第４回スタートアップゼミ

アルゴリズム・Rの応用

2012年5月25日

M1 伊藤 創太

○本日のメニュー

第1回
クロス集計
行動モデル基礎

第2回
Rの使い方
MNLモデル推定

第3回
Eclipseの使い方
Javaの基礎

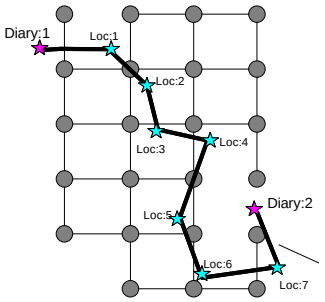
第4回

■プログラム・アルゴリズム

- ダイクストラ法
- マップマッチング

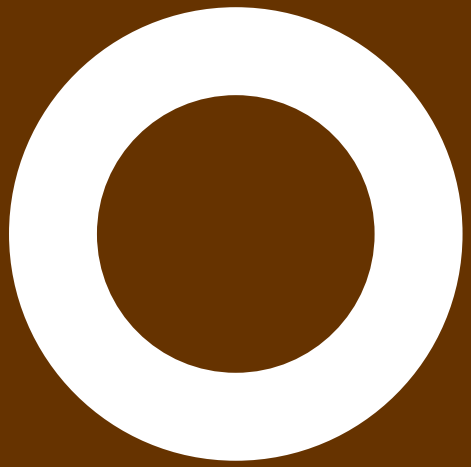
■Rの応用

- 関数の作り方
- 最適化関数




```
> ## 結果の出力
> ##ρ^2値
> print((L0-LL)/L0)
[1] 0.2809693
> ## 修正済ρ^2値
> print((L0-(LL-length(b)))/L0)
[1] 0.2571551
>
>
> print(res)
$par
 [1]  1.73583335  0.08750992  1.48217027  1.18417522 -0.79413283
 [6] -1.18444108 -0.95048348 -0.84977429 -0.88747670  1.34897282
[11]  1.73061252

$value
[1] -332.1265
```



アルゴリズム

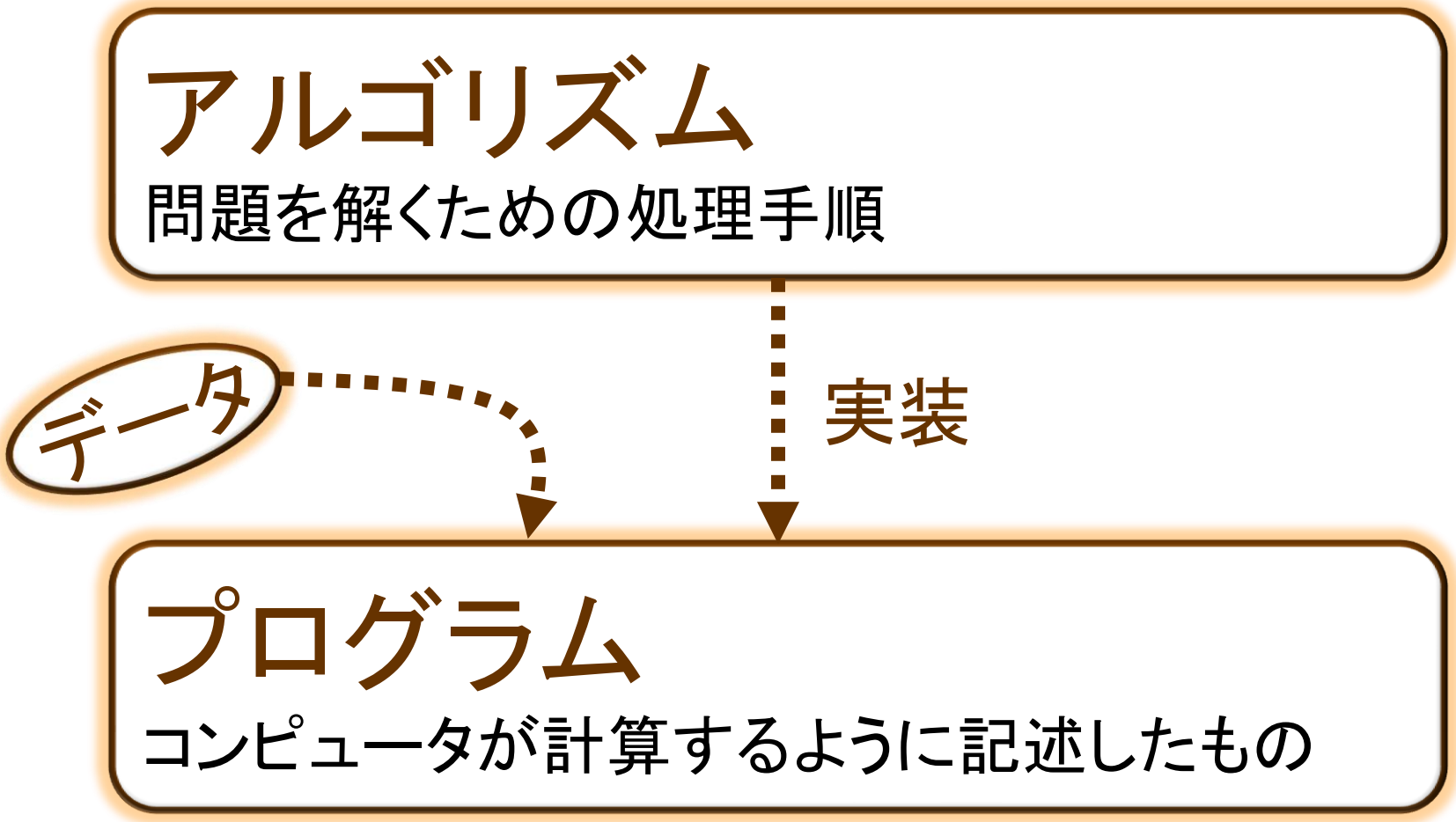
○プログラムとアルゴリズム

プログラムは所詮手段。

やっていることの中身が大事。

ブラックボックスにしない。

○プログラムとアルゴリズム



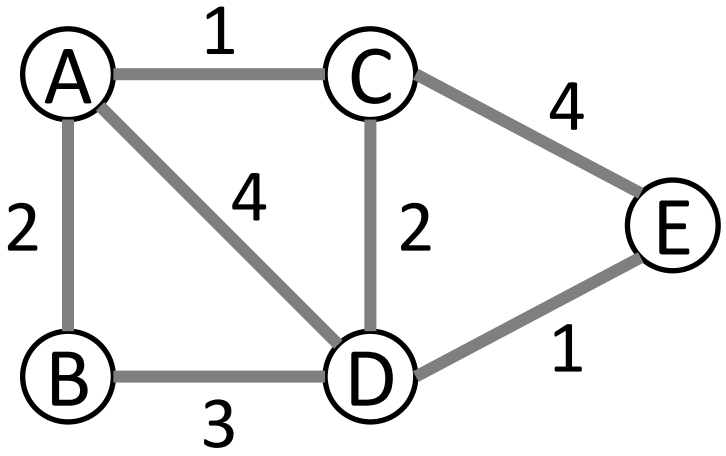
○ダイクストラ法

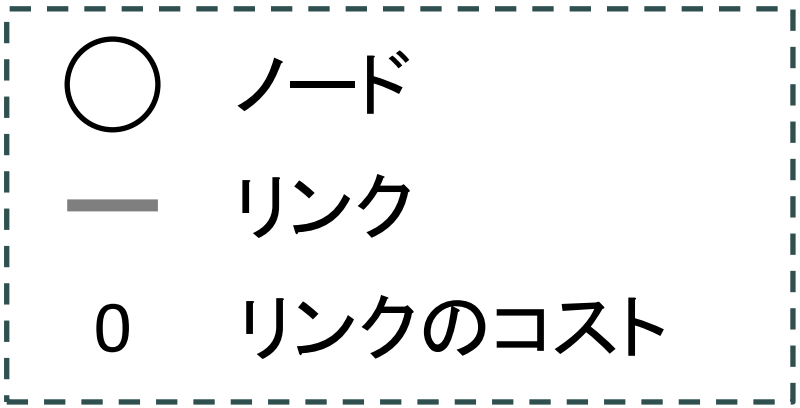
▼ダイクストラ法

グラフの最短距離・経路を求めるアルゴリズム

ダイクストラ（蘭）が1958年に考案

（例） AからEまで行く最短距離は？
そしてその経路は？

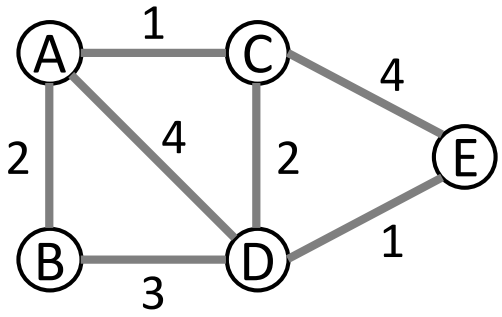




○ダイクストラ法

▼（補足）グラフのデータ構造

- リストによる構造/行列による構造



リスト

各リンクの起終点ノードで格納
疎なネットワーク向き

ID	起点	終点	コスト
1	A	B	2
2	B	A	2
:	:	:	:
13	D	E	1
14	E	D	1

行列

各OD間コストを格納
密なネットワーク向き

起点ノード＼終点ノード	A	B	C	D	E
A	0	2	1	3	∞
B	2	0	∞	3	∞
C	1	∞	0	2	4
D	3	3	2	0	1
E	∞	∞	4	1	0

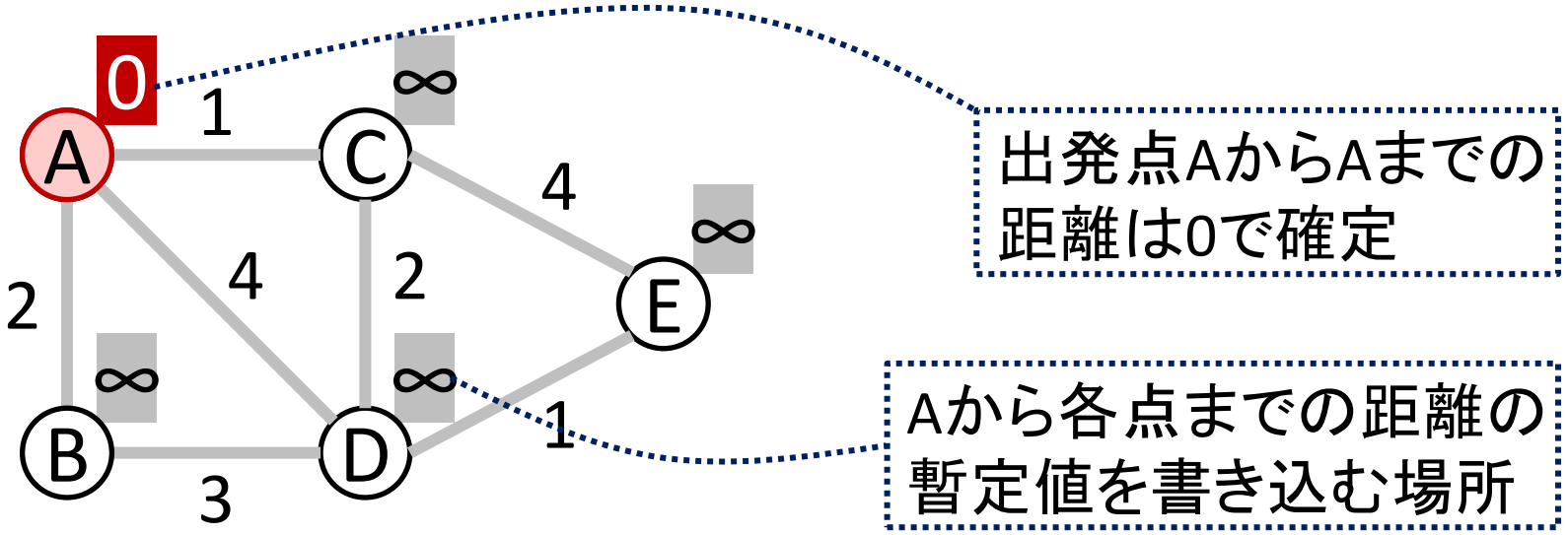
○ダイクストラ法

▼ダイクストラ法の考え方

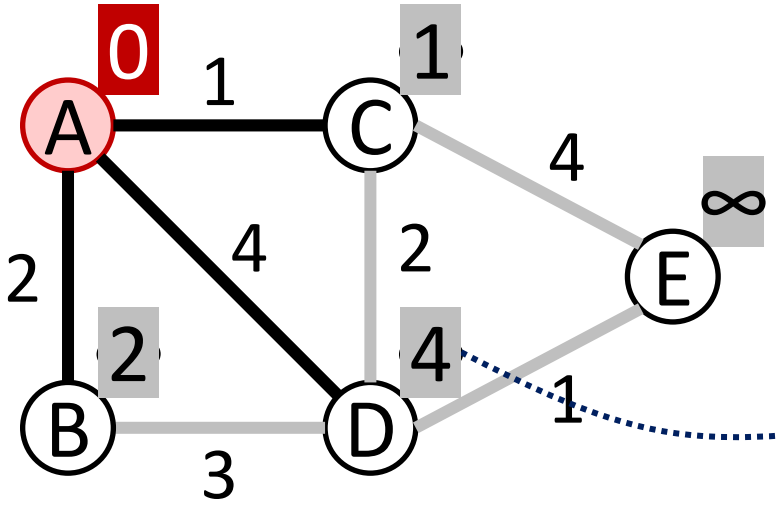
＜わかった最短経路から確定させていく＞

【Step1】 出発点から各点の距離の暫定値を設定

（赤色：確定済のノード・値、灰色：未確定）



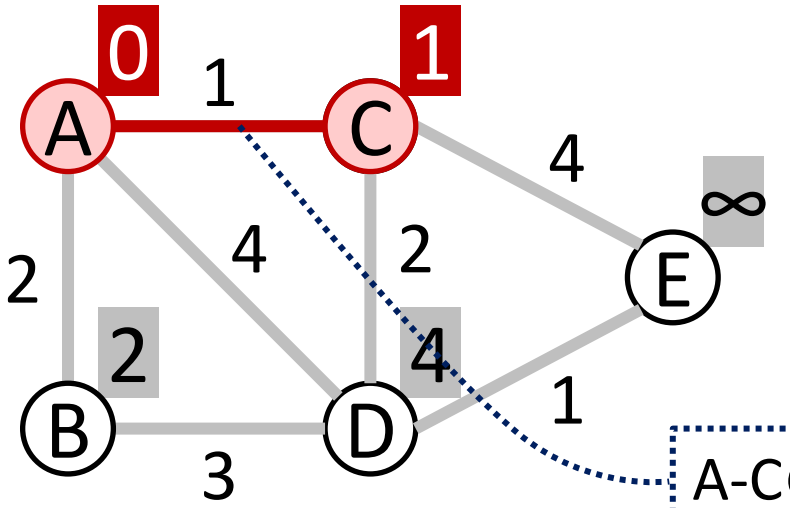
○ダイクストラ法



【Step2】

確定済の点から経路をたどり暫定値を更新

例えば現時点でのAからDの最短距離は4

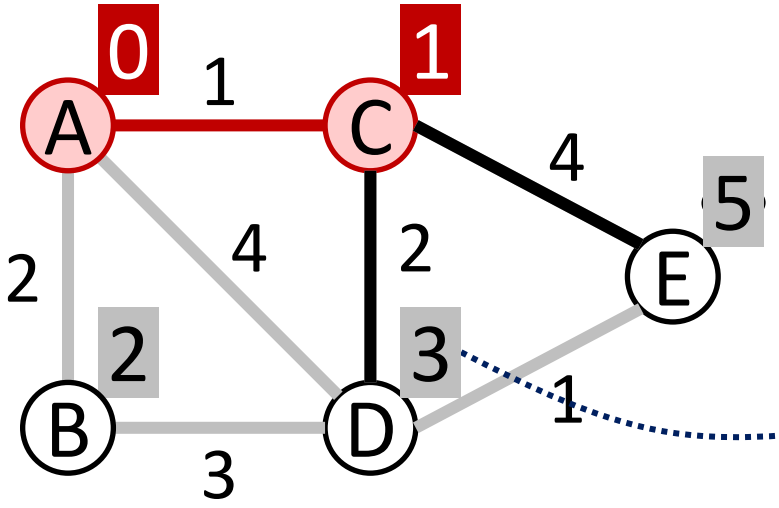


【Step3】

未確定の中で最小距離の点を確定させる

A-Cの最短経路も確定

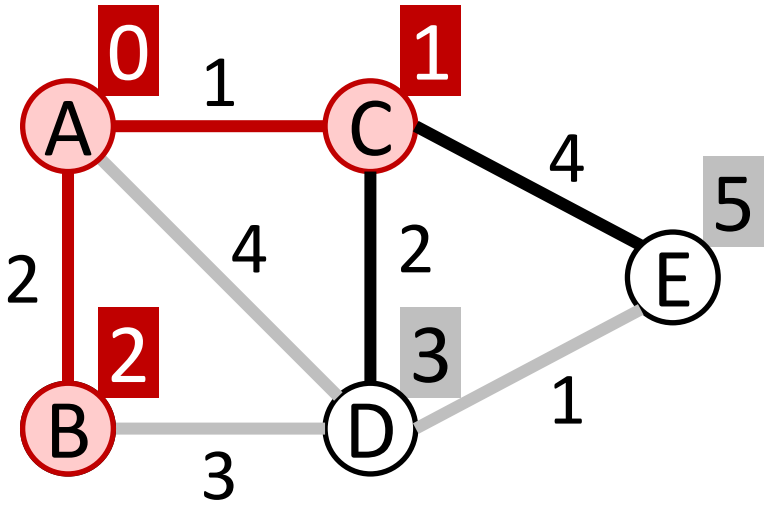
○ダイクストラ法



【Step4】

確定済の点から経路をたどり暫定値を更新

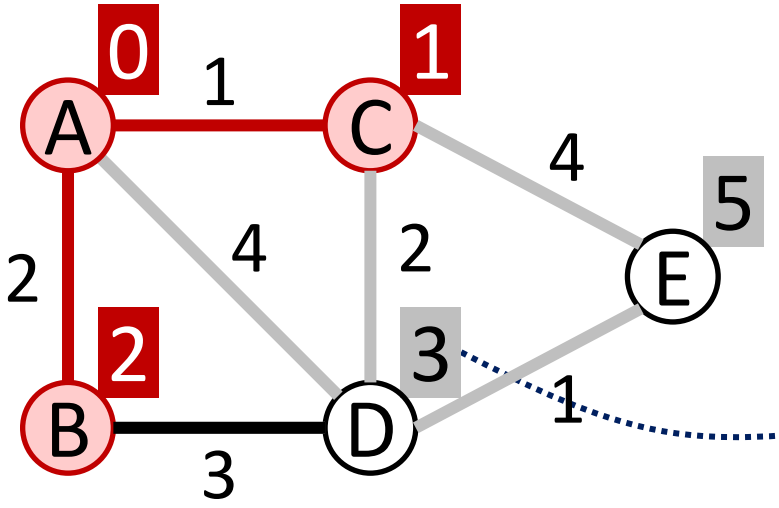
A-Dの4よりA-C-Dの3の方が小さいため、3に更新



【Step5】

未確定の中で最小の点を確定させる

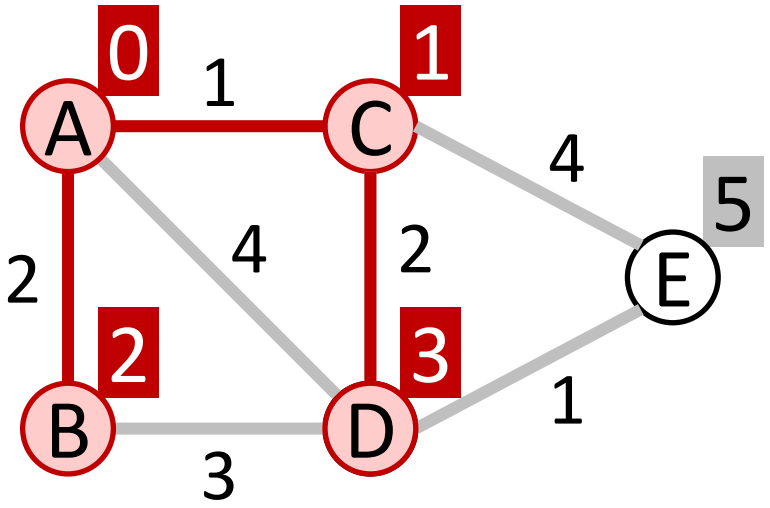
○ダイクストラ法



【Step6】

確定済の点から経路をたどり暫定値を更新

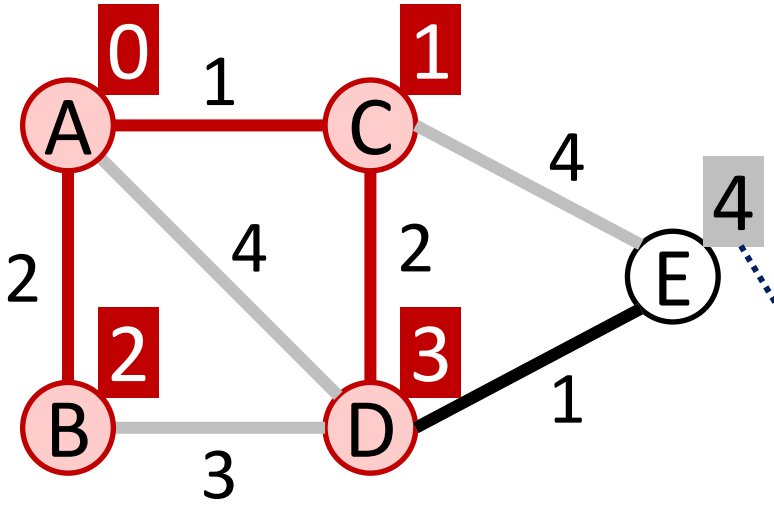
A-C-Dの3よりA-B-Dの5の方が大きいため更新されず



【Step7】

未確定の中で最小の点を確定させる

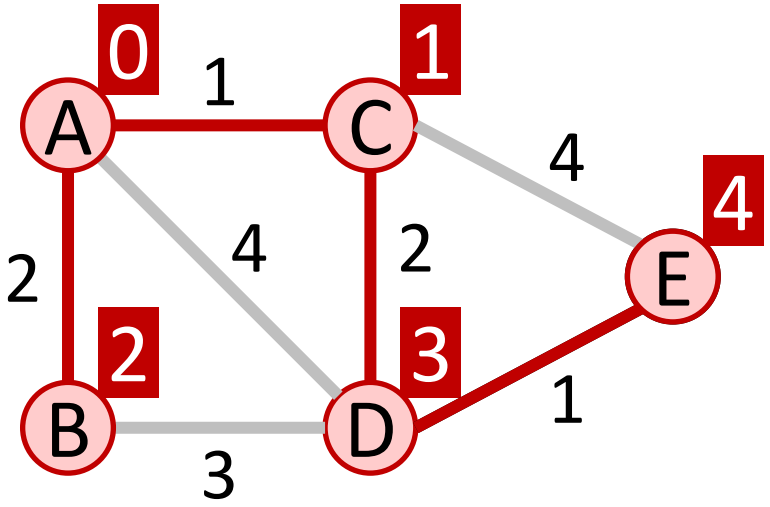
○ダイクストラ法



【Step8】

確定済の点から経路をたどり暫定値を更新

A-C-Eの5よりA-C-D-Eの4の方が小さいため更新



【Step9】

未確定の中で最小の点を確定させる

完成

AからEの最短距離は4で経路はA-C-D-E

○ダイクストラ法

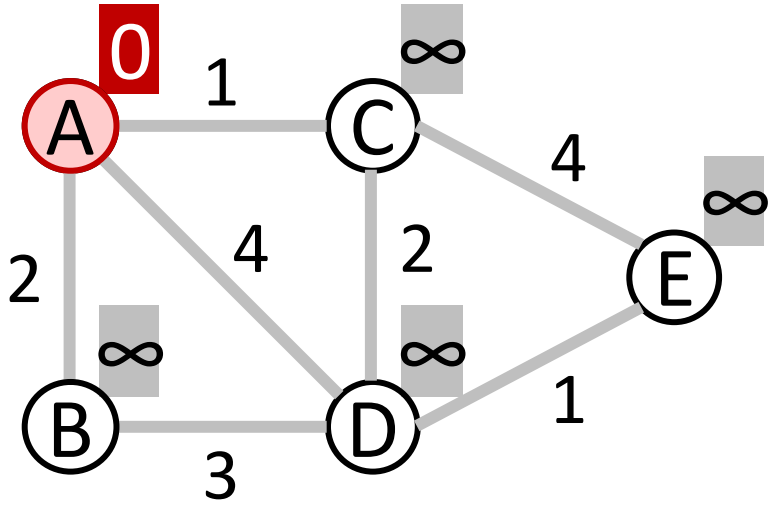
▼データ格納の例

リンク

A	B	2
B	A	2
:	:	:
D	E	1
E	D	1

最短経路かどうか

1
0
:
1
0



ノード

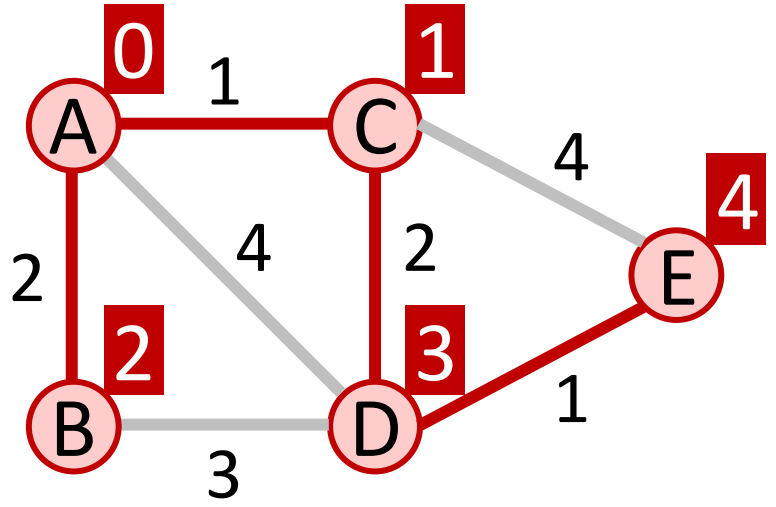
A
B
C
D
E

最短距離

0
2
1
3
4

確定済かどうか

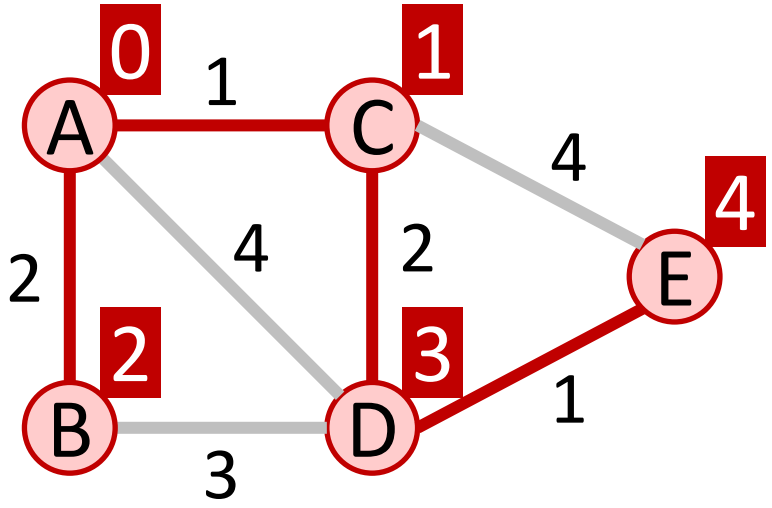
1
1
1
1
1



○ダイクストラ法

▼ダイクストラ法の特徴

- 1対多の探索に有効
- 負のコストは扱えない



▼ダイクストラ法の利用

- カーナビゲーションの経路探索
- 鉄道の経路案内
- マップマッチング
- OD交通量の配分

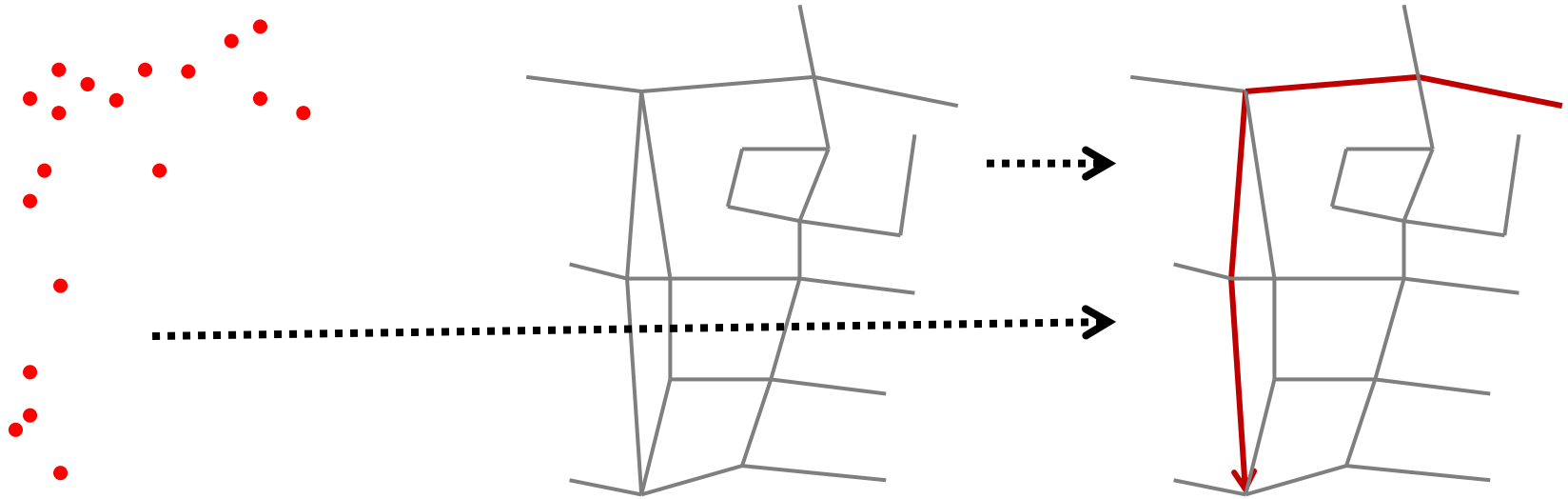
○マップマッチング

▼マップマッチングとは何か？やりたいことは？

位置データから経路を特定したい

経路は？リンク所要時間は？

観測データ　　ネットワークデータ　　経路



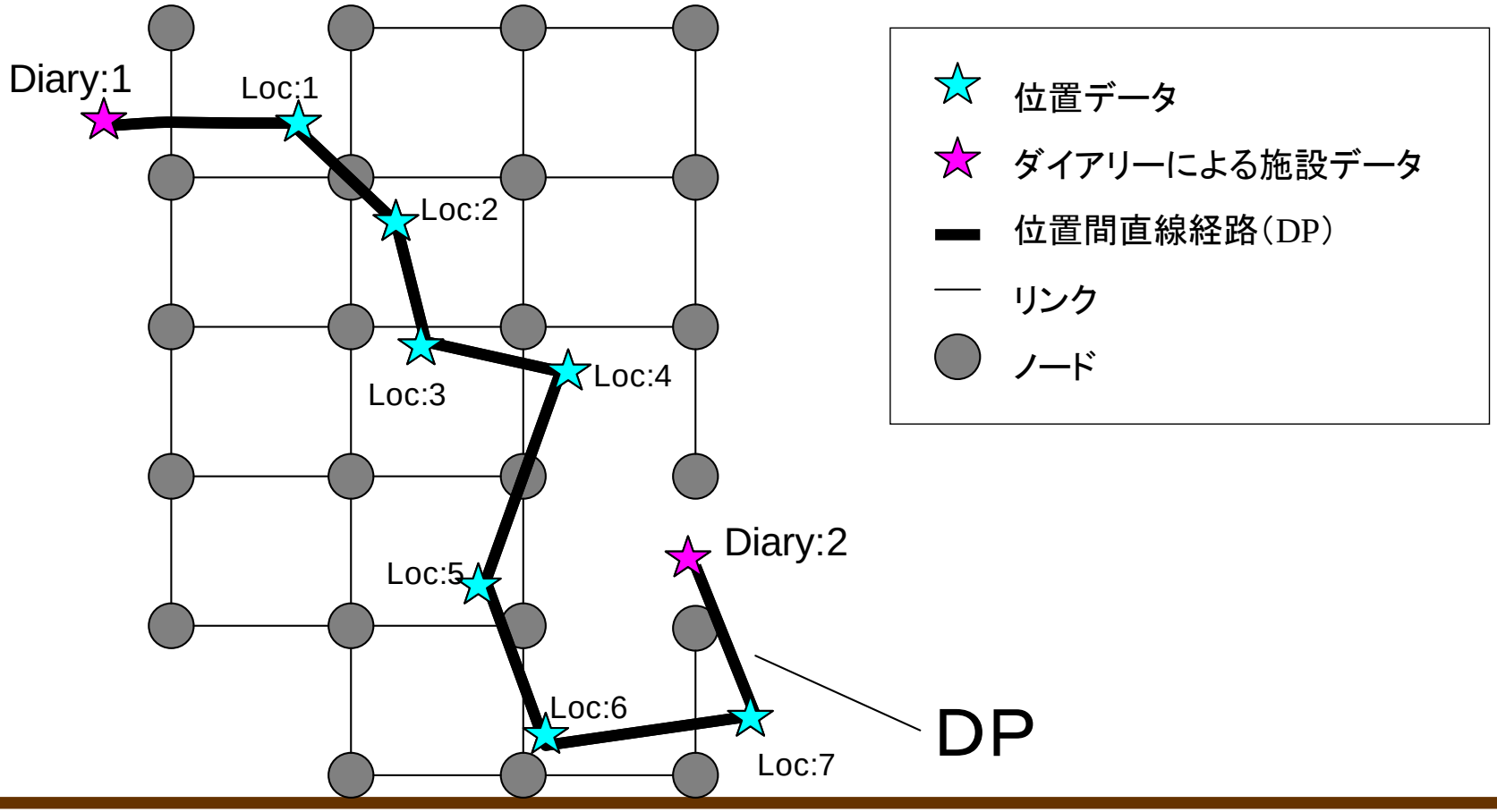
ここでは三谷(2006)のアルゴリズムを紹介

○マップマッチング

【Step1】位置データの取得

GPS携帯から位置データを，webダイアリーから施設の位置データを取得する.

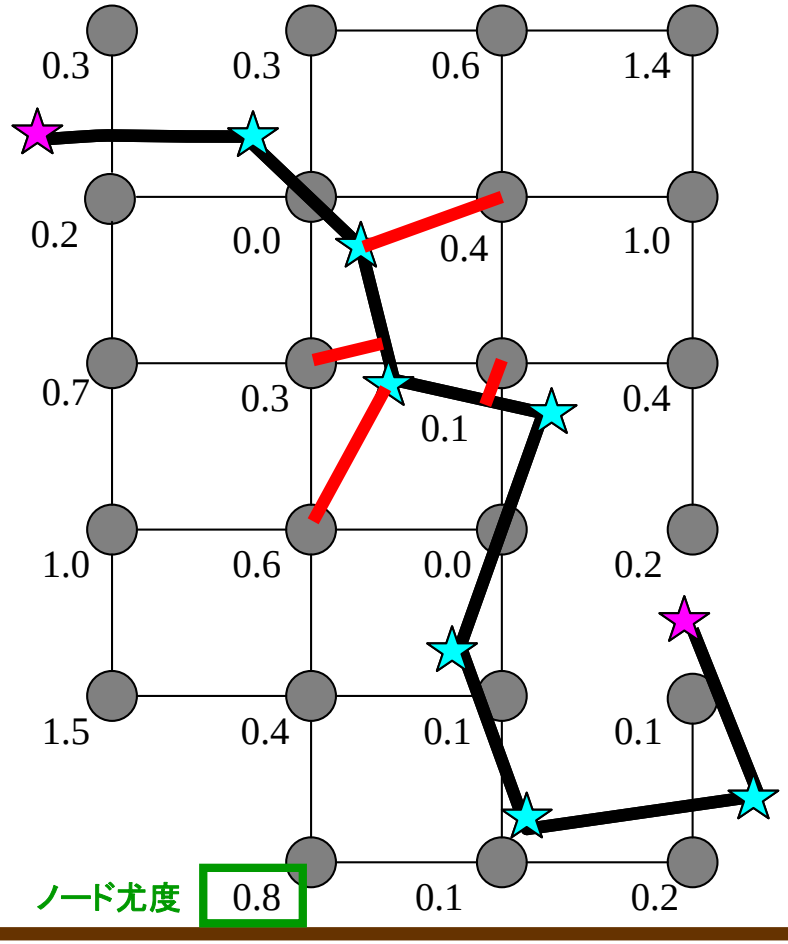
両方のデータを時間ごとに結び位置間直線経路（DP）とする.

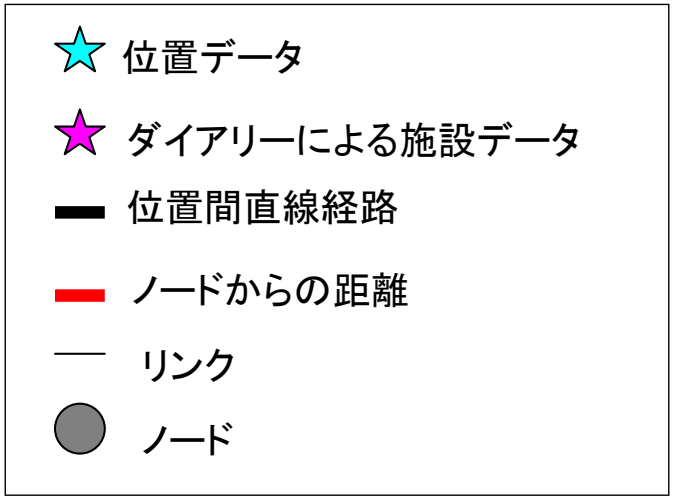


○マップマッチング

【Step2】ノード尤度の算出

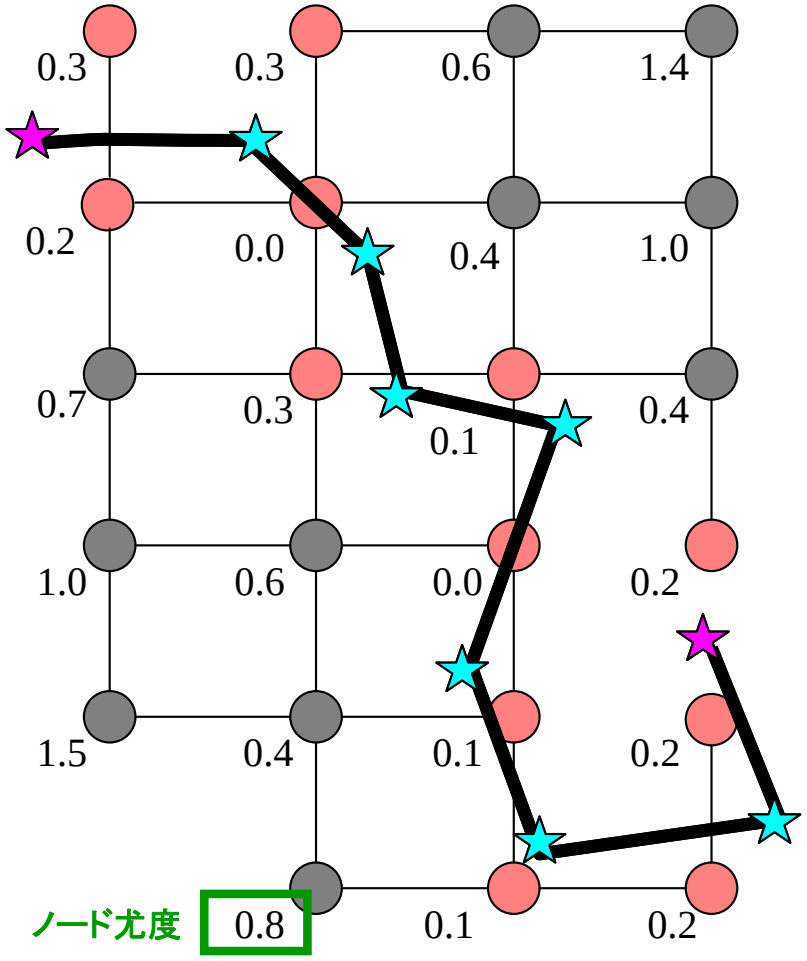
DPから各ノードまでの距離を計算し，最小値を各ノードのノード尤度とする．





○マップマッチング

【Step3】ノードの抽出



ノード尤度が閾値以下のノード

位置データ

ダイアリーによる施設データ

位置間直線経路

リンク

ノード

ノード抽出条件

$$\overline{NDP_i} \leq L_{\max}$$

$\overline{NDP_i}$ ：ノード尤度

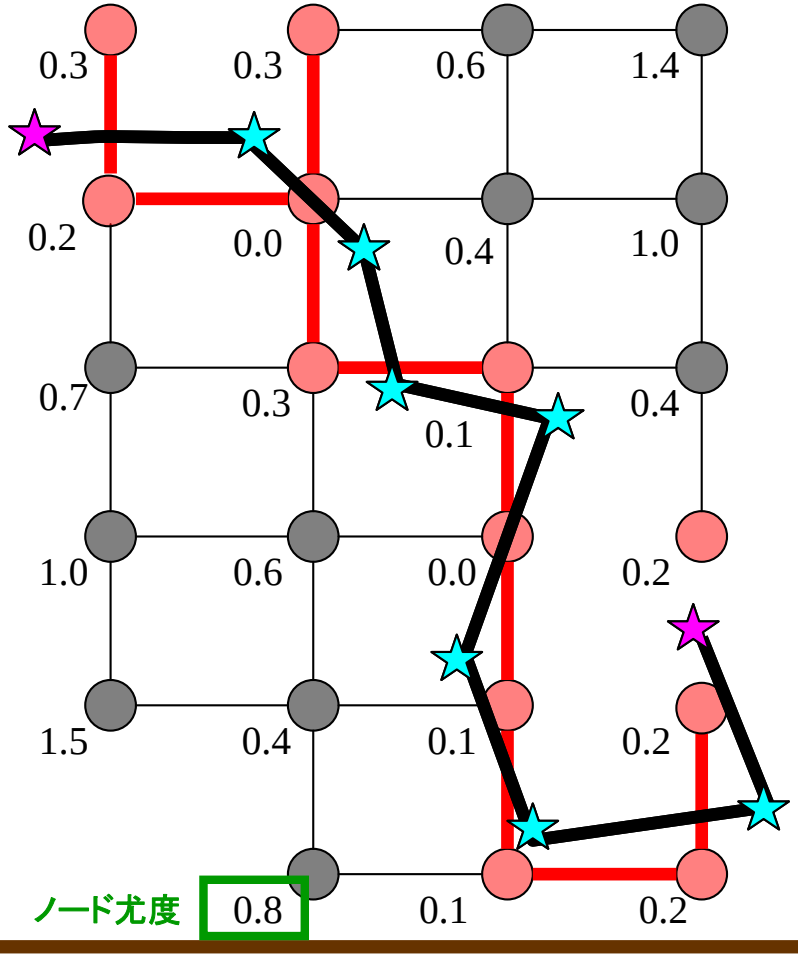
$L_{\max}$ ：閾値

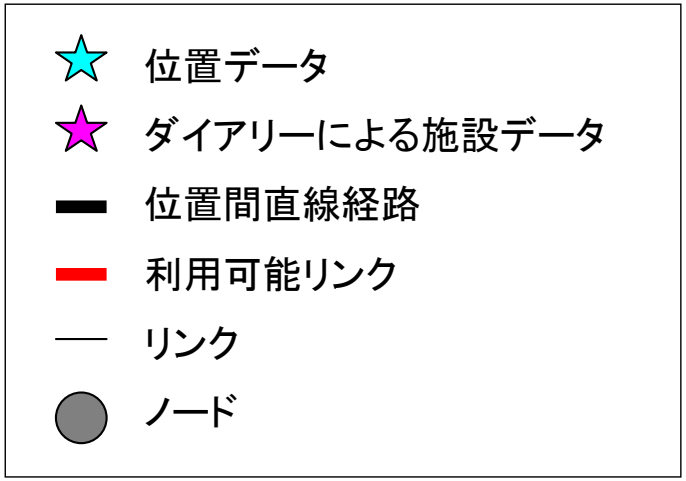
i ：ノード

○マップマッチング

【Step4】リンクの抽出

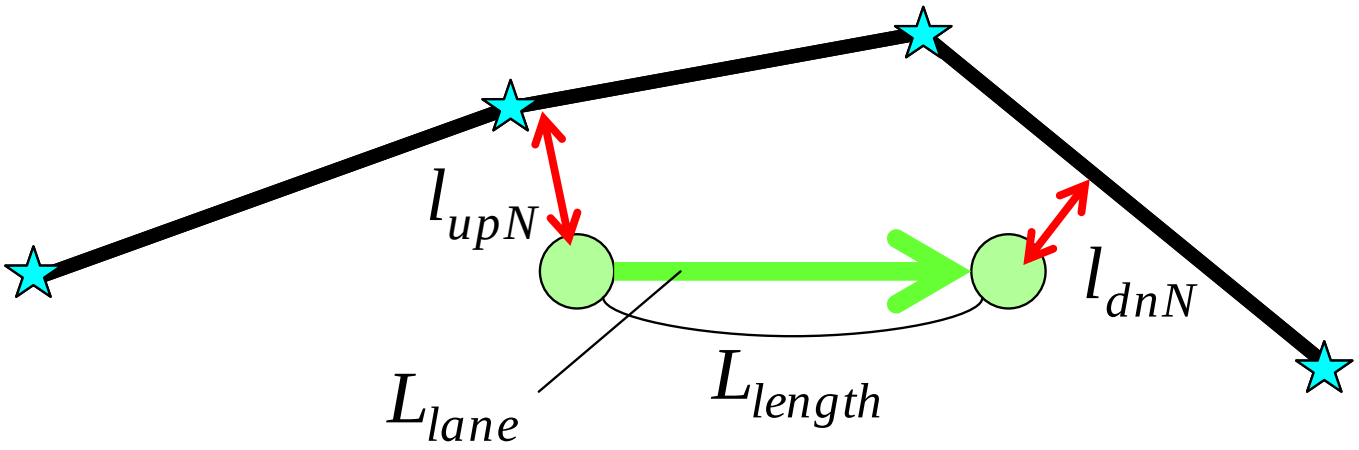
両端のノードが抽出されているリンクを抽出する.





○マップマッチング

【Step5】リンク尤度の算出



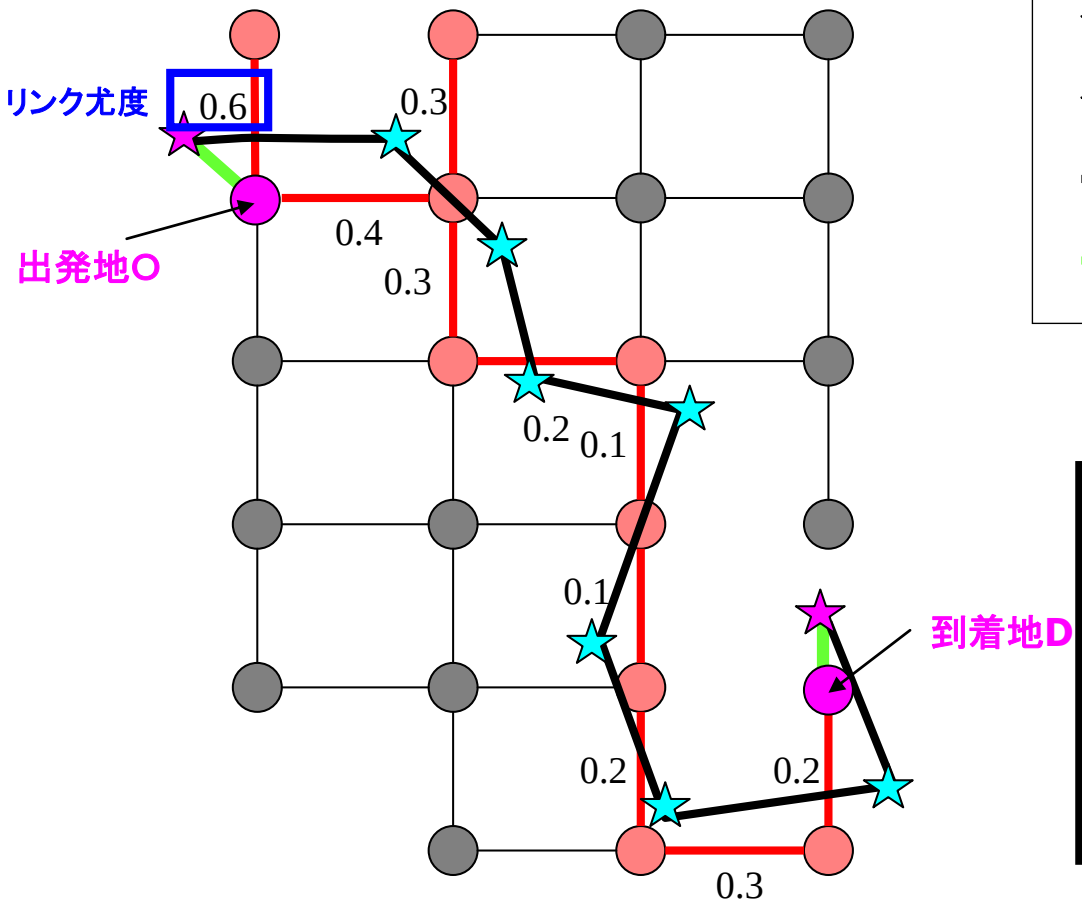
$$LL = (l_{upN} + l_{dnN}) * L_{length} / L_{lane}$$

- l_{upN} ：起点ノードから位置間直線経路までの距離
- l_{dnN} ：終点ノードから位置間直線経路までの距離
- L_{length}：リンク距離
- L_{lane} ：リンク車線数

○マップマッチング

【Step6】ODノードの決定

Webダイアリーによる施設データから最も近いノードをODノードとする.



位置データ

ダイアリーによる施設データ

位置間直線経路

ODノードからの距離

経路データ作成条件

$$Node_{Length} \leq Node_{\max Length}$$

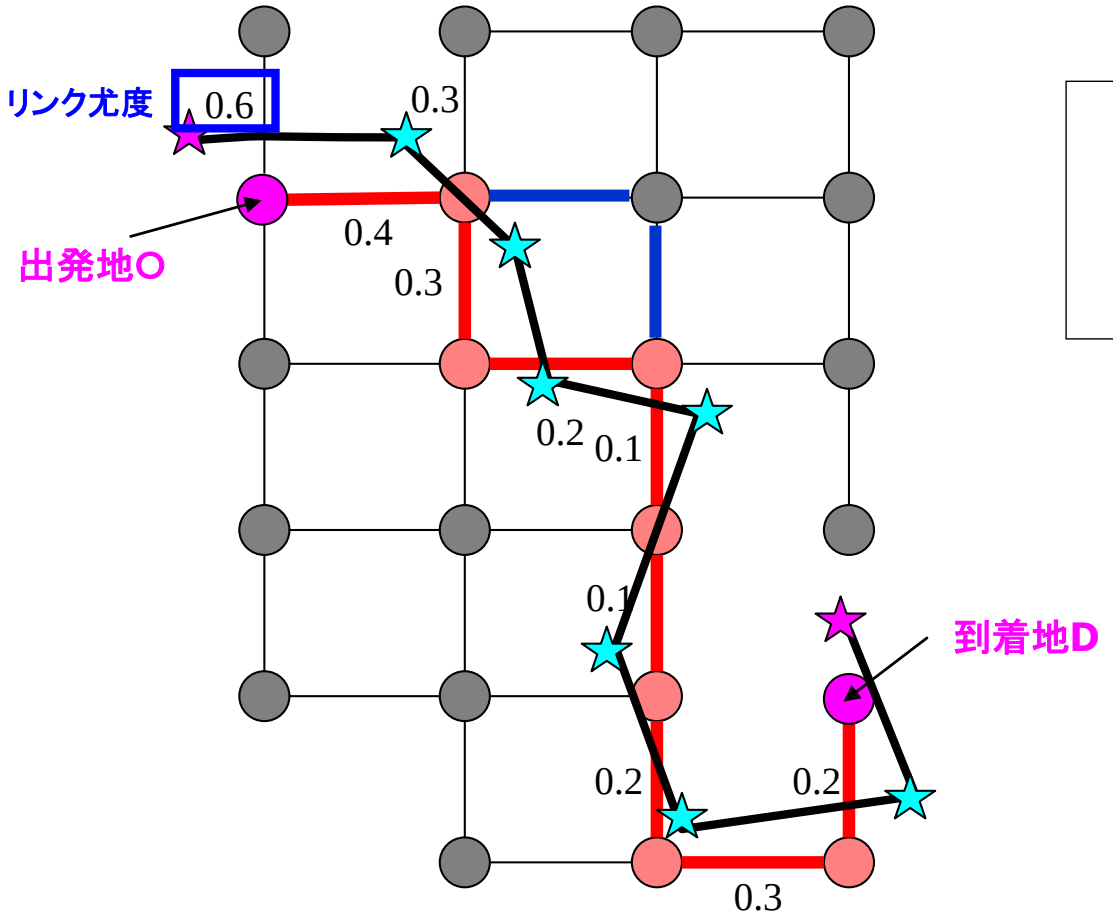
$Node_{Length}$:ODノードからの距離

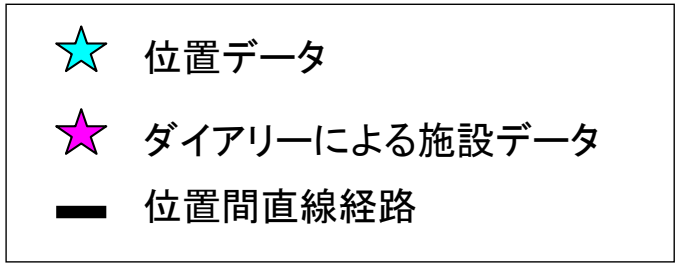
$Node_{\max Length}$: 閾値

○マップマッチング

【Step7】最短経路探索による経路の特定

リンク尤度をリンクコストとして最短経路探索（ダイクストラ法）により経路を特定する.

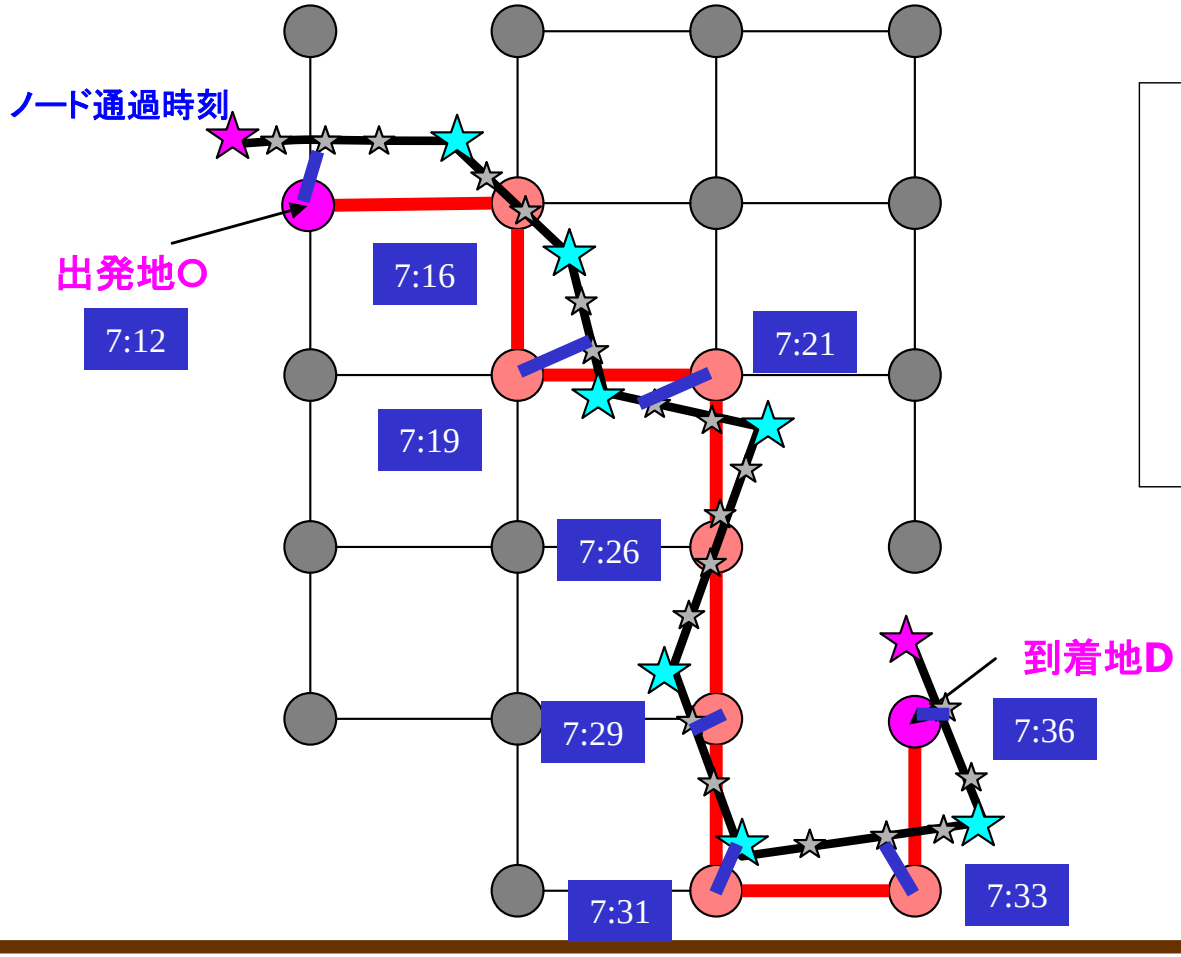




○マップマッチング

【Step8】リンク旅行時間の算出

ノードに最も近い点をノード通過時刻として，リンク所要時間を算出

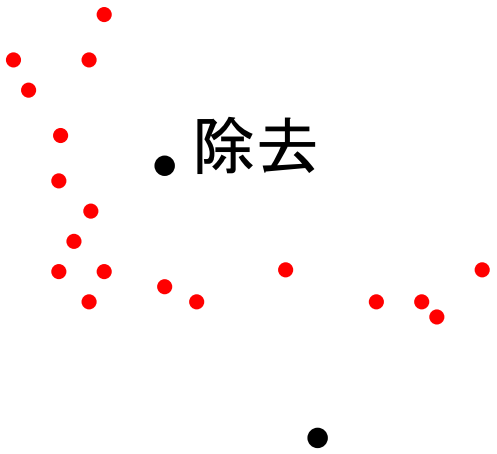


○マップマッチング

▼マップマッチング精度向上のために

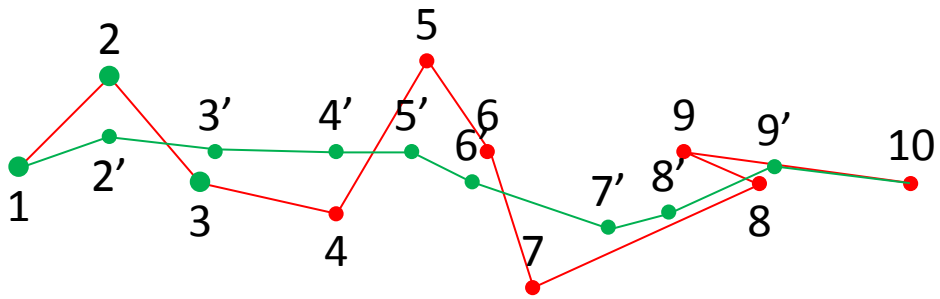
- 特異点除去

  前の点からXm以上離れた
  点を削除など



- 移動点平均

  前後X点を含めた平均をとる



赤：補正前
緑：補正後

○マップマッチング

▼ODの与え方

・出発・到着地はどこか

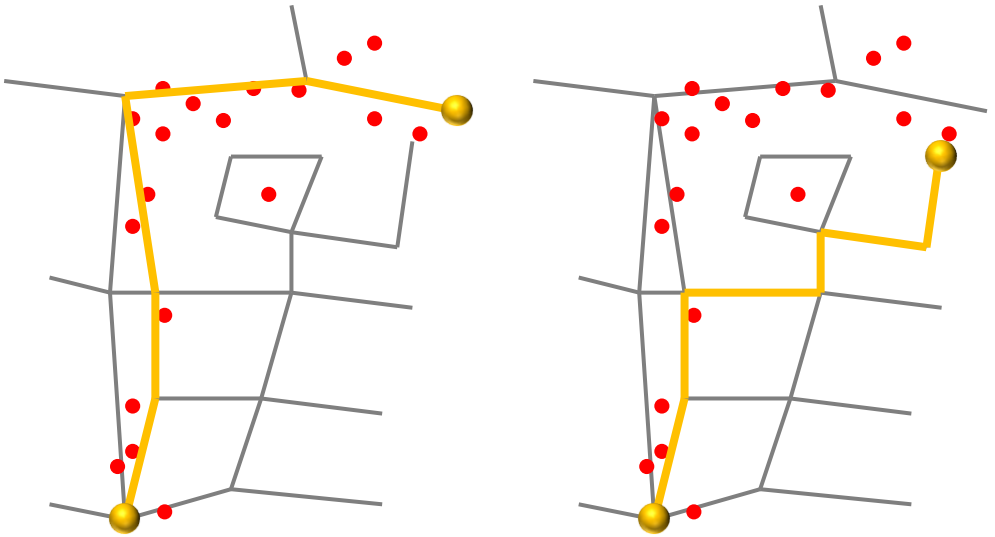
ダイアリデータの出発・到着施設？

トリップで取得された最初と最後の位置データ？

・ネットワークとの関係性

発着ノードはどこ？

結果に大きく
影響することも。



○マップマッチング

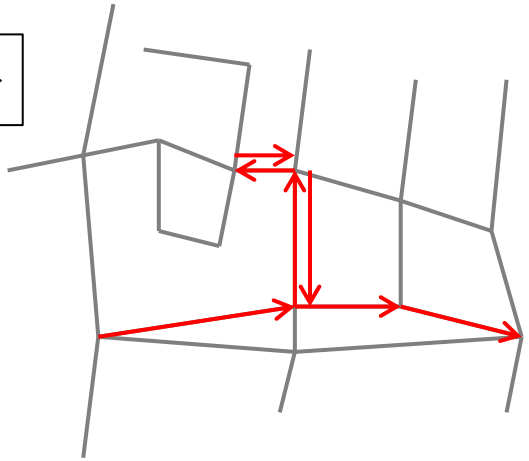
▼マップマッチングの課題

・折返し・ループ経路が捨象される

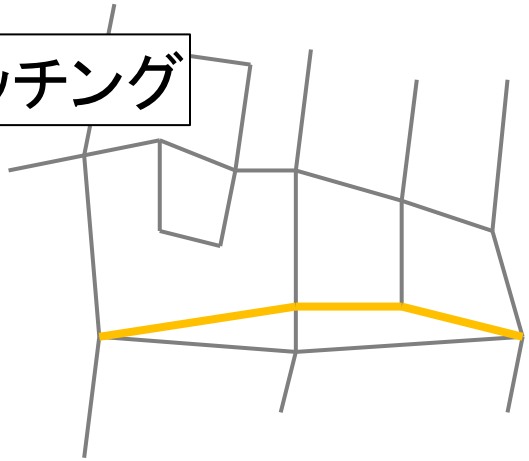
避難行動・回遊行動ではよくあるトリップ

→トリップ分割などの必要性

実際

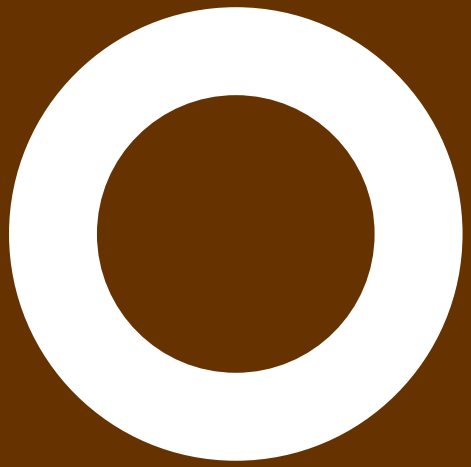


マッチング



・リンク内の挙動を再現できない

→計算方法の改良



Rの関数

○第2回の復習

RのMNLモデル推定コードを動かした
推定でどんな計算を行っているか、
関数の計算をもう少し詳しくみてみる。

```
### Logit model estimation

### データファイルの読み込み
Data <- read.csv("C:/R/kari_sapporomodel.csv",header=T)

## データ数:Dataの行数を数える
hh <- nrow(Data)


## 今回用いる選択肢の数
ch <- 3

## パラメータの初期値の設定
b0<-c(0, 0, 0, 0)
# b0 <- numeric(6)


### Logit model の対数尤度関数の定義

fr <- function(x) {
  ### パラメータの宣言:
  ## 定数項
  b1 <- x[1]
  b2 <- x[2]


  ## 車線数
  b3 <- x[3]


  ## リンク長
  b4 <- x[4]

  ##方向保持
  ##b5 <- x[5]

  ## 対数尤度のための変数を宣言
  LL=0


  ### 今回用いる目的地は以下の5つ.
  ## 右折
  ## 直進
  ## 左折


  ## 効用の計算:説明変数にしたい列を入れる.
                                                                                  # 定数項
  rightturn  <- Data$利用可能性rightturn*exp(b3*Data$車線数rightturn  +b4*Data$リンク長righttu
  straight    <- Data$利用可能性straight*exp(b3*Data$車線数straight +b4*Data$リンク長straight
  leftturn    <- Data$利用可能性leftturn*exp(b3*Data$車線数leftturn +b4*Data$リンク長leftturn)
```

○関数の定義

関数の
つくり方：

```
関数名 <- function(引数) {
   (関数の内容)
}
```

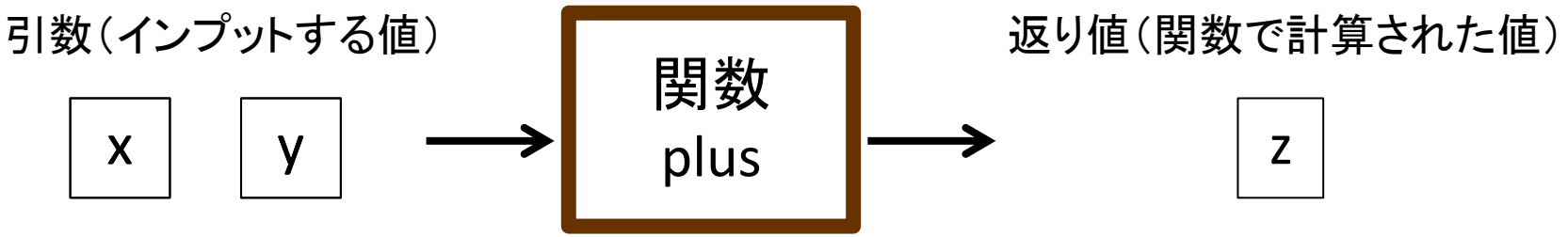
（例） xとyを足し算する関数をつくる

```
plus <- function(x,y){
   z <- x + y
   z
}
```

←plusという名の関数、x・yが引数

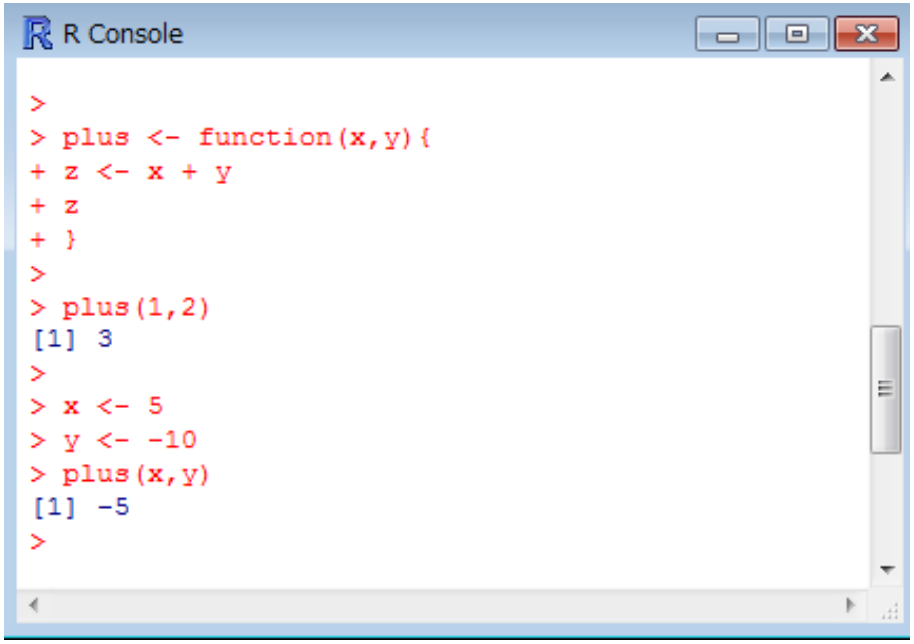
←最後の実行結果が返り値になる

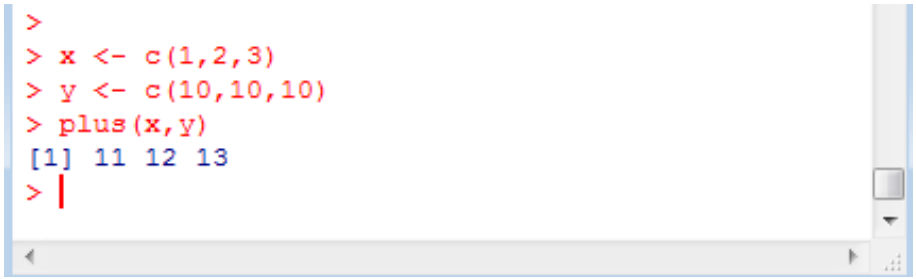
イメージ



○関数の定義

Rで入力してみる





(1) まずは関数を定義

(2) 数字を引数にして与えてみる

(3) 変数を引数にして与えてみる

(4) ベクトルを引数にして
与えてみる

○関数の定義

第2回の推定コードでは・・・

```
### Logit model の対数尤度関数の定義
fr <- function(x) {
  ### パラメータの宣言
  ## 定数項
  b1 <- x[1]
  b2 <- x[2]
  b3 <- x[3]
  b4 <- x[4]
  ## 異なる
  d1
  Pjusco   <- (Pjusco  !=0)*Pjusco
  Pjoepla  <- (Pjoepla !=0)*Pjoepla

  ## 選択結果
  Ctakashi <- Data[,7]==5212
  Cmitsuko <- Data[,7]==5213
  Cmatsu   <- Data[,7]==5202
  Cjusco   <- Data[,7]==5272
  Cjoepla  <- Data[,7]==5303

  ## 対数尤度の計算
  LL <- colSums(Ctakashi*log(Ptakashi) + Cmitsuko*log(Pmit
                Cmatsu  *log(Pmatsu)   + Cjusco  *log(Pjus
}
```

frという関数名、引数は推定するパラメータ変数であるベクトルx

ロジットモデルでの選択確率の計算

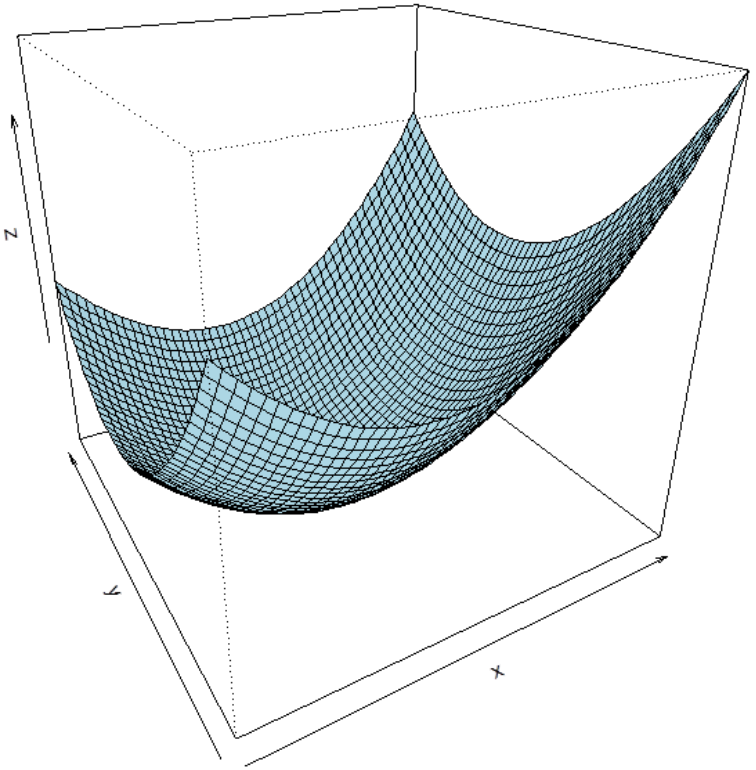
最終行での対数尤度の計算の結果LLが関数の返り値になる

○最適化関数

▼関数の最小化（最大化）計算

例： $f(x, y) = (x - 2)^2 + (y - 3)^2$ の最小値は？

xとyについて[0,5]の
区間で $z = f(x, y)$ として
3次元描画した図



Rの最適化関数を使って求めてみる

○最適化関数

最適化関数optim()の使い方：

```
optim(par, fn, gr = NULL, method = "BFGS",
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

par	初期値（必須）
fn	最適化する目的関数（必須）
gr	一階偏微分関数の指定（NULLでデフォルト関数使用）
method	最適化手法の指定（5種類から選ぶ）
	・Nelder-Mead（デフォルト）
	・BFGS
	・CG
	・L-BFGS-B
	・SANN

○最適化関数

最適化関数optim()の使い方：

```
optim(par, fn, gr = NULL, method = "BFGS",
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

lower	L-BFGS-B法での変数の下限（デフォルトは-Inf）
upper	L-BFGS-B法での変数の上限（デフォルトはInf）
control	制御パラメータ fnscale：関数に与える比例定数 optim関数は最小化を行うので、最大化のときは control=list(fnscale=-1)と指定する
hessian	最適解のヘッセ行列を返すかどうか パラメータ推定では、t値計算にヘッセ行列が必要 なので、hessian = TRUE

○最適化関数

▼簡単な最適化計算をRでやってみる

例： $f(x, y) = (x - 2)^2 + (y - 3)^2$ の最小値は？

```
f <- function(par){
   (par[1] - 2)^2 + (par[2] - 3)^2
}
optim( c(0,0) , f , method = "BFGS" )
```

・関数fで関数を定義、このとき変数は2変数なのでpar(x,y)の形でベクトル化して何番目の要素かで表現
・optim関数で初期値(x,y)=(0,0)を与えてfを最小化
（Rではベクトルはc(要素,要素,…)で定義される）

○最適化関数

例: $f(x, y) = (x - 2)^2 + (y - 3)^2$ の最小値は？

```
> f <- function(par){
+    (par[1] - 2)^2 + (par[2] - 3)^2
+ }
> optim( c(0,0) , f , method = "BFGS")
$par
[1] 2 3

$value
[1] 1.577722e-30

$counts
function gradient
      10        3

$convergence
[1] 0

$message
NULL
```

結果の解釈

par	最適解(x,y)=(2,3)
value	最適値 $1.58 \times 10^{-30} \fallingdotseq 0$
counts	
function	BFGS法の繰り返し計算10回
gradient	1階偏微分の計算3回
convergence	0ならば収束している
message	その他のメッセージなし

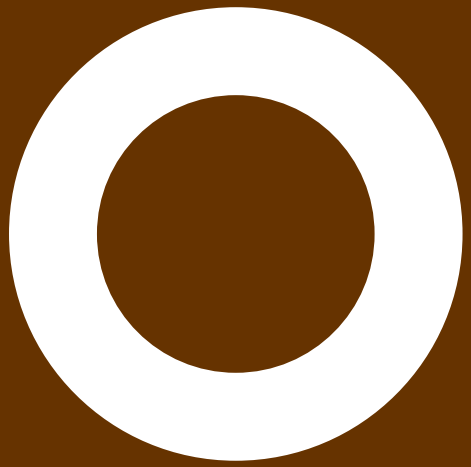
○最適化関数

第2回の推定コードでは・・・

```
## 対数尤度関数frの最大化
res <- optim(b0,fr, method = "BFGS", hessian = TRUE, control=list(fnscale=-1))
```

```
optim(b0, fr, method = "BFGS", hessian = TRUE
      control=list(fnscale=-1))
```

b0	初期値にベクトルb0を指定
fr	対数尤度関数frを最適化する
method	BFGS法での最適化計算を指定
hessian	ヘッセ行列を返す
control	対数尤度を最大化したいので、スケールに-1を指定



おわり