Reduction of Kernel Models

Mitchinson B., Dodd T. J., Harrison R. F.

Department of Automatic Control and Systems Engineering
The University of Sheffield
Mappin Street, Sheffield, UK, S1 3JD

Research Report 836

May 27, 2003

Abstract

Kernel models can be expensive to compute, and, in non-stationary environments, can become unmanageably large. Here, we present several previously reported techniques for reducing the complexity of these models in a common framework. This reformulation leads to the development of further related reduction techniques and clarifies the relationships between these and the existing techniques.

1 Introduction

A kernel model of the stationary Multiple Input Single Output (MISO) function $f(\mathbf{x})$ can be written

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{v}_i, \mathbf{x}) \tag{1}$$

where $\mathbf{x} \in \mathbb{X} \in \mathbb{R}^D$ is an input sample, $\{\mathbf{v}_i \in \mathbb{R}^D\}$ a set of expansion vectors, $\{k(\mathbf{v}_i, .)\}$ a corresponding set of kernel functions, and $\{\alpha_i \in \mathbb{R}\}$ a set of (unknown) multipliers. $N \geq 0$ is the cardinality of each of these sets.

We use, here, a regularised iterative Least Squares (LS) technique (Kernel Ridge Regression [1]) for building this model. It is given by

$$e_p = f(\mathbf{x}_p) - \hat{f}(\mathbf{x}_p) \tag{2}$$

$$\mathbf{v}_{N+1} = \mathbf{x}_p \tag{3}$$

$$\alpha_{N+1} = \eta e_p \tag{4}$$

$$\alpha_i = (1 - \eta\rho)\alpha_i \ , \quad i \in [1, N] \tag{5}$$

$$N \leftarrow N + 1 \tag{6}$$

with η a learning rate parameter and p the index of a newly presented sample. For a complete derivation see e.g. [1, 2]. An important feature of using this technique is that $\{\mathbf{v}_i \in \mathbb{X}\}$. This technique leads to a model complexity (N) that increases as training continues. Non-stationary problems require continual training, and thus N may increase without bound. Here, we are concerned with reducing the complexity to ensure the model remains computable (reducing N).

In Section 2, we summarise and relate some techniques for reducing kernel models. In Section 3 we compare their performances on some toy problems. In Section 4 we discuss their implementation. In Section 5 we summarise our conclusions and make recommendations for further investigation.

2 Set Reduction

We refer to reducing N as Set Reduction (SR). Any SR technique requires the removal of expansion vectors from the model, and thus has the potential to degrade the quality of the model. The approach used to minimise this degradation neatly dichotomises SR techniques, as follows.

Most simply, we can choose to remove only those expansion vectorsthat make small contributions to the model in $\mathbb{X}$; we call this, here, *Truncation*, after the *truncation error* of [3]. An alternative approach is to remove kernel functions that can be reasonably well represented as combinations of the kernel functions to be retained; we call this, here, Sparsity Control. Such techniques require the modification of the retained multipliers and tend, thus, to be more computationally demanding than truncation techniques.

In practice, we have found that a combination of Truncation and Sparsity Control minimises computational effort in achieving a given performance.

Truncation

Assuming $\alpha_i \neq 0$, discarding the ith vector from the expansion changes the model $\hat{f}(\mathbf{x})$. We say that the contribution of the ith vector is negligible if its removal (setting the ith multiplier to zero) has a negligible effect on the value of $\hat{f}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{X}$.

In practice, we will identify vectors that make negligible contributions by some indirect and computationally cheap means. We can choose those vectors with corresponding multipliers below some threshold on the grounds that a small multiplier will typically be associated with a small contribution to the model[1]. We call this, here, Magnitude Truncation (MT). If $\rho > 0$, (this can be viewed either as a regularisation or exponential forgetting parameter) all multipliers approach zero over time, and all vectors involved in the expansion will eventually have negligible contributions. In this case, we can choose to eliminate from the expansion those vectors that have been involved in the model for longer than some maximum time. We call this, here, Age Truncation (AT). A third approach is to set a hard limit on N, and to remove that expansion vector with the smallest value multiplier whenever this limit is exceeded; we call this Number Truncation (NT).

MT and AT are clearly closely related (for $\rho > 0$). Using MT tends to lead to a set of recently presented expansion vectors. Using AT tends to lead to a set of expansion vectors with non-small corresponding multipliers. This is illustrated in Figure 1 which shows a typical set of multipliers for a model trained in the way described above. We see that the set of vectors with age greater than 35 and that with corresponding multipliers less than 0.1 have a large intersection. Using small value multipliers to identify less important kernel functions has also been used in [4], though in a quite different context.

Sparsity Control

If, on the other hand, the contribution of the ith kernel function is non-negligible, simply discarding the ith vector will result in some (generally) undesirable degradation to the model. In this case, we may be able to distribute the contribution of the ith kernel function amongst the retained kernel functions by adjustment of the retained multipliers, to result in a model that is not degraded as much. We call the removal of kernel functions that can be reasonably well represented by a combination of the remaining kernel functions *sparsity control*; when the kernel function has local support, sparsity can be thought of as the reciprocal of the density of packing of expansion vectors.

We use $k_i(.) = k(\mathbf{v}_i, .)$ and $k_{ij} = k(\mathbf{v}_i, \mathbf{v}_j)$ for brevity, and thus write the kernel Gram matrix as

[1] At least for kernel functions with local support, a small multiplier implies a small contribution throughout $\mathbb{X}$.

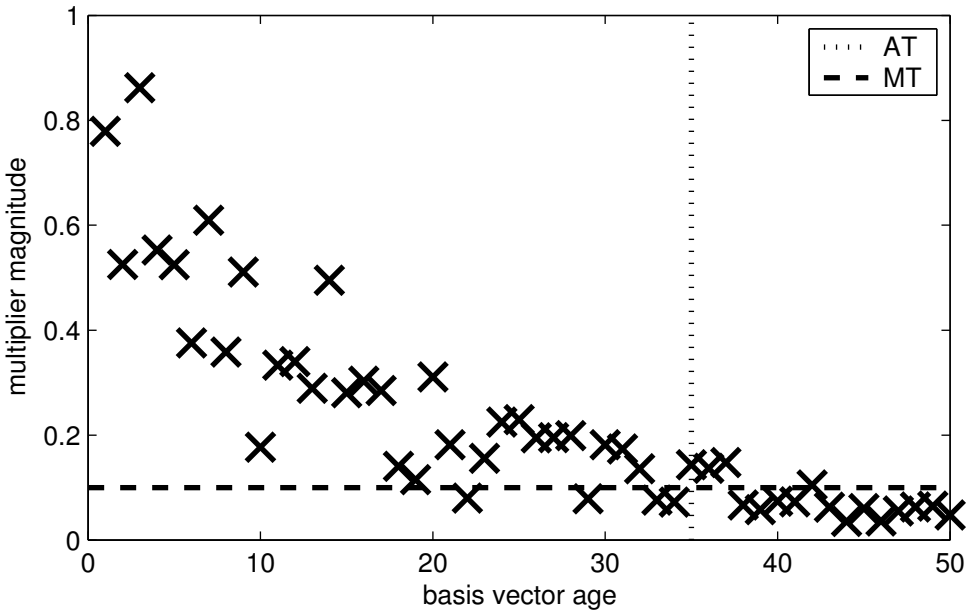

Figure 1: Illustration of truncation techniques.

$$K = \begin{bmatrix} k_{11} & k_{12} & . & k_{1N} \\ k_{21} & k_{22} & . & k_{2N} \\ . & . & . & . \\ k_{N1} & k_{N2} & . & k_{NN} \end{bmatrix} = K^T \tag{7}$$

where the superscript T indicates the transpose operation. Noting that we can always re-index the model such that the ith kernel function becomes the Nth, we consider the removal of the Nth kernel function only, without loss of generality. We write K in partitioned form

$$K = \left[\begin{array}{c|c} K_{\mathrm{RR}} & K_{\mathrm{RN}} \\ \hline K_{\mathrm{NR}} & k_{\mathrm{NN}} \end{array}\right] \tag{8}$$

After removal of the Nth kernel function, we can choose a new set of $N-1$ multipliers $\{\beta_i\}$ such that the reduced model

$$\hat{f}'(\mathbf{x}) = \sum_{i=1}^{N-1} \beta_i k(\mathbf{v}_i, \mathbf{x}) \tag{9}$$

is, in some sense, the best possible approximation to the original model. Ideally, the discrepancy $\delta(\mathbf{x}) = 0 \; \forall \; \mathbf{x} \in \mathbb{X}$ where

$$\delta(.) = \sum_{i=1}^{N} \alpha_i k(\mathbf{v}_i, .) - \sum_{i=1}^{N-1} \beta_i k(\mathbf{v}_i, .) \tag{10}$$

However, in general, this will not be possible and we will have to settle for approximate agreement between the old and new models at a finite set of locations in input space.

Since the expansion vectors $\{\mathbf{v}_i\}$ lie in $\mathbb{X}$, they are an obvious choice for this set of locations. We can write the discrepancy between the old and new models at the N expansion vectors as

$$\boldsymbol{\delta} = \begin{bmatrix} \boldsymbol{\delta}_R \\ \delta_N \end{bmatrix} = K\boldsymbol{\alpha} - \begin{bmatrix} K_{\mathrm{RR}} \\ K_{\mathrm{NR}} \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} \begin{bmatrix} K_{\mathrm{RR}} & K_{\mathrm{RN}} \end{bmatrix} \boldsymbol{\alpha} - K_{\mathrm{RR}}\boldsymbol{\beta} \\ \begin{bmatrix} K_{\mathrm{NR}} & k_{\mathrm{NN}} \end{bmatrix} \boldsymbol{\alpha} - K_{\mathrm{NR}}\boldsymbol{\beta} \end{bmatrix} \tag{11}$$

where $\boldsymbol{\alpha} = [\alpha_1 \ ... \ \alpha_N]^T$ and $\boldsymbol{\beta} = [\beta_1 \ ... \ \beta_{N-1}]^T$.

A reduction technique must define not only the new multipliers $\{\boldsymbol{\beta}\}$ in terms of the original model (a projection), but also a deterioration metric, λ, for the reduction, which measures how damaging to the quality of the model the reduction is. One adaptive strategy is to fix a maximum permissible deterioration, $\lambda_{\max}$, and remove a vector whenever this maximum deterioration is not exceeded; the model grows and shrinks as necessary. Another strategy, which may be more applicable when implementing the algorithm on hardware with hard resource limits, is to fix an upper limit for N, $N_{\max}$, and whenever N reaches this limit, to remove the kernel function that results in the minimum deterioration.

To illustrate, we write the deterioration metric $\lambda_{\mathrm{M}} = \alpha_N$ and the projection $\boldsymbol{\beta} = \begin{bmatrix} I & 0 \end{bmatrix} \boldsymbol{\alpha}$. For the strategy of fixed $\lambda_{\max}$, this defines MT; for fixed $N_{\max}$, it defines NT.

2.1 Techniques

The Kernel Least Mean Square (KLMS) technique described in [2] obtains the $\boldsymbol{\beta}$ which results in $\boldsymbol{\delta}_R = 0$, i.e. the reduced kernel model has the same output as the original model at the expansion vectors that are retained, and is erroneous only at the removed expansion vector. This projection is given by

$$\boldsymbol{\beta} = {K_{\mathrm{RR}}}^{-1} \begin{bmatrix} K_{\mathrm{RR}} & K_{\mathrm{RN}} \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} I & {K_{\mathrm{RR}}}^{-1} K_{\mathrm{RN}} \end{bmatrix} \boldsymbol{\alpha} \tag{12}$$

For this choice, we can write the remaining element of the discrepancy as

$$\begin{aligned} \delta_N &= \begin{bmatrix} K_{\mathrm{NR}} & k_{\mathrm{NN}} \end{bmatrix} \boldsymbol{\alpha} - K_{\mathrm{NR}} \begin{bmatrix} I & {K_{\mathrm{RR}}}^{-1} K_{\mathrm{RN}} \end{bmatrix} \boldsymbol{\alpha} \\ &= \begin{bmatrix} K_{\mathrm{NR}} & k_{\mathrm{NN}} \end{bmatrix} \boldsymbol{\alpha} - \begin{bmatrix} K_{\mathrm{NR}} & K_{\mathrm{NR}} {K_{\mathrm{RR}}}^{-1} K_{\mathrm{RN}} \end{bmatrix} \boldsymbol{\alpha} \\ &= \left(k_{\mathrm{NN}} - K_{\mathrm{NR}} {K_{\mathrm{RR}}}^{-1} K_{\mathrm{RN}}\right) \alpha_N \\ &= \kappa_N \alpha_N \end{aligned} \tag{13}$$

The deterioration metric is given by $\lambda_{\mathrm{KLMS}} = \kappa_N$.

Using the same projection, but the modified deterioration metric $\lambda_{\mathrm{MKLMS}} = \kappa_N \alpha_N^2$, gives the technique described in [5], which we call, here, Modified KLMS (MKLMS). This technique has a strong theoretical justification in terms of finding the unique orthogonal projection of the current model onto the space of models constructed by the removal of the kernel vector.

Another reasonable choice is that which minimises the norm of the discrepancy vector, $\|\boldsymbol{\delta}\|$, i.e. the reduced kernel model is the best LS match for the original kernel model (as measured at the original model expansion vectors). This projection is given by

$$\boldsymbol{\beta} = \left[\begin{array}{c} K_{\mathrm{RR}} \\ K_{\mathrm{NR}} \end{array} \right]^{\dagger} K\boldsymbol{\alpha} \tag{14}$$

where superscript † represents the Moore-Penrose pseudo-inverse. This may be better behaved than that of Equation 12 but is typically more expensive to compute. A sensible deterioration metric is the Mean Squared Error (MSE) as measured over all the bases of the original kernel model; i.e. $\lambda_{\mathrm{LS}} = \|\boldsymbol{\delta}\|^2/N$.

The Fast Kernel Least Mean Square (FKLMS) Rule [6] is applicable only for certain kernel functions. The following properties are sufficient to allow its use:

- $k_{ii} = 1$
- $k_i(.) \geq 0$
- $\|\mathbf{x}_a - \mathbf{v}_i\| > \|\mathbf{x}_b - \mathbf{v}_i\| \Rightarrow k_i(\mathbf{x}_a) < k_i(\mathbf{x}_b)$

The Gaussian kernel

$$k(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(\frac{\|\mathbf{x}_a - \mathbf{x}_b\|^2}{2\sigma^2}\right) \tag{15}$$

for instance, has these properties. We obtain FKLMS from KLMS by making the assumption that all the off-diagonal terms of K_{RR} are small. In this case, $K_{\mathrm{RR}} \approx I$, and we can rewrite the KLMS projection as

$$\boldsymbol{\beta} = \left[\begin{array}{cc} I & K_{\mathrm{RN}} \end{array} \right] \boldsymbol{\alpha} \tag{16}$$

The FKLMS deterioration metric is chosen as

$$\lambda_{\mathrm{FKLMS}} = 1 - \max_{i \in [1, N-1]} \{k_{iN}\} \tag{17}$$

In the case of fixed $\lambda_{\max}$, this restricts the off-diagonal elements of K_{RR} to be smaller than $1 - \lambda_{\max}$, fixing the validity of the assumption. In the case of fixed $N_{\max}$, each reduction step reduces the value of the maximum off-diagonal element of K_{RR}, which will tend to increase the validity of the assumption.

Analogous to the FKLMS algorithm, we can write a fast version of the MKLMS algorithm by choosing the deterioration

$$\lambda_{\mathrm{FMKLMS}} = \alpha_N^2 \left(1 - \max_{i \in [1, N-1]} \{k_{iN}\}\right) \tag{18}$$

and using the FKLMS projection. We call this Fast MKLMS (FMKLMS).

To illustrate these projection techniques, we choose a simple model given by $V = \{-2, -1, 0, 1, 2\}$, $\boldsymbol{\alpha} = [1, 1, 1, 1, 1]^T$, using the Gaussian kernel with $\sigma = 0.9$. We then remove the fourth of these vectors, $\mathbf{v}_4 = 1$. Figure 2 shows the output of the model after removal

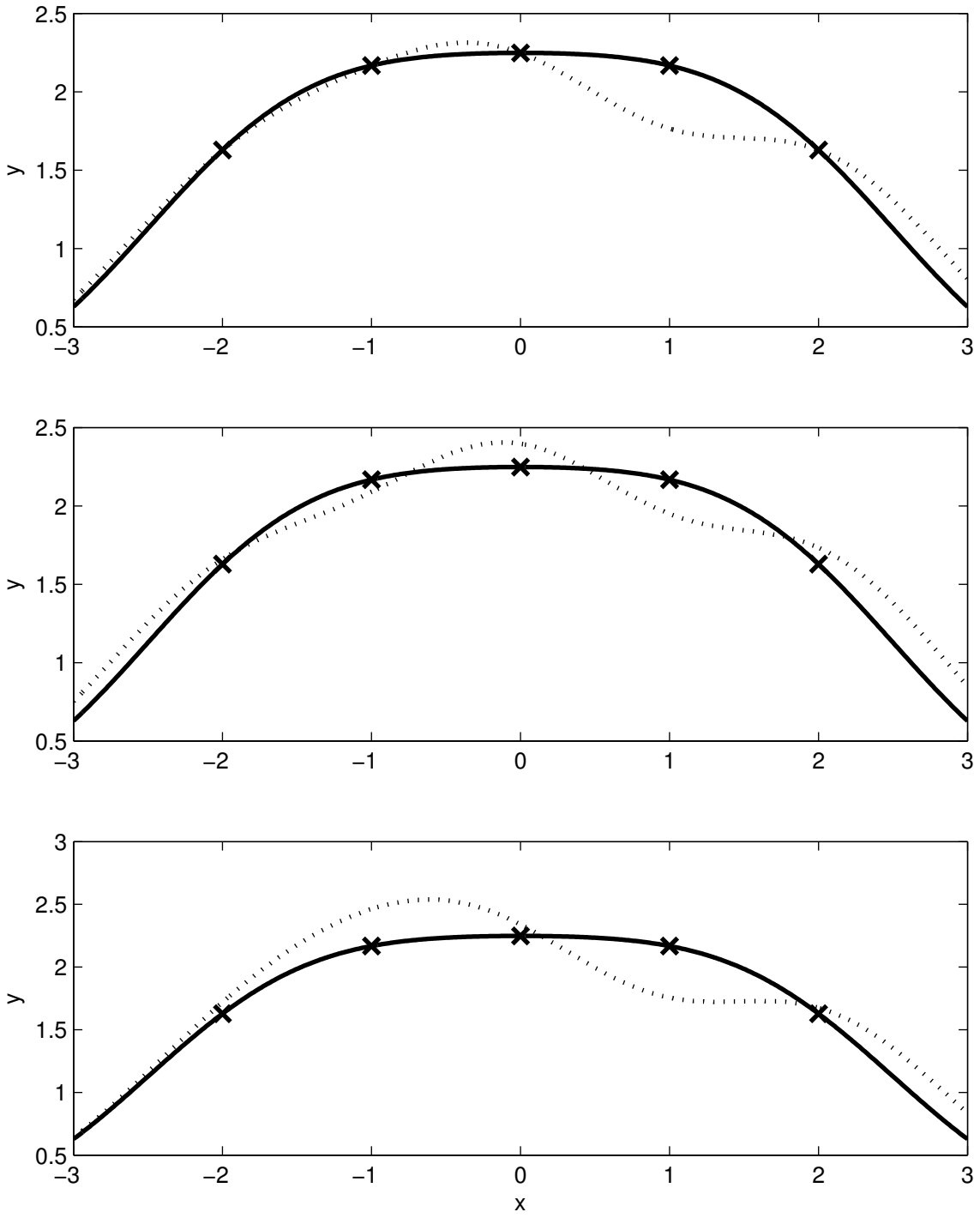

Figure 2: Illustration of (a) KLMS, (b) LS, and (c) FKLMS projections with strong coupling. Solid line in each case is output of original model with five expansion vectors. Expansion vectors are shown as crosses on this line. Output of model after removal of one vector shown as dotted line in each case.

and application of different projections. Note that the FKLMS projection is a fairly poor approximation to the KLMS projection in this case because the assumption of small off-diagonal terms in the Gram matrix is not fulfilled for this problem. We then repeat this

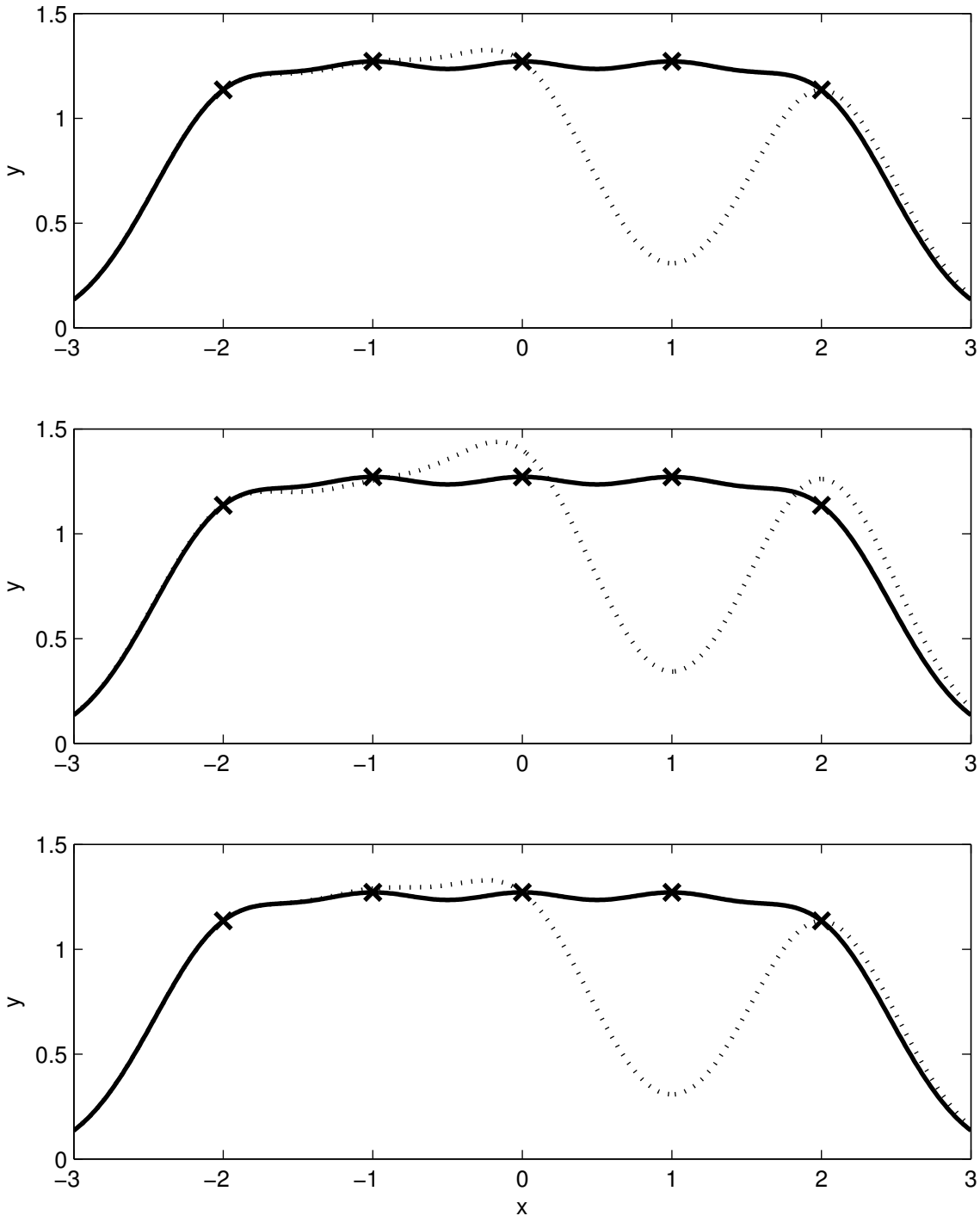

Figure 3: Illustration of (a) KLMS, (b) LS, and (c) FKLMS projections with moderate coupling. Solid line in each case is output of original model with five expansion vectors. Expansion vectors are shown as crosses on this line. Output of model after removal of one vector shown as dotted line in each case.

demonstration with $\sigma = 0.5$, and plot the results in Figure 3. In this case, the assumption holds well, and FKLMS is a good approximation to KLMS.

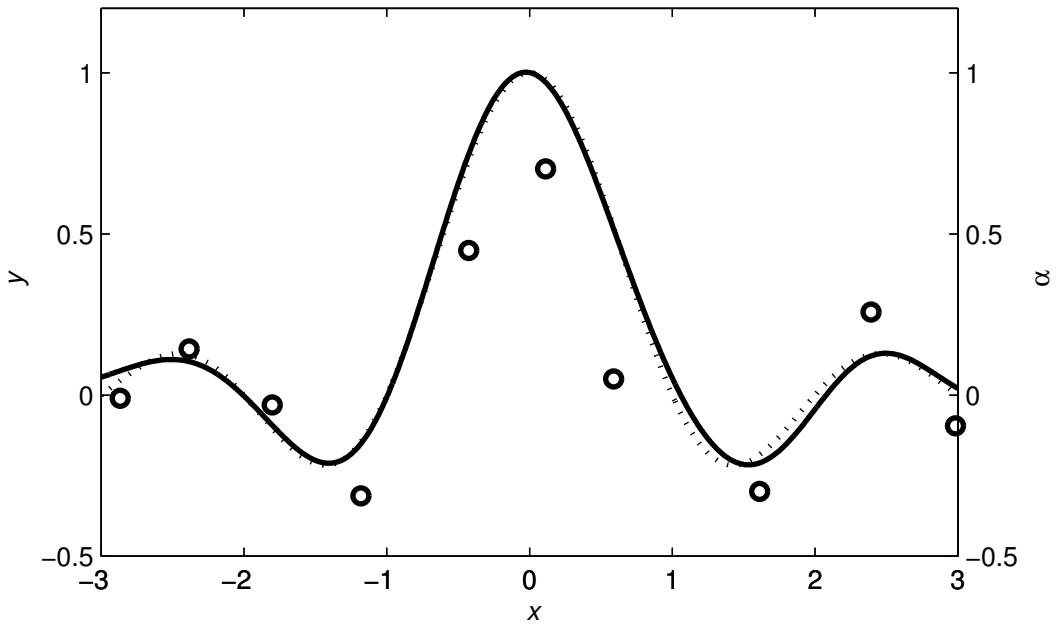

Figure 4: Fitting the sinc function (dotted line) using FKLMS (solid line). Vectors selected plotted against corresponding multipliers (circles).

3 Comparative Performance

To compare the performance of these algorithms, we use them to fit the (noiseless) function

$$f(x) = \frac{\sin(\pi x)}{\pi x} \tag{19}$$

with x in the domain $\mathbb{X} = [-3, 3]$. On the presentation of a new training sample x_p, we use the LS learning rule given above with $\eta = 0.1$ to update the model. In all cases, we use the reduction strategy of fixing $N_{\max}$; before the presentation step, if $N = N_{\max}$, we remove that expansion vector that results in the minimum deterioration. Figure 4 shows a typical result. Here we have used FKLMS, with 10 presentations of $N_{\text{tr}} = 100$ training examples and $N_{\max} = 10$. The selected expansion vectors are evenly distributed across the domain.

We repeat (using i.i.d. data) the above simulation 100 times for each of the algorithms with $N_{\max}$ chosen from $[4, 24]$, and record the final Mean Square Error (MSE) and the execution time in each case. We then take the mean of these results over the 100 realisations. The MSE is measured over 601 linearly spaced examples chosen from $\mathbb{X}$. We also perform 100 realisations setting $N_{\max} = 100$, in which case the projection involved is not relevant since it is never used; in other words, all the techniques tend to the same algorithm as $N_{\max} \to N_{\text{tr}}$. This *reference technique* amounts to not performing a reducing step.

Some of the results for execution time are plotted in Figure 5. Some results are excluded for clarity: The MKLMS and FMKLMS algorithms take very nearly the same amount of time to compute as the KLMS and FKLMS algorithms, respectively; this is as we expect. The execution time of FKLMS is the shortest by some margin, and increases approximately linearly with $N_{\max}$. KLMS takes longer, and LS is the slowest technique; both of these

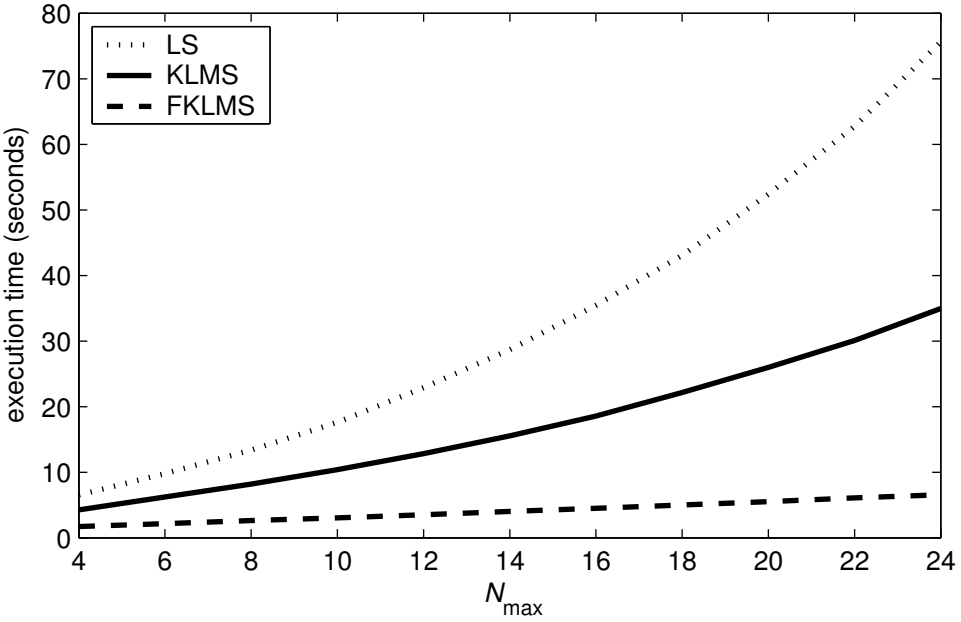

Figure 5: Execution time in seconds against $N_{\max}$ for each of the techniques.

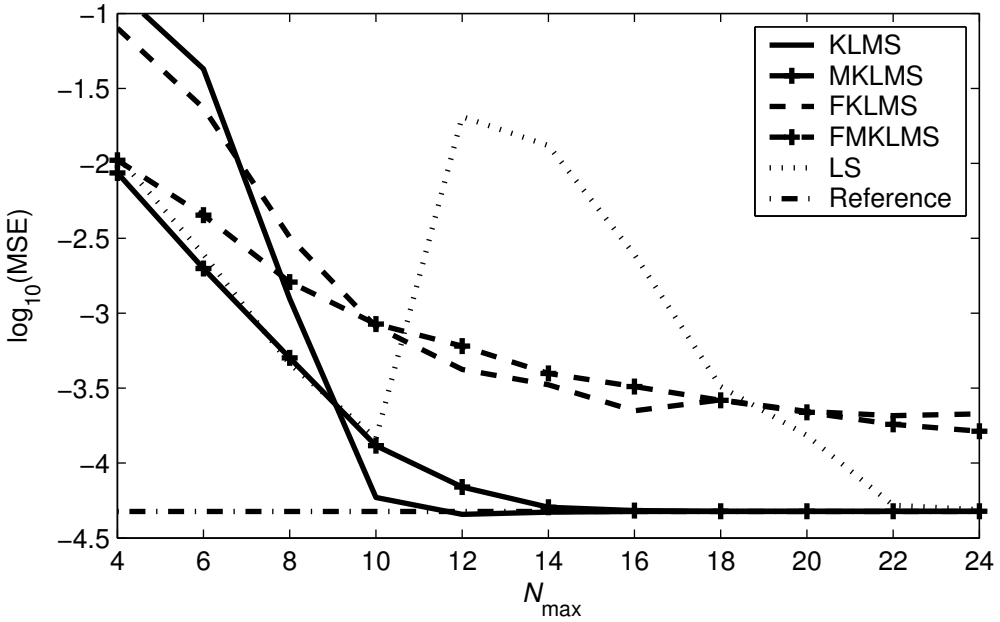

Figure 6: MSE against $N_{\max}$ for each of the techniques.

techniques have execution time increasing at a greater than linear rate with $N_{\max}$.

In Figure 6, we plot the MSE returned by some of the techniques along with the MSE returned by the reference technique ($\text{MSE}_{\text{REF}} = 4.76 \times 10^{-5}$). At $N_{\max} = 14$ and above, KLMS and MKLMS achieve similar performance, closely in line with that of the reference technique. In the same region, FKLMS and FMKLMS achieve similar performance, less good than the KLMS and MKLMS algorithms, but still very much useful. The main

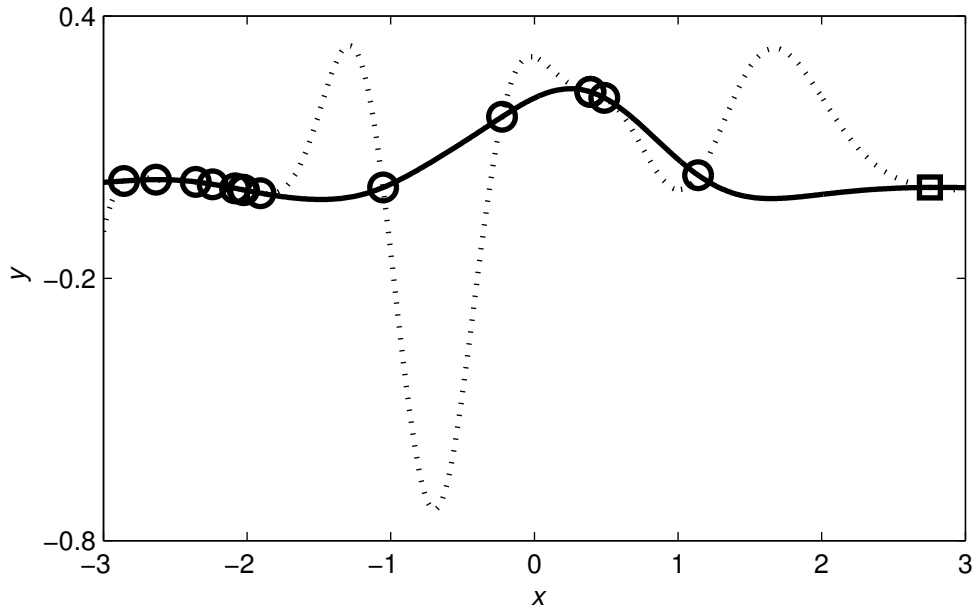

Figure 7: Poor performance of unregularised LS projection under adverse conditions.

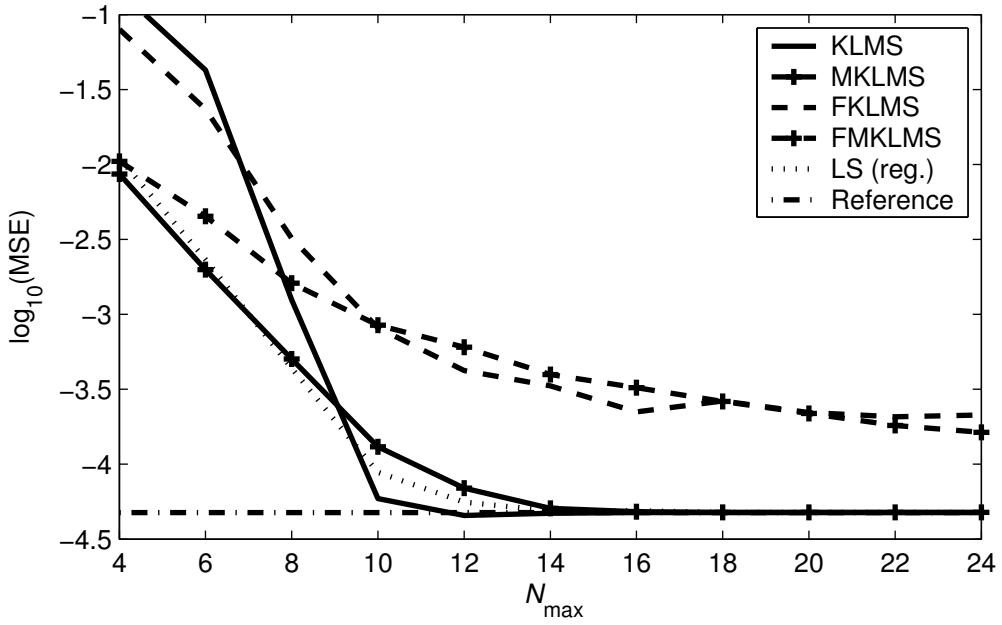

Figure 8: MSE against $N_{\max}$ for each of the techniques.

feature in the left half of the graph is that for both the fast and the original versions of the algorithms, MKLMS deals rather better with insufficient expansion vectors than does KLMS.

The LS technique is also shown in the figure, and shows performance in line with MKLMS, except in the range $N_{\max} \in [12, 20]$. Repeating some of these experiments and recording more detailed results revealed that the LS technique performs poorly under certain conditions. We graph a single LS projection in Figure 7. The circles and square taken

together mark the original expansion vectors; the solid line shows the output function of the machine. We then remove the rightmost expansion vector, marked with a square. The circles are thus the expansion vectors of the new machine; the dotted line shows its output function. Since none of the retained kernel functions has a strong response at the location of the removed expansion vector, very large changes of the retained multipliers are needed to produce small changes in the output function at that point. The LS projection chooses, thus, very large value multipliers (some $\sim 10^6$). This results in extremely poor performance out-of-sample (which, in this case, means at any point other than at an expansion vector), as shown by the output function of the reduced machine.

It is this effect that causes the anomalous results in Figure 6. We can counter this by adding an additional term to the LS deterioration metric that penalises the occurrence of large value multipliers after the projection[2]; it becomes

$$\lambda_{\mathrm{LS}} = \|\boldsymbol{\delta}\|^2/N + \frac{C}{N-1}\sum_{i=1}^{N-1}\beta_i^2 \tag{20}$$

which can be viewed as a regularised deterioration. Using this modified deterioration metric, and setting $C = 10^{-6}$, we repeat the above simulation for the LS technique only. The new results are plotted in Figure 8, and are similar to those of MKLMS.

4 Implementation

The implementations of MKLMS and KLMS are nearly identical; a single scalar multiplication per projection separates them. Fixing $N_{\max}$ requires computation of the deterioration metric for removal of each vector after each presentation. This requires the construction and inversion of N $(N-1)\times(N-1)$ matrices. However, since the matrix corresponding to each vector is K with a single row and column removed, K can be cached, effectively reducing the computation to N inversions of $N-1$ square matrices. This is an expensive computation, nonetheless, $\mathcal{O}(N^4)$.

Fixing $\lambda_{\max}$, a substantial reduction in complexity can be made by assuming that the removal of the most recently added vector will always be the lowest cost removal. This assumption does not hold in general, but is reasonably reliable in practice, and allows the reduction of N inversions to a single inversion at each presentation. Furthermore, since much of the matrix to be inverted is the same at each presentation, much of the inversion can be cached, reducing the computation from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$. Details of two techniques for achieving this reduction can be found in [2, 5].

The deterioration metric and projection of FKLMS involve only a single row of the K matrix; furthermore, the deterioration metric requires only examination of these values, and no computation beyond that which must be performed anyway in order to evaluate the machine at a new sample. This independence leads to a computational requirement of approximately $\mathcal{O}(N)$ for fixed $\lambda_{\max}$ and fixed $N_{\max}$ implementations (fixed $N_{\max}$ implementation has a slightly larger load of comparison operations, but no further floating point

[2]This is distinct from the regularisation by multiplier decay of Equation 5, which penalises large value multipliers during learning.

operations). For fixed $\lambda_{\max}$ implementation, FKLMS has a very low memory requirement; since no matrix operations are involved, the algorithm can be performed within little more than $N_{\max}(D+1)$ memory locations. KLMS and MKLMS, conversely, require the storage of the $(N-1) \times (N-1)$ Gram matrix and space to invert it, on top of the FKLMS requirement.

Our implementation of the LS technique has so far been restricted to a simple implementation of the algorithm using the MatLab `pinv()` function; we have not yet considered computational requirements or optimisations.

5 Conclusions and Recommendations

Casting the KLMS and MKLMS algorithms into a consistent framework reveals that they are closely related, despite quite distinct derivations. Casting the FKLMS algorithm into the same framework clarifies its relationship with KLMS, and leads immediately to a Fast version of the MKLMS algorithm, by analogy. The LS projection arises out of the same framework. This commonality will facilitate more extensive experimental and theoretical comparisons between the algorithms in future.

Also, we have highlighted the independence between the technique used to learn a kernel model (here, LS supervised learning) and the technique(s) used to reduce its complexity. Furthermore, we can see that the deterioration metric and projection of the reduction technique are also independent (theoretically, if not necessarily computationally). This frees us to mix components to form a complete algorithm, whilst clearly identifying the context in which we might invent low complexity approximations to the theoretically well-founded but sometimes computationally expensive techniques.

The LS projection, introduced here, has not been investigated in any depth. Intuition suggests that it should match or exceed the performance of the KLMS/MKLMS projections when the performance measure is square error. Results herein are inconclusive, but suggest that this is the case. To discover whether this is true in general will require more work. A new approximate technique (FMKLMS) is also very briefly introduced above; this warrants investigation, given the successful application of the FKLMS algorithm [6].

We also note that application of the approximation $K_{\mathrm{RR}} \approx I$ to the deterioration metric of KLMS (to give $\lambda = k_{\mathrm{NN}} - \|K_{\mathrm{NR}}\|^2$) as well as to the projection yields an alternative fast approximation to the KLMS algorithm. An analogous alternative fast approximation to the MKLMS also exists. These algorithms have not yet been investigated.

Finally, performance on the fitting problem indicates that the performance of MKLMS degrades less than that of KLMS when the number of expansion vectors used is insufficient to represent the function well. The reasons for this are not immediately obvious, and should be uncovered.

References

[1] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, 2000.

[2] P. M. L. Drezet. *Kernel Methods and their Application to Systems Identification and Signal Processing.* PhD thesis, The University of Sheffield, United Kingdom, June 2001.

[3] B. Schölkopf and A. J. Smola. *Learning With Kernels.* The MIT Press, 2002.

[4] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse least squares support vector machine classifiers. In *Proceedings of the European Symposium on Artificial Neural Networks, Bruges, Belgium (ESANN 2000)*, pages 37–42, April 2000.

[5] T. J. Dodd, V. Kadirkamanathan, and R. F. Harrison. Function estimation in Hilbert space using sequential projections. In *Proceedings of ICONS, Portugal*, 2003.

[6] B. Mitchinson. *Iterative Kernel Techniques in Application to Channel Equalisation.* PhD thesis, The University of Sheffield, United Kingdom, September 2002.