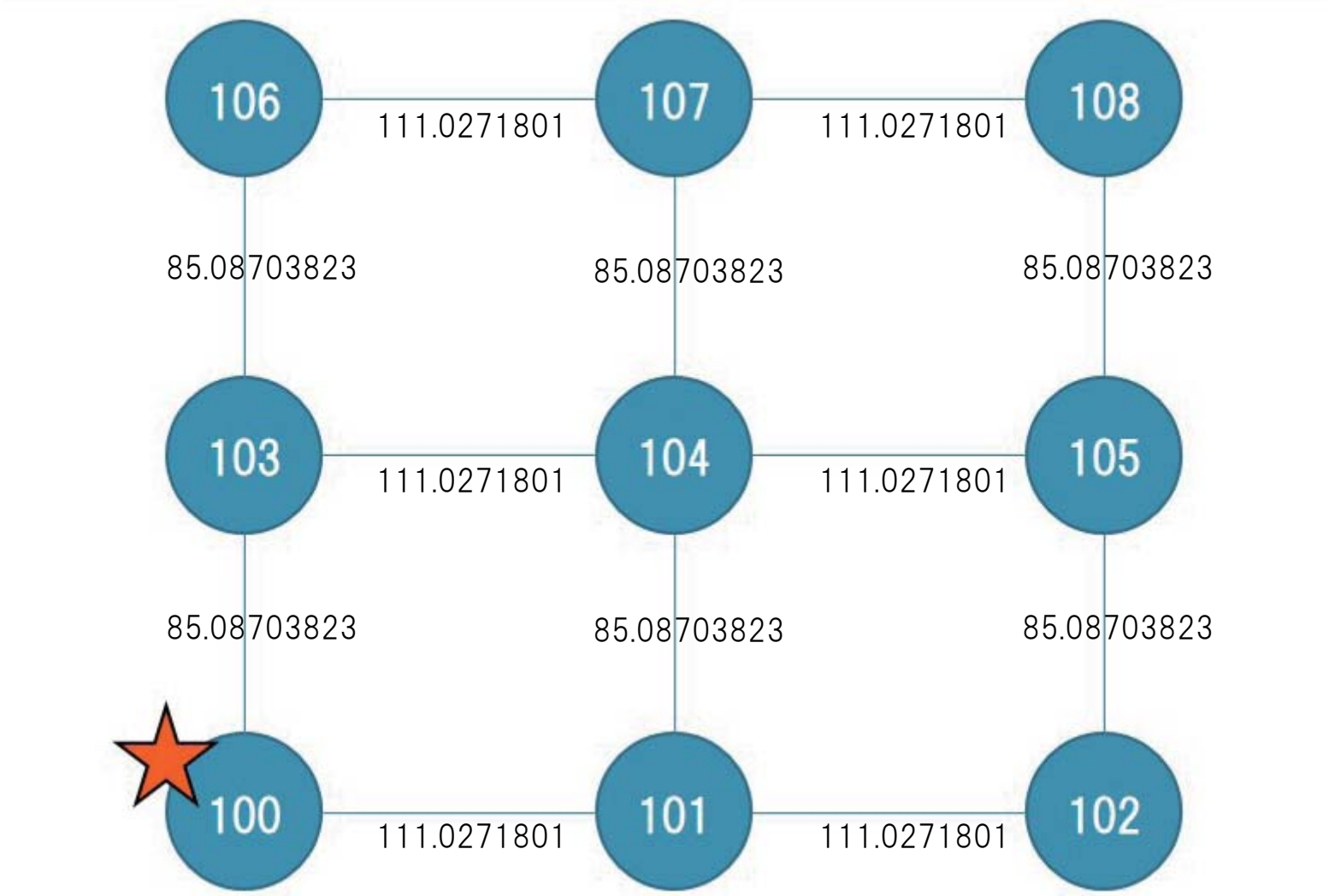
① Link 間距離を求める

羽佐田紘之

結果



②最短経路長を計算

羽佐田紘之

スクリプト（1/3）

```
####################################################
### パッケージの読み込み ######
####################################################
import numpy as np
from scipy.stats import rankdata
# from numpy import *
import pandas as pd
import itertools
import math
```

```
####################################################
### データ読み込み ######
####################################################

m = [
    [0, 111.0271801, 0, 85.08703823, 0, 0, 0, 0, 0],
    [111.0271801, 0, 111.0271801, 0, 85.08703823, 0, 0, 0, 0],
    [0, 111.0271801, 0, 0, 0, 85.08703823, 0, 0, 0],
    [85.08703823, 0, 0, 0, 111.0271801, 0, 85.08703823, 0, 0],
    [0, 85.08703823, 0, 111.0271801, 0, 111.0271801, 0, 85.08703823, 0],
    [0, 0, 85.08703823, 0, 111.0271801, 0, 0, 0, 85.08703823],
    [0, 0, 0, 85.08703823, 0, 0, 0, 111.0271801, 0],
    [0, 0, 0, 0, 85.08703823, 0, 111.0271801, 0, 111.0271801],
    [0, 0, 0, 0, 0, 85.08703823, 0, 111.0271801, 0],
]

   # ノード数は隣接行列の要素数です
m_node = len(m)

OD = pd.DataFrame(np.loadtxt( "Trip.csv", delimiter="," ,skiprows=1))
OD.columns = ["ID","male","age","O", "D"]

Olist = OD['O'] - 100
Dlist = OD['D'] - 100

pathlist = []
costlist = []
```

②最短経路長を計算

羽佐田紘之

スクリプト(2/3)

```
#####################################################
### 定義 ######
#####################################################

# ネットワークmにおいて出発地 ori から目的地 des までの最短経路を探していきます
def search(m, ori, des):
    # 初期設定
    max_cost = float('inf') # 初期コストは無限大
    unchecked = [False] * m_node  # 未確定ノードの集合
    cost = [max_cost] * m_node  # 起点から各ノードへの最小費用
    f_prev = [None] * m_node  # 最短経路を列挙するための配列
    cost[ori] = 0  # 起点のノードの距離は 0 とする
    f_prev[ori] = ori  # 起点前のノードは起点とする
    now = ori  # 現在地を起点　とする
    while True:
        min = max_cost  # 変数 min は現段階の最小費用を表す（初期は無限大）
        next = -1  # next は起点から最小費用のあるノードを表す ( 初期ー 1)
        unchecked[now] = True
        for i in range(m_node):  # 変数 i はノード（0~4）
            if unchecked[i]: continue  # ノードが確定していない場合ループが続く
            if m[now][i]:  # 今のノード i と起点 (now=ori) が接続しているかどうか
                tmp_cost = m[now][i] + cost[now]  # 一時的の費用を計算し、最小費用の更新
                if cost[i] > tmp_cost:
                    cost[i] = tmp_cost
                    f_prev[i] = now  # 直前のノードに更新
            if min > cost[i]:  # 現段階の最小費用と最小費用を持つノードを更新
                min = cost[i]
                next = i
        now = next  # 確定された最小費用持つノードが新しい”起点”となる
        if next == -1: break  # ノード番号がー 1 になるまで ( すべてノードが確認される )
    #print_path(f_prev, cost)  # 各接続しているノード間ルートの費用
    print(get_path(ori, des, f_prev), cost[des])
    return [get_path(ori, des, f_prev), cost[des]]

# 結果の出力
def print_path(f_prev, cost):
    for i in range(len(f_prev)):
        print("%d, prev = %d, cost = %d" % (i, f_prev[i], cost[i]))

def get_path(ori, des, f_prev):
    path = []
    now = des
    path.append(now)  # path の最後に now を加えている
    while True:
        path.append(f_prev[now])  # f_prev には一つ前のノードが入ってる。
        if f_prev[now] == ori: break  # 格納されたノードを逆にたどっていけば経路が求められる
        now = f_prev[now]
    path.reverse()
    return path
```

②最短経路長を計算

羽佐田紘之

スクリプト(3/3)

```
#####################################################
### 実行 ######
#####################################################

for v in range(len(Olist)):

    origin = int(Olist[v])
    destination = int(Dlist[v])
    print(origin, destination)  # 確認用
    path, cost = search(m, origin, destination)  # 返り値を格納

    # path を文字列化
    mapped_path = map(str, path)
    joined_path = ','.join(mapped_path)

    #path と cost をリスト化
    pathlist.append(joined_path)
    costlist.append(cost)

# 出力したいデータをまとめる
data = [OD["ID"], OD["male"], OD["age"], OD["O"], OD["D"], pathlist, costlist]
data = list(map(list,zip(*data)))
```

```
#####################################################
###csv に出力 ######
#####################################################

import csv

# ヘッダー
header = ["TripID","male","age","oNode","dNode","path","cost"]

# ファイルを書き込みモードでオープン
with open('result.csv', 'w') as f:

  writer = csv.writer(f,lineterminator='¥n')  # writer オブジェクトを作成
  writer.writerow(header) # ヘッダーを書き込む
  writer.writerows(data)
```

②最短経路長を計算

羽佐田紘之

結果

TripID	male	age	oNode	dNode	path	cost
10001	1	26	100	101	0,1	111.0272
10002	1	24	100	104	0,3,4	196.1142
10003	1	24	100	105	0,3,4,5	307.1414
10004	1	29	100	108	0,3,6,7,8	392.2284
10005	1	23	100	108	0,3,6,7,8	392.2284
10006	1	31	100	101	0,1	111.0272
10007	1	35	100	102	0,1,2	222.0544
10008	1	31	100	104	0,3,4	196.1142
10009	1	37	100	104	0,3,4	196.1142
10010	1	37	100	108	0,3,6,7,8	392.2284
10011	1	47	100	101	0,1	111.0272
10012	1	48	100	102	0,1,2	222.0544
10013	1	48	100	103	0,3	85.08704
10014	1	47	100	104	0,3,4	196.1142
10015	1	44	100	106	0,3,6	170.1741
10016	1	54	100	101	0,1	111.0272
10017	1	50	100	101	0,1	111.0272
10018	1	56	100	103	0,3	85.08704
10019	1	54	100	104	0,3,4	196.1142
10020	1	57	100	106	0,3,6	170.1741
10021	1	67	100	101	0,1	111.0272
10022	1	66	100	101	0,1	111.0272
10023	1	68	100	101	0,1	111.0272
10024	1	64	100	103	0,3	85.08704
10025	1	63	100	103	0,3	85.08704

TripID	male	age	oNode	dNode	path	cost
10026	0	20	100	102	0,1,2	222.0544
10027	0	20	100	104	0,3,4	196.1142
10028	0	27	100	104	0,3,4	196.1142
10029	0	23	100	107	0,3,6,7	281.2013
10030	0	22	100	108	0,3,6,7,8	392.2284
10031	0	33	100	101	0,1	111.0272
10032	0	31	100	104	0,3,4	196.1142
10033	0	33	100	104	0,3,4	196.1142
10034	0	37	100	106	0,3,6	170.1741
10035	0	38	100	108	0,3,6,7,8	392.2284
10036	0	45	100	101	0,1	111.0272
10037	0	43	100	103	0,3	85.08704
10038	0	41	100	104	0,3,4	196.1142
10039	0	42	100	104	0,3,4	196.1142
10040	0	49	100	106	0,3,6	170.1741
10041	0	50	100	101	0,1	111.0272
10042	0	52	100	103	0,3	85.08704
10043	0	58	100	104	0,3,4	196.1142
10044	0	53	100	104	0,3,4	196.1142
10045	0	53	100	106	0,3,6	170.1741
10046	0	62	100	101	0,1	111.0272
10047	0	67	100	101	0,1	111.0272
10048	0	64	100	101	0,1	111.0272
10049	0	69	100	103	0,3	85.08704
10050	0	66	100	104	0,3,4	196.1142

③目的地選択 MNL モデルの推定　　羽佐田紘之

スクリプト(1/3)

```
### Multinomial Logit model estimation
### ?f?[?^?t?@?C??? フ読み ????
Data1 <- read.csv("C:/Users/hiroyuki/Documents/urban analysis/Startupzemi/#1/
SampleData2/sample.csv",header=TRUE)
Data2 <- read.csv("C:/Users/hiroyuki/Documents/urban analysis/Startupzemi/#1/
SampleData2/custom.csv",header=TRUE)
## ?f?[?^?????????
hh <- nrow(Data1)

## ?p?????[?^??? フ ????????????l??????????????x?N?g???ub0?v???? 成
b0 <- numeric(12)

##### Logit model? フ対 ??? ゛度 ? ヨ ??? フ ???`#####

fr <- function(x) {
  ### ?p?????[?^? フ宣言
  ## ? 關 ???
  b1 <- x[1]
  b2 <- x[2]
  b3 <- x[3]
  b4 <- x[4]
  b5 <- x[5]
  b6 <- x[6]
  b7 <- x[7]
  ## ?? ソス ? ソス ? ソス ? マ ???p?????[?^
  d2 <- x[8]
  d3 <- x[9]
  d6 <- x[10]
  d8 <- x[11]
  d9 <- x[12]

## ?? ソス ? ソス ? ソス ? マ ???p?????[?^
d2 <- x[8]
d3 <- x[9]
d6 <- x[10]
d8 <- x[11]
d9 <- x[12]

## ? ホ ??? ゛度 ? フ ??? °の変 ????? 骭セ
LL = 0

### ????? フ目的 ?n? へ以 ??? フ8??
## 5202(dN1)
## 5204(dN2)
## 5206(dN3)
## 5208(dN4)
## 5212(dN5)
## 5213(dN6)
## 5272(dN7)
## 5303(dN8)
```

③目的地選択 MNL モデルの推定　　羽佐田紘之

スクリプト（2/3）

```
## ???p? フ計 ?Z?F?? ソス ? ソス ? ソス ? マ ??? ノ ???????????????
# ????        ?@?@?@?@    # ? 關 ???
dN1   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,15]*0.001  +d8*matrix(Data2[1,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[1,4],nrow =hh,ncol=1)  +b1*matrix(1,nrow =hh,ncol=1))
dN2   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,16]*0.001  +d8*matrix(Data2[2,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[2,4],nrow =hh,ncol=1)  +b2*matrix(1,nrow =hh,ncol=1))
dN3   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,17]*0.001  +d8*matrix(Data2[3,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[3,4],nrow =hh,ncol=1)  +b3*matrix(1,nrow =hh,ncol=1))
dN4   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,18]*0.001  +d8*matrix(Data2[4,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[4,4],nrow =hh,ncol=1)  +b4*matrix(1,nrow =hh,ncol=1))
dN5   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,13]*0.001  +d8*matrix(Data2[5,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[5,4],nrow =hh,ncol=1)  +b5*matrix(1,nrow =hh,ncol=1))
dN6   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,14]*0.001  +d8*matrix(Data2[6,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[6,4],nrow =hh,ncol=1)  +b6*matrix(1,nrow =hh,ncol=1))
dN7   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,19]*0.001  +d8*matrix(Data2[7,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[7,4],nrow =hh,ncol=1)  +b7*matrix(1,nrow =hh,ncol=1))
dN8   <- exp(d2*Data1[,3] +d3*Data1[,4] +d6*Data1[,20]*0.001  +d8*matrix(Data2[8,3],nrow =hh,ncol=1)*0.0001  +d9*matrix(Data2[8,4],nrow =hh,ncol=1)                    )

### ?l???m??? フ計 ?Z
## ????? ニな ???A?e?X??exp(V)? フ和 ??? ッ ????
deno <- (dN1 + dN2 + dN3 + dN4 + dN5 + dN6 + dN7 + dN8)

## ??? 黷シ ???v?Z????
PdN1   <- dN1 / deno
PdN2   <- dN2 / deno
PdN3   <- dN3 / deno
PdN4   <- dN4 / deno
PdN5   <- dN5 / deno
PdN6   <- dN6 / deno
PdN7   <- dN7 / deno
PdN8   <- dN8 / deno
```

③目的地選択 MNL モデルの推定　　羽佐田紘之

スクリプト(3/3)

```
## ?l???m????0 ? ノな゛ソス ? ソスト ??? ワ゛ソス ? ソス ?? 鼾 ?? ノ起 ????????? フ ????
## ??? フ場合 ? ノは1??? ヤ ??? ト対 ??
PdN1   <- (PdN1!=0)*PdN1  + (PdN1==0)
PdN2   <- (PdN2!=0)*PdN2  + (PdN2==0)
PdN3   <- (PdN3!=0)*PdN3  + (PdN3==0)
PdN4   <- (PdN4!=0)*PdN4  + (PdN4==0)
PdN5   <- (PdN5!=0)*PdN5  + (PdN5==0)
PdN6   <- (PdN6!=0)*PdN6  + (PdN6==0)
PdN7   <- (PdN7!=0)*PdN7  + (PdN7==0)
PdN8   <- (PdN8!=0)*PdN8  + (PdN8==0)

## ?l?? ソス ? ソス ? ソス ? ソス ?
CdN1   <- Data1[,8] == 5202
CdN2   <- Data1[,8] == 5204
CdN3   <- Data1[,8] == 5206
CdN4   <- Data1[,8] == 5208
CdN5   <- Data1[,8] == 5212
CdN6   <- Data1[,8] == 5213
CdN7   <- Data1[,8] == 5272
CdN8   <- Data1[,8] == 5303

## ? ホ ??? ゛度 ? フ計 ?Z
LL <- colSums(CdN1 *log(PdN1)  +CdN2 *log(PdN2)  +CdN3 *log(PdN3)
           +CdN4 *log(PdN4)  +CdN5 *log(PdN5)  +CdN6 *log(PdN6)
           +CdN7 *log(PdN7)  +CdN8 *log(PdN8)                )
}
```

```
##### ? ホ ??? ゛度 ? ヨ ??fr? フ最大化 #####

## ?p?????[?^?l? フ最適 ?? ?@? ナ適 ??? ヨ ??optim?i?d?v? ネので要 ?m?F?j
res <- optim(b0,fr,gr=NULL, method = "Nelder-Mead", hessian = TRUE,
control=list(fnscale=-1))
## ?p?????[?^?????l?A?w?b?Z?s??
b   <- res$par
hhh <- res$hessian
## t?l? フ計 ?Z
tval <- b/sqrt(-diag(solve(hhh)))
## ????? ゛度
L0 <- fr(b0)
## ? ナ終 ? ゛度
LL <- res$value

##### ??? ハの出 ?? #####
print(res)
## ????? ゛度
print(L0)
## ? ナ終 ? ゛度
print(LL)
## ??^2?l
print((L0-LL)/L0)
## ?C??? マみ ??^2?l
print((L0-(LL-length(b)))/L0)
## ?p?????[?^?????l
print(b)
## t?l
print(tval)
```

③目的地選択 MNL モデルの推定　　羽佐田紘之

結果

説明変数	パラメータ	t値
年齢	−0.330	−1.50E−06
職業コード	−0.272	−8.28E−06
出発地からの距離（m/1000）	−0.849	−11.96 **
到着地敷地面積（m2/10000）	−1.128	−0.08
到着地開店時間（h）	−0.162	−0.15
定数項（ﾌｼﾞｸﾞﾗﾝ松山店）	0.579	0.12
定数項（ﾌｼﾞｸﾞﾗﾝ重信SC）	0.250	0.13
定数項（ﾊﾟﾙﾃｨ姫原SC）	−1.045	−0.27
定数項（ﾊﾟﾙﾃｨ衣山SC）	0.566	0.11
定数項（伊予鉄高島屋）	0.773	0.07
定数項（三越　松山店）	−0.707	−0.19
定数項（ジャスコ松山店）	0.569	0.24
サンプル数	287	
初期尤度	−596.80	
最終尤度	−406.91	
尤度比	0.318	
修正済み尤度比	0.298	

** 5%有意

④重要事項の整理

羽佐田紘之

尤度関数

全ての個人が、選択肢の中からそれぞれ実際に選んだ選択肢を選ぶ確率。選択肢（選択変数）に影響を与える未知パラメータの関数となっている。最尤法ではこれの最大化問題を解くことによって適当なパラメータ値を求める。

ヘッセ行列

$H(f)= \nabla_i \nabla_j f$ で与えられる行列。固有値の符号が全て負のとき関数fは極大をとる。

Nelder-Mead method ではヘッセ行列がなくても最適化することができる。

ρ（適合度）

$$\rho^2 = \frac{L0 - LL}{L0}$$

L0：全ての未知パラメータを 0 とおいたときの対数尤度関数の値

LL：推定されたパラメータによる対数尤度関数の値

尤度比とも。離散選択モデルがどの程度データの程度を上手く説明するかを表す指標。

0-1 区間内の値をとり、大きいほど適合度が高い。

④重要事項の整理

羽佐田紘之

t 値

パラメータを標本分散で除した値。

t 値が有意水準をを超えていれば、そのパラメータは統計的に有意であり、意味をもつと言える。

選択集合の選び方

MNL モデルでは、それぞれの選択肢の誤差項に独立で同一な (IID) ガンベル分布を仮定している。（二項分布と仮定すると積分が現れてしまうため。）この IID 条件は、各選択肢がそれぞれの効用のみに影響を受けていることを示すが、類似した選択肢が存在し、選択肢の誤差項が独立であるという仮定が誤っていた場合に類似選択肢の選択確率が過大評価されてしまうという短所がある。

この IID 条件によって導かれる、無関係な選択肢からの独立性（IIA）条件を検定するために、ハウスマン・テストを用いる。IIA 条件が成立すれば、選択肢の部分集合を用いた推定値は、選択肢の完全集合を用いた推定値と統計的に異ならないという性質を利用している。

$$\text{ハウスマン統計量}[\beta_u - \beta_r]'[V_r - V_u]^{-1}[\beta_u - \beta_r]$$

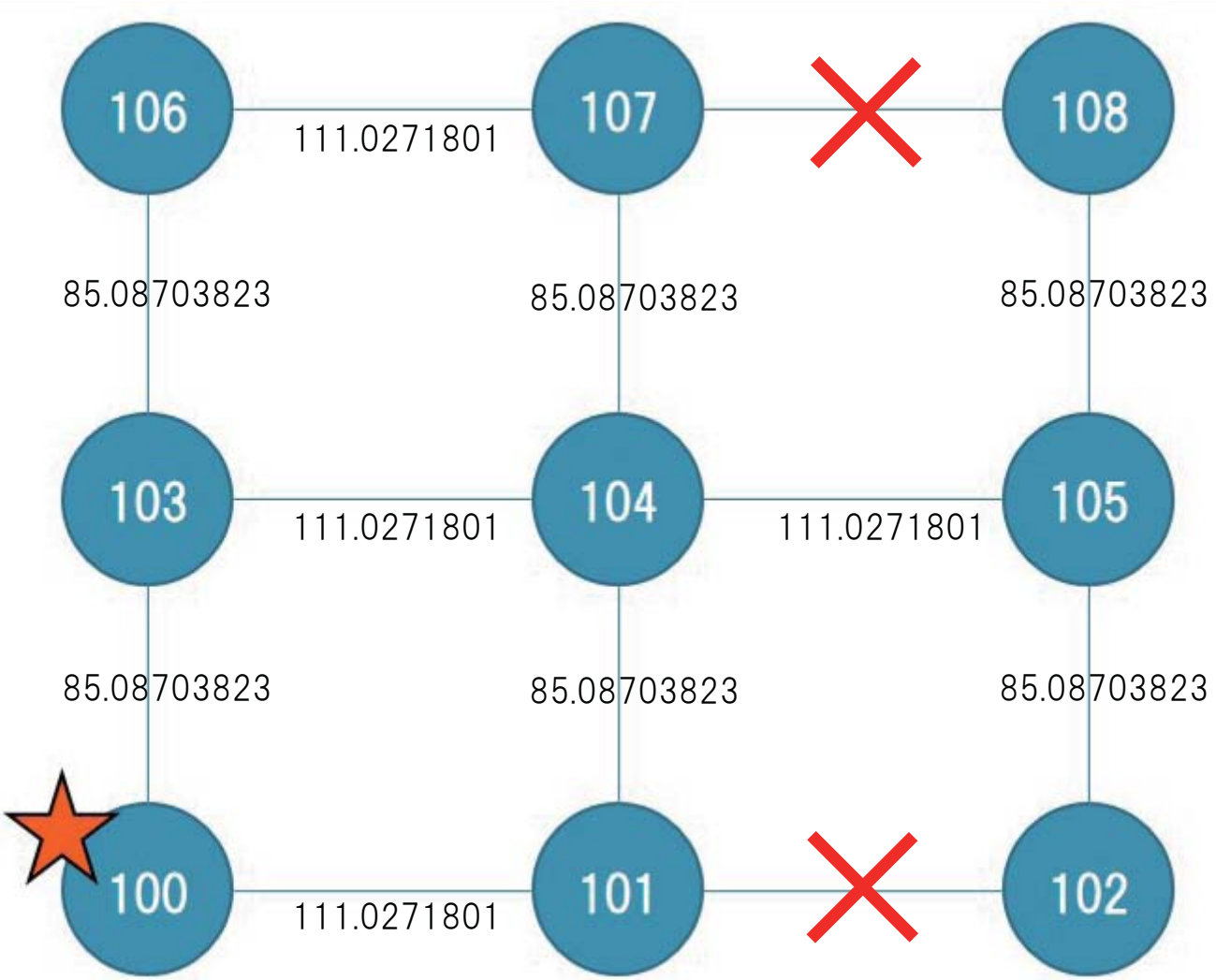
ハウスマン統計量は χ^2 分布に従うので、推定されるパラメータの数だけの自由度を持つ片側 5%の χ^2 臨界値との大小を比較し、ハウスマン統計量が χ^2 臨界値よりも大きければ IIA 条件は成立していないことになる。

参考文献；
依田高典「ブロードバンド・ビックバンの経済学」
BinN studies シリーズ　ネットワーク行動学 - 都市と移動 -
http://www.turbare.net/transl/scipy-lecture-notes/advanced/mathematical_optimization/index.html

⑤応用

羽佐田紘之

仮定


106
107
108
111.0271801
85.08703823
85.08703823
85.08703823
103
104
105
111.0271801
111.0271801
85.08703823
85.08703823
85.08703823
100
101
102
111.0271801

⑤応用

羽佐田紘之

結果

TripID	male	age	oNode	dNode	path	cost
10001	1	26	100	101	0,1	111.0272
10002	1	24	100	104	0,3,4	196.1142
10003	1	24	100	105	0,3,4,5	307.1414
10004	1	29	100	108	0,3,4,5,8	392.2284
10005	1	23	100	108	0,3,4,5,8	392.2284
10006	1	31	100	101	0,1	111.0272
10007	1	35	100	102	0,3,4,5,2	392.2284
10008	1	31	100	104	0,3,4	196.1142
10009	1	37	100	104	0,3,4	196.1142
10010	1	37	100	108	0,3,4,5,8	392.2284
10011	1	47	100	101	0,1	111.0272
10012	1	48	100	102	0,3,4,5,2	392.2284
10013	1	48	100	103	0,3	85.08704
10014	1	47	100	104	0,3,4	196.1142
10015	1	44	100	106	0,3,6	170.1741
10016	1	54	100	101	0,1	111.0272
10017	1	50	100	101	0,1	111.0272
10018	1	56	100	103	0,3	85.08704
10019	1	54	100	104	0,3,4	196.1142
10020	1	57	100	106	0,3,6	170.1741
10021	1	67	100	101	0,1	111.0272
10022	1	66	100	101	0,1	111.0272
10023	1	68	100	101	0,1	111.0272
10024	1	64	100	103	0,3	85.08704
10025	1	63	100	103	0,3	85.08704

TripID	male	age	oNode	dNode	path	cost
10026	0	20	100	102	0,3,4,5,2	392.2284
10027	0	20	100	104	0,3,4	196.1142
10028	0	27	100	104	0,3,4	196.1142
10029	0	23	100	107	0,3,6,7	281.2013
10030	0	22	100	108	0,3,4,5,8	392.2284
10031	0	33	100	101	0,1	111.0272
10032	0	31	100	104	0,3,4	196.1142
10033	0	33	100	104	0,3,4	196.1142
10034	0	37	100	106	0,3,6	170.1741
10035	0	38	100	108	0,3,4,5,8	392.2284
10036	0	45	100	101	0,1	111.0272
10037	0	43	100	103	0,3	85.08704
10038	0	41	100	104	0,3,4	196.1142
10039	0	42	100	104	0,3,4	196.1142
10040	0	49	100	106	0,3,6	170.1741
10041	0	50	100	101	0,1	111.0272
10042	0	52	100	103	0,3	85.08704
10043	0	58	100	104	0,3,4	196.1142
10044	0	53	100	104	0,3,4	196.1142
10045	0	53	100	106	0,3,6	170.1741
10046	0	62	100	101	0,1	111.0272
10047	0	67	100	101	0,1	111.0272
10048	0	64	100	101	0,1	111.0272
10049	0	69	100	103	0,3	85.08704
10050	0	66	100	104	0,3,4	196.1142