Trabalho_1_Análise_de_Eficiência_Produtiva

Trabalho 1

Fernando Söndahl Brito

DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

ANÁLISE DE EFICIÊNCIA PRODUTIVA

PROFª. LÍDIA

Ajuste iniciais

- Pacotes e bibliotecas
- Dados

Instalação do framework e solver:

[ ]:
```
!apt-get install -y -qq glpk-utils
!pip install -q pyomo
```

[2]:
```
import pandas as pd
import pyomo.environ as pyo
from pyomo.environ import *
from pyomo.opt import SolverFactory
import numpy as np

pd.options.mode.chained_assignment = None
```

Conexão com os dados do problema:

[4]:
```
dados = "https://github.com/fersondahl-uff/DEA/raw/main/Dados-Distribuidores.
↪xlsx"

dados_df = pd.read_excel(io=dados)

dados_df = dados_df.rename(columns={'Distribuidora': 'DMU' ,'INPUT 1': 'Input␣
↪1',  'INPUT 2': 'Input 2',  'OUTPUT': 'Output'})
```

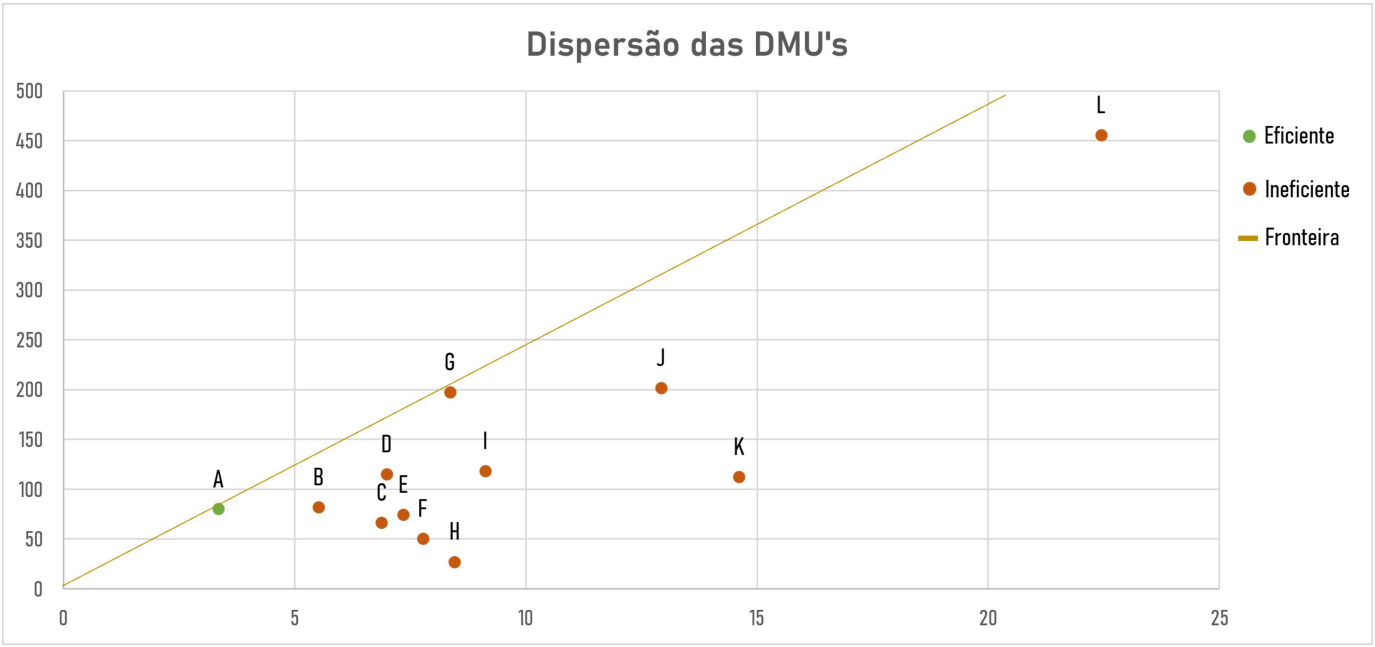
```
dados_df
```

[4]:

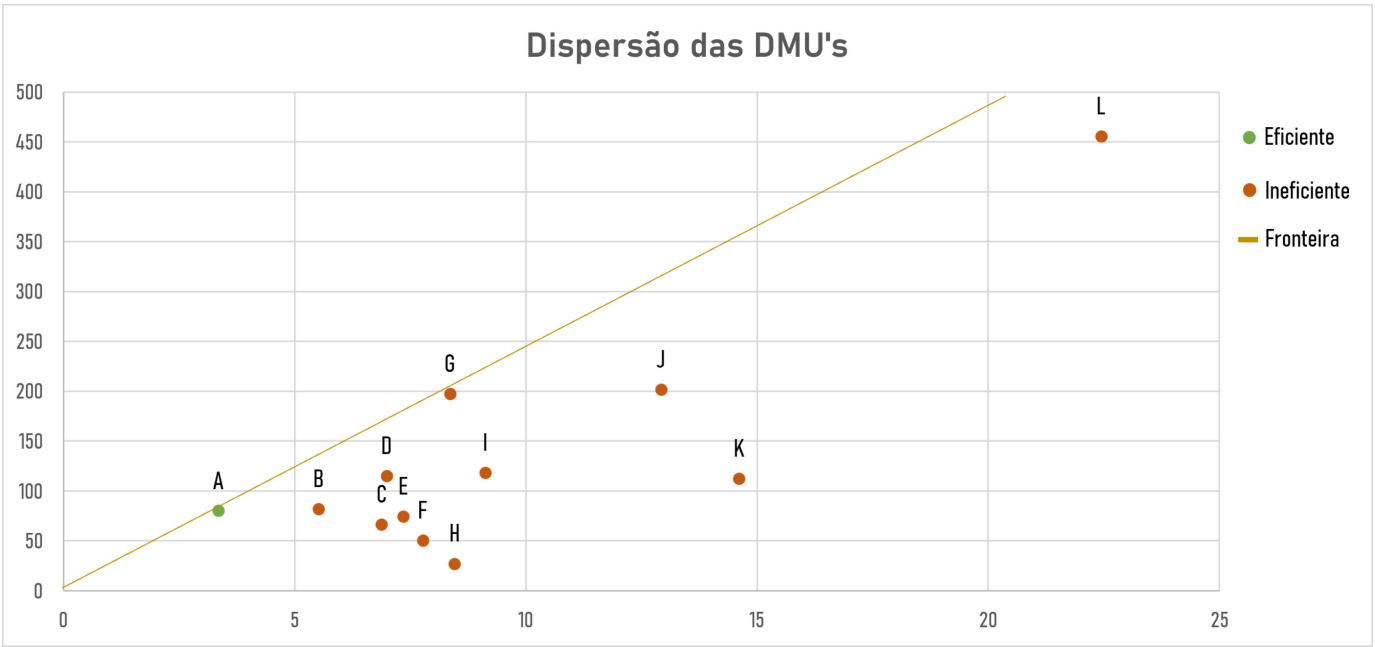
	DMU	Input 1	Input 2	Output
0	A	79.95	77.0	3.35
1	B	81.76	77.5	5.52
2	C	66.71	70.5	6.88
3	D	114.97	131.5	6.99
4	E	74.70	56.5	7.35
5	F	50.23	117.5	7.77
6	G	197.27	106.5	8.36
7	H	27.22	72.0	8.46
8	I	118.39	141.0	9.12
9	J	201.86	220.5	12.92
10	K	112.61	110.5	14.61
11	L	455.44	334.5	22.44

1.

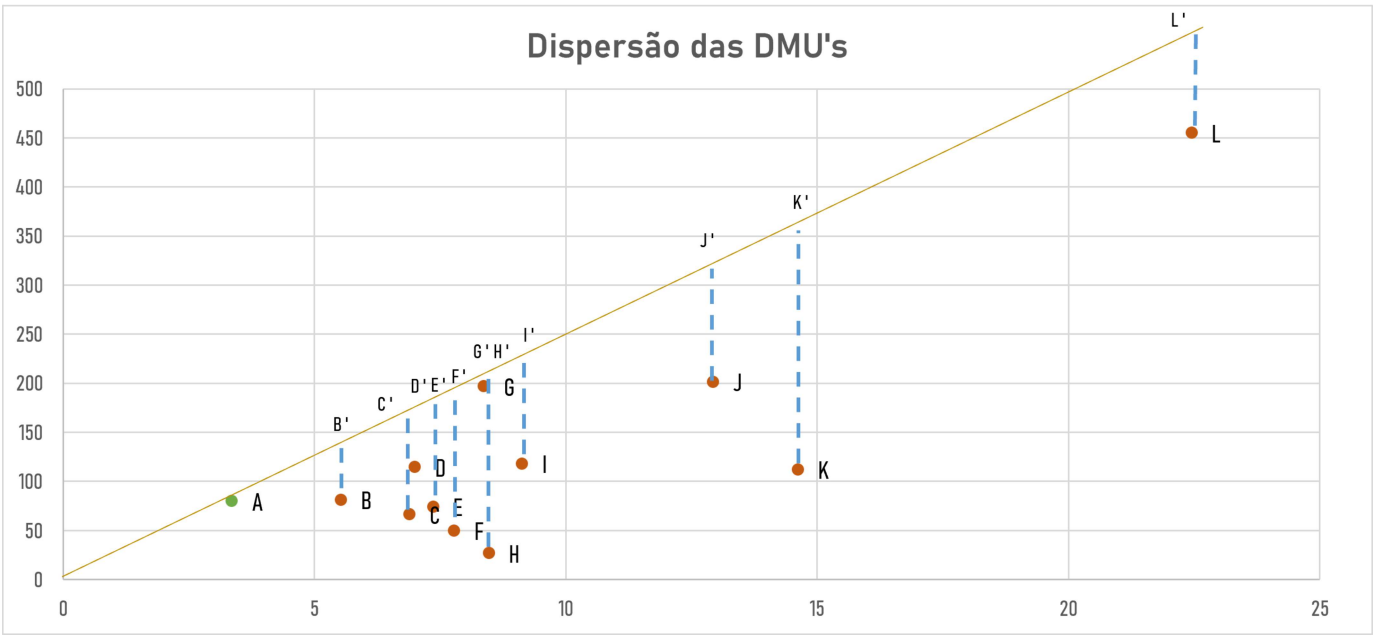
A.


Dispersão das DMU's
500
450
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
A
B
C
D
E
F
G
H
I
J
K
L
Eficiente
Ineficiente
Fronteira

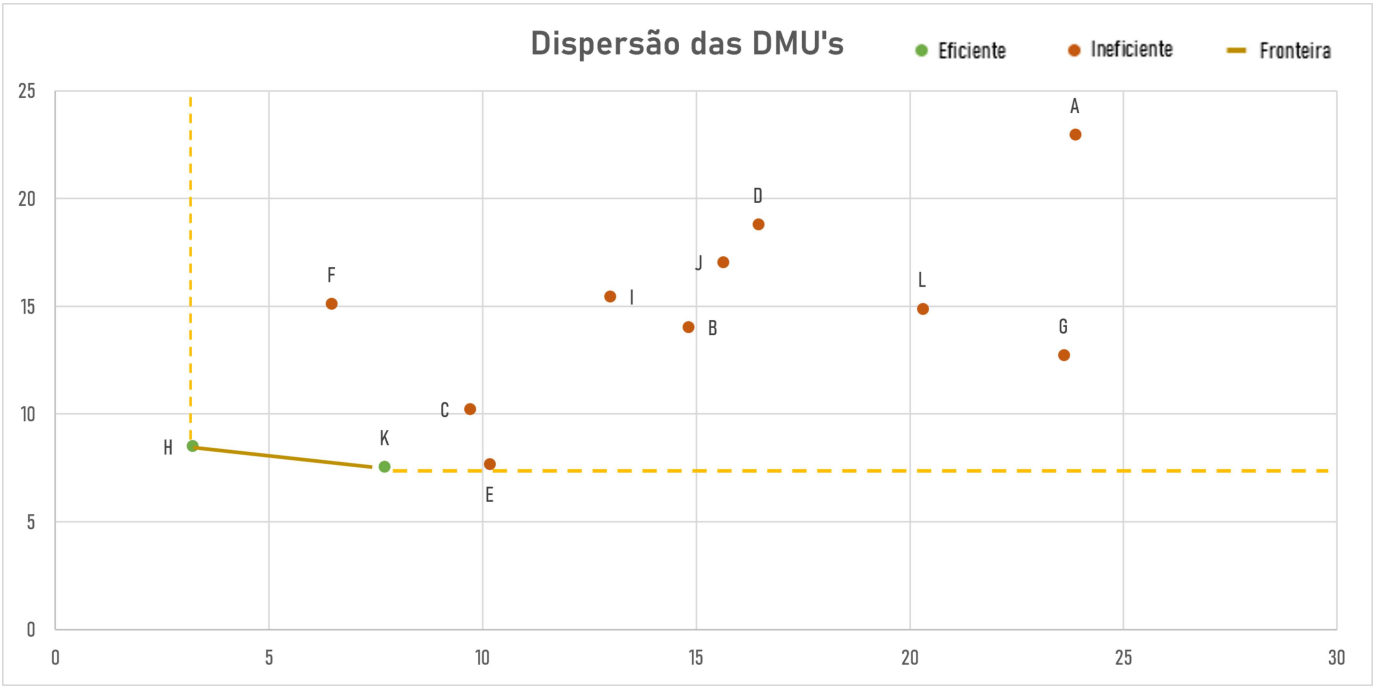
B.


Dispersão das DMU's
500
450
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
A
B
C
D
E
F
G
H
I
J
K
L
Eficiente
Ineficiente
Fronteira

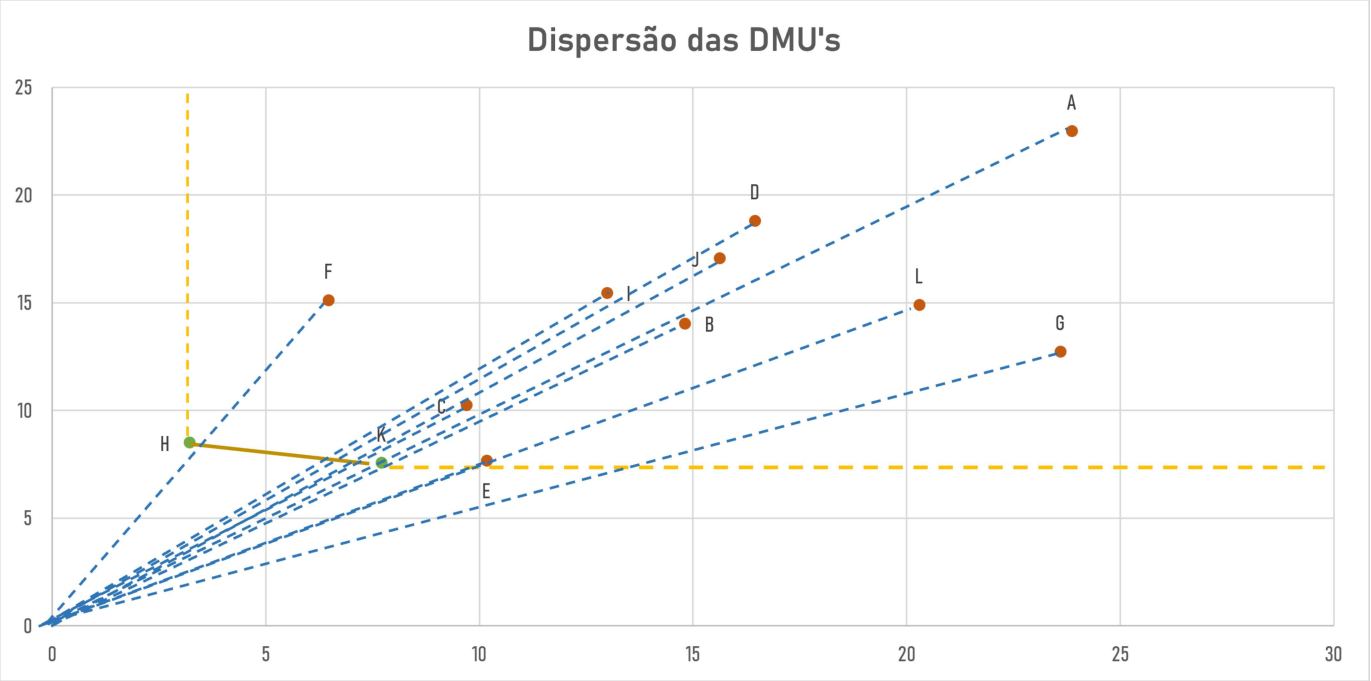
C.


Dispersão das DMU's
500
450
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
A
B
C
D
E
F
G
H
I
J
K
L
B'
C'
D'
E'
F'
G'
H'
I'
J'
K'
L'

D.


Dispersão das DMU's
Eficiente
Ineficiente
Fronteira
25
20
15
10
5
0
0
5
10
15
20
25
30
A
B
C
D
E
F
G
H
I
J
K
L

E.



DMU Ineficiente	Conjunto Benchmark
A	K
B	K
C	H-K
D	H-K
E	K
F	H-K
G	K
I	H-K
J	H-K
L	K

2.

Resolução do problema utilizando o Método CCR do envelope Orientado à *Inputs*

Exemplo do modelo de resolução para a DMU A

```
min h

st

79.95*lambdas[0] + 81.76*lambdas[1] + 66.71*lambdas[2] + 114.97*lambdas[3] + 74.7*lambdas[4] +
50.23*lambdas[5] + 197.27*lambdas[6] + 27.22*lambdas[7] + 118.39*lambdas[8] +
201.86*lambdas[9] + 112.61*lambdas[10] + 455.44*lambdas[11] - 79.95*h_dmu + slack[0] = 0

77.0*lambdas[0] + 77.5*lambdas[1] + 70.5*lambdas[2] + 131.5*lambdas[3] + 56.5*lambdas[4] +
117.5*lambdas[5] + 106.5*lambdas[6] + 72.0*lambdas[7] + 141.0*lambdas[8] +
220.5*lambdas[9] + 110.5*lambdas[10] + 334.5*lambdas[11] - 77.0*h_dmu + slack[1] = 0

3.35*lambdas[0] + 5.52*lambdas[1] + 6.88*lambdas[2] + 6.99*lambdas[3] + 7.35*lambdas[4] +
7.77*lambdas[5] + 8.36*lambdas[6] + 8.46*lambdas[7] + 9.12*lambdas[8] +
12.92*lambdas[9] + 14.61*lambdas[10] + 22.44*lambdas[11] - (3.35 + slack[2]) = 0
```

[ ]:
```python
## Modelo CCR envolope orientado à input
resultado = {'DMU':[], 'h': [], 'lambda': [], 'folgas': []}
n_input = 2

dmus = range(len(dados_df))
for problem in dmus:

    model = pyo.ConcreteModel()

    model.lambdas = pyo.Var(dmus, bounds=(0, np.inf))
    model.h_dmu = pyo.Var()

    model.slack = pyo.Var(range(3), bounds=(0, np.inf))


    lambdas = model.lambdas
```

```
    h_dmu = model.h_dmu
    folga = model.slack

    model.C1 = pyo.ConstraintList()
    for j in range(1, n_input+1):
      model.C1.add(expr= sum(lambdas[i]*dados_df[f'Input {j}'][i] for i in␣
 ↪dmus) - h_dmu * dados_df[f'Input {j}'][problem] + folga[j-1] ==0)

    model.C2 = pyo.Constraint(expr= sum(lambdas[i]*dados_df['Output'][i] for i␣
 ↪in dmus) == dados_df['Output'][problem]  + folga[2] )


    model.obj = pyo.Objective(expr=h_dmu, sense=minimize)

    opt = SolverFactory('glpk')
    opt.solve(model)


    lambdas_result = []
    for i in dmus:
        lambdas_result.append(pyo.value(lambdas[i]))

    s_result = []
    for s in range(n_input + 1):
        s_result.append(pyo.value(folga[s]))

    resultado['h'].append(pyo.value(h_dmu))
    resultado['lambda'].append(lambdas_result)
    resultado['folgas'].append(s_result)
    resultado['DMU'] = dados_df['DMU']

resultado_df = pd.DataFrame(data=resultado)

## Tratamento dos Resultados



lambda_list = []
column_name = []
for i in resultado_df.index:

    lambda_list.append(list(resultado_df.query(f'index == {i}')['lambda'].
 ↪explode('lambda')))
    column_name.append(f'lambda_{i}')

lambdas_df = pd.DataFrame(lambda_list, columns=column_name)
```

```
folga_list = []
column_name = []
for i in resultado_df.index:
    folga_list.append(list(resultado_df.query(f'index == {i}')['folgas'].
 ↪explode('folgas')))

for i in range(len(folga_list[0])):
    column_name.append(f'folga_{i}')



resultado_df = pd.merge(left=resultado_df.drop(['lambda', 'folgas'], axis=1),
                        right=pd.merge(left=lambdas_df, right=pd.
 ↪DataFrame(folga_list, columns=column_name), how='left', left_index=True,␣
 ↪right_index=True),
                        how='left', left_index=True, right_index=True).
 ↪set_index('DMU')

resultado_df
```

[ ]:

DMU	h	lambda_0	lambda_1	lambda_2	lambda_3	lambda_4	lambda_5
A	0.329053	0.0	0.0	0.0	0.0	0.0	0.0
B	0.538703	0.0	0.0	0.0	0.0	0.0	0.0
C	0.747550	0.0	0.0	0.0	0.0	0.0	0.0
D	0.412404	0.0	0.0	0.0	0.0	0.0	0.0
E	0.983900	0.0	0.0	0.0	0.0	0.0	0.0
F	0.557405	0.0	0.0	0.0	0.0	0.0	0.0
G	0.593702	0.0	0.0	0.0	0.0	0.0	0.0
H	1.000000	0.0	0.0	0.0	0.0	0.0	0.0
I	0.504935	0.0	0.0	0.0	0.0	0.0	0.0
J	0.451286	0.0	0.0	0.0	0.0	0.0	0.0
K	1.000000	0.0	0.0	0.0	0.0	0.0	0.0
L	0.507386	0.0	0.0	0.0	0.0	0.0	0.0

DMU	lambda_6	lambda_7	lambda_8	lambda_9	lambda_10	lambda_11	folga_0
A	0.0	0.000000	0.0	0.0	0.229295	0.0	0.486895
B	0.0	0.000000	0.0	0.0	0.377823	0.0	1.497668
C	0.0	0.083189	0.0	0.0	0.422739	0.0	0.000000
D	0.0	0.170135	0.0	0.0	0.379922	0.0	0.000000
E	0.0	0.000000	0.0	0.0	0.503080	0.0	16.845481
F	0.0	0.839505	0.0	0.0	0.045707	0.0	0.000000
G	0.0	0.000000	0.0	0.0	0.572211	0.0	52.682993
H	0.0	1.000000	0.0	0.0	0.000000	0.0	0.000000
I	0.0	0.276809	0.0	0.0	0.463942	0.0	0.000000

```
J          0.0  0.223424       0.0       0.0  0.754951       0.0   0.000000
K          0.0  0.000000       0.0       0.0  1.000000       0.0   0.000000
L          0.0  0.000000       0.0       0.0  1.535934       0.0  58.122486
```

```
     folga_1  folga_2
DMU
A        0.0      0.0
B        0.0      0.0
C        0.0      0.0
D        0.0      0.0
E        0.0      0.0
F        0.0      0.0
G        0.0      0.0
H        0.0      0.0
I        0.0      0.0
J        0.0      0.0
K        0.0      0.0
L        0.0      0.0
```

A.

Através do método do envelope orientado à *input*, a eficiência de cada DMU será o valor da função objetivo após a otimização, ou o próprio valor de h.

[ ]:
```
eficiencia_df = resultado_df[['h']].apply(lambda linha: round(linha*100, 2)).
↪rename(columns={'h': 'Eficiência %'})
eficiencia_df
```

[ ]:
```
     Eficiência %
DMU
A           32.91
B           53.87
C           74.76
D           41.24
E           98.39
F           55.74
G           59.37
H          100.00
I           50.49
J           45.13
K          100.00
L           50.74
```

B.

Os lambdas definirão os *benchmark* de cada DMU, sendo caracterizados em 3 casos:

1. A DMU é eficiente:

Ela será seu próprio *Benchmark*, seu respectivo lambda igual a 1 e os outros iguais a zero;

2. A DMU é fracamente eficiente:

Por mais que sua eficiência seja igual a 100%, ela terá outra DMU como *benchmark*;

3. A DMU é ineficiente:

Terá outra DMU ou uma combinação linear de 2 DMU's como *benchmark*.

[ ]:
```
lambda_dmu = { 0: 'A', 1: 'B', 2:'C', 3:'D',4:'E', 5:'F', 6:'G', 7:'H', 8:'I',␣
↪9:'J', 10:'K', 11:'L'}


bench_df = resultado_df[['lambda_0', 'lambda_1', 'lambda_2', 'lambda_3',␣
↪'lambda_4',
       'lambda_5', 'lambda_6', 'lambda_7', 'lambda_8', 'lambda_9', 'lambda_10',
       'lambda_11']]

for i in range(12):
  bench_df[f'lambda_{i}'] = bench_df[f'lambda_{i}'].apply(lambda linha:␣
↪lambda_dmu[i] if linha > 0 else '')

bench_df['benchmark'] = bench_df[['lambda_0', 'lambda_1', 'lambda_2',␣
↪'lambda_3', 'lambda_4',
       'lambda_5', 'lambda_6', 'lambda_7', 'lambda_8', 'lambda_9', 'lambda_10',
       'lambda_11']].agg(''.join, axis=1).apply(lambda linha: '-'.join(linha))

bench_df = pd.merge(left=bench_df[['benchmark']], right=eficiencia_df,␣
↪how='left', right_index=True, left_index=True).reset_index()
bench_df['status_dmu'] = np.where(bench_df['DMU'] == bench_df['benchmark'],␣
↪'Eficiente', np.where(bench_df['Eficiência %'] == 100, 'Fracamente␣
↪Eficiente', 'Ineficiente'))

bench_df
```

[ ]:

	DMU	benchmark	Eficiência %	status_dmu
0	A	K	32.91	Ineficiente
1	B	K	53.87	Ineficiente
2	C	H-K	74.76	Ineficiente
3	D	H-K	41.24	Ineficiente
4	E	K	98.39	Ineficiente
5	F	H-K	55.74	Ineficiente
6	G	K	59.37	Ineficiente
7	H	H	100.00	Eficiente
8	I	H-K	50.49	Ineficiente
9	J	H-K	45.13	Ineficiente
10	K	K	100.00	Eficiente
11	L	K	50.74	Ineficiente

C.

O cálculo do alvo será feito em duas partes:

1. Projeção Radial:

Multiplicar o valor do *input* pelo fator de redução h;

2. Alvo:

Somar a projeção radial com a respectiva folga.

Para o *output*, basta somar seu valor inicial com sua folga.

[ ]:
```
alvos_df = pd.merge(left=dados_df.set_index('DMU'), right=resultado_df[['h',␣
↪'folga_0', 'folga_1', 'folga_2']], how='left', right_index=True,␣
↪left_index=True)

for i in range(1, n_input+1):
  alvos_df[f'projecao radial ^X{i}'] = (alvos_df[f'Input {i}']*alvos_df['h'])
  alvos_df[f'alvo ^X{i}'] = (alvos_df[f'Input {i}']*alvos_df['h']) +␣
↪alvos_df[f'folga_{i-1}']

alvos_df['^Y'] = alvos_df['Output'] + alvos_df[f'folga_{n_input}']

alvos_df
```

[ ]:

DMU	Input 1	Input 2	Output	h	folga_0	folga_1	folga_2
A	79.95	77.0	3.35	0.329053	0.486895	0.0	0.0
B	81.76	77.5	5.52	0.538703	1.497668	0.0	0.0
C	66.71	70.5	6.88	0.747550	0.000000	0.0	0.0
D	114.97	131.5	6.99	0.412404	0.000000	0.0	0.0
E	74.70	56.5	7.35	0.983900	16.845481	0.0	0.0
F	50.23	117.5	7.77	0.557405	0.000000	0.0	0.0
G	197.27	106.5	8.36	0.593702	52.682993	0.0	0.0
H	27.22	72.0	8.46	1.000000	0.000000	0.0	0.0
I	118.39	141.0	9.12	0.504935	0.000000	0.0	0.0
J	201.86	220.5	12.92	0.451286	0.000000	0.0	0.0
K	112.61	110.5	14.61	1.000000	0.000000	0.0	0.0
L	455.44	334.5	22.44	0.507386	58.122486	0.0	0.0

DMU	projecao radial ^X1	alvo ^X1	projecao radial ^X2	alvo ^X2	^Y
A	26.307805	26.794699	25.337098	25.337098	3.35
B	44.044362	45.542029	41.749487	41.749487	5.52
C	49.869083	49.869083	52.702298	52.702298	6.88
D	47.414053	47.414053	54.231086	54.231086	6.99
E	73.497329	90.342810	55.590349	55.590349	7.35

F	27.998438	27.998438	65.495051	65.495051	7.77
G	117.119653	169.802646	63.229295	63.229295	8.36
H	27.220000	27.220000	72.000000	72.000000	8.46
I	59.779278	59.779278	71.195863	71.195863	9.12
J	91.096629	91.096629	99.508604	99.508604	12.92
K	112.610000	112.610000	110.500000	110.500000	14.61
L	231.084046	289.206532	169.720739	169.720739	22.44

D.

Resolução do problema utilizando o Método CCR de multiplicadores Orientado à *Inputs*

Exemplo do modelo de resolução para a DMU A

```
max 5.52*u

st

81.76*V1 + 77.5*V2 = 1

3.35*u - 79.95*V1 - 77.0*V2 <= 0

5.52*u - 81.76*V1 - 77.5*V2 <= 0

6.88*u - 66.71*V1 - 70.5*V2 <= 0

6.99*u - 114.97*V1 - 131.5*V2 <= 0

7.35*u - 74.7*V1 - 56.5*V2 <= 0

7.77*u - 50.23*V1 - 117.5*V2 <= 0

8.36*u - 197.27*V1 - 106.5*V2 <= 0

8.46*u - 27.22*V1 - 72.0*V2 <= 0

9.12*u - 118.39*V1 - 141.0*V2 <= 0

12.92*u - 201.86*V1 - 220.5*V2 <= 0

14.61*u - 112.61*V1 - 110.5*V2 <= 0

22.44*u - 455.44*V1 - 334.5*V2 <= 0
```

[ ]:
```
## MODELO CCR MULTIPLICADORES ORIENTADO À INPUT

dmus = range(len(dados_df))
```

```
resultado = {"DMU": [], "v1": [], "v2": [], "u": [], "eficiencia %": []}

for problem in dmus:


    model = pyo.ConcreteModel()

    model.inputs = pyo.Var(range(2), bounds=(0, np.inf))
    model.u = pyo.Var(bounds=(0, np.inf))
    u = model.u

    inputs = model.inputs

    model.C1 = pyo.Constraint(expr = dados_df['Input 1'][problem]*inputs[0] +␣
 ↪dados_df['Input 2'][problem]*inputs[1] == 1 )

    model.C2 = pyo.ConstraintList()
    for dmu in dmus:
        model.C2.add(expr= dados_df['Output'][dmu]*u - dados_df['Input␣
 ↪1'][dmu]*inputs[0] - dados_df['Input 2'][dmu]*inputs[1] <= 0)



    model.obj = pyo.Objective(expr=dados_df['Output'][problem]*u ,␣
 ↪sense=maximize)

    opt = SolverFactory('glpk')
    opt.solve(model)

    resultado['v1'].append(pyo.value(inputs[0]))
    resultado['v2'].append(pyo.value(inputs[1]))
    resultado['u'].append(pyo.value(u))
    resultado['DMU'] = dados_df['DMU']
    resultado['eficiencia %'].append(round(100*pyo.value(model.obj),2))


resultado_mult_input = pd.DataFrame(data=resultado).set_index('DMU')

resultado_mult_input['status'] = np.where(resultado_mult_input['eficiencia %']␣
 ↪== 100, 'Eficiente', 'Ineficiente')

print("Tabela com os resultados do método dos Multilplicadores (Orientação:␣
 ↪Input)")
resultado_mult_input
```

```
Tabela com os resultados do método dos Multilplicadores (Orientação: Input)
```

[ ]:

DMU	v1	v2	u	eficiencia %	status
A	0.000000	0.012987	0.098225	32.91	Ineficiente
B	0.000000	0.012903	0.097591	53.87	Ineficiente
C	0.002495	0.011824	0.108656	74.76	Ineficiente
D	0.001355	0.006420	0.058999	41.24	Ineficiente
E	0.000000	0.017699	0.133864	98.39	Ineficiente
F	0.001647	0.007807	0.071738	55.74	Ineficiente
G	0.000000	0.009390	0.071017	59.37	Ineficiente
H	0.002714	0.012863	0.118203	100.00	Eficiente
I	0.001271	0.006025	0.055366	50.49	Ineficiente
J	0.000802	0.003801	0.034929	45.13	Ineficiente
K	0.000000	0.009050	0.068446	100.00	Eficiente
L	0.000000	0.002990	0.022611	50.74	Ineficiente

3.

Resolução do problema utilizando o Método CCR do envelope Orientado à *Outputs*

Exemplo do modelo de resolução para a DMU A

```
max fi

st

79.95*lambdas[0] + 81.76*lambdas[1] + 66.71*lambdas[2] + 114.97*lambdas[3] + 74.7*lambdas[4] +
50.23*lambdas[5] + 197.27*lambdas[6] + 27.22*lambdas[7] + 118.39*lambdas[8] +
201.86*lambdas[9] + 112.61*lambdas[10] + 455.44*lambdas[11] - 79.95 + slack[0] = 0

77.0*lambdas[0] + 77.5*lambdas[1] + 70.5*lambdas[2] + 131.5*lambdas[3] + 56.5*lambdas[4] +
117.5*lambdas[5] + 106.5*lambdas[6] + 72.0*lambdas[7] + 141.0*lambdas[8] +
220.5*lambdas[9] + 110.5*lambdas[10] + 334.5*lambdas[11] - 77.0 + slack[1] = 0

3.35*lambdas[0] + 5.52*lambdas[1] + 6.88*lambdas[2] + 6.99*lambdas[3] + 7.35*lambdas[4] +
7.77*lambdas[5] + 8.36*lambdas[6] + 8.46*lambdas[7] + 9.12*lambdas[8] +
12.92*lambdas[9] + 14.61*lambdas[10] + 22.44*lambdas[11] - 3.35*fi - slack[2] = 0
```

[ ]:
```python
## Modelo CCR envolope orientado à output

resultado = {'DMU':[], 'fi': [], 'lambda': [], 'folgas': []}
n_input = 2

dmus = range(len(dados_df))
for problem in dmus:
```

```
    model = pyo.ConcreteModel()

    model.lambdas = pyo.Var(dmus, bounds=(0, np.inf))
    model.fi = pyo.Var()

    model.slack = pyo.Var(range(3), bounds=(0, np.inf))


    lambdas = model.lambdas
    fi = model.fi
    folga = model.slack


    model.C1 = pyo.ConstraintList()
    for j in range(1, n_input+1):
      model.C1.add(expr= sum(lambdas[i]*dados_df[f'Input {j}'][i] for i in␣
 ↪dmus) - dados_df[f'Input {j}'][problem] + folga[j-1] ==0)

    model.C2 = pyo.Constraint(expr= sum(lambdas[i]*dados_df['Output'][i] for i␣
 ↪in dmus) - dados_df['Output'][problem]*fi - folga[2] == 0)


    model.obj = pyo.Objective(expr=fi, sense=maximize)

    opt = SolverFactory('glpk')
    opt.solve(model)


    lambdas_result = []
    for i in dmus:
        lambdas_result.append(pyo.value(lambdas[i]))

    s_result = []
    for s in range(n_input +1):
        s_result.append(pyo.value(folga[s]))

    resultado['fi'].append(pyo.value(fi))
    resultado['lambda'].append(lambdas_result)
    resultado['folgas'].append(s_result)
    resultado['DMU'] = dados_df['DMU']

resultado_out_df = pd.DataFrame(data=resultado)


lambda_list = []
column_name = []
for i in resultado_out_df.index:
```

```
    lambda_list.append(list(resultado_out_df.query(f'index == {i}')['lambda'].
 ↪explode('lambda')))
    column_name.append(f'lambda_{i}')

lambdas_df = pd.DataFrame(lambda_list, columns=column_name)


folga_list = []
column_name = []
for i in resultado_out_df.index:
     folga_list.append(list(resultado_out_df.query(f'index == {i}')['folgas'].
 ↪explode('folgas')))

for i in range(len(folga_list[0])):
     column_name.append(f'folga_{i}')



resultado_out_df = pd.merge(left=resultado_out_df.drop(['lambda', 'folgas'],␣
 ↪axis=1),
                         right=pd.merge(left=lambdas_df, right=pd.
 ↪DataFrame(folga_list, columns=column_name), how='left', left_index=True,␣
 ↪right_index=True),
                         how='left', left_index=True, right_index=True).
 ↪set_index('DMU')

resultado_out_df
```

[ ]:

DMU	fi	lambda_0	lambda_1	lambda_2	lambda_3	lambda_4	lambda_5
A	3.039022	0.0	0.0	0.0	0.0	0.0	0.0
B	1.856310	0.0	0.0	0.0	0.0	0.0	0.0
C	1.337703	0.0	0.0	0.0	0.0	0.0	0.0
D	2.424809	0.0	0.0	0.0	0.0	0.0	0.0
E	1.016363	0.0	0.0	0.0	0.0	0.0	0.0
F	1.794029	0.0	0.0	0.0	0.0	0.0	0.0
G	1.684346	0.0	0.0	0.0	0.0	0.0	0.0
H	1.000000	0.0	0.0	0.0	0.0	0.0	0.0
I	1.980452	0.0	0.0	0.0	0.0	0.0	0.0
J	2.215889	0.0	0.0	0.0	0.0	0.0	0.0
K	1.000000	0.0	0.0	0.0	0.0	0.0	0.0
L	1.970885	0.0	0.0	0.0	0.0	0.0	0.0

DMU	lambda_6	lambda_7	lambda_8	lambda_9	lambda_10	lambda_11	folga_0
A	0.0	0.000000	0.0	0.0	0.696833	0.0	1.479683

```
B         0.0  0.000000       0.0       0.0  0.701357       0.0    2.780136
C         0.0  0.111282       0.0       0.0  0.565500       0.0    0.000000
D         0.0  0.412546       0.0       0.0  0.921237       0.0    0.000000
E         0.0  0.000000       0.0       0.0  0.511312       0.0   17.121131
F         0.0  1.506097       0.0       0.0  0.082000       0.0    0.000000
G         0.0  0.000000       0.0       0.0  0.963801       0.0   88.736380
H         0.0  1.000000       0.0       0.0  0.000000       0.0    0.000000
I         0.0  0.548207       0.0       0.0  0.918815       0.0    0.000000
J         0.0  0.495082       0.0       0.0  1.672887       0.0    0.000000
K         0.0  0.000000       0.0       0.0  1.000000       0.0    0.000000
L         0.0  0.000000       0.0       0.0  3.027149       0.0  114.552715
```

```
     folga_1  folga_2
DMU
A        0.0      0.0
B        0.0      0.0
C        0.0      0.0
D        0.0      0.0
E        0.0      0.0
F        0.0      0.0
G        0.0      0.0
H        0.0      0.0
I        0.0      0.0
J        0.0      0.0
K        0.0      0.0
L        0.0      0.0
```

A.

As eficiências das DMU's não mudam com o modelo orientado à *output*, são exatamente as mesmas.

[ ]:
```
eficiencia_out_df = pd.DataFrame()
eficiencia_out_df['Eficiencia output %'] = resultado_out_df[['fi']].
↪apply(lambda linha: round(100/linha, 2))
eficiencia_out_df = pd.merge(left=eficiencia_out_df, right=eficiencia_df,␣
↪how='left', right_index=True, left_index=True)

eficiencia_out_df['Teste eficiencia'] = eficiencia_out_df['Eficiencia output␣
↪%'] == eficiencia_out_df['Eficiência %']
eficiencia_out_df
```

[ ]:
```
     Eficiencia output %  Eficiência %  Teste eficiencia
DMU
A                  32.91         32.91              True
B                  53.87         53.87              True
C                  74.76         74.76              True
D                  41.24         41.24              True
```

E	98.39	98.39	True
F	55.74	55.74	True
G	59.37	59.37	True
H	100.00	100.00	True
I	50.49	50.49	True
J	45.13	45.13	True
K	100.00	100.00	True
L	50.74	50.74	True

B.

O cálculo do alvo será feito da seguinte maneira para cada caso:

1. *Inputs*:

Os *inputs* iniciais serão somados com suas respectivas folgas;

2. *Output*:

O *output* inicial de cada DMU será multiplicado por seu fi e somado com a sua folga.

[ ]:
```
lambda_dmu = { 0: 'A', 1: 'B', 2:'C', 3:'D',4:'E', 5:'F', 6:'G', 7:'H', 8:'I',␣
 ↪9:'J', 10:'K', 11:'L'}


bench_out_df = resultado_out_df[['lambda_0', 'lambda_1', 'lambda_2',␣
 ↪'lambda_3', 'lambda_4',
       'lambda_5', 'lambda_6', 'lambda_7', 'lambda_8', 'lambda_9', 'lambda_10',
       'lambda_11']]

for i in range(12):
  bench_out_df[f'lambda_{i}'] = bench_out_df[f'lambda_{i}'].apply(lambda linha:␣
 ↪lambda_dmu[i] if linha > 0 else '')

bench_out_df['benchmark'] = bench_out_df[['lambda_0', 'lambda_1', 'lambda_2',␣
 ↪'lambda_3', 'lambda_4',
       'lambda_5', 'lambda_6', 'lambda_7', 'lambda_8', 'lambda_9', 'lambda_10',
       'lambda_11']].agg(''.join, axis=1).apply(lambda linha: '-'.join(linha))

bench_out_df = pd.merge(left=bench_out_df[['benchmark']], right=eficiencia_df,␣
 ↪how='left', right_index=True, left_index=True).reset_index()
bench_out_df['status_dmu'] = np.where(bench_out_df['DMU'] ==␣
 ↪bench_out_df['benchmark'], 'Eficiente', np.where(bench_out_df['Eficiência␣
 ↪%'] == 100, 'Fracamente Eficiente', 'Ineficiente'))

alvos = []
alvos_out = pd.merge(left=dados_df.set_index('DMU'),␣
 ↪right=resultado_out_df[['fi', 'folga_0', 'folga_1', 'folga_2']], how='left',␣
 ↪right_index=True, left_index=True)
```

```
for i in range(1, n_input+1):
  alvos_out[f'alvo ^X{i}'] = alvos_out[f'Input {i}'] + alvos_out[f'folga_{i-1}']
  alvos.append(f'alvo ^X{i}')

alvos_out['alvo ^Y'] = alvos_out['Output']*alvos_out['fi'] +␣
 ↪alvos_out[f'folga_{n_input}']
alvos.append('alvo ^Y')


bench_out_df = pd.merge(left=bench_out_df.set_index('DMU'),␣
 ↪right=alvos_out[alvos], how='left', right_index=True, left_index=True)

bench_out_df
```

[ ]:

DMU	benchmark	Eficiência %	status_dmu	alvo ^X1	alvo ^X2	alvo ^Y
A	K	32.91	Ineficiente	81.429683	77.0	10.180724
B	K	53.87	Ineficiente	84.540136	77.5	10.246833
C	H-K	74.76	Ineficiente	66.710000	70.5	9.203394
D	H-K	41.24	Ineficiente	114.970000	131.5	16.949412
E	K	98.39	Ineficiente	91.821131	56.5	7.470271
F	H-K	55.74	Ineficiente	50.230000	117.5	13.939603
G	K	59.37	Ineficiente	286.006380	106.5	14.081131
H	H	100.00	Eficiente	27.220000	72.0	8.460000
I	H-K	50.49	Ineficiente	118.390000	141.0	18.061724
J	H-K	45.13	Ineficiente	201.860000	220.5	28.629283
K	K	100.00	Eficiente	112.610000	110.5	14.610000
L	K	50.74	Ineficiente	569.992715	334.5	44.226652

Resolução do problema utilizando o Método CCR de multiplicadores Orientado à *outputs*

Exemplo do modelo de resolução para a DMU A

```
min 81.76*V1 + 77.5*V2

st

3.35*u = 1

3.35*u - 79.95*V1 - 77.0*V2 <= 0

5.52*u - 81.76*V1 - 77.5*V2 <= 0

6.88*u - 66.71*V1 - 70.5*V2 <= 0

6.99*u - 114.97*V1 - 131.5*V2 <= 0
```

```
7.35*u - 74.7*V1 - 56.5*V2 <= 0

7.77*u - 50.23*V1 - 117.5*V2 <= 0

8.36*u - 197.27*V1 - 106.5*V2 <= 0

8.46*u - 27.22*V1 - 72.0*V2 <= 0

9.12*u - 118.39*V1 - 141.0*V2 <= 0

12.92*u - 201.86*V1 - 220.5*V2 <= 0

14.61*u - 112.61*V1 - 110.5*V2 <= 0

22.44*u - 455.44*V1 - 334.5*V2 <= 0
```

[ ]:
```
## MODELO CCR MULTIPLICADORES ORIENTADO À OUTPUT

dmus = range(len(dados_df))

resultado = {"DMU": [], "v1": [], "v2": [], "u": [], "eficiencia %": []}

for problem in dmus:


    model = pyo.ConcreteModel()

    model.inputs = pyo.Var(range(2), bounds=(0, np.inf))
    model.u = pyo.Var(bounds=(0, np.inf))
    u = model.u

    inputs = model.inputs

    model.C1 = pyo.Constraint(expr = dados_df['Output'][problem]*u == 1 )

    model.C2 = pyo.ConstraintList()
    for dmu in dmus:
        model.C2.add(expr= dados_df['Output'][dmu]*u - dados_df['Input␣
↪1'][dmu]*inputs[0] - dados_df['Input 2'][dmu]*inputs[1] <= 0)


    model.obj = pyo.Objective(expr= dados_df['Input 1'][problem]*inputs[0] +␣
↪dados_df['Input 2'][problem]*inputs[1], sense=minimize)

    opt = SolverFactory('glpk')
    opt.solve(model)
```

```
    resultado['v1'].append(pyo.value(inputs[0]))
    resultado['v2'].append(pyo.value(inputs[1]))
    resultado['u'].append(pyo.value(u))
    resultado['DMU'] = dados_df['DMU']
    resultado['eficiencia %'].append(round(100/pyo.value(model.obj), 2))


resultado_mult_output = pd.DataFrame(data=resultado).set_index('DMU')

resultado_mult_output['status'] = np.where(resultado_mult_output['eficiencia␣
↪%'] == 100, 'Eficiente', 'Ineficiente')

print("Tabela com os resultados do método dos Multilplicadores (Orientação:␣
↪Output)")
resultado_mult_input

resultado_mult_output
```

Tabela com os resultados do método dos Multilplicadores (Orientação: Output)

[ ]:

DMU	v1	v2	u	eficiencia %	status
A	0.000000	0.039468	0.298507	32.91	Ineficiente
B	0.000000	0.023952	0.181159	53.87	Ineficiente
C	0.003337	0.015817	0.145349	74.76	Ineficiente
D	0.003284	0.015568	0.143062	41.24	Ineficiente
E	0.000000	0.017989	0.136054	98.39	Ineficiente
F	0.002955	0.014005	0.128700	55.74	Ineficiente
G	0.000000	0.015815	0.119617	59.37	Ineficiente
H	0.036738	0.000000	0.118203	100.00	Eficiente
I	0.002517	0.011932	0.109649	50.49	Ineficiente
J	0.001777	0.008423	0.077399	45.13	Ineficiente
K	0.001571	0.007448	0.068446	100.00	Eficiente
L	0.000000	0.005892	0.044563	50.74	Ineficiente