Artículo **Revista de la Invención Técnica**
Marzo 2017 Vol.1 No.1 10-18

Python como una alternativa factible en el análisis de sistemas eléctricos de potencia

GARCÍA-GUZMÁN, José Miguel†, VARGAS-RODRÍGUEZ, José Reyes, ORTEGA-HERRERA, F. Javier y GONZÁLEZ-PONCE, María del Refugio

Instituto Tecnológico Superior de Irapuato, Guanajuato, México. Carr. Irapuato - Silao, El Copal, 36821 Irapuato, Gto.

Recibido 3 de Enero, 2017; Aceptado 2 de Marzo, 2017

Resumen

En este trabajo se presentan implementaciones y casos de estudio de algunos modelos utilizados en el análisis de los sistemas eléctricos de potencia utilizando Python y Matlab, con el fin de mostrar las bondades del primero para ser utilizado como una alternativa factible en el análisis de estado estacionario y transitorio de sistemas eléctricos de potencia. Los análisis empleados en estre trabajo son el Despacho Óptimo de Generación con pérdidas y sin pérdidas, así como el análisis de flujos de potencia convencional con el modelo del Transformador Cambiador de Taps. El algoritmo computacional empleado en estos análisis es el mismo tanto en Python como en Matlab, ya que la finalidad es determinar el tiempo de cómputo hacia la convergencia de ambos lenguajes de programación con dichos algoritmos. Diversos casos de estudio son llevados a cabo con los sistemas de potencia de 3, 5, 30 y 39 nodos. Los resultados y gráficas obtenidas de estos casos de estudio muestran que Python requiere menor tiempo de cómputo para converger a la solución y que es equiparable con Matlab en la generación de gráficos utilizados en el análisis de los sistemas eléctricos de potencia.

Python, Matlab, sistemas eléctricos de potencia

Abstract

In this work several implementations and study cases of some models used in power systems analysis are presented by using Python and Matlab, in order to show the capabilities of Python to be used as feasible alternative in steady-state and transient power system analysis. The analyzes used in this work are Optimal Dispatch with losses and without losses, as well as the conventional power flow analysis including the Tap Changing Transformer model. The computational algorithm used in these analyzes, both in Python and Matlab, is the same because the purpose is to determine the computation time towards the convergence of both programming languages with these algorithms. Several study cases with the power systems of 3, 5, 30 and 39 nodes are carried out. The results of these cases show that Python requires lower computation time to find the solución, in addition, the Optimal Dispatch of the power system of 39 nodes and the nodal voltage profile of the system of 30 nodes are plotted in Python and these graphics are compared with Matlab, in order to visualize the potencial of Python to generate graphics that can be used in the power system analysis.

Python Matlab, electric power systems

Citación: GARCÍA-GUZMÁN, José Miguel, VARGAS-RODRÍGUEZ, José Reyes, ORTEGA-HERRERA, F. Javier y GONZÁLEZ-PONCE, María del Refugio . Python como una alternativa factible en el análisis de sistemas eléctricos de potencia. Revista de la Invención Técnica 2017. 1-1:10-18

† Investigador contribuyendo como primer autor.

Introducción

Los sistemas eléctricos de potencia (SEPs) consisten principalmente de generadores, transformadores, compensadores y cargas, los cuales están interconectados mediante líneas de transmisión. La interacción entre estos elementos hace que el modelado matemático de los SEPs sea complejo y depediente de una gran cantidad de variables. Además, la cantidad de variables depende de manera directa del tamaño de los SEPs, que varía desde sistemas de prueba del IEEE con pocos nodos hasta sistemas reales de gran escala con gran cantidad de nodos. En ambos casos, los modelos matemáticos asociados con los SEPs son resueltos, en su mayoría, en forma iterativa mediante métodos numéricos implementados en algoritmos computacionales desarrollados en distintos lenguajes de programación o mediante toolboxes de paquetes computacionales comerciales, por lo que, actualmente el análisis de los SEPs depende de manera importante de las técnicas computacionales, el análisis numérico y los lenguajes de programación (Pai & Dheeman, 2014; Saadat, 2010, Arrillaga *et al*, 1991). Alrededor del mundo, Matlab® es el paquete computacional más comúnmente utilizado por ingenieros, investigadores, empresas y académicos de prácticamente todas las áreas, ya que es un ambiente amplio de cómputo científico y numérico, así como un lenguaje de programación (MathWorks, 2017, Pyzo, 2017). Sin embargo, la principal desventaja de Matlab es que es un software comercial y su licencia tiene un costo, lo cual puede representar un problema para estudiantes o incluso académicos e investigadores que no cuenten con el recurso económico para adquirir el licenciamiento.

En las décadas recientes Python se ha convertido en una alternativa altamente viable de utilizarse como software de cómputo científico y de ingeniería, ya que es un lenguaje multiparadigma que permite desarrollar programación orientada a objetos, imperativa e incluso funcional, lo que le ha valido la confianza de ser utilizado por grandes empresas a nivel internacional como Youtube.com, Google, IronPort, Industrial Light & Magic, entre otras (PSF, 2017). A diferencia de Matlab, Python posee una licencia de código abierto denominada Python Software Foundation License, por lo que, es fácil de conseguir de manera gratuita (PSF, 2017). Python por definición es un lenguaje de programación, pero es posible adicionar una extensa variedad de paquetes, módulos o librerías como NumPy (NumPy developers, 2017), SciPy (SciPy developers, 2017), matplotlib (Hunter, *et al*, 2017), Pyomo (Hart, *et al*, 2012), entre otras, que convierten a Python en un software de cómputo científico comparable con paquetes comerciales como Matlab u Octave. Incluso con estas librerías Python puede sustituir a estos paquetes comerciales en aplicaciones donde se requiera el uso del cómputo científico.

Lo anterior, le ha permitido a Python ganar terreno en el gusto de los investigadores, ingenieros y académicos de las distintas áreas del conocimiento alrededor del mundo, sin embargo, en el análisis de los sistemas eléctricos de potencia el software más utilizado es Matlab y el uso de Python es muy escaso, probablemente debido al uso histórico de Matlab y a la poca difusión de la utilidad, facilidad y confiabilidad de Python en el análisis de los SEPs, al grado de que solo muy pocas paquetes comutacionales como Dome (Milano, 2016) y PyPSA (FRESNA, 2017) enfocados a esta área están desarrolladas en este lenguaje.

ISSN 2523-6792

En este contexto, en el presente trabajo se presentan implementaciones y casos de estudio de diversos modelos utilizados en el análisis de los SEPs, hechas en Python y Matlab, con el fin de mostrar las bondades del primero que le permitan ser considerado como una alternativa factible para ser utilizado en el análisis de los sistemas eléctricos de potencia, tanto en estado estacionario como transitorio.

Python como software de cómputo científico

Como se mencionó antes, Python es escencialmente un lenguaje de programación fácil de aprender y leer, el cual puede ser utilizado en aplicaciones de cómputo científico mediante la adición de librerías númericas especializadas. Cuando se instala Python estas librerías no se incluyen implícitamente en dicha instalación y por tanto es necesario instalarlas, circunstancia que se traduce en una desventaja de Python frente a otros paquetes comerciales. Sin embargo, con el paso de los años se han desarrollado algunas distribuciones de Python que incluyen, además del mismo lenguaje de programación, diversas librerías orientadas al cómputo cienfíco y que están, por mencionarlo de alguna manera, listas para usar. Estas distribuciones siempre incluyen las librerías base como NumPy, Scipy y Matplotlib en determinados ambientes de programación como Spyder u otro ADI (Ambiente de Desarrollo Integrado) (Rodríguez, 2014).

Algunas de las distribuciones más comunes son: Python(x,y) (Raybaut, P. & Davar, G., 2015), Enthought Canopy (ENTHOUGHT, 2016), WinPython (Raybaut, P., 2014) y Anaconda (Continuum Analytics, 2017).

De estas, Anaconda es la más completa, es posible instalarla junto con otras distribuciones de Python, tiene un administrador de librerías denominado Conda que permite instalar, crear y actualizar paquetes y librerías de manera muy sencilla (Rodríguez, 2014).

Con las librerías adecuadas Python puede sustuir al lenguaje de programación y los toolboxes de Matlab, excepto Simulink. NumPy es una librería que permite realizar operaciones con vectores y matrices en forma similar a Matlab, mientras Scipy es una librería que contiene herramientas de fácil uso para el usuario, con las cuales es posible llevar a cabo integración numérica, optimización, procesamiento de señales, cálculo simbólico, entre otras. Esta librería que Python ofrece es equivalente a algunos de los toolboxes de Matlab. Por otro lado, Matplotlib es la librería de Python que permite obtener gráficas en 2D y 3D, en un ambiente visual similar a Matlab. Además de estas librerías, existen otras más especializadas que se pueden instalar a través de Anaconda. Algunas de las librerías que se pueden mencionar son a) Pyomo: optimización compleja, similar con AMPL (KNITRO, 2017) y GAMS (Drud, 1996), b) DEAP: programación evolutiva (Fortin, *et al*, 2012), c) Pandas: análisis y estructura de datos (McKinney, 2012), d) Kivy: desarrollo de apps para Android (DO, 2017), e) PyWheater: recolección y análisis de datos meteorológicos (PSF, 2017), f) PyQt: Interfaces gráficas multiplataforma (PyQt, 2016), además las mencionadas PyPSA y Dome y muchas más librerías disponibles manejadas también como software libre.

Lo antes expresado muestra que Python puede ser utilizado de manera satisfactoria para llevar a cabo estudios relacionados con el análisis de SEPs.

ISSN 2523-6792

Breve comparativa entre Python y Matlab

En este trabajo se compara Python con Matlab, ya que este último es el paquete computacional más utilizado en el análisis de los sistemas de potencia y se pretende mostrar el potencial de Python para ser considerado como un paquete computacional alterno en el análisis de los SEPs. Algunas de las diferencias y semejanzas entre tales paquetes se describen enseguida.

La principal diferencia entre ambos paquetes es que la licencia de Python es de código abierto y, por tanto, gratuita, mientras que el licenciamiento de Matlab tiene costo. Otra de las diferencias muy marcadas entre ellos es que Matlab ofrece, en una sola instalación, un paquete completo de cómputo científico que incluye toolboxes, lenguaje de programación y de un ADI. Mientras que en el caso de Python es necesario instalar una a una, además del lenguaje de programación, las librerías para tener un software de cómputo científico, aunque se tienen distribuciones en las que en una sola instalación se incluyen la mayoría de librerías de Python, pero no se incluye el ADI, de modo que también es necesario instalarlo de manera independiente. En lo que respecta al lenguaje de programación ambos son fáciles de leer; Python utiliza palabras en lugar de símbolos, por ejemplo, los operadores lógicos se escriben como tal "and", "or", "nor", mientras que en Matlab se escriben como &&, ||, !, respectivamente. Los dos paquetes pueden manejar matrices, vectores y variables dinámicas, pero en Python los índices de los arrays comienzan en cero y no en uno como en el caso de Matlab.

El código de Python tiende a ser más compacto y legible (Feldman, 2016), aunque para el usuario es un poco más sencillo implementar código en Matlab, ya que Python es un lenguaje de propósito general (Mathworks, 2017), esto último hace que las estructuras de datos en Python sean superiores a las de Matlab y brinda la posibilidad de tener un mejor control y mantenimiento del código (Feldman, 2016). En cuanto a las librerías, en Python se trabaja con interfaces distintas dependiendo de la utilizada y, en su naturaleza, cada librería es independiente entre sí, aunque pueden enlazarse para trabajar juntas sin mayor problema. Mientras que en Matlab se tiene una interfaz común y los toolboxes son diseñados para trabajar juntos entre sí (Mathworks, 2017). En cuanto a simulaciones numéricas, se debe mencionar que Python ejecuta estas en un menor tiempo de cómputo que Matlab, lo cual es importante en aplicaciones donde se requiere resolver modelos que implican un esfuerzo computacional considerable. En caso de que desee profundizar en las similitudes y diferencias entre ambos paquetes es recomendable revisar a detalle en la bibliografía indicada anteriormente.

Análisis de sistemas eléctricos de potencia

La solución de flujos de potencia convencional y el Despacho Óptimo de Generación son unas de las herramientas más utilizadas por los investigadores, empresas y académicos alrededor del mundo para llevar a cabo el análisis de los sistemas eléctricos de potencia, motivo por el cual se utilizan estos para ser implementados en Python.

Análisis de flujos de potencia convencional

El análisis de FP se basa en las ecuaciones de inyección de potencia siguientes (Saadat, 2010),

$$P_i = \sum_{j=1}^{n} Y_{ij} V_i V_j \cos\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{1}$$

ISSN 2523-6792

$$Q_i = -\sum_{j=1}^{n} Y_{ij} V_i V_j \sin\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2}$$

Estas ecuaciones forman un sistema de ecuaciones no lineales dependiente del ángulo de fase δ y de la magnitud del voltaje $|V|$, el cual es resuelto mediante el método de Newton-Raphson. Al resolver este sistema se debe establecer el siguiente balance de potencia activa y reactiva en el nodo *i*.

$$\Delta P_i = P_{Gi} - P_{Di} - \sum_{j=1}^{N_b} P_i^{iny\ j} = 0 \tag{3}$$

$$\Delta Q_i = Q_{Gi} - Q_{Di} - \sum_{j=1}^{N_b} Q_i^{iny\ j} = 0 \tag{4}$$

donde el subíndice *G* y *D* representan la potencia generada y demandada respectivamente, y el término de la sumatoria representa la potencia inyectada.

El proceso iterativo se ejecuta n-veces hasta que el balance de potencia activa y reactiva dado por (3) y (4) sea menor a una tolerancia de convergencia especificada.

Despacho Óptimo de Generación

El Despecho Óptimo de Generación (DOG) consiste en asignar un valor de potencia de generación a cada generador de un SEP, de manera que la suma de estas potencias satisfaga la demanda de potencia con el mínimo costo de generación. Este problema se formula matemáticamente como un problema de optimización como sigue (Saadat, 2010),

$$min\ \ F(P_{Gi}) = \sum_{i=1}^{ng} \left(a_i + b_i P_{Gi} + c_i P_{Gi}^2\right) \tag{5}$$

$$\text{sujeto a } \sum_{i=1}^{ng} P_{Gi} = P_D + P_L \tag{6}$$

$$P_{Gi}^{\min} \le P_{Gi} \le P_{Gi}^{\max} \tag{7}$$

Las Ecuaciones (5), (6) y (7) representan el costo de generación, el balance de potencia activa entre potencia demandada y generada, y los límites de generación, respectivamente.

Casos de estudio

Con el fin de ilustrar la factibilidad del uso de Python en el análisis de sistemas eléctricos de potencia se llevan a cabo diversos casos de estudio en los que se a) detemina el costo mínimo de operación cuando se satisface la demanda de potencia y b) el punto de operación de estado estacionario de sistemas de potencia. Los casos de estudio son llevados a cabo con el sistema de prueba de 3 y 5 nodos (Stagg & El-Abiad, 1968; Saadat, 2010), el sistema de 30 nodos del IEEE, el cual es una sección de la Corporación de Servicios de Potencia Eléctrica Americana (CSPEA, por sus siglas en inglés) (Saadat, 2010) y el sistema de Nueva Inglaterra de 39 nodos y 10 generadores (Pai, 1989).
Todos casos de estudio presentados en este trabajo son llevados a cabo con una tolerancia de convergencia $1x10^{-9}$.

Casos de estudio para el DOG

En la Tabla 1 se presenta una comparación de resultados del DOG sin pérdidas y con pérdidas, ambos considerando los límites de generación, utilizando Python y Matlab. Para el caso de estudio sin pérdidas se utiliza el sistema de Nueva Inglaterra de 39 nodos y se satisface una demanda de 6,097.10 MW, mientras que para el caso con pérdidas se utiliza el sistema de prueba de 3 nodos y se satisface una demanda de 150 MW.

Parámetro	**Python ® (MW)**	**Matlab ® (MW)**
Sistema de 39 nodos (sin pérdidas)		
Costo ($/MWhr)	15.8668	15.8668
Costo ($/hr)	60171.52	60171.52
Tiempo (seg)	0.001052	0.022847

ISSN 2523-6792

Sistema de 3 generadores (con pérdidas)		
Pérdidas	2.6687	2.6687
Costo ($/MWhr)	7.7678	7.7677
Costo ($/hr)	1599.98	1599.98
Tiempo (seg)	0.003591	0.033418

Tabla 1 Comparación de resultados del DOG considerando y sin considerar pérdidas.

Los resultados muestran que los valores obtenidos con ambos lenguajes son idénticos, ya que estos dependen directamente del algoritmo de solución y no del lenguaje de programación utilizado. Sin embargo, en la tabla anterior se muestra que el tiempo de cómputo requerido para converger a la solución del problema de DOG es menor en Python que en Matlab. Se debe notar que en este caso, el tiempo de cómputo empleado por ambos lenguajes no es significativo porque el problema de DOG no es un problema complejo, sin embargo, en algunos análisis de SEPs de gran escala el tiempo de cómputo es considerable y se convierte en un parámetro importante a tomarse en cuenta por los investigadores, empresas y académicos que trabajan con los sistemas de potencia. En este sentido, de acuerdo a los resultados obtenidos en este trabajo, al parecer Python ofrece la ventaja de alcanzar la convergencia en un menor tiempo de cómputo que Matlab.

Por otro lado, en las Figuras 1 y 2 se muestran las gráficas de barras del DOG del sistema de 39 nodos obtenidos utilizando Python y Matlab, respectivamente. Se debe notar que Python ofrece un potencial similar para graficar que puede ser comparado con Matlab y que lo vuelve factible para obtener cualquier tipo de gráfico empleado en el análisis de sistemas de potencia.

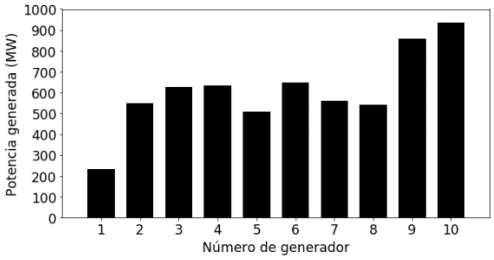

Gráfico 1 DOG del sistema de 39 nodos obtenido con Python.

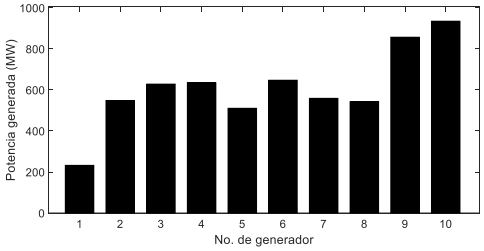

Gráfico 2 DOG del sistema de 39 nodos obtenido con Matlab

Casos de estudio para el análisis de flujos de potencia

En los casos de estudio de flujos de potencia convencional se utiliza el sistema de prueba de 5 nodos y una sección del sistema de la CSPEA de 30 nodos. En este sistema están instalados seis generadores, dos bancos de compensación y cuatro transformadores cambiadores de taps, cuyo modelo es considerado en la formulación de flujos de potencia, mientras que en el sistema de 5 nodos no hay transformadores. La Tabla 2 muestra los resultados de flujos de potencia obtenidos mediante Python y Matlab. De igual manera que en el caso del DOG, el tiempo de cómputo requerido para encontrar la solución de flujos de potencia es menor en Python.

Como se observa, en este caso el tiempo de cómputo comienza a incrementarse, ya que el análisis de flujos de potencia es un problema que involucra más ecuaciones y variables, lo que lo hace más complejo desde una perspectiva computacional.

ISSN 2523-6792

Parámetro	Python®	Matlab®
Sistema de 5 nodos		
Generación (MW)	171.122	171.122
Generación (MVAR)	29.223	29.223
Demanda (MW)	165	165
Demanda (MVAR)	40	40
Pérdidas (MW)	6.122	6.122
Pérdidas (MVAR)	-10.777	-10.777
Tiempo (segundos)	0.015703	0.176936
Sistema de 30 nodos		
Generación (MW)	300.999	300.999
Generación (MVAR)	125.144	125.144
Demanda (MW)	283.400	283.400
Demanda (MVAR)	126.200	126.200
Pérdidas (MW)	17.599	17.599
Pérdidas (MVAR)	22.244	22.244
Tiempo (segundos)	0.084653	0.228167

Tabla 2 Resumen de resultados del punto de operación obtenido del análisis de flujos de potencia para el sistema de 5 y 30 nodos.

Los gráficos 3 y 4 muestran los perfiles de voltaje del sistema de potencia 30 nodos mediante Python y Matlab, en forma respectiva.

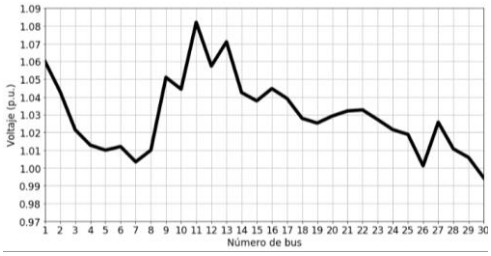

Gráfico 3 Perfil de voltaje nodal del SEP de 30 nodos obtenido con Python

Gráfico 4 Perfil de voltaje nodal del SEP de 30 nodos obtenido con Matlab.

Se debe mencionar que los casos de estudio se llevaron a cabo utilizando una computadora personal marca Dell con procesador Intel (R) Core i7-5500U CPU a 2.40 GHz y una memoria RAM de 6 GB en un sistema operativo de 64 bits.

Conclusiones

En este trabajo diversos casos de estudio relacionados con el análisis de sistemas eléctricos de potencia han sido presentados utilizando Python y Matlab como lenguaje de programación, con la finalidad de mostrar la factibilidad del uso de Python en el análisis de los SEPs. Una de las características más importantes que presenta Python respecto a Matlab es la de ser un software de cómputo científico completamente gratuito, mientras que la licencia de Matlab tiene costo. Por otro lado, el lenguaje de programación utilizado en Python tiende a ser más compacto y legible, sin embargo, resulta un poco más sencillo implementar código en Matlab. Los resultados obtenidos en los casos de estudio muestran que el tiempo de cómputo requerido por el lenguaje de programación de Python es menor que el requerido por el lenguaje de Matlab.

En los análisis de sistemas de potencia presentados el tiempo de cómputo no resultó significativo, debido a que dichos análisis no son numéricamente complejos y los SEPs utilizados no son de gran escala. Esta condición cambia cuando se trabaja, por ejemplo, con sistemas de gran escala o cuando se realiza análisis transitorio, en donde el tiempo de cómputo se convierte en un parámetro de gran importancia para ser considerado por los investigadores, empresas y académicos que trabajan con los sistemas de potencia.

ISSN 2523-6792

Por ultimo, es importante resaltar que Python ofrece una librería denominada Matplotlib que brinda un potencial similar a Matlab que permite obtener cualquier tipo de gráfico en dos y tres dimensiones que resultan muy útiles en el análisis de sistemas de potencia.

Referencias

Arrillaga, J., Arnold, C. P. & Harker, B. J. (1991). *Computer Modelling of Electrical Power Systems.* Nueva Dehli: John Wiley & Sons.

Continuum Analytics. *ANACONDA Leading Open Data Science Platform Powered by Python*. (2017). Recuperado de https://www.continuum.io/anaconda-overview

Digital Ocean (DO). *Kivy - Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps*. (2017). Recuperado de https://kivy.org/#home

Drud, A. S.*GAMS/CONOPT (1996)*, Bagsvaerd: Denmark. ARKI Consulting and Development. Recuperado de http://www.gams.com/

ENTHOUGHT. *Enthought Canopy - Proven Scientific Python Distribution Plus Integrated Analysis Environment*. (2016). Recuperado de https://www.enthought.com/products/canopy/

Feldman, P. M. *Eight Advantages of Python Over Matlab*. (2016). Recuperado de http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html

FIAS - Renewable Energy System and Network Analysis: RESNA (FRESNA). *PyPSA: Python for Power System Analysis*. (2017). Recuperado de https://www.pypsa.org/

Fortin, F. A., De Rainville, F. M., Gardner, M. A., Parizeau, M. & Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy, *Journal of Machine Learning Research*, 13(1), pp. 2171-2175.

Hart, E. W., Laird, C., Watson, J. P. & Woodruff, A. L. (2012). *Pyomo – Optimization Modeling in Python.* Nueva York: Springer.

Hunter, J., Dale, D., Firing, E., Droettboom, M. & Matplotlib development team. (2017). *Matplotlib User's Guide*.

KNITRO. (2017). En *Composants Numériques de Artelys Tools*.Recuperado de

http://www.artelys.com/index.php?page=knitro&hl=en_EN

Mathworks Inc. *MATLAB vs. Python: Top Reasons to Choose MATLAB*. (2017). Recuperado de https://es.mathworks.com/products/matlab/matlab-vs-python.html#comparison_table

McKinney, W. (2012). *Python for Data Analysis*. Massachusetts: O'Reilly Media.

Milano, F. *Dome*. (2016). Recuperado de http://faraday1.ucd.ie/dome.html

NumPy developers. *NumPy*. (2017). Recuperado de http://www.numpy.org/#

Pai, M. A & Dheeman, C. (2014). *Computer Techniques in Power System Analysis*. India: McGraw-Hill.

Pai, M. A. (1989). *Energy Function Analysis for Power System Stability*. Norwell: Kluwer Academic Publishers.

Python Software Foundation (PSF). *Documents of Python*. (2017). Recuperado de www.python.org.

ISSN 2523-6792

Python Software Foundation (PSF). *Documents of Python*. (2017). Recuperado de https://docs.python.org/3/license.html

Python Software Foundation (PSF). *PyWeather 0.7.0: Weather related functions, console readers, and service publishers*. (2017). Recuperado de https://pypi.python.org/pypi/PyWeather

Pyzo. *Python vs Matlab.* (2017). Recuperado de http://www.pyzo.org/python_vs_matlab.html

Raybaut, P. & Davar, G. *Python(x,y) - the scientific Python distribution*. (2015). Recuperado de http://python-xy.github.io/

Raybaut, P. *WinPython*. (2014). Recuperado de http://winpython.github.io/

Riverbank Computing Limited (RCL). *PyQT*. (2016). Recuperado de https://riverbankcomputing.com/news

Rodríguez O. A. *Cómputo científico con Python y Anaconda*. (2014). Recuperado de http://gatomontez.com/articulo/2014/02/16/computo-cientifico-con-python-y-anaconda/#.WUfzu1U1-Um

Saadat, H. (2010). *Power System Analysis*. USA: PSA Publishing LLC.

SciPy developers. *SciPy: Scientific Computing Tools for Python*. (2017). Recuperado de https://scipy.org/about.html

Stagg, G.W. & El-Abiad A. H. (1968). *Computer Methods in Power System Analysis*. USA: McGraw-Hill.

ISSN 2523-6792