A Square-Root-Free QRD-LSL Interpolation Algorithm

Jenq-Tay Yuan, *Senior Member, IEEE*, Chih-An Chiang and Chang-Hung Wu*

Department of Electrical Engineering, Fu Jen Catholic University, Taipei 24205, Taiwan, R.O.C.
E-mail: yuan@ee.fju.edu.tw, E-mail*: pillar1997@hotmail.com

ABSTRACT

This paper develops a fast order-recursive square-root-free (SRF) QR-decomposition-based least-squares lattice (QRD-LSL) interpolation algorithm that significantly improves the numerical stability of its square-root (SR) version. The proposed interpolation algorithm can be used to implement the widely known recursive least-squares (RLS) algorithm with a computational complexity of $O(N\log_2 N)$ per iteration, where N is the order of the RLS filter.

Index Terms – Interpolation, least-squares lattice (LSL), prediction, QR-decomposition (QRD), recursive least-squares (RLS), square-root-free (SRF).

1. INTRODUCTION

Linear interpolation estimates an unknown data sample based on a weighted sum of the surrounding data samples. In one-dimensional signal processing, $(p,f)^{th}-order$ linear interpolation is defined by linearly estimating the present input data samples $x(i)$ from its p past and f future neighboring data samples with the prewindowing condition on the data, viz. $\hat{x}_{p,f}(i)=-\sum_{\substack{k=-p\\k\neq 0}}^{f} b^*_{(p,f),k}(n-f)x(i+k)$, $1-f\le i\le n-f$, where $b_{(p,f),k}(n-f)$ is the interpolation coefficient at time n. The $(p,f)^{th}-order$ interpolation error at each time unit can thus be written as $e^I_{p,f}(i)=x(i)-\hat{x}_{p,f}(i)=\mathbf{b}^H_{p,f}(n-f)\mathbf{x}_{N+1}(i+f), 1-f\le i\le n-f$, where $\mathbf{b}^H_{p,f}(n-f)=[b^*_{(p,f),f}(n-f),...,b^*_{(p,f),1}(n-f),1,b^*_{(p,f),-1}(n-f),$ $...,b^*_{(p,f),-p}(n-f)]$ and the $(N+1)\times 1$ input vector is given as $\mathbf{x}_{N+1}(i)=[x(i),x(i-1),...,x(i-N)]^T$ in which $N=p+f$. In the sequel, we refer to any interpolation filter that operates on the present data sample as well as p past and f future data samples to produce the $(p,f)^{th}-order$ interpolation errors at its output as a $(p,f)^{th}-order$ *interpolator*. A *least squares lattice (LSL)* interpolator to compute the $(p,f)^{th}-order$ interpolation errors based on the *QR-decomposition (QRD)* was first introduced in [1]. However, the *square-root (SR)* QRD-LSL interpolation algorithm developed in [1] requires explicit square root computations, which constitute a computational bottleneck and are undesirable from the perspective of practical VLSI circuit design. Moreover, the SR QRD-LSL interpolation algorithm must execute the division by a cosine value $c(n)$ [see (27) and (28) in Table 1]. Owing to the division, the factor $\left|\frac{1}{c(n)}\right|\ge 1$ may amplify errors that arise in its corresponding numerator when using finite-precision arithmetic. Consequently, the SR QRD-LSL interpolation algorithm may not be numerically robust enough in a finite precision environment, restricting its application. The primary purpose of this paper is to develop a *square-root-free (SRF)* version of the QRD-LSL interpolation algorithm that avoids the division by $c(n)$; reduces computational complexity and, more importantly, significantly improves the numerical stability of the SR version.

The proposed SRF QRD-LSL interpolator outperforms its prediction counterpart in applications such as data compression, speech (image) coding and restoration, in terms of minimum mean-square error (MMSE), because the interpolator makes better use of correlation between the nearest neighboring samples than the predictor. One additional application of the SRF QRD-LSL interpolator is the implementation of the widely known *recursive least-squares (RLS)* algorithm. The majority of the computation of the RLS algorithm is the calculation of the Kalman gain vector that is required at each time step to update the filter weights [2]. Skidmore and Proudler [3] first realized the crucial fact that the Kalman gain vector can be calculated as a particular set of normalized least-squares interpolation errors. The SRF QRD-LSL interpolation algorithm can therefore be adopted to generate the set of least-squares interpolation errors in an *order-recursive* manner to calculate the Kalman gain vector. The resulting RLS algorithm requires $O(N\log_2 N)$ operations per iteration.

2. LSL INTERPOLATOR AND SRF GIVENS ROTATION

2.1. QRD-LSL Interpolator

A computationally efficient LSL interpolator developed in [1] requiring only *O(N)* operations via "intermediate predictions" can be employed to compute the exact order-updated interpolation errors, where N is the order of the interpolation lattice. The LSL interpolator computes the order-recursive interpolation errors as an additional *past* data sample and an additional *future* data sample are taken into account, respectively, as follows

$$e^I_{p+1,f}(n-f)=e^I_{p,f}(n-f)-k^*_{p+1,f}(n)e_{b,N+1}(n,n-f) \quad (1)$$

$$e^I_{p,f+1}(n-f-1)=e^I_{p,f}(n-f-1)-k^*_{p,f+1}(n)e_{f,N+1}(n,n-f-1) \quad (2)$$

where both $k_{p+1,f}(n)$ and $k_{p,f+1}(n)$ are the complex-valued

This work was supported by National Science Council (NSC), Taiwan, R.O.C. under contract NSC 96-2221-E-030-004.

coefficients; $e_{f,N+1}(i,i-f-1)=x(i)+\sum_{\substack{k=1\\k\neq f+1}}^{N+1} a^*_{N+1,k}(n)x(i-k)$, is referred to as the $(N+1)^{st}-order$ intermediate forward prediction error, as it is the prediction error of $x(i)$ based on a weighted linear combination of its (*N+1*) previous data samples without considering the present data sample *x(i-f-1)*, where $a_{N+1,k}(n), k=1,2,...,f,f+2,...,N$ are intermediate forward prediction coefficients. Similarly, the $(N+1)^{st}-order$ intermediate backward prediction error is defined as $e_{b,N+1}(i,i-f)=x(i-N-1)+\sum_{\substack{k=1\\k\neq p+1}}^{N+1} c^*_{N+1,k}(n)x(i+k-N-1)$, where $c_{N+1,k}(n), k=1,2,...,p,p+2,...,N$ are $(N+1)^{st}-order$ intermediate backward prediction coefficients.

Both the intermediate forward and backward prediction errors must be computed before the *order-updated* interpolation errors in (1) and (2) can be computed. They can be computed by using the conventional forward and backward prediction errors, as follows.

$$e_{b,N+1}(n,n-f)=e_{b,N+1}(n)+l^*_{b,N+1}(n)e^I_{p,f}(n-f) \quad (3)$$

$$e_{f,N+1}(n,n-f-1)=e_{f,N+1}(n)+l^*_{f,N+1}(n)e^I_{p,f}(n-f-1) \quad (4)$$

where $l_{b,N+1}(n)$ and $l_{f,N+1}(n)$ are complex-valued coefficients; $e_{b,N+1}(n)$ and $e_{f,N+1}(n)$, which are conventional backward and forward prediction errors, respectively, are directly accessible from an $(N+1)^{st}-order$ *LSL predictor* [2] that can be embedded into an LSL interpolator. Notably, both $e^I_{p,f}(n-f)$ and $e^I_{p,f}(n-f-1)$ are already computed from the previous interpolation lattice stage of the LSL interpolator. Equations (1)–(4), together with the well-known LSL predictor, constitute an *order-recursive LSL interpolator*. Due to space limitations, only the implementation of (2) and (4) as $(p,f)\to(p,f+1)$ using the SR version of the QRD-LSL interpolation algorithm developed in [1] is summarized in the Int(F) block and IFP block, respectively, (see the first column of Table 1).

2.2. SRF Givens Rotation with Feedback Mechanism

Consider first a Givens rotation matrix given by $\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix}$, where the two parameters of the Givens rotation are the real cosine parameter *c* and the complex sine parameter *s* such that $c^2+|s|^2=1$. A Givens rotation used to zero out the element at the (2,1) location is an elementary transformation of the form

$$\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix}\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_p \\ \beta_1 & \beta_2 & \cdots & \beta_p \end{bmatrix}=\begin{bmatrix} \alpha_1' & \alpha_2' & \cdots & \alpha_p' \\ 0 & \beta_2' & \cdots & \beta_p' \end{bmatrix} \quad (5)$$

where α_1 and α_1' are defined to be real and nonnegative, whereas $\alpha_2\cdots\alpha_p,\ \alpha_2'\cdots\alpha_p',\ \beta_1\cdots\beta_p,\ \beta_2'\cdots\beta_p'$ are all complex. Substituting $s=\frac{c\beta_1}{\alpha_1}$ into $c^2+|s|^2=1$ yields $c=\frac{\alpha_1}{\alpha_1'}$, where $\alpha_1'\triangleq\sqrt{\alpha_1^2+|\beta_1|^2}$. The SR Givens rotation of complex version, which requires a square root operation in the generic formulation, is summarized in the first column of Table 2. The SR operation may occupy a large area in a VLSI chip and may require many cycles to complete such computations; consequently, the operation is slow. Proudler *et al* [4] demonstrated that a finite-precision implementation of an SRF lattice algorithm achieved better numerical results than were obtained using the conventional Givens rotation. Hsieh, Liu and Yao [5] thus proposed a systematic way of generating a unified SRF Givens rotation to avoid the square root operation. In this section, we generalize the SRF Givens rotation developed in [5] to a complex form and extend it to include a feedback mechanism, which is known to have a stabilizing effect when errors are made in the QRD-based recursive least squares estimation, because of finite-precision effects [6], [7]. We then apply the generalized SRF Givens rotation with feedback to develop the SRF QRD-LSL interpolation algorithm in Section 3.

By taking out a scaling factor from each row of the matrices on both sides of (5), the two rows before and after the Givens rotation is denoted, respectively, by $\begin{bmatrix} \alpha_1\ \alpha_2\ ...\ \alpha_p \\ \beta_1\ \beta_2\ ...\ \beta_p \end{bmatrix}=\begin{bmatrix} \sqrt{k_a} & 0 \\ 0 & \sqrt{k_b} \end{bmatrix}\cdot\begin{bmatrix} a_1\ a_2\ ...a_p \\ b_1\ b_2\ ...\ b_p \end{bmatrix}$ and $\begin{bmatrix} \alpha_1'\ \alpha_2'\ ...\ \alpha_p' \\ 0\ \beta_2'\ ...\ \beta_p' \end{bmatrix}=\begin{bmatrix} \sqrt{k_a'} & 0 \\ 0 & \sqrt{k_b'} \end{bmatrix}\begin{bmatrix} a_1'\ a_2'\ ...a_p' \\ 0\ b_2'\ ...\ b_p' \end{bmatrix}$,

where k_a, k_b, k_a', and k_b' are the scaling factors resulting in SRF operations, and α_i' and β_i' are the updated α_i and β_i when β_1 is zeroed out. $k_a', k_b', a_1', \{(a_i', b_i'), i=2,...,p\}$ are determined [5] such that the SRF Givens rotation may be updated by using (19)–(21) [in the second column of Table 2] and

$$a_i'=\bar{c}a_i+\bar{s}^*b_i, i=2,...,p \quad (6)$$

$$b_i'=b_i-b_1a_i, i=2,...,p \quad (7)$$

without square root operation. The above results would be consistent with those of Gentleman [8] and McWhirter [9]. The SRF Givens rotation with feedback mechanism can be derived by substituting (7) into (6), which yields

$$a_i'=a_i\left(\bar{c}+\bar{s}^*b_1\right)+\bar{s}^*b_i'=a_i+\bar{s}^*b_i' \quad (8)$$

where we have used $\bar{c}+\bar{s}^*b_1=\frac{k_a}{k_a'}+\frac{k_bb_1^*}{k_a'}b_1=1$. For notational convenience, taking the complex conjugate on both sides of (7) and (8) yields (22) and (23). Equations (19)–(23) summarized in the second column of Table 2 are the complex version of the *SRF Givens rotation with a feedback mechanism*.

3. SRF QRD-LSL INTERPOLATOR FOR RLS SOLUTION

3.1. SRF QRD-LSL Interpolation Algorithm

The SRF QRD-LSL interpolation algorithm consists of six blocks: (a) forward prediction (FP) block and backward prediction (BP) block to compute both conventional forward and backward prediction errors; (b) intermediate forward prediction (IFP) block and interpolation [Int(F)] block as an additional "future" stage is increased [i.e., $(p,f)\to(p,f+1)$] to compute the interpolation error, $\bar{\varepsilon}^I_{p,f+1}(n-f-1)$; (c) intermediate backward prediction (IBP) block and interpolation [Int(P)] block as an additional "past" stage is increased [i.e., $(p,f) \rightarrow (p+1,f)$] to compute the interpolation error, $\overline{\varepsilon}^{I}_{p+1,f}(n-f)$. Only IFP and Int(F) blocks are summarized in the second column of Table 1. Throughout this work, the terms "e" and "$\overline{\varepsilon}$" represent the "a posteriori" and "a priori" estimation errors, respectively, whereas the term "ε" represents the "SR version of the angle-normalized" estimation error. Notably, the SRF QRD-LSL interpolation algorithm requires only 17 multiplication operations without any square root operation, while the SR QRD-LSL interpolation algorithm requires 20 multiplication operations and five square root operations per iteration. Both algorithms require six additions with an $O(N)$ computational complexity per iteration. Using exact arithmetic, both algorithms yield exactly the same result. However, computer simulations indicated that the SRF QRD-LSL interpolation algorithm is much more numerically robust than the SR QRD-LSL interpolation algorithm when finite arithmetic is used.

The SRF QRD-LSL interpolation algorithm can be rigorously derived by applying a sequence of SRF Givens rotations with a feedback mechanism. However, the derivation is lengthy and, therefore, is omitted. A less rigorous approach to verifying the derived SRF QRD-LSL interpolation algorithm, presented in the second column of Table 1, can be achieved by direct transformation from the original SR QRD-LSL interpolation algorithm in [1] by using the relationship between the SR and SRF Givens rotations described in the third column of Table 2. This paper demonstrates only how the Int(F) block of the SRF interpolation algorithm can be verified by directly transforming from the Int(F) block of the SR interpolation algorithm in [1], where both Int(F) blocks are presented in Table 1. Other blocks of the SRF interpolation algorithm can be similarly confirmed.

First, (29), (14) and (19) along with their relationship described in the third column of Table 2, yield $k_a' = \left(\alpha_1'\right)^2 = F_{N+1}(n,n-f-1)$, $k_a = \alpha_1^2 = \lambda F_{N+1}(n-1,n-f-2)$ and $\beta_1 = \varepsilon^{*}_{f,N+1}(n,n-f-1) \leftrightarrow b_1 \triangleq \overline{\varepsilon}^{*}_{f,N+1}(n,n-f-1)$. Second, the relationship among (34), (16) and (21) reveals $\gamma = k_{p,f}(n-f-1) \leftrightarrow k_b \triangleq k_{p,f}(n-f-1)$ and $\gamma' = k_{p,f+1}(n-f-1) \leftrightarrow k_b' \triangleq k_{p,f+1}(n-f-1)$. Substituting the above results into (20), (19) and (21), respectively, yield (36), (37), (35) and (40). Notably, the order-updated formula in (40), corresponding to the SR interpolation algorithm, is (34), which corresponds to $\gamma' = c \cdot \gamma$ in (16) of Table 2. Third, (17), (22) and (32), along with their relationship described in the third column of Table 2, reveals

$$\beta_i' = \varepsilon^{I^*}_{p,f+1}(n-f-1) \leftrightarrow b_i' \triangleq \overline{\varepsilon}^{I^*}_{p,f+1}(n-f-1) \qquad (9)$$

$$\alpha_i = \lambda^{1/2}\rho_{p,f+1}(n-1) \leftrightarrow a_i \triangleq \overline{\rho}_{p,f+1}(n-1) \qquad (10)$$

and $\beta_i = \varepsilon^{I^*}_{p,f}(n-f-1) \leftrightarrow b_i = \overline{\varepsilon}^{I^*}_{p,f}(n-f-1)$. Substituting the above results into (22) yields (38). Finally, (18), (23) and (33) along with their relationship described in the third column of Table 2 reveal $\alpha_i' = \rho_{p,f+1}(n) \leftrightarrow a_i' = \overline{\rho}_{p,f+1}(n)$. Substituting this result and (9) and (10) into (23) yields (39). Equations (35)–(40) summarized in the second column of Table 1, constitute the Int(F) block of the SRF QRD-LSL interpolation algorithm.

3.2. RLS Solution Based on SRF QRD-LSL Interpolator

The SRF QRD-LSL interpolator can be applied to implement the RLS algorithm in a transversal structure. The $(N+1)^{st}-order$ RLS algorithm with the tap-weight vector at time *n*, $\mathbf{w}_{N+1}(n)$, can be calculated recursively in time using [2] $\mathbf{w}_{N+1}(n) = \mathbf{w}_{N+1}(n-1) + \mathbf{k}_{N+1}(n)\overline{\varepsilon}^{*}(n)$, where $\mathbf{k}_{N+1}(n)$ is the Kalman gain vector. The majority of the computation of the RLS algorithm is the calculation of $\mathbf{k}_{N+1}(n)$, which can also be calculated as a particular set of normalized least-squares interpolation errors $\mathbf{k}^{T}_{N+1}(n) = [\frac{e^{I}_{N,0}(n)}{I_{N,0}(n)},\ldots,\frac{e^{I}_{p+1,f-1}(n-f+1)}{I_{p+1,f-1}(n-f+1)}, \frac{e^{I}_{p,f}(n-f)}{I_{p,f}(n-f)},\frac{e^{I}_{p-1,f+1}(n-f-1)}{I_{p-1,f+1}(n-f-1)},\ldots,\frac{e^{I}_{0,N}(n-N)}{I_{0,N}(n-N)}]$ [3] in which $I_{p,f}(n-f)$ is the sum of interpolation error squares of order *(p, f)* [see (41) in Table 1]. Notably, the posteriori interpolation errors in the Kalman gain vector can be computed by using $e^{I}_{p,f+1}(n-f-1) = k_{p,f+1}(n-f-1)\overline{\varepsilon}^{I}_{p,f+1}(n-f-1)$. [See (38) and (40) in Table 1.] Also note that the elements of $\mathbf{k}_{N+1}(n)$ are responsible for the fast convergence of the RLS algorithm, since that the Kalman gain vector is the most decorrelated version of the normalized input vector, because each element of $\mathbf{k}_{N+1}(n)$ is the optimum two-sided prediction residual of each corresponding element of the input vector $\mathbf{x}_{N+1}(n)$. Accordingly, the Kalman gain vector corresponds to the least redundant version of the input vector. The SRF QRD-LSL interpolation algorithm can be used to efficiently calculate $\mathbf{k}_{N+1}(n)$ in an *order-recursive* manner using a divide-and-conquer approach; the resulting RLS algorithm requires $O(N\log_2 N)$ operations for a transversal filter of order *N*.

4. SIMULATIONS

All simulations are run on a PC with a floating-point processor. To observe the finite-precision effects, the effective number of mantissa bits in the floating-point representation is reduced by truncating the mantissa at a predefined position without affecting the exponent according to pp.611 of [2]. A polar form pseudo-random signal *d*(*n*) is applied to a channel having unit pulse response [10] $h_n = \begin{cases} \frac{1}{2}\left[1+\cos\left(\frac{2\pi}{W}(n-2)\right)\right], n=1,2,3 \\ 0\text{ , }otherwise \end{cases}$, where *W* was set equal to 3.5 to yield an eigenvalue ratio of 46.82. The observation *x*(*n*) is the sum of the channel output and an independent white Gaussian noise with variance 0.001. The adaptive equalizer attempts to correct the distortion produced by the channel and the additive noise. In all simulations, 11-tap equalizers were used. Figure 1 compares the learning curves (obtained by ensemble-averaging the squared-error of the equalizer output over 200 independent trials of the experiment) using a conventional RLS algorithm requiring $O(N^2)$ operations per iteration and the RLS algorithm based on both the SR QRD-LSL interpolation algorithm and the SRF QRD-LSL interpolation algorithm, all with a forgetting factor of $\lambda = 0.996$. Our simulations revealed that using exact arithmetic, all three algorithms yielded exactly the same result, but they exhibited different unstable behaviors in finite precision computation. While the conventional RLS algorithm became unacceptable for less than 12 mantissa bits, the SRF (QRD-LSL)-interpolation-based RLS algorithm using 10 mantissa bits ran successfully for two million iterations (at which point the experiment terminated) in a limited-precision environment, making it potentially suitable for real-world applications. By contrast, the SR (QRD-LSL)-interpolation-based RLS algorithm with a mantissa length of 20 bits started diverging around $n = 1.4 \times 10^6$. This result confirms that the SRF QRD-LSL interpolation algorithm is much more numerically robust than the SR QRD-LSL interpolation algorithm when finite-precision arithmetic is used.

5. REFERENCES

[1] J.-T. Yuan, "QR-decomposition-based least-squares lattice interpolators," *IEEE Trans. Signal Processing*, vol. 48, pp. 70-79, Jan. 2000.

[2] S. Haykin, *Adaptive Filter Theory*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[3] I. D. Skidmore and I. K. Proudler, "The KaGE RLS algorithm," *IEEE Trans. Signal Processing*, vol. 51, pp. 3094-3104, Dec. 2003.

[4] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, "The QRD-based least squares lattice algorithm: Some computer simulations using finite wordlengths," *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 258-261, May 1990.

[5] S. F. Hsieh, K. J. R. Liu, and K. Yao, "A unified square-root-free approach for QRD-based recursive least squares estimation," *IEEE Trans. Signal Processing*, vol. 41, pp. 1405-1409, March 1993.

[6] F. Ling, D. Manloakis, and J. G. Proakis, "Numerically robust least-squares lattice-ladder algorithms with direct updating of the reflection coefficients," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 4, pp. 837-845, August 1986.

[7] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, "Computationally efficient QR-decomposition approach to least squares adaptive filtering," *IEE Proceedings-F*, vol. 138, no. 4, August 1991.

[8] W. M. Gentleman, "Least-squares computations by Givens transformations without square roots," *J. Inst. Math. Appl.,* vol. 12. pp. 329-336, 1973.

[9] J. G. McWhirter, "Recursive least squares minimization using a systolic array," *Proc. SPIE Int. Soc. Opt. Eng.,* vol. 431, pp. 105-112, 1983.

[10] E. H. Satorius and S. T. Alexander, "Channel equalization using adaptive lattice algorithms," *IEEE Trans. Commun.*, vol. COMM-27, pp. 899-905, June 1979..

Table 1. Summary of both SR and SRF QRD-LSL Interpolation Algorithms as $(p, f) \rightarrow (p, f+1)$

SR QRD-LSL Interpolation Algorithm		SRF QRD-LSL Interpolation Algorithm	
IFP block:		IFP block:	
$I_{p,f}(n-f-1) = \lambda I_{p,f}(n-f-2) + \left\|\varepsilon^I_{p,f}(n-f-1)\right\|^2$	(24)	$I_{p,f}(n-f-1) = \lambda I_{p,f}(n-f-2) + k_{p,f}(n-f-1)\left\|\bar{\varepsilon}^I_{p,f}(n-f-1)\right\|^2$	(41)
$c_{I,N}(n-1) = \dfrac{\lambda^{1/2} I^{1/2}_{p,f}(n-f-2)}{I^{1/2}_{p,f}(n-f-1)}$	(25)	(not needed)	
$s_{I,N}(n-1) = \dfrac{\varepsilon^{I*}_{p,f}(n-f-1)}{I^{1/2}_{p,f}(n-f-1)}$	(26)	$\bar{s}_{I,N}(n-1) = k_{p,f}(n-f-1)\dfrac{\bar{\varepsilon}^{I*}_{p,f}(n-f-1)}{I_{p,f}(n-f-1)}$	
$\varepsilon_{f,N+1}(n,n-f-1) = \dfrac{\varepsilon_{f,N+1}(n) + \lambda^{1/2} s^*_{I,N}(n-1)\Delta^*_{f,N+1}(n-1)}{c_{I,N}(n-1)}$	(27)	$\bar{\varepsilon}_{f,N+1}(n,n-f-1) = \bar{\varepsilon}_{f,N+1}(n) + \bar{\varepsilon}^I_{p,f}(n-f-1)\bar{\Delta}^*_{f,N+1}(n-1)$	
$\Delta^*_{f,N+1}(n) = \dfrac{\lambda^{1/2}\Delta^*_{f,N+1}(n-1) + s_{I,N}(n-1)\varepsilon_{f,N+1}(n)}{c_{I,N}(n-1)}$	(28)	$\bar{\Delta}^*_{f,N+1}(n) = \bar{\Delta}^*_{f,N+1}(n-1) + \bar{s}_{I,N}(n-1)\bar{\varepsilon}_{f,N+1}(n)$	
Int(F) block:		Int(F) block:	
$F_{N+1}(n,n-f-1) = \lambda F_{N+1}(n-1,n-f-2) + \left\|\varepsilon_{f,N+1}(n,n-f-1)\right\|^2$	(29)	$F_{N+1}(n,n-f-1) = \lambda F_{N+1}(n-1,n-f-2) + k_{p,f}(n-f-1)\left\|\bar{\varepsilon}_{f,N+1}(n,n-f-1)\right\|^2$	(35)
$c'_{f,N+1}(n) = \dfrac{\lambda^{1/2} F^{1/2}_{N+1}(n-1,n-f-2)}{F^{1/2}_{N+1}(n,n-f-1)}$	(30)	$\bar{c}'_{f,N+1}(n) = \dfrac{\lambda F_{N+1}(n-1,n-f-2)}{F_{N+1}(n,n-f-1)}$	(36)
$s'_{f,N+1}(n) = \dfrac{\varepsilon^*_{f,N+1}(n,n-f-1)}{F^{1/2}_{N+1}(n,n-f-1)}$	(31)	$\bar{s}'_{f,N+1}(n) = k_{p,f}(n-f-1)\dfrac{\bar{\varepsilon}^*_{f,N+1}(n,n-f-1)}{F_{N+1}(n,n-f-1)}$	(37)
$\varepsilon^I_{p,f+1}(n-f-1) = c'_{f,N+1}(n)\varepsilon^I_{p,f}(n-f-1) - \lambda^{1/2} s'^*_{f,N+1}(n)\rho^*_{p,f+1}(n-1)$	(32)	$\bar{\varepsilon}^I_{p,f+1}(n-f-1) = \bar{\varepsilon}^I_{p,f}(n-f-1) - \bar{\varepsilon}_{f,N+1}(n,n-f-1)\bar{\rho}^*_{p,f+1}(n-1)$	(38)
$\rho^*_{p,f+1}(n) = \lambda^{1/2} c'_{f,N+1}(n)\rho^*_{p,f+1}(n-1) + s'_{f,N+1}(n)\varepsilon^I_{p,f}(n-f-1)$	(33)	$\bar{\rho}^*_{p,f+1}(n) = \bar{\rho}^*_{p,f+1}(n-1) + \bar{s}'_{f,N+1}(n)\bar{\varepsilon}^I_{p,f+1}(n-f-1)$	(39)
$k_{p,f+1}(n-f-1) = c'_{f,N+1}(n)k_{p,f}(n-f-1)$	(34)	$k_{p,f+1}(n-f-1) = \bar{c}'_{f,N+1}(n)k_{p,f}(n-f-1)$	(40)

Table 2. Summary of SR and SRF Givens rotations and their relationship

SR Givens rotation		SRF Givens rotation with feedback		Relations
$(\alpha'_1)^2 = \alpha_1^2 + \left\|\beta_1\right\|^2$	(14)	$k'_a = k_a + k_b\left\|b_1\right\|^2$	(19)	$(\alpha'_1)^2 = k'_a$, $\alpha_1^2 = k_a$,
$c = \dfrac{\alpha_1}{\alpha'_1}$ and $s = \dfrac{\beta_1}{\alpha'_1}$	(15)	$\bar{c} = \dfrac{k_a}{k'_a}$ and $\bar{s} = b_1 \cdot \dfrac{k_b}{k'_a}$	(20)	$c = \sqrt{\bar{c}}$, $s = \sqrt{b_1 \bar{s}}$, $\alpha_i \leftrightarrow a_i$, $\beta_i \leftrightarrow b_i$,
$\gamma' = c \cdot \gamma$	(16)	$k'_b = \bar{c} \cdot k_b$	(21)	$\alpha'_i \leftrightarrow a'_i$, $i = 1, 2, .., p$,
$(\beta'_i)^* = c(\beta_i)^* - s^*\alpha^*_i, i = 2, ..., p$	(17)	$(b'_i)^* = (b_i)^* - (b_1)^*(a_i)^*, i = 2, ..., p$	(22)	$\beta'_i \leftrightarrow b'_i$, $i = 2, .., p$,
$(\alpha'_i)^* = c\alpha^*_i + s\beta^*_i, i = 2, ..., p$	(18)	$(a'_i)^* = a^*_i + \bar{s} \cdot (b'_i)^*, i = 2, ..., p$	(23)	$\gamma' \leftrightarrow k'_b$, $\gamma \leftrightarrow k_b$

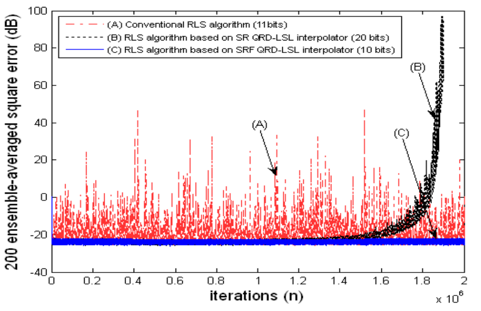

Fig. 1. Learning curves of three RLS algorithms in a finite precision environment.