IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, May 9-10, 1991

Dynamically Scaled Fixed Point Arithmetic

Darrell Williamson
Interdisciplinary Engineering Program
Australian national University
The Faculties, GPO box 4, Canberra, ACT 2601

1 Introduction

Digital filters and their implementation with finite precision arithmetic have been analysed for many years. The most common simple implementation use either fixed-point or floating point arithmetic, but the logarithmic number system [1] and the residue number system [2] have also been considered.

Fixed point arithmetic can provide a fast cheap and simple implementation. The accuracy of the implementation depends upon the input quantization error, the coefficient quantization error and the arithmetic roundoff error. Moreover, internal arithmetic quantizations can result in limit cycles due to the effects of underflow, or overflow oscillations due to the effect of overflow. The most severe effects occur as a result of overflow and therefore a great deal of consideration is spent on scaling a fixed point implementation to avoid overflow. If the scaling policy is too conservative, then accuracy will be lost but a too optimistic scaling policy will result in signal distortion from overflow.

Floating point arithmetic has the advantage of having a larger dynamic range but suffers from speed limitations. Also, if one compares the accuracy of a fixed point implementation and a floating point implementation whose mantissa has the same fixed point wordlength, then greater accuracy can be achieved with fixed point. The reason is because during the evaluation of an inner product $w = \Sigma_i x_i y_i$, a floating point quantization error occurs after every multipllication and addition, whereas with fixed point arithmetic, w can be evaluated exactly. Specifically, if x_i and y_i are t_1 but fixed point numbers, then an inner product w of n_1 terms can be exactly computed using an accumulator of $2t_1 + r_1$ bits where r_1 is the smallest integer which is not less than $\log_2(n_1)$.

Limit cycles can occur in fixed point implementations, but a floating point implementation is also not excluded from this phenomenon [3]. Clearly, one would be much more confident if better scaling procedures would be designed for a fixed point implementation.

The aim of this paper is to present a technique for dynamically scaling a fixed point filter at each stage of the iteration such that overflow is no more likely as when floating point arithmetic is used. The results are a generalization of earlier work [4,5].

2 Dynamic Scaling

Consider a digital filter $H(z)$ having a minimal state space representation $\{A, B, C\}$. That is

$$H(z) = C(zI_n - A)^{-1}B \tag{2.1}$$

where I_n denotes the $n \times n$ identity matrix is represented in state space form by

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \;\; ; \;\; x(k)\varepsilon R^n \\ y(k) &= Cx(k) \end{aligned} \tag{2.2}$$

Define the $n \times n$ *scaling matrix* $S(k)$, and the $n \times n$ *incremental scaling matrix* $\Lambda(k)$, and the new state $z(k)$ by

$$z(k) \triangleq S(k)x(k) \;\; ; \;\; S(k) \triangleq \Lambda(k)S(k-1) \tag{2.3}$$

Then (2.2) can be written in the form

$$\begin{aligned} p(k) &= S(k)AS^{-1}(k)z(k) + S(k)Bu(k) \\ z(k+1) &= \Lambda(k+1)p(k) \\ y(k) &= CS^{-1}(k)z(k) \end{aligned} \tag{2.4}$$

Now suppose

$$S(0) = I_n \tag{2.5a}$$

then

$$\Lambda(k) = \mathrm{diag}\{\lambda_1(k), \lambda_2(k), \cdots, \lambda_n(k)\} \tag{2.5b}$$

implies

$$S(k) = \mathrm{diag}\{s_1(k), s_2(k), \cdots, s_n(k)\} \tag{2.5c}$$

Given the components $p_j(k)$ of $p(k)$ in (2.4), define the integers $\beta_j(k+1)$ such that for all j:

$$2^{\beta j(k+1)-1} \leq |p_j(k)| < 2^{\beta j(k+1)} \tag{2.6a}$$

Then if $\lambda_j(k+1)$ in (2.5b) is defined by

$$\lambda_j(k+1) \triangleq 2^{-\beta j(k+1)} \tag{2.6b}$$

it follows from (2.4) that all components $z_j(k+1)$ of $z(k+1)$ are bounded according to

$$0.5 \leq |z_j(k+1)| < 1 \tag{2.7}$$

Now, let the components $s_j(k)$ of $S(k)$ in (2.5c) be defined for some integer $\gamma_j(k)$ by

$$s_j(k) \triangleq 2^{-\gamma_j(k)} \tag{2.8a}$$

Then from (2.3), (2.5) and (2.6b), the integer $\gamma_j(k)$ and $\beta_j(k)$ are related by

$$\gamma_j(k+1) = \beta_j(k+1)\gamma_j(k) \tag{2.8b}$$

At each step of the iteration in (2.4), the n integer values $z_j(k)$ and the n integers $\gamma_j(k)$ must be stored.

CH2954-6/91-0000-315 $1.00

3 Fixed Point Implementation

A fixed point finite wordlength implementation of (2.4) - (2.8) where both input and coefficient quantization is ignored is described by

$$\begin{aligned} \hat{p}(k) &= S(k)AS^{-1}(k)Q[\hat{z}(k)] + S(k)Bu(k) \\ \hat{z}(k+1) &= \Lambda(k+1)\hat{p}(k) \\ \hat{y}(k) &= CS^{-1}(k)Q[\hat{z}(k)] \\ Q[\hat{z}(k)] &= \hat{z}(k) - \varepsilon(k) \end{aligned} \tag{3.1}$$

where $S(k)$ and $\Lambda(k)$ given by (2.5) - (2.8). Observe that from (2.5c) and (2.8a) the i-th component $\hat{p}_i(k)$ of $\hat{p}(k)$ is given by

$$\hat{p}_i(k) = 2^{-\gamma_i(k)}\{\sum_{j=1}^{n} 2^{\gamma_j(k)} a_{ij} Q[\hat{z}_j(k)] + b_i' u(k)\} \tag{3.2}$$

where b_i' is the i-th row of B. Based on the definition of $\gamma_j(k)$ in section 2

$$0.5 \le |Q[\hat{z}_j(k)]| < 1 \tag{3.3}$$

Therefore the most significant fractional bit of $Q[\hat{z}_j(k)]$ is always 1 and therefore need not be stored. That is, with 1 bit for the sign and t_1 bits for the fractional part of $Q[\hat{z}_j(k)]$, the quantization error is bounded by

$$|\varepsilon_j(k)| < 2^{-(t_1+2)} \tag{3.4}$$

Suppose the components $u_j(k)$ of $u(k)$ are also fixed point fractions, coefficient a_{ij} and each component b_{mi} of b_i' are fixed point fractions of t_0 bits (not counting the sign bit) and

$$|\gamma_j(k)| \le \gamma \tag{3.5}$$

Then the inner product in (3.2) can be computed *exactly* using fixed point multiplications and additions and shifting operations if the accumulator has

$$t_o + t_1 + 2\gamma + n_{\text{add}}$$

bits where n_{add} is the smallest integer not less than $\log_2(n+1)$.

The computation of $\beta_j(k+1)$ as given by

$$2^{\beta_j(k+1)-1} \le |\hat{p}_j(k)| < 2^{\beta_j(k+1)} \tag{3.6}$$

is similar to the alignment associated with the normalization of the mantissa in floating point arithmetic. Once the $\beta_j(k+1)$ are determined, the new values for $z_j(k+1)$ are determined from (2.4), and the new values of $\gamma_j(k+1)$ from (2.8b).

In summary, the state of (3.1) is represented for each j by a signed t_1 bit normalized fixed point 'mantissa' $Q[\hat{z}_j(k)]$, and a signed γ bit fixed point exponent $\gamma_j(k)$. Only fixed point operations are involved in computing the inner product in (3.2), and normalization only occurs after the entire inner product is computed. This is quite different than in the case of a floating point implementation of an inner product where normalization and quantization (and hence error) occurs after *every* multiplication and addition.

The time needed to compute $\hat{p}_i(k)$ in (3.2) is similar to that required in the unscaled fixed point case (i.e. when $\gamma_m(k) = 0$ for all m). The only difference being a shift of $\gamma_j(k)$ bits after computing each product $a_{ij}Q[\hat{z}_j(k)]$. The main overhead in time when compared to a fixed point implementation is in the computartion of $\beta_j(k+1)$ in (3.5) given $\hat{p}_j(k)$. This may be done either in software or special purpose hardware designed in association with the fixed point multiplier/accummulator.

The implementation in (3.1) may be further improved by the addition of integer residue feedback [6] by modifying the equation for $\hat{p}(k)$ and $\hat{y}(k)$ according to

$$\begin{aligned} \hat{p}(k) &= \{S(k)AS^{-1}(k)Q[\hat{z}(k)] + \\ &\qquad S(k)Bu(k)\} + \psi(k)\varepsilon(k) \\ \hat{z}(k+1) &= \Lambda(k+1)\hat{p}(k) \\ \hat{y}(k) &= \{CS^{-1}(k)Q[\hat{z}(k)]\} + \theta(k)\varepsilon(k) \\ Q[\hat{z}(k)] &= \hat{z}(k) - \varepsilon(k) \end{aligned} \tag{3.7}$$

where $\psi(k)$ and $\theta(k)$ are matrices of *integers*. The choice for $\psi(k)$ and $\theta(k)$ means that (3.6) is *wordlength consistent* in that $\{\cdot\}$ and $\psi(k)\varepsilon(k)$ in the expression for $\hat{p}(k)$, and $\{\cdot\}$ and $\theta(k)\varepsilon(k)$ in the expression for $\hat{y}(k)$ always have the *same* frational wordlength representation.

Note that the practical limitation γ in (3.4) of the incremental scaling means that the *rate of increase* of the scaling matrix $S(k)$ in (2.3) is limited. Furthermore, the practical limitation

$$|\beta_j(k)| \le \beta \tag{3.8}$$

means that the scaling matrix itself is also limited. Thus as with floating point arithmetic, the practical limitation on the 'exponent' wordlength γ means that overflow is still possible but certainly very unlikely when compared to a constant scaled fixed point implementation.

4 Error Analysis

Define the error terms $\tilde{E}(k)$ and $R(k)$ by

$$\begin{aligned} \tilde{E}(k) &\triangleq S^{-1}(k)z(k) - S^{-1}(k)\hat{z}(k) \\ R(k) &\triangleq y(k) - \hat{y}(k) \end{aligned} \tag{4.1}$$

Then from (2.4) and (3.6) it follows that

$$\begin{aligned} \tilde{E}(k+1) &= A\tilde{E}(k) + (A - \psi(k))\tilde{\varepsilon}(k) \\ R(k) &= C\tilde{E}(k) + (C - \theta(k)S(k))\tilde{\varepsilon}(k) \end{aligned} \tag{4.2a}$$

where

$$\tilde{\varepsilon}(k) \triangleq S^{-1}(k)\varepsilon(k) \tag{4.2b}$$

The objective of using the integer residue feedback terms $\psi(k)$ and $\theta(k)$ is to control the output error sequence $R(k)$. Since $\psi(k)$ and $\theta(k)$ are matrices of integers, a useful *sub-optimal* solution is to select

$$\begin{aligned} \psi(k) &= \psi \triangleq \mathrm{INT}(A) \\ \theta(k) &= \mathrm{INT}(CS^{-1}(k)) \end{aligned} \qquad (4.3)$$

where $\mathrm{INT}(M)$ denotes the matrix whose components are integers which are equal to the nearest integer to the components of M. This choice in (4.3) guarantees that the terms driving $\tilde{E}(k)$ and $R(k)$ in (4.2) are kept small. Even in the fixed point case, it is known [6] that the choice in (4.3) is *not* optimal. However the choice of $\psi(k) = \psi$ as a *constant* matrix is certainly advantageous.

Note that silly conclusions can result if the wordlength consistency constraint in (3.6) is ignored. For example, just looking at (4.2a), the optimal choice is $\psi(k) = A$ in which case $\tilde{E}(0) = 0$ implies $\tilde{E}(k) = 0$ for all $k \geq 0$. However this choice for $\psi(k)$ in (3.6) will mean that $\{\cdot\}$ and $\psi(k)\varepsilon(k)$ in the expression for $\hat{p}(k)$ will *not* always have the same fractional wordlength unless all components of A are integers.

Notice that for a constant scaled fixed point implementation, $S(k)$ in (4.2b) is constant so that the bound $|\tilde{\varepsilon}_j(k)|$ on the components of $\tilde{\varepsilon}(k)$ is fixed. In particular, if $S(k) = I_n$, then

$$|\tilde{\varepsilon}_j(k)| \leq 2^{-(t_1+1)}$$

More generally, from (2.8a) and (4.2b)

$$|\tilde{\varepsilon}_j(k)| \leq 2^{-(t_1+2-\beta_j(k))}$$

Now the choice of $S(k)$ guarantees the bound in (3.3). Hence if the magnitude of any component $x_j(k)$ in the ideal representation (2.2) exceeds unity, then $s_j(k)$ in (2.5c) also exceeds unity, while if $|x_j(k)|$ becomes less than 0.5 then $s_j(k)$ becomes less than unity. If under 'expected' inputs $u(k)$, the components $x_j(k)$ of $x(k)$ in (2.2) are well scaled so that

$$0.5 \leq |x_j(k)| < 1$$

then any unforeseen circumstance which would cause $|x_j(k)|$ to exceed 1 means $|s_j(k)| > 1$ or $|s_j^{-1}(k)| < 1$. Thus in an average sense, one would expect all components $|s_j^{-1}(k)|$ to exceed unity which would imply that each component $|\tilde{\varepsilon}_j(k)|$ in (4.2b) would be *less* than $|\varepsilon_j(k)|$.

Example

The block floating point algorithm can be illustrated for the second order difference equation

$$y(k) + ay(k-1) + by(k-2) = v(k-1)$$

which has a state space realization given by (2.2) with

$$A = \begin{bmatrix} 0 & 1 \\ -b & -a \end{bmatrix} \quad ; \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad ; \quad C = [0 \quad 1]$$

With

$$s(k) \triangleq s_1(k) = s_2(k) \quad ; \quad \lambda(k) \triangleq \lambda_1(k) \triangleq \lambda_1(k) = \lambda_2(k)$$

in (2.5), a FWL implementation (3.7) is then described by

$$\begin{aligned} p(k) &= aQ[\hat{z}(k)] + Q[\lambda(k)bQ[\hat{z}(k-1)]] \\ &\quad + s(k)\hat{v}(k) + \psi_a\epsilon(k) + Q[\lambda(k)\psi_b\epsilon(k-1)] \\ \hat{z}(k+1) &= \lambda(k)p(k-1) \\ \hat{y}(k) &= s^{-1}(k)Q[\hat{z}(k)] \\ s(k) &= \lambda(k)s(k-1) \end{aligned}$$

where

$$\epsilon(k) = \hat{z}(k) - Q[\hat{z}(k)]$$

and

$$\psi_a = \mathrm{INT}(a) \quad ; \quad \psi_b = \mathrm{INT}(b)$$

are integers which provide state error feedback.

Specifically, consider the second order all pole system described by $a = -1.8$ and $b = 0.95$. The ideal response for a step input $v(k) = 0.125$ for $k \geq 0$ is compared in Figure 1a with fixed point arithmetic incorporating saturation arithmetic, and with block floating arithmetic. Note that the block floating implementation tracks the ideal solution while fixed point arithmetic introduced severe signal distortion. The variations of the instantaneous scale factor $s(k)$ for the block floating point implementation are illustrated in Figure 1b.

Noise Model

(i) The noise sources $\{\tilde{\epsilon}_m(k)\}$ in (4.2b) are *uncorrelated* for all m and k.

(ii)

$$\xi\{\tilde{\epsilon}_m^2(k)\} = \xi\{\epsilon_m^2(k)\}\xi\{s_m^{-2}(k)\}$$

(iii)

$$s_m^{-2} \triangleq \lim_{k\to\infty} \xi\{s_m^{-2}(k)\}$$

exists

(iv)

$$\lim_{k\to\infty} \xi\{\epsilon_m^2(k)\} = q_1 \triangleq \frac{1}{12}2^{-2t_1}$$

where t_1 bits are used for the fractional representation for the states $Q[\hat{z}_m(k)]$.

The value of s_m^{-2} will depend on both the input signal characteristics and the filter coefficients, and no theoretical estimates are available. However numerical studies indicate that block floating point arithmetic can be better than *fixed point* arithmetic in terms of output arithmetic noise even when overflow (in the fixed point implementation) rarely occurs.

5 References

[1] Lamaire, R.O. and Lang, J.H. (1986). "Performance of digital linear regulators which use logarithmic arithmetic", *IEEE Trans. on Automatic Control*, Vol. AC-31, pp. 394-400.

[2] Jenkins, W.K. (1979). "Recent advances in residue number techniques for recursive digital filtering", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-27, pp. 19-30.

[3] Kaneko, T. (1973). "Limit cycle oscillations in floating-point digital filters", *IEEE Trans. Audio and Electro-Acoust.*, Vol. AV-21, No. 2, pp. 100-106.

[4] Williamson, D., Sridharan, S. and McCrea, P.G. (1985). "A new approach to block floating-point arithmetic in recursive filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-32, No. 7, pp. 719-723.

[5] Sridharan, S. and Williamson, D. (1986). "Implementation of high order direct form digital filter structures", *IEEE Trans. Circuits and Systems*, Vol. CAS-33, pp. 818-822.

[6] Williamson, D. (1986). "Roundoff noise minimization in fixed point digital filters using residues feedback", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-34, No. 5, pp. 1210-1220.