DEA-ASSIGNMENT-QMM-03

ANURODH-SINGH

2023-11-03

CONCLUSION

'T_output' variable table shows that the 3 resultant peers or frontiers or boundary conditions for each energy policy are obtained as a 3-dimensional reference following x, y, and z axis, depending on the inputs ("D.C size" and "#Shutdowns") and outputs ("Computing Time(h)", "MWh Consumed", and "Queue time(ms)").

DEA analysis results show that the efficiencies for all 18 energy policies are [1.0000 1.0000 0.9991 0.4818 1.0000 0.4872 1.0000 0.9826 0.9578 1.0000 0.9806 0.4754 1.0000 0.9944 1.0000 0.9970 0.5290 0.4783] respectively. This means that 7 energy policies are 100% efficient and will be treated as the frontiers or boundary conditions, while the other 11 need to improve their performance.

The following table shows the amount/percentage of improvement required by each inefficient energy policy:

Energy Policy Amount/Percentage of Improvement Required 3 Gamma Omega High 0.09%
4 Always Mono. Low 51.82%
6 Load Omega Low 51.28%
8 Gamma Mono. High 1.74%
9 Random Mesos High 4.22%
11 ExponentialOmega High 1.94%
12 Margin Omega Low 52.46%
14 Gamma Mono. High 0.56%
16 Gamma Omega High 0.3%
17 Gamma Mesos Low 47.1%
18 Random Omega Low 52.17%
The amount/percentage of improvement using the peers energy policies is found out using the lambda function with Constant Return to scale (CRS), which is the biggest umbrella. This is stored in the 'CRS_Weights' variable.

The result implies that for example if energy policy "8 Gamma Mono. High" is chosen then it needs to learn or follow around 22.1% of energy policy "2 Margin Mesos High", 59.15% of energy policy "10 Margin Omega High", and 17.35% of energy policy "13 Margin Mono. High". The same applies to other policies as well.

In other words, energy policy "8 Gamma Mono. High" can improve its performance by learning from the best practices of energy policies "2 Margin Mesos High", "10 Margin Omega High", and "13 Margin Mono. High".

SUMMARY

- First of all The library "Benchmarking" is installed.

- Then two matrices, 'x' and 'y', are created and combined using the 'cbind' function.
- 'x' contains data related to data center size and shutdowns.
- 'y' contains data related to computing time, MWh consumption, and queue time.
- Row names are assigned to 'T_output' for data point identification.
- 'T_output' is printed to display the combined data.
- DEA (Data Envelopment Analysis) is conducted on 'x' and 'y' using the 'dea' function with the "RTS" parameter set to "crs" for Constant Returns to Scale.
- Efficiency scores generated by DEA are printed using the 'print' function.
- The 'peers' function is utilized to identify peers for each data point based on relative efficiency scores.
- Efficiency scores for each data point are extracted and stored in the 'CRS_Weights' variable using the 'lambda' function

---

#Installing required packages

```
#install.packages("Benchmarking")
library(Benchmarking)
```

```
## Loading required package: lpSolveAPI
```

```
## Loading required package: ucminf
```

```
## Loading required package: quadprog
```

```
# Create matrices with the same number of rows
x <- matrix(c(1000, 1000, 1000, 1000, 1000, 1000, 5000, 5000, 5000, 5000, 5000, 5000, 10000, 10000, 100
y <- matrix(c(104.42, 104.26, 104.17, 49.25, 49.63, 49.34, 99.96, 99.96, 100.03, 100.26, 100.26, 46.7, 

colnames(y) <- c("Computing Time(h)", "MWh Consumed", "Queue time(ms)")
colnames(x) <- c("D.C size", "#Shutdowns")

print(x) # Print the values of 'X'
```

```
##       D.C size #Shutdowns
##  [1,]     1000      37166
##  [2,]     1000      13361
##  [3,]     1000      14252
##  [4,]     1000      36404
##  [5,]     1000      19671
##  [6,]     1000      32407
##  [7,]     5000       6981
##  [8,]     5000       9877
##  [9,]     5000      33589
## [10,]     5000       8578
## [11,]     5000      11863
## [12,]     5000      15452
## [13,]    10000       9680
```

```
## [14,]    10000      11388
## [15,]    10000      18150
## [16,]    10000      18409
## [17,]    10000      29707
## [18,]    10000      40772
```

```r
print(y) # Print the values of 'Y'
```

```
##       Computing Time(h) MWh Consumed Queue time(ms)
##  [1,]            104.42        49.01           90.1
##  [2,]            104.26        49.65         1093.0
##  [3,]            104.17        49.60            0.1
##  [4,]             49.25        23.92           78.3
##  [5,]             49.63        24.65         1188.7
##  [6,]             49.34        24.19            1.1
##  [7,]             99.96       237.09          126.2
##  [8,]             99.96       235.92          129.8
##  [9,]            100.03       234.90         1122.6
## [10,]            100.26       239.13            0.7
## [11,]            100.26       236.95            1.0
## [12,]             46.70       115.82            0.5
## [13,]            101.56       481.36          325.2
## [14,]            101.56       479.36          327.9
## [15,]            101.63       486.11            2.6
## [16,]            101.63       484.69            2.5
## [17,]             45.83       228.31         1107.6
## [18,]             46.09       233.50            3.8
```

```r
# Assuming the matrices 'x' and 'y' have the same number of rows, you can cbind them
T_output <- cbind(x, y)
row.names(T_output) <- c("1 Always Monolithic High", "2 Margin Mesos High", "3 Gamma Omega High", "4 Al

T_output
```

```
##                          D.C size #Shutdowns Computing Time(h) MWh Consumed
## 1 Always Monolithic High     1000      37166            104.42        49.01
## 2 Margin Mesos High          1000      13361            104.26        49.65
## 3 Gamma Omega High           1000      14252            104.17        49.60
## 4 Always Mono. Low           1000      36404             49.25        23.92
## 5 ExponentialMesos Low       1000      19671             49.63        24.65
## 6 Load Omega Low             1000      32407             49.34        24.19
## 7 Margin Mono. High          5000       6981             99.96       237.09
## 8 Gamma Mono. High           5000       9877             99.96       235.92
## 9 Random Mesos High          5000      33589            100.03       234.90
## 10 Margin Omega High         5000       8578            100.26       239.13
## 11 ExponentialOmega High     5000      11863            100.26       236.95
## 12 Margin Omega Low          5000      15452             46.70       115.82
## 13 Margin Mono. High        10000       9680            101.56       481.36
## 14 Gamma Mono. High         10000      11388            101.56       479.36
## 15 Margin Omega High        10000      18150            101.63       486.11
## 16 Gamma Omega High         10000      18409            101.63       484.69
## 17 Gamma Mesos Low          10000      29707             45.83       228.31
## 18 Random Omega Low         10000      40772             46.09       233.50
```

```
##                           Queue time(ms)
## 1 Always Monolithic High           90.1
## 2 Margin Mesos High              1093.0
## 3 Gamma Omega High                  0.1
## 4 Always Mono. Low                 78.3
## 5 ExponentialMesos Low           1188.7
## 6 Load Omega Low                    1.1
## 7 Margin Mono. High               126.2
## 8 Gamma Mono. High                129.8
## 9 Random Mesos High              1122.6
## 10 Margin Omega High                0.7
## 11 ExponentialOmega High            1.0
## 12 Margin Omega Low                 0.5
## 13 Margin Mono. High              325.2
## 14 Gamma Mono. High               327.9
## 15 Margin Omega High                2.6
## 16 Gamma Omega High                 2.5
## 17 Gamma Mesos Low               1107.6
## 18 Random Omega Low                 3.8
```

```
CRS <- dea(x,y, RTS = "crs")
print(CRS)
```

```
##  [1] 1.0000 1.0000 0.9991 0.4818 1.0000 0.4872 1.0000 0.9826 0.9578 1.0000
## [11] 0.9806 0.4754 1.0000 0.9944 1.0000 0.9970 0.5290 0.4783
```

```
peers(CRS)
```

```
##       peer1 peer2 peer3
##  [1,]     1    NA    NA
##  [2,]     2    NA    NA
##  [3,]     1     2    NA
##  [4,]     2    NA    NA
##  [5,]     5    NA    NA
##  [6,]     2    NA    NA
##  [7,]     7    NA    NA
##  [8,]     2    10    13
##  [9,]     2    15    NA
## [10,]    10    NA    NA
## [11,]     2    13    15
## [12,]     2    15    NA
## [13,]    13    NA    NA
## [14,]     2    13    15
## [15,]    15    NA    NA
## [16,]     2    15    NA
## [17,]     2    13    NA
## [18,]     2    15    NA
```

```
CRS_Weights <- lambda(CRS)
CRS_Weights
```

```
##               L1         L2 L5 L7       L10       L13       L15
```

```
##  [1,] 1.000000000 0.00000000  0  0 0.0000000 0.0000000 0.00000000
##  [2,] 0.000000000 1.00000000  0  0 0.0000000 0.0000000 0.00000000
##  [3,] 0.009970484 0.98915099  0  0 0.0000000 0.0000000 0.00000000
##  [4,] 0.000000000 0.48177241  0  0 0.0000000 0.0000000 0.00000000
##  [5,] 0.000000000 0.00000000  1  0 0.0000000 0.0000000 0.00000000
##  [6,] 0.000000000 0.48721047  0  0 0.0000000 0.0000000 0.00000000
##  [7,] 0.000000000 0.00000000  0  1 0.0000000 0.0000000 0.00000000
##  [8,] 0.000000000 0.22098286  0  0 0.5914729 0.1734861 0.00000000
##  [9,] 0.000000000 2.03346741  0  0 0.0000000 0.0000000 0.27553094
## [10,] 0.000000000 0.00000000  0  0 1.0000000 0.0000000 0.00000000
## [11,] 0.000000000 0.53626578  0  0 0.0000000 0.4082527 0.02840485
## [12,] 0.000000000 0.26256674  0  0 0.0000000 0.0000000 0.21144095
## [13,] 0.000000000 0.00000000  0  0 0.0000000 1.0000000 0.00000000
## [14,] 0.000000000 0.04516562  0  0 0.0000000 0.8554257 0.13443418
## [15,] 0.000000000 0.00000000  0  0 0.0000000 0.0000000 1.00000000
## [16,] 0.000000000 0.02236541  0  0 0.0000000 0.0000000 0.99479451
## [17,] 0.000000000 0.89985422  0  0 0.0000000 0.3814863 0.00000000
## [18,] 0.000000000 0.93720988  0  0 0.0000000 0.0000000 0.38461980
```