stackoverflow.com

Transportation cost flow optimization using Python Scipy Minimize

6-8 minutes

---

1. **your communities**

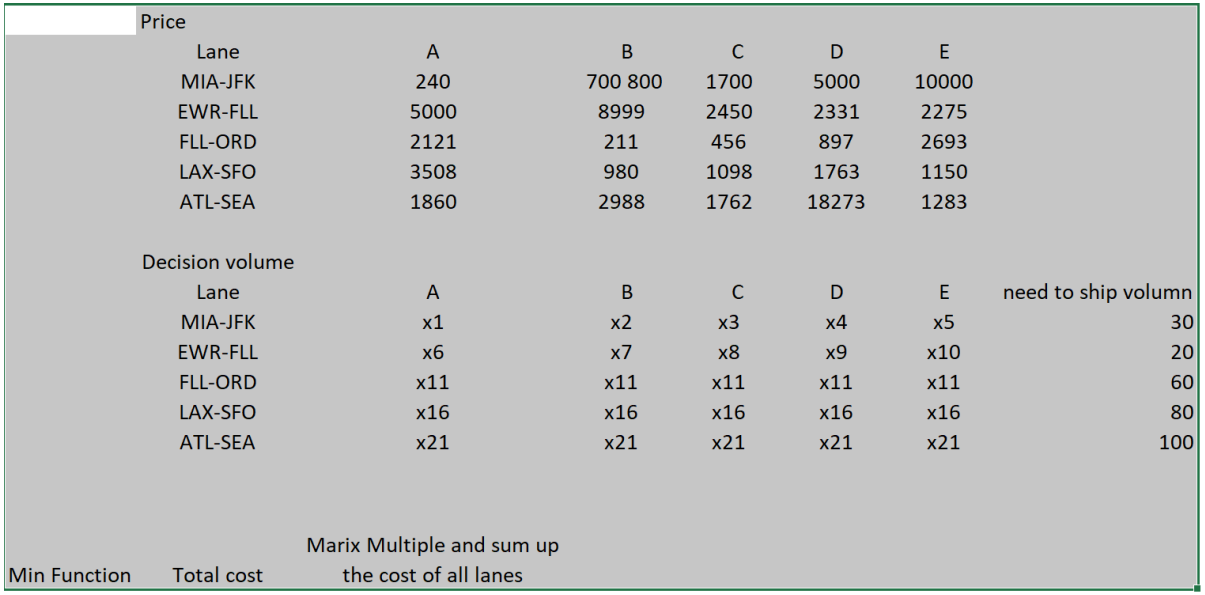
2. 🔍

3. Log in

4. Sign up

Asked 4 years, 5 months ago

I have a transportation cost flow problem and the purpose is to minimize overall transportation cost from 5 carriers and more than 3000 transport lanes (Ex. NY to MIA) I will mock up some sample data from my dataset to help you have a better understanding of the problem. Please see my data image here



	Price						
	Lane	A	B	C	D	E	
	MIA-JFK	240	700 800	1700	5000	10000	
	EWR-FLL	5000	8999	2450	2331	2275	
	FLL-ORD	2121	211	456	897	2693	
	LAX-SFO	3508	980	1098	1763	1150	
	ATL-SEA	1860	2988	1762	18273	1283	
	Decision volume						
	Lane	A	B	C	D	E	need to ship volumn
	MIA-JFK	x1	x2	x3	x4	x5	30
	EWR-FLL	x6	x7	x8	x9	x10	20
	FLL-ORD	x11	x11	x11	x11	x11	60
	LAX-SFO	x16	x16	x16	x16	x16	80
	ATL-SEA	x21	x21	x21	x21	x21	100
Min Function	Total cost	Marix Multiple and sum up the cost of all lanes					

I have tried Lonprog but it only works for lane by lane not for matrix decision variables

Please advise proper way to solve the problem without commercial solvers (standard excel solver has 200 variables limit)

Thanks

3

1 Answer 1

Your problem is a nicely structured transportation problem. It can be tackled in various ways.

If you want to solve it with linear programming, you can use `scipy.optimize.linprog`. Encoding the variables is a little more difficult with multi dimensional decision variables.

With `scipy.optimize.linprog` you could model and solve you problem like this:

```
import random
import numpy as np
import scipy.optimize
```

```
LANES = 30
CARRIERS = 6

cost = np.random.rand(LANES, CARRIERS) # c
demand = np.random.rand(LANES) # b_eq
capacity = [250, 300, 500, 750, 100, 200] # b_ub

A_eq = np.zeros(LANES*CARRIERS*LANES).reshape(LANES,
LANES*CARRIERS)
# Constraint for each lane, sum over the available
carriers
for l in range(LANES):
    for var in range(l*CARRIERS, l*CARRIERS+CARRIERS):
        A_eq[l, var] = 1

A_ub =
np.zeros(CARRIERS*LANES*CARRIERS).reshape(CARRIERS,
LANES*CARRIERS)
# Constraint for each carrier, sum over the lanes
for c in range(CARRIERS):
    for var in range(c, LANES*CARRIERS, CARRIERS):
```

```
        A_ub[c, var] = 1

print(scipy.optimize.linprog(cost.flatten(),
A_eq=A_eq, b_eq=demand,
    A_ub=A_ub, b_ub=capacity, options={"maxiter":
10000}))
```

We need a total of `LANES*CARRIERS` variables, which can be represented in a one-dimensional array. The variable that expresses how much is transported on lane `l` with carrier `c` has the index `l*LANES + c`. Under this assumption the constraints can be added. As the full problem matrix has `LANES*CARRIERS*(LANES+CARRIERS)` elements, the `linprog` function may not be suited for the problem size. You can increase the `maxiter` parameter, but you could run into other issues like numerical problems, though I did not read the source.

A faster and more robust free solver is bundled with `PuLP`. You can install `PuLP` with `easy_install pulp`. The problem can also be expressed in a more natural way, as `PuLP` has convenience functions for declaring variable dictionaries. While commercial solvers are faster than the one bundled with `PuLP`, your problem is a pure linear program and relatively "easy" even with 3000 lanes

and 6 carriers.

In PuLP it can be implemented in a more natural way:

```
from pulp import *
import numpy as np
from itertools import product

LANES = 30
CARRIERS = 6

cost = 100 * np.random.rand(LANES, CARRIERS) # c
demand = 10 * np.random.rand(LANES) # b_eq
capacity = [250, 300, 500, 750, 100, 200] # b_ub

prob = LpProblem("Transportation",LpMinimize)
x = LpVariable.dicts("Route", product(range(LANES),
range(CARRIERS)), 0, None)

prob += lpSum(cost[l, c] * x[l, c] for l in
range(LANES) for c in range(CARRIERS))

for l in range(LANES):
```

```
    prob += lpSum(cost[l, c] * x[l, c] for c in
range(CARRIERS)) == demand[l]

for c in range(CARRIERS):
    prob += lpSum(cost[l, c] * x[l, c] for l in
range(LANES)) <= capacity[c]

prob.solve()

# Get optimal solution
if LpStatus[prob.status] == "Optimal":
    x = {(l, c): value(x[l, c]) for l in range(LANES)
for c in range(CARRIERS)}
else:
    print("Optimization failed.")
```

answered Feb 21, 2018 at 22:58

TristanTristan

1,55688 silver badges1212 bronze badges

1

Not the answer you're looking for? Browse other questions tagged python numpy optimization scipy solver **or** ask your own question**.**