week05-00–restaurant-example-recap

February 20, 2025

1 George McNinch Math 87 - Spring 2025

2 Week 5 - Last week's Restaurant network flow example recapitulated

I'd like to recapitulate the "restaurant tablecloths" example, to show you a bit more of how we've represented the data describing the linear program that we use as parameters for the `linprog` function.

So: we need the objective function, a vector `c`.

And we need our inequality constraints, determined by a matrix `Aub` and a vector `bub`

And we need equality constraints (coming from the conservation laws). These are described by a matrix `Aeq` and a vector `beq` (in fact, `beq` is the zero vector of the right size).

We first import the code we used to solve `network flow` problems. This main function is `runNetworkFlow` which takes as arguments `vertices` and `edges`.

Here `vertices` should be a list of `strings`.

And `edges` should be a list of `python dictionaries` each of the form:

```
{ "from": "from vertex",
  "to":   "to vertex",
  "label": "edge label",
  "weight": 0,
  # "lower": 0,
  # "upper": np.inf
   }
```

where `lower` and or `upper` may be omitted.

My goal here is to show you how we can extract `c`, `Aub`, `bub`, etc. from the data `vertices` and `edges`. (Of course, this is precisely what the `runNetworkFlow` function does).

Here is the code, copied from the previous notebook:

[3]:
```
import numpy as np
from scipy.optimize import linprog
from pprint import pprint
```

```
float_formatter = "{:.2f}".format
np.set_printoptions(formatter={'float_kind':float_formatter})

import math


def sbv(index,size):
    return np.array([1.0 if i == index else 0.0 for i in range(size)])

# objective vector
def objective(edges):
    return sum([e["weight"]*sbv(edges.index(e),len(edges)) for e in edges])

def getIncoming(vertex,edges):
    return [ e for e in edges if e["to"] == vertex ]

def getOutgoing(vertex,edges):
    return [ e for e in edges if e["from"] == vertex ]

def isSource(vertex,edges):
    return getIncoming(vertex,edges) == []

def isSink(vertex,edges):
    return getOutgoing(vertex,edges) == []

def interiorVertices(vertices,edges):
    return [ v for v in vertices if not( isSource(v,edges) or isSink(v,edges) )␣
 ↪]

def conservationLaw(vertex,edges):
    ii = sum([ sbv(edges.index(e),len(edges)) for e in␣
 ↪getIncoming(vertex,edges) ])
    oo = sum([ sbv(edges.index(e),len(edges)) for e in␣
 ↪getOutgoing(vertex,edges) ])
    return ii - oo

def conservationMatrix(vertices,edges):
    return np.array([conservationLaw(v,edges) for v in␣
 ↪interiorVertices(vertices,edges) ])


def lowerBound(edge):
    if 'lower' in edge.keys():
        return edge['lower']
    else:
        return -math.inf
```

```
def upperBound(edge):
    if 'upper' in edge.keys():
        return edge['upper']
    else:
        return math.inf

def ineqConstraints(edges):
    m = np.array([*[ sbv(edges.index(e),len(edges))
                     for e in edges
                     if not upperBound(e) == math.inf ],
                  *[ -sbv(edges.index(e),len(edges))
                     for e in edges
                     if not lowerBound(e) == -math.inf ]
                 ])

    b = np.array([ *[ upperBound(e)
                      for e in edges
                      if not upperBound(e) == math.inf],
                   *[ -lowerBound(e)
                      for e in edges
                      if not lowerBound(e) == -math.inf]
                 ])

    return m,b

def reportEdge(edge):
    if "label" in edge.keys():
        return f"{edge['label']} ({edge['from']} --> {edge['to']})"
    else:
        return f"     ({edge['from']} --> {edge['to']})"


def runNetworkFlow(vertices,edges,maximize=False):
    obj = objective(edges)
    Aeq = conservationMatrix(vertices,edges)
    Aub,bub = ineqConstraints(edges)

    beq = np.zeros(len(interiorVertices(vertices,edges)))

    sgn = -1 if maximize else 1

    lr  = linprog(sgn*obj,
                  A_eq = Aeq,
                  b_eq = beq,
                  A_ub = Aub,
                  b_ub = bub
                  )
```

```
    if lr.success:
        optimal_value = sgn*lr.fun
        return [ f"optimal value: {optimal_value}" ] + [ (reportEdge(e),␣
↪float(lr.x[edges.index(e)])) for e in edges]
    else:
        print("Linear program failed")
```

And here is the code describing the `vertices` and `edges` for the restaurant example:

[2]:
```
# usage requirements

tt = {1: 10,
      2: 10,
      3: 15,
      4: 20,
      5: 40,
      6: 40,
      7: 30
     }


source = [ 'source' ]
cleanVert = [ f"day {n} clean" for n in range(1,8) ]
usedVert = [ f"day {n} used" for n in range(1,8) ]

# in python, addition of lists amounts to concatentation
# e.g. [1,2] + [3,4] = [1,2,3,4].
#
restaurant_vertices = source + cleanVert + usedVert


edges_from_source = [ {"from": 'source',
                       "to": f"day {n} clean",
                       'label': "purchase",
                       "weight": 5}
                       for n in range(1,8)
                    ]

edges_carryover = [ {"from": f"day {n} clean",
                     "to":   f"day {n+1} clean",
                     "label": "carryover",
                     "weight": 0
                    }
                    for n in range(1,7)
                    ]
```

```
edges_use = [ { "from": f"day {n} clean",
                "to":   f"day {n} used",
                "label": "tablecloth use",
                "weight": 0,
                "lower": tt[n]
                }
                for n in range(1,8)
            ]
edges_fast_laundry = [ { "from": f"day {n} used",
                         "to": f"day {n+1} clean",
                         "label": "fast laundry",
                         "weight": 2
                       }
                       for n in range(1,7)
                     ]

edges_slow_laundry = [ { "from": f"day {n} used",
                         "to":   f"day {n+2} clean",
                         "label": "slow laundry",
                         "weight": 1
                         }
                        for n in range(1,6)
                    ]

restaurant_edges = edges_from_source + edges_carryover + edges_use +␣
 ↪edges_fast_laundry + edges_slow_laundry
```

We keep the edges in the list `restaurant_edges`. We can see in what order the edges appear as follows:

[4]:
```
restaurant_vertices
```

[4]:
```
['source',
 'day 1 clean',
 'day 2 clean',
 'day 3 clean',
 'day 4 clean',
 'day 5 clean',
 'day 6 clean',
 'day 7 clean',
 'day 1 used',
 'day 2 used',
 'day 3 used',
 'day 4 used',
 'day 5 used',
 'day 6 used',
 'day 7 used']
```

[9]:
```
d  = { "name": "George",
       "address": "121 Oak st",
     }

[d["name"], d["address"] ]
```

[9]:
```
['George', '121 Oak st']
```

[5]:
```
my_vertices = [ "source", "day1c", "day2c" ]
```

[7]:
```
for e in restaurant_edges:
    print(f"{e['from']} -> {e['to']}")
```

```
source -> day 1 clean
source -> day 2 clean
source -> day 3 clean
source -> day 4 clean
source -> day 5 clean
source -> day 6 clean
source -> day 7 clean
day 1 clean -> day 2 clean
day 2 clean -> day 3 clean
day 3 clean -> day 4 clean
day 4 clean -> day 5 clean
day 5 clean -> day 6 clean
day 6 clean -> day 7 clean
day 1 clean -> day 1 used
day 2 clean -> day 2 used
day 3 clean -> day 3 used
day 4 clean -> day 4 used
day 5 clean -> day 5 used
day 6 clean -> day 6 used
day 7 clean -> day 7 used
day 1 used -> day 2 clean
day 2 used -> day 3 clean
day 3 used -> day 4 clean
day 4 used -> day 5 clean
day 5 used -> day 6 clean
day 6 used -> day 7 clean
day 1 used -> day 3 clean
day 2 used -> day 4 clean
day 3 used -> day 5 clean
day 4 used -> day 6 clean
day 5 used -> day 7 clean
```

And we can see the *objective function*:

[42]:
```
c=objective(restaurant_edges)
c
```

[42]:
```
array([5.00, 5.00, 5.00, 5.00, 5.00, 5.00, 5.00, 0.00, 0.00, 0.00, 0.00,
       0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 2.00, 2.00,
       2.00, 2.00, 2.00, 2.00, 1.00, 1.00, 1.00, 1.00, 1.00])
```

We can see the *conservation matrix*:

[43]:
```
## remember that the rows of the conservation matrix are determined by the␣
 ↪interior vertices
## so the length of the zero-vector veq should be the number of interior␣
 ↪vertices.

Aeq = conservationMatrix(restaurant_vertices,restaurant_edges)
beq = np.zeros(len(interiorVertices(restaurant_vertices,restaurant_edges)))
Aeq,beq
```

[43]:
```
(array([[1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00, 0.00,
         0.00, 0.00, 0.00, -1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
         0.00, 0.00, 0.00, 0.00, 0.00, 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, 1.00, -1.00, 0.00,
         0.00, 0.00, 0.00, 0.00, -1.00, 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, 0.00, 0.00, 0.00],
        [0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00, -1.00,
         0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00, 0.00, 0.00, 0.00,
         0.00, 1.00, 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, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00,
         -1.00, 0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00, 0.00, 0.00,
         0.00, 0.00, 1.00, 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, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00,
         -1.00, 0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00, 0.00, 0.00,
         0.00, 0.00, 1.00, 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, 0.00, 0.00, 0.00, 0.00, 0.00,
         1.00, -1.00, 0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00, 0.00,
         0.00, 0.00, 0.00, 1.00, 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, 0.00, 0.00, 0.00, 0.00,
         0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, -1.00, 0.00,
         0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 1.00],
        [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 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, -1.00,
         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, 0.00, 0.00, 0.00, 0.00, 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,
         -1.00, 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, 0.00, 0.00, 0.00, 0.00, 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,
         -1.00, 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, 0.00, 0.00, 0.00, 0.00, 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, -1.00, 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, 0.00, 0.00, 0.00, 0.00, 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, -1.00, 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, 0.00, 0.00, 0.00, 0.00,
         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, -1.00, 0.00, 0.00, 0.00, 0.00, 0.00]]),
 array([0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
        0.00, 0.00]))
```

[44]:
```
Aub,bub = ineqConstraints(restaurant_edges)
Aub,bub
```

[44]:
```
(array([[-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -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, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -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, -0.00, -0.00, -0.00, -0.00, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -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, -0.00, -0.00, -0.00, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00,
         -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, -0.00, -0.00, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00,
         -0.00, -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, -0.00, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00,
         -0.00, -0.00, -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, -0.00,
         -0.00, -0.00, -0.00, -0.00],
        [-0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00, -0.00,
         -0.00, -0.00, -0.00, -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,
         -0.00, -0.00, -0.00, -0.00]]),
 array([-10, -10, -15, -20, -40, -40, -30]))
```

And we could have just directly run `linprog` to solve the linear program, with these data:

[4]:
```
c = objective(restaurant_edges)
Aeq = conservationMatrix(restaurant_vertices,restaurant_edges)
Aub,bub = ineqConstraints(restaurant_edges)
beq = np.zeros(len(interiorVertices(restaurant_vertices,restaurant_edges)))

lr  = linprog(c,
              A_eq = Aeq,
              b_eq = beq,
              A_ub = Aub,
              b_ub = bub
              )
lr
```

[4]:
```
         message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
         success: True
          status: 0
             fun: 435.0
               x: [ 1.000e+01  1.000e+01 …  0.000e+00 -0.000e+00]
             nit: 21
           lower:  residual: [ 1.000e+01  1.000e+01 …  0.000e+00
                               -0.000e+00]
                  marginals: [ 0.000e+00  0.000e+00 …  3.000e+00
                               0.000e+00]
           upper:  residual: [       inf        inf …        inf
                                     inf]
                  marginals: [ 0.000e+00  0.000e+00 …  0.000e+00
                               0.000e+00]
           eqlin:  residual: [ 0.000e+00  0.000e+00 …  0.000e+00
                               0.000e+00]
                  marginals: [ 5.000e+00  5.000e+00 … -1.000e+00
                              -2.000e+00]
         ineqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                               0.000e+00  0.000e+00  1.000e+01]
                  marginals: [-1.000e+00 -1.000e+00 -1.000e+00 -2.000e+00
                              -6.000e+00 -3.000e+00 -0.000e+00]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

To get detailed information about the optimal values of the *variables* (associated with the edges) you must inspect the `x` field of the result from `linprog`. The report you see when you just examine the result elides the details – you need to examine the values themselves. This can be done as follows:

[5]:
```
lr.fun,lr.x
```

[5]:
```
(435.0,
 array([10.00, 10.00, 5.00, 15.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
```

```
5.00, 0.00, 0.00, 10.00, 10.00, 15.00, 20.00, 40.00, 40.00, 40.00,
0.00, 0.00, 0.00, 20.00, 40.00, 40.00, 10.00, 10.00, 15.00, 0.00,
-0.00]))
```

The point of this discussion is just to observe that you don't have to rely on describing the data via `vertices` and `edges`, you can simply type the vector `c`, the matrices `Aub` and `Aeq` and so on. In order to calculate those matrices, you'll be well-served by drawing the `network flow` diagram.

(Finally, notice that we get the same answer as `runNetworkFlow`:

[6]:
```
runNetworkFlow(restaurant_vertices,restaurant_edges)
```

[6]:
```
['optimal value: 435.0',
 ('purchase (source --> day 1 clean)', 10.0),
 ('purchase (source --> day 2 clean)', 10.0),
 ('purchase (source --> day 3 clean)', 5.0),
 ('purchase (source --> day 4 clean)', 15.0),
 ('purchase (source --> day 5 clean)', 0.0),
 ('purchase (source --> day 6 clean)', 0.0),
 ('purchase (source --> day 7 clean)', 0.0),
 ('carryover (day 1 clean --> day 2 clean)', 0.0),
 ('carryover (day 2 clean --> day 3 clean)', 0.0),
 ('carryover (day 3 clean --> day 4 clean)', 0.0),
 ('carryover (day 4 clean --> day 5 clean)', 5.0),
 ('carryover (day 5 clean --> day 6 clean)', 0.0),
 ('carryover (day 6 clean --> day 7 clean)', 0.0),
 ('tablecloth use (day 1 clean --> day 1 used)', 10.0),
 ('tablecloth use (day 2 clean --> day 2 used)', 10.0),
 ('tablecloth use (day 3 clean --> day 3 used)', 15.0),
 ('tablecloth use (day 4 clean --> day 4 used)', 20.0),
 ('tablecloth use (day 5 clean --> day 5 used)', 40.0),
 ('tablecloth use (day 6 clean --> day 6 used)', 40.0),
 ('tablecloth use (day 7 clean --> day 7 used)', 40.0),
 ('fast laundry (day 1 used --> day 2 clean)', 0.0),
 ('fast laundry (day 2 used --> day 3 clean)', 0.0),
 ('fast laundry (day 3 used --> day 4 clean)', 0.0),
 ('fast laundry (day 4 used --> day 5 clean)', 20.0),
 ('fast laundry (day 5 used --> day 6 clean)', 40.0),
 ('fast laundry (day 6 used --> day 7 clean)', 40.0),
 ('slow laundry (day 1 used --> day 3 clean)', 10.0),
 ('slow laundry (day 2 used --> day 4 clean)', 10.0),
 ('slow laundry (day 3 used --> day 5 clean)', 15.0),
 ('slow laundry (day 4 used --> day 6 clean)', 0.0),
 ('slow laundry (day 5 used --> day 7 clean)', -0.0)]
```

[ ]:
```

```