vector de tiempo de las líneas de tracción DC. La interpolación se realiza usando el método *interp()* del módulo *numpy*.

3.3.5. *RedAC.FlujosAC(Sbase, t)*

Esta función permite resolver únicamente los flujos de la RedAC para un tiempo dado del escenario de simulación definido. En este caso no se resuelven primero los estados de las líneas de tracción DC conectadas a la red de suministro de media tensión AC.

3.3.6. *RedAC.addLinea(Linea)*

Esta función permite conectar una línea de tracción DC a una red de suministro en media tensión AC. Esto conecta los SER de la línea a su respectivo



terminal en la RedAC para luego ser considerado como un consumo equivalente de potencia activa.

3.3.7. *RedAC.plotresults*

Equivalente al método descrito en la sección 3.2.5

3.3.8. *RedAC.saveresults*

Equivalente al método descrito en la sección 3.2.6, salvo por la estructura del diccionario de resultados *Results* que se muestra a continuación.

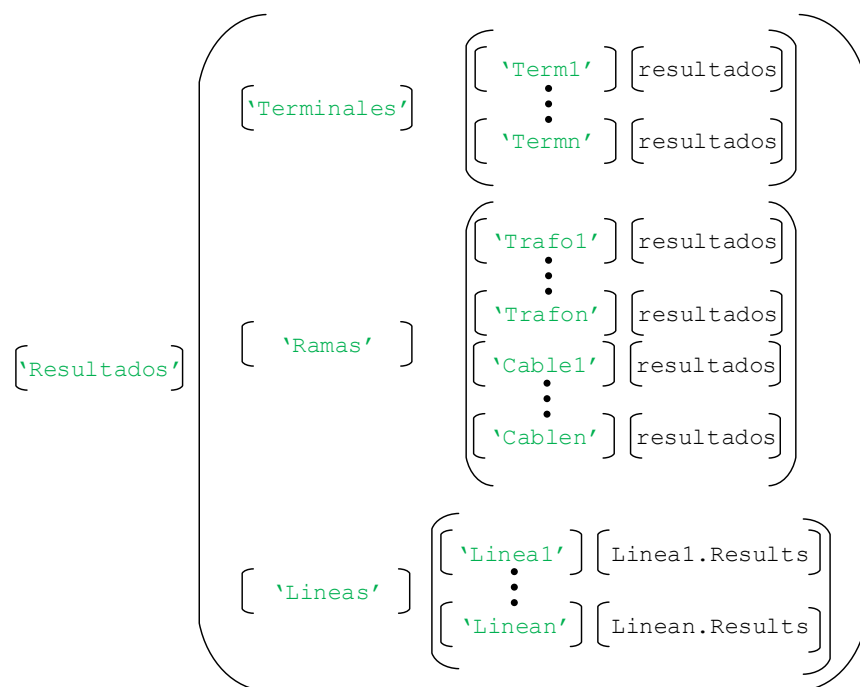


Figura 19: Estructura diccionario de resultados para red de suministro en media tensión AC.

En el caso en que existan líneas de tracción DC conectadas a la Red AC y que tengan resultados de simulación, estos también se guardan el diccionario de resultados de la Red AC respectiva.

Capítulo 4: Fundamentos teóricos

A continuación se detalla el método de resolución numérico de las líneas de tracción DC. En primer lugar se detalla la representación nodal de la línea de tracción con sus componentes, y como esta se actualiza conforme cambian las posiciones de los trenes. Luego se procede a detallar el procedimiento iterativo que se emplea para resolver el estado de la red eléctrica nodal.

4.1. Representación nodal de línea de tracción DC

Para realizar la representación nodal de una línea de tracción DC se define una clase *Nodo*. Un *Nodo* se define como un conjunto de elementos de la línea de tracción DC que, para un mismo tiempo, en una misma posición de la línea, comparten un conjunto de vías no vacío. A continuación se muestra un ejemplo de diferentes configuraciones de nodos.

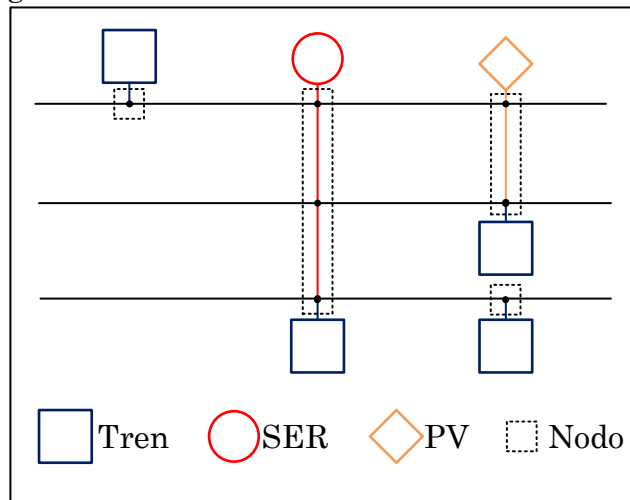


Figura 20: Representación nodal de línea de tracción DC.

De esta forma, cada vez que un tren se desplaza en la línea cambiando su posición, basta verificar si en esa posición se encuentra un nodo, y si el elemento en cuestión comparte alguna de las vías del conjunto de vías de ese nodo. De cumplir esta condición el elemento pasa a formar parte del nodo encontrado, de lo contrario permanece en su nodo actual en una nueva posición en la línea.

Luego, es posible construir la matriz de admitancia de la red eléctrica calculando la distancia entre nodos adyacentes en cada una de las diferentes vías. Para esto, se mantiene una lista para cada vía de sus nodos activos (con al menos un elemento en él) junto con su respectiva posición, la cual es uno de los atributos del nodo (*Nodo.pos*). La lista luego se puede ordenar en función de la posición de los nodos. Finalmente, en cada nodo se recorren su conjunto de vías, buscando en la lista de nodos de cada vía el nodo siguiente, para calcular la diferencia entre las posiciones de estos para obtener el largo del tramo de vía que

los conecta. Luego la admitancia se puede calcular como el inverso del largo por la resistividad de los rieles.

4.2. Solución numérica de red nodal

Una vez que se tiene la representación nodal de la línea de tracción DC, se deben cumplir las ecuaciones nodales de Kirchhoff, así como la de potencia inyectada por los módulos PV –dada su potencia disponible– y el consumo de los trenes, es decir:

$$P_{pv} = V_{pv} \cdot I_{pv} \quad (1)$$

$$P_{tren} = V_{tren} \cdot I_{tren} \quad (2)$$

$$\sum_{i \in N} I_i = 0 \quad (3)$$

De forma más general, estas ecuaciones se pueden combinar en un sistema de ecuaciones no lineales como se muestra en el ejemplo a continuación.

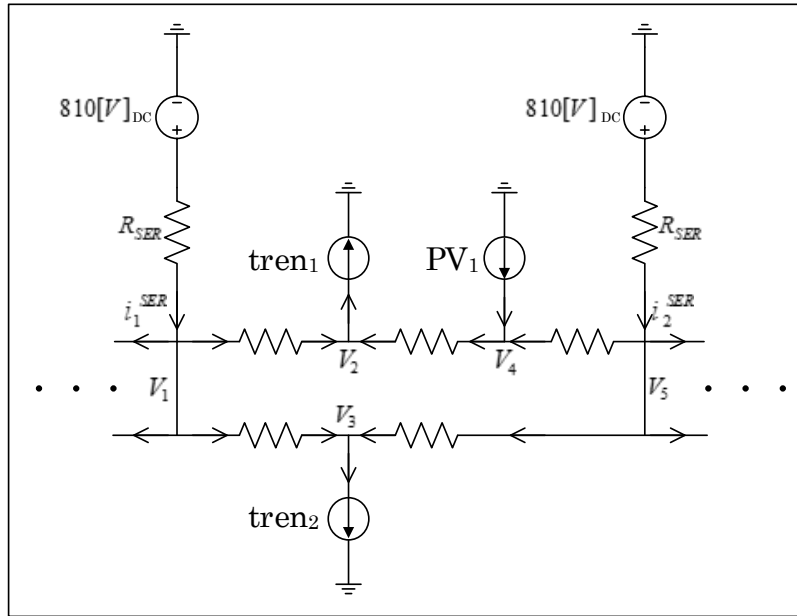


Figura 21: Ejemplo solución de representación nodal.

El sistema anterior queda definido por el siguiente sistema de ecuaciones:



CENTRO DE ENERGÍA

$$\begin{bmatrix} \frac{1}{R_{12}} + \frac{1}{R_{13}} + \frac{1}{R_{SER}} & -\frac{1}{R_{12}} & -\frac{1}{R_{13}} & 0 & 0 \\ -\frac{1}{R_{12}} & \frac{1}{R_{12}} + \frac{1}{R_{24}} & 0 & -\frac{1}{R_{24}} & 0 \\ -\frac{1}{R_{13}} & 0 & \frac{1}{R_{13}} + \frac{1}{R_{35}} & 0 & -\frac{1}{R_{35}} \\ 0 & -\frac{1}{R_{24}} & 0 & \frac{1}{R_{24}} + \frac{1}{R_{45}} & -\frac{1}{R_{45}} \\ 0 & 0 & -\frac{1}{R_{35}} & -\frac{1}{R_{45}} & \frac{1}{R_{35}} + \frac{1}{R_{45}} + \frac{1}{R_{SER}} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{bmatrix} = \begin{bmatrix} \frac{810}{R_{SER}} \\ 0 \\ 0 \\ 0 \\ \frac{810}{R_{SER}} \end{bmatrix} + \begin{bmatrix} 0 \\ -i_{t_1} \\ -i_{t_2} \\ i_{pv_1} \\ 0 \end{bmatrix} \quad (4)$$

Donde:

$$i_{t_1} = \frac{P_{t_1}}{V_{t_1}}, i_{t_2} = \frac{P_{t_2}}{V_{t_2}} \text{ y } i_{pv_1} = \frac{P_{pv_1}}{V_{pv_1}}$$

El sistema anterior se puede resolver usando Newton-Raphson, o bien, de forma iterativa. En este caso, dado que se conoce el voltaje operacional de la línea de tracción, se puede iniciar el método iterativo desde ese valor y converger a una buena solución en pocas iteraciones. A continuación se muestra un ejemplo de la solución numérica con este método iterativo de forma gráfica.

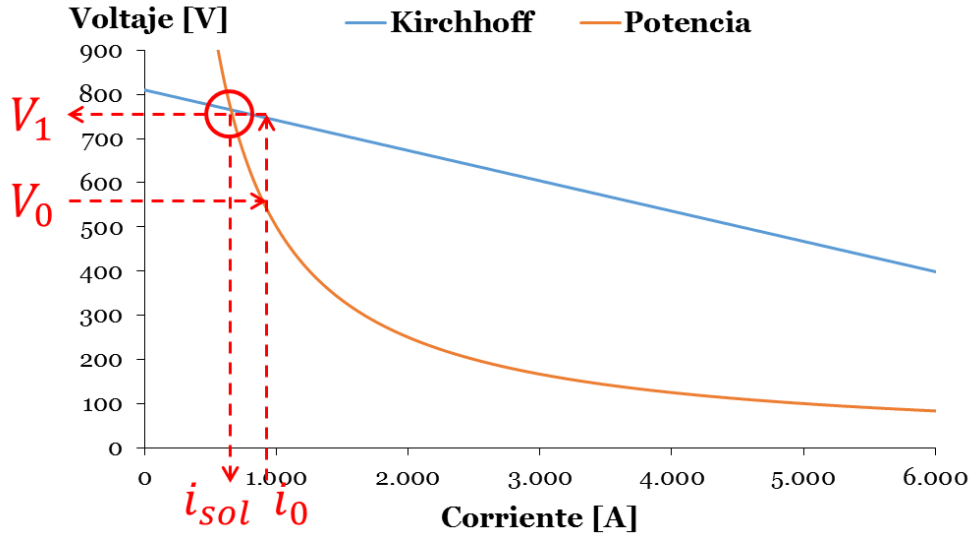


Figura 22: Solución iterativa de representación nodal de línea de tracción DC.