In [1]:
```
import numpy as np
import scipy.optimize as opt
```

In [2]:
```
# parameters
ca = np.array([1,3,2,4,5])
q = np.array([2,2])
b = (0,4)
bnds = (b,b,b,b,b)
f0 = np.array([1,1,1,1,1])
delta = np.array([[1,0,0,0,0],
                  [0,1,1,0,0],
                  [0,0,1,1,1],
                  [0,1,0,0,0],
                  [0,0,0,1,1]])
```

In [3]:
```
def obj(f):
    # link vs path
    xa = np.dot(f,delta)

    # objective function
    obj = np.sum(xa+0.03*ca*(xa/ca)**5)
    return obj

def con1(f):
    # path flow
    path_1 = f[0]+f[1]+f[2]-q[0]
    return path_1

def con2(f):
    path_2 = f[3]+f[4]-q[1]
    return path_2
```

In [4]:
```
cons = ({'type': 'eq', 'fun': con1},
        {'type': 'eq', 'fun': con2})
```

In [5]:
```
sol = opt.minimize(obj, constraints=cons, x0=f0, bounds=bnds)
```

In [6]:
```
path_flow = sol.x
```

In [7]:
```
link_flow = np.dot(sol.x,delta)
```

In [8]:
```
travel_time = np.dot((1+0.15*(link_flow/ca)**4), delta)
```

In [9]:
```
print('The path flow:')
for i in range(5):
    print('Path ' + str(i+1) + ': ' + str(round(path_flow[i],2)))
print('')
```

```
print('The link flow:')
for i in range(5):
    print('Link ' + str(i+1) + ': ' + str(round(link_flow[i],2)))
print('')
print('The travel time of each path:')
for i in range(5):
    print('Path ' + str(i+1) + ': ' + str(round(travel_time[i],2)))
```

```
The path flow:
Path 1: 1.63
Path 2: 0.37
Path 3: 0.0
Path 4: 2.0
Path 5: 0.0

The link flow:
Link 1: 1.63
Link 2: 2.37
Link 3: 0.37
Link 4: 0.0
Link 5: 0.0

The travel time of each path:
Path 1: 2.06
Path 2: 2.06
Path 3: 2.06
Path 4: 2.0
Path 5: 2.0
```

In [1]:
```
import numpy as np
import scipy.optimize as opt
```

In [2]:
```
# parameters
ca = np.array([1,3,2,4,5])
node_flow = np.array([2,-4,2,0])
xa0 = np.array([1.63, 2.37, 0.37, 0, 0])
b = (0,4)
bnds = (b,b,b,b,b)

# network
connection = np.array([[1,0,1,0,0],
                       [-1,-1,0,0,-1],
                       [0,1,-1,1,0],
                       [0,0,0,-1,1]])
connection = connection.T
```

In [3]:
```
def obj(xa):
    obj = np.sum(xa + 0.03*ca*(xa/ca)**5)
    return obj

def con1(xa):
    node = xa@connection - node_flow
    #node_2 = xa@connection - node_flow[1]
    return node
```

In [4]:
```
cons = [{'type':'eq', 'fun':con1}]
# cons = ({'type':'eq', 'fun': lambda x: x[0]+x[2]-2},
#         {'type':'eq', 'fun': lambda x: -x[0]-x[1]-x[4]+4},
#         {'type':'eq', 'fun': lambda x: -x[2]+x[1]+x[3]-2},
#         {'type':'eq', 'fun': lambda x: -x[3]+x[4]})
sol = opt.minimize(obj, constraints=cons, x0=xa0, bounds=bnds)
```

In [5]:
```
print('Flow on each link:')
for i in range(5):
    print('Link ' + str(i+1) + ': ' + str(round(sol.x[i],2)))
```

```
Flow on each link:
Link 1: 1.63
Link 2: 2.37
Link 3: 0.37
Link 4: 0.0
Link 5: 0.0
```

In [ ]:
```
```

In [1]:

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

In [2]:

```
# parameters
ca = np.array([1,3,2,4,5])
delta = np.array([[1,0,0,0,0],
                  [0,1,1,0,0],
                  [0,0,1,1,1],
                  [0,1,0,0,0],
                  [0,0,0,1,1]])
od_demand = np.array([2,2])

# Initialization
free_flow_time_link = np.ones(5)
free_flow_time_path = free_flow_time_link @ delta.T

min_path_ind = np.array([np.argmin(free_flow_time_path[:3]), np.argmin(free_flov
path_flow_0 = np.zeros(5)
for i in range(2):
    path_flow_0[min_path_ind[i]] = od_demand[i]

link_flow_0 = path_flow_0@delta

# Iteration 1
# step 1: Update travel time
updated_travel_time_link = 1+0.15*(link_flow_0/ca)**4
updated_travel_time_path = updated_travel_time_link @ delta.T

# step 2: Perform all-or-nothing assginment based on updated travel time
min_path_ind = np.array([np.argmin(updated_travel_time_path[:3]), np.argmin(upda
direction_path = np.zeros(5)
for i in range(2):
    direction_path[min_path_ind[i]] = od_demand[i]

direction_link = direction_path @ delta
movement = direction_link - link_flow_0

# step 3: Line search finding alpha
def obj(alpha):
    upper_bound = link_flow_0 + alpha*movement
    obj = np.sum(upper_bound + 0.03*(upper_bound/ca)**5)
    return obj

b = (0,1)
bnd = [b]
sol = opt.minimize(obj,bounds=bnd,x0=0)
alpha = sol.x

# step 4: compute new link flow
link_flow_1 = link_flow_0 + alpha*movement

# Iteration 2
# step 1: Update travel time
updated_travel_time_link = 1+0.15*(link_flow_1/ca)**4
updated_travel_time_path = updated_travel_time_link @ delta.T

# step 2: Perform all-or-nothing assignment based on updated travel time
```

```
min_path_ind = np.array([np.argmin(updated_travel_time_path[:3]), np.argmin(upda
direction_path = np.zeros(5)
for i in range(2):
    direction_path[min_path_ind[i]] = od_demand[i]

direction_link = direction_path @ delta
movement = direction_link - link_flow_1

# step 3: Line search finding alpha
def obj(alpha):
    upper_bound = link_flow_1 + alpha*movement
    obj = np.sum(upper_bound + 0.03*(upper_bound/ca)**5)
    return obj

b = (0,1)
bnd = [b]
sol = opt.minimize(obj,bounds=bnd,x0=0)
alpha = sol.x

# step 4: compute new link flow
link_flow_2 = link_flow_1 + alpha*movement
```

In [3]:

```
print('The link flow:')
for i in range(5):
    print('Link ' + str(i+1) + ': ' + str(round(link_flow_2[i],2)))
```

```
The link flow:
Link 1: 1.61
Link 2: 2.39
Link 3: 0.39
Link 4: 0.0
Link 5: 0.0
```

Problem 2: Steepest Descent Method

$\min \nabla f(x) = 4(x_1-10)^2 + (x_2-4)^2$, $x^{(0)} = (0,0)$

Iteration 1

$\nabla f(x) = [8(x_1-10), 2(x_2-4)]^T$, therefore $d^{(0)} = -\nabla f(x^{(0)}) = [80, 8]^T$

Find a step size $\alpha^{(0)}$ such that $\min f(x^{(0)} + \alpha^{(0)} \cdot d^{(0)})$

$x^{(0)} + \alpha^{(0)} \cdot d^{(0)} = [0, 0]^T + \alpha^{(0)}[80, 8]^T = [80\alpha^{(0)}, 8\alpha^{(0)}]^T$

$\min\limits_{\alpha} f(\alpha) = 4(80\alpha^{(0)} - 10)^2 + (8\alpha^{(0)} - 4)^2$

$= 400(8\alpha^{(0)} - 1)^2 + 16(2\alpha^{(0)} - 1)^2$

$df(\alpha)/d\alpha = 6400(8\alpha^{(0)} - 1) + 64(2\alpha^{(0)} - 1) = 0$

$\Rightarrow 64 \cdot 802 \cdot \alpha^{(0)} = 64 \cdot 101 \Rightarrow \alpha^{(0)} = \frac{101}{802} \doteq 0.126$

$\Rightarrow x^{(1)} = x^{(0)} + \alpha^{(0)} \cdot d^{(0)} = [0, 0]^T + 0.126[80, 8]^T = [10.08, 1.008]^T$

Iteration 2

$\nabla f(x) = [8(x_1-10), 2(x_2-4)]^T$, $d^{(1)} = -\nabla f(x^{(1)}) = [-0.64, 5.984]^T$

Find a step size $\alpha^{(1)}$ such that $\min f(x^{(1)} + \alpha^{(1)} d^{(1)})$

$x^{(1)} + \alpha^{(1)} d^{(1)} = [10.08, 1.008]^T + \alpha^{(1)}[-0.64, 5.984]^T$

$= [10.08 - 0.64\alpha^{(1)}, 1.008 + 5.984\alpha^{(1)}]^T$

$\min\limits_{\alpha} f(\alpha) = 4(0.08 - 0.64\alpha^{(1)})^2 + (-2.992 + 5.984\alpha^{(1)})^2$

$df(\alpha)/d\alpha = -5.12(0.08 - 0.64\alpha^{(1)}) + 11.968(-2.992 + 5.984\alpha^{(1)}) = 0$

$\Rightarrow 74.893\alpha^{(1)} = 36.218 \Rightarrow \alpha^{(1)} = 0.484$

$\Rightarrow x^{(2)} = x^{(1)} + \alpha^{(1)} \cdot d^{(1)} = [10.08, 1.008]^T + 0.484[-0.64, 5.984]^T$

$= [9.77, 3.904]$

$f(x^{(2)}) = 4 \times (0.23)^2 + (0.096)^2 = 0.221$ ■