THE GAUSS-SEIDEL FAST AFFINE PROJECTION ALGORITHM

Felix Albu[1], Milan Tichy[2], Nick Coleman[3], Anthony Fagan[1]

[1]DSP Group, UCD, Belfield 4, Dublin, Ireland, felix_albu@ieee.org
[2]UTIA, Pod vodarenskou vezi 4, 182 08 Prague 8, Czech Republic, tichy@utia.cas.cz
[3]Dept EE, The University, Newcastle, NE1 7RU, UK, j.n.coleman@ncl.ac.uk

ABSTRACT

In this paper we propose a new stable Fast Affine Projection algorithm based on Gauss-Seidel iterations (GSFAP). We investigate its implementation using the logarithmic number system (LNS) and compare it with other two FAP algorithms. A method to simplify its implementation is also proposed. We show that the 32-bit or 20-bit LNS implementation of the GSFAP algorithm is superior to those of other FAP algorithm. Its application for acoustic echo cancellation is also investigated.

1. INTRODUCTION

Adaptive filtering is widely used in echo cancellation, noise cancellation, system identification, active noise control, channel equalization and in products like data communications systems, network echo cancellers, acoustic echo cancellers for hands-free telephones. The performance of a specific adaptive filtering system is affected by the choice of its adaptation algorithm. The well-known normalized LMS (NLMS) algorithm has been widely used but it has slow asymptotic convergence. The affine projection algorithm (APA) [1] can be considered as a generalization of the NLMS algorithm. However, its fast version [2], when implemented with an embedded FRLS (Fast Recursive Least Squares) algorithm suffers from numerical instability [2]. The complexity of this algorithm is $2L + 20N$, where L is the length of the filter and N is the projection order. Other difficulties are its memory requirements and code overhead. Because of these disadvantages using an FRLS procedure instead of the standard RLS procedure do not necessarily represent the most economical solution. Other forms of the standard FAP that use sliding-window RLS type approach have been proposed in [3] and [4]. These alternate FAP algorithms lead to a more accurate estimation of the auto-correlation matrix inverse but have no feedback incorporated. If the estimations deviate from the accurate value, the errors propagate to the next iterations, causing the adaptive filter to fail sometimes [5]. A frequently proposed remedy is to re-start periodically a new inversion process. Even so, the numerical errors accumulate so fast sometimes that the re-starting period would have to be made very small. Therefore the complexity associated with this procedure is high. Another improved FAP algorithm using the conjugate gradient (CG) method to do the matrix inversion was proposed in [5]. It was called CGFAP and it was proved that it is stable and easy to implement in comparison with other FAP algorithms. It uses a feedback scheme so that the numerical errors do not accumulate. CG is a non-linear programming method that seeks the minimum of a quadratic cost function iteratively [6]. It has been verified on floating point and fixed-point DSP platforms including 16 and 24 bits ones [5]. In this paper we study the behaviour of 20-bit or 32-bit LNS or FLOAT implementations of the classical FAP algorithm [4], the CGFAP algorithm and our new proposed GSFAP algorithm (Gauss-Seidel Fast Affine Projection) algorithm. Also, we propose a version for multi-input system and a filtered-x system version of the GSFAP algorithm. We investigate its performances for an acoustic echo cancellation and we present the multi-input case and the filtered-x version.

In section 2 we briefly review the development of the GSFAP algorithm. The logarithmic number system is presented in section 3. The results of the implementation of GSFAP algorithms using logarithmic arithmetic and conventional 32-bit floating-point are provided in section 4. Section 5 concludes this work.

2. GSFAP ALGORITHM

We will use most of the notations and the definitions presented in [5]. It is shown there that the matrix inversion problem implied by FAP algorithms reduce to solving a set of N linear equations $\mathbf{R}(n)\underline{P}(n) = \underline{b}$, where $\underline{b}$ is a N vector with only one non-zero element, which is unity at the top and $\mathbf{R}(n)$ is symmetric and positive definite. Compared with the Jacobi method for the same class of problem, the Gauss-Seidel method offers faster

convergence. It uses updated values as soon as they are available [7]. If we have to solve $\mathbf{A}x = b$, where **A** is an NxN symmetric matrix, the Gauss-Seidel method computes $x_i^{(k)} = \left(b_i - \sum_{j<i} a_{ij} x_j^{(k)} - \sum_{j>i} a_{ij} x_j^{(k-1)} \right) / a_{ii}$. It is known that if the matrix A is symmetric and positive definite, the GS iteration is guaranteed to converge [7]. We perform just one iteration and use as an initial estimation for $\underline{P}(n)$ its previous value. The GSFAP algorithm is presented in the following lines:

Initialisation

$$\underline{V}(-1) = \underline{0}, \underline{\eta}(-1) = 0, \underline{s}(-1) = 0,$$
$$\mathbf{R}(-1) = \delta \mathbf{I}, \alpha = 1, \underline{P}(-1) = \underline{b} / \delta \quad (1)$$

Processing in sampling interval *n*

$$\mathbf{R}(n) = \mathbf{R}(n-1) + \underline{\xi}(n)\underline{\xi}^T(n) - \underline{\xi}(n-L)\underline{\xi}^T(n-L) \quad (2)$$

Solve $\mathbf{R}(n)\underline{P}(n) = \underline{b}$ using one GS iteration [7] (3)

$$\underline{V}(n) = \underline{V}(n-1) + \alpha \eta_{N-1}(N-1)\underline{X}(n-N) \quad (4)$$

$$e(n) = d(n) - \underline{V}^T(n)\underline{X}(n) - \alpha \underline{\eta}^{-T}(n-1)\underline{\tilde{R}}(n) \quad (5)$$

$$\underline{\varepsilon}(n) = e(n)\underline{P}(n) \quad (6)$$

$$\underline{\eta}(n) = \begin{bmatrix} 0 \\ \underline{\overline{\eta}}(n-1) \end{bmatrix} + \underline{\varepsilon}(n) \quad (7)$$

Table 1. The GSFAP algorithm.

where $\mathbf{I}$ is an NxN identity matrix, δ is a regularization factor that prevents the input auto-correlation matrix $\mathbf{R}(n)$ from becoming ill-conditioned. $\underline{\tilde{R}}(n)$ is an N-1 vector that consist of the N-1 lower-most elements of the N vector $\underline{R}(n)$, which is the left column of $\mathbf{R}(n)$. $\underline{\overline{\eta}}(n)$ is an N-1 vector that consist of the N-1 upper-most elements of the N vector $\underline{\eta}(n)$, the scalar $\eta_{N-1}(n)$ is the lower-most element of $\underline{\eta}(n)$, $x(n)$ is the input signal and $d(n)$ is the desired output signal. The normalized step size can be chosen within a range from 0.7 to 1. There is a wide acceptable range for the regularised factor that prevents the input auto-correlation matrix from becoming ill-conditioned. The algorithm can be easily modified for multi-input system case by using the way reported in [8]. The only difference is the shifting property of the input vector $\underline{\xi}(n)$. Also, as it is shown in [8] the recursive formula for step (2) becomes

$$\mathbf{R}(n) = \mathbf{R}(n-1) + \sum_i (\underline{\xi}_i(n)\underline{\xi}_i^T(n) - \underline{\xi}_i(n-L)\underline{\xi}_i^T(n-L)) \quad (8)$$

where $\underline{\xi}_i(n)$ is the corresponding vector for the input *i*. However, the added complexity is low. Unlike the algorithm proposed in [8] that use the matrix inversion lemma twice more for each added input the modified multi-input GSFAP algorithm solve equation 3 with only one iteration. Therefore the complexity of this step is reduced from $4N^2 i + 4Ni$ divisions and multiplications (where *i* is the number of the inputs) to only about N^2 multiplications and division. The same modifications proposed in [8] for the two output filtered-x system can be applied in order to obtain the filtered-x GSFAP version. Identical formulas presented in [8] are needed to generate the outputs of the corresponding filtered-x system. The total computation for the CGFAP algorithm is $2L + 2N^2 + 9N + 1$ MACs and 1 division [5]. The GSFAP algorithm has $2L + N^2 + 4N - 1$ MACs. It can be efficiently implemented with only 1 division if efficiently implemented because of $\mathbf{R}(n)$ special structure. GSFAP algorithm could use higher values of the projection order than CGFAP for the same computational effort because of its reduced complexity (it has $N^2 + 5N + 2$ MACs less than CGFAP). Therefore it is suitable to be implemented with most commercial DSPs. Also, we investigated the GSFAP behavior in 32-bit or 20-bit logarithmic number system.

3. THE LOGARITHMIC NUMBER SYSTEM

Contemporary microprocessors perform real arithmetic using the floating-point system. Although this method has served well over the past decades, it suffers from a number of disadvantages which render it unsuitable for very high-speed computation and which inhibit its more widespread use, for example in application-specific integrated circuits or smaller microprocessor devices. Floating-point circuits are large, complex and much slower than fixed-point units; they require separate circuitry for the add/ subtract, multiply, divide, and square-root operations; and all floating-point operations are liable to a maximum half-bit rounding error.

As an alternative to floating-point, the logarithmic number system offers the potential to perform real multiplication, division and square-root at fixed-point speed and, in the case of multiply and divide, with no rounding error at all. These advantages are, however, offset by the problem of performing logarithmic addition

and subtraction. Hitherto this has been slower or less accurate than floating-point, or has required very cumbersome hardware. Following the way reported in [9] and [10] it is now possible to perform logarithmic addition and subtraction with speed and accuracy equivalent to that of floating-point. In [10] we also described a 20-bit LNS implementation in which the addition-subtraction operation is performed with only 11 kbits of ROM and a small amount of additional circuitry. In view of the suitability of this scheme for an ASIC implementation, we have used it in this paper to demonstrate the GSFAP algorithm in practice. The 32-bit floating-point representation consists of a sign, 8-bit biassed exponent, and 23-bit mantissa. The LNS format is similar in structure (see Fig. 1).

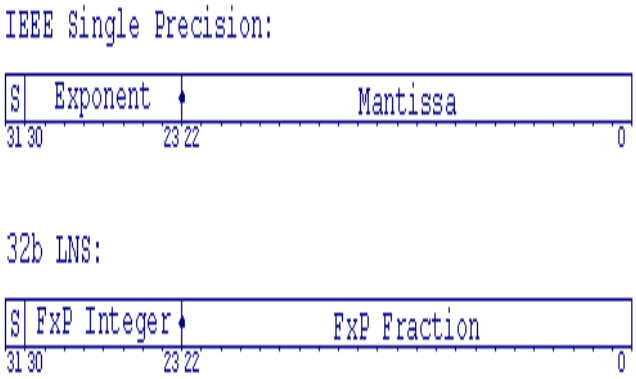

Fig. 1 IEEE standard single precision floating point representation and the 32-bit LNS format

x + y	ADD	Lz = Lx + log(1+2^(Ly-Lx)), Sz depends on sizes of x,y
x - y	SUB	Lz = Lx + log(1-2^(Ly-Lx)), Sz depends on sizes of x,y
x * y	MUL	Lz = Lx + Ly, Sz = Sx OR Sy
x / y	DIV	Lz = Lx - Ly, Sz = Sx OR Sy
x^2	SQU	Lx << 1, Sz = Sx
x^0.5	SQRT	Lx >> 1, Sz = Sx
x^-1	RECIP	Lz = Lx, Sz = ¬Sx
x^-0.5	RSQRT	Lz = Lx >> 1, Sz = ¬Sx

Table 1. LNS Arithmetic Operations

The 'S' bit again indicates the sign of the real value represented, with the remaining bits forming a 31-bit fixed point word in which the size of the value is encoded as its base-2 logarithm in 2's complement format. Since it is not possible to represent the real value zero in the logarithmic domain, the 'spare' (most negative) code in the 2's complement fixed point part is used for this purpose, which is convenient since smaller real values are represented by more negative log-domain values. The chosen format compares favorably against its floating-point counterpart, having greater range and slightly smaller representation error. A 20-bit LNS format is similar. It maintains the same range as the 32-bit, but has precision reduced to 11 fractional bits. This is comparable to the 16-bit formats used on commercial DSP devices. The 20-bit version requires just 10,920 bits of lookup tables. The 32-bit LNS implementation uses 321,536 bits of lookup tables. Table 1 presents the LNS arithmetic operations. More details about the logarithmic number system are available at http://www.ncl.ac.uk/eece/elm/.

4. SIMULATIONS

In these simulations the excitation signal is amplitude normalised speech, sampled at 8 kHz, the echo path has the length *L*, the projection number is N. The convergence of the algorithms were compared by using the squared norm of the difference between the LEM model and the adaptive filter (in dB) [11]. The parameter μ for the all FAP and NLMS algorithms was set to 1. The CGFAP algorithm performs one division per sample. This division is not performed and zero is assigned if the denominator is not positive or lower than a specified threshold. This threshold was fixed to 10^{-10} in our simulations. The echo path represents a room impulse response and is taken from [11]. The projection order is N=10. We found that the 32-bit GSFAP or CGFAP finite implementations (FLOAT or LNS) have virtually identical performances (Figs. 2-3). As expected, its initial convergence is better than that of the NLMS algorithm (see Fig. 2). The figure shows some losses in performances due to lower finite precision of 20-bit versions in comparison with their 32-bit versions. Also, our simulations shown that the CGFAP and GSFAP have virtually identical performance. The classical FAP algorithm uses a sliding window fast RLS algorithm that is difficult to implement, memory intensive and potentially numerically unstable. The 32-bit LNS or FLOAT implementations of classical FAP algorithm (without the restarting procedure) is unstable sometimes. In all cases where the 32-bit or 20-bit FLOAT or LNS implementation of Gay's basic FAP algorithm were unstable, the GSFAP algorithm implementations remained stable. As an iterative method, GS method approaches $\underline{P}(n)$ with a delay and this tracking error isn't a problem since $\mathbf{R}(n)$ varies at a

slower rate because $L >> N$. This delay allows us not to update $\underline{P}(n)$ every sample.

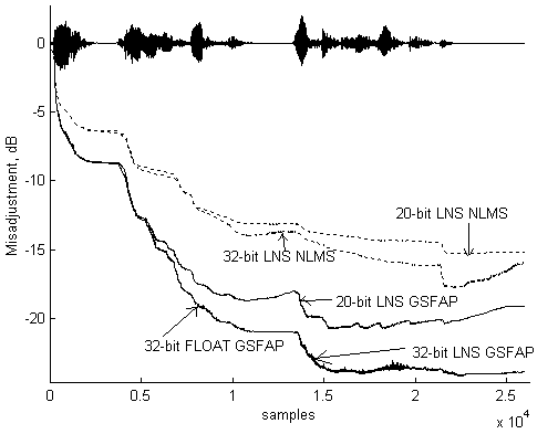

Fig. 2. The learning curves for 32-bit and 20-bit implementations of GSFAP and NLMS algorithms (32-bit FLOAT and 32-bit LNS curves almost co-incidental, L=1000, N=10)

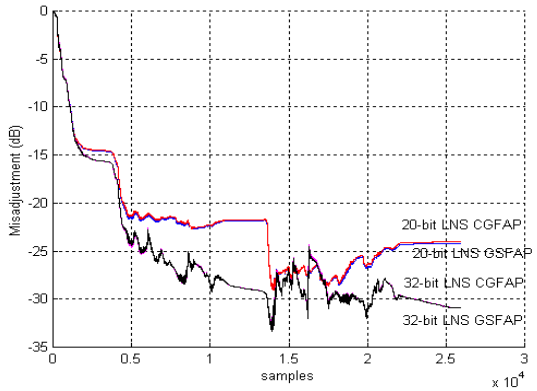

Fig. 3. The learning curves for 32-bit and 20-bit LNS implementations of GSFAP and CGFAP algorithms (32-bit or 20-bit LNS curves almost co-incidental for both algorithms, L=256, N=10, full echo path)

Figs. 4-5 show that updating $\underline{P}(n)$ every second up to fourth sample does not have a significant effect on the output error (Fig. 5 illustrate an example for the 20-bit GSFAP implementation). The less updating iterative method still approximates well the exact solution, although the error and the number of iterations to converge are higher (see Fig. 4). Therefore, the average number of MACs and divisions is p times smaller for GS section $(2L + N^2 / p + (5 - 1/p)N - 1$ MACs and 1division, where p depends on L or N values, usually between 2 and 5). By updating $\underline{P}(n)$ every fifth sample, the difference is increased to at least $1.8N^2 + 4.2N + 2$ MACs per sample in comparison with CGFAP. Higher values of p are possible, especially for high values of L or N. The reduction in complexity is important.

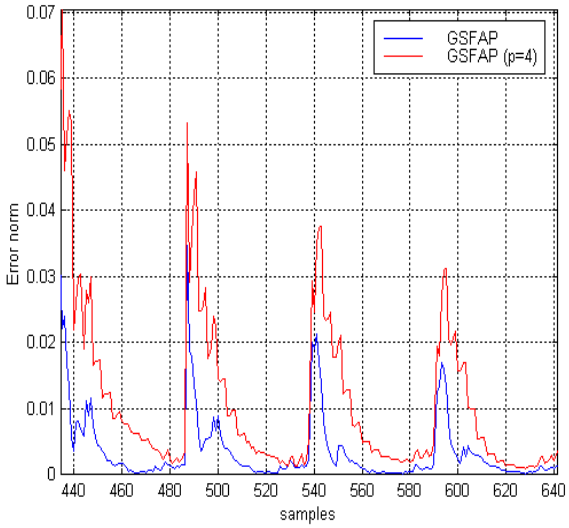

Fig. 4. The error norm between the exact solution (double precision) and the iterated solution of the linear system for different values of p (p=1 and p=4)

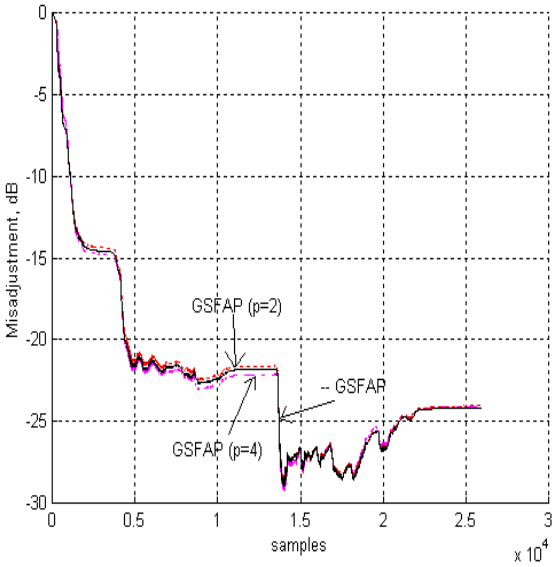

Fig. 5. Convergence of 20-bit LNS GSFAP implementation for different values of p (the curves are almost co-incidental most of time)

For example, with L=1000 and N=10, an NLMS needs 2025 MACs (assuming 25 MACs for a division), the CGFAP needs 2316 MACs. In comparison, the GSFAP needs 2154 MACs, while the GSFAP with p=2 needs 2119 MACs and GSFAP with p=5 needs only 2092 MACs. Therefore in this case, the increase is just about 2 % for GSFAP with p=5 in comparison with NLMS. The GSFAP algorithm needs the

least amount of computation of the three algorithms as long as $N \leq 16$ (see Fig. 6).

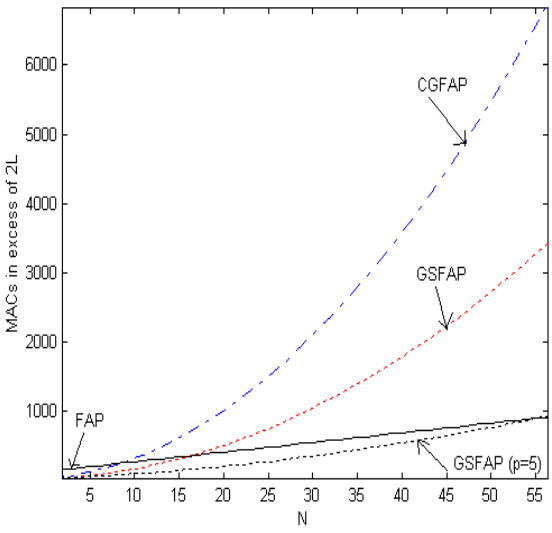

Fig. 6 Real time requirements of 3 FAPs

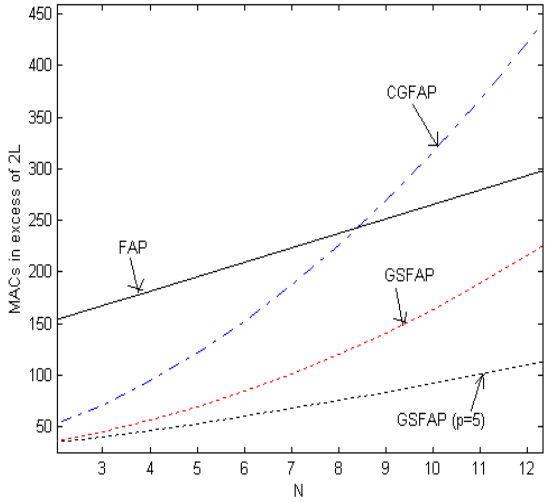

Fig. 7 Real time requirements of 3 FAPs (zoom for low values of N)

The value of corresponding *N* grows as *p* grows (it is 55 for *p*=5). However, N=10 is good enough for most voice applications. It can be seen from Fig. 7 that GSFAP algorithm is superior to all other considered FAP algorithms for this range of values for the projection order.

The algorithm was run on the European Logarithmic Microprocessor (ELM) simulator for the case L=1000 and N=12. Design of the microprocessor is in the final stages of layout, fabrication will be scheduled shortly, and a physically-knowledgeable VHDL simulator indicates a clock speed of 200 MHz. The inputs were quantized to 16 bits and the run took 12,286 cycles per timestep. That would allow a 16.2 KHz sampling rate at 200 MHz.

5. CONCLUSIONS

It has been verified by simulations that the GSFAP and its simplified forms is a stable alternative to the classical FAP algorithm without the restarting procedure. The GSFAP algorithms could provide a low-cost, efficient solution for other voice applications. It is only marginally more complex than NLMS and suitable for low-cost processors. Their 20-bit LNS implementation appears to offer a very attractive alternative to conventional arithmetic if the precision is not a major issue. This 20-bit implementation is particularly suitable for ASIC implementation, requiring 11 kbits of ROM and a minimal amount of additional circuitry for a complete add/subtract/multiply/divide unit. Our future work will be focused in implementing the GSFAP on FPGA following the way reported in [12]. We also intend to investigate its stability in 16-bit fixed-point precision and develop a multi-channel GSFAP algorithm suitable for active noise control systems.

6. ACKNOWLEDGMENTS

This work has been performed under the EU ESPRIT 33544 HSLA Long-term research project (http://www.ncl.ac.uk/eece/elm/), coordinated by the University of Newcastle, UK.

7. REFERENCES

[1] K. Ozeki, T. Umeda, 'An adaptive Filtering Algorithm Using an Orthogonal Projection to an Affine Subspace and its Properties,' Electronics and Communications in Japan, Vol. 67-A, No.5, 1984

[2] S. Gay, S. Tavathia, 'The Fast Affine Projection Algorithm', pp. 3023–3026, *ICASSP'95 Proceedings*

[3] Q.G. Liu, B. Champagne, and K. C. Ho, " On the use of a modified FAP algorithm in subbands for acoustic echo cancellation," in Proc. 7th IEEE DSP Workshop, Loen, Norway, 1996, pp. 2570-2573

[4] M. Ghanassi, B. Champagne, "On the Fixed-Point Implementation of a Subband Acoustic Echo Canceler Based on a Modified FAP Algorithm", 1999 IEEE Workshop on Acoustic Echo and Noise Control, Pocono Manor, Pennsylvania, USA pp. 128-131

[5] Heping Ding, "A stable fast affine projection adaptation algorithm suitable for low-cost processors", ICASSP 2000, Turkey, pp. 360-363

[6] David Luenberger, "Linear and Non-linear Programming", 2nd Edition, Addison-Wesley, 1984.

[7] R.Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, 'Templates for the solutions of linear systems: Building blocks for iterative methods', SIAM, 1994
[8] Y. Kaneda, M. Tanaka, J. Kojima, 'An Adaptive Algorithm with Fast Convergence for Multi-input Sound Control', Active95, pp. 993-1004, Newport Beach, California, USA.
[9] J.N. Coleman, E.I.Chester, 'A 32-bit Logarithmic Arithmetic Unit and Its Performance Compared to Floating-Point', *14th Symposium on Computer Arithmetic'*, Adelaide, April 1999
[10] J.N.Coleman, E.Chester, C.Softley and J.Kadlec, "Arithmetic on the European Logarithmic Microprocessor", IEEE Trans. Comput. Special Edition on Computer Arithmetic, July 2000, vol. 49, no. 7, pp. 702-715; and erratum October 2000, vol. 49, no. 10, p.1152
[11] C. Breining, P. Dreitseitel, E. Hansler, A. Mader, B. Nitsch, H. Pudeer, T. Scheirtler, G. Schmidt, and J.Tilp, ' Acoustic echo control- An application of very high order adaptive filters,' *IEEE Signal Processing Magazine*, pp. 42-69, July 1999
[12] F. Albu, J. Kadlec, N. Coleman, A. Fagan, "Pipelined Implementations of the Modified EF-LSL Algorithm", ICASSP2002, pp. 2681-2684, May 2002, Orlando, U.S.A

The matlab code and the speech file used in the Gauss-Seidel Fast Affine Projection algorithm can be found at http://falbu.50webs.com/fap/felix_gs.html

The reference for the paper is: F. Albu, Jiri Kadlec, Nick Coleman, Anthony Fagan, "The Gauss-Seidel Fast Affine Projection Algorithm", IEEE Workshop, SIPS 2002, pp. 109-114, San Diego, U.S.A, October 2002