LEAST SQUARES ORTHOGONAL POLYNOMIAL COMPONENT ANALYSIS IN R

JAN DE LEEUW AND KEKONA KAI SORENSON

Abstract. We present a simple algorithm (with code in R) to fit multivariate polynomial components or factors to a rectangular data matrix.

1. Introduction

In Least Squares Orthogonal Polynomial Component Analysis [De Leeuw and Sorenson, 2011] we minimize a loss function of the form

$$\sigma(X,B) = \mathbf{SSQ}(Y - \mathcal{P}(X)B'). \tag{1}$$

Here Y is an $n \times m$ *data matrix*, and X is an $n \times p$ matrix of *component scores*, satisfying $X'X = I$. **SSQ**() is the the sum of squares.

$\mathcal{P}$ is *polynomial expansion*, and $\mathcal{P}(X)$ is an $n \times R$ matrix in which row i contains R monomials in the p variables $x_{i1}, \cdots, x_{ip}$. The same monomials are used in each row. Thus we can write $\{\mathcal{P}(X)\}_{ij} = \pi_j(x_i)$, where each π_j is a monomial and x_i is row i of X. If $\mathcal{P}$ is the identity then we fit an ordinary linear principal component analysis.

Date: Monday 7th May, 2012 — 10h 51min — Typeset in Lucida Bright.
2000 *Mathematics Subject Classification.* 65C60,62H25.
Key words and phrases. component analysis, factor analysis, optimization, fit.

The matrix B is an $m \times R$ matrix of *component loadings*.

2. Polynomial Patterns

From the computational point of view we first need to decide how to code the monomials in $\mathcal{P}$. We could use the `multipol` package [Hankin, 2009], but for our purposes the representation of multivariate polynomials used in `multipol` is somewhat wasteful. Instead we code $\mathcal{P}$ using a list of R matrices. Each matrix has two columns and a number of rows equal to the number of factor in the monomial. The first column indicates which variables are included, the second column the power to which they are raised. Thus x_3 becomes $\begin{bmatrix}3 & | & 1\end{bmatrix}$, while $x_2^2 x_5^3$ is $\begin{bmatrix}2 & 2\\5 & 3\end{bmatrix}$.

Code segment 1 creates the *pattern* for a polynomial of a single variable of degree p.

$\boxed{\text{INSERT CODE SEGMENT 1 ABOUT HERE}}$

Code segment 2 can be used to create a quadratic polynomial in p variables. Note that both polynomials do not have a term of degree zero ("intercept"), which our algorithms will handle separately.

$\boxed{\text{INSERT CODE SEGMENT 2 ABOUT HERE}}$

3. Algorithm

Special purpose algorithms, adapted to the structure of the problem at hand, are usually faster than the general optimization methods that come with R. In this paper, however, we will simply use the `BFGS` method in the `optimx` package [Nash and Varadhan, 2011] to get an idea about the difficulty of the optimization problem at hand, and to analyze some relative small examples.

We will minimize the function $\sigma_\star(X) = \min_B \sigma(X, B)$ over all $X'X = I$. This seems contrary to the idea of using BFGS, which is an optimization method for unconstrained problems. It seems that a package like `GrassmanOptim` [Adragni and Wu, 2012] would be more appropriate. We reduce the problem to an unconstrained one by minimizing $\sigma_\star(\mathcal{Q}(Z))$ over Z, where $\mathcal{Q}$ is the *Procrustus transformation*. If $Z = K\Lambda L'$ is the singular value decomposition of Z, then $\mathcal{Q}(Z) = KL'$.

We will use `BFGS` from `optimx` without computing analytic derivatives, although the necessary formulas for the derivatives are given in De Leeuw [2012]. The function to evaluate the loss function is in code segment 3.

$\boxed{\text{INSERT CODE SEGMENT 3 ABOUT HERE}}$

4. EXAMPLES

Code segment 4 generates data using $Y = \mathcal{P}(X)B'$. Thus the algorithm should be able to find a solution with perfect fit if we use the correct polynomial pattern.

$\boxed{\text{INSERT CODE SEGMENT 4 ABOUT HERE}}$

We first try the `polynomialOne` pattern, with the code in segment 5.

$\boxed{\text{INSERT CODE SEGMENT 5 ABOUT HERE}}$

5. UNIQUE FACTORS

References

K.P. Adragni and S. Wu. *GrassmannOptim: Grassmann Manifold Optimization*, 2012. URL `http://CRAN.R-project.org/package=GrassmannOptim`. R package version 1.1.

J. De Leeuw. Derivatives of the Procrustus Transformation, with Applications. Unpublished, 2012.

J. De Leeuw and K. K. Sorenson. Matrix Approximation by Polynomial Factor Analysis. UCLA Department of Statistics, November 2011.

R.K.S. Hankin. *multipol: multivariate polynomials*, 2009. URL `http://CRAN.R-project.org/package=multipol`. R package version 1.0-4.

J.C. Nash and R. Varadhan. Unifying Optimization Algorithms to Aid Software System Users: optimx for R. *Journal of Statistical Software*, 43(9):1–13, 2011. URL `http://www.jstatsoft.org/v43/i09/paper`.

Appendix A. Code

Code Segment 1 polynomialOne Pattern

```
polynomialOne <- function (p) {
    pattern <- rep (list (0), p)
    for (i in 1 : p) {
        pattern[[i]] <- matrix (c (1, i), 1, 2)
    }
    return (pattern)
}
```

Code Segment 2 fullQuadratic Pattern

```
fullQuadratic <- function (p) {
    pp <- p + p * (p + 1) / 2
    pattern <- rep (list (0), pp)
    for (i in 1 : p) {
        pattern[[i]] <- matrix (c (i, 1), 1, 2)
        pattern[[p + i]] <- matrix (c (i, 2), 1, 2)
    }
    k <- 1
    for (i in 2 : p) {
        for (j in 1 : (i - 1)) {
            pattern[[2 * p + k]] <- matrix (c (i, j, 1, 1),
                2, 2)
            k <- k + 1
        }
    }
    return (pattern)
}
```

Code Segment 3 Loss Function Computation

```
polyfit <- function (x, y, pattern, intercept = TRUE,
    unique = FALSE) {
    n <- nrow (y)
    p <- length (x) / n
    x <- matrix (x, n, p)
    s <- svd (x)
    px <- tcrossprod (s $ u, s $ v)
    np <- length (pattern)
    qx <- matrix (0, n, np)
    for (i in 1 : np) {
        qx[, i] <- 1
        pmat <- pattern[[i]]
        for (j in 1 : nrow (pmat)) {
            qx[, i] <- qx[, i] * px [, pmat [j, 1]] ^ pmat
                [j, 2]
        }
    }
    if (intercept) {
        qx <- cbind (1, qx)
    }
    b <- qr.solve (qx, y)
    res <- y - qx %*% b
    return (sum (res * res))
}
```

Code Segment 4 Generate Perfect Fit Data

```
make_initial <- function (n, m, p, pattern, seed = 12345,
    intercept = TRUE, random = TRUE) {
    set.seed (seed)
    px <- qr.Q (qr (matrix (rnorm (n * p), n, p)))
    np <- length (pattern)
    qx <- matrix (0, n, np)
    for (i in 1 : np) {
        qx[, i] <- 1
        pmat <- pattern[[i]]
        for (j in 1 : nrow (pmat)) {
            qx[, i] <- qx[, i] * px [, pmat [j, 1]] ^ pmat
                [j, 2]
        }
    }
    if (intercept) {
        qx <- cbind (1, qx)
    }
    nq <- ncol (qx)
    mq <- nq * m
    b <- matrix (sample (1 : mq, mq, replace = FALSE), nq,
        m)
    y <- qx %*% b
    if (random) {
        x <- qr.Q (qr (matrix (rnorm (n * p), n, p)))
    }
    else {
        x <- (svd (y) $ u)[, 1 : p]
    }
    return (list (x = x, px = px, b = b, y = y))
}
```

Code Segment 5 Example with polynomialOne pattern

```
pattern <- polynomialOne (4)
ini <- make_initial (100, 4, 1, seed = 54321, pattern,
    random = TRUE)
res <- optimx (par = as.vector (ini $ x), fn = polyfit,
    method = "BFGS",
     control = list (trace = 1, maxit = 1500), y = ini $ y,
         pattern = pattern)
xopt <- matrix ((res $ par) [[1]] , 100, 1)
popt <- xopt / sqrt ( sum (xopt ^ 2))
px <- ini $ px
np <- length (pattern)
plot (px, popt)
qopt <- matrix (0, 100, np)
for (i in 1 : np) {
    qopt[, i] <- 1
    pmat <- pattern[[i]]
    for (j in 1 : nrow (pmat)) {
        qopt[, i] <- qopt[, i] * popt [, pmat [j, 1]] ^
            pmat [j, 2]
    }
}
qopt <- cbind (1, qopt)
print (bopt <- qr.solve (qopt, ini $ y))
```

Code Segment 6 Example with fullQuadratic pattern

```
pattern <- fullQuadratic (2)
ini <- make_initial (100, 4, 2, pattern, random = FALSE)
res <- optimx (par = as.vector (ini $ x), fn = polyfit,
    method = "BFGS",
    control = list (trace = 1, maxit = 1500), y = ini $ y,
        pattern = pattern)
xopt <- matrix (res $ par[[1]], 100, 2)
px <- ini $ px
np <- length (pattern)
print (qr.solve (xopt, px) %*% qr.solve (px, xopt))
sopt <- svd (xopt)
popt <- tcrossprod (sopt $ u, sopt $ v)
qopt <- matrix (0, 100, np)
for (i in 1 : np) {
    qopt[, i] <- 1
    pmat <- pattern[[i]]
    for (j in 1 : nrow (pmat)) {
        qopt[, i] <- qopt[, i] * popt [, pmat [j, 1]] ^
            pmat [j, 2]
    }
}
qopt <- cbind (1, qopt)
print (bopt <- qr.solve (qopt, ini $ y))
```

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: `deleeuw@stat.ucla.edu`

E-mail address, Kekona Kai Sorenson: `kekona@gmail.com`

URL, Jan de Leeuw: `http://gifi.stat.ucla.edu`

URL, Kekona Kai Sorenson: `http://www.kekonakai.com/Site/Welcome.html`