# KAGE: A NEW FAST RLS ALGORITHM

*I. D. Skidmore, I. K. Proudler.*

DERA, St. Andrews Road, Malvern, Worcs, WR14 3PS, United Kingdom.
skidmore@signal.dera.gov.uk ; proudler@signal.dera.gov.uk

## ABSTRACT

A new fast Recursive Least Squares (RLS) algorithm is introduced. By making use of RLS interpolation as well prediction, the algorithm generates the transversal filter weights without suffering the poor numerical attributes of the FTF algorithm. The Kalman gain vector is generated at each time step in terms of interpolation residuals. The interpolation residuals are calculated in an order recursive manner. For an $N$th order problem the procedure requires O($N$log$N$) operations. This is achieved via a divide and conquer approach. Computer simulations suggest the new algorithm is numerically robust, running successfully for many millions of iterations.

## 1. INTRODUCTION

Adaptive filtering algorithms have numerous applications in fields such as telecommunications, radar, sonar and speech processing [1]. Recursive Least Squares (RLS) algorithms generate the least squares optimal solution to the adaptive filtering problem at each time instant and are therefore highly desirable. We consider here the special case of the single channel adaptive filtering problem.

Conventional RLS algorithms require $O(N^2)$ operations per time step which may be prohibitive for real time implementation of large adaptive filters. Fast algorithms exist for the single channel problem which require just $O(N)$ operations per time step. (The reduction is attained by utilising LS prediction.) These can be split into two categories: Fast Transversal Filter (FTF) algorithms and Least Squares Lattice (LSL) algorithms [1].

LSL lattice algorithms output the RLS filtering residual but not the LS transversal filter weights directly. They can be implemented in a numerically stable manner. FTF algorithms generate the transversal filter weights. However, they are notoriously unstable in finite precision implementations [2] and so usually considered unsuitable for real world application.

For some applications, such as acoustic echo cancellation, it is sufficient for an algorithm to output the filtering residual. For others, such as system identification, the adaptive filter weights are also required.

The KaGE (**Ka**lman **G**ain **E**stimator) RLS algorithm is a new fast RLS algorithm which generates the Kalman gain vector, and therefore the transversal filter weights, in a novel, numerically robust manner, using $\mathrm{O}(N\log_2 N)$ operations per time step. This is achieved by making use of RLS interpolation as well as prediction.

In this paper we define RLS interpolation; express the Kalman gain vector in terms of interpolation residuals; and present an $\mathrm{O}(N\log_2 N)$ method for generating this vector.

### 1.1. RLS Interpolation

Define $e_{p,\,f}(t)$ as the least squares interpolation residual for a time series element $x(t)$ using $p$ past and $f$ future samples. That is:

$$e_{p,\,f}(t) = x(t) - \underline{\hat{w}}^T_{p,\,f}(t)\underline{x}_{p+f}(t+f,\,t) \qquad (1)$$

where $\underline{\hat{w}}^T_{p,\,f}(t)$ is a vector of length $p+f$ chosen according to the least squares criteria. The vector $\underline{x}_{p+f}(t+f,\,t)$ is a modified data vector consisting of all the elements of the conventional single channel data vector at time $(t+f)$: $\underline{x}_{p+f}(t+f) = [x(t+f), \ldots, x(t-p)]^T$ **except** for the element $x(t)$.

Define

$$\underline{w}^T_{p,\,f}(t) = [-\hat{w}_1(t)\ldots, -\hat{w}_f(t), 1 \;, -\hat{w}_{f+1}(t), \ldots, -\hat{w}_{f+p}(t)] \qquad (2)$$

where the $\hat{w}_i(t)$ are the elements of $\underline{\hat{w}}_{p,\,f}(t)$.

Equation (1) can be re-written as

$$e_{p,\,f}(t) = \underline{x}^T_{p+f+1}(t+f)\underline{w}_{p,\,f}(t)\,. \qquad (3)$$

The unknown elements of the least squares interpolation error weight vector $\underline{w}_{p,\,f}(t)$ are defined by the following rearrangement of the augmented normal equations [1]

$$M_{p+f+1}(t+f)\underline{w}_{p,\,f}(t) = \begin{bmatrix} \underline{0}_f \\ E_{p,\,f}(t) \\ \underline{0}_p \end{bmatrix} \qquad (4)$$

where $M_n(t) = \sum_{i=0}^{t} \lambda^{t-i}\underline{x}_n(i)\underline{x}^T_n(i)$ is the data covariance matrix, and $E_{p,\,f}(t)$ is the least squares interpolation residual power associated with $\underline{w}_{p,\,f}(t)$.

Define also the *a priori* interpolation residual

$$\bar{e}_{p,\,f}(t) = \underline{x}^T_{p+f+1}(t+f)\underline{w}_{p,\,f}(t-1)\,. \qquad (5)$$

| Stage | Operations at time $t$. | Computational Cost |
|---|---|---|
| 1 | Input $x(t)$. | |
| 2 | Generate $\{e_i^f(t), e_i^b(t) : (1 \le i \le N-1)\}$ using LS Lattice. | $(N-1)(14M + 6A + 4D)$ |
| 3 | Generate $\hat{\underline{k}}^N(t)$ using divide and conquer approach. | $(N(\log_2(N) - 2) + 2)(13M + 6A + 3D)$ |
| 4 | Update normalisation coefficients using equation (18). | $N(1M + 1A + 0D)$ |
| 5 | Generate $\underline{k}_N(t)$ using equation (10). | $N(1M + 0A + 1D)$ |
| 6 | Update filter weights using equation (6). | $N(2M + 2A + 0D)$ |

**Table 1: Full algorithm structure.**

## 2. KALMAN GAIN VECTOR

For the general $N$th order RLS adaptive filtering problem, the Kalman gain vector $\underline{k}_N(t)$ is the vector required for updating the LS adaptive filter weights at each time instance. That is

$$\underline{w}_N(t) = \underline{w}_N(t-1) + \bar{e}_N(t)\underline{k}_N(t) \tag{6}$$

where $\bar{e}_N(t)$ is the *a priori* filtering residual given by $\bar{e}_N(t) = y(t) - \underline{w}_N^T(t-1)\underline{x}_N(t)$ and $y(t)$ is the desired signal. By definition

$$\underline{k}_N(t) = M_N^{-1}(t)\underline{x}_N(t). \tag{7}$$

Consider $k_i^N(t)$, the $i$th element of $\underline{k}_N(t)$

$$\begin{aligned} k_i^N(t) &= [\underline{0}_{i-1}^T, 1, \underline{0}_{N-i}^T]\underline{k}_N(t) \\ &= [\underline{0}_{i-1}^T, 1, \underline{0}_{N-i}^T]M_N^{-1}(t)\underline{x}_N(t) \end{aligned} \tag{8}$$

From equation (4) we then have that

$$k_i^N(t) = \frac{\underline{w}_{N-i,i-1}^T(t-i+1)\underline{x}_N(t)}{E_{N-i,i-1}(t-i+1)} \tag{9}$$

and it follows from equation (3) that

$$k_i^N(t) = \frac{e_{N-i,i-1}(t-i+1)}{E_{N-i,i-1}(t-i+1)}. \tag{10}$$

Equation (10) is crucial in what follows. It states that the Kalman gain vector can be calculated as a particular set of normalised LS interpolation residuals.

## 3. GENERATING THE KALMAN GAIN VECTOR

It is possible to derive a lattice like structure to generate RLS interpolation residuals recursively in both time and in order [3]. Summarising the pertinent information in [3], it is possible to define two functions with the following input/output relationships:

$$e_{p+1,f}(t-f) = \text{incp}(e_{p,f}(t-f), e_{p+f+1}^b(t)) \tag{11}$$

$$e_{p,f+1}(t-f-1) = \text{incf}(e_{p,f}(t-f), e_{p+f+1}^f(t)). \tag{12}$$

where $e_{p+f+1}^f(t)$ and $e_{p+f+1}^b(t)$ are respectively the forwards and backwards LS prediction residuals of order $p+f+1$ at time $t$.

Note that the outputs of a LSL algorithm are prediction residuals, *i.e.* interpolation residuals with either $p$ or $f$ equal to zero. The set of residuals $\{e_{i,0}(t), e_{0,i}(t-i) | 0 \le i \le n\}$ is output by an $n$th order LSL algorithm. Hence any interpolation residual can be generated using either of equations (11) or (12) starting from the outputs of a LSL of sufficient order.

Define the $N$th order un-normalised Kalman gain vector $\hat{\underline{k}}^N(t)$ with elements

$$\{\hat{k}_i^N(t) = e_{N-i,i-1}(t-i+1) | 1 \le i \le N\} \tag{13}$$

Using the outputs of a LSL of order $N-1$ it is then possible to generate each element of $\hat{\underline{k}}^N(t)$. For example, to calculate $\hat{k}_j^N(t) = e_{N-j,j-1}(t-j+1)$ for $1 < j < N$ one could generate the following sequence of parameters:

$$\begin{aligned} e_{0,j-1}(t-j+1) &\rightarrow e_{1,j-1}(t-j+1) \\ &\rightarrow \ldots \rightarrow e_{N-j,j-1}(t-j+1) \end{aligned}$$

via repeated use of (11). Calculation of all $N$ elements of $\hat{\underline{k}}^N(t)$ in this way requires a total of $O(N^2)$ operations. Equally, $\hat{k}_j^N(t)$ could be calculated by repeated use of equation (12) via:

$$\begin{aligned} e_{N-j,0}(t) &\rightarrow e_{N-j,1}(t-1) \\ &\rightarrow \ldots \rightarrow e_{N-j,j-1}(t-j+1) \end{aligned}$$

To calculate all $N$ elements of $\hat{\underline{k}}^N(t)$ in this method would also require $O(N^2)$ operations.

### 3.1. Divide and Conquer

The KaGE RLS algorithm uses a divide and conquer approach to reduce the operation count to $O(N\log_2 N)$. The scheme is described for filter lengths $N = 2^k$ where $k \in \mathbb{N}$. Problems with other filter lengths can be solved via the same basic principle.

Consider the following set of RLS interpolation residuals:

$$I = \{e_{p,f}(t-f) | (p,f) \in \mathbb{N}^2\}. \tag{14}$$

This can be considered as a two dimensional "interpolation space" of discrete points. Note that the elements of $\hat{\underline{k}}^N(t)$ as well as the prediction residuals $e_{0,n}(t-n)$ and $e_{n,0}(t)$ are all contained by the set $I$. Note also that at time instant $t$ all elements of this set are realisable.

Figure 1a shows the residuals that form the un-normalised Kalman gain vector $\hat{\underline{k}}^N(t)$, defined by equation (13), within this space. These are denoted by the diagonal line. Each point in the space represents an interpolation residual. The horizontal position of a point on the diagram represents

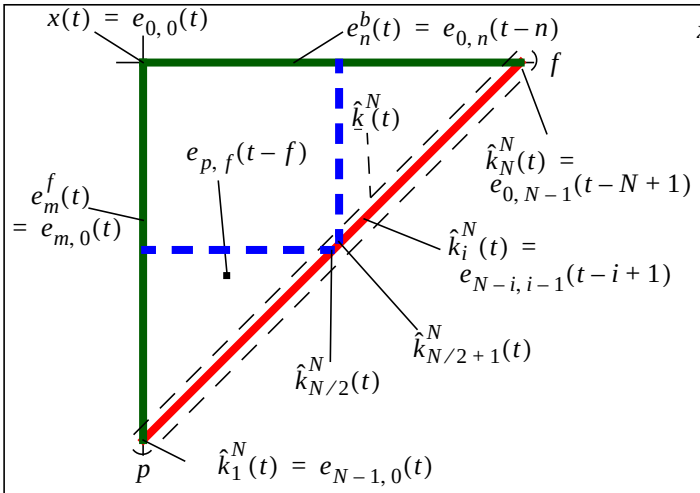


**Figure 1a: Interpolation Space**

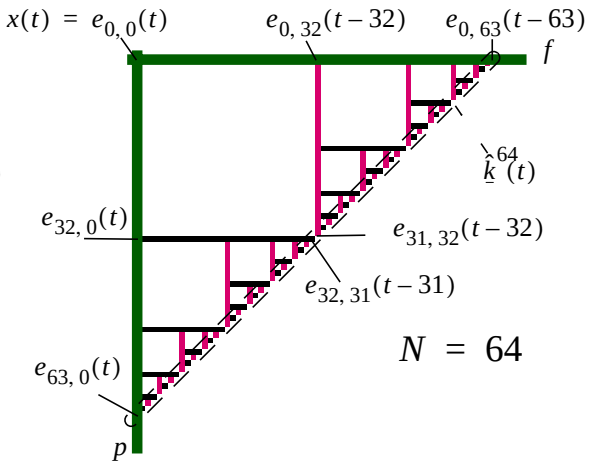


**Figure 1b: $O(N\log_2 N)$ structure for $N = 64$.**

the number of future samples of the interpolation residual. The vertical position represents the number of past samples of the interpolation residual.

Also shown are the positions of the outputs of a conventional LS prediction lattice within the interpolation space. The solid horizontal line depicts the positions of the backwards prediction residuals and the solid vertical line depicts the positions of the forwards prediction residuals.

Given a residual in this space, equations (11) and (12) allow the generation of the residuals either immediately below or to the right on the diagram.

The divide and conquer method works as follows: starting from known residuals output by the LSL generate the two central elements of $\hat{\underline{k}}^N(t)$ using equations (11) and (12). The particular set of residuals to be calculated at this stage is marked by dashed lines on figure 1a. What remains are two sub-problems of order $N/2$. These are then each divided into two further sub-problems of size $N/4$, and so on. We now describe the full process more precisely.

Consider an isosceles, right angled triangle in $I$, with catheti of length $n$, oriented as in figure 1a. Define a vector function $\underline{G}$ which at a given time instant calculates the residuals on the diagonal of the triangle from known residuals on the catheti of the triangle and the set of forwards and backwards prediction residuals. Let the function be expressed in terms of input parameters $\underline{G}(n, \underline{H}, \underline{V}, p, f, \underline{e}^f, \underline{e}^b)$. The parameters are defined as follows:

$n$: The order of the problem in hand.

$\underline{H}$: The vector of length $n$ of known interpolation residuals on the horizontal cathetus.

$\underline{V}$: The vector of length $n$ of known interpolation residuals on the vertical cathetus.

$p$, $f$: the coordinates in $I$ of the right angle of the triangle.

$\underline{e}^f$, $\underline{e}^b$: the vectors of forward and backward prediction residuals respectively.

Let the elements of each of these vectors be indexed from 0. Then for a problem of order $2n$, $n \neq 1$, the output vector can be expressed in terms of lower order vectors by

$$\underline{G}(2n, \underline{H}, \underline{V}, p, f, \underline{e}^f, \underline{e}^b) = \begin{bmatrix} \underline{G}(n, \underline{H}', \underline{V}(n, \ldots, 2n-1), (p+n), f, \underline{e}^f, \underline{e}^b) \\ \underline{G}(n, \underline{H}(n, \ldots, 2n-1), \underline{V}', p, (f+n), \underline{\bar{e}}^f, \underline{\bar{e}}^b) \end{bmatrix} \quad (15)$$

where the vectors $\underline{H}'$ and $\underline{V}'$ are vectors of length $n$ calculated by the following

$H_0' = V_n$; $H_i' = \text{incf}(H_{j-1}', e_{p+f+n+j}^f)$ for $1 \le j < n$

$V_0' = H_n$; $V_j' = \text{incp}(V_{j-1}', e_{p+f+n+j}^b)$ for $1 \le j < n$.

The case $n = 2$ is of course trivial. The diagonal consists of the two elements $V_1$ and $H_1$.

$$\underline{G}(2, \underline{H}, \underline{V}, p, f, \underline{e}^f, \underline{e}^b) = [V_1, H_1]^T \quad (16)$$

The un-normalised Kalman gain vector of length $N$ can be expressed as

$$\hat{\underline{k}}^N = \underline{G}(N, \underline{e}_N^b, \underline{e}_N^f, 0, 0, \underline{e}_N^f, \underline{e}_N^b). \quad (17)$$

If $N \in \{2^i \mid i \in \mathbb{N}\}$ then from equation (15) it is clear that the vector can be calculated entirely via a recursive method. The procedure is illustrated for $N = 64$ in figure 1b.

## 4. FULL ALGORITHM

Equation (10) states that the $i$th element of $\underline{k}_N(t)$, can be expressed as an interpolation residual $e_{N-i,i-1}(t-i+1)$ divided by that interpolation residual's power $E_{N-i,i-1}(t-i+1)$. The standard time recursion for the RLS residual power [1] applied to $E_{p,f}(t)$ is

$$E_{p,f}(t) = \lambda E_{p,f}(t-1) + \bar{e}_{p,f}(t-f)e_{p,f}(t-f) \quad (18)$$

This recursion must be applied to each of the $N$ residuals which form $\hat{\underline{k}}^N(t)$. At each time step the elements of $\underline{k}_N(t)$ are then available via equation (10). The filter weights can therefore be updated via equation (6).

The full $N$th order algorithm can be split into several concatenated stages. A breakdown of the full procedure is given in

table 1. The total operation count where $M$ is the number of multiplies, $A$ additions and $D$ divides is

$$((13\log_2 N - 8)N + 12)M + ((6\log_2 N - 3)N + 6)A + ((3\log_2 N - 1)N + 2)D \quad (19)$$

which for large $N$ is dominated by $O(N\log_2 N)$ terms.

### 4.1. Numerical properties

As with a conventional LSL algorithm, there are several possible implementations of the KaGE algorithm. In the present work we have used an implementation based on Square Root Free (SRF) QR-RLS [4]. SRF QR-RLS was chosen for its good numerical properties plus ease of initialization. The standard recursions are well known and are guaranteed stable. Each instantiation of these recursions solves a first order LS problem.

To implement the KaGE algorithm a set of modified first order LS problems must also be solved. Instead of the conventional two inputs, the modified problem has as inputs one of the conventional inputs and the conventional output. From this we must calculate the other conventional input. A modified set of update recursions ensue. (Note that this is not a LS downdate.)

The modified recursions have slightly degraded numerical properties. One stored quantity in each instantiation of the recursions is simply an accumulation of the product of input terms. Consequently this term exhibits linear error growth with time. This in turn leads to linear error growth in $\hat{\underline{k}}^N(t)$. This is in contrast to the conventional FTF algorithm which exhibits exponential error growth. The Stabilised FTF algorithm has better initial numerical properties but can still be unstable unless its parameters are tuned to the data [5].

## 5. SIMULATIONS

A 64th order noise cancellation problem was simulated. The desired signal strength was chosen to be $-60$dB to observe numerical effects. At $t = 5\times10^5$ a change of data statistics and of desired filter weights occurred. The graph of figure 2 shows the square of the difference between the filtering residual and the desired signal for both the KaGE algorithm and the Stabilised FTF (SFTF) algorithm as specified in [5]. A forget factor of $\lambda = 0.995$ was used. All functions were performed using 32 bit, single precision, floating point arithmetic.

Both algorithms converged quickly as expected to the same solution. The SFTF RLS algorithm diverged from the correct solution at $t \approx 2.9\times10^5$. (After this point the algorithm output is not shown.) This was in contrast to the KaGE RLS algorithm which ran successfully for $1\times10^6$ iterations (at which point the experiment stopped), quickly re-adapting to the change of problem at $t = 5\times10^5$.

Eventually, the KaGE algorithm would diverge when the noise on the Kalman gain vector exceeded a certain threshold. For such a problem we have not seen this within $O(10^7)$ iterations. The divergence occurs much sooner if lower precision arithmetic is used. the The KaGE algorithm implemented using 8 bit mantissa arithmetic diverged within $1\times10^5$ iterations in similar simulations.

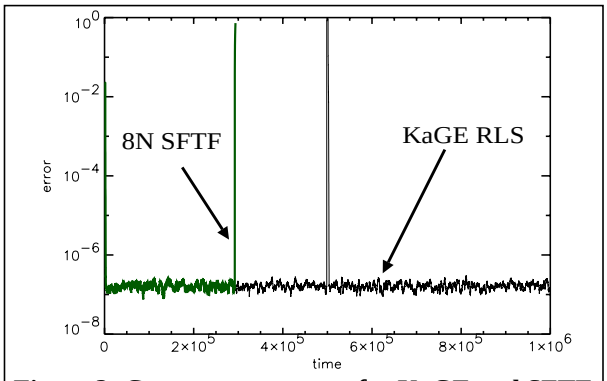


**Figure 2: Convergence curves for KaGE and SFTF.**

## 6. CONCLUSIONS

We have introduced the KaGE RLS algorithm. By making use of interpolation as well as prediction the KaGE algorithm generates the Kalman gain vector via numerically well behaved SRF QR RLS techniques using $O(N\log_2 N)$ operations. The transversal filter weights follow immediately. Assuming that a divide is on average equivalent to 5 multiplies or additions, the total operation count for the KaGE algorithm is approximately $(34\log_2 N - 16)N$. For $N = 1024$ this is approximately 7 times more than the SRF QR LSL algorithm and 20 times more than the SFTF algorithm which require approximately $46N$ and $16N$ operations per iteration respectively. However, whilst the FTF algorithms exhibit exponential error growth with time, the KaGE RLS algorithm exhibits only linear error growth, and is thus better suited to many real world applications.

## 7. REFERENCES


[1] Simon Haykin, “Adaptive filter Theory.”, 2nd Edition, Prentice-Hall, 1991.

[2] M D Levin and C F N Cowan, “Instability Trends in the Fast Kalman and FTF Algorithms” IEE Colloquium Digest, no. 1992/197, IEE, London, pp.3/1-3/5, Nov 1992.

[3] J-T Yuan, “Asymmetric Interpolation Lattice”, IEEE Trans. Sig Proc., vol 44, no.5, pp. 1256-1261, May 1996.

[4] S Haykin, J Litva, and T J Shepherd, “Radar Array Processing”, Springer-Verlag, Berlin, 1993, pp. 169-173.

[5] K Maouche and D. T. M. Slock, “Fast Subsampled-Updating Stabilized Fast Transversal Filter (FSU SFTF) RLS Algorithm for Adaptive Filtering”, IEEE Trans. Signal Proc., vol. 48, no. 8, pp. 2248-2257, Aug 2000.