UNIVERSIDAD DE OVIEDO

Escuela de
Ingeniería de Minas, Energía y Materiales de Oviedo

Máster en Ingeniería de Minas

Trabajo Fin de Máster

Aplicación en Tiempo Real de Técnicas de Estimación de Estado en Redes de Distribución

Autor: Francisco Pello de la Roz

Tutor: José Manuel Cano Rodríguez

Co-Tutor: Carlos Hiram Rojas García

Oviedo, julio de 2018

Índice de la Memoria

Índice de las Ilustraciones

Índice de las Tablas

1. Introducción y Objetivo

Actualmente el uso de la **Energía Eléctrica** resulta imprescindible en nuestras vidas puesto que es muy accesible y sencilla de utilizar.

Esta energía nos facilita nuestro día a día debido a que la amplia mayoría de los objetos que nos rodean, tanto en el ámbito doméstico (luz, electrodomésticos, televisión, aire acondicionado…) como en el ámbito industrial y comercial, funcionan gracias a ella.

Podríamos decir que actualmente la energía eléctrica puede ser considerada como un bien de consumo esencial. Además presenta unas características que la convierten en un bien peculiar. En primer lugar, la electricidad ha de producirse y transportarse en el mismo instante en que es consumida. Ésta es la característica más destacable a la hora de configurar, planificar, organizar y gestionar los sistemas de energía eléctrica. Otro aspecto peculiar de la electricidad es que para su transporte se han de emplear líneas eléctricas u otro tipo de instalaciones que han de cumplir unas leyes físicas fundamentales (*Leyes de Kirchhoff*) las cuales provocan que exista una estrecha relación entre las distintas vías de transporte de energía eléctrica, de manera que cualquier perturbación en estas líneas o equipos de transporte provoca inmediatamente efectos negativos en el suministro de la electricidad.

El suministro de energía eléctrica desde los centros de producción hasta los usuarios finales se lleva a cabo mediante el *Sistema de Suministro Eléctrico*. A este sistema están conectadas las centrales de producción de energía eléctrica, las cuales pueden ser *hidroeléctricas, nucleares y térmicas.* Estas centrales generan un sistema trifásico sinusoidal de tensiones, con una frecuencia (50Hz en Europa y gran parte de América del Sur y 60Hz en América del Norte y Central y Brasil) y una amplitud de onda estrictamente estandarizada y controlada.

Posteriormente para poder transportar esta energía se han de emplear unos transformadores que eleven la tensión a la que es generada (entre 6 y 20 kV), a una tensión más adecuada para su transporte. Habitualmente estas centrales de producción se encuentran alejadas de los núcleos de consumo por lo que se requiere de la utilización de una *Red de Transporte.* Dicha red está formada por los siguientes elementos:

- Subestación de Distribución: conjunto de elementos (transformadores, interruptores, seccionadores…) cuya misión es reducir los niveles de alta tensión de las líneas de transmisión hasta niveles de media tensión para su ramificación en múltiples salidas.
- Circuito Primario.
- Circuito Secundario.

A la red de transporte se conectan directamente algunas grandes instalaciones de consumo, y también los transformadores, que se encargan de reducir las altas tensiones de producción a tensiones adecuadas (132kV, 66kV, 45kV), para su distribución a otros grandes consumidores, o a otras subestaciones transformadoras de carga inferior.

Esta red de carga inferior, denominada *Red de Distribución,* se configura de manera distinta a la red de transporte, y es la encargada de suministrar la energía eléctrica a los lugares más recónditos desde donde es demandada. A esta red de carga inferior se conectan instalaciones generadoras de menor envergadura que las conectadas a la red de transporte como son: *generadores eólicos, instalaciones industriales con cogeneración, pequeñas centrales hidráulicas, pequeñas centrales térmicas* que utilizan biomasa u otros residuos como combustibles, *centrales solares* y en menor medida *pequeños generadores* que utilizan combustibles fósiles e *instalaciones fotovoltaicas*.

Desde las subestaciones transformadoras de esta red parte una red de media tensión (20kV, 15kV, 6,6kV) cuyas tensiones se aproximan a las tensiones finales de consumo.

Finalmente, desde esta red se procede a disminuir de nuevo la tensión (en centros de transformación) para alimentar en baja tensión (400V, 230V) a los consumidores domésticos, industriales, comerciales, etc.

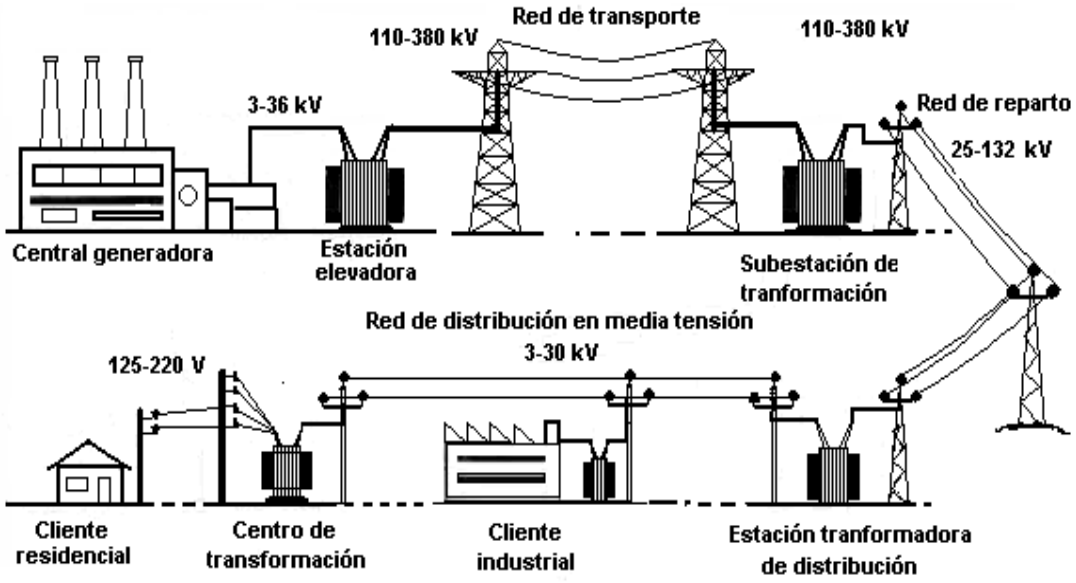

Ilustración 1. Diagrama esquematizado del Sistema de suministro eléctrico. Fuente [1]

El objetivo de este ***Trabajo Fin de Máster*** será la implementación del algoritmo de estimación de estado de una Red de Distribución Eléctrica mediante el método ***WLS*.**

La finalidad de lo que se conoce como estimación de estado es estimar las tensiones complejas en todos los nodos eléctricos del sistema. Esto se logra procesando las medidas disponibles y la información sobre el estado de los interruptores, seccionadores y tomas de transformadores, así como los parámetros de líneas, transformadores, bancos de condensadores y reactancias.

Para ello se procederá al modelado de la red, empleando el sistema por unidad, e incluyendo un método de resolución que permite representar este tipo de elementos (**Direct Approach**).

De cara a implementar el algoritmo de estimación de estado ***WLS***, el funcionamiento de la red eléctrica será evaluado inicialmente a partir de la generación de medidas sintéticas con errores generados de forma estadística. A continuación, el algoritmo desarrollado será probado en una red de distribución sintética en la que la medida de tensión y potencia de alguno de los nodos será obtenida desde dispositivos reales. De esta forma se podrá garantizar la capacidad de la técnica de estimación de estado implementada para funcionar en tiempo real.

2. Descripción de la Red Eléctrica Española

Red Eléctrica de España es un grupo empresarial fundado en 1985 que actúa como operador del sistema eléctrico en el mercado eléctrico español. Se encarga de asegurar el correcto funcionamiento del sistema de suministro eléctrico y garantiza la continuidad y seguridad del suministro de energía eléctrica.

Su función como operador es establecer las previsiones de la demanda de energía eléctrica y operar en tiempo real las instalaciones de generación y transporte eléctrico, para que la producción programada en las centrales eléctricas coincida en cada instante con la demanda de los consumidores.

La Red de Transporte está compuesta por más de **40 000** kilómetros de líneas de alta tensión, más de **5 000** subestaciones y tiene una capacidad de transformación de más de **80 000** MVA.

Ilustración 2. Esquema Distribución Eléctrica en España. Fuente [2]

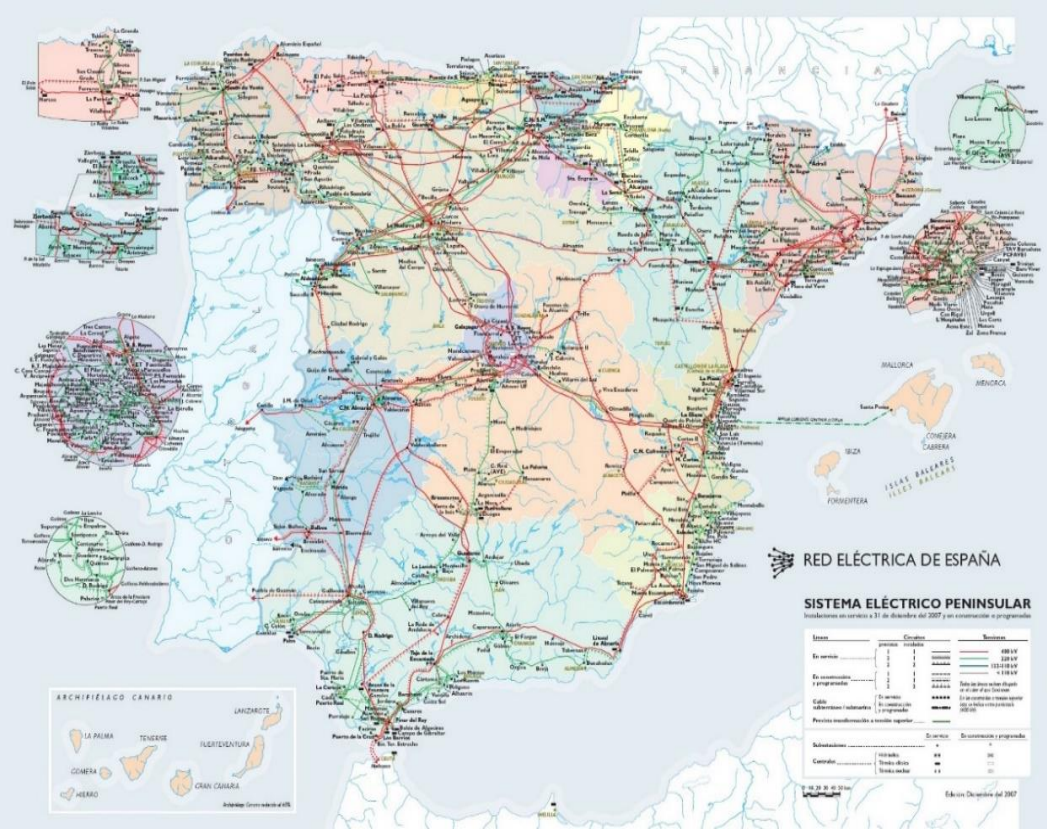

Ilustración 3. Sistema Eléctrico Peninsular. Fuente [2]

Km de circuito	*2011*	*2012*	*2013*	*2014*	*2015*	*2016*	*2017*
400kV	19.671	20.109	20.639	21.094	21.179	21.619	21.728
220kV	18.410	18.779	19.053	19.192	19.387	19.479	19.507
150 - 132 - 110kV	272	272	272	272	398	523	523
< 110kV	2.011	2.014	2.014	2.014	2.022	2.025	2.035
Total	**40.364**	**41.174**	**41.978**	**42.572**	**42.986**	**43.646**	**43.793**

Tabla 1. Red de Transporte Peninsular y no Peninsular. Fuente [2]

Número de posiciones	*2011*	*2012*	*2013*	*2014*	*2015*	*2016*	*2017*
400kV	1.253	1.319	1.374	1.394	1.441	1.458	1.484
220kV	2.813	2.936	3.026	3.077	3.124	3.152	3.180
150 - 132 - 110kV	52	52	52	52	84	84	110
< 110 kV	743	743	745	769	779	797	827
Total	**4.861**	**5.050**	**5.197**	**5.292**	**5.428**	**5.491**	**5.601**

Tabla 2. Posiciones de subestaciones peninsulares y no peninsulares. Fuente [2]

Potencia (MVA)	*2011*	*2012*	*2013*	*2014*	*2015*	*2016*	*2017*
Total	**72.869**	**78.629**	**81.289**	**83.939**	**84.544**	**85.444**	**86.654**

Tabla 3. Capacidad de transformación peninsular y no peninsular. Fuente [2]

3. Modelado de las Redes de Distribución

El sistema de transmisión de energía eléctrica requiere de un modelado adecuado que depende de parámetros como la distancia y la frecuencia.

Las redes de distribución eléctrica constituyen el elemento más significativo de un sistema eléctrico de potencia. Si éstas tuviesen una distancia mínima se podría considerar que los conductores de la misma serían ideales, pero puesto que las redes de distribución se construyen con el fin de transportar energía entre las subestaciones y el consumidor, la realidad es otra.

En el modelado de las redes de distribución influyen 4 fenómenos físicos que dependen de la distancia y del voltaje de operación. Estos fenómenos físicos son:

- ***Efecto resistivo:*** es el responsable del calentamiento y la caída de tensión de los conductores. Este efecto es dominante sobre los demás en las redes de distribución de baja tensión.
- ***Efecto inductivo:*** se debe a los enlaces de flujo que rodean al conductor, creados por su propia corriente y por las corrientes de otros conductores. Este efecto no se suele tener en cuenta en las redes de distribución de baja tensión, puesto que el efecto resistivo es mayor que el inductivo. Se empieza a tener en cuenta cuando ambos efectos presentan valores comparables, cosa que sucede en las líneas de distribución. Y finalmente en las líneas de alta tensión el efecto inductivo limita las transferencias de potencia activa.
- ***Efecto capacitivo:*** se debe a las corrientes de desplazamiento en derivación que aparecen entre conductores, y entre conductores y el suelo. Estas corrientes hacen que los conductores se carguen, incluso cuando las líneas se encuentran en vacío. El efecto capacitivo provoca un aumento de la tensión en el extremo de la carga en vacío, que depende de la longitud de la línea. Para redes de menos de 80 km, este efecto supone un 0,5% por lo que se puede considerar despreciable, y sólo se empieza a tener en cuanta cuando las redes de distribución superan los 80 km. Sin embargo, este efecto se ha de tener en cuenta cuando se trabajamos con cables aislados debido a que las consideraciones de la longitud ya no nos resultan válidas.

- ***Efecto conductivo:*** es provocado por las corrientes de fuga que aparecen por las características del aislamiento de la red. Estas corrientes aparecen a causa de la contaminación del medio ambiente que rodea a la red de distribución. Normalmente no se tiene en cuenta a la hora de representar la línea en su normal funcionamiento, sin embargo se ha de tener en cuenta en las líneas de alta tensión, puesto que a la hora de seleccionar los conductores para las mismas, este efecto conductivo provoca pérdidas de potencia activa.

Una vez estudiados estos cuatro efectos se puede llevar a cabo el diseño de la red de distribución eléctrica. Para llevar a cabo el modelado de la misma los elementos básicos que se han de tener en cuenta son: líneas eléctricas, transformadores de potencia, máquinas síncronas, máquinas asíncronas o de inducción y consumos.

3.1 Valores por Unidad

Para llevar a cabo el modelado realizaremos una normalización de todos los factores que interviene en el cálculo del mismo (tensión, potencias, corrientes) convirtiéndolos en valores por unidad (**p.u**). Dicha normalización se establece a través del cociente que relaciona la variable a estudiar y un valor que se define como base. Los valores base se establecen en ***notación fasorial*** y se corresponden con los valores de módulo de las magnitudes eléctricas básicas (tensión eficaz, intensidad o corriente eficaz, potencia aparente, módulo de la impedancia y módulo de la admitancia). De esta manera resulta mucho más sencilla su utilización a la hora de realizar cálculos, ya que todos los valores utilizados se encuentran en torno a la unidad **[3]**.

Las operaciones realizadas para llevar a cabo las transformaciones de todos los parámetros a valores **p.u** son las siguientes **[3]**:

- En primer lugar se ha de establecer un binomio de valores base, potencia aparente o compleja (**S**) y tensión (**U**), potencia aparente o compleja (**S**) e intensidad de corriente (**I**), tensión (**U**) e intensidad de corriente (**I**), que han de cumplir las ecuaciones de un sistema monofásico de energía eléctrica:

$$S_{base} = U_{base} \times I_{base}$$
$$U_{base} = Z_{base} \times I_{base}$$
$$I_{base} = Y_{base} \times U_{base}$$

Por lo general, en el análisis de las Redes Eléctricas se establecen como parámetros base la potencia aparente o compleja (**S_{base}**) y la tensión (**U_{base}**). A partir de estos dos parámetros podemos obtener una impedancia base (**Z_{base}**) y una intensidad de corriente base **(I_{base})** a través de las siguientes transformaciones:

$$Z_{base} = \frac{U_{base}^2}{S_{base}}$$

$$I_{base} = \frac{S_{base}}{U_{base}}$$

- Una vez obtenidos estos parámetros estamos en disposición de transformar todos los valores del circuito a valores **p.u**. Para calcular estos valores simplemente basta con dividir el valor del parámetro (impedancia, admitancia, reactancia) o de la variable (intensidad de corriente, tensión, potencia) por su respectiva base. Las transformaciones para obtener las principales variables son las siguientes :

$$U_{pu} = \frac{U}{U_{base}}$$

$$I_{pu} = \frac{I}{I_{base}}$$

$$P_{pu} = \frac{P}{S_{base}}$$

$$Q_{pu} = \frac{Q}{S_{base}}$$

$$Z_{pu} = \frac{Z}{Z_{base}}$$

Como resulta evidente, los resultados de los cálculos obtenidos estarán en valores por unidad, para obtener los resultados en las unidades habituales simplemente habrá que realizar la operación contraria.

La utilización de los valores **p.u** ayuda a simplificar los cálculos en los sistemas eléctricos de potencia, principalmente a la hora de realizar estos cálculos sobre los transformadores, ya que si se eligen los valores adecuados para la base, desaparece la relación de transformación (**r_t**) entre los devanados del transformador. A continuación se muestra un ejemplo de cómo se llevarán a cabo estos cálculos:

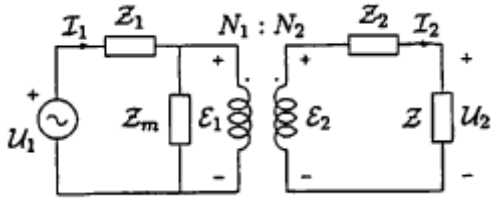

Ilustración 4. Representación circuito monofásico. Fuente [3]

En la *Ilustración 4* se muestra un circuito monofásico compuesto por una carga **Z**, alimentada por una fuente de tensión (**U_1**), a través de un transformador ideal con relación de transformación **N_1/N_2**, que además consta de dos impedancias en serie (**Z_1** y **Z_2**), y de una impedancia en paralelo (**Z_m**).

Aplicando la ***Ley de Tensiones de Kirchhoff*** obtenemos las siguientes ecuaciones para el transformador:

$$U_1 = I_1 \cdot Z_1 + \varepsilon_1$$

$$\varepsilon_2 = I_2 \cdot Z_2 + U_2$$

donde:

- $\varepsilon_1 \rightarrow$ Tensión en la bobina del devanado primario
- $\varepsilon_2 \rightarrow$ Tensión en la bobina del devanado secundario

Si utilizamos como valores base la potencia aparente (**S_{base}**) para el devanado primario, y la tensión (**U_{base2}**) para el devanado secundario, obtenemos la siguiente relación:

$$\frac{U_{base1}}{U_{base2}} = \frac{E_1}{E_2} = \frac{N_1}{N_2} = r_t$$

donde:

- N_1 → Número de espiras del devanado primario.
- N_2 → Número de espiras del devanado secundario.

Calculamos las intensidades base:

$$I_{base1} = \frac{S_{base}}{U_{base1}}$$

$$I_{base2} = \frac{S_{base}}{U_{base2}}$$

$$\frac{I_{base1}}{I_{base2}} = \frac{U_{base2}}{U_{base1}} = \frac{1}{r_t}$$

Y las impedancias base:

$$Z_{base1} = \frac{U_{base1}^2}{S_{base}}$$

$$Z_{base2} = \frac{U_{base2}^2}{S_{base}}$$

$$\frac{Z_{base1}}{Z_{base2}} = \frac{U_{base2}^2}{U_{base1}^2} = r_t^2$$

Si sustituimos estas ecuaciones en las ecuaciones obtenidas por la *Ley de Tensiones de Kirchhoff* para el transformador, obtenemos lo siguiente:

$$\frac{U_1}{U_{base1}} = \frac{I_1}{I_{base1}} \cdot \frac{Z_1}{Z_{base1}} + \frac{\varepsilon_1}{U_{base1}}$$

$$\frac{\varepsilon_2}{U_{base2}} = \frac{I_2}{I_{base2}} \cdot \frac{Z_2}{Z_{base2}} + \frac{U_2}{U_{base2}}$$

Finalmente obtenemos la siguiente expresión en valores **p.u**:

$$U_{1p.u} = I_{1p.u} \cdot Z_{1p.u} + I_{2p.u} \cdot Z_{2p.u} + U_{2p.u}$$

Como podemos observar desaparece la relación de transformación (**r_t**) de nuestros cálculos, y el circuito quedará representado de la siguiente manera:

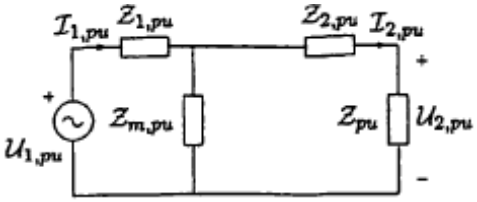

Ilustración 5. Representación circuito monofásico simplificado a valores p.u. Fuente [3]

Por otra parte, puesto que en la Red Eléctrica Española las redes utilizadas son redes trifásicas, los cálculos de los valores **p.u** varían un poco en relación a las redes monofásicas. Para realizar los cálculos se toman como bases las tensiones de línea (**U_{base}**) y las potencias trifásicas (**S_{base}**). De esta manera obtenemos las siguientes expresiones:

$$Z_{base} = \frac{U_{base}^2}{S_{base}}$$

$$I_{base} = \frac{S_{base}}{\sqrt{3} \cdot U_{base}}$$

Como podemos observar, la impedancia base (**Z_{base}**) será la misma que en el caso de los circuitos monofásicos, mientras que la intensidad de corriente base (**I_{base}**) se obtiene de aplicar la relación trifásica para circuitos eléctricos.

Utilizando estos valores base trifásicos, podemos obtener definitivamente los valores en **p.u** del circuito. Las ecuaciones que relacionarían tensión, intensidad de corriente y potencia serían las siguientes:

$$S_{pu} = \frac{S}{S_{base}} = \frac{\sqrt{3} \cdot U \cdot I}{\sqrt{3} \cdot U_{base} \cdot I_{base}} = U_{pu} \cdot I_{pu}$$

$$I_{pu} = \frac{I}{I_{base}} = \frac{\frac{U}{\sqrt{3} \cdot Z}}{\frac{U_{base}}{\sqrt{3} \cdot Z_{base}}} = \frac{U_{pu}}{Z_{pu}}$$

Como se puede observar el término $\sqrt{3}$ desaparece de nuestras ecuaciones, por lo tanto se obtiene un circuito monofásico equivalente. Sin embargo, una vez resuelto éste, al pasar los valores p.u a valores físicos, los resultados obtenidos serán los del circuito trifásico.

3.2 Transformadores de Potencia

Uno de los elementos principales de las redes de distribución eléctrica es el transformador de potencia, cuya misión es elevar, reducir o regular los niveles de tensión en la red.

Un transformador monofásico de potencia está constituido principalmente por dos devanados enrollados sobre un núcleo de material ferromagnético.

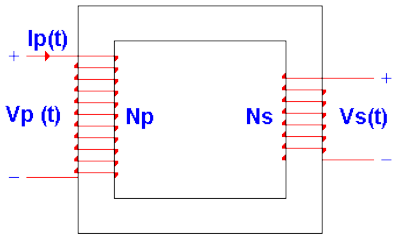

Ilustración 6. Transformador monofásico de potencia. Fuente [1]

Los transformadores monofásicos ideales han de cumplir los siguientes aspectos:

- Los devanados tienen resistencia nula.
- Nos existe flujo de dispersión.
- El núcleo tiene una reluctancia magnética nula.
- No existen pérdidas por histéresis, ni corrientes parásitas en el núcleo.

Cumpliendo estas condiciones la potencia aparente primaria ($\mathbf{S_p}$) y secundaria ($\mathbf{S_s}$) del transformador serán iguales **($\mathbf{S_p = S_s}$),** y se obtendrán las siguientes relaciones [3]:

$$\frac{U_p}{U_s} = \frac{N_p}{N_s} = r_t$$

$$\frac{I_p}{I_s} = \frac{N_s}{N_p} = \frac{1}{r_t}$$

siendo:

- U_p → tensión en el devanado primario [V]
- U_s → tensión en el devanado secundario [V]
- N_p → número de espiras en el devanado primario.
- N_s → número de espiras en el devanado secundario.

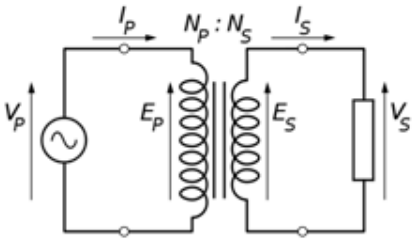

Ilustración 7. Circuito eléctrico de transformador monofásico ideal. Fuente [3]

Al trabajar con transformadores reales, y aplicar una tensión en uno de sus devanados mientras que el otro se está cargando, se observará que éstas relaciones no son del todo ciertas, por lo que el transformador se ha de modificar para mostrar un modelo más cercano a la realidad.

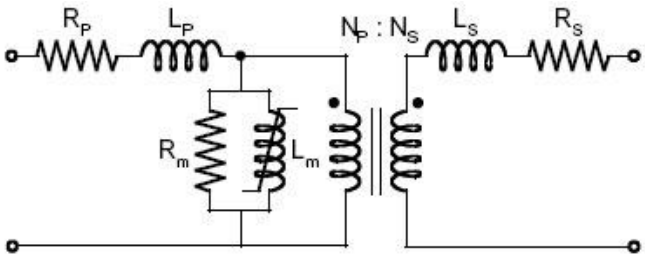

Ilustración 8. Circuito eléctrico de transformador monofásico real. Fuente [3]

En este circuito se modelan las pérdidas óhmicas en los devanados y los flujos de dispersión mediante la inclusión de las resistencias **R_p** y **R_s**, y de las reactancias **L_p** (**X_p**) **y L_s** (**X_s**). La rama que forman la resistencia **R_m** (resistencia del hierro **R_{fe}**) y la reactancia **L_m** (**X_m**) representan las pérdidas que se producen en el propio núcleo ferromagnético por corrientes de *Foucault* y fenómenos de histéresis, y también debido al hecho de que la reluctancia magnética del circuito magnético no es nula.

Para calcular el modelo de transformador real se usan dos tipos de ensayo [3]:

- *Ensayo de vacío:* consiste en alimentar únicamente uno de los devanados (por ejemplo el primario) del transformador a su tensión nominal ($\mathbf{U_{pn}}$), mientras que el otro devanado se mantiene en vacío. Como resultado se obtienen una intensidad de vacío ($\mathbf{I_0}$), una tensión en el secundario ($\mathbf{U_{s0}}$), una potencia activa ($\mathbf{P_0}$) que tiene un valor muy bajo, y una potencia reactiva ($\mathbf{Q_0}$) con un valor bajo. Con estos parámetros obtenidos se pueden obtener las siguientes relaciones:

$$\frac{U_{pn}}{U_{s0}} \approx \frac{N_p}{N_s}$$

$$R_{Fe} \approx \frac{U_{pn}^2}{U_0}$$

$$X_m \approx \frac{U_{pn}^2}{Q_0}$$

- *Ensayo de cortocircuito:* consiste en dejar el devanado secundario en cortocircuito, mientras que el devanado primario es alimentado mediante una tensión $\mathbf{U_{pcc}}$, fijada para que por este devanado circule una intensidad nominal $\mathbf{I_{pn}}$. Como resultado de este ensayo se obtienen una intensidad en el devanado secundario ($\mathbf{I_{scc}}$), prácticamente igual a la nominal, una potencia reactiva ($\mathbf{Q_{cc}}$), y una potencia activa ($\mathbf{P_{cc}}$) que tiene un valor bajo e inferior al de la potencia reactiva. A partir de los valores obtenidos en este ensayo se pueden calcular la resistencia en cortocircuito ($\mathbf{R_{cc}}$), y la reactancia en cortocircuito ($\mathbf{X_{cc}}$) reducida al primario. Las expresiones obtenidas son las siguientes:

$$R_{cc} \approx \frac{P_{cc}}{I_{pn}^2} = \frac{R_p \cdot I_{pn}^2 + R_s \cdot I_{scc}^2}{I_{pn}^2} \approx R_p + r_t^2 \cdot R_s$$

$$X_{cc} \approx \frac{Q_{cc}}{I_{pn}^2} = \frac{X_p \cdot I_{pn}^2 + X_s \cdot I_{scc}^2}{I_{pn}^2} \approx X_p + r_t^2 \cdot X_s$$

Lo habitual es expresar el resultado del ensayo de cortocircuito en tanto por ciento. Este porcentaje se obtiene de multiplicar por cien el cociente entre la tensión de alimentación del ensayo de cortocircuito (**U_{pcc}**), y la tensión nominal de ese devanado (**U_{pn}**), es decir:

$$Z_{cc\%} = \frac{U_{pcc}}{U_{pn}} \cdot 100 = \frac{I_{pcc} \cdot Z_p + \frac{N_p}{N_s} \cdot I_{sn} \cdot Z_s}{\frac{N_p}{N_s} \cdot U_{sn}} \cdot 100 = \frac{\left(\left(\frac{N_p}{N_s}\right)^2 \cdot Z_p + Z_s\right) \cdot S_{sn}}{U_{sn}^2} \cdot 100$$

3.2.1 Transformadores Trifásicos

Los transformadores trifásicos de potencia son una evolución de los transformadores monofásicos. Como principales modificaciones se observan el aumento en el número de devanados, y el distinto tipo de conexiones de que disponen (triángulo y estrella). Los transformadores trifásicos se clasifican en función de su tipo de núcleo, diferenciándose en:

- *Banco trifásico:* constituido por tres transformadores monofásicos.

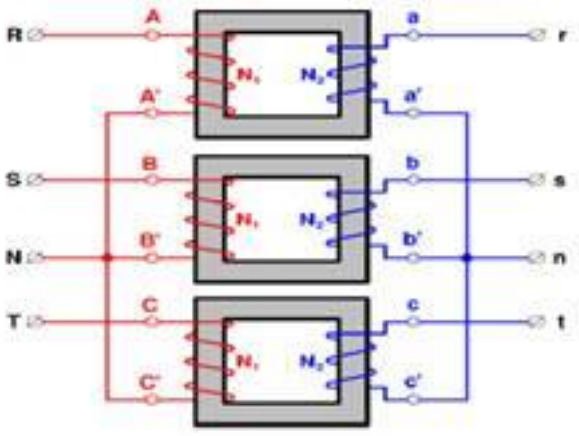

Ilustración 9. Banco trifásico. Fuente [1]

- *Transformador de 3 columnas:* contiene un único núcleo con una columna por fase.

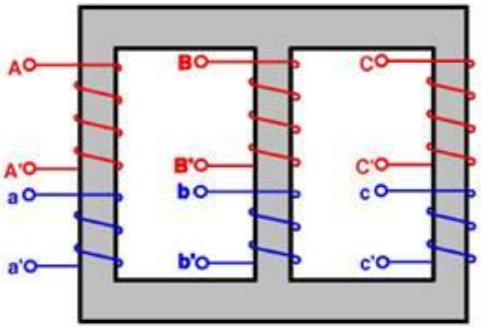

Ilustración 10. Transformador de 3 columnas. Fuente [1]

- *Transformados de 5 columnas:* el núcleo está formado por 5 columnas (3 para las fases y 2 adicionales para los extremos).

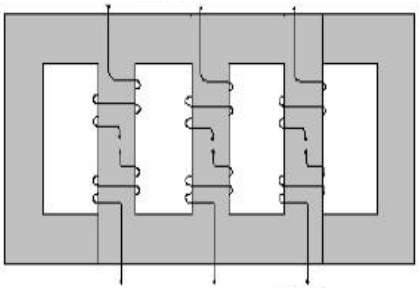

Ilustración 11. Transformador de 5 columnas. Fuente [1]

Existe otra clasificación de los transformadores trifásicos en función del número de devanados por columna que se dispongan en el núcleo:

- Transformadores de dos devanados separados (primario-secundario).
- Transformadores de tres devanados separados (primario-secundario-terciario).
- Transformadores de devanado continuo.

Por último, los transformadores trifásicos se pueden clasificar en función del tipo de conexión existente entre sus devanados:

- Transformadores con conexión estrella-triángulo.
- Transformadores con conexión estrella-estrella.
- Transformadores con conexión triángulo-triángulo.

En función del tipo de conexión entre devanados, aparecerán distintos desfases entre las tensiones primarias y secundarias, lo que supondrá que existan diferentes relaciones de transformación de los módulos. Esto provoca que en los transformadores trifásicos existan relaciones de transformación tanto de módulo como de ángulo. Éstas últimas se expresan mediante un coeficiente que indica el ángulo de desfase entre el devanado primario y el secundario en variaciones de **30º** [3].

3.2.2 Transformadores con Regulación de Módulo

Existen transformadores de potencia que se usan especialmente en las redes eléctricas de distribución. Estos transformadores se denominan *transformadores de regulación*, y se distinguen en dos tipos:

- Transformadores de regulación de módulo.
- Transformadores de regulación de ángulo.

Para el desarrollo de nuestro ***Trabajo Fin de Máster*** nos centraremos en los *transformadores de regulación de módulo* que son con mucho los más habituales.

En este tipo de transformadores uno de sus devanados tiene varias tomas correspondientes a distintos números de espiras. De esta manera, se puede modificar el módulo de la tensión en el devanado secundario ($\mathbf{U_s}$), suponiendo fija la tensión en el devanado primario ($\mathbf{U_p}$), mediante la modificación del número de espiras del devanado primario $\mathbf{N_p}$ (lado de alta tensión). De esta manera también se modifica la relación entre las espiras $\mathbf{N_p/N_s}$ [3].

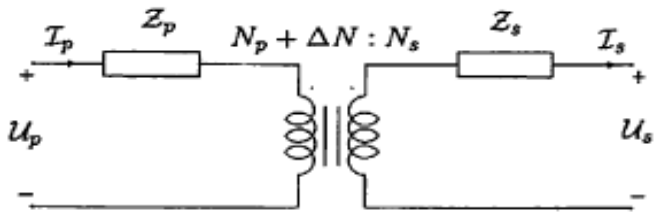

Ilustración 12. Circuito de transformador con regulación de tomas. Fuente [3]

Si pasamos este circuito a valores **p.u** (utilizando las tensiones base y manteniendo la relación de transformación N_p/N_s), podemos obtener el siguiente circuito:

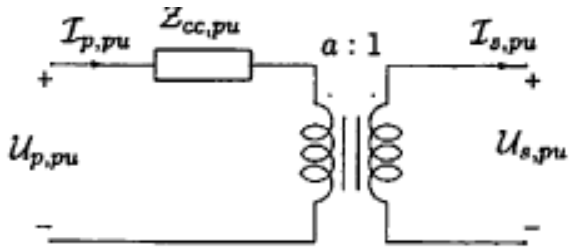

Ilustración 13. Circuito de transformador con regulación de tomas en p.u. Fuente [3]

donde:

- **Z_{ccpu}** → impedancia de cortocircuito del transformador en **p.u**.
- **a** → cambiador de tomas del transformador (***tap***). Regulador entre tensiones primaria y secundaria, donde:

$$a = 1 + \frac{\Delta N_p}{N_p}$$

Si el valor de **a** se sitúa por debajo de **1** disminuirá la tensión en el devanado primario del transformador y por tanto aumentará la tensión en el devanado secundario. Por el contrario, si el valor de **a** es mayor a **1** aumentará la tensión en el devanado primario y disminuirá en el devanado secundario.

3.2.3 Transformadores Modelo en π

Si se realiza un estudio del esquema genérico del transformador con regulación de módulo, se puede obtener un esquema más simple del mismo, a partir de la siguiente ecuación matricial **[3]**:

$$\begin{pmatrix} I_{pp.u} \\ I_{sp.u} \end{pmatrix} = \begin{pmatrix} \gamma & -a\gamma \\ a\gamma & -a^2\gamma \end{pmatrix} \cdot \begin{pmatrix} U_{pp.u} \\ U_{sp.u} \end{pmatrix}$$

donde:

$$\boldsymbol{\gamma = \frac{1}{Z_{cc,p.u}}} \rightarrow \textbf{\textit{Admitancia}}$$

De estas ecuaciones se puede obtener el siguiente circuito equivalente que se denomina ***Modelo en π***, y será el utilizado en el desarrollo de nuestro ***Trabajo Fin de Máster***.

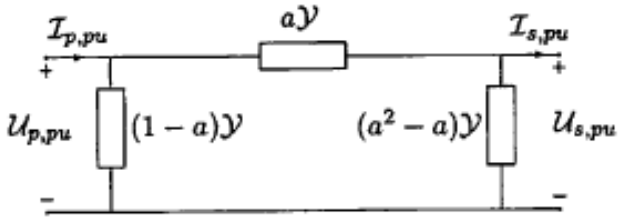

Ilustración 14. Modelo en π del circuito de transformador con regulación de tomas en p.u. Fuente [3]

Corrección del Modelo en π

Se ha observado que a la hora de llevar a cabo el análisis del **Modelo en π**, se han de realizar unas pequeñas modificaciones en el mismo para que los resultados obtenidos se asemejen a la realidad. Para llevar a cabo estas modificaciones el **Modelo en π** que aparece en la *Ilustración 14* se sustituye por el siguiente **[4]**:

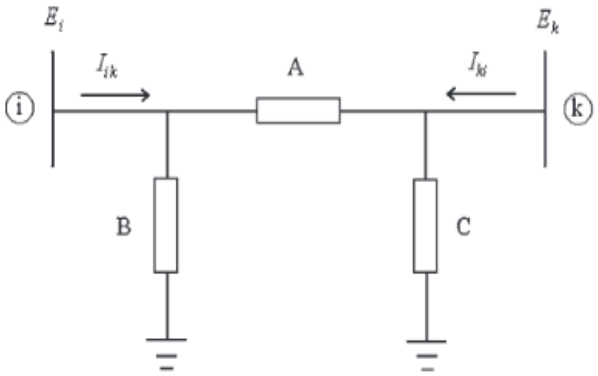

Ilustración 15. Corrección del modelo en π. Fuente [4]

donde: $A = \frac{1}{a} \cdot \gamma$ $B = \frac{1-a}{a^2} \cdot \gamma$ $C = \frac{a-1}{a} \cdot \gamma$

4. Flujo de Cargas en Redes Eléctricas

El ***Flujo de Cargas*** consiste en obtener las condiciones de operación en régimen permanente de un sistema de energía eléctrica. Concretamente, dados los consumos en cada nodo, y la potencia generada por los alternadores, se trata de obtener las tensiones en los nodos y los flujos de potencia de las líneas y los transformadores.

El flujo de cargas consta de dos etapas:

- *Primera etapa:* consiste en obtener las tensiones complejas en todos los nodos del circuito. Esto no se puede realizar utilizando las herramientas convencionales de análisis de circuitos lineales, ya que las restricciones de contorno no se especifican en términos de impedancias (cargas) y fuentes de tensión (generadores), sino de potencias, lo cual implica la obtención de un sistema de ecuaciones no lineales.
- *Segunda etapa:* consiste en el cálculo de todas las magnitudes de interés del circuito, como son los flujos de potencia activa y reactiva, las pérdidas, los consumos, etc.

Para la resolución del ***problema de Flujo de Cargas*** se emplean diferentes técnicas numéricas las cuales se empezaron a desarrollar por ingenieros eléctricos a principios de los años 70.

El uso de la técnica de ***Flujo de Cargas*** será necesaria en nuestro ***Trabajo Fin de Máster*** en el proceso de generación de medidas sintéticas, las cuales serán utilizadas en el desarrollo del algoritmo de estimación de estado ***WLS***.

4.1 Desarrollo del Problema de Flujo de Cargas

Para la formulación del problema de flujo de cargas partimos de una red eléctrica de ***n*** nodos, de la que conocemos las tensiones complejas en todos ellos. Aplicando las *Leyes de Kirchhoff* y los modelos para componente de la red, se obtienen las siguientes ecuaciones complejas para los diferentes nodos de la red **[3]**:

$$I = \gamma \cdot U$$

$$I_i = \sum_{i=1}^{n} \gamma_{ij} \cdot U_j \qquad \text{i} = 1, 2, \dots, \text{n}$$

donde:

- I → vector de intensidades netas inyectadas en los nodos.
- U → vector de tensiones nodales.
- γ → matriz *n x n* de admitancias de los nodos.

Además en cada nudo se cumple que:

$$S_i = S_{Gi} - S_{Ci} = U_i \cdot I_i^*$$

donde:

- S_i → potencia aparente neta inyectada en el *nodo i* [KVA]
- S_{Gi} → potencia aparente generada en el *nodo i*.
- S_{Ci} → potencia aparente consumida en el *nodo i*.
- I^*_i → conjugado del vector de intensidades netas inyectadas en los nodos.

Aplicando la ecuación anterior a todos los nodos del circuito, se obtiene la siguiente expresión matricial:

$$S = diag(U) \cdot I^*$$

donde:

- S → vector de potencias aparentes nodales.
- diag(U) → matriz diagonal formada por los elementos del vector de tensiones nodales.

Una vez conocemos la matriz de admitancias (**γ**), las ecuaciones de I y S constituyen un sistema de **2n** ecuaciones complejas, con **3n** incógnitas complejas (S, U e I). Para resolver este sistema de ecuaciones no lineales, introducimos la matriz de admitancias en la expresión del vector de potencias aparentes nodales, obteniendo la siguiente ecuación:

$$S = diag(U) \cdot [\gamma \cdot U]^*$$

El siguiente paso es descomponer la potencia compleja en parte real e imaginaria, y la matriz de admitancias en sus coordenadas cartesianas:

$$S = P + Qj$$

$$\gamma = G + Bj$$

Sustituyendo:

$$P + Qj = diag(U) \cdot [G - Bj] \cdot U^*$$

donde:

- P → potencia activa [KW]
- Q → potencia reactiva [KVAr]
- G → conductancia [S]
- B → susceptancia [S]

Los métodos iterativos utilizados para resolver el problema de flujo de cargas no pueden trabajar con ecuaciones complejas, por lo que es necesario convertirlas a ecuaciones reales. Normalmente las tensiones se expresan en coordenadas polares. Las ecuaciones resultantes son las siguientes:

$$P_i + j \cdot Q_i = U_i \cdot \sum_{j=1}^{n} \left[G_{ij} - j \cdot B_{ij}\right] \cdot U_j^* \quad \text{i} = 1, 2, \dots, \text{n}$$

$$P_i = V_i \cdot \sum_{j=1}^{n} V_j \cdot \left(G_{ij} \cdot cos\Theta_{ij} + B_{ij} \cdot sen\Theta_{ij}\right) \quad \text{i} = 1, 2, \dots, \text{n}$$

$$Q_i = V_i \cdot \sum_{j=1}^{n} V_j \cdot \left(G_{ij} \cdot sen\Theta_{ij} + B_{ij} \cdot cos\Theta_{ij}\right) \quad \text{i} = 1, 2, \dots, \text{n}$$

Se puede observar que de cada nodo se extraen dos ecuaciones y cuatro incógnitas, por lo que se han de establecer dos magnitudes por nodo para que las ecuaciones puedan ser resueltas. Para ello hemos de distinguir dos tipos principales de nodos:

- *Nodos de consumo o nodos PQ:* se conoce el consumo de potencia activa ($\mathbf{P_{ci}^{esp}}$), y de potencia reactiva ($\mathbf{Q_{ci}^{esp}}$), siendo nula la potencia generada $(P_{Gi} = Q_{Gi} = 0)$. Se cumple por tanto:

  $$P_i^{esp} = -P_{ci}^{esp} \; ; \; Q_i^{esp} = -Q_{ci}^{esp}$$

  La gran mayoría de los nodos de una red son de este tipo.

- *Nodos de generación o nodos PV:* en ellos un generador regula la tensión a un valor especificado ($\mathbf{V_i^{esp}}$) e inyecta una potencia activa ($\mathbf{P_{Gi}^{esp}}$). Y por tanto se cumple:

  $$P_i^{esp} = P_{Gi}^{esp} - P_{ci}^{esp} \; ; \; V_i = V_i^{esp}$$

Si solo consideráramos este tipo de nodos, las potencias activas se deberían de especificar de antemano, cosa que resulta imposible ya que las pérdidas de la red, que son potencias aportadas por los generadores, no se pueden conocer hasta obtener los flujos de potencia de cada elemento. Es decir, al menos la potencia activa de un generador no puede ser especificada y debe de ser calculada al final del problema. Esta incógnita se resuelve gracias a que a la hora de trabajar con **fasores**, uno de los ángulos lo podemos tomar como origen de fases (habitualmente **0º**). Normalmente se toma como origen de fases el nodo de generación.

Este nodo se denomina **nodo *SLACK***. Una vez fijado el nodo *SLACK* las ecuaciones que intervienen en el problema del flujo de cargas serán:

$$P_i^{esp} = V_i \cdot \sum_{j=1}^{n} V_j \cdot \left(G_{ij} \cdot cos\Theta_{ij} + B_{ij} \cdot sen\Theta_{ij}\right) \quad \text{i} = 1, 2, \dots, \text{nd} + \text{ng}$$

$$Q_i^{esp} = V_i \cdot \sum_{j=1}^{n} V_j \cdot \left(G_{ij} \cdot sen\Theta_{ij} + B_{ij} \cdot cos\Theta_{ij}\right) \quad \text{i} = 1, 2, \dots, \text{nd}$$

donde:

$$n_g = n - n_d - 1$$

- n_g → número de nodos de generación.
- n → número total de nodos.
- n_d → número de nodos de consumo.

La solución al problema consiste en encontrar los desfases (**Θ_i**), y los módulos de tensión (**V_i**).

Puesto que las ecuaciones resultantes son no lineales, su solución será iterativa, por lo que será necesario establecer unos valores iniciales para las variables a resolver. Para ello se establece que **$\Theta_i^0 = 0$** para todos los nodos, y que **$V_i^0 = 1$ p.u** para todos los nodos de consumo.

Finalmente los flujos de potencia para un elemento conectado entre los nodos i y j se obtendrá de las siguientes expresiones:

$$P_{ij} = V_i \cdot V_j \cdot \left(G_{ij} \cdot cos\Theta_{ij} + B_{ij} \cdot sen\Theta_{ij}\right) - G_{ij} \cdot V_i^2$$

$$Q_i = V_i \cdot V_j \cdot \left(G_{ij} \cdot sen\Theta_{ij} + B_{ij} \cdot cos\Theta_{ij}\right) + V_i^2 \cdot \left(B_{ij} - b_{pij}\right)$$

donde:

- b_p → susceptancia en paralelo del modelo de transformador en π.

Para finalizar las pérdidas totales del sistema pueden calcularse, una vez hallada la potencia en el nodo ***SLACK***, como suma de las potencias inyectadas en todos los nodos, o bien como suma de las pérdidas de cada elemento.

4.2 Método Direct Approach

Como hemos explicado anteriormente para resolver el problema del flujo de cargas se necesita del uso de métodos iterativos. A continuación se expone el método usado en la realización de este ***Trabajo Fin de Máster***.

Se trata de un algoritmo novedoso para la resolución de los problemas de flujo de cargas, basado en las técnicas clásicas de resolución. Este algoritmo está especialmente orientado a resolver redes radiales y por tanto resulta mucho más adecuado en el desarrollo de nuestro ***Trabajo Fin de Máster***.

Los únicos datos que se toman como entrada son los valores de las ramas y los nodos del sistema eléctrico a resolver. Haciéndolo de esta manera se consigue resolver el flujo de cargas de una manera más rápida ya que no se pierde tiempo en iterar y resolver la matriz de admitancias γ.

DESARROLLO DEL MÉTODO DIRECT APPROACH [5]

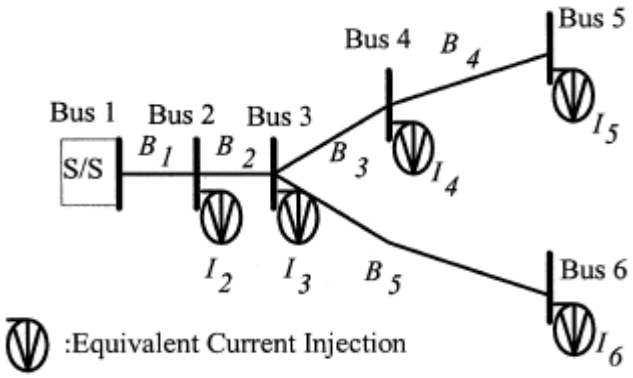

Ilustración 16. Sistema de distribución simple. Fuente [5]

El sistema de distribución de la *Ilustración 16* se tomará como ejemplo para el desarrollo del método Direct Approach.

El método toma como base dos matrices, la matriz de corrientes de rama y corrientes inyectadas, y la matriz de corrientes de rama y tensiones inyectadas.

Para las redes de distribución se parte de las siguientes condiciones:

$$S_i = P_i + Q_i \cdot j \quad i = 1, \dots, n$$

$$I_i^k = I_i^r \cdot \left(V_i^k\right) + j \cdot I_i^j \cdot \left(V_i^k\right) = \left(\frac{P_i + j \cdot Q_i}{V_i^k}\right)^*$$

donde:

- S_i → potencia aparente
- V_i^k → tensión equivalente del *nodo i* en la iteración *k*
- I_i^k → corriente equivalente del *nodo i* en la iteración *k*
- I_i^r → parte real de la corriente inyectada en la iteración *k*
- I_i^i → parte imaginaria de la corriente inyectada en la iteración *k*

Relaciones entre las Matrices

Las potencias inyectadas pueden ser convertidas en corrientes inyectadas gracias a la anterior ecuación, y la relación existente entre el nodo de corrientes inyectadas y las corrientes de rama se puede obtener aplicando la *Ley de Corrientes de Kirchhoff* en sistema de distribución. De esta manera la relación se puede expresar matricialmente de la siguiente manera:

$$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \end{bmatrix}$$

Este desarrollo matricial se puede expresar como:

$$[B] = [BIBC] \cdot [I]$$

donde:

- B → corrientes de los nodos
- BIBC → matriz que relaciona al nodo de inyección con las corrientes de rama. (Sólo puede contener 1 y 0)
- I → corrientes de rama

La relación entre las corrientes de rama y las tensiones inyectadas por los nodos se obtiene de la siguiente expresión matricial:

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{bmatrix} - \begin{bmatrix} V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{bmatrix} = \begin{bmatrix} Z_{12} & 0 & 0 & 0 & 0 \\ Z_{12} & Z_{23} & 0 & 0 & 0 \\ Z_{12} & Z_{23} & Z_{34} & 0 & 0 \\ Z_{12} & Z_{23} & Z_{34} & Z_{45} & 0 \\ Z_{12} & Z_{23} & 0 & 0 & Z_{56} \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{bmatrix}$$

donde:

- V_i → tensiones en los nodos.
- Z_{ij} → impedancia entre *nodo i* y *nodo j*.

Este desarrollo matricial se puede expresar como:

$$[\Delta V] = [BCBV] \cdot [B]$$

donde:

- BCBV → matriz de impedancias compartidas entre nodos

<u>Formulación del Problema</u>

La construcción del algoritmo para desarrollar las matrices **BIBC** y **BCBV** consta de los siguientes pasos:

- ***Paso 1****:* para un sistema que conste de ***m-ramas*** y ***n-nodos*** la dimensión de la matriz será ***m x (n-1).***
- ***Paso 2:*** si el tramo de la línea de distribución (B_k) está situada entre el *nodo i* y el *nodo j*, se copiará la columna de las ***i*** de la matriz BIBC en la columna de las ***j***. A continuación se añadirá ***a+1*** en la posición de la fila ***k*** y la columna ***j*** del nodo.
- ***Paso 3:*** repetir el procedimiento anterior hasta que todas las secciones lineales estén incluidas en la matriz BIBC.
- ***Paso 4:*** para un sistema de distribución con una sección de ***m-ramas*** y ***n-nodos***, la dimensión de la matriz BCBV será ***(n-1) x m.***
- ***Paso 5:*** si el tramo de la línea de distribución (B_k) está localizada entre el *nodo i* y el *nodo j*, se copiará la fila de las ***i*** de la matriz BCBV en la fila de las ***j***.

Posteriormente se ha de añadir la línea de impedancias (Z_{ij}), en la posición de la fila ***j***, y de la columna ***k*** del nodo.

- ***Paso 6:*** repetir este procedimiento hasta que todas las secciones lineales estén incluidas en la matriz BCBV.

Solución del Problema

Las matrices ***BIBC*** y ***BCBV*** se desarrollan en base a la estructura topológica de los sistemas de distribución. La matriz **BIBC** representa la relación entre las corrientes inyectadas en el nodo y las corrientes que circulan por las ramas. Las variaciones sufridas por las corrientes de rama, generadas por las variaciones de las corrientes inyectadas, pueden calcularse directamente gracias a la matriz **BIBC.**

La matriz **BCBV** representa la relación entre las corrientes de rama y las tensiones en los nodos. Las variaciones correspondientes a las tensiones de los nodos que son generadas por las variaciones de las corrientes de rama, se pueden calcular directamente gracias a la matriz **BVBC**.

La relación entre las corrientes y las tensiones inyectadas en los nodos se puede expresar de la siguiente manera:

$$[\Delta V] = [BCBV] \cdot [BIBC] \cdot [I] = [DLF] \cdot [I]$$

donde:

- DLF → combinación de las matrices BCBV y BIBC

Finalmente la solución para el problema de flujo de carga en los sistemas de distribución se puede obtener resolviendo iterativamente las siguientes ecuaciones:

$$I_i^k = I_i^r \cdot \left(V_i^k\right) + j \cdot I_i^j \cdot \left(V_i^k\right) = \left(\frac{P_i + j \cdot Q_i}{V_i^k}\right)^*$$

$$[\Delta V^{k+1}] = [DLF] \cdot [I^k]$$

$$[V^{k+1}] = [V^0] + [\Delta V^{k+1}]$$

5. Estimación de Estado

Los primeros sistemas de transporte de energía eléctrica, inicialmente aislados entre sí, se habían ido interconectando por motivos de seguridad y economía. Sin embargo, uno de los principales inconvenientes de la interconexión consiste en la posibilidad de que un incidente se extienda a áreas mucho mayores, lo que obligó desde un principio a realizar un seguimiento continuo del funcionamiento de dichos sistemas.

Gracias a los primeros ordenadores, se pudieron instalar los denominados *Sistemas de Supervisión, Control y Adquisición de Datos* (**SCADA**), presentes hoy en día en innumerables instalaciones industriales de cierta complejidad, como centrales nucleares, sistemas ferroviarios, gaseoductos, etc. Las funciones más primitivas de un sistema SCADA consisten en la captura de todos los datos relevantes del sistema supervisado, mediante unidades normalmente remotas, el mantenimiento de una base de datos, la presentación en pantallas gráficas de la información disponible, resaltando posibles alarmas o eventos importantes, y facilitar al operador la actuación sobre elementos de control del sistema para modificar su evolución.

En el caso de los sistemas eléctricos, los **SCADA** incorporaban además ciertas funciones propias, como el control automático de la generación y el despacho económico. Para ello, además de los estados de los interruptores, se monitorizaban la frecuencia del sistema y las potencias activas de los generadores.

Ciertos incidentes pusieron de manifiesto que había que prestar muchas más atención a la seguridad de operación del sistema, lo que requería sistemas **SCADA** más sofisticados que los entonces existentes. Se empezaron a capturar, a intervalos de tiempo menores, un mayor número de medidas, incluyendo flujos de potencia por las líneas, y se desarrollaron nuevas herramientas informáticas, que eventualmente permitían analizar la seguridad de la red, los riesgos de inestabilidad, las pérdidas, etc.

Todo este nuevo entramado, se basaba fundamentalmente, en el conocimiento del estado del sistema, determinado completamente por las tensiones complejas en todos los nodos. Los primeros intentos de obtener dicho estado mediante un flujo de cargas *on-line* estuvieron plagados de problemas, como consecuencia de la carencia de ciertas medidas,

la inconsistencia de otras y la imposibilidad de aprovechar toda la información disponible (el flujo de cargas no utiliza por ejemplo los flujos por las líneas).

Un estimador de estado, trabajando *on-line*, permite obtener una base de datos fiable y completa, imprescindible para el correcto funcionamiento de todas las actividades involucradas en el control y operación del sistema eléctrico, empezando por la evolución de seguridad. Hasta tal punto el análisis de seguridad depende de los resultados del estimador de estado, que usualmente ambas herramientas aparecen y se describen como una sola.

Pero el registro histórico de toda la información generada resulta también útil para la mayoría de funciones relacionadas con la planificación (predicción de la demanda, estudios de fiabilidad, ampliación de sistemas de generación y transporte, etc.), y la gestión de los nuevos mercados de electricidad. La incorporación de los sistemas SCADA convencionales de todas estas funciones, junto al avance espectacular en arquitecturas de ordenadores han dado lugar a los modernos sistemas de *Gestión de Energía* (EMS).

Un estimador de estado incluye básicamente las siguientes funciones:

- Prefiltrado de Medidas
- Procesador Topológico
- Análisis de Observabilidad
- Estimación de Estado
- Procesador de Medidas Erróneas

El estimador de estado, por tanto, actúa como un filtro entre las medidas de campo y todas las aplicaciones del *EMS*, que requieren la base de datos más fiable posible. Sirve además para que los operadores del sistema aumenten su confianza en los valores que se les muestran y dispongan de magnitudes importantes que no siempre se miden **[3]**.

5.1 Método WLS

El objetivo de lo que se conoce como estimación de estado es estimar las tensiones complejas en todos los nodos eléctricos del sistema. Esto se logra procesando las medidas disponibles y la información sobre el estado de los interruptores, seccionadores y tomas de transformadores, así como los parámetros de líneas, transformadores, bancos de condensadores y reactancias.

Los tipos de medidas más comúnmente utilizadas son los siguientes:

- *Flujos:* flujos de potencia activa y reactiva medidas en ambos extremos de líneas y transformadores.
- *Inyecciones:* potencia neta activa y reactiva inyectada en los nudos.
- *Módulos de tensiones:* lecturas de los voltímetros en los embarrados.
- *Módulos de corriente:* lecturas de amperímetros en ambos extremos de líneas y transformadores.

Todas las medidas llevan asociado un cierto error, que proviene de los transformadores de medida (tensión e intensidad), del propio transductor, del proceso de conversión analógico-digital, y del posible sesgo o ruido introducido por el sistema de comunicaciones, como consecuencia del lapso de tiempo transcurrido entre la primera y la última medida.

Además de estas medidas mencionadas, existen ciertas magnitudes que, sin proceder de un aparato de medida, pueden utilizarse como medidas en el proceso de estimación. Éstas son:

- *Medidas virtuales:* valores que vienen impuestos por restricciones de la propia red.
- *Pseudo-medidas:* valores basados en datos históricos o en predicciones utilizados para mejorar la redundancia en zonas pobremente monitorizadas.

El error más comúnmente asociado tanto a les medidas de tensiones, como a las medidas de potencia activa y reactiva es el siguiente **[6]**:

- **0.1 %** de error en las medidas de las tensiones en las redes de distribución.
- **2%** de error en las medidas de potencia activa y reactiva en las redes de distribución.

DESARROLLO DEL MÉTODO [3]

Dado el modelo de la red, todas las medidas pueden expresarse como funciones, generalmente no lineales, del estado del sistema. Estas expresiones no tienen en cuenta los posibles errores de las medidas, que deben modelarse como un término adicional.

Si se considera un vector ***z,*** compuesto por ***m*** medidas, éstas pueden ser expresadas en función de los ***n*** componentes del vector de estado ***x*** como se muestra a continuación (**n = 2N-1**, siendo **n** el número de nodos):

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} h_1(x_1, x_2 \dots, x_n) \\ h_2(x_1, x_2 \dots, x_n) \\ \vdots \\ h_m(x_1, x_2 \dots, x_n) \end{bmatrix} + \begin{bmatrix} e_1 \\ e_1 \\ \vdots \\ e_m \end{bmatrix} = h(x) + e$$

donde:

- $h_i(x)$:es la función no lineal que relaciona la medida i con el vector x.
- x_i: es el vector de estado.
- e_i: es el vector de errores en las medidas.

Utilizando coordenadas polares para las tensiones, y cartesianas para los elementos de la matriz de admitancias de nudos, las funciones $h_i(x)$ relativas a medidas de potencia son las siguientes:

- Medidas de inyección neta en el nodo i:

$$P_i = \sum_{j=1}^{n} V_i \cdot V_j \cdot \left(G_{ij} \cdot cos\Theta_{ij} + B_{ij} \cdot sen\Theta_{ij}\right)$$

$$Q_i = \sum_{j=1}^{n} V_i \cdot V_j \cdot \left(G_{ij} \cdot sen\Theta_{ij} + B_{ij} \cdot cos\Theta_{ij}\right)$$

donde:

- V_i,V_j: módulos de las tensiones en los nudos i y j.
- θ_{ij}: θ_i - θ_i el desfase entre los nodos i y j.
- G_{ij} + B_{ij}: el elemento i,j-ésimo de la matriz de admitancias de nudos.

Las medidas de tensión están relacionadas trivialmente con la variable de estado respectiva, y las medidas de intensidad no son frecuentes en redes de transporte.

Resulta habitual hacer las siguientes suposiciones sobre las propiedades estadísticas de los errores de las medidas:

- Los errores siguen una distribución normal.
- El valor esperado de todos los errores es nulo, es decir, $\boldsymbol{E(e_i)=0}$, $i = 1, 2, \dots, m$
- Los errores son independientes, es decir, $\boldsymbol{E(e_i,e_j)=0}$.

Por tanto:

$$Cov(e) = E(e \cdot e^T) = R = diag\{\sigma_1^2, \sigma_2^2, \dots, \sigma_m^2\}$$

donde:

- σ_i: desviación típica de cada medida. Se calcula para reflejar la precisión esperada de los aparatos de medida involucrados.

La formulación del problema de estimación de estado se basa en el concepto de *estimación de máxima verosimilitud*. El estimador de máxima verosimilitud de una variable aleatoria maximiza una función de probabilidad que se define en base a las hipótesis realizadas para el problema en cuestión. Lógicamente, formulaciones basadas en hipótesis distintas darán resultados distintos. El desarrollo que sigue es el más común, y se basa en las hipótesis sobre los errores de medidas enunciados anteriormente.

La primera suposición es que los errores siguen una distribución gaussiana o normal. Una variable aleatoria sigue un distribución normal si su función de densidad de probabilidad ***f(z)*** viene dada por:

$$f(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$$

donde:

- z: variable aleatoria.
- μ: valor medio o esperado de z_i E(z).
- σ: desviación típica de z.

Mediante el siguiente cambio de variable se obtiene una distribución normal estandarizada, **Φ(u)**, con lo que sólo es preciso considerar una única distribución normal.

Sea $u = \frac{z-\mu}{\sigma}$, entonces E(u)=0, Var(u)=1.0

$$\Phi(u) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\mu^2}{2}}$$

La segunda suposición se incorpora fácilmente haciendo cero al valor esperado de los errores en las anteriores expresiones.

La tercera suposición implica que la *función de probabilidad* del conjunto de *m* medidas se obtiene simplemente tomando el producto de las funciones correspondientes a cada medida, es decir:

$$f_m(z) = f(z_1)f(z_2) \dots f(z_m)$$

Para simplificar la aritmética, se usa el logaritmo de dicha función, que se expresa entonces como:

$$\mathcal{L} = log f_m(z) = \sum_{i=1}^{m} log\ f(z_i) = -\frac{1}{2}\sum_{i=1}^{m}\left(\frac{z_i - \mu_i}{\sigma_i}\right)^2 - \frac{m}{2} log 2\pi - \sum_{i=1}^{m} log\sigma_i$$

El estado de máxima verosimilitud ẋ se puede obtener maximizando cualquiera de las dos funciones para un conjunto dado de observaciones $z_1, z_2, \dots, z_m$. Esto constituye un problema de optimización que puede formularse como:

$$\text{maximizar } log f_m(z)$$

o

$$\text{minimizar } \sum_{i=1}^{m} \left(\frac{z_i - \mu_i}{\sigma_i}\right)^2$$

Si definimos el residuo de la medida i como:

$$r_i = z_i - E(z_i)$$

donde:

- $E(z_i) = h_i(x)$ y h_i es la función no lineal que relaciona el vector de estado **x** con la medida i.

Las inversas de las varianzas de las medidas pueden considerarse como "pesos" asignados a cada medida individual: valores altos para medidas precisas con pequeña varianza y pesos pequeños para medidas con gran incertidumbre. Denotamos este peso para la medida i como $W_{ii} = \sigma_i^{-2}$.

Con la notación y variables introducidas, el estimador de máxima verosimilitud puede formularse como el siguiente problema de optimización:

$$\text{minimizar } J(x) = \sum_{i=1}^{m} W_{ii} r_i^2$$

$$\text{sujeto a } z_i = h_i(x) + r_i, \ i = 1, \dots, m$$

La solución de este problema se conoce como el ***Estimador de Mínimos Cuadrados Ponderados*** (conocido por las siglas inglesas **WLS**) ***de x***.

RESOLUCIÓN DEL MÉTODO MEDIANTE LAS ECUACIONES NORMALES [6]

La estimación de estado cuando los errores en las medidas son independientes y siguen una distribución normal consiste en resolver el problema de mínimos cuadrados, cuya función objetivo se puede reescribir del siguiente modo:

$$j(x) = [z - h(x)]^T W[z - h(x)] = \sum_{i=1}^{m} \frac{[z_i - h_i(x)]^2}{\sigma_i^2}$$

donde $\boldsymbol{W = R^{-1}}$.

En el mínimo, deben cumplirse las n condiciones de optimalidad de primer orden que en este caso son

$$\frac{\partial J(x)}{\partial x} = 0 \Rightarrow H^T(x)W[z - h(x)] = 0$$

donde:

- $\boldsymbol{H(x) = \frac{\partial h(x)}{\partial x}}$, es la matriz jacobiano del vector *h(x)*, de dimensión *m x n*.

Se trata de encontrar el valor de ẋ que satisface la anterior ecuación. Despreciando los términos donde aparecen segundas derivadas de h(x), el sistema lineal de n ecuaciones, denominadas *ecuaciones normales*, que deber resolverse en cada iteración es el siguiente:

$$G(x^k)\Delta x^k = H^T(x^k)W[z - h(x^k)]$$

donde:

- x^k denota el valor de x en la iteración k-ésima
- $G(x) = H^T(x)WH(x)$ se conoce como ***matriz de ganancia***

Una vez resuelto el sistema, el vector de estado debe actualizarse para la siguiente iteración:

$$x^{k+1} = x^k + \Delta x^k$$

El procedimiento completo para resolver el problema se resume en los siguientes pasos:

1. Tomar el perfil plano como valor inicial para $\boldsymbol{x^0}$ ($\boldsymbol{V_i}$ ***=1 p.u,*** $\boldsymbol{\Theta_i}$ ***= 0***). Hacer $k = 0$.
2. Calcular los residuos $\Delta z^k = z - h(x^k)$.
3. Obtener H y calcular $G = H^T W H$.
4. Resolver el sistema $G\Delta x^k = H^T W \Delta z^k$.
5. Actualizar el vector de estado $x^{k+1} = x^k + \Delta x^k$ y hacer $k = k + 1$.
6. Si alguno de los elementos de Δx es mayor que una determinada tolerancia, volver al paso 2. En caso contrario, o si k excede un valor prefijado, detener el proceso.

Los términos del jacobiano $\boldsymbol{H(x)} = \frac{\boldsymbol{\partial h(x)}}{\boldsymbol{\partial x}}$, correspondientes a medidas de inyección se muestran a continuación:

- *Parciales respecto a U:*

$$\frac{\partial U_{i,j}}{\partial \Theta_{i,j}} = 0$$

$$\frac{\partial U_i}{\partial U_i} = 1$$

$$\frac{\partial U_i}{\partial U_j} = 0$$

- *Parciales respecto a P*

$$\frac{\partial P_i}{\partial \Theta_i} = \sum_{j=1}^{N} U_i U_j \left(-G_{i,j} \cdot sin\Theta_{i,j} + B_{i,j} \cdot cos\Theta_{i,j}\right) - U_i^2 B_{ii}$$

$$\frac{\partial P_i}{\partial \Theta_j} = U_i U_j \left(G_{i,j} \cdot sin\Theta_{i,j} - B_{i,j} \cdot cos\Theta_{i,j}\right)$$

$$\frac{\partial P_i}{\partial U_i} = \sum_{j=1}^{N} U_j \left(G_{i,j} \cdot cos\Theta_{i,j} + B_{i,j} \cdot sin\Theta_{i,j}\right) - U_i G_{ii}$$

$$\frac{\partial P_i}{\partial U_j} = U_i \left(G_{i,j} \cdot cos\Theta_{i,j} + B_{i,j} \cdot sin\Theta_{i,j}\right)$$

- *Parciales respecto a Q*

$$\frac{\partial Q_i}{\partial \Theta_i} = \sum_{j=1}^{N} U_i U_j \left(G_{i,j} \cdot cos\Theta_{i,j} + B_{i,j} \cdot sin\Theta_{i,j}\right) - U_i^2 G_{ii}$$

$$\frac{\partial Q_i}{\partial \Theta_j} = U_i U_j \left(-G_{i,j} \cdot cos\Theta_{i,j} - B_{i,j} \cdot sin\Theta_{i,j}\right)$$

$$\frac{\partial Q_i}{\partial U_i} = \sum_{j=1}^{N} U_j \left(G_{i,j} \cdot sin\Theta_{i,j} - B_{i,j} \cdot cos\Theta_{i,j}\right) - U_i B_{ii}$$

$$\frac{\partial Q_i}{\partial U_j} = U_i \left(G_{i,j} \cdot sin\Theta_{i,j} - B_{i,j} \cdot cos\Theta_{i,j}\right)$$

6. Caso de Estudio

En este ***Trabajo Fin de Máster*** se realizará la implementación del algoritmo de estimación de estado en una *Red de Distribución* mediante el método ***WLS*** (*Weighted Least Squares*). La implementación será genérica, pero se tomará como base una pequeña *Red de Distribución Eléctrica* formada por **9** nodos que incluye **4** transformadores con cambiadores de tomas (*taps*). El trabajo a desarrollar se ha dividido en las siguientes tareas:

- Resolución del problema de flujo de cargas gracias a la aplicación del método iterativo ***Direct Approach,*** mediante el uso del sistema p.u, con el fin de obtener los valores de corrientes, tensiones y potencias que afectan a la *Red de Distribución*.
- Estos valores se tomarán como referencia para generar unas medidas sintéticas en las que se introducirá el error asociado tanto a las medidas de tensión como a las medidas de potencia, que serán utilizadas para desarrollar el algoritmo de estimación de estado con el cual vamos a trabajar posteriormente.
- Desarrollo del algoritmo de estimación de estado ***WLS*** en ***Matlab*** a partir de las medidas sintéticas generadas tras resolver el problema de flujo de cargas.
- Resolución del problema de flujo de cargas gracias a la aplicación del método iterativo ***Direct Approach*** en el cual se introducirá un perfil de cargas diario que constará de 1440 pares de medidas de potencia activa y reactiva. Tras obtener las tensiones asociadas a cada par de medidas de potencia mediante la resolución del problema de flujo de cargas, se introducirá el error asociado a dichas medidas. Las medidas corrompidas serán utilizadas para desarrollar el algoritmo final de estimación de estado.
- Desarrollo en ***Matlab*** del algoritmo de estimación de estado en el cual se introducirán las medidas corrompidas del perfil de cargas diario.
- Desarrollo en ***Python*** del algoritmo de estimación de estado en el cual se introducirán las medidas corrompidas del perfil de cargas diario.
- Introducción del medidor ***Power Meter PowerLogic PM5560*** en nuestro algoritmo de estimación de estado. Gracias a este medidor podremos introducir

medidas en tiempo real tanto de potencias activas y reactivas como de tensiones en nuestro algoritmo. Estas medidas serán tomadas de una bombilla doméstica conectada a la red de distribución eléctrica. Para introducir estas medidas en nuestro algoritmo se sustituirán los datos de uno de los nodos de la Red de Distribución que hemos usado como base, en concreto el nodo 9, por los datos proporcionados por el medidor.

- Finalmente se procederá a la introducción de nuestro algoritmo de estimación de estado en el dispositivo electrónico ***Raspberry Pi3 Model B+.*** Gracias a este dispositivo podremos visualizar en tiempo real la evolución tanto de las medidas de tensión como de las medidas de potencia activa y reactiva de la bombilla doméstica conectada a la red de distribución eléctrica.

A continuación se muestra la ***Red de Distribución*** a analizar:

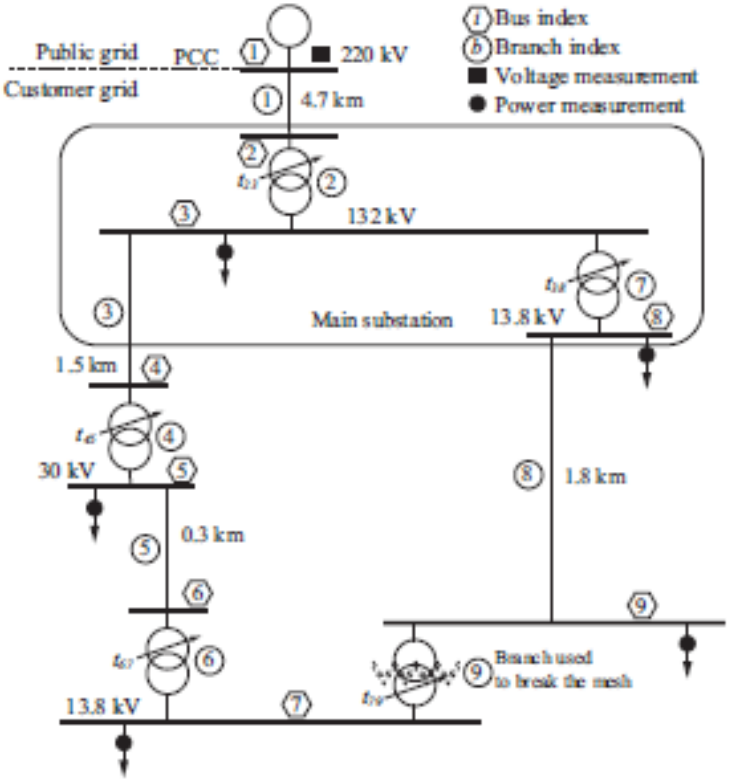

Ilustración 17. Red de Distribución Eléctrica de 9 nodos. Fuente: [Propia]

Los datos de partida se muestran en las siguientes tablas:

Líneas	Ω/km	*km*
Línea 12	0.025+0.240j	4.7
Línea 34	0.161+0.151j	1.5
Línea 56	0.568+0.133j	0.3
Línea 89	0.161+0.112j	1.8

Tabla 4. Líneas de la Red de Distribución. Fuente: [Propia]

Transformadores	*Sn (MVA)*	*Rsc (%)*	*Xsc (%)*	*a (tap)*	*Θ (deg)*
Transformador 23	2 x 270	0.90	12.9	0.98	-30
Transformador 45	3 x 37.5	0.90	9	0.99	0
Transformador 67	10	0.95	4.8	1	30
Transformador 38	3 x 50	0.92	8.5	0.98	30

Tabla 5. Transformadores de la Red de Distribución. Fuente: [Propia]

Donde:

- 2 x… → la Red cuenta con dos transformadores en paralelo.
- 3 x… → la Red cuenta con tres transformadores en paralelo.
- R_{sc} → valor en p.u de la resistencia referido a los transformadores.
- X_{sc} → valor en p.u de la impedancia referida a los transformadores

Cargas	*P (MW)*	*Q (MVAr)*
Nodo 3	84	26
Nodo 5	34	12
Nodo 7	7.5	5
Nodo 8	52	39
Nodo 9	1.7	1.5

Tabla 6. Cargas de la Red de Distribución. Fuente: [Propia]

6.1 Modelado de la Red de Distribución

A continuación se procede a resolver el problema de flujo de cargas en la ***Red de Distribución*** mediante el empleo del método *Direct Approach*:

6.1.1 Método Direct Approach

En las siguientes tablas se muestran los resultados obtenidos de las tensiones (**U**) y corrientes (**I**) tras resolver el problema de flujo de cargas en la Red de Distribución mediante el empleo del método *Direct Approach*. Estos valores se han calculado gracias al programa *Matlab*, cuyo desarrollo se puede observar en el **ANEXO I**.

Nodos	*Tensiones p.u (Upu)*	*Tensiones Eficaces (V)*
1	1	220 000
2	0.9973	219 400
3	0.9925	131 010
4	0.9917	130 900
5	0.9836	29 507
6	0.9819	29 456
7	0.9485	13 089
8	0.9857	13 603
9	0.9815	13 544

Tabla 7. Tensiones en los nodos de la Red calculadas por el método de DA. Fuente: [Propia]

Líneas	*Corriente p.u (Ipu)*	*Corrientes Eficaces (A)*
12	2.0594	540.45
34	0.4633	202.62
56	0.0950	182.89
89	0.0231	96.643

Tabla 8. Corrientes en las líneas de la Red calculadas por el método de D.A. Fuente: [Propia]

Transformadores	*Corriente p.u*	*Corrientes Eficaces (A)*
23 (devanado primario)	2.0594	540.45
23 (devanado secundario)	2.0182	882.74
45 (devanado primario)	0.4633	202.62
45 (devanado secundario)	0.4586	882.63
67 (devanado primario)	0.0950	182.89
67 (devanado secundario)	0.0950	397.59
38 (devanado primario)	0.6964	304.59
38 (devanado secundario)	0.6825	2855.2

Tabla 9. Corrientes en los transformadores de la Red calculadas por el método de D.A. Fuente: [Propia]

6.2 Verificación de la Red de Distribución

Para verificar que los datos obtenidos tras resolver el problema de flujo de cargas con el método iterativo *Direct Approach* son correctos, los compararemos con los datos que nos proporciona el programa informático ***PowerWorld.***

PowerWorld es un software comercial de análisis y simulación que permite realizar análisis técnicos (límites de tensión, repartos de carga, etc) y económicos en los sistemas de energía eléctrica.

Este programa permite resolver el problema de flujo de cargas ya que con él se pueden representar sistemas de hasta **60 000** nodos, obteniendo posteriormente los resultados en modo de texto o en modo de gráfico. Adicionalmente, permite estudiar la evolución del sistema a lo largo del tiempo, resolviendo, de forma sucesiva e independiente, un flujo de cargas para cada intervalo de análisis **[7]**.

A la hora de llevar a cabo el modelado de la Red de Distribución en *PowerWorld* los datos han de ser introducidos en el siguiente orden:

- En primer lugar se han de representar los nodos de la red.
- A continuación, una vez ubicados los nodos, se procede a insertar las uniones entre ellos, esto es, las líneas y los transformadores, introduciendo en cada uno de ellos sus valores característicos.
- Por último se inserta el generador (*SLACK*) y las cargas que existen en cada nodo.

Una vez se han introducido todos los datos de la Red se puede proceder a la simulación de la misma con el fin de obtener los resultados de las tensiones que aparecen en los nodos. Ésta simulación muestra el flujo de cargas de forma animada, con flechas de colores en las líneas de transporte, las cargas y los generadores, que indican la dirección que sigue dicho flujo a lo largo de la Red.

6.2.1 Verificación Direct Approach con Power World

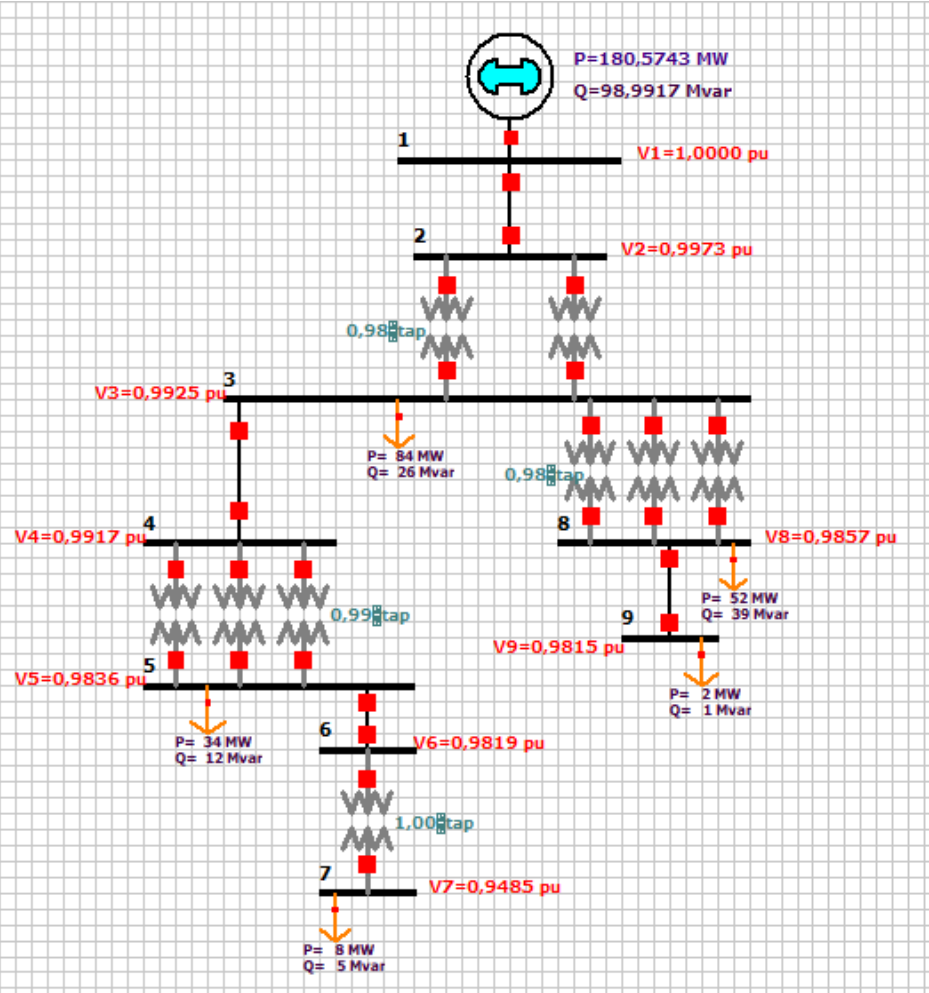

Ilustración 18. Representación en PowerWorld de la Red de Distribución. Fuente [Propia]

	Number	Name	Area Name	Nom kV	PU Volt	Volt (kV)	Angle (Deg)	Load MW	Load Mvar	Gen MW	Gen Mvar
1	1	1	1	220,00	1,00000	220,000	0,00			180,57	98,99
2	2	2	1	220,00	0,99726	219,398	-0,23				
3	3	3	1	132,00	0,99250	131,010	-2,58	84,00	26,00		
4	4	4	1	132,00	0,99167	130,900	-2,60				
5	5	5	1	30,00	0,98359	29,508	-4,45	34,00	12,00		
6	6	6	1	30,00	0,98188	29,456	-4,41				
7	7	7	1	13,80	0,94852	13,090	-6,33	7,50	5,00		
8	8	8	1	13,80	0,98573	13,603	-4,19	52,00	39,00		
9	9	9	1	13,80	0,98148	13,544	-4,16	1,70	1,50		

Tabla 10. Resultados del análisis de la Red de distribución con Power World. Fuente: [Propia]

Como podemos observar si comparamos los resultados obtenidos en la ***Tabla 7*** con los resultados de la ***Tabla 10***, hay una correspondencia perfecta entre ambos programas. Por ello, podemos afirmar que se ha resuelto de manera correcta el problema de flujo de cargas mediante el uso del método iterativo *Direct Approach*.

6.3 Implementación algoritmo WLS mediante Matlab

Para la implementación del método ***WLS*** mediante el uso del programa informático *Matlab*, en primer lugar hemos tenido que corromper tanto las medidas de Potencia Activa y Reactiva, así como las tensiones en los nodos de la Red, puesto que como se ha mencionado anteriormente éstas siguen una distribución normal y por tanto llevarán asociado cierto error, que al resolverlo mediante el método iterativo Direct Approach no se ha tenido en cuenta. Este error será de un **2%** para las medidas de potencia y de un **0.1%** para las medidas de tensión **[6]**. La obtención de estas medidas corrompidas se puede observar en la parte final del **ANEXO I**.

En las siguientes tablas se muestran los resultados obtenidos de las tensiones (**U**) , corrientes (**I**), y potencias (**P** y **Q**) tras realizar la estimación de estado en la Red de Distribución mediante el empleo del algoritmo ***WLS*** desarrollado en *Matlab*. El desarrollo del programa necesario para el cálculo de estos valores se puede observar en el **ANEXO II**.

Nodos	*Tensiones p.u (Upu)*	*Tensiones Eficaces (V)*
1	1.0006	220 120
2	0.9978	219 520
3	0.9929	131 060
4	0.9921	130 950
5	0.9838	29 513
6	0.9821	29 463
7	0.9491	13 098
8	0.9859	13 606
9	0.9809	13 537

Tabla 11. Tensiones en los nodos de la Red calculadas mediante WLS. Fuente: [Propia]

Líneas	*Corriente p.u (Ipu)*	*Corrientes Eficaces (A)*
12	1.173	307.84
34	0.4656	194.91
56	0.0864	166.33
89	0.0271	113.23

Tabla 12. Corrientes en las líneas de la Red calculadas mediante WLS. Fuente: [Propia]

Transformadores	*Corriente p.u*	*Corrientes Eficaces (A)*
23 (devanado primario)	1.0765	282.51
23 (devanado secundario)	1.055	461.43
45 (devanado primario)	0.2301	100.64
45 (devanado secundario)	0.2278	438.38
67 (devanado primario)	0.0674	129.7
67 (devanado secundario)	0.0674	281.95
38 (devanado primario)	0.4876	213.26
38 (devanado secundario)	0.4778	1999.1

Tabla 13. Corrientes en los transformadores de la Red calculadas mediante WLS. Fuente: [Propia]

Cargas	*P (MW)*	*Q (MVar)*
Nodo 1	82.91	26.04
Nodo 2	34.8	12.23
Nodo 3	7.38	4.97
Nodo 4	50.85	39.03
Nodo 5	1.67	1.53

Tabla 14. Cargas en los nodos de la Red calculadas mediante WLS. Fuente: [Propia]

6.4 Implementación algoritmo WLS mediante Matlab y Python con Perfil de Cargas Diario

Para la implementación del algoritmo ***WLS*** en *Matlab* y *Python* hemos utilizado un perfil de cargas diario que ha sido obtenido de la base de datos austriaca ***ADRES-Concept - Autonomous Decentralised Renewable Energy Systems.***

El conjunto de datos ADRES incluye la potencia activa y reactiva eléctrica. Las mediciones se tomaron durante una semana en el invierno (entre septiembre y diciembre de 2009) y una en el verano (entre mayo y octubre de 2010) en 30 hogares diferentes en la Alta Austria. Se tomaron con una resolución de 1 segundo (valores RMS)**[8]**.

Gracias a esta base de datos se nos proporcionan una serie de medidas de potencia activa y reactiva (1440) para los nodos de la red. Como constamos de 1440 pares de medidas de potencias, tendremos que resolver el problema de flujo de cargas en la *Red de Distribución* para obtener las tensiones de funcionamiento de la misma asociadas a estas potencias. Además una vez obtenidas, estas medidas han de ser escaladas en primer lugar, para adaptarse a los valores de nuestra *Red de Distribución*, y corrompidas posteriormente incluyéndoles el error asociado a cada una de ellas. La resolución del problema de flujo de cargas (*método Direct Approach*), así como la modificación de las medidas de tensión se pueden observar en el **ANEXO III**. La modificación de las medidas de potencia se puede observar en el **ANEXO IV**. Gracias a estas medidas podremos proceder a realizar una estimación de estado diaria de la *Red de Distribución Eléctrica*.

En las siguientes tablas se muestran los resultados obtenidos en la última medición de las tensiones (**U**), corrientes (**I**), y potencias (**P** y **Q**) tras realizar la estimación de estado en la Red de Distribución mediante el empleo del algoritmo ***WLS*** desarrollado tanto en *Matlab* como en *Python*. El desarrollo de los programas necesarios para el cálculo de estos valores se puede observar en los **ANEXOS V** y **VI**.

Nodos	*Tensiones p.u (Upu)*	*Tensiones Eficaces (V)*
1	0.9999	219 970
2	0.9991	219 798
3	1.0122	133 626
4	1.0120	133 589
5	1.0173	30 518
6	1.0167	30 500
7	1.0084	13 915
8	1.0254	14 150
9	1.0237	14 127

Tabla 15. Tensiones en los nodos de la Red calculadas mediante WLS para perfil de cargas diario. Fuente: [Propia]

Líneas	*Corriente p.u (Ipu)*	*Corrientes Eficaces (A)*
12	0.3336	87.55
34	0.1492	65.26
56	0.0296	57.04
89	0.0091	38.84

Tabla 16. Corrientes en las líneas de la Red calculadas mediante WLS para perfil de cargas diario. Fuente: [Propia]

Transformadores	*Corriente p.u*	*Corrientes Eficaces (A)*
23 (devanado primario)	0.3047	79.97
23 (devanado secundario)	0.2984	130.68
45 (devanado primario)	0.0627	27.44
45 (devanado secundario)	0.0621	119.53
67 (devanado primario)	0.0169	32.61
67 (devanado secundario)	0.0169	70.90
38 (devanado primario)	0.1354	59.21
38 (devanado secundario)	0.1327	555.05

Tabla 17. Corrientes en los trafos de la Red calculadas mediante WLS para perfil de cargas diario. Fuente: [Propia]

Cargas	*P (MW)*	*Q (MVar)*
Nodo 1	28.373	8.8627
Nodo 2	13.317	3.5073
Nodo 3	2.791	1.2529
Nodo 4	24.472	10.381
Nodo 5	0.6263	0.5356

Tabla 18. Cargas en los nodos de la Red calculadas mediante WLS para perfil de cargas diario. Fuente: [Propia]

6.5 Datos tratados con Raspberry

Una vez se ha desarrollado en Python el algoritmo de estimación de estado *WLS*, se procede a introducir en el mismo un código asociado a un medidor en tiempo real, el cual se comunica con nuestro algoritmo gracias al protocolo ***moodbus-TCP/IP***. El nombre de este medidor es *Power Meter PowerLogic PM5560 de Schneider*. Se trata de una central de medida que proporciona datos en tiempo real de potencia activa y reactiva, tensión, corriente y frecuencia **[9]**.

Ilustración 19. Medidor Power Meter PowerLogic PM5560 de Schneider. Fuente: [9]

Este medidor nos proporcionará unas medidas reales de potencia activa y reactiva, y de tensión, las cuales serán intercambiadas por las utilizadas en el nodo **9** de nuestra *Red de Distribución.* Las medidas serán obtenidas de una bombilla doméstica conectada a la red de distribución eléctrica y será necesario realizar un escalado de las mismas para poder adaptarlas a los valores de nuestra *Red de Distribución*. Se puede observar como se ha realizado este escalado en el **ANEXO VII**.

Finalmente este código será introducido en nuestra ***Raspberry Pi3 Model B***$^{+}$ gracias a la cual podremos visualizar en tiempo real la evolución de dichas medidas.

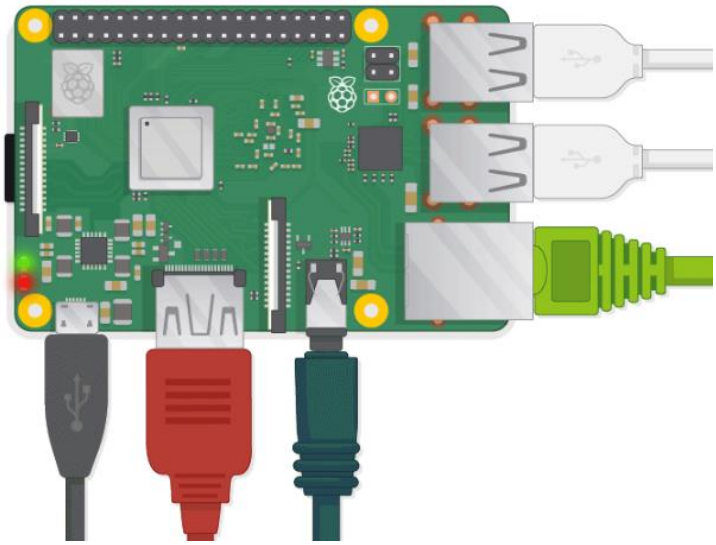

Ilustración 20. Raspberry Pi3 Model B+. Fuente: [10]

La ***Raspberry Pi3 Model B***$^{+}$ es un pequeño ordenador que consta de **[10]**:

- 4 puertos USB: se usan para conectar un ratón y un teclado necesarios para poder operar la Raspberry Pi. En ellos también se pueden conectar memorias USB.
- Ranura para tarjetas SD: aquí es donde se almacenan el software del sistema operativo y sus archivos.
- Puerto Ethernet: se usa para conectar las Raspberry Pi a internet. También puede ser conectada mediante Wifi.
- Puerto HDMI: lugar donde se conecta el monitor para la visualización de los datos.
- Conector de alimentación micro USB: lugar para conectar la fuente de alimentación.
- Puertos GPIO: permiten conectar componentes electrónicos como LED's a la Raspberry Pi.

El desarrollo del programa necesario para el cálculo de los valores de la *Red de Distribución* en tiempo real se puede observar en el **ANEXO VII**.

Los resultados obtenidos de las tensiones (**U**), corrientes (**I**), y potencias (**P** y **Q**) tras realizar la estimación de estado en tiempo real de la *Red de Distribución* mediante el empleo del algoritmo ***WLS*** desarrollado en *Python* podrán ser visualizados in situ durante la defensa del ***Trabajo Fin de Máster***.

A continuación se muestran unas gráficas de cómo será la visualización en tiempo real de la evolución de las potencias y las tensiones de 70 muestras tras utilizar la estimación de estado desarrollada en *Python*.

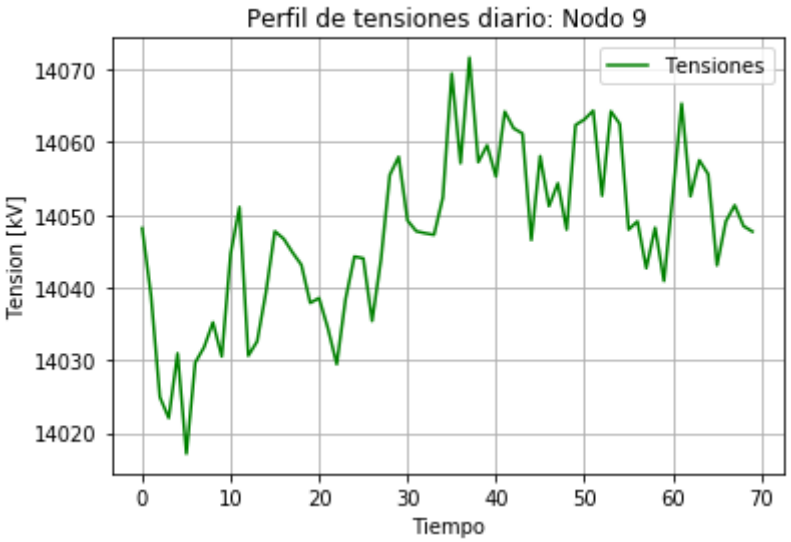

Ilustración 21. Perfil de tensiones diarias en tiempo real. Fuente: [Propia]

Ilustración 22. Perfil de potencias diarias en tiempo real. Fuente: [Propia]

7. Conclusiones

Una vez se ha llevado a cabo el análisis del problema de flujo de cargas en la *Red de Distribución* a través del método iterativo **Direct Approach** y tras el desarrollo del algoritmo de estimación de estado **WLS** se ha llegado a las siguientes conclusiones.

7.1 Resolución del Flujo de Cargas mediante Direct Approach

Podemos observar que el método *Direct Approach* resulta ser un método muy rápido para la resolución del problema de flujo de cargas de una Red de Distribución Eléctrica. Tan sólo emplea **10** iteraciones y un tiempo de **0.66** segundos para resolver el problema, lo que supone una mejora bastante notable respecto a los métodos tradicionales. Esto se debe a que sólo emplea como datos de partida los valores de corrientes y tensiones de las ramas y los nodos de la Red.

7.2 Algoritmo de Estimación de Estado WLS

El hecho de desarrollar el algoritmo en *Python* tras haberlo desarrollado en *Matlab* se debe a que gracias a su desarrollo en *Python*, podemos disponer de un dispositivo autónomo que puede realizar el algoritmo de estimación de estado *WLS* sin necesidad de depender de un software propietario, como es el caso de *Matlab*.

Podemos observar que mediante el uso del lenguaje de programación *Python* y con un código relativamente sencillo, podemos desarrollar el algoritmo de estimación de estado *WLS* que nos permite obtener rápidamente las tensiones, corrientes de y las potencias de funcionamiento de la Red de Distribución Eléctrica. Además estas medidas serán muy aproximadas a las medidas reales (que no pueden ser obtenidas con precisión absoluta), puesto que ya llevan incorporado el error procedente de los transformadores de la Red.

7.3 Algoritmo de Estimación de Estado WLS en Tiempo Real

Gracias a la implementación de nuestro algoritmo *WLS* desarrollado en *Python* y con dos pequeños dispositivos electrónicos como son el *Power Meter PowerLogic PM5560 y* la *Raspberry Pi3 Model* B^+ , podemos obtener y visualizar en tiempo real las tensiones, corrientes y potencias de funcionamiento de un dispositivo eléctrico como en nuestro caso el de una bombilla doméstica. Este algoritmo nos puede resultar de gran utilidad puesto que si en lugar de una bombilla, midiésemos las tensiones, corrientes y potencias de otros aparatos eléctricos, podríamos estar visualizando en tiempo real su consumo, y de esta manera podríamos detectar fallos en el correcto funcionamiento de los mismos, puesto que las medidas obtenidas serían mayores que las teóricas.

ANEXO I: Programación Direct Approach mediante Matlab

A continuación se muestra el desarrollo del método Direct Approach resuelto mediante el programa *MATLAB*. De esta forma se obtienen los valores de las tensiones y corrientes en la Red de Distribución. Al final del mismo se calculan los valores corrompidos de las tensiones y las potencias utilizadas para el desarrollo del algoritmo de estimación de estado WLS de manera genérica.

```
%RESOLUCIÓN DEL PROBLEMA DE FLUJO DE CARGAS MEDIANTE EL MÉTODO DIRECT
%APPROACH, PARA OBTENER LAS POTENCIAS Y TENSIONES DE FUNCIONAMIENTO DE
UNA
%RED DE DISTRIBUCIÓN DE 9 NODOS
clear all
tic
%Introducimos la Potencia Compleja que usaremos como base:
Sbase=100e6;
%Introducimos los datos de la Red de Distribución y calculamos las
%impedancias y admitancias en p.u
%Transformador 23
a23=0.98;
Ubase23(1)=220000;
Ubase23(2)=132000;
Ibase23=Sbase./(sqrt(3)*Ubase23);
Zbase23=(Ubase23.*Ubase23)/Sbase;
S23=270e6;
Zbaset1=(Ubase23(1)^2)/S23;
Z23=(0.009+0.129j)/2;
Z23o=Z23*Zbaset1;
Z23pu=Z23o/Zbase23(1);
Y23=1/Z23pu;
%Transformador 45
a45=0.99;
Ubase45(1)=132000;
Ubase45(2)=30000;
Ibase45=Sbase./(sqrt(3)*Ubase45);
Zbase45=(Ubase45.*Ubase45)/Sbase;
S45=37.5e6;
Zbaset2=(Ubase45(1)^2)/S45;
Z45=(0.009+0.09j)/3;
Z45o=Z45*Zbaset2;
Z45pu=Z45o/Zbase45(1);
Y45=1/Z45pu;
%Transformador 67
a67=1;
Ubase67(1)=30000;
Ubase67(2)=13800;
Ibase67=Sbase./(sqrt(3)*Ubase67);
Zbase67=(Ubase67.*Ubase67)/Sbase;
S67=10e6;
```

```
Zbaset3=(Ubase67(1)^2)/S67;
Z67=0.0095+0.048j;
Z67o=Z67*Zbaset3;
Z67pu=Z67o/Zbase67(1);
Y67=1/Z67pu;
%Transformador 38
a38=0.98;
Ubase38(1)=132000;
Ubase38(2)=13800;
Ibase38=Sbase./(sqrt(3)*Ubase38);
Zbase38=(Ubase38.*Ubase38)/Sbase;
S38=50e6;
Zbaset4=(Ubase38(1)^2)/S38;
Z38=(0.0092+0.085j)/3;
Z38o=Z38*Zbaset4;
Z38pu=Z38o/Zbase38(1);
Y38=1/Z38pu;
%Linea 12
Z12=(0.025+0.240j)*4.7;
Z12pu=Z12/Zbase23(1);
Y12=1/Z12pu;
%Linea 34
Z34=(0.161+0.151j)*1.5;
Z34pu=Z34/Zbase23(2);
Y34=1/Z34pu;
%Linea 56
Z56=(0.568+0.133j)*0.3;
Z56pu=Z56/Zbase45(2);
Y56=1/Z56pu;
%Linea 89
Z89=(0.161+0.112j)*1.8;
Z89pu=Z89/Zbase38(2);
Y89=1/Z89pu;

%Calculamos el valor de las admitancias en los transformadores
%Transformador 23
Y123=((1-a23)/a23^2)*Y23;
Y223=(1/a23)*Y23;
Y323=((a23-1)/a23)*Y23;
Z123=1/Y123;
Z223=1/Y223;
Z323=1/Y323;
%Transformador 45
Y145=((1-a45)/a45^2)*Y45;
Y245=(1/a45)*Y45;
Y345=((a45-1)/a45)*Y45;
Z145=1/Y145;
Z245=1/Y245;
Z345=1/Y345;
%Transformador 67
Y167=((1-a67)/a67^2)*Y67;
Y267=(1/a67)*Y67;
Y367=((a67-1)/a67)*Y67;
Z167=1/Y167;
Z267=1/Y267;
Z367=1/Y367;
%Transformador 38
```

```
Y138=((1-a38)/a38^2)*Y38;
Y238=(1/a38)*Y38;
Y338=((a38-1)/a38)*Y38;
Z138=1/Y138;
Z238=1/Y238;
Z338=1/Y338;
%Introducimos el valor de los consumos que se producen en cada nodo.
%Nodos:
P3=84e6;
Q3=26e6;
P5=34e6;
Q5=12e6;
P7=7.5e6;
Q7=5e6;
P8=52e6;
Q8=39e6;
P9=1.7e6;
Q9=1.5e6;
%Construimos un vector con los valores en p.u de las potencias activas
y
%reactivas de la red. El signo - se debe a que se trata de nodos de
%consumo.
Ppu=[0,-P3/Sbase,0,-P5/Sbase,0,-P7/Sbase,-P8/Sbase,-P9/Sbase];
Qpu=[0,-Q3/Sbase,0,-Q5/Sbase,0,-Q7/Sbase,-Q8/Sbase,-Q9/Sbase];
Spu=Ppu+Qpu*1i;

%Cálculo matrices para desarrollar el método de Direct Approach. Tiene
%dimensión 8x8, ya que no se ha de introducir el nodo SLACK.
BIBC= [1 1 1 1 1 1 1 1
    0 1 1 1 1 1 1 1
    0 0 1 1 1 1 0 0
    0 0 0 1 1 1 0 0
    0 0 0 0 1 1 0 0
    0 0 0 0 0 1 0 0
    0 0 0 0 0 0 1 1
    0 0 0 0 0 0 0 1];

BCBV= [Z12pu 0 0 0 0 0 0 0
    Z12pu Z223 0 0 0 0 0 0
    Z12pu Z223 Z34pu 0 0 0 0 0
    Z12pu Z223 Z34pu Z245 0 0 0 0
    Z12pu Z223 Z34pu Z245 Z56pu 0 0 0
    Z12pu Z223 Z34pu Z245 Z56pu Z267 0 0
    Z12pu Z223 0 0 0 0 Z238 0
    Z12pu Z223 0 0 0 0 Z238 Z89pu];

%Calculamos el valor de las tensiones en los nodos.
N=8;
tolerancia=10e-10;
iteraciones=0;
error=tolerancia+1;
IncVant=0;
V0=ones(1,N);
Upu=V0;
while error>tolerancia;
    iteraciones=iteraciones+1;
    Ibus1=conj(Spu./Upu);
```

```
    %Vector impedancias transformadores con tomas
    YT=[Y123 Y323+Y138 Y145 Y345 Y167 Y367 Y338 0];
    IT=YT.*Upu;
    Ibusfinal=Ibus1-IT;
    B=BIBC*Ibusfinal.';
    IncV=BCBV*B;
    Upu=V0+IncV.';
    error=max(abs(IncV-IncVant));
    IncVant=IncV;
end
%Una vez resuelto el problema de flujo de cargas, ya puedo calcular el
%valor de las tensiones y de las corrientes que tendremos en cada nodo
de la Red de
%distribución.
%Tensiones
U1=Ubase23(1);
U2=Ubase23(1)*abs(Upu(1));
U3=Ubase23(2)*abs(Upu(2));
U4=Ubase45(1)*abs(Upu(3));
U5=Ubase45(2)*abs(Upu(4));
U6=Ubase67(1)*abs(Upu(5));
U7=Ubase67(2)*abs(Upu(6));
U8=Ubase38(2)*abs(Upu(7));
U9=Ubase38(2)*abs(Upu(8));
Ufinales=[U1;U2;U3;U4;U5;U6;U7;U8;U9];
%Corrientes en los nodos de la Red y en los Transformadores.
%Linea12
I12pu=(V0(1)-Upu(1))/Z12pu;
I12puRMS=abs(I12pu);
I12=I12pu*Ibase23(1);
I12RMS=abs(I12);
%Transformador23
%Devanado primario
I23pup=((Upu(1)-Upu(2))*Y223)+(Upu(1)*Y123);
I23puRMSp=abs(I23pup);
I23p=I23pup*Ibase23(1);
I23RMSp=abs(I23p);
%Devanado secundario
I23pus=((Upu(1)-Upu(2))*Y223)-(Upu(2)*Y323);
I23puRMSs=abs(I23pus);
I23s=I23pus*Ibase23(2);
I23RMSs=abs(I23s);
%Linea34
I34pu=(Upu(2)-Upu(3))/Z34pu;
I34puRMS=abs(I34pu);
I34=I34pu*Ibase23(2);
I34RMS=abs(I34);
%Transformador45
%Devanado primario
I45pup=((Upu(3)-Upu(4))*Y245)+(Upu(3)*Y145);
I45puRMSp=abs(I45pup);
I45p=I45pup*Ibase45(1);
I45RMSp=abs(I45p);
%Devanado secundario
I45pus=((Upu(3)-Upu(4))*Y245)-(Upu(4)*Y345);
I45puRMSs=abs(I45pus);
I45s=I45pus*Ibase45(2);
```

```
I45RMSs=abs(I45s);
%Linea56
I56pu=(Upu(4)-Upu(5))/Z56pu;
I56puRMS=abs(I56pu);
I56=I56pu*Ibase45(2);
I56RMS=abs(I56);
%Transformador67
%Devanado primario
I67pup=((Upu(5)-Upu(6))*Y267)+(Upu(5)*Y167);
I67puRMSp=abs(I67pup);
I67p=I67pup*Ibase67(1);
I67RMSp=abs(I67p);
%Devanado secundario
I67pus=((Upu(5)-Upu(6))*Y267)-(Upu(6)*Y367);
I67puRMSs=abs(I67pus);
I67s=I67pus*Ibase67(2);
I67RMSs=abs(I67s);
%Transformador38
%Devanado primario
I38pup=((Upu(2)-Upu(7))*Y238)+(Upu(2)*Y138);
I38puRMSp=abs(I38pup);
I38p=I38pup*Ibase38(1);
I38RMSp=abs(I38p);
%Devanado secundario
I38pus=((Upu(2)-Upu(7))*Y238)-(Upu(7)*Y338);
I38puRMSs=abs(I38pus);
I38s=I38pus*Ibase38(2);
I38RMSs=abs(I38s);
%Linea89
I89pu=(Upu(7)-Upu(8))/Z89pu;
I89puRMS=abs(I89pu);
I89=I89pu*Ibase38(2);
I89RMS=abs(I89);
%Creo vectores con las corrientes finales en las líneas y en los
%transformadores:
Ilineaspu=[I12puRMS,I34puRMS,I56puRMS,I89puRMS];
Ilineas=[I12RMS,I34RMS,I56RMS,I89RMS];
Itrafospup=[I23puRMSp,I45puRMSp,I67puRMSp,I38puRMSp];
Itrafosp=[I23RMSp,I45RMSp,I67RMSp,I38RMSp];
Itrafospus=[I23puRMSs,I45puRMSs,I67puRMSs,I38puRMSs];
Itrafoss=[I23RMSs,I45RMSs,I67RMSs,I38RMSs];
toc

%Para poder llevar a cabo la resolución del método de estimación de
estado
%necesito corromper los valores obtenidos tras resolver el problema de
%flujo de cargas.
%Una vez tengo resuelta la red ya puedo calcular los valores
corrompidos de la misma
%Corrompo los valores de P, los cuales presentarán una variación de un
2%
%respecto a los originales.
rng('default')
rng(1)
eP=normrnd(0,0.02,[1 5]); %Distribución normal de media 0 y varianza
0.02
P3c=P3+P3*eP(1);
```

```
P5c=P5+P5*eP(2);
P7c=P7+P7*eP(3);
P8c=P8+P8*eP(4);
P9c=P9+P9*eP(5);
%Corrompo los valores de Q, los cuales presentarán una variación de un
2%
%respecto a los originales.
rng('default')
rng(3)
eQ=normrnd(0,0.02,[1 5]); %Distribución normal de media 0 y varianza
0.02
Q3c=Q3+Q3*eQ(1);
Q5c=Q5+Q5*eQ(2);
Q7c=Q7+Q7*eQ(3);
Q8c=Q8+Q8*eQ(4);
Q9c=Q9+Q9*eQ(5);
%Corrompo los valores de U, los cuales presentarán una variación de un
0.1%
%respecto a los originales.
rng('default')
rng(5)
eU=normrnd(0,0.001,[1 9]); %Distribución normal de media 0 y varianza
0.001
U1c=U1+U1*eU(1);
U2c=U2+U2*eU(2);
U3c=U3+U3*eU(3);
U4c=U4+U4*eU(4);
U5c=U5+U5*eU(5);
U6c=U6+U6*eU(6);
U7c=U7+U7*eU(7);
U8c=U8+U8*eU(8);
U9c=U9+U9*eU(9);

%Ahora guardo los valores corrompidos
save('Pcfile.mat','P3c','P5c','P7c','P8c','P9c'); %Aquí guardo los
valores de las Potencias Activas corrompidas
save('Qcfile.mat','Q3c','Q5c','Q7c','Q8c','Q9c'); %Aquí guardo los
valores de las Potencias Reactivas corrompidas
save('Ucfile.mat','U1c','U2c','U3c','U4c','U5c','U6c','U7c','U8c','U9c
'); %Aquí guardo los valores de las Tensiones corrompidas
```

El error cometido por el programa es de **$3.015 \cdot 10^{-10}$.**

El tiempo de ejecución del programa es de **0.6633 segundos**.

El número de iteraciones realizadas es de: **10**.

ANEXO II: Programación WLS Genérico mediante Matlab

A continuación se muestra el desarrollo del algoritmo WLS resuelto mediante el programa *MATLAB*. De esta forma se obtienen los valores de las tensiones y corrientes en la Red de Distribución.

```
%DESARROLLO DEL ALGORITMO DE ESTIMACIÓN DE ESTADO PARA UNA RED DE
%DISTRIBUCIÓN ELÉCTRICA DE 9 NODOS
tic
%Introducimos la Potencia Compleja que usaremos como base:
Sbase=100e6;
%Introducimos los datos de la Red de Distribución y calculamos las
%impedancias y admitancias en p.u
%Transformador 23
a23=0.98;
Ubase23(1)=220000;
Ubase23(2)=132000;
Ibase23=Sbase./(sqrt(3)*Ubase23);
Zbase23=(Ubase23.*Ubase23)/Sbase;
S23=270e6;
Zbaset1=(Ubase23(1)^2)/S23;
Z23=(0.009+0.129j)/2;
Z23o=Z23*Zbaset1;
Z23pu=Z23o/Zbase23(1);
Y23=1/Z23pu;
%Transformador 45
a45=0.99;
Ubase45(1)=132000;
Ubase45(2)=30000;
Ibase45=Sbase./(sqrt(3)*Ubase45);
Zbase45=(Ubase45.*Ubase45)/Sbase;
S45=37.5e6;
Zbaset2=(Ubase45(1)^2)/S45;
Z45=(0.009+0.09j)/3;
Z45o=Z45*Zbaset2;
Z45pu=Z45o/Zbase45(1);
Y45=1/Z45pu;
%Transformador 67
a67=1;
Ubase67(1)=30000;
Ubase67(2)=13800;
Ibase67=Sbase./(sqrt(3)*Ubase67);
Zbase67=(Ubase67.*Ubase67)/Sbase;
S67=10e6;
Zbaset3=(Ubase67(1)^2)/S67;
Z67=0.0095+0.048j;
Z67o=Z67*Zbaset3;
Z67pu=Z67o/Zbase67(1);
Y67=1/Z67pu;
%Transformador 38
a38=0.98;
```

```
Ubase38(1)=132000;
Ubase38(2)=13800;
Ibase38=Sbase./(sqrt(3)*Ubase38);
Zbase38=(Ubase38.*Ubase38)/Sbase;
S38=50e6;
Zbaset4=(Ubase38(1)^2)/S38;
Z38=(0.0092+0.085j)/3;
Z38o=Z38*Zbaset4;
Z38pu=Z38o/Zbase38(1);
Y38=1/Z38pu;
%Linea 12
Z12=(0.025+0.240j)*4.7;
Z12pu=Z12/Zbase23(1);
Y12=1/Z12pu;
%Linea 34
Z34=(0.161+0.151j)*1.5;
Z34pu=Z34/Zbase23(2);
Y34=1/Z34pu;
%Linea 56
Z56=(0.568+0.133j)*0.3;
Z56pu=Z56/Zbase45(2);
Y56=1/Z56pu;
%Linea 89
Z89=(0.161+0.112j)*1.8;
Z89pu=Z89/Zbase38(2);
Y89=1/Z89pu;

%Cálculo de las admitancias en los transformadores
%Transformador 23
Y123=((1-a23)/a23^2)*Y23;
Y223=(1/a23)*Y23;
Y323=((a23-1)/a23)*Y23;
Z123=1/Y123;
Z223=1/Y223;
Z323=1/Y323;
%Transformador 45
Y145=((1-a45)/a45^2)*Y45;
Y245=(1/a45)*Y45;
Y345=((a45-1)/a45)*Y45;
Z145=1/Y145;
Z245=1/Y245;
Z345=1/Y345;
%Transformador 67
Y167=((1-a67)/a67^2)*Y67;
Y267=(1/a67)*Y67;
Y367=((a67-1)/a67)*Y67;
Z167=1/Y167;
Z267=1/Y267;
Z367=1/Y367;
%Transformador 38
Y138=((1-a38)/a38^2)*Y38;
Y238=(1/a38)*Y38;
Y338=((a38-1)/a38)*Y38;
Z138=1/Y138;
Z238=1/Y238;
Z338=1/Y338;
```

```
%Construimos la matriz de admitancias:
Y= [Y12 -Y12 0 0 0 0 0 0 0
    -Y12 Y12+Y123+Y223 -Y223 0 0 0 0 0 0
    0 -Y223 Y223+Y323+Y34+Y138+Y238 -Y34 0 0 0 -Y238 0
    0 0 -Y34 Y34+Y145+Y245 -Y245 0 0 0 0
    0 0 0 -Y245 Y245+Y345+Y56 -Y56 0 0 0
    0 0 0 0 -Y56 Y56+Y167+Y267 -Y267 0 0
    0 0 0 0 0 -Y267 Y267+Y367 0 0
    0 0 -Y238 0 0 0 0 Y238+Y338+Y89 -Y89
    0 0 0 0 0 0 0 -Y89 Y89];

%Buses P Q y U. Tomo como datos de partida los valores corrompidos
obtenidos
%en Direct Approach
load('Pcfile.mat')
load('Qcfile.mat')
load('Ucfile.mat')
%Potencias activas
P3pu=-P3c/Sbase;
P5pu=-P5c/Sbase;
P7pu=-P7c/Sbase;
P8pu=-P8c/Sbase;
P9pu=-P9c/Sbase;
Ppu=[0;P3pu;0;P5pu;0;P7pu;P8pu;P9pu]; %Vector de potencias activas p.u
%Potencias reactivas
Q3pu=-Q3c/Sbase;
Q5pu=-Q5c/Sbase;
Q7pu=-Q7c/Sbase;
Q8pu=-Q8c/Sbase;
Q9pu=-Q9c/Sbase;
Qpu=[0;Q3pu;0;Q5pu;0;Q7pu;Q8pu;Q9pu]; %Vector de potencias reactivas
p.u
%Tensiones
U1puc=U1c/Ubase23(1);
U2puc=U2c/Ubase23(1);
U3puc=U3c/Ubase23(2);
U4puc=U4c/Ubase45(1);
U5puc=U5c/Ubase45(2);
U6puc=U6c/Ubase67(1);
U7puc=U7c/Ubase67(2);
U8puc=U8c/Ubase38(2);
U9puc=U9c/Ubase38(2);
Upuc=[U1puc;U2puc;U3puc;U4puc;U5puc;U6puc;U7puc;U8puc;U9puc]; %Vector
de tensions p.u

%COMIENZO IMPLEMENTACIÓN WLS
N=9; %número de buses y de U
Gpu=real(Y); %Parte real de la matriz de admitancias
Bpu=imag(Y); %Parte imaginaria de la matriz de admitancias.
R=zeros(25); %Matriz diagonal, cuyos valores son las varianzas de las
P, Q y U. Tengo 25 medidas. Tengo 8P, 8Q y 9U
varpq=0.02; %Covarianza para las medidas de P y Q
varu=0.001; %Covarianza para las medidas de U
varo=0.001/10; %Covarianza para los buses en que no tenemos P o Q (10
veces más pequeña que la de U)
%Introduzco las covarianzas para P
for i=1;
```

```
    R(i,i)=varo;
end
for i=2
    R(i,i)=varpq;
end
for i=3
    R(i,i)=varo;
end
for i=4
    R(i,i)=varpq;
end
for i=5
    R(i,i)=varo;
end
for i=6:8
    R(i,i)=varpq;
end
%Introduzco las covarianzas para Q
for i=9;

    R(i,i)=varo;
end
for i=10
    R(i,i)=varpq;
end
for i=11
    R(i,i)=varo;
end
for i=12
    R(i,i)=varpq;
end
for i=13
    R(i,i)=varo;
end
for i=14:16
    R(i,i)=varpq;
end
%Introduzco las covarianzas para U
for i=17:25
    R(i,i)=varu;
end
W=inv(R); %Matriz inversa de la matriz de covarianzas.
z=[Ppu;Qpu;Upuc]; %Vector de partida formado por los valores de P, Q y
U.
Upu=ones(N,1); %Iniciamos en arranque plano V=1. Valores en p.u
teta=zeros(N,1); %Iniciamos en arranque plano teta=0
X=[teta(2:end);Upu]; %Vector estado (2N-1; 8 tetas (0 SLACK) y 9 U)
tolerancia=10e-6;
error=tolerancia+1;
iteraciones=0;
%Defino un vector con las posiciones de mis P y Q
busPQ=[2;3;4;5;6;7;8;9];
numberPQ=length(busPQ);

while error>tolerancia;
    iteraciones=iteraciones+1;
```

```
    %tengo que calcular las h que dependen de U, P y Q
    hu=Upu;
    hp=zeros(numberPQ,1); %tengo 8 medidas de P
    hq=zeros(numberPQ,1); %tengo 8 medidas de Q

    for i = 1:numberPQ %tengo 8 hp, ya que tengo 8P
(P2,P3,P4,P5,P6,P7,P8,P9)
     j=busPQ(i);
     for k = 1:N
            hp(i) = hp(i) + Upu(j)*Upu(k)*(Gpu(j,k)*cos(teta(j)-
teta(k)) + Bpu(j,k)*sin(teta(j)-teta(k)));
     end
    end

    for i = 1:numberPQ %tengo 8 hq, ya que tengo 8Q
      j=busPQ(i);
      for k = 1:N
            hq(i) = hq(i) + Upu(j)*Upu(k)*(Gpu(j,k)*sin(teta(j)-
teta(k)) - Bpu(j,k)*cos(teta(j)-teta(k)));
      end
    end
    h=[hp;hq;hu];

    r=z-h;%residuo

    %Tengo que calcular ahora el Jacobiano H
    HUteta = zeros(N,N-1); %derivada de U respecto a teta (9x8, ya que
teta1)

    HUU = zeros(N,N); %derivada de U respecto a U (9x9)
    for i=1:N
        for k = 1:N
            if i == k
                HUU(i,k) = 1;
            end
        end
     end

     HPteta = zeros(numberPQ,N-1); %Derivada de P respecto a teta
(8x8) %Tengo 8 medidas de P
        for i = 1:numberPQ
            m=busPQ(i); %Estas son las posiciones de mis tensiones
            for k = 1:N-1 %Estos son mis ángulos
                if k+1 == m
                    for n=1:N
                        HPteta(i,k) = HPteta(i,k) + Upu(m)* Upu(n)*(-
Gpu(m,n)*sin(teta(m)-teta(n)) + Bpu(m,n)*cos(teta(m)-teta(n)));
                    end
                    HPteta(i,k)= HPteta(i,k)- Upu(m)^2*Bpu(m,m);
                else
                    HPteta(i,k) = Upu(m)*
Upu(k+1)*(Gpu(m,k+1)*sin(teta(m)-teta(k+1)) - Bpu(m,k+1)*cos(teta(m)-
teta(k+1)));
                end
            end
        end
```

```
      HPU = zeros(numberPQ,N); %Derivada de P respecto a U (8x9)
        for i = 1:numberPQ
            m=busPQ(i);
            for k = 1:N
                if k == m
                    for n=1:N
                        HPU(i,k) = HPU(i,k) +
Upu(n)*(Gpu(m,n)*cos(teta(m)-teta(n)) + Bpu(m,n)*sin(teta(m)-
teta(n)));
                    end
                    HPU(i,k)=HPU(i,k)+ Upu(m)*Gpu(m,m);
                else
                    HPU(i,k) = Upu(m)*(Gpu(m,k)*cos(teta(m)-teta(k)) +
Bpu(m,k)*sin(teta(m)-teta(k)));
                end
            end
        end

      HQteta = zeros(numberPQ,N-1); %Derivada de Q respecto a teta
(8x8)
        for i = 1:numberPQ
            m=busPQ(i);
            for k = 1:N-1
                if k+1 == m
                    for n=1:N
                        HQteta(i,k) = HQteta(i,k) + Upu(m)*
Upu(n)*(Gpu(m,n)*cos(teta(m)-teta(n)) + Bpu(m,n)*sin(teta(m)-
teta(n)));
                    end
                    HQteta(i,k) = HQteta(i,k)- Upu(m)^2*Gpu(m,m);
                else
                    HQteta(i,k) = Upu(m)* Upu(k+1)*(-
Gpu(m,k+1)*cos(teta(m)-teta(k+1)) - Bpu(m,k+1)*sin(teta(m)-
teta(k+1)));
                end
            end
        end

      HQU = zeros(numberPQ,N); %Derivada de Q respecto a U (8x9)
        for i = 1:numberPQ
            m = busPQ(i);
            for k = 1:N
                if k == m
                    for n = 1:N
                        HQU(i,k) = HQU(i,k) +
Upu(n)*(Gpu(m,n)*sin(teta(m)-teta(n)) - Bpu(m,n)*cos(teta(m)-
teta(n)));
                    end
                    HQU(i,k)= HQU(i,k) - Upu(m)*Bpu(m,m);
                else
                    HQU(i,k) = Upu(m)*(Gpu(m,k)*sin(teta(m)-teta(k)) -
Bpu(m,k)*cos(teta(m)-teta(k)));
                end
            end
        end
```

```
        %Construyo el Jacobiano
        H=[HPteta HPU; HQteta HQU; HUteta HUU];
        %Calculo G
        G = transpose(H)*W*H;
        %Calculo el incremento de X (vector de estado)
        incX=G\(transpose(H)*W*r);
        X=X+incX;
        %Obtengo las medidas de los ángulos.
        teta(2:end)=X(1:8);
        %Obtengo las medidas de las tensiones.
        Upu=X(9:end);
        error=max(abs(incX));
end
%Una vez resuelto el problema de flujo de cargas, ya puedo calcular el
%valor de las tensiones y de las corrientes que tendremos en cada nodo
%y cada líneas de la Red de Distribución.
%Tensiones
U1=Ubase23(1)*abs(Upu(1));
U2=Ubase23(1)*abs(Upu(2));
U3=Ubase23(2)*abs(Upu(3));
U4=Ubase45(1)*abs(Upu(4));
U5=Ubase45(2)*abs(Upu(5));
U6=Ubase67(1)*abs(Upu(6));
U7=Ubase67(2)*abs(Upu(7));
U8=Ubase38(2)*abs(Upu(8));
U9=Ubase38(2)*abs(Upu(9));
Ufinal=[U1;U2;U3;U4;U5;U6;U7;U8;U9];
%Corrientes
%Linea12
I12pu=(Upu(1)-Upu(2))/Z12pu;
I12puRMS=abs(I12pu);
I12=I12pu*Ibase23(1);
I12RMS=abs(I12);
%Transformador23
%Devanado primario
I23pup=((Upu(2)-Upu(3))*Y223)+(Upu(2)*Y123);
I23puRMSp=abs(I23pup);
I23p=I23pup*Ibase23(1);
I23RMSp=abs(I23p);
%Devanado secundario
I23pus=((Upu(2)-Upu(3))*Y223)-(Upu(3)*Y323);
I23puRMSs=abs(I23pus);
I23s=I23pus*Ibase23(2);
I23RMSs=abs(I23s);
%Linea34
I34pu=(Upu(3)-Upu(4))/Z34pu;
I34puRMS=abs(I34pu);
I34=I34pu*Ibase23(2);
I34RMS=abs(I34);
%Transformador45
%Devanado primario
I45pup=((Upu(4)-Upu(5))*Y245)+(Upu(4)*Y145);
I45puRMSp=abs(I45pup);
I45p=I45pup*Ibase45(1);
I45RMSp=abs(I45p);
%Devanado secundario
```

```
I45pus=((Upu(4)-Upu(5))*Y245)-(Upu(5)*Y345);
I45puRMSs=abs(I45pus);
I45s=I45pus*Ibase45(2);
I45RMSs=abs(I45s);
%Linea56
I56pu=(Upu(5)-Upu(6))/Z56pu;
I56puRMS=abs(I56pu);
I56=I56pu*Ibase45(2);
I56RMS=abs(I56);
%Transformador67
%Devanado primario
I67pup=((Upu(6)-Upu(7))*Y267)+(Upu(6)*Y167);
I67puRMSp=abs(I67pup);
I67p=I67pup*Ibase67(1);
I67RMSp=abs(I67p);
%Devanado secundario
I67pus=((Upu(6)-Upu(7))*Y267)-(Upu(7)*Y367);
I67puRMSs=abs(I67pus);
I67s=I67pus*Ibase67(2);
I67RMSs=abs(I67s);
%Transformador38
%Devanado primario
I38pup=((Upu(3)-Upu(8))*Y238)+(Upu(3)*Y138);
I38puRMSp=abs(I38pup);
I38p=I38pup*Ibase38(1);
I38RMSp=abs(I38p);
%Devanado secundario
I38pus=((Upu(3)-Upu(8))*Y238)-(Upu(8)*Y338);
I38puRMSs=abs(I38pus);
I38s=I38pus*Ibase38(2);
I38RMSs=abs(I38s);
%Linea89
I89pu=(Upu(8)-Upu(9))/Z89pu;
I89puRMS=abs(I89pu);
I89=I89pu*Ibase38(2);
I89RMS=abs(I89);
%Creo vectores con las corrientes finales en las líneas y en los
%transformadores:
Ilineaspu=[I12puRMS,I34puRMS,I56puRMS,I89puRMS];
Ilineas=[I12RMS,I34RMS,I56RMS,I89RMS];
Itrafospup=[I23puRMSp,I45puRMSp,I67puRMSp,I38puRMSp];
Itrafosp=[I23RMSp,I45RMSp,I67RMSp,I38RMSp];
Itrafospus=[I23puRMSs,I45puRMSs,I67puRMSs,I38puRMSs];
Itrafoss=[I23RMSs,I45RMSs,I67RMSs,I38RMSs];
toc
```

El error cometido por el programa es de **$1.931 \cdot 10^{-6}$.**

El tiempo de ejecución del programa es de **0.148 segundos**.

El número de iteraciones realizadas es de : **3**.

ANEXO III: Programación del Método Direct Approach incluyendo un Perfil de Cargas Diario mediante Matlab

A continuación se muestra el desarrollo del método Direct Approach resuelto mediante el programa *MATLAB*. En este programa se han utilizado 1440 medidas de P y Q utilizando un perfil de cargas diario. De esta forma se obtienen los valores de las tensiones en la Red de Distribución, las cuales se corrompen al final del mismo y serán utilizadas para desarrollar el algoritmo de estimación de estado WLS en el cual se introducirá un perfil de cargas diario.

```
%RESOLUCIÓN DEL PROBLEMA DE FLUJO DE CARGAS MEDIANTE EL MÉTODO DIRECT
%APPROACH, PARA OBTENER LAS POTENCIAS Y TENSIONES DE FUNCIONAMIENTO DE
UNA
%RED DE DISTRIBUCIÓN DE 9 NODOS, EN LA CUAL HEMOS INTRODUCIDO UN
PERFIL DE
%CARGAS DIARIO QUE NOS PROPORCIONA 1440 MEDIDAS DE P Y Q.
clear all
tic
%Introducimos la Potencia Compleja que usaremos como base:
Sbase=100e6;
%Introducimos los datos de la Red de Distribución y calculamos las
%impedancias y admitancias en p.u
%Transformador 23
a23=0.98;
Ubase23(1)=220000;
Ubase23(2)=132000;
Ibase23=Sbase./(sqrt(3)*Ubase23);
Zbase23=(Ubase23.*Ubase23)/Sbase;
S23=270e6;
Zbaset1=(Ubase23(1)^2)/S23;
Z23=(0.009+0.129j)/2;
Z23o=Z23*Zbaset1;
Z23pu=Z23o/Zbase23(1);
Y23=1/Z23pu;
%Transformador 45
a45=0.99;
Ubase45(1)=132000;
Ubase45(2)=30000;
Ibase45=Sbase./(sqrt(3)*Ubase45);
Zbase45=(Ubase45.*Ubase45)/Sbase;
S45=37.5e6;
Zbaset2=(Ubase45(1)^2)/S45;
Z45=(0.009+0.09j)/3;
Z45o=Z45*Zbaset2;
Z45pu=Z45o/Zbase45(1);
Y45=1/Z45pu;
%Transformador 67
```

```
a67=1;
Ubase67(1)=30000;
Ubase67(2)=13800;
Ibase67=Sbase./(sqrt(3)*Ubase67);
Zbase67=(Ubase67.*Ubase67)/Sbase;
S67=10e6;
Zbaset3=(Ubase67(1)^2)/S67;
Z67=0.0095+0.048j;
Z67o=Z67*Zbaset3;
Z67pu=Z67o/Zbase67(1);
Y67=1/Z67pu;
%Transformador 38
a38=0.98;
Ubase38(1)=132000;
Ubase38(2)=13800;
Ibase38=Sbase./(sqrt(3)*Ubase38);
Zbase38=(Ubase38.*Ubase38)/Sbase;
S38=50e6;
Zbaset4=(Ubase38(1)^2)/S38;
Z38=(0.0092+0.085j)/3;
Z38o=Z38*Zbaset4;
Z38pu=Z38o/Zbase38(1);
Y38=1/Z38pu;
%Linea 12
Z12=(0.025+0.240j)*4.7;
Z12pu=Z12/Zbase23(1);
Y12=1/Z12pu;
%Linea 34
Z34=(0.161+0.151j)*1.5;
Z34pu=Z34/Zbase23(2);
Y34=1/Z34pu;
%Linea 56
Z56=(0.568+0.133j)*0.3;
Z56pu=Z56/Zbase45(2);
Y56=1/Z56pu;
%Linea 89
Z89=(0.161+0.112j)*1.8;
Z89pu=Z89/Zbase38(2);
Y89=1/Z89pu;
%Cálculo de las admitancias en los transformadores
%Transformador 23
Y123=((1-a23)/a23^2)*Y23;
Y223=(1/a23)*Y23;
Y323=((a23-1)/a23)*Y23;
Z123=1/Y123;
Z223=1/Y223;
Z323=1/Y323;
%Transformador 45
Y145=((1-a45)/a45^2)*Y45;
Y245=(1/a45)*Y45;
Y345=((a45-1)/a45)*Y45;
Z145=1/Y145;
Z245=1/Y245;
Z345=1/Y345;
%Transformador 67
Y167=((1-a67)/a67^2)*Y67;
Y267=(1/a67)*Y67;
```

```
Y367=((a67-1)/a67)*Y67;
Z167=1/Y167;
Z267=1/Y267;
Z367=1/Y367;
%Transformador 38
Y138=((1-a38)/a38^2)*Y38;
Y238=(1/a38)*Y38;
Y338=((a38-1)/a38)*Y38;
Z138=1/Y138;
Z238=1/Y238;
Z338=1/Y338;

%Cálculo matrices para desarrollar el método de Direct Approach. Tiene
%dimensión 8x8, ya que no se ha de introducir el nodo SLACK.
BIBC= [1 1 1 1 1 1 1 1
    0 1 1 1 1 1 1 1
    0 0 1 1 1 1 0 0
    0 0 0 1 1 1 0 0
    0 0 0 0 1 1 0 0
    0 0 0 0 0 1 0 0
    0 0 0 0 0 0 1 1
    0 0 0 0 0 0 0 1];

BCBV= [Z12pu 0 0 0 0 0 0 0
    Z12pu Z223 0 0 0 0 0 0
    Z12pu Z223 Z34pu 0 0 0 0 0
    Z12pu Z223 Z34pu Z245 0 0 0 0
    Z12pu Z223 Z34pu Z245 Z56pu 0 0 0
    Z12pu Z223 Z34pu Z245 Z56pu Z267 0 0
    Z12pu Z223 0 0 0 0 Z238 0
    Z12pu Z223 0 0 0 0 Z238 Z89pu];

%Creamos un bucle con 1440 medidas para obtener los valores de las
%tensiones para cada uno de los valores de P y Q.
for i = 1:1440;
%Introducimos el valor de los consumos que se producen en cada nodo.
%Con este comando cargamos las 1440 medidas de P y Q.
load('Potencias.mat')
P3=P3e;
Q3=Q3e;
P5=P5e;
Q5=Q5e;
P7=P7e;
Q7=Q7e;
P8=P8e;
Q8=Q8e;
P9=P9e;
Q9=Q9e;
%Construimos un vector con los valores en p.u de las potencias activas
y
%reactivas de la red. El signo - se debe a que se trata de nodos de
%consumo.
Ppu=[0,-P3(i)/Sbase,0,-P5(i)/Sbase,0,-P7(i)/Sbase,-P8(i)/Sbase,-
P9(i)/Sbase];
Qpu=[0,-Q3(i)/Sbase,0,-Q5(i)/Sbase,0,-Q7(i)/Sbase,-Q8(i)/Sbase,-
Q9(i)/Sbase];
Spu=Ppu+Qpu*1i;
```

```
%Calculo las tensiones en los nodos
N=8;
tolerancia=10e-10;
iteraciones=0;
error=tolerancia+1;
IncVant=0;
V0=ones(1,N);
Upu=V0;
while error>tolerancia;
    iteraciones=iteraciones+1;
    Ibus1=conj(Spu./Upu);
    %Vector impedancias transformadores con tomas
    YT=[Y123 Y323+Y138 Y145 Y345 Y167 Y367 Y338 0];
    IT=YT.*Upu;
    Ibusfinal=Ibus1-IT;
    B=BIBC*Ibusfinal.';
    IncV=BCBV*B;
    Upu=V0+IncV.';
    %Una vez resuelto el problema de flujo de cargas, ya puedo
calcular el
    %valor de las tensiones y de las corrientes que tendremos en cada
nodo de la Red de
    %distribución.
    U1=Ubase23(1);
    U2=Ubase23(1)*abs(Upu(1));
    U3=Ubase23(2)*abs(Upu(2));
    U4=Ubase45(1)*abs(Upu(3));
    U5=Ubase45(2)*abs(Upu(4));
    U6=Ubase67(1)*abs(Upu(5));
    U7=Ubase67(2)*abs(Upu(6));
    U8=Ubase38(2)*abs(Upu(7));
    U9=Ubase38(2)*abs(Upu(8));
    U=[U1;U2;U3;U4;U5;U6;U7;U8;U9];
    error=max(abs(IncV-IncVant));
    IncVant=IncV;
end
%Creo una matriz donde introduciré las 1440 medidas de tensión
%correspondiente a cada par de valores de P y Q del perfil de cargas
diario.
   Utotal(:,i)=U;
end
toc

%Corrompo los valores de U, los cuales presentarán una variación de un
0.1%
%respecto a los originales. Estos valores de las tensiones serán los
que
%utilice para desarrollar mi algoritmo de estimación de estado
utilizando el perfil de cargas diario.
rng('default')
rng(5)
eU=normrnd(0,0.001,[9 1440]); %Distribución normal de media 0 y
varianza 0.001
U1c=Utotal(1,:)+Utotal(1,:).*eU(1,:);
U2c=Utotal(2,:)+Utotal(2,:).*eU(2,:);
U3c=Utotal(3,:)+Utotal(3,:).*eU(3,:);
U4c=Utotal(4,:)+Utotal(4,:).*eU(4,:);
```

```
U5c=Utotal(5,:)+Utotal(5,:).*eU(5,:);
U6c=Utotal(6,:)+Utotal(6,:).*eU(6,:);
U7c=Utotal(7,:)+Utotal(7,:).*eU(7,:);
U8c=Utotal(8,:)+Utotal(8,:).*eU(8,:);
U9c=Utotal(9,:)+Utotal(9,:).*eU(9,:);
Utotaldc=[U1c;U2c;U3c;U4c;U5c;U6c;U7c;U8c;U9c];
save('Ucfilediario.mat','Utotaldc'); %Aquí guardo los valores de las
Tensiones corrompidas
```

El error cometido por el programa es de **$2.4578 \cdot 10^{-10}$.**

El tiempo de ejecución del programa es de **2.761 segundos**.

El número de iteraciones realizadas es de: **10**.

ANEXO IV: Modificación de las Potencias Activas y Reactivas utilizadas en WLS incluyendo un Perfil de Cargas Diario

```
%EN ESTE ARCHIVO CALCULAREMOS LOS VALORES ESCALADOS Y CORROMPIDOS DE
LAS P Y Q PROPORCIONADAS POR NUESTRO PERFIL DE CARGAS DIARIO.
clear all
load('ADRES_2sets_1min.mat')
P3=84e6;
Q3=26e6;
P5=34e6;
Q5=12e6;
P7=7.5e6;
Q7=5e6;
P8=52e6;
Q8=39e6;
P9=1.7e6;
Q9=1.5e6;
P3d=P_1_o_av*1e6;
P5d=P_2_o_av*1e6;
P7d=P_3_o_av*1e6;
P8d=P_4_o_av*1e6;
P9d=P_5_o_av*1e6;
Q3d=Q_1_o_av*1e6;
Q5d=Q_2_o_av*1e6;
Q7d=Q_3_o_av*1e6;
Q8d=Q_4_o_av*1e6;
Q9d=Q_5_o_av*1e6;

%Ahora calculo las constantes de escalado, las cuales se obtienen
%dividiendo nuestras potencias iniciales entre el valor máximo de cada
%muestra de 1440 datos.
k3p=P3/max(P3d);
k5p=P5/max(P5d);
k7p=P7/max(P7d);
k8p=P8/max(P8d);
k9p=P9/max(P9d);
k3q=Q3/max(Q3d);
k5q=Q5/max(Q5d);
k7q=Q7/max(Q7d);
k8q=Q8/max(Q8d);
k9q=Q9/max(Q9d);

%Las potencias escaladas serán:
P3e=P3d*k3p;
P5e=P5d*k5p;
P7e=P7d*k7p;
P8e=P8d*k8p;
P9e=P9d*k9p;
Q3e=Q3d*k3q;
Q5e=Q5d*k5q;
Q7e=Q7d*k7q;
```

```
Q8e=Q8d*k8q;
Q9e=Q9d*k9q;

%Una vez tengo escalados los valores de las potencias los corrompo
%Corrompo los valores de P, los cuales presentarán una variación de un
2% respecto a los originales.
rng('default')
rng(1)
eP=normrnd(0,0.02,[5 1440]); %Distribución normal de media 0 y
varianza 0.02
P3dec=P3e+P3e.*eP(1,:);
P5dec=P5e+P5e.*eP(2,:);
P7dec=P7e+P7e.*eP(3,:);
P8dec=P8e+P8e.*eP(4,:);
P9dec=P9e+P9e.*eP(5,:);
Pdec=[P3dec;P5dec;P7dec;P8dec;P9dec];
%Corrompo los valores de Q, los cuales presentarán una variación de un
2%
%respecto a los originales.
rng('default')
rng(3)
eQ=normrnd(0,0.02,[5 1440]); %Distribución normal de media 0 y
varianza 0.02
Q3dec=Q3e+Q3e.*eQ(1,:);
Q5dec=Q5e+Q5e.*eQ(2,:);
Q7dec=Q7e+Q7e.*eQ(3,:);
Q8dec=Q8e+Q8e.*eQ(4,:);
Q9dec=Q9e+Q9e.*eQ(5,:);
Qdec=[Q3dec;Q5dec;Q7dec;Q8dec;Q9dec];
%Ahora guardo los valores escalados que utilizaré como punto de
partida del
%perfil de cargas diario en D.A
save('Potencias.mat','P3e','P5e','P7e','P8e','P9e','Q3e','Q5e','Q7e','
Q8e','Q9e');
%Ahora guardo los valores escalados corrompidos que utilizaré en mi
WLS
%diario
save('Pdiaria.mat','Pdec');
save('Qdiaria.mat','Qdec');
```

ANEXO V: Programación del WLS incluyendo un Perfil de Cargas Diario mediante Matlab

A continuación se muestra el desarrollo del algoritmo WLS resuelto mediante el programa *MATLAB*. En este programa se han utilizado las medidas del perfil de cargas diario ya corrompidas en los ANEXOS III y IV. De esta forma se obtienen los valores de las tensiones, las corrientes, los ángulos y las potencias en la Red de Distribución.

```
%DESARROLLO DEL ALGORITMO DE ESTIMACIÓN DE ESTADO PARA UNA RED DE
%DISTRIBUCIÓN ELÉCTRICA DE 9 NODOS, INTRODUCIENDO UN PERFIL DE CARGAS
%DIARIO QUE CONSTA DE 1440 MEDIDAS DE POTENCIA ACTIVA Y REACTIVA
clear all
tic

%Introducimos la Potencia Compleja que usaremos como base:
Sbase=100e6;

%Introducimos los datos de la Red de Distribución y calculamos las
%impedancias y admitancias en p.u
%Transformador 23
a23=0.98;
Ubase23(1)=220000;
Ubase23(2)=132000;
Ibase23=Sbase./(sqrt(3)*Ubase23);
Zbase23=(Ubase23.*Ubase23)/Sbase;
S23=270e6;
Zbaset1=(Ubase23(1)^2)/S23;
Z23=(0.009+0.129j)/2;
Z23o=Z23*Zbaset1;
Z23pu=Z23o/Zbase23(1);
Y23=1/Z23pu;
%Transformador 45
a45=0.99;
Ubase45(1)=132000;
Ubase45(2)=30000;
Ibase45=Sbase./(sqrt(3)*Ubase45);
Zbase45=(Ubase45.*Ubase45)/Sbase;
S45=37.5e6;
Zbaset2=(Ubase45(1)^2)/S45;
Z45=(0.009+0.09j)/3;
Z45o=Z45*Zbaset2;
Z45pu=Z45o/Zbase45(1);
Y45=1/Z45pu;
%Transformador 67
a67=1;
Ubase67(1)=30000;
Ubase67(2)=13800;
Ibase67=Sbase./(sqrt(3)*Ubase67);
Zbase67=(Ubase67.*Ubase67)/Sbase;
S67=10e6;
Zbaset3=(Ubase67(1)^2)/S67;
```

```
Z67=0.0095+0.048j;
Z67o=Z67*Zbaset3;
Z67pu=Z67o/Zbase67(1);
Y67=1/Z67pu;
%Transformador 38
a38=0.98;
Ubase38(1)=132000;
Ubase38(2)=13800;
Ibase38=Sbase./(sqrt(3)*Ubase38);
Zbase38=(Ubase38.*Ubase38)/Sbase;
S38=50e6;
Zbaset4=(Ubase38(1)^2)/S38;
Z38=(0.0092+0.085j)/3;
Z38o=Z38*Zbaset4;
Z38pu=Z38o/Zbase38(1);
Y38=1/Z38pu;
%Linea 12
Z12=(0.025+0.240j)*4.7;
Z12pu=Z12/Zbase23(1);
Y12=1/Z12pu;
%Linea 34
Z34=(0.161+0.151j)*1.5;
Z34pu=Z34/Zbase23(2);
Y34=1/Z34pu;
%Linea 56
Z56=(0.568+0.133j)*0.3;
Z56pu=Z56/Zbase45(2);
Y56=1/Z56pu;
%Linea 89
Z89=(0.161+0.112j)*1.8;
Z89pu=Z89/Zbase38(2);
Y89=1/Z89pu;

%Cálculo de las admitancias en los transformadores
%Transformador 23
Y123=((1-a23)/a23^2)*Y23;
Y223=(1/a23)*Y23;
Y323=((a23-1)/a23)*Y23;
Z123=1/Y123;
Z223=1/Y223;
Z323=1/Y323;
%Transformador 45
Y145=((1-a45)/a45^2)*Y45;
Y245=(1/a45)*Y45;
Y345=((a45-1)/a45)*Y45;
Z145=1/Y145;
Z245=1/Y245;
Z345=1/Y345;
%Transformador 67
Y167=((1-a67)/a67^2)*Y67;
Y267=(1/a67)*Y67;
Y367=((a67-1)/a67)*Y67;
Z167=1/Y167;
Z267=1/Y267;
Z367=1/Y367;
%Transformador 38
Y138=((1-a38)/a38^2)*Y38;
```

```
Y238=(1/a38)*Y38;
Y338=((a38-1)/a38)*Y38;
Z138=1/Y138;
Z238=1/Y238;
Z338=1/Y338;

%Construimos la matriz de admitancias:
Y= [Y12 -Y12 0 0 0 0 0 0 0
    -Y12 Y12+Y123+Y223 -Y223 0 0 0 0 0 0
    0 -Y223 Y223+Y323+Y34+Y138+Y238 -Y34 0 0 0 -Y238 0
    0 0 -Y34 Y34+Y145+Y245 -Y245 0 0 0 0
    0 0 0 -Y245 Y245+Y345+Y56 -Y56 0 0 0
    0 0 0 0 -Y56 Y56+Y167+Y267 -Y267 0 0
    0 0 0 0 0 -Y267 Y267+Y367 0 0
    0 0 -Y238 0 0 0 0 Y238+Y338+Y89 -Y89
    0 0 0 0 0 0 0 -Y89 Y89];


%Buses P Q y U. Tomo como datos de partida los valores corrompidos
obtenidos
%en mi archivo Perfilcargadiario.mat, y en la resolución del método
Direct
%Approch con las 1440 muestras.
load('Pdiaria.mat')
load('Qdiaria.mat')
load('Ucfilediario.mat')
%Potencias activas
P3pu=-Pdec(1,:)/Sbase;
P5pu=-Pdec(2,:)/Sbase;
P7pu=-Pdec(3,:)/Sbase;
P8pu=-Pdec(4,:)/Sbase;
P9pu=-Pdec(5,:)/Sbase;
%Vector de potencias activas p.u
Q3pu=-Qdec(1,:)/Sbase;
Q5pu=-Qdec(2,:)/Sbase;
Q7pu=-Qdec(3,:)/Sbase;
Q8pu=-Qdec(4,:)/Sbase;
Q9pu=-Qdec(5,:)/Sbase;
%Tensiones
U1puc=Utotaldc(1,:)/Ubase23(1);
U2puc=Utotaldc(2,:)/Ubase23(1);
U3puc=Utotaldc(3,:)/Ubase23(2);
U4puc=Utotaldc(4,:)/Ubase45(1);
U5puc=Utotaldc(5,:)/Ubase45(2);
U6puc=Utotaldc(6,:)/Ubase67(1);
U7puc=Utotaldc(7,:)/Ubase67(2);
U8puc=Utotaldc(8,:)/Ubase38(2);
U9puc=Utotaldc(9,:)/Ubase38(2);


%Creamos un bucle con 1440 medidas para obtener los valores de las
%tensiones para cada uno de los valores de P y Q.
for i = 1:1440;
%COMIENZO WLS
%Vector de Potencias Activas en p.u de la Red
Ppu=[0;P3pu(i);0;P5pu(i);0;P7pu(i);P8pu(i);P9pu(i)];
```

```
%Vector de Potencias Reactivas en p.u de la Red
Qpu=[0;Q3pu(i);0;Q5pu(i);0;Q7pu(i);Q8pu(i);Q9pu(i)];
%Vector de tensiones de partida en p.u de la Red.
Upuc=[U1puc(i);U2puc(i);U3puc(i);U4puc(i);U5puc(i);U6puc(i);U7puc(i);U
8puc(i);U9puc(i)];
N=9;%Número de buses y de U
Gpu=real(Y); %Parte real de la matriz de admitancias.
Bpu=imag(Y); %Parte imaginaria de la matriz de admitancias.
R=zeros(25); %Matriz diagonal, cuyos valores son las varianzas de las
P, Q y U. Tengo 25 medidas. Tengo 8P, 8Q y 9U
varpq=0.02; %Covarianza para las medidas de P y Q
varu=0.001; %Covarianza para las medidas de U
varo=0.001/10; %Covarianza para los buses en que no tenemos P o Q (10
veces más pequeña que la de U)
%Introduzco las covarianzas para P
    for i=1;

        R(i,i)=varo;
    end
    for i=2
        R(i,i)=varpq;
    end
    for i=3
        R(i,i)=varo;
    end
    for i=4
        R(i,i)=varpq;
    end
    for i=5
        R(i,i)=varo;
    end
    for i=6:8
        R(i,i)=varpq;
    end
    %Introduzco las covarianzas para Q
    for i=9;

        R(i,i)=varo;
    end
    for i=10
        R(i,i)=varpq;
    end
    for i=11
        R(i,i)=varo;
    end
    for i=12
        R(i,i)=varpq;
    end
    for i=13
        R(i,i)=varo;
    end
    for i=14:16
        R(i,i)=varpq;
    end
    %Introduzco las covarianzas
    for i=17:25
```

```
        R(i,i)=varu;
    end
    W=inv(R); %Matriz inversa de la matriz de covarianzas.
    z=[Ppu;Qpu;Upuc]; %Vector de partida formado por los valores de P,
Q y U.
    Upu=ones(N,1); %Iniciamos en arranque plano V=1. Valores en p.u
    teta=zeros(N,1); %Iniciamos en arranque plano teta=0
    X=[teta(2:end);Upu]; %Vector estado (2N-1; 8 tetas (0 SLACK) y 9
U)
    tolerancia=10e-6;
    error=tolerancia+1;
    iteraciones=0;
    %Defino un vector con las posiciones de mis P y Q
    busPQ=[2;3;4;5;6;7;8;9];
    numberPQ=length(busPQ);

    while error>tolerancia;
        iteraciones=iteraciones+1;
        %tengo que calcular las h que depende de U, P y Q
        hu=Upu;
        hp=zeros(numberPQ,1); %tengo 8 medidas de P
        hq=zeros(numberPQ,1); %tengo 8 medidas de Q

        for i = 1:numberPQ %tengo 8 hp, ya que tengo 8P
(P2,P3,P4,P5,P6,P7,P8,P9)
            j=busPQ(i);
            for k = 1:N
                hp(i) = hp(i) + Upu(j)*Upu(k)*(Gpu(j,k)*cos(teta(j)-
teta(k)) + Bpu(j,k)*sin(teta(j)-teta(k)));
            end
        end

        for i = 1:numberPQ %tengo 8 hq, ya que tengo 8Q
            j=busPQ(i);
            for k = 1:N
                hq(i) = hq(i) + Upu(j)*Upu(k)*(Gpu(j,k)*sin(teta(j)-
teta(k)) - Bpu(j,k)*cos(teta(j)-teta(k)));
            end
        end
        h=[hp;hq;hu];

        r=z-h;%residuo

        %Tengo que calcular ahora el Jacobiano H
        HUteta = zeros(N,N-1); %derivada de U respecto a teta (9x8, ya
que teta1)

        HUU = zeros(N,N); %derivada de U respecto a U (9x9)
        for i=1:N
            for k = 1:N
                if i == k
                    HUU(i,k) = 1;
                end
            end
        end
```

```
        HPteta = zeros(numberPQ,N-1); %Derivada de P respecto a teta
(8x8) %Tengo 8 medidas de P
        for i = 1:numberPQ
            m=busPQ(i); %Estas son las posiciones de mis tensiones
            for k = 1:N-1 %Estos son mis ángulos
                if k+1 == m
                    for n=1:N
                        HPteta(i,k) = HPteta(i,k) + Upu(m)* Upu(n)*(-
Gpu(m,n)*sin(teta(m)-teta(n)) + Bpu(m,n)*cos(teta(m)-teta(n)));
                    end
                    HPteta(i,k)= HPteta(i,k)- Upu(m)^2*Bpu(m,m);
                else
                    HPteta(i,k) = Upu(m)*
Upu(k+1)*(Gpu(m,k+1)*sin(teta(m)-teta(k+1)) - Bpu(m,k+1)*cos(teta(m)-
teta(k+1)));
                end
            end
        end


        HPU = zeros(numberPQ,N); %Derivada de P respecto a U (8x9)
        for i = 1:numberPQ
            m=busPQ(i);
            for k = 1:N
                if k == m
                    for n=1:N
                        HPU(i,k) = HPU(i,k) +
Upu(n)*(Gpu(m,n)*cos(teta(m)-teta(n)) + Bpu(m,n)*sin(teta(m)-
teta(n)));
                    end
                    HPU(i,k)=HPU(i,k)+ Upu(m)*Gpu(m,m);
                else
                    HPU(i,k) = Upu(m)*(Gpu(m,k)*cos(teta(m)-teta(k)) +
Bpu(m,k)*sin(teta(m)-teta(k)));
                end
            end
        end

        HQteta = zeros(numberPQ,N-1); %Derivada de Q respecto a teta
(8x8)
        for i = 1:numberPQ
            m=busPQ(i);
            for k = 1:N-1
                if k+1 == m
                    for n=1:N
                        HQteta(i,k) = HQteta(i,k) + Upu(m)*
Upu(n)*(Gpu(m,n)*cos(teta(m)-teta(n)) + Bpu(m,n)*sin(teta(m)-
teta(n)));
                    end
                    HQteta(i,k) = HQteta(i,k)- Upu(m)^2*Gpu(m,m);
                else
                    HQteta(i,k) = Upu(m)* Upu(k+1)*(-
Gpu(m,k+1)*cos(teta(m)-teta(k+1)) - Bpu(m,k+1)*sin(teta(m)-
teta(k+1)));
                end
            end
        end
```

```
        HQU = zeros(numberPQ,N); %Derivada de Q respecto a U (8x9)
        for i = 1:numberPQ
            m = busPQ(i);
            for k = 1:N
                if k == m
                    for n = 1:N
                        HQU(i,k) = HQU(i,k) +
Upu(n)*(Gpu(m,n)*sin(teta(m)-teta(n)) - Bpu(m,n)*cos(teta(m)-
teta(n)));
                    end
                    HQU(i,k)= HQU(i,k) - Upu(m)*Bpu(m,m);
                else
                    HQU(i,k) = Upu(m)*(Gpu(m,k)*sin(teta(m)-teta(k)) -
Bpu(m,k)*cos(teta(m)-teta(k)));
                end
            end
        end

        %Construyo el Jacobiano
        H=[HPteta HPU; HQteta HQU; HUteta HUU];
        %Calculo G
        G = transpose(H)*W*H;
        %Calculo el incremento de X (vector de estado)
        incX=G\(transpose(H)*W*r);
        X=X+incX;
        %Obtengo las medidas de los ángulos.
        teta(2:end)=X(1:8);
        %Obtengo las medidas de las tensiones.
        Upu=X(9:end);
        error=max(abs(incX));
    end

end

%Una vez resuelto el problema de flujo de cargas, ya puedo calcular el
%valor de las tensiones y de las corrientes que tendremos en cada nodo
%y cada líneas de la Red de distribución.
%Tensiones
U1=Ubase23(1)*abs(Upu(1));
U2=Ubase23(1)*abs(Upu(2));
U3=Ubase23(2)*abs(Upu(3));
U4=Ubase45(1)*abs(Upu(4));
U5=Ubase45(2)*abs(Upu(5));
U6=Ubase67(1)*abs(Upu(6));
U7=Ubase67(2)*abs(Upu(7));
U8=Ubase38(2)*abs(Upu(8));
U9=Ubase38(2)*abs(Upu(9));
Ufinal=[U1;U2;U3;U4;U5;U6;U7;U8;U9];
%Corrientes
%Linea12
I12pu=(Upu(1)-Upu(2))/Z12pu;
I12puRMS=abs(I12pu);
I12=I12pu*Ibase23(1);
I12RMS=abs(I12);
%Transformador23
%Devanado primario
```

```
I23pup=((Upu(2)-Upu(3))*Y223)+(Upu(2)*Y123);
I23puRMSp=abs(I23pup);
I23p=I23pup*Ibase23(1);
I23RMSp=abs(I23p);
%Devanado secundario
I23pus=((Upu(2)-Upu(3))*Y223)-(Upu(3)*Y323);
I23puRMSs=abs(I23pus);
I23s=I23pus*Ibase23(2);
I23RMSs=abs(I23s);
%Linea34
I34pu=(Upu(3)-Upu(4))/Z34pu;
I34puRMS=abs(I34pu);
I34=I34pu*Ibase23(2);
I34RMS=abs(I34);
%Transformador45
%Devanado primario
I45pup=((Upu(4)-Upu(5))*Y245)+(Upu(4)*Y145);
I45puRMSp=abs(I45pup);
I45p=I45pup*Ibase45(1);
I45RMSp=abs(I45p);
%Devanado secundario
I45pus=((Upu(4)-Upu(5))*Y245)-(Upu(5)*Y345);
I45puRMSs=abs(I45pus);
I45s=I45pus*Ibase45(2);
I45RMSs=abs(I45s);
%Linea56
I56pu=(Upu(5)-Upu(6))/Z56pu;
I56puRMS=abs(I56pu);
I56=I56pu*Ibase45(2);
I56RMS=abs(I56);
%Transformador67
%Devanado primario
I67pup=((Upu(6)-Upu(7))*Y267)+(Upu(6)*Y167);
I67puRMSp=abs(I67pup);
I67p=I67pup*Ibase67(1);
I67RMSp=abs(I67p);
%Devanado secundario
I67pus=((Upu(6)-Upu(7))*Y267)-(Upu(7)*Y367);
I67puRMSs=abs(I67pus);
I67s=I67pus*Ibase67(2);
I67RMSs=abs(I67s);
%Transformador38
%Devanado primario
I38pup=((Upu(3)-Upu(8))*Y238)+(Upu(3)*Y138);
I38puRMSp=abs(I38pup);
I38p=I38pup*Ibase38(1);
I38RMSp=abs(I38p);
%Devanado secundario
I38pus=((Upu(3)-Upu(8))*Y238)-(Upu(8)*Y338);
I38puRMSs=abs(I38pus);
I38s=I38pus*Ibase38(2);
I38RMSs=abs(I38s);
%Linea89
I89pu=(Upu(8)-Upu(9))/Z89pu;
I89puRMS=abs(I89pu);
I89=I89pu*Ibase38(2);
I89RMS=abs(I89);
```

```
%Creo vectores con las corrientes finales en las líneas y en los
%transformadores:
Ilineaspu=[I12puRMS,I34puRMS,I56puRMS,I89puRMS];
Ilineas=[I12RMS,I34RMS,I56RMS,I89RMS];
Itrafospup=[I23puRMSp,I45puRMSp,I67puRMSp,I38puRMSp];
Itrafosp=[I23RMSp,I45RMSp,I67RMSp,I38RMSp];
Itrafospus=[I23puRMSs,I45puRMSs,I67puRMSs,I38puRMSs];
Itrafoss=[I23RMSs,I45RMSs,I67RMSs,I38RMSs];
toc
```

El error cometido por el programa es de **$5.5889 \cdot 10^{-7}$.**

El tiempo de ejecución del programa es de **4.429 segundos**.

El número de iteraciones realizadas es de: **3**.

ANEXO VI: Programación del WLS incluyendo un Perfil de Cargas Diario mediante Python

A continuación se muestra el desarrollo del algoritmo WLS resuelto mediante el programa *PYTHON*. En este programa se han utilizado las medidas del perfil de cargas diario ya corrompidas en los Anexos III y IV. De esta forma se obtienen los valores de las tensiones, las corrientes, los ángulos y las potencias en la Red de Distribución.

```
"""
Created on Fri Jun 8 12:05:15 2018

@author: Fran Pello
"""
#DESARROLLO DEL ALGORITMO DE ESTIMACIÓN DE ESTADO PARA UNA RED DE
#DISTRIBUCIÓN ELÉCTRICA DE 9 NODOS, INTRODUCIENDO UN PERFIL DE CARGAS
#DIARIO QUE CONSTA DE 1440 MEDIDAS DE POTENCIA ACTIVA Y REACTIVA

#Importamos los diferentes módulos de Python que se necesitarán para
desarrollar nuestro programa
import time
start = time.time()
import math
import numpy as np
#Introducimos la Potencia Compleja que usaremos como base:
Sbase = 100e6
#Introducimos los datos de la Red de Distribución y calculamos las
#impedancias y admitancias en p.u
#Transformador 23
a23 = 0.98
Ubase23 = np.array([220000,132000],dtype = np.int64)
Ibase23 = Sbase/(math.sqrt(3)*Ubase23)
Zbase23 = (Ubase23*Ubase23)/Sbase
S23 = 270e6
Zbaset1 = (Ubase23[0]**2)/S23
Z23 = (0.009+0.129j)/2
Z23o = Z23*Zbaset1
Z23pu = Z23o/Zbase23[0]
Y23 = 1/Z23pu
#Transformador 45
a45 = 0.99
Ubase45 = np.array([132000,30000],dtype = np.int64)
Ibase45 = Sbase/(math.sqrt(3)*Ubase45)
Zbase45 = (Ubase45*Ubase45)/Sbase
S45 = 37.5e6
Zbaset2 = (Ubase45[0]**2)/S45
Z45 = (0.009+0.09j)/3
Z45o = Z45*Zbaset2
Z45pu = Z45o/Zbase45[0]
Y45 = 1/Z45pu
#Transformador 67
a67 = 1
```

```
Ubase67 = np.array([30000,13800],dtype = np.int64)
Ibase67 = Sbase/(math.sqrt(3)*Ubase67)
Zbase67 = (Ubase67*Ubase67)/Sbase
S67 = 10e6
Zbaset3 = (Ubase67[0]**2)/S67
Z67 = 0.0095+0.048j;
Z67o = Z67*Zbaset3
Z67pu = Z67o/Zbase67[0]
Y67 = 1/Z67pu
#Transformador 38
a38 = 0.98
Ubase38 = np.array([132000,13800],dtype = np.int64)
Ibase38 = Sbase/(math.sqrt(3)*Ubase38)
Zbase38 = (Ubase38*Ubase38)/Sbase
S38 = 50e6
Zbaset4 = (Ubase38[0]**2)/S38
Z38 = (0.0092+0.085j)/3
Z38o = Z38 * Zbaset4
Z38pu = Z38o / Zbase38 [0]
Y38 = 1 / Z38pu
#Linea 12
Z12 = (0.025 + 0.240j) * 4.7
Z12pu = Z12 / Zbase23 [0]
Y12 = 1 / Z12pu
#Linea 34
Z34 = (0.161 + 0.151j) * 1.5
Z34pu = Z34 / Zbase23 [1]
Y34 = 1 / Z34pu
#Linea 56
Z56 = (0.568 + 0.133j) * 0.3
Z56pu = Z56 / Zbase45 [1]
Y56 = 1 / Z56pu
#Linea 89
Z89 = (0.161 + 0.112j) * 1.8
Z89pu = Z89 / Zbase38 [1]
Y89 = 1 / Z89pu

#Cálculo de las admitancias en los transformadores
#Transformador 23
Y123 = ((1 - a23) / a23 ** 2) * Y23
Y223 = (1 / a23) * Y23
Y323 = ((a23 - 1) / a23) * Y23
#Transformador 45
Y145 = ((1 - a45) / a45 ** 2) * Y45
Y245 = (1 / a45) * Y45
Y345 = ((a45 - 1) / a45) * Y45
#Transformador 67
Y167 = ((1 - a67) / a67 ** 2) * Y67
Y267 = (1 / a67) * Y67
Y367 = ((a67 - 1) / a67) * Y67
#Transformador 38
Y138 = ((1 - a38) / a38 ** 2) * Y38
Y238 = (1 / a38) * Y38
Y338 = ((a38 - 1) / a38) * Y38

#Construimos la matriz de admitancias:
Y = np.matrix([[Y12, - Y12, 0, 0, 0, 0, 0, 0, 0],
```

```
                [-Y12, Y12 + Y123 + Y223, - Y223, 0, 0, 0, 0, 0, 0],
                [0, - Y223, Y223 + Y323 + Y34 + Y138 + Y238, - Y34, 0,
0, 0, - Y238, 0],
                [0, 0, - Y34, Y34 + Y145 + Y245, - Y245, 0, 0, 0, 0],
                [0, 0, 0, - Y245, Y245 + Y345 + Y56, - Y56, 0, 0, 0],
                [0, 0, 0, 0, - Y56, Y56 + Y167 + Y267, - Y267, 0, 0],
                [0, 0, 0, 0, 0, - Y267, Y267 + Y367, 0, 0],
                [0, 0, - Y238, 0, 0, 0, 0, Y238 + Y338 + Y89, - Y89],
                [0, 0, 0, 0, 0, 0, 0, - Y89, Y89]],dtype =
np.dtype('c16'))

#Nodos P Q y U. Tomo como datos de partida los valores corrompidos
obtenidos
#en mi archivo Perfilcargadiario.mat, y en la resolución del método
Direct
#Approch con las 1440 muestras. Para ello cargo los datos de un
fichero txt como se muestra a continuación.
P = np.loadtxt("Pdiaria.txt", dtype = 'i', delimiter = ',')
P3pu = -P[0]/Sbase
P5pu = -P[1]/Sbase
P7pu = -P[2]/Sbase
P8pu = -P[3]/Sbase
P9pu = -P[4]/Sbase
#Potencias reactivas corrompidas
Q = np.loadtxt("Qdiaria.txt", dtype = 'i', delimiter = ',')
Q3pu = -Q[0]/Sbase
Q5pu = -Q[1]/Sbase
Q7pu = -Q[2]/Sbase
Q8pu = -Q[3]/Sbase
Q9pu = -Q[4]/Sbase
#Tensiones corrompidas
U = np.loadtxt("tensionesdiarias.txt", dtype = 'i', delimiter = ',')
U1puc = U[0]/Ubase23[0]
U2puc = U[1]/Ubase23[0]
U3puc = U[2]/Ubase23[1]
U4puc = U[3]/Ubase45[0]
U5puc = U[4]/Ubase45[1]
U6puc = U[5]/Ubase67[0]
U7puc = U[6]/Ubase67[1]
U8puc = U[7]/Ubase38[1]
U9puc = U[8]/Ubase38[1]


#COMIENZO WLS
#Creamos un bucle con 1440 medidas para obtener los valores de las
#tensiones para cada uno de los valores de P y Q.
for i in range (1440):
    #Vector de Potencias Activas en p.u de la Red
    Ppu = np.array([0, P3pu[i], 0, P5pu[i], 0, P7pu[i], P8pu[i],
P9pu[i]])
    #Vector de Potencias Reactivas en p.u de la Red
    Qpu = np.array([0, Q3pu[i], 0, Q5pu[i], 0, Q7pu[i], Q8pu[i],
Q9pu[i]])
    #Vector de tensiones de partida en p.u de la Red.
    Upuc = np.array([U1puc[i], U2puc[i], U3puc[i], U4puc[i], U5puc[i],
U6puc[i], U7puc[i], U8puc[i], U9puc[i]])
    N = 9 #número de buses y de U
```

```
    Gpu = Y.real #Parte real de la matriz de admitancias
    Bpu = Y.imag #Parte imaginaria de la matriz de admitancias
    R = np.zeros((25, 25))#Matriz diagonal, cuyos valores son las
varianzas de las P, Q y U. Tengo 25 medidas. Tengo 8P, 8Q y 9U
    varpq = 0.02 #Covarianza para las medidas de P y Q
    varu = 0.001 #Covarianza para las medidas de U
    varo = 0.001 / 10 #Covarianza para los nodos en que no tenemos P o
Q (10 veces más pequeña que la de U)
    #Introduzco las covarianzas para P
    R [0][0] = varo
    R [1][1] = varpq
    R [2][2] = varo
    R [3][3] = varpq
    R [4][4] = varo
    R [5][5] = varpq
    R [6][6] = varpq
    R [7][7] = varpq
    #Introduzco las covarianzas para Q
    R [8][8] = varo
    R [9][9] = varpq
    R [10][10] = varo
    R [11][11] = varpq
    R [12][12] = varo
    R [13][13] = varpq
    R [14][14] = varpq
    R [15][15] = varpq
    R [16][16] = varpq
    #Introduzco las covarianzas para U
    R [17][17] = varu
    R [18][18] = varu
    R [19][19] = varu
    R [20][20] = varu
    R [21][21] = varu
    R [22][22] = varu
    R [23][23] = varu
    R [24][24] = varu

    W = np.linalg.inv(R) #Matriz inversa de la matriz de covarianzas
    z = np.concatenate([Ppu, Qpu, Upuc]) #Vector de partida formado
por los valores de P, Q y U
    Upu = np.ones(N) #Iniciamos en arranque plano V=1. Valores en p.u
    teta = np.zeros(N) #Iniciamos en arranque plano teta=0
    X = np.concatenate([teta[1:], Upu]) #Vector estado (2N-1; 8 tetas
(0 SLACK) y 9 U)
    tolerancia = 10e-6
    error = tolerancia + 1
    iteraciones = 0
    #Defino un vector con las posiciones de mis P y Q
    busPQ = ([1, 2, 3, 4, 5, 6, 7, 8])
    numberPQ = len(busPQ)

    while error > tolerancia:
        iteraciones = iteraciones + 1
    #tengo que calcular las h que depende de U, P y Q
        hu = Upu #tengo 9 medidas de U
        hp = np.zeros(numberPQ)#tengo 8 medidas de P
        hq = np.zeros(numberPQ)#tengo 8 medidas de Q
```

```
        for i in range(numberPQ):#tengo 8 hp, ya que tengo 8P
(P2,P3,P4,P5,P6,P7,P8,P9)
            j = busPQ[i]
            for k in range(N):
                hp[i] = hp[i] + Upu[j] * Upu[k] * (Gpu[j, k] *
math.cos(teta[j] - teta[k]) + Bpu[j, k] * math.sin(teta[j] - teta[k]))

        for i in range(numberPQ): #tengo 8 hq, ya que tengo 8Q
            j = busPQ[i]
            for k in range(N):
                hq[i] = hq[i] + Upu[j] * Upu[k] * (Gpu[j, k] *
math.sin(teta[j] - teta[k]) - Bpu[j, k] * math.cos(teta[j] - teta[k]))

        h = np.concatenate([hp, hq, hu])

        r = np.array(z - h) #residuo

    #Tengo que calcular ahora el Jacobiano H
        HUteta = np.zeros((N, N - 1))#derivada de U respecto a teta
(9x8)

        HUU = np.zeros((N, N))#derivada de U respecto a U (9x9) Tengo
9 medidas de U
        for i in range(N):
            for k in range(N):
                if i == k:
                    HUU[i, k]= 1

        HPteta = np.zeros((numberPQ, N - 1)) #Derivada de P respecto a
teta (8x8)
        for i in range(numberPQ):
            m = busPQ[i] #Estas son las posiciones de mis tensiones
            for k in range(N - 1):
                if k + 1 == m:
                    for n in range(N):
                        HPteta[i, k] = HPteta[i, k] + Upu[m] * Upu[n]
* (-Gpu[m, n] * math.sin(teta[m] - teta[n]) + Bpu[m, n] *
math.cos(teta[m] - teta[n]))

                    HPteta[i, k] = HPteta[i, k] - Upu[m] ** 2*Bpu[m,
m]
                else:
                    HPteta[i, k] = Upu[m] * Upu[k + 1] * (Gpu[m, k +
1] * math.sin(teta[m] - teta[k + 1]) - Bpu[m, k + 1] *
math.cos(teta[m] - teta[k + 1]))

        HPU = np.zeros((numberPQ, N))#Derivada de P respecto a U (8x9)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N):
                if k == m:
                    for n in range(N):
                        HPU[i, k] = HPU[i, k] + Upu[n] * (Gpu[m, n] *
math.cos(teta[m] - teta[n]) + Bpu[m, n] * math.sin(teta[m] - teta[n]))

                    HPU[i, k] = HPU[i, k] + Upu[m] * Gpu[m, m]
```

```
                else:
                    HPU[i, k] = Upu[m] * (Gpu[m, k] * math.cos(teta[m]
- teta[k]) + Bpu[m, k] * math.sin(teta[m] - teta[k]))

        HQteta = np.zeros((numberPQ, N - 1))#Derivada de Q respecto a
teta (8x8)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N - 1):
                if k + 1 == m:
                    for n in range(N):
                        HQteta[i, k] = HQteta[i, k] + Upu[m] * Upu[n]
* (Gpu[m, n] * math.cos(teta[m] - teta[n]) + Bpu[m, n] *
math.sin(teta[m] - teta[n]))

                    HQteta[i, k] = HQteta[i, k] - Upu[m] ** 2 * Gpu[m,
m]
                else:
                    HQteta[i, k] = Upu[m] * Upu[k + 1] * (-Gpu[m, k +
1] * math.cos(teta[m] - teta[k + 1]) - Bpu[m, k + 1] *
math.sin(teta[m] - teta[k + 1]))

        HQU = np.zeros((numberPQ, N))#Derivada de Q respecto a U (8x9)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N):
                if k == m:
                    for n in range(N):
                        HQU[i, k] = HQU[i, k] + Upu[n] * (Gpu[m, n] *
math.sin(teta[m] - teta[n]) - Bpu[m, n] * math.cos(teta[m] - teta[n]))

                    HQU[i, k] = HQU[i, k] - Upu[m] * Bpu[m, m]
                else:
                    HQU[i, k] = Upu[m] * (Gpu[m, k] * math.sin(teta[m]
- teta[k]) - Bpu[m, k] * math.cos(teta[m] - teta[k]))

    #Construyo el Jacobiano
        H = np.block([[HPteta, HPU],[HQteta, HQU], [HUteta, HUU]])
    #Calculo G
        G = np.dot((np.dot(np.transpose(H), W)), H)
    #Calculo el incremento de X (vector de estado)
        G1 = (np.dot((np.dot(np.transpose(H),  W)), r))
        incX = np.dot(np.linalg.inv(G), G1)
        X = X + incX
        #Obtengo las medidas de los ángulos
        teta[1:] = X[0:8]
        #Obtengo las medidas de las tensiones
        Upu[0:] = X[8:]
        error = max(abs(incX))

#Una vez resuelto el problema de flujo de cargas, ya puedo calcular el
#valor de las tensiones y de las corrientes que tendremos en cada nodo
#y cada línea de la Red de Distribución.
#Tensiones
U1 = Ubase23[0] * abs(Upu[0])
U2 = Ubase23[0] * abs(Upu[1])
U3 = Ubase23[1] * abs(Upu[2])
```

```
U4 = Ubase45[0] * abs(Upu[3])
U5 = Ubase45[1] * abs(Upu[4])
U6 = Ubase67[0] * abs(Upu[5])
U7 = Ubase67[1] * abs(Upu[6])
U8 = Ubase38[1] * abs(Upu[7])
U9 = Ubase38[1] * abs(Upu[8])
Ufinales = np.array([U1, U2, U3, U4, U5, U6, U7, U8, U9])
print ("El valor de las tensiones en [kV] en los nodos
será:",np.reshape(Ufinales,(9,1)))

#Corrientes
#Linea12
I12pu=(Upu[0]-Upu[1]) / Z12pu;
I12puRMS=abs (I12pu);
I12=I12pu*Ibase23[0];
I12RMS=abs(I12);
#Transformador23
#Devanado primario
I23pup=((Upu[1]-Upu[2])*Y223) + (Upu[1]*Y123);
I23puRMSp=abs(I23pup);
I23p=I23pup * Ibase23[0];
I23RMSp=abs(I23p);
#Devanado secundario
I23pus=((Upu[1]-Upu[2])*Y223) - (Upu[2]*Y323);
I23puRMSs=abs(I23pus);
I23s=I23pus*Ibase23[1];
I23RMSs=abs(I23s);
#Linea34
I34pu=(Upu[2]-Upu[3]) / Z34pu;
I34puRMS=abs(I34pu);
I34=I34pu * Ibase23[1];
I34RMS=abs(I34);
#Transformador45
#Devanado primario
I45pup=((Upu[3]-Upu[4])*Y245) + (Upu[3]*Y145);
I45puRMSp=abs(I45pup);
I45p=I45pup*Ibase45[0];
I45RMSp=abs(I45p);
#Devanado secundario
I45pus=((Upu[3]-Upu[4])*Y245) - (Upu[4]*Y345);
I45puRMSs=abs(I45pus);
I45s=I45pus*Ibase45[1];
I45RMSs=abs(I45s);
#Linea56
I56pu=(Upu[4]-Upu[5]) / Z56pu;
I56puRMS=abs(I56pu);
I56=I56pu*Ibase45[1];
I56RMS=abs(I56);
#Transformador67
#Devanado primario
I67pup=((Upu[5]-Upu[6])*Y267) + (Upu[5]*Y167);
I67puRMSp=abs(I67pup);
I67p=I67pup*Ibase67[0];
I67RMSp=abs(I67p);
#Devanado secundario
I67pus=((Upu[5]-Upu[6])*Y267) - (Upu[6]*Y367);
I67puRMSs=abs(I67pus);
```

```
I67s=I67pus*Ibase67[1];
I67RMSs=abs(I67s);
#Transformador38
#Devanado primario
I38pup=((Upu[2]-Upu[7])*Y238) + (Upu[2]*Y138);
I38puRMSp=abs(I38pup);
I38p=I38pup*Ibase38[0];
I38RMSp=abs(I38p);
#Devanado secundario
I38pus=((Upu[2]-Upu[7])*Y238) - (Upu[7]*Y338);
I38puRMSs=abs(I38pus);
I38s=I38pus*Ibase38[1];
I38RMSs=abs(I38s);
#Linea89
I89pu=(Upu[7]-Upu[8])/Z89pu;
I89puRMS=abs(I89pu);
I89=I89pu*Ibase38[1];
I89RMS=abs(I89);
#Creo vectores con las corrientes finales en las líneas y en los
transformadores:
Ilineaspu=([I12puRMS, I34puRMS, I56puRMS, I89puRMS]);
Ilineas=([I12RMS, I34RMS, I56RMS, I89RMS]);
Itrafospup=([I23puRMSp, I45puRMSp, I67puRMSp, I38puRMSp]);
Itrafosp=([I23RMSp, I45RMSp, I67RMSp, I38RMSp]);
Itrafospus=([I23puRMSs, I45puRMSs, I67puRMSs, I38puRMSs]);
Itrafoss=([I23RMSs, I45RMSs, I67RMSs, I38RMSs]);
print ("El valor de las corrientes en  [A] en las líneas
será:",np.reshape(Ilineas,(4,1)))
print ("El valor de las corrientes en  [A] en el devanado primario de
los trafos será:",np.reshape(Itrafosp,(4,1)))
print ("El valor de las corrientes en  [A] en el devanado secundario
de los trafos será:",np.reshape(Itrafoss,(4,1)))

#Conocemos también el valor de los Ángulos en cada nodo de la red.
# Vamos a convertir los ángulos de radianes a grados
Tetaradianes = teta
Tetagrados = teta*180/np.pi
Desfase = [0,0,-30,-30,-30,-30,-30+30,-30+30,-30+30] # Incluimos el
índice horario de los trafos
Tetagrados = Tetagrados - Desfase
print ("Los ángulos de las tensiones en grados [°]
serán:",np.reshape(Tetagrados,(9,1)))

#Conocemos las Cargas Activas y Reactivas en los nodos de la red.
Pfinales = (Ppu * Sbase * -1)/(1e6)
Qfinales = (Qpu * Sbase * -1)/(1e6)
print ("Los valores de las potencias activas en [MW]
serán:",np.reshape(Pfinales,(8,1)))
print ("Los valores de las potencias reactivas en [MVar]
serán:",np.reshape(Qfinales,(8,1)))

print ("El número de iteraciones será de:",iteraciones)
print ("El error cometido ha sido:", error)
#Aquí calculamos el tiempo de ejecución del programa
end=time.time()
elapsed=end-start
print ("Tiempo de ejecución del programa: ",elapsed)
```

El error cometido por el programa es de **$7.051 \cdot 10^{-7}$.**

El tiempo de ejecución del programa es de **12.92 segundos**.

El número de iteraciones realizadas es de: **3**.

ANEXO VII: Programación del WLS incluyendo un Perfil de Cargas Diario mediante Python e Introducción de Medidas en Tiempo Real

A continuación se muestra el desarrollo del algoritmo WLS resuelto mediante el programa *PYTHON*, mediante el cual se nos proporcionarán medidas reales de potencia activa y reactiva, y de tensión, las cuales serán utilizadas en el nodo 9 de nuestra Red de Distribución. Esto se hace gracias a la inclusión del medidor Power Meter PowerLogic PM5560 de Schneider, que ha sido introducido dentro de nuestro algoritmo. De esta forma se obtienen los valores de las tensiones, las corrientes, los ángulos y las potencias en la Red de Distribución. Este código será introducido en nuestra Raspberry Pi gracias a la cual podremos visualizar en tiempo real la evolución de las medidas.

```
"""
Created on Fri Jun 15 17:39:20 2018

@author: Fran Pello
"""
#DESARROLLO DEL ALGORITMO DE ESTIMACIÓN DE ESTADO PARA UNA RED DE
#DISTRIBUCIÓN ELÉCTRICA DE 9 NODOS, INTRODUCIENDO UN PERFIL DE CARGAS
#DIARIO QUE CONSTA DE 1440 MEDIDAS DE POTENCIA ACTIVA Y REACTIVA.
#LOS DATOS INTRODUCIDOS EN EL NODO 9 DE LA RED DE DISTRIBUCIÓN SERÁN
DATOS REALES
#PROPORCIONADOS POR EL CONSUMO DE UNA BOMBILLA, Y ESTO SE CONSIGUE
GRACIAS AL USO
#DEL POWERMETER
#Importamos los diferentes módulos de Python para poder desarrollar
nuestro programa
import time
start = time.time()
import math
import numpy as np
#Introducimos la Potencia Compleja que usaremos como base
Sbase = 100e6
#Introducimos los datos de la Red de Distribución y calculamos las
#impedancias y admitancias en p.u
#Transformador 23
a23 = 0.98
Ubase23 = np.array([220000,132000],dtype = np.int64)
Ibase23 = Sbase/(math.sqrt(3)*Ubase23)
Zbase23 = (Ubase23*Ubase23)/Sbase
S23 = 270e6
Zbaset1 = (Ubase23[0]**2)/S23
Z23 = (0.009+0.129j)/2
Z23o = Z23*Zbaset1
Z23pu = Z23o/Zbase23[0]
```

```
Y23 = 1/Z23pu
#Transformador 45
a45 = 0.99
Ubase45 = np.array([132000,30000],dtype = np.int64)
Ibase45 = Sbase/(math.sqrt(3)*Ubase45)
Zbase45 = (Ubase45*Ubase45)/Sbase
S45 = 37.5e6
Zbaset2 = (Ubase45[0]**2)/S45
Z45 = (0.009+0.09j)/3
Z45o = Z45*Zbaset2
Z45pu = Z45o/Zbase45[0]
Y45 = 1/Z45pu
#Transformador 67
a67 = 1
Ubase67 = np.array([30000,13800],dtype = np.int64)
Ibase67 = Sbase/(math.sqrt(3)*Ubase67)
Zbase67 = (Ubase67*Ubase67)/Sbase
S67 = 10e6
Zbaset3 = (Ubase67[0]**2)/S67
Z67 = 0.0095+0.048j;
Z67o = Z67*Zbaset3
Z67pu = Z67o/Zbase67[0]
Y67 = 1/Z67pu
#Transformador 38
a38 = 0.98
Ubase38 = np.array([132000,13800],dtype = np.int64)
Ibase38 = Sbase/(math.sqrt(3)*Ubase38)
Zbase38 = (Ubase38*Ubase38)/Sbase
S38 = 50e6
Zbaset4 = (Ubase38[0]**2)/S38
Z38 = (0.0092+0.085j)/3
Z38o = Z38 * Zbaset4
Z38pu = Z38o / Zbase38 [0]
Y38 = 1 / Z38pu
#Linea 12
Z12 = (0.025 + 0.240j) * 4.7
Z12pu = Z12 / Zbase23 [0]
Y12 = 1 / Z12pu
#Linea 34
Z34 = (0.161 + 0.151j) * 1.5
Z34pu = Z34 / Zbase23 [1]
Y34 = 1 / Z34pu
#Linea 56
Z56 = (0.568 + 0.133j) * 0.3
Z56pu = Z56 / Zbase45 [1]
Y56 = 1 / Z56pu
#Linea 89
Z89 = (0.161 + 0.112j) * 1.8
Z89pu = Z89 / Zbase38 [1]
Y89 = 1 / Z89pu

#Cálculo de las admitancias en los transformadores
#Transformador 23
Y123 = ((1 - a23) / a23 ** 2) * Y23
Y223 = (1 / a23) * Y23
Y323 = ((a23 - 1) / a23) * Y23
#Transformador 45
```

```
Y145 = ((1 - a45) / a45 ** 2) * Y45
Y245 = (1 / a45) * Y45
Y345 = ((a45 - 1) / a45) * Y45
#Transformador 67
Y167 = ((1 - a67) / a67 ** 2) * Y67
Y267 = (1 / a67) * Y67
Y367 = ((a67 - 1) / a67) * Y67
#Transformador 38
Y138 = ((1 - a38) / a38 ** 2) * Y38
Y238 = (1 / a38) * Y38
Y338 = ((a38 - 1) / a38) * Y38

#Construimos la matriz de admitancias:
Y = np.matrix([[Y12, - Y12, 0, 0, 0, 0, 0, 0, 0],
               [-Y12, Y12 + Y123 + Y223, - Y223, 0, 0, 0, 0, 0, 0],
               [0, - Y223, Y223 + Y323 + Y34 + Y138 + Y238, - Y34, 0,
0, 0, - Y238, 0],
               [0, 0, - Y34, Y34 + Y145 + Y245, - Y245, 0, 0, 0, 0],
               [0, 0, 0, - Y245, Y245 + Y345 + Y56, - Y56, 0, 0, 0],
               [0, 0, 0, 0, - Y56, Y56 + Y167 + Y267, - Y267, 0, 0],
               [0, 0, 0, 0, 0, - Y267, Y267 + Y367, 0, 0],
               [0, 0, - Y238, 0, 0, 0, 0, Y238 + Y338 + Y89, - Y89],
               [0, 0, 0, 0, 0, 0, 0, - Y89, Y89]],dtype =
np.dtype('c16'))

# La variable 'muestras' recoge el número de medidas de potencias que
queremos
# utilizar para el flujo de cargas. Utilizaremos 70 medidas, aunque
este programa
# está preparado para leer 1440 medidas.
muestras = 70
# Estos vectores nos servirán para dibujar el perfil de cargas y
tensiones diario al final del programa.
P_plot=np.zeros((muestras))
Q_plot=np.zeros((muestras))
U_plot=np.zeros((muestras))

#A CONTINUACIÓN SE INTRODUCE EL CÓDIGO QUE NOS PERMITE MEDIR DATOS
REALES DE UNA BOMBILLA
#Y ASÍ PODER INCORPORARLOS A NUESTRO PROGRAMA
#Cargamos los diferentes módulos que necesitamos para el correcto
funcionamiento del programa
from pymodbus3.client.sync import ModbusTcpClient
from pymodbus3.constants import Endian
from pymodbus3.payload import BinaryPayloadDecoder
import threading
from datetime import datetime
from datetime import timedelta


def boolList2BinString(lst):
      return '0b' + ''.join(['1' if x else '0' for x in lst])


#IP de la que obtenemos los datos de P y Q
client = ModbusTcpClient('80.31.186.204')
#Creamos un registro de las diferentes medidas obtenidas
```

```
f = open('registro', 'a')

now_ref = datetime.now()
#Retraso necesario para el correcto funcionamiento del programa
total_delay = 4
for x in range(samples):
# Reading date and time from smart meter
# Reading registers
      #now = datetime.now()
    #Cargamos los módulos que nos dirán el momento exacto en que
tomamos las medidas
    year = client.read_holding_registers(1844,1)
    m_day = client.read_holding_registers(1845,1)
    h_min = client.read_holding_registers(1846,1)
    mili = client.read_holding_registers(1847,1)
# Las medidas se guardan en formato texto y las pasamos e formato
numérico
    year_decoder = BinaryPayloadDecoder.from_registers(year.registers,
endian=Endian.Big)
    m_day_decoder =
BinaryPayloadDecoder.from_registers(m_day.registers,
endian=Endian.Big)
    h_min_decoder =
BinaryPayloadDecoder.from_registers(h_min.registers,
endian=Endian.Big)
    mili_decoder = BinaryPayloadDecoder.from_registers(mili.registers,
endian=Endian.Big)
# Decoding year
    yr = year_decoder.decode_bits() # no borrar
    yr = year_decoder.decode_bits() # no borrar
    yr_m = yr[6],yr[5],yr[4],yr[3],yr[2],yr[1],yr[0]
# Decoding month and day
    mt = m_day_decoder.decode_bits()
    dy = m_day_decoder.decode_bits()
    mt_m = mt[5],mt[4],mt[3],mt[2],mt[1],mt[0]
    dy_m = dy[4],dy[3],dy[2],dy[1],dy[0]
# Decoding hour and minute
    hr = h_min_decoder.decode_bits()
    hr_m = hr[4],hr[3],hr[2],hr[1],hr[0]
    mn = h_min_decoder.decode_bits()
    mn_m = mn[5],mn[4],mn[3],mn[2],mn[1],mn[0]
# Decoding miliseconds
    ml = mili_decoder.decode_16bit_uint()
# Variables are represented as integer values
    year = int(boolList2BinString(yr_m),2)
    month = int(boolList2BinString(mt_m),2)
    day = int(boolList2BinString(dy_m),2)
    hour = int(boolList2BinString(hr_m),2)
    minute = int(boolList2BinString(mn_m),2)
    miliseconds = ml
# Shown date and time
#     print str(day)+'/'+str(month)+'/'+str(year)+'
'+str(hour)+':'+str(minute)+':'+str(miliseconds)

#A continuación se muestran las medidas de tensión y potencia activa y
reactiva medidas en tiempo real.
# Reading voltage and active power from the smartmeter
```

```
# Read registers
      # Voltage phase 'a' to neutral (Van)
    Van = client.read_holding_registers(3027, 2)
      # Current phase 'a' (Ia)
    Ia = client.read_holding_registers(2999, 2)
      # Active power phase 'a' (Pa)
    Pa = client.read_holding_registers(3053, 2)
      # Reactive power phase 'a' (Qa)
    Qa = client.read_holding_registers(3061, 2)
# Decoding (FLOAT32)
    #Los datos son transformados del formato de texto al formato
numérico
    Van_decoder = BinaryPayloadDecoder.from_registers(Van.registers,
endian=Endian.Big)
    Ia_decoder = BinaryPayloadDecoder.from_registers(Ia.registers,
endian=Endian.Big)
    Pa_decoder = BinaryPayloadDecoder.from_registers(Pa.registers,
endian=Endian.Big)
    Qa_decoder = BinaryPayloadDecoder.from_registers(Qa.registers,
endian=Endian.Big)
    Van_f = Van_decoder.decode_32bit_float()
    Ia_f = Ia_decoder.decode_32bit_float()
    Pa_f = Pa_decoder.decode_32bit_float()
    Qa_f = Qa_decoder.decode_32bit_float()
    # Mostramos los valores de V,P,Q. Estas medidas son las que
asignamos al nodo 9.
    print ("V9[V]:"+ str(Van_f)+"   ","P9[kW]:"+str(Pa_f)+"
","Q9[kvar]:"+str(Qa_f))
    #Buses P Q y U. Tomo como datos de partida los valores corrompidos
obtenidos
    #en mi archivo Perfilcargadiario.mat, y en la resolución del
método Direct
    #Approch con las 1440 muestras. Para ello cargo los datos de un
fichero txt como se muestra a continuación.
    #Potencias activas corrompidas
    P = np.loadtxt("Pdiaria.txt", dtype = 'i', delimiter = ',')
    P3pu = -P[0]/Sbase
    P5pu = -P[1]/Sbase
    P7pu = -P[2]/Sbase
    P8pu = -P[3]/Sbase
    #El valor de la potencia activa en el nodo 9 será el proporcionado
en tiempo real
    #Para hacer el escalado tomo el valor de P9 de la red genérica que
es de 1.7MW
    P9 = 1.7*1e6
    c=(P9)/18
    k9p=1e3*c
    P9pu = -(Pa_f*k9p)/Sbase
    #Potencias reactivas corrompidas
    Q = np.loadtxt("Qdiaria.txt", dtype = 'i', delimiter = ',')
    Q3pu = -Q[0]/Sbase
    Q5pu = -Q[1]/Sbase
    Q7pu = -Q[2]/Sbase
    Q8pu = -Q[3]/Sbase
    #El valor de la potencia reactiva en el nodo 9 será el
proporcionado en tiempo real
    k9q = k9p
```

```
    Q9pu = -(Qa_f*k9q)/Sbase
    #Tensiones corrompidas
    U = np.loadtxt("tensionesdiarias.txt", dtype = 'i', delimiter =
','')
    U1puc = U[0]/Ubase23[0]
    U2puc = U[1]/Ubase23[0]
    U3puc = U[2]/Ubase23[1]
    U4puc = U[3]/Ubase45[0]
    U5puc = U[4]/Ubase45[1]
    U6puc = U[5]/Ubase67[0]
    U7puc = U[6]/Ubase67[1]
    U8puc = U[7]/Ubase38[1]
    #El valor de la tensión en el nodo 9 será el proporcionado en
tiempo real
    #Para hacer el escalado tomo el valor de U9 de la red genérica que
es 13.8kV
    k9u = 13800/230
    U9puc = (Van_f*k9u)/Ubase38[1]

    #COMIENZO WLS
    #Vector de Potencias Activas en p.u de la Red
    Ppu = np.array([0, P3pu[x], 0, P5pu[x], 0, P7pu[x], P8pu[x],
P9pu])
    #Vector de Potencias Reactivas en p.u de la Red
    Qpu = np.array([0, Q3pu[x], 0, Q5pu[x], 0, Q7pu[x], Q8pu[x],
Q9pu])
    #Vector de tensiones de partida en p.u de la Red
    Upuc = np.array([U1puc[x], U2puc[x], U3puc[x], U4puc[x], U5puc[x],
U6puc[x], U7puc[x], U8puc[x], U9puc])
    N = 9 #número de buses y de U
    Gpu = Y.real #Parte real de la matriz de admitancias
    Bpu = Y.imag #Parte imaginaria de la matriz de admitancias
    R = np.zeros((25, 25))#Matriz diagonal, cuyos valores son las
varianzas de las P, Q y U. Tengo 25 medidas. Tengo 8P, 8Q y 9U
    varpq = 0.02 #Covarianza para las medidas de P y Q
    varu = 0.001 #Covarianza para las medidas de U
    varo = 0.001 / 10 #Covarianza para los nodos en que no tenemos P o
Q (10 veces más pequeña que la de U)
    #Introduzco las covarianzas para P
    R [0][0] = varo
    R [1][1] = varpq
    R [2][2] = varo
    R [3][3] = varpq
    R [4][4] = varo
    R [5][5] = varpq
    R [6][6] = varpq
    R [7][7] = varpq
    #Introduzco las covarianzas para Q
    R [8][8] = varo
    R [9][9] = varpq
    R [10][10] = varo
    R [11][11] = varpq
    R [12][12] = varo
    R [13][13] = varpq
    R [14][14] = varpq
    R [15][15] = varpq
    R [16][16] = varpq
```

```
    #Introduzco las covarianzas para U
    R [17][17] = varu
    R [18][18] = varu
    R [19][19] = varu
    R [20][20] = varu
    R [21][21] = varu
    R [22][22] = varu
    R [23][23] = varu
    R [24][24] = varu

    W = np.linalg.inv(R) #Matriz inversa de la matriz de covarianzas
    z = np.concatenate([Ppu, Qpu, Upuc]) #Vector de partida formado
por los valores de P, Q y U
    Upu = np.ones(N) #Iniciamos en arranque plano V=1. Valores en p.u
    teta = np.zeros(N) #Iniciamos en arranque plano teta=0
    X = np.concatenate([teta[1:], Upu]) #Vector estado (2N-1; 8 tetas
(0 SLACK) y 9 U)
    tolerancia = 10e-6
    error = tolerancia + 1
    iteraciones = 0
    #Defino un vector con las posiciones de mis P y Q
    busPQ = ([1, 2, 3, 4, 5, 6, 7, 8])
    numberPQ = len(busPQ)

    while error > tolerancia:
        iteraciones = iteraciones + 1
        #tengo que calcular las h que depende de U, P y Q
        hu = Upu
        hp = np.zeros(numberPQ)#tengo 8 medidas de P
        hq = np.zeros(numberPQ)#tengo 8 medidas de Q

        for i in range(numberPQ):#tengo 8 hp, ya que tengo 8P
(P2,P3,P4,P5,P6,P7,P8,P9)
            j = busPQ[i]
            for k in range(N):
                hp[i] = hp[i] + Upu[j] * Upu[k] * (Gpu[j, k] *
math.cos(teta[j] - teta[k]) + Bpu[j, k] * math.sin(teta[j] - teta[k]))

        for i in range(numberPQ):#tengo 8 hq, ya que tengo 8Q
            j = busPQ[i]
            for k in range(N):
                hq[i] = hq[i] + Upu[j] * Upu[k] * (Gpu[j, k] *
math.sin(teta[j] - teta[k]) - Bpu[j, k] * math.cos(teta[j] - teta[k]))

        h = np.concatenate([hp, hq, hu])

        r = np.array(z - h) #residuo

        #Tengo que calcular ahora el Jacobiano H
        HUteta = np.zeros((N, N - 1))#derivada de U respecto a teta
(9x8, ya que teta1)

        HUU = np.zeros((N, N)) #derivada de U respecto a U (9x9)
        for i in range(N):
            for k in range(N):
                if i == k:
                    HUU[i, k]= 1
```

```
        HPteta = np.zeros((numberPQ, N - 1)) #Derivada de P respecto a
teta (8x8) %Tengo 8 medidas de P
        for i in range(numberPQ):
            m = busPQ[i] #Estas son las posiciones de mis tensiones
            for k in range(N - 1):
                if k + 1 == m:
                    for n in range(N):
                        HPteta[i, k] = HPteta[i, k] + Upu[m] * Upu[n]
* (-Gpu[m, n] * math.sin(teta[m] - teta[n]) + Bpu[m, n] *
math.cos(teta[m] - teta[n]))

                    HPteta[i, k] = HPteta[i, k] - Upu[m] ** 2 * Bpu[m,
m]
                else:
                    HPteta[i, k] = Upu[m] * Upu[k + 1] * (Gpu[m, k +
1] * math.sin(teta[m] - teta[k + 1]) - Bpu[m, k + 1] *
math.cos(teta[m] - teta[k + 1]))

        HPU = np.zeros((numberPQ, N))#Derivada de P respecto a U (8x9)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N):
                if k == m:
                    for n in range(N):
                        HPU[i, k] = HPU[i, k] + Upu[n] * (Gpu[m, n] *
math.cos(teta[m] - teta[n]) + Bpu[m, n] * math.sin(teta[m] - teta[n]))

                    HPU[i, k] = HPU[i, k] + Upu[m] * Gpu[m, m]
                else:
                    HPU[i, k] = Upu[m] * (Gpu[m, k] * math.cos(teta[m]
- teta[k]) + Bpu[m, k] * math.sin(teta[m] - teta[k]))

        HQteta = np.zeros((numberPQ, N - 1)) #Derivada de Q respecto a
teta (8x8)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N - 1):
                if k + 1 == m:
                    for n in range(N):
                        HQteta[i, k] = HQteta[i, k] + Upu[m] * Upu[n]
* (Gpu[m, n] * math.cos(teta[m] - teta[n]) + Bpu[m, n] *
math.sin(teta[m] - teta[n]))

                    HQteta[i, k] = HQteta[i, k] - Upu[m] ** 2 * Gpu[m,
m]
                else:
                    HQteta[i, k] = Upu[m] * Upu[k + 1] * (-Gpu[m, k +
1] * math.cos(teta[m] - teta[k + 1]) - Bpu[m, k + 1] *
math.sin(teta[m] - teta[k + 1]))

        HQU = np.zeros((numberPQ, N))#Derivada de Q respecto a U (8x9)
        for i in range(numberPQ):
            m = busPQ[i]
            for k in range(N):
                if k == m:
                    for n in range(N):
```

```
                        HQU[i, k] = HQU[i, k] + Upu[n] * (Gpu[m, n] *
math.sin(teta[m] - teta[n]) - Bpu[m, n] * math.cos(teta[m] - teta[n]))

                    HQU[i, k] = HQU[i, k] - Upu[m] * Bpu[m, m]
                else:
                    HQU[i, k] = Upu[m] * (Gpu[m, k] * math.sin(teta[m]
- teta[k]) - Bpu[m, k] * math.cos(teta[m] - teta[k]))

        #Construyo el Jacobiano
        H1 = np.hstack([HPteta, HPU])
        H2 = np.hstack([HQteta, HQU])
        H3 = np.hstack([HUteta, HUU])
        H4 = np.vstack([H1, H2])
        H = np.vstack([H4, H3])
        # H = np.block([[HPteta, HPU],[HQteta, HQU], [HUteta, HUU]])
        #Calculo G
        G = np.dot((np.dot(np.transpose(H), W)), H)
        #Calculo el incremento de X (vector de estado)
        G1 = (np.dot((np.dot(np.transpose(H),  W)), r))
        incX = np.dot(np.linalg.inv(G), G1)
        X = X + incX
        #Obtengo las medidas de los ángulos
        teta[1:] = X[0:8]
        #Obtengo las medidas de las tensiones
        Upu[0:] = X[8:]
        error = max(abs(incX))

        #Una vez resuelto el problema de flujo de cargas, ya puedo
calcular el valor de las tensiones y de las corrientes que tendremos
en cada nodo y  cada línea de la Red de Distribución
        #Tensiones
        U1 = Ubase23[0] * abs(Upu [0])
        U2 = Ubase23[0] * abs(Upu [1])
        U3 = Ubase23[1] * abs(Upu [2])
        U4 = Ubase45[0] * abs(Upu [3])
        U5 = Ubase45[1] * abs(Upu [4])
        U6 = Ubase67[0] * abs(Upu [5])
        U7 = Ubase67[1] * abs(Upu [6])
        U8 = Ubase38[1] * abs(Upu [7])
        U9 = Ubase38[1] * abs(Upu [8])
        Ufinal = np.array([U1, U2, U3, U4, U5, U6, U7, U8, U9])
        print ("El valor de las tensiones en
kV:",np.reshape(Ufinal,(9,1)))
        #Corrientes
        #Linea12
        I12pu=(Upu[0]-Upu[1]) / Z12pu;
        I12puRMS=abs (I12pu);
        I12=I12pu*Ibase23[0];
        I12RMS=abs(I12);
        #Transformador23
        #Devanado primario
        I23pup=((Upu[1]-Upu[2])*Y223) + (Upu[1]*Y123);
        I23puRMSp=abs(I23pup);
        I23p=I23pup * Ibase23[0];
        I23RMSp=abs(I23p);
        #Devanado secundario
        I23pus=((Upu[1]-Upu[2])*Y223) - (Upu[2]*Y323);
```

```
        I23puRMSs=abs(I23pus);
        I23s=I23pus*Ibase23[1];
        I23RMSs=abs(I23s);
        #Linea34
        I34pu=(Upu[2]-Upu[3]) / Z34pu;
        I34puRMS=abs(I34pu);
        I34=I34pu * Ibase23[1];
        I34RMS=abs(I34);
        #Transformador45
        #Devanado primario
        I45pup=((Upu[3]-Upu[4])*Y245) + (Upu[3]*Y145);
        I45puRMSp=abs(I45pup);
        I45p=I45pup*Ibase45[0];
        I45RMSp=abs(I45p);
        #Devanado secundario
        I45pus=((Upu[3]-Upu[4])*Y245) - (Upu[4]*Y345);
        I45puRMSs=abs(I45pus);
        I45s=I45pus*Ibase45[1];
        I45RMSs=abs(I45s);
        #Linea56
        I56pu=(Upu[4]-Upu[5]) / Z56pu;
        I56puRMS=abs(I56pu);
        I56=I56pu*Ibase45[1];
        I56RMS=abs(I56);
        #Transformador67
        #Devanado primario
        I67pup=((Upu[5]-Upu[6])*Y267) + (Upu[5]*Y167);
        I67puRMSp=abs(I67pup);
        I67p=I67pup*Ibase67[0];
        I67RMSp=abs(I67p);
        #Devanado secundario
        I67pus=((Upu[5]-Upu[6])*Y267) - (Upu[6]*Y367);
        I67puRMSs=abs(I67pus);
        I67s=I67pus*Ibase67[1];
        I67RMSs=abs(I67s);
        #Transformador38
        #Devanado primario
        I38pup=((Upu[2]-Upu[7])*Y238) + (Upu[2]*Y138);
        I38puRMSp=abs(I38pup);
        I38p=I38pup*Ibase38[0];
        I38RMSp=abs(I38p);
        #Devanado secundario
        I38pus=((Upu[2]-Upu[7])*Y238) - (Upu[7]*Y338);
        I38puRMSs=abs(I38pus);
        I38s=I38pus*Ibase38[1];
        I38RMSs=abs(I38s);
        #Linea89
        I89pu=(Upu[7]-Upu[8])/Z89pu;
        I89puRMS=abs(I89pu);
        I89=I89pu*Ibase38[1];
        I89RMS=abs(I89);
        #Creo vectores con las corrientes finales en las líneas y en
los transformadores:
        Ilineaspu=([I12puRMS, I34puRMS, I56puRMS, I89puRMS]);
        Ilineas=([I12RMS, I34RMS, I56RMS, I89RMS]);
        Itrafospup=([I23puRMSp, I45puRMSp, I67puRMSp, I38puRMSp]);
        Itrafosp=([I23RMSp, I45RMSp, I67RMSp, I38RMSp]);
```

```
        Itrafospus=([I23puRMSs, I45puRMSs, I67puRMSs, I38puRMSs]);
        Itrafoss=([I23RMSs, I45RMSs, I67RMSs, I38RMSs]);
        print ("El valor de las corrientes en  [A] en las líneas
será:",np.reshape(Ilineas,(4,1)))
        print ("El valor de las corrientes en  [A] en el devanado
primario de los trafos será:",np.reshape(Itrafosp,(4,1)))
        print ("El valor de las corrientes en  [A] en el devanado
secundario de los trafos será:",np.reshape(Itrafoss,(4,1)))

        #Conocemos también el valor de los Ángulos en cada nodo de la
red.
        # Vamos a convertir los ángulos de radianes a grados
        Tetaradianes = teta
        Tetagrados = teta*180/np.pi
        Desfase = [0,0,-30,-30,-30,-30,-30+30,-30+30,-30+30] #
Incluimos el índice horario de los trafos
        Tetagrados = Tetagrados - Desfase
        print ("Los ángulos de las tensiones en grados
[°]:",np.reshape(Tetagrados,(9,1)))

        #Conocemos las Cargas Activas y Reactivas en los nodos de la
red.
        Pfinales = (Ppu * Sbase * -1)/ 1e6
        Qfinales = (Qpu * Sbase * -1)/ 1e6
        print ("Los valores de las potencias activas en
[MW]:",np.reshape(Pfinales,(8,1)))
        print ("Los valores de las potencias reactivas en
[MVar]:",np.reshape(Qfinales,(8,1)))

        print ("El número de iteraciones:",iteraciones)
        print ("El error cometido ha sido:", error)
```

Modificamos el vector que recoge los resultados de cada iteración del nodo 9 para posteriormente visualizar el perfil de cargas diario de este nodo.

```
    P_plot[x]=Pfinales[7]
    Q_plot[x]=Qfinales[7]

   # Modificamos el vector que recoge los resultados de cada iteración
en el nodo 9 para posteriormente visualizar el perfil de tensiones
diario de este nodo
    U_plot[x]=Ufinal[8]
    # Showing values
    # print "Van",Van_f
    # print "Pa",Pa_f
    # Writing to the database
      # Convert to string
    yr_st=str(year)
    mt_st=str(month)
    dy_st=str(day)
    hr_st=str(hour)
    mn_st=str(minute)
    ml_st=str(miliseconds)
      # Fill with '0s' for uniformity
    while len(yr_st) < 2:
            yr_st='0'+yr_st
```

```
    while len(mt_st) < 2:
            mt_st='0'+mt_st
    while len(dy_st) < 2:
            dy_st='0'+dy_st
    while len(hr_st) < 2:
            hr_st='0'+hr_st
    while len(mn_st) < 2:
            mn_st='0'+mn_st
    while len(ml_st) < 5:
            ml_st='0'+ml_st
    f.write(dy_st+'/'+mt_st+'/'+yr_st+' '+hr_st+':'+mn_st+':'+ml_st+'
'+str("%.5f" % Van_f)+'    '+str("%.5f" % Ia_f)+'    '+str("%.5f" %
Pa_f)+'    '+str("%.5f" % Qa_f)+'\n')
    now2 = datetime.now()
    delay = total_delay*x+total_delay-(now2-now_ref).total_seconds()
       #print ("%.6f" % delay)
    time.sleep(delay)

f.close()
client.close()
# Dibujamos el perfil de potencia activa (P y Q) del nodo 9
# Para ello importamos los ficheros necesarios en Python
import matplotlib.pyplot as plt
# Introducimos esta línea para dibujar conjuntamente
#las gráficas de potencias y tensiones
plt.fugure(1)
# Dibujamos P en color rojo
plt.plot(P_plot,'r',label='P')
# Dibujamos Q en color azul
plt.plot(Q_plot,'b',label='Q')
# Añadimos líneas de cuadriculas a la gráfica
plt.grid()
# Añadimos el título de la gráfica, el nombre de los ejes, y la
leyenda
plt.title('Perfil de cargas diario: Nodo 9')
plt.xlabel('Tiempo')
plt.ylabel('Potencia [MW / Mvar]')
plt.legend()


# Introducimos esta línea para dibujar conjuntamente
#las gráficas de potencias y tensiones
plt.fugure(2)
# Dibujamos el perfil de tensión del nodo 9
# Dibujamos U en color verde
plt.plot(U_plot,'g',label='Tensiones')
# Añadimos líneas de cuadriculas a la gráfica
plt.grid()
# Añadimos el título de la gráfica, el nombre de los ejes, y la
leyenda
plt.title('Perfil de tensiones diario: Nodo 9')
plt.xlabel('Tiempo')
plt.ylabel('Tension [kV]')
plt.legend()
plt.show()
```

```
#Aquí calculamos el tiempo de retraso que necesita nuestro programa
para su correcto funcionamiento
end=time.time()
elapsed=end-start  # Tiempo transcurrido (en segundos) una vez
ejecutado el programa
if elapsed<=60:
    print ("Elapsed time:",'%.2f' % elapsed,"seconds")
elif elapsed>60 and elapsed<=3600:
    elapsed=elapsed/60
    print ("Elapsed time:",'%.2f' % elapsed,"minutes")
elif elapsed>3600 and elapsed <=86400:
    elapsed=elapsed/3600
    print ("Elapsed time:",'%.2f' % elapsed,"hours")
elif elapsed>86400:
    elapsed=elapsed/86400
    print ("Elapsed time:",'%.2f' % elapsed,"days")
#Aquí calculamos el tiempo de ejecución del programa
elapsed=end-start
print ("Tiempo de ejecución del programa: ",elapsed)
```

El error cometido por el programa es de **$1.2518 \cdot 10^{-6}$.**

El tiempo de ejecución del programa para **70** muestras es de **140.34 segundos**.

El número de iteraciones para cada una de estas **70** muestras es de: **3**.

Referencias

Publicaciones

[3] Gómez Expósito, Antonio. (2002). *Análisis y operación de sistemas de energía eléctrica.* Mc Graw Hill.

[4] Luciano V. Barboza, Hans H. Zürn, Roberto Salgado. (2001). Load Tap Change Transformers: A Modeling Reminder. *IEEE Power Engineering Review*.

[5] Jen-Hao Teng. (2003). A Direct Approach for Distribution System Load. *IEEE Power Engineering Review*.

[6] Expósito, A. A. (2004). *Power System State Stimation. Theory and Implementation.* Marcel Denker.

Referencias de Internet

[1] http://es.wikipedia.org

[2] http://www.ree.es/es/

[7] http://www.powerworld.com

[8] https://www.ea.tuwien.ac.at/projects/adres_concept/EN/

[9]https://www.schneider-electric.es/es/product/METSEPM5560/pm5560-analizador-con-1mod2eth---hasta-63th-h---1%2C1m-4di-2do-52-alarmas---panel

[10] https://www.raspberrypi.org/