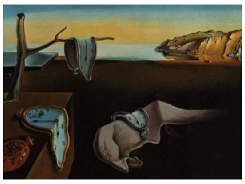


Journal of Statistical Software

October 2015, Volume 67, Code Snippet 1. *doi: 10.18637/jss.v067.c01*

The Forward Search for Very Large Datasets

Marco Riani
Università di Parma

Domenico Perrotta
European Commission
Joint Research Centre

Andrea Cerioli
Università di Parma

Abstract

The identification of atypical observations and the immunization of data analysis against both outliers and failures of modeling are important aspects of modern statistics. The forward search is a graphics rich approach that leads to the formal detection of outliers and to the detection of model inadequacy combined with suggestions for model enhancement. The key idea is to monitor quantities of interest, such as parameter estimates and test statistics, as the model is fitted to data subsets of increasing size. In this paper we propose some computational improvements of the forward search algorithm and we provide a recursive implementation of the procedure which exploits the information of the previous step. The output is a set of efficient routines for fast updating of the model parameter estimates, which do not require any data sorting, and fast computation of likelihood contributions, which do not require matrix inversion or *qr* decomposition. It is shown that the new algorithms enable a reduction of the computation time by more than 80%. Furthemore, the running time now increases almost linearly with the sample size. All the routines described in this paper are included in the **FSDA** toolbox for MATLAB which is freely downloadable from the internet.

Keywords: fast updating, **FSDA**, linear and logical indexing, order statistics, MATLAB.

1. Introduction

The forward search is a powerful general method for detecting anomalies in structured data (Atkinson and Riani 2000; Atkinson, Riani, and Cerioli 2004; Riani, Atkinson, and Cerioli 2009; Atkinson, Riani, and Cerioli 2010), which relies on a simple and attractive idea. In a nutshell, we are given a sample of n observations and we postulate a generating model for them. The method starts from a subset of cardinality $m_0 \ll n$, which is robustly chosen to contain only observations coming from the assumed model. This subset is used for fitting the model and the likelihood contribution (or alternatively a deviance measure) for each observation is computed. The subsequent fitting subset is then obtained by taking the $m_0 +$

1 observations with the largest likelihood contribution (smallest deviance measures). The algorithm iterates this fitting and updating scheme until all the observations are used in the fitting subset, thus yielding the classical statistical summary of the data. A major advantage of the forward search is its diagnostic power, because it shows the impact that each unit exerts on the fitting process, with outliers and other peculiar observations entering in the last steps of the algorithm. The definition of proper deviance residuals depends on the specific model for the data: in regression they are squared residuals, while in multivariate analysis we take Mahalanobis distances. Also the initialization step is model-dependent, and the most frequent choices are least trimmed squares for regression (Hubert, Rousseeuw, and Aelst 2008) and robust bivariate projections in multivariate analysis (Zani, Riani, and Corbellini 1998).

Recent applications of the forward search include systematic outlier detection in official Census data (Torti, Perrotta, Francescangeli, and Bianchi 2015), and the analysis of international trade markets (Cerioli and Perrotta 2014), where important issues such as incorrect declarations, tax evasion and money laundering are at the forefront. In both these instances the number of datasets to be analyzed is of the order of hundreds of thousands, while the sample size of each dataset ranges from less than 10 observations to more than 100000. It is thus crucial to improve the computational features of the methodology and to dramatically reduce its computation time. Otherwise, online monitoring and outlier detection, which are essential requirements for the successful implementation of statistical methods in these fields, would be unfeasible.

The goal of this work is to provide the computational and algorithmic advances that are necessary to apply the forward search to the massive datasets arising in applications like those sketched above. We thus provide the user with:

1. An analysis of the computation time required to perform the forward search for a wide set of sample sizes;
2. Efficient algorithms for both subset updating and computation of Mahalanobis distances or residuals; these algorithms do not need matrix inversion, but simple matrix multiplications, and avoid the use of sorting procedures;
3. An evaluation of the computation savings that derive from the use of the new algorithms.

All the routines described in this paper are included in the **FSDA** toolbox for MATLAB (Riani, Perrotta, and Torti 2012). This new software library, which extends MATLAB and its Statistics Toolbox to support a robust and efficient analysis of complex datasets, affected by different sources of heterogeneity, is freely downloadable from the websites `http://www.riani.it/MATLAB/` and `http://fsda.jrc.ec.europa.eu/`.

The structure of the paper is as follows. In Section 2 we give the details of our procedure for efficient updating, avoiding the sorting of Mahalanobis distances, and compare the new procedure to the current one in terms of computation time. In Section 3 we suggest an iterative method to compute likelihood contributions (deviance measures) at each step without resorting to the computation of inverse matrices. We show that the new approach leads to dramatic time savings in large samples. Section 4 discusses our findings. We also provide four appendices. The first three focus on the MATLAB implementation of the new software. In Appendix A, given that we make substantial use of submatrices, we compare logical and linear indexing. In Appendix B we analyze the difference in performance of the internal

MATLAB function `@bsxfun` with element by element multiplication `'.*'`, for small and large sample sizes. In Appendix C we give the details of the MATLAB implementation of a new algorithm to find an order statistic tailored to the forward search context; this implementation turns out to be faster than the use of the internal compiled MATLAB function based on `sort`. The Appendix D deals with the HTML documentation of the new functions which have been written.

2. Fast subset updating

Let $L_n(\theta)$ be the loglikelihood of all observations. The forward search fits subsets of observations of size m to the data, with $m = m_0, m_0 + 1, \dots n$. Let S_m be the subset of size m, for which the maximum likelihood estimate (m.l.e.) of the $p \times 1$ parameter θ is $\hat{\theta}(m)$. Then

$$L_m\{\hat{\theta}(m)\} = \sum_{i \in S_m} l_i\{\hat{\theta}(m)\}. \tag{1}$$

Loglikelihood contributions $l_i\{\hat{\theta}(m)\}$ can be calculated for all observations including those not in S_m. The search moves forward with the augmented subset S_{m+1} consisting of the observations with the $m+1$ largest values of $l_i\{\hat{\theta}(m)\}$.

In regression $y = X\beta + \epsilon$, where y is the $n \times 1$ vector of responses, X is an $n \times p$ full-rank matrix of known constants, with ith row $x_i^\top$, and β is a vector of p unknown parameters. The normal theory assumptions state that the errors ϵ_i are i.i.d. $N(0, \sigma^2)$. For regression the loglikelihood increment (1), omitting constants not depending on i, becomes

$$l_i\{\hat{\theta}(m)\} = -\{y_i - x_i^\top \hat{\beta}(m)\}^2 / 2s^2(m) = -e_i^2(m)/2s^2(m). \tag{2}$$

In (2) $\hat{\beta}(m)$ and $s^2(m)$ are the parameter estimates from subset S_m. The search thus moves forward with the augmented subset S_{m+1} consisting of the observations with the $m+1$ smallest absolute values of $e_i(m)$. An inferentially important consequence is that the estimates of the parameters are based only on those observations giving the m central residuals.

In the case of a sample $y = (y_1, \dots, y_n)^\top$ of v-variate observations from the multinormal distribution $N(\mu, \Sigma)$, the likelihood contribution (1), again omitting constants not depending on i, becomes

$$l_i\{\hat{\mu}(m), \hat{\Sigma}(m)\} = -\{y_i - \hat{\mu}(m)\}^\top \hat{\Sigma}(m)^{-1} \{y_i - \hat{\mu}(m)\}/2, \tag{3}$$

where $\hat{\mu}(m)$ and $\hat{\Sigma}(m)$ are the estimates of μ and Σ computed from the fitting subset S_m. Comparison with (2) shows that squared scaled residuals are now replaced by the squared Mahalanobis distances

$$d_i^2(m) = \{y_i - \hat{\mu}(m)\}^\top \hat{\Sigma}(m)^{-1} \{y_i - \hat{\mu}(m)\} \qquad i = 1, \dots, n, \tag{4}$$

which are used for progressing in the search and for detecting multivariate outliers.

In the current version of the algorithm, all the n squared residuals (Mahalanobis distances) are sorted and the units corresponding to the smallest $m+1$ deviance measures are considered, in order to select the new subset of size $m+1$. This procedure requires $O(n \log n)$ operations and becomes largely inefficient when n is very large, because the main quantity of interest is the $(m+1)$th order statistic of the deviance measures. In the theorem below we show that we do not require any sorting operations to update the subset in each step, but we simply need some logical operations and the possible calculation of minima.

Theorem 1. *Assume that just one unit joins the subset from step m to step $m+1$. Then, S_{m+1} can be found with $2n+1$ logical operations and the computation of a sum.*

On the other hand, if $k > 1$ new units join the subset, it is necessary to compute k additional minima and k additional logical operations to find S_{m+1}.

Proof. Let us consider a logical vector `bsbT` of length n containing a `true` in correspondence of the units forming the fitting subset at a certain step. Once the minimum deviance outside the subset has been calculated, the idea is to count how many units of the current subset have a deviance measure smaller than or equal to the minimum. Let this number be m'. Computation of m' can be done using $2 \times n$ logical operations. Let us call `rankgap` the difference between $m+1$ and m'. If just one new unit joins the subset, `rankgap` is equal to 1. The logical vector containing the units forming the new subset, therefore, can be obtained by adding a `true` in correspondence of the unit associated to the minimum deviance outside the subset.

The corresponding operations can be easily understood in the code snippet below, where `MD` is the vector of Mahalanobis distances for all the observations computed using centroid and covariance matrix based on S_m. We assume that the minimum Mahalanobis distance (`minMD`), and the index associated to the minimum, `minMDindex`, have already been computed as follows:

```
MDmod = MD; MDmod(bsbT) = Inf; [minMD, minMDindex(:)] = min(MDmod);
```

The logical vector containing the subset of units which have a Mahalanobis distance smaller than the minimum, and then `rankgap`, can be found as follows:

```
bsbriniT = (MD <= minMD) & oldbsbT; rankgap = m + 1 - sum(bsbriniT)
```

So, if `rankgap = 1` the new subset can be found from the simple logical instruction

```
bsbT(minMDindex) = true;
```

When `rankgap` is equal to $k > 1$, this means that k additional units must be added to form the new subset; that is, k additional `true` must be set in the logical vector `bsbriniT`. In order to find the next unit which will join the subset it is necessary to compute the minimum of the vector of modified Mahalanobis distances which has an `Inf` in correspondence of the m' units (`MDmod` in the code snippet below), and add a `true` and an `Inf` in vectors `bsbriniT` and `MDmod`, respectively.

```
MDmod = MD;
MDmod(bsbriniT) = Inf;
[~, minMDindex(:)] = min(MDmod);
bsbriniT(minMDindex) = true;
MDmod(minMDindex) = Inf;
```

The code above is inserted in a loop and repeated k times in order to find the new subset of size $m+1$. □

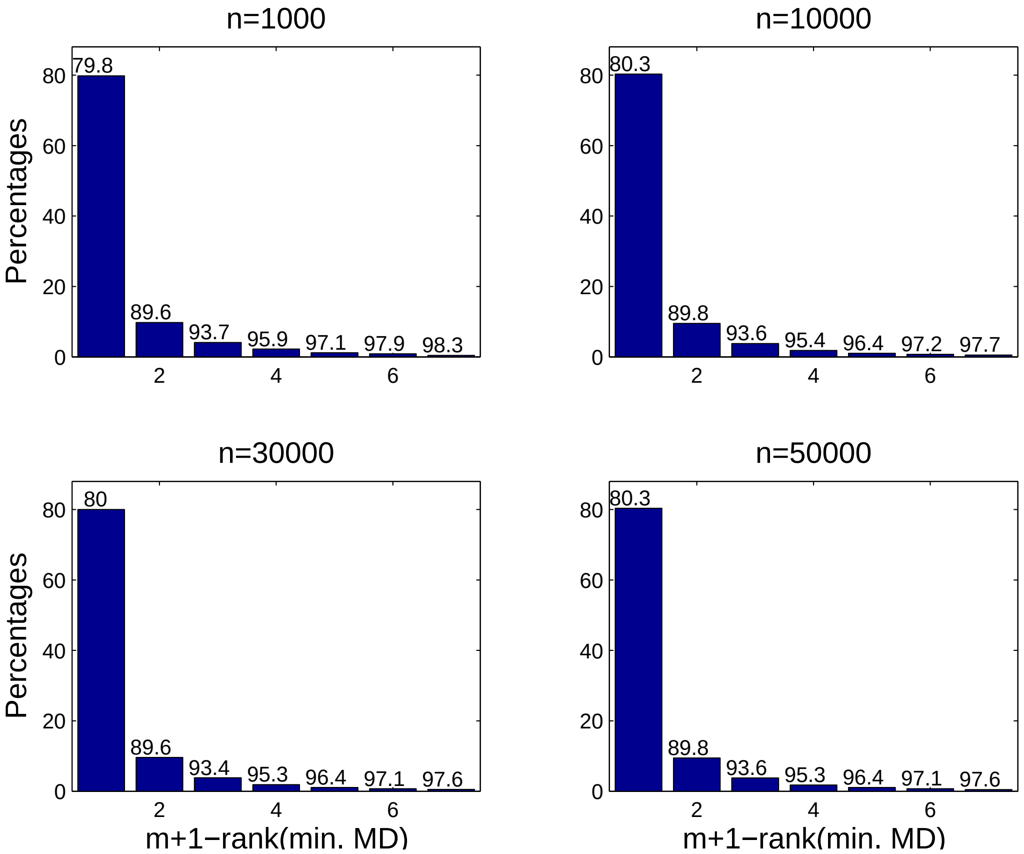


Figure 1: Analysis of the frequency distribution of `rankgap` for 4 samples sizes $n = 1000$, 10000, 30000 and 50000. The numbers on top of the bars are the cumulative percentages.

The new updating strategy poses the question of the distribution of `rankgap` along the search. Figure 1 addresses this problem. The plot considers the frequency distribution of `rankgap` when $n = 1000$, 10000, 30000 and 50000, $v = 2$, and the data are normally distributed. Without loss of generality, the data are standardized because of the invariance properties of the deviance measures (4). Similar results have been obtained also with $v = 10$, but are not reported here. This figure clearly shows that in about 80% of the steps the value of `rankgap` is 1, and that the cases in which `rankgap` ≤ 2 involve about 90% of all the steps of the forward search. This implies that nine times out of ten on average we simply need to compute 1 minimum and $2n + 2$ logical operations in order to find the new subset. The situations in which `rankgap` is greater than 6 are just roughly 2%.

The second question that the new procedure naturally raises is up to which value of `rankgap` it is convenient to compute the new subset using repeated minima, and when it is better to find the $(m+1)$th order statistic directly. In Appendix C we give a MATLAB implementation of a new algorithm to find an order statistic, specifically tailored to the forward search. Our analyses, not given here, show that it is better to resort to repeated minima when `rankgap` is smaller than or equal to 10. Otherwise, we resort to our new algorithm for the $(m + 1)$th order statistic.

The third question that the new procedure implies is up to which point of the search it is convenient to rely on linear indexing instead of logical indexing. In linear indexing the elements to be extracted from a vector are indicated through a set of integers, which correspond to the element position in the vector. Logical indexing extracts the elements using a Boolean vector of the same length as the original one. The results in Appendix A show that the computing time of linear indexing increases linearly with the number of elements to extract, as expected, and that linear indexing is quicker than logical indexing if the percentage of elements to extract is not greater than 50%. This implies that in the forward search we must find the step up to which linear indexing is more convenient. Figure 2 gives computing time as a function of the proportion of subsequent steps for which linear indexing is adopted. It shows that the optimal proportion is about 0.85. Clearly, the difference in computing time is negligible when $n \leq 10000$, but it becomes important when n is very large. Therefore, we suggest to use linear indexing until $m/n = 0.85$ and then to revert to logical indexing. This adjustment implies that to extract the units of S_m, for $m = m_0, m_0 + 1 \ldots, n$, we need to add the conditional statement:

```
if m <= percn;
    Yb = Y(bsb, :);
else
    Yb = Y(bsbT, :);
end
```

where `percn` $= 0.85 \times n$ and `bsb` is an int16 or int32 vector of size m.

We now compare the computing time of our new procedure to update the subset against the currently available one, which is based on the use of MATLAB function `sort`. More precisely, in the traditional implementation the new subset of size $m + 1$, given S_m, was found by the following code[1]:

```
[~, malasorind(:)] = sort(MD);
bsb = malasorind(1:m + 1);
```

Figure 3 shows that the reduction of computation time ensured by the use of the new procedure is huge. The left panel refers to sample sizes n in the range 100–100000. The right panel is a zoom referred to the interval 100–10000. The new updating procedure is almost 10 times faster than the current one. An additional important property of the new procedure is the time is now a roughly linear function of n.

3. Fast deviance measures updating

The purpose of this section is to show, given the subset S_{m+1}, how it is possible to efficiently compute the n corresponding deviance measures using the quantities available from the previous step m. As we have seen, in most moves from S_m to S_{m+1} just one new unit joins the subset. In this case simple deletion formulae can be used (Atkinson and Riani 2000; Atkinson *et al.* 2004). However, in presence of a complicated structure based on more than one group, there be can steps in which k units leave the subset and $k + 1$ join it, when passing from S_m

[1] In order to save space `malasorind` is predefined as int16 or int32 signed integer arrays. The instruction `malasorind(:)` ensures that the output of `sort` is not implicitly converted to a double precision array.

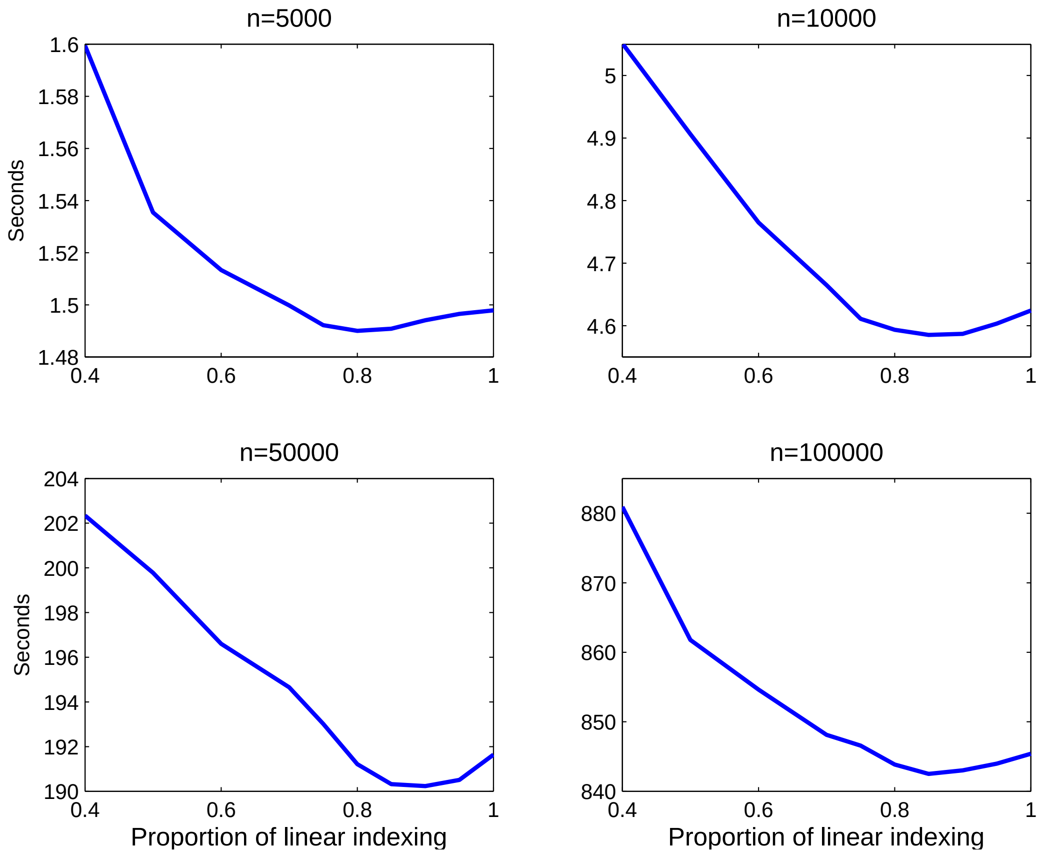


Figure 2: Computing time to perform the forward search as a function of the proportion of linear indexing.

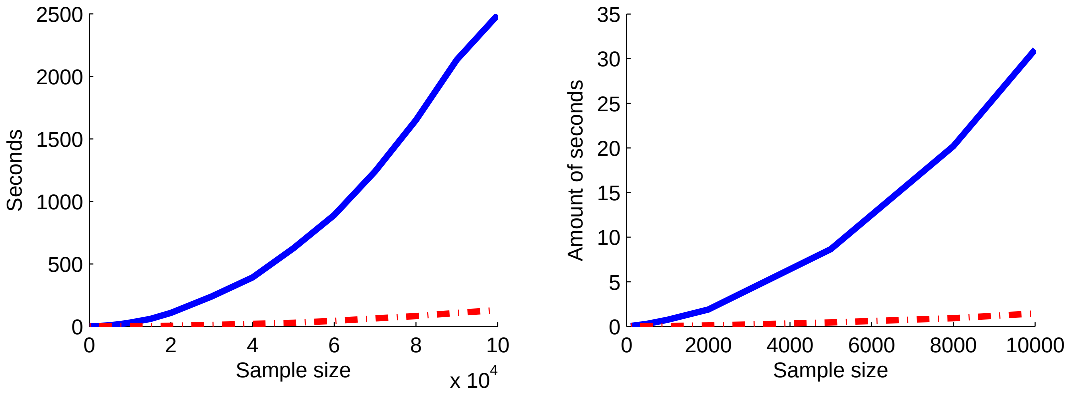


Figure 3: Comparison of computing time between traditional (solid line) and new (dash-dotted line) subset updating.

to S_{m+1}. This event is often called an "interchange". It is thus necessary to generalize the traditional deletion formulae to deal with interchanges. In what follows we concentrate on multivariate analysis, but our approach can be easily extended to regression.

Let $\hat{\Sigma}(m)_{x\in S_m}$ be the covariance matrix based on the units which are inside subset S_m. We can see by algebraic manipulations that this matrix can be decomposed as follows:

$$\begin{aligned}(m-1)\hat{\Sigma}(m)_{x\in S_m} &= (m-k-1)\hat{\Sigma}(m-k)_{x\in S_m\cap x\in S_{m+1}} + (k-1)\hat{\Sigma}(k)_{x\in S_m\cap x\notin S_{m+1}} + \\ &+ \frac{k(m-k)}{m}(\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}} - \hat{\mu}(k)_{x\in S_m\cap x\notin S_{m+1}})\times \\ &\qquad (\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}} - \hat{\mu}(k)_{x\in S_m\cap x\notin S_{m+1}})^\top, \quad (5)\end{aligned}$$

where $\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}}$ and $\hat{\Sigma}_{x\in S_m\cap x\in S_{m+1}}$ are, respectively, the centroid and the covariance matrix of the $(m-k)$ units which remain in the subset at step $m+1$, while $\hat{\mu}(k)_{x\in S_m\cap x\notin S_{m+1}}$ and $\hat{\Sigma}_{x\in S_m\cap x\notin S_{m+1}}$ are the centroid and the covariance matrix of the k units which leave the subset when passing from S_m to S_{m+1}. Similarly, there is the following relationship between the covariance matrix based on S_{m+1}, i.e., $\hat{\Sigma}(m+1)_{x\in S_{m+1}}$, and $\hat{\Sigma}(m-k)_{x\in S_m\cap x\in S_{m+1}}$:

$$\begin{aligned}m\hat{\Sigma}(m+1)_{x\in S_{m+1}} &= (m-k-1)\hat{\Sigma}(m-k)_{x\in S_m\cap x\in S_{m+1}} + k\hat{\Sigma}(k+1)_{x\notin S_m\cap x\in S_{m+1}} \\ &+ \frac{(k+1)(m-k)}{m+1}(\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}} - \hat{\mu}(k+1)_{x\notin S_m\cap x\in S_{m+1}})\times \\ &\qquad (\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}} - \hat{\mu}(k+1)_{x\notin S_m\cap x\in S_{m+1}})^\top \quad (6)\end{aligned}$$

where $\hat{\mu}(k+1)_{x\notin S_m\cap x\in S_{m+1}}$ and $\hat{\Sigma}_{x\notin S_m\cap x\in S_{m+1}}$ are, respectively, the centroid and the covariance matrix of the $k+1$ new units which join the subset when passing from step m to step $m+1$.

It is well known that the inverse of a $v\times v$ matrix $D=(C+\alpha xx^\top)$ can be expressed as follows:

$$(C+\alpha xx^\top)^{-1} = C^{-1} - \frac{\alpha C^{-1}xx^\top C^{-1}}{1+\alpha x^\top C^{-1}x},$$

where the $v\times v$ matrix C is of full rank, $\alpha\in\Re$ and x is a $v\times 1$ vector. This expression is particularly useful if C^{-1} is already available. In general, if we need to compute the inverse of $D=(C+\sum_{i=1}^r \alpha_i z_i z_i^\top)$ we can exploit the following identities

$$\begin{aligned}(D-\sum_{i=2}^r \alpha_i z_i z_i^\top)^{-1} &= (C+\alpha_1 z_1 z_1^\top)^{-1} = C_1^{-1} \\ (D-\sum_{i=3}^r \alpha_i z_i z_i^\top)^{-1} &= (C_1+\alpha_2 z_2 z_2^\top)^{-1} = C_2^{-1} \\ &\vdots \\ (D-\sum_{i=r-1}^r \alpha_i z_i z_i^\top)^{-1} &= (C_{r-3}+\alpha_{r-2} z_{r-2} z_{r-2}^\top)^{-1} = C_{r-2}^{-1} \\ (D-\alpha_r z_r z_r^\top)^{-1} &= (C_{r-2}+\alpha_{r-1} z_{r-1} z_{r-1}^\top)^{-1} = C_{r-1}^{-1} \\ D^{-1} &= (C_{r-1}+\alpha_r z_r z_r^\top)^{-1} = C_r^{-1}.\end{aligned}$$

As a consequence, if the inverse of C is already available, matrix D^{-1} can be efficiently obtained through the following recursion:

$$C_j^{-1} = (C_{j-1}+\alpha_j z_j z_j^\top)^{-1} = \frac{C_{j-1}^{-1} - \alpha_j C_{j-1}^{-1} z_j z_j^\top C_{j-1}^{-1}}{1+\alpha z_j^\top C_{j-1}^{-1} z_j} \qquad j=1,\ldots,r, \qquad (7)$$

with $C_r^{-1} = D^{-1}$ and $C_0^{-1} = C^{-1}$.

We can apply these results to the updating process implied by the forward search. If k units leave the fitting subset when passing from S_m to S_{m+1}, we can easily obtain matrix $\{(m-k-1)\hat{\Sigma}(m-k)_{x\in S_m \cap x \in S_{m+1}}\}^{-1}$ from $\{(m-1)\hat{\Sigma}(m)_{x\in S_m}\}^{-1}$ using Equation 5 and recursion (7) with

$$r = k+1, \qquad \alpha_1 = -\frac{k(m-k)}{m}, \qquad z_1 = (\hat{\mu}(m-k)_{x\in S_m\cap x\in S_{m+1}} - \hat{\mu}(k)_{x\in S_m \cap x \notin S_{m+1}}),$$

and, for $j = 2, \ldots, k+1$,

$$\alpha_j = -1, \qquad z_j = x_j - \hat{\mu}(k)_{x\in S_m\cap x\notin S_{m+1}}, \qquad \text{with } \{x_j : x_j \in S_m \cap x_j \notin S_{m+1}\}.$$

Similarly, using Equation 6, if we start from $\{(m-k-1)\hat{\Sigma}(m-k)_{x\in S_m\cap x\in S_{m+1}}\}^{-1}$ we can easily compute $\{m\hat{\Sigma}(m+1)_{x\in S_{m+1}}\}^{-1}$ by putting

$$r = k+2, \qquad \alpha_1 = \frac{(k+1)(m-k)}{m+1}, \qquad z_1 = (\hat{\mu}(m-k)_{x\notin S_m\cap x\in S_{m+1}} - \hat{\mu}(k)_{x\in S_m\cap x\notin S_{m+1}}),$$

and, for $j = 2, \ldots, k+2$,

$$\alpha_j = 1, \qquad z_j = x_j - \hat{\mu}(k)_{x\notin S_m\cap x\in S_{m+1}}, \qquad \text{with } \{x_j : x_j \notin S_m \cap x_j \in S_{m+1}\}.$$

If k units (with $k = 2, 3, \ldots$) leave the fitting subset we need to run recursion (7) $2k+3$ times. It is also worthwhile to notice that, if in passing from S_m to S_{m+1} just one unit leaves the subset, then Equation 5 reduces to

$$\begin{aligned}(m-1)\hat{\Sigma}(m)_{x\in S_m} = {}& (m-2)\hat{\Sigma}(m-1)_{x\in S_m\cap x\in S_{m+1}} + \\ & + \frac{(m-1)}{m}(\hat{\mu}(m-1)_{x\in S_m\cap x\in S_{m+1}} - x_{x\in S_m\cap x\notin S_{m+1}})\times \\ & (\hat{\mu}(m-1)_{x\in S_m\cap x\in S_{m+1}} - x_{x\in S_m\cap x\notin S_{m+1}})^\top, \end{aligned} \tag{8}$$

so just four iterations of Equation 7 are needed. Finally, in normal progression (no interchange), Equation 6 reduces to

$$\begin{aligned}m\hat{\Sigma}(m+1)_{x\in S_{m+1}} = {}& (m-1)\hat{\Sigma}(m)_{x\in S_m\cap x\in S_{m+1}} \\ & + \frac{2(m-1)}{m+1}(\hat{\mu}(m)_{x\in S_m\cap x\in S_{m+1}} - x_{x\notin S_m\cap x\in S_{m+1}})\times \\ & (\hat{\mu}(m)_{x\in S_m\cap x\in S_{m+1}} - x_{x\notin S_m\cap x\in S_{m+1}})^\top \end{aligned} \tag{9}$$

and just one iteration of (7) is needed.

Remark: The well known formula of Sherman-Morrison-Woodbury (see for example Equation 2.31 in Atkinson and Riani 2000) states that:

$$(A - UV^\top)^{-1} = A^{-1} + A^{-1}U(I_r - V^\top A^{-1}U)^{-1}V^\top A^{-1}, \tag{10}$$

where A is $p \times p$ and it is assumed that all necessary inverses exist. The advantage of using the matrix form is thought to be that if r is smaller than p the required inverse is $r \times r$. Translated to our notation, A is a $v \times v$ function of the covariance matrix at step m, while

$UV^\top$ is a function of the covariance matrix of the units which leave or enter the subset from step m to step $m+1$. Therefore, Equation 10 does not bring any computational advantage because, even if A^{-1} is precomputed, this formula requires the inverse of $I_r - V^\top A^{-1}U$, which can be of size greater than v in presence of considerable interchange. The iterative method suggested above avoids the computation of such an inverse. The numerical properties of the Sherman-Morrison-Woodbury formula are nicely discussed by Yip (1986) and Hager (1989), in terms of the well known condition estimator or condition number (Hager 1984, function `cond` in MATLAB).

Figure 4 compares the new way of computing Mahalanobis distances with the traditional one which is based on the `qr` decomposition. More precisely, if `bsbT` is defined as in the previous section, `Y` is the $n \times v$ data matrix, `ym` is the mean of the units forming S_m, and `Ym` is the matrix which contains the deviations from the means computed using the units in S_m, the snippet of the currently available code to compute the distances is given below

```
Yb = Y(bsbT, :);
ym = sum(Yb, 1)/m;
Ym = bsxfun(@minus,Y, ym);
[~, R] = qr(Ym(bsbT, :), 0);
u = (Ym/R)'
% Compute squared Mahalanobis distances
MD = (m - 1) * sum(u.^2, 1);
```

(11)

In the new way, once the inverse of the new covariance matrix has been computed recursively (which we denote with `S`), the Mahalanobis distances are computed using the following instruction:

```
MD = (m - 1) * sum((Ym * S) .* Ym, 2);
```

(12)

Figure 4 compares the amount of seconds required by the two procedures. The left panel of this figure considers a sample size in the range 100–100000, while the right panel is the zoom of the previous plot in the interval $100 \leq n \leq 10000$. These two figures show that if we use the new updating formulae the time is reduced by almost two third.

Snippet 11 of the traditional Mahalanobis distances code shows that we avoided the direct computation of the covariance matrix inverse by using the `qr` decomposition. The choice is motivated by the good numerical stability of the decomposition, providing minimal amplification of the error inherent in the original subset matrix *Ybsbm* (that is `Ym(bsbT, :)`). In fact, cond(*Ybsbm*) = cond(QR) = cond(R) with Q being an orthogonal basis for the space of *Ybsbm* (so cond(Q) = 1) and R an upper triangular matrix. Then, the covariance matrix inverse is implicit in the backsolve line `u = (Ym/R)`. Clearly, multicollinearity or other data pathologies leading to very ill-conditioned covariance matrices may still determine very high condition estimates and, so, very poor Mahalanobis distance values (with almost random digits for condition numbers approaching the reciprocal of the machine epsilon).

With the new approach the Mahalanobis distances are computed, after the first step, with the updating formulae and the numerical stability is dominated by the matrix multiplications `Ym * S`. The QR decomposition is essentially reducible to matrix multiplications of the same complexity (Miller 1975). In addition, it can be shown (see e.g. Demmel, Dumitriu, and Holtz 2007) that if the matrix multiplication is numerically stable, then essentially all related

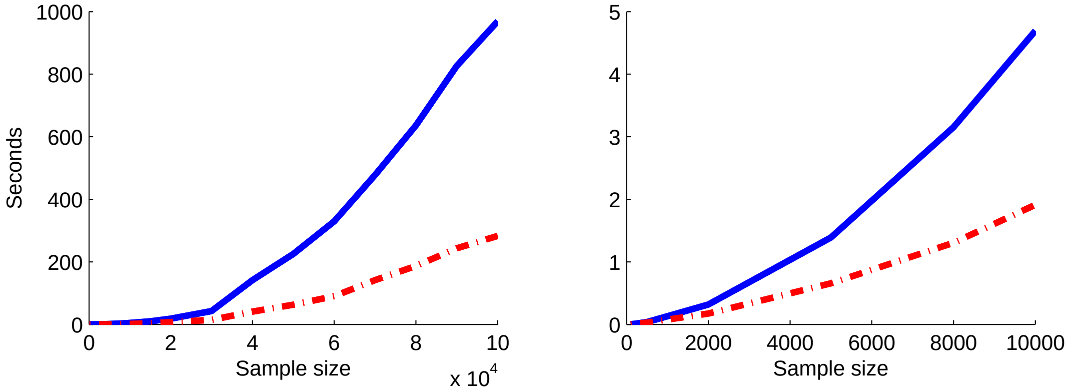


Figure 4: Comparison of computing time between the traditional (solid line) approach based on the qr decomposition and the new approach (dash-dotted line) based on recursive updating. The left panel considers a sample size which ranges from 100 to 100000, while the right panel is a zoom referred to sample sizes in the interval 100–10000.

linear algebra operations, including the `qr` decomposition, can also be done stably. In other words, although we have not assessed in detail the numerical stability of the two approaches (theoretically and empirically, by monitoring the condition estimates), it is reasonable to think that they are comparable, with a slight improvement provided by our updating formulae.

4. Conclusions

We have proposed some computational improvements to the forward search methodology, that are essential for practical use with the large datasets arising in modern application fields. The suggested improvements cover different features of the forward search code and include:

- fast subset updating;
- fast computation of deviance measures, such as Mahalanobis distances, using simple updating formulae;
- selection of the best mixture of linear indexing and logical indexing along the search;
- choice of the best alternative between the matrix multiplication operator `*.` and the MATLAB function `@bsxfun`;
- a new procedure for finding the kth order statistics, to be used when computation of repeated minima becomes too expensive.

In the previous sections we have shown the contribution of each of these modifications individually. It is also instructive to see the overall improvement provided by the new algorithms. Figure 5 contrasts the computation time of the currently available version of the software, the **FSDA** toolbox for MATLAB, with that obtained after implementation of all our new algorithms, for different sample sizes and $v = 2$. As expected, the gain is huge and, even more importantly, it increases with n. In fact, the overall time required by our algorithms is roughly a linear function of n, a remarkable improvement over the traditional ones.

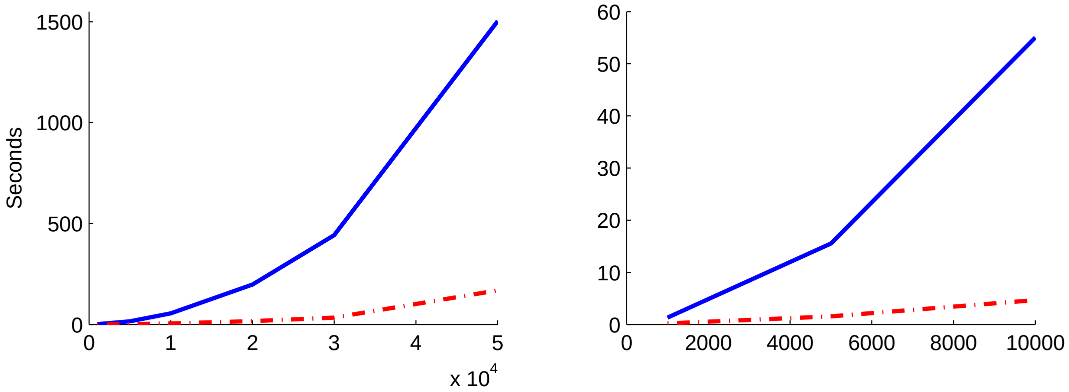


Figure 5: Comparison of computation time between the currently available version of the **FSDA** software (solid line) and the new routines proposed in this paper (dash-dotted line), when they are all implemented. The left panel considers a sample size which ranges from 1000 to 50000, while the right panel is a zoom referred to sample sizes in the interval 1000–10000.

Acknowledgments

This work was jointly supported by the project MIUR PRIN *MISURA – Multivariate models for risk assessment*, by the 2014-2020 JRC Multiannual Work Programme and by the OLAF-JRC project *Automated Monitoring Tool on External Trade*. The authors thank Dr. Ivano Azzini for his contribution to the development of Appendix C.

References

Atkinson AC, Riani M (2000). *Robust Diagnostic Regression Analysis*. Springer-Verlag, New York.

Atkinson AC, Riani M, Cerioli A (2004). *Exploring Multivariate Data with the Forward Search*. Springer-Verlag, New York.

Atkinson AC, Riani M, Cerioli A (2010). "The Forward Search: Theory and Data Analysis." *Journal of the Korean Statistical Society*, **39**(2), 117–134.

Blum M, Floyd RW, Pratt VR, Rivest RL, Tarjan RE (1973). "Time Bounds for Selection." *Journal of Computer and System Sciences*, **7**(4), 448–461.

Cerioli A, Perrotta D (2014). "Robust Clustering Around Regression Lines with High Density Regions." *Advances in Data Analysis and Classification*, **8**(1), 5–26.

Demmel J, Dumitriu I, Holtz O (2007). "Fast Linear Algebra is Stable." *Numerische Mathematik*, **108**(1), 59–91.

Dromey RG (1986). "An Algorithm for the Selection Problem." *Software-Practice & Experience*, **16**(11), 981–986.

Floyd RW, Rivest RL (1975). "Expected Time Bounds for Selection." *Communications of the ACM*, **18**(3), 165–172.

Hager W (1984). "Condition Estimates." *SIAM Journal on Scientific and Statistical Computing*, **5**(2), 311–316.

Hager W (1989). "Updating the Inverse of a Matrix." *SIAM Review*, **31**(2), 221–239.

Hoare CAR (1961). "Algorithm 64: Quicksort." *Communications of the ACM*, **4**(7), 321–322.

Hoare CAR (1971). "Proof of a Program: FIND." *Communications of the ACM*, **14**(1), 39–45.

Hubert M, Rousseeuw PJ, Aelst SV (2008). "High-Breakdown Robust Multivariate Methods." *Statistical Science*, **23**(1), 92–119.

Knuth DE (1981). *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. 2nd edition. Addison-Wesley.

Miller W (1975). "Computational Complexity and Numerical Stability." *SIAM Journal on Computing*, **4**(2), 97–107.

Press WH, Teukolski SA, Vetterling WT, Flannery BP (1992). *Numerical Recipes in C*. 2nd edition. Cambridge University Press.

Riani M, Atkinson AC, Cerioli A (2009). "Finding an Unknown Number of Multivariate Outliers." *Journal of the Royal Statistical Society B*, **71**(2), 447–466.

Riani M, Perrotta D, Torti F (2012). "**FSDA**: A MATLAB Toolbox for Robust Analysis and Interactive Data Exploration." *Chemometrics and Intelligent Laboratory Systems*, **116**, 17–32.

Torti F, Perrotta D, Francescangeli P, Bianchi G (2015). "A Robust Procedure Based on the Forward Search to Detect Outliers in Census Data." Submitted.

Yip E (1986). "A Note on the Stability of Solving a Rank-p Modification of a Linear System by the Sherman-Morrison-Woodbury Formula." *SIAM Journal on Scientific and Statistical Computing*, **7**(2), 507–513.

Zani S, Riani M, Corbellini A (1998). "Robust Bivariate Boxplots and Multiple Outlier Detection." *Computational Statistics & Data Analysis*, **28**(3), 257–270.

A. Linear indexing vs. logical indexing

During the forward search we extensively use the extraction of subsets. The purpose of this appendix is to compare linear extraction with logical extraction. In more detail, we investigate how the amount of time in logical extraction depends on the number of `true` or `false`, and how it relates to linear extraction. The execution of the code below

```
nsimul = 10000;          % Number of simulations to avoid random oscillations
n      = 20000;          % n = number of units
v      = 2;              % v = number of variables
nn     = 1000:1000:n;    % Range of subsample sizes
Y      = randn(n, v);     % Y = dataset
Time   = zeros(length(nn), 3);

for i = 1:length(nn)              % i is linked to the number of observations
    seq = randsample(n, nn(i)); % extract nn(i) values out of nn
    aa = 0; bb = 0; cc = 0;     % Reset timers at each fraction
    bsba = false(n, 1);         % Initialize arrays
    bsba(seq) = true;
    bsbb = true(n, 1);
    bsbb(seq) = false;
    for k = 1:nsimul
        % Logical extraction function of the number of true
        a  = tic;
        Y1 = Y(bsba, :);
        aa = aa + toc(a);
        % Logical extraction function of the number of false
        b  = tic;
        Y2 = Y(bsbb, :);
        bb = bb + toc(b);
        % Linear extraction
        c  = tic;
        Y3 = Y(seq, :);
        cc = cc + toc(c);
    end
    Time(i, :) = [aa bb cc];
end
```

produces the output given in Figure 6. The solid line is associated with `cc` in the code snippet above (amount of time necessary to compute index extraction), while the two other lines with square and diamonds markers refer to logic extraction as function of the number of `true` and `false`, respectively (`aa` and `bb` in the code snippet above).

The figure shows very clearly that if the number of elements to extract is smaller than 50% of the length of the matrix (in this case less than 10000) it is better to use linear indexing. On the other hand, when the percentage of elements to extract is greater than 50% it is better to use logical indexing. Finally, when the percentage of elements to extract is high, logical indexing based on the fraction of `false` is slightly quicker than logical indexing based on fraction of `true` and vice versa.

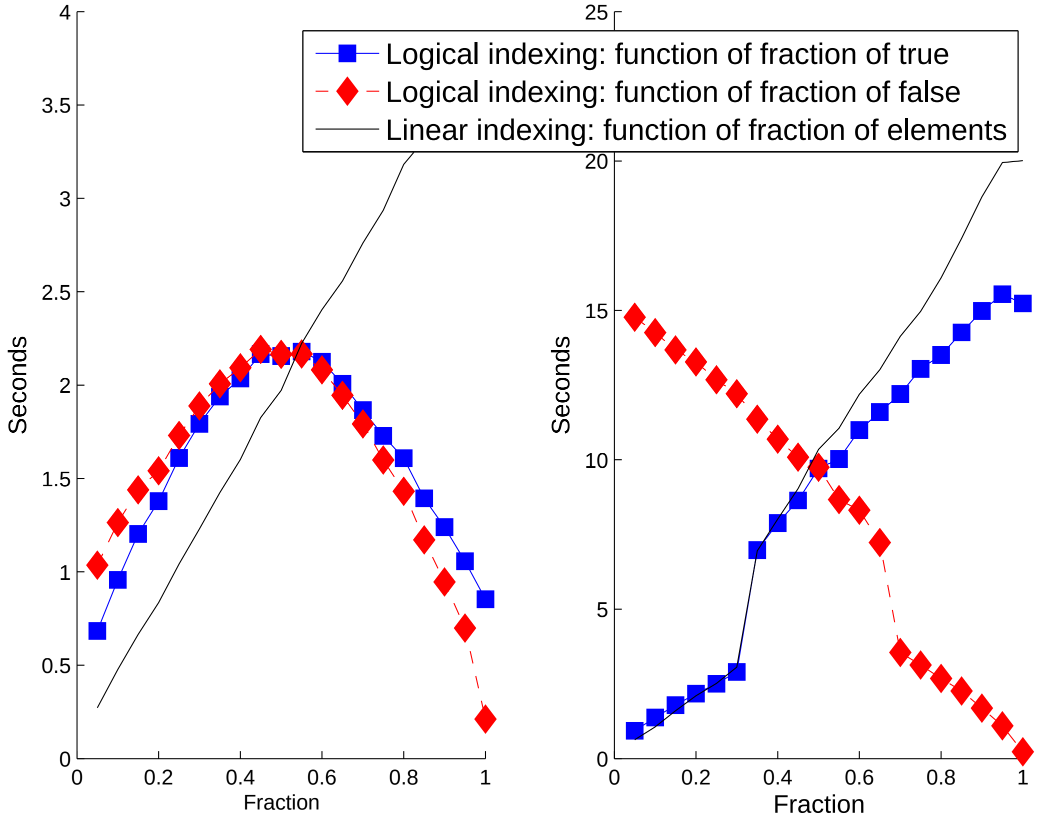


Figure 6: Analysis of the optimal proportion of using linear indexing vs. logical indexing as a function of proportion of `true` or `false`. The left panel is for $v = 2$, while the right panel is for $v = 10$.

The figure also shows a, perhaps surprising, change of behavior when moving from $v = 2$ to $v = 10$. Our explanation is that the internal memory representation of arrays (`MxArray`) can be organized in a single block only when the number of elements is small. In our application this happens when $v = 2$, where the symmetric shape of elapsed time is due to the fact that logical indexing processes only the elements (either `true` or `false`) which appear in the minority of the array cells. Otherwise, the internal memory representation requires several non-contiguous blocks and only the elements for which there is `true` are processed. The same feature also explains the discontinuity observed in the case $v = 10$. Indeed, the discontinuity point approaches the limit 0.5 when the analysis is performed on a 64-bit machine (not shown here), which can address larger blocks of memory.

B. Element multiplication based on $*$. or `@bsxfun`

In the forward search procedure, almost 20% of the overall computation time comes from the computation of the Mahalanobis distances of all the observations in each step. In MATLAB, the instructions given in block (12) for computing the Malahanobis distances can be

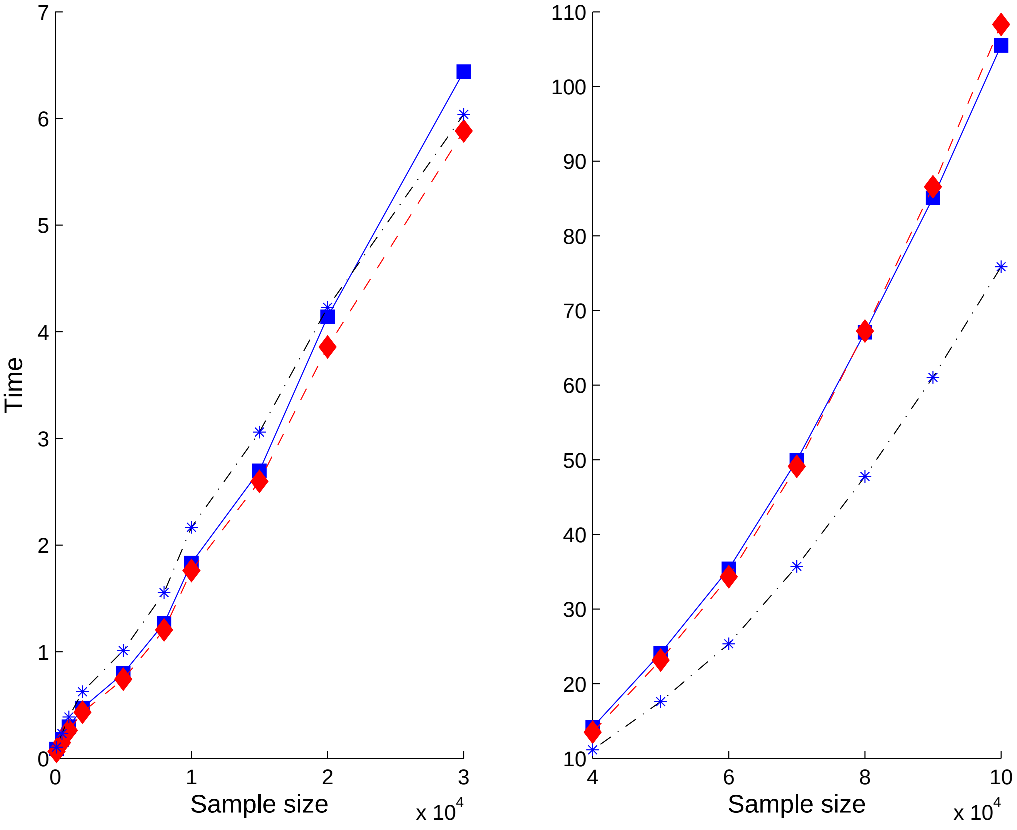


Figure 7: Analysis of computation time of three ways of multiplying matrices. The line with marker $*$ is associated with implementation based on Equation 13, while lines with square and diamond markers use, respectively, Equations 14 and 12.

implemented either as

```
MD = (m - 1) * sum(bsxfun(@times,mtimes(Ym, S), Ym), 2);
```
(13)

or, keeping into account that `*.` is equivalent to use instruction `times`, as

```
MD = (m - 1) * sum(times(mtimes(Ym, S), Ym), 2);
```
(14)

In this Appendix we compare the computation time of operation `*.` to that of (13) and (14). Figure 7 shows that up to a sample size of $n = 30000$ it is faster to use `*.`, while from $n = 30000$ onward it is much faster to use `bsxfun`.

Remark: According to the Mathworks technical support, the two procedures take a different computing time due to the so-called MATLAB accelerator, introduced from R2011b onwards[2]. Just if the accelerator is turned off using the undocumented instruction: `feature accelerator off` the computing time of the ways of multiplying matrices becomes comparable.

[2] For further details see http://blogs.mathworks.com/community/2011/12/19/matlab-startup-accelerator/.

n	NRC	**FSDA**
100	0.1043	0.0554
1000	0.3190	0.1896
5000	1.2391	0.9326
10000	2.3450	1.7673

Table 1: Average elapsed time in seconds of 10000 runs of the MATLAB porting of the Numerical Recipes in C algorithm `select` (NRC) and of our algorithm (**FSDA**). The data are generated to be normally distributed and ordered up to $n/2$, to mimic the framework of the forward search.

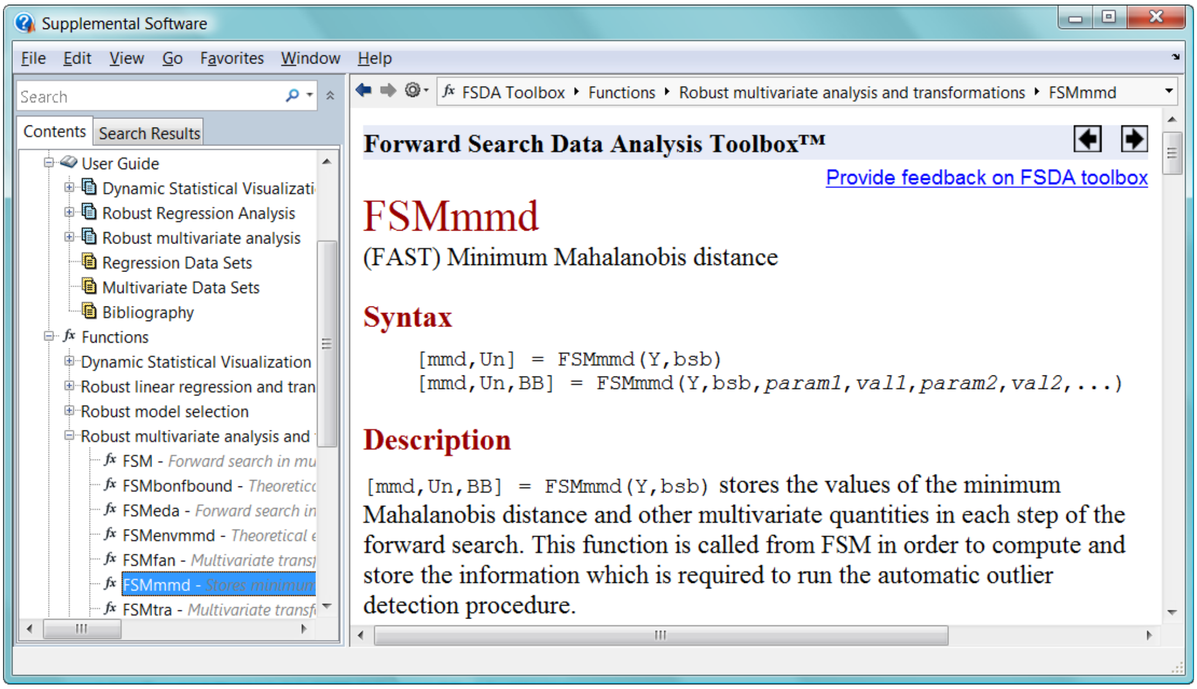
C. Procedure to find the kth order statistic

In computer science the problem of determining the kth smallest (or largest) element in a totally ordered set of n values is called *selection*. The literature on this fundamental problem is wide, with interesting historical roots (Knuth 1981, pp. 207–219), and covers algorithms aimed at finding the solution in $O(n)$ time, without sorting the entire set of values. A consolidated approach to the problem is an adaptation of *quicksort* (Hoare 1961, 1971; Dromey 1986) based on a partitioning, divide and conquer, process. In *quicksort* the idea is to move an element to its final position, say s, and at the same time rearrange the other elements so that those in positions 1 to $s-1$ will be smaller than those in positions $s+1$ to n. In this way, the same technique can be re-applied (recursively or iteratively) to the partitioned subsets. If we are only interested in the kth smallest element, then:

- if $s < k$ we look for the $(k-s)$th element of the "right side" subset,
- if $s > k$ we look for the kth element of the "left side" subset,
- if $s = k$ we stop partitioning and return the element in that position as result.

The complexity of the algorithms is usually measured in terms of the (maximum or average) number of comparisons or in-place exchanges between elements, or in terms of number of partition passes. Blum, Floyd, Pratt, Rivest, and Tarjan (1973) and Floyd and Rivest (1975) have shown that superior performances are obtained with strategies that can somehow choose the element around which partitioning is made (the so called pivot) close to the kth position. Typical strategies and complexity results for the selection problem assume random order of the elements in the set. However, in the forward search we have to deal with sets in which a progressively increasing subset is already sorted, at least to a considerable extent, and we look for the next smallest element to add to such a subset. Therefore, if we are at the forward search step m and we look for the $(m+1)$th order statistic, it is natural to use as pivot the index of the unit which has the minimum Mahalanobis distance outside the subset. With this idea in mind, we have implemented an iterative selection algorithm that has no freedom in the choice of the pivot.

To quantify the advantage of our fixed-pivot algorithm in the forward search context, we report in Table 1 the average elapsed time from 10000 applications on datasets of different sizes. The data are generated from a normal distribution. NRC stands for the implementation of algorithm `select` from Numerical Recipes in C (Press, Teukolski, Vetterling, and Flannery 1992, pp. 341–345), ported into MATLAB and run with random choice of the pivot. **FSDA** is



Supplemental Software
File Edit View Go Favorites Window Help
Search
Contents
Search Results
User Guide
Dynamic Statistical Visualizati
Robust Regression Analysis
Robust multivariate analysis
Regression Data Sets
Multivariate Data Sets
Bibliography
Functions
Dynamic Statistical Visualization
Robust linear regression and tran
Robust model selection
Robust multivariate analysis and
FSM - Forward search in mu
FSMbonfbound - Theoretica
FSMeda - Forward search in
FSMenvmmd - Theoretical e
FSMfan - Multivariate transf
FSMmmd - Stores minimum
FSMtra - Multivariate transf
FSDA Toolbox ▸ Functions ▸ Robust multivariate analysis and transformations ▸ FSMmmd

Forward Search Data Analysis Toolbox™

Provide feedback on FSDA toolbox

FSMmmd

(FAST) Minimum Mahalanobis distance

Syntax

```
[mmd,Un] = FSMmmd(Y,bsb)
[mmd,Un,BB] = FSMmmd(Y,bsb,param1,val1,param2,val2,...)
```

Description

`[mmd,Un,BB] = FSMmmd(Y,bsb)` stores the values of the minimum Mahalanobis distance and other multivariate quantities in each step of the forward search. This function is called from FSM in order to compute and store the information which is required to run the automatic outlier detection procedure.

Figure 8: HTML documentation of function FSMmmd. For those who use MATLAB R2012b+ the HTML help files can be found in the Supplemental Software tab which appears at the bottom of the Doc Center home page. Those who use releases of MATLAB earlier than 2012b can find the documentation in the same place as all the other official Mathworks toolboxes.

the new selection algorithm, where the pivot is fixed using the index of the unit associated with the minimum Mahalanobis distance outside the subset.

The code snippet of our fixed-pivot algorithm is reported below.

```
function kE = quickselectFS(A, k, kiniindex)
% Initialise the two sentinels
left    = 1;
right   = numel(A);

% if we know that element in position kiniindex is "close" to the desired
% order statistic k, than swap A(k) and A(kiniindex).
if nargin > 2
    Ak    = A(k);
    A(k)  = A(kiniindex);
    A(kiniindex) = Ak;
end

% pivot is chosen at fixed position k.
pivotIndex = k;

position = -999;
while ((left < right) && (position ~= k))
```

```
    pivot = A(pivotIndex);

    % Swap right sentinel and pivot element
    A(k)     = A(right);
    A(right) = pivot;

    position = left;
    for i = left:right
        if (A(i) < pivot)
            % Swap A(i) with A(position)!
            % A([i, position]) = A([position, i]) would be more elegant
            % but slower
            Ai          = A(i);
            A(i)        = A(position);
            A(position) = Ai;

            position = position + 1;
        end
    end

    % Swap A(right) with A(position)
    A(right)    = A(position);
    A(position) = pivot;

    if  position < k
        left  = position + 1;
    else % this is 'elseif pos > k' as pos == k cannot hold (see 'while')
        right = position - 1;
    end

end

kE = A(k);

end
```

When `rankgap` > 10 the routine at step m is called using the instruction

```
kE = quickselectFS(MD, m + 1, minMDindex)
```

D. HTML help files

We have added appropriate documentation to all routines which have been written. For example, all routines described in this paper have been included in function `FSMmmd` which can be found in the section “Robust multivariate analysis and transformations” of the **FSDA** toolbox (see Figure 8). The standard routine, which does not use the recursive implementation has been renamed `FSMmmdeasy` because it is much easier to follow and can help the novice to better understand how the procedure works. Finally, a file named "Installation-Notes.pdf", which describes what you should get when **FSDA** is installed manually by unpacking the compressed tar file FSDA.tar.gz, or automatically with our setup program for Windows platforms, can be downloaded from the web sites http://www.riani.it/MATLAB/ and http://fsda.jrc.ec.europa.eu/.

Affiliation:

Marco Riani, Andrea Cerioli
Dipartimento di Economia
Area di Statistica e Informatica
Università di Parma
Via Kennedy 6
43125 Parma, Italy
E-mail: mriani@unipr.it, andrea.cerioli@unipr.it

Domenico Perrotta
European Commission, Joint Research Centre
Institute for the Protection and Security of the Citizen
Global Security and Crisis Management Unit
Via E. Fermi 2749
21027 Ispra, Italy
E-mail: domenico.perrotta@ec.europa.eu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
October 2015, Volume 67, Code Snippet 1 *Submitted:* 2013-04-01
doi:10.18637/jss.v067.c01 *Accepted:* 2014-10-31