«بسمه‌تعالی»

آشنایی با پیاده‌سازی مدل‌های بهینه‌سازی در pyomo

Algebraic Modeling Language (AML) قابلیت بیان مدل‌های بهینه‌سازی با یک زبان برنامه‌نویسی سطح بالا را فراهم می‌کند که بسیار حائز اهمیت قرار دارد زیرا بدون آن، اگر بخواهیم یک مدل بهینه‌سازی را با یک سالور مشخص حل کنیم، باید آن مدل را در قالب فرمتی که برای آن سالور قابل قبول است، بیان داریم و به عنوان ورودی به سالور بدهیم در حالی که نوشتن مدل با فرمت سالور به خصوص برای مسائل بزرگ مقیاس، سخت و زمانبر است و ممکن است با خطای زیادی همراه باشد. به علاوه، سالورهای مختلف بعضاً فرمت‌های مختلفی را برای ورودی استفاده می‌کنند و در صورتی که بخواهیم عملکرد چند سالور مختلف را در حل مدلمان با یکدیگر مقایسه کنیم، لازم است آماده‌سازی ورودی هر سالور را به صورت جداگانه انجام دهیم. همچنین، اعتبارسنجی مدل (تضمین آن که مدلی که به سالور ارسال می‌شود دقیقاً همان مدلی است که طراحی شده) بدون یک زبان برنامه‌نویسی سطح بالا کاری سخت است.

AMLهای تجاری مانند AIMMS، AMPL و GAMS قابلیت بیان مدل‌های بهینه‌سازی با یک زبان برنامه‌نویسی سطح بالا را فراهم می‌کنند اما pyomo این قابلیت را از طریق زبان برنامه‌نویسی پایتون فراهم می‌کند.

pyomo قابلیت پیاده‌سازی انواع مختلف مدل‌های بهینه‌سازی را دارد از جمله:

Linear programming
Mixed integer linear programming
Nonlinear programming
Mixed integer nonlinear programming

pyomo مدل‌های بهینه‌سازی را حل نمی‌کند بلکه ورودی مناسب برای سالورها را ایجاد و فرآیند حل را به سالورها واگذار می‌نماید و سپس جوابی را که سالور بازمی‌گرداند گزارش می‌کند. امکان برقراری ارتباط با سالورهای مختلف در pyomo فراهم است البته لازم است این سالورها ابتدا نصب گردند و در صورت نیاز به لایسنس فعال‌سازی شوند. GLPK یک سالور متن‌باز است که در آن قابلیت حل مسائل LP و MILP وجود دارد و IPOPT نیز برای حل مسائل غیرخطی استفاده می‌شوند.

در منابع زیر اطلاعات جامعی درباره کتابخانه pyomo ارائه شده است:

Pyomo website: http://www.pyomo.org
Pyomo's open source software: https://github.com/Pyomo/pyomo
https://pyomo.readthedocs.io/en/stable/working_models.html

همچنین، کتاب زیر اطلاعات مفیدی را دربردارد.

Pyomo — Optimization Modeling in Python

مسأله بهینه‌سازی زیر را در نظر بگیرید:

$$\min z = 2x_1 + 10x_2$$
$$s.t.$$
$$x_1 + x_2 \leq 10$$
$$x_1 - 2x_2 \geq 4$$
$$x_1, x_2 \geq 0$$

دستورات زیر یک استفاده مقدماتی از کتابخانه pyomo را برای پیاده‌سازی مدل ساده فوق نشان می‌دهد:

```
import pyomo.environ as pyo
model=pyo.ConcreteModel()
model.x1=pyo.Var(domain=pyo.NonNegativeReals)
model.x2=pyo.Var(domain=pyo.NonNegativeReals)
model.obj=pyo.Objective(expr=2*model.x1+10*model.x2, sense=pyo.minimize)
model.const1=pyo.Constraint(expr=model.x1+model.x2<=10)
model.const2=pyo.Constraint(expr=model.x1-2*model.x2>=4)

solvername = 'glpk'
opt = pyo.SolverFactory(solvername)
result=opt.solve(model, 'glpk');
model.display()

print(pyo.value(model.x1), pyo.value(model.x2))
print(pyo.value(model.obj))
```

خروجی دستورات فوق به صورت زیر است:

```
Model unknown

  Variables:
    x1 : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :   4.0 :  None : False : False : NonNegativeReals
    x2 : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :   0.0 :  None : False : False : NonNegativeReals

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True :   8.0

  Constraints:
    const1 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  4.0 :  10.0
    const2 : Size=1
        Key  : Lower : Body : Upper
        None :   4.0 :  4.0 :  None
4.0 0.0
8.0
```

دستورات فوق را می‌توان به صورت زیر نیز نوشت که در آن برای تعریف ضابطه تابع هدف و قیود به جای expr از rule استفاده شده است:

```
import pyomo.environ as pyo
model=pyo.ConcreteModel()
model.x1=pyo.Var(domain=pyo.NonNegativeReals)
model.x2=pyo.Var(domain=pyo.NonNegativeReals)
def obj_rule(model):
    return 2*model.x1+10*model.x2

def const1_rule(model):
    return model.x1+model.x2<=10

def const2_rule(model):
    return model.x1-2*model.x2>=4


model.obj=pyo.Objective(rule=obj_rule, sense=pyo.minimize)
model.const1=pyo.Constraint(rule=const1_rule)
model.const2=pyo.Constraint(rule=const2_rule)

solvername = 'glpk'
opt = pyo.SolverFactory(solvername)
result=opt.solve(model, 'glpk');
model.display()

print(pyo.value(model.x1), pyo.value(model.x2))
print(pyo.value(model.obj))
```

دقت کنید که برای اجرای دستورات فوق لازم است پکیج‌ها نصب شده باشند:

```
conda install pyomo
conda install –c conda-forge glpk
```

در ادامه، می‌آموزیم که چگونه از قابلیت‌های پکیج pyomo به صورت پیشرفته و حرفه‌ای استفاده کنیم که به ما کمک می‌کند به جای فرم گسترده با فرم فشرده مدلها کار کنیم.

مطالعه این قسمت به بعد اختیاری است.

در pyomo مدل‌های بهینه‌سازی را به دو صورت می‌توان بیان کرد:

- ConcreteModel: در این حالت، مدل به ازای یک dataset مشخص نوشته می‌شود و امکان ساخت نمونه‌های مختلفی از مدل به طور همزمان به ازای چند dataset مختلف وجود ندارد.
- AbstractModel: در این حالت، امکان ایجاد نمونه‌های مختلفی از مدل به طور همزمان به ازای چند dataset فراهم است.

در اینجا شیوه کار با ConcreteModel را به طور کامل شرح می‌دهیم.

برای ایجاد یک ConcreteModel دستورات زیر را می‌نویسیم:

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

سپس، بخش‌های اصلی مدل (مجموعه اندیس، پارامتر، متغیر تصمیم، تابع هدف و قید) را از طریق کلاس‌های زیر تعریف می‌کنیم. در ادامه، هر کلاس را با ذکر جزئیات شرح می‌دهیم.

کلاس	نقش
Set	معرفی مجموعه اندیس
Var	معرفی متغیر تصمیم
Param	معرفی پارامتر
Objective	معرفی تابع هدف
Constraint	معرفی قید

کلاس Set

از Set برای بیان مجموعه اندیس در مدل‌های بهینه‌سازی استفاده می‌شود. مثلاً اگر بخواهیم مجموعه‌ای به نام iset را با اعضای {1,2,3} برای model تعریف کنیم، از دستور زیر استفاده می‌کنیم:

model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5])

آرگومان‌های مهم کلاس Set عبارتند از:

آرگومان	شرح	مقادیر قابل قبول
initialize	اعضای مجموعه	در قالب list شامل رشته‌ها یا اعداد یا tuple یا ... بیان می‌شود
ordered	بیان می‌کند که آیا ترتیب اولیه اعضای مجموعه حفظ شود یا خیر	True/False

برای دسترسی به اعضای مجموعه، بعد از نام مجموعه از data() استفاده می‌کنیم. به عنوان مثال:

دستور

```
for i in model.iset.data():
    print(i)
```

خروجی

```
1
2
3
4
5
```

یا به عنوان مثالی دیگر،

دستور

```
print(model.iset.data())
```

خروجی

```
(1, 2, 3, 4, 5)
```

تذکر: فرض کنید

model.A = pyo.Set(initialize=[3,2,1], ordered=True)

در این صورت، به خروجی دستورات زیر توجه کنید:

```
print(model.A.first()) # 3
print(model.A.last()) # 1
print(model.A.next(2)) # 1
print(model.A.prev(2)) # 3
print(model.A.nextw(1)) # 3
print(model.A.prevw(3)) # 1
```

```
print(model.A.ord(3)) # 1
print(model.A.ord(1)) # 3
print(model.A[1]) # 3
print(model.A[3]) # 1
```

nextw و prevw بیانگر wrap around هستند و ord ترتیب هر عضو را در یک مجموعه مرتب نشان می‌دهد.

این دستورات، در مدل‌هایی که دربردارنده عباراتی مانند $x_{i-1} + x_{i+1}$ هستند، قابل استفاده هستند.

کلاس Param

پارامترها در مقادیر سمت راست و نیز به عنوان ضرایب متغیرها در تابع هدف و قیود مدل ظاهر می‌شوند. مثلاً اگر بخواهیم پارامتری با نام a_i را با فرض $i \in iset$ و با مقادیر زیر تعریف کنیم ابتدا یک دیکشنری تعریف می‌کنیم که داده‌های این پارامتر را در بربگیرد.

i	1	2	3	4	5
a_i	10	20	30	40	50

```
data_a={1: 10, 2: 20, 3: 30, 4:40, 5: 50}
```

و سپس از دستور زیر استفاده می‌کنیم:

```
model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5])
model.a=pyo.Param(model.iset,initialize= data_a)
```

همان‌طور که در دستور فوق دیده می‌شود، اگر یک پارامتر وابسته به یک یا چند اندیس باشد، ابتدا مجموعه‌های متناظر با این اندیس‌ها را در آرگومان‌های ورودی Param بیان می‌کنیم و سپس سایر آرگومان‌ها را می‌نویسیم که برخی از آرگومان‌های مهم در جدول زیر آمده است:

آرگومان	شرح	مقادیر قابل قبول
initialize	مقدار پارامتر	اسکالر، دیکشنری یا rule function
default	مقادیر پیش‌فرض را تعیین می‌کند که اگر مقداری برای پارامتر تعیین نشد، استفاده شود	اسکالر، دیکشنری یا rule function
mutable	بیانگر آن است که آیا مقدار پارامتر می‌تواند بین دفعات مختلف فراخوانی سالور تغییر داده شود.	True/False

برای دسترسی مقادیر پارامتر، از pyo.value استفاده می‌کنیم. به عنوان مثال:

```
for i in model.iset:
    print(i, pyo.value(model.a[i]))
```

و خروجی به صورت زیر خواهد بود:

```
1 10
2 20
3 30
4 40
5 50
```

مثال‌های زیر روش‌های مختلف مقداردهی به پارامتر را نشان می‌دهند:

دستور

```
import pyomo.environ as pyo
model=pyo.ConcreteModel()
model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5])
model.jset=pyo.Set(initialize=[1, 2, 3])
model.a=pyo.Param(initialize=10)

data_b={1: 10, 2:20}
model.b=pyo.Param(model.iset, default=0, initialize=data_b)
data_c={(1,1): 10, (1,2): 20, (1,3): 30, (2,1):40, (2,2):50, (2,3): 60}
model.c=pyo.Param(model.iset, model.iset, default=0, initialize=data_c)

def d_init_rule(model, i, j):
    if i==1 or j==1:
        return i*j
    return i*j + model.b[j]

model.d = pyo.Param(model.iset, model.jset, initialize=d_init_rule)
for i in model.iset:
    for j in model.jset:
        print(i, j, pyo.value(model.d[i,j]))
```

خروجی

```
1 1 1
1 2 2
1 3 3
2 1 2
2 2 24
2 3 6
3 1 3
3 2 26
3 3 9
4 1 4
4 2 28
4 3 12
5 1 5
5 2 30
5 3 15
```

مثال زیر نشان می‌دهد که چگونه می‌توان این امکان را فراهم کرد که مقدار یک پارامتر در طول برنامه قابل تغییر باشد.

دستور	import pyomo.environ as pyo model=pyo.ConcreteModel() model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5]) data_b={1: 10, 2:20} model.b=pyo.Param(model.iset, default=0, initialize=data_b, mutable=True) model.b[3]=100 for i in model.iset: print(pyo.value(model.b[i]))
خروجی	10 20 100 0 0

کلاس Var

دستور کلی برای تعریف متغیر تصمیم فاقد اندیس w و متغیر y_i که $i \in iset$ و متغیر $x_{i,j}$ که $i \in iset$ و $j \in jset$ با فرض آن که هر سه نامنفی باشند، به صورت زیر است:

```
model.w=pyo.Var(domain=pyo.NonNegativeReals)
model.y=pyo.Var(model.iset, domain=pyo.NonNegativeReals)
model.p=pyo.Var(model.iset, model.jset, domain=pyo.NonNegativeReals)
```

همان‌طور که در دستور فوق دیده می‌شود، اگر یک پارامتر وابسته به یک یا چند اندیس باشد، ابتدا مجموعه‌های متناظر با این اندیس‌ها را در آرگومان‌های ورودی Var بیان می‌کنیم و سپس سایر آرگومان‌ها را می‌نویسیم که برخی از آرگومان‌های مهم در جدول زیر آمده است:

آرگومان	شرح	مقادیر قابل قبول
within یا domain	دامنه مقادیر متغیر	pyomo set python list مقادیر از قبل تعریف شده rule function
initialize	مقدار اولیه برای متغیر	اسکالر، دیکشنری یا rule function
bounds	کران‌های پایین و بالا برای متغیر	tuple دو مؤلفه‌ای یا rule function
mutable	بیانگر آن است که آیا مقدار پارامتر می‌تواند بین دفعات مختلف فراخوانی سالور تغییر داده شود.	True/False

مقادیر از قبل تعریف شده برای within و domain به شرح زیر است:

Reals	The set of floating point values
PositiveReals	The set of strictly positive floating point values
NonPositiveReals	The set of non-positive floating point values
NegativeReals	The set of strictly negative floating point values
NonNegativeReals	The set of non-negative floating point values
PercentFraction	The set of floating point values in the interval [0,1]
UnitInterval	The same as 'PercentFraction'
Integers	The set of integer values
PositiveIntegers	The set of positive integer values
NonPositiveIntegers	The set of non-positive integer values
NegativeIntegers	The set of negative integer values
NonNegativeIntegers	The set of non-negative integer values
Boolean	The set of boolean values, which can be represented as False/True, 0/1, 'False'/'True' and 'F'/'T'
Binary	The same as 'Boolean'

برای دسترسی مقادیر پارامتر، از pyo.value استفاده می‌کنیم. به عنوان مثال:

```
for i in model.iset:
    print(i, pyo.value(model.w[i]))
```

مثال زیر شیوه تعیین کران را هنگام تعریف متغیرها را نشان می‌دهد. در این مثال، برای متغیر x_i کران پایین 1 و کران بالای ۱۰ و برای متغیر w_i کران پایین l_i و کران بالای u_i در نظر گرفته شده است:

```
import pyomo.environ as pyo
model=pyo.ConcreteModel()
model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5])

model.x=pyo.Var(model.iset, domain=pyo.NonNegativeReals, bounds=(1, 10))

data_lower={1: 1, 2: 3, 3: 10, 4: 5, 5:5}
data_upper={1: 10, 2: 30, 3: 100, 4: 50, 5:50}
def bound_y_rule(model, i):
    return (data_lower[i], data_upper[i])

model.y=pyo.Var(model.iset, domain=pyo.NonNegativeReals, bounds=bound_y_rule)
```

مثال زیر نشان می‌دهد که چگونه می‌توان مقدار متغیر و کران‌های بالا و پایین آن را چاپ نمود:

```
for i in model.iset:
    print(pyo.value(model.y[i]))
    print(model.y[i].lb)
    print(model.y[i].ub)
```

با فرض آن که data_y یک دیکشنری باشد، می‌توان به همه مؤلفات متغیر y به صورت یک جا با دستور زیر مقداردهی کرد:

```
model.y.set_values(data_y)
```

کلاس Objective

با فرض آن که تابع هدف را obj بنامیم، برای تعریف آن از دستور model.obj=pyo.Objective() استفاده می‌شود که آرگومان‌های ورودی Objective به شرح زیر است:

آرگومان	شرح	مقادیر قابل قبول
expr	بیانگر ضابطه تابع هدف	pyomo expression
rule	بیانگر ضابطه تابع هدف	function rule
sense	نوع تابع هدف	minimize/maximize

با در نظر گرفتن ضابطه تابع هدف به صورت $\sum_{i \in iset} c_i x_i$، مثال زیر دو روش rule و expr را برای تعریف ضابطه تابع هدف نشان می‌دهد.

```
import pyomo.environ as pyo
model=pyo.ConcreteModel()
model.iset=pyo.Set(initialize=[1, 2, 3, 4, 5])

model.c=pyo.Param(model.iset, initialize={1: 10, 2:20, 3:30, 4:40, 5:50})
model.x=pyo.Var(model.iset, domain=pyo.Binary)

def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.iset)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.minimize)

model.obj2=pyo.Objective(expr=sum(model.c[i]*model.x[i] for i in model.iset), sense=pyo.
minimize)
```

کلاس Constraint

با فرض آن که قید مسأله را const1 بنامیم، برای تعریف آن از دستور model.const1=pyo.Constraint() استفاده می‌شود. اگر یک قید به ازای اندیس‌های مشخصی بیان شده باشد، ابتدا مجموعه‌های متناظر با این اندیس‌ها را در آرگومان‌های ورودی Constraint بیان می‌کنیم و سپس سایر آرگومان‌ها را می‌نویسیم که به شرح جدول زیر است:

آرگومان	شرح	مقادیر قابل قبول
expr	بیانگر ضابطه تابع قید	pyomo expression
rule	بیانگر ضابطه تابع قید	function rule

علامت قید به یکی از حالات ==، =< و >= بیان می‌گردد.

به عنوان مثال، برای بیان قید $\forall j \in jset \quad \sum_{i \in iset} y_i \leq p$، به صورت زیر عمل می‌کنیم:

```
def const1_rule(model):
    return sum(model.y[i] for i in model.iset) <= model.p
model.const1 = Constraint(rule=const1_rule)
```

برای بیان قید $\forall j \in jset \quad \sum_{i \in iset} x_{i,j} = 1$، به صورت زیر عمل می‌کنیم:

```
def const2_rule(model, j):
    return sum(model.x[i,j] for i in model.iset) == 1
model.const2 = Constraint(model.jset, rule=const2_rule)
```

برای بیان قید $\forall i \in iset, \forall j \in jset \quad x_{i,j} \leq y_i$، به صورت زیر عمل می‌کنیم:

```
def const3_rule(model, i, j):
    return model.x[i,j] <= model.y[i]
model.const3 = Constraint(iset, model.jset, rule=const3_rule)
```

در برخی از مدل‌های بهینه‌سازی، ممکن است یک قید لزوماً به ازای همه مقادیر یک اندیس تعریف نشود. در این صورت، در rule function از constraint skip استفاده می‌کنیم. دستورات زیر این موضوع را برای قید $\forall i \in iset, \forall j \in jset: i \neq j \quad x_{i,j} \leq y_i$ نشان می‌دهند.

```
Def const5_rule(model, i, j):
    if i==j:
        return pyo.Constraint.Skip
    else:
        return model.x[i,j] <= model.y[i]
model.const5 = pyo.Constraint(model.iset, model.jset, rule=const5_rule)
```

برای تعیین مقدار ضابطه قید به ازای جواب به دست آمده و نیز اختلاف آن با کران‌های بالا و پایین از body، lslack و uslack استفاده می‌شود. مثال زیر این موضوع را نشان می‌دهد:

```
for j in model.jset:
    print(value(model.const1[j].body))
    print(model.const1[j].lslack())
    print(model.const1[j].uslack())
```

دستور حل مدل

بعد از معرفی بخش‌های مختلف مدل، می‌توان آن را با سالورهای مختلف حل کرد.

سالور glpk یک سالور متن‌باز مختص مسائل LP و MILP است که برای استفاده از آن، ابتدا باید با دستور زیر آن را نصب کنیم:

```
conda install –c conda-forge glpk
```

و سپس، می‌توان از دستورات زیر برای حل مدل استفاده نمود:

```
solvername = ‘glpk’
opt = pyo.SolverFactory(solvername)
result=opt.solve(model, ‘glpk’, keepfiles=True, tee=True)
```

مثال: پیاده‌سازی مدل مسأله مکان‌یابی مراکز خدماتی در pyomo

مدل مسأله مکان‌یابی مراکز خدماتی به صورت زیر است:

$$\min \sum_{i\in\mathbb{I}}\sum_{j\in\mathbb{J}} c_{i,j}x_{i,j}$$

$$s.t.$$

$$\sum_{i\in\mathbb{I}} y_i \le p$$

$$\sum_{i\in\mathbb{I}} x_{i,j} = 1 \quad \forall j \in \mathbb{J}$$

$$x_{i,j} \le y_i \quad \forall i \in \mathbb{I}, \forall j \in \mathbb{J}$$

$$0 \le x_{i,j} \le 1 \quad \forall i \in \mathbb{I}, \forall j \in \mathbb{J}$$

$$y_i \in \{0,1\} \quad \forall i \in \mathbb{I}$$

فرض کنید داده‌های مسأله به صورت زیر در یک فایل اکسل در مسیر 'C:\\Users\\F\\Desktop\\FacilityLocation.xlsx' ذخیره شده است:

	A	B	C	D	E	F	G
1		C1	C2	C3	C4	C5	C6
2	F1	9	7	1	2	4	10
3	F2	6	10	9	5	6	6
4	F3	10	20	3	9	8	7

برای پیاده‌سازی مدل فوق، دستورات زیر را می‌نویسیم:

```
import pyomo.environ as pyo
import pandas as pd

df = pd.read_excel('C:\\Users\\F\\Desktop\\FacilityLocation.xlsx', header=0, index_col=0)
data_iset=list(df.index.map(str))
data_jset=list(df.columns.map(str))
data_c={(i,j):df.at[i,j] for i in data_iset for j in data_jset}

model =pyo.ConcreteModel()
model.iset=pyo.Set(initialize=data_iset)
model.jset=pyo.Set(initialize=data_jset)
model.c = pyo.Param(model.iset, model.jset, initialize=data_c)
model.p = pyo.Param(initialize=2, mutable=True)
model.x = pyo.Var(model.iset, model.jset, bounds=(0,1))
model.y = pyo.Var(model.iset, within=pyo.Binary)

def obj_rule(model):
    return sum(model.c[i,j]*model.x[i,j] for i in model.iset for j in model.jset)
model.obj = pyo.Objective(rule=obj_rule)
```

```
def const1_rule(model):
    return sum(model.y[i] for i in model.iset) <= model.p
model.const1 = pyo.Constraint(rule=const1_rule)

def const2_rule(model, j):
    return sum(model.x[i,j] for i in model.iset) == 1
model.const2 = pyo.Constraint(model.jset, rule=const2_rule)

def const3_rule(model, i, j):
    return model.x[i,j] <= model.y[i]
model.const3 = pyo.Constraint(model.iset, model.jset, rule=const3_rule)

solvername = 'glpk'
opt = pyo.SolverFactory(solvername)
result=opt.solve(model,'glpk', keepfiles=True, tee=True)
model.display()

y_result={(i): pyo.value(model.y[i]) for i in data_iset}
print(y_result)
```

چاپ وضعیت سالور

برای چاپ وضعیت سالور، از دستور زیر استفاده می‌کنیم:

```
print (str(result.solver))
```

که مثلاً منجر به خروجی زیر می‌شود:

```
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.03952527046203613
```