グラフのゼータ函数

リーマンのゼータ函数

$$\zeta_R(s) = \sum_{i=1}^{\infty} \frac{1}{i^s} = \sum_{z \in Z} \frac{1}{z^s} = \prod_{p \in P} \frac{1}{(1 - p^{-s})}$$

すべての自然数に関する函数

$$\sum_{z \in Z} \frac{1}{z^s}$$

が、すべての素数の函数

$$\prod_{p \in P} \frac{1}{(1 - p^{-s})}$$

で表されると言っている。

すべての自然数に関する方は「和」を取っており、

すべての素数に関する方は「積」を取っている。

すべての自然数は素数の積で表されるという性質がある。

その性質を利用して、$\prod_{p \in P}(1 - p^{-s})$が、式展開するとすべての自然数の多項式になることで、２つの表現の一致が説明される。

グラフのゼータ函数

グラフのゼータ函数では、グラフ上のサイクル(あるノードからぐるりと回って元のノードに返ってくるパス)について考える。

グラフの上にある１つのサイクルがあるとき、それは「素数相当のサイクル」。(ただし、後に、サイクルと言ってもある特定の性質を持つサイクルのみに限定することになる。今はその詳細には触れない。)

グラフの上には、複数のサイクルを同時に乗せることができる。 同時に乗せるとは、乗せたときに複数のサイクルが重複しない範囲で乗せることとする。(この重複の定義も後述する。)

このグラフに同時に乗せることができる複数のサイクルの組を、「素数相当のサイクルの積」と見なすことで、「素数ではないサイクル集合」というものが定まる。 これが、グラフにおける「素数の積〜自然数」と考える(ようだ)。

自然数の場合と違って、「素数相当のサイクル」の積は、与えられたグラフに重複なく乗る、という制約があるので、かなり限定されたものとなる。 また、組み合わせられる「素数相当サイクル」の組もあれば、「組み合わせられない組もあることになる。

グラフのゼータ函数のための定義

対象とするグラフ

ここで考えるグラフは

- ノード数が有限
- 連結
- 次数1のノードはない(行き止まりになるノードがない)
- ランクが1以上(ランクとは、グラフに(最大)全域木を取ったとき、最小全域木に入っていない辺の数)

とする。

「素数」なサイクル

- サイクルは、グラフ上のある点から出発して戻ってくるパス
- ただし、ぐるりと回って戻ってくるパスにいくつかの条件を付ける。
- Closed: 出発ノードと終着ノードは同じであるとする
- Backtrackless: あるエッジを辿って、すぐさまそのエッジを逆向きに辿ってはいけない
- Tailless: 戻って来るときに、最後のエッジが最初のエッジの逆向きエッジではいけない(出発点・到着点のところは、ふくらみを持たないといけない)
- Primitive: 上記の条件を満足する周回パスを、複数回、回ってできるパスも上記の条件を満足してしまうが、複数周回のものは除き、単回周回のものを指してprimitiveと言う

これをClosed backtrackless tailess primitive paths (CBTPP) と呼ぶ。

今、あるCBTPPがあったとき、その始点終点をCBTPP上の別点に置き換えたものも、同じCBTPPとみなす(Equivalence class)ことにする。

このように、CBTPPでequivalence classesによって集約したclassesをグラフの「素数」とする。

グラフのゼータ函数は多項式の逆数であり、また、行列式で求められる

ゼータ函数の値の求め方

- 無向グラフGを対称Digraph$D$にする。対称Digraphとは、無向グラフの各エッジを有向エッジ２本としたグラフ。
- Dのエッジのペアについて考える。このエッジペアを行列W_Eで表現することにする。片方のエッジの終点ともう片方のエッジの始点とが同じであって、片方のエッジの始点ともう片方のエッジの終点が異なる場合、行列W_Eの該当セルに1を立て、それ以外のセルは0とする。

$$\zeta_E(u)^{-1} = \det\left(I - uW_E\right); u \in C$$

Rで書いてみる

ゼータ函数値を返す函数

まず、無向グラフを対称Digraphにする函数を書く。

```
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 3.4.4
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
my.bigraph <- function(g){
  el <- as_edgelist(g)
  el2 <- rbind(el,cbind(el[,2],el[,1]))
  ret <- graph.edgelist(el2)
  return(ret)
}
```

次に、有向グラフのW_Eを返す函数を書く。

```
my.WE <- function(g){
  el <- as_edgelist(g)
  n.e <- length(el[,1])
  edge.mat <- matrix(0,n.e,n.e)
    for(i in 1:n.e){
        st <- el[i,1]
        ed <- el[i,2]
        tmp <- which(el[,1]==ed & el[,2]!=st)
        edge.mat[i,tmp] <- 1
    }
  return(edge.mat)
}
```

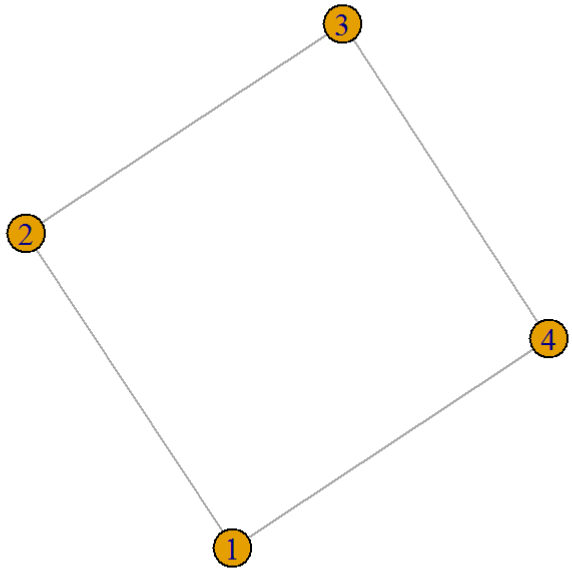
ついで、ゼータ函数の逆数を返す関数を書く。

```
library(complexplus) # 複素行列計算用
```

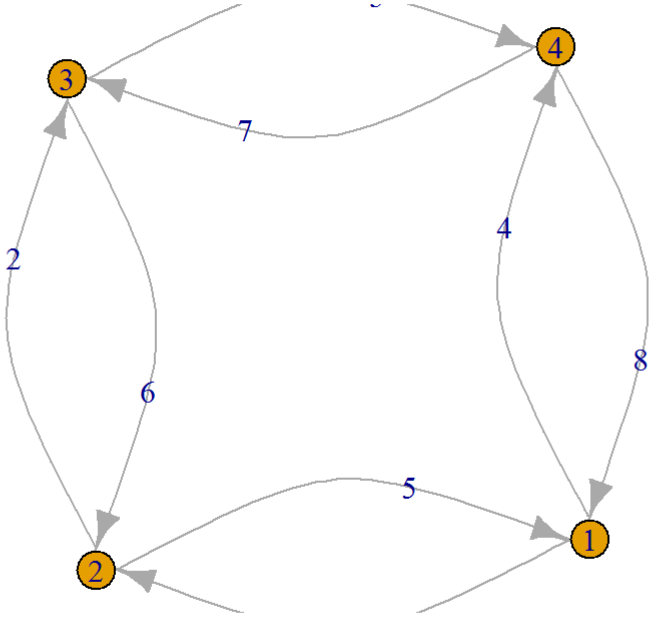
```
## Warning: package 'complexplus' was built under R version 3.4.4
```

```
my.Ihara.zeta.e <- function(g,u){
  g.bi <- my.bigraph(g) # 両方向グラフにする
  we <- my.WE(g.bi) # WE 行列を作る
  return(Det(diag(rep(1,length(we[,1]))) - we * u)) # 行列式を計算して返す
}
```

```
#el <- rbind(c(1,2),c(2,3),c(3,4),c(4,1),c(1,3))
el <- rbind(c(1,2),c(2,3),c(3,4),c(4,1))
g <- graph.edgelist(el,directed =FALSE)
plot(g)
```




```
g.bi <- my.bigraph(g)
plot(g.bi,edge.label=1:8,edge.curved=TRUE)
```



```
WE <- my.WE(g.bi)
WE
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    1    0    0    0    0    0    0
## [2,]    0    0    1    0    0    0    0    0
## [3,]    0    0    0    0    0    0    0    1
## [4,]    0    0    0    0    0    0    1    0
## [5,]    0    0    0    1    0    0    0    0
## [6,]    0    0    0    0    1    0    0    0
## [7,]    0    0    0    0    0    1    0    0
## [8,]    1    0    0    0    0    0    0    0
```

```
u <- rnorm(1) + 1i * rnorm(1)
my.Ihara.zeta.e(g,u)
```

```
## [1] 1.622717-0.357285i
```

ゼータ函数の逆数である多項式を返す関数を書く

ゼータ函数の逆数である多項式の係数が整数係数であることを利用して、適当な複素数を複数指定し、それに対応するゼータ函数値を求め、 その複素数とゼータ函数値のペアが作る連立方程式を解くことで整数係数を求めることができる。

以下の函数はその要領で整数係数を求めるとともに、その整数係数多項式を持ちいて、ゼータ函数値を算出する函数である。

```
my.Ihara.zeta.poly <- function(g){
  n.e <- length(E(g))*2
  us <- rnorm(n.e)
  U <- matrix(0,n.e,n.e)
  for(i in 1:length(us)){
    #U[i,] <- (-us[i])^(1:n.e)
    U[i,] <- (us[i])^(1:n.e)
  }
  vs <- rep(0,n.e)
  for(i in 1:n.e){
    vs[i] <- my.Ihara.zeta.e(g,us[i])
  }
  A <- solve(U) %*% (vs-1)

  return(c(1,A))
}

my.Ihara.zeta.poly.calc <- function(g,u=2){
  A <- round(my.Ihara.zeta.poly(g))
  #v <- sum(A * (-u)^(0:(length(A)-1)))
  v <- sum(A * (u)^(0:(length(A)-1)))
  return(list(v=v,A=A))
}
```

```
my.Ihara.zeta.poly.calc(g,u)
```

```
## $v
## [1] 1.622717-0.357285i
##
## $A
## [1]  1  0  0  0 -2  0  0  0  1
```

この出力は以下の整数係数多項式を意味している(Aの第一成分がu^0$の項)。

$$1 \times u^0 + (-2) \times u^4 + 1 \times u^8$$

ゼータ函数が計算している内容を確認する

$$\begin{aligned} &\zeta_E(u)^{-1} \\ &= \det(I - uW_E) \\ &= \sum_{\sigma \in S} \prod (diag((I - W_E)[, \sigma])) \\ &= \sum_{i=0} a_i u^i \end{aligned}$$

ゼータ函数は、行列W_Eを含んだ行列の行列式で表される。

その内実を展開すれば、対称Digraphのすべてのエッジの順列 $\sigma \in S$について、行列の列の置換を行い、その対角成分の積を足し合わせたものである。

そして、その個々の項は、$0, u^0, \ldots, u^N$(ただしNはDigraphのエッジ本数)のいずれかとなるので$u^0, \ldots, u^N$の整数係数多項式になる、という意味である。

$\prod (diag((I - W_E)[, \sigma]))$ が1になる場合とは、σが置換ではない、オリジナルの並び方の場合である。

一方、$\prod (diag((I - W_E)[, \sigma]))$が$u^k; k > 0$となる場合は、1つ以上のサイクル(Closed backtrackless tailess primitive paths (CBTPP))が存在し、そのサイクルの辺の長さの和がkとなる場合である。

このとき、複数のサイクルは対象Digraphの辺を共有しない。逆に言うと、サイクルのべき集合のうち、相互に重複しないようにグラフ上に配置できるものが$\sigma \in S$にて列挙されることがわかる。

したがって、$\sum_{i=0} a_i u^i$における$\sum a_i$は、そのような条件を満たすべき集合要素の総数に相当する。

今、重複せずに配置できるサイクルの組み合わせを、基本サイクル(素数的サイクル)の積であると見なせば、

基本サイクル$p \in [P]$に関する $\prod_{p \in [P]} (1 - u^{|p|})$として、重複せずに配置できるサイクル組み合わせに関する式となることがわかる。

配置可能サイクル集合を考慮したゼータ函数計算方法をRで実装しておく

順列のうち、非0を返すもののみを選別するために、検討するべき置き換えパターンを行ごとに出力する函数。

```
my.ok.loc <- function(WE){
  tmp <- diag(rep(1,length(WE[,1]))) - WE * 2
  tmp2 <- apply(apply(tmp,2,function(x)x!=0),2,function(x)which(x))
  if(is.matrix(tmp2)){
    tmp3 <- list()
    for(i in 1:length(tmp2[1,])){
      tmp3[[i]] <- tmp2[,i]
    }
    ret <- tmp3
  }
  else{
    ret <- tmp2
  }
  return(ret)
}
```

非0を返す順列のみを取り出す函数。

```
my.ok.perm <- function(WE){
  tmp <- my.ok.loc(WE)
  tmp2 <- expand.grid(tmp)
  tmp3 <- apply(tmp2,1,sort)
  tmp4 <- apply(tmp3,2,diff)
  tmp5 <- apply(tmp4,2,prod)
  okperm <- tmp2[which(tmp5!=0),]
  sgn <- rep(0,length(okperm[,1]))
  nu <- sgn
  for(i in 1:length(sgn)){
    ok <- unlist(okperm[i,])
    sgn[i] <- det((diag(rep(1,length(ok))))[,ok])
    nzero <- length(which(ok-(1:length(ok)) == 0))
    nu[i] <- length(ok)-nzero
  }
  return(list(okperm=okperm,sgn=sgn,nu=nu))
}
```

上記２函数を用いて、非0を返す順列を返しつつ、それを用いたゼータ函数値を計算する函数。

```
my.Ihara.zeta.okperm <- function(g,u=2){
  g.bi <- my.bigraph(g)
  WE <- my.WE(g.bi)
  ok <- my.ok.perm(WE)
  v <- sum(ok$sgn * (-u)^ok$nu)
  return(list(zeta=v,okperm=ok,WE=WE))
}
```

```
my.Ihara.zeta.e(g,u)
```

```
## [1] 1.622717-0.357285i
```

```
my.Ihara.zeta.poly.calc(g,u)
```

```
## $v
## [1] 1.622717-0.357285i
##
## $A
## [1]  1  0  0  0 -2  0  0  0  1
```

```
ok <- my.Ihara.zeta.okperm(g,u)
ok
```

```
## $zeta
## [1] 1.622717-0.357285i
##
## $okperm
## $okperm$okperm
##     Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
## 58     8    1    2    5    6    7    4    3
## 66     8    1    2    4    5    6    7    3
## 191    1    2    3    5    6    7    4    8
## 199    1    2    3    4    5    6    7    8
##
## $okperm$sgn
## [1]  1 -1 -1  1
##
## $okperm$nu
## [1] 8 4 4 0
##
##
## $WE
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    1    0    0    0    0    0    0
## [2,]    0    0    1    0    0    0    0    0
## [3,]    0    0    0    0    0    0    0    1
## [4,]    0    0    0    0    0    0    1    0
## [5,]    0    0    0    1    0    0    0    0
## [6,]    0    0    0    0    1    0    0    0
## [7,]    0    0    0    0    0    1    0    0
## [8,]    1    0    0    0    0    0    0    0
```

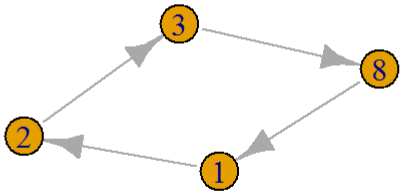
非0を返す順列のそれぞれについて、対角成分のすべては非0になる様子を示しておく。

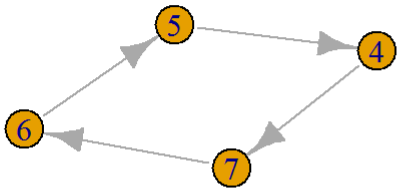
```
for(i in 1:(length(ok$okperm$sgn))){
  print((diag(rep(1,length(ok$WE[1,])))-2*ok$WE)[unlist(ok$okperm$okperm[i,]),])
}
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   -2    0    0    0    0    0    0    1
## [2,]    1   -2    0    0    0    0    0    0
## [3,]    0    1   -2    0    0    0    0    0
## [4,]    0    0    0   -2    1    0    0    0
## [5,]    0    0    0    0   -2    1    0    0
## [6,]    0    0    0    0    0   -2    1    0
## [7,]    0    0    0    1    0    0   -2    0
## [8,]    0    0    1    0    0    0    0   -2
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   -2    0    0    0    0    0    0    1
## [2,]    1   -2    0    0    0    0    0    0
## [3,]    0    1   -2    0    0    0    0    0
## [4,]    0    0    0    1    0    0   -2    0
## [5,]    0    0    0   -2    1    0    0    0
## [6,]    0    0    0    0   -2    1    0    0
## [7,]    0    0    0    0    0   -2    1    0
## [8,]    0    0    1    0    0    0    0   -2
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1   -2    0    0    0    0    0    0
## [2,]    0    1   -2    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0   -2
## [4,]    0    0    0   -2    1    0    0    0
## [5,]    0    0    0    0   -2    1    0    0
## [6,]    0    0    0    0    0   -2    1    0
## [7,]    0    0    0    1    0    0   -2    0
## [8,]   -2    0    0    0    0    0    0    1
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1   -2    0    0    0    0    0    0
## [2,]    0    1   -2    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0   -2
## [4,]    0    0    0    1    0    0   -2    0
## [5,]    0    0    0   -2    1    0    0    0
## [6,]    0    0    0    0   -2    1    0    0
## [7,]    0    0    0    0    0   -2    1    0
## [8,]   -2    0    0    0    0    0    0    1
```

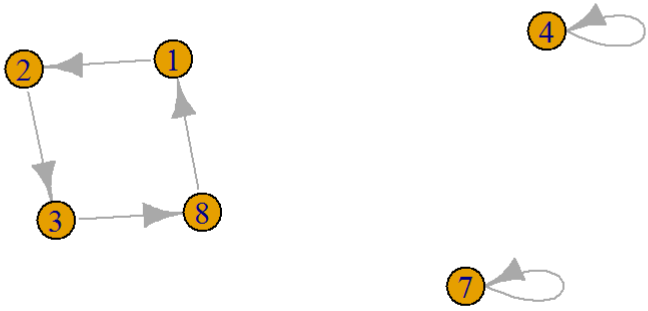
また、それぞれの状態ごとに、どのノードセットがサイクルをなすか(残りのノードは孤立する)、を表示しておく。

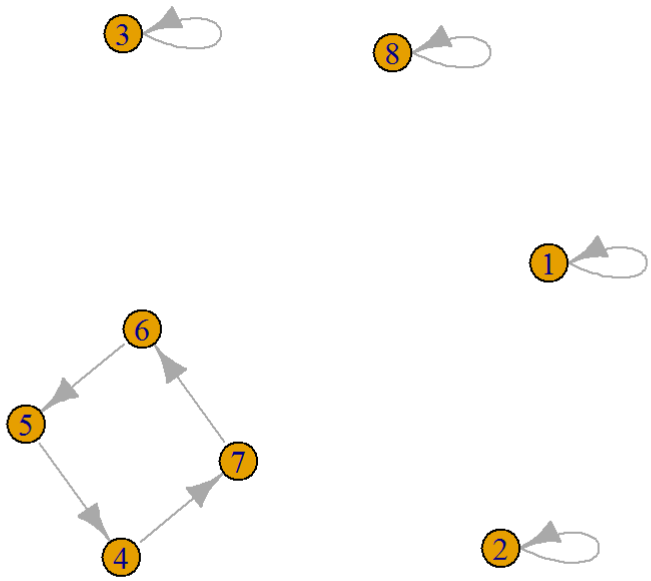
```
for(i in 1:(length(ok$okperm$sgn))){
  oo <- unlist(ok$okperm$okperm[i,])
  el <- cbind(oo, 1:length(oo))
  plot(graph.edgelist(el))
}
```

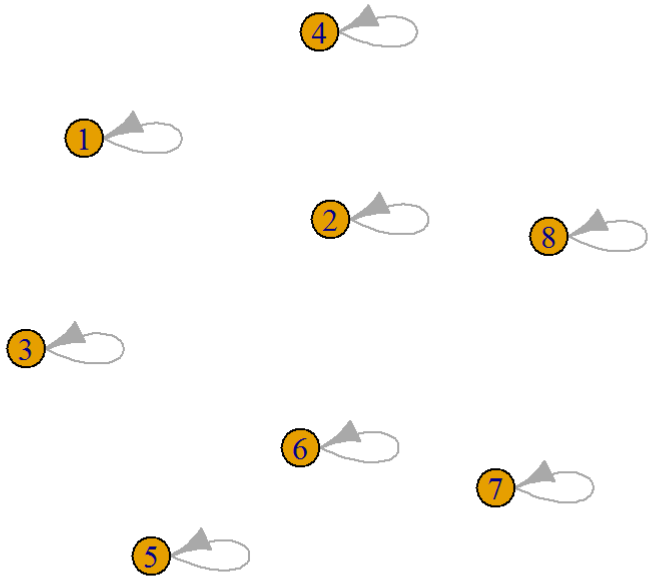

3
8
2
1


5
4
6
7


5
6


4
2
1
3
8
7





グラフのゼータ函数値の別計算法

上述のグラフのゼータ函数値計算法は、グラフの対象性Digrahpのエッジ連結行列を用いたものであったが、

その他に、グラフのノードに関する隣接行列を用いる方法、グラフの中に全域木を定め、全域木に含まれないエッジのみを用いて計算量を抑える方法の２方法が知られている。

以下にそれぞれを実装しておく。

ノードを用いた計算法

Iharaのゼータ函数は、Barholdiのゼータ函数の特殊形なので、２通りの方法で実装しておく

```
library(igraph)
library(complexplus)
my.Ihara.zeta.elem <- function(g){
    A <- as.matrix(as_adjacency_matrix(g))
    n.v <- length(V(g))
    n.e <- length(E(g))
    r <- n.e - n.v
    D <- diag(degree(g))
    H <- diag(degree(g)-1)
    I <- diag(rep(1,n.v))
    return(list(r=r,A=A,H=H,I=I,D=D,n.v=n.v,n.e=n.e))
}
my.Ihara.zeta.ori <- function(g,u){
    elem <- my.Ihara.zeta.elem(g)
    1/((1-u^2)^elem$r * Det(elem$I-u*elem$A + u^2*elem$H))
}

my.Bartholdi.zeta <- function(g,u,t){
    elem <- my.Ihara.zeta.elem(g)
    tmp <- (1-(1-u)^2*t^2)^elem$r * Det(elem$I-t*elem$A+(1-u)*(elem$D-(1-u)*elem$I)*t^2)
    return(1/tmp)
}

my.Ihara.zeta <- function(g,u){
    my.Bartholdi.zeta(g,0,u)
}
```

エッジを用いた計算法

エッジに長さの重みをつけて計算することもできる

サイクルの長さの計算方法に２法あるので、それを別々に実装しておく

```
# (エッジ長ｘ２）^2 行列でのゼータ関数
my.Ihara.zeta.e <- function(g,u){
    el <- as_edgelist(g)
    el2 <- rbind(el,cbind(el[,2],el[,1]))
    n.e <- length(el[,1])
    edge.mat <- matrix(0,n.e*2,n.e*2)
    for(i in 1:(n.e*2)){
        st <- el2[i,1]
        ed <- el2[i,2]
        tmp <- which(el2[,1]==ed & el2[,2]!=st)
        edge.mat[i,tmp] <- 1
    }
    I <- diag(rep(1,n.e*2))
    tmpmat <- I - edge.mat * u
    return(1/Det(tmpmat))
}
# エッジにウェイトがある場合
# Wマトリックスの要素がエッジウェイトによって値を変える
# エッジペアがつながっていたら、２つのエッジの長さの平均を重みにとる
my.Ihara.zeta.e.w <- function(g,u){
    w <- E(g)$weight
    w <- c(w,w)
    if(is.null(w)){
        w <- rep(1,length(E(g))*2)
    }
    el <- as_edgelist(g)
    el2 <- rbind(el,cbind(el[,2],el[,1]))
    n.e <- length(el[,1])
    edge.mat <- matrix(0,n.e*2,n.e*2)
    for(i in 1:(n.e*2)){
        st <- el2[i,1]
        ed <- el2[i,2]
        tmp <- which(el2[,1]==ed & el2[,2]!=st)
        edge.mat[i,tmp] <- u^((w[i]+w[tmp])/2) # u^w[i]にしてもよい
    }
    I <- diag(rep(1,n.e*2))
    tmpmat <- I - edge.mat
    return(1/Det(tmpmat))
}
my.Ihara.zeta.e.w2 <- function(g,u){
    w <- E(g)$weight
    w <- c(w,w)
    if(is.null(w)){
        w <- rep(1,length(E(g))*2)
    }
    el <- as_edgelist(g)
    el2 <- rbind(el,cbind(el[,2],el[,1]))
    n.e <- length(el[,1])
    edge.mat <- matrix(0,n.e*2,n.e*2)
    for(i in 1:(n.e*2)){
        st <- el2[i,1]
        ed <- el2[i,2]
        tmp <- which(el2[,1]==ed & el2[,2]!=st)
        edge.mat[i,tmp] <- u^(w[i]) # u^w[i]にしてもよい->my.Ihara.zeta.e.w2()
    }
    I <- diag(rep(1,n.e*2))
    tmpmat <- I - edge.mat
```

```
    return(1/Det(tmpmat))
}
```

全域木を取り出して軽くする方法。パスを基準にする方法と呼ばれる。

```
my.Ihara.zeta.path <- function(g,u){
    # まずはedge zetaと同じ処理
    # その後で全域木(minimum spaning tree)を取り出して、その補を取る
    w <- E(g)$weight
    if(is.null(w)){
        w <- rep(1,length(E(g)))
    }
    mst <- mst(g,weights=w)

    comst <- g- mst
    n.e.co <- length(E(comst))
    Z <- matrix(0,n.e.co*2,n.e.co*2)
    w.co <- E(comst)$weight
    w.co <- c(w.co,w.co)
    #diag(Z) <- u^w.co
    el <- as_edgelist(comst)
    el2 <- rbind(el,cbind(el[,2],el[,1]))

    for(i in 1:((n.e.co*2))){
        for(j in 1:(n.e.co*2)){
            if(abs(i-j) == n.e.co){
                Z[i,j] <- 0
            #}else if(i==j){

            }else{
                a <- el2[i,2]
                b <- el2[j,1]
                tmp <- w.co[i]/2 + w.co[j]/2 + shortest.paths(mst,a,b)
                #tmp <- shortest.paths(mst,a,b,weights=E(mst)$weight)
                Z[i,j] <- u^tmp
                #a <- el2[j,2]
                #b <- el2[i,1]
                #tmp <- w.co[i] + w.co[j] + shortest.paths(mst,a,b)
                #Z[j,i] <- u^tmp
            }
        }
    }
    I <- diag(rep(1,n.e.co*2))
    tmpmat <- I - Z
    return(1/Det(tmpmat))
}

#my.Ihara.zeta.path(g,xy.[30])
#my.Ihara.zeta.e.w(g,xy.[30])

# エッジに整数長さがあるときに、すべてのエッジ長が１となるように
# ノードを加えたグラフに変換する
# E(g)$weight <- sample(1:20,n.e)

my.graph.inflation.weight <- function(g){
    w <- E(g)$weight
    el <- as_edgelist(g)
    A <- as.matrix(as_adjacency_matrix(g))
    for(i in 1:length(w)){
        if(w[i] > 1){
            nv <- w[i]-1
            st <- el[i,1]
```

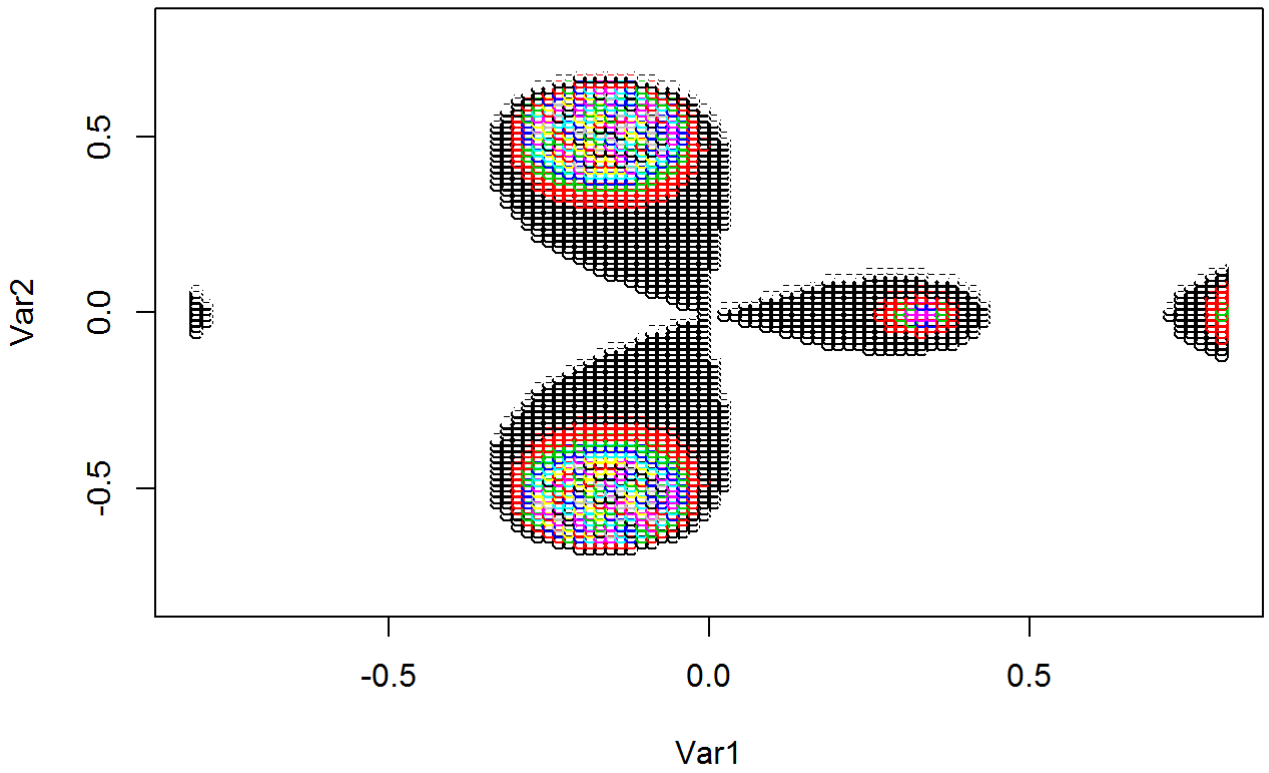
```
            ed <- el[i,2]
            A[st,ed] <- A[ed,st] <- 0
            n <- length(A[,1])
            newA <- matrix(0,n+nv,n+nv)
            newA[1:n,1:n] <- A
            newA[n+1,st] <- newA[st,n+1] <- 1
            newA[n+nv,ed] <- newA[ed,n+nv] <- 1
            if(nv > 1){
                for(j in 1:(nv-1)){
                    newA[n+j,n+j+1] <- newA[n+j+1,n+j] <-1
                }
            }
            A <- newA
        }
    }
    newg <- graph.adjacency(A,mode="undirected")
    return(newg)
}

#E(g)$weight <- rep(3,n.e)
#newg <- my.graph.inflation.weight(g)


fu <- function(u){
    tmp <- (-1)*(3*u-1)*(u+1)^5*(u-1)^6*(3*u^2+u+1)^4
    return(1/tmp)
}

#g <- sample_gnp(10, 2/10)
n <- 5
adj <- matrix(1,n,n)
diag(adj) <- 0
g <- graph.adjacency(adj,mode="undirected")

x <- y <- seq(from=-0.8,to=0.8,length=100)

xy <- expand.grid(x,y)
xy. <- xy[,1] + 1i*xy[,2]

vs <- rep(0,length(xy.))
vs2 <- vs
for(i in 1:length(vs)){
    vs[i] <- my.Ihara.zeta(g,xy.[i])
    vs2[i] <- fu(xy.[i])
}
my.Ihara.zeta(g,vs[55])
```

```
## [1] 1.000005+0.000003i
```

```
fu(vs[55])
```

```
## [1] 1.000005+0.000003i
```

```
plot(xy,col=abs(Mod(vs)))
```




```
fu2 <- function(u){
    tmp <- (1-u^10)^5*(1-3*u^5)*(1-u^5)*(1+u^5+3*u^10)
    return(1/tmp)
}


fu(vs[515]^5)
```

```
## [1] 1+0i
```

```
fu2(vs[515])
```

```
## [1] 1+0i
```