ブラウニアンマップの構成

ryamada

*2017*年*8*月*26*日

1 ブラウニアンマップの構成手順

ブラウニアンマップはS2同相なランダムな曲面。

このブラウニアンマップが、以下に示すランダムな木グラフ作成とそのグラフのノードへのランダムな数値ラベル付与をしたものに定義づけた距離関係情報の極限であることが知られている。

1.1 ランダムな木グラフの作成

0からスタートし0に戻る1次元ブラウン運動のうち、0より大の値のみを取るそれを、ブラウン散歩(Brownian excursion)と呼ぶ。

このブラウン散歩から木グラフを構成することができる(構成法は後述する)。

この木グラフが、ブラウニアンマップ構成の1段階目である。

1.2 木構造を周回コースとしてのグラフ

作成したランダムな木グラフに沿って、ブラウン散歩を実行すると、すべてのエッジは両方向に1回ずつ、計2回、歩まれる。

エッジを2回歩くことを、エッジの両側面を歩くことだとみなすと、ランダム木グラフ上のブラウン散歩は、ルートノードから、木の周囲をぐるりと歩くことに相当する。

この周回コースは有向グラフとみなせる。

1.3 ランダム数値のラベル付与

木グラフの直線成分ごとに酔歩をすることで、木グラフのノードにランダムな値が付与できる。

1.4 周回グラフ上の点の間の距離関係

周回グラフでは木グラフ上は同一の点であっても周回路の上では異なる点に相当することがある。 ブラウニアンマップの構成は、この周回グラフ上の点の間の距離関係を定めることにより実現される。

木グラフのノードに付与された値(この値は、周回グラフでは区別されていても、木の上で同一であれば、同じ値を持つことになる)と、木の構造とからノード間距離を定めるルールを導入する。このルールを定めることで、その木グラフが配置された二次元面に距離空間ができる。

この距離空間がブラウニアンマップである。

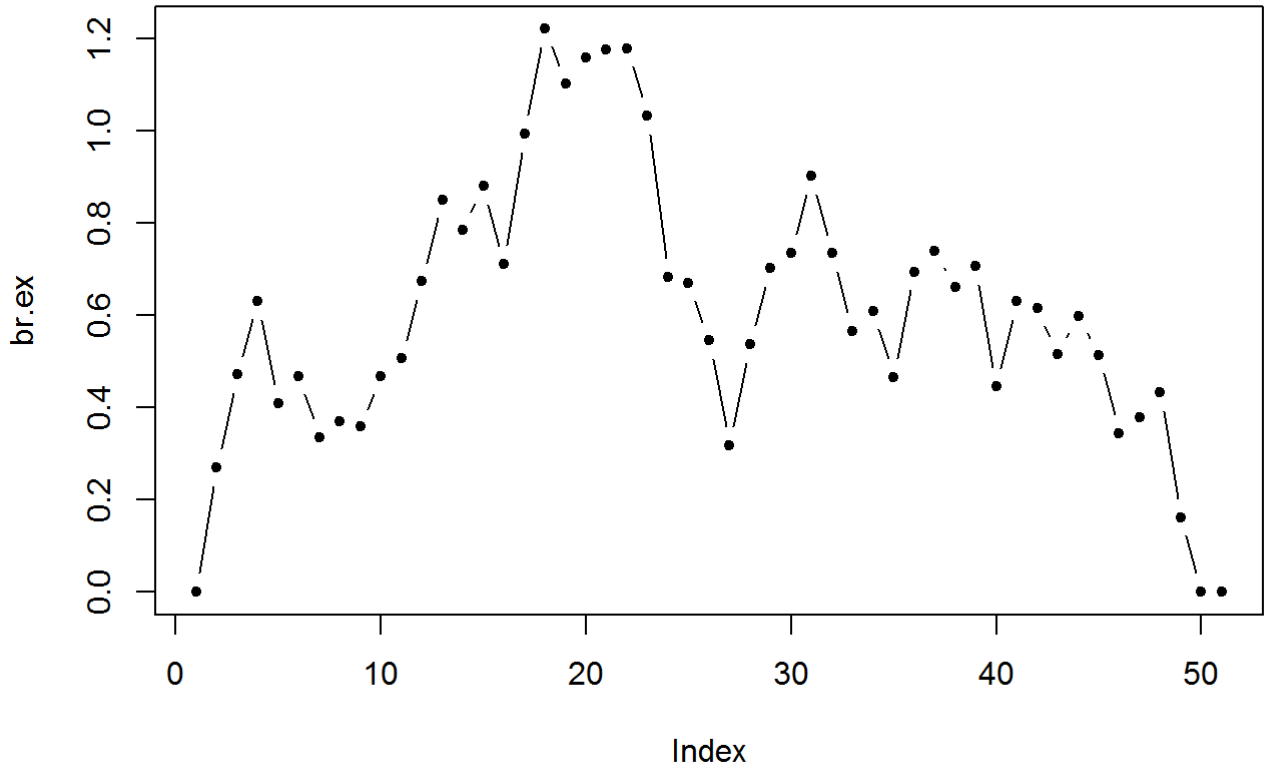
2 ブラウン散歩によるランダムな木グラフの構成

2.1 ブラウン散歩

0を出発して0に戻るブラウン運動はブラウニアンブリッジ(ブラウン橋)と言う。

ブラウン散歩のシミュレーション作成は、ブラウン橋を作成し、正の領域のみを通過するものができるまで、作成を繰り返すことで実現する。

```
# Wiener bridge作成関数を持つパッケージ
#library(e1071)
my.rexcursion <- function(frequency = 1000, end = 1){
    succeeded <- FALSE
    while(!succeeded)
    {
        bridge <- rbridge(end = end, frequency = frequency)
        succeeded=all(bridge>=0)||all(bridge<=0)
    }
    return(c(0,c(abs(bridge))))
}
my.rwiener <- function(frequency = 1000, end = 1){
    c(0,cumsum(rnorm(end * frequency)/sqrt(frequency)))
}

fr <- 50
br.ex <- my.rexcursion(frequency=fr)
plot(br.ex,pch=20,type="b")
```



2.2 ブラウン散歩から木を構成する

2.2.1 ブラウン散歩上の点間距離の定義

ブラウン散歩で座標が大きくなるときには、未踏のエッジを作成し、 座標が小さくなるときには、来た道を戻ることを繰り返す。

このような木を作成するにあたり、以下の方法を採用する。

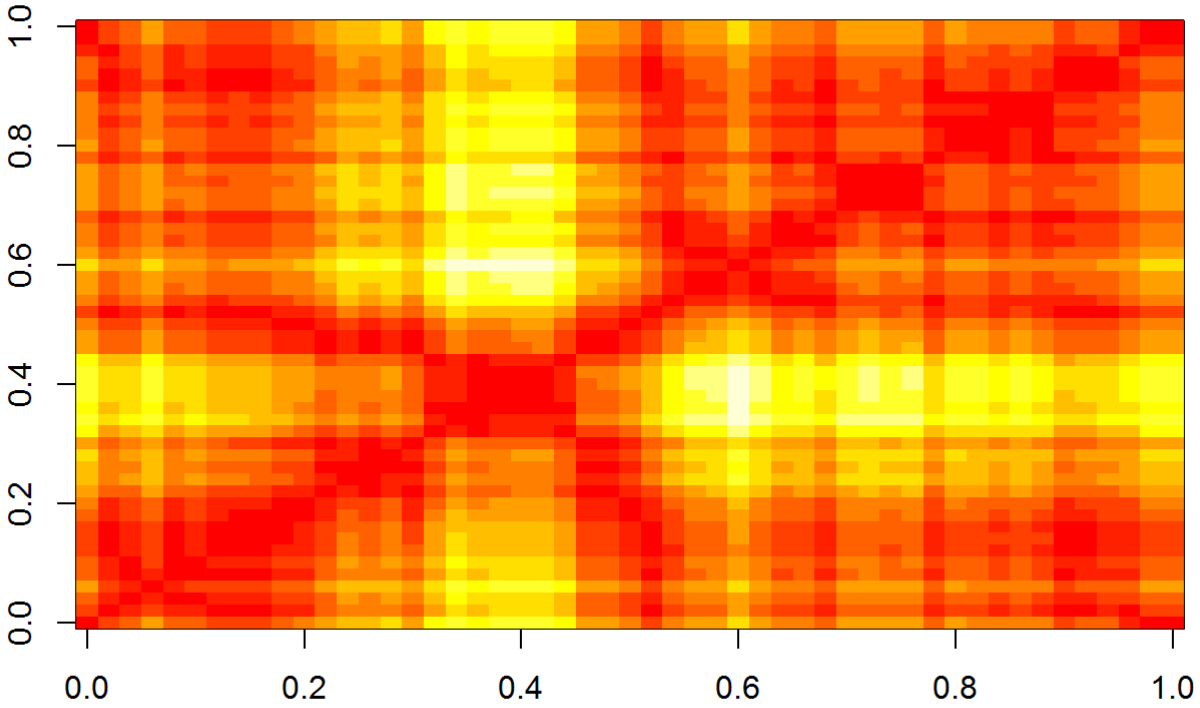
ブラウン散歩e上の2点s, t間の距離$d_e(s,t)$を以下のように定める。

$$d_e(s,t) = x(s) + x(t) - 2 \times min_{u \in [s,t]} x(u)$$

ただし、$x(u)$はパラメタuでのブラウン散歩の座標とする。

ブラウン散歩上の点間の距離$d_e(s,t)$が定まった。

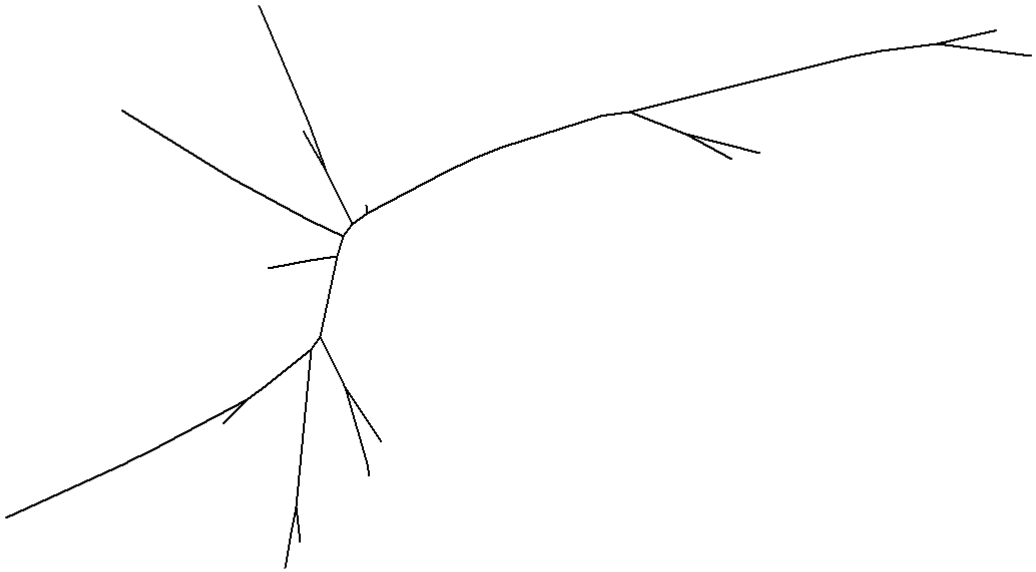
```
my.Rtree.dist <- function(g,s,t){
    g[s] + g[t] - 2 * min(g[s:t])
}

my.Rtree.dist.mat <- function(g){
    L <- length(g)
    D.mat <- matrix(0,L,L)

    for(i in 1:(L-1)){
        for(j in (i+1):L){
            D.mat[i,j] <- D.mat[j,i] <- my.Rtree.dist(g,i,j)
        }
    }
    return(D.mat)
}
D.mat <- my.Rtree.dist.mat(br.ex)
image(D.mat)
```



2.2.2 NJ法による木構造の作成

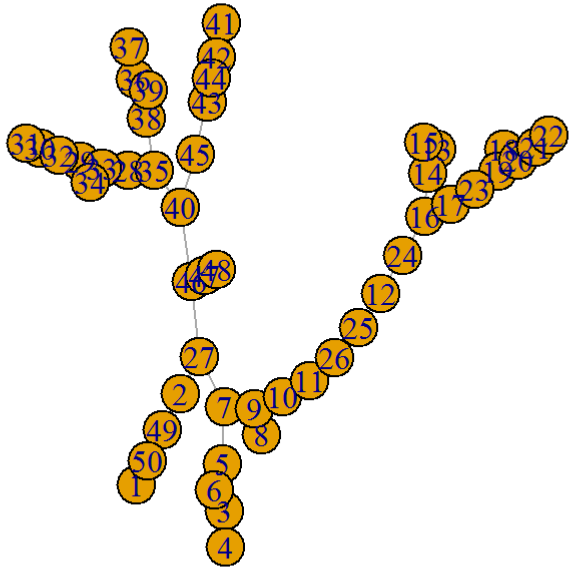
これを満足する木構造が存在することが知られているので、それを満足する木グラフをNJ法にて構成することができる。

```
# nj法の関数を持つパッケージ
#library(ape)
tr <- nj(D.mat)

plot(nj(my.Rtree.dist.mat(br.ex)),type="u",show.tip.label=FALSE)
```



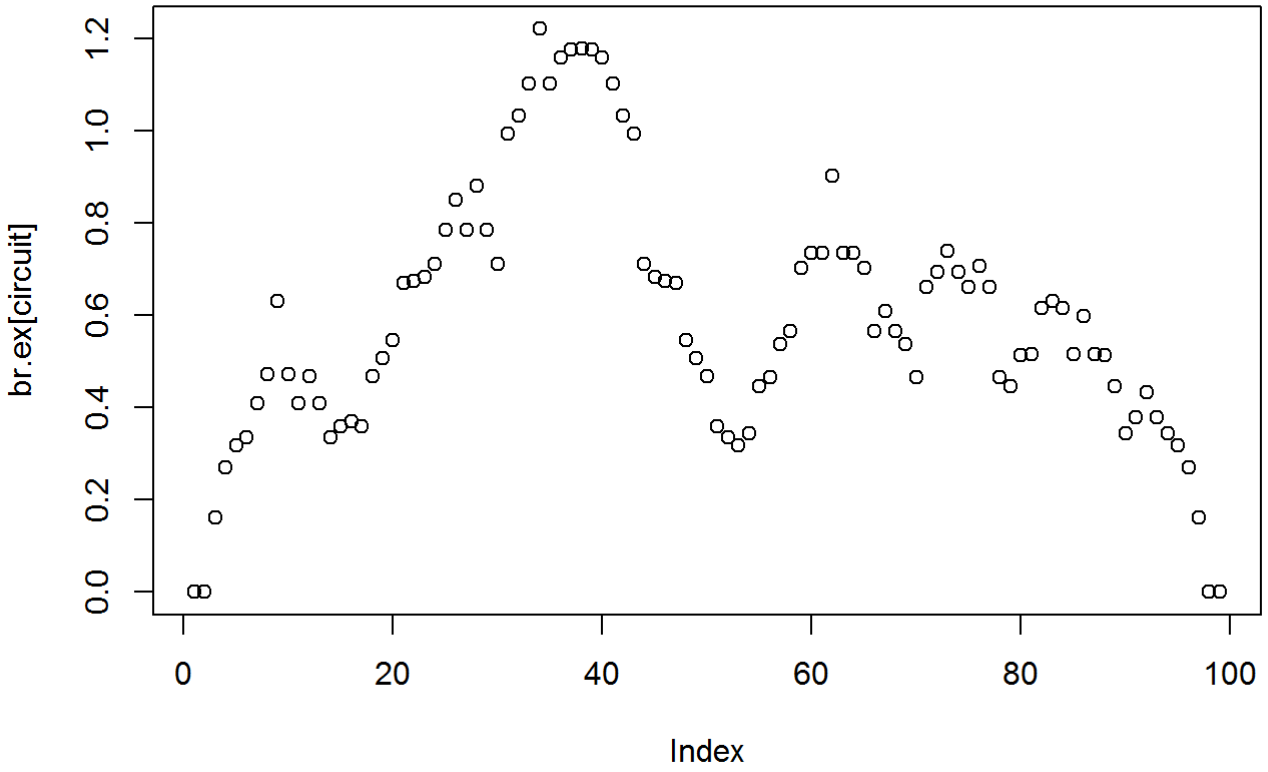
この木構造では、ブラウン散歩上の点に多数のノードが追加されるが、それらは、散歩上の点であることも知られているから、以下のような要領で、追加ノードを散歩上の点に対応するノードに変換する。

```
# library(igraph)
my.extree <- function(br.ex,minval = 10^(-15)){
    n <- length(br.ex)
    D.mat <- my.Rtree.dist.mat(br.ex)
    tr <- nj(D.mat)
    el <- tr$edge
    el[which(el==n)] <- 1
    L <- tr$edge.length
    shortL <- (L < minval)
    while(max(el) > n){
        el <- t(apply(el,1,sort))
        tmp <- el - (n+0.5)
        tmp2 <- tmp[,1] * tmp[,2]
        tmp3 <- which(shortL & (tmp2 < 0))
        for(i in 1:length(tmp3)){
            el[which(el==el[tmp3[i],2])] <- el[tmp3[i],1]
        }
    }
    el.true <- which(el[,1]!=el[,2])
    el2 <- el[el.true,]
    L2 <- L[el.true]
    return(list(el=el2,len=L2))
}


extree <- my.extree(br.ex)
g <- graph.edgelist(extree$el,directed=FALSE)
g$weight <- extree$len
plot(g)
```

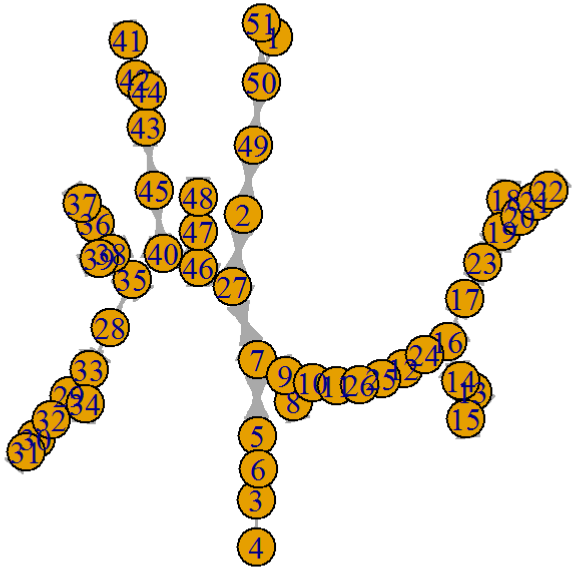


3 木の周り周回コースとしてのグラフ

```
my.circuit.extree <- function(br.ex){
    extree <- my.extree(br.ex)
    g <- graph.edgelist(extree$el,directed=FALSE)
    n <- length(br.ex)
    ret <- c(1)
    for(i in 1:(n-2)){
        tmp <- shortest_paths(g,from=i,to=i+1)[[1]][[1]]
        ret <- c(ret,tmp[-1])
    }
    tmp <- shortest_paths(g,from=n-1,to=1)[[1]][[1]]
    ret <- c(ret,tmp[-1])
    ret[length(ret)] <- n
    return(ret)
}

circuit <- my.circuit.extree(br.ex)
plot(br.ex[circuit])
```




```
g.circuit <- graph.edgelist(cbind(circuit[1:(length(circuit)-1)],circuit[2:length(circuit)]))
plot(g.circuit) # 第１ノードと最終ノードは別ノードとして扱っているが、同一地点に相当することに注意
```



```
circuit
```

```
##  [1]  1 50 49  2 27  7  5  3  4  3  5  6  5  7  9  8  9 10 11 26 25 12 24
## [24] 16 14 13 14 15 14 16 17 23 19 18 19 20 21 22 21 20 19 23 17 16 24 12
## [47] 25 26 11 10  9  7 27 46 40 35 28 33 29 32 30 31 30 32 29 33 34 33 28
## [70] 35 38 36 37 36 38 39 38 35 40 45 43 42 41 42 43 44 43 45 40 46 47 48
## [93] 47 46 27  2 49 50 51
```

4 ランダムな木グラフのノードへのランダムな値付与

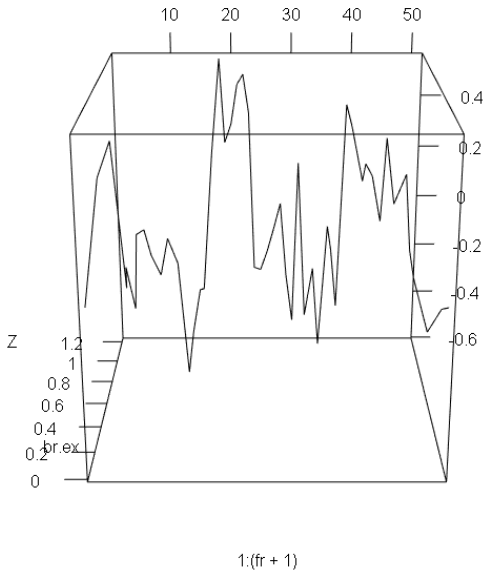
木グラフのエッジの長さに応じて1次元ブラウン運動をさせ、それにより、エッジごとにエッジ始点からエッジ終点までの値の相対的増減値を発生させる。

その値を用いて、ルートノードの値を0としたときの各ノードの値を算出する。

```
my.edgeWiener <- function(lens,frequency=1000){
    ret <- rep(0,length(lens))
    for(i in 1:length(ret)){
        tmp <- rwiener(lens[i],frequency=frequency/lens[i])
        ret[i] <- tmp[length(tmp)]
    }
    return(ret)
}
my.edgeWiener(extree$len)
```

```
##  [1]  0.100725478 -0.087853713 -0.019420842 -0.019758987  0.113740206
##  [6] -0.076505147  0.151081702  0.641249564  0.090494262  0.042445182
## [11] -0.123777236 -0.429639007 -0.604435266  0.177486093  0.152113764
## [16]  0.339003629 -0.210581789  0.084495173  0.562325064 -0.004512839
## [21] -0.289511041  0.055472887 -0.142720792 -0.064407796  0.015046887
## [26] -0.044603321  0.150328769 -0.026479083 -0.154726070  0.019393483
## [31]  0.042367280 -0.249433434 -0.061926828 -0.824575579  0.098858461
## [36] -0.018789771  0.145869738  0.052719669 -0.156384353  0.066004291
## [41]  0.094863225  0.160124081 -0.143728681 -0.154107351  0.152556563
## [46] -0.068618990  0.296581865 -0.713453242  0.023499753
```

```
my.extreeZ <- function(br.ex){
    extree <- my.extree(br.ex)
    g <- graph.edgelist(extree$el,directed=FALSE)
    edgeZlen <- my.edgeWiener(extree$len)
    path.from.1 <- shortest_paths(g,from=1,output="epath")
    Z <- rep(0,length(br.ex)-1)
    for(i in 2:length(Z)){
        tmp <- as_ids(path.from.1[[2]][[i]])
        Z[i] <- sum(edgeZlen[tmp])
    }
    Z <- c(Z,0)
    return(Z)
}

Z <- my.extreeZ(br.ex)
plot3d(1:(fr+1),br.ex,Z,type="l")
```



5 周回グラフ上の点間距離

周回グラフ上の点間距離(擬距離)は2段階の定義になっている。

5.1 D°

これは、ブラウン散歩のときに散歩座標から木グラフを作るときに導入した距離に似ている。

違いは、周回グラフであるので、順方向・逆方向の両方向を考慮して、短くなる方を採用するようになっていることである。

今、周回グラフがあって、ノードp, qに値$Z(p), Z(q)$がラベル付けされているとする。

この2点間に以下のような距離を定める。

$$D^\circ(p,q) = Z(p) + Z(q) - 2 \times max(min_{r \in \{p \to q\}} Z(r), min_{r \in \{q \to p\}} Z(r))$$

ただし$r \in \{p \to q\}$はpからqへと時計回りに巡回したときの周回グラフの部分を表し、 $r \in \{q \to p\}$はqからpへと時計回りに巡回したときの周回グラフの部分を表す。

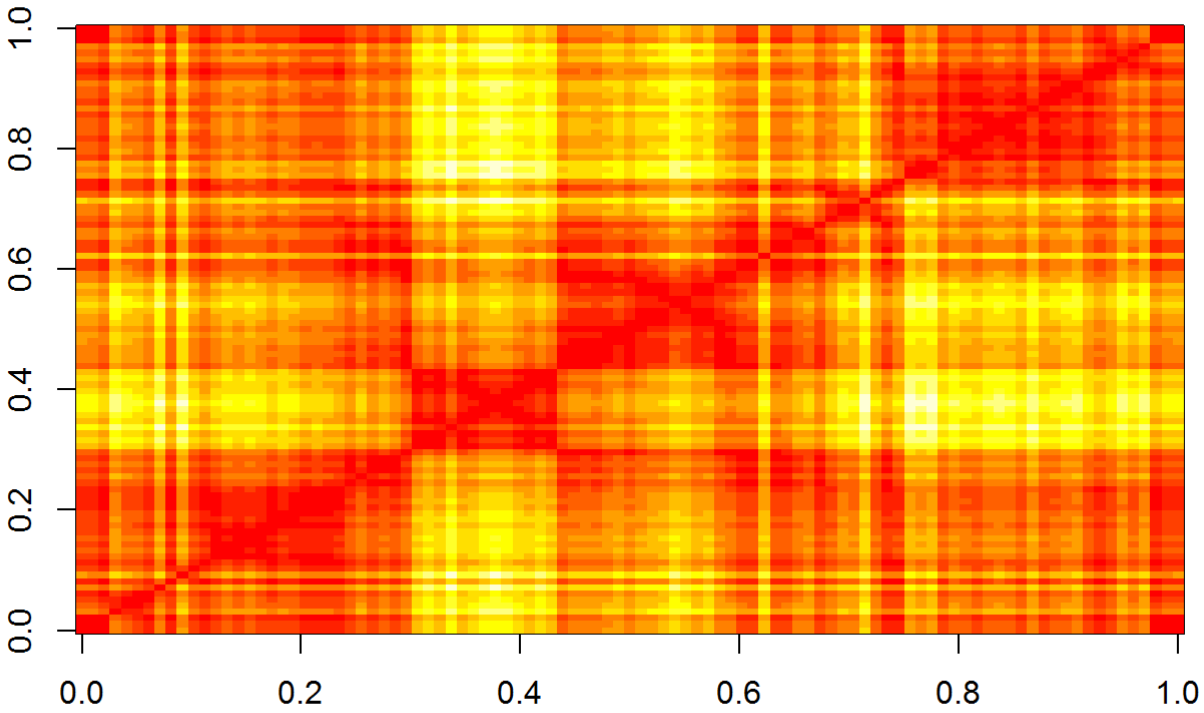
2つの周回路でのZ値の最小値のうち大きい方を取ることで、2点間の距離としては、「短め」になるような定義になっている。

$Z(p) = Z(q)$であって、どちらかの周回路での最小値が$Z(p) = Z(q)$であるときに、$D^\circ(p,q) = 0$となることもわかる。

それ以外の場合は正の値を取る。

以下で、すべてのノード間のD°を計算し、その行列を作成する。

```
my.D.circuit <- function(v,Z,i,j){
    n <- length(v)
    Z. <- Z[v]
    vij <- i:j
    vji <- c(1:n,1:n)[j:(length(v)+i)]
    tmp <- max(min(Z.[vij]),min(c(Z.,Z.)[vji]))
    return(Z.[i]+Z.[j]-2*tmp)
}
# すべての
my.D.circuit.mat <- function(v,Z){
    n <- length(v)
    ret <- matrix(0,n,n)
    for(i in 1:(n-1)){
        for(j in (i+1):n){
            ret[i,j] <- ret[j,i] <- my.D.circuit(v,Z,i,j)
        }
    }
    return(ret)
}

Dcmat <- my.D.circuit.mat(circuit,Z)
image(Dcmat)
```



5.2 $D(p,q)$

D° 自体を用いると、

$max(min_{r\in\{p\to q\}}Z(r), min_{r\in\{q\to p\}}Z(r))$の値が小さくなることがあり(周回路上に小さなZ値が出てくるため)、結果として、ある2点間の距離(擬距離)が長くなり過ぎて、三角不等式を満足することがなくなるなど、不具合があるという。

それを回避するために、D°を使いつつ、異なる定義をする。

ブラウン散歩にて同一視された点s,tは$d_e(s,t)=0$の関係に関係にある、と表せるが、そのような点なら、バイパスしてもよいと言うルールで、以下のように定める。

$$D(s,t) = Inf(\sum_{i=1}^{k} D^{\circ}(s_i,t_i)); s_1 = s, t_k = k, d_e(s_{i+1},t_i) = 0$$

ただし、kの値は1でも良いし、他のどんな正の整数でもよいものとする。

$k = 1$のときには、バイパスはせずに、$D^{\circ}(s,t)$を用いるし、$k > 1$のときには、どこかしらの$d_e(s_{i+1},t_i) = 0$な関係にある周回2点t_i, s_{i+1}でバイパスをする。

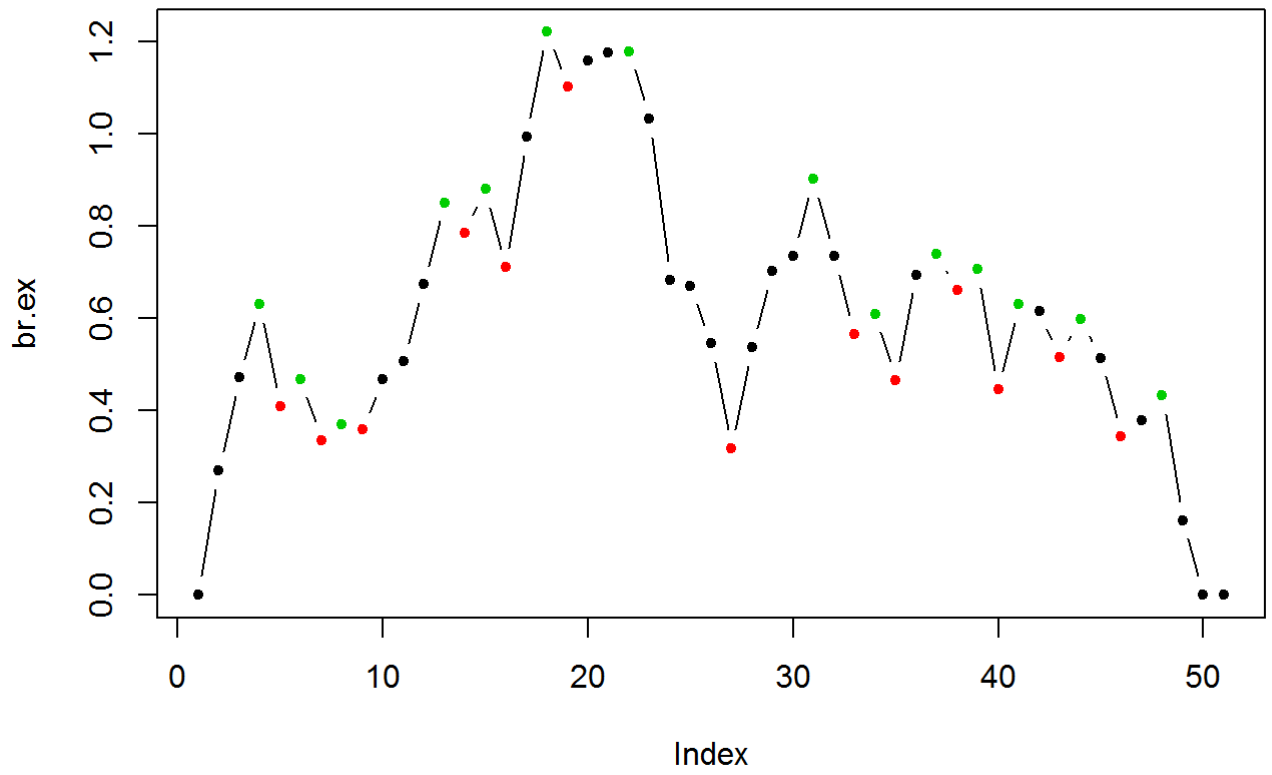
このInfを取るにあたり、s,tの2点と、$d_e(s_{i+1},t_i) = 0$を満足することから、バイパス候補になる点のみを取り出して、そのペア間にD°の重みを持つ完全グラフを作成した上で、s,t間の最短距離を求める。

$d_e(s,t)$なる点の組み合わせを列挙する。

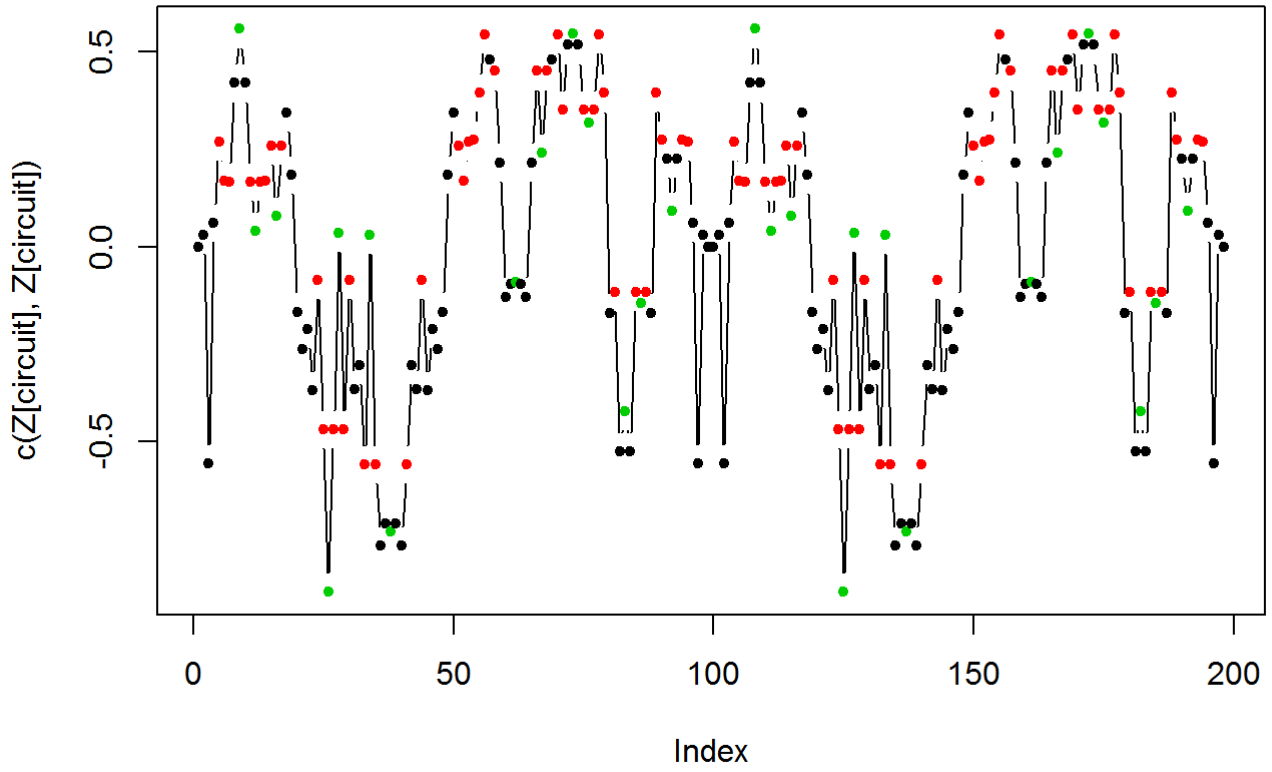
```
Dcmat <- my.D.circuit.mat(circuit,Z)

Z <- my.extreeZ(br.ex)
```

まず、ブラウン散歩で下りから上りに転ずる点を識別する

```
# d_e(s,t)=0のIDを取り出す
my.exUpDown <- function(br.ex){
    L <- length(br.ex)
    d <- diff(br.ex)
    #print(d)
    d.sign <- sign(d)
    #print(d.sign)
    d.d.sign <- diff(d.sign)
    dd.positive <- which(d.d.sign>0) + 1
    dd.negative <- which(d.d.sign<0) + 1
    #print(dd.positive)
    #print(dd.negative)
    col <- rep(1,L)
    col[dd.positive] <- 2
    col[dd.negative] <- 3
    #plot(br.ex,col=col,pch=20,type="b")
    return(col)
}
col <- my.exUpDown(br.ex)
plot(br.ex,col=col,pch=20,type="b")
```




```
plot(c(Z[circuit],Z[circuit]),type="b",pch=20,col=c(col[circuit],col[circuit]))
```



この識別情報から、周回路のノードIDで同等扱いするノードペアを列挙する

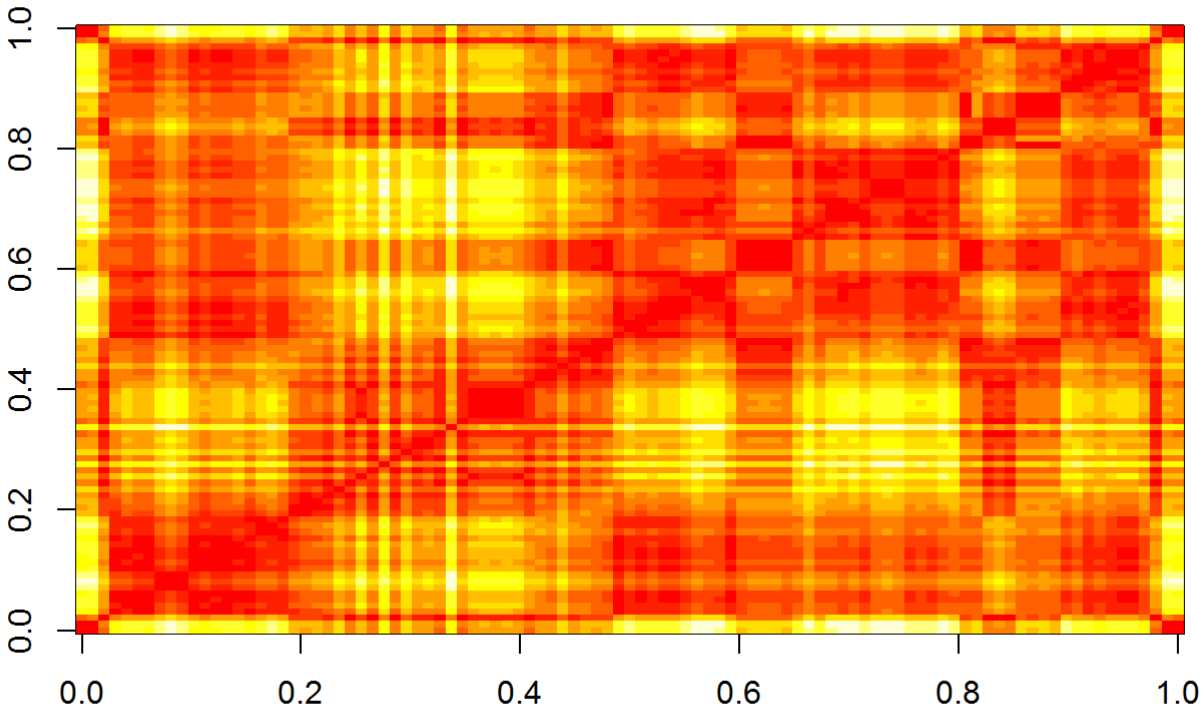
```
valleyID <- which(my.exUpDown(br.ex)==2)
# d_e(s,t)に相当するcircuitノードペアを作る
my.valley.ids <- function(br.ex){
  valleyID <- which(my.exUpDown(br.ex)==2)
  el <- matrix(0,0,2)
  for(i in 1:length(valleyID)){
      tmp <- which(circuit==valleyID[i])
      tmp.comb <- combinations(length(tmp),2)
      el <- rbind(el,cbind(tmp[tmp.comb[,1]],tmp[tmp.comb[,2]]))
  }
  return(list(id=unique(c(el)),pairs=el))
}
my.valley.ids(br.ex)
```

```
## $id
##  [1]  7 11  6 14 15 17 25 27 24 30 33 35  5 53 58 66 56 70 71 75 55 79 81
## [24] 85 54 90 13 52 51 29 44 41 95 68 78 77 89 87 94
##
## $pairs
##       [,1] [,2]
##  [1,]    7   11
##  [2,]    7   13
##  [3,]   11   13
##  [4,]    6   14
##  [5,]    6   52
##  [6,]   14   52
##  [7,]   15   17
##  [8,]   15   51
##  [9,]   17   51
## [10,]   25   27
## [11,]   25   29
## [12,]   27   29
## [13,]   24   30
## [14,]   24   44
## [15,]   30   44
## [16,]   33   35
## [17,]   33   41
## [18,]   35   41
## [19,]    5   53
## [20,]    5   95
## [21,]   53   95
## [22,]   58   66
## [23,]   58   68
## [24,]   66   68
## [25,]   56   70
## [26,]   56   78
## [27,]   70   78
## [28,]   71   75
## [29,]   71   77
## [30,]   75   77
## [31,]   55   79
## [32,]   55   89
## [33,]   79   89
```

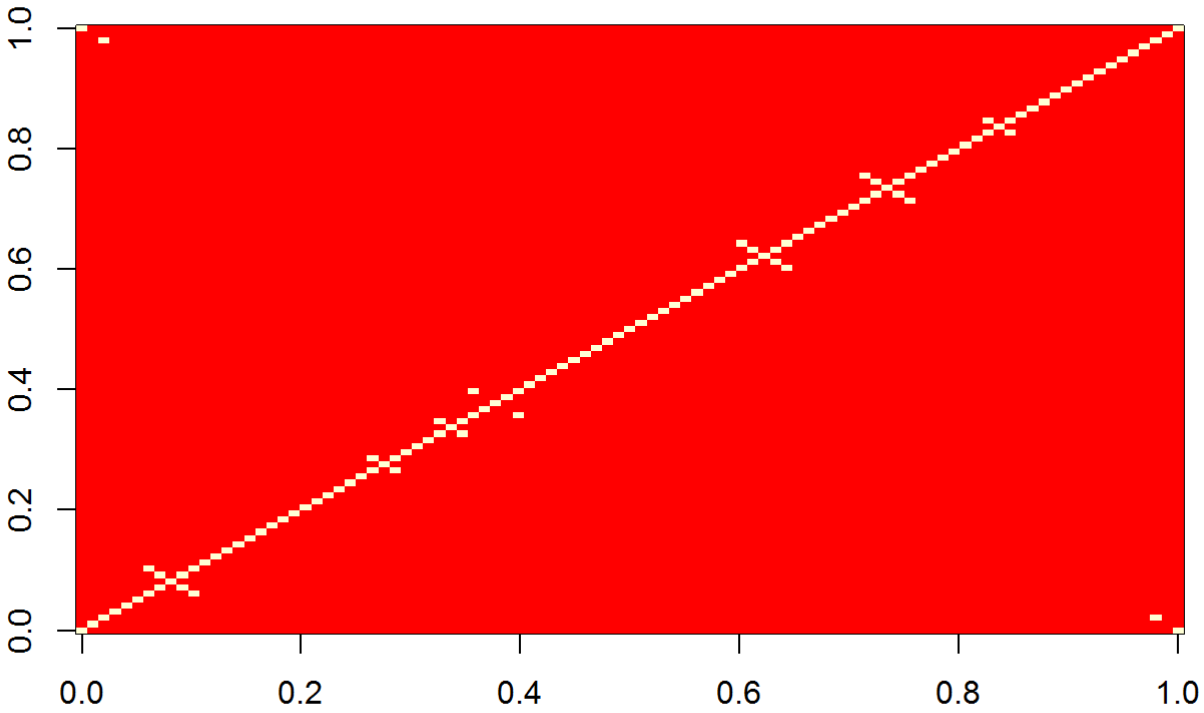
```
## [34,]   81   85
## [35,]   81   87
## [36,]   85   87
## [37,]   54   90
## [38,]   54   94
## [39,]   90   94
```

得られた周回路ノードと、着目２ノードとからなるサブグラフ(完全グラフ)を作り、最短距離を求める。

```
#Z <- my.extreeZ(br.ex)

my.Dc.circuit <- function(i,j,Dcmat,valleys){
  tmp <- Dcmat[c(i,j,valleys$id),c(i,j,valleys$id)]
  n <- length(tmp[,1])
  pairs <- as.matrix(expand.grid(1:n,1:n))
  g <- graph.edgelist(pairs,directed=FALSE)
  ret <- distances(g,1,2,weights=c(tmp))
  return(ret)
}
my.Dc.circuit.mat <- function(br.ex,Z){
  valleys <- my.valley.ids(br.ex)
  circuit <- my.circuit.extree(br.ex)
  n <- length(circuit)
  ret <- matrix(0,n,n)
  Dcmat <- my.D.circuit.mat(circuit,Z)
  Dcmat2 <- Dcmat
  Dcmat2[(valleys$pairs[,1]-1) * n + valleys$pairs[,2]] <- 0

  for(i in 1:(n-1)){
    for(j in (i+1):n){
      Dcmat3 <- Dcmat2
      Dcmat3[i,] <- Dcmat[i,]
      Dcmat3[j,] <- Dcmat[j,]
      Dcmat3[,i] <- Dcmat[,i]
      Dcmat3[,j] <- Dcmat[,j]
      ret[i,j] <- ret[j,i] <- my.Dc.circuit(i,j,Dcmat3,valleys)
    }
  }
  ret
}
D <- my.Dc.circuit.mat(br.ex,Z)
image(D)
```




```
image(D==0)
```



6 検算

Z値が一番小さいノードからの距離は、単純に、Z値の差の絶対値になっている

```
min.Z.id <- which(Z==min(Z))
circuit.min.Z.id <- which(circuit==min.Z.id)
for(i in 1:length(circuit.min.Z.id)){
  print(range(abs(Z[circuit]-Z[min.Z.id]) - D[circuit.min.Z.id[i],]))
  print(range(abs(Z[circuit]-Z[min.Z.id]) - Dcmat[circuit.min.Z.id[i],]))
}
```

```
## [1] -2.220446e-16  4.440892e-16
## [1] -0.9266857  1.1169423
```

周回路上、隣り合うノードの距離は、単純にZ値の差の絶対値

```
for(i in 2:length(circuit)){
  print(D[i-1,i] - abs(Z[circuit[i-1]]-Z[circuit[i]]))
}
```

```
## [1] 9.714451e-17
## [1] 0
## [1] 0
## [1] -2.220446e-16
## [1] 0
## [1] 1.110223e-16
## [1] -1.110223e-16
## [1] 1.110223e-16
## [1] 1.110223e-16
## [1] -1.110223e-16
## [1] 0
## [1] 0
## [1] 1.110223e-16
## [1] 5.551115e-17
## [1] -1.110223e-16
## [1] -1.110223e-16
## [1] -5.551115e-17
## [1] -1.665335e-16
## [1] -5.551115e-17
## [1] 0
## [1] -2.775558e-17
## [1] -1.387779e-16
## [1] 0
## [1] 0
## [1] -1.110223e-16
## [1] -1.110223e-16
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] -1.110223e-16
## [1] 1.110223e-16
## [1] 1.110223e-16
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0
## [1] -1.110223e-16
## [1] 0
## [1] 0
## [1] 0
## [1] -1.387779e-16
## [1] -2.775558e-17
## [1] 0
## [1] -5.551115e-17
## [1] -1.665335e-16
## [1] -5.551115e-17
## [1] 5.551115e-17
## [1] 0
## [1] -5.551115e-17
## [1] 1.110223e-16
## [1] 5.551115e-17
## [1] 5.551115e-17
## [1] 0
## [1] -1.665335e-16
## [1] 5.551115e-17
## [1] 9.714451e-17
## [1] 1.387779e-17
## [1] 1.387779e-17
## [1] 9.714451e-17
## [1] 5.551115e-17
## [1] 5.551115e-17
## [1] 1.110223e-16
## [1] 1.110223e-16
## [1] 0
## [1] 5.551115e-17
## [1] -1.665335e-16
## [1] -1.665335e-16
## [1] 1.110223e-16
## [1] 1.110223e-16
## [1] -1.665335e-16
## [1] 5.551115e-17
## [1] 5.551115e-17
## [1] -1.665335e-16
## [1] 5.551115e-17
```

```
## [1] 0
## [1] -6.938894e-17
## [1] 1.110223e-16
## [1] 5.551115e-17
## [1] 5.551115e-17
## [1] 1.110223e-16
## [1] -6.938894e-17
## [1] -6.938894e-17
## [1] -6.938894e-17
## [1] 0
## [1] 1.110223e-16
## [1] -5.551115e-17
## [1] -8.326673e-17
## [1] -8.326673e-17
## [1] -5.551115e-17
## [1] -5.551115e-17
## [1] 0
## [1] 0
## [1] 0
## [1] 9.714451e-17
```

第一ノードと周回路最終ノードとの距離は0

```
print(D[1,length(circuit)])
```

```
## [1] 0
```

距離0のペアを検証する

```
zero.pairs <- which(D==0,arr.ind=TRUE)
# 自身とのペアを除く
n <- length(circuit)
tmp <- zero.pairs[,1] != zero.pairs[,2]
zero.pairs. <- zero.pairs[tmp,]
zero.pairs. <- t(apply(zero.pairs.,1,sort))
for(i in 1:length(zero.pairs.[,1])){
  vij <- zero.pairs.[i,1]:zero.pairs.[i,2]
  vji <- c(1:n,1:n)[zero.pairs.[i,2]:(length(circuit)+zero.pairs.[i,1])]
  print(zero.pairs.[i,])
  print(circuit[zero.pairs.[i,]])
  print(min(Z[circuit[zero.pairs.[i,]]])-max(min(Z[circuit[vij]]),min(Z[circuit[vji]])))
}
```

```
## [1]  1 99
## [1]  1 51
## [1] 0
## [1]  3 97
## [1] 49 49
## [1] 0
## [1]  7 11
## [1] 5 5
## [1] 0
## [1]  8 10
## [1] 3 3
## [1] 0
## [1]  8 10
## [1] 3 3
## [1] 0
## [1]  7 11
## [1] 5 5
## [1] 0
## [1] 27 29
## [1] 14 14
## [1] 0
## [1] 27 29
## [1] 14 14
## [1] 0
## [1] 33 35
## [1] 19 19
## [1] 0
## [1] 33 35
## [1] 19 19
## [1] 0
## [1] 36 40
## [1] 20 20
## [1] 0
## [1] 36 40
## [1] 20 20
## [1] 0
## [1] 60 64
## [1] 32 32
## [1] 0
```

```
## [1] 61 63
## [1] 30 30
## [1] 0
## [1] 61 63
## [1] 30 30
## [1] 0
## [1] 60 64
## [1] 32 32
## [1] 0
## [1] 71 75
## [1] 38 38
## [1] 0
## [1] 72 74
## [1] 36 36
## [1] 0
## [1] 72 74
## [1] 36 36
## [1] 0
## [1] 71 75
## [1] 38 38
## [1] 0
## [1] 82 84
## [1] 42 42
## [1] 0
## [1] 82 84
## [1] 42 42
## [1] 0
## [1]  3 97
## [1] 49 49
## [1] 0
## [1]  1 99
## [1]  1 51
## [1] 0
```

$d_e(s1, s2) = d_e(s2, s3) = d_e(s3, s1) = 0$なる3ノードが周回グラフには生じるが、すべてで$D(si, sj) = 0$になるわけではないことが確認できる。 それと、「自身より大きなZ値なら迂回路を引ける」ということが対応することも確認できる。