Python Codes for FN14 of AEJ Paper

These are the Python codes (and outputs) as referred to in footnote 14 (FN14) of the Accounting Educators' Journal (AEJ) paper.

In [1]:

```
# Direct method
# Illustration based on data taken from Bhimani et al. (2019)
# Case with 2 support departments (PM and IS) and 2 operating departments (Machining and Assem
bly)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([600000,116000,400000,200000])
serv_perc = np.array([[-1.00,0.20,0.30,0.50],
                      [0.10,-1.00,0.80,0.10],
                      [0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Auxiliary calculations

aux_a = serv_perc[0, 2:].sum()
aux_b = serv_perc[1, 2:].sum()

# Calculate the allocation percentages

serv_perc[:, 0] = 0.00
serv_perc[0, 0] = -1.00
serv_perc[:, 1] = 0.00
serv_perc[1, 1] = -1.00
serv_perc[0, 2] = (serv_perc[0, 2] / aux_a)
serv_perc[0, 3] = (serv_perc[0, 3] / aux_a)
serv_perc[1, 2] = (serv_perc[1, 2] / aux_b)
serv_perc[1, 3] = (serv_perc[1, 3] / aux_b)
print("\nAllocation percentages:")
print(serv_perc.round(2))

# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0]],
              [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1]],
              [0,0,dep_costs[2],0],
              [0,0,0,dep_costs[3]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the direct method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col3_sum = all_costs[:, 2].sum()
col4_sum = all_costs[:, 3].sum()
print("\nFinal costs of the Machining department: $" + str(int(col3_sum)))
print("Final costs of the Assembly department: $" + str(int(col4_sum)))
print("\nTotal costs (grand total): $" + str(int((col3_sum + col4_sum))))
```

```
Department costs:
[600000 116000 400000 200000]

Allocation percentages:
[[-1.    0.    0.37  0.62]
 [ 0.   -1.    0.89  0.11]
 [ 0.    0.    1.    0.  ]
 [ 0.    0.    0.    1.  ]]

Cost allocation (based on the direct method):
[[-600000       0  224999  375000]
 [      0 -116000  103111   12888]
 [      0       0  400000       0]
 [      0       0       0  200000]]

Final costs of the Machining department: $728111
Final costs of the Assembly department: $587888

Total costs (grand total): $1316000
```

In [2]:

```
# Step-down method
# Illustration based on data taken from Bhimani et al. (2019)
# Case with 2 support departments (PM and IS) and 2 operating departments (Machining and Assem
bly)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([600000,116000,400000,200000])
serv_perc = np.array([[-1.00,0.20,0.30,0.50],
                      [0.10,-1.00,0.80,0.10],
                      [0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Calculate the allocation percentages

serv_perc[1:, 0] = 0.00
serv_perc[2:, 1] = 0.00

aux_a = serv_perc[1, 2:].sum()

serv_perc[1, 2] = (serv_perc[1, 2] / aux_a)
serv_perc[1, 3] = (serv_perc[1, 3] / aux_a)

dep_costs[1] = dep_costs[1] + serv_perc[0,1] * dep_costs[0]

print("\nAllocation percentages:")
print(serv_perc.round(2))

# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0]],
              [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1]],
              [0,0,dep_costs[2],0],
              [0,0,0,dep_costs[3]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the step-down method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col3_sum = all_costs[:, 2].sum()
col4_sum = all_costs[:, 3].sum()
print("\nFinal costs of the Machining department: $" + str(int(col3_sum)))
print("Final costs of the Assembly department: $" + str(int(col4_sum)))
print("\nTotal costs (grand total): $" + str(int((col3_sum + col4_sum))))
```

```
Department costs:
[600000 116000 400000 200000]

Allocation percentages:
[[-1.    0.2   0.3   0.5 ]
 [ 0.   -1.    0.89  0.11]
 [ 0.    0.    1.    0.  ]
 [ 0.    0.    0.    1.  ]]

Cost allocation (based on the step-down method):
[[-600000  120000  180000  300000]
 [      0 -236000  209777   26222]
 [      0       0  400000       0]
 [      0       0       0  200000]]

Final costs of the Machining department: $789777
Final costs of the Assembly department: $526222

Total costs (grand total): $1316000
```

In [3]:

```
# Reciprocal method
# Illustration based on data taken from Bhimani et al. (2019)
# Case with 2 support departments (PM and IS) and 2 operating departments (Machining and Assem
bly)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([600000,116000,400000,200000])
serv_perc = np.array([[-1.00,0.20,0.30,0.50],
                      [0.10,-1.00,0.80,0.10],
                      [0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Calculate the negative transpose matrix of the interdepartmental services

A = (serv_perc[0:2, 0:2].transpose()) * - 1.0
print("\nNegative transpose matrix:")
print(A)

# Calculate the allocated costs

B = np.array([dep_costs[0],dep_costs[1]])
C = np.linalg.solve(A, B)
print("\nReciprocated costs of the support departments:")
print(C.astype(int))

D = np.array([[C[0],C[0],C[0],C[0]],
              [C[1],C[1],C[1],C[1]],
              [0,0,dep_costs[2],0],
              [0,0,0,dep_costs[3]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the reciprocal method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col3_sum = all_costs[:, 2].sum()
col4_sum = all_costs[:, 3].sum()
print("\nFinal costs of the Machining department: $" + str(int(col3_sum)))
print("Final costs of the Assembly department: $" + str(int(col4_sum)))
print("\nTotal costs (grand total): $" + str(int((col3_sum + col4_sum))))
```

```
Department costs:
[600000 116000 400000 200000]

Negative transpose matrix:
[[ 1.  -0.1]
 [-0.2  1. ]]

Reciprocated costs of the support departments:
[624081 240816]

Cost allocation (based on the reciprocal method):
[[-624081  124816  187224  312040]
 [  24081 -240816  192653   24081]
 [      0       0  400000       0]
 [      0       0       0  200000]]

Final costs of the Machining department: $779877
Final costs of the Assembly department: $536122

Total costs (grand total): $1316000
```

In [4]:

```
# Reciprocal method
# Illustration based on data taken from Bent & Caplan (2017) and Togo (2010)
# Case with 4 support departments (A, B, C and D) and 3 operating departments (X, Y, Z)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([900000,600000,500000,300000,8500000,5000000,4200000])
serv_perc = np.array([[-1.00,0.08,0.07,0.05,0.40,0.25,0.15],
                      [0.09,-1.00,0.03,0.06,0.30,0.35,0.17],
                      [0.05,0.06,-1.00,0.04,0.40,0.25,0.20],
                      [0.03,0.07,0.04,-1.00,0.32,0.40,0.14],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Calculate the negative transpose matrix of the interdepartmental services

A = (serv_perc[0:4, 0:4].transpose()) * - 1.0
print("\nNegative transpose matrix:")
print(A)

# Calculate the allocated costs

B = np.array([dep_costs[0],dep_costs[1],dep_costs[2],dep_costs[3]])
C = np.linalg.solve(A, B)
print("\nReciprocated costs of the support departments:")
print(C.astype(int))

D = np.array([[C[0],C[0],C[0],C[0],C[0],C[0],C[0]],
              [C[1],C[1],C[1],C[1],C[1],C[1],C[1]],
              [C[2],C[2],C[2],C[2],C[2],C[2],C[2]],
              [C[3],C[3],C[3],C[3],C[3],C[3],C[3]],
              [0,0,0,0,dep_costs[4],0,0],
              [0,0,0,0,0,dep_costs[5],0],
              [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the reciprocal method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department X: $" + str(int(col5_sum)))
print("Final costs of department Y: $" + str(int(col6_sum)))
print("Final costs of department Z: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```

```
Department costs:
[ 900000  600000  500000  300000 8500000 5000000 4200000]

Negative transpose matrix:
[[ 1.   -0.09 -0.05 -0.03]
 [-0.08  1.   -0.06 -0.07]
 [-0.07 -0.03  1.   -0.04]
 [-0.05 -0.06 -0.04  1.  ]]

Reciprocated costs of the support departments:
[1010299  746799  609913  419719]

Cost allocation (based on the reciprocal method):
[[-1010299    80823    70720    50514   404119   252574   151544]
 [   67211  -746799    22403    44807   224039   261379   126955]
 [   30495    36594  -609913    24396   243965   152478   121982]
 [   12591    29380    16788  -419719   134310   167887    58760]
 [       0        0        0        0  8500000        0        0]
 [       0        0        0        0        0  5000000        0]
 [       0        0        0        0        0        0  4200000]]

Final costs of department X: $9506435
Final costs of department Y: $5834320
Final costs of department Z: $4659244

Total costs (grand total): $20000000
```

In [5]:

```
# Lattice allocation method
# Illustration based on data taken from Bent & Caplan (2017) and Togo (2010)
# Case with 4 support departments (A, B, C and D) and 3 operating departments (X, Y, Z)

import numpy as np

# Cost data, support services provided and desired accuracy

dep_costs = np.array([900000,600000,500000,300000,8500000,5000000,4200000])
serv_perc = np.array([[-1.00,0.08,0.07,0.05,0.40,0.25,0.15],
                      [0.09,-1.00,0.03,0.06,0.30,0.35,0.17],
                      [0.05,0.06,-1.00,0.04,0.40,0.25,0.20],
                      [0.03,0.07,0.04,-1.00,0.32,0.40,0.14],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])
cut_off = 0.001

print("Department costs:")
print(dep_costs)

# Create the lattice allocation matrix

serv_perc[0, 0] = 0.00
serv_perc[1, 1] = 0.00
serv_perc[2, 2] = 0.00
serv_perc[3, 3] = 0.00
print("\nLattice allocation matrix:")
print(serv_perc)

# Matrix multiplication(s) needed to obtain the desired accuracy

col1_sum = serv_perc[:, 0].sum()
col2_sum = serv_perc[:, 1].sum()
col3_sum = serv_perc[:, 2].sum()
col4_sum = serv_perc[:, 3].sum()
n = 0
while (col1_sum > cut_off) | (col2_sum > cut_off) | (col3_sum > cut_off) | (col4_sum > cut_off
):
    n += 1
    serv_perc_adj = np.linalg.matrix_power(serv_perc, n)
    col1_sum = serv_perc_adj[:, 0].sum()
    col2_sum = serv_perc_adj[:, 1].sum()
    col3_sum = serv_perc_adj[:, 2].sum()
    col4_sum = serv_perc_adj[:, 3].sum()
print("\nNumber of matrix multiplications needed to obtain the desired accuracy: " + str(n))
print("\nLattice allocation matrix at the desired accuracy level:")
print(serv_perc_adj.round(2))

# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],d
ep_costs[0]],
              [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],d
ep_costs[1]],
              [dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],d
ep_costs[2]],
              [dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],d
ep_costs[3]],
              [0,0,0,0,dep_costs[4],0,0],
              [0,0,0,0,0,dep_costs[5],0],
              [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc_adj, D)
print("\nCost allocation (based on the lattice allocation method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department X: $" + str(int(col5_sum)))
print("Final costs of department Y: $" + str(int(col6_sum)))
print("Final costs of department Z: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```

```
Department costs:
[ 900000  600000  500000  300000 8500000 5000000 4200000]

Lattice allocation matrix:
[[0.   0.08 0.07 0.05 0.4  0.25 0.15]
 [0.09 0.   0.03 0.06 0.3  0.35 0.17]
 [0.05 0.06 0.   0.04 0.4  0.25 0.2 ]
 [0.03 0.07 0.04 0.   0.32 0.4  0.14]
 [0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.  ]]

Number of matrix multiplications needed to obtain the desired accuracy: 4

Lattice allocation matrix at the desired accuracy level:
[[0.   0.   0.   0.   0.48 0.33 0.19]
 [0.   0.   0.   0.   0.38 0.42 0.2 ]
 [0.   0.   0.   0.   0.46 0.31 0.23]
 [0.   0.   0.   0.   0.38 0.45 0.17]
 [0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.  ]]

Cost allocation (based on the lattice allocation method):
[[    239     235     164     197  433192  294423  171547]
 [    113     171     127     117  227770  249299  122400]
 [     99     108      80      87  230873  154546  114203]
 [     61      57      41      50  113770  135312   50706]
 [      0       0       0       0 8500000       0       0]
 [      0       0       0       0       0 5000000       0]
 [      0       0       0       0       0       0 4200000]]

Final costs of department X: $9505606
Final costs of department Y: $5833581
Final costs of department Z: $4658857

Total costs (grand total): $19998045
```