第10回基礎ゼミ

Exercising UE with Python

平成30年6月1日(金)
朝倉研究室　修士1年　小池卓武

利用者均衡配分とベクトル･リンクコスト関数

プログラミングに移る前に…

今までは..リンクコストはそのリンク交通量のみに依存すると仮定..

$$t_a = t_a(x_a)$$

しかし…！！

リンクコストは他のリンク交通量にも影響を受けるのでは？
(例えば交差点遅れ，バスと乗用車が混在する車線等..)

$$t_a = t_a(x_1, \dots x_a, \dots x_L)\ \forall \mathrm{a} \in A$$

↓一般化すると

$$\boldsymbol{t}(\boldsymbol{x}): R_+^L \to R_+^L \qquad R_+^L: \text{非負実数空間}$$

上記のようなベクトル・リンクコスト関数(リンクコスト写像)を持つネットワーク上での利用者均衡を議論したい…！

利用者均衡配分とベクトル･リンクコスト関数

リンクコストが複数の交通量に影響を受ける..
いきなり難しい...
→<u>経路コストと経路交通量に限定</u>して考えてみる.

①非線形相補性問題$(UE/FD-NCP)$

利用者均衡条件

$$f_k^{rs} \cdot (C_k^{rs}(\boldsymbol{f}) - u_{rs}) = 0 \qquad k \in K_{rs} \quad \forall \mathrm{rs} \in W$$

$$\left.\begin{array}{l} C_k^{rs}(\boldsymbol{f}) - u_{rs} = 0 \leftrightarrow f_k^{rs} > 0 \\ C_k^{rs}(\boldsymbol{f}) - u_{rs} > 0 \leftrightarrow f_k^{rs} = 0 \end{array}\right\} \; C_k^{rs}(\boldsymbol{f}) - u_{rs} \geqq 0 \quad f_k^{rs} \geqq 0 \quad (1)$$

フロー保存条件

$$u_{rs} \cdot \left(\sum_{k \in K_{rs}} f_k^{rs} - q_{rs} \right) = 0$$

$$\left.\begin{array}{l} \sum_{k \in K_{rs}} f_k^{rs} - q_{rs} = 0 \leftrightarrow u_{rs} > 0 \\ \sum_{k \in K_{rs}} f_k^{rs} - q_{rs} > 0 \leftrightarrow u_{rs} = 0 \end{array}\right\} \; \sum_{k \in K_{rs}} f_k^{rs} - q_{rs} \geqq 0 \quad u_{rs} \geqq 0 \quad (2)$$

(1)(2)は相補性形式&変数$(\boldsymbol{f}, \boldsymbol{u})$の許容領域が非負実数空間$R_+^K \times R_+^M$(要は負にならない)

$(UE/FD-NCP)$

$Find\ \boldsymbol{X} = (\boldsymbol{f}, \boldsymbol{u}) \in R_+^K \times R_+^M$

$such\ that \quad \boldsymbol{X} \cdot \mathrm{F}(\mathbf{X}) = 0, \boldsymbol{X} \geqq 0\ \mathrm{F}(\boldsymbol{X}) \geqq 0$

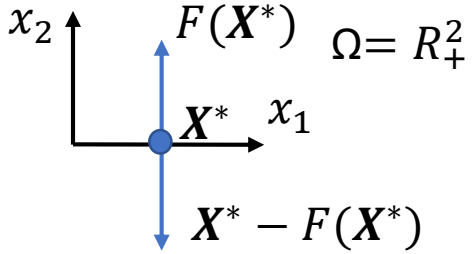
利用者均衡配分とベクトル・リンクコスト関数

②変分不等式問題$(UE/FD-VIP)$

$$Find\ \boldsymbol{x} \in \Omega \quad such\ that \quad \mathrm{F}(\boldsymbol{x}) \cdot (\boldsymbol{y}-\boldsymbol{x}) = 0,\ \forall \boldsymbol{y} \in \Omega$$

$\Omega \to R_+^n$とすれば,
VIPはNCPと等価問題となる

(R_+^n：非負実数空間)



ベクトル場Fと凸集合Ωが直行する点を求める問題に同義
(VIPはNCPを特殊ケースとして含んだ一般的枠組みと言える)

$$(UE/FD-VIP)$$
$$Find\ \boldsymbol{X}^* = (\boldsymbol{f}^*, \boldsymbol{u}^*) \in R_+^K \times R_+^M$$
$$such\ that \quad \mathrm{F}(\boldsymbol{X}^*) \cdot (\boldsymbol{X}-\boldsymbol{X}^*) \geqq 0,\ \forall \boldsymbol{X} \in \Omega$$

詳しく書くと…

$$\sum_{rs}\sum_{k}(C_k^{rs}(\boldsymbol{f}^*) - u_{rs}^*)(f_k^{rs} - f_k^{rs*}) + \sum_{rs}(\sum_{k} f_k^{rs*} - q_{rs}) \cdot (u_{rs} - u_{rs}^*) \geqq 0$$
$$\forall (\boldsymbol{f}, \boldsymbol{u}) \in R_+^K \times R_+^M$$

利用者均衡配分とベクトル・リンクコスト関数

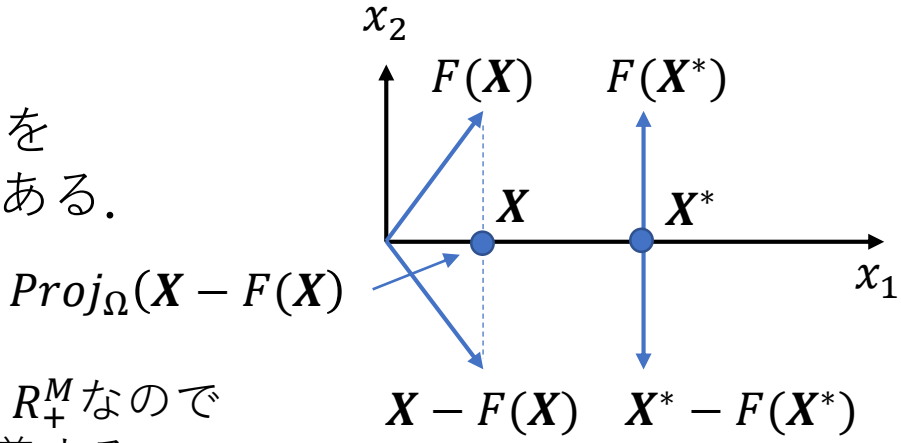
③不動点問題$(UE/FD-FPP)$

$$Find\ \boldsymbol{x} \in \Omega \quad such\ that \quad \boldsymbol{x} = Proj_{\Omega,Q}(\boldsymbol{x} - Q^{-1}F(\boldsymbol{x}))$$

※$Proj_{\Omega,Q}\boldsymbol{x} = arg.min_z.\{(z-x)\cdot Q(z-x) \quad s.t.\ z \in \Omega\}$

→凸集合Ω上への$\boldsymbol{x}$の正射影

$FPP(F,\Omega)$と$F(\boldsymbol{X})$がΩに直交している点を
求める問題であり，$VIP(F,\Omega)$と等価である.



利用者均衡問題は集合Ωが非負実数空間$R_+^K \times R_+^M$なので
集合Ωへの射影は，以下のような演算子に帰着する

$$Proj_{R_+^K \times R_+^M,Q}\left(\boldsymbol{x} - F(\boldsymbol{x})\right) = [\boldsymbol{x} - Q^{-1}F(\boldsymbol{x})]$$

例えば非負実数空間$\Omega = R_+^2$
においてVIPとFPPは等価

$$(UE/FD-FPP)$$
$$Find \boldsymbol{X} = (\boldsymbol{f},\boldsymbol{u}) \in R_+^K \times R_+^M$$
$$such\ that \quad \boldsymbol{X} = [\boldsymbol{X} - Q^{-1}F(\boldsymbol{X})]$$

リンク間相互干渉のある利用者均衡問題

不等変分式問題(VIP)としての表現方法

$\boldsymbol{f}$および$\boldsymbol{f}^*$がフロー保存条件と非負条件を満たした凸領域Ω_{p0}に属するとする

$$\sum_{k \in K_{rs}} f_k^{rs} - q_{rs} = 0 \quad \forall \mathrm{rs} \in W$$

$$f_k^{rs} > 0 \quad k \in K_{rs} \quad \forall \mathrm{rs} \in W$$

$UE/FD-VIP$より，下記のVIP式の第２項が消去され，

$$\sum_{rs}\sum_{k}(C_k^{rs}(\boldsymbol{f}^*) - u_{rs}^*)(f_k^{rs} - f_k^{rs*}) + \sum_{rs}(\sum_{k} f_k^{rs*} - q_{rs}) \cdot (u_{rs} - u_{rs}^*) \geqq 0$$

（$\sum_{k} f_k^{rs*} - q_{rs}$ → 0）

$$\sum_{rs}\sum_{k} u_{rs}^* \, (f_k^{rs} - f_k^{rs*}) = \sum_{rs} u_{rs}(q_{rs} - q_{rs}) = 0$$

したがって，経路交通量パターンf_k^{rs*}が$UE/FD-VIP$の解ならば下記が成立

$$\sum_{rs}\sum_{k} C_k^{rs}(\boldsymbol{f}^*) \, (f_k^{rs} - f_k^{rs*}) \geqq 0 \qquad \boldsymbol{f}, \boldsymbol{f}^* \in \Omega_{p0}$$

$\boldsymbol{f}^*$がこの式の解ならば$\boldsymbol{f}^*$は利用者均衡解！

リンク間相互干渉のある利用者均衡問題

ここで，$\boldsymbol{f}^*$が利用者均衡解ではないと仮定してみると..

$$f_k^{rs*} > 0,\ \ C_k^{rs}(\boldsymbol{f}^*) > C_l^{rs}(\boldsymbol{f}^*)$$

となる経路k, lが存在する．経路kの交通量を経路lに移すと，コストは$C_k^{rs}(\boldsymbol{f}^*) - C_l^{rs}(\boldsymbol{f}^*) > 0$だけ小さくなり，下記の式が成り立つ．

$$\sum_{rs}\sum_{k} C_k^{rs}(\boldsymbol{f}^*)\cdot f_k^{rs*} < \sum_{rs}\sum_{k} C_k^{rs}(\boldsymbol{f}^*) f_k^{rs*}$$

↑ $\sum_{rs}\sum_{k}(C_k^{rs}(\boldsymbol{f}^*) - u_{rs}^*)(f_k^{rs} - f_k^{rs*}) \geqq 0$が成立しないため，
利用者均衡配分問題は変分不等式問題$VIP(c, \Omega_{p0})$
と等価できる．

リンク間相互干渉のある利用者均衡問題

経路コストからリンクコストへの適用例

$$C_k^{rs} = \sum_{a \in A} t_a \delta_{a,k}^{rs}$$

経路コストがリンクコストと加算的関係の時,
ネットワーク総所要費用は全経路についての足し合わせなので

$$\sum_{rs} \sum_{k} C_k^{rs}(\boldsymbol{f}) \cdot f_k^{rs} = \sum_{a \in A} t_a(x) \cdot x_a$$

この関係を変分不等式問題$VIP(c, \Omega_{p0})$に適応すれば，以下のVIPが得られる.

$(UE/FD - VI - Priminal)$

$$Find\ \boldsymbol{x}^*\ such\ that \sum_{a \in A} t_a(\boldsymbol{x}^*) \cdot (x_a - x_a^*) \geqq 0 \qquad \boldsymbol{x} \in \Omega_p$$

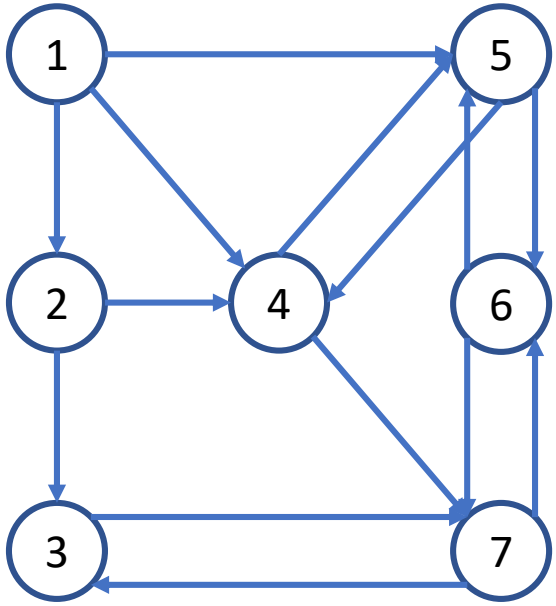
※Ω_pは経路コストとリンクコストの関係式，フロー保存条件,
非負条件を満足したリンク交通量ベクトルの集合

ここからが確定的UEのプログラミングになります！
Jupyter notebook を起動してください！

(※先ほどまでのベクトル・リンクコストではなく,
今回はノーマルなリンクコストでコードを作っています)

使用データの概要

ネットワークデータはcsvファイルとして配布



入ノード 出ノード

Init_node	Term_node	Capacity	Free_Flow	alpha	beta
1	2	10	50	0.15	4
1	4	20	40	0.15	4
1	5	10	50	0.15	4
2	3	20	50	0.15	4
2	4	60	10	0.15	4

交通容量Cap_a

自由走行時間t_a^0

形状パラメータα, β

↓BPR関数の定数として用いられる

$$t_a(x_a) = t_a^0 \left\{ 1 + \alpha \left(\frac{x_a}{Cap_a} \right)^{\beta} \right\}$$

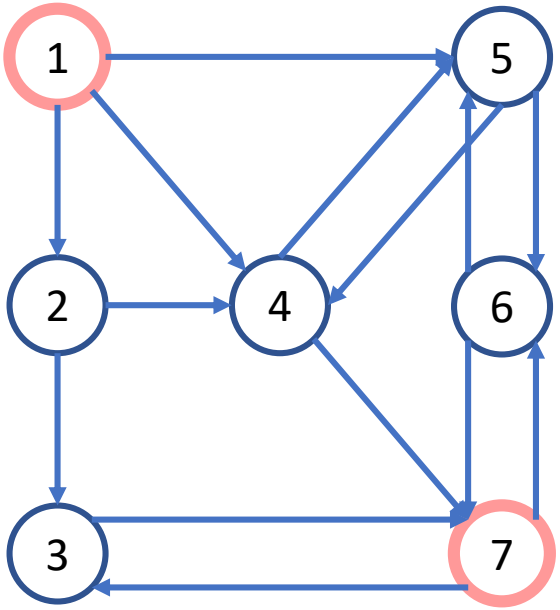
※入ノード，出ノードはODではなくリンクの入口，出口の意味！

使用データの概要

出発地　目的地　OD交通量

origin	destinatio	OD_vol
1	7	100
5	3	100
2	6	100
5	2	100

← ODデータも
CSVファイルとして配布



UEコード


```python
import csv
import copy
import networkx as nx

# CSVファイルの読み込み
def read_csv(file_name):
    data = []
    f = open(file_name, 'r', encoding="utf-8")
    reader = csv.reader(f)
    header = next(reader)
    for row in reader:
        data.append(row)
    f.close()
    return data

# 配分結果の書き込み
def write_res(file_name, data):
    header = ['Init node', 'Term node', 'link_vol']
    f = open(file_name, 'w')
    writer = csv.writer(f)
    writer.writerow(header)
    for row in data:
        writer.writerow(row[:3])
    f.close()
```

コードで用いるモジュールを
事前にインポート

csv ： csvファイルの読み書きに必要
$copy$ ：データをコピーする関数を内蔵
$networkx$ ：ネットワークを扱う際に便利

csvファイル(ネットワークデータ， ODデータ)
を読み込む関数

UEの配分結果をファイルに出力する関数

def _関数名(引数):

　　算術式

　　$return$ _返り値

自身で関数を定義！

(引数)は()でも実行可能..関数のタイプによる.

多くの関数は$return$で目的関数値を返す.

UEコード


```
# リンク交通量格納配列
def create_linkvol(data):
    link_vol = []
    for x in data:
        link_vol.append([x[0], x[1], 0, 0, 0, 0])
    return link_vol

# グラフデータの作成

def create_graph(data):
    network = nx.DiGraph()
    for x in data:
        network.add_edge(x[0], x[1], cap = float(x[2]),
                         FFcost = float(x[3]), weight = float(x[3]),
                         alpha = float(x[4]), beta = float(x[5]))
    return network
```

UEの配分結果の出力先を設定.
各リンクを構成する入ノード$x[0]$,
出ノード$x[1]$を設定！

ネットワークデータから
$networkx$モジュールを用いて
ネットワークを作成！

今回はこっち！

$.DiGraph$：有向リンク(行ったり来たり..向きのあるリンク)

$.Graph$ ：無向リンク

UEコード


```python
# 最短経路探索
def dijkstra(network, origin, destination):
    try:
        path = nx.dijkstra_path(network, origin, destination, weight='weight')
        return path
    except Exception as e:
        return 99999

# All or Nothing 配分
def AON(network, od, link_vol, flag):
    for od_i in od:
        res = dijkstra(network, od_i[0], od_i[1])
        if res == 99999:
            continue

        od_vol = float(od_i[2])
        for i in range(len(res)-1):
            for vol_i in link_vol:
                if vol_i[0] == res[i] and vol_i[1] == res[i+1]:
                    vol_i[flag] += od_vol
                    break
```

ダイクストラ法で最短経路探索！
$networkx$モジュールには
ダイクストラ法を実行する関数
$(.dijkstra_path)$が内蔵されている

最短経路に$All\ or\ Nothing$配分を実行し，
各リンクの交通量x_aを算出！
→リンクコスト(所要時間)の更新に用いる

$$t_a(x_a) = t_a^0\left\{1 + \alpha\left(\frac{x_a}{Cap_a}\right)^{\beta}\right\}$$

毎度おなじみBPR

UEコード


```python
# リンクコストの更新
def update_cost(network, link_vol):
    for vol_i in link_vol:
        target_link = network[vol_i[0]][vol_i[1]]
        target_link['weight'] = target_link['FFcost'] * (
            1 + target_link['alpha'] * (vol_i[2]/target_link['cap']) ** target_link['beta'])
    return network


# 探索方向ベクトルの計算
def calc_d(link_vol):
    for vol_i in link_vol:
        vol_i[5] = vol_i[4]-vol_i[2]

    return link_vol
```

*All or Nothing*配分の結果を用いてリンクコストを更新する関数

所要時間最小化問題を解く際に用いる探索方向ベクトルd_a^nを算出

$$d_a^n = y_a^n - x_a^n$$

y_a^nはリンクコスト更新後のリンクaの交通量
x_a^nはリンクコスト更新前のリンクaの交通量
nは繰り返し計算回数の添え字(nループ目)

UEコード


```python
# ステップ幅の計算
def obj_func(network, link_vol, xi):
    func = 0
    for vol_i in link_vol:
        target_link = network[vol_i[0]][vol_i[1]]
        xx = vol_i[2] + xi * vol_i[5]
        func += target_link['FFcost'] * (
            1 + target_link['alpha'] * (xx/target_link['cap']) ** target_link['beta'])
    return func

def calc_xi(network, link_vol):
    min_val = [0.0001,1000000]
    val = 0.0
    for i in range(0,100000):
        val += 0.0001
        res = obj_func(network, link_vol, val)
        if min_val[1] > abs(res):
            min_val[0] = val
            min_val[1] = abs(res)
        else:
            break
    return min_val[0]
```

下式の最適化問題を満足するような
ステップ幅ξ^n(読みはクサイ，グザイ)を直線探索により求める！

$$minZ(x) = min\sum_a t_a(x_a) = min\sum_a \int_0^{x_a^{n+1}=x_a^n+\xi^n d_a^n} t_a(w)dw$$

UEコード


```
# リンク交通量の更新
def update_link_vol(link_vol, xi):
    for vol_i in link_vol:
        vol_i[3] = vol_i[2] + xi * vol_i[5]
    return link_vol

# リンク交通量を用いた収束判定
def ck(link_vol):
    flag = 0
    for vol_i in link_vol:
        if abs(vol_i[3]-vol_i[2])>0.1:
            flag += 1

    if flag == 0:
        return 0, link_vol
    else:
        for vol_i in link_vol:
            vol_i[2] = copy.copy(vol_i[3])
            vol_i[3] = 0
            vol_i[4] = 0
            vol_i[5] = 0
        return 1, link_vol
```

下式のようにリンク交通量を更新

$$x_a^{n+1} = x_a^n + \xi^n d_a^n$$

※$x_a^{n+1} - x_a^n < \varepsilon$となるまで(繰り返し計算前後で交通量の差が$\varepsilon$より小さくなるまで繰り返し計算)

なお，εは自身で設定する収束判定値
ε小さい=判定厳しい↔収束しにくい

UEコード


```
def UEanalysis(network_file, od_file):
    #------------------------------------------------#
    # 初期設定 #
    #------------------------------------------------#
    print("Network Model: User Equilibrium Analysis")
    print("")
    print("Setting")
    # データの読み込み
    network_data = read_csv(network_file)
    od_data = read_csv(od_file)

    # 経路探索用データの生成
    graph_data = create_graph(network_data)

    print("Initial Assignment")
    # 配分結果
    link_vol= create_linkvol(network_data)

    # 初回のAON配分とコスト更新
    link_vol = AON(graph_data, od_data, link_vol, 2)

    #------------------------------------------------#
    # 繰り返し計算 #
    #------------------------------------------------#
    print("Iterative Assignment")
    for n in range(1,10000):
        print("  Ite:", n)
        # リンクコスト更新
        graph_data = update_cost(graph_data, link_vol)
        # 補助解の算出
        link_vol = AON(graph_data, od_data, link_vol, 4)
        # 探索ベクトルとステップサイズの決定
        link_vol = calc_d(link_vol)
        step_size = calc_xi(graph_data, link_vol)
        # 交通量の更新
        link_vol = update_link_vol(link_vol, step_size)
        # 収束判定
        conv_flag, link_vol = ck(link_vol)
        if n > 5 and conv_flag == 0:
            print('Fin!')
            break
    write_res('res_link_vol.csv', link_vol)
```

事前に作った関数を組み合わせて
にUEを実行する関数を作成

ネットワーク内のリンクデータ&にODデータ作成

ノードを繋いでネットワークに

配分結果の出力先を作成

AON配分&配分結果x_a^nよりリンクコストを更新

更新したコストをネットワークに反映

補助解y_a^nを算出

降下方向ベクトル，ステップ幅を算出

リンク交通量を更新→x_a^{n+1}

収束？Yes→結果を出力，No→コスト更新へ戻る

(

```
#write_res('res_link_vol.csv', link_vol)
return link_vol
```

にするとノートブック上に出力)

UEコード


```python
#文字列(str型)リストを数値(float型)リストに変換
def str_float(data):
    list_float = []
    for i in range(len(data)):
        transrate = list(map(float,data[i]))
        list_float.append(transrate)
    return list_float


#float型リストをint型リストに変換
def float_int(data):
    list_float = []
    for i in range(len(data)):
        transrate = list(map(int,data[i]))
        list_float.append(transrate)
    return list_float
```

```python
def create_total_TT(network_file,od_file):

    a = read_csv(network_file)
    b = str_float(a)
    c = float_int(b)

    link_data = []#ネットワーク内のリンク情報
    for i in range(len(c)):
        link_data_i = [c[i][0],c[i][1]]
        link_data_i.append(b[i][2])
        link_data_i.append(b[i][3])
        link_data_i.append(b[i][4])
        link_data_i.append(b[i][5])

        link_data.append(link_data_i)


    #UEによる交通量データ
    UE_result = UEanalysis(network_file,od_file)


    total_TT = 0
    #全リンク分の所要時間を計算
    for a in range(len(link_data)):

        #リンクごとBPRパラメータを用意
        link_data_a = link_data[a]
        cap = link_data_a[2]
        free_flow = link_data_a[3]
        alpha = link_data_a[4]
        beta = link_data_a[5]

        #リンク交通量を用意
        xa = UE_result[a][2]

        #BPR関数にリンク交通量を入力
        t_xa = free_flow*( 1 + (alpha*(xa/cap))**beta)

        #総所要時間を順繰り計算
        total_TT = total_TT + t_xa

    return total_TT
```

総所要時間を計算！

リンク情報の取得
(入出ノード，交通容量，
自由走行時間，形状パラメータ)

UEを実行

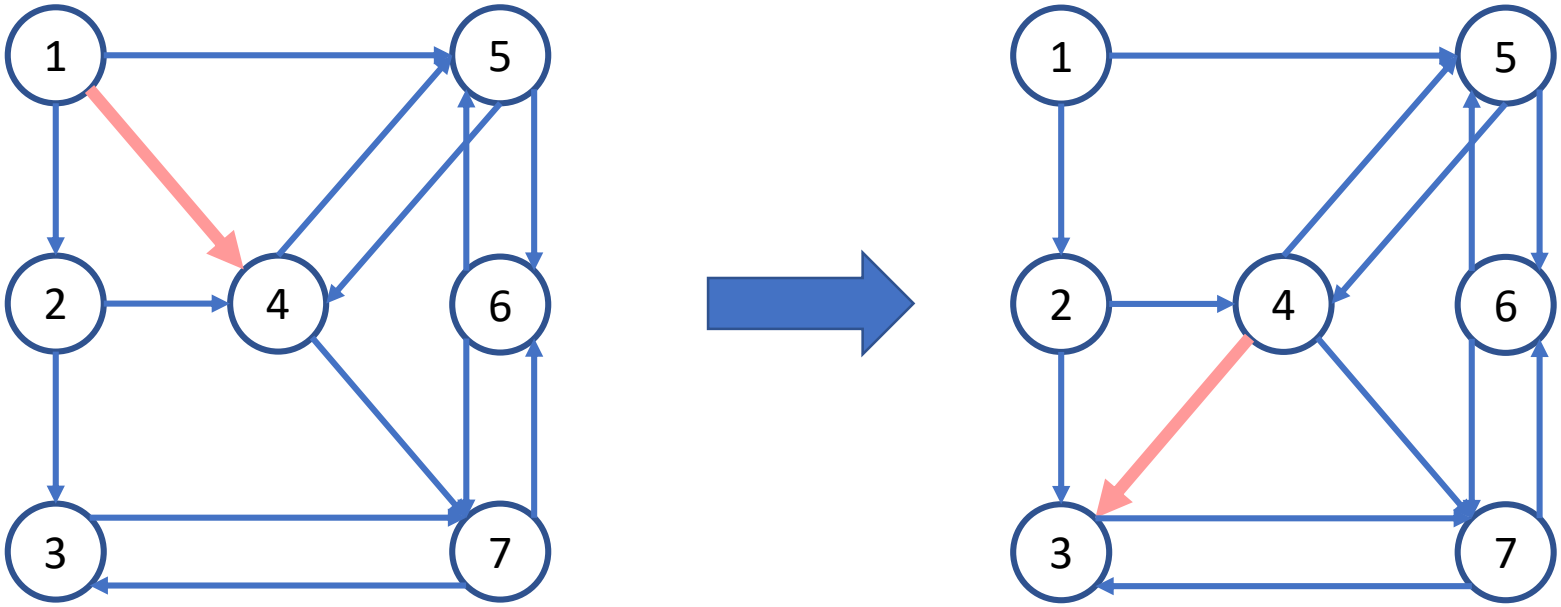
UEの結果から
リンク交通量x_aを得る

リンク交通量x_aをBPR関数に入力し，リンクコストt_a(リンク所要時間)を得る

リンクコストt_aを合計し，総所要時間を得る

演習課題1

CSVファイルを操作して
ネットワーク構造を変更してみよう！



UEを実行し，リンク交通量の変化を見てみよう！

演習課題2

CSVファイルを操作して
OD交通量，　ODパターンを変更してみよう！

↑例えば×OD_vol100倍とか，　1→6のODを追加してみるとか..

出発地　目的地　OD交通量

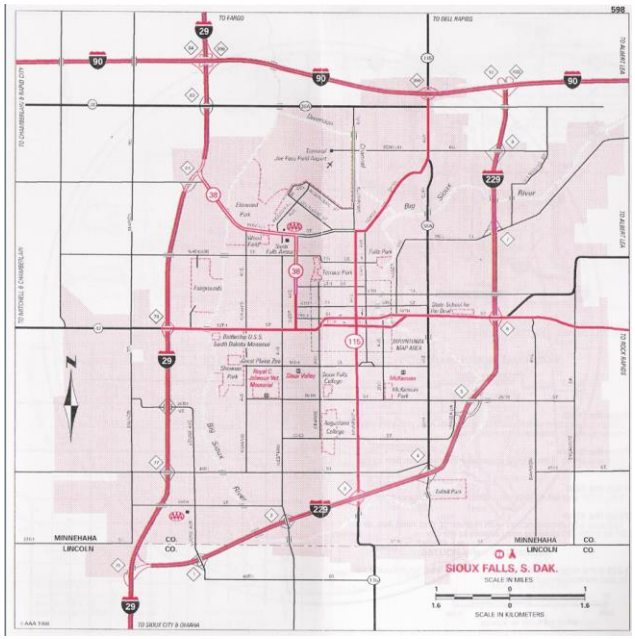
origin	destinatio	OD_vol
1	7	100
5	3	100
2	6	100
5	2	100

?

UEを実行し，収束までの時間の違いを体感しよう！

演習課題3

スーフォールズネットワークを
UEに適応してみよう！



アメリカ，サウスダコタ州のとある町のネットワーク．交通量配分のベンチマークとして広く用いられる．

SiouxFalls_network.csv
SiouxFalls_od.csv

※UEの収束に時間がかかるので実行は各自に任せます！

参考文献

参考文献：

1)土木学会：交通ネットワークの均衡分析−最新の理論と解法−

2) 小池卓武，柳沼秀樹：道路階層化を念頭に置いた
ネットワーク設計モデルの構築

3) SiouxFalls Network：
https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls