Demanda de electricidad estocástica

Alejandro Campos, Darian Dominguez, Daniel Dominguez

Facultad de Matemática y Computación
Universidad de la Habana
2022

Resumen

Para estudiar los mercados de electricidad desregulados, se han propuesto modelos. La mayoría de ellos corresponden a un llamado juego de líder-seguidor en el que un Operador de Sistema Independiente (ISO) juega un papel central. Nuestro objetivo es considerar funciones de oferta cuadráticas junto con las pérdidas de transmisión en el juego multi-líder-seguidor y, a partir de estas, calcular la producción garantizando que su costo sea mínimo. Usaremos la librería de python GEKKO para lograrlo.

I. Introducción

Con la desregulación y privatización del mercado eléctrico en muchos países desde mediados de la década de 1980, han aparecido nuevos modelos en la literatura. Estos modelos se basan fundamentalmente en juegos no cooperativos que se pueden describir de la siguiente manera: el mercado de la electricidad es centralizado por un Operador de Sistema Independiente (ISO), donde cada agente ofrece su costo de producción y demanda al ISO que calcula la mejor producción, con el fin de minimizar el costo de esta. Esto lleva a un juego llamado multi-líder-seguidor en el que cada agente se enfrenta a un problema de dos niveles.

Los consumidores y los generadores de electricidad están ubicados en diferentes nodos conectados por líneas de transmisión de energía. En la mayoría de los modelos existentes se supone que las ofertas de los generadores siguen leyes lineales o afines. Sin embargo, es interesante considerar los mercados al contado, en los que las funciones de oferta de la generación (producción) o de los consumidores (demanda) están más estructurados que los lineales. Por lo que, en este problema trabajamos con funciones de demanda y costo cuadráticas.

Por otro lado, en algunas situaciones como transporte de larga distancia o alta intensidad, las pérdidas de transmisión a lo largo de las líneas no pueden pasarse por alto. Por ello, la influencia de las pérdidas de transmisión en el equilibrio del sistema se ha tenido en cuenta también en este problema. Nuestro objetivo es trabajar simultáneamente funciones de oferta no lineales para generadores y las pérdidas de transmisión en la red.

En las diferentes secciones del presente trabajo nos concentraremos en explicar de una forma más precisa este problema de demanda de electricidad, proponiendo un

modelo matemático que se ajusta a este. A este modelo le daremos solución mediante la librería GEKKO del lenguaje de progamación python. No sin antes explicar cómo esta librería resuelve los problemas de optimización. Además, explicaremos un ejemplo muy sencillo para un mejor entendimiento de toda la teoría que abordaremos. Por último, abordaremos las distintas distribuciones para la generación de la demanda de electricidad de cada nodo.

II. Modelo matemático

Consideramos un mercado puntual de electricidad basado en una red de transmisión. Los agentes de los mercados son los productores. Cada nodo de la red está compuesto por un productor y un consumidor (eventualmente con una demanda nula). Se supone que las líneas de la red están orientadas. Cada línea tiene una capacidad de transmisión máxima y se consideran las pérdidas térmicas. De acuerdo a modelos clásicos las pérdidas térmicas son proporcionales al cuadrado de la transmisión que fluye a lo largo de la línea.

El mercado de electricidad está regulado por un operador de sistema independiente (ISO). Por lo tanto cualquier agente (como productor) proporciona al ISO una función de oferta cuadrática dada por los parámetros a_i y b_i. Luego, el ISO calcula el conjunto de producciones admisibles para cada nodo i (q_i) y transmisiones entre cada nodo i y j (t_{ij}), conociendo:

- N número de agentes del mercado de electricidad,
- L conjunto de pares $< i, j >$ que representan las líneas de electricidad entre el agente i y el agente j,
- L_{ij} coeficiente de pérdida térmica correspondiente a la línea $< i, j >$, y
- D_i demanda de electricidad correspondiente al nodo i

Se sabe, además, que el costo de generación de cada nodo está dado por la expresión $a_i q_i + b_i q_i^2$ con $i = 1, ..., N$. Donde a_i y b_i corresponden a la i-ésima posición de los vectores A y B, de tamaño N, conocidos.

Por último, el coeficiente de pérdida térmica viene dado por $L_{ij} t_{ij}^2$ y es asumido equitativamente por los nodos i y j.

Así, el ISO, conociendo los parámetros anteriores, calcula $q = (q_i)_{i \in N}$ y $t = (t_{ij})_{ij \in L}$, no negativos, de forma tal que se minimice el costo total de generación y se satisfaga la demanda de electricidad de cada nodo, es decir, resolver el siguiente problema de optimización:

$$\begin{aligned} \min_{q,t} \quad & \sum_{i=1}^{N} a_i q_i + b_i q_i^2 \\ & s.a. \;\; q_i \geq 0, i = 1, ..., N \\ & t_{ij} \geq 0, ij \in L \\ q_i - \sum_{k:ik \in L} (t_{ik} + \frac{L_{ik}}{2} t_{ik}^2) + & \sum_{k:ki \in L} (t_{ki} - \frac{L_{ki}}{2} t_{ki}^2) \geq D_i, \\ & i = 1, ..., N \end{aligned}$$

En este problema la única restricción no trivial es la última. Esta asegura que se satisfaga la demanda de cada nodo i, planteando que la energía de un nodo i debe ser mayor o igual que la demanda de este. En esta restricción tenemos dos sumatorias, la primera representa la energía que sale de un nodo i, es decir, se suman los flujos de las líneas de salida de este hacia otros nodos sumando, además, lo que aporta este nodo a la pérdida térmica de dichas líneas que, como se asume equitativamente, solo aporta la mitad de la pérdida térmica. La segunda sumatoria representa la energía que entra al nodo i, o sea, se suman los flujos de las líneas que llegan a este nodo, quitando la mitad de la pérdida térmica que debe asumir dicho nodo en el flujo de las líneas en cuestión. Luego, la energía total que sale del nodo se representa con signo negativo, mientras que la que entra a este se representa con signo positivo. Por tanto, la energía de cada nodo corresponde a lo que

produce menos la energía total que sale de este sumado a la energía total que llega.

III. GEKKO

GEKKO es un software de optimización para ecuaciones algebraicas diferenciales y enteras mixtas. Se combina con solucionadores a gran escala para programación de enteros lineales, cuadráticos, no lineales y mixtos (LP, QP, NLP, MILP, MINLP). Los modos de operación incluyen reconciliación de datos, optimización en tiempo real, simulación dinámica y control predictivo no lineal.

GEKKO es una abstracción de alto nivel de problemas de optimización matemática. Los valores de los modelos se definen mediante constantes, parámetros y variables. Estos están relacionados entre sí por intermedios o ecuaciones. Las funciones objetivas se definen para maximizar o minimizar ciertos valores. Los objetos son colecciones integradas de valores (constantes, parámetros y variables) y relaciones (intermedios, ecuaciones y funciones objetivas). Los objetos pueden basarse en otros objetos con relaciones orientadas a objetos.

En el back-end GEKKO compila un modelo a código de bytes y realiza la reducción del modelo en base al análisis de la estructura de dispersión (incidencia de variables en ecuaciones o función objetivo) del modelo. El núcleo de todos los modos es el modelo no lineal. Una vez que la solución está completa, se escriben los resultados en `results.json` que GEKKO vuelve a cargar en las variables de Python.

Al combinar los enfoques de los lenguajes de modelado algebraico (AML) típicos y los paquetes de control óptimos, GEKKO facilita enormemente el desarrollo y la aplicación de herramientas como el control predicativo de modelos no lineales (NMPC), la optimización en tiempo real (RTO) y simulación dinámica.

Como GEKKO constituye un lenguaje de modelado algebraico (AML), nos permite plantear problemas de optimización en modelos sencillos, basados en ecuaciones orientadas a objetos, para interactuar con potentes solucionadores de optimización incorporados, que tienen la capacidad para ejecutar control predictivo de modelos y optimización en tiempo real.

GEKKO, al ser un AML, facilita la interfaz entre los solucionadores avanzados y los usuarios. Los solucionadores de gama alta requieren una amplia información sobre el problema, incluidos los límites de las variables, las funciones de restricción y las funciones objetivas, todo en un formato coherente. GEKKO simplifica el proceso al permitir que el modelo se escriba en un formato simple e intuitivo. El lenguaje de modelado acepta un modelo (restricciones) y un objetivo a optimizar. GEKKO maneja los enlaces al binario del solucionador, mantiene el formato requerido de los solucionadores y expone las funciones necesarias. Las llamadas de función necesarias incluyen residuales de restricción, valores de función objetivo y derivadas. La mayoría de los lenguajes de modelado modernos, como GEKKO, aprovechan la diferenciación automática para facilitar gradientes exactos sin una definición de derivada explícita por parte del usuario.

Dado que Python está diseñado para la legibilidad y la facilidad en lugar de la velocidad, la librería GEKKO de python se convierte en una representación de bajo nivel en el back-end de Fortran para acelerar las llamadas a funciones. La diferenciación automática proporciona los gradientes necesarios, exactos a la precisión de la máquina, sin trabajo adicional por parte del usuario. Luego, GEKKO interactúa con los solucionadores integrados de código abierto, comerciales y personalizados a gran escala para la optimización en el back-end. Posteriormente, los resultados de la optimización, como ya se dijo, se vuelven a cargar en Python para facilitar el acceso y un mayor análisis o manipulación.

Por tanto podemos concluir que GEKKO es una biblioteca de Python orientada a objetos que ofrece construcción de modelos,

herramientas de análisis y optimización que ofrece muchas ventajas. Es por ello que utilizamos GEKKO para resolver nuestro problema de optimización del mercado de electricidad.

IV. Resolución del modelo

Para la resolución del modelo, como afirmamos en la sección anterior, nos auxiliamos de la librería GEKKO de python. La implementación se puede encontrar en `utils/solveModel.py`.

Primeramente creamos un objeto de tipo GEKKO que consistirá nuestro modelo y, posteriormente vamos añadiendo a este las diferentes variables, restricciones y la función objetivo. Guardamos los q_i en una lista de tamaño N, estos son objetos de tipo GEKKO.Var con límite inferior igual a 0. Los t_{ij} se crean de forma análoga y los guardamos en una matriz de NxN que contiene la variable de GEKKO si $< i, j > \in L$ y -1 en caso contrario.

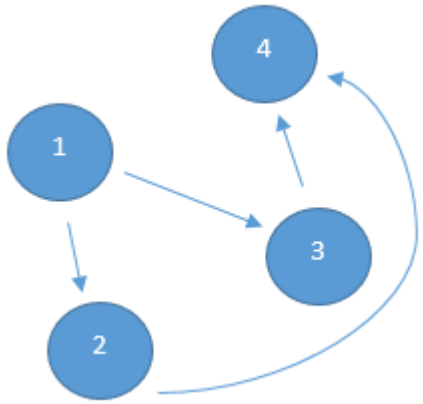
Despúes de añadir las variables a nuestro modelo, le agregamos las restricciones. En este caso, como definimos nuestras variables no negativas, solo nos queda adicionar la restricción de satisfacción de la demanda. Recordemos que es una restricción para cada nodo, por tanto tenemos que añadir N restricciones. Con ayuda de GEKKO.Equation creamos las restricciones tal y como se definieron en la sección II. Por último, le agregamos la función objetivo al modelo como se expuso en dicha sección.

Antes de resolver el modelo se define el modo de funcionamiento de GEKKO. Esta librería tiene varios modos de funcionamiento, ajustables con el parámetro model.options.IMODE que define el tipo de problema. Cada tipo de problema trata las clases de variables de manera diferente y crea ecuaciones para cumplir con el objetivo particular heredado de este. En este caso escogimos el modo 3 ya que es el estado estable para resolver problemas de optimización en tiempo real. Finalmente, resolvemos el problema con GEKKO.solve().

La siguiente sección muestra un ejemplo donde se reflejan los resultados de ejecutar la implementación antes expuesta.

V. Ejemplo

Veamos el siguiente ejemplo en el que tenemos 4 agentes, que presentan las siguientes conexiones:



Supongamos, además, que las demandas de los agentes 2 y 4 tienen demanda nula y la demanda del resto de los nodos viene dada por: $D_1 = 5$, $D_3 = 1$. Tenemos que los vectores A y B son $A = \{1, 4, 5, 8\}$, $B = \{2, 4, 4, 1\}$. Por último, desestimemos la pérdida térmica de todas las líneas menos de la línea $< 1, 3 >$ que es igual a 2.

Ya tenemos todos los datos que necesitamos para resolver el modelo. Si le pasamos estos datos a nuestro método solveModel y miramos en la consola, obtenemos:

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 1.8109982e-001 5.02e+000 4.37e+000   0.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 3.8886858e-001 4.98e+000 4.42e+000   0.0 2.98e+000    -  4.98e-003 8.94e-003h  1
   2 7.3842557e+001 3.15e-002 2.17e+001  -0.6 5.40e+000    -  5.36e-002 1.00e+000h  1
   3 6.7233978e+001 2.69e-002 1.17e+001  -2.6 1.55e+000    -  1.00e+000 1.71e-001f  1
   4 6.3414013e+001 3.10e-002 2.88e+000  -2.8 2.70e-001    -  1.00e+000 7.31e-001f  1
   5 6.4098932e+001 2.13e-006 2.94e-003  -2.0 6.99e-002    -  1.00e+000 1.00e+000h  1
   6 6.4002246e+001 2.04e-008 3.02e-004  -4.1 1.03e-002    -  9.98e-001 9.94e-001f  1
   7 6.4000002e+001 1.25e-010 5.35e-007  -9.7 3.78e-004    -  1.00e+000 9.99e-001f  1

Number of Iterations....: 7

                                   (scaled)                 (unscaled)
Objective...............:  6.4000001551989470e+001   6.4000001551989470e+001
Dual infeasibility......:  5.3483850814828315e-007   5.3483850814828315e-007
Constraint violation....:  1.2470700564913275e-010   1.2470700564913275e-010
Complementarity.........:  5.7882404618494599e-007   5.7882404618494599e-007
Overall NLP error.......:  5.7882404618494599e-007   5.7882404618494599e-007


Number of objective function evaluations             = 8
Number of objective gradient evaluations             = 8
Number of equality constraint evaluations            = 8
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 8
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 7
Total CPU secs in IPOPT (w/o function evaluations)   =      0.143
Total CPU secs in NLP function evaluations           =      0.002

EXIT: Optimal Solution Found.

 The solution was found.

 The final value of the objective function is  64.00000155198947

--------------------------------------------------------
Solver         :  IPOPT (v3.12)
Solution time  :  0.2809000000000004 sec
Objective      :  64.00000157772452
Successful solution
--------------------------------------------------------


[[4.9999999846], [4.9458055967e-07], [0.99999999409], [0.0]]
[[-1, [0.0], [0.0], -1], [-1, -1, -1, [3.2389481721e-07]], [-1, -1, -1, [0.0]], [-1, -1, -1, -1]]
```

Podemos observar que llegamos a una solución optima. GEKKO nos muestra, además, algunos datos de interés entre los que se encuentran la cantidad de iteraciones que se realizaron para llegar a la solución, en este caso 7. También notamos que muestra los detalles de cada iteración, donde podemos ver, por ejemplo, la evaluacioón de la función objetivo en cada una. Esta se evaluó 8 veces hasta obtener la solución final, al igual que su gradiente. Obteniendo un costo mínimo de 64 aproximadamente. El tiempo de solución fue de 0.28 segundos.

Además de la información que GEKKO nos brinda, imprimimos en consola los resultados de las variables. La primera lista corresponde a la producción de cada nodo, mientras que la segunda corresponde con los flujos asociados a cada línea.

2

VI. Generación de la demanda

Según se plantea en el problema, la demanda se genera atendiendo a una distribución seleccionada por el usuario. Se deben escoger entre las siguientes distribuciones:

- uniforme
- exponencial
- gamma
- normal
- binaria
- geométrica
- por escenarios

En el caso de la última se debe poner la demanda y la probabilidad con la que esta aparecerá. La implementación de cada una de las distribuciones anteriores se encuentra en `utils/random_variables.py`. Todas las distribuciones que describen la demanda fueron implementadas tal cual se muestra en el capítulo 5 de [1].

References

[1] Ross 2010 A First Course in Probability 8th Ed