# NSLMS: a Proportional Weight Algorithm for Sparse Adaptive Filters

R. K. Martin and C. R. Johnson, Jr.*
School of Electrical Engineering
Cornell University
Ithaca, NY 14853
{frodo,johnson}@ece.cornell.edu

**Abstract**

*In this paper we discuss a proportional weight algorithm that is similar to LMS. The distinction is that the new algorithm (called normalized sparse LMS, or NSLMS) has a time-varying vector stepsize, whose coefficients are proportional to the magnitudes of the current values of the tap estimates.*

*We show that when the system to be identified is sparse, NSLMS converges faster than LMS (to the same asymptotic MMSE for both algorithms). We also discuss the effect of the initialization on the performance of NSLMS.*

## 1 Introduction

Researchers in fields such as wireless communication, echo cancellation, and underwater acoustic communication have noted that transmission channels are often sparse [1], [2], [3]. Often, sparsity means that if the channel is represented as a finite impulse response (FIR) filter, then there will be a few large taps, separated by many much smaller taps, as exemplified by Figure 1. In this paper, we will use a broader definition of sparsity, which is that a given tap in a sparse channel has a high probability of being small and a low probability of being high. When there is any ambiguity, these two definitions can be called "conventional sparsity" and "probabilistic sparsity," respectively, and in general we will use "sparsity" to refer to the latter definition. A conventionally sparse filter is also probabilistically sparse, but the converse is not necessarily true.

Typical adaptive filter algorithms such as the Least Mean Square (LMS) algorithm, the Constant Modulus Algorithm (CMA), and Decision Directed (DD) algorithms do not exploit this a priori information. Many researchers have tried to modify existing algorithms or create new ones, with the intent to reduce the complexity or to improve performance. A summary of this work can be found in Chapter 2 of [4].

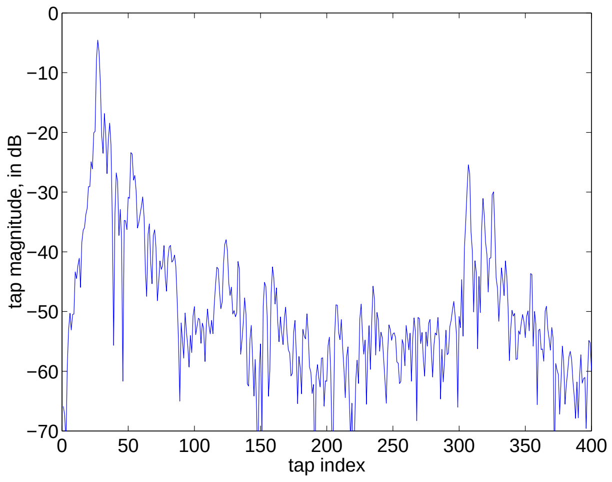

Figure 1: Example of a conventionally sparse channel.

These approaches to sparsity can be divided into two philosophies. Typical approaches to exploiting sparsity are motivated by complexity reduction (at the expense of a possible performance loss), which is often accomplished by only updating a subset of the filter taps [1], [2], [3]. The alternate philosophy is to improve performance at the expense of retaining a possibly high complexity. For example, the Exponentiated Gradient (EG) algorithm [5] has recently been shown to have better performance than typical gradient methods when the target weight vector is sparse [6], [7], though it does not decrease the required complexity. Our approach in this paper is similar. We will propose an algorithm similar in form to LMS (but with slightly higher complexity) that performs better than LMS when the target system is sparse.

*This work was supported in part by NSF grant ECS-9811297, Applied Signal Technology, and NxtWave Communications.

## 2 System model and algorithm

Consider a standard channel identification (or equalization) setting, where $\mathbf{w}$ is a length $N$ vector representing the channel estimate (or equalizer) and $\mathbf{w}^*$ is the optimal value of the unknown system. The output of the adapted filter $y_k$, the desired signal $d_k$, and the error signal are generated according to

$$\begin{aligned} y_k &= \mathbf{X}_k^T\ \mathbf{w}_k \\ d_k &= \mathbf{X}_k^T\ \mathbf{w}^* \\ e_k &= d_k - y_k, \end{aligned} \tag{1}$$

where $\mathbf{X}_k = [x(k) \cdots x(k-N+1)]$ is the regressor of the input to $\mathbf{w}$ and $\mathbf{w}^*$. We need not assume $d_k$ is generated by an FIR parameterization, but it greatly simplifies the analysis. The most common algorithm for determining $\mathbf{w}$ is the LMS algorithm,

$$\begin{aligned} \mathbf{w}_{k+1} &= \mathbf{w}_k - \mu\ \hat{\bigtriangledown}_k \\ &= \mathbf{w}_k + \mu\ e_k \mathbf{X}_k, \end{aligned} \tag{2}$$

where $\hat{\bigtriangledown}_k$ is the estimate of the gradient of the MSE cost surface at time $k$ with respect to $\mathbf{w}_k$. Now consider the vector update rule

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu\ D\ \hat{\bigtriangledown}_k, \tag{3}$$

where $D$ is a diagonal matrix which may depend on the current value of the weight estimates, $\mathbf{w}$. This is a more general form of the LMS update rule (for which $D = I$). Note that $\mu D$ is effectively a time-varying vector stepsize.

Many new algorithms of the form of (3) were introduced in [8] and [4]. The algorithm we will consider in this paper was introduced in [4], and is called "Normalized Sparse LMS", or NSLMS. The name was chosen since the algorithm was found to perform well in sparse environments, though in this paper we consider non-sparse environments as well.

NSLMS is defined by the parameterization of $D$ as

$$D_{ii} = \frac{|w_k^i| + \epsilon}{\text{AVG}(|w_k^j| + \epsilon)}, \tag{4}$$

where AVG(.) means $\frac{1}{N}\sum_{j=1}^{N}(.)$, $\epsilon$ is a small constant, and the off-diagonal elements of $D$ are zero. The corresponding NSLMS algorithm is

$$w_{k+1}^i = w_k^i + \mu \left( \frac{|w_k^i| + \epsilon}{\text{AVG}(|w_k^j| + \epsilon)} \right) e_k\ x_k^i. \tag{5}$$

The essential advantage of (5) is that large taps get large updates, and small taps get small updates. Heuristically, this says that the MSE should decrease rapidly, since the taps currently perceived to be important (i.e. the largest) get an advantage, though this is at the expense of the less important (i.e. smaller) taps. (Note that there is some similarity between NSLMS and the $EG^{\pm}$ algorithm in equations (3) and (4) of [6].) There is of course a disadvantage: if the optimal weight vector changes drastically, then the estimate will temporarily have the wrong impression of which taps are important. Fortunately, gradual changes do not cause such an effect.

There are three key points to note regarding (5). First of all, the denominator in (4) was chosen to keep the sum over $i$ of the effective stepsizes ($\mu D_{ii}$) constant as the actual weights vary. This keeps the sum of the stepsizes from becoming small near the origin (as it would without normalization). The second point is that the term $\epsilon$ keeps the stepsizes of *individual* taps from going to zero as individual taps go to zero. The third point is that according to equations given in [6], [8], and [9], NSLMS and LMS have the same asymptotic MSE, so long as the stepsize $\mu$ is the same for both algorithms.

Section 3 analytically characterizes the situations in which NSLMS outperforms LMS, in terms of convergence speed. Note that this is a fair comparison, since the two algorithms have the same asymptotic MSE. Section 4 compares the complexity of implementing LMS versus NSLMS. Section 5 provides simulations of situations in which NSLMS performs better and in which LMS performs better, and Section 6 concludes.

## 3 Analysis

In this section, we will determine the regions of the parameter space in which NSLMS moves faster than LMS. We will perform this analysis in the error space, $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}^*$, and we will consider the average system (i.e. replace the estimate of the gradient with the true gradient). The general update equation becomes

$$\mathbf{v}_{k+1} = (I - \mu D R)\ \mathbf{v}_k,$$

where $D = I$ for LMS. Define the change in the error vector from time $k$ to time $k+1$ by

$$\Delta \mathbf{v}_k = -\mu\ D\ R\ \mathbf{v}_k.$$

The MSE's at times $k$ and $k+1$ are [9]

$$\xi_k = \mathbf{v}_k^T\ R\ \mathbf{v}_k,$$

$$\xi_{k+1} = (\mathbf{v}_k + \Delta\mathbf{v}_k)^T\ R\ (\mathbf{v}_k + \Delta\mathbf{v}_k).$$

This means that if we are at $\mathbf{w}_k$ at time $k$, then the change in the MSE ($\xi_{k+1} - \xi_k$) will be

$$\begin{aligned} (\Delta\xi)_k &= (\mathbf{v}_k + \Delta\mathbf{v}_k)^T\ R\ (\mathbf{v}_k + \Delta\mathbf{v}_k) - \mathbf{v}_k^T\ R\ \mathbf{v}_k \\ &= -\mu \mathbf{v}_k^T\ R(2D - \mu DRD)R\ \mathbf{v}_k. \end{aligned} \tag{6}$$

If we assume $\mu$ is small, then we get

$$(\Delta\xi)_k = -2\mu \mathbf{v}_k^T \ R \ D \ R \ \mathbf{v}_k. \tag{7}$$

NSLMS will move faster than LMS when $(\Delta\xi)_{NSLMS} < (\Delta\xi)_{LMS}$ (remember $\Delta\xi < 0$). Dividing by $-2\mu$, this occurs when

$$\mathbf{v}_k^T \ R \ (D - I) \ R \ \mathbf{v}_k > 0 \tag{8}$$

Henceforth, we will drop the $k$'s for simplicity. To visualize the results, consider the case $N = 2$. Let the input autocorrelation be

$$R = \begin{bmatrix} 1 & \beta \\ \beta & 1 \end{bmatrix}. \tag{9}$$

The 1's on the diagonal amount to assuming the input has unit power, $\mathrm{E}\left[x^2(k)\right] = 1$. Since $\beta = \mathrm{E}\left[x(k)x(k-1)\right]$, we must have $|\beta| \le 1$, with equality only in the degenerate case. Then (8) becomes

$$\begin{aligned} &\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 1 & \beta \\ \beta & 1 \end{bmatrix} \left( \left( \frac{|w_1| - |w_2|}{|w_1| + |w_2| + 2\epsilon} \right) \right. \\ &\left. \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right) \begin{bmatrix} 1 & \beta \\ \beta & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} > 0 \end{aligned} \tag{10}$$

Since $|w_1| + |w_2| + 2\epsilon > 0$, we can multiply it out. Simplifying what remains yields

$$(|w_1| - |w_2|)\left(1 - \beta^2\right)\left(v_1^2 - v_2^2\right) > 0. \tag{11}$$

Recall that $|\beta| \le 1$, so $\left(1 - \beta^2\right)$ is always positive (or zero in the degenerate case). Since $v_i = w_i - w_i^*$, the condition for NSLMS to be faster than LMS (for the case $n = 2$) is

$$(|w_1| - |w_2|) \cdot \left((w_1 - w_1^*)^2 - (w_2 - w_2^*)^2\right) > 0. \tag{12}$$

Since $\left(w_1^2 - w_2^2\right)$ has the same sign as $(|w_1| - |w_2|)$, we can factor this condition as

$$\begin{aligned} &(w_1 + w_2) \cdot [(w_1 - w_1^*) + (w_2 - w_2^*)] \\ &\cdot (w_1 - w_2) \cdot [(w_1 - w_1^*) - (w_2 - w_2^*)] > 0. \end{aligned} \tag{13}$$

For this to be positive, an even number of the factors must be positive. Thus, the plane should be divided by four lines:

$$\begin{aligned} w_2 &= w_1, \\ w_2 &= -w_1, \\ w_2 &= w_2^* + (w_1 - w_1^*), \\ w_2 &= w_2^* - (w_1 - w_1^*). \end{aligned} \tag{14}$$

A plot of the region in which NSLMS is faster than LMS is given in Figure 2, where the optimal weight

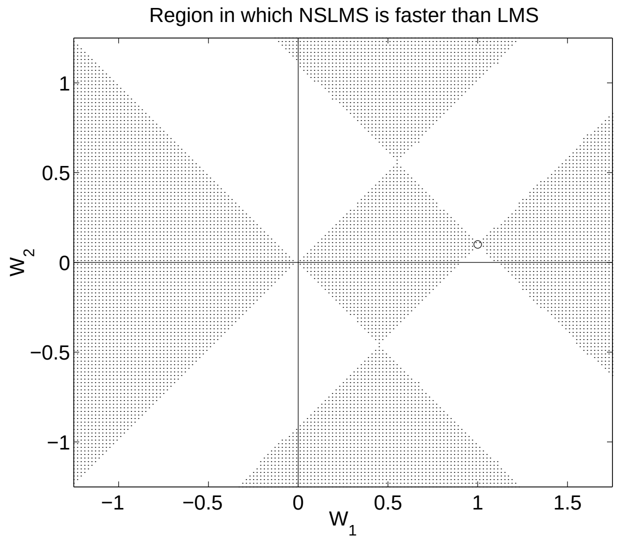

Figure 2: Region (shaded) in which NSLMS outperforms LMS. The circle is the true weight vector $\mathbf{w}^*$.

vector $\mathbf{w}^* = [1, 0.1]$ is marked by a circle. If you are in the shaded region at time $k$, then the decrease in MSE from time $k$ to time $k+1$ will be larger for NSLMS.

There are two points to make regarding Figure 2. The first is that if we initialize at the origin, the path that NSLMS traces out on its way to the optimal solution will be contained within the central shaded diamond, indicating that NSLMS moves faster than LMS along the entirety of the path, and thus converges faster. The second point is that NSLMS displays a directionality. That is, near the optimal solution, the speed of the algorithm will depend upon which parameter has a larger error. If the large tap has a larger error (i.e., an offset mostly in the horizontal direction), then the algorithm will converge quickly. If the small tap has a larger error (i.e., an offset mostly in the vertical direction), then the algorithm will converge slowly.

Now consider larger values of $n$. The analysis becomes intractable unless we assume $R = I$. By an analysis similar to that above, it can be shown that NSLMS is the faster algorithm when

$$\sum_{i=1}^{n} (w_i - w_i^*)^2 \sum_{j=1}^{n} (|w_i| - |w_j|) > 0. \tag{15}$$

(Compare to (11) with $\beta = 0$, $n = 2$.) In Figure 2, the line connecting the origin and the optimal weight vector is contained entirely within the shaded region. This is also true for $n > 2$. To see this, note that the line in question consists of all $\mathbf{w}$ such that $\mathbf{w} = \alpha\mathbf{w}^*$.

Then (15) becomes

$$|\alpha|\,(\alpha-1)^2 \sum_{i=1}^{n} (w_i^*)^2 \sum_{j=1}^{n} \left(|w_i^*| - |w_j^*|\right) > 0. \qquad (16)$$

We can ignore $|\alpha|\,(\alpha-1)^2$ since it is non-negative. It can then be shown that (16) is equivalent to

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1} \left(|w_i^*| - |w_j^*|\right) \left[(w_i^*)^2 - (w_j^*)^2\right] > 0. \qquad (17)$$

Note that for a given term in the summation, the first factor has the same sign as the second factor, so each term in the summation is non-negative. Thus, the condition is always met (except sometimes the summation could be zero, meaning both algorithms move at the same speed). What we have shown is that for arbitrary $n$, the NSLMS update is better than the LMS update if the current weight vector estimate lies between the origin and the optimal weight vector. If we initialize at the origin, the weight vector will be relatively close to this line (and in fact when R = I, LMS moves exactly along this line, and the similarity of LMS and NSLMS suggests that NSLMS will stay near this line), so this result suggests that for a zero initialization, NSLMS will reach the optimal setting faster than LMS (with the same asymptotic MSE).

## 4 Complexity

In a hardware implementation, we must consider the number of additions, multiplications, and divisions separately (rather than just counting "flops", as in a software simulation). In LMS, to compute $y_k$ and $e_k$, we must compute a dot product of size $n$ and perform an additional subtraction, which requires $n$ additions and $n$ multiplications. The update rule for each time step requires one multiplication to compute $\mu \cdot e_k$, then another $n$ multiplications and $n$ additions to multiply this by $\mathbf{X}_k$ and add it to $\mathbf{w}_k$. The total complexity (per iteration) of LMS is shown in Table 1.

We can rewrite the NSLMS update as

$$w_{k+1}^i = w_k^i + \left(\frac{\hat{\mu}\; e_k}{\sum_j |w_k^j| + \hat{\epsilon}}\right)\left(|w_k^i| + \epsilon\right)\; x_k^i, \qquad (18)$$

where $\hat{\mu} = n\mu$ and $\hat{\epsilon} = n\epsilon$. We can store $\hat{\epsilon}$ in memory to save 1 multiplication per iteration. Assuming the parameters are real (which simplifies calculating $|w_j|$), we must expend an additional $n$ additions and a single division to determine the first quantity in parenthesis. We also need an additional $n$ multiplications and additions to compute $(|w_i| + \epsilon)\; x_k^i$ (one each for each $i$), so NSLMS takes $2n$ additions, $n$ multiplications, and a division in addition to the complexity of plain LMS. This is shown in Table 1.

The use of a division is not wholly unprecedented among the LMS family; normalized LMS (NLMS) [1] uses normalization also, but with respect to the input $\mathbf{X}$ rather than the weight vector $\mathbf{w}$.

Table 1: Complexity per iteration.

| | **LMS** | **NSLMS** |
|---|---|---|
| Additions | $2n$ | $4n$ |
| Multiplications | $2n$+1 | $3n$+1 |
| Divisions | 0 | 1 |

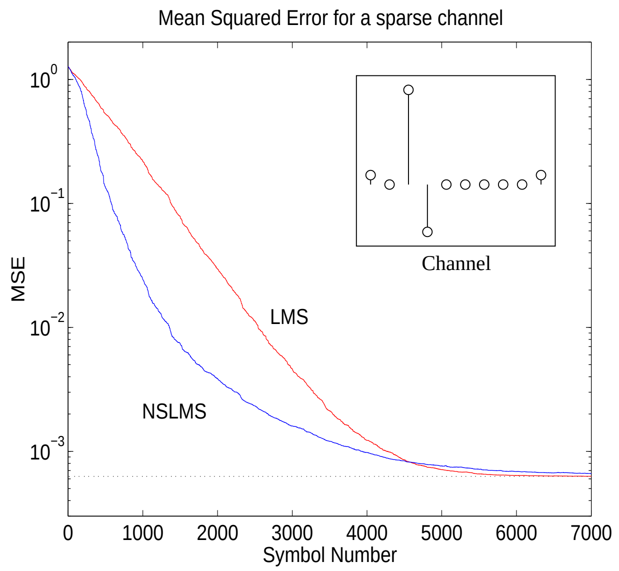

Figure 3: MSE vs. time for LMS and NSLMS for a sparse channel.

## 5 Simulations

Figure 3 shows the MSE vs. time for both algorithms, when the channel is sparse: $\mathbf{w}^* = [0.1, 0, 1, -0.5, 0, 0, 0, 0, 0, 0.1]$. Initialization was at the origin, $\mu = 0.001$, and $\epsilon = \frac{1}{16} = 0.0625$. Observe that for this channel, NSLMS converges much faster than LMS. We can think of NSLMS as dividing its stepsize up among the different taps. If there are only a few large taps, they get the lion's share of the stepsize, and can move very quickly. Of course the large number of small taps will all move slowly, but small taps are less significant in terms of reducing the error (if their optimal values are also small).

Figure 4 shows the MSE vs. time for both algorithms, averaged over 200 random ten-tap channels. Each channel tap of each ensemble member was chosen from a uniform distribution over $[-1, 1]$, leading

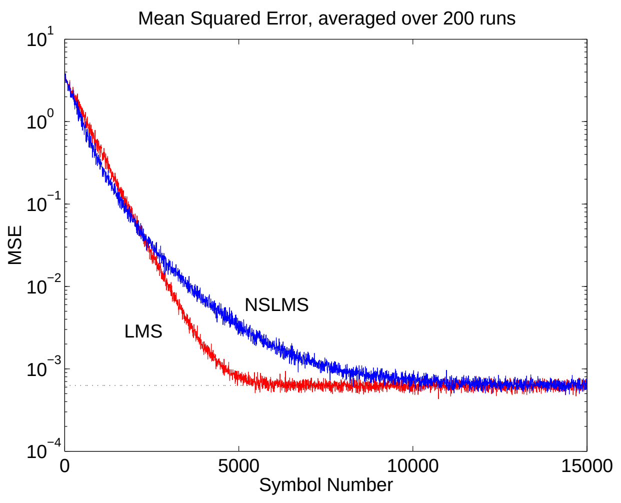

Figure 4: MSE vs. time for LMS and NSLMS, averaged over 200 random (non-sparse) channels.

to non-sparse channels in general. The initializations were all at the origin. The point of this plot is that for non-sparse channels, NSLMS suffers a slight performance loss compared to LMS. The reason NSLMS does not excel for non-sparse channels is that the stepsize is rationed out to many large taps, so each large tap only gets a small benefit. The small taps still move slowly, so the small increase in speed of the large taps is more or less matched by the small decrease in speed of the smaller taps. For a visual interpretation, imagine moving the circle in Figure 2 to $(0.5, 0.5)$ (which is a non-sparse system). Then the central diamond disappears entirely, so NSLMS will move slower than LMS along the path from the origin to $(0.5, 0.5)$.

From these results, it is evident that NSLMS can achieve performance gains over LMS in sparse environments. However, in non-sparse environments, NSLMS may perform worse than LMS.

## 6 Conclusions

We have introduced the normalized sparse LMS algorithm (NSLMS), which is obtained from LMS by modifying the stepsize to be a time-varying vector stepsize. We have shown analytically and via simulations that if the system is sparse and the initialization is at the origin, then NSLMS will converge to the optimal solution faster than LMS for the same asymptotic total MSE; and if the system is not sparse, NSLMS converges slower than LMS. The faster speed of NSLMS is achieved at the expense of an increase in complexity; notably, there is a division in each iteration of the update routine, which can be anathema for a hardware implementation.

## Acknowledgments

The authors would like to thank Bill Sethares (University of Wisconsin) and Bob Williamson (Australian National University) for their work on the precursor to this paper [8], and Jai Balakrishnan and Wonzoo Chung from the Cornell University Blind Equalization Research Group for their comments.

## References

[1] T. Aboulnasr and K. Mayyas, “Complexity Reduction of the NLMS Algorithm via Selective Coefficient Update,” *IEEE Transactions on Signal Processing*, vol. 47, no. 5, pp. 1421–1424, May 1999.

[2] S. Ariyavisitakul, N. R. Sollenberger, and L. J. Greenstein, “Tap-Selectable Decision-Feedback Equalization,” *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1497–1500, Dec. 1997.

[3] T. J. Endres, R. A. Casas, S. N. Hulyalkar, and C. H. Strolle, “On Sparse Equalization Using Mean-Square-Error and Constant Modulus Criteria,” in *The 34th Annual Conference on Information Sciences and Systems*, Princeton, NJ, 2000, vol. 1, pp. TA7b–7–12.

[4] R. K. Martin, “Exploiting Sparsity in Adaptive Filters,” M.S. thesis, Cornell University, 2001.

[5] J. Kivinen and M. K. Warmuth, “Exponentiated Gradient Versus Gradient Descent for Linear Predictors,” *Information and Computation*, vol. 132, no. 1, pp. 1–64, Jan. 1997.

[6] S. I. Hill and R. C. Williamson, “Convergence of Exponentiated Gradient Algorithms,” *IEEE Transactions on Signal Processing*, vol. 49, no. 6, pp. 1208–1215, June 2001.

[7] R. E. Mahony and R. C. Williamson, “Riemannian Structure of Some New Gradient Descent Learning Algorithms,” in *Adaptive Systems for Signal Processing, Communication and Control Symposium*, Lake Louise, Alberta, Canada, 2000, pp. 197–202.

[8] R. K. Martin, W. A. Sethares, R. C. Williamson, and C. R. Johnson, Jr., “Exploiting Sparsity in Adaptive Filters,” in *The 2001 Conference on Information Sciences and Systems*, Baltimore, MD, 2001.

[9] B. Widrow, J. McCool, M. G. Larimore, and C. R. Johnson, Jr., “Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter,” *Proceedings of the IEEE*, vol. 64, no. 8, pp. 1151–1162, Aug. 1976.