基礎ゼミ⑬

Frank Wolfeアルゴリズムの実装 (Python)

2017/06/28

朝倉研究室M1　尾頭尚人

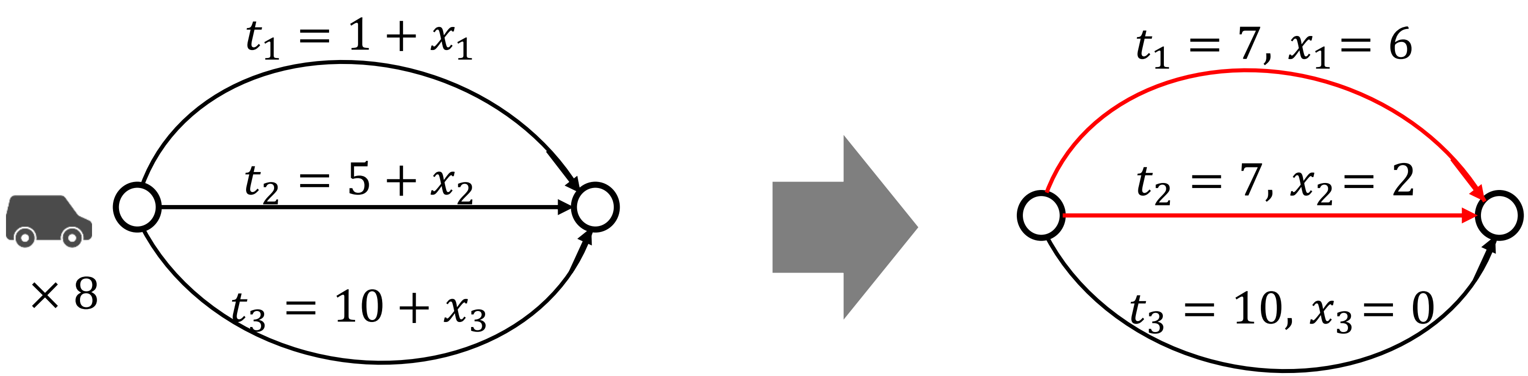
目次

- 前回の復習(利用者均衡)
- Frank Wolfe法
- Frank Wolfe法の実装

(復習)利用者均衡

Wardropの第一原則

利用される経路の旅行時間は全て等しく、利用されない経路の旅行時間よりも小さいか、せいぜい等しい



利用者均衡配分

（x: リンク交通量　t: リンク走行時間）

(復習)利用者均衡

利用者均衡時の各リンクの交通量は
以下の**数理最適化問題**の解として求められる

(交通ネットワークの均衡分析:p.42を参照)

【目的関数】

$$\min Z(x_a) = \sum_{a \in A} \int_0^{x_a} t_a(w)dw$$

各リンクの所要時間の総和を最小にする

【制約条件】

$$\sum_{k \in K_{rs}} f_k^{rs} - Q_{rs} = 0$$

$$x_a = \sum_{k \in K_{rs}} \sum_{rs \in} \delta_{a,k}^{rs} f_k^{rs}$$

交通量保存則
(OD交通量、リンク交通量)

$$f_k^{rs} \geq 0, \qquad x_a \geq 0$$

非負制約
(経路交通量、リンク交通量)

Frank Wolfe法

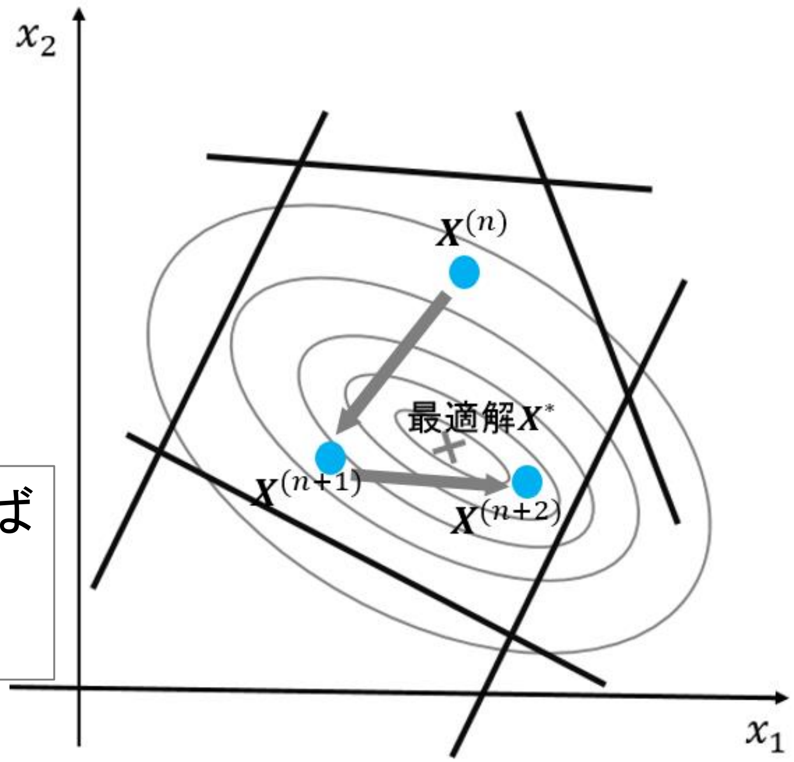
- p.4の数理最適化問題を解く代表的なアルゴリズム

非線形最適化問題の
一般的解法に沿っている

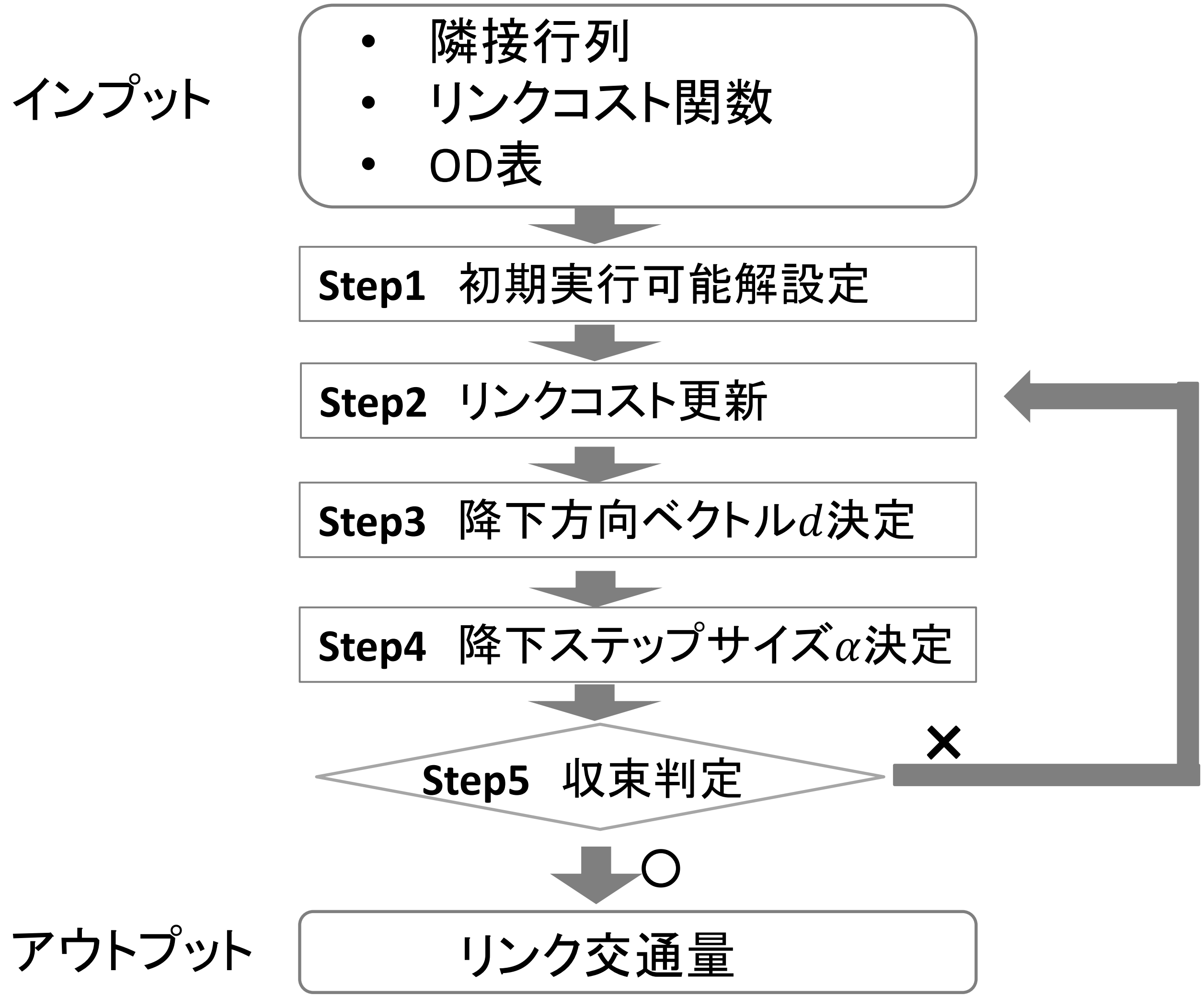
$$\boldsymbol{X}^{(n+1)} = \boldsymbol{X}^{(n)} + \alpha \cdot \boldsymbol{d}$$

どのくらい進めるか
(ステップサイズ)

どの方向に向かえば
最適解に近づくか
(降下方向ベクトル)



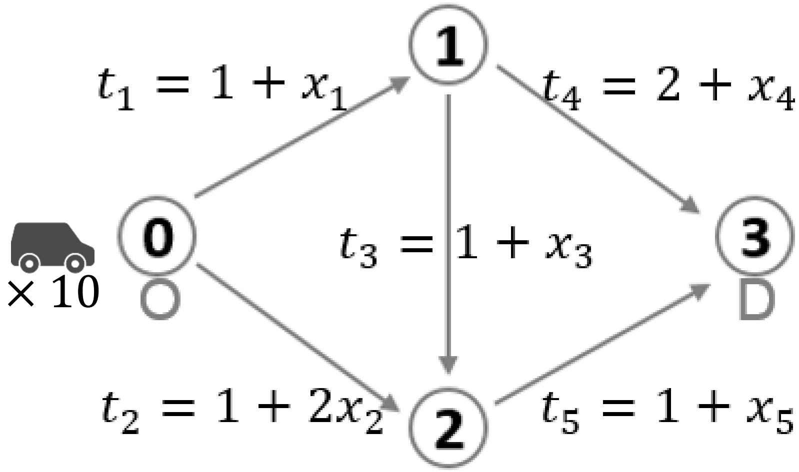
Frank Wolfe法



Frank Wolfe法

例
ノード0からノード3へ10の車が走行

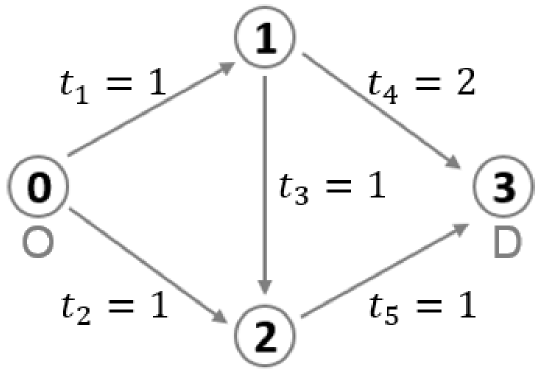
配分量(リンク交通量)をFrank Wolfe法で求める



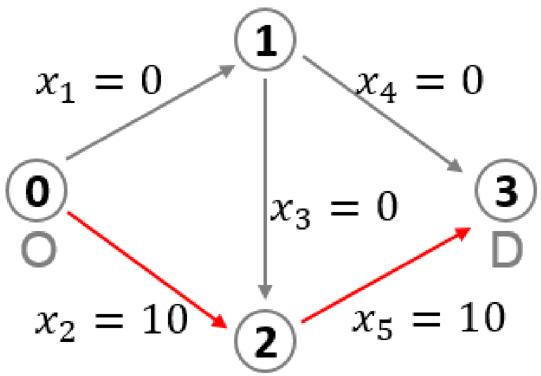
Frank Wolfe法

Step1 初期実行可能解設定

①すべてのリンク交通量を0とした時の各リンクの所要時間を求める



②OD間の最短経路を求め, そこにOD交通量をすべて流す(all or nothing配分)



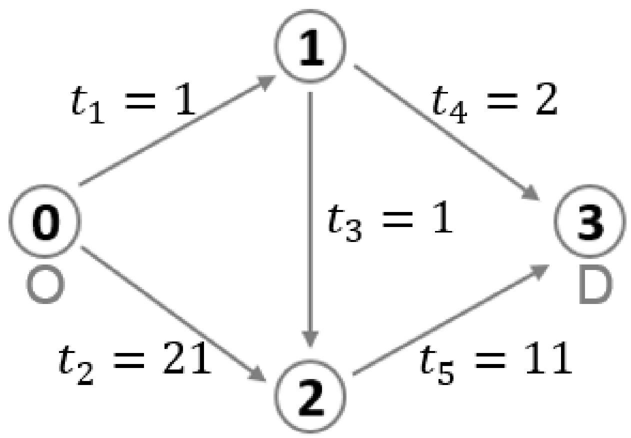
③②によりリンク交通量 $\boldsymbol{X}^{(1)}$が決まる

$$\boldsymbol{X}^{(1)} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 10 \\ 0 \\ 0 \\ 10 \end{pmatrix}$$

Frank Wolfe法

Step2　リンクコスト更新

$\boldsymbol{X}^{(1)}$に対する、各リンクの所要時間を求める

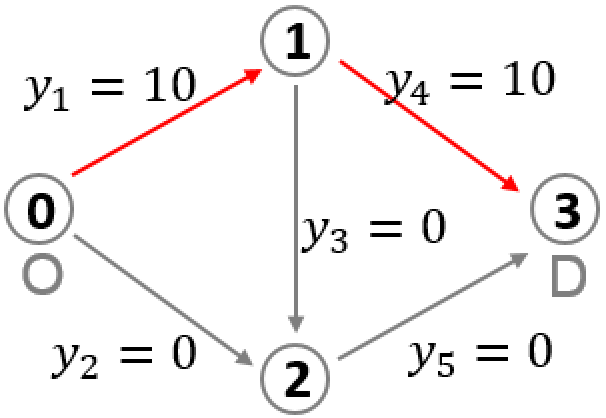


Frank Wolfe法

Step3　降下方向ベクトルd決定

①OD間の最短経路を求め, そこに

　OD交通量をすべて流す(all or nothing配分)



②①の配分量のベクトルを$\boldsymbol{y}^{(1)}$とする

③降下方向ベクトル$\boldsymbol{d}^{(1)}$は以下となる

$$\boldsymbol{d}^{(\mathbf{1})} = \boldsymbol{y}^{(1)} - \boldsymbol{x}^{(1)} = \begin{pmatrix} 10 \\ 0 \\ 0 \\ 10 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 10 \\ 0 \\ 0 \\ 10 \end{pmatrix} = \begin{pmatrix} 10 \\ -10 \\ 0 \\ 10 \\ -10 \end{pmatrix}$$

Frank Wolfe法

Step4　降下ステップサイズα決定

①リンク交通量は次式で更新

$$\boldsymbol{X}^{(2)} = \boldsymbol{X}^{(1)} + \alpha \boldsymbol{d}^{(1)} = \begin{pmatrix} 10\alpha \\ 10 - 10\alpha \\ 0 \\ 10\alpha \\ 10 - 10\alpha \end{pmatrix}$$

②これをp.4の目的関数に代入

$$Z\left(\boldsymbol{X}^{(2)}\right) = \boxed{\sum_{a \in A} \int_0^{x_a^{(1)} + \alpha d_a^{(1)}} t_a(w)dw}$$

$\alpha(0 \leq \alpha \leq 1)$の関数

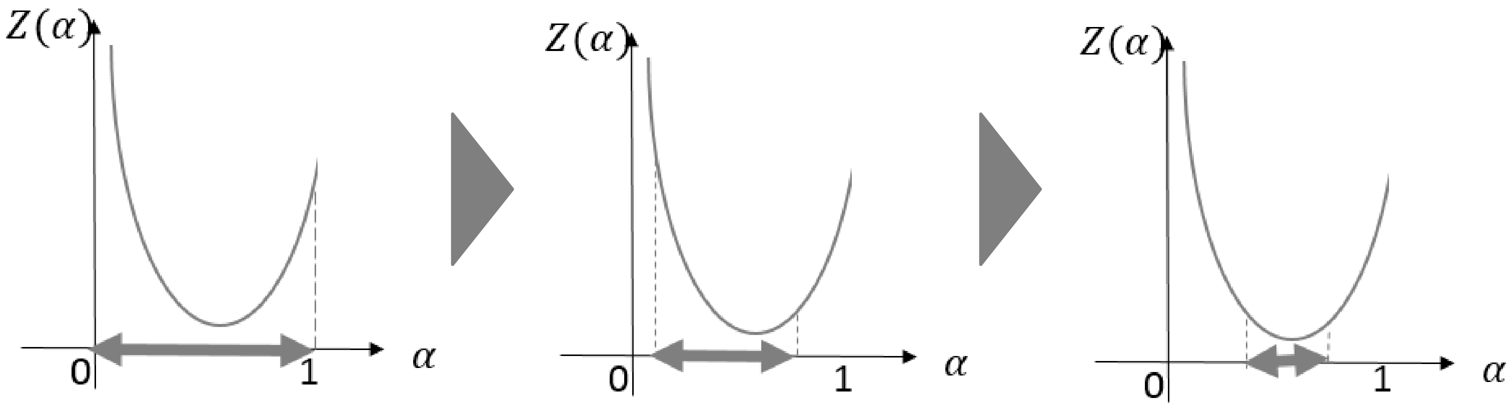
一次元探索アルゴリズム(黄金分割法)を用いて、
Zを最小にするαを求める (α=0.58となる)

Frank Wolfe法

※黄金分割法

一変数関数のある区間内の最小値を、

黄金比を利用して区間を狭めていくことで求める方法



Frank Wolfe法

Step5　収束判定

α=0.58より、リンク交通量を次式で更新

$$\boldsymbol{X}^{(2)} = \boldsymbol{X}^{(1)} + \alpha \boldsymbol{d}^{(1)} = \begin{pmatrix} 5.8 \\ 4.2 \\ 0 \\ 5.8 \\ 4.2 \end{pmatrix}$$

次の収束判定を満たすとき、計算終了

$$\max_{a} \left| \frac{x_a^{(2)} - x_a^{(1)}}{x_a^{(1)}} \right| \leq 0.001$$

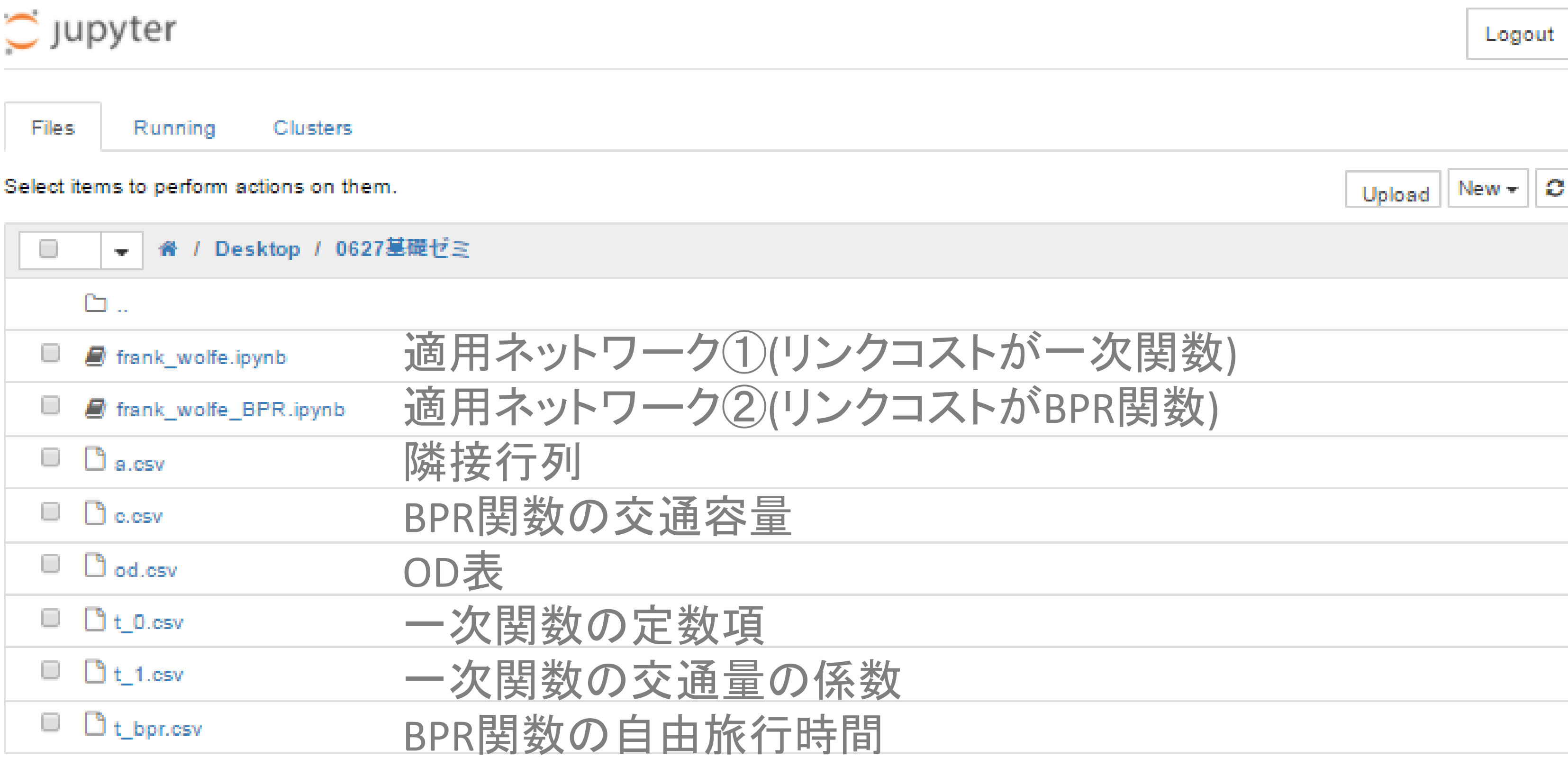
更新前後における、
リンク交通量の変化で判断
(更新前の交通量に対する相対的な変化がかなり小さければ計算終了)

満たさないとき、再びStep2に戻って繰り返す

Frank Wolfe法の実装

◆IpythonでFrank Wolfe法

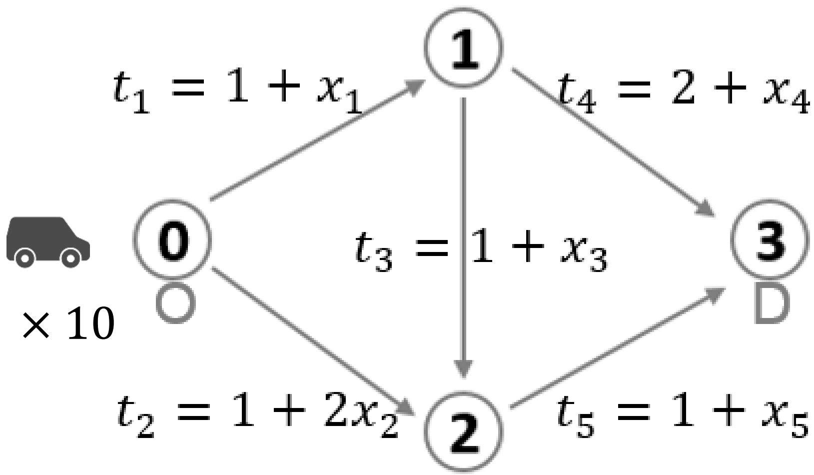
- コマンドプロンプトでipython notebook
- 配布した　0627基礎ゼミ　を選択



Frank Wolfe法の実装

- ✓ frank_wolfe.ipynb を選択

適用ネットワーク①(リンクコストが一次関数の場合)



```python
# coding: shift_jis
import numpy as np
from numpy import genfromtxt
import networkx as nx
from scipy import integrate
import matplotlib.pyplot as plt

print 'start'


def frank_wolfe(a,t_0,t_1,od):

    """
    a: adjacency matrix(隣接行列)
    t_0,t_1: coefficient of link cost function(1次のリンクコスト関数の係数)
    od: OD matrix(OD表)
    """
    x = np.zeros([len(a[0]),len(a[0])])#link flow matrix (initial value)
    t = t_0 + t_1*x#link cost

    G = nx.DiGraph(t)
    for i in range(a.shape[0]):
        for j in range(a.shape[0]):
            if od[i,j] != 0:
                shortest_path = nx.dijkstra_path(G,i,j,'weight')
                for k in range(len(shortest_path)-1):
                    x[shortest_path[k], shortest_path[k+1]] += od[i,j]


    n = 1#Repeat number of calculations
    while True:
        """
        Decision on descent direction vector(降下方向ベクトル決定)
        """
        t = t_0 + t_1*x
        G = nx.DiGraph(t)
        y = np.zeros([len(a[0]),len(a[0])])#link flow based on all or nothing (initial value)

        for i in range(a.shape[0]):
            for j in range(a.shape[0]):
                if od[i,j] != 0:
                    shortest_path = nx.dijkstra_path(G,i,j,'weight')
                    for k in range(len(shortest_path)-1):
                        y[shortest_path[k], shortest_path[k+1]] += od[i,j]

        d = y-x#descent direction vector
```

ライブラリのインポート
(numpy, matplotlib, networkx)

Frank Wolfe法
開始

Step1 初期実行可能解設定

$\boldsymbol{X}^{(1)}$を与える

繰り返し処理
開始

Step2 リンクコスト更新

Step3
降下方向ベクトル$d^{(n)}$ 決定

```
"""
Decision on descent step size by Golden section method(黄金分割法による降下ステップサイズ決定)
"""
s = (5.0**0.5-1)/2#Golden ratio(黄金比)
l = 0.0#Initial lower limit value(初期下限値)
m = 1.0#Initial upper limit value(初期上限値)
g = m-s*(m-l)
h = l+s*(m-l)

z_g = 0
z_h = 0

for i in range(a.shape[0]):
    for j in range(a.shape[0]):
        if a[i,j] != 0:
            def z(a_n):
                z = integrate.quad(lambda w: t_0[i,j] + t_1[i,j]*w,
                                   0.0,a_n*y[i,j] + (1-a_n)*x[i,j])#目的関数zはa_nの関数
                return z
            z_g += z(g)[0]
            z_h += z(h)[0]

###Repeating Golden select method
while True:
    if m-l < 0.001:
        a_n = (l+m)/2.0#Decision on decent step size(収束判定を満たすとき降下ステップサイズa_n決定)
        break

    else:
        if z_g >= z_h:
            l = g
            g = h
            m = m
            h = l+s*(m-l)
            z_g = z_h
            z_h = 0.0
            for i in range(a.shape[0]):
                for j in range(a.shape[0]):
                    if a[i,j] != 0:
                        def z(a_n):
                            z = integrate.quad(lambda w: t_0[i,j] + t_1[i,j]*w,
                                               0.0,a_n*y[i,j] + (1-a_n)*x[i,j])
                            return z
                        z_h += z(h)[0]
        elif z_g <= z_h:
            l = l
            m = h
            h = g
            g = m-s*(m-l)
            z_h = z_g
            z_g = 0.0
            for i in range(a.shape[0]):
                for j in range(a.shape[0]):
                    if a[i,j] != 0:
```

Step4
ステップサイズα決定
(黄金分割法を利用)

```python
                        for j in range(a.shape[0]):
                            if a[i,j] != 0:
                                def z(a_n):
                                    z = integrate.quad(lambda w: t_0[i,j] + t_1[i,j]*w,
                                                       0.0,a_n*y[i,j] + (1-a_n)*x[i,j])
                                    return z
                                z_g += z(g)[0]
        """
        Updating the value of x(xを更新)
        """
        x_2 = x + a_n*d
```

Step4

リンク交通量を更新

$$x^{(n+1)} = x^{(n)} + \alpha d^{(n)}$$

```python
        """
        Convergence Criteria(収束判定)
        """
        syusoku_list = []

        for i in range(a.shape[0]):
            for j in range(a.shape[0]):
                if x[i,j] != 0:
                    syusoku = np.absolute((x_2[i,j]-x[i,j])/x[i,j])
                    syusoku_list.append(syusoku)
        print max(syusoku_list)
        if max(syusoku_list) < 0.001: #収束条件
            break
        else:
            x = x_2
            n += 1
    return x

x = frank_wolfe(genfromtxt('a.csv', delimiter=',')
                ,genfromtxt('t_0.csv', delimiter=',')
                ,genfromtxt('t_1.csv', delimiter=',')
                ,genfromtxt('od.csv', delimiter=','))

print x


print 'end'
```

Step5
収束判定

収束判定を満たせば、
繰り返し計算終了

インプットデータ
隣接行列，リンクコスト関数，OD表

Frank Wolfe法の実装

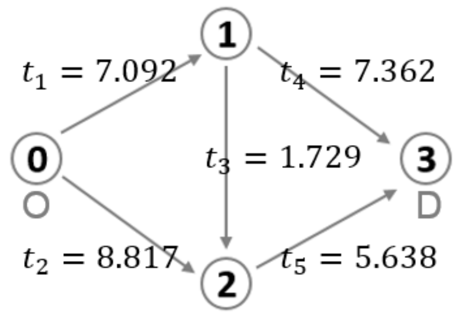
結果

繰り返し計算5回で、均衡時のリンク交通量が求まる

```
start
0.580119946268
0.100595283918
0.00429662280464
0.00468192215199
0.000366568717929

リンク所要時間
[[ 0.          7.09161601  8.81676798  0.        ]
 [ 0.          0.          1.72949525  7.36212076]
 [ 0.          0.          0.          5.63787924]
 [ 0.          0.          0.          0.        ]]

リンク交通量
[[ 0.          6.09161601  3.90838399  0.        ]
 [ 0.          0.          0.72949525  5.36212076]
 [ 0.          0.          0.          4.63787924]
 [ 0.          0.          0.          0.        ]]
end
```



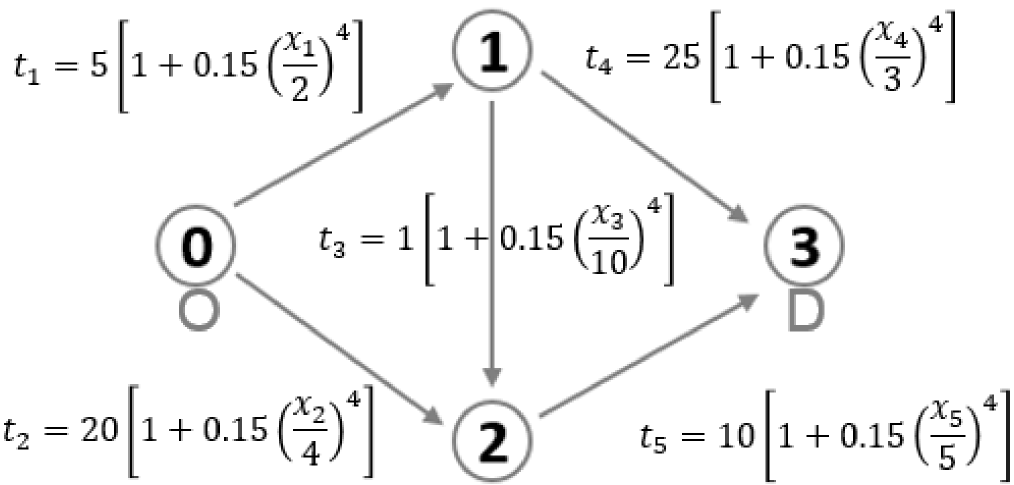
所要時間

- 経路0→1→3
  $t_1 + t_4 = 14.454$
- 経路0→2→3
  $t_2 + t_5 = 14.455$
- 経路0 → 1 → 2 → 3
  $t_1 + t_3 + t_5 = 14.459$

Frank Wolfe法の実装

✓frank_wolfe_BPR.ipynb　を選択

適用ネットワーク②(リンクコストがBPR関数の場合)



まとめ

本日のゼミでは、

- 利用者均衡とその数理最適問題
- Frank Wolfe法の具体的なアルゴリズム
- Frank Wolfe法の適用

  (リンクコスト関数が1次関数・BPR型の場合)

について扱った