Sparse Partial Least Squares Regression for On-Line Variable Selection with Multivariate Data Streams

Brian McWilliams and Giovanni Montana*

Statistics Section, Department of Mathematics, Imperial College London, London, UK

Received 20 August 2009; revised 25 February 2010; accepted 4 March 2010
DOI:10.1002/sam.10074
Published online 12 May 2010 in Wiley InterScience (www.interscience.wiley.com).

Abstract: Data streams arise in several domains. For instance, in computational finance, several statistical applications revolve around the real-time discovery of associations between a very large number of co-evolving data feeds representing asset prices. The problem we tackle in this paper consists of learning a linear regression function from multivariate input and output streaming data in an incremental fashion while also performing dimensionality reduction and variable selection. When input and output streams are high-dimensional and correlated, it is plausible to assume the existence of hidden factors that explain a large proportion of the covariance between them. The methods we propose build on recursive partial least squares (PLS) regression. The hidden factors are dynamically inferred and tracked over time and, within each factor, the most important streams are recursively identified by means of sparse matrix decompositions. Moreover, the recursive regression model is able to adapt to sudden changes in the data generating mechanism and also identifies the number of latent factors. Extensive simulation results illustrate how the methods perform and compare with alternative penalized regression models for streaming data. We also apply the algorithm to solve a multivariate version of the *enhanced index tracking* problem in computational finance. © 2010 Wiley Periodicals, Inc. Statistical Analysis and Data Mining 3: 170–193, 2010

Keywords: data streams; dynamic regression; partial least squares; variable selection; index tracking

1. INTRODUCTION

Streaming data arise in several application domains, including web analytics [1], healthcare monitoring [2], and asset management [3], among others. In all such contexts, large quantities of data are continuously collected, monitored, and analyzed over time. When dealing with data streams, a common and important task consists of learning a regression function that explains the linear relationship between a number of incoming streams, regarded as predictive variables or covariates, and a number of co-evolving data streams, regarded as responses. In this paper, we envisage a real-time system that imports p input and q output data streams at discrete time points. The input data vector observed at time t is denoted by $x_t \in \mathbb{R}^{1\times p}$ where the subscript refers to the time stamp and the dimension p may be very large. The output streams are collected in a vector $y_t \in \mathbb{R}^{1\times q}$, which may also be very high-dimensional although q is generally much smaller than p. Our objective is to recursively estimate a regression function of form $y_t = f(x_t) + \epsilon_t$ where $f(\cdot)$ is linear in its parameters and each ϵ_t is an independent noise component. Our fundamental assumption is that, at any given time, only a few selected components of x_t, or variables, contain enough predictive power, and only those should be selected to build the regression model.

Our motivating application is an extension of the *index tracking* or *equity index replication* problem arising in the domain of computational finance. Index tracking is a passive investment strategy that consists of purchasing all or some of the assets which are contained in a market index. An index tracking portfolio is built by weighting the selected assets such that the portfolio optimally replicates the behavior of the index according to some performance measure. A tracking portfolio is then periodically rebalanced and its weights re-adjusted to achieve optimal performance over time. *Full replication* of the index can be obtained by investing in all the assets that comprise the index, for instance, all the 500 equities included in the computation of the S&P 500 market index. Although perfect replication can be obtained in this way, a full replication strategy involves high initial costs, and the portfolio is difficult to rebalance when changes are made to the index. An alternative approach is that of *partial replication*, in which an investment is made in only a small proportion of all the

Correspondence to: Giovanni Montana
(g.montana@imperial.ac.uk)

available assets, while attempting to replicate the performance of the index as closely as possible. In this case, the problem is twofold and involves the initial selection of a small number of assets to be included in the portfolio and then the estimation of the optimal portfolio weights.

Recently, the problem of asset selection and portfolio optimization have been cast in the framework of variable selection using penalized regression methods, where the adopted performance measure is the mean squared error. Penalized regression involves modifying the ordinary least squares solution by applying a restriction on the size of the regression parameter. The Lasso [4] is one such technique that imposes a restriction on the L_1 norm of the regression parameter which for a large enough penalty forces some of the regression coefficients to be exactly zero, effectively performing variable selection. In the case where the predictors are uncorrelated, the Lasso solution is a soft-thresholded version of the least squares solution. When independence cannot be assumed, the Lasso solution can be found efficiently using an algorithm such as least angle regression (LARS) [5] or coordinate descent [6]. Recently, DeMiguel *et al.* [7] has proposed to determine the portfolio weights as the coefficients from a linear regression and restricted the norm of the coefficients using the Lasso penalty to achieve a sparse portfolio. A similar study has been described by Brodie *et al.* [8], who also formulate the asset selection problem as a Lasso penalized regression and apply this portfolio to index tracking. These approaches assume that all the data are available for fitting the regression model, and no incremental learning is considered. Moreover, only one index can be tracked, whereas the problem of simultaneously tracking multiple indices, which can be formulated as a regression problem with multivariate responses, has not been investigated. The ability to track multiple indices is beneficial as it opens the possibility of identifying which assets are important in multiple separate markets and thus generate returns on more than one index using a single portfolio of assets common to several indices.

Formulated in terms of streaming data, where each stream represents the observed daily prices of a given asset, the *multivariate index tracking* problem consists of selecting a small number of predictive streams which optimally replicates multiple indices. Following DeMiguel *et al.* [7] and Brodie *et al.* [8], we approach this problem as a sparse regression problem. However, contrary to these approaches, we are interested in performing sparse regression with multiple responses and derive recursive solutions that are suitable in an on-line setting so that the regression model can be recursively updated at every time point. Further clarifications regarding this application and experimental results are found in Section 5.

There are a number of challenges arising in this setting which we intend to tackle and discuss in this paper. First, a decision has to be made on how to select the truly important predictive components of the input data streams that best explain the multivariate response in a computationally efficient manner. As mentioned before, we embrace a sparse regression approach where the unimportant variables are excluded from the model by forcing their coefficients to be exactly zero. Second, because the components of x_t may be highly correlated, variable selection arises in an ill-posed problem and special care is needed to deal with this difficulty. As will be clear later, we take a dimensionality reduction approach. Third, the relationship between input and output streams is expected to change quite frequently over time, with the frequency of change depending on the specific application domain and nature of the data. This aspect requires the development of adaptive methods that are able to deal with possible nonstationarities and the notion of *concept drift*, that is, the time-dependency of the underlying data generating process [9].

When input and output streams are high-dimensional, hence highly correlated, it is plausible to assume the existence of *latent factors* that explain a large proportion of the covariance between them. The latent factors are uncorrelated and, although not directly observed, may relate to a hidden physical quantity or underlying phenomenon. For instance, in computational finance, latent factor models have been extensively used to explain and model asset prices [10]. The methodology we propose builds on partial least squares (PLS) regression for the efficient estimation of such latent factors. By extracting orthogonal latent factors that explain a large proportion of the covariance between the covariates and the response, PLS overcomes the problems introduced by high-dimensional and collinear predictors.

Building on PLS regression based on a singular value decomposition (SVD), we propose an on-line learning algorithm that extracts the latent factors and tracks them over time in an adaptive fashion. So that within each factor, the algorithm incrementally selects the most important variables as new data points become available. To the best of our knowledge, although the problems of adaptively tracking latent factors and performing variable selection have been tackled separately in the context of on-line learning in high dimensions, little work has been carried out toward the development of a methodology that addressed all these issues in a unified framework.

The problem of tracking latent structures that explain important variation in time-varying data streams has been approached in several different ways in the literature. For instance, a number of approaches to on-line principal component analysis (PCA) have been proposed in the areas of image analysis [11] and data stream mining [12], among others. The problem of tracking latent factors in time has mostly been studied in the context of PCA in the field of

subspace tracking [13]. In several of these works, the eigenvector problem involved in a PCA is recast as a regression problem with the aim of minimizing the sum of squares between the matrix of covariates X and its low-dimensional reconstruction $v^{(1)}v^{(1)^{\mathrm{T}}}X$, where $v^{(1)}$ is the largest eigenvector of $X^{\mathrm{T}}X$. When the data arrives sequentially, $v^{(1)}$ can be estimated recursively using a modified recursive least squares (RLS) algorithm, as in Refs. [14] and [15], among others.

Unlike on-line PCA, not many incremental versions of PLS regression have been suggested in the literature. An adaptive PLS algorithm called recursive PLS was proposed by Dayal and Macgregor [16] for real-time chemical process control. More recently, an efficient tracking procedure was proposed within the locally weighted projection regression (LWPR) algorithm of Ref. [17] which is motivated by the problem of learning a model of inverse dynamics for real-time robotics. In LWPR, PLS regression is updated efficiently at each time point using previous solutions. However, the LWPR algorithm is only suitable in situations where the response is univariate.

Tracking and performing regression in the streaming data setting is also well studied with the most well-known technique being recursive least squares (RLS); see, for example, Ref. [18]. However, to date, the problem of selecting variables on-line has been somewhat less studied. To the best of our knowledge, only two relatively recent works address this issue within a penalized regression framework. One method for on-line Lasso by Kim *et al.* [19] updates the LARS solution as a new data point becomes available simply by performing a single iteration of the LARS algorithm. More recently, Anagnostopoulos *et al.* [20] developed an alternative approach to on-line Lasso based on RLS. They solve the Lasso problem using coordinate descent which they combine with an adaptive RLS algorithm that can adapt to changes in the data over time. Neither approach considers a multivariate response or the issue of multicollinearity among covariates which are two key issues addressed in our work.

The format of this paper is as follows. First, in Section 2, we briefly review multivariate linear regression and PLS regression with emphasis on situations where the number of factors which can be extracted simultaneously by the PLS algorithm is limited by the rank of the data matrix. In Section 2.1, we propose a method of overcoming this limitation using a technique similar to ridge regression which allows all PLS factors to be extracted simultaneously using only one SVD. We then present a new algorithm to perform efficient sparse PLS regression in Section 3. We achieve sparsification of the regression coefficients by means of a soft-thresholding rule in the computation of the SVD. This rule effectively applies a Lasso-like penalty although many other penalties could be easily used within the same framework. Then, in Section 3.2, an incremental and adaptive version of our sparse PLS algorithm called incremental sparse PLS (iS-PLS) is proposed for real-time applications. The final algorithm is based on the adaptive simultaneous iterations method for sequential updating of the eigenstructure of a covariance matrix [21]. This has the effect of introducing an adaptive behavior so that changes in the important variables can be tracked in a timely manner. Experimental results using both artificial and real data are presented in Sections 4 and 5 and conclusive remarks are found in Section 6.

2. SVD-BASED PLS REGRESSION

In this section, we motivate the PLS regression approach and describe a SVD-based algorithm that will be further elaborated on in the following sections. We consider a regression setting when both the design matrix, $X \in \mathbb{R}^{n\times p}$, and the response, $Y \in \mathbb{R}^{n\times q}$, are multivariate. Assuming centered data, the standard regression approach consists of fitting a multivariate linear regression (MLR) model, that is $Y = X\beta + \epsilon$, where $\beta \in \mathbb{R}^{p\times q}$ is a matrix of regression coefficients and $\epsilon \in \mathbb{R}^{n\times q}$ is the matrix of uncorrelated, mean-zero errors. However, a key result (see, for instance, Ref. [22]) shows that the columns, $[\beta_1, \ldots, \beta_q]$, of the matrix of regression coefficients β are in fact the coefficients of the univariate regressions of X on the individual response variables, $[y_1, \ldots, y_q]$, respectively. This implies that the least squares estimate for β is equivalent to performing q separate univariate regressions on the individual columns of Y. Therefore, the MLR solution contains no information about the correlation between variables in the response and performing MLR in situations where the variables are highly correlated is unsuitable because of high variance caused by inverting $X^{\mathrm{T}}X$ when the covariates are poorly conditioned because of multicollinearity.

A common solution to the multicollinearity problem involves constraining the rank of the matrix of coefficients so that $\mathrm{rank}(\beta) = t \leq p$, which is known as reduced-rank regression (RRR) [22]. If the rank parameter, t is decreased we effectively perform dimensionality reduction. As a special case of RRR we obtain principal component regression (PCR), which is a well-known dimensionality reduction and regression method based on a PCA. PCA finds the orthogonal directions of largest variance in the data which correspond to the largest eigenvectors of the covariance matrix, $X^{\mathrm{T}}X$. The latent factors are then obtained by projecting the covariates onto the eigenvectors. PCR overcomes the problem introduced by multicollinear variables by performing regression using the uncorrelated, low-dimensional latent factors. However, in certain situations, we may have reason to assume that the important variation in both the

covariates and the response is controlled by a handful of common factors. In these cases, PCR ignores the variation in the response and is not able to exploit this information to improve prediction. In PLS regression, instead of explaining the directions of largest variation in X, the latent factors correspond to the directions of largest covariance between the covariates and the response.

In what follows, we first introduce PLS in the simplest case where the response y is univariate and then provide the details for the general case of a multivariate response Y. The assumption underlying this form of PLS is that both X and the response y are generated by R latent factors in the following way:

$$X = \sum_{r=1}^{R} s^{(r)} b^{(r)\mathrm{T}} + D,\ y = \sum_{r=1}^{R} s^{(r)} w^{(r)\mathrm{T}} + e,$$

where $s^{(r)} \in \mathbb{R}^{n\times 1}$ are the latent factors and $b^{(r)} \in \mathbb{R}^{p\times 1}$ and $w^{(r)} \in \mathbb{R}^{1\times 1}$ are loading vectors of X and y, respectively. D and e are residuals for which we do not assume a distribution. The PLS algorithm iteratively finds R latent factors of X such that $s^{(r)} = X^{(r)}u^{(r)}$, where $u^{(r)}$ is a vector of weights corresponding to the direction of largest covariance between $X^{(r)}$ and y. The weights are estimated by solving the following optimization problem:

$$u^{(r)} = \max_{u} [\mathrm{cov}(X^{(r)}u, y)]^2 \text{s.t. } \|u\| = 1. \qquad (1)$$

In order to extract the full complement of latent factors, each one must be extracted sequentially. Once a factor has been extracted, a rank one deflation of the X matrix is usually performed by subtracting the contribution of the current factor from the data to give $X^{(r+1)} = X^{(r)} - s^{(r)}b^{(r)\mathrm{T}}$, where $X^{(1)} = X$, and a new iteration begins. The first latent factor explains most of the total covariance between X and y with successive latent factors explaining progressively less of the total covariance.

The PLS literature is extensive and many methods exist for extracting the latent factors (see, e.g. Ref. [23] for a recent review of PLS variants). Many of the various algorithms yield identical results for the first factor, but often they differ in computation of successive factors by how the data matrices are deflated. Because it is assumed that X and y are related through the latent factors, and the factors underlying X are a good predictor of y, the response can be rewritten as $y = XUW + e = SW + e$, where $U = [u^{(1)}, \ldots, u^{(R)}]$, $W = [w^{(1)}, \ldots, w^{(R)}]$, and $S = [s^{(1)}, \ldots, s^{(R)}]$. This leads to the regression model $\hat{y} = X\hat{\beta} + e$, where $\hat{\beta} = \tilde{U}\hat{W}$ are the estimated coefficients.

We first concentrate on the PLS-1 algorithm [24] which is a reinterpretation of the original nonlinear iterative partial least squares (NIPALS) algorithm introduced in Ref. [25] for a univariate response. The NIPALS algorithm finds the PLS weights by performing the power method for finding the largest eigenvector of the covariance matrix $X^{\mathrm{T}}yy^{\mathrm{T}}X$ [24]. For all values of $r = 1, 2, \ldots, R$, we define $M^{(r)} = X^{(r)\mathrm{T}}y$, that is, the covariance matrix between the covariates and response. PLS-1 finds the rth weight vector $u^{(r)}$ by solving the optimization problem in Eq. (1). This is equivalent to finding the solution for

$$\arg\max_{u} \left[u^{\mathrm{T}}M^{(r)}M^{(r)\mathrm{T}}u\right] \quad \text{s.t.} \quad \|u\| = 1,$$

that is, the normalized eigenvector corresponding to the largest eigenvalue of $M^{(r)}M^{(r)\mathrm{T}}$. This requires finding the first right singular vector of the SVD of $M^{(r)}$. The loading vectors for both y and X are estimated by performing univariate regressions of the latent factors onto y and $X^{(r)}$, respectively. After the extraction of the first factor, in order to extract subsequent factors, X must be deflated by subtracting the contribution of the current latent factor. The same procedure is then repeated until all required factors are extracted.

The above setting can be extended to situations where the response is multivariate so that

$$Y = \sum_{r=1}^{R} s^{(r)} w^{(r)\mathrm{T}} + E,$$

where $Y \in \mathbb{R}^{n\times q}$ and $w^{(r)} \in \mathbb{R}^{1\times q}$. In this case, we need to obtain

$$u^{(r)} = \max_{u} [\mathrm{cov}(X^{(r)}u, Y)]^2 \quad \text{s.t.} \quad \|u\| = 1. \qquad (2)$$

The latent factors and X and Y loading vectors, $s^{(r)}$, $b^{(r)}$ and $w^{(r)}$, respectively, are computed as before. In order to extract successive latent factors, the data matrix must be deflated by subtracting the contribution of the current latent factor as in the univariate case. Clearly, this is not very efficient because the SVD of $X^{(r)\mathrm{T}}Y$ must be recomputed at each iteration to find the subsequent PLS weight [26]. In order to extract R latent factors, the SVD must be computed R times with the computational cost of each SVD being $O(np^2)$. Therefore, our first step toward a computationally efficient implementation of PLS is to propose a SVD-based PLS algorithm that extracts the latent factors in a noniterative way.

2.1. Efficient PLS Regression for Multivariate Responses

In the previous section, we reviewed PLS as a regression technique with favorable properties when dealing with high-dimensional or ill-conditioned data. A key part of the PLS

algorithm is that to extract more than one latent factor, the data matrix must be deflated and new PLS weights must be computed using the SVD. The deflation step is necessary because often in regression problems, $\text{rank}(Y) < \text{rank}(X)$. In this case, the covariance matrix MM^{T} will be rank deficient and so the $[\text{rank}(X) - \text{rank}(Y)]$th to $\text{rank}(X)$th singular vectors will not explain any covariance between X and Y. In such cases, the number of PLS components that can be extracted without deflation will be limited to $\text{rank}(Y)$ [26].

Obtaining the PLS weights amounts to finding the principal eigenvector of the matrix $MM^{\mathrm{T}} = X^{\mathrm{T}}YY^{\mathrm{T}}X$. In many real situations, the number of response variables is fewer than the number of predictors so YY^{T} will cause MM^{T} to become rank deficient. This problem is generally solved by deflating the data matrix, that is, subtracting the contribution of the previously extracted latent factor. The next PLS weight can be extracted by computing the principal eigenvector of the new covariance matrix, $M^{(r)}M^{(r)^{\mathrm{T}}} = X^{(r)^{\mathrm{T}}}YY^{\mathrm{T}}X^{(r)}$. However, as mentioned in Section 2, to extract R latent factors using this method, R different SVD computations are required which is computationally expensive.

In this section, we introduce an efficient new algorithm for performing PLS using only one single SVD computation. We address this situation by noting that a similar problem arises in ordinary least squares fitting problems: when the matrix of covariates X is ill-conditioned, the covariance matrix $X^{\mathrm{T}}X$ is rank deficient and therefore cannot be inverted. This situation is generally resolved by applying a small positive constant to the diagonal of the covariance matrix which has the effect of reducing the variance in the solution by adding some bias. We use a technique similar to ridge regression to regularize MM^{T} by adding a small constant term to the covariance matrix, in the following way:

$$H = X^{\mathrm{T}}\left[\alpha I + (1-\alpha)YY^{\mathrm{T}}\right]X \tag{3}$$

where $0 \le \alpha \le 1$. It can be noticed that this has the effect of adding a small constant to the diagonal of YY^{T}. With this modification, it follows that $\text{rank}(\alpha I + (1-\alpha)YY^{\mathrm{T}}) = \text{rank}(X^{\mathrm{T}}X)$, and this prevents H from becoming rank deficient [27]. Rearranging Eq. (3), we obtain a new covariance matrix:

$$H = \alpha C + (1-\alpha)MM^{\mathrm{T}}, \tag{4}$$

where $C = X^{\mathrm{T}}X$. In this form, it can be noted that H is a weighted sum of the covariance matrix of X and the covariance matrix of X and Y. The parameter α has a clear interpretation: when $\alpha = 0$, this yields regular PLS; when $\alpha = 1$, this yields a principal components regression (PCR); when $0 < \alpha < 1$, we obtain a trade-off between PLS and PCR. Therefore, using this new covariance matrix can be thought of as biasing the PLS solution (which takes into consideration the response in deriving the latent factors and therefore yields better predictive performance) toward the PCR solution (which does not use the response in deriving the latent factors). Because MM^{T} is larger than C, provided α is not too large, the contribution of MM^{T} to H is bigger than the contribution of C. Therefore, the specific value chosen for α, as long as this is not too close to 1, will not have any noticeable effect on the solution; see also Section 4.4 for further comments and a sensitivity analysis. However, this simple modification will ensure that the covariance matrix H is always full rank and the PLS weights can then be obtained in one single step by solving the following optimization problem:

$$\tilde{U} = \arg\max_{U}\left(U^{\mathrm{T}}HU\right) \text{ s.t. } \|U\| = 1, \tag{5}$$

so that $\tilde{U} = [\tilde{u}^{(1)}, \ldots, \tilde{u}^{(R)}]$ are the first R eigenvectors of H. The latent factors, S, are then computed as $X\tilde{U}$. The corresponding Y-loadings are $\hat{W} = \left(S^{\mathrm{T}}S\right)^{-1}S^{\mathrm{T}}Y$ and the final PLS regression coefficients are $\hat{\beta} = \tilde{U}\hat{W}$.

A further computational advantage stems from the fact that it is no longer necessary to compute the X-loadings which were only required to deflate X. Removing the necessity to perform $R-1$ additional SVD computations in the off-line case yields a saving in computation time of $O(Rnp^2)$. Moreover, reducing the problem to a single SVD enables us to propose an efficient algorithm for on-line learning.

3. SPARSE PLS REGRESSION USING SPARSE MATRIX FACTORIZATION

3.1. Off-Line Learning

We have previously introduced an efficient method of performing PLS regression which finds the PLS weights by means of a single SVD computation. We now observe that the PLS weights can be made sparse by using a penalized form of the SVD which leads to a novel and efficient method of variable selection based on the PLS framework. Sparse matrix factorization methods have recently been introduced in Refs. [28] and [29]. Specifically, Shen and Huang [28] formulates a sparse matrix factorization using the best low-rank approximation property of the SVD; this is achieved by reformulating the SVD problem as a regression where the aim is to minimize the sum of squared errors between X and its best low-rank approximation. Witten *et al.* [29] propose a more general framework for

sparse matrix factorization which includes the sparse SVD method of Ref. [28] as a special case.

Recently, a sparse PLS algorithm based on sparse SVD has been proposed in Ref. [30] which computes the PLS weights using the standard PLS algorithm described in Section 2. Because the covariance matrix used to extract the PLS weights is not regularized, the problems of rank-deficiency described in Section 2.1 are still present and so R separate SVD computations are required to extract all R latent factors. In this section, we use sparse SVD to achieve an efficient variable selection algorithm within the PLS framework described in the previous section.

Sparse SVD is motivated by first rewriting the SVD as a regression problem. We calculate H as in Eq. (4) and define the SVD of $H = UDV^{\mathrm{T}}$ where $U = [u^{(1)}, \ldots, u^{(n)}] \in \mathbb{R}^{n\times n}$ and $V = [v^{(1)}, \ldots, v^{(p)}] \in \mathbb{R}^{p\times p}$ are orthogonal matrices. $D \in \mathbb{R}^{n\times p}$ is a diagonal matrix whose entries are the singular values of H. The PLS criterion in Eq. (5) can be written as a regression whereby the criterion to be minimized is the residual sum of squares between H and its low-rank approximation, as follows:

$$\min_{u,v} \left\| H - \tilde{u}\tilde{v}^{\mathrm{T}} \right\|^2, \tag{6}$$

where $\tilde{u}$ and $\tilde{v} \in \mathbb{R}^{p\times 1}$ are the estimates of $u^{(1)}$ and $v^{(1)}$ and are restricted to be vectors with unit norm so that a unique solution may be obtained. This can be solved iteratively for $\tilde{u}$ and $\tilde{v}$ using ordinary least squares in the following way: minimizing Eq. (6) for a given v we obtain $2Hv - 2\tilde{u}v^{\mathrm{T}}v = 0$ but because we impose the constraint that $\|v\| = 1$ and $\|\tilde{u}\| = 1$ we obtain $\tilde{u} = Hv/\|Hv\|$. For a given $\tilde{u}$, we obtain a similar result for $\tilde{v}$, namely, $\tilde{v} = H\tilde{u}/\|H\tilde{u}\|$. It is known that the product of the first left and right singular vectors, $u^{(1)}v^{(1)}$, is the best rank one approximation of H [28]. Therefore, the solution to problem (6) is equivalent to setting $\tilde{u} = u^{(1)}$ and $\tilde{v} = v^{(1)}$. Because Eq. (6) is an ordinary least squares problem, we obtain a sparse solution by imposing a penalty on $\tilde{u}$, as follows:

$$\min_{\tilde{u},\tilde{v}} \left\| H - \tilde{u}\tilde{v}^{\mathrm{T}} \right\|^2 + p(\tilde{u}) \quad \text{s.t.} \quad \|\tilde{v}\| = 1, \tag{7}$$

where $p(\cdot)$ is one of the number of penalty functions which restrict the size of the coefficients, as mentioned in Section 2. In this work, we concentrate on the Lasso penalty, which places a restriction on the L_1 norm of $\tilde{u}$. This amounts to the following optimization problem:

$$\min_{\tilde{u},\tilde{v}} \left\| H - \tilde{u}\tilde{v}^{\mathrm{T}} \right\|^2 + \gamma^{(1)} \|\tilde{u}\|_1 \quad \text{s.t.} \quad \|\tilde{v}\| = 1, \tag{8}$$

where $\gamma^{(1)}$ is a parameter which controls the sparsity of the solution. If $\gamma^{(1)}$ is large enough, it will force some variables to be exactly zero. The problem of Eq. (8) can be solved in an iterative fashion by first setting $\tilde{u} = u^{(1)}$ and $\tilde{v} = v^{(1)}$ as before. The Lasso penalty can then be applied as an iterative component-wise soft-thresholding operation on the elements of $\tilde{u}$ (see, for instance, Ref. [6]). The solutions then are given by

$$\tilde{u}^* = \operatorname{sgn}(Hv)\left(|Hv| - \gamma^{(1)}\right)_+, \qquad \tilde{v}^* = H\tilde{u}^* / \left\| H\tilde{u}^* \right\|.$$

We then set $\tilde{u} = \tilde{u}^*$ and $\tilde{v} = \tilde{v}^*$ and iterate until $\|\tilde{u}^* - \tilde{u}\| < \tau$, where τ is an arbitrarily small constant. Assuming known sparsity parameters $\gamma^{(1)}, \ldots, \gamma^{(R)}$, the procedure above allows all the PLS weight vectors to be extracted and made sparse at once without the need to recompute an SVD for each dimension. The remaining steps of the PLS algorithm proceeds as before, using the newly calculated sparse weights. This leads to latent factors $S = XU$, and the matrix of Y loadings is $W = (S^{\mathrm{T}}S)^{-1}S^{\mathrm{T}}Y$. The final sparse PLS regression coefficients are $\hat{\beta} = UW$.

The number R of PLS components to be retained and the sparsity parameters $\gamma^{(r)}, r = 1, \ldots, R$ must be preselected. As usually done in off-line PLS, we suggest to determine the initial R using historical data and K-fold cross-validation (CV) [30]. Cross-validation involves constructing K training and test sets from a single data set, each training set consisting of $(K-1)/K$ samples from the data and the testing set consisting of the remaining $1/K$ samples. For each $k = 1, \ldots, K$ training set, we compute the S-PLS regression coefficients and obtain a prediction error using the kth testing set. The mean cross-validation error is

$$\mathrm{CV} = \frac{1}{K}\sum_{k=1}^{K}(Y_k - X_k\beta_{-k})^2,$$

where $-k$ denotes parameter β_{-k} was estimated using the kth training set. Typically, only a small number of PLS components are selected as the first few eigenvectors of H explain most of the covariance between X and Y. In Section 3.2, we discuss how the proposed on-line algorithm is able to track changes in both the number of important latent factors and their weights, over time.

The other parameters to be selected are the $\gamma^{(r)}, r = 1, \ldots, R$, which control the amount of sparsity in the input. In several applications, the number of variables to include in the model is controlled by the user because this yields results that can be more easily interpreted; for instance, in genomics applications [29] and in some financial applications such as index tracking and portfolio optimization (see Section 5). When the user wishes to select a specific number of variables for the rth PLS component, the corresponding $\gamma^{(r)}$ can be obtained by finding the roots of the following function related to the following

thresholding function

$$f[\gamma^{(r)}] = \sum_{i=1}^{p} \mathbb{I}\left[\text{sgn}(u_i^{(r)})(|u_i^{(r)}| - \gamma^{(r)})_+ > 0\right] - \theta,$$

where $\mathbb{I}$ is an indicator function which finds the non-zero elements of u after the threshold has been applied. The desired value of $\gamma^{(r)}$ is such that $f(\gamma^{(r)}) = 0$. In other applications, the objective is to identify as many important variables as possible to achieve accurate predictions. When the degree of sparsity has to be inferred from historical data, K-fold cross-validation can again be used before on-line learning, then the optimal value of $\gamma^{(r)}, r = 1, \ldots, R$, minimizes the cross-validated prediction error. We next discuss how our on-line algorithm can automatically track the most important latent factors and, within each, the most important variables.

3.2. On-Line Learning

In an on-line learning setting, we no longer assume we have access to the full data matrices $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^{n\times q}$. Instead the data stream arrives sequentially at each time point, t, as $x_t \in \mathbb{R}^{1\times p}$. Similarly, the response is observable only at discrete time points as $y_t \in \mathbb{R}^{1\times q}$. The use of on-line methods for prediction in a streaming data setting is desirable as off-line methods require a full model fit to be performed every time a new data point arrives which can be computationally expensive.

There are three important issues involved in performing sparse and adaptive incremental learning with PLS regression: (i) latent factors must be recursively computed because they may be changing with time; (ii) important predictors must be recursively extracted from the most important latent factors; (iii) adaptation to changes in the data generating mechanism must be data-driven. In this section, we propose an on-line learning procedure which combines all three of the above criteria in the context of sparse PLS. We call the resulting algorithm incremental sparse PLS (iS-PLS). To our knowledge, this is the first such method which combines tracking of latent factors with variable selection in an adaptive fashion for data streams. Operating on streaming data also offers some computational advantages. Because each observed data vector is of rank one, certain operations involved in updating the PLS solution at each time point are simplified somewhat compared with performing PLS on the full $(n \times p)$ data matrix. For instance, the matrix of latent factors is computed as $S = XU \in \mathbb{R}^{1\times R}$. This means the matrix inversion required for the computation of the Y-loading matrix reduces to a division by a scalar at each time point.

Before moving on to the description of the algorithm, we state two assumptions about the nature of the data stream:

(A1) The number of important latent factors underlying the data as well as their weights can evolve over time.

(A2) The important variables to be retained within each latent factors can evolve over time, but their number does not change.

Assumption (A1) is required to allow for time-dependent changes in the covariance structures of the data streams. Assumption (A2) also specifies a time-varying functional relationship between the input and the output. However, to simplify the estimation procedure, we impose that the number of important variables to be extracted from each latent factor does not change over time; that is, each parameter $\gamma^{(r)}, r = 1, \ldots, R$ is time-invariant. Potential solutions for releasing this constraint are discussed in Section 6.

In order to track the latent factors, we must first compute the PLS weights which are found as a result of solving the optimization problem (5). In the on-line setting, this amounts to updating the SVD of the time-varying covariance matrix, H_t. When a new data point x_t and its corresponding response y_t arrives, we update the individual covariance matrices as follows:

$$C_t = \lambda C_{t-1} + x_t^{\mathrm{T}} x_t, \quad M_t = \lambda M_{t-1} + x_t^{\mathrm{T}} y_t, \qquad (9)$$

where $0 \le \lambda \le 1$ is a forgetting factor which downweights the contribution of past data points. We elaborate further on the use of the forgetting factor in Section 3.2.1. We then construct the PLS covariance matrix H_t of Eq. (4), by summing the weighted PCA and PLS covariance matrices C_t and $M_t M_t^{\mathrm{T}}$, which leads to:

$$H_t = \alpha C_t + (1-\alpha) M_t M_t^{\mathrm{T}}. \qquad (10)$$

A problem we face with applying the sparse PLS algorithm to streaming data consists in updating the SVD of H_t recursively. Because H_t is a weighted sum between two covariance matrices, we are unable to find its eigenvectors using a standard RLS method. RLS requires as input the current estimate of the inverse covariance matrix and the new data observation, whereas we essentially only have access to a time-varying covariance matrix, H_t. We resolve this issue by means of algorithms derived from the power method (see, e.g. Ref. [31]), an iterative procedure used to compute the principal eigenvector of a symmetric matrix. As such, the power method can be used in situations when only a covariance matrix is available. The first iteration of the power method proceeds by initializing u_0 as a column vector with unit norm and then multiplying the covariance matrix H by u_{t-1} as $u_t = H u_{t-1}$. After each iteration, u_t is re-normalized to have unit norm. By repeating this procedure, u_t converges to the primary eigenvector of H.

Further eigenvectors may be extracted using a multivariate extension of the power method called simultaneous iterations method (SIM) [31]. The SIM algorithm proceeds by initializing $U_0 = [u^{(1)}, \ldots, u^{(R)}]$ as the identity matrix and at each time point, the estimate of the eigenvectors of the covariance matrix, H, are updated by performing one power iteration as $Q = HU_{t-1}$. Without further modification, each individual sequence $Hu^{(r)}$ will converge to the primary eigenvector therefore, after each iteration, each column of Q must be orthogonalized with respect to the other columns to ensure that they converge to different eigenvectors. This requirement is necessary as the true eigenvectors of H form an orthogonal basis. The updated estimate of the eigenvectors is obtained by $U_t = \text{orth}(Q)$, where orth(Q) is any orthogonalization procedure which causes the columns of the matrix Q to be mutually orthogonal. This ensures the columns of U_t will converge to different eigenvectors of H. Specifically, in our implementation we use the Gram–Schmidt orthogonalization procedure which has a computational complexity of $O(pR^2)$ [31].

In order to extract and track the eigenvectors of H in situations where the data generating mechanism is changing, we use the adaptive simultaneous iterations method (SIM) algorithm [21]. Adaptive SIM replaces the covariance matrix H with its time-varying estimate at time, t, H_t as in Eq. (10). The estimates of the eigenvectors at time t, U_t are then updated by performing a single iteration of the SIM algorithm. At each time t, once the weight vectors U_t have been updated, they are made sparse using a modified version of the iterative regularized SVD algorithm used for sparse PLS in Section 3. Because our algorithm is on-line and the solution is updated when a new data point arrives, we no longer iteratively apply the thresholding operation and instead apply it directly to the sparse estimate of the eigenvector found at the previous time point. When the data stream is stationary, this procedure quickly converges to the optimal solution as we effectively perform the iterations in time; simulation studies showing that the convergence takes place are presented in Section 4. In the case of a regime change, the proposed algorithm is able to quickly adapt to changes and quickly converges to the new solution (see Sections 3.2.1 and 4).

The final steps of the PLS algorithm proceed as in the off-line case. The latent vectors S are computed as $S = XU$. Because the number of observations at each update is effectively one, S will be an R-vector and the Y-loadings can be computed as $W = Y^{\mathrm{T}}S/(S^{\mathrm{T}}S)$. The sparse PLS regression coefficients are $\hat{\beta} = UW$ so that the regression estimate at time t is $\hat{y}_t = x_t UW$. Algorithm () details the resulting iS-PLS procedure in full.

A few comments about the initialization of the algorithm are in order. We set $U_0 = \left[u_0^{(1)}, \ldots, u_0^{(R)}\right] = I_{p\times R}$ to ensure that the initial estimates of the eigenvectors are mutually orthogonal. We also initialize $m_0 = 0$ and $C_0 = 0$. In the off-line case, the complexity introduced by the penalization function is $O(Rnp)$. In the on-line case, we operate only on a single data point at a time this reduces the complexity of the penalization function at each time point to $O(Rp)$.

Initialize $U_0 = I$, $m_0 = 0$, $C_0 = 0$;
Data: Input x_t and output y_t at time t
Result: Sparse regression coefficients β_t at time t
$\lambda_t \longleftarrow \texttt{selfTune}(x_t, y_t, \beta_{t-1})$ (Algorithm 4);
$m_t \longleftarrow \lambda_t m_{t-1} + x_t^{\mathrm{T}} y_t$;
$C_t \longleftarrow \lambda_t C_{t-1} + x_t^{\mathrm{T}} x_t$;
$H_t \longleftarrow \alpha C_t + (1-\alpha) m_t m_t^{\mathrm{T}}$;
for $r \leftarrow 1$ **to** R **do**
  $a^{(r)} \leftarrow H_t u^{(r)}$;
  $q^{(r)} \leftarrow \left[I_{p\times p} - \sum_{k=1}^{r-1} u^{(k)} u^{(k)\mathrm{T}}\right] a^{(r)}, \quad q^{(1)} \leftarrow a^{(1)}$;
  $u^{(r)} \leftarrow q^{(r)} / \left\|q^{(r)}\right\|$;
  $\gamma^{(r)} \leftarrow \texttt{findRoot}(u^{(r)})$;
  $u^* \leftarrow \text{sgn}\left(u^{(r)}\right)\left(|u^{(r)}| - \gamma^{(r)}\right)_+$;
  $u^* \leftarrow \frac{u^*}{\|u^*\|}$;
  $u_t^{(r)} \leftarrow u^*$;
end
$s \leftarrow xU_t$;
$W \leftarrow \frac{ys}{s^{\mathrm{T}}s}$;
$\beta_t \leftarrow U_t W^{\mathrm{T}}$;

Algorithm 1: The iS-PLS algorithm.

3.2.1. *Adaptive behavior using self-tuning forgetting factors*

The problem of adapting the learning algorithm to changes in the data has often been addressed in the context of recursive least squares (RLS) [18]. RLS is an efficient method of updating a least squares estimate as new data points become available. One motivation behind RLS can be seen by factorizing the least squares solution $\beta = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$ into two constituent quantities: the covariance matrix, $C = X^{\mathrm{T}}X$, and the covariance between the data and the response, $M = X^{\mathrm{T}}y$. Both these quantities can be updated efficiently as a new data point becomes available using *forgetting factors* as in Eq. (9). In its simplest form, a forgetting factor downweights the contribution of past data by a constant, $0 \leq \lambda \leq 1$, so that in the case where the data generating mechanism is changing, the newer relevant data contributes more to the solution than older irrelevant data. In the case where the forgetting factor, $\lambda = 1$, no discounting of past data takes place and when

$\lambda = 0$ the effective sample size is reduced to the current data point only. At time point t, the ith observation is weighted by λ^{t-i} which results in an exponential forgetting of past data by the factor $e^{(t-i)\log\lambda}$ [32].

The choice of an appropriate forgetting factor has been addressed in Ref. [32], among others, who suggest setting λ between 0.98 and 0.995 so that the RLS algorithm is able to accurately track slowly changing data. However, if the data changes quickly, better tracking can be achieved if λ is updated incrementally in response to these changes. Several schemes for *self-tuning* the forgetting factor have been suggested in the literature; for example, Haykin [18] proposes a gradient descent method which at each time point adjusts the forgetting factor in the direction which minimizes the residual sum of squares by some small constant value.

The motivation for introducing adaptive behavior within iS-PLS is to allow us to detect changes in the latent factors underlying the data by tracking some measure of the performance of the algorithm and noticing when there is a decrease in performance signaling a change. When a change is detected, the forgetting factor is modified so that the currently computed latent factors are discarded and the soft-thresholding algorithm starts recursively computing the new solution starting from the current data point.

In order to introduce such adaptive behavior, we have adapted the self-tuning forgetting factor algorithm proposed in Ref. [33] to be used within our iS-PLS algorithm. This procedure for self-tuning of the forgetting factor is motivated by the aim to correctly identify the true least squares regression coefficients and additive noise when the algorithm converges. This is achieved by formulating an expression for the forgetting factor in terms of the ratio between two error quantities estimated using RLS: the variance of the *a priori* error and the variance of the *a posteriori* error at each time point. The *a priori* error, defined as $e_t = y_t - x_t\beta_{t-1}$, is the prediction error of the current time point using the previously estimated coefficient, whereas the *a posteriori* error, defined as $\epsilon_t = y_t - x_t\beta_t$, is the residual error using the regression coefficient estimated at the current time point. The self-tuning forgetting factor, λ_t, can then be updated at each time point as:

$$\lambda_t = \frac{\sigma_q \sigma_\epsilon}{\sigma_e - \sigma_\epsilon}, \tag{11}$$

where σ_ϵ^2 is the variance of the *a posteriori* error and σ_e^2 is the variance of the *a priori* error. σ_q^2 is the variance of the quantity $q_t = x_t C_t^{-1} x_t^{\mathrm{T}}$. When the data is stationary or changing slowly, the difference between the *a priori* error and the *a posteriori* error is small. In this case, the numerator dominates the ratio in Eq. (11) which leads to a larger value of λ_t. In the case where the data is non-stationary, the difference between the *a priori* error and the *a posteriori* error will increase. This causes the denominator to dominate the ratio in Eq. (11) which leads to a smaller value of λ_t. Because inverting C_t at each time point is a computationally expensive operation, the Sherman–Morrison formula [34] can be used instead to efficiently obtain C_t^{-1} as each new data point arrives. In order to ensure λ_t remains between 0 and 1, we take $\lambda_t = \min\{\lambda_t, 0.999\}$. All the estimates of the noise variances featuring in Eq. (11) can be updated recursively using the following equations: $\sigma_{q,t}^2 = a\sigma_{q,t-1}^2 + (1-a)q_t^2$, $\sigma_{e,t}^2 = a\sigma_{e,t-1}^2 + (1-a)e_t^2$ and $\sigma_{\epsilon,t}^2 = b\sigma_{\epsilon,t-1}^2 + (1-b)e_t^2$, where $0 < a < 1$ and $a < b < 1$ are constant terms which determine the effective sample size for the recursive noise estimates.

The parameters a and b are selected so that the system noise is estimated using a relatively long exponential window, as suggested in Ref. [35]. The *a priori* error is the current prediction error and is tracked recursively using a short exponential time window which assigns small weights to previous estimates; in this way, the *a priori* error is sensitive to sudden changes in the current prediction error. Accordingly, the algorithm is able to adjust the forgetting factor. Assuming stationary data, in the limit $t \to \infty$, the *a posteriori* error is the true system noise when the estimate of β has converged to its true value. In this limit, the *a priori* error tends toward the *a posteriori* error. On the basis of this observation, we can estimate the *a posteriori* error as the *a priori* error over a longer exponential window to provide a stable estimate of the error which will not be affected by sudden changes in the current prediction error. It can be seen that the hyper-parameters a and b control how quickly the error estimates converge to their true values; provided that $b > a$, their specific values do not greatly affect the sensitivity of the self-tuning algorithm after the convergence period. If $a = b$, then $\sigma_{e,t}^2 = \sigma_{\epsilon,t}^2$ which implies the data is stationary and so λ_t is prevented from adapting. If $a > b$, the error estimates will not converge to their true values and the algorithm will attempt to continuously adjust the forgetting factor.

In our experiments, we set $a = 0.5$ which is a short exponential window allowing $\sigma_{e,t}^2$ to accurately track the *a priori* error; we also set $b = 0.9$ thus ensuring that $\sigma_{\epsilon,t}^2$ will converge to the *a posteriori* error. Experimental results in Section 4 show that the performance of the algorithm is not affected greatly by the specific choice of these parameters as long as the constraints are obeyed. We modify the self-tuning forgetting factor algorithm for use in the iS-PLS algorithm by computing e_t and ϵ_t using the PLS regression coefficients. The self-tuning forgetting factor is described in full in Ref. [33].

3.2.2. *Detecting changes in the number of important latent factors*

The weights determining the PLS latent factors can be time-dependent and the algorithm presented so far, coupled with an adaptive forgetting factor, is able to detect and adapt to changes over time. According to our assumption (A1), the number of the most important factors that are being retained for prediction is also allowed to change over time. As pointed out before, off-line cross-validation over a training period can be generally used to obtain an initial value R_0, prior to on-line learning.

In this section, we describe how the algorithm adjusts R_t on-line. At each time step, we suggest to recursively update the mean prediction errors

$$E_t^{(r)} = \lambda_t E_{t-1}^{(r)} + \left(y_t - \hat{y}_t^{(r)}\right),$$

where $\hat{y}_t^{(r)}$ is the prediction of y_t obtained at time $t-1$ and r indicates how many latent factors are retained. In our procedure, we use $r = R_t - 1, R_t, R_t + 1$ to determine when to decrease or increase the current value R_t by one. It can be noted that past errors are included in $E_t^{(r)}$, but these are downweighted using the adaptive forgetting factor while more weight is given to the current prediction error term.

The error ratios $Q_t^{(R_t-1)} = E_t^{(R_t-1)}/E_t^{(R_t)}$ and $Q_t^{(R_t+1)} = E_t^{(R_t+1)}/E_t^{(R_t)}$ can then be computed. Values of these ratios above 1 indicate that the current R_t is appropriate and should not be changed. Conversely, values below 1 suggest that a change in R_t is needed, as this adaptation will improve the prediction performance. Accordingly, we define two truncated error ratios, $A_t^{(R_t-1)} = \min(1, Q_t^{(R_t-1)})$ and $A_t^{(R_t+1)} = \min(1, Q_t^{(R_t+1)})$, and trigger a change in R_t when either one deviates substantially away from one. Extreme deviations in $A_t^{(R_t-1)}$ and $A_t^{(R_t+1)}$ that trigger a change in R_t are detected by using a cumulative sum (CUSUM) approach for change detection as commonly used in statistical process control; see, for instance, Ref. [36]. Monte Carlo simulations presented in Section 4.6 demonstrate that this procedure works well in practice. It can be interpreted as an incremental approximation to leave-one-out cross-validation using the past data as training samples and the new data arriving at time t as test data.

4. EXPERIMENTAL RESULTS WITH SIMULATED DATA

In this section, we report on simulation experiments designed to demonstrate the performance of the sparse PLS regression algorithm as variable selection method in both the off-line case, where all the data has been observed, and the on-line case, where the data is assumed to arrive sequentially at discrete time points. Both univariate and multivariate responses will be considered.

4.1. Ability to Track the Important Explanatory Variables

First, we propose a simulation setting whereby groups of explanatory variables are generated from three distinct latent factors. In this setting, we impose that, each time step, the response depends on only two out of the three existing latent factors, which we call here the *active* factors. This is obtained by setting the regression coefficients of the variables corresponding to the remaining group of variables, associated with the *inactive* factor, exactly to zero. The non-zero regression coefficients associated with the active variables are simulated so that one of the two sets of variables is more strongly correlated with the response than the other. A similar simulation framework has been described in Ref. [37].

A more precise description of the simulation scheme follows. In order to generate data that are evolving over time, the three hidden factors are assumed to follow an autoregressive (AR) process of first order so that factor j is given by $F_{t,j} = \delta_j F_{t-1,j} + \epsilon_{t,j}$ with $t = 2 \ldots, 400$ and starting with an arbitrary initial value at $t = 1$. This is performed independently for each j, with $j = 1, 2, 3$. The parameter δ_j is the autoregressive coefficient for factor j, and we use $\delta_1 = 0.1$, $\delta_2 = 0.4$, $\delta_3 = 0.2$. The error terms in each factor follow a normal distribution with different means but same variance. Each explanatory variable is then generated as $x_{t,i} = F_{t,j} + \eta_{t,i}$, where $x_{t,i}$ indicates the values of data stream i at observation t, for $t = 1, \ldots, 400$ and $i = 1, \ldots, 300$. Here η_t is a standard normal variate. The index j indicates that each stream depends only on a given latent factor. Specifically, we create three groups of data streams by collecting the indices representing the variables into three non-overlapping sets, $\mathcal{F}_1 = \{1, \ldots, 100\}$, $\mathcal{F}_2 = \{101, \ldots, 200\}$, and $\mathcal{F}_3 = \{201, \ldots, 300\}$. Finally, the univariate response is simulated as

$$y_t = \sum_{j=1}^{2} \sum_{i \in \mathcal{F}_j} \beta_i x_{t,i} + \epsilon_t,$$

where the regression coefficients in the index sets $\mathcal{F}_1$ and $\mathcal{F}_2$ are sampled from normal distributions with means 10 and 5, respectively, and with common variance. Using the simulated data streams, we are able to show that the off-line sparse PLS regression procedure accurately selects the correct active variables. In the on-line case, for which we propose an incremental soft-thresholding procedure, we are also able to demonstrate how, when the underlying data is stationary, the on-line solution converges to the

off-line solution. The sparsity parameter, which determines how many variables are included in the regression model, is selected using the 10-fold cross-validation procedure described in Section 3.

On the basis of a Monte Carlo simulation study involving 500 data sets, the cross-validated mean number of active variables in the first component turned out to be 105.37 with a standard error of 19.50. We also compared the performance of our algorithm with that of two variable selection methods: standard Lasso regression (using the LARS algorithm of Ref. [5]) and sparse PCA (S-PCA) using the sparse SVD method of Ref. [28] described in Section 3. Figure 1 compares the receiver operating characteristic (ROC) curve obtained by S-PLS with that of LARS and S-PCA. As before, we report on average results based on a Monte Carlo simulation consisting of 500 simulated data sets. The ROC curve plots the sensitivity against 1−specificity. The sensitivity measure is defined as the proportion of correctly selected variables, whereas the specificity measure is defined as the proportion of variables which are correctly assigned a zero coefficient and excluded from the model (i.e. 1−specificity can be thought of as the false positive rate). It can be seen that S-PLS (solid line) performs better than LARS (dashed line) in cases, such as ours, where the explanatory variables are highly correlated. As expected, S-PCA (dot-dashed line) exhibits zero sensitivity for a portion of the curve. This is because S-PCA is selecting the variables in X whose projection explains a large proportion of the input variance; by construction, these variables are not those most correlated with the output. More generally, we expect S-PCA to exhibit much lower sensitivity than S-PLS except in the case where the variables associated with largest variance in the input are also able to explain a large proportion of variability in the output.

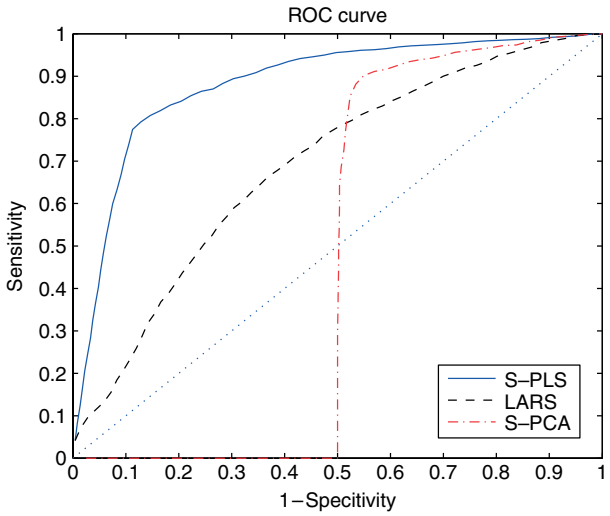

Fig. 1 Receiver operating characteristic (ROC) curve obtained by S-PLS, LARS, and S-PCA, reported are averages for a Monte Carlo simulation of 500 runs. For the simulated data, S-PLS (solid line) has a larger sensitivity for smaller values of specitivity than LARS (dashed line) which means it selects the correct variables and selects fewer incorrect variables. In the portion where S-PCA (dot-dashed line) has zero sensitivity, it is selecting the variables in X with the largest variance which by design are not the variables which are most correlated with the response.

4.2. Convergence of the Incremental Soft-Thresholding Update

In order to test the convergence of the iS-PLS algorithm to the off-line algorithm, we consider the same simulation setting introduced in Section 4.1 and assume that all observations arrive sequentially. Figure 2 shows the result of a Monte Carlo simulation consisting of 500 runs of the on-line iS-PLS algorithm with a forgetting factor $\lambda = 1$ (because the data stream is stationary and no forgetting is required). The shaded area shows the Monte Carlo error. It can be seen that the sensitivity of the on-line algorithm reaches its maximum value after observing 25 data points. These results, as well other extensive experimental results (not shown here), suggest that, in the presence of stationary data, the iS-PLS solution converges to the solution found by iterative soft-thresholding in an off-line setting only after a brief learning period. In the next section, we show how the on-line algorithm is also able to adapt to changes.

4.3. Ability to Adapt to Changes

In order to test the adaptive behavior of the iS-PLS algorithm in high dimensions, we generated a non-stationary output by introducing time-dependent regression coefficients, according to the following scheme: $\beta_{t,i}$, for $t = 1, \ldots, 100$, is drawn from a $N(10, 0.25)$ when $i \in \mathcal{F}_1$ and from a $N(5, 0.25)$ when $i \in \mathcal{F}_2$, and is set to zero otherwise. All the non-zero coefficients in the two groups of active variables are swapped at $t = 101$ so that $\beta_{t,i}$, for $t = 101, \ldots, 300$, is drawn from a $N(5, 0.25)$ when $i \in \mathcal{F}_1$ and from a $N(10, 0.25)$ when $i \in \mathcal{F}_2$, and is set to zero otherwise. Finally, at $t = 301$, another change of variables occurs and $\beta_{t,i}$, for $t = 301, \ldots, 400$, is drawn from a $N(5, 0.25)$ when $i \in \mathcal{F}_2$ and from a $N(10, 0.25)$ when $i \in \mathcal{F}_3$, and set to zero otherwise. In this way, the important predictors change over time and we expect these changes to be picked up in real-time by the algorithm.

The coefficient values are represented graphically in Fig. 3: in plot (a) the black-shaded area represents variables with a large coefficient, the gray-shaded area represents variables with a smaller coefficient and the unshaded area represents variables with a zero coefficient (inactive variables). In this setting, we set $R = 2$ and the sparsity

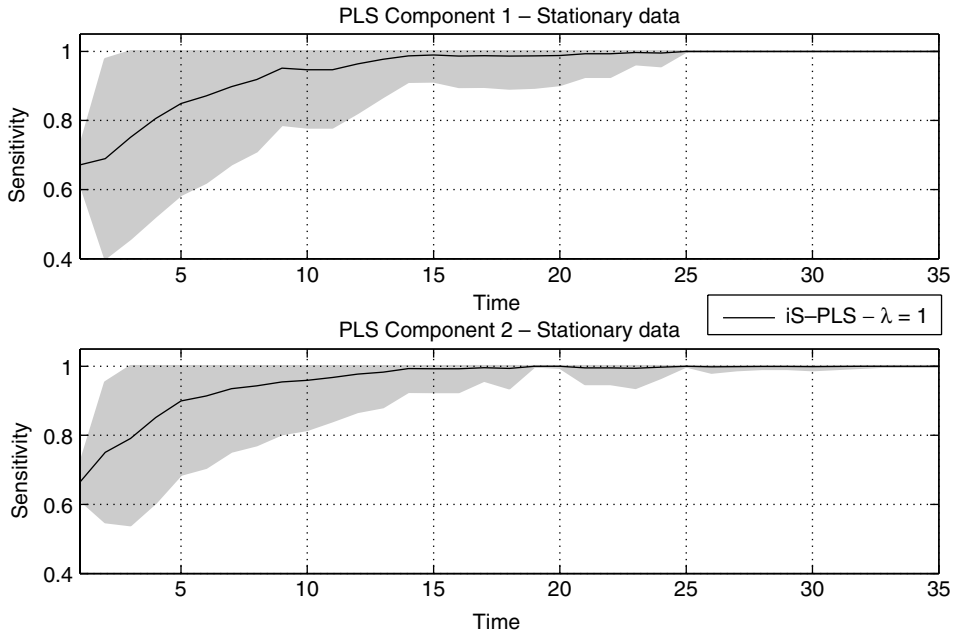

Fig. 2 Sensitivity of the iS-PLS algorithm. The shaded area shows the Monte Carlo error of the sensitivity index. It can be seen that after $t = 25$ data points, the iS-PLS result achieves maximum sensitivity and converges to the same solution obtained by off-line learning, when all data are available.

parameters $\gamma^{(r)}$ are chosen so that, at any given time, exactly 100 variables are selected. The forgetting factor λ is updated to ensure a rapid adjustment when the coefficients switch while also keeping the switching frequency low when the data is stationary to gain stability in the selected variables. Figure 3 also shows the results of a single run of this experiment. Clearly, the first PLS component is able to accurately select the most important group of variables. The second component always selects the second most important group of variables while mostly ignoring the group of variables selected by the first component. Neither component selects the inactive variables suggesting the algorithm is correctly able to distinguish important predictors from noise. As the regression coefficients switch, the algorithm only requires few data points before it detects the changes and adapts.

Figure 4 reports on the mean sensitivity of the iS-PLS algorithm in a Monte Carlo simulation consisting of 500 runs of this experiment for different values of λ. The solid line shows the mean percentage of correctly selected variables by the first and second PLS components. The shaded area shows the Monte Carlo error. This result clearly illustrates the limitations of using a fixed λ. Plot (a) shows the result when $\lambda = 1$. Here, during the first stationary period, the selected components are stable. However, when the coefficients suddenly transit to another stationary period, the algorithm adapts very slowly. Plot (b) shows the result when $\lambda = 0.9$. Although the algorithm is able to adapt quickly to changes in the coefficients, during stationary periods the estimated coefficients have higher variance. This can be seen by observing the larger Monte Carlo error during the periods of stationary data. However, the reported mean squared error (MSE) shows that, despite the higher variability, the ability to adapt to changes in the data (using a forgetting factor $\lambda = 0.9$, in this case) results in a smaller prediction error.

Figure 5 reports on the sensitivity of the iS-PLS algorithm when using the adaptive forgetting factor introduced in Section 3.2.1. Plot (a) shows the value of λ over time as a result of a self-tuning algorithm. During stationary periods, λ is kept very close to 1, as we would expect. When the coefficients "jump", λ is automatically set to a very small value for a short period of time and then set back to its previous value. This allows iS-PLS to adjust very quickly to sudden changes in the data while allowing a low-variance solution during stationary periods. Plot (b) shows the sensitivity measure obtained by iS-PLS when λ is adjusted using the self-tuning forgetting factor and takes the values reported in Plot (a). It is clear that in the stationary periods, iS-PLS correctly selects the important variables with very little error. In response to a change in the important factors, the percentage of correctly selected variables instantly decreases and quickly adapts to the new data. The algorithm eventually selects the correct variables after a short settling time. However, during this time the estimation variance increases. The reported MSE (91.94) shows that the ability to dynamically adapt the forgetting factor results in

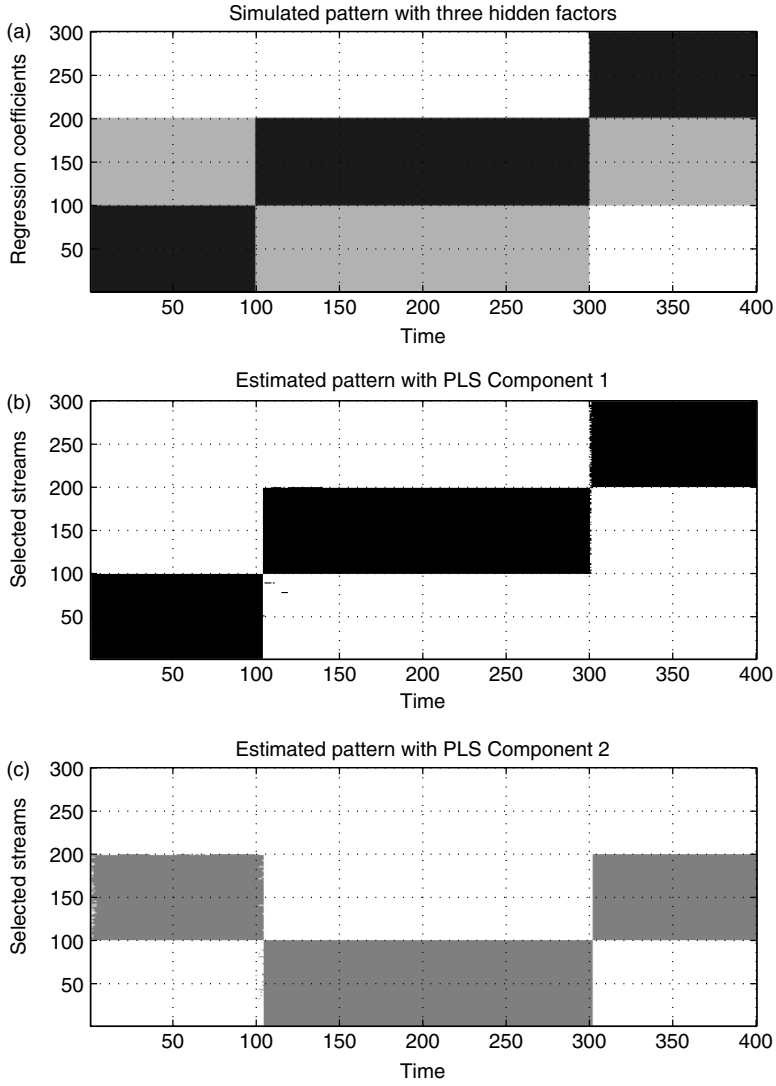

Fig. 3 Figure (a) shows a block-wise representation of input data streams: active streams having larger coefficients (black blocks), smaller regression coefficients (gray blocks), and inactive streams (white blocks) which only contributes to noise. Each block is related to a different hidden factor. Figures (b) and (c) shows the data streams selected on-line by the first and second PLS component, respectively, in a typical run. Noise occurs in the solution between $t = 0$ and $t = 10$ which occurs before the iS-PLS solution has converged as described in Figure 2. There is also noise present when the factors switch at $t = 100$ before the algorithm adjusts.

a smaller prediction error compared with the fixed λ case (Fig. 4).

In order to evaluate the performance of the iS-PLS algorithm in a more challenging setting, we devised a simulation where the number of change points is a random outcome governed by a geometric distribution with a parameter 0.1, so that the mean number of change points is 10 over a period of 1000 data points. When a change point occurs, the active variables also switch randomly. Furthermore, we restrict the number of variables which contribute to the response to be some percentage of the active variables, ranging between 1% (three variables) and 30% (90 variables). In this case, many of the variables are effectively noise. Our results are benchmarked against the recursive LARS (R-LARS) algorithm of Ref. [19] and the adaptive Lasso (aLasso) algorithm of Ref. [20]. In order to make the comparison meaningful and simple to interpret, we let the number of variables to be selected by each algorithm to be equal to the true number of active variables.

In Fig. 6, Plot (a) reports on the mean sensitivity, averaged over 1000 data points and over 500 Monte Carlo simulations. The shaded, colored areas represent the Monte Carlo errors for each competing algorithm. It can be seen that iS-PLS always achieves higher sensitivity compared with R-LARS or aLasso, even when the number of active variables is small. This suggests that the ability to identify

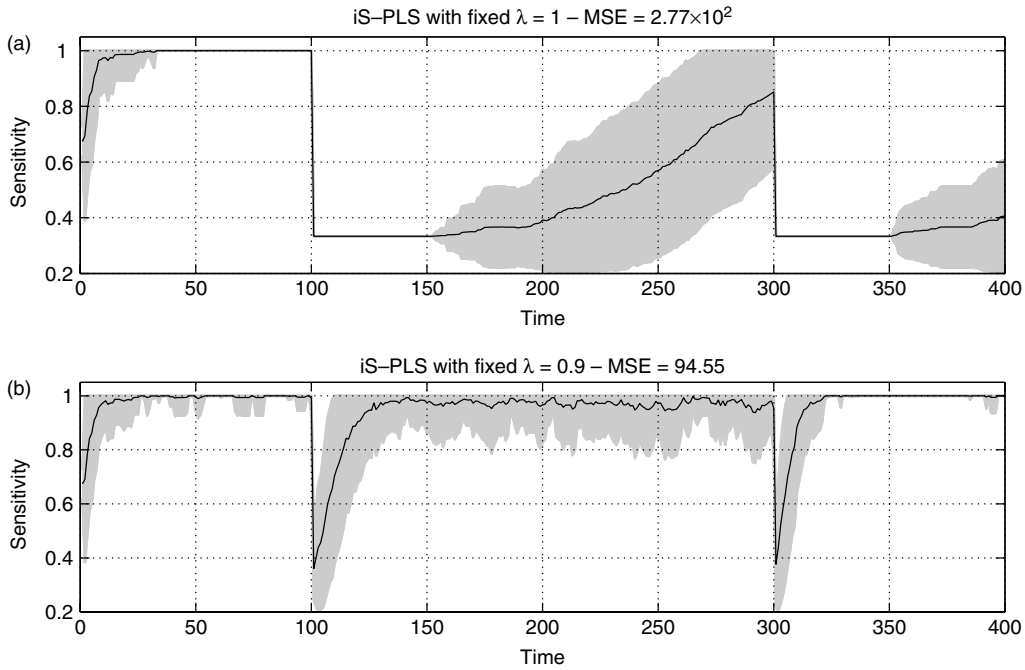

Fig. 4 Sensitivity of iS-PLS for different values of λ averaged over 500 Monte Carlo simulations where the shaded area shows the Monte Carlo error. In plot (a), $\lambda = 1$ which causes the solution to converge with low variance up until the coefficients change. It then is not able to quickly or accurately adjust to the new important factors. In plot (b), $\lambda = 0.9$ which allows the solution to rapidly adjust to changes; however, there is a lot of variance in the solution during periods when the coefficients are not changing due to the small forgetting factor. Despite the large variance, when $\lambda = 0.9$ the algorithm achieves greater sensitivity throughout the simulation which achieves a smaller prediction error as more of the correct variables are being selected compared with the case where there is no forgetting.

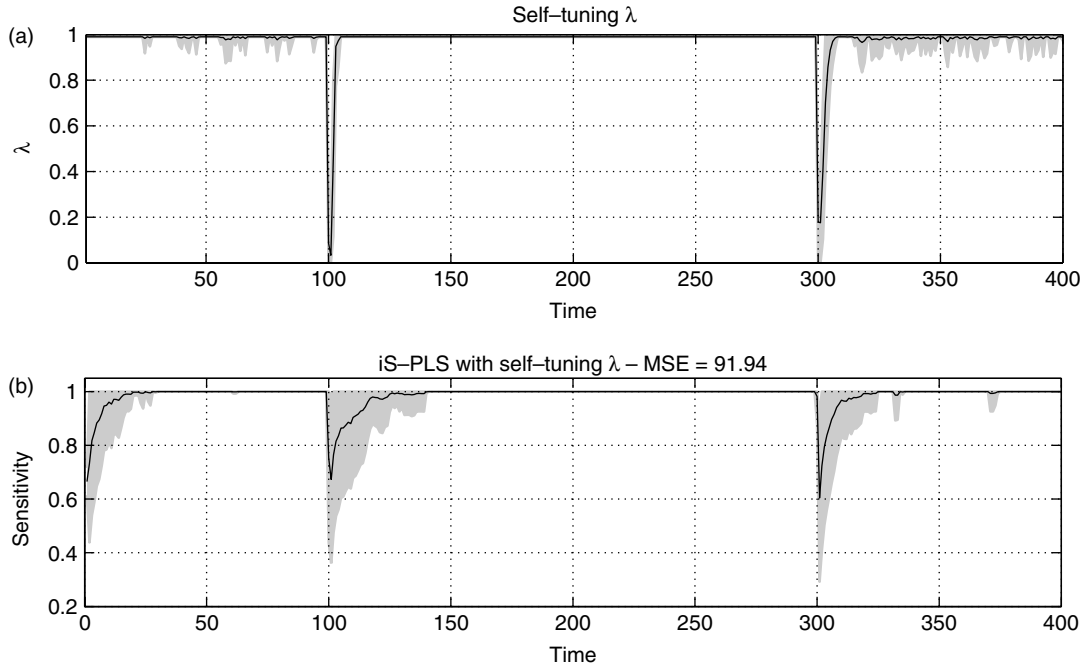

Fig. 5 Sensitivity of iS-PLS when using a self-tuning λ averaged over 500 Monte Carlo simulations where the shaded area shows the Monte Carlo error. Plot (a) shows how the value of the self-tuning forgetting factor changes over time. Plot (b) shows that when λ is dynamically adjusted using the variable forgetting factor algorithm, the solution adjusts rapidly and with little error to changes in the coefficients. During periods when the data is stationary, the variance in the solution is small.

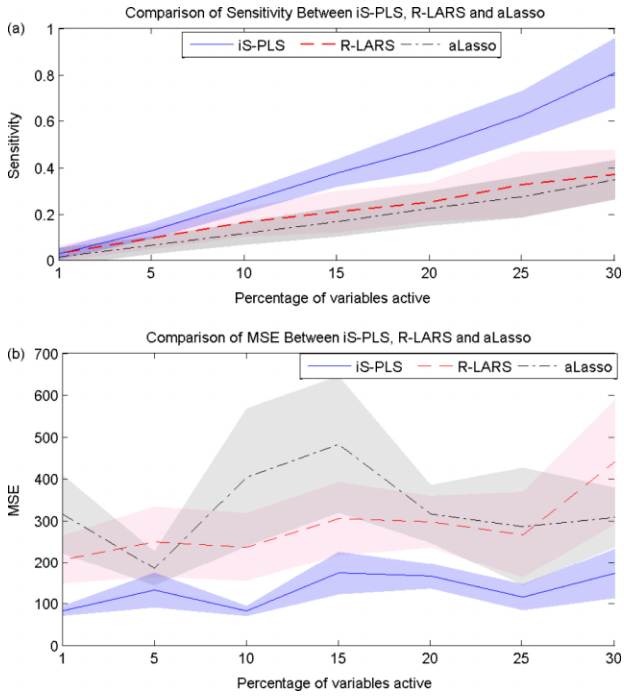

Fig. 6 Comparison of mean sensitivity (plot a) and MSE (plot b) over 1000 data points between iS-PLS, R-LARS and aLasso averaged over 500 Monte Carlo simulations where the shaded areas show the Monte Carlo error. Plot (a) shows that for fewer active variables in the simulation, iS-PLS achieving a higher sensitivity than R-LARS and aLasso. As the percentage of active variables increases toward 30% (90 variables), the sensitivity of all three algorithms increases. This suggests that iS-PLS is better able to select the variables corresponding to the important latent factor in the presence of many noisy and correlated variables. Plot (b) shows that iS-PLS achieves a smaller MSE with lower variance than R-LARS and aLasso; however, the error is not greatly affected by an increase in the sensitivity, which suggests that iS-PLS is better suited to situations where there are many correlated variables.

important latent factors is beneficial, especially in cases where a large number of the variables only contribute to noise and do not affect the response. Furthermore, these results illustrate the ability of iS-PLS to adapt to changes in the latent factors. As the number of variables increases, iS-PLS approaches quickly reaches maximum sensitivity, whereas the other methods do not perform well. Plot (b) reports on the corresponding MSE. The MSE achieved by iS-PLS is smaller and has lower variance, compared with other methods; however, the MSE is not greatly affected by an increase in the sensitivity.

4.4. Sensitivity Analysis

In order to study how the algorithm depends on the hyper-parameters, we carried out a sensitivity analysis. First, we examined the effect of α that regularizes the covariance matrix so that the full PLS solution may be extracted using only one SVD. On the basis of 500 Monte Carlo replicates, when α varies from 10^{-5} and 10^{-1}, the corresponding average MSEs (and their standard deviations) were 1.551(2.253) and 1.624(2.393), respectively; when α takes larger values, such as 0.5 and 1, the MSE gets as large as 7.899(32.551) and 32.331(187.559), respectively. Clearly, as α is kept very small, there is very little effect on the overall prediction error; however, as the solution becomes closer to the PCR solution, the MSE and standard deviation increase dramatically. Every value of α tested yields a covariance matrix H which is always of full rank; hence, it is always sensible to choose the smallest value of α to achieve a solution as close as possible to the true PLS solution.

Second, we look at the effect of the hyper-parameters a and b controlling the effective memory length in the self-tuning forgetting factor algorithm. We have studied the sensitivity of the iS-PLS algorithm using 500 Monte Carlo simulations as a function a and b, for $b \leq 1$ and $a \leq b$. The overall sensitivity ranges between 0.80 and 0.91. When $b > a$, the range of sensitivity is between 0.85 and 0.91. In the case where $a = b$, the *a priori* error and the *a posteriori* error in Eq. (11) become equal which prevents λ_t from adapting. Similarly when $b = 1$, the estimate for the *a posteriori* error does not discount past data which in turn stops λ_t from adapting. At these extreme cases, the algorithm achieves its lowest sensitivity of 0.80. In our simulations, we set $a = 0.5$ and $b = 0.9$ which lies close to the region of maximum sensitivity; we expect this to be the case because, to obtain an accurate estimate of the *a posteriori* error, we must use a long exponential time window, whereas the *a priori* error is an instantaneous estimate of the current error and can be obtained using an extremely short window.

To conclude our sensitivity analysis, we demonstrate how the sparsity parameter and the noise level affect the performance. Figure 7 contains a heatmap of the mean sensitivity of the S-PLS algorithm as a function of both the number of selected variables and the signal-to-noise ratio. For simplicity, we only consider the first latent factor. It clearly emerges that when the signal-to-noise ratio is high, the algorithm always correctly selects variables from the active group when less than 100 variables are selected (the true number of active variables in the first latent factor). As the noise level increases, the sensitivity of the algorithm decreases to the point where maximum sensitivity is only reached when all 300 variables are selected; this implies that S-PLS is no longer able to distinguish the active variables from the inactive variables. The optimal degree of sparsity should be selected using cross-validation and, following our assumption (A2), will be kept fixed over time.

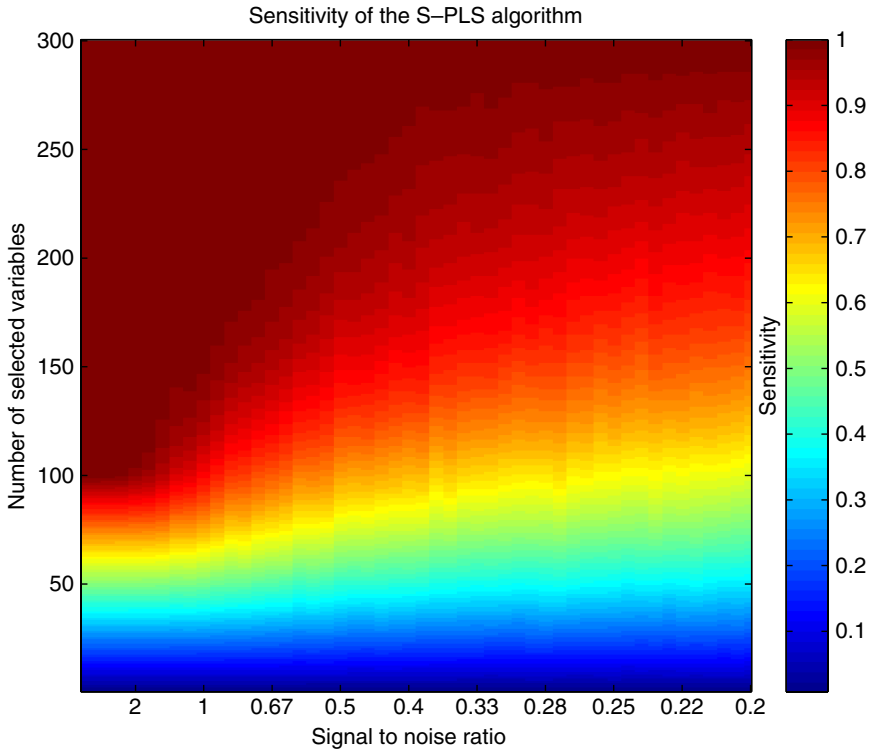

Fig. 7 Sensitivity of the S-PLS algorithm as a function of the number of variables selected and the signal-to-noise ratio in the first latent factor. Reported is an average over 500 Monte Carlo simulations. When the signal-to-noise ratio is high, S-PLS achieves maximum sensitivity when approximately 100 variables are selected (where 100 is the true number of active variables). As the noise increases to the point where the variance of the noise is four times that of the signal, the sensitivity decreases so that maximum sensitivity is only achieved when all variables are selected.

4.5. Performance with High-Dimensional Responses

In order to test the ability of iS-PLS to track important variables in a setting where multiple correlated responses are generated by the same underlying processes, we set up the following simulation. Two independent latent factors F_1 and F_2 of length $T = 400$ were simulated from a bivariate normal distribution with means $\mu_1 = 3$ and $\mu_2 = 5$ and identity covariance. Two Y-loading vectors v_1 and v_2 of length 50 were sampled from a standard normal distribution and orthogonalized with respect to each other using the QR decomposition. The 50 responses were generated as

$$Y = \sum_{j=1}^{2} F_j v_j + \epsilon,$$

where the errors are independently and identically distributed standard normals. Two time-varying $X-$loading vectors $u_{t,1}$ and $u_{t,2}$, both of length 300, have non-zero elements simulated from a normal distribution. Specifically, for $t = 1, \ldots, 100$, the non-zero elements of $u_{t,1}$ are drawn from a $N(10, 0.25)$, whereas the non-zero elements of $u_{t,2}$ are drawn from a $N(5, 0.25)$. All the non-zero loadings are then swapped at $t = 101$ and a final change of variables occurs at time $t = 301$. The predictor variables were then generated as

$$X = \sum_{t=1}^{T} \sum_{j=1}^{2} F_{t,j} u_{t,j} + \eta_t,$$

where $\eta_t \in \mathbb{R}^{1\times p}$ is standard normal noise. In this way, only two groups of variables are correlated with the response through the latent factors, F_j, at any one time.

We again compare the performance of iS-PLS with R-LARS and aLasso. Because these are univariate techniques, we run R-LARS and aLasso for each response separately and obtain the mean result. In Fig. 8, plot (a) shows the mean sensitivity of iS-PLS compared with R-LARS and aLasso as a result of 500 Monte Carlo simulations. It can be seen that iS-PLS is able to accurately select the important variables during periods where the loading coefficients are stationary, achieving close to maximum sensitivity. The iS-PLS algorithm is able to quickly adapt when the loadings change. Both R-LARS and aLasso achieve much lower sensitivity than iS-PLS and are much slower to adapt in response to changes in the active latent factor. This is expected as neither method is able to estimate the

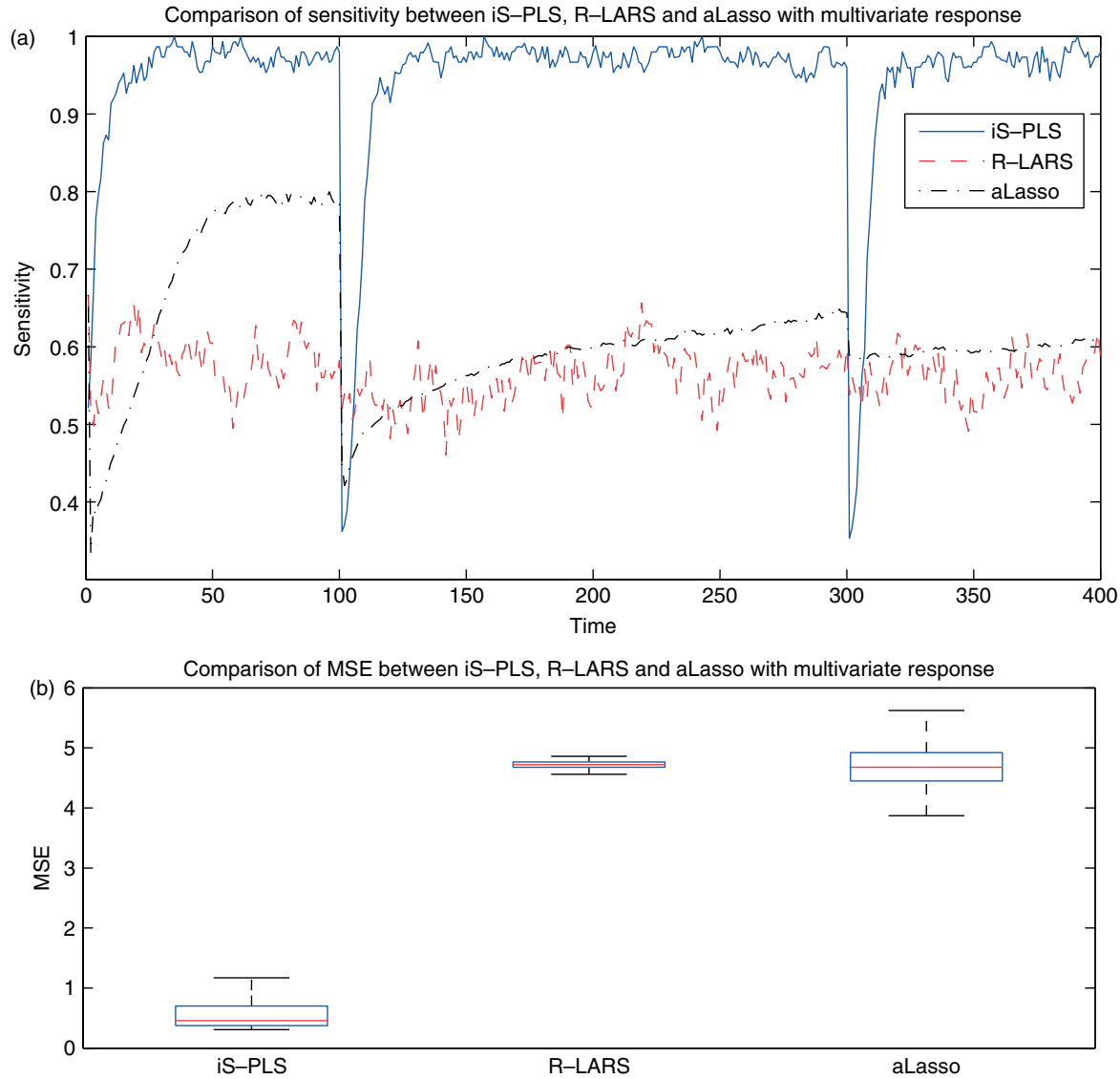

Fig. 8 Performance with high-dimensional responses. Plot (a) reports on a comparison of mean sensitivity between iS-PLS, R-LARS and aLasso averaged over 500 Monte Carlo simulations in a setting where there are 50 correlated responses which are generated by the same latent factors. iS-PLS achieves close to maximum sensitivity when tracking the latent factors using all 50 responses. As the important variables change, iS-PLS is able to quickly adapt. Neither R-LARS or aLasso are able to uncover the important variables as they do not assume the existence of latent factors and are unable to use information in all 50 responses simultaneously. Plot (b) reports on the distribution of MSE of the different methods over the Monte Carlo simulations. The iS-PLS algorithm achieves a consistently lower error than either R-LARS or aLasso due to its accuracy in tracking the latent factors and ability to quickly adapt to changes in the data.

latent factors which affect the important variables. Plot (b) shows the distribution of mean squared errors for the three methods. It can be seen that iS-PLS achieves a consistently lower error than either of the univariate techniques which is expected considering the accuracy of the tracking displayed by iS-PLS compared with R-LARS and aLasso.

We also report on the mean computational time required to run each algorithm once for 50 responses averaged over the same 500 Monte Carlo simulations. A key advantage of iS-PLS is its ability to handle high-dimensional inputs and responses simultaneously with relatively low computational cost. For a univariate response, the computational complexity of R-LARS and aLasso per time point is $O(p)$ and $O(p^2)$ respectively, compared with a complexity of $O(R^2 p)$ for iS-PLS. However, iS-PLS has a further advantage in that it can handle multivariate response with no further increase in complexity, whereas both other methods will scale linearly in the number of responses. In this setting, iS-PLS took 8.31s, whereas R-LARS took 556.9s and aLasso took 2391s. These timings confirm that iS-PLS is

significantly faster than both other methods when dealing with multivariate responses.

4.6. Ability to Track the Number of Important Latent Factors

Finally, we test the ability of iS-PLS to adaptively tune the number of latent factors R. Using the same simulation framework, we simulate a third uncorrelated latent factor in the same way as before and simulate loadings such that at time $t = 1, \ldots, 100$, only the first loading contains non-zero elements. At time $t = 101, \ldots, 300$, the first two factors contain non-zero elements and at $t = 301, \ldots, 400$ all three loading vectors contain non-zero elements.

In Fig. 9, plot (a) shows the mean estimated value of R_t through time compared with the true value of R_t. It can be seen that the iS-PLS can quickly and accurately adjust the number of latent factors active in the data as R_t increases from 1 to 3 and then decreases again. Plot (b) shows the mean value of the truncated error ratios $A_t^{(R_t-1)}$ and $A_t^{(R_t+1)}$. A sufficient negative gain in either ratio (as determined using CUSUM, see Section 3.2.2) indicates when the algorithm should increment or decrement the value of R_t. As more factors are added, the resulting negative gains in the error ratios gets smaller. This is because each subsequent latent factor added by iS-PLS accounts for progressively less of the total variance between the explanatory variables and the response. This accounts for the slower adjustments made when R_t increases from 2 to 3.

5. AN APPLICATION TO INDEX TRACKING

We have applied the iS-PLS algorithm to the problems of both univariate and multivariate index tracking. As mentioned in Section 1, a financial index consists of a portfolio of constituent stocks whose daily price is determined as a weighted sum of the prices of those constituents. A commonly used measure of performance of the index is the daily index return, y_t which is defined as the percentage gain (or loss) of the index price each day. Similarly, the daily portfolio returns, x_t, are the corresponding daily percentage gains in the individual stock prices. The makeup of an index is determined by a number of factors such as market capitalization and asset price and as such, stocks may be added or removed from an index based on whether they meet the criteria for inclusion in the index. Similarly, the

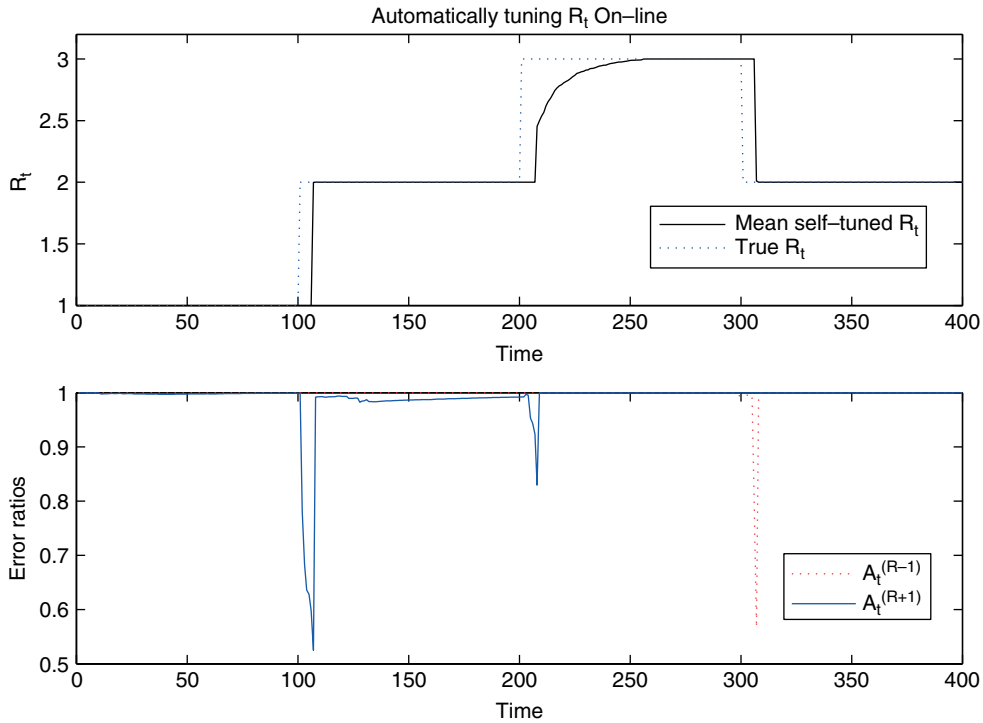

Fig. 9 Self-tuning the number of latent factors. Plot (a) shows how the mean value of R_t is adjusted in response to changes in the number of latent factors in the data over 500 Monte Carlo simulations. Starting with one important latent factor, at $t = 100$ and $t = 200$ we increment the number of important latent factors by 1 and at $t = 300$ we decrement the number of latent factors by 1. Plot (b) shows the truncated error ratios which determines when to update R_t. iS-PLS is able to quickly adapt as the number of latent factors in the data changes.

weights assigned to each asset are also frequently adjusted. These changes in the index cause its returns to be non-stationary.

There are two interconnected problems associated with index tracking: *asset selection* and *asset allocation*. Asset selection involves selecting a subset of p out of n available assets to construct a tracking portfolio. The asset allocation problem involves investing a proportion of the total available capital in each one of the p selected assets by estimating the index weight assigned to that asset. The overall goal of performing index tracking is to reproduce as accurately as possible the returns of the index. The constituents and weights of the tracking portfolio can be selected by attempting to minimize the *tracking error* [38], that is, the error between the index returns y_t and the tracking portfolio returns $\hat{y}$,

$$\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2, \tag{12}$$

where T is the period over which the returns are observed. Following this setting, the problem of asset allocation becomes a standard regression problem with the portfolio weights being the parameters to be estimated. Traditionally, the problem of selecting a small number of assets from a large index such as the S&P 500 has been performed using search algorithms such as simulated annealing [39,40]. However, this is prohibitively expensive. Therefore, we must look at methods of automatically selecting assets at low computational cost. Automatic asset selection is not a prevalent topic in the literature with most methods either falling into the category of full-index replication (no asset solution), computationally intensive search or proprietary algorithms.

Our motivation for using a sparse PLS algorithm for index tracking is that a number of studies of financial markets have suggested the existence of latent factors. For instance, Alexander and Dimitru [41,42] use standardized principal component weights as portfolio weights. They motivate their approach by providing evidence that the principal eigenvector of the covariance matrix of asset returns generally captures the underlying *market factor*.

Latent factor models related to principal components have also been suggested for the construction of an index tracking portfolio in Refs. [43] and [44]. However, to our knowledge PLS has not been used in the context of financial index tracking. Given the evidence for a latent factor underlying the market, we expect that PLS is better suited for index tracking applications as it takes into account the covariance between the constituent asset returns and the index returns as opposed to just the asset returns as in PCA. This means PLS should identify the assets which are contributing most importantly to the variation in the index returns.

For this application, we have used publicly available data sets from the S&P 100 and Nikkei indices, as previously described in Ref. [39]. The S&P 100 index has 98 constituents and the Nikkei has 225. To motivate the need

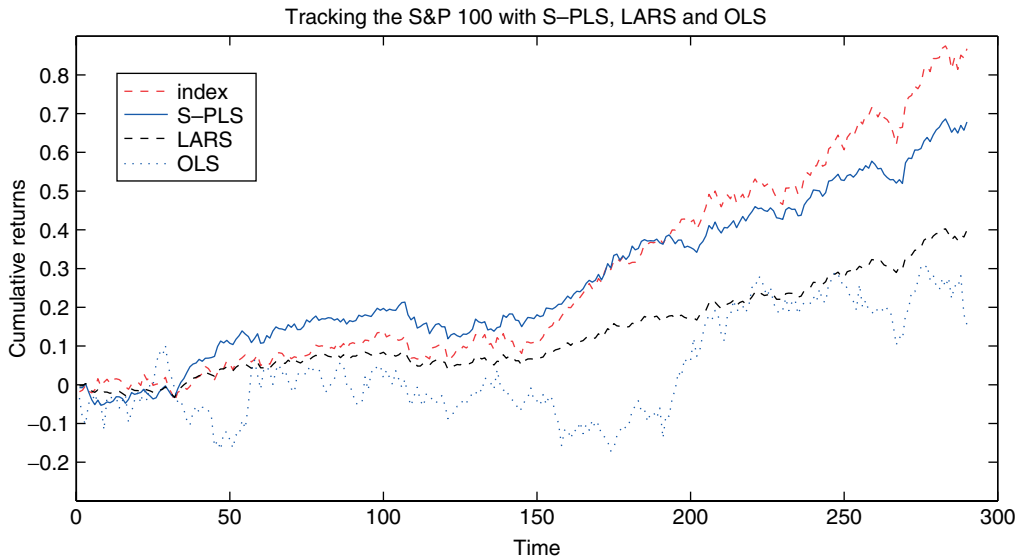

Fig. 10 Comparison of enhanced tracking (+15% annual returns) of the S&P 100 index using a static portfolio of 10 stocks chosen out of 98 using S-PLS and LARS. S-PLS performs well during the first half of the tracking period; however, as the entire period is used for training, a change in the"market factor" at around $t = 150$ is not picked up by the off-line S-PLS algorithm which causes tracking performance to degrade over the second half of the tracking period. Similarly, the performance of LARS gets worse after $t = 150$. The OLS portfolio exhibits poor tracking performance with high variance due to the correlated nature of the asset returns.

for an incremental variable selection algorithm for index tracking, we start by presenting an example of tracking with two off-line variable selection methods, our sparse PLS algorithm using one latent factor and the LARS algorithm of Ref. [5]. We also compare these methods to an ordinary least squares (OLS) fit which does not perform variable selection. We performed *enhanced* tracking of the S&P 100 index where the aim is to overperform the index returns by a certain percentage. This is achieved by creating a new target asset by adding a percentage of its annual returns to the index, we then aim to track this *enhanced* index [45]. We enhance the S&P 100 by an additional 15% annual returns.

Figure 10 shows the in-sample results of enhanced tracking of the S&P 100 index using a static portfolio of 10 stocks selected from 98 using S-PLS and LARS and full-index replication using OLS. Despite having access to all of the data, it is clear that using a static portfolio for a long period of time leads to poor tracking performance and in all three cases the artificial portfolios underperform the index. This result confirms that a scheme for periodically rebalancing the portfolio is necessary to produce better tracking

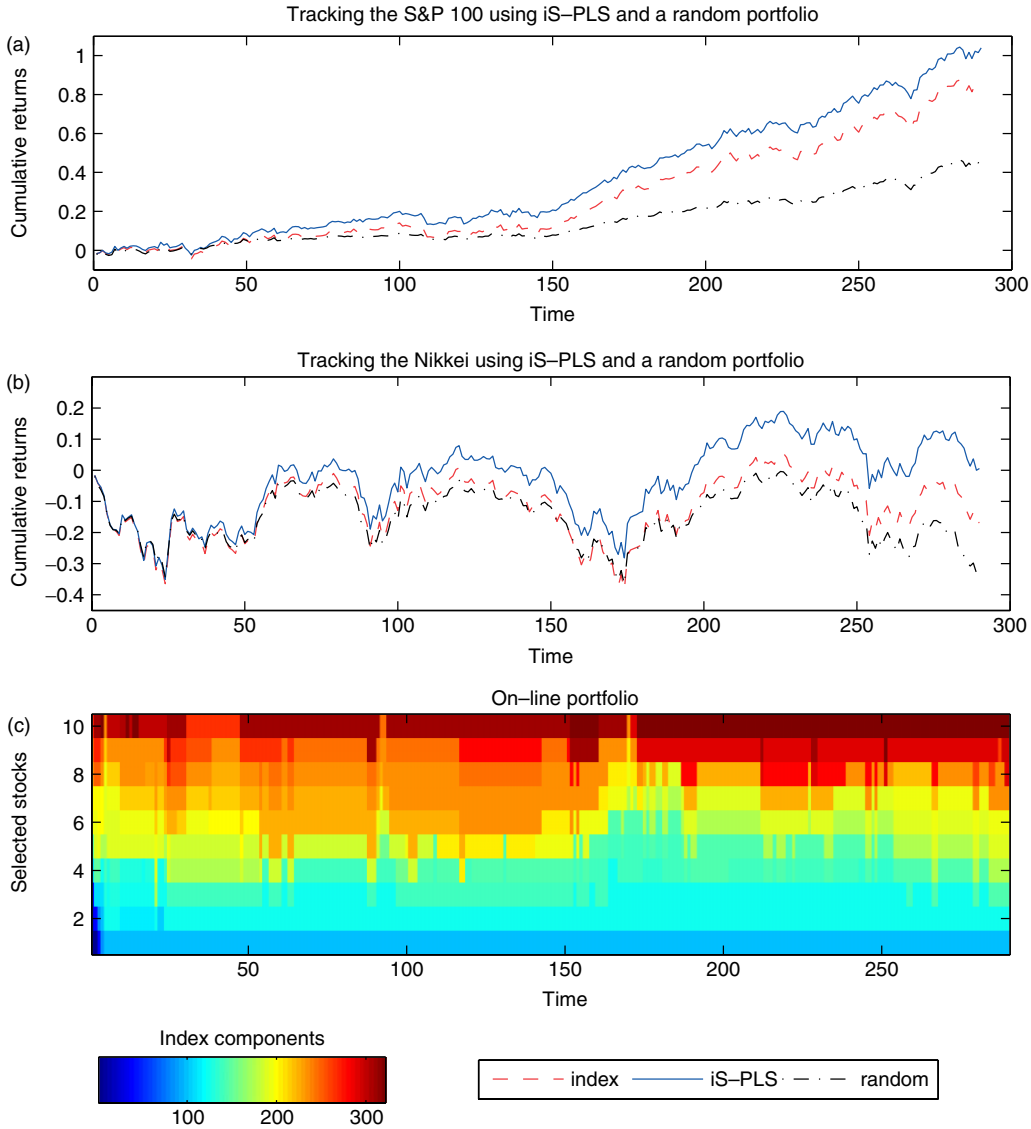

Fig. 11 A comparison between iS-PLS and an averaged random portfolio performing bivariate enhanced tracking (+15% annual returns) of the S&P 100 (Plot a) and Nikkei (Plot b) indices using a dynamic portfolio of 10 stocks out of 323. The dashed red line shows the returns of the index over the tracking period. The solid blue line and the dashed black shows the returns of the iS-PLS and random portfolios, respectively. For both indices, iS-PLS achieves the target returns, whereas the random portfolio underperforms the index. Plot (c) shows how the stocks selected by iS-PLS change over the tracking period, notably at $t = 150$ in response to the S&P 100 and at $t = 95$ and $t = 110$ in response to the Nikkei.

performance. The OLS tracking performance exhibits high variance due to the many correlated data assets which further suggests that a latent factor approach which represents correlated assets using a single factor is appropriate.

We have tested the iS-PLS algorithm in the multivariate case where two indices, the S&P 100 and the Nikkei, are simultaneously tracked. Both benchmark indices have been *enhanced* by adding 15% annual returns as previously described. The total combined number of available stocks is 323 and we set the portfolio size to 10. We constrain the selected stocks to be associated to the main latent factor only, so that $R = 1$, as in suggested in Ref. [41]. In order to assess whether iS-PLS selects the important variables over time more accurately than simple random guessing, we compared its performance with the average returns obtained from a population of 1000 random portfolios of the same size, with each portfolio being made of a randomly selected subset of assets. To make sure that the comparison is fair, the portfolio weights are also time-varying and are obtained by using a RLS method with the same λ parameter. Figure 11 shows the results of this test. It can be seen that iS-PLS consistently overperforms both indices and selects a small portfolio achieving the target annual returns of $+15\%$. In comparison, the mean returns of the 1000 random portfolios underperforms the S&P index by 32.07% and the Nikkei by 8.42%.

Our empirical results suggest that the importance of certain stocks in the index is not constant over time so the ability to detect and adapt to these changes is certainly advantageous. Using a model that assumes a time-varying latent factor driving the asset returns is also advantageous in this setting, because its existence in real markets has been heavily documented in the financial literature. The bottom plot of Fig. 11 is a heatmap illustrating how the makeup of the portfolio selected by iS-PLS changes during the entire period. Specifically, it shows the existence of a few important stocks that are held for the majority of the period, whereas other assets are picked and dropped more frequently throughout the period, further suggesting that it is advantageous to be able to adapt the constituents of a tracking portfolio.

Finally, we also compared the iS-PLS algorithm against the R-LARS and aLasso algorithms. The purpose of this comparison is again to determine whether assuming a time-varying latent factor which explains covariance between the data and the response is beneficial compared with two online variable selection techniques which do not make this assumption. Figure 12 compares the performance of the

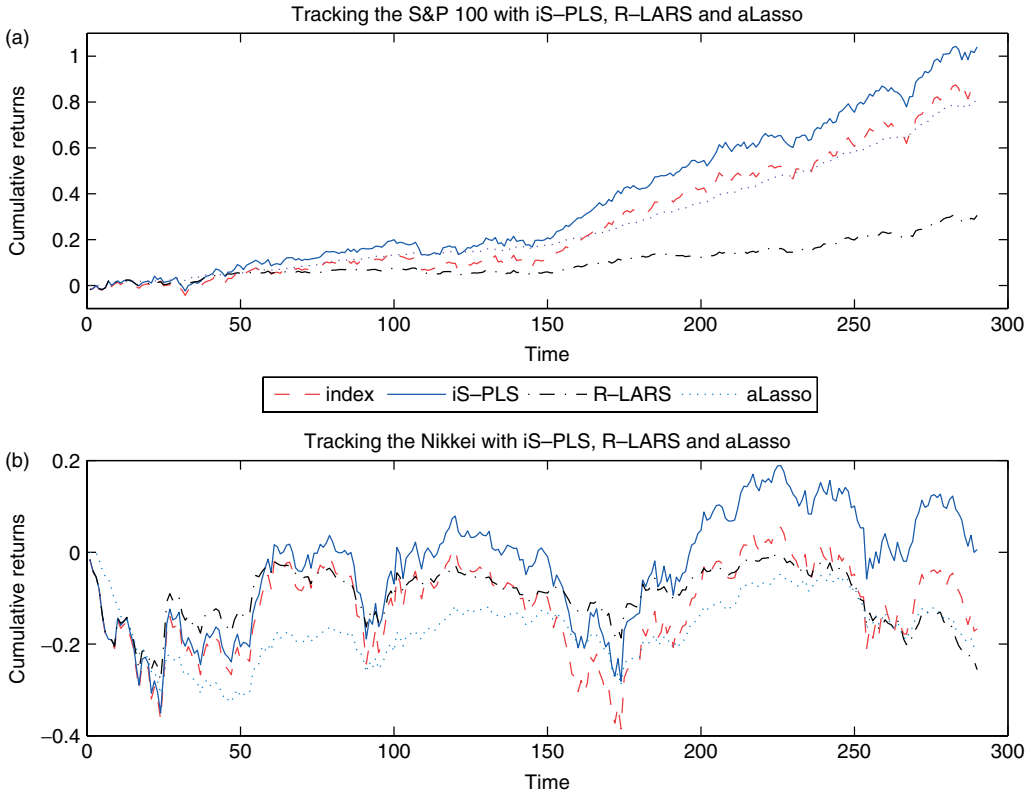

Fig. 12 A comparison of bivariate enhanced tracking (+15% annual returns) between iS-PLS, recursive LARS (R-LARS) and adaptive Lasso (aLasso) tracking the S&P 100 (Plot a) and Nikkei indices (Plot b). R-LARS and aLasso underperform both indices which suggests that the ability of iS-PLS to identify latent factors which contribute to the response is important in performing index tracking.

iS-PLS algorithm against that of R-LARS and aLasso for the same bivariate index tracking problem. Because both R-LARS and aLasso are univariate techniques, they cannot perform bivariate index tracking. Therefore, we compared iS-PLS performing bivariate index tracking against univariate tracking performed by R-LARS and aLasso on each of the indices separately. We set the portfolio size for each R-LARS and aLasso portfolio equal to 10. It can be seen that R-LARS and aLasso underperforms both indices suggesting that assuming a latent factor which explains the important covariation between the asset returns and the index returns is beneficial to performing accurate enhanced index tracking. It can be seen that aLasso performs slightly better than R-Lars which suggests that the use of a self-tuning forgetting factor is also beneficial to performing accurate tracking; however, it is not as important as the ability to identify the important latent factors.

6. CONCLUSIONS

In this work, we have presented an efficient on-line learning algorithm for variable selection in a multivariate regression context based on streaming data. The proposed method can be interpreted as an on-line and adaptive version of two-block PLS regression with sparsity constraints. As far as we are aware, this is the first such algorithm which combines dimensionality reduction and variable selection for data streams in a unified framework. iS-PLS has advantages, over other incremental algorithms, in that it is able to deal with high-dimensional and multicollinear data. The algorithm is also able to self-adjust some essential parameters, such as the number of important latent factors that need to be retained to obtain accurate predictions, and the forgetting factor needed to detect changes in the underlying data generating mechanism. Extensive simulation results show that the algorithm is able to accurately detect and track the important variables that should enter the regression model, even when sudden changes occur in the data. It also outperforms competing methods for on-line variable selection that do not assume the existence of latent factors.

The iS-PLS algorithm requires the specification of some parameters. Of these, the ridge parameter α and the forgetting factor memory length parameters a and b have been shown not to play a critical role in the performance of the algorithm. The iS-PLS algorithm is then able to accurately track any time-varying change in the covariance matrices and self-adjust the number and weights of latent factors. The user is only required to select the initial number of important PLS components to be retained. Prior to on-line learning, this value may be selected using domain-specific knowledge or prior information. For instance, in the financial index tracking application, it is well known that the first latent factor is representative of the entire market [41]; in other areas, such as genomic [29] and chemical process control [16], the selection of R is also guided by external information. Alternatively, the optimal R can be obtained using cross-validation [30].

In the existing literature, other approaches for on-line tracking of latent factors assume that the number of factors is fixed and these are updated in time without removing or adding new factors [11,13,46]. Recently, there have been some attempts to update R_t on-line in the context of PCA; for instance, in Refs [12], [47], and [48], the proportion of variance explained by the current number of factors is estimated and drives the tuning of R_t. However, these approaches are not immediately applicable to PLS regression. Other than tacking the PLS weights, our proposed iS-PLS algorithm is also able to improve on the prediction error by decreasing and increasing R_t as needed to achieve good predictive performance; this is obtained by means of an on-line approximation to cross-validation.

The only parameters that are assumed to be time-independent are $\gamma^{(r)}, r = 1, \ldots, R$, which controls the degree of sparsity in the PLS solutions. As with R, these parameters can be either selected using domain-specific information or, prior to on-line learning, using cross-validation. For instance, Waaijenborg anZwinderman [49] present a cross-validation method to select the sparsity parameter which achieves the lowest mean squared prediction error; however, they also suggest that some parameter tuning is performed by the user based on the number of desired variables. Recently, Witten *et al.* [29] present an application to genomic data where both the number of latent factors and the degree of sparsity is also selected by the user. In the relevant on-line learning literature, very little work has been carried out for adapting the degree of sparsity in recursive regression problems, especially in high-dimensional settings. To our knowledge, the only method of adapting this parameter over time has been proposed in Ref. [20], but they deal with a simpler on-line regression problem in which the response is univariate and the explanatory variables are uncorrelated. In their work, the sparsity parameter is selected to minimize a running estimate of the AIC criterion which quantifies the goodness of fit of the regression model. Although a similar solution may be investigated for our setting, this seems much harder to achieve in practice because our PLS model assumes R_t latent factors, which may change over time; the algorithm would then need to track the optimal sparsity parameter $\gamma_t^{(r)}$ for each factor $r = 1, \ldots, R$. More work needed to be performed toward the development a fully automated procedure.

We have also presented an application of iS-PLS to the bivariate index tracking problem in computational finance. On that task, iS-PLS overperforms both target indices

by a pre-selected amount. It also improves upon two competing on-line variable selection methods based on penalized regression, namely, recursive LARS [19] and adaptive Lasso [20], which consistently underperform the target indices.

Acknowledgments

We wish to thank Christoforos Anagnostopoulos for providing us with Matlab code for the adaptive Lasso algorithm and Nick Heard and Dimitris Tasoulis for helpful discussions and advice concerning simulations. We also wish to thank the anonymous referees and associate editor for insightful comments and suggestions which have helped improve the paper.

REFERENCES

[1] O. Nasraoui, C. Rojas, and C. Cardona, A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation, J Comput Network, 50(10) (2006), 1425–1652. (Special Issue on Web Dynamics).

[2] S.-K. Ng, G. J. McLachlan, and A. H Lee, An incremental em-based learning approach for on-line prediction of hospital resource utilization, Artif Intell Med 36 (2006), 257–267.

[3] Y. Zhu and D. Shasha, Statstream: statistical monitoring of thousands of data streams in real time, In Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.

[4] R Tibshirani, Regression shrinkage and selection via the lasso, J Roy Stat Soc B, 58(1) (1996), 267–288.

[5] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, Least angle regression, Ann Stat, 32 (2004), 407–499.

[6] J. Friedman, E. Hastie, H. Höfling, and R. Tibshirani, Pathwise coordinate optimization, Ann Appl Stat, 1(2) (2007), 302–332.

[7] V. DeMiguel, L. Garlappi, F. J. Nogales, and R. Uppal, A generalized approach to portfolio optimization: Improving performance by constraining portfolio norms, Manage Sci, 55(5) (2007), 798–812.

[8] J. Brodie, I. Daubechies, C. De Mol, C. Giannone, and I. Loris, Sparse and stable Markowitz portfolios. European Central Bank Working Paper Series 936, 2008.

[9] H. Wang, W. Fan, P. S. Yu, and J. Han, Mining concept-drifting data streams using ensemble classifiers, In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, DC, 2003, 226–235.

[10] J. Knight and S. Satchell, eds. Linear Factor Models in Finance, Oxford, UK, A Butterworth-Heinemann Title, 2004.

[11] J. Weng, Y. Zhang, and W. S. Hwang, Candid covariance-free incremental principal component analysis, IEEE Trans Pattern Anal Machine Intell, 25(8) (2003), 1034–1040.

[12] S. Papadimitriou, J. Sun, and C. Faloutsos, Streaming pattern discovery in multiple time-series, In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 2005, 697–708.

[13] B. Yang, Projection approximation subspace tracking, IEEE Trans Signal Process, 43 (1995), 95–107.

[14] A. Shu-Yan Wong, K. Wong, and C. Leung, A practical sequential method for principal component analysis, Neural Process Lett 11 (2000), 107–112.

[15] Z. Wang, Y. Lee, S. Fiori, C. S. Leung, and Y.-S. Zhu, An improved sequential method for principal component analysis, Pattern Recognit Lett, 24 (2003), 1409–1415.

[16] B. S. Dayal and J. F. Macgregor, Improved PLS algorithms, J Chemom, 11(1) (1997), 73–85.

[17] S. Vijayakumar, A. D'Souza, and S. Schaal, Incremental online learning in high dimensions, Neural Comput, 17 (2005), 2602–2634.

[18] S. Haykin, Adaptive Filter Theory, Upper Saddle River, New Jersey, Prentice Hall, 2001.

[19] S.-P. Kim, Y. N. Rao, D. Erdogmus, and J. C. Principe, Tracking of multivariate time-variant systems based on on-line variable selection, In 2004 IEEE Workshop on Machine Learning for Signal Processing, Sao Luis, Brazil, 2004, 123–132.

[20] C. Anagnostopoulos, D. Tasoulis, D. J. Hand, and N. M. Adams, Online optimisation for variable selection on data streams, Proceedings of the 18th European Conference on Artificial Intelligence, Patros, Greece, 2008, 132–136.

[21] S Erlich and K Yao. Convergences of adaptive block simultaneous iteration method for eigenstructure decomposition. Signal Process, 37 (1994), 1–13.

[22] A. J. Izenman, Regression, classification, and Manifold learning, Modern Multivariate Statistical Techniques, Springer Texts in Statistics, chapter 6, 2008, 107–190.

[23] R. Rosipal and N. Krämer, Overview and recent advances in partial least squares, Subspace, Latent Structure and Feature Selection Techniques, Berlin, Springer, 2006, 34–51.

[24] A. Hoskuldsson, PLS regression methods, J Chemom, 2(3) (1988), 211–228.

[25] H. Wold, Estimation of principal components and related models by iterative least squares, Multivariate Analysis, New York, Academic Press, 1966.

[26] J. Wegelin, A survey of partial least squares (PLS) methods, with emphasis on the two-block case, Technical report, University of Washington, 2000.

[27] L. Gidskehaug, H. Stdkilde-Jrgensen, M. Martens, and H. Martens, Bridge-PLS regression: two-block bilinear regression without deflation, J Chemom, 18 (2004), 208–215.

[28] H. Shen and J. Huang, Sparse principal component analysis via regularized low rank matrix approximation, J Multivariate Anal, 99 (2008), 1015–1034.

[29] D. M. Witten, R. Tibshirani, and T. Hastie, A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis, Biostatistics 10 (2009), 515–534.

[30] K. Lê Cao, D. Rossouw, C. Robert-Granié, and P. Besse, Sparse PLS: variable selection when integrating omic data, Technical report, INRA, 2008.

[31] G. Golub and C. F. Van Loan, Matrix Computations, Baltimore, The Johns Hopkins University Press, 1996.

[32] L. Ljung. System Identification: Theory for the User (2nd ed.), Upper Saddle River, New Jersey, Prentice Hall PTR, 1998.

[33] C. Paleologu, J. Benesty, and S. Ciochina, A robust variable forgetting factor recursive least-squares algorithm for system identification, IEEE Signal Process Lett, 15 (2008), 597–600.

[34] J. Sherman and W. J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, Ann Math Stat, 21 (1950), 124–127.
[35] S.-H. Leung and C.F. So, Gradient-based variable forgetting factor rls algorithm in time-varying environments, IEEE Trans Signal Process, 53(8) (2005), 3141–3150.
[36] M. Basseville and I.V. Nikiforov, Detection of abrupt changes: theory and application, New Jersey, Prentice-Hall, 1993.
[37] E. Bair, T. Hastie, D. Paul, and R. Tibshirani, Prediction by supervised principal components, J Amer Stat Assoc, 101 (2006), 119–137.
[38] J. Miao and C. L. Dunis, Volatility filters for dynamic portfolio optimization, Appl Financial Econ Lett, 1(2) (2005), 111–119.
[39] J.E. Beasley, N. Meade, and T. J. Chang, An evolutionary heuristic for the index tracking problem, Eur J Oper Res, 148 (2003), 621–643.
[40] M. Gilli and E. Këllezi, Threshold accepting for index tracking, Technical report, Department of Econometrics, University of Geneva, Switzerland, 2001.
[41] C. Alexander and A. Dimitriu, Sources of over-performance in equity markets: mean reversion, common trends and herding, Technical report, ISMA Center, University of Reading, UK, 2005.
[42] C. Alexander and A. Dimitriu, Equity indexing: optimising passive investments. Quant Finance 4(3) (2004), 30–33.
[43] A. Rudd, Optimal selection of passive portfolios, Financial Manage, 1 (1980), 57–66.
[44] R. A. Haugen and N.L. Baker, Dedicated stock portfolios, J Portfolio Manage, 16(4) (1990), 17–22.
[45] C. Alexander and A. Dimitriu, The cointegration alpha: enhanced index tracking and long-short equity market neutral strategies. Social Science Research Network Working Paper Series, June 2002.
[46] M. Brand, Fast online svd revisions for lightweight recommender systems, Technical report, Mitsubishi Electric Research Laboratory, 2003.
[47] M. Shen, D. Zhu, and Z. Zhu, Reduced-rank space-time adaptive processing using a modified projection approximation subspace tracking deflation approach, IET Radar, Sonar Navigation, 3(1) (2008), 93–100.
[48] A. Valizadeh and Mahmood Karimi, Fast subspace tracking algorithm based on the constrained projection approximation, EURASIP J Adv Signal Process (2009).
[49] S. Waaijenborg and A. Zwinderman, Penalized canonical correlation analysis to quantify the association between gene expression and dna markers, BMC Proceed, 1(Suppl 1) (2007), S122.