連立多項式、グレブナー基底？

グラフ距離行列とエッジ長の連立方程式

頂点数 n のグラフ距離行列は、対角成分が0の$n \times n$対称正方行列である。

頂点V_iから頂点V_jへのグラフ距離を$d_{i,j}$とする。

$d_{i,j} = d_{j,i}$であるので、$i < j$の場合のみを考えることにする。

$\frac{n(n-1)}{2}$個の$d_{i,j}$を考えればよい。

今、点V_iからV_jへの最短経路$P_{i,j}$が

$$P_{i,j} = (V^{i,j}_{s_0} \rightarrow V^{i,j}_{s_1} \rightarrow \ldots \rightarrow V^{i,j}_{s_{m_{i,j}}})$$
$$s_0 = i, s_{m_{i,j}} = j$$

と$S^{i,j} = (s_0, s_1, \ldots, s_{m_{i,j}})$という数列で表されているとき、次の等式が成り立つ。

$$d_{i,j} = \sum_{t=0}^{m_{i,j}-1} d_{s_t, s_{t+1}}$$

この等式を列挙したものは変数の数がｴｯｼﾞ数の１次連立方程式になるだろう。

大きな連立方程式とグレブナー基底

グレブナー基底ってなんなのか、あまりわかっていないけれど、計算統計学的には、どっさりある多項式から、あるルールで「いい感じに大事なエッセンス」を引き出す仕組みと関係すると理解している。

じゃあ、やってみればいいのでは・・・

ユーティリティ関数

```
# 頂点数 N のグラフのペアワイズ距離をID化する
my.vpairs <- function(N){
  m <- matrix(0,N,N)
  pairs <- which(upper.tri(m),arr.ind=TRUE)
  return(pairs)
}
my.edge.val <- function(e,N){
  e[1] + (e[2]-1)*N
}
my.edge.info <- function(N){
  pairs <- my.vpairs(N)
  vals <- apply(pairs,1,my.edge.val,N)
  return(list(ids=1:length(vals),vals=vals,pairs=pairs,N=N))
}
my.edge.id <- function(e,edge.info){
  v <- my.edge.val(e,edge.info$N)
  return(which(edge.info$vals==v))
}

my.graph.poly <- function(g){# g$weight に長さ情報あり
  n.edge <- length(E(g))

    nv <- length(V(g))
    edge.info <- my.edge.info(nv)
    n.pairs <- length(edge.info$vals)
    paths <- list()
    #for(i in 1:nv){
    #   paths[[i]] <- all_shortest_paths(g,from = i)
    #}

    #dep.array <- array(0,rep(nv,3))
    #paths.mat <- matrix(0,0,nv)
    ne <- length(E(g))

    polyeq <- matrix(0,0,n.pairs)
    el <- get.edgelist(g)

    el.eid <- rep(0,ne)
    for(i in 1:ne){
      el.eid[i] <- my.edge.id(el[i,],edge.info)
    }

    for(i in 1:(nv-1)){
        for (j in (i+1):nv){
          this.edge <- my.edge.id(c(i,j),edge.info)
          #print(this.edge)
            #tmp <- all_shortest_paths(g,from=i,to=j)[[1]]
            tmp. <- shortest_paths(g,from=i,to=j,output="epath")[[2]][[1]]
            tmp.el.eids <- el.eid[as_ids(tmp.)]
            tmp.2 <- rep(0,n.pairs)

            tmp.2[tmp.el.eids] <- 1
            tmp.2[this.edge] <- tmp.2[this.edge] - 1
            if(sum(tmp.2^2)!=0){
              polyeq <- rbind(polyeq,tmp.2)
            }
```

```
            #dep.array[i,j,unlist(tmp)] <- 1
            #if(length(unlist(tmp)>2)){
            #   tmp2 <- rep(0,nv)
            #   tmp2[unlist(tmp)] <- 1
            #   paths.mat <- rbind(paths.mat,tmp2)
            #}
        }
    }
    return(polyeq)
}
#my.vpairs(3)
#my.edge.info(3)
#my.edge.id(c(1,3),my.edge.info(3))
```

```
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
```

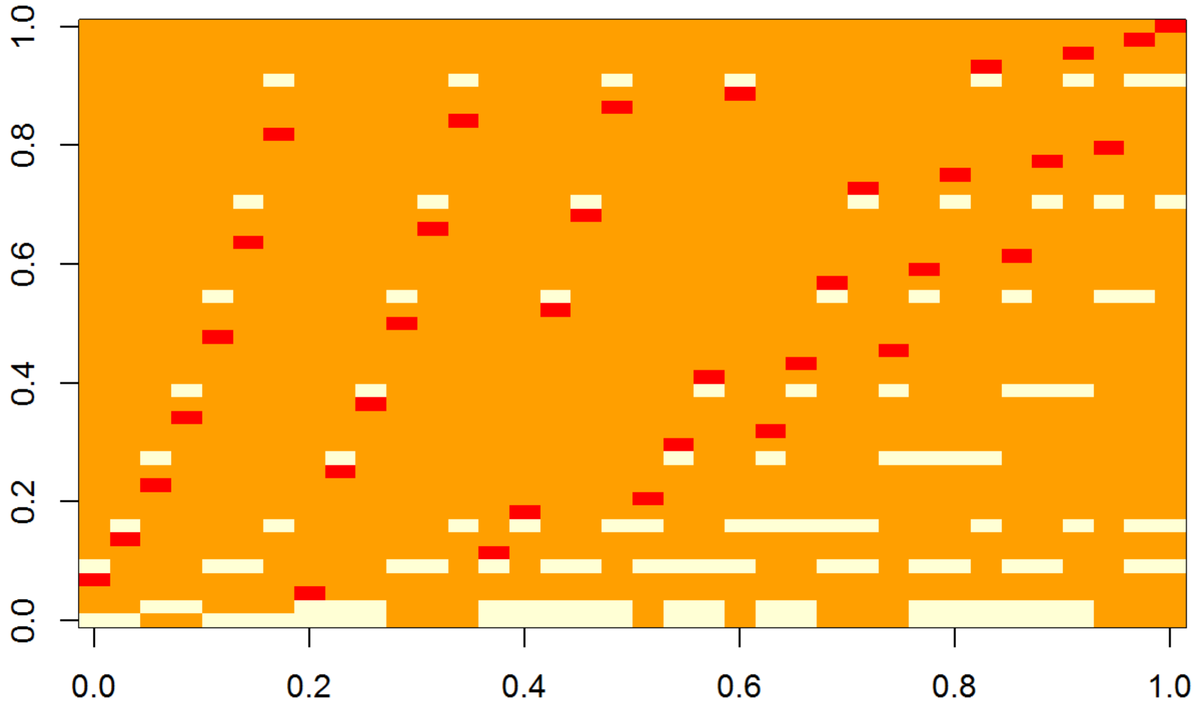
```
g <- make_tree(10, 2, mode = "undirected")


n.edge <- length(E(g))
e.len <- runif(n.edge)

g$weight <- e.len

polyeq <- my.graph.poly(g)
```

```
image(polyeq)
```




```
#library(devtools)
# install_github("ryamada22/Ronlyryamada") 初回はインストールする
library(Ronlyryamada)
library(RFOC)
```

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

```
n <- 3 # メッシュの複雑さを指定(大きいと凹凸の周期が細かくなる)
k <- 2 # メッシュの複雑さを指定(大きいと真球に近くなる)
n.mesh <- 8 # メッシュの細かさを指定
A. <- matrix(runif(n^2), n, n)
A.[1, 1] <- k
A. <- A. + rnorm(n^2, 0, 0.05)
xxx <- my.spherical.harm.mesh(A = A., n = n.mesh)
el <- xxx$edge
rmv <- which(el[,1]==el[,2])
el <- el[-rmv,]
g6 <- graph.edgelist(el,directed=FALSE)
#plot(g)
X <- xxx[[1]]
e.len6 <- rep(0,length(xxx$edge[,1]))
for(i in 1:length(e.len6)){
    e.len6[i] <- sqrt(sum((X[xxx$edge[i,1],]-X[xxx$edge[i,2],])^2))
}
g6$weight <- e.len6
```

```
polyeq2 <- my.graph.poly(g6)
image(polyeq2)
```