Received: 11 August 2016 | Revised: 19 January 2017 | Accepted: 17 March 2017

DOI: 10.1002/cem.2898

RESEARCH ARTICLE

WILEY Journal of CHEMOMETRICS

Fast and stable partial least squares modelling: A benchmark study with theoretical comments

Åke Björck[1] | Ulf G. Indahl[2]

[1]Department of Mathematics, Linköping University, Linköping, S-581 83, Sweden

[2]Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, N-1432, Norway

Correspondence
Ulf G. Indahl, Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, N-1432 Ås, Norway.
Email: ulf.indahl@nmbu.no

Funding information
Research Council of Norway, Grant/Award Number: 239070

Algorithms for partial least squares (PLS) modelling are placed into a sound theoretical context focusing on numerical precision and computational efficiency. NIPALS and other PLS algorithms that perform deflation steps of the predictors (**X**) may be slow or even computationally infeasible for sparse and/or large-scale data sets. As alternatives, we develop new versions of the Bidiag1 and Bidiag2 algorithms. These include full reorthogonalization of both score and loading vectors, which we consider to be both necessary and sufficient for numerical precision. Using a collection of benchmark data sets, these 2 new algorithms are compared to the NIPALS PLS and 4 other PLS algorithms acknowledged in the chemometrics literature. The provably stable Householder algorithm for PLS regression is taken as the reference method for numerical precision. Our conclusion is that our new Bidiag1 and Bidiag2 algorithms are the methods of choice for problems where both efficiency and numerical precision are important. When efficiency is not urgent, the NIPALS PLS and the Householder PLS are also good choices. The benchmark study shows that SIMPLS gives poor numerical precision even for a small number of factors. Further, the nonorthogonal scores PLS, direct scores PLS, and the improved kernel PLS are demonstrated to be numerically less stable than the best algorithms. Prototype MATLAB codes are included for the 5 PLS algorithms concluded to be numerically stable on our benchmark data sets. Other aspects of PLS modelling, such as the evaluation of the regression coefficients, are also analyzed using techniques from numerical linear algebra.

KEYWORDS
Bidiag1, Bidiag2, bidiagonalization, deflation, Golub-Kahan Householder, Lanczos process, NIPALS, PLS, regression, reorthogonalization, stability

1 | INTRODUCTION

The first rigorous description of a partial least squares (PLS) algorithm was the NIPALS algorithm given in 1984 by Wold et al.[1] Since then, a great number of different, but analytically equivalent, PLS algorithms have been suggested in the literature. Even minor analytically equivalent modifications of a PLS algorithm may give (very) different computed results for the regression coefficients and fitted values. It is, therefore, important that all new PLS algorithms are tested, preferably on a shared set of benchmark problems in the public domain. In 2009, Andersson[2] presented a comparison of 9 different PLS algorithm with respect to speed and numerical precision. The aim of the present paper is to follow up on Andersson's pioneering initiative, but our evaluation differs in several aspects. We take the PLS algorithm based on Householder transformations (HHPLS) as our reference algorithm. The HHPLS is computationally slow but, more importantly, has been proven by Björck[3] to be numerically stable in the mixed forward-backward sense.

For sparse and/or large-scale data sets, the deflation steps of the predictors ($\mathbf{X}$) as implemented in the NIPALS PLS is slow and may even be computationally unfeasible. Wold et al[1] noted that in such situations the LSQR algorithm by Paige and Saunders[4] might be an attractive alternative. LSQR builds on the Bidiag1 algorithm, a Lanczos-type (see Lanczos[5]) bidiagonalization algorithm proposed in the seminal paper by Golub and Kahan.[6] Manne[7] suggested that instead using an adaptation of the analytically equivalent Bidiag2 algorithm (also proposed in Golub and Kahan[6]). Eldén[8] also focused on the equivalence between NIPALS PLS and Bidiag2 for the purpose of exploring the shrinkage properties of PLS in comparison to principal component regression. Wu and Manne[9] commented on the numerical instability problems associated with the Bidiag2 version of PLS but refrained from including stabilizing reorthogonalization steps. In the survey by Andersson,[2] where Bidiag2 was tested without reorthogonalization, it was demonstrated to be fast but gave considerably poorer numerical precision than the other algorithms.

The tested algorithms analyzed and tested in the present paper include the 2 most widely used choices for PLS modelling—the NIPALS PLS algorithm by Wold et al[1] and the SIMPLS algorithm by de Jong[10] as well as 2 new algorithms based on the Bidiag2 and Bidiag1. These perform full reorthogonalization, which we consider essential for any successful large scale application of PLS. Code for a stabilized version of Bidiag2 with reorthogonalization of both the scores ($\mathbf{T}$) and the weights ($\mathbf{W}$) was also presented by Indahl.[15, appendix A.5]

2 | MATHEMATICAL PRELIMINARIES

Consider a linear model $\mathbf{Xb} = \mathbf{y}$, where $\mathbf{X} \in \mathbf{R}^{n\times p}$ and $\mathbf{y} \in \mathbf{R}^n$ and the related least squares problem

$$\min \|\mathbf{b}\|_2 \quad \text{subject to} \quad \|\mathbf{Xb} - \mathbf{y}\|_2 = \min . \tag{1}$$

Independent of the size and rank of $\mathbf{X}$, this problem always has a unique solution $\mathbf{b}^\dagger$ called the pseudoinverse solution characterized by the two conditions

$$\mathbf{X}'\mathbf{Xb} = \mathbf{X}'\mathbf{y}, \qquad \mathbf{b} \in \text{span}(\mathbf{X}') \subseteq \mathbf{R}^p, \tag{2}$$

ie, $\mathbf{b}^\dagger$ is the solution of the associated normal equations contained in the row subspace of $\mathbf{X}$.

The PLS approximations $\mathbf{b}_k$, $k = 1, 2, \ldots$ to the least squares problem in Equation 1 can be defined as the estimates generated by k steps of the NIPALS PLS algorithm. We prefer a definition of the approximate k-component PLS solutions to be independent of a particular algorithm and define the PLS approximation $\mathbf{b}_k$ as the solution of the subproblem

$$\min_{\mathbf{b}} \|\mathbf{Xb} - \mathbf{y}\|_2, \quad \text{subject to} \quad \mathbf{b} \in \mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y}). \tag{3}$$

Here, $\mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y}) \subseteq \mathbf{R}^p$ denotes the so-called Krylov subspace* spanned by the first k Krylov vectors

$$\mathbf{X}'\mathbf{y}, (\mathbf{X}'\mathbf{X})\mathbf{X}'\mathbf{y}, \ \ldots\ , (\mathbf{X}'\mathbf{X})^{k-1}\mathbf{X}'\mathbf{y}, \tag{4}$$

The corresponding residual vector $\mathbf{r} = \mathbf{y} - \mathbf{Xb}_k \in \mathcal{K}_{k+1}(\mathbf{XX}', \mathbf{y}) \subseteq \mathbf{R}^n$ is contained in the subspace spanned by the Krylov vectors

$$\mathbf{y}, (\mathbf{XX}')\mathbf{y}, (\mathbf{XX}')^2\mathbf{y}, \ \ldots\ , (\mathbf{XX}')^k\mathbf{y}. \tag{5}$$

An infinite sequence of Krylov vectors is called a Krylov sequence. A Krylov sequence always has a first vector that can be expressed as a linear combination of the preceding ones. Hence, for some index $K \geqslant 1$, it holds that the subspaces $\mathcal{K}_{K+1}(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y}) = \mathcal{K}_K(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y})$. The latter means that K is both the maximum rank for the particular Krylov subspaces and the maximum number of possible PLS components for the particular $(\mathbf{X}, \mathbf{y})$ data set. It can be shown (see Björck[3]) that with this maximum number (K) of PLS components, the pseudoinverse solution and the PLS solution coincide, ie, $\mathbf{b}_K = \mathbf{b}^\dagger$.

For $1 \leqslant k < K$ PLS components, the subproblems 3 have full rank, and hence, the corresponding PLS solution $\mathbf{b}_k$ is uniquely defined. Note that, although the pseudoinverse solution $\mathbf{b}^\dagger$ is a linear mapping of $\mathbf{y}$, the intermediate PLS approximations $\mathbf{b}_k$ depend nonlinearly on $\mathbf{y}$ in a nontrivial way, as explained in Eldén.[8, section 4]

In exact arithmetic, there are two situations where PLS terminates in less than rank($\mathbf{X}$) steps. The first is when $\mathbf{X}$ has 1 or more multiple singular values; the second when $\mathbf{y}$ is orthogonal to some left singular vectors of $\mathbf{X}$. The following lemma is proved in Björck[3]:

Lemma 1. *Let $X \in \mathbf{R}^{n\times p}$ have s distinct (possibly multiple) nonzero singular values $\sigma_1 > \sigma_2 > \ldots > \sigma_s$. Denote by c_i, the norm of the orthogonal projection of $\mathbf{y}$ onto the left singular subspace corresponding to σ_i. Then, in exact arithmetic, PLS terminates with $\mathbf{b}_r$, where $r \leqslant s$ is the number of nonzero coefficients c_i.*

Special cases of Lemma 1 occur when $\mathbf{X}$ equals the identity matrix or is a design matrix from a factorial experimental design. In these situations, all the singular values are identical, ie, $s = 1$ and PLS stops already after 1 step. It should be noted that applications of PLS in the analysis of experimental designs, and the fact that PLS extracts the ordinary least squares solution in its first step, has been an issue in chemometrics since Martens and Næs[11] published their well-known

*Named after the Russian mathematician Aleksei Nikolaevich Krylov (1863-1945), Maritime Academy of St. Petersburg, who pioneered the use of such subspaces in scientific computing.

book on multivariate calibration. In other types of applications, PLS is usually stopped well before the (theoretical) maximum number of terms have been computed.

Lanczos and conjugate gradient methods are closely related and are frequently referred to as Krylov subspace methods. These methods were introduced in the early 1950s by Lanczos[5] and Hestenes and Stiefel[12] for the purpose of solving large systems of linear equations and eigenvalue problems. Today, Krylov subspace methods play a dominant role in nearly all scientific computations; see Golub and O'Leary[13] for a survey on the fundamental developments of the topic.

Orthogonal bases for the Krylov subspaces $\mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y})$ and $\mathcal{K}_k(\mathbf{X}\mathbf{X}', \mathbf{X}\mathbf{X}'\mathbf{y})$ play a central role in PLS algorithms. They are uniquely defined until the PLS algorithm terminates. In theory, they can be computed by applying Gram-Schmidt orthogonalization to the sequence of Krylov vectors

$$\begin{aligned} &\mathbf{X}'\mathbf{y}, (\mathbf{X}'\mathbf{X})\mathbf{X}'\mathbf{y}, \ \dots\ , (\mathbf{X}'\mathbf{X})^{k-1}\mathbf{X}'\mathbf{y}, \\ &\mathbf{X}\mathbf{X}'\mathbf{y}, (\mathbf{X}\mathbf{X}')^2\mathbf{y}, \ \dots\ , (\mathbf{X}\mathbf{X}')^k\mathbf{y}. \end{aligned} \tag{6}$$

It is well known that these sequences of Krylov vectors converge to the left and right singular vectors of the largest singular value of $\mathbf{X}$. Hence, they rapidly become nearly linearly dependent. Even when using double numerical precision computation with relative numerical precision 10^{-16}, there may be a complete loss of orthogonality after only few steps. This is because the loss of orthogonality grows exponentially with the number of steps. This will cause a corresponding loss of orthogonality as in the Gram-Schmidt process; see eg, Björck[14] section 2.3.5. Indeed, loss of orthogonality in the computed basis vectors is the most common cause of low-numerical precision for several proposed PLS algorithms.

Krylov subspaces have a useful and important invariance property with respect to a change of basis. Let $\mathbf{T} \in \mathbf{R}^{n\times n}$ and $\mathbf{W} \in \mathbf{R}^{p\times p}$ be any pair of square orthogonal matrices representing orthogonal bases for $\mathbf{R}^n$ and $\mathbf{R}^p$, respectively. Starting with the original system $\mathbf{X}\mathbf{b} = \mathbf{y}$ and the indicated subsequent orthogonal transformations based on $\mathbf{T}$ and $\mathbf{W}$, we have the following equivalences

$$\mathbf{X}\mathbf{b} = \mathbf{y} \Longleftrightarrow \mathbf{T}'\mathbf{X}\mathbf{b} = \mathbf{T}'\mathbf{y} \Longleftrightarrow \mathbf{T}'\mathbf{X}\mathbf{W}\mathbf{W}'\mathbf{b} = \mathbf{T}'\mathbf{y} \Longleftrightarrow \mathbf{B}\mathbf{z} = \mathbf{q}, \tag{7}$$

where we have defined $\mathbf{B} = \mathbf{T}'\mathbf{X}\mathbf{W}$, $\mathbf{z} = \mathbf{W}'\mathbf{b}$, and $\mathbf{q} = \mathbf{T}'\mathbf{y}$. The Krylov subspaces for the leftmost and rightmost systems in Equation 7 are related by

$$\mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y}) = \mathbf{W}\mathcal{K}_k(\mathbf{B}'\mathbf{B}, \mathbf{B}'\mathbf{q}).$$

The PLS approximations for the original and transformed systems are related according to $\mathbf{b}_k = \mathbf{W}\mathbf{z}_k$, where $\mathbf{z}_k$ is the solution of the subproblem

$$\min_{\mathbf{z}} \|\mathbf{B}\mathbf{z} - \mathbf{q}\|_2, \quad \text{subject to} \quad \mathbf{z} \in \mathcal{K}_k(\mathbf{B}'\mathbf{B}, \mathbf{B}'\mathbf{q}). \tag{8}$$

If $\mathbf{T}$ and $\mathbf{W}$ are chosen as the matrices of left and right singular vectors, respectively, then $\mathbf{B} = \Sigma$ is diagonal. The transformed system in this case becomes $\Sigma\mathbf{z} = \mathbf{q}$, where

$$\mathbf{q} = \mathbf{T}'\mathbf{y}, \quad \Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \Sigma_1 = \operatorname{diag}(\sigma_1, \ \dots\ , \sigma_r) > 0,$$

and $r = \operatorname{rank}(\mathbf{X}) \leqslant \min\{p, n\}$. Hence, if the singular value decomposition (SVD) of $\mathbf{X}$ is known, then explicit expressions for the PLS approximations, $\mathbf{b}_k$ can be obtained. However, even computing a low-rank PLS approximation with this approach requires the full SVD of $\mathbf{X}$. Hence, this relationship is most interesting for the theoretical aspects of the PLS approximations. For real applications, it is more attractive to require $\mathbf{B}$ to be bidiagonal, since the corresponding orthogonal basis matrices can be obtained by computationally fast forward recursions.

3 | PLS ALGORITHMS

3.1 | The NIPALS PLS

The NIPALS PLS algorithm by Wold et al[1] takes $\mathbf{X}_0 = \mathbf{X}$, $\mathbf{y}_0 = \mathbf{y}$, and for $k = 1, 2, \dots$, generates

$$\mu_k\mathbf{w}_k = \mathbf{X}'_{k-1}\mathbf{y}_{k-1}, \qquad \rho_k\mathbf{t}_k = \mathbf{X}_{k-1}\mathbf{w}_k, \tag{9}$$

$$(\mathbf{X}_k, \mathbf{y}_k) = (\mathbf{I} - \mathbf{t}_k\mathbf{t}'_k)(\mathbf{X}_{k-1}, \mathbf{y}_{k-1}), \tag{10}$$

where μ_k and ρ_k are normalizing constants. (The original NIPALS algorithm differs slightly from this description in that only the weights $\mathbf{w}_k$ are normalized. To be consistent with the algorithms considered below we also normalize the scores $\mathbf{t}_k$. This will not affect the numerical precision of the algorithm and the extra computational overhead is negligible.) In Equation 10, $\mathbf{X}_{k-1}$ and $\mathbf{y}_{k-1}$ are deflated by subtracting their orthogonal projections onto $\mathbf{t}_k$. This can also be written as

$$(\mathbf{X}_k, \mathbf{y}_k) = (\mathbf{X}_{k-1}, \mathbf{y}_{k-1}) - \mathbf{t}_k(\mathbf{p}'_k, \eta_k), \tag{11}$$

$$\mathbf{p}'_k = \mathbf{t}'_k\mathbf{X}_{k-1}, \quad \eta_k = \mathbf{t}'_k\mathbf{y}_{k-1}. \tag{12}$$

We note that if $\mathbf{t}'_k\mathbf{X}_{k-1}\mathbf{w}_k \neq 0$, then the rank of $\mathbf{X}_k$ is exactly 1 less than that of $\mathbf{X}_{k-1}$. Summing Equations 11 and 12 and setting $\mathbf{T}_k = (\mathbf{t}_1, \dots, \mathbf{t}_k)$, $\mathbf{P}_k = (\mathbf{p}_1, \dots, \mathbf{p}_k)$, and $\mathbf{q}_k = (\eta_1, \dots, \eta_k)'$ give

$$\mathbf{X} = \mathbf{T}_k\mathbf{P}'_k + \mathbf{X}_k, \qquad \mathbf{y} = \mathbf{T}_k\mathbf{q}_k + \mathbf{y}_k. \tag{13}$$

These relations hold to working numerical precision and do not rely on orthogonality. The matrix $\mathbf{T}_k\mathbf{P}'_k$ is a rank k approximation to the data matrix $\mathbf{X}$. The regression coefficients are $\mathbf{b}_k = \mathbf{W}_k\mathbf{z}_k$, where $\mathbf{z}_k$ is obtained by solving the linear system

$$\left(\mathbf{P}_k'\mathbf{W}_k\right)\mathbf{z}_k = \mathbf{q}_k. \tag{14}$$

Orthogonal bases for the Krylov subspaces $\mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y})$ and $\mathcal{K}_k(\mathbf{X}\mathbf{X}', \mathbf{X}\mathbf{X}'\mathbf{y})$ are given by the matrices

$$\mathbf{T}_k = (\mathbf{t}_1, \ldots, \mathbf{t}_k) \text{ and } \mathbf{W}_k = (\mathbf{w}_1, \ldots, \mathbf{w}_k), \tag{15}$$

generated by Equations 9 to 10; see Eldén.[8, proposition 3.1]

3.2 | The Householder bidiagonalization PLS

Golub and Kahan[6] gave a numerically stable algorithm for the bidiagonalization of a matrix $\mathbf{X}$ using products of Householder reflections (see Björk[14, section 2.3.1]) from left and right. After k steps, the first k rows and columns are transformed to upper bidiagonal form

$$\mathbf{B}_k = \begin{pmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \rho_{k-1} & \theta_k \\ & & & & \rho_k \end{pmatrix}. \tag{16}$$

The first k left and right Householder transformations implicitly define bases matrices $\mathbf{T}_k$ and $\mathbf{W}_k$ for the Krylov subspaces. These matrices need not be formed explicitly and hence are orthogonal by definition.

As in the NIPALS, the HHPLS algorithm performs explicit modifications of the $(\mathbf{X}, \mathbf{y})$ data in the intermediate calculations. Because the HHPLS is numerically stable in a strong sense (Björck[3]), it is chosen as our reference algorithm. Note that the Householder reduction to bidiagonal form is also the first step in standard algorithms for computing the SVD. Therefore, the HHPLS gives at least as good precision for PLS, although SVD may provide more complete information about the data matrix $\mathbf{X}$.

3.3 | The Golub-Kahan Lanczos bidiagonalization for PLS

Golub and Kahan[6] also proposed an algorithm using a Lanczos-type recursive process for the bidiagonalization of a given matrix, in which only matrix-vector products with the original data $\mathbf{X}$ and $\mathbf{X}'$ are used. This is highly advantageous when $\mathbf{X}$ is sparse or otherwise structured, because such matrix-vector products may then be performed very efficiently.

There are 2 alternative procedures for adaption of the Golub-Kahan Lanczos bidiagonalization to PLS:

The *Bidiag2*–procedure starts by computing

$$\theta_1\mathbf{w}_1 = \mathbf{X}'\mathbf{y}, \quad \rho_1\mathbf{t}_1 = \mathbf{X}\mathbf{w}_1, \tag{17}$$

and then for $i = 1, 2, \ldots$, it computes

$$\theta_{i+1}\mathbf{w}_{i+1} = \mathbf{X}'\mathbf{t}_i - \rho_i\mathbf{w}_i, \quad \rho_{i+1}\mathbf{t}_{i+1} = \mathbf{X}\mathbf{w}_{i+1} - \theta_{i+1}\mathbf{t}_i. \tag{18}$$

The scalars ρ_i and θ_i are normalizing constants and give the elements in the ith row of the upper bidiagonal matrix $\mathbf{B}_k$. The unit vectors $\mathbf{w}_i$ and $\mathbf{t}_i$ are the desired orthogonal basis vectors for the corresponding Krylov subspaces $\mathcal{K}_k(\mathbf{X}'\mathbf{X}, \mathbf{X}'\mathbf{y})$ and $\mathcal{K}_k(\mathbf{X}\mathbf{X}', \mathbf{X}\mathbf{X}'\mathbf{y})$, respectively. We may rewrite the above recursion (Equations 17-18) as

$$\begin{aligned} \mathbf{W}_k(\theta_1\mathbf{e}_1) = \mathbf{X}'\mathbf{y}, \quad \mathbf{X}\mathbf{W}_k = \mathbf{T}_k\mathbf{B}_k, \\ \mathbf{X}'\mathbf{T}_k = \mathbf{W}_k\mathbf{B}_k' + \theta_{k+1}\mathbf{w}_{k+1}\mathbf{e}_k', \end{aligned} \tag{19}$$

where $\mathbf{e}_1$ and $\mathbf{e}_k$ denotes the first and kth standard basis vectors in $\mathbf{R}^k$, respectively. As noted in Indahl,[15, eq. 1] the PLS $\mathbf{X}$-loadings are $\mathbf{P}_k = \mathbf{X}'\mathbf{T}_k$. Hence, by left multiplication with the orthogonal scores matrix $\mathbf{T}_k'$ in the middle equation of Equation 19, it follows that the bidiagonal matrix can be expressed as

$$\mathbf{B}_k = \mathbf{P}_k'\mathbf{W}_k. \tag{20}$$

The *Bidiag1*–procedure starts with

$$\gamma_1\mathbf{u}_1 = \mathbf{y}, \quad \alpha_1\mathbf{w}_1 = \mathbf{X}'\mathbf{u}_1, \tag{21}$$

and then for $i = 1, 2, \ldots$, computes

$$\gamma_{i+1}\mathbf{u}_{i+1} = \mathbf{X}\mathbf{w}_i - \alpha_i\mathbf{u}_i, \quad \alpha_{i+1}\mathbf{w}_{i+1} = \mathbf{X}'\mathbf{u}_{i+1} - \gamma_{i+1}\mathbf{w}_i. \tag{22}$$

For $i = 1, \ldots, k$, the normalizing constants α_i and γ_{i+1} are elements in the ith column of the lower bidiagonal matrix

$$\mathbf{C}_k = \begin{pmatrix} \alpha_1 & & & & \\ \gamma_2 & \alpha_2 & & & \\ & \gamma_3 & \ddots & & \\ & & \ddots & \alpha_{k-1} & \\ & & & \gamma_k & \alpha_k \\ & & & & \gamma_{k+1} \end{pmatrix}. \tag{23}$$

The recursion (Equations 21-22) may be rewritten in matrix form as

$$\begin{aligned} \mathbf{U}_{k+1}(\gamma_1\mathbf{e}_1) = \mathbf{y}, \quad \mathbf{X}\mathbf{W}_k = \mathbf{U}_{k+1}\mathbf{C}_k, \\ \mathbf{X}'\mathbf{U}_{k+1} = \mathbf{W}_k\mathbf{C}_k' + \alpha_{k+1}\mathbf{w}_{k+1}\mathbf{e}_k'. \end{aligned} \tag{24}$$

The vectors $\mathbf{w}_i$ are the same as in Bidiag2, while $\mathbf{u}_i$ are orthogonal basis vectors for the Krylov subspace $\mathcal{K}_k(\mathbf{X}\mathbf{X}', \mathbf{y})$. Bidiag2 corresponds to the procedure originally given by Golub and Kahan,[6] but either procedure may be derived from the other by interchanging $\mathbf{X}$ and $\mathbf{X}'$ and choosing the appropriate starting vector; see Paige and Saunders.[4]

The upper bidiagonal matrix $\mathbf{B}_k$ and the vectors $\mathbf{t}_i$ in Bidiag2 can be obtained from $\mathbf{C}_k$ and the $\mathbf{u}_i$ using a sequence of Givens rotations (see Björk[14, section 2.3.1]). This process is

illustrated below for $k = 2$. We first rotate the first 2 rows in $\mathbf{C}_k$ to zero out the element γ_2. To preserve the product, we apply the same rotation to the corresponding columns in $\mathbf{U}_k$. After these operations, we have obtained $\mathbf{t}_1$, ρ_1, and θ_1. Next, we rotate rows 2 and 3 in the (transformed) matrix $\mathbf{C}_k$ to zero out γ_3 and again apply the same rotations to columns 2 and 3 in $\mathbf{U}_k$. This will produce $\mathbf{t}_2$, ρ_2, and θ_2.

$$(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)\begin{pmatrix} \alpha_1 & & \\ \gamma_2 & \alpha_2 & \\ & \gamma_3 & \end{pmatrix} \Rightarrow (\mathbf{t}_1, \tilde{\mathbf{u}}_2, \mathbf{u}_3)\begin{pmatrix} \rho_1 & \theta_2 & \\ 0 & \tilde{\alpha}_2 & \\ & \gamma_3 & \end{pmatrix}$$
$$\Rightarrow (\mathbf{t}_1, \mathbf{t}_2, \tilde{\mathbf{u}}_3)\begin{pmatrix} \rho_1 & \theta_2 & \\ 0 & \rho_2 & \\ & 0 & \end{pmatrix}.$$

Clearly, the given rotations can be interleaved with the recursions in Bidiag1. Because $\mathbf{y} = \gamma_1\mathbf{u}_1$, the generation of the right-hand side vector $\mathbf{X}\mathbf{q}_k$ in the linear system 14 also differs from that in Bidiag2. A more complete description of the above process is given in the LSQR paper of Paige and Saunders.[4] However, in the LSQR algorithm, the basis vectors $\mathbf{T}_k$ are not computed because they are not used.

In Bidiag1 and Bidiag2, the analytically orthogonal basis vectors are directly generated from the 2-term recursions and the Krylov vectors (Equation 6) are never formed. Still, there will be a gradual loss of numerical orthogonality in the basis vectors. This loss is closely related to the convergence of singular values of the bidiagonal matrices to the singular values of $\mathbf{X}$. A satisfactory analysis of this behaviour is challenging, and was first given by Paige[16] nearly 20 years after the publication of the Lanczos process in Lanczos.[5]

The original PLS paper[1] recommended using LSQR in situations where the deflation of the predictors $\mathbf{X}$ in NIPALS is expensive. However, the LSQR algorithm is not directly suitable as a PLS substitute for several reasons:

- LSQR is designed to be used for a different purpose, namely, for the efficient solution of large-scale least squares problems $\min \|\mathbf{X}\mathbf{b} - \mathbf{y}\|_2$.
- LSQR does not save the vectors $\mathbf{w}_i$ and $\mathbf{u}_i$. The iterations are continued even after the orthogonality of the bases vectors have been completely lost. For ill-conditioned problems, LSQR can take many more than rank($\mathbf{X}$) iterations to converge.
- Several features included in LSQR, such as stopping criteria and condition estimation are of less interest in the typical PLS modelling context, where comparatively few factors are used.

Because of the inference purposes of PLS, the bases vectors $\mathbf{t}_k$ and $\mathbf{w}_k$ need to be saved and their orthogonality preserved. This can be achieved by reorthogonalizing $\mathbf{t}_k$ and $\mathbf{w}_k$ against all previous basis vectors $\mathbf{t}_1, \ldots, \mathbf{t}_{k-1}$ and $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$ as soon as they have been computed. If we define $\mathbf{T}_{k-1} = [\mathbf{t}_1\mathbf{t}_2 \ldots \mathbf{t}_{k-1}]$ and $\mathbf{W}_{k-1} = [\mathbf{w}_1\mathbf{w}_2 \ldots \mathbf{w}_{k-1}]$, the reorthogonalizations of $\mathbf{t}_k$ and $\mathbf{w}_k$ can be expressed as $\mathbf{t}_k = \mathbf{t}_k - \mathbf{T}_{k-1}(\mathbf{T}_{k-1}'\mathbf{t}_k)$ and $\mathbf{w}_k = \mathbf{w}_k - \mathbf{W}_{k-1}(\mathbf{W}_{k-1}'\mathbf{w}_k)$, respectively, followed by a normalization step. This adds an arithmetic cost of about $4(n + p)k^2$ flops for k factors and is usually affordable in typical applications of PLS (but not for the problems traditionally solved by LSQR).

3.4 | A note on matrix residuals

Although the NIPALS and the proposed alternative algorithms compute the same approximate solutions $\mathbf{b}_k$, they differ in the way that the residual of the data matrix is approximated. With the $\mathbf{T}_k$ matrix taken to be orthogonal, the data residuals $\mathbf{X}_k$ in NIPALS are given by

$$\mathbf{X}_k = \mathbf{X} - \mathbf{T}_k\mathbf{P}_k' = (\mathbf{I} - \mathbf{T}_k\mathbf{T}_k')\mathbf{X}, \tag{25}$$

ie, a projection in the column space of $\mathbf{X}$. For the Householder and Bidiag versions the residual is obtained from the rank-k approximations $\mathbf{X} \approx \mathbf{T}_k\mathbf{B}_k\mathbf{W}_k'$. Using $\mathbf{T}_k\mathbf{B}_k = \mathbf{X}\mathbf{W}_k$, we obtain the data residual

$$\mathbf{E}_k = \mathbf{X} - (\mathbf{T}_k\mathbf{B}_k)\mathbf{W}_k' = \mathbf{X} - (\mathbf{X}\mathbf{W}_k)\mathbf{W}_k' = \mathbf{X}(\mathbf{I} - \mathbf{W}_k\mathbf{W}_k'), \tag{26}$$

ie, a projection in the row space of $\mathbf{X}$. See Indahl[15] note 8 for a more detailed discussion of the two $\mathbf{X}$-approximation alternatives $\mathbf{T}_k\mathbf{P}_k'$ and $(\mathbf{T}_k\mathbf{B}_k)\mathbf{W}_k'$.

It is a common assumption that the PLS data residuals $\mathbf{X}_k$ and $\mathbf{E}_k$ will always become small when $k \to$ rank($\mathbf{X}$). The original "Algorithm PLS" in Wold et al[1] says that the iterations are to be continued until $\|(\mathbf{X}_k, \mathbf{y}_k)\|$ is small, where "small" in the numerical sense can be decided upon by comparing the ratios $\|(\mathbf{X}_k, \mathbf{y}_k)\|/\|(\mathbf{X}, \mathbf{y})\|$ to some prespecified positive tolerance τ ($0 < \tau \ll 1$). However, according to Lemma 1, there are circumstances where PLS instead terminates after $k \ll$ rank($\mathbf{X}$) steps independent of the magnitude of the ratio $\|(\mathbf{X}_k, \mathbf{y}_k)\|/\|(\mathbf{X}, \mathbf{y})\|$.

In most practical applications of PLS, one is primarily interested in good predictive performance with respect to $\mathbf{y}$. Some appropriate model validation-and-selection strategy should, therefore, be preferred over the theoretical termination alternatives just mentioned. Cross validation is a popular alternative often used with PLS.

4 | COMPUTING THE REGRESSION COEFFICIENTS

After k steps of the Bidiag2 algorithm, the regression coefficients are $\mathbf{b}_k = \mathbf{W}_k\mathbf{z}_k$, where $\mathbf{z}_k$ satisfies the upper bidiagonal system

$$\mathbf{B}_k\mathbf{z}_k = \mathbf{q}_k, \quad \mathbf{q}_k = \mathbf{T}_k'\mathbf{y} = (\eta_1, \ldots, \eta_k)'. \tag{27}$$

The bidiagonal system $\mathbf{B}_k\mathbf{z}_k = \mathbf{q}_k$ can be solved with minimum computational effort using back substitution. But $\mathbf{z}_k$ will differ in all entries from $\mathbf{z}_{k-1}$ and computing $\mathbf{b}_k = \mathbf{W}_k\mathbf{z}_k$ has to be done from scratch, which may be computationally expensive when dealing with large modelling problems. In LSQR, a more efficient updating formula is used for $\mathbf{b}_k$ that can be applied also in the Bidiag1 and Bidiag2 algorithms presented here. If we define $\mathbf{D}_k = (\mathbf{d}_1, \ldots, \mathbf{d}_k) = \mathbf{W}_k\mathbf{B}_k^{-1}$, then $\mathbf{W}_k = \mathbf{D}_k\mathbf{B}_k$ and

$$\mathbf{b}_k = \mathbf{W}_k\mathbf{B}_k^{-1}\mathbf{q}_k = \mathbf{D}_k\mathbf{q}_k = \mathbf{b}_{k-1} + \eta_k\mathbf{d}_k. \tag{28}$$

For the first column, we obtain $\mathbf{d}_1 = (1/\rho_1)\mathbf{w}_1$, and the resulting vector of regression coefficients $\mathbf{b}_1 = \eta_1\mathbf{d}_1$. For $k > 1$, the matrix identity $\mathbf{W}_k = \mathbf{D}_k\mathbf{B}_k$ can be expressed as

$$(\mathbf{W}_{k-1}, \quad \mathbf{w}_k) = (\mathbf{D}_{k-1}, \quad \mathbf{d}_k)\begin{pmatrix} \mathbf{B}_{k-1} & \theta_k\mathbf{e}_{k-1} \\ 0 & \rho_k \end{pmatrix}, \tag{29}$$

where $\mathbf{e}_{k-1}$ is the last standard basis vector of $\mathbf{R}^{k-1}$. Equating the first block columns shows that $\mathbf{W}_{k-1} = \mathbf{D}_{k-1}\mathbf{B}_{k-1}$. Hence, the first $k-1$ columns of $\mathbf{D}_k$ equal $\mathbf{D}_{k-1}$. Because $\mathbf{D}_{k-1}\mathbf{e}_{k-1} = \mathbf{d}_{k-1}$ (the last column of $\mathbf{D}_{k-1}$), equating the last columns of $\mathbf{D}_k$ and $\mathbf{W}_k$ and solving for $\mathbf{d}_k$ gives

$$\mathbf{d}_k = (\mathbf{w}_k - \theta_k\mathbf{d}_{k-1})/\rho_k, \quad k > 1. \tag{30}$$

Note that storing the entire matrix $\mathbf{D}_k$ is unnecessary (only the most recent iteration for the column $\mathbf{d}_{i-1}$ needs to be saved to compute the subsequent column $\mathbf{d}_i$).

According to the $\mathbf{b}_k$ updating rule given in Equation 28 and the formula for $\mathbf{d}_k$ in Equation 30, the complete updating formulas for the PLS regression coefficients become

$$\begin{aligned} \mathbf{b}_1 &= \eta_1\mathbf{d}_1 = (\eta_1/\rho_1)\mathbf{w}_1, \\ \mathbf{b}_k &= \mathbf{b}_{k-1} + \eta_k\mathbf{d}_k = \mathbf{b}_{k-1} \\ &\quad + (\eta_k/\rho_k)(\mathbf{w}_k - \theta_k\mathbf{d}_{k-1}), \quad k > 1. \end{aligned} \tag{31}$$

When coding in MATLAB it is possible to perform the vector updates (Equation 31) by using the expression

```
b = cumsum(bsxfun(@times,W/B, y' * T),2);
```

This executes very efficiently, but it must be noted that the `cumsum`–construction (for cumulative summation) is not available in most programming languages. It should also be noted that using W * inv(B) instead of W/B is not good programming practice, because the inverse of the bidiagonal matrix $\mathbf{B}$ is a full upper triangular matrix and calculating it will be a waste of both time numerical precision. We remark that many other alternatives for computing the regression coefficients have been suggested in the literature, but using Equation 31, seems to be the best choice for speed and numerical precision.

The original NIPALS algorithm differs in that the regression coefficients $\mathbf{b}_k = \mathbf{W}_k\mathbf{z}_k$ are obtained from the linear system

$$\left(\mathbf{P}_k'\mathbf{W}_k\right)\mathbf{z}_k = \mathbf{q}_k, \tag{32}$$

where $\mathbf{P}_k'\mathbf{W}_k$ is treated as a full matrix. By uniqueness (see Equation 20), it follows that $\mathbf{P}_k'\mathbf{W}_k$ analytically equals the bidiagonal matrix $\mathbf{B}_k$. However, because of loss of orthogonality in floating-point arithmetic, small but nonzero off-bidiagonal entries will appear. Björck[3] showed that neglecting all off-bidiagonal entries in $\mathbf{P}_k'\mathbf{W}_k$ leads to a loss of numerical precision, because they compensate for the loss of orthogonality. On the other hand, the lower triangular elements of $\mathbf{P}_k'\mathbf{W}_k$ are of the order of unit roundoff, whereas the off-bidiagonal elements in the upper triangular part can be much larger. Therefore, in our NIPALS MATLAB code (see section A.2 below), we suggest that $\mathbf{P}_k'\mathbf{W}_k$ is treated as a full upper triangular matrix. In that case, the back substitution alternative for solving Equation 32 is still applicable. The recursion 29 is also still valid, and with MATLAB code, we can use the expression

```
beta = cumsum(bsxfun(@times,W/triu(P' *
                    W), y' * T),2);
```

for the evaluation of the regression coefficients. If wanted, this can of course be rewritten as an equivalent vector recursion.

5 | THE ALGORITHMS TO BE COMPARED

The algorithms can be split into 2 groups depending on whether or not they "deflate" the data $\mathbf{X}$ and $\mathbf{y}$ as in the NIPALS and Householder algorithms or just use the data matrix $\mathbf{X}$ and its transpose in matrix-vector operations.

- Algorithms deflating both $\mathbf{X}$ and $\mathbf{y}$:
  1. The mixed forward-backward stable reference method HHPLS by Björck[3] (accurate but slow).
  2. The original NIPALS by Wold et al[1] with normalized scores ($\mathbf{T}$) and weights ($\mathbf{W}$) conjectured to be stable in Björck.[3]
  3. The nonorthogonal scores PLS by Martens.[11]
- Additional algorithms considered to be stable in Anderson[2]:

4. SIMPLS by de Jong[10] known as one of the fastest PLS algorithms according to Andersson.[2]
5. Direct scores PLS (DSPLS) by Andersson—reported to be among the fastest PLS algorithms in Andersson.[2]
6. The improved kernel PLS (IKPLS) by Dayal and MacGregor[17]—reported to be among the fastest PLS algorithms in Andersson.[2]

- Algorithms without **X** deflation but including full reorthogonalization (all comparable to the algorithms 4-6 in terms of speed):

7. Bidiag2 adapted from Golub and Kahan[6] including full reorthogonalization of score and loading vectors.
8. Bidiag1 adapted from Paige and Saunders[4] including full reorthogonalization of score and loading vectors.
9. The PLSHY—a hybrid bidiagonalization algorithm obtained by a minor modification of Bidiag2 to include explicit **y** deflation.

Prototype MATLAB code for the algorithms 1, 2, and 7–9 is included in the appendix. For the other algorithms, we use the MATLAB code given by Andersson.[2] Manne[7] observed that in the NIPALS algorithm deflation of **y** is analytically unnecessary. Andersson[2] omits this deflation in his code as does several other authors. However, as was shown by Björck,[3] this omission may substantially increase the loss of orthogonality in the basis vectors $\mathbf{T}_k$ and $\mathbf{W}_k$ computed by NIPALS. Hence, our implementation of the NIPALS PLS performs deflation of **y**, as in the original algorithm,[1] (Note: No particular stopping criterion is implemented in the various algorithms. Deciding when to stop a PLS algorithm is problem dependent and considered to be outside the scope of this paper.)

To improve the performance of MATLAB codes, it is usually recommended to preallocate arrays to avoid repeated resizing in loops. We found that preallocation actually increased running times for our codes, which is why it is consistently omitted in all the algorithms we present.

6 | DATA SETS

We will consider five real data sets and one constructed data set for the PLS benchmark study:

1. The Near-infrared (NIR)/octane measurements data set provided by Kalivas.[18] This is a MATLAB example data set available from the *Statistics and Machine Learning Toolbox* by the command: `load spectra`. Here **X** is a matrix of 60 gasoline samples measured at 401 NIR wavelengths and **y** is a vector of 60 corresponding chemical measurements of octane numbers.
2. The Beer data set in Nørgaard et al.[19]**X** is a matrix of 60 samples measured at 926 NIR wavelengths, and **y** is a vector of 60 corresponding extract measurements. The data are available from the iToolbox at http://www.models.life.ku.dk/itoolbox.
3. The Melter data set from Eigenvector Research, Inc https://software.eigenvector.com/toolbox/download/, accessible from the demo-version of the PLS_Toolbox. **X** is a matrix of 300 samples measured at 20 temperatures in a Slurry Fed Ceramic Melter, and **y** is a vector of corresponding level values. This data set was considered in the EVRI-blog by Wise at http://www.eigenvector.com/evriblog/?p=268 called "Accuracy of PLS algorithms" for comparing some of the PLS algorithms considered in Andersson.[2]
4. The small round blue cell tumors microarray data set of Khan et al.[20]**X** is a matrix of 63 samples measured at 2318 genes, and **y** is a vector of class labels from $\{1, 2, 3, 4\}$ indicating various types of cancer. The data set can be downloaded from data sets for "The Elements of Statistical Learning": http://statweb.stanford.edu/~tibs/ElemStatLearn/data.html.
5. The ovarian cancer case vs high-risk control data set (WCX2 protein array experiment) from the FDA-NCI Clinical Proteomics Program Databank http://home.ccr.cancer.gov/ncifdaproteomics/ppatterns.asp. This is a MATLAB example data set available from the *Bioinformatics Toolbox* by the commands: `load ovariancancer, X = double(obs); y = strcmpi('Cancer',grp)*1;`**X** is a matrix of 216 samples where the ion intensity level is measured at 4000 specific mass-charge values for each sample. **y** is a corresponding vector of class labels from $\{0, 1\}$ representing the case/control status.
6. The contrived data set from Björk.[3] In this, **X** is a 50×8 matrix with singular values $\sigma_i = 10^{-i+1}$, $i = 1 : 8$ and $\mathbf{y} = \mathbf{X}\mathbf{e}$, where $\mathbf{e} = (1, \ldots, 1)'$. This data set is generated by the function `[X,y] = testp(50,8);` given in the appendix.

For each of the data sets 1 to 5, we also generated an associated poorly conditioned artificial data set by computing the reduced SVD of the (uncentered) **X** matrix and replacing its original r (nonzero) singular values blueσ_i, $i = 1, \ldots, r$, by a set of designed singular values 10^{p_i}, where $p_1 > p_2 > \ldots > p_r$ are evenly spaced numbers (powers) from $p_1 = 3$ to $p_r = -15$. The purpose of the manipulated data sets is to challenge the numerical precision of the tested algorithms with some nearly rank deficient data matrices for detection of any potential instability.

7 | RESULTS

From the Figures 1 and 2 below, we can compare the algorithms in terms of regression coefficients and fitted values for

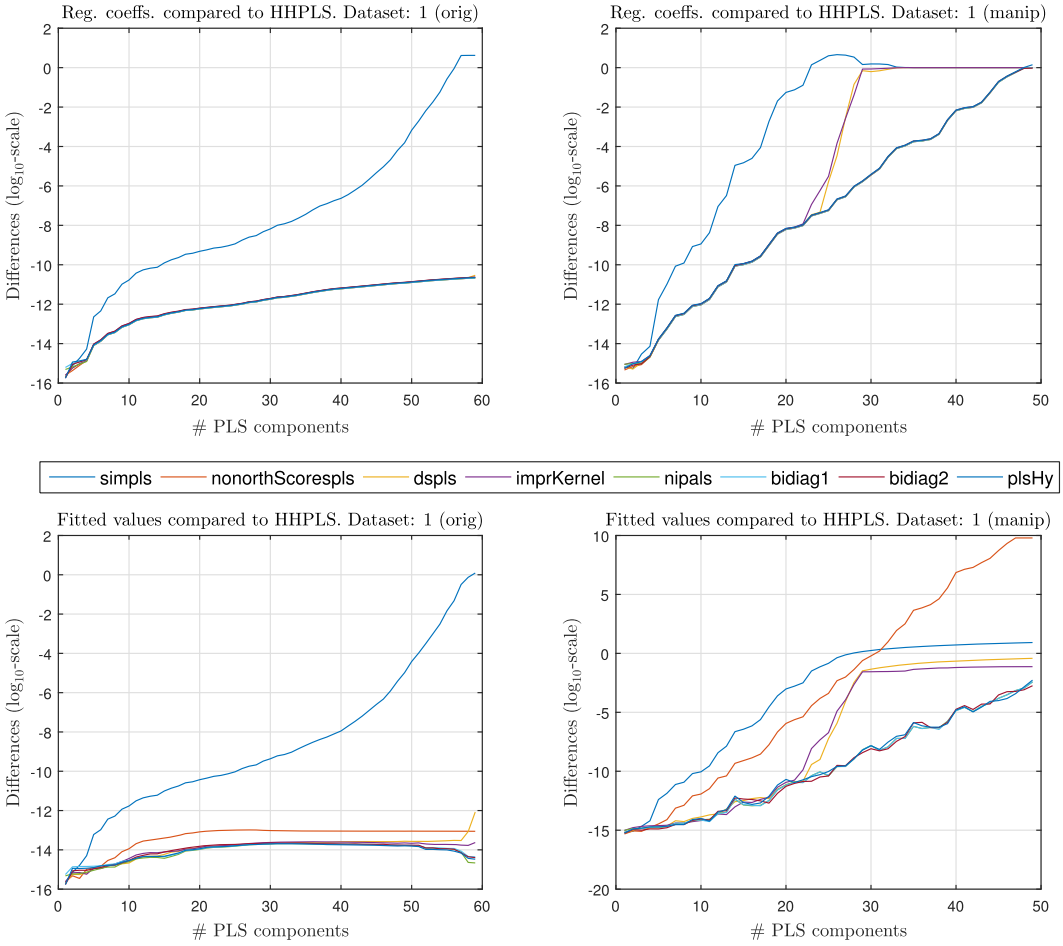

FIGURE 1 The NIR/Octane measurements—data set 1. The original data shows problems with the SIMPLS (blue). The manipulated data shows problems also for the nonorthogonal scores partial least squares (PLS) (red), the direct scores PLS (DSPLS) (orange), and the improved kernel PLS (violet). The other methods cluster well with the NIPALS PLS (green) and indicate similar stability properties

the data sets 1 and 5. The numerical precision of the computed regression coefficients $\mathbf{b}_{\text{Meth}}$ and fitted values $\mathbf{X}\mathbf{b}_{\text{Meth}}$ is measured by the normwise relative differences

$$\|\mathbf{b}_{HHPLS} - \mathbf{b}_{Meth}\|_2 / \|\mathbf{b}_{HHPLS}\|_2,$$
$$\|\mathbf{X}\mathbf{b}_{HHPLS} - \mathbf{X}\mathbf{b}_{Meth}\|_2 / \|\mathbf{X}\mathbf{b}_{HHPLS}\|_2$$

between the HHPLS and the other methods (Meth) for the various number of components. Corresponding graphs for the data sets 2 to 4 (available as supplementary material) confirm the patterns shown in Figures 1 and 2.

With the exception of SIMPLS, the tested algorithms gave about the same numerical precision in the regression coefficients and fitted values for data sets 1 to 5. SIMPLS showed poor numerical precision even with a fairly small number of factors for all the considered data sets. The instability of SIMPLS was also noticed by Andersson.[2] De Jong[10, p. 258] remarks that "SIMPLS does a similar job (as Bidiag2)" but starts from $\mathbf{X}\mathbf{X}'\mathbf{y}$ generating $\mathbf{T}$ and $\mathbf{X}'\mathbf{X}\mathbf{X}'\mathbf{y}$ generating the nonorthogonal weights $\mathbf{V}$. This use of higher order Krylov vectors as the initial vectors is a possible explanation of the observed loss of stability. The use of reorthogonalization in SIMPLS proposed by Faber and Ferré[21] improved its numerical precision considerably (data not shown), but even then, SIMPLS gave the worst numerical precision of all our tested algorithms.

For Bidiag1 and Bidiag2 we also tried 1-sided reorthogonalization, ie, reorthogonalizing only 1 set of basis vectors, either $\mathbf{W}$ or $\mathbf{T}$. This was found to lead to a substantial loss of numerical precision compared to full reorthogonalization (data not shown). Contrary to the remark by Wu and Manne[9] that "Explicit reorthogonalization … postpones the problems, but does not eliminate them," our results indicate that for algorithms such as Bidiag1 and Bidiag2 that do not deflate the data $\mathbf{X}$ and $\mathbf{y}$, full reorthogonalization of the basis matrices $\mathbf{W}$ and $\mathbf{T}$ is both necessary and sufficient for stability.

The PLS problems generated by data sets 1 to 5 are all fairly well conditioned, ie, the solution is not overly sensitive to small perturbations. This is natural, because PLS is a regularization method that projects the original data onto subspaces of smaller dimensions. The artificially ill-conditioned data sets shows problems also for the nonorthogonal scores

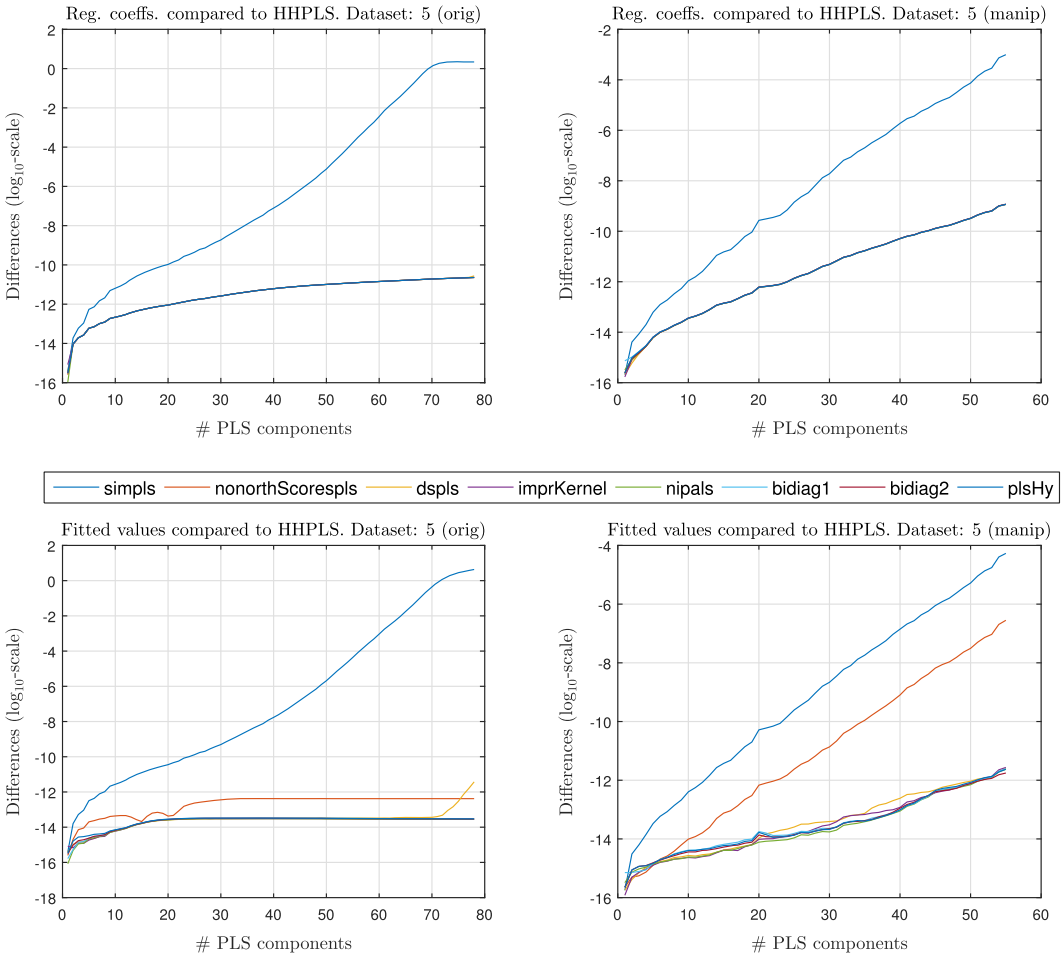

FIGURE 2 The ovarian cancer data—data set 5. The original data shows problems with the SIMPLS (blue). The manipulated data shows problems also for the nonorthogonal scores partial least squares (PLS) (red). The other methods cluster well with the NIPALS PLS (green). DSPLS, direct scores partial least squares

PLS (red), DSPLS (orange), and the IKPLS (violet). The other methods cluster well with the NIPALS (green) and indicate similar good stability properties.

As a complement, all algorithms were also tested on the contrived ill-conditioned data set 6 used by Björck.[3] This data set has $n = 50$ rows, $p = 8$ columns, and the exact vector of regression coefficients $\mathbf{b} = [1 \quad 1 \quad \ldots \quad 1]' \in \mathbf{R}^8$ is known by choice. Table 1 shows the relative error in the regression coefficients $\|\mathbf{b} - \mathbf{b}_{Meth}\|_2/\|\mathbf{b}\|_2$ for the different algorithms. The results confirms the conclusions based on the data sets 1 to 5, except that the previously observed lack of stability for the nonorthogonal scores PLS did not show up.

For the the same ill-conditioned data set 6, Björck[3] showed that the NIPALS PLS including $\mathbf{y}$ deflation gave a loss of orthogonality of the order 10^{-10} using double numerical precision. Omitting the $\mathbf{y}$ deflation turned out to be disastrous, as the loss of orthogonality then increased to about 10^{-2}.

In further tests on data set 6 with the NIPALS algorithm (with $\mathbf{y}$ deflation included), we computed the norms NU and NL of the off-bidiagonal elements in the upper and lower and triangular parts of $\mathbf{P}_k'\mathbf{W}_k$. We found that $NU = 6.4659e-11$, ie, of the same size as the loss of orthogonality in the basis vectors. However, $NL = 1.0191e-17$, ie, close to unit round off in double numerical precision. This observation supports

TABLE 1 Numerical precision of different methods when applied to example data from Björck[3]

Method	Numerical Precision
HHPLS	$5.6077e-11$
SIMPLS	$2.7735e+04$
nonorth scores PLS	$9.1559e-11$
DSPLS	0.0023
IKPLS	$1.5521e-04$
NIPALS (full)	$2.2247e-11$
NIPALS (triangular)	$9.4026e-11$
Bidiag1	$7.6880e-11$
Bidiag2	$2.3657e-11$
PLSHY	$9.3793e-11$

Abbreviations: DSPLS, direct scores partial least squares; HHPLS, Householder transformations partial least squares; IKPLS, improved kernel partial least squares; PLS, partial least squares.

TABLE 2 Five repeated runs of accumulated execution times (over the data sets 1-5) and the associated mean values (in sec) for all algorithms. The last column shows the mean execution times in percent (%) of the NIPALS result

Method	Run1	Run2	Run3	Run4	Run5	Mean	NIPALS, %
HHPLS	3.45	3.34	3.36	3.31	3.34	3.36	435.2
SIMPLS	0.34	0.37	0.35	0.35	0.34	0.35	45.0
nonorth scores PLS	0.76	0.79	0.75	0.76	0.77	0.77	99.3
DSPLS	0.20	0.20	0.19	0.19	0.20	0.19	25.2
IKPLS	0.18	0.19	0.18	0.18	0.18	0.18	23.4
NIPALS	0.76	0.79	0.76	0.78	0.77	0.77	100.0
Bidiag1	0.18	0.18	0.18	0.18	0.18	0.18	23.2
Bidiag2	0.17	0.17	0.17	0.17	0.17	0.17	22.2
PLSHY	0.17	0.17	0.17	0.17	0.18	0.17	22.6

Abbreviations: DSPLS, direct scores partial least squares; HHPLS, Householder transformations partial least squares; IKPLS, improved kernel partial least squares; PLS, partial least squares.

treating $\mathbf{P}_k'\mathbf{W}_k$ in NIPALS PLS as an upper triangular matrix. However, as shown in Table 1, the traditional choice of considering $\mathbf{P}_k'\mathbf{W}_k$ as a full matrix in the NIPALS PLS gave a slightly smaller error in the regression coefficients for data set 6.

Regarding computational efficiency, Table 2 shows the accumulated execution times (for 5 repeated runs) over the collection of data sets (1-5, including both the original and manipulated versions of each data set) for all the considered algorithms. In the last column (based on the mean of the five repeated runs), the mean execution time of the NIPALS algorithm is set to 100% to indicate a relative mean execution time for the other algorithms.

Further laptop tests (data not shown) with some larger simulated data sets (matrices of dimensions 10000×30000 and 30000×10000 with 100 extracted components) indicate that the NIPALS and nonorthogonal scores PLS can be slower by almost a factor 7 when compared to the Bidiag2. For the smallest data sets, we observed that that timings in the group of fast algorithms could vary with as much as 30% by changing only minor details in the MATLAB code. Therefore, we stress that the reported differences in timings are relatively coarse estimates. Finally, we note that although MATLAB is a natural choice for prototyping algorithms, a more realistic timings comparison should be done using implementations in C++ or Fortran.

8 | CONCLUSIONS

In this paper, we have performed tests on 6 benchmark data sets of several widely used PLS algorithms as well as modified versions of Bidiag1 and Bidiag2 with reorthogonalization of both score and loading vectors. The algorithms consistently showing good numerical stability for all data sets in our benchmark study were

- Householder PLS (HHPLS), NIPALS PLS, Bidiag1, Bidiag2, and hybrid bidiagonalization (PLSHY).

These algorithms may be used interchangeably when numerical precision is the only concern. Our new versions of Bidiag1 and Bidiag2 were among the best in numerical precision of all our tested algorithms. However, the use of reorthogonalization is essential. In previous tests by Andersson[2] of Bidiag2 without reorthogonalization, the computed regression vectors were imprecise, as were the score and loading vectors. Also note that, as demonstrated in Björk,[3] the otherwise excellent stability properties of the NIPALS PLS may be destroyed if the deflation of $\mathbf{y}$ is omitted.

The reference algorithm HHPLS is the slowest among the stable alternatives. The details of our time comparisons were shown in Table 2. The algorithms concluded to be both numerically stable and fast were

- Bidiag1, Bidiag2, and PLSHY.

These algorithms are also the methods of choice for PLS modelling with sparse or structured $\mathbf{X}$ data (a common situation in Big Data applications), as deflation of $\mathbf{X}$ is avoided, and the matrix-vector multiplications in the Equations 17 and 18 for such cases can be implemented with particular efficiently. As Bidiag2 has the slightly simpler implementation, it may be the preferred alternative over Bidiag1.

The SIMPLS, DSPLS, and IKPLS (that are all avoiding deflation of the $\mathbf{X}$ data) were also among the fastest in our benchmark study (see Table 2), but the stability of these algorithms were demonstrated to fail in several situations. The nonorthogonal scores PLS was demonstrated to be both slow and some times unstable.

Finally, we note that model selection for PLS is often based on some s-fold cross-validation strategy. Our experience with real data sets indicate that by choosing, say, $s = 5$ cross-validation segments, the model building and selection by the stable and fast bidiagonalization algorithms will be

almost as efficient as doing just one single NIPALS model fitting (without the cross validation).

ACKNOWLEDGEMENT

The Research Council of Norway (project number 239070) provided financial support for this work.

REFERENCES

1. Wold S, Ruhe A, Wold H, Dunn WJ. The collinearity problem in linear regression, the partial least squares (PLS) approach to generalized inverses. *SIAM J Sci Stat Comput*. 1984;5:735–743.
2. Andersson M. A comparison of nine PLS1 algorithms. *J Chemom*. 2009;23:518–529.
3. Björck Å. Stability of two direct methods for bidiagonalization and partial least squares. *SIAM J Matrix Anal Appl*. 2014;35(1):279–291.
4. Paige CC, Saunders MA. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans Math Softw*. 1982;8:43–71.
5. Lanczos C. Solution of systems of linear equations by minimized iterations. *J Res Nat Bur Standards, Sect B*. 1952;49:33–53.
6. Golub GH, Kahan W. Calculating the singular values and pseudoinverse of a matrix. *SIAM J Numer Anal Ser B*. 1965;2:205–224.
7. Manne R. Analysis of two partial-least-squares algorithms for multivariate calibration. *Chemom Intell Lab Syst*. 1987;2:187–197.
8. Eldén L. Partial least-squares vs. Lanczos bidiagonalization—I: analysis of a projection method for multiple regression. *Comput Statist Data Anal*. 2004;46:11–31.
9. Wu W, Manne R. Fast regression methods in a Lanczos (or PLS-1) basis. Theory and applications. *Chemom Intell Lab Syst*. 2000;51:145–161.
10. de Jong S. SIMPLS: An alternative approach to partial least squares regression. *Chemom Intell Lab Syst*. 1993;18(3):251–263.
11. Martens H, Næs T. *Multivariate Calibration*. 2nd ed. Chichester, UK: Wiley; 1989.
12. Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. *J Res Nat Bur Standards, Sect B*. 1952;49:409–436.
13. Golub GH, O'Leary DP. Some history of the conjugate gradient and Lanczos algorithms: 1948–1976. *SIAM Review*. 1989;31:50–102.
14. Björck Å.. *Numerical Methods in Matrix Computations*, of Texts in Applied Mathematics, vol. 59. Berlin, Heidelberg, New York: Springer; 2014.
15. Indahl UG. The geometry of PLS1 explained properly: 10 key notes on mathematical properties and some alternative algorithmic approaches to PLS1 modelling. *J Chemom*. 2014;28:168–180.
16. Christopher C. Paige. The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices. *PhD thesis*, University of London, 1971.
17. Dayal BS, MacGregor JF. Improved PLS algorithms. *J Chemom*. 1997;11:73–85.
18. Kalivas JH. Two data sets of near infrared spectra. *Chemom Intell Lab Syst*. 1997;37:255–259.
19. Nørgaard L, Saudland A, Wagner J, Nielsen JP, Munck L, Engelsen SB. Interval partial least squares regression (iPLS): a comparative chemometric study with an example from near-infrared spectroscopy. *Appl Spectrosc*. 2000;37:413–419.
20. Khan J, Wei JS, Ringner M. et al. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nat Med*. 2001;7:673–679.
21. Faber NM, Ferré J. On the numerical stability of two widely used PLS algorithms. *J Chemom*. 2008;22:101–105.

SUPPORTING INFORMATION

Additional Supporting Information may be found online in the supporting information tab for this article.

How to cite this article: Björck Å, Indahl UG. Fast and stable partial least squares modelling: A benchmark study with theoretical comments. *Journal of Chemometrics*. 2017;e2898. https://doi.org/10.1002/cem.2898

APPENDIX WITH MATLAB CODE

A.1 The Golub Kahan Householder PLS

```
function [beta,W,T,B] = hhpls(X,y,A)
% -------------------------------------------------------------
% ----------------- Ake Bjorck 9/6-2016 -----------------------
% -------------------------------------------------------------
[n,p] = size(X);
X = [y'*X; X];          % Append row on top of X.
w = zeros(p,1);  t = zeros(n,1);
B = zeros(A,2);         % B stored by diagonals
% ---------------- Start bidiagonalization --------------------
for a = 1:A
% ----------Generate and apply right Householder reflection----
   if a < p,
      [w(a:p), X(a,a), theta] = hhgen(X(a,a:p)');
       X(a+1:n+1,a:p) = hhapp(X(a,a),w(a:p),X(a+1:n+1,a:p)')';
       X(a,a+1:p) = w(a+1:p)';  ar = a; % Save right HH vector
   end
   if a == p, theta = X(p,p); end,
  B(a,2) = theta;
% ----------Generate and apply left Householder reflection-----
   if a < n,
      [t(a:n), X(a+1,a), rho] = hhgen(X(a+1:n+1,a));
      X(a+1:n+1,a+1:p) = hhapp(X(a+1,a),t(a:n),X(a+1:n+1,a+1:p));
      y(a:n) = hhapp(X(a+1,a),t(a:n),y(a:n));
      X(a+2:n+1,a) = t(a+1:n);  al = a; % Save left HH vector
   end
   if a == n,  rho = X(n+1,n); end
   B(a,1) = rho;
end
% ----------- Generate orthogonal matrices T and W -------------
W = eye(p,A);  T = eye(n,A);
for a = A:-1:1
   if a <= ar,
      w(a:p) = [1; X(a,a+1:p)'];
      W(a:p,a:A) = hhapp(X(a,a),w(a:p),W(a:p,a:A));
   end
   if a <= al,
      t(a:n) = [1; X(a+2:n+1,a)];
      T(a:n,a:A) = hhapp(X(a+1,a),t(a:n),T(a:n,a:A));
   end
end
% -----------------Generate regression coefficients--------------
beta = zeros(p,A);  s = 0;  d = 0;
for a = 1:A
   d = (W(:,a) - B(a,2)*d)/B(a,1);
   s = s + y(a)*d;  beta(:,a) = s;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [u,beta,sigma] = hhgen(x)
%   Constructs reflector H = I - beta*u*u' such that
%   H*x = sigma*e_1, with sigma = ||x||_2, u_1 = 1.
% -------------------------------------------------------------------
u = x;  sigma = norm(x);
u(1) = sigma + abs(x(1));
beta = u(1)/sigma;
if x(1) < 0, u(1) = -u(1);
else sigma = -sigma;
end
u = u/u(1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = hhapp(tau,u,X)
% Applies Householder reflection
%  (I - tau*u'*u)*X
% ---------------------------------------------------------
X = X - (tau*u)*(u'*X);
```

A.2 NIPALS with normalization

```
function [beta,W,T,P] = nipalsN(X,y,A)
% ---------------------------------------------------------
% --------------- Ake Bjorck 31/3-2016 --------------------
% ---------------------------------------------------------
% ------------- Solution of the PLS1-problem --------------
for a = 1:A
    w = X'*y;  w = w/norm(w);  W(:,a) = w;
    t = X*w;   t = t/norm(t);  T(:,a) = t;
% ------------------- Deflate X and y ----------------------
    P(:,a) = X'*t;  X = X - t*P(:,a)';
    q(a) = y'*t;    y = y - q(a)*t;
end
% ---------- Calculate regression coefficients ------------
beta = cumsum(bsxfun(@times,(W/triu(P'*W)), q),2);
```

A.3 Bidiag2 with reorthogonalization

```
function [beta,W,T,B] = bidiag2(X,y,A)
% ------------------------------------------------------------
% ---------------- Ake Bjorck 9/6-2016 -----------------------
% ------------------------------------------------------------
B = zeros(A,2);   % B stored by diagonals
w = X'*y;  w = w/norm(w);  W = w;
t = X*w;  rho = norm(t);  t = t/rho; T = t;
B(1,1) = rho;
d = w/rho;  beta = (t'*y)*d;
% ---------------- Continue bidiagonalization ----------------
for a = 2:A
    w = X'*t - rho*w; w = w - W*(W'*w);  % Reorthogonalize w
    theta = norm(w);  w = w/theta;  W(:,a) = w;
    t = X*w - theta*t; t = t - T*(T'*t);  % Reorthogonalize t
    rho = norm(t);  t = t/rho; T(:,a) = t;
    B(a-1,2) = theta;  B(a,1) = rho;
% --------------- Update regression coefficients ---------------
    d = (w - theta*d)/rho;
    beta(:,a) = beta(:,a-1) + (t'*y)*d;
end
```

A.4 Bidiag1 with reorthogonalization

```
function [beta,W,T,B] = bidiag1(X,y,A)
% ------------------------------------------------------------
% ----------------- Ake Bjorck 9/6-2016 ----------------------
% ------------------------------------------------------------
B = zeros(A,2);   % B stored by diagonals
gamma = norm(y);  t = y/gamma;  T = t;
w = X'*t;  alpha = norm(w);  w = w/alpha;
W = w;  d = w;  reg = 0;
rhobar = alpha;  phibar = gamma;
% ---------------- Start bidiagonalization -------------------
for a = 1:A
    t = X*w - alpha*t;  t = t - T*(T'*t);  % Reorthogonalize t
    gamma = norm(t);  t = t/gamma;  T(:,a+1) = t;
    w = X'*t - gamma*w;  w = w - W*(W'*w);  % Reorthogonalize w
    alpha = norm(w);  w = w/alpha;  W(:,a+1) = w;
% -------------- Construct and apply i:th Givens rotation ------
    rho = norm([rhobar,gamma]);
    cos = rhobar/rho; sin = gamma/rho;
    theta = sin*alpha; rhobar = -cos*alpha;
    phi = cos*phibar; phibar = sin*phibar;
    G = [cos, sin; sin, -cos];
    T(:,a:a+1) = T(:,a:a+1)*G;
    B(a-1,2) = theta;  B(a,1) = rho;
% ------------- Update regression coefficients ----------------
    reg = reg + (phi/rho)*d;  beta(:,a) = reg;
    d = w - (theta/rho)*d;
end
```

A.5 The hybrid PLS with reorthogonalization and y-deflation

```
function [beta,W,T,B,q] = plsHy(X,y,A)
% ------------------------------------------------------------
% -------------- Ulf Indahl 20/06-2016 -----------------------
% ------------------------------------------------------------
B = zeros(A,2);   % B stored by diagonals
w0 = X'*y; w = w0/norm(w0);
t = X*w; rho = norm(t); t = t/rho; q(1) = y'*t;
W = w; T = t; B(1,1) = rho;
d = w/rho;  beta(:,1) = (t'*y)*d;
y = y - t*q(1);
% -------------------- Solution of the PLS1-problem --------------
for a = 2:A,
    w1 = X'*y; w = (w0-w1)/q(a-1) - rho*w; w0 = w1;   % w = X'*t - rho*w;
    w = w - W*(W'*w);  theta = norm(w); w = w/theta;  % Reorthogonalize & normalize w
    t = X*w; t = t - T*(T'*t); % Reorthogonalize t    %(t = X*w - theta*t;)
    rho = norm(t); t = t/rho; q(a) = y'*t;
    W(:,a) = w; T(:,a) = t;
    B(a-1,2) = theta;  B(a,1) = rho;
% ---------------- Update regression coefficients ----------------
    d = (w - theta*d)/rho;
    beta(:,a) = beta(:,a-1) + q(a)*d;
    y = y - t*q(a);                 % Deflate y
end
```

A.6 Generate contrived data set 6

```
function [X,y] = testp(n,p);
% ------------------------------------------------------------
% -------------- Ake Bjorck 03/10-2010 -----------------------
% ------------------------------------------------------------
for i = 1:p
    z(i) = cos(4*pi*i/p);
    sigma(i) = 10.^(1-i);    % Generate singular values
end
for j = 1:n
    w(j) = sin(4*pi*j/n);
end
%----------------- Normalize vectors w,z--------------------------
w = w'/norm(w);  z = z'/norm(z);
W = eye(n) - 2*w*w';  Z = eye(p) - 2*z*z';
X = W*[diag(sigma), zeros(p,n-p)]'*Z;
y = X*ones(p,1);
```