

# Lecture\_notes6

10.03-2021.

# Recall Gram Schmidt and QR

The orthogonal vectors of the  $Q$ -matrix produced by the Gram-Schmidt orthogonalization process can be written in terms of projections as follows

$$\mathbf{q}_1 = \frac{1}{\|P_1 \mathbf{a}_1\|} P_1 \mathbf{a}_1, \mathbf{q}_2 = \frac{1}{\|P_2 \mathbf{a}_2\|} P_2 \mathbf{a}_2, \dots, \mathbf{q}_n = \frac{1}{\|P_n \mathbf{a}_n\|} P_n \mathbf{a}_n,$$

where  $P_1 = I_m$ ,  $P_j = I_m - Q_{j-1} Q_{j-1}^t$  and  $Q_{j-1} = [\mathbf{q}_1 \ \mathbf{q}_2 \ \dots \ \mathbf{q}_{j-1}]$ .

## Exercise 1:

What are the  $Q_{j-1}$ -coordinates of  $\hat{\mathbf{a}} = Q_{j-1} Q_{j-1}^t \mathbf{a}$ ?

How can we conclude the following matrix identity

$$Q_{j-1} Q_{j-1}^t = \sum_{k=1}^{j-1} \mathbf{q}_k \mathbf{q}_k^t?$$

## Exercise 2:

Verify that  $P_j = I_m - Q_{j-1} Q_{j-1}^t$  is a projection matrix, i.e show that  $P_j^2 = P_j$  and  $P_j = P_j^t$ .

**Exercise 3:**

Recall that the orthogonal complement of a subspace  $W \subseteq \mathbf{R}^m$  is defined as

$$W^\perp = \{\mathbf{v} \in \mathbf{R}^m \mid \mathbf{v}^t \mathbf{w} = 0 \text{ for all } \mathbf{w} \in W\}.$$

Verify that  $P_j$  is the projection onto  $Col([\mathbf{q}_1 \dots \mathbf{q}_{j-1}])^\perp = Col([\mathbf{a}_1 \dots \mathbf{a}_{j-1}])^\perp$ .

# The modified Gram Schmidt

The modified Gram-Schmidt (MGS) algorithm works by converting the matrix  $A$  into the matrix  $Q$ . Once  $\mathbf{q}_k$  is found, the remaining  $A$ -columns (of index  $j > k$ ) are modified to be orthogonal to that  $\mathbf{q}_k$ . This is called a *deflation* of  $A$  w.r.t.  $\mathbf{q}_k$ .

**Exercise 4:**

Define the projection  $P_{\perp \mathbf{q}_k} = I_m - \mathbf{q}_k \mathbf{q}_k^t$ . Show that

$$\mathbf{q}_k^t P_{\perp \mathbf{q}_k} A = 0,$$

and that

$$P_j = P_{\perp \mathbf{q}_{j-1}} P_{\perp \mathbf{q}_{j-2}} \dots P_{\perp \mathbf{q}_2} P_{\perp \mathbf{q}_1}.$$

**Exercise 5:**

Make a Julia-function that does the QR-factorization according to MGS and returns the matrices  $Q$  and  $R$ .

```
function MGS(A)
```

## Julia-code for QR by the CGS

```
1 using LinearAlgebra
2 function CGS(A)
3 # QR-factorization of input-matrix A by the ...
  classical Gram Schmidt algorithm (CGS).
4 # The function returns Q and R in the QR-factorization.
5 m, n = size(A);
6 p = min(m,n)
7 Q = zeros(m,p);
8 R = zeros(p,n);
9 R[1,1] = norm(A[:,1]);
10 Q[:,1] = A[:,1]./R[1,1];
11 for i = 2:n
12     if i ≤ p
13         R[1:i-1,i] = Q[:,1:i-1]'A[:,i];
14         v = A[:,i]-Q[:,1:i-1]*R[1:i-1,i]; # ...
          Orthogonalize i-th column wrt Q[:,1:i-1].
15         R[i,i] = norm(v);
16         Q[:,i] = v./R[i,i];
17     else
18         R[:,i] = Q'A[:,i];           # Find ...
          Q-coordinates of A[:,i]
19     end
20 end
21 return Q, R;
22 end
```

### Exercise 6:

Let  $\epsilon = 10^{-7}$ ,  $A = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix}$ ,  $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$  and  $\mathbf{b} = A\mathbf{x}$ .

Solve the system  $A\mathbf{x} = \mathbf{b}$  by

1. the normal equations  $A^t A \mathbf{x} = A^t \mathbf{b}$ .
2. by QR-factorization obtained from the CGS.
3. by QR-factorization obtained from the MGS.
4. by the QR-factorization in Julia.
5. by the *backslash* "`\`" operator.
6. by the pseudo-inverse of  $A$ .

Compare and comment on the results.

The MGS is known to be numerically more stable than the classical Gram-Schmidt (CGS). This can be illustrated by running both algorithms on a matrix  $A$  with decreasingly small singular values.

We can design such a matrix  $A = USV^t$  (say  $60 \times 60$ ) by generating random data and extract appropriate orthogonal right and left singular vectors. As singular values we may use the values  $s_i = 2^{-i}$  for  $i = 1, \dots, 60$ .

### Exercise 7:

1. Make a Julia-script that generates  $U$ ,  $S$  and  $V$  to design the desired matrix  $A = USV^t$ .
2. Apply your Julia QR-factorization functions based on CGS and MGS to  $A$  in addition to the Julia-internal QR.
3. Compare the 60 diagonal elements of the resulting  $R$ -matrices by plotting them together (use log10-scale for the plotting).

## Partial Least Squares (PLS)

Partial least squares (PLS, see **Fast and stable partial least squares modelling: A benchmark study with theoretical comments, by Björck & Indahl, 2017**) is a method for computing approximate least squares solutions. It is much used in fields like **Chemometrics** and **Genomics** where prediction modelling based on high-dimensional data and "wide" datamatrices  $\mathbf{X}$  ( $n \gg m$ ) is quite common.

The idea behind PLS is to take advantage of the responses  $y$  to obtain projection models requiring fewer and more appropriate components than PCR to obtain good regression models for predicting  $y$ -values from  $X$ -data.

Most PLS-methods/algorithms have been developed "empirically" by people that were not experts in numerical linear algebra.

The Björck–Indahl collaboration was initiated as a consequence of the following two publications:

**Stability of Two Direct Methods for Bidiagonalization and Partial Least Squares, by Björck (2014)** presenting an investigation of the stability issues for two PLS algorithms (the *Golub-Kahan Householder bidiagonalization method* and the *NIPALS PLS*) is presented.

**The geometry of PLS1 explained properly: 10 key notes on mathematical properties of and some alternative algorithmic approaches to PLS1 modelling, by Indahl (2014)** focusing on understanding various aspects of the PLS-methodology in terms of elementary linear algebra.



# The NIPALS PLS algorithm

The classical algorithm for PLS modelling was published by Wold et al. in 1984.  $\mathbf{X}_0$  and  $\mathbf{y}_0$  represent the mean-centered data (predictors) and responses.

```

for  $a = 1 : k$       ( $k$ , # components to be extracted)
  1.  $\mathbf{w}_a = \mathbf{X}_{a-1}^t \mathbf{y}_{a-1}$  ( $\mathbf{y}$ ,  $\mathbf{X}$  inner prods  $\sim$  covariances)
  2.  $\mathbf{w}_a = \mathbf{w}_a / \|\mathbf{w}_a\|$  (normalizing the vector from 1)
  3.  $\mathbf{t}_a = \mathbf{X}_{a-1} \mathbf{w}_a$  ( $\mathbf{X}$ -comb wrt weights from 2)
  4.  $\mathbf{t}_a = \mathbf{t}_a / \|\mathbf{t}_a\|$  (normalizing the vector from 3)
  5.  $\mathbf{p}_a = \mathbf{X}_{a-1}^t \mathbf{t}_a$  ("projection-loadings" used in 6)
  6.  $\mathbf{X}_a = \mathbf{X}_{a-1} - \mathbf{t}_a \mathbf{p}_a^t$  (deflate  $\mathbf{X}$  to be orthogonal to  $\mathbf{t}_a$ )
  7.  $q_a = \mathbf{t}_a^t \mathbf{y}_{a-1}$  (reg. coeff wrt score vector  $\mathbf{t}_a$ )
  8.  $\mathbf{y}_a = \mathbf{y}_{a-1} - \mathbf{t}_a q_a$  (make  $\mathbf{y}$  orthogonal to  $\mathbf{t}_a$ )

```

end

Organize vectors and numbers into the matrices:

```

 $\mathbf{T}_k = [\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_k]$       (the orthonormal scores)
 $\mathbf{W}_k = [\mathbf{w}_1 \ \mathbf{w}_2 \ \dots \ \mathbf{w}_k]$     (the orthonormal weights)
 $\mathbf{P}_k = [\mathbf{p}_1 \ \mathbf{p}_2 \ \dots \ \mathbf{p}_k]$     (the  $\mathbf{X}$ -projection loadings)
 $\mathbf{q}_k^t = [q_1 \ q_2 \ \dots \ q_k]$       (the  $\mathbf{T}$  regression coeffs)

```

Finally calculate the regression coeffs for the original  $\mathbf{X}$ -data by:  $\beta_k = \mathbf{W}_k (\mathbf{P}_k^t \mathbf{W}_k)^{-1} \mathbf{q}_k$  and  $\beta_{0,k} = \bar{y} - \bar{x} \beta_k$ , where  $\bar{y}$  and  $\bar{x}$  denote  $\mathbf{y}$  and  $\mathbf{X}$  column means.

The NIPALS algorithm computing a  $k$ -component PLS model with orthonormal scores ( $\mathbf{T}_k$ ) and weights ( $\mathbf{W}_k$ ).

The above algorithm extracts weights  $\mathbf{W}_k$  and scores  $\mathbf{T}_k$  ( $\mathbf{W}_k^t \mathbf{W}_k = \mathbf{I}_k$  and  $\mathbf{T}_k^t \mathbf{T}_k = \mathbf{I}_k$ ) representing orthogonal bases for the so-called **Krylov subspaces**  $\mathcal{K}_k(\mathbf{X}^t \mathbf{X}, \mathbf{X}^t \mathbf{y})$  and  $\mathcal{K}_k(\mathbf{X} \mathbf{X}^t, \mathbf{X} \mathbf{X}^t \mathbf{y})$ , respectively.

The weights  $\mathbf{w}_a$  found in steps 1. and 2. of the NIPALS algorithm provide a linear combination of the  $\mathbf{X}_{a-1}$ -columns that maximizes the covariance with  $\mathbf{y}_{a-1}$ .

**NOTE 1:**

The Krylov subspace

$$\begin{aligned}\mathcal{K}_k(\mathbf{X}^t \mathbf{X}, \mathbf{X}^t \mathbf{y}) &= \text{Span}\{\mathbf{X}' \mathbf{y}, (\mathbf{X}^t \mathbf{X}) \mathbf{X}^t \mathbf{y}, \dots, (\mathbf{X}^t \mathbf{X})^{k-1} \mathbf{X}' \mathbf{y}\} \\ &= \text{Col}(\mathbf{W}_k)\end{aligned}$$

is a subspace of the vector space  $\text{Col}(\mathbf{X}^t)$  spanned by the rows of  $\mathbf{X}$ , and

$$\begin{aligned}\mathcal{K}_k(\mathbf{X} \mathbf{X}^t, \mathbf{X} \mathbf{X}^t \mathbf{y}) &= \text{Span}\{(\mathbf{X} \mathbf{X}^t) \mathbf{y}, (\mathbf{X} \mathbf{X}^t)^2 \mathbf{y}, \dots, (\mathbf{X} \mathbf{X}^t)^k \mathbf{y}\} \\ &= \text{Col}(\mathbf{T}_k).\end{aligned}$$

is a subspace of the vector space  $\text{Col}(\mathbf{X})$  spanned by the columns of  $\mathbf{X}$ .

---

After the orthogonal scores in  $\mathbf{T}_k$  are calculated, the matrix of projection loadings ( $\mathbf{P}_k$ ) satisfies the identity:

$$\mathbf{P}_k^t = \mathbf{T}_k^t \mathbf{X}.$$

The columns of  $\mathbf{T}_k^* = \mathbf{XW}_k$  are often referred to at the *non-orthogonal scores*. Obviously, the columns of  $\mathbf{T}_k^*$  and  $\mathbf{T}_k$  (the orthogonal scores from the NIPALS-algorithm) span the same subspace of  $\mathbb{R}^m$ .

### **NOTE 2:**

Clearly (why)

$$\mathbf{T}_k^* = \mathbf{T}_k \mathbf{T}_k^t \mathbf{T}_k^* = \mathbf{T}_k \mathbf{P}_k^t \mathbf{W}_k.$$

The above relationship between  $\mathbf{T}_k$  and  $\mathbf{T}_k^*$  corresponds to a QR-factorization of  $\mathbf{T}_k^*$  where  $\mathbf{T}_k$  plays the role of  $\mathbf{Q}$  and  $\mathbf{B}_k$  the role of  $\mathbf{R}$ :

$$\mathbf{T}_k^* = \mathbf{T}_k \mathbf{B}_k, \text{ where } \mathbf{B}_k = \mathbf{P}_k^t \mathbf{W}_k.$$

It turns out that the  $k \times k$  upper triangular  $\mathbf{B}_k$  matrix is actually upper bi-diagonal (only the main diagonal of  $\mathbf{B}_k$  and the first super-diagonal above it are non-zero) in this case. Such bi-diagonal matrices can be inverted very effectively when necessary.

The projection of  $\mathbf{y}$  onto the column space of  $\mathbf{T}_k$  is

$$\hat{\mathbf{y}} = \mathbf{T}_k \mathbf{T}_k^t \mathbf{y} = \mathbf{T}_k \mathbf{q}_k.$$

Here the  $\mathbf{T}_k$ -regression coeffs  $\mathbf{q}_k = \mathbf{T}_k^t \mathbf{y}$  are often referred to as the  $\mathbf{y}$ -loadings.

Because the orthogonal scores  $\mathbf{T}_k = \mathbf{XW}_k(\mathbf{P}_k^t \mathbf{W}_k)^{-1}$ , we therefore have

$$\hat{\mathbf{y}} = \mathbf{T}_k \mathbf{q}_k = \mathbf{XW}_k(\mathbf{P}_k^t \mathbf{W}_k)^{-1} \mathbf{q}_k,$$

where the PLSR regression coeffs are directly expressed as

$$\beta_k = \mathbf{W}_k(\mathbf{P}_k^t \mathbf{W}_k)^{-1} \mathbf{q}_k.$$

For predictions with uncentered  $\mathbf{X}$ -data, the associated constant term for the  $k$ -component PLS regression (PLSR) model is

$$\beta_{0,k} = \bar{y} - \bar{\mathbf{x}}\beta_k,$$

and the PLSR predictions based on  $k$  PLS-components for new data points  $\mathbf{x}$  (a row-vector in  $\mathbb{R}^n$ ) is

$$\hat{y} = \beta_{0,k} + \mathbf{x}\beta_k.$$

### **NOTE 3:**

The NIPALS algorithm actually implements a Gram-Schmidt (GS) process deriving the orthogonal scores ( $\mathbf{T}_k$ ) by starting with linear combinations of  $\mathbf{X}$ -columns rather than just a single  $\mathbf{X}$ -column. The steps 5 and 6 in NIPALS represent a GS-step "deflating" the  $(m \times n)$ -matrix  $\mathbf{X}_{a-1}$  with respect to  $\mathbf{t}_a$  to assure that the subsequent scores ( $\mathbf{T}_k$ -columns) are orthogonal. The main difference from the MGS algorithm for QR-factorization is that each GS-step in the MGS only "deflate" a subset of the matrix columns.

### **NOTE 4:**

From the identity  $\mathbf{X}\mathbf{W}_k = \mathbf{T}_k^*$  we obtain an approximation  $\tilde{\mathbf{X}}_k$  of  $\mathbf{X}$  by multiplying both sides of the equation

from the right with  $\mathbf{W}_k^t$ :

$$\mathbf{X} \approx \mathbf{X}\mathbf{W}_k\mathbf{W}_k^t = \mathbf{T}_k\mathbf{B}_k\mathbf{W}_k^t \stackrel{\text{def}}{=} \tilde{\mathbf{X}}_k.$$

Here the right-multiplication  $\mathbf{W}_k\mathbf{W}_k^t$  projects the  $\mathbf{X}$ -rows onto  $\mathcal{K}_k(\mathbf{X}^t\mathbf{X}, \mathbf{X}^t\mathbf{y})$ .

**Exercise 8:**

Recall from Lecture-notes4 the (truncated) pseudo-inverse for the rank  $k$  approximation of  $\mathbf{X}$  obtained by the SVD.

Suggest a similar definition  $\tilde{\mathbf{X}}_k^\dagger$  based on the bi-diagonal factorization of  $\tilde{\mathbf{X}}_k = \mathbf{T}_k\mathbf{B}_k\mathbf{W}_k^t$  that is consistent with calculation of the PLSR regression coeffs  $\beta_k$ .

# The PLS regression problem

## Recall:

The ordinary least squares (OLS) problem for any linear system  $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$  corresponds to solving

$$\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2 \quad \text{subject to} \quad \min \|\boldsymbol{\beta}\|_2$$

always has a unique solution  $\hat{\boldsymbol{\beta}}$  called the *pseudoinverse solution*. This solution is characterized by satisfying the two conditions

$$\mathbf{X}^t \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^t \mathbf{y}, \quad \boldsymbol{\beta} \in \text{Span}(\mathbf{X}^t) \subseteq \mathbb{R}^n,$$

i.e.,  $\hat{\boldsymbol{\beta}}$  is the solution of the associated normal equations that is also contained in the row subspace of  $\mathbf{X}$ .

---

The **PLS approximations**  $\boldsymbol{\beta}_k$ ,  $k = 1, 2, \dots$  to the least squares problem can be defined as the estimates generated by  $k$  steps of the NIPALS PLS algorithm.

However, a definition of the approximate  $k$ -component PLS solutions should be independent of a particular algorithm.

We therefore define the *PLS approximation*  $\boldsymbol{\beta}_k$  as *the solution of the restricted problem*

$$\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2, \quad \text{subject to} \quad \boldsymbol{\beta} \in \mathcal{K}_k(\mathbf{X}^t \mathbf{X}, \mathbf{X}^t \mathbf{y}).$$

From latter definition, alternative algorithms for finding the PLS approximations can be studied (see Björck & Indahl, 2017)).

## Julia-code for the NIPALS PLS

```
1 using LinearAlgebra, Statistics
2 function PLS_nip(X, y; mc = 2)
3 # b0, B, T, W, P, q = PLS_nip(X,y; mc = 6)
4 m,n = size(X)
5 mc = min(mc, min(n,m)-1)
6 # Scores (T), weights (W) and X-loadings (P):
7 T = zeros(m,mc); W = zeros(n,mc); P = zeros(n,mc)
8 q = zeros(1,mc); # the y-loadings
9 mx = mean(X, dims=1) # - X-column mean values.
10 my = mean(y, dims=1)[1] # - mean of responses y.
11 y = y.- my; # - centered response vector.
12 X = X.- mx; # - centered X-data
13 for a = 1:mc
14     w = X'y; w = w/norm(w); W[:,a] = w;
15     t = X*w; t = t/norm(t); T[:,a] = t;
16     P[:,a] = X't; X = X - t*P[:,a]';
17     q[a] = (y't)[1]; y = y - q[a].*t;
18 end
19 # The regression-coeffs (B) and constant terms (b0):
20 B = cumsum((W/triu(P'W)).*q, dims = 2);
21 b0 = my .- mx*B;
22 return b0, B, T, W, P, q;
23 end
```

**Exercise 9:**

Extend last weeks programming exercise with the Julia-script `RR_PCR_Exercise.jl` to include PLS-modelling with the same number of components as for PCR.

## Some useful "local" contributions to PLS-modelling:

- **Much faster cross-validation in PLSR-modelling by avoiding redundant calculations, see Liland et al. (2020).** About fast selection/validation of PLS models.
- **ROSA - a fast extension of partial least squares regression for multiblock data analysis, Liland et al. (2016).** Multiblock data analysis (a.k.a. *data fusion*) is about combining different measurements (data matrices) obtained from the same set of samples.
- **Canonical partial least squares - a unified PLS approach to classification and regression problems, see Indahl et al. (2009).** About estimating latent variables in multivariate classification and regression problems where more than one response variable is available.