3Dオブジェクトの形と配置のデータ保管方法

3Dオブジェクトを球面調和関数分解して把握する

あるオブジェクトO1がある大きさV1であり、ある形S1をとり、それが3D空間中であるおかれ方P1をしているとする。

ただし、ここで言うS1とは、体積を標準化した形であり、また、3D空間での回転によって一致するものは同一の形でみなしたものとする。

また、置かれ方P1とは次のように定める。

01を構成するボクセルの体積を合算したものをV1とする。

01の三角メッシュの座標は重心を原点に取り、その上で、$v1^{1/3}$することで、重心中心の体積１のメッシュとする。

これを球化処理し、その上で球面に張りつけた3D空間のx,y,z座標を球面調和関数分解すると、 $k/times3$の球面調和関数係数行列 Sh1が得られる。

このSh1の各行を3次元ベクトルと見たときに、Sh1はk個の3次元ベクトルの集合である。

今、このk個の3次元ベクトルを回転することで、第1ベクトルをx軸に、第2ベクトルをxy平面に置くことができる。

この回転により、Sh1行列がSh_s1 行列に標準化したと称することにする。

このSh_s1行列は、[1,2],[1,3],[2,3]要素が0の$k \times 3$行列である。

今、ある回転行列P1があって

$$Sh1 = P1Sh_s1$$

となっている。

このP1をオブジェクト01の配置P1とする。

このようにして、オブジェクト01には以下の形・配置属性が得られる。

- Voxel のリスト
- Voxel ベースでの体積 V1
- オリジナルサイズのメッシュ(頂点座標と、三角形の頂点セット)
- 体積標準化後のメッシュ
- 体積標準化後のメッシュから得られる球化メッシュ
- 球面調和関数分解係数ベクトル
- 体積標準化後のメッシュ頂点座標(x_s,y_s,z_s)を分解したもの
- メッシュ頂点の平均曲率を球面調和関数分解したもの(この計算もSPHARMでやりたいですが、どうやるか不明　→　藤井さんと相談)(ここでの平均曲率の球面調和関数分解は、平均曲率からむりやり3D座標を作り出しての分解とは異なります。無理やり作り出しての分解は、平均曲率場によるアラインメントのための作業であって、平均曲率場の球面調和関数分解ではないからです)
- 同、メッシュ頂点のガウス曲率を球面調和関数分解したもの
- 同、平均曲率を無理やり3D座標化して球面調和関数分解しもの

- 同、ガウス曲率を無理やり3D座標化して球面調和関数分解したもの

$k \times 3$ 行列を回転して標準化する

今、3×3行列 Mを

QR分解

$$M = QR$$

すると、Qは直交行列、Rは上三行列となるから

Sh1の第1,2,3行のみを取り出した行列を転置したものをMとすればよいことがわかる。

$$Sh_s1^T = Q^{-1}Sh1^T$$

であるから、置かれ方P1は

$$Sh1^T = P1Sh_s1^T$$

なので、

$$P1 = Q$$

である。

やってみる。

```
k <- 10
d <- 3
Sh1 <- matrix(rnorm(k*d),ncol=d)
M <- t(Sh1[1:3,])
qrout <- qr(M)
Q <- qr.Q(qrout)
print("Identity Matrix")
```

```
## [1] "Identity Matrix"
```

```
Q %*% t(Q) # 単位行列
```

```
##               [,1]          [,2]          [,3]
## [1,]  1.000000e+00 -2.116363e-16 -5.551115e-17
## [2,] -2.116363e-16  1.000000e+00 -9.020562e-17
## [3,] -5.551115e-17 -9.020562e-17  1.000000e+00
```

```
R <- t(Q) %*% M
print("triangular")
```

```
## [1] "triangular"
```

```
R # 上三角
```

```
##                [,1]          [,2]       [,3]
## [1,] -9.285515e-01  1.191054e+00 -0.5201902
## [2,]  0.000000e+00 -1.491232e+00  0.7058025
## [3,] -1.110223e-16  1.457168e-16 -0.1199628
```

```
Sh_s1 <- t(t(Q) %*% t(Sh1)) # t(Q) = Q^{-1}
print("Sh_s1")
```

```
## [1] "Sh_s1"
```

```
Sh_s1
```

```
##             [,1]        [,2]          [,3]
##  [1,] -0.9285515  0.00000000 -1.110223e-16
##  [2,]  1.1910539 -1.49123249  1.457168e-16
##  [3,] -0.5201902  0.70580250 -1.199628e-01
##  [4,]  1.0378142  0.56079489 -5.995400e-01
##  [5,]  1.8809152  0.99484300 -7.817416e-02
##  [6,] -0.2881588 -1.17950137  1.026867e+00
##  [7,]  0.9028599  0.43144856  1.350549e+00
##  [8,] -0.1779854 -0.06863067 -5.774168e-01
##  [9,] -0.9809691 -0.95533771 -5.503899e-01
## [10,]  0.2882034 -1.25259415 -6.665569e-01
```

```
round(Sh_s1,5)
```

```
##           [,1]     [,2]     [,3]
##  [1,] -0.92855  0.00000  0.00000
##  [2,]  1.19105 -1.49123  0.00000
##  [3,] -0.52019  0.70580 -0.11996
##  [4,]  1.03781  0.56079 -0.59954
##  [5,]  1.88092  0.99484 -0.07817
##  [6,] -0.28816 -1.17950  1.02687
##  [7,]  0.90286  0.43145  1.35055
##  [8,] -0.17799 -0.06863 -0.57742
##  [9,] -0.98097 -0.95534 -0.55039
## [10,]  0.28820 -1.25259 -0.66656
```

上記処理の確認が取れたので、関数化しておく。

```
my.standardSh <- function(Sh){
  M <- t(Sh[1:3,])
  qrout <- qr(M)
  Q <- qr.Q(qrout)

  Sh_s <- t(t(Q) %*% t(Sh))
  return(list(Sh_s=Sh_s,P=Q))
}
```

アラインメント～２つのオブジェクトの形を比べるための相対的配置～

今、２つのオブジェクトがあるとする。

(O1,V1,S1,P1), (02,V2,S2,P2)

である。

特に、S1,S2は、球面調和関数係数行列の[1,2][1,3],[2,3]成分が0であることを強調して

(O1,V1,Sh_s1,P1), (02,V2,Sh_s2,P2) と書き直しておく。

アラインメントは、01と02の標準化した形情報であるSh_s1とSh_s2とを比較することとする。

今、01を基準にして、

O2を回転R(2|1)することで、O2がO1に似るようにすることを考える。

この$R(2|1)$はAlthloothiの方法によって推定される。

もし、O1,O2が3D空間にP1,P2で配置されているときに、02を回転して、O1と『アラインメント』するためには

02を$P2^{-1}$回転して、02の標準的なおき方にし、

さらに$R(2|1)$回転して、O1の標準的なおき方とうまくアラインメントさせ、されに$P1$して、O1の置かれ方にあわせる必要がある。

一連の回転は

$$P1R(2|1)P2^{-1}$$

である。

念のため、ここにAlthloothiの方法の実装を藤井さんコード、それを簡略化した山田コードで示しておく。

```
######
## 藤井さん版
######

# Althoothi によるrobust estimation
# SPHARM-PDM による球面調和関数分解では, 係数について8パターンのflip を生じる
# そのうちのひとつが, 最適な距離と回転を推定するので、すべてのflip を試して
# index にはいっている値が, 最小の距離を与えるflip である
# return には1番目にその最小推定が来るように並び替えているので
# $result[[1]] がその最小の結果である

### R
QSigma <- function(clm1, clm2, l=NULL, scale=TRUE, l0=FALSE){
  # l: degree
  # scale: compute mean ¥mu
  # l0: degree l=0 is the varicenter. include or not
  # flip: try all 8 flip pattern for alignment
  Lmax <- max(nrow(clm1), nrow(clm2))
  if(nrow(clm1) < nrow(clm2)){
    clm1 <- rbind(clm1, replicate(ncol(clm1), rep(0, Lmax-nrow(clm1))))
  } else if (nrow(clm1) > nrow(clm2)){
    clm2 <- rbind(clm2, replicate(ncol(clm2), rep(0, Lmax-nrow(clm2))))
  }
  if(is.null(l)){
    Lmax <- max(nrow(clm1), nrow(clm2))
  } else {
    Lmax <- (l+1)^2
  }
  clm1 <- as.matrix(clm1[(1+!l0):Lmax,,drop=FALSE])
  clm2 <- as.matrix(clm2[(1+!l0):Lmax,,drop=FALSE])
  if(scale){
    clm1_scale <- scale(clm1, scale=FALSE)
    clm2_scale <- scale(clm2, scale=FALSE)
    dist1 <- sum((clm1_scale - clm2_scale)^2) # 平方根とらないL2 norm
    L2 <- sum(clm1_scale^2) + sum(clm2_scale^2)
    Sigma <- t(clm1_scale) %*% clm2_scale
  } else {
    dist1 <- sum((clm1 - clm2)^2) # 平方根とらないL2 norm
    L2 <- sum(clm1^2) + sum(clm2^2)
    Sigma <- t(clm1) %*% clm2
  }
  Aij <- Sigma - t(Sigma)
  Delta <- c(Aij[2, 3], Aij[3, 1], Aij[1, 2])
  res <- diag(0, 4)
  res[1, 1] <- sum(diag(Sigma))
  res[1, 2:4] <- Delta
  res[2:4, 1] <- Delta
  res[-1, -1] <- Sigma + t(Sigma) - res[1,1]*diag(1, 3)
  eig <- eigen(res)
  theta <- acos(eig$vector[1,1])*2
  u <- eig$vector[-1,1]/sin(theta/2)
  if(any(is.nan(u))) u <- c(1, 0, 0)
  optdist <- L2 - 2*eig$value[1]
  return(list(mat=res, v=eig$value, e=eig$vector, theta=theta, u=u, distance=dist1, optdist=optdist))
}

flipQSigma <- function(clm1, clm2, scale=TRUE, l0=FALSE){
```

```
# degree
  n <- 1:nrow(clm2)
  flip_ll <- function(n) floor(sqrt(n-1))
  flip_mm <- function(n) floor(n - flip_ll(n)^2 - 1)
  flip_m2 <- function(n) floor((n - flip_ll(n)^2)/2)
  flip_fu0 <- (-1)^ifelse(flip_mm(n)==0, flip_ll(n), ifelse(flip_mm(n)%%2, flip_ll(n)+flip_m2(n), flip
_ll(n)+flip_m2(n)+1))
  flip_fu1 <- (-1)^ifelse(flip_mm(n)==0, flip_ll(n), ifelse(flip_mm(n)%%2, flip_ll(n), flip_ll(n)+1))
  flip_fu2 <- (-1)^flip_m2(n)
  flip_reflect <- ifelse(flip_mm(n)==0, 1, ifelse(flip_mm(n)%%2, 1, -1))
  fu012 <- rbind(abs(flip_fu0), flip_fu0, flip_fu1, flip_fu2)
# possible 8 patterns
  pat <- rbind(fu012, sweep(fu012, 2, flip_reflect, "*"))
  ret <- mapply(function(z) QSigma(clm1, clm2*pat[z,]), 1:nrow(pat), SIMPLIFY=FALSE)
  idx <- which.min(sapply(ret, "[[", "optdist"))
  retclm <- list(clm1, clm2*pat[idx,])
  return(list(clm=retclm, index=idx, result=ret[replace(1:8, c(1, idx), c(idx, 1))]))
}


# 検算用
flippattern <- function(l){
# degree
  n <- 1:l
  flip_ll <- function(n) floor(sqrt(n-1))
  flip_mm <- function(n) floor(n - flip_ll(n)^2 - 1)
  flip_m2 <- function(n) floor((n - flip_ll(n)^2)/2)
  flip_fu0 <- (-1)^ifelse(flip_mm(n)==0, flip_ll(n), ifelse(flip_mm(n)%%2, flip_ll(n)+flip_m2(n), flip
_ll(n)+flip_m2(n)+1))
  flip_fu1 <- (-1)^ifelse(flip_mm(n)==0, flip_ll(n), ifelse(flip_mm(n)%%2, flip_ll(n), flip_ll(n)+1))
  flip_fu2 <- (-1)^flip_m2(n)
  flip_reflect <- ifelse(flip_mm(n)==0, 1, ifelse(flip_mm(n)%%2, 1, -1))
  fu012 <- rbind(abs(flip_fu0), flip_fu0, flip_fu1, flip_fu2)
# possible 8 patterns
  pat <- rbind(fu012, sweep(fu012, 2, flip_reflect, "*"))
  return(pat)
}
```

```
####
# 山田版
####
library(onion)
# X1を回してX2に近づける
my.althlooti <- function(X1,X2){
    M <- t(X2) %*% X1
    N <- matrix(0,4,4)

    N[1,1] <- M[1,1] + M[2,2] + M[3,3]
    N[1,2] <- M[2,3] - M[3,2]
    N[1,3] <- M[3,1] - M[1,3]
    N[1,4] <- M[1,2] - M[2,1]
    N[2,1] <- N[1,2]
    N[2,2] <- M[1,1] - M[2,2] - M[3,3]
    N[2,3] <- M[1,2] + M[2,1]
    N[2,4] <- M[3,1] + M[1,3]
    N[3,1] <- N[1,3]
    N[3,2] <- N[2,3]
    N[3,3] <- -M[1,1] + M[2,2] - M[3,3]
    N[3,4] <- M[2,3] + M[3,2]
    N[4,1] <- N[1,4]
    N[4,2] <- N[2,4]
    N[4,3] <- N[3,4]
    N[4,4] <- -M[1,1] - M[2,2] + M[3,3]

    eigen.out <- eigen(N)
    q <- Re(eigen.out[[2]][,1])
    qh <- q[1] + Hi * q[2] + Hj * q[3] + Hk * q[4]
    Xh <- Hi * X1[,1] + Hj * X1[,2] + Hk * X1[,3]

    RotX1 <- Conj(qh) * Xh * qh
    RotX1.mat <- cbind(i(RotX1),j(RotX1),k(RotX1))

    D0 <- sqrt(sum((X1-X2)^2))
    Dal <- sqrt(sum((RotX1.mat-X2)^2))
    return(list(X1=X1,X2=X2,M=M, N=N, eigen.out=eigen.out,q =qh,RotX1 = RotX1.mat,D0=D0,Dal=Dal))
}

# 四元数から対応する３ｘ３回転行列に変換
my.q2rotmat <- function(q){
  x <- Re(q)
  y <- i(q)
  z <- j(q)
  w <- k(q)
  R <- matrix(c(x^2+y^2-z^2-w^2,2*(y*z-x*w),2*(x*z+y*w),
                2*(x*w+y*z),x^2-y^2+z^2-w^2,2*(-x*y+z*w),
                2*(y*w-x*z),2*(z*w+x*y),x^2-y^2-z^2+w^2),
              byrow=TRUE,3,3)
  return(t(R))

}

q <- runif(1) + Hi*runif(1)+Hj*runif(1)+Hk*runif(1)
p <- Hi*runif(1)+Hj*runif(1)+Hk*runif(1)

Conj(q) * p * q
```

```
##             [1]
## Re 6.245005e-17
## i  6.056123e-01
## j  5.817151e-01
## k  3.098281e-01
```

```
R <- my.q2rotmat(q)

R %*% c(i(p),j(p),k(p))
```

```
##           [,1]
## [1,] 0.6056123
## [2,] 0.5817151
## [3,] 0.3098281
```

練習として、以下のように、Sh1,Sh2が得られているときに、それらをSh_s1,Sh_s2に変換した上で、AlthloothiしてR(2|1)を求め、$P1R(2|1)P2^{-1}$の変換も確認しておく。

```
k <- 7
d <-3
Sh1 <- matrix(rnorm(k*d),ncol=d)
library(GPArotation)
R <- Random.Start(d) # ランダムな回転行列
# Sh2 はSh1を回転したものに似ているようにしておく
Sh2 <- t(R %*% t(Sh1)) + rnorm(k*d)*0.0

# Sh_s1, Sh_s2

tmp <- my.standardSh(Sh1)
Sh_s1 <- tmp$Sh_s
P1 <- tmp$P
tmp2 <- my.standardSh(Sh2)
Sh_s2 <- tmp2$Sh_s
P2 <- tmp2$P

tmp3 <- my.althlooti(Sh_s2,Sh_s1) # 引数の順番に注意
R2_1.q <- tmp3$q
R2_1 <- my.q2rotmat(R2_1.q)
```

検算する。

```
tmp3$RotX1  - Sh_s1
```

```
##               [,1]          [,2]          [,3]
## [1,] -2.220446e-16 -1.864032e-16  1.703239e-16
## [2,]  2.775558e-16 -1.110223e-16 -1.876291e-16
## [3,] -4.440892e-16  0.000000e+00  1.387779e-16
## [4,]  0.000000e+00 -2.775558e-16  2.220446e-16
## [5,] -2.220446e-16 -4.440892e-16  2.220446e-16
## [6,]  4.440892e-16  4.440892e-16 -2.636780e-16
## [7,]  0.000000e+00  5.551115e-17  0.000000e+00
```

確かに、Sh_s2を回転してSh_s1に近くなっている。

```
Sh_s2.rot <- t(R2_1 %*% t(Sh_s2))

Sh_s2.rot - Sh_s1 # 行列に近いはず
```

```
##               [,1]          [,2]          [,3]
## [1,] -2.220446e-16 -1.864032e-16  1.703239e-16
## [2,]  2.220446e-16  0.000000e+00 -1.876291e-16
## [3,] -4.440892e-16  0.000000e+00  1.387779e-16
## [4,]  0.000000e+00 -3.330669e-16  2.220446e-16
## [5,]  0.000000e+00 -4.440892e-16  2.220446e-16
## [6,]  3.330669e-16  2.220446e-16 -2.567391e-16
## [7,] -2.220446e-16  5.551115e-17  0.000000e+00
```

次に、$P1R(2|1)P2^{-1}$にて Sh2を回して、Sh1に近づける。

```
P1RP2 <- P1 %*% R2_1 %*% t(P2)
Sh2.rot <- t(P1RP2 %*% t(Sh2))
Sh2.rot - Sh1
```

```
##               [,1]          [,2]          [,3]
## [1,] -3.330669e-16  3.330669e-16  0.000000e+00
## [2,]  0.000000e+00  1.110223e-16 -1.110223e-16
## [3,]  4.440892e-16 -5.759282e-16  2.220446e-16
## [4,]  4.440892e-16 -4.440892e-16  0.000000e+00
## [5,]  6.661338e-16 -9.992007e-16  4.440892e-16
## [6,] -2.220446e-16  2.775558e-16  0.000000e+00
## [7,]  5.065393e-16 -4.440892e-16  0.000000e+00
```

3個以上のオブジェクトのアラインメント

一般にn個のオブジェクトのアラインメントは、個々の要素を回転行列とした、$n \times n$行列で表すこととする。

このアラインメン行列の行列を、Aと呼ぶことにする。

Aの対角行列は、3×3の単位行列である。 Aの[i,j]要素はR(j|i)とする。R(j|i)とは、Ojを回転してOiに近づけるような行列である。

なお、一般に

$$R(k|i) \neq R(k|j)R(j|i)$$

であることに注意する。