The Modular Model
Manual

Laboratorio de Optimización, Control y Mercados
Pontificia Universidad Católica de Chile

Manual preparado por
Felipe Verástegui

August 14, 2018

1 Introducción

Este documento contiene información sobre la estructura y componentes de The Modular Model, diseñado para el estudio de SSCC. Los principales objetivos de estos códigos son:

- Permitir la construcción y solución de un problema de pre-despacho y/o despacho conjunto, considerando distintos productos de reserva.
- Tener una estructura lo suficientemente modular para poder correr casos de estudio considerando o sacando dichos productos de reserva cambiando pocos parámetros.

A continuación se presentan las características específicas desarrolladas para cumplir con estos objetivos.

2 Programación de Modelo de Optimización

2.1 Pyomo

The Modular Model usa el paquete Pyomo para la construcción de un modelo de optimización. En particular, se usa la estructura de *AbstractModel* propia del paquete en cuestión.

Por otro lado, también se crea un objeto del tipo *DataPortal* propia de Pyomo también. Este objeto se hace para cargar los números y se le indica que son esos numeros en relación a un modelo.

Estos dos procesos paralelos se juntan en una línea de código, donde en base a los componentes específicados y datos cargados, se generan todas las variables, sets, parámetros, restricciones y funcion objetivo específicas para la instancia concreta del problema. Esta instancia es la que luego pasará a ser resuelta.

2.2 Reglas del codigo

Se siguen estas reglas a la hora de escribir el codigo relativo al modelo:

- **Parámetros**: Los nombres de los parámetros se definen todo junto en minúscula (ej: *startupcost, zonedemand, zonereservedemand*)
- **Variables**: Los nombres de las variables se definen usando todo junto en mayúsculas y minúsculas (ej: *GenCommit, GenStartUp, GenBasicReserve, PowerFlow, LoadShedding*)
- **Sets**: Los nombres de los sets se definen todo en mayúscula, usando guión abajo para los espacios (ej: *LOADZONES, TIMEPOINT, GEN, VARIABLE_GENS, GENS_AT_ZONE*)
- **Reglas**: Los nombres de las reglas (funciones iterables en pyomo para construir componentes) están definidas todas en minúscula, usando guión abajo para los espacios (ej: *dc_power_flow_rule, gen_ramp_up_rule*)
- **Restricciones, listas dinámicas y expresiones**: Los nombres de los demás componentes del modelo están definidas usando mayúsculas y minúsculas, usando guión abajo para los espacios (ej: *Zone_Power_Plus, Gen_Min_Start_Up*)

2.2.1 Solver

Por el momento, el modelo utiliza el solver comercial Gurobi. Esto puede ser cambiado en la programación misma del modelo (ver *main.py*).

3 Estructura Principal

The Modular Model se compone de tres bloques principales:

- *main.py*
- *core_module.py*
- modules (*unit_commitment.py, basic_reserve.py, basic_economic_dispatch.py*)

El proposito de la estructura es poder añadir y quitar componentes al modelo o al problema a resolver simplemente especificando que módulos deben ser considerados para la instancia a construir.

3.1 Main.py

A continuacion se describe el archivo *main.py* y su estructura a grandes rasgos.

3.1.1 Descripción de main.py

El archivo *main.py* se encarga de servir como punto de encuentro de todos los módulos y maneja los objetos tipo *AbstractModel* y *DataPortal* a medida que se va construyendo. Por ahora, también en este archivo se encuentra el código que explícitamente ordena al modelo a buscar la solución para el problema creado.

3.1.2 Estructura de main.py

El código en *main.py* se estructura de la siguiente manera:

- Se inicia el proceso creando los objetos tipo *AbstractModel* y *DataPortal*.
- Se llama a los modulos respectivos al problema, para que aumenten el *AbstractModel* con componentes del modelo tales como variables y restricciones.
- Se llama a los modulos respectivos al problema para que carguen datos de entrada al *DataPortal*.
- Se crea una instancia especifica combinando *AbstractModel* y *DataPortal*.
- Se resuelve dicha instancia con los parametros del solver definidos y se llama a cada modulo para que exporte sus resultados.

3.1.3 Llamados de main.py

El código en *main.py* hace todos los llamados, que son a:

- *core_module.py*
- Módulos individuales

3.2 Core_module.py

A continuacion se describe el archivo *core_module.py* y su estructura a grandes rasgos.

3.2.1 Descripción de core_module.py

El archivo de código *core_module.py* tiene dos funciones principales. En primer lugar, es llamado al principio para aumentar el *AbstractModel* con listas dinámicas y componentes claves, que son básicos para los problemas de operación a estudiar. Estos són:

- Set de buses (*LOADZONES*)
- Set de tiempo (*TIMEPOINT*)
- Costos para la función objetivo (*Costs*)
- Inyecciones y retiros de energía (*Zone_Power_Plus,Zone_Power_Minus*)

La segunda función principal es que una vez que han sido cargados todos los módulos (junto con sus costos a la lista dinámica *Costs*, y sus inyecciones o retiros de energía a las listas dinámicas de *Zone_Power_Plus,Zone_Power_Minus*), es construir la restriccion de balance de energía y la función objetivo en base a los componentes de las listas dinámicas.

Este archivo es necesario ya que de esta manera todos los módulos pueden referenciar a estas listas sin depender de lo que hacen los demás módulos o los componentes que construyen (por ahora).

3.2.2 Estructura de core_module.py

El código en *core_module.py* se estructura en base a dos funciones principales:

- build_model(*AbstractModel*): Esta función aumenta el objeto tipo *AbstractModel* agregandole componentes clave para los problemas operacionales a estudiar.
- complete_model(*AbstractModel*): Esta función aumenta por última vez el objeto tipo *AbstractModel*, constituyendo la función objetivo y la restricción de balance de energía.

3.2.3 Llamados de core_module.py

El codigo en *core_module.py* no hace llamados a otros bloques. Dado que se construye en base a funciones, es llamado por otros bloques (*main.py*).

3.3 Módulos

A continuacion se describen a grandes rasgos los modulos de forma generica y su estructura.

3.3.1 Descripción de los módulos

Los módulos individuales estan programados para definir las caracteristicas de cada problema de optimizacion que se desee resolver. En base a una defínción del usuario se cargan los módulos deseados y esto determina los componentes del problema a resolver.

Los módulos existentes a la fecha son:

- *unit_commitment.py*
- *basic_reserve.py*
- *basic_economic_dispatch.py*

3.3.2 Estructura de los módulos

Los módulos individuales se programan en base a funciones. Cada módulo debe tener las tres siguientes funciones principales:

- build_model(*AbstractModel*): Esta función aumenta el objeto tipo *AbstractModel* agregandole los componentes propios del módulo.
- load_data(*AbstractModel*, *DataPortal*): Esta función carga los datos necesarios para utilizar el módulo, aumentando el objeto tipo *DataPortal*.
- export_results(*SolvedInstance*): Esta función exporta en forma de archivos los resultados del problema de optimización.

3.3.3 Llamados de los módulos

El codigo en los módulos no hace llamados a otros bloques. Dado que se construyen en base a funciones, son llamados por otros bloques (*main.py*).

3.4 Flujo del proceso

En base a los bloques comentados anteriormente, el flujo del proceso es el siguiente:

- Se ejecuta el archivo *main.py*. Dicho archivo crea los objetos *AbstractModel* y *DataPortal*.
- Se llama al archivo *core_module.py*, que se encarga de aumentar el *AbstractModel* con componentes claves.
- Se lee el archivo *modules.txt* para analizar la estructura que se le desea dar al problema de optimización. Esto genera la lista de **módulos**.
- Para cada elemento en la lista de **módulos**, se le solicita que aumente el *AbstractModel*.

- Para cada elemento en la lista de **módulos**, se le solicita que aumente el *DataPortal*.
- Se crea una instancia combinando *AbstractModel* y *DataPortal* y se resuelve usando el solver.
- Para cada elemento en la lista de **módulos**, se le solicita que exporte los resultados generados.

4 Formato de Inputs

Se describen a continuación los archivos necesarios como input para cada módulo.

4.1 Unit Commitment

Los inputs para *unit_commitment.py* deben estar en una carpeta llamada *uc_inputs* y contener los siguientes archivos:

- *gen.csv*
- *line.csv*
- *load_zones.csv*
- *timepoints.csv*
- *zone_demand.csv*
- *variable_capacity_factors.csv* (opcional)
- *variable_water_cost.csv* (opcional)

4.2 Basic Reserve

Los inputs para *basic_reserve.py* deben estar en una carpeta llamada *uc_inputs* y contener los siguientes archivos:

- *zone_reserve_demand.csv* (opcional)

4.3 Basic Economic Dispatch

Los inputs para *basic_economic_dispatch.py* deben estar en una carpeta llamada *ed_inputs* y contener los siguientes archivos:

- *gen.csv*
- *line.csv*
- *load_zones.csv*
- *timepoints.csv*
- *zone_demand.csv*

- *gen_commit.tab* *
- *gen_start_up.tab* *
- *gen_shut_down.tab* *
- *variable_capacity_factors.csv* (opcional)
- *variable_water_cost.csv* (opcional)

*: Cabe señalar que si se corre un Unit Commitment antes del Economic Dispatch, este automáticamente traspasará estos archivos (que son resultados de dicho problema) a la carpeta *ed_inputs*.

5 Formato de Outputs

5.1 Unit Commitment

Los outputs para *unit_commitment.py* se generarán en una carpeta llamada *uc_outputs* y contendrá los siguientes archivos:

- *gen_commit.tab* *
- *gen_start_up.tab* *
- *gen_shut_down.tab* *
- *gen_pg.tab*
- *load_shedding.tab*
- *over_gen.tab*
- *power_flow.tab*
- *theta.tab*

*: Cabe señalar que estos outputs serán también puestos en una carpeta llamada *ed_inputs* en caso de que se quisiera correr un Economic Dispatch con los resultados señalados.

5.2 Basic Reserve

Los outputs para *basic_reserve.py* se generarán en una carpeta llamada *uc_outputs* y contendrá los siguientes archivos:

- *gen_basic_reserve.tab*

5.3 Basic Economic Dispatch

Los outputs para *basic_economic_dispatch.py* se generarán en una carpeta llamada *ed_outputs* y contendrá los siguientes archivos:

- *gen_pg.tab*
- *load_shedding.tab*
- *over_gen.tab*
- *power_flow.tab*
- *theta.tab*

6 Tutorial para uso y desarrollo

En esta sección se describe el repositorio donde se aloja The Modular Model, y se dan indicaciones para el desarrollo o el uso del mismo.

6.1 Git

Todo el código y desarrollos relativos a The Modular Model están en el Git del proyecto (https://github.com/sacordov/UC-OCM)) En dicho repositorio, encontrarán tres carpetas que se describen a continuación.

6.1.1 stable

En la carpeta *stable* se aloja la última versión funcional y estable de The Modular Model. Estos archivos serán usados para correr los casos en la medida en que sea necesario. La idea es no modificar estos archivos durante el desarrollo de nuevos módulos, y si se hace un cambio, solo incluirlo cuando este funcionando en perfecta sintonía con los demas bloques del programa.

6.1.2 development

En la carpeta *development* se aloja la versión en desarrollo de The Modular Model, la cual no necesariamente será funcional al ser consultada. Aquí también habrán módulos que se estén probando o archivos de código para usos prácticos (visualización de resultados, debbugging).

6.1.3 examples

A la fecha hay dos casos de estudio, contenidos en carpetas que se describen a continuación.

- *3 Bus Toy*: Este caso es un caso de juguete con tres zonas (Norte, Centro, Sur). Se pueden probar todas las funcionalidades del modelo (generadores con costo variable de agua, generadores variables, requerimientos básicos de reserva).
- *135 Chilean Case* : Este caso es un caso para probar la escalabilidad del modelo, se hizo una versión del SIC en base a datos de un estudio de la CNE, tiene 135 barras y muchos generadores. Varios parámetros faltantes fueron inventados y no se consideran generadores variables ni hidro.

6.2 Instrucciones de uso

Por ahora, para usar The Modular Model, se deben seguir los siguientes pasos:

- Crear una carpeta donde se alojarán los inputs y outputs del problema.
- Copiar en dicha carpeta todos los archivos presentes en la carpeta *stable*, que corresponde a la versión estable de The Modular Model
- Dentro de la carpeta, crear las carpetas *uc_inputs* y *ed_inputs* según sea necesario. En dichas carpetas dejar los archivos de entrada para el caso de estudio.
- Modificar el archivo *modules.txt*, especificando los módulos a utilizar en la construcción del problema. *E*jecutar el archivo *main.py*.

6.3 Manual

El manual está hecho en LaTeX. https://www.sharelatex.com/9975932792vnbkdrnrmpcn