

# Introducción a la programación matemática con



## Agenda

	Pág.
I    ¿Por qué Pyomo?	3
II   Programación matemática	6
III   Fundamentos de Python	9
IV   Fundamentos de Pyomo	10
V    Casos de estudio	12
VI   Recursos de aprendizaje	47

# I ¿Por qué Pyomo?

## Python



Fácil de usar



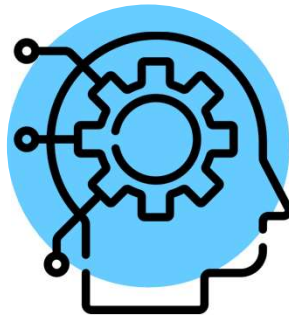
Eficiencia



Multiplataforma



Variedad de  
paquetes y  
librerías



Uso en ML y Big  
Data



Soporte y  
documentación

# I ¿Por qué Pyomo?

## Pyomo

**1** Gratuito y bien documentado

**3** Comunicación con los principales solvers

**2** Fácil instalación  
`pip install pyomo`

**4** Adaptabilidad y múltiples extensiones



IDAES/**idaes-pse**

The IDAES Process Systems Engineering Framework



Pyomo/**mpi-sppy**

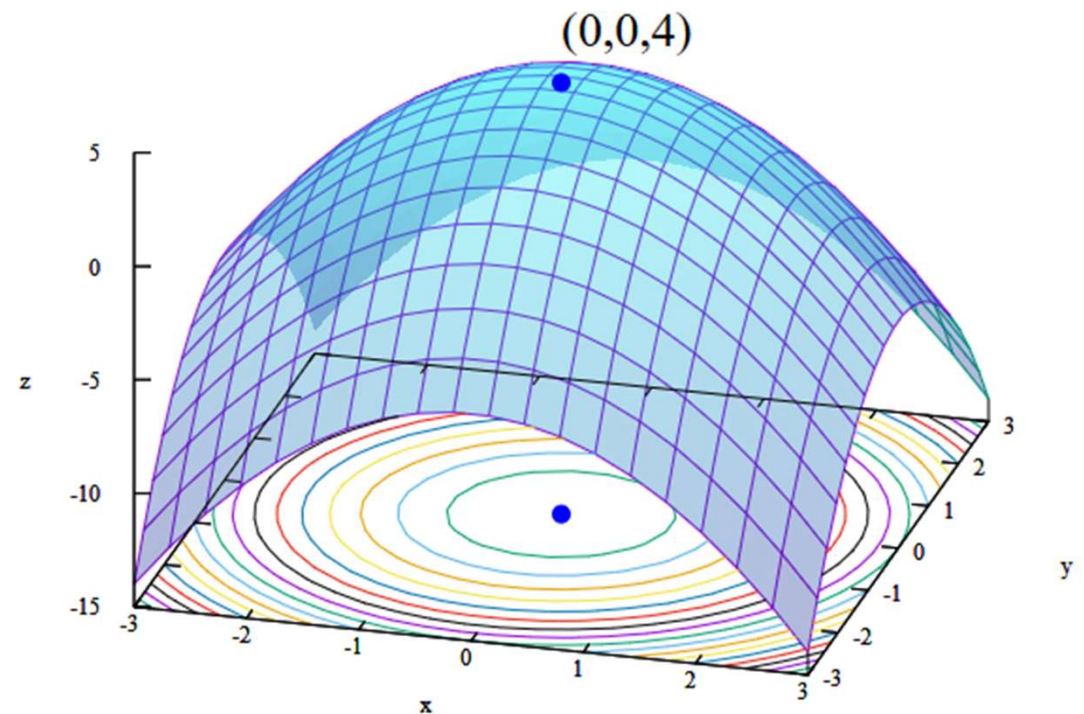
MPI-based Stochastic Programming in PYthon



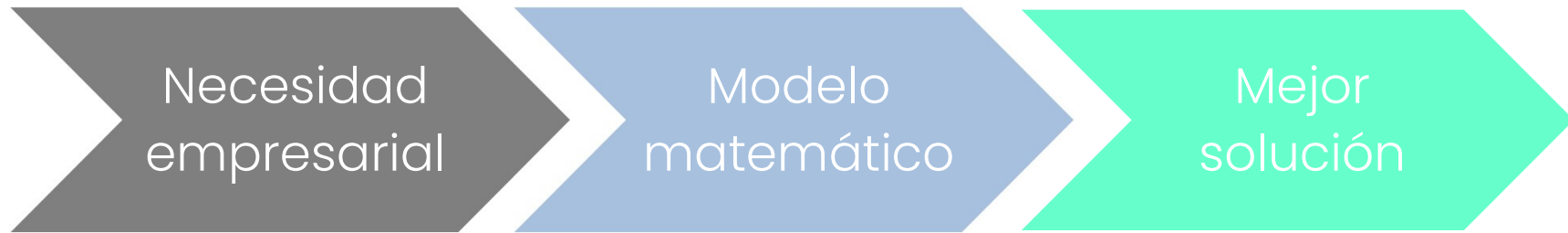
## II Programación matemática

### ¿Por qué es importante la optimización?

- ¿Cuál es el programa de producción óptimo para maximizar la utilización de los equipos?
- ¿Cuál es la mezcla óptima cuando se trabaja con ingredientes naturales?
- ¿Cuál es el horario óptimo para planificar la plantilla?
- ¿Cuál es el mejor medio de transporte para entregar los pedidos?
- ....



## II Programación matemática



$$\begin{array}{ll} \min & f(x) \quad \leftarrow \text{Función objetivo} \\ s.a., & h(x) = 0 \quad \leftarrow \text{Restricciones de igualdad} \\ & g(x) \leq 0 \quad \leftarrow \text{Restricciones de desigualdad} \\ & x^L \leq x \leq x^U \quad \leftarrow \text{Límites de las variables} \end{array}$$

### Clasificación

- Restricciones y función objetivo: Lineales / No lineales
- Variables: Contínuas / Discretas / Mixtas

## II Programación matemática

### Programación lineal (LP)

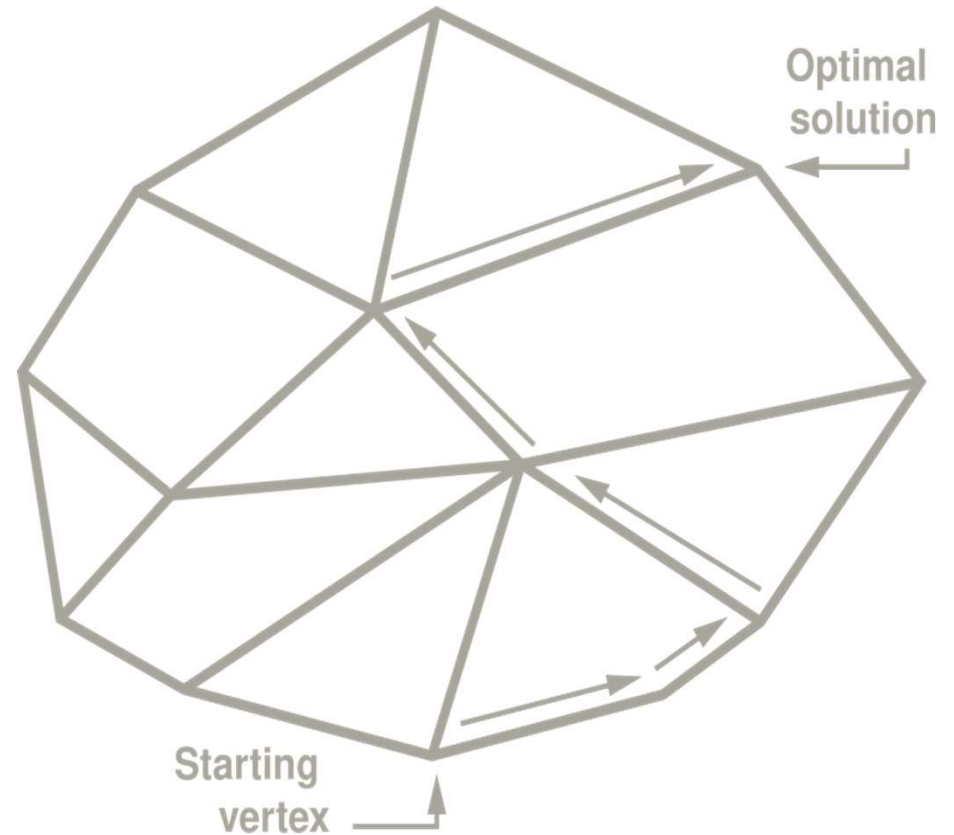
$$\begin{array}{ll}\min & c^T x \\ \text{s.a.}, & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{R}^n\end{array}$$



Función objetivo y  
restricciones lineales

### Ejemplo

Problema de transporte



### Resolución

Algoritmo Simplex

## II Programación matemática

### Programación no lineal (NLP)

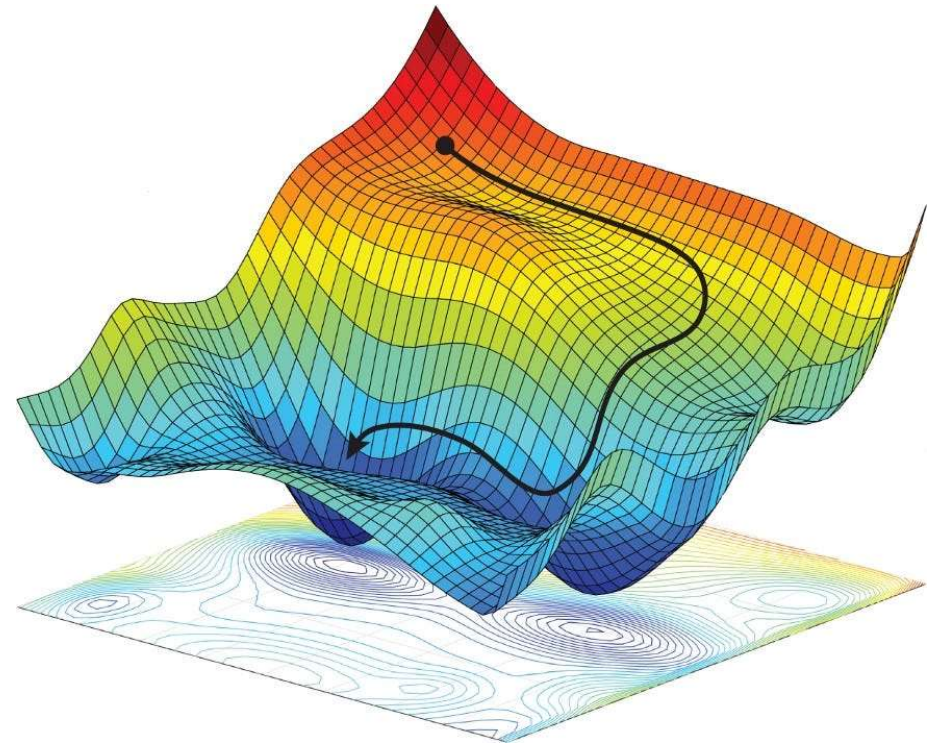
$$\begin{aligned} \min \quad & f(x) \\ \text{s.a.}, \quad & h(x) = 0 \\ & g(x) \leq 0 \\ & x^L \leq x \leq x^U \end{aligned}$$



Función objetivo y/o restricciones  
lineales y no lineales

### Ejemplo

Diseño RCTA



### Resolución

Condiciones KKT



## II Programación matemática

Programación lineal entera mixta (MILP)

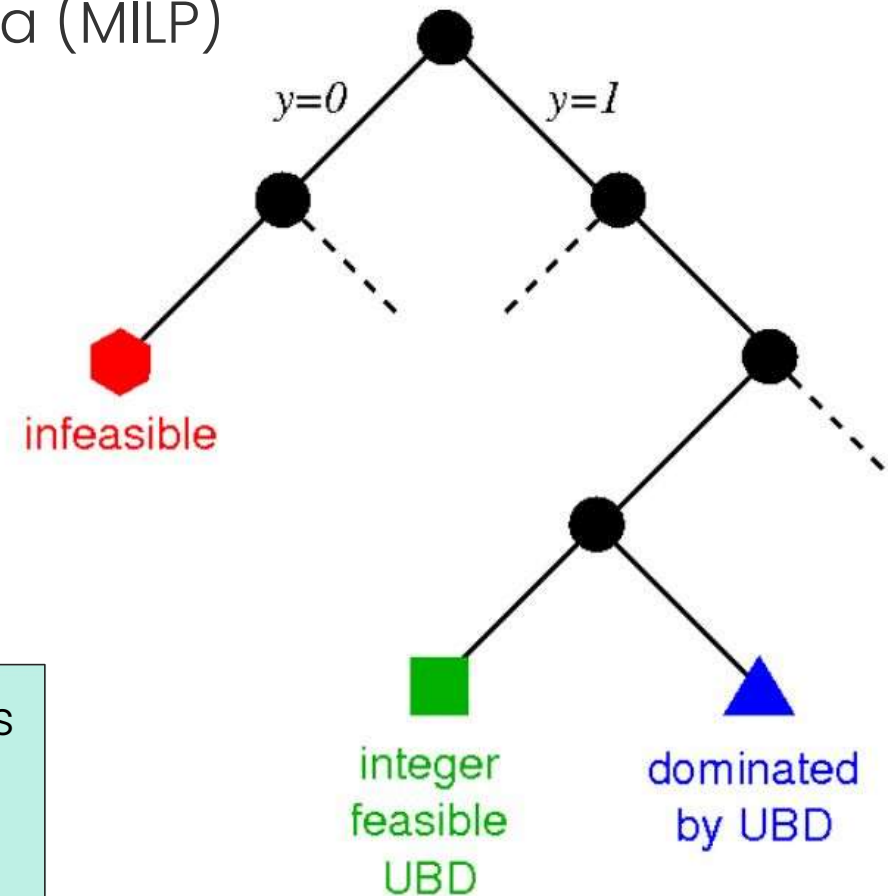
$$\begin{aligned} \min \quad & c^T x + d^T y \\ \text{s.t.}, \quad & Ax + By \leq b \\ & x \geq 0, \quad x \in \mathbb{R}^n \\ & y \in \{0,1\}^m \end{aligned}$$



Función objetivo y restricciones lineales  
+ Decisiones lógicas incorporando  
variables binarias

### Ejemplo

Selección equipos



### Resolución

Branch and Bound

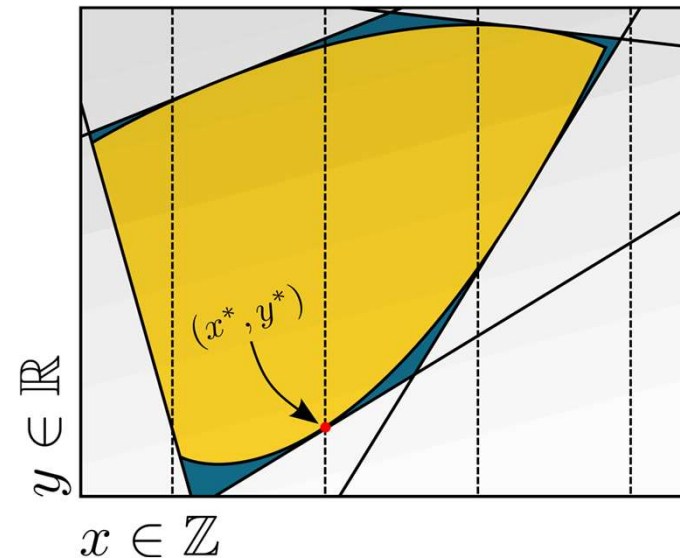
# I Programación matemática

## Programación no lineal entera mixta (MINLP)

$$\begin{aligned} \min \quad & f(x) + d^T y \\ \text{s.a.}, \quad & h(x) + By = 0 \\ & g(x) + Dy \leq 0 \\ & x \in \mathbb{R}^n \\ & y \in \{0,1\}^m \end{aligned}$$



Función objetivo y/o  
restricciones lineales y no  
lineales  
+ Decisiones lógicas  
incorporando variables binarias



### Ejemplo

Síntesis diagramas flujo

### Resolución

Outer Approximation

## II Programación matemática

SOLVER	LP	MILP	NLP	MINLP
ALPHAECP				X
ANTIGONE			X	X
BARON	X	X	X	X
CONOPT4	X		X	
CPLEX	X	X		
DICOPT				X
GUROBI	X	X	X	X
GLPK	X	X		
IPOPT	X		X	
SCIP		X	X	X
CBC		X		X

## II Programación matemática

### Programación disyuntiva generalizada (GDP)

$$\begin{array}{ll} \min & z = f(x) \\ \text{s.a.}, & h(x) = 0 \\ & g(x) \leq 0 \end{array} \left. \vphantom{\begin{array}{l} \min \\ \text{s.a.}, \end{array}} \right\} \text{Función objetivo y restricciones globales}$$
$$\begin{array}{ll} \bigvee_{i \in D_k} \left[ \begin{array}{c} Y_{k,j} \\ r_{k,j}(x) \leq 0 \end{array} \right] & k \in K \\ \bigvee_{i \in \overline{D}_k} Y_{k,j} & k \in K \end{array} \left. \vphantom{\begin{array}{l} \bigvee_{i \in D_k} \\ \bigvee_{i \in \overline{D}_k} \end{array}} \right\} \begin{array}{l} \text{Disyunciones} \\ \bullet \text{ Restricciones } r(x) \text{ solo se aplican} \\ \text{si } Y_{k,j} = \text{True} \end{array}$$
$$\begin{array}{l} \Omega(Y) = \text{True} \\ x^L \leq x \leq x^U \\ x \in \mathbb{R}^n \\ Y \in \{\text{True}, \text{False}\} \quad k \in K, i \in D_k \end{array} \leftarrow \text{Proposiciones lógicas}$$

## II Programación matemática

### Programación disyuntiva generalizada (GDP)

- Los modelos de GDP suelen reformularse como MILP/MINLP

#### GDP

$$\begin{aligned} \min : z &= f(x) \\ \text{s.t.} \quad g(x) &\leq 0 \\ h(x) &= 0 \end{aligned}$$

$$\bigvee_{i \in D_k} \left[ \begin{array}{c} Y_{k,i} \\ r_{i,k}(x) \leq 0 \end{array} \right] \quad k \in K$$

$$\bigvee_{i \in D_k} Y_{k,i} \quad k \in K$$

$$\begin{aligned} \Omega(Y) &= \text{True} \\ x^{lo} &\leq x \leq x^{up}, \quad x \in \mathbb{R}^n \\ Y &\in \{\text{True}, \text{False}\} \quad k \in K, i \in D_k \end{aligned}$$

#### Big M

$$\begin{aligned} \min : z &= f(x) \\ \text{s.t.} \quad g(x) &\leq 0 \\ h(x) &= 0 \end{aligned}$$

$$r_{ki}(x) \leq M^{ki} (1 - y_{ki}) \quad k \in K, i \in D_k$$

$$\sum_{i \in D_k} y_{ki} = 1 \quad k \in K$$

$$\begin{aligned} Hx &\geq h \\ x^{lo} &\leq x \leq x^{up}, \quad x \in \mathbb{R}^n \\ y_{ki} &\in \{0, 1\} \quad k \in K, i \in D_k \end{aligned}$$

#### Envolvente convexa

$$\begin{aligned} \min : z &= f(x) \\ \text{s.t.} \quad g(x) &\leq 0 \\ h(x) &= 0 \end{aligned}$$

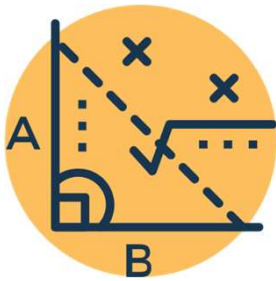
$$\begin{aligned} x &= \sum_{i \in D_k} v^{ki} \quad k \in K \\ y_{ki} r_{ki} \left( v^{ki} / y_{ki} \right) &\leq 0 \quad k \in K, i \in D_k \\ x^{lo} y_{ki} &\leq v^{ki} \leq x^{up} y_{ki} \quad k \in K, i \in D_k \\ \sum_{i \in D_k} y_{ki} &= 1 \quad k \in K \end{aligned}$$

$$\begin{aligned} Hx &\geq h \\ x &\in \mathbb{R}^n \\ y_{ki} &\in \{0, 1\} \quad k \in K, i \in D_k \end{aligned}$$

## II Programación matemática

### Lenguajes de modelado algebraico (AML)

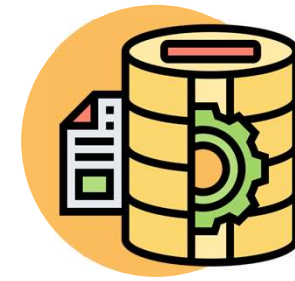
#### Objetivos



Forma natural de describir modelos matemáticos



Interfaz con numerosos solucionadores



Separar el modelado y las declaraciones de datos.

#### Ejemplos

- Lenguajes de modelado específicos
  - **GAMS**, AMPL, AIMMS
- Paquetes de AML para lenguajes de programación genéricos
  - FlopCPP (C++), OptimJ (Java), JuMP (Julia), PuLP (Python), **Pyomo** (Python)

### III Fundamentos básicos de Python

1. Creación de un entorno de trabajo e instalación de paquetes
2. Tipos de datos
3. Manipulación de datos
4. Definición de funciones



<https://github.com/cursos-COnCEPT/curso-Pyomo>

## IV Fundamentos de Pyomo

### Mezcla de cerveza

- Determinar proporción en la que se mezclan  $C$  componentes para satisfacer la demanda y especificaciones
- Minimizar costes de producción

	Coste (€/m <sup>3</sup> )	APV (%)
Cerveza A	0.32	4.5
Cerveza B	0.25	3.7
Agua	0.05	0.0



$$\begin{aligned} \min \quad & \sum_{c \in C} x_c P_c \\ \text{s.t.}, \quad & \sum_{c \in C} x_c = V \\ & A^* = \frac{\sum_{c \in C} x_c A_c}{\sum_{c \in C} x_c} \end{aligned}$$



## IV Fundamentos de Pyomo

### Mezcla de cerveza

- Modelo
- Sets
- Variables
- Parámetros
- Objetivo
- Restricciones

#### Nomenclatura

- $\mathcal{C}$  Set de componentes
- $x_c$  Volumen de componente  $c$  (m<sup>3</sup>)
- $P_c$  Coste del componente  $c$  (€/m<sup>3</sup>)
- $A_c$  Graduación del componente  $c$
- $V$  Demanda del cliente (m<sup>3</sup>)
- $A^*$  Graduación deseada

	Coste (€/m <sup>3</sup> )	APV (%)
Cerveza A	0.32	4.5
Cerveza B	0.25	3.7
Agua	0.05	0.0

$$\begin{aligned} \min \quad & \sum_{c \in \mathcal{C}} x_c P_c \\ \text{s.a.}, \quad & \sum_{c \in \mathcal{C}} x_c = V \\ & A^* = \frac{\sum_{c \in \mathcal{C}} x_c A_c}{\sum_{c \in \mathcal{C}} x_c} \end{aligned}$$

## IV Fundamentos de Pyomo

### Mezcla de cerveza

#### 1. Importar librerías que se van a utilizar

#### 2. Crear modelo

Opción 1

```
import pyomo.environ  
model = pyomo.environ.ConcreteModel()
```

Indicar a Pyomo qué librerías se van a utilizar

Opción 2

```
from pyomo.environ import *  
model = ConcreteModel()
```

Crear una instancia de `ConcreteModel`

- Los datos se deben especificar a medida que se construye el modelo

Opción 3

```
import pyomo.environ as pyo  
model = pyo.ConcreteModel()
```

Variable local que contendrá toda la información del modelo que vamos a crear

## IV Fundamentos de Pyomo

### Mezcla de cerveza

1. Importar librerías que se van a utilizar
2. Crear modelo
3. Definir los sets (si hay)

```
m.C= pyo.Set( initialize = ['A','B','W'] , doc = "Componentes" )
```

**IMPORTANTE:**

Distinción entre MAYÚSCULAS y minúsculas

Indicar los elementos del set por medio de un iterable (por ejemplo, una lista)

Descripción (opcional)

## IV Fundamentos de Pyomo

### Mezcla de cerveza

1. Importar librerías que se van a utilizar
2. Crear modelo
3. Definir los sets (si hay)
4. Definir las variables

```
m.x= pyo.Var(m.C, within=pyo.NonNegativeReals, doc = "Volumen m3")
```

Sets de los cuales  
depende la variable

Dominio de la  
variable (opcional)  
'within' y 'domain'  
pueden usarse  
indistintamente

Dominios predefinidos:

- NonNegativeReals
- Binary
- Integers

Descripción  
(opcional)

Definición alternativa:

```
m.x= pyo.Var(m.C , bounds=(0, None))
```

## IV Fundamentos de Pyomo

### Mezcla de cerveza

1. Importar librerías que se van a utilizar
2. Crear modelo
3. Definir los sets (si hay)
4. Definir las variables
5. Definir los parámetros

```
m.P = pyo.Param(m.C,
                initialize = {'A': 0.32, 'B': 0.25, 'C': 0.05},
                mutable= True ,
                doc = "Coste €/m3")
```

Sets de los cuales  
depende el  
parámetro

Python dict con  
los valores  
correspondientes

Descripción  
(opcional)

Indica que puede que decidas  
cambiar el valor del parámetro  
en algún momento


## IV Fundamentos de Pyomo

### Mezcla de cerveza

## 6. Definir la función objetivo

```
m.obj = pyo.Objective( expr= sum(m.x[i]*m.P[i] for i in m.C),  
                        sense=pyo.maximize)
```

‘expr’ es una función



Dirección de la optimización



## 7. Definir las restricciones

```
m.con_demand = pyo.Constraint(expr = vol == sum(m.x[c] for c in m.C))
```

## 8. Seleccionar solver y resolver el modelo

```
opt = pyo.SolverFactory('glpk')  
results= opt.solve(m)
```

## IV Fundamentos de Pyomo

Mezcla de cerveza

**6. Definir la función objetivo**

**7. Definir las restricciones**

**8. Seleccionar solver y resolver el modelo**

**9. Mostrar los resultados**

Todo el modelo

```
m.pprint()
```

Valor de la función objetivo

```
m.obj()
```

Valor que toman las variables

```
m.x.pprint()
```

Status del solver

```
print(results.solver.status)
```

```
print(results.solver.termination_condition)
```

# I Fundamentos de Pyomo

## Mezcla de cerveza

```
import pyomo.environ as pyo

m = pyo.ConcreteModel()

m.C = pyo.Set( initialize = ['A','B','C'], doc = 'Components')

m.x= pyo.Var(m.C, within=pyo.NonNegativeReals, doc = 'Volumen m3')

m.P = pyo.Param(m.C, initialize = {'A': 0.32, 'B': 0.25, 'C': 0.05},
                doc = 'Coste €/m3')
m.apv = pyo.Param( m.C, initialize = {'A': 4.5, 'B': 3.7, 'C': 0.},
                doc = 'Graduación')
m.V = pyo.Param(initialize = 100, doc = 'Demanda m3')
m.apv_spec = pyo.Param(initialize = 4., doc = 'Graduación deseada')

m.obj = pyo.Objective(expr = sum(m.x[c]*m.P[c] for c in m.C), sense=pyo.minimize)

m.con_demand = pyo.Constraint(expr = m.V == sum(m.x[c] for c in m.C))
m.con_spec = pyo.Constraint(
    expr = 0 == sum(m.x[c]*(m.apv[c] - m.apv_spec) for c in m.C))

opt = pyo.SolverFactory('glpk')
results= opt.solve(m)
```



## I Fundamentos de Pyomo

### ConcreteModel

Data first, then model

- 1 paso
  - Todos los datos se deben especificar **antes** de que Pyomo procese el modelo
  - Los objetos se construyen a medida que se declaran
- Más sencillo de escribir
- Similar a la filosofía de GAMS

### AbstractModel

Model first, then data

- 2 pasos
  - PASO 1:** se crea un “patrón”
    - Ejemplo: se declara una variable  $x$  que depende de un set  $I$ , sin indicar aún los componentes de ese set.
  - PASO 2:** se **cargan** los **datos** conocidos para crear una instancia **ConcreteModel**
- Facilita la reutilización del modelo
- Similar a la filosofía de AMPL

## I Fundamentos de Pyomo

### Mezcla de cerveza como AbstractModel

## PASO 1: Definir modelo (sets, variables, parámetros...)

```
import pyomo.environ as pyo

m = pyo.AbstractModel()

m.C = pyo.Set( initialize = ['A','B','C'], doc = 'Components')

m.x= pyo.Var(m.C, within=pyo.NonNegativeReals, doc = 'Volumen m3')

m.P = pyo.Param(m.C, initialize = {'A': 0.32, 'B': 0.25, 'C': 0.05},
                doc = 'Coste €/m3')
m.apv = pyo.Param( m.C, initialize = {'A': 4.5, 'B': 3.7, 'C': 0.},
                doc = 'Graduación')
m.V = pyo.Param(initialize = 100, doc = 'Demanda m3')
m.apv_spec = pyo.Param(initialize = 4., doc = 'Graduación deseada')

m.obj = pyo.Objective(expr = sum(m.x[c]*m.P[c] for c in m.C),
sense=pyo.minimize)

m.con_demand = pyo.Constraint(expr = m.V == sum(m.x[c] for c in m.C))
m.con_spec = pyo.Constraint(
    expr = 0 == sum(m.x[c]*(m.apv[c] - m.apv_spec) for c in m.C))
```

# I Fundamentos de Pyomo

Mezcla de cerveza como `AbstractModel`

## PASO 2: Cargar datos

- Para cargar los datos se utiliza el método `create_instance`

`m.create_instance()` o bien `m.create_instance( data )`

### Opcional

Si no se indica, tomará los valores que se hayan indicado como `'initialize=...'`.

Si no, sobrescribe los valores por defecto.

- Pueden utilizarse diferentes formatos para especificar los datos
  - Python `dict`
  - Archivo DAT
  - Importar desde `.JSON`, `.CSV` ...

# I Fundamentos de Pyomo

## Mezcla de cerveza

### Python dict

```
data = { None: {  
    'C': ['A', 'B', 'W'],  
    'vol': {None: 100},  
    'grd': {None: 4},  
    'P': {'A': 0.32,  
          'B': 0.25;  
          'W': 0.05},  
    'apv': {'A': 0.045 ,  
            'B': 0.037,  
            'W': 0.0 }},  
}
```

### Archivo DAT

```
set C := A B W ;  
  
param vol := 100 ;  
param grd := 4 ;  
  
param cost apv :=  
A      0.32  0.045  
B      0.25  0.037  
W      0.05  0.0  
;
```

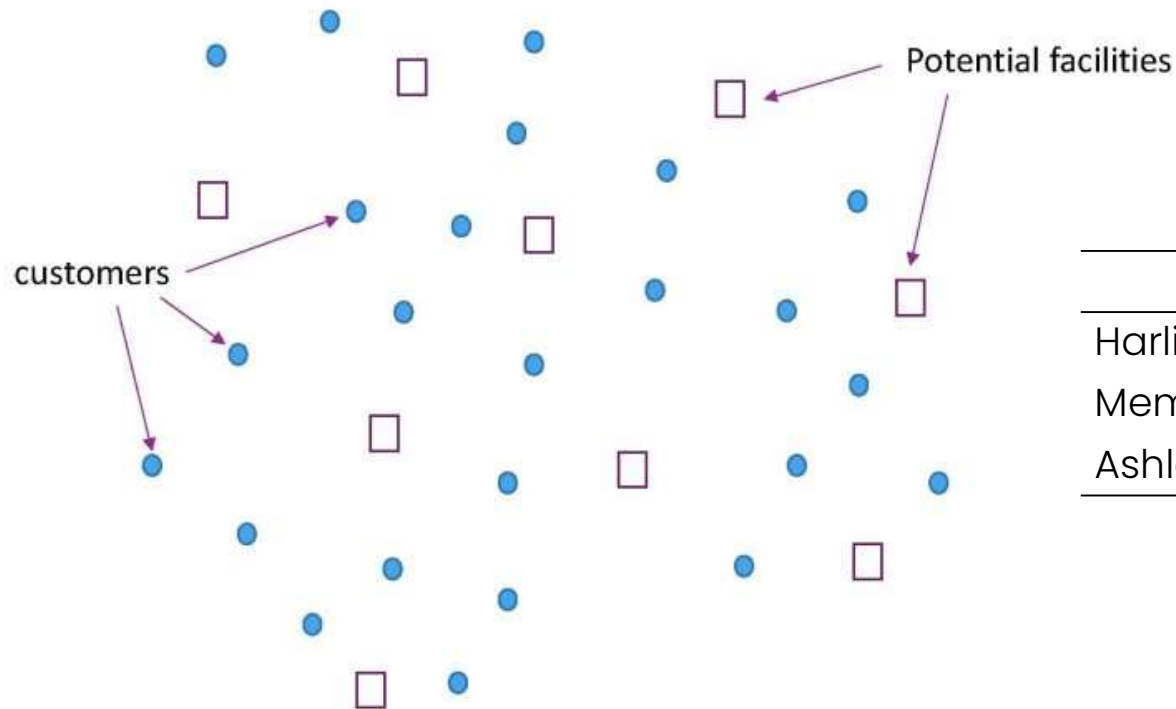
Nombre del archivo  
data.dat

```
instance = m.create_instance(data) # si se usa un dict  
  
instance = m.create_instance(data.dat) # si se utiliza un archivo .dat  
  
opt = pyo.SolverFactory('glpk')  
results= opt.solve(instance)
```

## V Ejemplo programación lineal

### Warehouse location

- Determinar dónde construir  $P$  almacenes entre  $W$  posibles ubicaciones
- Minimizar costes de distribución a todos los consumidores  $C$



	Costes de distribución (UM)			
	NYC	LA	Chicago	Houston
Harlingen	1956	1606	1410	330
Memphis	1095	1792	531	667
Ashland	485	2322	324	1236

## V Ejemplo programación lineal

### Warehouse location

- Determinar dónde construir  $P$  almacenes entre  $W$  posibles ubicaciones
- Minimizar costes de distribución a todos los consumidores  $C$

$$\begin{aligned} \min \quad & \sum_{w \in W} \sum_{c \in C} d_{wc} x_{wc} \\ \text{s.a.}, \quad & \sum_{w \in W} x_{wc} = 1 \quad \forall c \in C \\ & x_{wc} \leq y_w \quad \forall w \in W, c \in C \\ & \sum_{w \in W} y_w = P \\ & 0 \leq x_{wc} \leq 1 \quad \forall w \in W, c \in C \\ & y_w \in \{0, 1\} \quad \forall w \in W \end{aligned}$$

#### Nomenclatura

$d_{wc}$  Costes de distribución desde el almacén  $w$  al cliente  $c$

$P$  Número de almacenes que se pretende construir

$x_{wc}$  fracción de la demanda del cliente  $c$  servida desde el almacén  $w$

$y_w$  1 si se construye un almacén en la ubicación  $w$

## V Ejemplo programación lineal

¿Diferencia?

$$\min \quad \text{cost} = \sum_{c \in C} x_c P_c$$

$$s.a., \quad \sum_{c \in C} x_c = V$$

$$A^* = \frac{\sum_{c \in C} x_c A_c}{\sum_{c \in C} x_c}$$

### Mezcla cerveza

Restricciones escalares (una sola ecuación)

$$\min \quad \sum_{w \in W} \sum_{c \in C} d_{wc} x_{wc}$$

$$s.a., \quad \sum_{w \in W} x_{wc} = 1 \quad \forall c \in C$$

$$x_{wc} \leq y_w \quad \forall w \in W, c \in C$$

$$\sum_{w \in W} y_w = P$$

$$0 \leq x_{wc} \leq 1 \quad \forall w \in W, c \in C$$

$$y_w \in \{0, 1\} \quad \forall w \in W$$

### Warehouse location

Múltiples ecuaciones  
definidas de forma indexada

## V Ejemplo programación lineal

### Warehouse location

```
model = pyo.AbstractModel(name="(WL)")

d = {'Harlingen':{'NYC': 1956, 'LA': 1606, 'Chicago': 1410 , 'Houston': 330},
     'Memphis': {'NYC': 1096, 'LA': 1792, 'Chicago': 531, 'Houston': 567},
     'Ashland': {'NYC': 485, 'LA': 2322, 'Chicago': 324, 'Houston': 1236 }
     }

data = {None: {'W': ['Harlingen', 'Memphis', 'Ashland'],
                'C': ['NYC', 'LA', 'Chicago', 'Houston'],
                'P': {None: 2} ,
                'cost': {(w,c):d[w][c] for w in d for c in d[w]}
                }}

model.C = pyo.Set(doc = 'Cities')
model.W = pyo.Set(doc = 'Warehouses')

model.x = pyo.Var(model.W, model.C, bounds=(0,1))
model.y = pyo.Var(model.W, within=pyo.Boolean)

model.cost = pyo.Param(model.W, model.C, doc='Coste')
model.P = pyo.Param(initialize=2, doc='Number of warehouses')
```



## V Ejemplo programación lineal

### Warehouse location

```
def obj_rule(m):  
    return sum(m.cost[w,c]*m.x[w,c] for w in m.W for c in m.C)  
model.obj = pyo.Objective(rule=obj_rule)  
  
def one_per_cust_rule(m, c):  
    return sum(m.x[w,c] for w in m.W) == 1  
model.one_per_cust = pyo.Constraint(model.C, rule=one_per_cust_rule)  
  
def num_warehouses_rule(m):  
    return sum(m.y[w] for w in m.W) <= m.P  
model.num_warehouses = pyo.Constraint(rule=num_warehouses_rule)  
  
def warehouse_active_rule(m, w, c):  
    return m.x[w,c] <= m.y[w]  
model.warehouse_active = pyo.Constraint(model.W, model.C, rule=warehouse_active_rule)  
  
instance = model.create_instance(data)  
pyo.SolverFactory('glpk').solve(instance)  
  
instance.y.pprint()  
instance.x.pprint()
```

## V Ejemplo programación disyuntiva I

### Problema de mezclado

- Determinar proporción en la que se mezclan  $C$  componentes para satisfacer las especificaciones del producto final
  - Cantidad máxima Vit A = 0.4
  - Cantidad mínima Vit B = 0.2
  - A y B no se pueden mezclar
- Minimizar costes de producción

Comp.	Coste (UM)	Vit A	Vit B
A	2.0	0.5	0.2
B	2.0	0.4	0.1
C	5.0	0.3	0.3

$$\begin{aligned} \min \quad & \sum_{c \in C} x_c P_c \\ \text{s.a.}, \quad & \sum_{c \in C} x_c = 1 \\ & \sum_{c \in C} x_c \text{Vit}A_c \leq 0.4 \\ & \sum_{c \in C} x_c \text{Vit}B_c \geq 0.2 \\ & \left[ \begin{array}{c} Y \\ x_B = 0 \end{array} \right] \underline{\vee} \left[ \begin{array}{c} \neg Y \\ x_A = 0 \end{array} \right] \\ & 0 \leq x_c \leq 1 \quad \forall c \in C \end{aligned}$$

## V Ejemplo programación disyuntiva I

Problema de mezclado

### Programación disyuntiva

$$\begin{aligned} \min \quad & \sum_{c \in C} x_c P_c \\ \text{s.a.}, \quad & \sum_{c \in C} x_c = 1 \\ & \sum_{c \in C} x_c \text{Vit}A_c \leq 0.4 \\ & \sum_{c \in C} x_c \text{Vit}B_c \geq 0.2 \\ & \left[ \begin{array}{c} Y \\ x_B = 0 \end{array} \right] \vee \left[ \begin{array}{c} \neg Y \\ x_A = 0 \end{array} \right] \\ & 0 \leq x_c \leq 1 \quad \forall c \in C \end{aligned}$$

### Reformulación Big-M

$$\begin{aligned} \min \quad & \sum_{c \in C} x_c P_c \\ \text{s.a.}, \quad & \sum_{c \in C} x_c = 1 \\ & \sum_{c \in C} x_c \text{Vit}A_c \leq 0.4 \\ & \sum_{c \in C} x_c \text{Vit}B_c \geq 0.2 \\ & x_A \leq My \\ & x_B \leq M(1 - y) \\ & 0 \leq x_c \leq 1 \quad \forall c \in C \\ & y \in \{0, 1\} \end{aligned}$$

## V Ejemplo programación disyuntiva I

### Problema de mezclado

## Manipulación de sets

Un set se puede definir a partir de otro (subset) de forma programática utilizando filtros.

1. Definir el conjunto principal

```
model.C = pyo.Set(initialize = ['A','B','C'], doc = 'Componentes')
```

2. Crear un filtro

```
def incompatible_filter(model, i, j):  
    return i == 'A' and j == 'B'
```

← Función que devuelve True si el elemento pertenece al subset

3. Definir el subconjunto

```
model.C_incompat = pyo.Set(initialize=model.C*model.C,  
                           filter=incompatible_filter,  
                           doc = 'Pares incompatibles')
```

← Operador \*: producto cartesiano

## V Ejemplo programación disyuntiva I

### Problema de mezclado: Restricciones globales

```
import pyomo.environ as pyo
model = pyo.AbstractModel()

model.C = pyo.Set(initialize = ['A','B','C'], doc = 'Componentes')
def incompatible_filter(model, i, j):
    return i == 'A' and j == 'B'
model.C_incompat = pyo.Set(initialize=model.C*model.C,
                           filter=incompatible_filter, doc = 'Pares incompatibles')

model.x = pyo.Var(model.C, bounds = (0.,1.))
model.y = pyo.Var(model.C_incompat, domain=pyo.Boolean)

model.P = pyo.Param(model.C, initialize = d['Coste'], doc='Coste UM')
model.VitA = pyo.Param(model.C, initialize = d['VitA'], doc='Contenido de VitA')
model.VitB = pyo.Param(model.C, initialize = d['VitB'], doc='Contenido de VitB')
model.VitA_UB = pyo.Param(initialize = 0.4, doc='Cantidad máxima de VitA')
model.VitB_LB = pyo.Param(initialize = 0.2, doc='Cantidad mínima de VitB')
model.BigM = pyo.Param(initialize = 10, doc = 'Coeficiente BigM')

def obj_rule(m):
    return sum(m.P[c]*m.x[c] for c in m.C)
model.obj = pyo.Objective(rule = obj_rule)

model.massfraction = pyo.Constraint(rule = lambda m: sum(m.x[c] for c in m.C) == 1)
model.comp_ub = pyo.Constraint(rule = lambda m: sum(m.VitA[c]*m.x[c] for c in m.C) <= m.VitA_UB )
model.comp_lb = pyo.Constraint(rule = lambda m: sum(m.VitB[c]*m.x[c] for c in m.C) >= m.VitB_LB )
```

## V Ejemplo programación disyuntiva I

Problema de mezclado: Reformulación Big M

### Reformulación Big M

```
model.ref_BigM = pyo.ConstraintList()
for pair in model.C_incompat:
    a, b = pair
    model.ref_BigM.add(model.x[a] <= model.BigM*model.y[pair])
    model.ref_BigM.add(model.x[b] <= model.BigM*(1-model.y[pair]))
```

## V Ejemplo programación disyuntiva I

Problema de mezclado: Disyunciones

$$\left[ \begin{array}{c} Y_1 \\ \exp(x_2) - 1 = x_1 \\ x_3 = x_4 = 0 \end{array} \right] \vee \left[ \begin{array}{c} Y_2 \\ \exp(2x_4) - 1 = x_3 \\ x_1 = x_2 = 0 \end{array} \right]$$

```
m.term1 = pyo.Disjunct()
```

Crear un elemento `Disjunct()`  
para cada bloque

1

```
m.term1.cons1 = pyo.Constraint(expr=exp(m.x[2]) - 1 == m.x[1])  
m.term1.cons2 = pyo.Constraint(expr= m.x[3] == 0)  
m.term1.cons3 = pyo.Constraint(expr= m.x[4] == 0)
```

Añadir  
restricciones

2

```
m.term2 = pyo.Disjunct()  
m.term2.cons1 = pyo.Constraint(expr=exp(2*m.x[4]) - 1 == m.x[3])  
m.term2.cons2 = pyo.Constraint(expr= m.x[2] == 0)  
m.term2.cons3 = pyo.Constraint(expr= m.x[1] == 0)
```

```
m.logicOR = pyo.Disjunction(expr = [m.term1, m.term2])
```

Combinar cada término  
en forma de lista

3



Por defecto, Pyomo aplica una disyunción exclusiva  $\vee$

## V Ejemplo programación disyuntiva I

Problema de mezclado: Reformulación Big M

### Reformulación Big M

```
model.ref_BigM = pyo.ConstraintList()
for pair in model.C_incompat:
    a, b = pair
    model.ref_BigM.add(model.x[a] <= model.BigM*model.y[pair])
    model.ref_BigM.add(model.x[b] <= model.BigM*(1-model.y[pair]))
```

### Disyunciones

```
# Importar funciones necesarias
from pyomo.gdp import Disjunct, Disjunction

# Término que se cumple si Y = True
model.term1 = Disjunct()
model.term1.consA = pyo.Constraint(expr= model.x['A'] == 0)
# Término que se cumple si Y = False
model.term2 = Disjunct()
model.term2.consB = pyo.Constraint(expr= model.x['B'] == 0)
# Añadir restricción en forma de disyunción
model.disyuncion = Disjunction(expr = [model.term1, model.term2])
```

Disjunct y Disjunction son comandos propios de la extensión de GDP para Pyomo



## V Ejemplo programación disyuntiva I

Problema de mezclado: Resolución

### Activar/Desactivar restricciones

```
model.cons_name.deactivate() // model.cons_name.activate()
```

- Las restricciones indexadas pueden desactivarse por completo o individualmente

```
model.cons_name[idx].deactivate() // model.cons_name[idx].activate()
```

- Las restricciones desactivadas **no se envían** al *solver* a la hora de resolver el modelo.

### Aplicar transformaciones

```
pyo.TransformationFactory(trf_name).apply_to(model)
```

- Transformaciones disponibles

Reformulación	trf_name
Big M (BM)	<code>gdp.bigm</code>
Envolvente convexa (HR)	<code>gdp.hull</code>
Híbrido BM/HR	<code>gdp.cuttingplane</code>

## V Ejemplo programación disyuntiva I

Problema de mezclado: Resolución

### Reformulación Big M (a mano)

```
# Desactivar restricciones
model.term1.deactivate()
model.term2.deactivate()
model.disyuncion.deactivate()

# Llamada al solver
opt = pyo.SolverFactory('cbc')
results = opt.solve(model)
```

### Extensión Pyomo GDP

```
# Activar/desactivar restricciones
model.ref_BigM.deactivate()

# Aplicar transformación
pyo.TransformationFactory('gdp.hull').apply_to(model)

# Llamada al solver
opt = pyo.SolverFactory('cbc')
results = opt.solve(model)
```



## V Ejemplo programación disyuntiva II

### Separación no total

$$\min \sum_{c \in C} Cost_c$$

$$s.a. \quad F_{0i} = S1_i + S2_i + S3_i + F_{c1,i} + F_{c2,i} + F_{c3,i} \quad \forall i \in I$$

$$F_{c4,i} = B_{c1,i} + B_{c2,i} \quad \forall i \in I$$

$$F_{c5,i} = D_{c2,i} + D_{c3,i} \quad \forall i \in I$$

$$ProdA_i = S1_i + D_{c1,i} + D_{c5,i} \quad \forall i \in I$$

$$ProdB_i = S2_i + D_{c4,i} + B_{c5,i} \quad \forall i \in I$$

$$ProdC_i = S3_i + B_{c3,i} + B_{c4,i} \quad \forall i \in I$$

$$\sum_{i \in I} SX_i F_{0i} = \sum_{i \in I} F_{0i} SX_i \quad \forall i \in I, SX = \{S1, S2, S3\}$$

$$\sum_{i \in I} F_{c,i} F_{0i} = \sum_{i \in I} F_{0i} F_{c,i} \quad \forall i \in I, c = \{c1, c2, c3\}$$

$$F_{c,i} = D_{c,i} + B_{c,i} \quad \forall c \in C, i \in I$$

$$D_{c,i} = F_{c,i} x_{c,i} \quad \forall c \in C, i \in I$$

$$ProdA_A \geq 0.85F_{0A} \quad ProdA_B \leq 0.1F_{0B} \quad ProdA_C \leq 0.1F_{0C}$$

$$ProdB_A \leq 0.1F_{0A} \quad ProdB_B \geq 0.8F_{0B} \quad ProdB_C \leq 0.1F_{0C}$$

$$ProdC_A \leq 0.05F_{0A} \quad ProdC_B \leq 0.1F_{0B} \quad ProdC_C \geq 0.8F_{0C}$$

Función objetivo

Restricciones  
globales

## V Ejemplo programación disyuntiva II

Separación no total

$$\left[ \begin{array}{l} Y_c \\ Cost_c = FC_c + FV_c \sum_{i \in I} D_{c,i} \\ F_{c,i} \leq F_{0i} \quad \forall i \in I \end{array} \right] \vee \left[ \begin{array}{l} \neg Y_c \\ Cost_c = 0 \\ F_{c,i} = 0 \quad \forall i \in I \end{array} \right] \quad \forall c \in C$$

$$Y_1 \vee Y_2 \vee Y_3$$

$$Y_1 \Rightarrow Y_4$$

$$Y_1 \Rightarrow \neg Y_5$$

$$Y_2 \Rightarrow Y_4$$

$$Y_2 \Rightarrow Y_5$$

$$Y_3 \Rightarrow Y_5$$

$$F_{ci}, D_{ci}, D_{c,i}, S1_i, S2_i, S3_i, ProdA_i, ProdB_i, Prodc_i, Cost_c \in \mathbb{R}$$

$$Y_{i,j} \in \{True, False\}$$

Disyunciones

Restricciones  
lógicas

## V Ejemplo programación disyuntiva II

Separación no total

$$\text{Cost}_c = FC_c y_c + VC_c \sum_{i \in I} D_{ci}$$

$$F_{ci} \leq F0_i y_c$$

$$y_1 + y_2 + y_3 = 1$$

$$y_1 - y_4 \leq 0$$

$$y_1 + y_5 \leq 1$$

$$y_2 - y_4 \leq 0$$

$$y_2 - y_5 \leq 0$$

$$y_3 - y_5 \leq 0$$

$$F_{ci}, D_{ci}, D_{c,i}, S1_i, S2_i, S3_i, ProdA_i, ProdB_i, ProDC_i, Cost_c \in \mathbb{R}$$

$$y_c \in \{0,1\}$$

Disyunciones  
(Envolvente  
convexa)

Restricciones  
lógicas

## VI Recursos de aprendizaje

- Pyomo documentation

<https://pyomo.readthedocs.io/en/stable/>

- ND Pyomo Cookbook

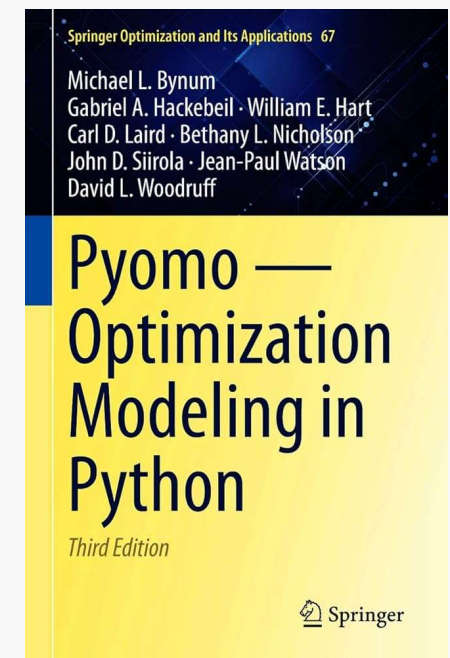
<https://jckantor.github.io/ND-Pyomo-Cookbook/README.html>

- Materiales Cacheme

<https://github.com/CACheME>

- Foros

- StackOverflow (“pyomo” tag)
- Pyomo Forum (pyomoforum@googlegroups.com)



# Introducción a la programación matemática con

