Equalisation Algorithms in Fixed Point Arithmetic

by

A.T. Markettos (CAI)

Fourth-year project in Group E, 2001/2002
Cambridge University Engineering Department

Abstract

The conversion from floating point to fixed point of Least Mean-Squares (LMS) and Recursive Least-Squares (RLS) equalisation algorithms is considered. Simulations suggest that the stability of LMS depends on the representation of its step size, whilst RLS is found to be stable for a wide range of parameter settings. RLS is implemented on a fixed-point DSP, and LMS on a custom fixed-point DSP core.

1 Introduction

Broadband wireless networks suffer multipath interference from reflections due to buildings, trees and vehicles in the transmission path. This causes inter-symbol interference (ISI) and nulls in the frequency band. Such high bitrate networks, particularly if they employ Single Carrier modulation schemes, require the use of an equaliser to cancel this ISI[2]. A time-domain equaliser consists of a filter trained by an adaptive equalisation algorithm such as Least Mean-Squares (LMS) or Recursive Least-Squares (RLS). It is desirable to be able to implement these on cheap fixed-point hardware – to do this we must first analyse whether they are stable in finite precision arithmetic, and use this to design an implementation.

2 Finite precision implementation

A system for evaluating the finite precision behaviour of equalisers for use on a fixed point processor has been developed. The aim was to examine the suitability of equalisation algorithms for implementation on cheap fixed point hardware.

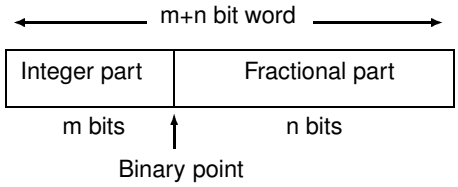


Figure 1: Fixed point representation

Fixed point values are represented using integers divided into integer and fractional parts (figure 1). Each fixed point variable has an associated point position, which can either be set globally or tailored for that variable. We can consider an arithmetic based on this representation, which deals with performing arithmetic on values a and b with different point positions p_a and p_b respectively ($\ll n$ and $\gg n$ represent left and right shifts by n places). Table 1 sets this out.

A common optimisation is that for addition, subtraction and assignment shifts are not required if the points for the two val-

Operation	Pre-operation	Arithmetical operator	Point of result
Assignment		$a = b \ll (p_a - p_b)$	p_a
Addition/ subtraction	$a' = a \ll (p_b - p_a)$ $b' = b \ll (p_a - p_b)$	$q = a' \pm b$ $q = a \pm b'$	p_b p_a
Multiplication		$q = a \times b$	$p_a + p_b$
Division	$a' = a \ll p_b$	$q = a' \div b$	p_a

Table 1: Fixed point arithmetic

ues are equal. We wish the C compiler to perform the work of optimising code using this arithmetic to remove unnecessary shifts, since the point positions of each variable are known on compilation.

A series of C preprocessor macros have been developed, which take on much of the work of converting a fixed point algorithm to floating point. The length of the integer representation can also be defined, and is typically set to the native word length of the machine in use. As well as shifting operations, the macros make use of C's predicate operator:

`(condition ? resultIfTrue : resultIfFalse)`

If the condition is known at compile time (such as comparing two different point positions), the compiler can ignore the half of the predicate which will never be evaluated, and optimise it away. Thus we can compare point positions and make the compiler take action on them, without any impact on the resulting code. The compiler will insert shifts only where necessary, with the result that if two values in an arithmetic operation have the same point position there will be no extra work carried out.

Each macro performs a normal arithmetic operation, such as assignment, addition or multiplication. Within the constraints of the C syntax, it is possible to directly replace an expression such as:

```
c = a+b;
```

with:

```
c = FIXED_ADD(a,POINT_A,b,POINT_B,POINT_C);
```

which performs the same function, but is also provided with the point positions of a, b and c. Thus clarity of the code is maintained. Underneath the macros perform the functions in the table, performing shifts as required.

3 Finite precision algorithms

These macros allow easy modification of existing floating point code, and allow the evaluation of different point positions. This was applied to two systems, a simple linear equaliser trained with the Least Mean-Squares (LMS) algorithm, and a decision feedback equaliser training with the Recursive Least-Squares (RLS) algorithm.

3.1 Least Mean-Squares (LMS)

A schematic diagram of the LMS-trained linear equaliser can be seen in figure 2. It is trained using a known training sequence sent from the transmitter, which is a fixed number of symbols long. Since fixed point

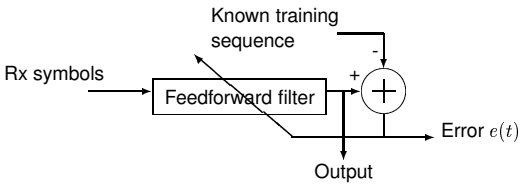


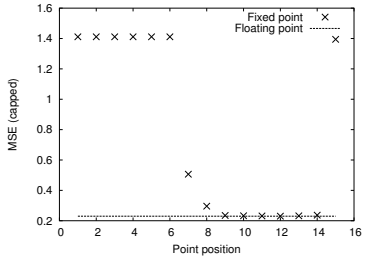
Figure 2: FIR-based linear equaliser

arithmetic only handles real values, it is necessary to split the calculation into real and imaginary parts, and a useful start is to write it in floating point arithmetic. Once done, the macros can be applied to the algorithm to convert it to fixed point.

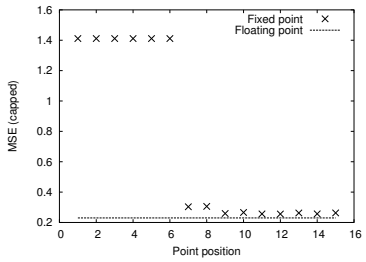
When converted to fixed point, we can examine its behaviour in relation to the point positions. Figure 3 shows its behaviour when varying the point positions of a number of parameters. The macros allow a default point to be specified, which is used for all values other than those under scrutiny. Unless otherwise stated the default point was 8 in these tests.

These plots show the mean squared error over training. To stop large values dwarfing small values, the MSE of each step is limited to 2 - values above this are taken as untrained. We can see that LMS is stable when given enough fractional bits - 8 or 9 in this 16 bit representation. Extra fractional bits make little difference to the result - either it trains correctly, or fails to train at all. This can be seen especially on the plot of μ, where when $\mu = 0.01$, 7 bits are required before this becomes non-zero in fixed point format.

This suggests μ becomes the limiting factor on the representation. To test this hypothesis, we notice that figure 3(a) has a step in MSE between correct and incorrect training. By differentiating this plot, we can find the position of this step, and



(a) Default point



(b) Point of step size μ

Figure 3: Effects of varying point position on 16 bit LMS. SNR=100dB, channel=[1,0,0.4], 5 taps, 1000 training symbols, 1000 data symbols, $\mu = 0.01$

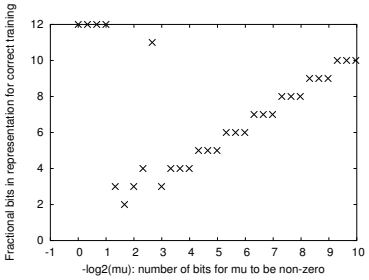
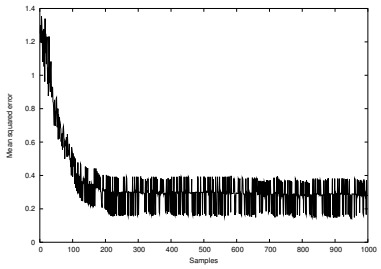
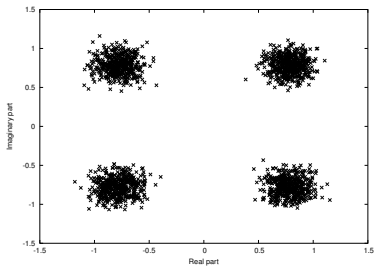


Figure 4: Fractional bits required for correct training when μ requires given number of bits to be non-zero. SNR=100dB, channel=[1,0,0.4], 5 taps, 1000 training symbols, 1000 data symbols

hence the fractional bits required for correct training. Figure 4 shows this number of fractional bits against the number of bits required to represent μ. The straight line agrees that μ is limiting.



(a) Training



(b) Constellation

Figure 5: LMS on DSP, noisy channel, small reflector: SNR=10dB, channel=[1,0,0.1], 1000 training symbols, 10000 data symbols, 5 taps

This shows that LMS is stable in finite precision arithmetic. A typical output from the filter can be seem in figure 5(a).

The frequency response of the resulting channel was also analysed (figure 6). From this is can be seen that at frequency nulls the (floating point) zero-forcing equaliser has a higher magnitude output than the fixed point LMS – this is due to restrictions on the magnitude of the representation. This does not affect the performance of the filter, since in the frequency nulls it would only amplify noise.

3.2 Recursive Least-Squares (RLS)

RLS is known to have stability problems when represented in finite precision arithmetic[1]. The fixed point macros can

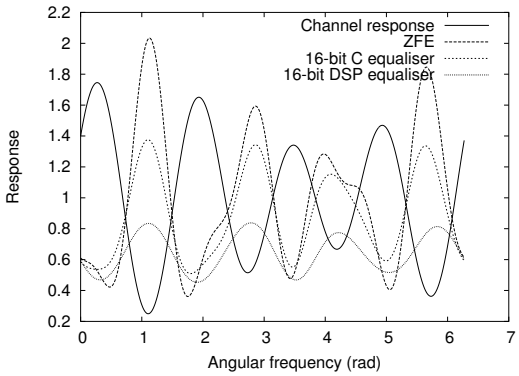


Figure 6: Frequency response of channel [1,0,0.2+0.1i,0.2+0.5i] and filters. SNR=100dB, 8 taps.

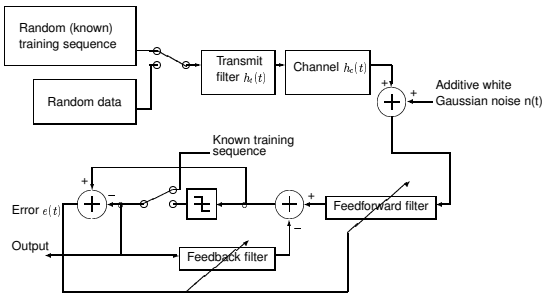
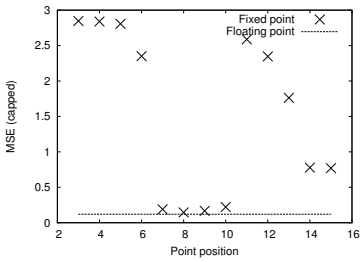


Figure 7: Decision feedback equaliser (DFE) filter system block diagram, trained with RLS

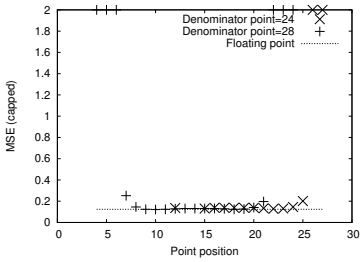
be used to investigate its behaviour in fixed point, and determine how it behaves when changing the point position of different variables. RLS was examined with a more complex decision feedback equaliser (figure 7).

A similar system was developed by converting floating point to fixed point with the macros. The results of altering various point positions were then noted.

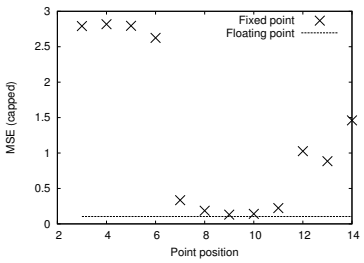
As figure 8(a) shows, the RLS was stable in 8.8 format in a 16 bit representation. These results were tested with a number of different channels and noise levels, and



(a) Default point, 16 bit datawords



(b) Default point, 32 bit datawords



(c) Point of inverse correlation matrix, 16 bit datawords

Figure 8: Effects of varying point position on RLS. SNR=20dB, channel=[0,0.2+0.1i,1,0.3+0.4i,0.2-0.3i], 5 taps, 32 training symbols, 100 data symbols

RLS was found to continue to be stable. The optimal point position was found to be 8 bits.

[1] suggests that the inverse correlation matrix P being non-positive definite is the cause of numerical instability, yet figure 8(c) suggests control of the point of this matrix can prevent this in common cases (although this is not exhaustive, and a more rigourous proof is required).

4 Fixed point implementation

Using the above results, it was possible to implement these algorithms on a fixed point digital signal processor (DSP). The processor used was part of a broadband wireless data link, to perform echo cancellation by training a filter from a known data sequence. A host processor pipes data into the memory of the DSP, starts it when a correllation event has occurred, and reads out the results for use by further hardware.

Originally this was an off-the-shelf floating point DSP. As described above, the RLS algorithm in floating point C was converted to fixed point. Following from the finite precision analysis of RLS, it was possible to compile this RLS algorithm for a substantially cheaper fixed point DSP. Using simulated received data, it was possible to ensure it converged as expected on this hardware under a wide variety of simulated conditions.

With the knowledge that LMS trains correctly using 8.8 arithmetic (8 bits integer, 8 bits fractional), it was possible to consider using a simple 16-bit DSP that was integrated into the modem FPGA. As no C compiler exists for this DSP, it was necessary to program the algorithm in assembler. The DSP architecture has 16-bit registers with a 32-bit accumulator for holding the result of multiply operations. To support 8.8 fixed point arithmetic in hardware, instructions were added to convert between 16 and 32 bit representations of 8.8 values, and these used to implement LMS on this DSP. Architectural and tool problems made this implementation and debugging difficult, but the results of the

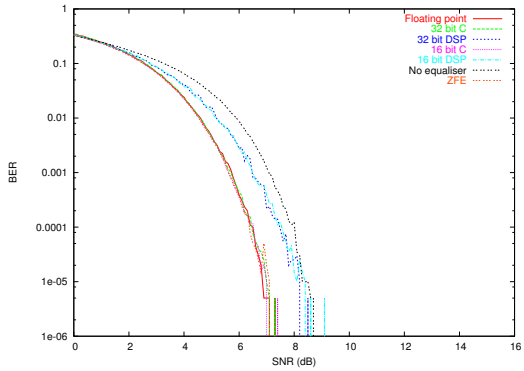


Figure 9: BER v SNR for 1000 training symbols, 10000 data symbols, 5 taps, averaged over 20 runs.

implementation can be seen in figure 9. This shows the 16-bit C implementation to be indistinguishable from the ZFE in terms of BER curve, while the DSP implementation is between C and no equaliser. It is suspected that this is due to bugs in the DSP implementation rather than a fundamental problem, but there was insufficient time for further debugging.

With the knowledge that 8.8 arithmetic is stable, it was possible to implement LMS with the linear equaliser on a custom digital signal processor with native support for 8.8 arithmetic. This showed similar behaviour to the version in C, except was more susceptible to bugs because of the difficulty of programming in assembler. Figure 9 shows that the BER performance of the LMS trained filter is almost identical to that of a Zero Forcing Equaliser, suggesting no substantial training errors.

The precision of the C and DSP versions was increased to 32 bit, with no noticeable difference in performance. This agrees with the statement above that once stable, LMS does not improve with extra precision.

5 Conclusion

We have demonstrated that:

- LMS is stable in finite precision arithmetic, the limit being the step size μ. The value of μ needed for correct training should be chosen, then the precision of the arithmetic decided from that.
- RLS is stable in 8.8 fixed point arithmetic over a wide range of channels and noise. Stability can be maintained by controlling the point of the inverse correlations matrix P. Further work is required to establish what limits the stability.
- Both have been implemented for use in the equaliser of a broadband wireless receiver, LMS and RLS on an off-the-shelf fixed point DSP and LMS on a small custom DSP. The custom DSP implementation of LMS needs more work for final debugging before its performance can be properly assessed.

References

[1] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall, fourth edition, 2002.

[2] Nigel King. Broadband business and residental radio access. *http://www.pipinghotnetworks.com/site/whitepaper/broadbandbusiness.pdf*, October 2000.