Comparing NIPALS functions in R

Kevin Wright

October 27, 2017

There are at least 4 R packages with a function for performing NIPALS on a matrix that contains missing values (and several other packages have functions which do not allow missing values). These functions have slightly different scalings for the returned values, and were written with different coding styles.

With careful attention to some of the scaling details of the returned values, all 4 packages produce the same results. However, there are dramatic differences in performance.

NOTE! These comparisons were made before `nipals::nipals` gained the Gram-Schmidt orthogonalization by default.

Example data

A small dataset with 2 missing values in the first column will be used to compare the numerical results from the 4 packages.

```
B <- matrix(c(50, 67, 90, 98, 120,
              55, 71, 93, 102, 129,
              65, 76, 95, 105, 134,
              50, 80, 102, 130, 138,
              60, 82, 97, 135, 151,
              65, 89, 106, 137, 153,
              75, 95, 117, 133, 155), ncol=5, byrow=TRUE)
rownames(B) <- c("G1","G2","G3","G4","G5","G6","G7")
colnames(B) <- c("E1","E2","E3","E4","E5")

B2 = B
B2[1,1] = B2[2,1] = NA
B2 <- as.matrix(B2)

same <- function(a,b, tol=1e-3){
  all.equal( abs(a), abs(b), tol=tol, check.attributes=FALSE)
}
```

Since principal components are only unique up to a change of sign, a small function `same()` has been defined to take absolute values before calling `all.equal`. The `same()` function will be used to compare results from the differnt functions. In the next 3 sections, the results from the `nipals` package are compared to the `ade4`, `plsdepot`, and `mixOmics` packages respectively.

ade4

The `ade4` package uses a maximum-likelihood scaling of the data which divides by `n` instead of `n-1`, so we need to scale the data by hand before using the `nipals` package. Note: only for ade4 version >= 1.7-10.

```
library(ade4)
made <- ade4::nipals(B2, nf=5, rec=TRUE, niter=500, tol=1e-9)
```

```r
B2a <- apply(B2, 2, function(x) {
  n <- sum(!is.na(x))
  x <- x - mean(x, na.rm=TRUE)
  x <- x / ( sd(x, na.rm=TRUE) * sqrt((n-1) / n ))
})

mnip <- nipals::nipals(B2a, ncomp=5, center=FALSE, scale=FALSE, fitted=TRUE, maxiter=500, tol=1e-9, gra
```

The eigenvalues reported by `ade4` are the squared singular values divided by $n-1$.

```r
# data
same(B2a, as.matrix(made$tab))
# TRUE

# eigenvalues, ade4 uses squared singular values / n-1
mnip$eig
# [1] 5.2913781 2.2555596 1.1651281 0.2590878 0.1563175
made$eig
# [1] 4.666454778 0.847924398 0.226254436 0.011187921 0.004072542
same(mnip$eig ^ 2 / (nrow(B2a)-1), made$eig)
# TRUE

# P loadings
same(mnip$loadings, made$c1)
# TRUE

# T scores. For nipals, sweep IN the eigenvalues
same( sweep(mnip$scores, 2, mnip$eig, "*"), made$li)
# TRUE
```

plsdepot

```r
library(plsdepot)
mpls <- plsdepot::nipals(B2, comps=5)
library(nipals)
mnip <- nipals::nipals(B2a, ncomp=5, maxiter=100, tol=1e-6, gramschmidt=FALSE)
```

The `plsdepot` package reports squared singular values.

```r
# eigenvalues
mnip$eig
# [1] 4.8762167 2.0442757 1.0728055 0.2369607 0.1432779
mpls$values[,1]
# [1] 3.963172007 0.696484184 0.191839875 0.009366425 0.003421661
same(mnip$eig, sqrt(mpls$values[,1] * 6) )
# TRUE

# P loadings
mnip$loadings
mpls$loadings
same(mnip$loadings, mpls$loadings, tol=1e-2 )
# TRUE
```

```
# T scores
mnip$scores
mpls$scores
same( sweep(mnip$scores, 2, mnip$eig, "*"), mpls$scores)
# TRUE
```

mixOmics

```
library(mixOmics)
library(nipals)
mnip <- nipals::nipals(B2, gramschmidt=FALSE)
mmix <- mixOmics::nipals(scale(B2), ncomp=5)
```

```
# eigenvalues
mnip$eig
mmix$eig
same(mnip$eig, mmix$eig)
# TRUE

# P loadings
mnip$loadings
mmix$p
same(mnip$loadings, mmix$p, tol=1e-2)
# TRUE

# T scores
mnip$scores
mmix$t
same(mnip$scores, mmix$t, tol=1e-2)
TRUE
```

Performance comparison

For the purpose of comparing performance of the functions, we simulate a 100 x 100 matrix and insert one missing value.

```
set.seed(43)
Bbig <- matrix(rnorm(100*100), nrow=100)
Bbig2 <- Bbig
Bbig2[1,1] <- NA
```

The `ade4::nipals` function uses `for` loops to loop over the columns of `X`, which results in very slow execution even when calculating only 1 principal component.

```
system.time(ade4::nipals(Bbig2, nf=1)) # Only 1 factor!
##  user  system elapsed
## 42.09    0.00   42.14
```

The `plsdepot::nipals` function is fast enough that all 100 PCs can be calculated.

```
system.time(plsdepot::nipals(Bbig2, comps=1)) # Only 1 factor !
#   user  system elapsed
```

```
#    0.5     0.0     0.5
system.time(plsdepot::nipals(Bbig2, comps=100)) # 100 factors
#   user  system elapsed
#  30.19    0.00   30.18
```

The `mixOmics::nipals` function uses `crossprod` and a few other tricks to improve performance.

```
system.time(mixOmics::nipals(scale(Bbig2), ncomp=100)) # 100 factors
#   user  system elapsed
#  20.70    0.00   20.81
```

The `nipals::nipals` function was optimized through extensive testing and is about 5 times faster!

```
system.time(nipals::nipals(Bbig2, ncomp=100, gramschmidt=FALSE)) # 100 factors
#   user  system elapsed
#   2.74    0.00    2.75
```