# Computing Partial Least Squares Approximations

Åke Björck

Seminarium, Son, Norge, 19–21 August 2015

Dedicated to the memory of Axel Ruhe (1943–2015)

# Outline

PLS and Krylov Subspaces

Orthogonalization in Finite Precision

The NIPALS PLS Algorithm

Golub–Kahan Bidiagonalization

# PLS and Krylov Subspaces

Consider the linear least squares problem

$$\min_b \|Xb - y\|_2, \quad X \in \mathbf{R}^{m\times n}, \quad y \in \mathbf{R}^m.$$

For any shape and rank of $X$ there is a unique solution $b^\dagger = X^\dagger y$ of minimum norm ($X^\dagger$ is the pseudoinverse of $X$).

We define the Partial Least Squares (PLS) approximations $b_k$ to be the solutions to the subproblems

$$\min_b \|Xb - y\|_2, \ \text{ subject to } \ b \in \mathcal{K}_k(X^T X, X^T y),$$

where $k = 1, 2, \ldots$ and

$$\mathcal{K}_k(X^T X, X^T y) = \operatorname{span}\{X^T y, (X^T X)X^T y, \ldots, (X^T X)^{k-1} X^T y\}.$$

# PLS and Krylov Subspaces

Subspaces of the form $\mathcal{K}_k(B, c) = \operatorname{span}\{c, Bc, \ldots, B^{k-1}c\}$ are known as Krylov subspaces after the Russian mathematician A. N. Krylov (1863–1945).

The residual vectors lie in the Krylov subspaces

$$r_k = y - Xb_k \in \mathcal{K}_{k+1}(XX^T, y), \quad k = 1, 2, \ldots.$$

Let $B^s c$ be the first Krylov vector that can be expressed as a linear combination of $c, Bc, \ldots, B^{(s-1)}c$. Then $s$ is the maximum dimension of the Krylov subspaces generated by $B$ and $c$.

# PLS and Krylov Subspaces

The following finite termination property holds for the PLS approximations.

**Theorem**

*The grade $s$ of the Krylov subspaces $\mathcal{K}(X^T X, y)$ equals the number of distinct nonzero singular values $\sigma_i$ of $X$ for which $y$ has a nonzero projection onto a left singular vector $u_i$. The PLS approximation $b_s$ equals the pseudoinverse solution $b^\dagger = X^\dagger y$.*

Note that $s$ may be smaller than the rank of $X$. For example, let $y = u_i$, where $u_i$ is a left singular vector of $X$. Then $u_j^T y = 0$, $j \neq i$, and $s = 1$.

# PLS and Krylov Subspaces

The Krylov vectors are very bad basis vectors for the subspaces $\mathcal{K}_k(B, c)$. This may lead to numerical problems for Krylov subspace algorithms. For example, let

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 10^{-4} \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The vectors $w_k = B^k c$, $k = 0, 1, 2, \ldots$, rapidly become almost linearly dependent, for example,

$$B^5 c = \begin{pmatrix} 1 \\ 10^{-20} \end{pmatrix} \approx \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The Krylov vectors converge to an eigenvector to the dominating eigenvalue of $B$; cf. the power method for computing eigenvalues!

# PLS and Krylov Subspaces

A useful invariance property of the PLS Krylov subspaces:

Let $U \in \mathbf{R}^{m \times m}$, $V \in \mathbf{R}^{n \times n}$ be square orthogonal matrices such that $X = UBV^T$. Then

$$\mathcal{K}_k(X^T X, X^T y) = V \mathcal{K}_k(B^T B, B^T c).$$

It follows that $Xb = y$ is equivalent to $Bz = d$, where

$$B = U^T X V, \quad z = V^T x, \quad d = U^T y.$$

The Krylov subspace approximations for the original and transformed systems are related by $b_k = V z_k$.

# PLS and Krylov Subspaces

If $U$ and $V$ are the matrices of singular vectors of $X$ and $p = \mathrm{rank}\,(X)$, then

$$X = U_p \Sigma V_p^T, \quad \Sigma = \mathrm{diag}\,(\sigma_1, \ldots, \sigma_p),$$

is the SVD of $X$ and the transformed system is diagonal $\Sigma z = d$.

Since the SVD is expensive to compute, this is mostly of theoretical interest. A better choice is to take $U$ and $V$ such that $X = UBV^T$, with $B$ is upper bidiagonal.

# Outline

PLS and Krylov Subspaces

Orthogonalization in Finite Precision

The NIPALS PLS Algorithm

Golub–Kahan Bidiagonalization

# Orthogonalization in Finite Precision

In floating-point arithmetic operations are performed as if the result were first calculated to infinite precision and then rounded to a floating-point number. Hence,

$$fl\,(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \leq \mathbf{u}, \tag{1}$$

where $\mathbf{u}$ is the unit roundoff and "op" stands for one of the four elementary operations $+$, $*$, $\cdot$, and $/$. In IEEE standard double precision $\mathbf{u} = 1.11 \cdot 10^{-16}$.

In general, mathematically equivalent algorithms are not numerically equivalent and may produce (very) different results in floating-point arithmetic.

# Orthogonalization in Finite Precision

Using floating-point computation there may be a severe loss of orthogonality in the Gram–Schmidt orthogonalization process.

Let $X = (a_1, a_2)$, where $a_1 = q_1$ and $a_2$ are vectors of unit length. Then the component of $a_2$ orthogonal to $q_1$ is

$$\hat{q}_2 = (I - q_1 q_1^T) a_2 = a_2 - r_{12} q_1, \quad r_{12} = q_1^T a_2.$$

Normalizing this vector to unit length, we obtain

$$q_2 = \hat{q}_2 / r_{22}, \quad r_{22} = \|\hat{q}_2\|_2 = \sin \angle(a_1, a_2).$$

If $r_{22} \ll \|a_2\|$ cancellation has occurred. Then $|q_1^T q_2| < n\mathbf{u}/r_{22}$, i.e., the computed vector $q_2$ is not orthogonal to $q_1$ to full precision.

# Orthogonalization in Finite Precision

When orthogonalizing more than two vectors, there are two different ways to proceed. Consider a third vector $a_3$.

1. Classical Gram–Schmidt (CGS). Compute projections

$$r_{13} = q_1^T a_3, \quad r_{23} = q_2^T a_3, \qquad \hat{q}_3 = a_3 - r_{13} q_1 - r_{13} q_1,$$

   and normalize.

2. Modified Gram–Schmidt (MGS).

$$\begin{aligned} r_{13} &= q_i^T a_3, \quad a_3^{(2)} = a_3 - r_{13} q_1, \\ \hat{q}_3 &= a_3^{(2)} - r_{23} q_2, \quad r_{23} = q_2^T a_3^{(2)}, \end{aligned}$$

   and normalize.

Note that in MGS vectors are deflated before the next projection is computed.

# Orthogonalization in Finite Precision

The loss of orthogonality will propagate differently to later vectors in CGS and MGS.

**Theorem (Björck (1967))**

*For MGS the loss of orthogonality is bounded by*

$$c(m,n)\kappa(X)\mathbf{u},$$

*where* $\kappa(X) = \sigma_{max}(X)/\sigma_{min}(X)$ *is the condition number of* $X$.

Since the distance from $X$ to a rank deficient matrix is $1/\kappa(X)$, MGS gives "good enough" orthogonality for many purposes. CGS is worse by more than a factor $\kappa(X)$.

# Outline

PLS and Krylov Subspaces

Orthogonalization in Finite Precision

The NIPALS PLS Algorithm

Golub–Kahan Bidiagonalization

# The NIPALS PLS Algorithm

The original NIPALS PLS algorithm is due to Svante Wold, Axel Ruhe, Harald Wold, and W. J. Dunn (1984).

$$
\begin{aligned}
&X_0 = X; \quad y_0 = y;\\
&k = 1, 2, \ldots, p\\
&\quad v_k = X_{k-1}^T y_{k-1}; \quad v_k = v_k / \|v_k\|_2;\\
&\quad u_k = X_{k-1} v_k; \quad u_k = u_k / \|v_k\|_2;\\
&\quad (p_k^T, \zeta_k) = u_k^T (X_{k-1}, y_{k-1});\\
&\quad (X_k, y_k) = (X_{k-1}, y_{k-1}) - u_k (p_k^T, \zeta_k);
\end{aligned}
$$

Here $u_k$ are *score vectors*, $v_k$, *loading weight* vectors, and $p_k$ *loading vectors*.

Each step needs three matrix-vector multiplies ($6mn$ flops), one matrix deflation ($2mn$ flops) plus some vector operations.

# The NIPALS PLS Algorithm

In the last two rows the orthogonal projections onto $u_k$ are subtracted from the (modified) data $X_{k-1}$ and $y_{k-1}$. This can be written (c.f. Modified Gram–Schmidt!)

$$(X_k, y_k) = (I - u_k u_k^T)(X_{k-1}, y_{k-1}).$$

Summing equations in the algorithm gives

$$X = U_k P_k^T + X_k, \qquad y = U_k z_k + y_k,$$

where $z_k^T = (\zeta_1, \ldots, \zeta_k)$, $P_k = (p_1, \ldots, p_k)$, and $U_k = (u_1, \ldots, u_k)$.

# The NIPALS PLS Algorithm

The PLS approximations $b_k = V_k c_k$, $V_k = (v_1, \ldots, v_k)$, are determined by $r_k = y - Xb_k \perp U_k$, i.e.,

$$U_k^T (y - XV_k c_k) = 0.$$

Here $U_k^T X V_k = P_k^T V_k = B_k$ is upper bidiagonal

$$B_k = \begin{pmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \rho_{k-1} & \theta_k \\ & & & & \rho_k \end{pmatrix} \in \mathbf{R}^{k \times k}.$$

Hence, $B_k c_k = z_k$. This sytem can be solved in $2k$ flops by backsubstitition. (The inverse $B_k^{-1}$ is full upper triangular.)

# The NIPALS PLS Algorithm

Alternatively a recursion can be used as in LSQR. Write

$$b_k = V_k B_k^{-1} z_k = D_k z_k, \quad D_k = (d_1, d_2, \ldots, d_k) = V_k B_k^{-1},$$

The columns of $D_k$ satisfy $D_k B_k = W_k$ or $B_k^T D_k^T = W_k^T$ with $B_k^T$ lower bidiagonal. This gives $\theta_k d_{k-1} + \rho_k d_k = w_k$. We obtain the forward recursions $d_0 = 0$, $b_0 = 0$,

$$\begin{aligned} d_k &= (w_k - \theta_k d_{k-1})/\rho_k, \\ b_k &= b_{k-1} + \zeta_k d_k, \quad k = 1, 2, \ldots. \end{aligned}$$

In MATLAB the solutions are often computes using the expression `cumsum(W*(inv(P'*W)*diag(q)),2)`. (In general \ should be used instead of `inv`.)

# The NIPALS PLS Algorithm

Many NIPALS codes do not deflate $y$. This is recommended by Manne (1987), Wold et al. (2001), used in the survey of Andersson (2009), etc. However, with this practice the loss of orthogonality in $u_k$ and $v_k$ becomes more harmful.

Without deflation, the full system $(P_k^T W_k)c_k = z_k$ need to be formed and solved, at a cost of $O(nk^2 + k^3)$ flops.

With deflation of $y$, $P_k^T W_k$ is more accurately upper bidiagonal. This allows all solutions to be computed in $O(k^2)$ flops.

# The NIPALS PLS Algorithm

## Theorem

*The matrices $U_k$ and $V_k$, generated by NIPALS are the unique orthonormal bases for the Krylov subspaces*

$$\mathcal{R}(V_k) = \mathcal{K}_k(X^T X, X^T b), \quad \mathcal{R}(U_k) = \mathcal{K}_k(XX^T, XX^T b). \tag{2}$$

*$k = 1, 2, \ldots$. The following relationships hold:*

$$XV_k = U_k B_k, \qquad X^T U_k = V_k B_k^T + \theta_{k+1} v_{k+1} e_k^T, \tag{3}$$

*where $B_k$ is upper bidiagonal.*

# Outline

PLS and Krylov Subspaces

Orthogonalization in Finite Precision

The NIPALS PLS Algorithm

Golub–Kahan Bidiagonalization

# Golub–Kahan Bidiagonalization

Two alternative algorithms to compute the same bidiagonal factorization were given by Golub and Kahan in 1965.

I. Apply sequences of Householder reflections

$$H = I - 2ww^T, \quad \|w\|_2 = 1,$$

to the left and right of $X$ to construct $U_k$, $V_k$, and $B$.

$H$ is by definition symmetric and orthogonal ($H = H^{-1} = H^T$) and is uniquely represented by the vector $w$. $H$ need never be explicitly formed, since products are formed according to

$$Hx = x - 2(w^T x)w.$$

Zeros can be introduced in a column (row) of a matrix by multiplying $X$ from right or left with a suitable matrix $H$. This is used for obtaining the bidiagonal form of a matrix as a first step towards computing the SVD.

Algorithms based on Householder matrices are known to have ideal stability properties (Wilkinson (1965)).

II. The second algorithm is derived from the relationships

$$\begin{aligned} X(v_1, \ldots, v_k) &= (u_1, \ldots, u_k)B_k, \\ X^T(u_1, \ldots, u_k) &= (v_1, \ldots, v_k)B_k^T + \theta_{k+1} v_{k+1} e_k^T. \end{aligned}$$

# Golub–Kahan Bidiagonalization

Recursion formulas for constructing $U_k$, $V_{k+1}$, and $B_k$ can be obtained by identifying successive columns in these equations. With

$$B_k = \begin{pmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \rho_{k-1} & \theta_k \\ & & & & \rho_k \end{pmatrix} \in \mathbf{R}^{k \times k}.$$

this gives $Xv_1 = \rho_1 u_1$, and

$$\begin{aligned} Xv_k &= \theta_k u_{k-1} + \rho_k u_k, \\ X^T u_k &= \rho_k v_k + \theta_{k+1} v_{k+1}, \quad k > 1 \end{aligned}$$

# Golub–Kahan Bidiagonalization

Solving for $v_{k+1}$ and $u_k$, we obtain the Bidiag2 algorithm:
$\theta_1 v_1 = X^T y$, $\rho_1 u_1 = X v_1$,

$$\begin{aligned} \theta_i v_i &= X^T u_{i-1} - \rho_{i-1} v_{i-1}, \\ \rho_i u_i &= X v_i - \theta_i u_{i-1}, \quad i = 2, \ldots, k \ldots. \end{aligned}$$

These equations uniquely determine $u_1, v_2, u_2, \ldots$ and the scalars $\theta_1, \rho_1, \theta_2, \rho_2, \ldots,$ in the PLS factorization.

Note that only matrix–vector multiplications of the form $Xv$ and $X^T u$, are used, i.e., no deflation of $X$. This is an advantage in particular if matrix-vector products are cheap.

# Golub–Kahan Bidiagonalization

In LSQR a similar recursion called Bidiag1 is used. This starts with $u_1 = y$ and generates the same $v$-vectors but different $u$-vectors and a lower bidiagonal matrix $B$.

Bidiag1 and Bidiag2 are special cases of Lanczos process (1952) applied to the symmetric Jordan–Wielandt matrix

$$\begin{pmatrix} A & 0 \\ 0 & A^T \end{pmatrix}.$$

Lanczos methods were developed by Lanczos (1952) and conjugate gradient methods by Hestenes and Stiefel (1952).

# Golub–Kahan Bidiagonalization

Lanczos- and CG-methods behave very differently in inexact arithmetic. Eventually there is a complete loss of orthogonality in the $u$ and $v$ vectors. The finite termination property is lost.

It took until the 1970s before the behavior of Lanczos and CG methods was sufficiently well understood for the methods to become widely used.

Lanczos methods are designed as iterative methods for solving linear systems and least squares problems. Accuracy of final solution not affected by loss of orthogonality. Only a small number of vectors can be saved.

# Golub–Kahan Bidiagonalization

In PLS a small number $\ll \min\{m, n\}$ of factors are desired. Since $u$- and $v$-vectors are saved, full reorthogonalization can be used. (Cost $O((m+n)k^2)$ flops).

Manne (1987), Wu and Manne (2000) and Eldén (2004) have suggested using Bidiag2 for PLS.

In tests by Andersson (2008) Bidiag2 was fast but gave inaccurate results. Similar results reported by Barry at Eigenvector (2013). No more systematic study of these methods seems to have been published.

# Golub–Kahan Bidiagonalization

Open questions:

- The method LSCG by Paige (1974) based on Bidiag2 is known to be numerically inferior to LSQR. Is Bidiag2 with reorthogonalization stable? Should an implementation use Bidiag1 as LSQR?
- Should one-sided reorthogonalization (either $u$-vectors or $v$-vectors) be an option? Big saving in case $m \gg n$ or $n \gg m$ (Simon and Zha (2000)).
- Indahl (2013) has given a hybrid algorithm that is free of deflation and may use reorthogonalization. How does this compare to Bidiag2 and Bidiag1?

Final Comments:

Theoretically valid modifications of PLS methods may result in algorithms with (very) different stability when used in actual computing.

Essential that all new algorithms are tested numerically, preferably on a shared set of test problems in the public domain.

Knowledge of numerical linear algebra lacking in many application areas.