GLS PCA

Jan de Leeuw - University of California Los Angeles
Jan Graffelman - Universitat Politècnica de Catalunya

26 September, 2025

Introduction

This document seeks low-rank approximations for a given matrix based on GLS by iteratively minimizing a loss function.

Code

```r
set.seed(12345)
x <- matrix(rnorm(40), 10, 4)
x <- apply(x, 2, function(x) x - mean(x))
u <- crossprod(matrix(rnorm(100), 10, 10)) / 10
v <- crossprod(matrix(rnorm(16), 4, 4)) / 4
```

```r
#install.packages("RSpectra")
library("RSpectra")

ulsPCA <- function(x, p) {
  x <- apply(x, 2, function(z) z - mean(z))
  s <- svd(x, nu = p, nv = p)
  a <- s$u
  b <- s$v %*% diag(s$d[1:p])
  return(list(a = a, b = b, ab = tcrossprod(a, b)))
}


ulsAdd <- function(x) {
  x <- apply(x, 2, function(z) z - mean(z))
  r <- apply(x, 1, mean)
  s <- apply(x, 2, mean)
  m <- mean(x)
  return(list(
  m = m,
  r = r - m,
  s = s - m,
  rs = outer(r, s, "+") + m))
}

ulsBoth <- function(x, p) {
  h1 <- ulsAdd(x)
  h2 <- ulsPCA(x - h1$rs, p)
  return(list(
    m = h1$m,
    r = h1$r,
    s = h1$s,
    a = h2$a,
    b = h2$b,
    y = h1$rs + h2$ab
  ))
}

glsLoss <- function(x, y, u, v) {
  d <- x - y
  return(sum(v * crossprod(d, (u %*% d))))
}

glsAdd <- function(x,
                   u,
                   v,
                   type = 3,
                   p = 2,
                   itmax = 10000,
                   eps = 1e-6,
                   verbose = FALSE) {
  yold <- switch(type,
                 ulsAdd(x)$rs,
                 ulsPCA(x, p)$ab,
                 ulsBoth(x, p)$y
                 )
  sold <- glsLoss(x, yold, u, v)
  lbdm <- eigs_sym(u, 1)$values * eigs_sym(v, 1)$values
  itel <- 1
  repeat {
    d <- x - yold
    g <- yold + u %*% d %*% v / lbdm
    ynew <- switch(type,
                   ulsAdd(g)$rs,
                   ulsPCA(g, p)$ab,
                   ulsBoth(g, p)$y
                   )
    snew <- glsLoss(x, ynew, u, v)
    if (verbose) {
      cat(
        "itel ",
        formatC(itel, format = "d"),
        "sold ",
        formatC(sold, digits = 10, format = "f"),
        "snew ",
        formatC(snew, digits = 10, format = "f"),
        "\n"
        )
    }
    if ((itel == itmax) || ((sold - snew) < eps)) {
      break
```

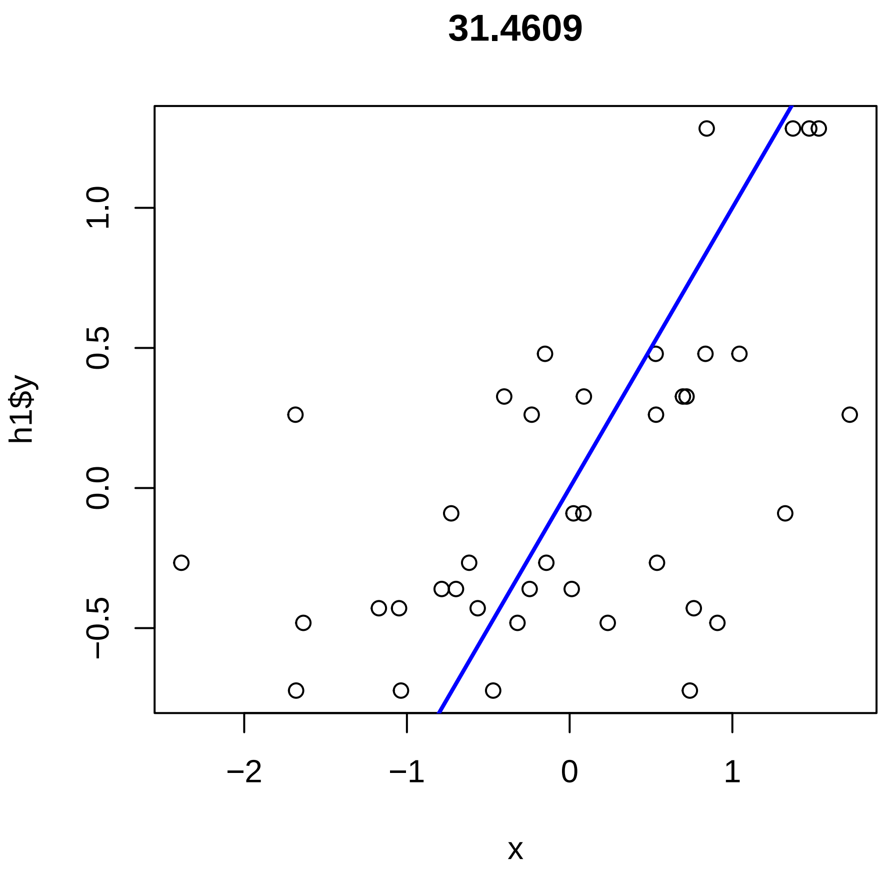
```r
    }
    itel <- itel + 1
    sold <- snew
    yold <- ynew
  }
  return(list(y = ynew, loss = snew, itel = itel))
}
```

```r
eig <- function (X) {
  out <- eigen(X)
  V <- out$vectors
  D <- diag(out$values)
  return(list(V = V, D = D))
}

matrank <- function(X, symmetric = FALSE, crit = 1e-10, verbose = FALSE)
{
    n <- nrow(X)
    p <- ncol(X)
    if (!symmetric) {
        if (n < p)
            A <- X %*% t(X)
        else A <- t(X) %*% X
    }
    else A <- X
    lam <- eigen(A)$values
    r <- sum(lam > crit)
    lmin <- min(lam)
    if (verbose) {
        cat("Matrix rank = ", r, "\n")
        cat("Smallest eigenvalue = ", lmin, "\n")
    }
    return(r)
}
```

```r
h1 <- glsAdd(x, u, v, type = 1, verbose = FALSE)
h2 <- glsAdd(x, u, v, type = 2, verbose = FALSE)
h3 <- glsAdd(x, u, v, type = 3, verbose = FALSE)

h1loss <- round(h1$loss,4)
h2loss <- round(h2$loss,4)
h3loss <- round(h3$loss,4)

plot(x,h1$y,main=toString(h1loss))
matrank(h1$y)
```

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

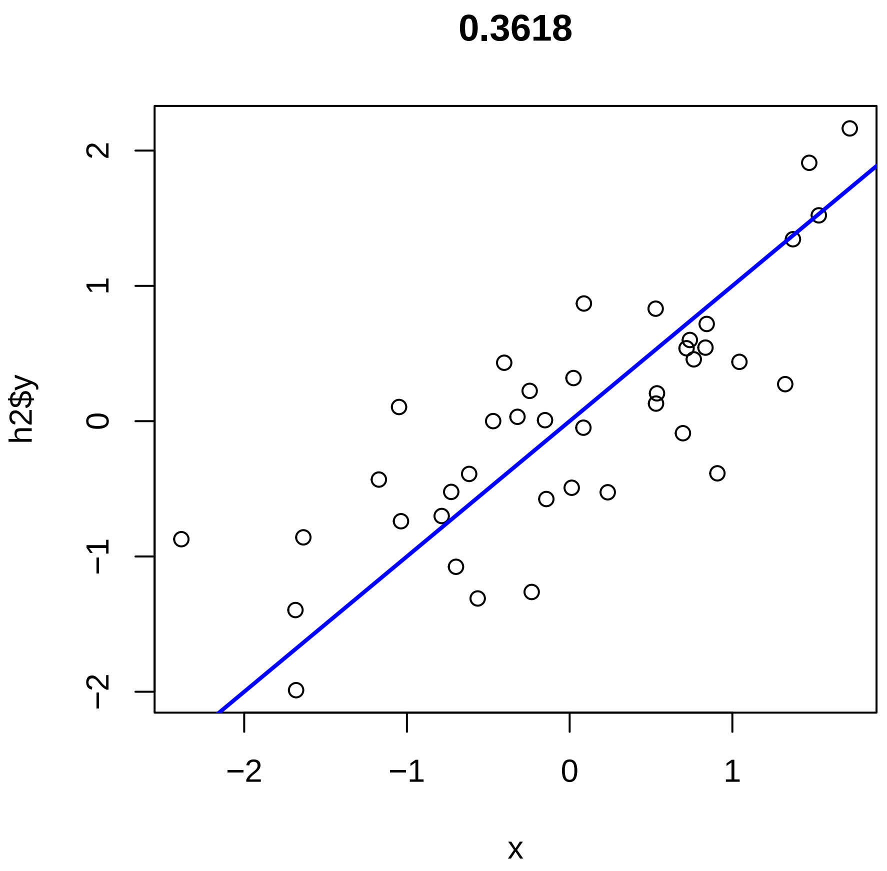
```r
abline(0,1,lwd=2,col="blue")
```



```
plot(x,h2$y,main=toString(h2loss))
matrank(h2$y)
```

```
## [1] 2
```

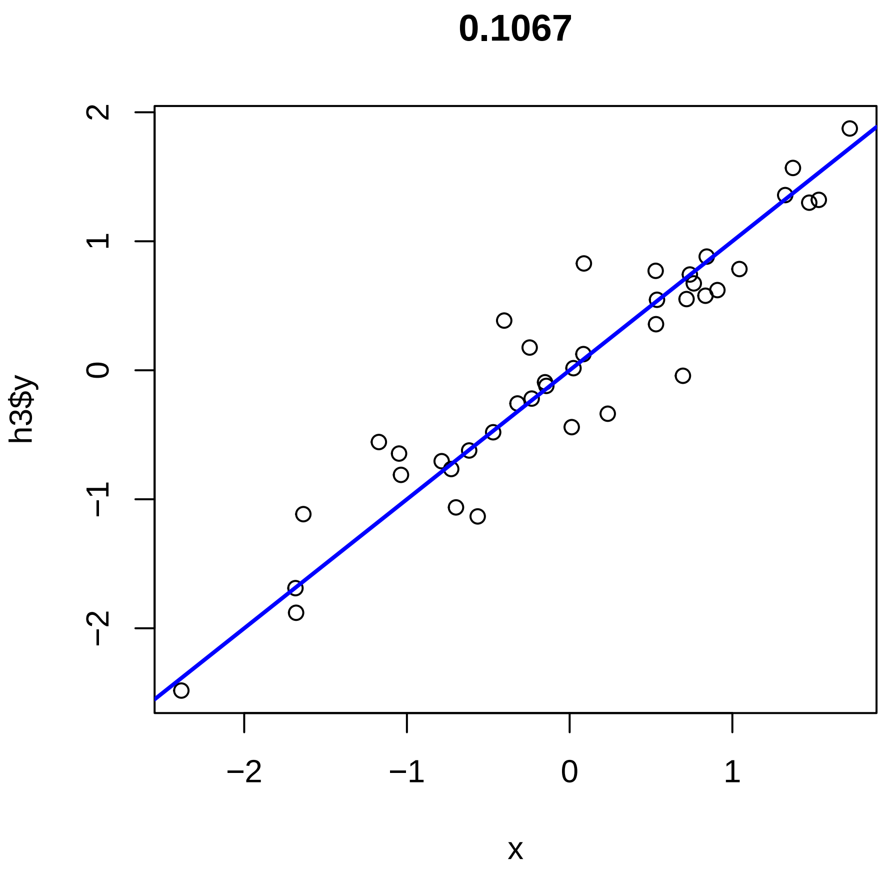
```
abline(0,1,lwd=2,col="blue")
```



```
plot(x,h3$y,main=toString(h3loss))
matrank(h3$y)
```

```
## [1] 3
```

```
abline(0,1,lwd=2,col="blue")
```



Classical solution with the SVD

The classical way to approximate a matrix by GLS with square invertible non-diagonal weight matrices for rows and columns is obtained by the SVD in combination with transformation and backtransformation:

$$\mathbf{X}_t = \mathbf{R}^{\frac{1}{2}}\mathbf{X}\mathbf{C}^{\frac{1}{2}} = \mathbf{U}\mathbf{D}\mathbf{V}',$$

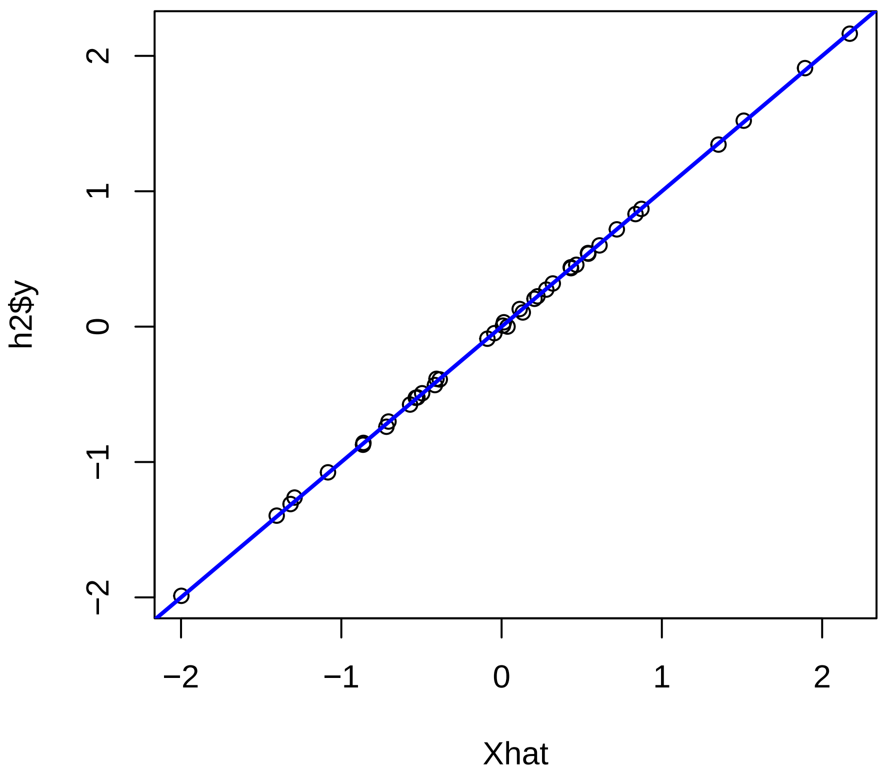
$$\tilde{\mathbf{U}} = \mathbf{R}^{-\frac{1}{2}}\mathbf{U} \quad \text{and} \quad \tilde{\mathbf{V}} = \mathbf{C}^{-\frac{1}{2}}\mathbf{V}$$

$$\hat{\mathbf{X}} = \tilde{\mathbf{U}}_{[,1:k]}\mathbf{D}_{[1:k,1:k]}\tilde{\mathbf{V}}_{[,1:k]}$$

This is coded below, and the solution is shown to be identical to the `glsAdd(x, u, v, type = 2)` solution.

```
u.specdec <- eig(u)
u.U <- u.specdec$V
u.D <- u.specdec$D
u.phalf <- u.U%*%sqrt(u.D)%*%t(u.U)
u.mhalf <- u.U%*%sqrt(diag(1/diag(u.D)))%*%t(u.U)

v.specdec <- eig(v)
v.U <- v.specdec$V
v.D <- v.specdec$D
v.phalf <- v.U%*%sqrt(v.D)%*%t(v.U)
v.mhalf <- v.U%*%sqrt(diag(1/diag(v.D)))%*%t(v.U)
```

```
Xt <- u.phalf%*%x%*%v.phalf

r <- 2
out.svd <- svd(Xt)
U  <- out.svd$u[,1:r]
Ds <- diag(out.svd$d[1:r])
V  <- out.svd$v[,1:r]

Xhat <- u.mhalf%*%U%*%Ds%*%t(V)%*%v.mhalf

plot(Xhat,h2$y)
abline(0,1,lwd=2,col="blue")
```



```
gof  <- sum(diag(Ds)^2)/sum(out.svd$d^2)
gof
```

```
## [1] 0.9931523
```

```
loss <- sum(out.svd$d[3:4]^2)
loss
```

```
## [1] 0.3616262
```

Mistake in `glsAdd(x, u, v, type = 1)` ?

In general, model 1 $\mu + \alpha_i + \beta_j$ is expected to give a rank 2 approximation. But due to the column-centring `x <- apply(x, 2, function(z) z - mean(z))` it will *always* give a rank 1 solution, for `s` and `m` will be structurally zero. It seems the column centring should not be applied. This indeed gives smaller loss and a rank 2 solution as shown by the code below with `ulsAdd` redefined. Note that the modification *increases* the loss for `type=3` which calls `ulsAdd`.

```
ulsAdd <- function(x) {
#  x <- apply(x, 2, function(z) z - mean(z))
  r <- apply(x, 1, mean)
  s <- apply(x, 2, mean)
  m <- mean(x)
  return(list(
  m = m,
  r = r - m,
  s = s - m,
  rs = outer(r, s, "+") + m))
}

h1.bis <- glsAdd(x, u, v, type = 1, verbose = FALSE)
h1.bis$loss
```

```
## [1] 31.41217
```

```
h1$loss
```

```
## [1] 31.46085
```

```
matrank(h1.bis$y)
```

```
## [1] 2
```

```
h3.bis <- glsAdd(x, u, v, type = 3, verbose = FALSE)
h3.bis$loss
```

```
## [1] 0.2753407
```

Notes

1. For the desired precision weighting (either in a compositional or a non-compositional context), I see I need to adapt the algorithm to allow for elementwise weighting, i.e., combine it with `wAddPCA`.
2. Canonical correlation analysis (CCo) is known to provide the “best” GLS low-rank approximation to the between-set correlation matrix $\mathbf{R}_{xy}$. This will no longer be true, for the model $\mu + \alpha_i + \beta_j + \sum a_{is}b_{js}$ with α_i and β_j set to zero can be expected to give a better fit, similar to the δ adjustment for $\mathbf{R}$ in our AM paper.
3. $\mathbf{U}$ and $\mathbf{V}$ are assumed positive definite. In the case of compositional canonical correlation analysis based on the centred logratio transformation, these matrices will be singular (their rows and columns sum to 0). This can be avoided by using a different transformation (the isometric logratio transformation).