JOURNAL OF CHEMOMETRICS, VOL. 8, 169–174 (1994)

SHORT COMMUNICATION

COMMENTS ON THE PLS KERNEL ALGORITHM

SIJMEN DE JONG
Unilever Research Laboratorium Vlaardingen, PO Box 114, NL-3130 AC Vlaardingen, Netherlands
AND
CAJO J. F. TER BRAAK
Agricultural Mathematics Group (GLW-DLO), PO Box 100, NL-6700 AC Wageningen, Netherlands

SUMMARY

Lindgren *et al.* (*J. Chemometrics*, 7, 45–59 (1993)) published a so-called kernel algorithm for PLS regression of $\mathbf{Y}$ against $\mathbf{X}$ when the number of objects is very large. The algorithm is based solely on deflation of the cross-product matrices $\mathbf{X}^T\mathbf{X}$, $\mathbf{Y}^T\mathbf{Y}$ and $\mathbf{X}^T\mathbf{Y}$. The algorithm is now described in a shorter and more transparent way and compared with a similar algorithm for the singular value decomposition of $\mathbf{X}^T\mathbf{Y}$.

KEY WORDS Kernel algorithm PLS SVD

1. INTRODUCTION

Lindgren *et al.*[1] recently published a so-called kernel algorithm for PLS regression that is particularly efficient when the number of objects is very much larger than the number of variables. In the standard NIPALS procedure[2] the data matrices $\mathbf{X}$ $(n \times p)$ and $\mathbf{Y}$ $(n \times m)$ are deflated by projecting out successive orthogonal component scores, $\mathbf{t}_a$, $a = 1, 2, \ldots$. When the number of objects becomes huge $(n \gg p)$, it pays to invest in the once-only construction of $\mathbf{X}^T\mathbf{X}$ $(p \times p)$, $\mathbf{X}^T\mathbf{Y}(p \times m)$ and $\mathbf{Y}^T\mathbf{Y}$ $(m \times m)$ and to extract the PLS model from these small cross-product (or 'kernel') matrices. This is the main idea behind the kernel approach. Under favourable conditions it may be up to four times as fast as efficient PLS algorithms based on a conjugate gradient approach such as BIDIAG2,[3,4] LSQR,[4] CGLS[4] and SIMPLS[5] and orders of magnitude faster than the standard NIPALS-PLS algorithm.

In this communication we propose a simplified estimation of the PLS model from the cross-products $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{Y}$ (Section 2). It closely follows the decomposition of cross-product matrices presented in Reference 6, leading to an elegant and concise reformulation of the kernel algorithm. The main thrust of this communication is not so much to introduce a new algorithm but rather to shed new light on the kernel approach and the PLS method as such. In Section 3 we compare the modified and the original kernel PLS algorithm. For both algorithms the results are identical to those obtained with standard NIPALS-PLS. The

CCC 0886–9383/94/020169–06 *Received 13 September 1993*
Accepted 18 November 1993

modified algorithm is consistently faster, but not much. The rate-determining step in the kernel algorithm is the *construction* of the cross-product matrices $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{Y}$ rather than their *decomposition*.

The PLS decomposition of the $\mathbf{X}^T\mathbf{Y}$ cross-product presented here can be nicely contrasted with the singular value decomposition (SVD) of $\mathbf{X}^T\mathbf{Y}$ (Section 4). Comparing these two decomposition algorithms reveals the similarities and differences between the two methods. Again, the value lies in adding further insight to the PLS method.

2. CROSS-PRODUCT DECOMPOSITION PLS ALGORITHM

In PLS regression one seeks a decomposition of the centred predictor data $\mathbf{X}$ in terms of component scores $\mathbf{t}_1, \mathbf{t}_2, \dots$ and associated loadings $\mathbf{p}_1, \mathbf{p}_2, \dots$. We will choose the component scores to be uncorrelated and of unit norm. The scores are collected in a column-orthonormal matrix $\mathbf{T}$, the loadings in a matrix $\mathbf{P}$. The components are constructed as linear combinations of the predictors using weights $\mathbf{w}_1, \mathbf{w}_2, \dots$. To ensure orthogonality of the scores, the weights apply to progressively deflated $\mathbf{X}$-matrices $\mathbf{X}_0$ $(\equiv \mathbf{X})$, $\mathbf{X}_1$, $\mathbf{X}_2, \dots$, where $\mathbf{X}_a$ indicates the residual matrix obtained by projecting out the first a score vectors. The (normed) weights $\mathbf{w}_a$ are chosen such as to maximize a covariance criterion, i.e. they correspond to the first eigenvector of $\mathbf{X}_{a-1}^T\mathbf{Y}_{a-1}\mathbf{Y}_{a-1}^T\mathbf{X}_{a-1}$. Since the scores need to be normed, the weights $\mathbf{w}_a$ have to be renormalized in the metric $\mathbf{X}_{a-1}^T\mathbf{X}_{a-1}$.

Note that $\mathbf{X}$ can be decomposed completely when $\mathbf{T}$ and $\mathbf{P}$ have the same rank (r) as $\mathbf{X}$. For $\mathbf{Y}$ this cannot be done generally, since $\mathbf{Y}$ will have a residual part $\mathbf{F}$ that is orthogonal to $\mathbf{X}$ and hence to $\mathbf{T}$. Thus we find

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_r\mathbf{p}_r^T \tag{1}$$

$$\mathbf{Y} = \mathbf{T}\mathbf{C}^T + \mathbf{F} = \mathbf{t}_1\mathbf{c}_1^T + \mathbf{t}_2\mathbf{c}_2^T + \cdots + \mathbf{t}_r\mathbf{c}_r^T + \mathbf{F} \tag{2}$$

where $\mathbf{P} = \mathbf{X}^T\mathbf{T}$ is the $p \times r$ matrix of X-loadings and $\mathbf{C} = \mathbf{Y}^T\mathbf{T}$ is the corresponding $m \times r$ matrix of Y-loadings. Using the fact that we have chosen $\mathbf{T}^T\mathbf{T} = \mathbf{I}_r$, we may develop the following decompositions of the cross-product matrices $\mathbf{X}^T\mathbf{X}$, $\mathbf{X}^T\mathbf{Y}$ and $\mathbf{Y}^T\mathbf{Y}$:

$$\mathbf{X}^T\mathbf{X} = \mathbf{P}\mathbf{T}^T\mathbf{T}\mathbf{P}^T = \mathbf{P}\mathbf{P}^T = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \cdots \tag{3}$$

$$\mathbf{X}^T\mathbf{Y} = \mathbf{P}\mathbf{T}^T\mathbf{T}\mathbf{C}^T = \mathbf{P}\mathbf{C}^T = \mathbf{p}_1\mathbf{c}_1^T + \mathbf{p}_2\mathbf{c}_2^T + \cdots \tag{4}$$

$$\mathbf{Y}^T\mathbf{Y} = \mathbf{C}\mathbf{T}^T\mathbf{T}\mathbf{C}^T + \mathbf{F}^T\mathbf{F} = \mathbf{c}_1\mathbf{c}_1^T + \mathbf{c}_2\mathbf{c}_2^T + \cdots + \mathbf{F}^T\mathbf{F} \tag{5}$$

These decompositions (see Reference 6, p. 228) suggest a PLS2 algorithm based on deflation of the cross-product matrices. On noting that the deflation of $\mathbf{X}_a$ and $\mathbf{Y}_a$ in the original PLS2 algorithm is a projection operation, we obtain

$$\begin{aligned}(\mathbf{X}^T\mathbf{X})_a = \mathbf{X}_a^T\mathbf{X}_a &= [(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{X}_{a-1}]^T(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{X}_{a-1} = \mathbf{X}_{a-1}^T(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{X}_{a-1}\\ &= \mathbf{X}_{a-1}^T\mathbf{X}_{a-1} - \mathbf{X}_{a-1}^T\mathbf{t}_a\mathbf{t}_a^T\mathbf{X}_{a-1} = (\mathbf{X}^T\mathbf{X})_{a-1} - \mathbf{p}_a\mathbf{p}_a^T\end{aligned} \tag{6}$$

$$\begin{aligned}(\mathbf{X}^T\mathbf{Y})_a = \mathbf{X}_a^T\mathbf{Y}_a &= [(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{X}_{a-1}]^T(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{Y}_{a-1} = \mathbf{X}_{a-1}^T(\mathbf{I}_n - \mathbf{t}_a\mathbf{t}_a^T)\mathbf{Y}_{a-1}\\ &= \mathbf{X}_{a-1}^T\mathbf{Y}_{a-1} - \mathbf{X}_{a-1}^T\mathbf{t}_a\mathbf{t}_a^T\mathbf{Y}_{a-1} = (\mathbf{X}^T\mathbf{Y})_{a-1} - \mathbf{p}_a\mathbf{c}_a^T\end{aligned} \tag{7}$$

The loadings in (6) and (7) are obtainable from the preceding cross-product matrices and the current weights

$$\mathbf{p}_a = \mathbf{X}_{a-1}^T\mathbf{t}_a = \mathbf{X}_{a-1}^T\mathbf{X}_{a-1}\mathbf{w}_a = (\mathbf{X}^T\mathbf{X})_{a-1}\mathbf{w}_a \tag{8}$$

$$\mathbf{c}_a = \mathbf{Y}_{a-1}^T\mathbf{t}_a = \mathbf{Y}_{a-1}^T\mathbf{X}_{a-1}\mathbf{w}_a = (\mathbf{Y}^T\mathbf{X})_{a-1}\mathbf{w}_a \tag{9}$$

The weights $\mathbf{w}_a$ follow from the eigenanalysis of $(\mathbf{X}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X})_{a-1} = (\mathbf{X}^T\mathbf{Y})_{a-1}(\mathbf{X}^T\mathbf{Y})_{a-1}^T$ and subsequent rescaling using $(\mathbf{X}^T\mathbf{X})_{a-1}$:

$$\mathbf{w}_a := \mathbf{w}_a / \mid \mathbf{X}_{a-1}\mathbf{w}_a \mid = \mathbf{w}_a / [\mathbf{w}_a^T(\mathbf{X}^T\mathbf{X})_{a-1}\mathbf{w}_a]^{1/2} \tag{10}$$

The individual terms in the expansions (3) and (5) can be used to compute the amount of variance accounted for by each PLS component:

$$\mathrm{tr}(\mathbf{X}^T\mathbf{X}) = \mathrm{tr}(\mathbf{P}\mathbf{T}^T\mathbf{T}\mathbf{P}^T) = \mathrm{tr}(\mathbf{P}\mathbf{P}^T) = \mathrm{tr}(\Sigma\ \mathbf{p}_a\mathbf{p}_a^T) = \Sigma\ \mathrm{tr}(\mathbf{p}_a\mathbf{p}_a^T) = \Sigma\ \mathbf{p}_a^T\mathbf{p}_a \tag{11}$$

$$\mathrm{tr}(\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}) = \mathrm{tr}(\mathbf{Y}^T\mathbf{T}\mathbf{T}^T\mathbf{Y}) = \mathrm{tr}(\mathbf{C}\mathbf{C}^T) = \mathrm{tr}(\Sigma\ \mathbf{c}_a\mathbf{c}_a^T) = \Sigma\ \mathrm{tr}(\mathbf{c}_a\mathbf{c}_a^T) = \Sigma\ \mathbf{c}_a^T\mathbf{c}_a \tag{12}$$

One may compute a matrix $\mathbf{R}$ of weight vectors ($\mathbf{W}^*$ of Reference 1) which give $\mathbf{T}$ directly when applied to the original $\mathbf{X}$:

$$\mathbf{T} = \mathbf{X}_0\mathbf{R} \tag{13}$$

These weights can also be used to calculate new scores for new data. $\mathbf{R}$ is given by[7]

$$\mathbf{R} = \mathbf{W}(\mathbf{P}^T\mathbf{W})^{-1} \tag{14}$$

and can be used to predict $\mathbf{Y}$ directly from $\mathbf{X}$, $\hat{\mathbf{Y}} = \mathbf{T}\mathbf{T}^T\mathbf{Y} = \mathbf{X}_0\mathbf{R}\mathbf{C}^T$, yielding a simple expression for the PLS multivariate regression model

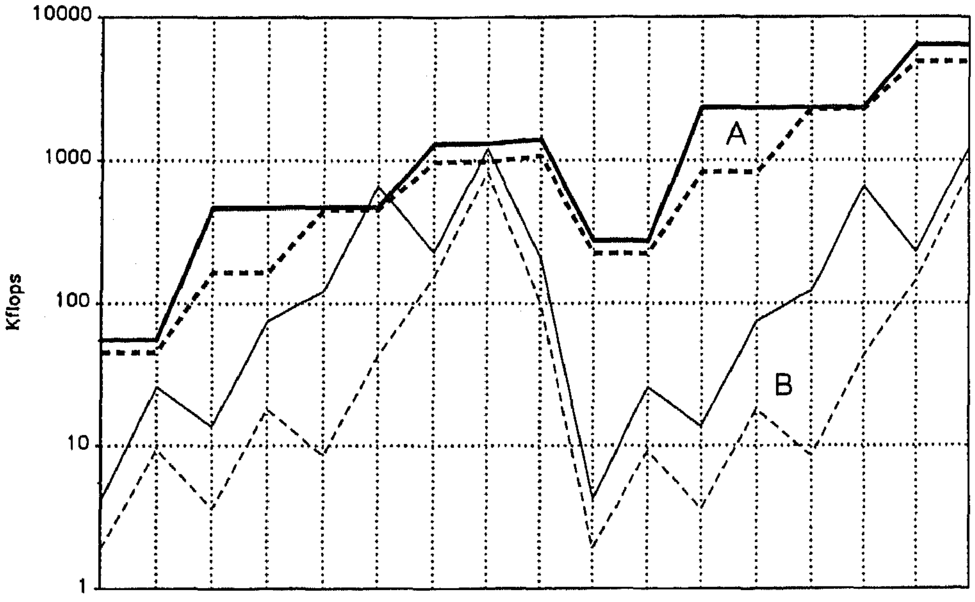
$$\mathbf{B} = \mathbf{R}\mathbf{C}^T \tag{15}$$

Given these relations, we now can present the simplified kernel algorithm (see Appendix). In the algorithm we have computed the weights $\mathbf{w}_a$ indirectly via the eigenanalysis of $(\mathbf{Y}^T\mathbf{X}\mathbf{X}^T\mathbf{Y})_{a-1} \equiv (\mathbf{X}^T\mathbf{Y})_{a-1}^T(\mathbf{X}^T\mathbf{Y})_{a-1}$, which generally has much smaller dimension than $(\mathbf{X}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X})_{a-1}$. Here we use the relation $\mathbf{w}_a \propto (\mathbf{X}^T\mathbf{Y})\mathbf{q}_a$, with $\mathbf{q}_a$ being the dominant eigenvector of $(\mathbf{Y}^T\mathbf{X}\mathbf{X}^T\mathbf{Y})_{a-1}$. Alternatively, $\mathbf{w}_a$ can be calculated as the rescaled first left singular vector of $(\mathbf{X}^T\mathbf{Y})_{a-1}$. Usually a predictive model can be built employing a few components only (A) rather than the maximum possible number (r).

The algorithm can be similarly applied when the $\mathbf{X}$ data set is very wide ($p \gg n$). In that case one first rotates the data to canonical space and then works with the matrix of principal component scores, $\mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{\Lambda}$, rather than $\mathbf{X}$ itself. Here $\mathbf{U}$ and diagonal $\mathbf{\Lambda}^2$ follow from eigenanalysis of $\mathbf{X}\mathbf{X}^T$. At the end of the PLS analysis one should back-rotate the weights $\mathbf{W}$ and $\mathbf{R}$, loadings $\mathbf{P}$ and regression coefficients $\mathbf{B}$ by premultiplying these matrices by the eigenvectors $\mathbf{V} = \mathbf{X}^T\mathbf{U}\mathbf{\Lambda}^{-1}$. This procedure is allowed, as we have checked, since PLS is invariant under orthogonal transformation of the X-variables and/or Y-variables.

3. COMPARISON AND EVALUATION

We have tested the performance of the new algorithm in comparison with that of the published algorithm. Both kernel algorithms give the same results as standard NIPALS-PLS using UNSCRAMBLER software. The speed of computation has been compared using the same factorial design as described by Lindgren *et al.*[1] Here the number of objects (n), the number of X-variables (p), the number of Y-variables (m) and the number of dimensions (A) have been varied according to a 2^4 design augmented with a centre point. The total number of NIPALS iterations for five dimensions was 142, close to the value of Reference 1. As responses we adopted the number of floating point operations (flops) consumed in constructing the cross-product matrices (stage A) and in deflating these matrices (stage B). The results for all design points are shown in Figure 1.



n	500	500	500	500	500	500	500	500	1500	2500	2500	2500	2500	2500	2500	2500	2500
p	5	5	5	5	25	25	25	25	15	5	5	5	5	25	25	25	25
m	5	5	25	25	5	5	25	25	15	5	5	25	25	5	5	25	25
A	1	5	1	5	1	5	1	5	3	1	5	1	5	1	5	1	5

Figure 1. Performance (in kflops) of original (———) and modified (–––) kernel algorithms. The contributions of the construction (A, thick lines) and decomposition (B, thin lines) stages are displayed separately

The modified algorithm is consistently more economic. For stage A the advantage stems from the fact that in the modified algorithm we only calculate the diagonal elements of $\mathbf{Y}^T\mathbf{Y}$ rather than the full cross-product matrix. For a larger number of Y-variables this can make a substantial difference. In the published kernel PLS algorithm[1] $\mathbf{Y}^T\mathbf{Y}$ is fully calculated though only the diagonal is required. The results for stage B focus on the essential difference between the two algorithms. Here the modified algorithm is at a clear advantage. However, the advantage loses importance when the number of objects increases and stage A consumes the larger part of the total flop count. When applying cross-validation, stage B may become more important and the advantage of the modified algorithm more prominent.

Inspection of both kernel algorithms discloses the more important factors contributing to the total number of floating point operations. These are the third-order terms np^2 (for constructing $\mathbf{X}^T\mathbf{X}$), npm ($\mathbf{X}^T\mathbf{Y}$), nm^2 or $nm(\mathbf{Y}^T\mathbf{Y})$ and, in stage B, Ap^2, Apm and Am^2. Regression analysis using these six high-order terms accounts for more than 99·99% of the variance for both algorithms. This six-factor model has a residual variance ($1 - R^2 < 0{\cdot}01\%$) that is much smaller than the ingenuous ten-factor model ($1 - R^2 = 1{\cdot}9\%$) considered by Lindgren *et al.*[1]

A few more comments on the kernel algorithm are in order.

1. With the current modifications the kernel algorithm has become about as efficient as the recently published SIMPLS algorithm.[5] The average performance in the above comparative evaluation was 1974 kflops for the original kernel algorithm, 1350 kflops for the modified kernel algorithm and 1162 kflops for the SIMPLS algorithm.

2. The modified algorithm requires less storage space since it does without the $n \times n$ matrices $\mathbf{Y}^{\mathrm{T}}\mathbf{Y}$, $(\mathbf{I}_p - \mathbf{w}_a\mathbf{p}_a^{\mathrm{T}})$ and $(\mathbf{X}^{\mathrm{T}}\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\mathbf{X})_a$.
3. The deflation of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ in equation (6) preserves symmetry exactly, also when computed with limited precision. For the original kernel PLS algorithm symmetry is preserved in theory. With the published code exact symmetry is lost.
4. MATLAB has no special built-in function to extract the first eigenvector only. One therefore needs to extract all eigenvectors after each deflation step and retain the dominant one only. For matrices of order less than 20 this appears just as efficient as using the power method. If programmed in FORTRAN or C, it is efficient to use specialized routines to extract the first eigenvector.[8,9]
5. For the calculation of mean-corrected sum-of-squares and product matrices it is numerically unstable[10] to follow equation (38) of Lindgren *et al.*[1] A more stable routine is given by Clarke.[11]

4. PLS2 AND THE SINGULAR VALUE DECOMPOSITION OF $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$

The novice to PLS2 often asks about the relation between PLS and the more familiar singular value decomposition[6,12] of $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$. The easy answer is of course that SVD is symmetric in $\mathbf{X}$ and $\mathbf{Y}$ whereas PLS2 is asymmetric. We make this answer more precise by comparing their respective decomposition algorithms.

The SVD of $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ is given by

$$\mathbf{X}^{\mathrm{T}}\mathbf{Y} = \mathbf{G}\,\boldsymbol{\Delta}\mathbf{H}^{\mathrm{T}} = \delta_1\mathbf{g}_1\mathbf{h}_1^{\mathrm{T}} + \delta_2\mathbf{g}_2\mathbf{h}_2^{\mathrm{T}} + \cdots \tag{16}$$

where $\mathbf{G}$ and $\mathbf{H}$ are orthonormal matrices ($\mathbf{G}^{\mathrm{T}}\mathbf{G} = \mathbf{H}^{\mathrm{T}}\mathbf{H} = \mathbf{I}_{r'}$) of order $p \times r'$ and $m \times r'$ respectively, with $r' = \min(p, m)$ containing the left and right singular vectors $\{\mathbf{g}_a\}$ and $\{\mathbf{h}_a\}$ ($a = 1, \ldots, r'$) respectively, and $\boldsymbol{\Delta}$ is a diagonal matrix with the singular values on the diagonal arranged in decreasing order ($\delta_1 \geqslant \delta_2 \geqslant \cdots \geqslant \delta_{r'} \geqslant 0$). The SVD can be obtained by progressive deflation of $\mathbf{S}_0$ ($\equiv \mathbf{X}^{\mathrm{T}}\mathbf{Y}$) into matrices $\mathbf{S}_1, \mathbf{S}_2, \ldots$ by the formula[8,12]

$$\mathbf{S}_a = \mathbf{S}_{a-1} - \delta_a\mathbf{g}_a\mathbf{h}_a^{\mathrm{T}} \tag{17}$$

For comparison with PLS2 (equations (6)–(10)) we express $\mathbf{g}_a$ and $\mathbf{h}_a$ in terms of $\mathbf{X}$ and $\mathbf{Y}$. Because $\mathbf{G}$ and $\mathbf{H}$ are orthonormal, $\mathbf{X}^{\mathrm{T}}\mathbf{Y}\mathbf{H} = \mathbf{G}\,\boldsymbol{\Delta}\mathbf{H}^{\mathrm{T}}\mathbf{H} = \mathbf{G}\,\boldsymbol{\Delta}$ and analogously $\mathbf{Y}^{\mathrm{T}}\mathbf{X}\mathbf{G} = \mathbf{H}\,\boldsymbol{\Delta}$, so that

$$\mathbf{g}_a = \delta_a^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}\mathbf{h}_a = \mathbf{X}^{\mathrm{T}}\tilde{\mathbf{u}}_a \tag{18}$$

$$\mathbf{h}_a = \delta_a^{-1}\mathbf{Y}^{\mathrm{T}}\mathbf{X}\mathbf{g}_a = \mathbf{Y}^{\mathrm{T}}\tilde{\mathbf{t}}_a \tag{19}$$

with

$$\tilde{\mathbf{u}}_a = \delta_a^{-1}\mathbf{Y}\mathbf{h}_a \tag{20}$$

$$\tilde{\mathbf{t}}_a = \delta_a^{-1}\mathbf{X}\mathbf{g}_a \tag{21}$$

The deflation in SVD uses $\mathbf{g}_1$ and $\mathbf{h}_1$. Equation (18) shows that $\mathbf{g}_1$ is a (normalized) loading vector of the X-variables with respect to $\tilde{\mathbf{u}}_1$, the score vector that belongs to the Y-variables (equation (20)). Likewise, equation (19) shows that $\mathbf{h}_1$ is a (normalized) loading vector of the Y-variables with respect to $\tilde{\mathbf{t}}_1$, the score vector that belongs to the X-variables (equation (21)). Thus the SVD deflation is symmetric. The decomposition of $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ in PLS2 uses $\mathbf{p}_1$ and $\mathbf{c}_1$ (equation (7)). These are also loading vectors of the X-variables and Y-variables respectively. However, both these loading vectors pertain to the same X-component score vector $\mathbf{t}_1$ (equations (8) and (9)). Thus the deflation in PLS2 is asymmetric. From the PLS2 algorithm in the Appendix it follows that $\mathbf{g}_1 \propto \mathbf{w}_1$, so that $\tilde{\mathbf{t}}_1 \propto \mathbf{t}_1$ and $\mathbf{h}_1 = \mathbf{q}_1 \propto \mathbf{c}_1$. Further, notice[6] that $\| \mathbf{S}_1 \|^2 \leqslant \| (\mathbf{X}^{\mathrm{T}}\mathbf{Y})_1 \|^2$. In other words, SVD is geared to the least-squares approximation[13] of $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ by lower-rank matrices, whereas PLS2 is geared to the prediction of $\mathbf{Y}$.

APPENDIX: SIMPLIFIED KERNEL ALGORITHM FOR PLS2 REGRESSION

The following MATLAB* code starts from the cross-product matrices XtX ($=\mathbf{X}^{\mathrm{T}}\mathbf{X}$) and XtY ($=\mathbf{X}^{\mathrm{T}}\mathbf{Y}$) and finds the first 'dim' components.

```
ssqX = sum(diag(XtX))                       % total X-variance
for a = 1:dim                               % dim = # components
   [Q,D] = eig(XtY' * XtY)                  % Y-weights
   q = Q(:,find(diag(D) = = max(diag(D))))  % dominant eigenvector
   w = XtY * q                              % X-weights
   w = w/sqrt(w' * XtX * w)                 % rescale weights
   p = XtX * w                              % X-loadings
   c = XtY' * w                             % Y-loadings
   XtX = XtX - p * p'                       % X'X deflation
   XtY = XtY - p * c'                       % X'Y deflation
   W = [W,w]                                % store weights
   P = [P,p]                                % and
   C = [C,c]                                % loadings
end
R = W * inv(P' * W)                         % modified X-weights
B = R * C'                                  % regression coefficients
R2X = sum(P.^ 2)/ssqX                       % coeff. of determin. (X)
R2Y = (ones (1,m) * C.^ 2)/sum(sum(Y.^ 2))  % ibid                (Y)
```

REFERENCES

1. F. Lindgren, P. Geladi and S. Wold, *J. Chemometrics*, **7**, 45–59 (1993).
2. H. Martens and T. Næs, *Multivariate Calibration*, Wiley, Chichester (1989).
3. C. Paige and M. A. Saunders, *ACM Trans. Math. Softw.* **8**, 43–47, 195–209 (1982).
4. R. Manne, *Chemometrics Intell. Lab. Syst.* **2**, 283–290 (1987).
5. S. de Jong, *Chemometrics Intell. Lab. Syst.* **18**, 251–263 (1993).
6. A. Høskuldsson, *J. Chemometrics*, **2**, 211–228 (1988).
7. I. S. Helland, *Commun. Stat.—Simul. Comput.* **17**, 581–607 (1988).
8. A. R. Gourlay and G. A. Watson, *Computational Methods for Matrix Eigenproblems*, Wiley, London (1973).
9. M. O. Hill, *DECORANA*, Microcomputer Power, Ithaca, NY (1979).
10. R. F. Ling, *J. Am. Stat. Assoc.* **69**, 859–866 (1974).
11. M. R. B. Clarke, *Appl. Stat.* **20**, 206–209 (1971).
12. K. V. Mardia, J. T. Kent and J. M. Bibby, *Multivariate Analysis*, Academic, London (1979).
13. C. Eckart and G. Young, *Psychometrika*, **1**, 211–218 (1936).

*MATLAB syntax: sum, summation of columns (matrix) or sum of elements (vector); diag, main diagonal of matrix; *, matrix multiplication; [X, x], add a column x to matrix **X**; .^ 2, elementwise square; ones(1,m), row of *m* ones.