NOISE REDUCTION IN THE GAMMA-RAY LOG BY MEANS OF NONLINEAR FILTERING

By

LARRY J. PADEN

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1979

Master of Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
1980

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1991

NOISE REDUCTION IN THE GAMMA-RAY LOG BY MEANS OF NONLINEAR FILTERING

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

[illegible]

[illegible]

Dean of the Graduate College

1402224

PREFACE

In January 1983, I returned to OSU to begin formal work on the degree of Doctor of Philosophy. My personal goal was to take a number of courses which I knew from experience would be useful in the burgeoning information age. Luckily for me, since without it I might never have collected and produced enough material to write this dissertation, I became involved with the Oklahoma State University Consortium for Enhancement of Well Log Data via Signal Processing. (We call it the Well Log Project for brevity.) This wonderful group of companies gave us much needed contact with researchers from companies such as Amoco Production Company; Arco Oil and Gas Company; Cities Service Oil and Gas Corporation; Conoco; Dresser-Atlas Company; Exxon Production Research Company; Gearhart Industries, Inc.; Halliburton; International Business Machines; Mobil Research and Development Corporation; Phillips Petroleum Corporation; Seismograph Service Corporation; Sohio Petroleum Company; Texaco Corporation; and the Oklahoma State University Center for Energy Research. My graduate research assistantship was funded by this consortium, and the support is gratefully acknowledged.

Ironically, some experts, along the very helpful and extremely detailed discussions, also advised that everything possible had been done for the gamma-ray log, and that I should pursue a more fruitful avenue of research. Instead of the desired effect, this made me more determined to do something which, I hope, is useful in the field. After all, the history of technology is filled with ironies like Einstein working at a patent office during an era when serious suggestions were being made that patent offices be closed since everything possible had been invented. One thing for which I feel indebted to these researchers is that in spite of any doubts as to the fruitfulness of this field of my endeavor, they did their utmost to assist me in every way possible.

After reading a large amount of literature on gamma-ray logging, I knew that the issue of what to do about the Poisson noise inherent in radioactive decay is an important problem to investigate. The problem is that this noise is small in comparison with the uncertainties involved in physical logging, so by late spring 1983 I began using the synthetic logs described here and measuring the results in Monte Carlo simulations. This was the turning point because it provided me with a relatively objective figure of merit of a filter.

And, although I touch on the subject of what input parameters should be provided to the synthetic log generator, I have never changed them from that first spring day when the program ran. This I have purposely *not* experimented with for fear of coming up with an optimized log instead of an improved filter. The question of how the different synthetic log parameters affect filtering may provide another interesting topic of research if couched in slightly different terms.

The advent of the synthetic log brought with it some startling conclusions: Ordinary median filters increased the noise; recursive median filters improved the noise level more than did the optimal time-invariant linear filter; but my own best filter thus far did little in comparison. That filter has long since been confined to mass storage, but it did introduce me to a useful methodology in inventing filters. One indication of the pathological features of this filter appears to be that the histograms results of the Monte Carlo simulation are multimodal. By keeping a record of the input seeds to the random number subsequently be regenerated and examined manually for the features that cause unusually beneficial or pathological behavior. This may seem like it would produce an unwieldy quantity of output, but for any filter that improved the results, less than 10% of the histogram's data points fell outside the Gaussian-like center hump, and often the examination of only a few of these would be sufficient to conjecture what might be improved. Once the histogram began to appear Gaussian, the filters often began to produce reasonable results on actual log data, also. Perhaps the process could be repeated with the tails of the histograms, but this is left as a potentially interesting problem for future work.

In addition to acknowledging the helpful discussions with the various members of the OSU Well Log Consortium, I would especially like to thank Eric Pasternak of Arco for providing suitable gamma-ray log data on which to illustrate the results. To all the faculty members at OSU I wish to extend my heartfelt thanks since I cannot think of a single class in which the professor did not take time to explain some question that I had which might not even be particularly related to the subject. In particular, and not necessarily in any particular order, I wish to thank certain members of that faculty for their special help. Dr. James Rowland made computer simulations a very exciting part of all his classes, and serving on my committee. Dr. Gary Stewart of the Geology Department spent many hours defining salient features in various logs and ultimately examining the results of the more meritorious filters to see if they made gross blunders. Dr. Stewart also served on my committee. Dr. David Soldan initially got me started in the gamma-ray log and helped with in the numerous discussions that save many precious hours when learning a new subject and initially acting as my thesis advisor. Dr. Allen Steinhardt came up with challenging problems that were, at the same time, unusual, relevant, and interesting. He subsequently served as my thesis advisor before joining MIT Lincoln Labs (now at Cornell University) and is especially remembered for good mathematically correct and relevant answers to many of my badly posed questions.

My debt of gratitude to the chairman of my committee, Dr. Rao Yarlagadda can never be repaid. He has contributed in so many ways to my education that I doubt I can name them all. He instigated the OSU Wellog Consortium, which has been so important, taught numerous courses, answered many questions, provided many books, and provided continuity in the mist of the many changes my committee has undergone. Without his patience and persistent encouragement, this work might have died on the verge of completion.

In my personal life, I must thank my loving wife Carol for her patience with all days I spent away from home in pursuit of this project. My father, Jack T. Paden, Jr. has also provided much needed encouragement to overcome those many petty adversities that once seemed too onerous to endure. My grandfather, Jack T. Paden, Sr. always encouraged me by instilling the philosophy that the purpose of education is to enable one to find that employment which best suits that particular individual, in order that she or he may get the most out of life. (Or, as I have come to think of it: The only real success in life is to be able to do exactly what one wants.)

Last, but not least, I express my heartfelt thanks to the many, many other people, going as far back as my school days in Sand Springs, Oklahoma, in Stillwater, and other places, who have assisted me over the years both in obtaining the requisite education for this research, as well as in the research itself.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1. The Model of Gamma-Ray Logging 3
1.1.1. Poisson Noise 3
1.1.2. Factors Affecting the Recorded Signal 4
1.2. Linear Filters for Noise Removal 7
1.3. Nonlinear Filters for Noise Removal 8
1.3.1. Median and Recursive Median Filters 8
1.3.2. Generalized Median Filters 10
1.4. The Monte Carlo Method 12
1.4.1. Theoretical Basis 13
1.4.2. Other Uses for these Results 14
1.5. Survey of Dissertation 14
1.5.1. Summary of the Introduction 14
1.5.2. Outline of Dissertation 15

II. SYNTHETIC LOGS 17

2.1. Construction of Synthetic Logs 18
2.2. Measurement of Noise on a Linear Scale 22
2.3. An Unattainable Minimum Noise Level 23
2.4. The Wiener Filter Applied 25
2.5. Chapter Summary 26

III. MONTE CARLO SIMULATION 28

3.1. Results of Recursive Median Filtering 29
3.2. Linear Combinations of Recursive Median Filters 30
3.3. Optimal Weighting for RM Filters 31
3.4. Twin Window Filtering 33
3.5. Monte Carlo Simulation of the Twin Window Filters 36
3.5.1. Mean Values for Synthetic Log 36
3.5.2. Standard Deviation for Synthetic Log 38

Chapter Page

3.5.3. Results for Synthetic Log with Slopes.................39
3.6. Twin Window Filtering Applied to Actual Logs 41
3.7. Chapter Summary 41

IV. OPTIMIZATION OF THE TWIN WINDOW AVERAGE FILTER.............44

4.1. A Special Synthetic Log..44
4.2. Monte Carlo Simulation...47
4.2.1. Program Description 47
4.2.2. Checking the Simulation..................................49
4.2.3. Consolidation of Data 50
4.3. Results of the Improved Filter.......................................51
4.4. A Method for Using the Filters 52
4.5. Chapter Summary 54

V. CONCLUSIONS AND FUTURE WORK..56

5.1. Recursive Median Filters 57
5.2. Twin Window Filters..57
5.3. Future Work..58

BIBLIOGRAPHY 61

APPENDIX A - PROGRAM TO EVALUATE THE TWIN WINDOW AVERAGE FILTER...68

APPENDIX B - PROGRAM TO CONSOLIDATE THE DATA.........................85

APPENDIX C - RESULTS OF RUNNING THE TWIN WINDOW AVERAGE FILTER...94

APPENDIX D - HISTOGRAMS OF TWIN WINDOW AVERAGE FILTER FOLLOWED BY A RECURSIVE MEDIAN 3 FILTER...107

APPENDIX E - PROGRAM TO OPTIMIZE THE TWIN WINDOW AVERAGE FILTER...119

LIST OF TABLES

Table Page

1. The means and standard deviations of the histograms of RMS error of selected RM filters. 29

2. The average values and standard deviations of RMS errors of 1000 runs of combinations of RM filters.....................................33

3. Designation for seven different TW filters. 36

4. RMS errors for seven different TW filters for nine different values of the filter parameter c. 38

5. The standard deviations of the results of Monte Carlo simulation of the seven filters in Table 3 for selected filter parameters...41

6. Average RMS errors for six of the filters in Table 3.........................42

7. Standard deviations for the data in Table 6. 42

8. Number of files created for each bed length.....................................49

9. Optimal values of the TW filter parameter for selected signal levels. 51

LIST OF FIGURES

Figure Page

1. Model of gamma-ray logging. 6
2. One of the 1000 different ideal synthetic logs used here. On the average, each of these 2048-point logs has 273 beds 18
3. Part of the sample ideal log at an expanded scale. 19
4. The noisy synthetic log. 20
5. Part of the noisy log on the expanded scale. 21
6. The autocorrelation function of the ideal synthetic log. 26
7. Illustration of the operation of the twin window filter. 34
8. Histogram of the RMS difference in 1000 synthetic logs filtered with TWAF (c=3.15625) 38
9. Schlumberger gamma-ray log, the log of the same borehole processed by another tool and TW filters, and a composite log with differences shaded. 42
10. Most (97.6%) of the revised ideal log, showing the extra beds at V=50 with L=5. 45
11. The first 256 points (12.5%) of the revised ideal log, with the added beds of length 5. 46
12. The first 256 points (12.5%) of the revised noisy log, with the added beds of length 5. 46
13. The center curve is RMS error plotted against variation of the optimal curve. The two outside lines are ±1 standard deviation. The value 3.10 is nominally optimal. 51

NOMENCLATURE

$\int_a^b f(x)dx$	Definite integral of f(x) from a to b with respect to x
$\lvert z \rvert$	Absolute value of z
α	Parameter in Conway's GIR approximation
σ_x^2	Variance of the random variable x
μ	Linear attenuation coefficient of gamma-rays
μ_x	Mean of the random variable x
ω	Frequency
$\Phi(z)$	Geologic impulse response as a function of the depth, z
$\prod_{k=1}^{N} f(k)$	Product of f(k) over the range k=1, 2, 3, ..., N
$\sum_{k=1}^{N} f(k)$	Summation of f(k) over the range from k=1, ..., N
ASSP	Acoustics, Speech, and Signal Processing (an IEEE society) , now simply Signal Processing.
e	Mean square error
$E_1(x)$	Exponential integral of order 1, $\int_x^\infty \frac{e^{-t}}{t}dt$.

$H(\omega)$	Wiener filter as a function of frequency
IEEE	Institute of Electrical and Electronics Engineers
$K_0(x)$	Modified Bessel function of the second kind
$K_1(x)$	Modified Bessel function of the second kind
GIR	Geologic Impulse Response
O	Order of the run time of a program
R	Borehole radius
RMn	Recursive Median of length n
$S_{nn}(\omega)$	Noise power spectrum, as a function of frequency
$S_{sn}(\omega)$	Cross-spectrum of the ideal and noisy signals, as a function of frequency
$S_{ss}(\omega)$	Ideal signal power spectrum, as a function of frequency
SPWLA	Society of Professional Well Log Analysts
STM	Standard type M filter
TW	Twin Window
TWAF	Twin Window Average Filter
X_k	Sequence of input data points, as a function of k
Y_k	Sequence of output data points, as a function of k
z	Depth (domain of a well logging function)

COLOPHON

Normally, a colophon tells how a document was typeset, but since understanding the programs used to produce this is important, the definition is used in a very broad sense here. Various editions of this work have been on DEC Runoff and AT&T Troff, but it may be of interest that the final edition was produced on a Gateway 2000 with a Micronics 80486 motherboard and 12 MB of DRAM. The display was a NEC 3D™ and hardcopy on a TI MicroLaser™ with 1.5 MB of DRAM and the PostScript™ option. The compiler for the C programs was Borland's Turbo C++ Professional™, with C++ turned *on* for its superior type-checking capabilities. Word processing was done in the Microsoft Office™ which includes Word for Windows™, Excel™ for plots, and PowerPoint™ for slides. The type used is 14-point Times-Roman set at 24-point pitch which gives approximately the same character density as double-space Elite typewriter output with much improved readability. The selection of capitalized titles set in the same size type is enforced by the OSU Graduate College which also mandated the removal of bold and italics which were originally used in the style common to current engineering texts. Finally, the character spacing was adjusted by hand on about 3 lines per page to avoid hyphenated words.

CHAPTER I

INTRODUCTION

At least two major types of gamma-ray logging tools are widely used in the oil field. The first of these, which produces what is known as the gamma-gamma log, has a radioactive source as well as a transducer that converts gamma-rays into electrical impulses. The second, which produces the gamma-ray log, has only the transducer, and will be discussed here. It is often used to study geologic formations involving highly radioactive elements such as those containing Uranium, as well as to do ordinary oil field logging. This tool is also made with multiple transducers, each sensitive to gamma-rays of particular energy. Due to the penetrating nature of gamma-rays, these tools are usually made cylindrically symmetric. As the tool is pulled up a borehole, the naturally occurring radiation is converted to electrical pulses, which are integrated by a simple RC circuit known as the ratemeter, and the result then recorded for subsequent interpretation. This produces relatively large signals for beds of shale, which are more radioactive than beds of sand.

The underlying cause of the problem is that the needs of the oil field are much different than the needs of Uranium mining. In Uranium mining, boreholes are often drilled solely for exploratory purposes preparatory to mining. These holes are available for many hours for logging purposes, and consequently the logging may be done much more slowly than in oil fields. This need for rapid logging in oil fields, along with the fact that uranium ore and other ore mined for radioactive elements are many times more radioactive than common shale or sand, produces a log at least an order of magnitude less noisy than that found in oil field logs. On the other hand, in the oil field, the time spent logging the well may often come at the expense of idle drilling equipment, costing many tens of thousands of dollars per day. Many of the researchers working with gamma-ray logs come from the mining environment, so they have little motivation to reduce the greatly increased variance associated with the rapidly done oil field log.

At first it may seem that logging the well more slowly is a reasonable solution, but in the oil field are a number of tools producing useful information when run at a rapid pace. These include sonic tools, resistivity tools, induction tools, borehole televiewers and others. Consequently, the choice invariably made is to run the log rapidly and process the results numerically. The other important observation in this scenario is that the driving function of this system—the relative radioactivity of each geologic bed—cannot be accurately deduced from other logs. This lack of knowledge of the input to the system and the desire to estimate it more precisely leads to the construction of synthetic logs in order to be able to evaluate the results precisely.

To lay the framework for this work, a model of gamma-ray logging is developed which takes into account the previous efforts in this field. Next a method of synthetic log construction is proposed that is rich in the smaller beds which most interest geologists. Having thus ascertained the driving function of the system, the measurement of the error of the filtered signal is defined. Then, since the filters of interest are nonlinear and have no closed form equation representing their capability for noise reduction, the Monte Carlo method is advocated, along with applications of the relevant equations to other parts of this work. After that, an unattainable minimum noise level is derived below which no filter can achieve additional noise reduction. And last, an introductory summary is given, along with an outline of the remainder of this work.

1.1. The Model of Gamma-Ray Logging

The model of gamma-ray logging used is highly dependent on research in that field. The first reference to practical gamma-ray well logging was published by Howell and Frosch (1939) at the annual meeting of the board of directors of the Humble Oil & Refining Co. in Oklahoma City. They first used an ionization chamber, which led to the construction of a more rugged apparatus containing two Geiger counters. This instrument was about 12 feet long and 3.5 inches in diameter, similar in size to modern instruments which often make use of scintillation counters.

1.1.1. Poisson Noise

The radioactive decay which drives these detectors has associated with it a phenomenon known as Poisson noise. This can be derived mathematically, as in Haight (1967), which starts with the basic physics in Evans (1955). Haight includes other important facts about the Poisson distribution and an extensive bibliography. If the mean of the Poisson distribution is sufficiently large, it may be effectively modeled by a Gaussian distribution with its mean and variance set equal to the mean of the Poisson distribution. The error in this approximation is derived and is discussed extensively in §119 and §120 of Fry (1965), which gives the warning that the largest percentage error is in the tails of the distribution, a caveat that does not apply here. The noisy signal is then affected by interaction with a number of parameters such as the borehole diameter, the length of the

Geiger counter tubes, the absorption of gamma radiation in the drilling mud, and a number of other factors.

1.1.2. Factors Affecting the Recorded Signal

Scott and his colleagues (1961) mention borehole diameter, medium filling the borehole, borehole casing, water content of ore, and the nonuniform distribution of radioactive material within a layer and give standard conditions for U. S. Atomic Energy Commission logging. However, such standard conditions are virtually impossible to achieve in an oil well. Rhodes and Mott (1966) quantify the effects of such less-than-ideal factors in oil well logging. This is largely based on work done around 1961. They quantify the effects of the borehole diameter, mud density, casing and cement thickness, bed thickness, and detector eccentricity. This is done for a number of different gamma-ray energies. By using these differing characteristics, the log analyst may determine the interactions of these different effects. The capability to do this demonstrates the usefulness of the spectral gamma logs.

Mathematical investigation of the effects of the ratemeter, the length of the detector, the size of the borehole, and the absorption of the drilling mud was done by Czubek (1964, 1971, 1972). Davydov (1970) investigated the one-dimensional problem in gamma-ray logging. Czubek and Zorski (1976) present a method of accounting for the above effects as well as logging velocity, absorption in the rock, and rock porosity. They present tables of the mathematical coefficients required for practical application of the method.

From the standpoint of signal processing, this is all greatly simplified by Conaway and Killeen (1978). These researchers define the geologic impulse response (GIR) as the response produced by infinitesimally thin bed on a one-dimensional detector. If this is done under the ideal conditions of noise-free counts, and infinitesimal sample interval, then the GIR, Φ, as a function of depth z is

$$\Phi(z) = \frac{\alpha}{2} e^{-\alpha|z|}, \tag{1}$$

where the infinitesimal bed is assumed to be at z = 0 and α, a constant to determine the shape of the double exponential. Essentially, this equation is the one found in Davydov (1970), who refers Suppe and Khaikovich (1960). They go on to derive the inverse digital operator for the GIR as

$$\left[\frac{-1}{(\alpha\Delta z)^2},\ 1 + \frac{2}{(\alpha\Delta z)^2},\ \frac{-1}{(\alpha\Delta z)^2}\right]. \tag{2}$$

The issue of how to relate α to Czubek's work is further examined by Conaway (1980). He points out that Czubek's expression for the GIR may be written:

$$\Phi(\mu,z) = \frac{E_1\left(\mu\sqrt{R^2 + z^2}\right)}{2\mu R\left[K_1(\mu R) - \int_{\mu R}^{\infty} K_0(x)dx\right]}, \tag{3}$$

where μ is the linear attenuation coefficient, R is the borehole radius, $K_0(x)$ and $K_1(x)$ are modified Bessel functions of the second kind, and $E_1(x)$ is the exponential integral of order 1 defined by

$$E_1(x) = \int_x^{\infty} \frac{e^{-t}}{t} dt. \tag{4}$$

Both the exponential integral and the modified Bessel function of the second kind are tabulated in Abramowitz and Stegun (1964). Czubek (1971) suggested approximating the GIR with the double-sided exponential and gave an expression for obtaining α from the borehole radius R and linear attenuation coefficient μ,

$$\alpha = \frac{E_1(\mu R)}{\mu R\left[K_1(\mu R) - \int_{\mu R}^{\infty} K_0(x)dx\right]}. \tag{5}$$

Although Czubek suggests obtaining α from normalizing the two curves, Conaway (1980) uses what he terms the semilogarithmic slope method. It turns out that either of these methods is more than what is required for the logs discussed later.

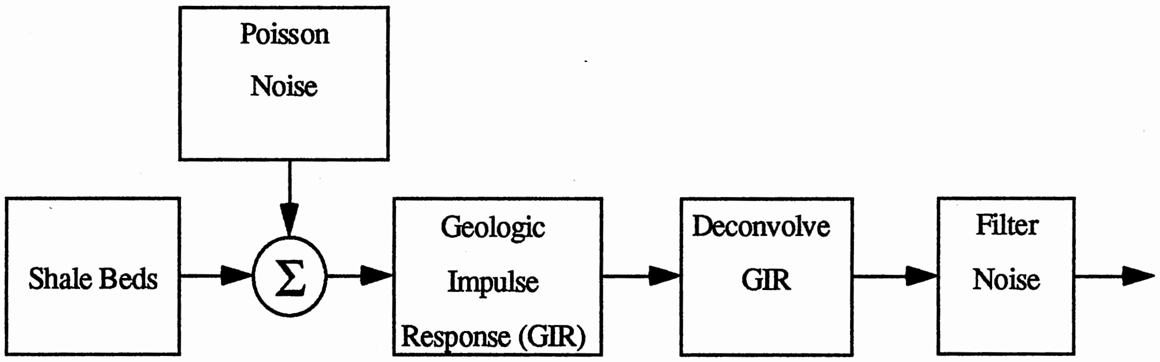


Figure 1. Model of gamma-ray logging.

Combining the various pieces gives a model for gamma-ray logging as illustrated in Figure 1. The shale beds generate the signal, which, by virtue of the discrete nature of the constant process of radioactive decay, has Poisson noise added to it. This, of course, neglects the extremely small second order effect that as decay occurs, a miniscule amount of material is transmuted to another element, thereby changing the rate. This combined signal then passes through the geologic features at various angles to strike the detector. Neglecting the second-order effects of the various beds having different attenuations of gamma-rays and assuming a two-dimensional problem as in the literature cited, implies convolution with a linear geologic impulse response (GIR). This may then be deconvolved by means of the three-point inverse function previously discussed and the noise subsequently filtered, giving the desired signal. The novel contribution of this thesis is in filtering Poisson noise added to a signal containing many discrete jumps, so techniques for filtering such noise that were found in existing literature will now be discussed.

1.2. Linear Filters for Noise Removal

Papoulis (1977) gives an account of the Wiener filter, which is the optimal linear filter for noise removal. If the signal and noise are uncorrelated, and the noisy log to be filtered is the sum of the signal and noise, then in terms of the ideal signal power spectrum $S_{ss}(\omega)$, the cross-spectrum of the ideal and noisy signals $S_{sx}(\omega)$, and the noise power spectrum $S_{nn}(\omega)$:

$$S_{sx}(\omega) = S_{ss}(\omega) \tag{6}$$

$$S_{xx}(\omega) = S_{ss}(\omega) + S_{nn}(\omega) \tag{7}$$

Then the Wiener filter is given by

$$H(\omega) = \frac{S_{ss}(\omega)}{S_{ss}(\omega) + S_{nn}(\omega)}, \tag{8}$$

and the resulting mean square error, e, is given by

$$e = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{S_{ss}(\omega) S_{nn}(\omega)}{S_{ss}(\omega) + S_{nn}(\omega)} d\omega. \tag{9}$$

This shows that once certain statistics of the signal and the noise are known, the noise level after filtering with the optimal linear filter may be obtained without actually implementing the filter. However, as will be shown later, linear filters are of limited use due to the sharp changes in the desired signal, so the literature concerning nonlinear filters is discussed next.

1.3. Nonlinear Filters for Noise Removal

Many different nonlinear filters are discussed in various widely different publications. The median filter is the most widely used of the nonlinear filters, so first it and the operationally similar recursive median filter (RM filter) are defined. But although the operations appear very similar, the results will later appear to be vastly different. After the median filter is presented, some generalized nonlinear filters will be presented.

1.3.1. Median and Recursive Median Filters

J. W. Tukey (1971) is the first to revive the median filter as a tool for time series analysis. Given the input sequence X_k and output sequence Y_k, the output of the median filter of length $W = 2N+1$ is defined as

$$Y_k = \text{median of } \{X_{k-N}, X_{k-N+1}, X_{k-N+2}, \ldots, X_{k-1}, X_k, X_{k+1}, \ldots, X_{k+N-2}, X_{k+N-1}, X_{k+N}\}. \tag{10}$$

The first and the last points of a noisy sequence are usually replicated before filtering to minimize end effects. The output of the recursive median filter is similarly defined as:

$$Y_k = \text{median of } \{Y_{k-N}, Y_{k-N+1}, Y_{k-N+2}, \ldots, Y_{k-1}, X_k, X_{k+1}, \ldots, X_{k+N-2}, X_{k+N-1}, X_{k+N}\}, \quad (11)$$

where the first N points of the median operator are now previously filtered points. These filters have been successfully applied in areas such as speech processing by Jayant (1976) and in digital image processing by Huang, et al. (1979). Huang suggests an algorithm for implementing a fast median filter, which is similar to that in Ataman, et al. (1980). This algorithm is shown by Bednar and Watt (1984) to reduce the computer time required from $O(L^{3/2})$ to O(L) where O( ) is the order as given by Aho, Hopcroft, and Ullman (1983) and L is the length of the median filter. The actual time saved is even greater than this seems to indicate, since every full sort routine must examine every element to be sorted, but the fast median algorithm needs, on the average, to examine only ½ of the elements.

Literature on the theory of median filters is less well-developed than that of linear filters. The typical techniques, such as superposition for linear systems, do *not* exist, making design difficult. However, knowledge of some of the published properties can greatly reduce the number of trials required to find useful filters. Some interesting signal structures and properties of median filters are given in Nodes and Gallagher (1982). These structures include:

1. A constant neighborhood is a region of at least N+1 consecutive points, all of which are identically valued.
2. An edge is a monotonically rising or falling set of points surrounded on both sides by constant neighborhoods.
3. An impulse is a set of N or less points whose values are different from the surrounding regions and whose surrounding regions are identically valued constant neighborhoods.

4. A root is a signal which is not modified by filtering.

Useful properties are:

1. Impulses are eliminated by both ordinary and recursive median filters.
2. Constant neighborhoods and edges are unperturbed.
3. Only signals composed solely of constant neighborhoods and edges are roots.
4. Any signal of length L is reduced to its root after at most $\frac{1}{2}(L-2)$ successive passes by any median filter.
5. A signal is invariant to recursive filtering if and only if it is invariant to standard filtering.
6. Any signal will be reduced to a root after one pass of a recursive median filter.

These properties are used extensively to make decisions regarding what nonlinear filters might reduce the noise level without significantly degrading the signal.

1.3.2. Generalized Median Filters

Other useful facts about nonlinear filters are given the paper by Peterson, et al. (1988). This work uses two broad generalized classes of median filters, called L filters, and standard type M filters (STM filters). As before if X_k is the input and Y_k is the output, then the L filter is defined as

$$Y_k = \sum_{j=1}^{W} A_j X_{(j)}^{k}. \qquad (12)$$

Here $X_{(j)}^{k}$ is the jth smallest sample from the W samples inside the window centered at k. A certain choice for the A_j coefficients yields an α-TM filter, given by

$$Y_k = \sum_{j=T+1}^{W-T} \frac{1}{2(N-T)+1} X_{(j)}^{k}, \tag{13}$$

where T is the largest integer which is less than or equal to αW, with α being constrained by $0 \leq \alpha \leq 0.5$. As in Bednar and Watt (1984), when α=0 the α-TM filter becomes the running mean filter; when α=0.5 it becomes the median filter. The output Y_k of an STM filter solves the equation

$$\sum_{j=k-N}^{k+N} \Psi(X_j - Y_k) = 0, \tag{14}$$

where

$$\Psi(x) = \begin{cases} 1, & x > p \\ x/p, & |x| \leq p \\ -1, & x < -p \end{cases} \tag{15}$$

with p some positive constant. This filter approaches the running mean filter as p approaches infinity or the median filter as p approaches 0. Peterson, et al. (1988) go on to derive a relation for the root mean square (RMS) error of the α-TM and STM filters, which are then graphed for W=5. This is done for signals containing a single edge, constructed as follows: The notable feature of these graphs is that

$$X_k = S_k + N_k \tag{16}$$

in which

$$S_k = \begin{cases} S, & k \le 0 \\ S+H, & k > 0 \end{cases} \tag{17}$$

and H is the height of a step at time k. The edge height of the filter and the RMS error are both normalized to the standard deviation of the zero-mean, Gaussian white noise. All the filters graphed in Figure 4 and Figure 7 in Peterson, et al. (1988), including the median, running mean, α-TM with T=1 and STM with p=1, 2, 3, 4, 5, and W=5 produce a greater RMS error after the operation for a normalized edge height of more than approximately 2.5. This means that the noise is greater in the output than the noise in the input. In terms of approximated Poisson noise, if the mean level is 100, the standard deviation is 10, so this represents a jump from 100 to 125, which is a relatively small change in the gamma-ray log. The typical gamma-ray log is filled with much larger changes, which as the graphs show, cause these filters even more problems. Although this does not rule out some other, probably heavily center-weighted combination of A_j coefficients for the L filter, it does eliminate all the nonlinear filters for which substantial published data is known.

1.4. The Monte Carlo Method

As can be seen in the derivation of Peterson's results, the algebra is extensive, and yields results which must be integrated numerically. In order to avoid such problems, especially when the results may not show improvement by lowering signal noise, a method of evaluation more dependent on machine computation is required. The method which is used is known as the Monte Carlo method after the famous gambling resort in Monaco. This method depends, as does the profit in such an establishment, in running the trials a large number of times, thereby reducing the variance of the simulated result.

1.4.1. Theoretical Basis

This result is important in a number of applications, as well as explaining a part of the practicality of the scientific method in general. The derivation given is due to Bendat and Piersol (1971). Given N uncorrelated random variables x_i, the expected value of the sample mean $\overline{x}$ is

$$E[\overline{x}] = E\left[\frac{1}{N}\sum_{i=1}^{N} x_i\right] = \frac{1}{N}E\left[\sum_{i=1}^{N} x_i\right] = \frac{1}{N}(N\mu_x) = \mu_x \tag{18}$$

$$E\left[(\overline{x}-\mu_x)^2\right] = E\left[\left(\frac{1}{N}\sum_{i=1}^{N} x_i-\mu_x\right)^2\right] = \frac{1}{N^2}E\left[\left(\sum_{i=1}^{N} x_i-\mu_x\right)^2\right] \tag{19}$$

Since the observations x_i are uncorrelated, the cross product terms in the last expression will have an expected value of zero. It then follows that

$$E\left[(\overline{x}-\mu_x)^2\right] = \frac{1}{N^2}E\left[\left(\sum_{i=1}^{N} x_i-\mu_x\right)^2\right] = \frac{1}{N^2}(N\sigma_x^2) = \frac{\sigma_x^2}{N} \tag{20}$$

In terms of a Monte Carlo simulation, this demonstrates that the average of many trials is an unbiased estimator of the mean of the random variable being simulated. Further, in N trials, the variance is reduced by $\frac{1}{N}$. Bendat and Piersol continue with the remark that this estimate of the mean is consistent and can be shown to be efficient.

1.4.2. Other Uses for these Results

These results have many uses. Aside from being the basis of Monte Carlo simulation, they assure that independent trials by different scientists or at different times will reduce the variance of the estimate of the mean of the distribution of the random variable under investigation. This is true regardless of the distribution, so long as the samples are uncorrelated. Later, after the synthetic log is defined, the result will be used again to show that if N samples from the same level are corrupted with white noise, that the estimate of the level has a variance reduced by $\frac{1}{N}$, when compared to the variance of the noise.

1.5. Survey of Dissertation

The material covered so far is the background for this dissertation. In order to give a clear exposition, this background is summarized, and the remainder of the dissertation described.

1.5.1. Summary of the Introduction

The basic method of gamma-ray logging was explained. The problems of logging in the oil field versus logging for rare earth mining have been discussed and provide the motivation for the balance of this work. The work done by Davydov, Czubek, Conaway, and others effectively gives the model of gamma-ray logging described. The true driving function of this model is the distinct beds of different types of sands and shales which each produce a characteristic signal level. These sharp boundaries produce a signal with unusually sharp changes in level. This signal is corrupted with the Poisson noise inherent in the process of radioactive decay. Deconvolution of the GIR can be done quickly with Conaway's method, leaving only the Poisson noise to corrupt the signal. These facts motivate the remainder of this dissertation.

1.5.2. Outline of Dissertation

Due to the probabilistic uncertainty associated with the true driving function of an actual gamma-ray log, a method of creating stochastic synthetic logs is given in Chapter II. A figure of merit is then presented, which when combined with the synthetic log and the Monte Carlo method, enables objective evaluation of novel filters. The same equations on which the Monte Carlo method is based are also used to calculate an unattainable lower bound for noise removal. The effects of different bed widths on the unattainable minimum noise level is then discussed. Chapter II closes with the evaluation of the prescribed figure of merit for the optimum stationary linear filter.

Within the framework for evaluation defined in the previous chapter, Chapter III evaluates the numerous filters. Median filters and α-trimmed mean filters are then evaluated by simulation, or the published literature. These filters are found to be worse on the average than no filter, at least for the synthetic log used. Recursive median filters are then shown in simulation to outperform the Wiener filter, and further reduction is achieved by the novel approach of simple (unity weighted) linear combinations of recursive median filters. Attention is then focused on the optimal linear combination of selected recursive median filters and it is shown that very little additional improvement is obtained.

Having exhausted all the traditional filters which are computationally simple to use, as well as the linear combination of recursive median filters, Chapter III continues by defining a novel class of filters named twin window filters. This method can be used with a number of more traditional filters as kernels, and these are shown in Monte Carlo simulation to achieve superior noise reduction. Cascading certain twin window filters with recursive median filters of length 3 achieves still better results. Attention is then focused on mathematical optimization of the twin window filter parameter with the running average as kernel. To produce this result, several different functions must be integrated numerically since the closed-form results are unknown. After avoiding many potential numerical traps involving loss of precision, the result is obtained. Unfortunately, this yields only a slight improvement in the figure of merit. Chapter IV presents a method of analysis for the Twin Window Average Filter (TWAF) and uses it to optimize the filter parameter with respect to signal level. The final chapter, Chapter V draws conclusions and makes suggestions for future work.

CHAPTER II

SYNTHETIC LOGS

Because the true input function—the shale beds in the model of gamma-ray logging given in the last chapter—is unknown, a method for constructing synthetic logs is developed. In order to use this to evaluate the effectiveness of the novel filters, a figure of merit is next devised. Before making use of these results in the next chapter, an unattainable minimum noise level is derived, below which no filter can reduce the noise. This and the residual noise level of the Wiener filter represent the greatest and least amount of noise reduction which filters of interest can achieve. The novel filters must do better than the relatively simple optimal stationary linear filter to be of interest. At the other end of the scale, the optimal nonstationary filter, the Kalman filter cannot do better than the relatively simply-derived unattainable minimum bound.

To compute the Kalman filter, full information regarding the method by which the ideal log is constructed must be used as input. This is, of course, absurd, since the synthetic log is not that closely modeled after actual logs. Also, the Kalman filter requires many more computations than any of the proposed filters, so even if it could reduce the noise to the unattainable minimum bound (which is impossible,) it is doubtful that it would be used when the novel filters presented here come relatively close. Note that in most of the proposed work linear filters are not as useful as the nonlinear filters for reasons to be discussed later. With the path to take thus carefully outlined, the construction of the synthetic log is now considered.

2.1. Construction of Synthetic Logs

Literature discussing the statistics of the distribution of shale, sand, and other geologic beds is unknown. Even if such data existed, it might not give sufficient correlation information between beds to enable construction of a realistic log. Therefore, in order to construct a random synthetic log, the desired properties are first enumerated, then a method proposed. From the viewpoint of the geologist, the salient features which must be preserved as much as possible are the bed boundaries, since these are often the basis for

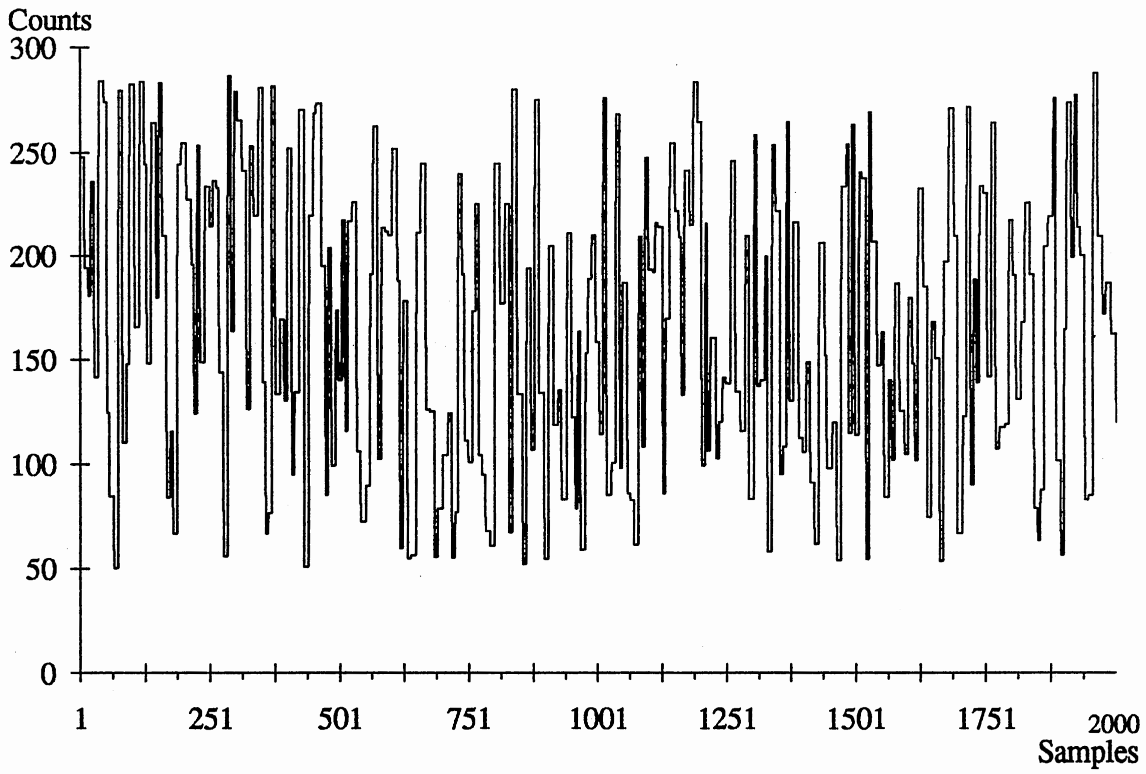


Figure 2. One of the 1000 different ideal synthetic logs used here. On the average, each of these 2048-point logs has 273 beds.

judgment on the best means of augmenting oil production.

Also, the differences in counts produced by adjacent beds are large; instantaneous changes by factors of two, three, or even more are not rare. To simulate these features, the ideal synthetic log is generated by taking the bed width to be an independent, uniformly distributed random variable between 5 and 10 samples in duration. The amplitude of each bed is also an independent, uniformly distributed random variable between 50 and 288 counts and is constant throughout the bed. Note that this is an idealized version. Every synthetic log in this dissertation contains 2048 samples which is equivalent to 1024 feet at 6 inches per sample. This is illustrated in Figure 2, but, for clarity, only an eighth of the log is shown in Figure 3.

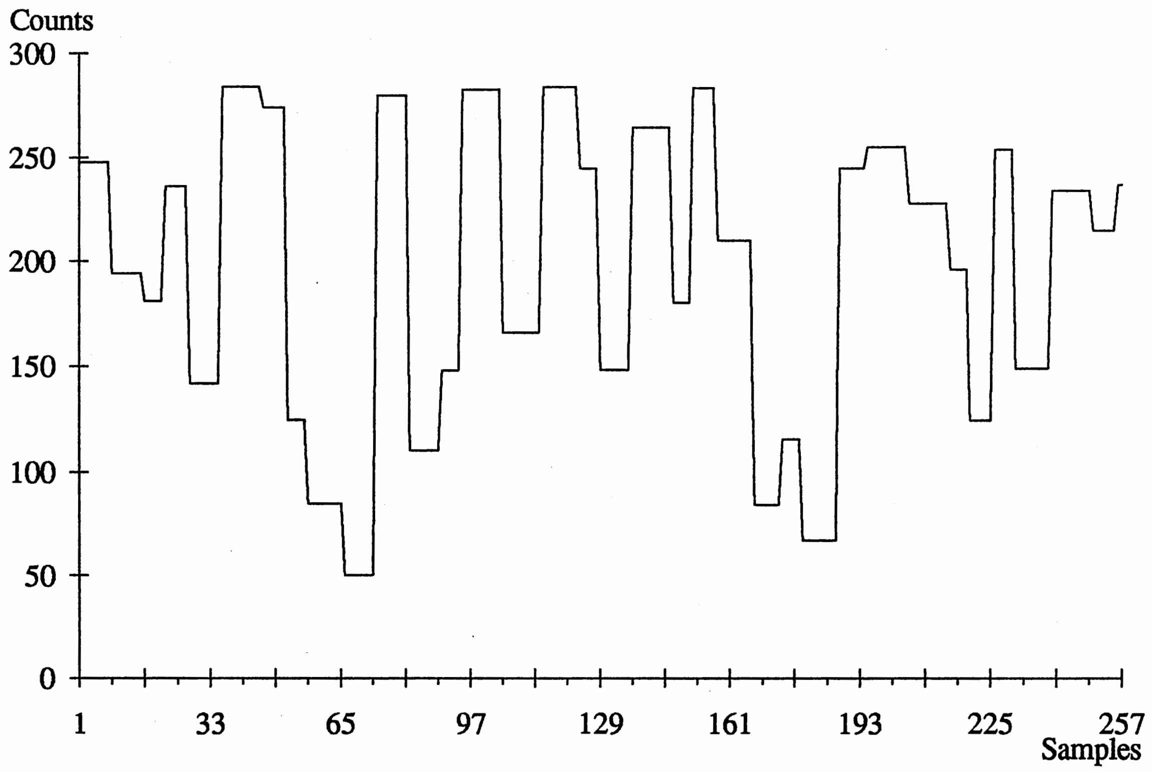


Figure 3. Part of the sample ideal log at an expanded scale.

Logs constructed in this manner were used to evaluate proposed novel filters. However, in actual logging, the bed breaks do not occur in such precise synchronization with the sampling. So to help ensure that this does not unduly influence the figure of merit of a filter, another synthetic log is constructed with the same parameters, but is moved back in time $\frac{1}{2}$ sample interval so that each and every pair of beds has a point between them in what is assumed to be the worst possible place: the average of the signal levels.

Whichever ideal log is used, the next step is the same. Simulated Poisson noise is added by using Gaussian noise with the variance set equal to the average signal level, as is illustrated in Figures 4 and 5. This is a reasonably good approximation for signals having a minimum level of 50 counts, as discussed in Haight (1967). In the present case, a large part of the error is exemplified by the possibility that the signal level of a bed will be reduced to a negative value, which is a physical impossibility. This

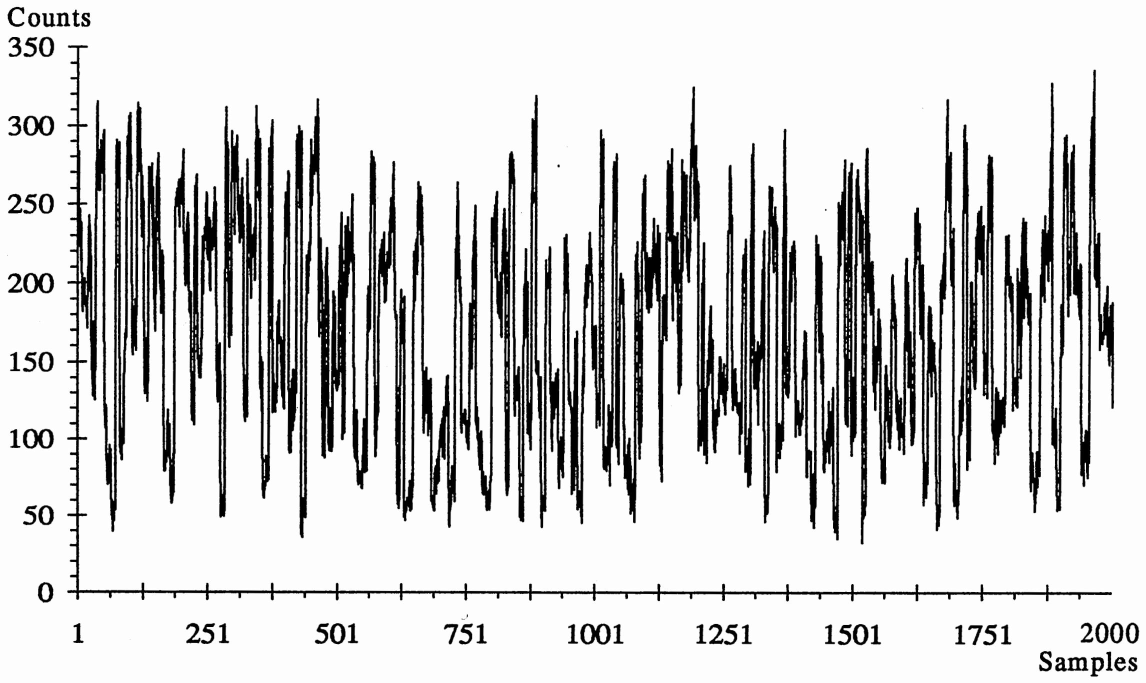


Figure 4. The noisy synthetic log.

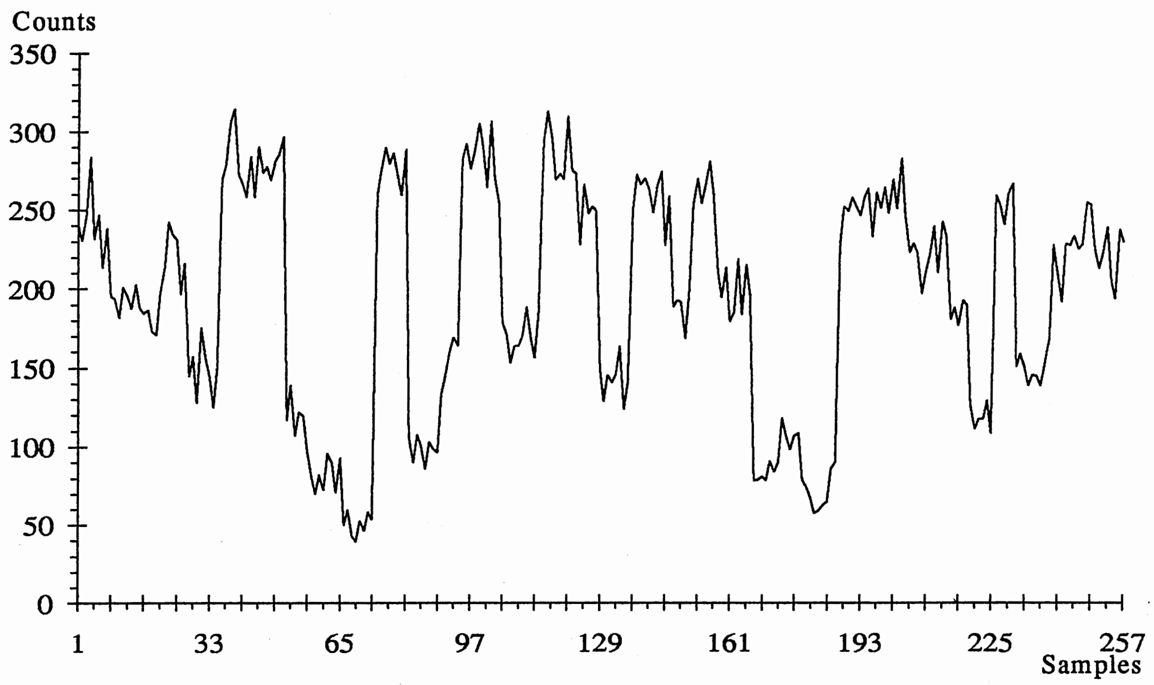


Figure 5. Part of the noisy log on an expanded scale.

possibility is extremely remote. In the worst case of a bed with an ideal signal of 50 counts, one standard deviation is $\sqrt{50} \approx 7.071$. So the impossible case occurs only when the noise is beyond 7 standard deviations, an exceedingly rare event.

Equation (20) shows that given N samples, each from a certain population with a standard deviation of σ_V, that the mean value of the samples is a random variable with a standard deviation of $\frac{\sigma_V}{\sqrt{N}}$. Moreover, this is true regardless of the distribution of the population from which the samples were drawn. To give a concrete example, if 1000 logs are processed by a certain filter, and the results have a mean of 9.00 and a standard deviation of 0.31, then the uncertainty associated with the estimate of the mean is $\frac{0.31}{\sqrt{1000}} \approx 0.01$. This assures us that, if the same experiment is performed a sufficiently large number of times, the results may be ascertained to any desired precision.

In the case of the gamma-ray logs, the Monte Carlo approach was used to obtain results sufficiently precise to distinguish between the various filters. More specifically, 1000 different noise sequences were each added separately to 1000 different synthetic logs to ensure that a particular pattern in a synthetic log would not give misleading results. All the logs were filtered with each of the proposed filters, and the mean and variance of each filter tabulated. Fortunately, the standard deviation associated with the best filters ranged from 0.27 to 0.32 and even the worst filter had a standard deviation of only 0.38. So the results given below are accurate ±0.03 in the worst cases. But before this principle can be used as assurance that the simulation is representative of the overall population of filtered logs, a figure of merit for noise reduction must be defined.

2.2. Measurement of Noise on a Linear Scale

One possible figure of merit is the signal-to-noise ratio (SNR) of the log. The noisy log constructed above has a SNR of 14.5 dB. To calculate this SNR, the expected value of the power of the zero-mean ideal log was divided by the expected value of the power of the noise. The logarithm of the result was then multiplied by ten. This is, of course, the typical way to figure the SNR and is certainly useful in comparing high quality signals in which the ratio might range over several decades.

Gamma-ray logging, however, does not produce the typical signal. It is strictly positive, often of short duration, and it will soon be shown that the SNR only ranges over a fraction of a single decade. Although the SNR is highly useful in many fields, the RMS error is more directly related to the visual noisiness in which we are interested, so it is the figure of merit used here. The RMS error of a filtered log, Y_k, with N points, whose ideal log is G_k, is defined by:

$$\text{RMS error} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[G_k - Y_k]^2} \tag{21}$$

The RMS error of the noisy log is 13.000 (22.2789 dB) which compares favorably with the simulated result of 12.995 (22.2755 dB).

2.3. An Unattainable Minimum Noise Level

Another use of Equation (20) is to compute a lower limit to which the RMS error can be reduced. Assuming that the bed boundaries are known precisely and because the log has a constant value within each bed, Equation (20) applies. To make use of this, first note that since the number of each of the six different sizes of layers is equal, the proportion of the total number of points contained in the wide layers is greater than that in the narrow beds. The fraction of the total number of points contained in each layer is proportional to the width of the layer. For instance, in the model used here, the bed widths are uniformly distributed between 5 and 10 samples, so $\frac{5}{45}$ of the total number of samples are contained in beds of width 5, $\frac{6}{45}$ of the samples are contained in beds of width 6, and so on through 10. However, Equation (20) assures us that the noise variance of each point in a bed of width n can be reduced by an average factor of $\frac{1}{n}$. In our example $\frac{5}{45}$ of the beds may achieve reduction by $\frac{1}{5}$. This continues for each bed width up to the example of $\frac{10}{45}$ of the beds may achieve reduction by $\frac{1}{10}$. For this particular example, the total factor by which the noise is reduced is

$$\frac{5}{45}\frac{1}{5} + \frac{6}{45}\frac{1}{6} + \frac{7}{45}\frac{1}{7} + \frac{8}{45}\frac{1}{8} + \frac{9}{45}\frac{1}{9} + \frac{10}{45}\frac{1}{10}.$$

In general, for uniformly distributed beds of constant amplitudes between n_1 and n_2 inclusive, the variance can be reduced by a factor of

$$\frac{\frac{n_1}{n_1} + \frac{n_1+1}{n_1+1} + \ldots + \frac{n_2-1}{n_2-1} + \frac{n_2}{n_2}}{n_1 + n_1+1 + \ldots + n_2-1 + n_2}. \tag{22}$$

Making use of the identity $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ and letting $n_3 = n_2+1$ gives

$$\frac{2(n_3-n_1)}{n_3(n_3-1)-n_1(n_1-1)} = \frac{2(n_3-n_1)}{n_3^2-n_3-n_1^2+n_1}$$

$$= \frac{2(n_3-n_1)}{(n_3^2-n_1^2)-(n_3-n_1)}$$

$$= \frac{2(n_3-n_1)}{(n_3+n_1)(n_3-n_1)-(n_3-n_1)}$$

$$= \frac{2(n_3-n_1)}{n_3+n_1-1}.$$

Finally, substituting for n_3 gives a noise reduction factor of:

$$\frac{\frac{n_1}{n_1} + \frac{n_1+1}{n_1+1} + \ldots + \frac{n_2-1}{n_2-1} + \frac{n_2}{n_2}}{n_1 + n_1+1 + \ldots + n_2-1 + n_2} = \frac{2}{n_2+n_1}, \tag{23}$$

which is the minimum, and unattainable, factor by which the variance can be reduced by any filter. Curiously, this happens to be exactly the same reduction that would be achieved in a log in which all beds are the same width: the average of the smallest and largest bed.

Returning to our particular example of $n_1 = 5$ and $n_2 = 10$ and making use of the fact that the original noise variance is 169 gives a minimum unattainable noise variance of

$$169\frac{2}{n_2+n_1} = 25.53. \tag{24}$$

This is equivalent to an RMS error of 4.75, or 36.5% of the original noise. This minimum value of noise can only be attained, on the average, when the bed boundaries are known. If they are not known, then some beds may be close enough together to allow the noise of one bed to raise its amplitude to that of the other bed. Therefore, the minimum value of residual noise represented by Equation (23) cannot be attained consistently and no filter of any type can be expected to achieve this level of noise reduction. Now that we have a bound on the maximum noise reduction, let us consider what can be achieved by the optimum linear filter, thereby establishing a benchmark that nonlinear filters must exceed to compensate for their added complexity.

2.4. The Wiener Filter Applied

As seen from Equation (9), the power spectral density is required to compute the residual error in the Wiener filter. To compute this, note that 45 different types of points exist in the synthetic log, each with equal probability. This makes it apparent that the autocorrelation function $R_{GG}(x)$ starts out dropping by $\frac{6}{45}$ until the points of beds of width 5 are exhausted, then drops by $\frac{5}{45}$ with each successive drop likewise decreasing. This is shown in Figure 6. This function was then transformed and numerically integrated:

$$e = \frac{169}{2\pi}\int_{-\infty}^{\infty}\frac{S_{GG}(\omega)S_{nn}(\omega)}{S_{GG}(\omega) + S_{nn}(\omega)}d\omega = 136. \tag{25}$$

So the mean square error of the residual noise after Wiener filtering is 136. This means that the optimum linear filter (which may not even be practical to implement) can only reduce the RMS error from 13 to 11.7, leaving 90%

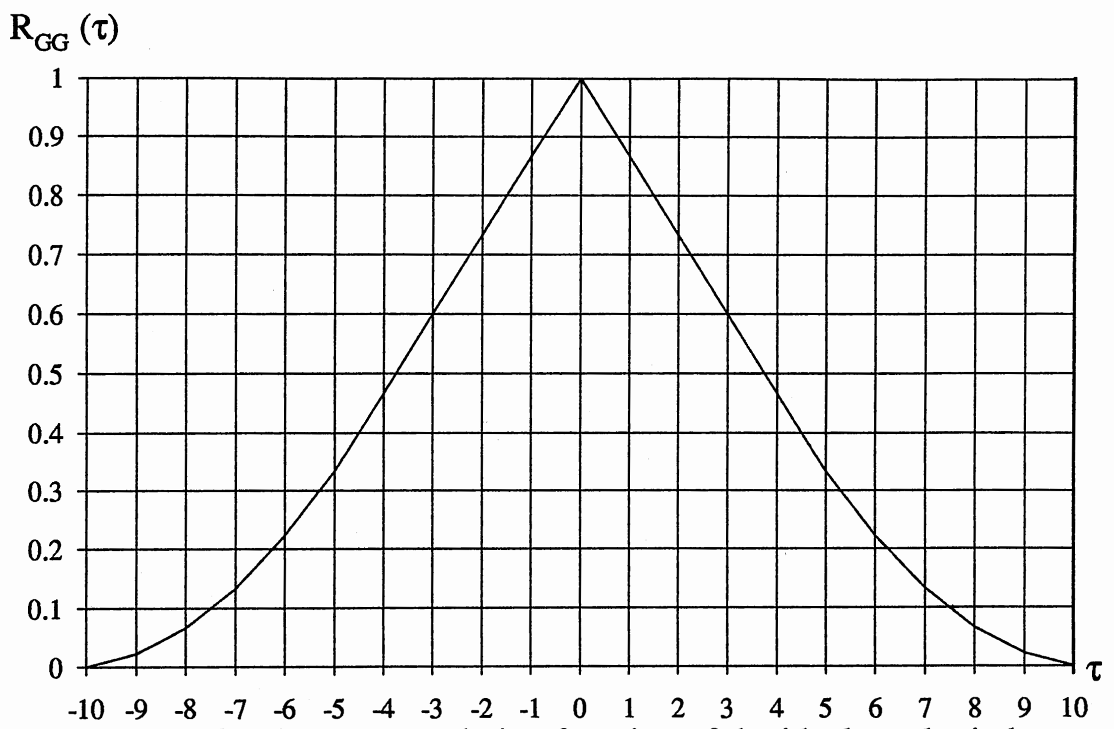


Figure 6. The autocorrelation function of the ideal synthetic log.

of the noise. The fact that this is still a large proportion of the original noise shows that nonlinear filters are attractive alternatives. While the computation of the residual error of the Wiener filter is comparatively simple, involving but one numerical integration, the corresponding calculations for the nonlinear filters to be discussed is many times more cumbersome, so the Monte Carlo method is used. Before examining these simulations, the chapter summary is given.

2.5. Chapter Summary

Due to the extreme difficulty of determining the true driving function of an actual gamma-ray log, a method of constructing synthetic logs was given. This method is readily applicable to a digital computer and can be used to generate an arbitrary number of ideal synthetic logs for study. The

RMS error was selected as the figure of merit for computer simulation, and its relative merit over measuring noise in decibels was discussed. This figure of merit was then used to quantify the unattainable minimum noise level, beyond which no filter of any type can, on the average reduce the noise in the synthetic log. At the other end of the scale, the Wiener filter quantifies the noise reduction expected from the optimum stationary linear filter. These two bounds are important since they represent the range of interesting novel filters. If a new filter cannot do better than the Wiener filter, it might be just as well to use a linear filter, while it is utterly futile to search for a filter to attempt to achieve an unattainable goal. With these results clearly in hand, Monte Carlo simulation will now be used to establish performance against these established benchmarks.

CHAPTER III

MONTE CARLO SIMULATION

Given the two important bounds of the previous section; and the fact that the true driving function of the system is known only for a synthetic log, at least for the immediate future; and the fact that an objective figure of merit is needed to access the results of the filtering in an unbiased measure; leads to the use of Monte Carlo Simulation. When this was applied to ordinary median filters, it was discovered that the noise increased, at least on this particular log with this particular type of noise. Therefore, these filters will not be discussed further. The filters that do show promise are the closely-related recursive median filters. These are evaluated, along with a filter comprised of a linear combination of RM filters. This brings up the question as to the best weighting for the linear combination, but multiple linear regression is used to show that optimal weighting will not produce much more than marginally better results, at least for this particular log and this particular type of noise. A novel filter, the Twin Window Filter is then introduced, and further simulation reveals that it performs the best thus far. This filter is then applied to actual data to show that, in addition to working well as measured by the chosen figure of merit, it also appear to visually improve actual data. Each of the nonlinear filters which lowered the RMS error of the synthetic log will now be defined and the significant results of the Monte Carlo simulations described.

3.1. Results of Recursive Median Filtering

One thousand noisy synthetic logs, each of 2048 points, were each filtered with RM filters of odd lengths up to 13. The mean and standard deviation of each resulting histogram is given in Table 1. For convenience in notation, an RM filter of length 1 is defined as the original, unfiltered data. As the window length of the RM filter increases, its performance deteriorates due to the inclusion of an increasing proportion of data points in the window that are not highly correlated with the data point being estimated. Although the results are significantly better than the optimal linear filter, they are not very close to the minimum noise level (36.5%) derived above. Even though this minimum is unattainable, we continue the search for a superior nonlinear filter by testing linear combinations of RM filters.

TABLE 1

THE MEANS AND STANDARD DEVIATIONS OF THE HISTOGRAMS OF RMS ERROR OF SELECTED RM FILTERS.

Length (Samples)	Average (Counts)	Percent of Original RMS Error	Standard Deviation (Counts)
1	13.00	100%	.27
3	9.54	73%	.28
5	10.04	77%	.37
7	11.79	91%	.45
9	13.59	104%	.51
11	23.43	180%	2.22
13	30.99	238%	2.53

3.2. Linear Combinations of Recursive Median Filters

Because median filters of different lengths pass different value through at the same point, an investigation of linear combinations of RM filters was undertaken. This method can take advantage of some of the noise suppressing benefits of moving average filters while retaining the full advantage of preserving sharp edges in the data. To describe this let Y'(k) be the final result of a linear combination of RM filtered data $Y_3(k)$, $Y_5(k)$, and $Y_7(k)$. That is

$$Y'(k) = \frac{a_1Y_3(k)+a_2Y_5(k)+a_3Y_7(k)}{a_1+a_2+a_3} \tag{26}$$

where the a_n's are the weighting parameters. If computer time is crucial in an application, it is good for the a_n's to be small and add up to a power of 2 for scaling. Another program was constructed to evaluate selected linear combinations (small integral weights) of the most promising lengths of 3, 5, and 7. It showed that some improvement could be made over the simple RM filter. Due to the results of previous experiments with RM filters, the original data was added as yet another term in the linear combination. This produces a reduction of RMS error by another 4% for $a_1 = a_2 = a_3 = 1$ in the 1000 log simulation.

If noncausal filtering is allowed, the data can be fed in reverse through the RM filters. If a RM filter of length 3 in reverse is denoted by $Y_{3r}(k)$, then

$$Y'(k) = \frac{a_1Y_1(k)+a_2Y_3(k)+a_3Y_{3r}(k)+a_4Y_5(k)+a_5Y_{5r}(k)}{a_1+a_2+a_3+a_4+a_5} \tag{27}$$

where $Y_1(k)$ is the unfiltered noisy file and again the a_n's are the weighting parameters. This combined forward/reverse technique offers additional noise suppression. In a Monte Carlo simulation of 1000 logs, it produces an average improvement of 3% for $a_n=1$.

3.3. Optimal Weighting for RM Filters

It is difficult to derive the best linear combination of RM filters on a theoretical basis. To calculate a lower bound on the performance of a selected combination of RM filters, we use multiple linear regression, as in Devore (1982). This procedure fits, by the least squares method, a linear equation of the form:

$$G(k) = b_1+b_2Y_3(k)+b_3Y_5(k)+b_4Y_7(k) \tag{28}$$

where G(k) is the ideal log, $Y_n(k)$ is a filtered sequence and b_i are the regression coefficients to be determined by solving the following system of equations:

$$\begin{bmatrix} \sum_{k=1}^{N} 1 & \sum_{k=1}^{N} Y_3(k) & \sum_{k=1}^{N} Y_5(k) & \sum_{k=1}^{N} Y_7(k) \\ \sum_{k=1}^{N} Y_3(k) & \sum_{k=1}^{N} Y_3(k)Y_3(k) & \sum_{k=1}^{N} Y_5(k)Y_3(k) & \sum_{k=1}^{N} Y_7(k)Y_3(k) \\ \sum_{k=1}^{N} Y_5(k) & \sum_{k=1}^{N} Y_3(k)Y_5(k) & \sum_{k=1}^{N} Y_5(k)Y_5(k) & \sum_{k=1}^{N} Y_7(k)Y_5(k) \\ \sum_{k=1}^{N} Y_7(k) & \sum_{k=1}^{N} Y_3(k)Y_7(k) & \sum_{k=1}^{N} Y_5(k)Y_7(k) & \sum_{k=1}^{N} Y_7(k)Y_7(k) \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} =$$

$$= \begin{bmatrix} \sum_{k=1}^{N} G(k) \\ \sum_{k=1}^{N} Y_3(k)G(k) \\ \sum_{k=1}^{N} Y_5(k)G(k) \\ \sum_{k=1}^{N} Y_7(k)G(k) \end{bmatrix} \tag{29}$$

Of course, since this requires the ideal log as input, it cannot serve as a practical filter. It is used only to compute a bound for a particular collection of RM filters. This can be expanded to any number of independent variables, so long as the number of data points, N, is sufficiently large. This was done with the noisy file and the four most promising filters, those of lengths 3, 5, 3r, and 5r. For the usual thousand files, multiple linear regression achieves a result of 8.16 RMS which is only 62.8% of the original noise. No linear combination can produce a better result than this, and since the weights were allowed to be different for each of the thousand files, a fixed combination will not do as well. Setting the weights to 1 gives:

$$Y'(k) = \frac{Y_1(k)+Y_5(k)+Y_7(k)+Y_9(k)}{4} \tag{30}$$

$$Y'(k) = \frac{Y_{1r}(k)+Y_{5r}(k)+Y_{7r}(k)+Y_{9r}(k)}{4} \tag{31}$$

$$Y'(k) = \frac{Y_1(k)+Y_7(k)+Y_{7r}(k)+Y_9(k)+Y_{9r}(k)}{5} \tag{32}$$

The results for these linear combinations are summarized in Table 2. Note that the spread of the histograms is reduced along with the reduction of the average value. Note that the spread of the histograms is reduced along with the reduction of the average value. However promising this result appears, a class of filters does exist which consistently does better than this unattainable minimum for linear combinations of RM filters of 8.16 RMS which is only 62.8% of the original 13 RMS error. For this new class, we have chosen the name twin window (TW) filters.

TABLE 2

THE AVERAGE VALUES AND STANDARD DEVIATIONS OF RMS ERRORS OF 1000 RUNS OF COMBINATIONS OF RM FILTERS.

Average of RMS errors	Percent of Original RMS Error	Standard Deviation	Description
13.00	100%	0.278	The original noisy log.
9.54	73%	0.284	RM filter of length 3.
9.091	70%	0.276	Linear combination of Equation (30).
9.087	70%	0.279	Linear combination of Equation (31).
8.64	66%	0.248	Linear combination of Equation (32).
8.16	63%	0.266	Multiple linear regression.

3.4. Twin Window Filtering

The advantages of the twin window filter are that it produces significantly better results on the synthetic logs than the various RM filters discussed, and it can be implemented in a fast method similar to that in Huang (1979), Ataman (1980) and Bednar (1984). Also, the computer time required is less than that required by the averaged recursive median filters. Similar filters are given by Pomalaza-Raez (1984) and Schultz (1981). The outer window (of length nine throughout this paper) moves along one point at a time, centered about the point to be estimated. The points in this window are sorted, which may not be necessary for computation, but is a convenient mechanism to portray the operation. Next, an inner window is formed by including the points which lie a certain number of standard deviations above and below the point being estimated. The number of standard deviations in half of the inner window defines the filter parameter c. As an example, Figure 7 illustrates how the TW filter operates.

... [19 20 15 12 **50** 45 49 68 93] ...
... [12 15 19 20 45 49 **50** 68 93] ...
... [12 15 19 20 (45 49 **50** 68) 93] ...

$$\frac{49+50}{2} = \mathbf{49.5}$$

Figure 7. Illustration of the operation of the twin window filter.

The first line in Figure 7 represents the original data, a sequence of three noisy steps in the middle of other data,. with the point to be estimated having the value 50, shown in boldface. The second line shows the data sorted. In the third line, the inner window is represented by the parenthesis. This window uses the filter parameter c = 3, and thus includes all points within $50 \pm c\sqrt{50}$, or approximately between 29 and 71. If, for example's sake, the ordinary twin window is used, which applies a median filter to the inner window, then because the window is already sorted and has an even number of points, the average of the two innermost points is used as the estimate. Note that the point being estimated, 50 in this case, may not be in the center of the window, as happens here. Instead of the median being applied to the inner window a variety of other methods may be applied to the inner window to obtain the estimate of the selected point.

Methods tried so far include the simple, unweighted average; the median; maximum likelihood; a method which allows the internal window to move based on the latest estimate of the selected point; and each of the first three followed by a three point RM filter. The RM filter can improve the RMS error by removing spikes caused by noise which exceed the bounds of the inner window. Continuing this example for each method, an unweighted average of the points in the inner window yields $\frac{45+49+50+68}{4} = 53.0$. For the median, since the window is an even number of points, the average of the two center points is used, giving $\frac{49+50}{2}$. For maximum likelihood, Equation (33) is used and gives

$$\frac{\sqrt{1+\frac{4}{k}\sum_{i=1}^{k} x_i^2} - 1}{2} = 53.2. \qquad (33)$$

The possibility also exists that a 3σ window centered around the newly formed estimate would add or omit one or more points compared with those in the original window. We will refer to this as a moving internal window. In this example, none of the estimates cause a point to be added to or deleted from the internal window, so no movement of the window would occur. If the internal window does change, the estimate calculated from the new window is used. It will be shown in Monte Carlo simulation that a moving internal window does not, in itself, reduce the RMS error. However, it does allow similar results using smaller internal windows as evidenced in the Monte Carlo simulation. This may prove to be useful in processing certain types of data. For instance, in the section which follows on the optimization of the twin window filter with respect to the filter parameter, it is found that larger values of c work better at low signal levels. So for a log of lower overall level, this filter should be more beneficial. Such investigation, however, would require altering the synthetic log, and so is considered beyond the scope of this work, therefore we return to the original log and simulation of the various twin window filters.

3.5. Monte Carlo Simulation of the Twin Window Filters

As was done with the RM filters, 1000 different 2048-point noisy synthetic logs were constructed and filtered. In order to express these results in uncluttered tables, the designations in Table 3 will be used.

TABLE 3

DESIGNATIONS FOR SEVEN DIFFERENT TW FILTERS.

Designation	Type of Filter
A	TW using average.
B	With maximum likelihood (ML).
C	Median.
D	Moving internal window.
E	Average followed by 3-point RM.
F	ML followed by 3-point RM.
G	Median followed by 3-point RM.

3.5.1. Mean Values for Synthetic Log

For the seven cases of TW filters described, the average RMS error for various filter parameters is given in Table 4. In these tables, "low" is the minimum value; "loc" is its filter parameter. The range of 7.49 to 6.87 is equivalent to 57.6% to 52.8% of the original noise. The fact that these are smooth, broad curves shows that selection of the filter parameter is not critical to filter performance. Further evidence that these filters are well-behaved is given in the histograms and examples in the Appendices of Paden and Steinhardt (1984). This work is further expanded in Paden (1985), and histograms are given in the Appendix C and Appendix D. An example histogram is shown in Figure 8. The filename, `D3_15`, in this figure is the result of the truncation of $3\frac{5}{32}$, which is the same filter parameter which gave the best result for twin window average filters in Table 4 where it was rounded to 3.16. The Gaussian curve in the figure is the curve implied by estimates of the mean and variance of the data in the histogram. Results based on only a single synthetic log indicate that another 2% reduction in noise may be gained by using averages of different filters as discussed under linear combinations of RM filters.

TABLE 4

RMS ERRORS FOR SEVEN DIFFERENT TW FILTERS FOR NINE DIFFERENT VALUES OF THE FILTER PARAMETER C.

c	A	B	C	D	E	F	G
2.00	8.76	8.82	9.02	8.06	7.42	7.48	7.88
2.25	8.17	8.23	8.47	7.60	7.13	7.18	7.61
2.50	7.74	7.79	8.05	7.34	6.94	6.99	7.42
2.75	7.46	7.50	7.77	7.27	6.87	6.91	7.31
3.00	7.32	7.35	7.59	7.36	6.90	6.93	7.26
3.25	7.31	7.33	7.51	7.56	7.02	7.03	7.27
3.50	7.41	7.41	7.49		7.20	7.20	7.30
3.75	7.59	7.58	7.51		7.45	7.44	7.37
4.00	7.84	7.82	7.57		7.75	7.72	7.46
low	7.30	7.33	7.49		6.86	6.91	7.26
loc	3.16	3.19	3.47		2.81	2.81	3.06

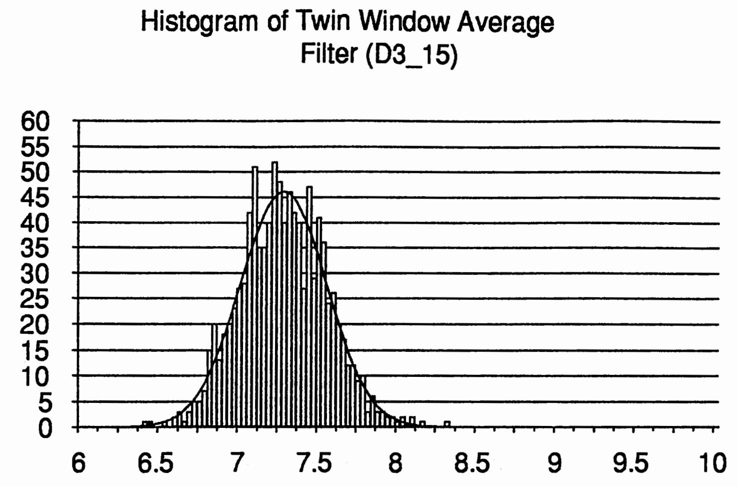


Figure 8. Histogram of the RMS difference in 1000 synthetic logs filtered with TWAF (c=3.15625).

3.5.2. Standard Deviation for Synthetic Log

Another measure the reliability of TW filters with regard to filtering the Poisson noise is the standard deviation of the RMS error measurements from the Monte Carlo simulation. This data is in Table 5. No filter produces much more spread in the error than that of the original data (0.267), and the spread associated with the lowest RMS averages is generally below that average spread. This is fortunate because it implies that the best filters also are the most dependable. As clearly seen in Table 4, the best filters are the TW using either the average or the maximum likelihood, followed by a three-point recursive median.

3.5.3. Results for Synthetic Log with Slopes

Before giving an example of an actual log filtered with a TW filter, let us consider one of the drawbacks of the above simulation. The original problem was that linear filters do not do well on logs with sharp corners. Consequently, the synthetic log was created to emphasize this problem. If another rule is used to construct the synthetic logs used in the simulations, substantially different results might be obtained. One very obvious problem with the log constructed of discrete levels (Figure 3) is that it implies that the bed boundaries always break synchronously with the sampling. In order to simulate this problem, the synthetic log was reconstructed with one point added between each bed, precisely half way between layers. The average RMS errors for 1000 of these logs are given in Table 6. Although this is higher that the corresponding table for the first synthetic log, it is comparable to the best RM filters. The spread of the data is given in Table 7, and while it shows somewhat less dependable filters, they should still be quite useful. The best of these

TABLE 5

THE STANDARD DEVIATIONS OF THE RESULTS OF MONTE CARLO SIMULATION OF THE SEVEN FILTERS GIVEN IN TABLE 3 FOR SELECTED FILTER PARAMETERS.

c	A	B	C	D	E	F	G
2.00	.300	.300	.310	.332	.303	.305	.318
2.25	.290	.290	.303	.322	.296	.298	.311
2.50	.284	.285	.296	.313	.292	.294	.304
2.75	.274	.275	.288	.307	.288	.290	.301
3.00	.268	.268	.281	.302	.285	.286	.297
3.25	.270	.271	.277	.312	.286	.286	.291
3.50	.274	.274	.274		.290	.291	.288
3.75	.281	.281	.274		.298	.297	.286
4.00	.295	.295	.278		.310	.309	.290

filters are applied to real gamma-ray logs in the next section to further illustrate their usefulness.

TABLE 6

AVERAGE RMS ERRORS FOR SIX OF THE FILTERS IN TABLE 3.

c	A	B	C	E	F	G
2.00	9.73	9.78	10.0	8.55	8.60	8.99
2.25	9.37	9.42	9.71	8.46	8.50	8.96
2.50	9.15	9.19	9.56	8.47	8.50	9.02
2.75	9.06	9.09	9.53	8.56	8.59	9.15
3.00	9.08	9.10	9.61	8.72	8.74	9.35
3.25	9.19	9.20	9.78	8.94	8.95	9.58
3.50	9.36	9.36	9.99	9.19	9.19	9.85
3.75	9.59	9.58	10.24	9.47	9.46	10.06
4.00	9.85	9.83	10.50	9.78	9.75	10.42
Low	9.05	9.08	9.53	8.45	8.49	8.96
Loc	2.84	2.84	2.72	2.34	2.38	2.19

TABLE 7

STANDARD DEVIATIONS FOR THE DATA IN TABLE 6.

c	A	B	C	E	F	G
2.00	.286	.287	.293	.288	.291	.299
2.25	.286	.288	.295	.287	.290	.302
2.50	.286	.286	.297	.287	.289	.302
2.75	.292	.293	.307	.296	.298	.312
3.00	.293	.293	.316	.299	.300	.322
3.25	.300	.301	.329	.306	.307	.334
3.50	.309	.310	.343	.316	.317	.347
3.75	.317	.317	.354	.323	.323	.356
4.00	.330	.330	.376	.337	.336	.379

3.6. Twin Window Filtering Applied to Actual Logs

In order to evaluate the results of filtering properly, two tools were run simultaneously in the same hole. One was processed through Schlumberger's filtering and is a conventional gamma-ray data curve; the other was not filtered and represents raw data. The raw data was then deconvolved using α=0.10 and for ease of comparison, scaled by a small constant to give a good match with the Schlumberger data. This was then filtered by a twin window filter using the average of the internal window. The filter parameter was 2.50 and the length of the outer window fixed at 9 points. The results are shown in Figure 9. The RMS difference between the filtered log and the Schlumberger log is only 7.23 and occurs primarily at the peaks and valleys. This is even slightly less than the RMS difference between the ideal and noisy logs with the same filter (7.30). Since visual inspection shows that this method of filtering produces sharper bed boundaries, substantiating the results of the Monte Carlo simulation, it indicates that the twin window filter is superior.

3.7. Chapter Summary

Monte Carlo simulation was applied to evaluate the results of the various nonlinear filters discussed. First, it was discovered that on this particular type of synthetic log that ordinary median filters do not improve the RMS error. However, recursive median filters of lengths 3, 5, and 7 do show varying degrees of improvement with respect to this figure of merit. Linear combinations of these recursive median filters were shown to decrease the error even more. In the quest for the perfect weighting for the linear combination, multiple linear regression was used to show that the optimal weighting produces only slight improvement over uniform weighting

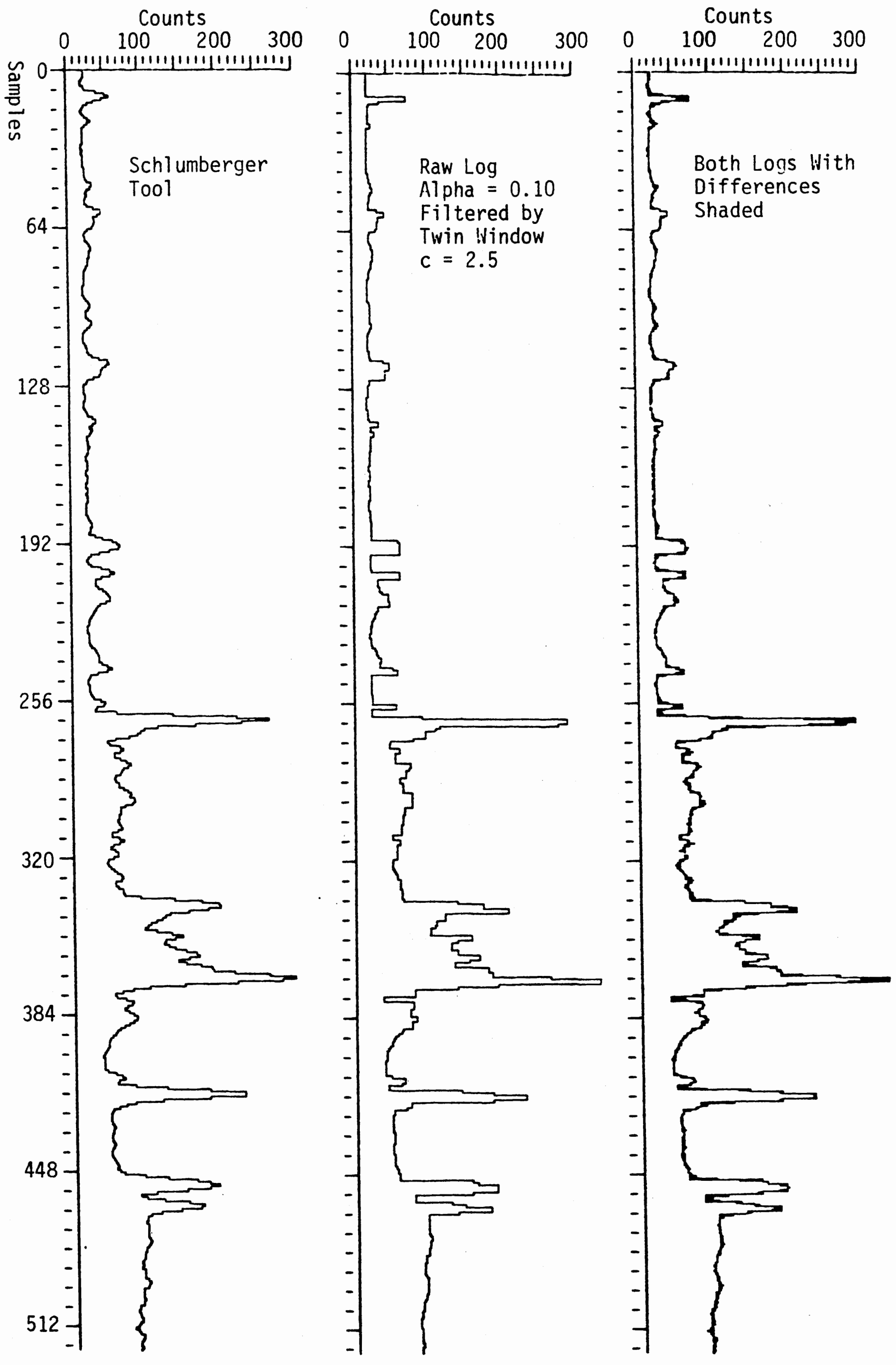


Figure 9. Schlumberger gamma-ray log, the log of the same borehole processed by another tool and TW filters, and a composite log with differences shaded.

for the particular filters used. The reduction in noise obtained by multiple linear regression is, in fact, impractical to implement as a filter since the weights obtained were allowed to, and did in fact, vary from log to log. Next, a novel filter, the twin window filter, was introduced and shown to be superior to even the impractical, but promising, results of the multiple linear regression. Further, the twin window offers the frame work in which to implement a number of different filters, which each produce better results on the synthetic log than any of the filters related to the recursive medians. This type of filter also has a filter parameter which can be adjusted to suit particular needs, or simply to improve the RMS difference. The most promising of these filters was applied to raw gamma-ray log data and seen to produce good results. Having used multiple linear regression to show how little optimal weighting improves linear combinations of recursive median filters, the twin window filter will, in the next chapter, be optimized with respect to the filter parameter and signal level.

CHAPTER IV

OPTIMIZATION OF THE TWIN WINDOW AVERAGE FILTER

The Twin Window Average Filter (TWAF) is defined such that the width of the internal window $W = 2c\sqrt{X_n}$ where c is the filter parameter and X_n is the value of the point in the center of the window as in the previous chapter. This leads to the observation that this window includes relatively fewer uncorrelated points when X_n is small than when it is large. Therefore making the inner window smaller for small X_n should improve the filter performance. This will be first be done using the Monte Carlo method.

4.1. A Special Synthetic Log

Since the idea behind the optimization is that the filter parameter may be beneficially adjusted for each level, a new type of synthetic log is constructed. Central to this new log is that it contains an excessive number of beds of a particular level which is denoted by V, for value. Each of the added beds in the synthetic log is the same length, L, to allow the statistics to be tabulated not only by level, but also by location within a particular bed. The first extra bed is after the *end* of the bed containing the 25th point of the log. The extra bed is generated again after the bed containing the 75th point of the log and the remaining log has an extra bed after the bed containing point at position 50N–25. The extra bed is only generated at the end of the bed at a particular position to preserve, to the greatest extent possible, the remaining characteristics of the synthetic log. So for the 2048-point logs used here, the new log will contain at least 41 beds of level V. Because the level of the random beds is a random double-precision floating point number, the chances of one of the other beds being generated at precisely that level are vanishing small. This special synthetic log with V=50 and L=5 is shown in Figure 10. For the purpose of showing greater detail, the first 256 points of same log are shown in Figure 11. The 256-point noisy log is shown with noise added in Figure 12.

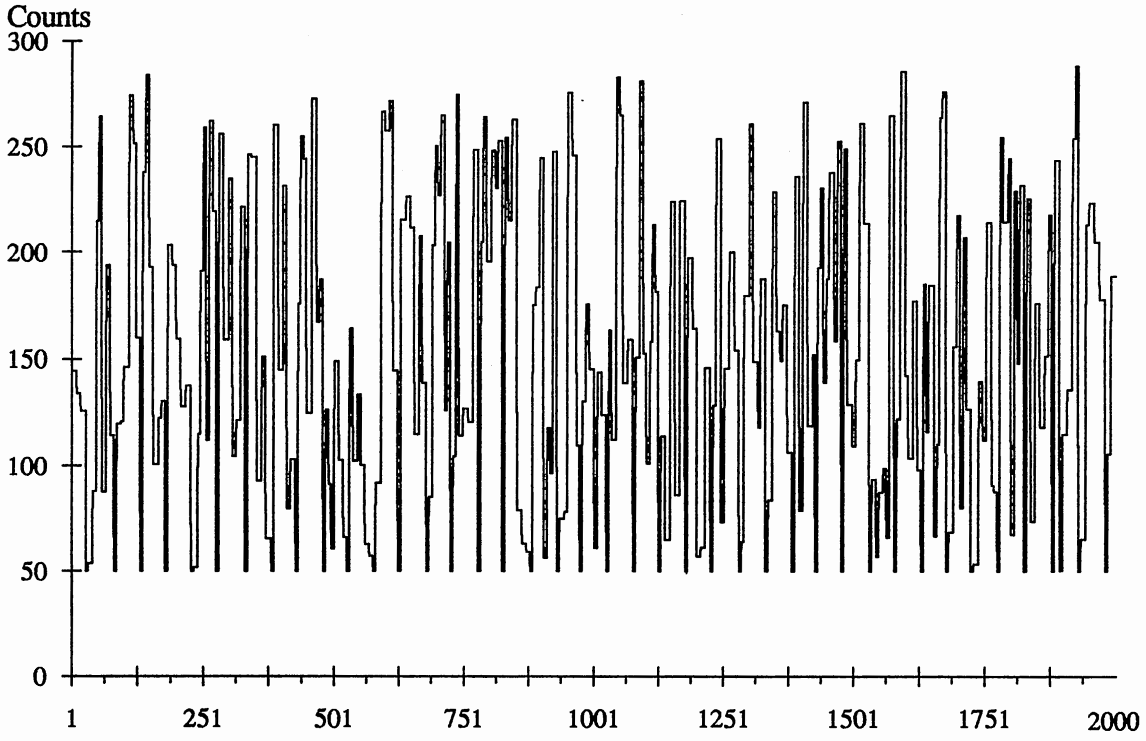


Figure 10. Most (97.6%) of the revised ideal log, showing the extra beds at V=50 with L=5.

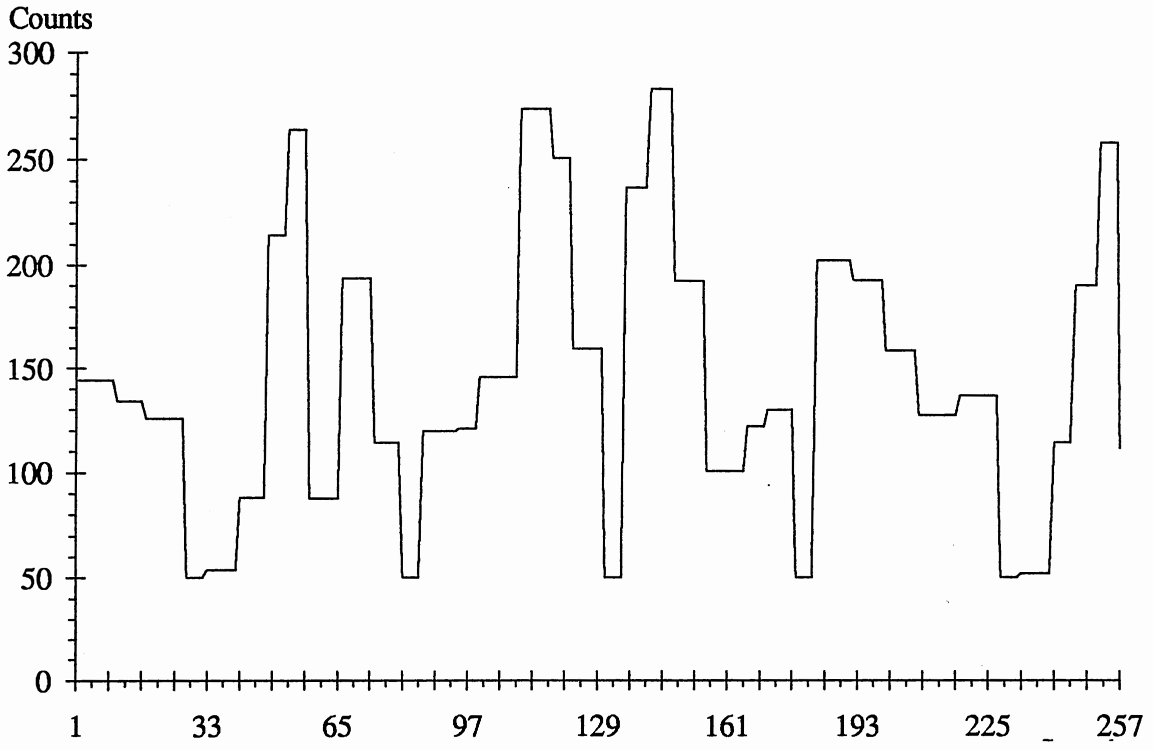


Figure 11. The first 256 points (12.5%) of the revised ideal log, with the added beds of length 5.

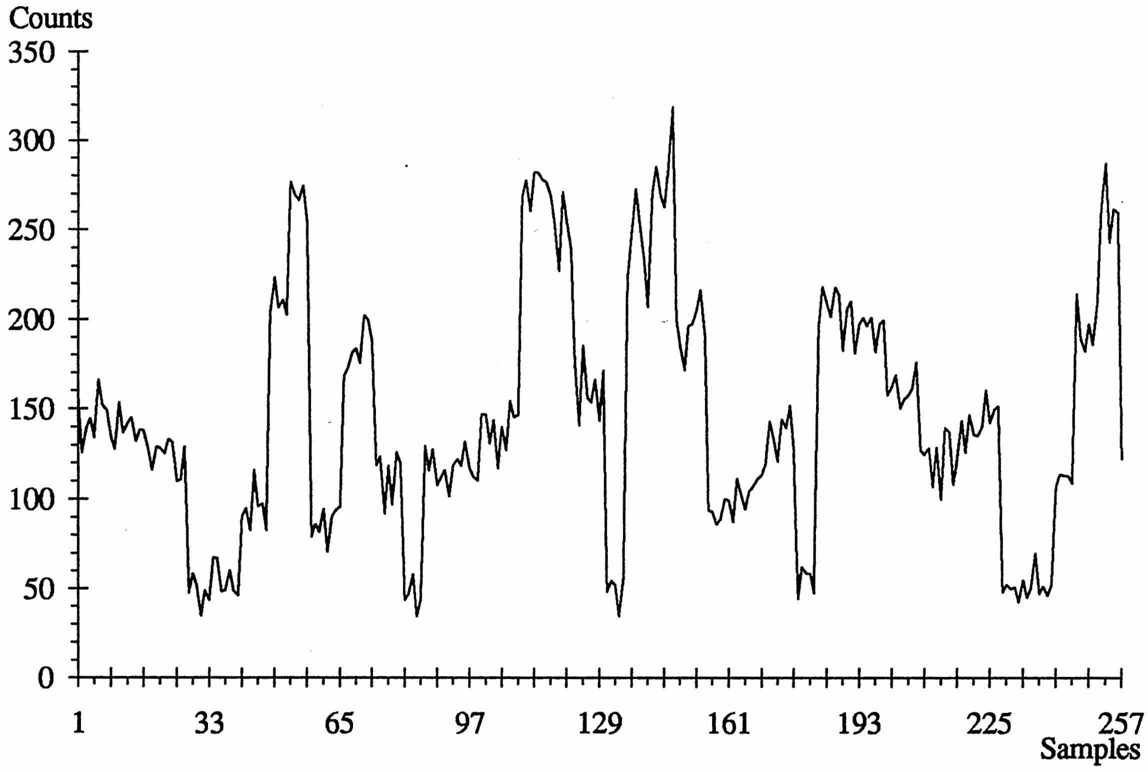


Figure 12. The first 256 points (12.5%) of the revised noisy log, with the added beds of length 5.

4.2. Monte Carlo Simulation

As in the previous cases, Monte Carlo simulation was used to model 1000 of these 2048-point logs. This undertaking was considerably complicated by the fact that statistics were collected for the mean and standard deviation for each individual point location in a bed for a particular signal level V and bed length L. This involved hundreds of files on disk for each level V, which each contain 4000 bytes. Each of these files was then summarized by its data count, mean, and standard deviation using the program `avg`, given in Appendix B. This summary was then further consolidated to produce the desired results. This results were then used to construct a modified version of the Twin Window filter which does, in fact perform better than the original. Taking these topics in order, the program to create the special synthetic log will now be explained.

4.2.1. Program Description

A suitable program was written to create the special synthetic log and was run to evaluate the mean and variance for all combinations of the values of these parameters:

$$c = 2, 2\tfrac{1}{8}, 2\tfrac{2}{8}, \ldots, 3\tfrac{6}{8}, 3\tfrac{7}{8}, 4$$

$$L = 5, 6, 7, 8, 9, 10$$

$$V = 50, 60, 90, 120, 150, 180, 210, 240, 270, 288$$

where c is the filter parameter, L is the width of the bed, and V is the level of the bed to be examined. It was run separately for each value of V since a run takes between 3 and 4 days, depending on how much cpu time was siphoned off for other activities. In addition to separate runs for each value V, a separate directory was used, although the file names are distinct.

As an example, let V=50 and L=5, for each of the 17 values of c, 12 files are populated with the results from 1000 different synthetic logs, giving a total of 204 files. Each set of 12 files includes the mean and standard deviation of these 6 results:

1. RMS difference of the ideal and filtered values of the first point in each bed of width 5. (Each log has 41 of this type of point.)
2. Similarly, the RMS difference concerning the second point.
3. Contains similar RMS difference for the third point.
4. Contains similar RMS difference for the fourth point.
5. Contains values for the fifth point.
6. RMS difference of the entire ideal and filtered log. (Each log has 2048 points, and each of them impact this evaluation.)

In addition to these important files of interest, 18 other files were created with various other values to check the results, for a total of 222 files.

TABLE 8

NUMBER OF FILES CREATED FOR EACH BED LENGTH.

Bed Length	Number of Files	Bytes
5	222	909,312
6	259	1,060,864
7	296	1,212,416
8	333	1,363,968
9	370	1,515,520
10	407	1,667,072
Total	1887	7,729,152

The number of files created for each bed length is given in Table 8. Because each file has one 4-byte floating point number for each of the usual 1000 logs, it is 4000 bytes long. Assuming a cluster size on MS-DOS of 4096, or a block size on Unix™, gives the total disk space consumed by the file, 4096 bytes, as in the third column. This excludes the space the directory entries require, which amounts to a little more than a single file in itself. For the ten values of V under consideration, the total requirement is well over 73 megabytes.

4.2.2. Checking the Simulation

Returning to the example with V=50 and L=5, one check that was made was the new mean value of the noisy synthetic log. Considering that each log has 2048 points, and 41 special beds of 5 points each, gives 1843 points uniformly distributed between 50 and 288, and 205 points having a mean value of 50. The expected mean value is

$$\frac{1843\frac{50+288}{2} + 205 \cdot 50}{2048} \approx 157.0884. \tag{34}$$

The square root of this value, 12.5335 is the expected RMS difference between the ideal and unfiltered noisy synthetic log. From the selected example file, for the 1000 logs, the mean RMS difference is 12.5265 with a standard deviation, per Equation (20), of $\frac{0.262372}{\sqrt{1000}} \approx 0.0083$, certainly a positive indication. In like manner, the mean value of the ideal log is 157.211 and the standard deviation is $\frac{3.88666}{\sqrt{1000}} \approx 0.1229$, so the Monte Carlo result is approximately 1 standard deviation from the expected result. Similarly, for the unfiltered noisy synthetic log, the numbers are remarkably close: 157.210 and $\frac{3.90131}{\sqrt{1000}} \approx 0.1234$, a remarkable one standard deviation from the expected. The overall result for the filtered log (c = 2.00), however, is 156.83 and $\frac{3.90224}{\sqrt{1000}} \approx 0.1234$, a little over 2 standard deviations from the expected result, but in the opposite direction, the first subtle hint of a possible bias in the estimator. This same bias in similar quantity is also seen across all the filter parameters (c=2.00 to 4.00). Cancelling this bias may be another means of improving filter performance, but first a means of consolidating this detailed, but imponderably large mass of data is given.

4.2.3. Consolidation of Data

The consolidation of this data occurs in two phases, the first combines the statistics of various points within a particular length bed, at a particular signal level, and the second combines the various length beds at a particular

TABLE 9

OPTIMAL VALUES OF THE TW FILTER PARAMETER FOR SELECTED SIGNAL LEVELS.

Signal Level	Filter Parameter
50	3.67
60	3.62
90	3.33
120	3.15
150	3.10
180	3.10
210	3.10
240	3.20
270	3.28
288	3.28

signal level. Once the data is consolidated, the required optimal values can be interpolated. These values are given in Table 9. So, as anticipated, the best values for the filter parameter are larger for the low signal levels, and smaller for high signal levels. But the optimal filter parameter also increases slightly again near a signal level of 240, for reasons, as yet, undetermined. Nevertheless, the above table was implemented in a Monte Carlo simulation and found to improve the results.

4.3. Results of the Improved Filter

To implement the improved filter, the existing code was modified by

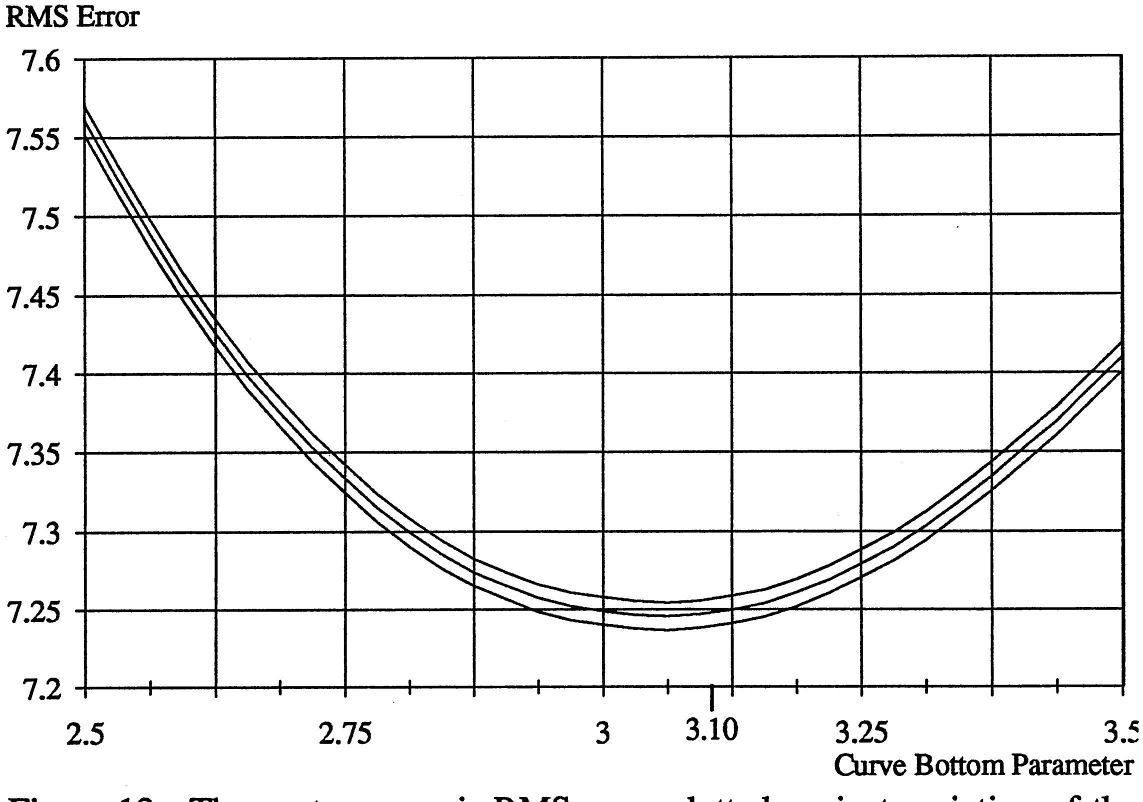


Figure 13. The center curve is RMS error plotted against variation of the optimal curve. The two outside lines are ±1 standard deviation. The value 3.10 is nominally optimal.

adding a subroutine called `optlvl` which given a signal level, returns the best filter parameter, interpolated from the data in Table 9. The altered routines are given in Appendix E. This program not only implements the filter, but varies the data in Table 9 by effectively adding a constant and evaluating the results. The data in Table 9 has the filter parameter value 3.10 as the bottom of a bathtub-shaped curve, so this value was selected as nominally optimal and a constant was added to the *entire* curve by the program. For instance, if the program parameter is 3.25, the entire curve is shifted upward 0.15. The program used parameters valued at every $\frac{1}{32}$ between 2.5 and 3.5, and the results are shown in Figure 13. While the graph of Figure 13 does not have its extremum precisely over 3.10, it is well within the ±1 standard deviation lines. It also clearly shows that the filter is somewhat tolerant to variation from the optimal in that perturbation of the optimal curve by as much as 0.10 produces only 0.02 change in RMS error, although further perturbation may cause substantially greater error.

The other thing to note is that, similarly to the optimization of the linear combinations of recursive median filters, the optimized filter only performs slightly better than the original. In the present case, the improved filter has an RMS error of slightly less than 7.25 while the original was 7.30. Since the standard deviation is approximately 0.009, these two estimates are less than 6 standard deviations apart. However, the improved filter requires only one additional subroutine call, which could easily be avoided by inline code, or the inline declaration in C++, and a table lookup. For such a small price, the improvement is probably worth the effort. With the improvement completed, let us summarize the steps involved to make practical use of the optimized twin window average filter.

4.4. A Method for Using the Filters

To make practical use of this filter, and to compare it to another log as done in the previous chapter, several steps are required. The first step is to scale the data in terms of depth to assure that similarly-numbered points correspond to the same geologic beds. This allows for the fact that even though the tool is first lowered and the well logged only as the tool is lifted, it may snag on the sides of the borehole and cable stretch may cause disparities in the data. This was not a problem in the previous chapter since both tools were run simultaneously. The second step is the selection of α in Equation (1). The exponential, when convolved with the log, gives the log the general appearance of indistinct beds, or sloping bed boundaries. Although scientific analysis based on the properties of the material surrounding the borehole may yield a proper value for α, simply applying different values and observing which value yields beds that appear distinct seems to work well. This was the method used in the previous chapter, and can be accomplished quite rapidly if the observer attaches a numeric value to the slope of the beds and tries to center the optimum α between values of α which gave similar slopes on opposite sides on the optimum.

The third problem is to relate the ordinate of the graph to the number of counts. This, too, turns out to be surprisingly simple if approached in the correct manner. The technique is to use the largest unchanging bed to approximate the mean, μ, and standard deviation, σ, for that one particular bed. If that bed is so narrow as to make the estimates uncertain, then several beds may be used, yielding the estimates μ_n, and σ_n. Once these are found, the required scaling factors, β_n, may be estimated as $\beta_n = \frac{\sigma_n}{\mu_n}$. These may be averaged using a weighted average which considers the number of points in the bed and the level of the bed, but for the data in the last chapter, the first two β_n's calculated were in close agreement, so this did not seem to be necessary in practice. Of course, if it is only desired to scale two logs of the same borehole to be equal, the scale factor may be taken as the square root of the ratio of the total power of each log.

The filter must then be selected. This should be done on the basis of the filters and parameters that appear to work best on the synthetic logs. The optimization of the filter parameter accomplished after the large expenditure of computer time in this chapter has two problems in practice. It is highly dependent on the distribution of geologic beds, which implies that a large number of actual logs are needed to estimate that distribution; and once the optimization is performed, it changes the performance of the filter only slightly. Therefore, the optimization is of little value unless one has a large quantity of data stored on a very large and fast computer. That is the reason that the log was filtered in the previous chapter, before obtaining this optimization.

4.5. Chapter Summary

Upon observation that due to the fact that the inner window of the Twin Window filter includes relatively more uncorrelated points for large signal values than small, a corresponding optimization of the filter with respect to the filter parameter was sought. The optimization was determined, in addition to numerous other statistics which, it was hoped, would be useful in better understanding the filter. After determining the optimal relationship between the filter parameter and the point to be estimated, the optimized filter was run over the usual 1000 synthetic logs to verify that it did, in fact, improve the performance. Also, the optimization function was perturbed, and it was shown that the perturbations degraded the performance within the certainty allowed by the Monte Carlo simulation. This simulation also showed that the improved version only did very slightly better than the original filter. However, the cost at execution time to implement the optimized filter is very low, so it is a worthwhile improvement, if the initial cost of finding the optimization can be justified.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

As Isaac Newton (1675) said, "If I have seen further than the others it is by standing on the shoulders of giants." From the literature, a basic model of gamma-ray logging was derived which showed how the Poisson noise could be viewed as adding to the signal emitted from the geologic beds. This is then convolved with the geologic impulse response which can be, with suitable restrictions, deconvolved to remove its effects. The problem of how to remove the Poisson noise without seriously degrading the signal which is composed of the sharp transitions representing the distinct layers of sand and shale was then addressed. The optimal stationary linear filter, designed by Wiener, removes only 90% of the noise, in the RMS sense.

Because so much noise remained, nonlinear filters were sought to improve the problem. The first of these, the median filters, actually made the noise worse. This was measured in Monte Carlo simulation, using a synthetic log due to the uncertainty associated with the true driving function of the physical system. The RMS difference was selected as the figure of merit. This made objective evaluation possible, and enabled the computer to evaluate large numbers of logs quickly. To quantify the amount of noise that it is possible to remove, an unattainable minimum noise level was derived, the value of which marked the best that could be achieved at 36.5% of the original noise. After ascertaining these two limits, work turned to finding filters which produced better results than linear filters. The first improvement was seen in the recursive median filters.

5.1. Recursive Median Filters

The recursive median filters which improve the noise level are the recursive medians of length 3, 5, and 7 with the RMS noise remaining of 73%, 77%, and 93%, respectively. Since these are not linearly dependent, linear combinations of these filters were tried and found to improve the noise level, leaving only 70% of the original noise. Also, since recursive median filtering in reverse does not produce the same result as forward filtering, the linear combinations were augmented with reverse recursive median filters for a greater improvement in noise level, with 66% of the noise remaining. So far, the weighting of the linear combinations has been uniform, so in an attempt to determine the optimum linear combination, a simulation was made with multiple linear regression used on each individual log. Even allowing the weights to change independently for each log only reduces the noise to 63%. This method is impractical to implement as a filter, and shows very little improvement over the simple uniform weights used previously. It has no known practical value, but it does demonstrate a limit to the amount of noise reduction available by means of such linear combinations. Having come to the apparent limit as to what can be expected in the realm of median filters, a novel filter, called the twin window, is introduced.

5.2. Twin Window Filters

The twin window filters were defined as a framework to surround various kernels, namely the average, maximum likelihood, and median, although others could easily be added in the future. These reduce the noise to 56.2%, 56.4%, and 57.6% of its original value, respectively. The inner window of the filter was allowed to vary, improving its position according to the best estimate of the current point, and reduced the noise to 55.9%, as well as producing these with a smaller value of filter parameter. Upon observation that the worst errors often occurred at single points, a 3-point recursive median was run after each of the first three twin window filters to achieve noise levels of 52.8%, 53.2%, and 55.8% respectively.

The observation that the twin window methodology allows more the uncorrelated points in the inner window at high signal level than at low levels leads to the optimization of the filter with respect to the filter parameter. This was carried out for the twin window average filter. The function of signal level versus filter parameter thus generated was found, when reapplied to the usual Monte Carlo simulation to reduce the noise to 55.7%. The optimal curve was then perturbed by adding a global constant and, within a fraction of 1 standard deviation, found to be consistent with the premise that the curve was indeed optimal. So a novel filter was introduced, investigated, optimized to a limited extent, and found to compare reasonably well to the unattainable minimum noise level of 36.5%.

5.3. Future Work

The state of knowledge on twin window filters presently is analogous to that of recursive medians before the investigation turned to multiple linear regression. However, the TW filter has a filter parameter, and thus, along with the various different kernels which may be used in the filter, leads to a larger system of regression equations. Once the resultant value is known, a decision to whether the potential reduction in noise is worth the added complexity of the filter. This complexity might be reduced by studying how much the filter parameter has to change to produce a filtered log of diversity sufficient to change the result of the regression.

Many times in the course of this work, investigation of the synthetic logs associated with the extreme tails of the histograms has provided insight into improvements in the filters. Often, this is because these logs represent a caricature of the important features which cause the extreme (good or bad) results. This will probably continue to be an important investigative tool. In addition to this, since the twin window filters have parameters, the histograms for different filter parameters could, in effect, be lined up and the little boxes reassigned to their particular log. Boxes from each histogram pertaining to the same log could be connected, and the correspondence of log to box rearranged within the column of each histogram to minimize the length of the connecting lines. Once this is done properly, if any long lines between boxes remain, they would be investigated individually to determine what physical features in the original log caused the sudden change in result. This method of investigation has been made practical only recently by the combination of high resolution graphics and fast processors on personal workstations.

Another aspect of the histograms that seems to demand investigation is the fact that reduced variance in the histogram seems to correlate with the greatest noise reduction. Further work in this area may lead to a simple heuristic to find the optimal filter parameter, or may lead to some totally unforeseen and better filter. Understanding of why this should be so certainly seems, at least at this point, to have some fundamental impact on nonlinear filters.

Leaving Poisson noise totally behind could also result in a very interesting investigation. The gamma-ray tool basically has not changed much since it was invented in 1939. A updated digital version could register every decay detected by sending the digitally-encoded time between decays. This leads to a different noise distribution, but has the advantage of less reduction of data before filtering, effectively destroying the role of the unattainable minimum bound, so prominently unassailable in this work.

The work of optimizing the twin window filter may be done analytically, although with numerous numerical integrations and convolutions, much more accurately and in less time. However, checking out such a program is very time-consuming and requires, among other things, functions which in fact evaluate accurately to nearly the full machine precision, unlike the error function often implemented on Unix™, which for some values, only produces nine or ten significant digits. An example of a correct function is given in Hart (1967), and is used here in the program `avg` for the more mundane and less demanding task of calculating the Gaussian curves approximating the histograms in the appendices. The time for such a program to do the same job, when carefully written, might be reduced by a factor of 25 or more for the 3-digit precision given, and would not increase as rapidly as the required precision increased. This method has the added advantage of generating intermediate results which might prove useful in improving the twin window filter in ways not now foreseen.

BIBLIOGRAPHY

Abramowitz, M. and Stegun, I.A. (1964), Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, National Bureau of Standards, Applied Math Series, No. 55.

Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. (1983), Data Structures and Algorithms, Addison-Wesley, Reading, Massachusetts.

Arce, G.R., and Gallagher, N.C., Jr. (1982), State Description for the Root-signal Set of Median Filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, No. 6, pp. 894-902.

Astola, J., and Campbell, T.G. (1989), On Computation of the Running Median, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 4, pp. 572-574.

Astola, J., Heinonen, P., and Neuvo, Y. (1989), Linear Median Hybrid Filters, IEEE Circuits and Systems, Vol. 36, No. 11, pp. 1430-1438.

Ataman, E., et al. (1980), Fast Method for Real-time Median Filtering Algorithm, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-28, pp. 415-420, August 1980.

Bednar, J.B., and Watt, T.L. (1984), Alpha-trimmed Means and Their Relationship to Median Filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, pp. 145-153, February 1984.

Bendat, Julius S., and Piersol, Allan G. (1971), Random Data: Analysis and Measurement Procedures, Wiley, New York, Chapter 4.

Berezin, I.S., and Zhidkov, N.P. (1965), Computing Methods, Vol. 1, translation of the Russian, Pergamon Press, Oxford, and New York, or Addison-Wesley Publishing Co, Inc, Reading, Massachusetts.

Billinton, Roy (1970), Power System Reliability Evaluation, Gordon and Breach, New York, p. 28.

Conaway, John G. and Killeen, P.G. (1978), Computer Processing of Gamma-ray Logs: Iteration and Inverse Filtering, Current Research, Part B, Geological Survey of Canada, Paper 78-1B, pp. 83-88.

Conaway, J.G. and Killeen, P.G. (1978), Quantitative Uranium Determinations from Gamma-ray Log by Application of Digital Time Series Analysis, Geophysics, Vol. 43, No. 6, pp. 1204-1221, October 1978.

Conaway, J.G., Bristow, Q., and Killeen, P.G. (1979), Optimization of Gamma-ray Logging Techniques for Uranium, draft, Geophysics, Vol. 45.

Conaway, J.G. (1979), Problems in Gamma-ray Logging: The Effect of Dipping Zones on the Accuracy of Ore Grade Determinations, Current Research, Part A, Geological Survey of Canada, Paper 79-1A, pp. 41-44.

Conaway, J.G. (1979), The Effects of Borehole Diameter, Borehole Fluid, and Casing Thickness on Gamma-ray Logs in Large Diameter Boreholes, Current Research, Part C, Geological Survey of Canada, Paper 79-1C, pp. 37-40.

Conaway, J.G. (1980), Uranium Concentrations and the System Response Function in Gamma-ray Logging, Current Research, Part A, Geological Survey of Canada, Paper 80-A, pp. 77-87.

Conaway, J.G. (1980), Exact Inverse Filters for the Deconvolution of Gamma-ray Logs, Geoexploration, Vol. 18, pp. 1-14.

Conaway, J.G. (1980), Direct Determination of the Gamma-ray Logging System Response Function in Field Boreholes, Geoexploration, Vol. 18, pp. 187-199, January 1980.

Conaway, John G. (1981), Deconvolution of Gamma-ray Logs in the Case of Dipping Radioactive Zones, Geophysics, Vol 46, No. 2, pp. 198-202, February 1981.

Conaway, John G. (1980), Uranium Concentrations and the System Response Function in Gamma-ray Logging, Current Research, Part A, Geological Survey of Canada, Paper 80-A, pp. 77-87.

Czubek, J.A. (1961), Some problems of the Theory and Quantitative Interpretation of the Gamma Ray Logs, Acta Geophysica Polonica, Vol. 9, No. 1/2, pp. 121-137.

Czubek, J.A. (1962), The Influence of the Drilling Fluid on the Gamma-ray Intensity in the Borehole, Acta Geophysica Polonica, Vol. 10, pp. 25-30.

Czubek, J.A. (1966), Physical Possibilities of Gamma-gamma Logging, Radioisotope Instruments in Industry and Geophysics, Vol. 2, International Atomic Energy Agency Proceedings Series, IAEA, Vienna, 1966.

Czubek, J.A. (1969), Influence of Borehole Construction on the Results of Spectral Gamma-logging, Nuclear Techniques and Mineral Resources, International Atomic Energy Agency Proceedings Series, IAEA, Vienna.

Czubek, J.A. (1971), Differential Interpretation of Gamma-ray Logs, I. Case of the Static Gamma-ray Curve, Report No. 760/I, Nuclear Energy Information Center, Polish Government Commission for the Use of Nuclear Energy, Warsaw.

Czubek, J.A. (1972), Differential Interpretation of Gamma-ray Logs, II. Case of the Static Gamma-ray Curve, Report No. 793/I, Nuclear Energy Information Center, Polish Government Commission for the Use of Nuclear Energy, Warsaw.

Czubek, J.A. (1983), Kernel Functions in Gamma-ray Logs, draft accepted for publication in Acta Geophysica Polonica.

Davis, Philip J. and, Rabinowitz, Philip (1967), Numerical Integration, Blaisdell Publishing Company, Waltham, Massachusetts.

Davisson, C.M. and Evans, R.D. (1976), Gamma-ray Absorption Coefficients, Reviews of Modern Physics, Vol. 24, No. 2, pp. 79-107.

Davydov, Y.B. (1970), One-Dimensional Inverse Problem of Gamma Logging of Borehole (translated from Odnomernaya obratnaya zadacha gamma-karotazha skvazhin. Izvestiya Vysshikh Uchebnykh Zavedenii.) Geologiya i Razvedka, No. 2.

Elphick, R.Y. (1987), Neutron/Density/GR Interpretation in Shaley [sic] Sands, Geobyte, Vol. 2(2), pp. 51-54.

Devore, J.L. (1982), Probability and Statistics for Engineering and the Sciences, Brooks/Cole, Monterey, California, pp. 485-491.

Evans, Robley D. (1955), The Atomic Nucleus, McGraw-Hill Book Co., New York.

Fike, C.T. (1968), Computer Evaluation of Mathematical Functions, Prentice-Hall, Inc, Englewood Cliffs, New, Jersey.

Flaum, C., Galford, J.E., and Hastings, A., (1989), Enhanced Vertical Resolution Processing of Dual Detector Gamma-Gamma Density Logs, The Log Analyst, Vol. 30(3), May-June 1989, pp. 139-149.

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. (1977), Computer Methods for Mathematical Computations, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 97-107.

Fry, Thornton C. (1965), Probability and Its Engineering Uses, D. Van Nostrand Company, Princeton, New Jersey, pp. 236-253.

Gallagher, N.C., and Wise, G.L. (1981), A Theoretical Analysis of the Properties of Median Filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, December 1981.

Grenander, Ulf (1982), Mathematical Experiments on the Computer, Academic Press, New York, pp. 430-435.

Hart, John F., Cheny, E.W., et al., (1968), Computer Approximations, John Wiley & Sons, New York.

Haight, Frank A. (1967), Handbook of the Poisson Distribution, John Wiley & Sons, Inc., New York.

Howell, Lynn G., and Frosch, Alex (1939), Gamma-Ray Well-Logging, Paper read at the Annual Meeting of the Board of Directors, Humble Oil & Refining Co., Oklahoma City, Oklahoma, March 23, 1939.

Huang, T. S., Yang, G. T., and Tang, G. Y. (1979), Fast Two-dimensional Median Filtering Algorithm, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-27, pp. 13-18, February 1979.

Huang, Thomas S. (1981), Editor, Two-Dimensional Digital Signal Processing II, Springer-Verlag, New York, pp. 161-217.

Jayant, N. S. (1976), Average and Median Based Smoothing Techniques for Improving Digital Speech Quality in the Presence of Transmission Errors, IEEE Transactions on Communication, Vol. COM-24, pp. 1043-1045, September 1976.

Hurst, A. (1990), Natural Gamma-Ray Spectrometry in Hydrocarbon-bearing Sandstones from the Norwegian Continental Shelf, Geological Applications of Wireline Logs, Geological Society of London Special Publication No. 48, pp. 211-222.

Karaman, M., and Onural, L. (1989), New Radix-2-Based McGraw for Fast Median Filtering, Electr. Lett., Vol. 25, No. 2, pp. 723-724.

Killeen, P.G. (1983), Borehole Logging for Uranium by Measurement of Natural Gamma-radiation, International Journal of Applied Radiation and Isotopes, Vol. 34, pp. 231-260, January 1983.

Koizumi, C.J. (1985), Computer Determination of Calibration and Environmental Corrections for A Natural Spectral Gamma Ray Logging System, Society of Petroleum Engineers, 60th Annual Meeting Preprint, Vol. SPE-14186. Also in SPE Formation Evaluation, Vol. 3, No. 3, 1988, pp. 637-650.

Kozhevnikov, D.A., and Shagin, V.L. (1989), A Method of Treating the Spectral Response of a Tool in Open and Cased Boreholes to Determine the Natural Radioactivity of Rocks, Nuclear Geophysics, Vol. 3, No. 1, pp. 17-29.

Krishnan, Venkatarama (1984), Nonlinear Filtering and Smoothing: An Introduction to Martingales, Stochastic Integrals and Estimation, John Wiley & Sons, New York, pp. 245-270.

Leithold, Louis (1972), The Calculus with Analytic Geometry, Second Edition, Harper & Row, New York.

Longbotham, Harold Gene, and Bovik, Alan Conrad (1989), IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 2, February, pp. 275-287.

Mathis, G.L. (1987), Smoothing Spectral Gamma Logs-A Simple but Effective Technique, Geophysics, Vol. 52, No. 3, pp. 363-367.

Mendel, Jerry M. (1987), Lessons in Digital Estimation Theory, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Milton, Roy C. (1970), Rank Order Probabilities, John Wiley & Sons, Inc., New York.

Newton, Isaac (1675), Letter to Robert Hooke, dated February 5.

Nodes, T.A., and Gallagher, N.C., Jr. (1982), Median Filters: Some Modifications and Their Properties, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, No. 5, pp. 739-746, October 1982.

Paden, Larry J., and Steinhardt, Allan O. (1984), Reduction of Poisson Noise in the Gamma-ray Log, 1983-84 Program Final Report, Oklahoma State University Research Consortium for Enhancement of Well Log Data via Signal Processing, Stillwater, Oklahoma, pp. 2.1-2.53.

Paden, Larry J. (1985), Reduction of Noise in the Gamma-Ray Log, SPWLA Symposium Record, Dallas.

Papoulis, Athanasios (1977), Signal Analysis, McGraw-Hill, New York, pp. 337-339.

Park, S.Y. and, Lee, Y.H., Double Smoothing of Images Using Median and Wiener Filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 6, pp. 943-946.

Peebles, Peyton Z. (1987), Probability, Random Variables, and Random Signal Principles, McGraw-Hill, New York.

Peterson, S.R., Lee, Y.H., and Kassam, S.A. (1989), Spectral Performance Characterizations of Some Generalized Median Filters, The Journal of the Franklin Institute, Vol. 326, No. 2, pp. 151-166.

Prudnikov, Anatolii Platonovich (1986), Translation of (1983) Integraly I Ryady, Gordon and Breach Science Publishers, New York.

Pomalaza-Raez, Carlos A., and McGillem, Clare D. (1984), An Adaptive, Nonlinear Edge-preserving Filter, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, pp. 571-576.

Ruckebusch, Guy (1983), A Kalman Filtering Approach to Natural Gamma Ray Spectroscopy in Well Logging, IEEE Transactions on Automatic Controls, Vol. AC-28, pp. 372-380.

Richards, D.S., (1990), VLSI Median Filters, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 38, No. 1, pp. 145-153.

Schlumberger (1982), Fundamentals of Natural Gamma Ray Spectrometry and Essentials of NGS Interpretation, Schlumberger Technical Services, Document number M-081025, Paris.

Schultz, Ward E. and Thadani, Suresh G. (1981), Applications of Digital Filtering Techniques to Nuclear Well Logs, SPWLA Logging Symposium Transactions.

Scott, J.H. (1963), Computer Analysis of Gamma-ray Logs, Geophysics, Vol. 28, pp. 457-465.

Scott, J.H., et al. (1961), Quantitative Interpretation of Gamma-ray Logs, Geophysics, Vol. 26, pp. 182-191.

Sinha, P.K. (1990), An Improved Median Filter, IEEE Transactions on Medical Imaging, Vol. 9, No. 3, pp. 345-346.

Snyder, Donald L. (1975), Random Point Processes, John Wiley & Sons, New York.

Tukey, J.W. (1971), Exploratory Data Analysis (preliminary ed.), Addison-Wesley, Reading, Mass.

Wang, D.J. (1990), On the Max Median Filter, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 38, No. 8, pp. 1473-1475.

Yung, W.F., Yuen, Y.K., Koh, S.N., and Lee B.S., Implementation of a Pseudo 2-D Median Filter for Image-Enhancement, IEEE Transactions on Consumer Electronics, Vol 35, No. 4, pp. 901-906.

Zorski, Tomasz (1980), Application of Discrete Fourier Transform in Solving the Inverse Problem in Gamma-Ray Logging, Acta Geophysica Polonica, Vol. 28, No. 1, pp. 57-66.

APPENDIX A

PROGRAM TO EVALUATE THE TWIN WINDOW AVERAGE FILTER

The program in this appendix is typical of those used to evaluate the filters presented here. It was originally written for a DEC Vax 11/750, then ported to an AT&T 3B2/400, and finally to an IBM AT clone from Gateway housing a Micronics motherboard with an Intel 80486 processor. The Vax was, of course, multiuser, so typically a run would actually get less than 30% to 50% of the total CPU time. The 3B2 was also multiuser, but due to lack of after hours use, often runs would get 90% to 95% of the CPU time. The overall performance was approximately half the speed for a given amount of CPU time. The AT runs Microsoft Windows/386 and the program approximately 10 to 12 times the speed of the Vax, a much needed improvement for some of the more complex programs.

The routines required to make a complete working program are:

```
twaf.c, bubble.c, cputime.c, daytime.c, flush.c, gamgev2.c, gauss.c, ncproces.c,
poisnois.c, randb.c, reload.c, rmfilt.c, rmssub.c, twaffilt.c.
```

The main routine is `twaf.c` and the others are subroutines listed in in alphabetical order for the convenience of the reader, although it may not be the most efficient from a machine's point of view. In order to assist the potential programmer to understand the dependencies, the following topological sort may be helpful:

```
twaf.c calls:
    cputime.c, daytime.c, flush.c, gamgev2.c, ncproces.c, poisnois.c, reload.c, rmfilt.c,
    rmssub.c, twaffilt.c.
cputime.c calls nothing.
daytime.c calls nothing.
flush.c calls nothing.
gamgev2.c calls
    gauss.c, randb.c.
ncproces.c calls nothing.
poisnois.c calls
    gauss.c.
reload.c calls nothing.
rmfilt.c calls
    bubble.c.
rmssub.c calls nothing.
twaffilt.c calls:
    bubble.c.
bubble.c calls nothing.
gauss.c calls:
    randb.c.
randb.c calls nothing.
```

This source code, when compiled with Borland's Turbo C++ Professional, combines to produce an executable program. Its original form was that of 14 separate source files and a projectfile which compiled each into an object file, then linked these 14 object files together to produce an executable. When run, produces 67 files requiring over 300k bytes of disk space. One file, called `datatext`, is a text file which gives a running account of the time at which the program last added to its data files. When run under a multitasking operating system, this file may be checked to monitor the progress of the program. A second file, called `dataseed`, contains the seed of the multiplicative congruential random number generator before the synthetic log was generated. It is often useful for reconstructing the log for which a given filter produces an exceptionally large or small result. The remaining 65 files are the results of varying the filter parameter between 2.00 and 4.00 by $\frac{1}{32}$, which corresponds to the file names `d2_00.ot`, `d2_03.ot`, ..., `d4_00.ot`, respectively.

This last group of 65 files represents the data central to the theme of the program. Each file consists of 1000 single-precision floating-point numbers stored in binary format. Each of these numbers represents the RMS average difference between an ideal synthetic log and the corresponding filtered noisy synthetic log. As an example, consider the file `d3_15.ot`. From the file name, the filter parameter can be deduced as $3\frac{3}{32}$, which, although somewhat cumbersome written on paper, has an exact, and simple, binary representation. Each set of four bytes in the file is the binary representation of a single-precision floating-point number. So bytes 9 to 12 represent a number (say 7.26, for example's sake.) This means that when the third 2048-point noisy synthetic log was filtered, that the RMS difference between it and the original ideal synthetic log was 7.26. Of course reading this binary data is somewhat difficult, so in Appendix B is the program listing for a utility that assists in this endeavor.

The main program `twaf.c`:

```
/*  program otwl    !Ordinary twin window. (Uses median without RM3.)
Ideal gamma-ray log generator.
Makes multiple runs with different noise files.  Finds the RMS average of
the difference between the ideal log and a simple combination of recursive
median filters.  Larry Paden 3/30/84.
    Changed to print out the seed before each pass.  LJP 8/23/83.
    Synthetic log generator changed to produce small layers.  LJP 10/5/83.
    Prints the unformatted files directly.  LJP 8/30/84.
    Some subroutines are in [LARRY.PADEN]LIBRARY
    RELOAD subroutine added to pick up where it left off.  LJP 10/2/84.
    PHOENIX added.  LJP 10/4/84.
    Converted to C.  ljp 4/5/87.*/
#include <stdio.h>
#include <sys\stat.h>
/* The directory for a RAM disk to cut down on hard disk use. */
#define DIR "D:\\twaf"
/* Signal length in samples. */
#define LENSIG 2048
/* Number of noisy logs. */
#define RUNS 1000
/* Controls verbosity of text output. */
#define IRMSVERB 0
```

```
/* The number of points in the outer window. */
#define LARGE 9
/* Lowest filter parameter. */
#define BOTTOM 2.0
/* Highest filter parameter. */
#define TOP 4.0
/* Filter parameter increment.  Change filename in sprintf %.2f below.*/
#define DELTA 0.03125
/* Minimum time in seconds between harddisk writes. */
#define TIMEALL (600.0)
/* Minimum slope. */
#define SLPMIN 0.0
/* Maximum slope. */
#define SLPMAX 0.0
/* Name of file for text output. */
#define LUWNAME "datatext"
char datafile[128] = "data";        /* Place to store a data file name. */
void main() {
    FILE *luwrite,              /* Logical unit for writing text. */
        *tempfile,              /* LU for other files. */
        *hardfile;              /* LU to copy to hard disk. */
    void flush();               /* For MS-DOS fflush only. */
    double cputime();           /* CPU (?) time used.  MS-DOS elapsed time. */
    double lasttime;                /* Save the last time files were transfered. */
    static double param,        /* Filter parameter. */
        rmsdiff,            /* RMS difference. */
        rmsavg1, rmsavg2,   /* RMS average of the two logs. */
        signal[LENSIG], ideal[LENSIG], xplt[LENSIG];
    float wrflt;                /* Convert to single precision. */
    int limit;                  /* Length of a log in points. */
    int irun;                   /* Log suite sequence number. */
    int junk;                   /* Info returned, but not used. */
    int kk;                     /* Loop counter through a log. */
    int errtmp;                 /* Error message data. */
    unsigned short iseed[3];    /* Random number generator seed. */
    int sprflng, sprfsht, sprfint;          /* For checking sprintf results. */
    int readdesc, writdesc, nitems, nindx, errnbr;  /* For copying files. */
    limit = LENSIG;
    /* Write various input parameters. */
    luwrite = fopen (LUWNAME, "a+");
    daytime (luwrite);
    fprintf (luwrite, "Hard disk is updated every %g seconds from %s.\n",
        TIMEALL, DIR);
    fprintf (luwrite, "The outer window is length:  %d.\n", LARGE);
    fprintf (luwrite, "Bottom, top, delta: %g, %g, %g\n",
        BOTTOM, TOP, DELTA);
    fprintf (luwrite, "Files have %d points; transitions %g to %g.\n",
        LENSIG, SLPMIN, SLPMAX);
    flush (luwrite);
    /* Initialize seed. */
    iseed[0] = 0xe66d;
    iseed[1] = 0xdeec;
    iseed[2] = 0x5;
    /* Initialize everyting else */
```

```
if (stat (DIR, (struct stat *)ideal) == 0) {
    fprintf (luwrite, "Directory %s already exists.  ", DIR);
    fprintf (luwrite, "Are other processes using it?\n");
    exit (-8);
}
if ((errtmp = mkdir (DIR)) != 0) {
    fprintf (luwrite, "Cannot make directory %s %d\n", DIR, errtmp);
    exit (-9);
}
sprflng = sprintf (datafile, "%s\\d%d_%.2d.ot", DIR, 4, 0);
sprfsht = sprintf (datafile, "%s\\d%d_%.2d.ot", ".", 4, 0);
lasttime = 0.0;
irun = 0;
ncprocessor();
/* Reload all arrays if data is available. */
irun = reload (luwrite, BOTTOM, TOP, DELTA, iseed);
fprintf (luwrite, "Run and seed: %d %ux %ux %ux\n",
    irun, iseed[2], iseed[1], iseed[0]);
daytime(luwrite); flush (luwrite);
/* Do the number of times in RUNS. */
for (irun=irun; irun<RUNS; irun++) {
    printf ("%d ", irun);
    daytime (stdout);
    fprintf (luwrite, "%d ", irun);
    daytime (luwrite);
    flush (luwrite);
    /* Save the seed. */
    if ((tempfile = fopen ("dataseed", "a+b")) == NULL) {
        fprintf (luwrite, "Stopped by fopen dataseed.\n");
        exit (-10);
    }
    if ((errtmp = fseek (tempfile, (long)(3*sizeof(short)*irun), 0))
        != 0) {
        fprintf (luwrite, "Seek error on dataseed.  %d\n", errtmp);
        exit (-11);
    }
    if (fwrite ((char *) iseed, sizeof(short), 3, tempfile) != 3) {
        fprintf (luwrite, "Write error on dataseed.\n");
        exit (-12);
    }
    if (fclose (tempfile) == EOF) {
        fprintf (luwrite, "Stopped by fclose dataseed.\n");
        exit (-13);
    }
    /* Check time. */
    if (cputime()-lasttime > TIMEALL) {
        lasttime = cputime();
        fprintf (luwrite, "Appending files.\n");
        fprintf (luwrite, "%g %d %ux %ux %ux\n",
            cputime(), irun, iseed[2], iseed[1], iseed[0]);
        daytime(luwrite); flush (luwrite);
        /* Append the RAM disk files to the files in this directory. */
        for (param=BOTTOM; param<=TOP; param+=DELTA) {
            sprfint = param;
```

```
if ((errtmp = sprintf (datafile, "%s\\d%d_%.2d.ot",
    ".", sprfint, (int)(100.*(param-sprfint)))) != sprfsht) {
    fprintf (luwrite,
        "Stopped due to sprintf %d return code -16.\n", errtmp);
    exit (-14);
}
if ((hardfile = fopen (datafile, "a+b")) == NULL) {
    fprintf (luwrite, "Stopped by fopen %s.\n", datafile);
    exit (-15);
}
if ((errtmp = sprintf (datafile, "%s\\d%d_%.2d.ot",
    DIR, sprfint, (int)(100.*(param-sprfint)))) != sprflng) {
    fprintf (luwrite,
        "Stopped due to sprintf %d return code -14.\n", errtmp);
    exit (-16);
}
if ((tempfile = fopen (datafile, "rb")) == NULL) {
fprintf (luwrite, "Stopped by fopen %s.\n", datafile);
    exit (-17);
}
readdesc = fileno(tempfile);
writdesc = fileno(hardfile);
while ((nitems =
    read (readdesc, (float*)ideal, (unsigned)LENSIG)) > 0) {
    if ((errnbr = write (writdesc, (float*)ideal,
        (unsigned)nitems)) != nitems) {
        fprintf (luwrite, "Bad write to %s %d/%d.\n",
            datafile, errnbr, nitems);
        fprintf (stderr, "Bad write to %s %d/%d.\n",
            datafile, errnbr, nitems);
        sleep (600);
        nindx = errnbr;
        nitems -= errnbr;
        /* While hard disk is full, give user a chance to fix. */
        while ((errnbr = write (writdesc,
            &((float*)ideal)[nindx],
            (unsigned)LENSIG)) != nitems) {
            fprintf (luwrite, ".");
            fprintf (stderr, ".");
            nindx += errnbr;
            nitems -= errnbr;
            sleep (600);
        }
    }
    if (fclose (tempfile) == EOF) {
        fprintf (luwrite, "Stopped by fclose tempfile.\n");
        exit (-18);
    }
    if (fclose (hardfile) == EOF) {
        fprintf (luwrite, "Stopped by fclose hardfile.\n");
        exit (-19);
    }
    unlink (datafile);
}
```

```
            }
        }
        /* Create synthetic log. */
        gamgev2 (ideal, limit, iseed, SLPMIN, SLPMAX);
        /* Copy the ideal synthetic log and add noise. */
        for (kk=0; kk<limit; kk++) signal[kk] = ideal[kk];
        poisnois (signal, limit, iseed);
        /* Compute RMS between noisy and ideal. */
        rmssub (ideal, signal, 0, limit, IRMSVERB,
            &rmsdiff, &rmsavg1, &rmsavg2);
        /* Save the RMS value. */
        if ((tempfile = fopen ("datanois", "a+b")) == NULL) {
            fprintf (luwrite, "Stopped by fopen datanois.\n");
            exit (-21);
        }
        if ((errtmp = fseek (tempfile, (long)(sizeof(wrflt)*irun), 0))
            != 0) {
            fprintf (luwrite, "Seek error on datanois.  %d\n", errtmp);
            exit (-22);
        }
        wrflt = rmsdiff;
        if ((errtmp = fwrite ((char *) (&wrflt), sizeof(wrflt),
            1, tempfile)) != 1) {
            fprintf (luwrite, "Stopped by fwrite datanois. %d\n", errtmp);
            exit (-23);
        }
        if (fclose (tempfile) == EOF) {
            fprintf (luwrite, "Stopped by fclose datanois.\n");
            exit (-24);
        }
        /* Copy the data so that the copy can be filtered. */
        for (param=BOTTOM; param<=TOP; param+=DELTA) {
            for (kk=0; kk<limit; kk++) {
                    xplt[kk] = signal[kk];
            }
            twaffilt (xplt, limit, LARGE, param, &junk, 0, &ideal[1836]);
/*          rmfilt (xplt, limit, 3, &junk, 1);*/
            rmssub (ideal, xplt, 0, limit, IRMSVERB,
                &rmsdiff, &rmsavg1, &rmsavg2);
            sprfint = param;
            if ((errtmp = sprintf (datafile, "%s\\d%d_%.2d.ot",
                DIR, sprfint, (int)(100.*(param-sprfint)))) != sprflng) {
                fprintf (luwrite,
                    "Stopped due to sprintf %d return code -30.\n", errtmp);
                exit (-30);
            }
            if ((tempfile = fopen (datafile, "a+b")) == NULL) {
                fprintf (luwrite, "Stopped by fopen %s.\n", datafile);
                exit (-31);
            }
            if ((errtmp = fseek (tempfile, (long)(sizeof(wrflt)*irun), 0))
                != 0) {
                fprintf (luwrite, "Seek error on %s.  %d\n", datafile, errtmp);
                exit (-32);
            }
            wrflt = rmsdiff;
            if ((errtmp = fwrite ((char *) (&wrflt), sizeof(wrflt),
                1, tempfile)) != 1) {
                fprintf (luwrite, "Stopped by fwrite %s.\n", datafile);
                exit (-33);
            }
            if (fclose (tempfile) == EOF) {
                fprintf (luwrite, "Stopped by fclose %s.\n", datafile);
                exit (-34);
            }
        }
    }
    daytime (luwrite);
    fprintf (luwrite, "Finished!!!");
    fclose (luwrite);
    system ("NEXT.BAT\n");
}
/* Deal with a peculiarity of MS-DOS.  See Turbo C++ help on fflush(). */
void flush(FILE *stream)
{
     int duphandle;

     /* flush the stream's internal buffer */
     fflush(stream);

     /* make a duplicate file handle */
     duphandle = dup(fileno(stream));

     /* close the duplicate handle to flush the DOS buffer */
     close(duphandle);
}
```

Subroutine `bubble.c`:

```
/*  Bubble sort.  By Larry Paden 5/19/83.  Translated to C 4/5/87. */
/*  This would probably run faster written with pointers. */
void bubble (double xx[], int istart, int iend)
/* double *xx;          /* Data to be sorted. */
/* int istart;          /* First location to be sorted. */
/* int iend;            /* Last location to be sorted. */
{
    double temp;    /* Temporary location. */
    int ii, jj;     /* Indices. */
    /* Do some parameter checking.  */
    if (istart >= iend) {
        printf ("Bubble did nothing. %d %d\n", istart, iend);
        return;
    }
    if (iend-istart >= 1024) {
        printf ("You could really speed this up with a binary sort.");
    }

    /* Sort it. */
    for (jj=istart+1; jj<=iend; jj++) {
        temp=xx[jj];
        ii = jj;
        while (temp < xx[ii-1]) {
            xx[ii] = xx[ii-1];
            ii = ii-1;
            if (ii <= istart) break;
        }
        xx[ii] = temp;
    }
}
```

Subroutine cputime.c:

```
/* Returns a double of elapsed time in MS-DOS.  Larry Paden 7/30/90. */
/* Bugs:  not really the cpu time, as on the multi-user systems of old. */
#include <dos.h>
#include <math.h>
#include <sys/timeb.h>
static double basis = 0.0;
double cputime () {
    struct timeb timebuf;
    ftime(&timebuf);
    if (basis != 0.0) {
        return (((double) timebuf.time + (double) timebuf.millitm / 1000.0) - basis);
    }
    else {
        basis = (double) timebuf.time + (double) timebuf.millitm / 1000.0;
        return (0.0);

    }
}
```

Subroutine daytime.c:

```
/* Prints time on input LU.  Larry Paden 4/10/86.*/
#include <stdio.h>
#include <time.h>
#include <sys/timeb.h>
void daytime(FILE *lu)
{
    long clock, time();
    char *ctime();
    struct tm *localtime(), *now;
    struct timeb timebuf;
    time(&clock);
    ftime(&timebuf);
    /* fprintf (lu, ctime(&clock)); Somewhat verbose. */
    now = localtime(&clock);
    fprintf (lu, "%.2d/%.2d/%.2d %.2d:%.2d:%.2d.%.3d\n",
        now->tm_year, now->tm_mon+1, now->tm_mday,
        now->tm_hour, now->tm_min, now->tm_sec,
        (int) timebuf.millitm);
    /* Military dates are readily machine sortable. */
}
```

Subroutine `flush.c`:

```
/* Deal with a peculiarity of MS-DOS.  See Turbo C++ help on fflush(). */
#include <stdio.h>
#include <io.h>
void flush(FILE *stream)
{
     int duphandle;
     fflush (stream);                /* flush the stream's internal buffer */
     duphandle = dup (fileno (stream));  /* make a duplicate file handle */
     close (duphandle);         /* close duplicate to flush the DOS buffer */
}
```

Subroutine `gamgev2.c`:

```
/* Generates random synthetic gamma-ray logs.  HIMIN and
 * HIMAX are chosen to make the average noise power of the
 * generated logs to be 13.  Larry Paden 10/5/83.
 * WIDMAX and initial width changed 5/17/84.  LJP
 * Gamge2 created to add random slopes between levels.  LJP 6/7/84.
 * Gamgevar to allow calling program to select width of slopes. LJP 8/5/84.
 * Gamgev2 to tidy up.  Parameters are the same, but fewer calls to erand()
 *  are made, so this will not generate the same synthetic log LJP 9/12/90.
 */
#include <math.h>
#define WIDMIN 5
#define WIDMAX 11
#define WIDE (WIDMAX-WIDMIN)
#define HIMIN 50
#define HIMAX 288
#define HIGH (HIMAX-HIMIN)
void gamgev2 (double xx[],  /* Incoming ideal log. */
    int isize,              /* Length in samples of the ideal log. */
    unsigned short iseed[], /* Seed for the random number generator. */
    double slpmin,          /* Minimum transition between levels. */
    double slpmax)          /* Maximum transition between levels. */
{
    int ii, jj;
    double erand48b(), slpwid, swidth, width, height, oldhi;
    slpwid=slpmax-slpmin;
    ii = 0;
    height = HIMIN+HIGH*erand48b(iseed);
    width = WIDMAX*erand48b(iseed);
/*  printf ("At %d W1, h1:  %g, %g\n", ii, width, height);*/
    while (ii < isize) {
        /* Generate width points on a level. */
        for (jj=ii; jj<=ii+width-1 && jj<isize; jj++) {xx[jj] = height;}
        ii = ii+width;
        oldhi = height;
        height = HIMIN+HIGH*erand48b(iseed);
        width = WIDMIN+WIDE*erand48b(iseed);
        if (slpmin+slpwid >= 0.0) {
            swidth = slpmin + (slpwid==0.0 ? 0.0 : slpwid*erand48b(iseed));
/*          printf ("At %d W2, h2:  %g, %g\n", ii, swidth, height);*/
            /* Generate swidth points on a slope. */
            for (jj=ii; jj<=ii+swidth-1 && jj<isize; jj++) {
                xx[jj] = oldhi + (jj-ii+1)*(height-oldhi)/(int)(swidth+1);}
            ii = ii+swidth;
        }
    }
}
```

Subroutine gauss.c:

```
/* Generates Gaussion RV.  From Fortran; Larry Paden 4/5/87 */
#include <math.h>
static double twopi = 0.0;
double gauss (
    unsigned short seed[],  /* Random number generator seed. */
    double s,               /* Standard deviation. */
    double am)              /* Mean value. */
{
    double erand48b(), atan(), log(), cos(), sqrt();
    if (twopi == 0.0) {
        twopi = 8.0*atan(1.0);
        printf ("Two pi is %18.17lg\n", twopi);
    }
    return (sqrt(-2.*log(erand48b(seed))) * s *
               cos(twopi*erand48b(seed))+am);
}
```

Subroutine ncproces.c:

```
/* Tells if a numeric coprocessor is found.  Larry Paden 7/30/90 */
#include <dos.h>
int ncprocessor () {
    if (_8087 > 0) {
        printf ("This program finds an 80%d87.\n", _8087);
    }
    else printf ("This program cannot find the 80x87!\n");
    return (_8087);
}
```

Subroutine poisnois.c:

```
/*  Adds Guassian noise to a synthetic gamma-ray log.  The noise
 *  variance is set to the square root of the log at each point
 *  to simulate the Poisson distribution.  Larry Paden 10/5/83.
 *  From Fortran 4/5/87 ljp. */
#include <math.h>
void poisnois (double xx[], int isize, unsigned short iseed[])
/*      double *xx;             /* An incoming synthetic log. */
/*      int isize;              /* Length of this log. */
/*      unsigned short *iseed;  /* Random number generator seed. */
{
    int ii;                     /* Array index. */
    double
        whino,                  /* Gaussian (0, 1) distributed. */
        gauss();            /* Noise generator. */
    for (ii=0; ii<isize; ii++) {
        whino = gauss (iseed, 1.0, 0.0);
        xx[ii] = whino*sqrt(xx[ii]) + xx[ii];
    }
}
```

Subroutine `randb.c`:

```
/* A simple version of the UNIX(tm) 48-bit multiplicative congruential random
 * number generator.  Takes 1.7 times longer to execute on the i486, but code
 * is considerably more readable.  Larry Paden 9/20/90.
 */
#include <math.h>
#define two_m16 ((long double)(1./65536.))
static double
    result = ((0x330eU*two_m16 + 0xabcdU) * two_m16 + 0x1234U) * two_m16,
    a2 = 0x0005U*65536.*65536., a1 = 0xdeecU*65536., a0 = 0xe66dU,
    carry = 0xb * two_m16 * two_m16 * two_m16;
double drand48b() {
    double b2, b1, b0, intprt;
    modf (a2*result, &intprt);
    b2 =  a2*result-intprt;         /* Fractional part of 16 by 48 bits. */
    modf (a1*result, &intprt);
    b1 =  a1*result-intprt;         /* Fractional part of 16 by 48 bits. */
    modf (a0*result+carry, &intprt);/* Works if carry < 2^16. */
    b0 =  a0*result+carry-intprt;   /* Fractional part of 16 by 48 bits. */
    result = modf (b2+b1+b0, &intprt);  /* Works if sum b[0-2] <= 2.0. */
    return (result);
}
double erand48b (unsigned short iseed[]) {
    a2 = iseed[2]*65536.*65536.; a1 = iseed[1]*65536.; a0 = iseed[0];
    return (drand48b());
}
```

Subroutine `reload.c`:

```
/* Checks the length of dataseed, datanois, and dataxxx files.
    If they are within one record of being the same length, the
    seed is reloaded and the return value is set accordingly.  The other
    files are not truncated since it is easier to seek to a given position
    and write than it is to truncate.  Larry Paden 4/8/87. */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
reload (
    FILE *luwrite,          /* LU for outputing text messages. */
    double bottom,          /* Least file (of form dataX.XX.) */
    double top,             /* Greatest file (of form dataX.XX.) */
    double delta,           /* Increment from bottom to top. */
    unsigned short iseed[]) /* Random number generator seed. */
{
```

```
int access(), stat(), open(), close(), seedrecs, records, errtmp;
double param;
char *tmp, filename[20];
FILE *fp;
struct stat statbuf;
if (access (tmp="dataseed", 06) == 0) {
    stat (tmp, &statbuf);
    seedrecs = statbuf.st_size/(sizeof(short)*3);
    if (seedrecs <= 0) {
        fprintf (luwrite, "Reload:  empty seed. %d\n",statbuf.st_size);
        exit (-1);
    }
    if ((fp = fopen ("dataseed", "r")) == NULL) {
        fprintf (luwrite, "Reload:  cannot open dataseed.\n");
        exit (-2);
    }
    if (fseek (fp, (seedrecs-1)*sizeof(short)*3, 0) != 0) {
        fprintf (luwrite, "Reload:  cannot seek on dataseed.\n");
        exit (-3);
    }
    if (fread ((char *)iseed, sizeof(short), 3, fp) != 3) {
        fprintf (luwrite, "Reload:  read wrong number of items.\n");
        exit (-4);
    }
    if (fclose (fp) != 0) {
        fprintf (luwrite, "Reload:  cannot close dataseed.\n");
        exit (-5);
    }
}
else {
    return (0);      /* No previous data. */
}
if (access (tmp="datanois", 06) == 0) {
    stat (tmp, &statbuf);
    records = statbuf.st_size/sizeof(float);
}
else {
    records = 0;
}
if (records>seedrecs || records<seedrecs-1) {
    fprintf (luwrite, "Reload failed:  datanois %d %d\n",
        records, seedrecs);
    exit(-6);
}
for (param=bottom; param<=top; param=param+delta) {
    if ((errtmp =sprintf (filename, "data%.2f", param)) != 8) {
        fprintf (luwrite, "Reload:  stopped at sprintf %d\n", errtmp);
        exit(-7);
    }
    if (access (filename, 06) == 0) {
        stat (filename, &statbuf);
        records = statbuf.st_size/sizeof(float);
        if (records>seedrecs || records<seedrecs-1) {
            fprintf (luwrite, "Reload failed on %s: %d %d\n",
                    filename, records, seedrecs);
                exit(-8);
            }
        }
        else {
            fprintf (luwrite, "Reload:  cannot access %s.\n", filename);
            exit (-9);
        }
    }
    if (seedrecs <= 0) seedrecs = 0;
    else seedrecs -= 1;
    return (seedrecs);
}
```

Subroutine rmfilt.c:

```
/*C Recursive median filter.  Larry Paden  6/15/87.
/*C The filtered data is input and returned in xx. */
#include <stdio.h>
#include <math.h>
#define BUFLEN 51
void rmfilt (
    double xx[],        /* The data to be filtered. */
    int last,           /* The length of the data. */
    int length,         /* The length of the large window. */
    int *differ,    /* Returns the number of points changed. */
    int verbos)         /* Prints certain statistics. */
{
    int ii, jj, haflen;
    double xsort[BUFLEN];       /* Buffer for sorting. */
    haflen = length/2;
    if (length > BUFLEN) {
        printf ("Rmfilt must be passed length < %d, not %d.  Stopped.\n",
            BUFLEN, length);
        exit (-70);
    }
    *differ = 0;            /* Zero the number of differing points. */
    /* Note that if the filter is run with the first and last points
        duplicated at the beginning and end, the first haflen points will
        always remain the same, but the last point needs to be saved
        for future reference. */
    /* The data point numbered 0 remains unchanged. */
    /* For each data point between 1 and haflen-1 filter. */
    for (ii=1; ii<haflen; ii++) {
        jj = ii-haflen;
        while (jj < 0) {
            xsort[jj-(ii-haflen)] = xx[0];
            jj++;
        }
        while (jj <= ii+haflen) {
            xsort[jj] = xx[ii+jj];
            jj++;
        }
        bubble (xsort, 0, length);
        if (xx[ii] != xsort[haflen]) {
            *differ+=1;
            xx[ii] = xsort[haflen];
        }
    }
    /* For each data point, filter.  (Except for first and last haflen.) */
    for (ii=haflen; ii<last-haflen; ii++) {
        for (jj=0; jj<length; jj++) {
            xsort[jj] = xx[ii-haflen+jj];
        }
        bubble (xsort, 0, length);
        if (xx[ii] != xsort[haflen]) {
            *differ+=1;
            xx[ii] = xsort[haflen];
        }
    }
    /* Filter last haflen points. */
    for (ii=last-haflen; ii<last-1; ii++) {
        jj = ii-haflen;
        while (jj < last) {
            xsort[jj-(ii-haflen)] = xx[ii+jj];
            jj++;
        }
        while (jj < ii+haflen) {
            xsort[jj] = xx[last-1];
            jj++;
        }
        bubble (xsort, 0, length);
        if (xx[ii] != xsort[haflen]) {
            *differ+=1;
            xx[ii] = xsort[haflen];
        }
    }
    /* The last data point (numbered last-1) remains unchanged. */
    /* Clean up. */
    if (verbos == 1) {
        if (*differ > 0) {
            printf ("%d points changed.\n", *differ);
            /*format (1h , i<log10(float(*differ))+1>,' points changed.')*/
        }
        else { printf ("0 points changed.  No difference in output!"); }
    }
}
```

Subroutine `rmssub.c`:

```
/* Given real arrays X1 and X2 and range ISTART to LIMIT, this calculates
 * the RMS average of the difference (RMSAVG), and the ordinary average
 * values of the two input files (AVG1 and AVG2.)  The results are labelled
 * and printed if VERBOSE > 0.  Larry Paden 6/22/83.
 * From Fortran.  ljp 4/5/87. */
#include <math.h>
void rmssub (
    double x1[],
    double x2[],                /* Incoming arrays. */
    int istart,                 /* First point in evaluation. */
    int limit,                  /* Last point NOT in evaluation. */
    int verbose,            /* Print if > 0. */
    double *rmsavg,
    double *avg1,
    double *avg2)               /* The three outputs. */
{
    double total;
    int ii;
    total = limit-istart;
    *rmsavg = 0.0;
    *avg1 = 0.0;
    *avg2 = 0.0;
    for (ii=istart; ii<limit; ii++) {
      *rmsavg = *rmsavg + (x1[ii]-x2[ii]) * (x1[ii]-x2[ii]);
      *avg1 = *avg1+x1[ii];
      *avg2 = *avg2+x2[ii];
    }
    *rmsavg = sqrt (*rmsavg/total);
    *avg1 = *avg1/total;
    *avg2 = *avg2/total;
    if (verbose > 0) printf ("RMS = %lg; averages = %lg %lg\n",
        *rmsavg, *avg1, *avg2);
}
```

Subroutine `twaffilt.c`:

```
/*C Twin window average filter.  Larry Paden 3/29/84.
C   The filtered data is returned in XX.
    From twinfilt 9/11/90.  */
#include <math.h>
#define IARRAY 2048
#define XSMAX 51
#define XI(qq) xi[(qq)+XSMAX]
void twaffilt (
    double xx[],         /* The data to be filtered. */
    int last,            /* The length of the data. */
    int large,           /* The length of the large window. */
    double range,        /* Maximum distance from the current point */
                         /* which is allowed in the little window. */
    int *differ,     /* Returns the number of points changed. */
    int verbos)          /* Prints certain statistics. */
{
    int ii, jj, top, btm, haflen;
    double fmod(), sqrt(), fabs();
    static double xi[IARRAY+2*XSMAX],    /* Orignally xi(-50:size+51).*/
        xsort[XSMAX*2+1], value;         /* Temporary locations. */
    /* Do some parameter checking. */
    if (last > IARRAY) {
        printf ("Input array is greater than %d.\n", IARRAY);
        return;
    }
    if (large > XSMAX*2+1) {
        large = XSMAX*2+1;
        printf ("The maximum filter length is:  %d\n", XSMAX*2+1);
    }

    /* Initialize and copy the array. */
    *differ = 0;            /* Zero the number of differing points. */
    haflen = large/2;   /* Center position in XSORT. */
    for (ii=0; ii<last; ii++) XI(ii) = xx[ii];

    /* Append end points. */
    for (ii=0; ii<haflen; ii++) {
        XI(-1-ii) = XI(0);
        XI(last+ii) = XI(last-1);
    }

    /* For each data point, filter. */
    for (ii=0; ii<last; ii++) {

        /* Initialize and sort the little sort array. */
        for (jj=0; jj<large; jj++) xsort[jj] = XI(ii+jj-haflen);
        bubble (xsort, 0, large);
        for (btm=0; btm<large; btm++) {
            if (fabs(xx[ii]-xsort[btm]) <= range*sqrt(xx[ii]))
                break;
        }
        for (top=large-1; top>=0; top--) {
            if (fabs(xx[ii]-xsort[top]) <= range*sqrt(xx[ii]))
                break;
        }
        /* Calculate average of inner window. */
        value = 0.0;
        for (jj=btm; jj<=top; jj++) {
            value += xsort[jj];
        }
        value /= (double)(top-btm+1);
        /* Check for difference and copy the point. */
        /* printf ("Value:  %lg\n", value); */
        if (xx[ii] != value) {
            *differ+=1;
            xx[ii] = value;
        }
    }
    /* Clean up. */
    if (verbos == 1) {
        if (*differ > 0) {
            printf ("%d points changed.\n", *differ);
            /*format (1h , i<log10(float(*differ))+1>,' points changed.')*/
        }
        else { printf ("0 points changed.  No difference in output!"); }
    }
}
```

APPENDIX B

PROGRAM TO CONSOLIDATE THE DATA

The program to consolidate the data produced by the code in Appendix A is `avg.c`, and when compiled with the ingenious Turbo C++ provisions for expanding command line arguments, might be invoked by:

```
avg -a -p d?_??.ot
```

which would print on its standard output the statistics for each of the files matching the specification by the wildcard characters. The `-p` switch toggles printing of the value of every point and the `-a` switch toggles printing of the mean and standard deviation of the set of points represented in the file. Other switches may be used to select contiguous subsets of the file. If the switches are forgotten, the program will list them using the `-?`, or any other unrecognized switch. The program to interpret the binary files produced by the routines in Appendix A is

Program `avg.c`:

```
/* Prints out the average and std. dev. of a file.  Larry Paden  6/1/87
 * Handles multiple files.  (Use avf data* for all.)  LJP 6/3/87
 * As arguments, -x drops the first x elements and +y stops after
 *  the yth element in the file and gives the statistics.  LJP 6/3/87
 * Converted to Turbo C++ LJP 10/10/90.
 * Bugs:  Only processes file once.  For data with mean/std~1e7 all
 * significance will be unnecessarily lost.
 */
#include <stdio.h>
#include <math.h>
#include <values.h>
#include <stdlib.h>
#include <alloc.h>
#define ULIMIT 1048576
```

```
/* Longest file length. */
#define BIG 1.0e30
#define LITTLE 1.0e-30
/* Too BIG to square! */
void main (int argc, char *argv[])
{
    FILE *tempfile;             /* Logical unit for writing text. */
    double sum, square, histlo, histhi, histstep, histmin, histmax, histtmp,
        norm (double, double, double, double);
    float xinput;
    long far *histpnt = NULL, histlen, histcnt, lngtmp;
    int errtmp, count, scnt, hi, ii, jj, start,
        avgflg, histflg, prtflg, newfile, lnnflg, process, zero;
    long stop;
    avgflg = 1;
    histflg = 0;
    histmin = MAXDOUBLE;
    histmax = - MAXDOUBLE;
    process = 1;
    if (argv[0][0] != 'a') avgflg = 0;
    if (argv[0][0] == 'p') prtflg = 1;
    else prtflg = 0;
    lnnflg = 1;
    zero = 1;
    start = 0;
    stop = ULIMIT;
    for (ii=1; ii<argc; ii++) {
        count = start;
        newfile = 1;
        if (histflg && histpnt == NULL) {
            /* Allocate the required array. */
            histlen = (histhi-histlo)/histstep + 1;
/*          printf("Available heap is, initially: %lu bytes.\n",
                farcoreleft());*/
            if ((histpnt = (long far *) farcalloc ((unsigned long)histlen,
                (unsigned long) (sizeof(*histpnt)))) == NULL) {
                    fprintf (stderr, "Histpnt is %Fp\n", histpnt);
                exit (10);
            }
/*          printf ("Available heap is: %lu bytes.\n", farcoreleft());*/
            for (histcnt=0; histcnt<histlen; histcnt++) histpnt[histcnt] = 0;
        }
        if (argv[ii][0] == '-') {
            jj=1;
            while (argv[ii][jj] != '\0') {
                if (argv[ii][jj] >= '0' && argv[ii][jj] <= '9') {
                    start = atoi (&argv[ii][jj]);
                    while (argv[ii][jj] >= '0'
                        && argv[ii][jj] <= '9'
                    /*  && argv[ii][jj] != '\0' */) {
                        jj++;
                    }
                    if (argv[ii][jj] == '\0') break;
                }
```

```
else if (argv[ii][jj] == 'a') {
    if (avgflg) avgflg = 0;
    else avgflg = 1;
}
else if (argv[ii][jj] == 'H') {
    if (histflg) histflg = 0;
    else histflg = 1;
    histlo = 0.0;
    histhi = 13.0;
    histstep = 0.25;
    hi = ii;
    jj = 2;
    histlo = atof (&argv[hi][jj]);
    while (argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.'
        || argv[hi][jj] == 'e') {
        jj++;
    }
    if (argv[hi][jj] == '\0') break;
    while (!(argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.')
        && argv[hi][jj] != '\0') {
        jj++;
    }
    if (argv[hi][jj] == '\0') break;
    histhi = atof (&argv[hi][jj]);
    while (argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.'
        || argv[hi][jj] == 'e') {
        jj++;
    }
    if (argv[hi][jj] == '\0') break;
    while (!(argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.')
        && argv[hi][jj] != '\0') {
        jj++;
    }
    if (argv[hi][jj] == '\0') break;
    histstep = atof (&argv[hi][jj]);
    while (argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.'
        || argv[hi][jj] == 'e') {
        jj++;
    }
    if (argv[hi][jj] == '\0') break;
    while (!(argv[hi][jj] >= '0'
        && argv[hi][jj] <= '9'
        || argv[hi][jj] == '.')
        && argv[hi][jj] != '\0') {
```

```
                        jj++;
                    }
                    if (argv[hi][jj] == '\0') break;
                }
                else if (argv[ii][jj] == 'l') {
                    if (lnnflg) lnnflg = 0;
                    else lnnflg = 1;
                }
                else if (argv[ii][jj] == 'p') {
                    if (prtflg) prtflg = 0;
                    else prtflg = 1;
                }
                else if (argv[ii][jj] == 'z') {
                    if (zero) zero = 0;
                    else zero = 1;
                }
                else {
                    fprintf (stderr, "%s:  Bad switch %s.\n",
                        argv[0], argv[ii]);
                    fprintf (stderr,
"Usage:  %s -alp\n\
\ta\ttoggle averaging\n\
\tH\tselect Histogram -Hlo:hi:step\n\
\tl\ttoggle line numbering\n\
\tp\ttoggle printing\n", argv[0]);
/*d toggle float/double
  u toggle short
  cXX set number of columns */
                }
                jj++;
            }
            continue;
        }
        else if (argv[ii][0] == '+') {
            stop = atoi (&argv[ii][1]);
            continue;
        }
        if (start >= stop) {
            fprintf (stderr, "%s:  %d >= %d in %s\n",
                argv[0], start, stop, argv[ii]);
            continue;
        }
        if ((tempfile = fopen (argv[ii], "rb")) == NULL) {
            fprintf (stderr, "%s:  Cannot open %s.\n",
                argv[0], argv[ii]);
            continue;
        }
        if (start != 0) {
            if ((errtmp = fseek
                (tempfile, (long)(sizeof(float)*start), 0)) != 0) {
                fprintf (stderr, "%s:  Cannot seek on %s.  %d\n",
                    argv[0], argv[ii], errtmp);
                continue;
            }
```

```
}
scnt = 0;
sum = square = 0.0;
while ((errtmp = fread ((&xinput), sizeof(xinput),
        1, tempfile)) == 1 && scnt < stop-start) {
    /* Don't bother with files with values without physical meaning.*/
    if (fabs(xinput) > BIG) {
        fprintf (stderr, "%s:  fabs(%s(%d)) ",
            argv[0], argv[ii], scnt+start);
        fprintf (stderr, "= %lg > %g\n", xinput, BIG);
        process = 0;
        break;
    }
    if (fabs(xinput) < LITTLE && xinput != 0.0) {
        fprintf (stderr, "%s:  fabs(%s(%d)) ",
            argv[0], argv[ii], scnt+start);
        fprintf (stderr, "= %lg < %g\n", xinput, LITTLE);
        process = 0;
        break;
    }
    if ((histflg || avgflg) && (zero || xinput != 0)) {
        scnt++;
        sum += xinput;
        square += (double)xinput*xinput;
    }
    if (histflg) {
        lngtmp = (long)((xinput-histlo)/histstep);
        if (lngtmp < 0) lngtmp = 0;
        if (lngtmp > histlen) lngtmp = histlen;
        histpnt[lngtmp]++;
        if (xinput < histmin) histmin = xinput;
        if (xinput > histmax) histmax = xinput;
    }
    if (prtflg) {
        if (lnnflg) {
            if (fmod((double)(count), 10.) == 0.)
                printf ("%.4d", count);
            else {
                if (newfile) {
                    printf ("%.4d", count);
                    for (jj=0; jj<fmod((double)(count),10.); jj++){
                        printf ("       ");
                    }
                }
            }
        }
        printf ("%7.5lg", xinput);
        if (fmod((double)(count), 10.) == 9.)
            printf ("\n");
        count++;
        newfile = 0;
    }
}
if (process) {
```

```
            if (prtflg) printf ("\n");
            if (!feof(tempfile) && scnt != stop-start) {
                fprintf (stderr, "%s:  data missing %s at %d. %d\n",
                    argv[0], argv[ii], start+scnt, errtmp);
                if (scnt == 0) continue;
            }
            if (avgflg) {
                if (scnt > 1) {
                    sum = sum/scnt;
                    square = sqrt((square-scnt*sum*sum)/(scnt-1));
                    fprintf (stdout, "%s %d %lg %lg\n",
                        argv[ii], scnt, sum, square);
                }
                else {
                    fprintf (stdout, "%s %d %lg\n", argv[ii], scnt, sum);
                }
            }
        }
        process = 1;
        if (prtflg) printf ("\n");
        if (fclose (tempfile) == EOF) {
            fprintf (stderr, "%s:  Cannot close %s.\n", argv[0], argv[ii]);
            continue;
        }
        if (histflg) {
            printf ("Histogram\t%s\n", argv[ii]);
            printf ("%g\t%g\t%g\tlo:hi:step\n", histlo, histhi, histstep);
            if (scnt > 1) {
                sum = sum/scnt;
                square = sqrt((square-scnt*sum*sum)/(scnt-1));
                fprintf (stdout, "%d\t%lg\t%lg\tcnt:avg:std\n",
                    scnt, sum, square);
            }
            else {
                fprintf (stdout, "%d\t%lg\t0\tcnt:avg:std\n", scnt, sum);
                square = 1.0;
            }
            printf ("%g\t%g\t\tmin:max\n", histmin, histmax);
            printf ("\nValue\tCount\tExpect\n");
            for (histcnt=0; histcnt<histlen; histcnt++) {
                histtmp = histlo+histcnt*histstep;
                printf ("%g\t%ld\t%g\n", histtmp, histpnt[histcnt],
                    scnt * (norm (sum, square, histtmp, histtmp+histstep)));
            }
        }
    }
}
```

Subroutine `erfast.c`:

```
/* Returns (2/sqrt(pi)) * integral from 0 to x of exp (-t^2) dt.  To avoid
 * loss of precision due to subtracting nearly equal numbers, erfc = 1-erf.
 * Coefficients are Hart & Cheney #5667 (18.72D).  Larry Paden 9/23/90.
 * Runs in about 70.8% of the time of its counterpart #5667 on Unix by using
 * constants, instead of arrays.  This version may also be 1 bit (out of 53)
 * more accurate.  LJP 9/23/90.
 * Testing Clenshaw's 20D formula from 0 to 0.5 by 0.00001 reveals a maximum
 * relative error of only 2.22e-16 and most of the time 0 relative error.
 * The trouble with #5667 is that the precision is absolute, so #5708 may be
 * used instead.  Tested from 0 to 8 by 0.001, it is off 13.8e-16 relative to
 * to function itself.  After 8, it quickly deteriorates so that by 26, it
 * only 10D relative accuracy.  Between 8 and 100, use #5725, which is faster
 * to compute anyway.  It is off by 2.2e-16 relative max.  Larry Paden 1/28/91.
 */

#include <math.h>
/* TOOBIG = sqrt (abs (ln (MINDOUBLE))) to prevent underflow. */
#define TOOBIG 26.6157

/* From C.W. Clenshaw, "Chebyshev Series for Mathematical Functions," National
   Physical Laboratory Mathematical Tables, London, 1962. */
#define NUMERATOR1(xx) ((( ((( ((\
    0.00754772803341863128783 4e0) * (xx) + \
   -0.28880513720759408492401 0e0) * (xx) + \
    0.14338384219174820557671 2e2) * (xx) + \
    0.38014031812390300824444 4e2) * (xx) + \
    0.30178278853650757780922 6e4) * (xx) + \
    0.74040714271015147008206 4e4) * (xx) + \
    0.80437363096084017283216 2e5)

/* First coefficient of DENOM1 is 1.0. */
#define DENOM1(xx) ((( ((\
                                    (xx) + \
    0.38019071395193940375346 8e2) * (xx) + \
    0.65807015545924050632693 7e3) * (xx) + \
    0.63796001732442827948712 0e4) * (xx) + \
    0.34216525792462853976900 6e5) * (xx) + \
    0.80437363096084017282626 6e5)

/* J.F. Hart, Computer Approximations, 1968, ERFC 5667, page 293. */
#define N5667(xx) ((( ((( ((\
    0.5641877825507397413087057563e0) * (xx) + \
    0.9675807882987265400604202961e1) * (xx) + \
    0.7708161730368428609781633646e2) * (xx) + \
    0.3685196154710010637133875746e3) * (xx) + \
    0.1143262070703886173606073338e4) * (xx) + \
    0.2320439590251635247384768711e4) * (xx) + \
    0.2898029329216765561127584 6e4  ) * (xx) + \
    0.1826334884229511259216899 9e4)

#define D5667(xx) ((( ((( ((\
                                        (xx) + \
    0.1714980943627607849376131193e2) * (xx) + \
    0.1371255960500622202878443578e3) * (xx) + \
    0.6617361207107653469211984771e3) * (xx) + \
    0.2094384367789539593790281779e4) * (xx) + \
    0.4429612803883682726711528526e4) * (xx) + \
    0.6089542423272443550463306 8e4  ) * (xx) + \
    0.4958827564721140714954384 22e4 ) * (xx) + \
    0.182633488422951125955764 38e4)

/* J.F. Hart, Computer Approximations, 1968, ERFC 5708, page 296. */
#define N5708(xx) ((( ((( (((\
    0.564189586761813613692546 5862e0) * (xx) + \
    0.100648589749095425355050 5591e2) * (xx) + \
    0.860827622119485951175545 307e2) * (xx) + \
    0.456261458706092630641800 311e3) * (xx) + \
    0.163176026875371469635150 913e4) * (xx) + \
    0.403226701083004973620957 28e4) * (xx) + \
    0.675821696411048588633275 86e4) * (xx) + \
    0.711366324695404987340998 6e4) * (xx) + \
    0.372350798155480672256717e4)

#define D5708(xx) ((( ((( (((\
                                        (xx) + \
    0.178394984391395565288423 8734e2) * (xx) + \
    0.153077710750362215856952 0624e3) * (xx) + \
    0.817622386304544077028250 2642e3) * (xx) + \
    0.296800490148230871642765 2719e4) * (xx) + \
    0.754247951019347575547208 583e4) * (xx) + \
    0.133493465612844573717217 317e5) * (xx) + \
    0.158025359994020425273588 457e5) * (xx) + \
    0.113151920818544054682014 43e5) * (xx) + \
    0.372350798155480654352472e4)

/* J.F. Hart, Computer Approximations, 1968, ERFC 5725, page 297. */
#define N5725(xx) ((( (((\
    0.564189583547755074125320 1704e0) * (xx) + \
    0.127536664472996595247958 5264e1) * (xx) + \
    0.501904972678426746345005 8e1) * (xx) + \
    0.616020985310963054409 06e1) * (xx) + \
    0.740974060596474179442 5e1) * (xx) + \
    0.297886562639399288 62e1)

#define D5725(xx) ((( (((\
                                        (xx) + \
    0.226052852076732696959186 6945e1) * (xx) + \
    0.939603401623505415043057 9648e1) * (xx) + \
    0.120489519278551290360340 491e2) * (xx) + \
    0.170814407474660043157109 5e2) * (xx) + \
    0.960896532719278787069 8e1) * (xx) + \
    0.336907520698275276 77e1)

double erf (register double xx)
{

    double erfc (double);
    int minusx = 0;
    if (xx < 0) {xx = -xx; minusx++;}
    if (xx > 0.5) {xx = 1.0-erfc(xx);}
    else {
        register double x2 = xx * xx;
        xx *= M_2_SQRTPI * NUMERATOR1 (x2) / DENOM1 (x2);
    }
    return (minusx ? -xx : xx);
}
double erfc (register double xx)
{
    if (xx < 0.5) return (1 - erf(xx));
    if (xx >= TOOBIG) return (0.0); /* exp (-xx^2) will underflow. */
    if (xx < 8.0) return (exp (-xx*xx) * N5708 (xx) / D5708 (xx));
    return (exp (-xx*xx) * N5725 (xx) / D5725 (xx));
}
```

Subroutine norm.c:

```
/* The nunit and norm functions.  Larry Paden 9/21/87  MS-DOS ljp 9/21/90. */
#include <stdio.h>
#include <math.h>
#define ABS(xx) ((xx>=0)?(xx):-(xx))
double erf (double), erfc (double);
double nunit (double x1, double x2)
/* Calculates the area under the standard normal curve, avoiding loss of
   precision except where both arguments are abs(x[12]) < 0.7.  This is
   not currently a problem. Note that the returned value is always
   positive notwithstanding the definition of erf(3C) and reguardless of
   the order of the arguments. */
{
    if ((x1>0) != (x2>0)) {     /* If signs are different */
        return ((erf(ABS(x2)/M_SQRT2) + erf(ABS(x1)/M_SQRT2))/2.0);
    }
    else {                      /* Signs are same */
        /* Need test here for both absolute values <0.7. */
        return ((ABS (erfc(ABS(x1)/M_SQRT2) - erfc(ABS(x2)/M_SQRT2)))/2.0);
    }
}
double norm (double mu, double sigma, double aa, double bb)
/* Calculates the area under the normal curve, with mean mu, variance
sigma^2, from aa to bb.  Uses method of nunit.  Larry Paden 1/23/88. */
{
    double x1, x2;
    if (sigma<=0.0) {
        fprintf (stderr, "Norm: Sigma %g!\n", sigma);
        sigma = -sigma;
    }
    x1 = (aa-mu)/sigma;
    x2 = (bb-mu)/sigma;
    if ((x1>0) != (x2>0)) {     /* If signs are different */
        return ((erf(ABS(x2)/M_SQRT2) + erf(ABS(x1)/M_SQRT2))/2.);
    }
    else {                      /* Signs are same */
        /* Need test here for both absolute values <0.7. */
        return ((ABS (erfc(ABS(x1)/M_SQRT2) - erfc(ABS(x2)/M_SQRT2)))/2.);
    }
}
```

APPENDIX C

RESULTS OF RUNNING THE TWIN WINDOW AVERAGE FILTER

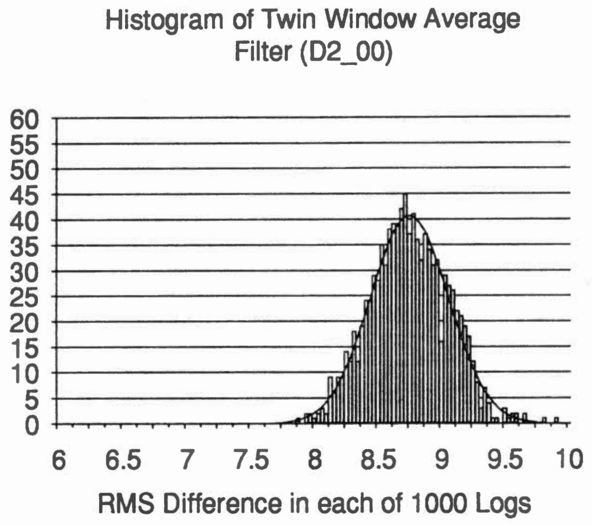
The Twin Window Average Filter was run on 1000 different synthetic logs with the filter parameter c varying between 2.0 and 4.0 by $\frac{1}{32}$. For each of these logs, the results were displayed in histograms. Care was taken to produce each of the graphs on the same scale to aid in visual comparison of the histograms. The filter parameter may be deduced from the title. For instance, on the second graph is "D2_03" which means that the parameter c was set to 2.03125 or $2\frac{1}{32}$. This was done so that the machine could handle the important steps in the process, reducing the possibility of human error.

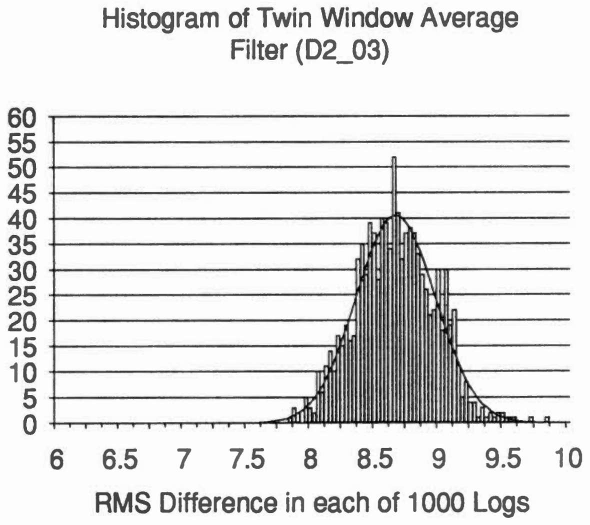
For each histogram, the mean and variance was estimated to use in computing the normal distribution. This distribution curve was then superimposed on each plot as an aid in visualizing the mean and standard deviation. It portrays graphically that the data in each histogram is very close to being normally distributed. Most importantly, the normal curves illustrate that as the filter parameter increases to its optimum value, the variance of the histogram decreases. This might provide insight as to what an appropriate value of the filter parameter should be in a practical application.

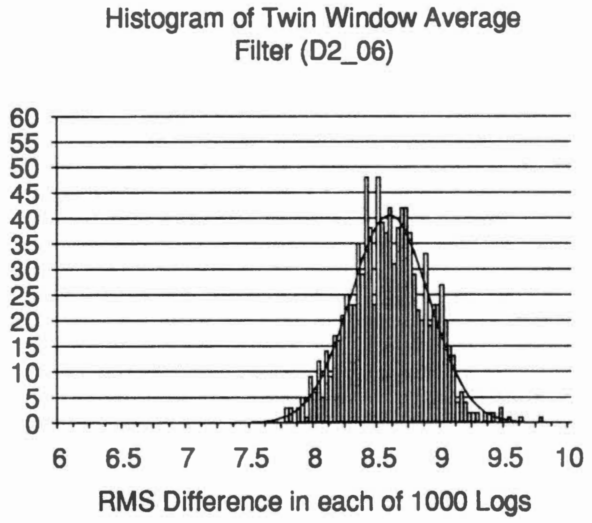
The program that summarizes the data for the histograms is given in Appendix B. To do this for a single file, it was invoked by:

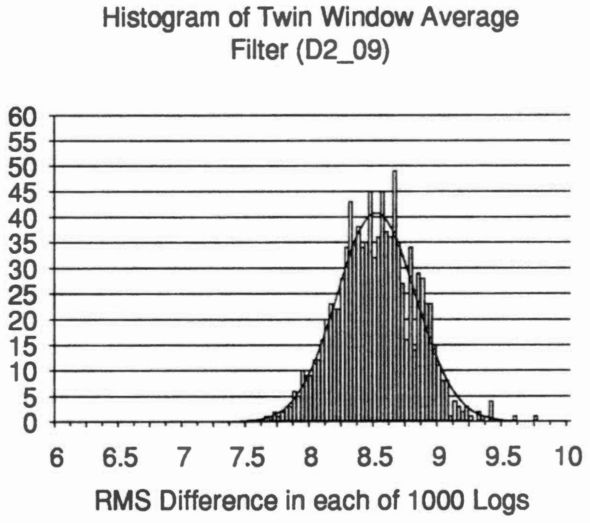
```
avg -H6.0:10.0:0.03125 d2_00.ot > d2_00.txt
```

for each of the 65 files. The program has a feature that piles any points beyond the range of the buckets into the first or last bucket, whichever is nearer. In this invocation, a point with a value of 5 would show up in the 6 bucket, or a point with a value of 22 would show up in the 10 bucket. The results are presented such that the diligent reader may verify that each histogram contains precisely 1000 points.

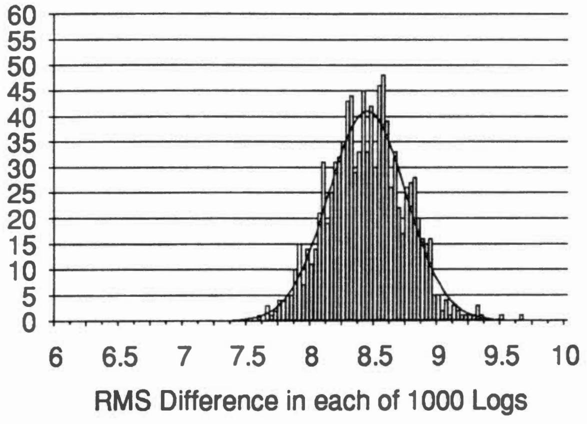




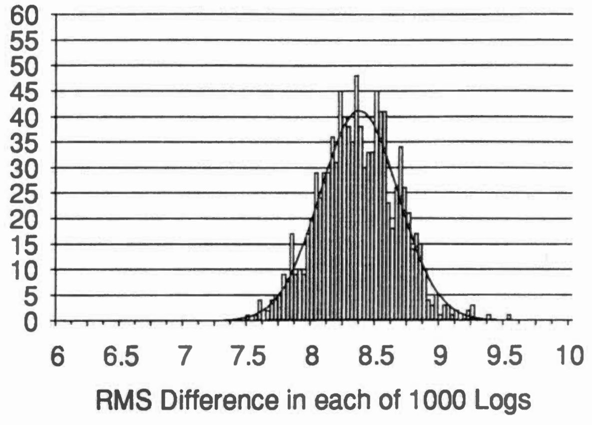




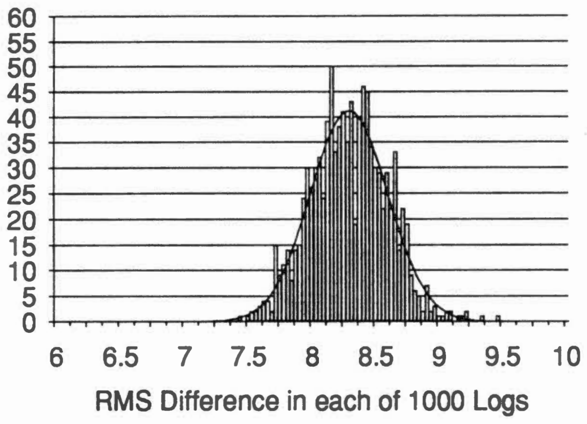
Histogram of Twin Window Average Filter (D2_12)



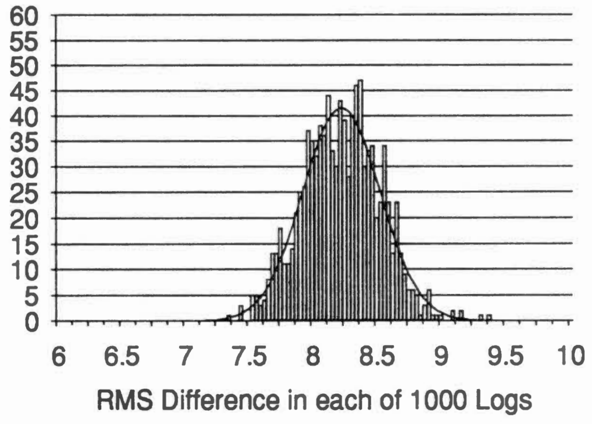
Histogram of Twin Window Average Filter (D2_15)



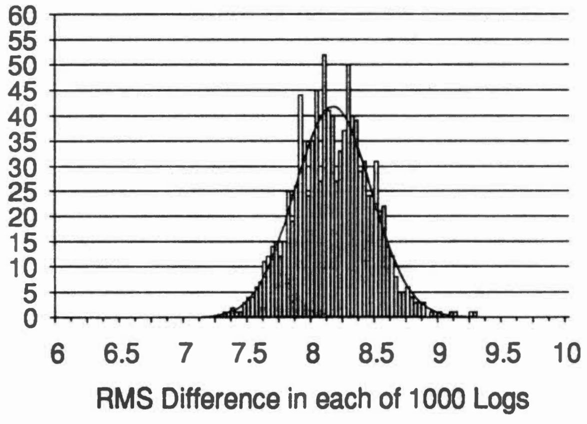
Histogram of Twin Window Average Filter (D2_18)



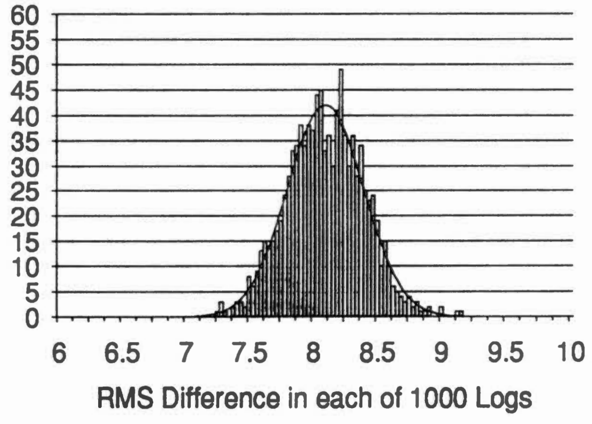
Histogram of Twin Window Average Filter (D2_21)



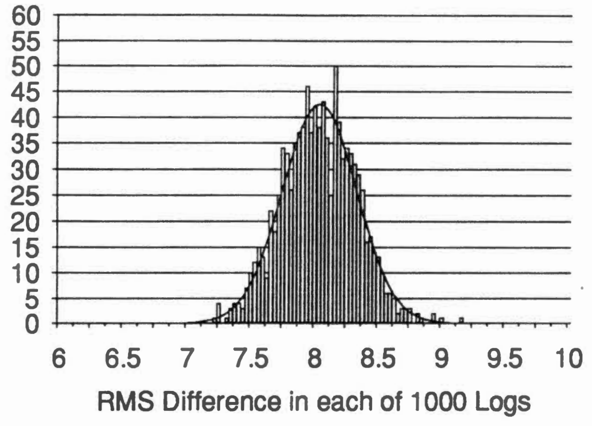
Histogram of Twin Window Average Filter (D2_25)



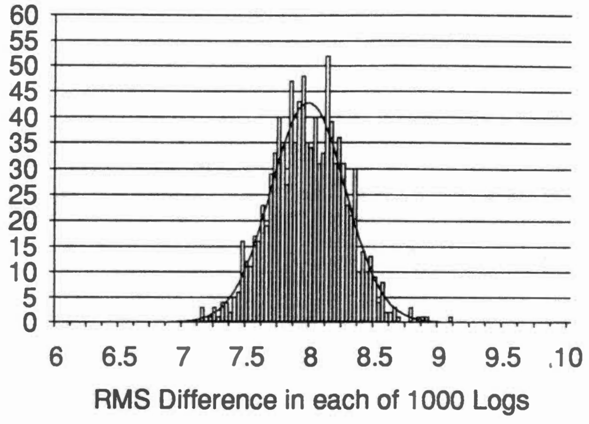
Histogram of Twin Window Average Filter (D2_28)



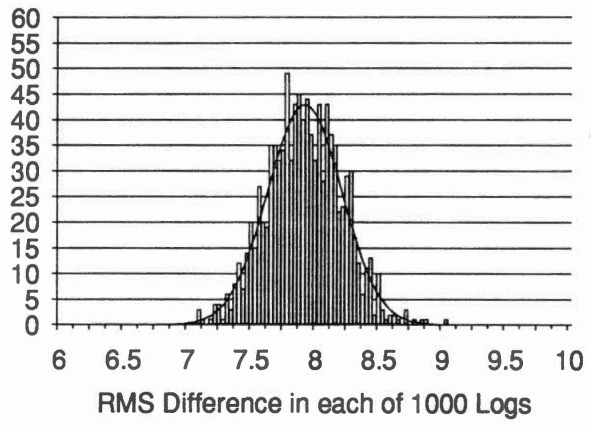
Histogram of Twin Window Average Filter (D2_31)



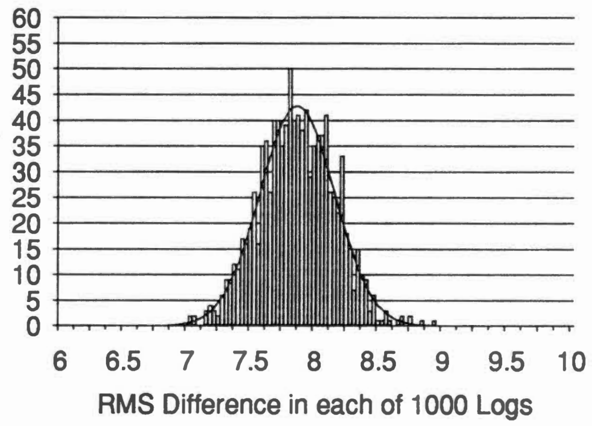
Histogram of Twin Window Average Filter (D2_34)



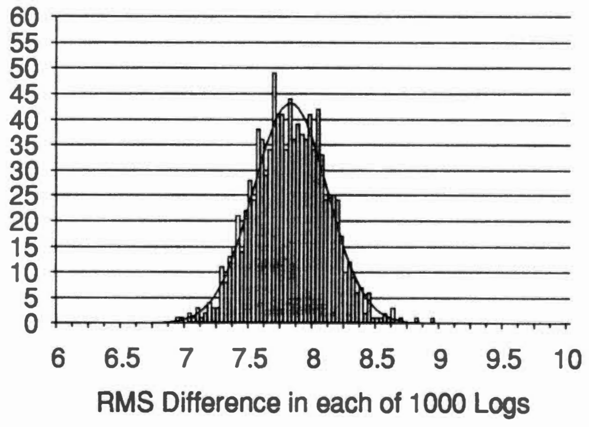
Histogram of Twin Window Average Filter (D2_37)



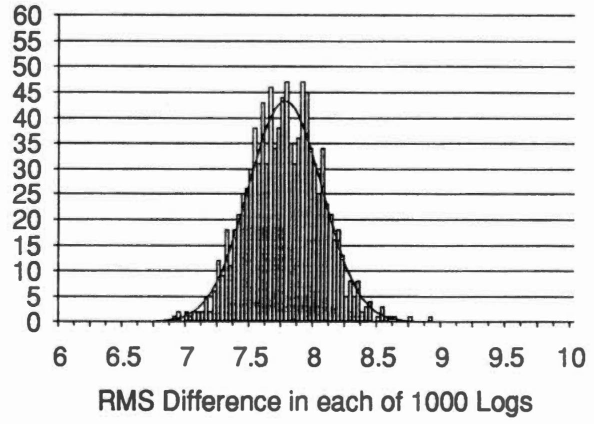
Histogram of Twin Window Average Filter (D2_40)

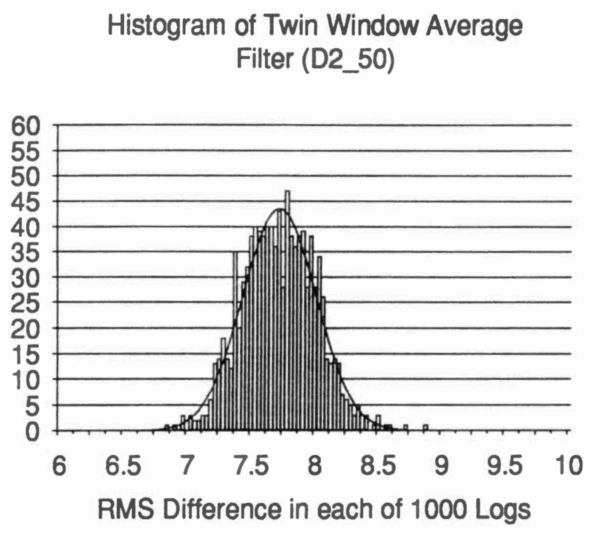


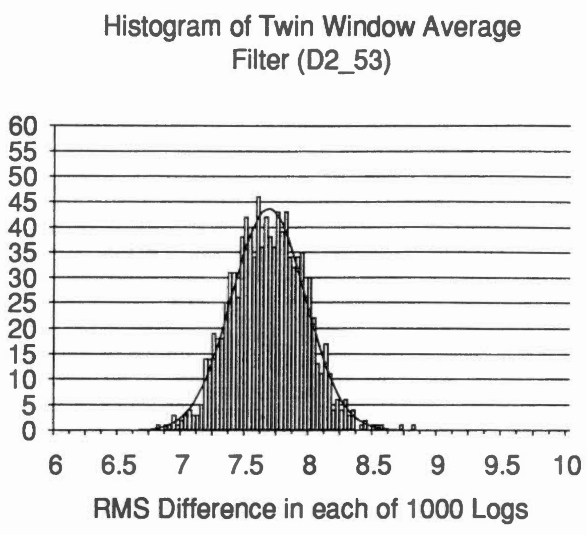
Histogram of Twin Window Average Filter (D2_43)

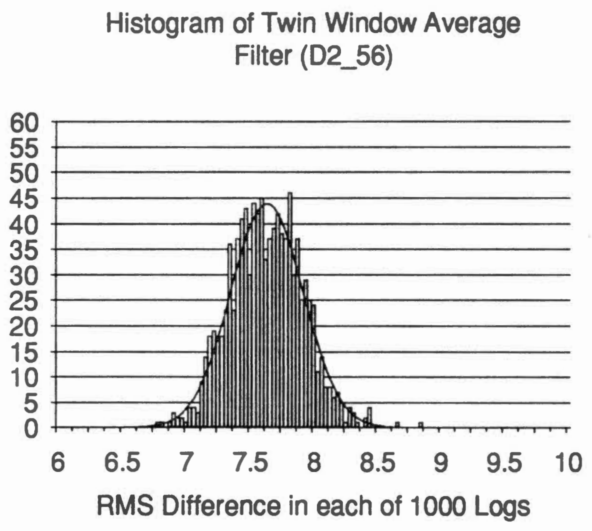


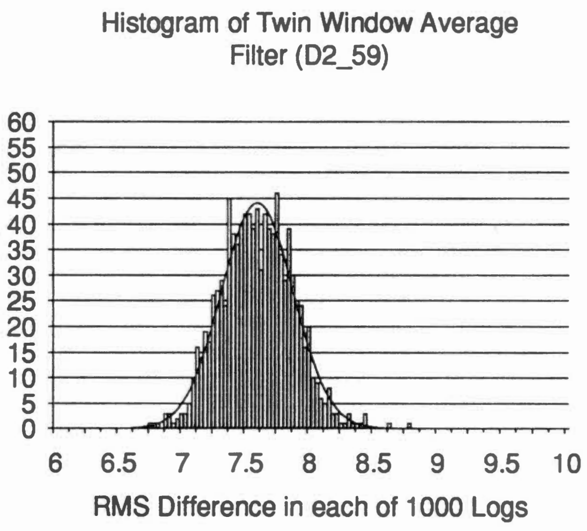
Histogram of Twin Window Average Filter (D2_46)

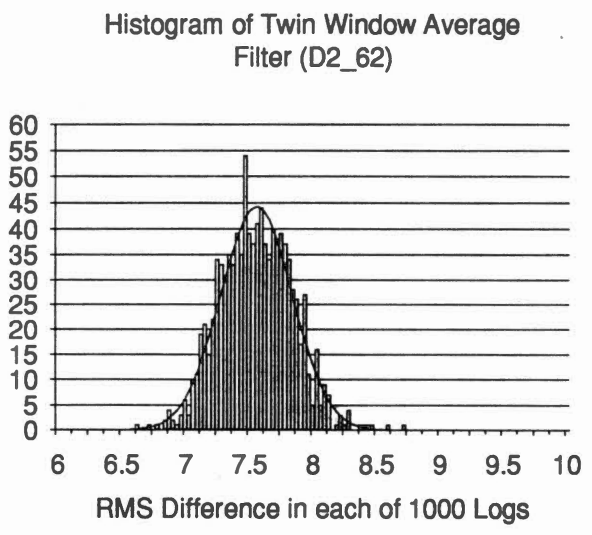


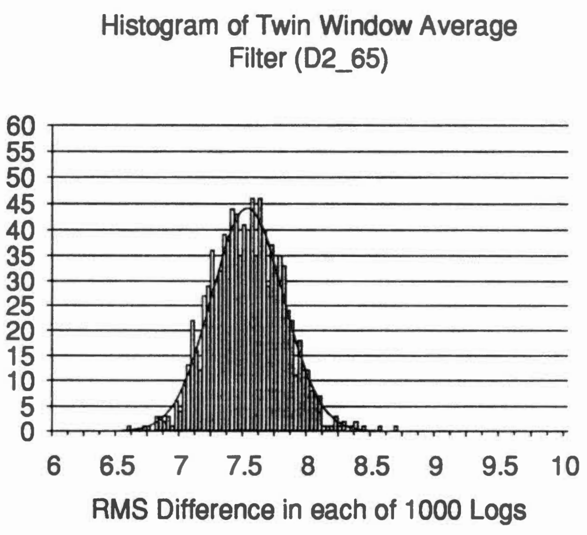

Histogram of Twin Window Average Filter (D2_50)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

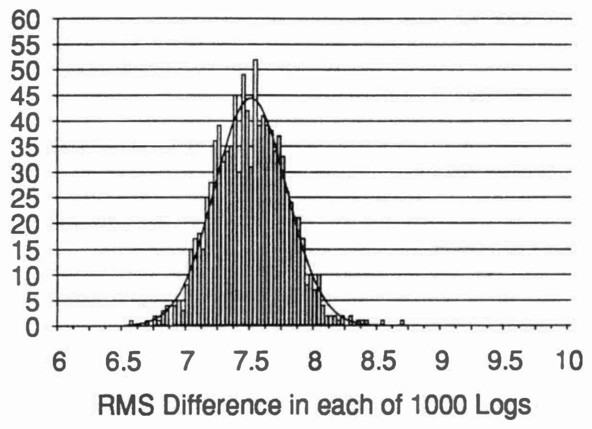

Histogram of Twin Window Average Filter (D2_53)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

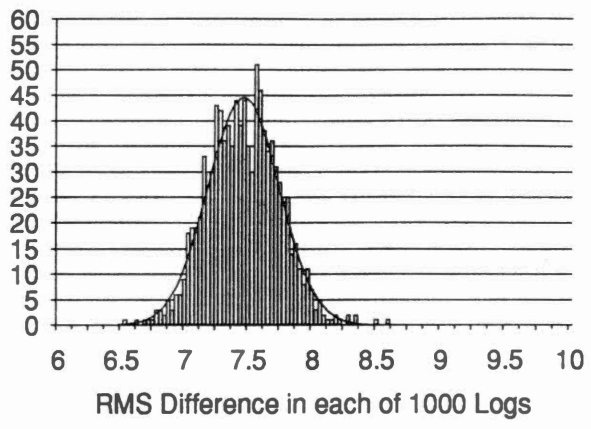

Histogram of Twin Window Average Filter (D2_56)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

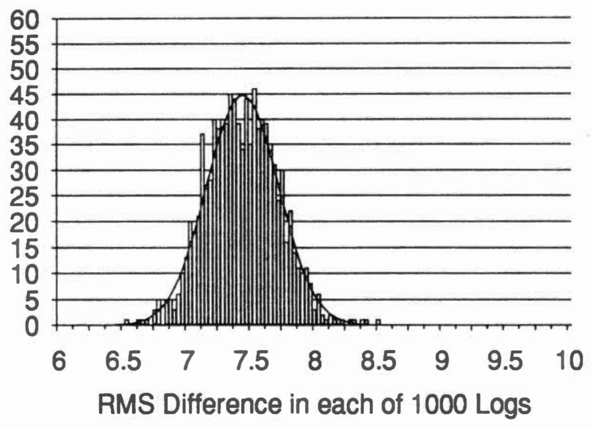

Histogram of Twin Window Average Filter (D2_59)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

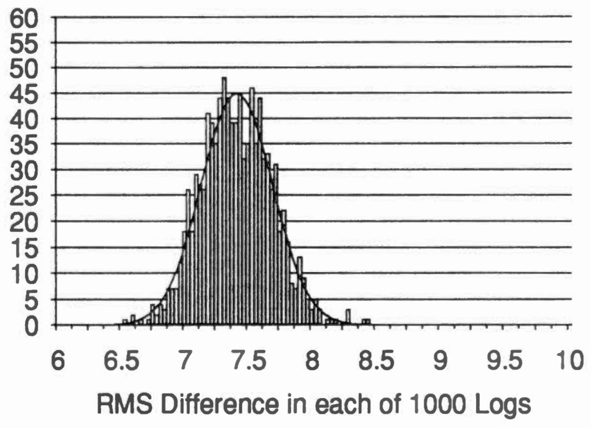

Histogram of Twin Window Average Filter (D2_62)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

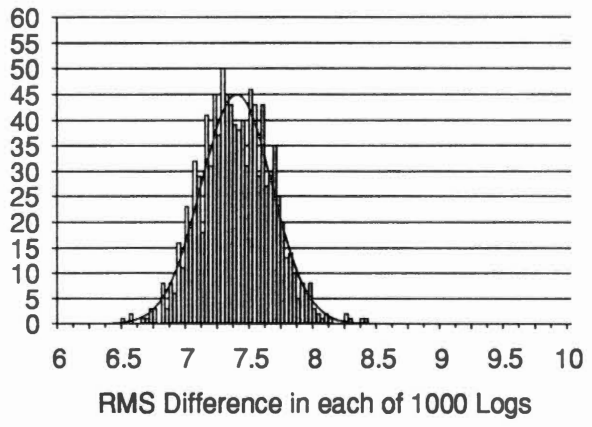

Histogram of Twin Window Average Filter (D2_65)
60 55 50 45 40 35 30 25 20 15 10 5 0
6 6.5 7 7.5 8 8.5 9 9.5 10
RMS Difference in each of 1000 Logs

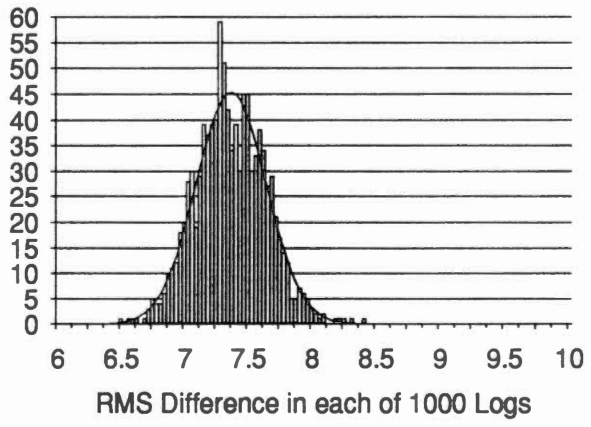

Histogram of Twin Window Average Filter (D2_68)


Histogram of Twin Window Average Filter (D2_71)

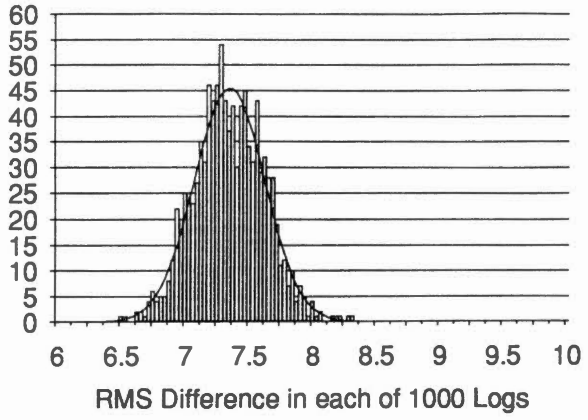

Histogram of Twin Window Average Filter (D2_75)


Histogram of Twin Window Average Filter (D2_78)

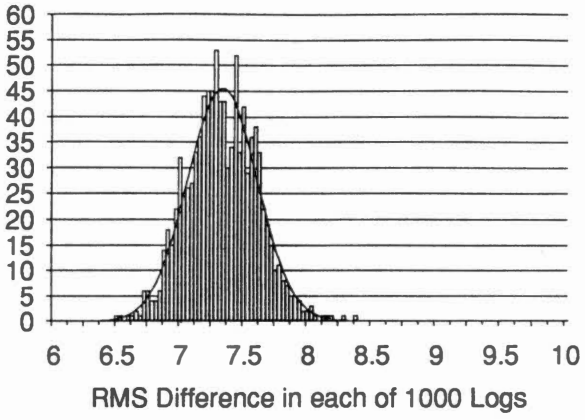

Histogram of Twin Window Average Filter (D2_81)


Histogram of Twin Window Average Filter (D2_84)

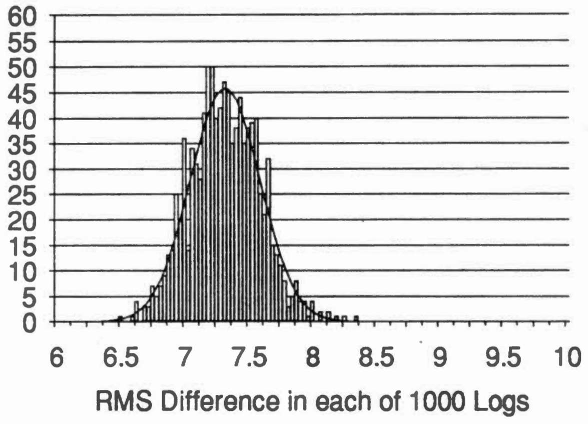
Histogram of Twin Window Average Filter (D2_87)



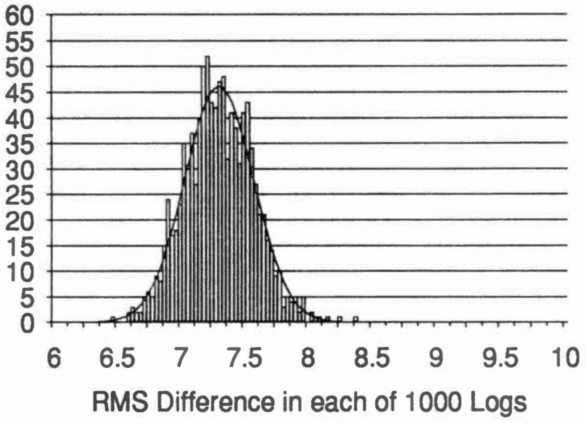
Histogram of Twin Window Average Filter (D2_90)



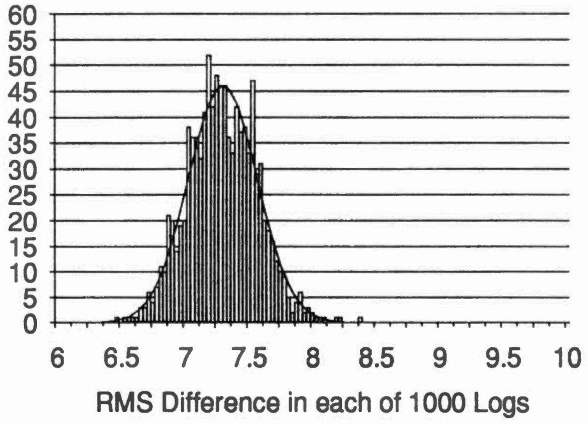
Histogram of Twin Window Average Filter (D2_93)



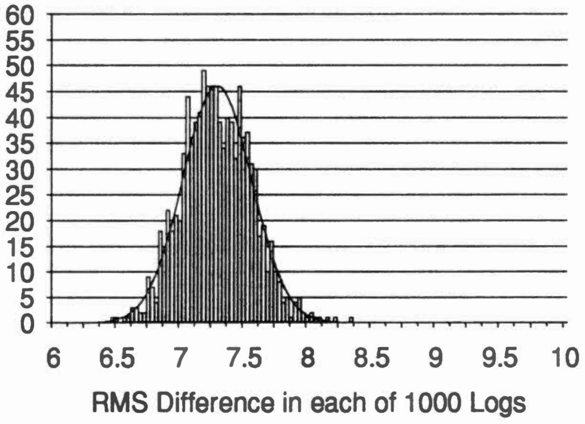
Histogram of Twin Window Average Filter (D2_96)



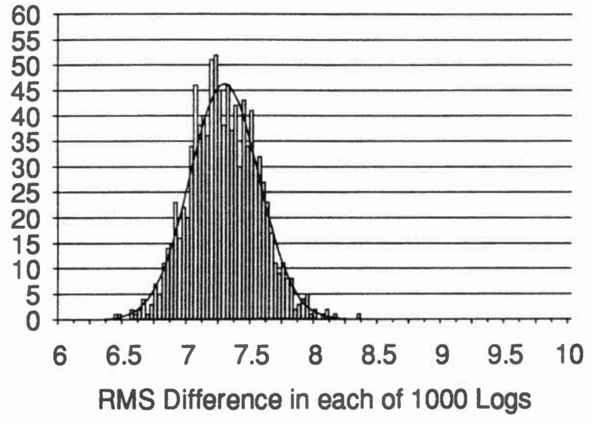
Histogram of Twin Window Average Filter (D3_00)



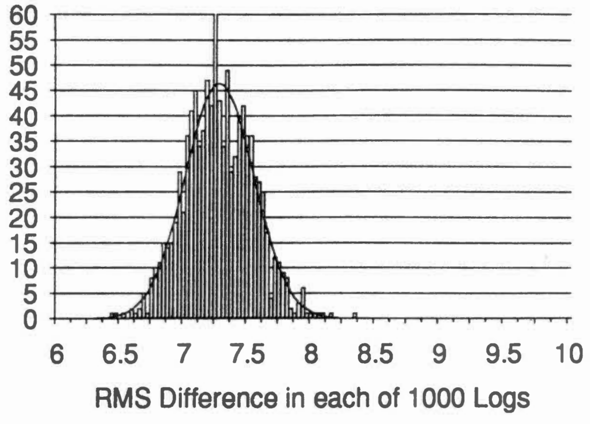
Histogram of Twin Window Average Filter (D3_03)



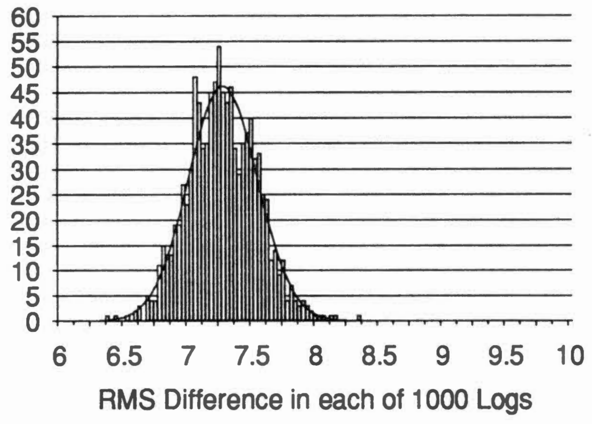
Histogram of Twin Window Average Filter (D3_06)



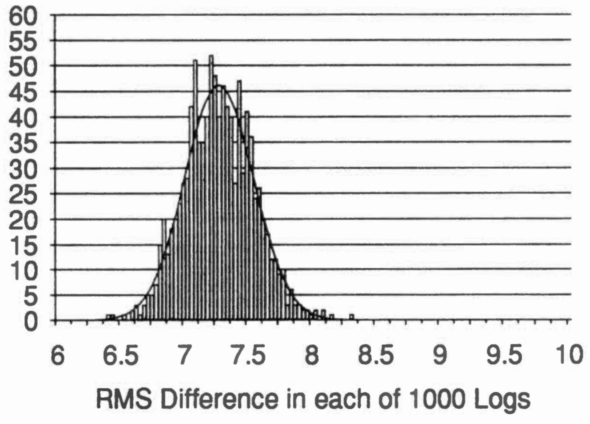
Histogram of Twin Window Average Filter (D3_09)



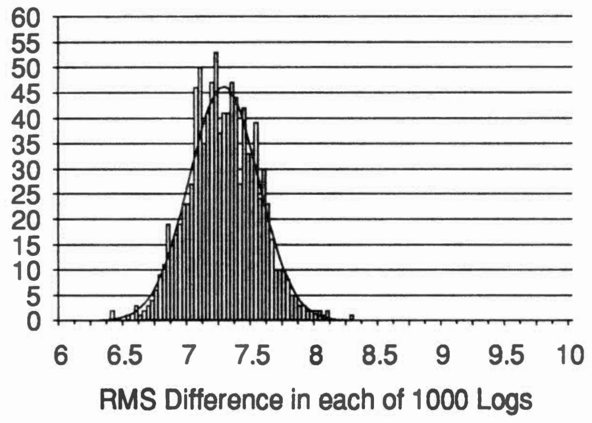
Histogram of Twin Window Average Filter (D3_12)



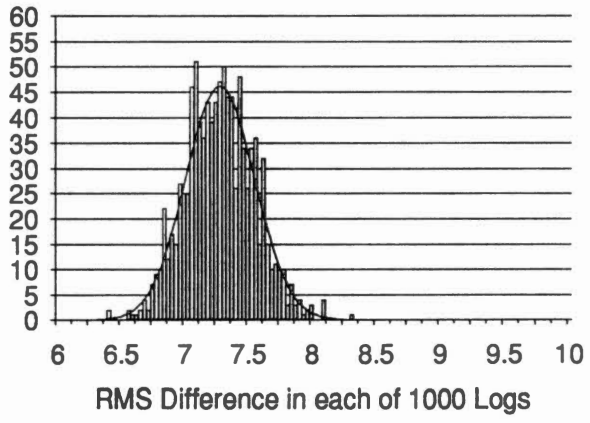
Histogram of Twin Window Average Filter (D3_15)



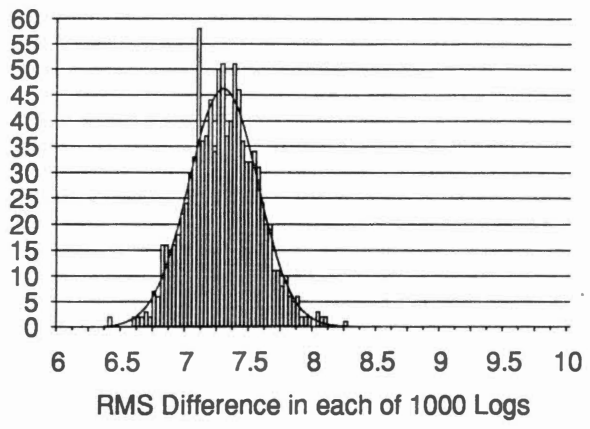
Histogram of Twin Window Average Filter (D3_18)



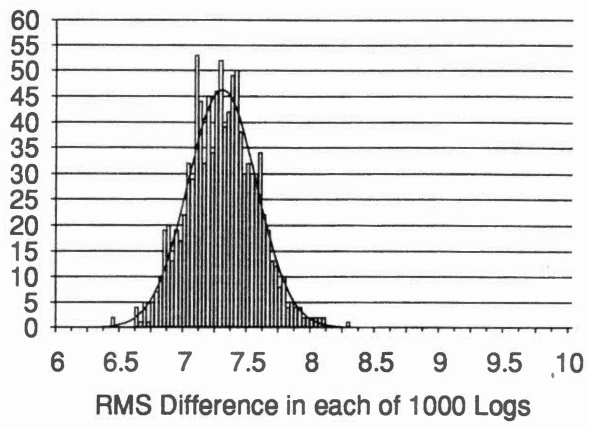
Histogram of Twin Window Average Filter (D3_21)



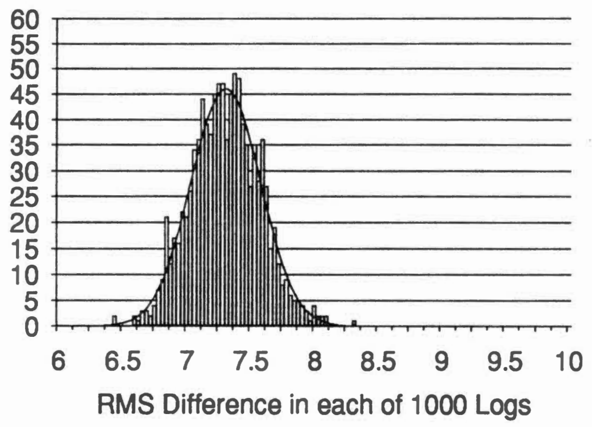
Histogram of Twin Window Average Filter (D3_25)



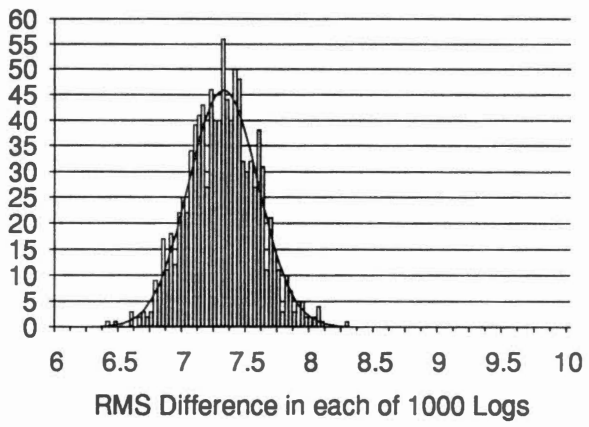
Histogram of Twin Window Average Filter (D3_28)



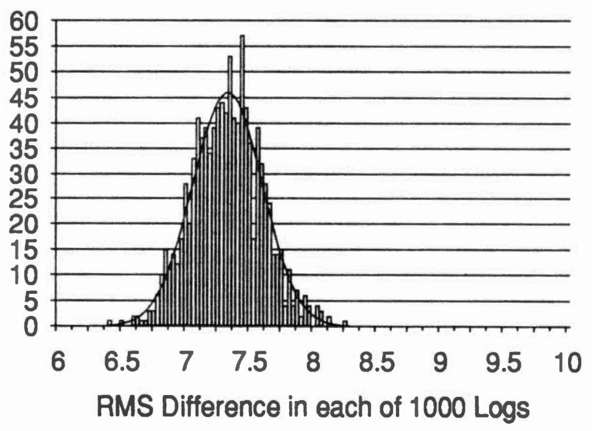
Histogram of Twin Window Average Filter (D3_31)



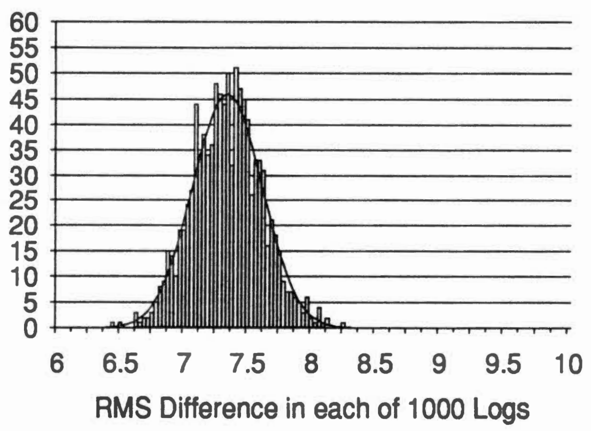
Histogram of Twin Window Average Filter (D3_34)



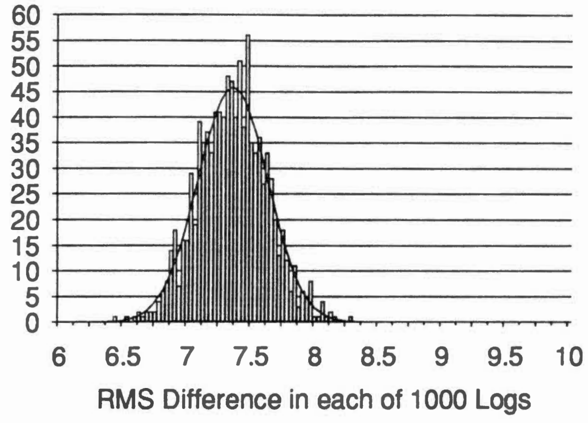
Histogram of Twin Window Average Filter (D3_37)



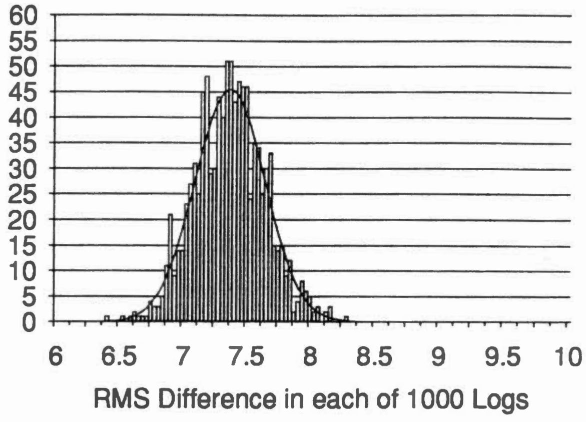
Histogram of Twin Window Average Filter (D3_40)



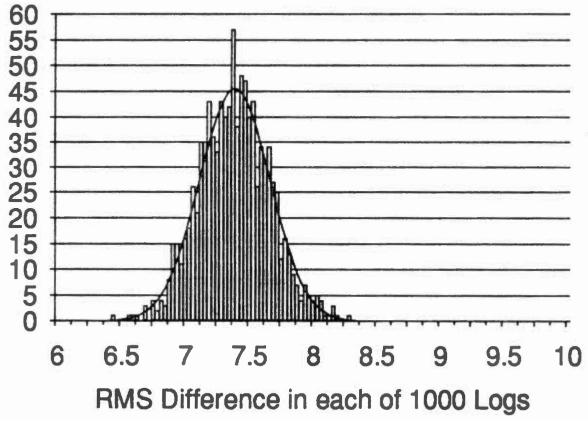
Histogram of Twin Window Average Filter (D3_43)



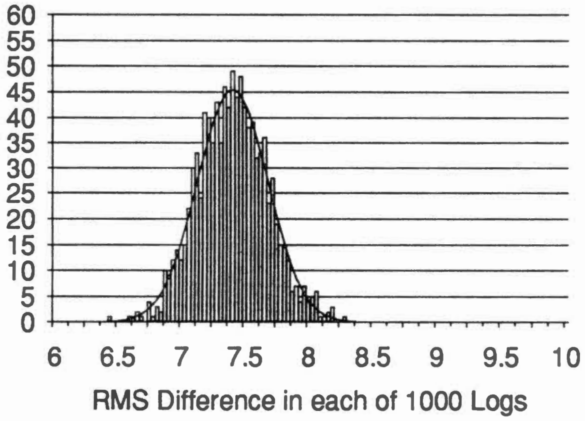
Histogram of Twin Window Average Filter (D3_46)



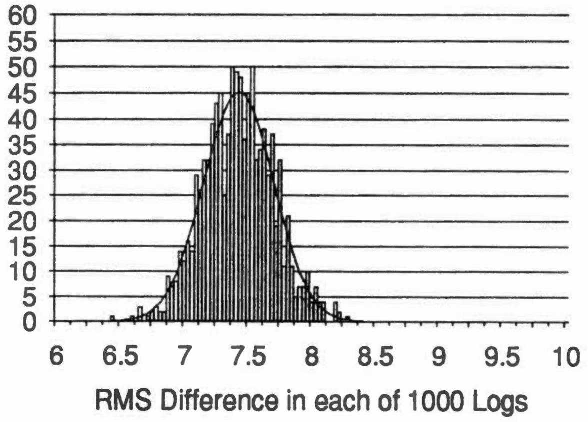
Histogram of Twin Window Average Filter (D3_50)



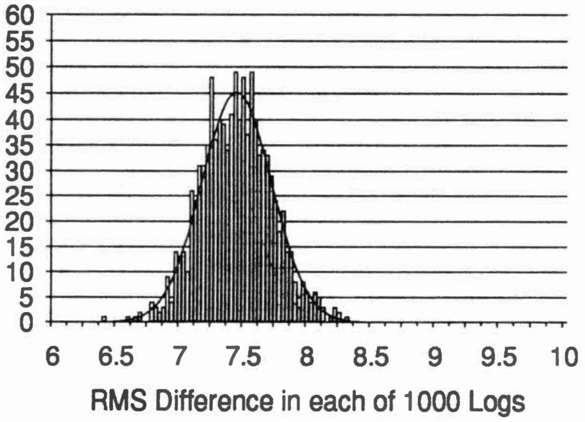
Histogram of Twin Window Average Filter (D3_53)



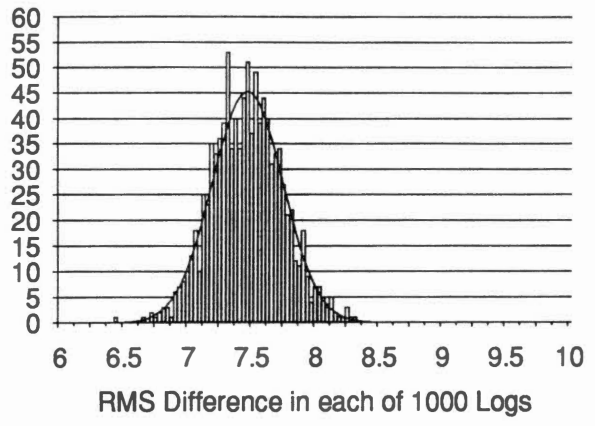
Histogram of Twin Window Average Filter (D3_56)



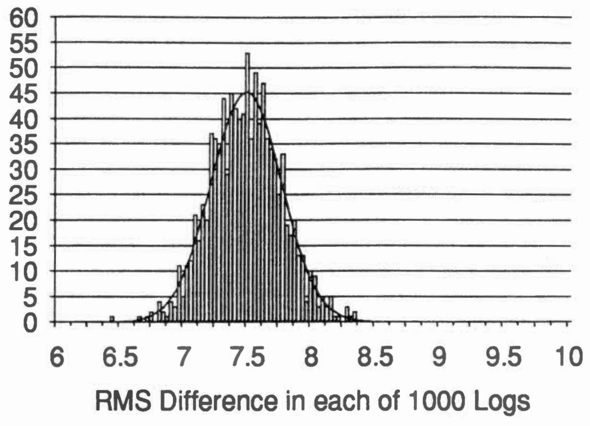
Histogram of Twin Window Average Filter (D3_59)



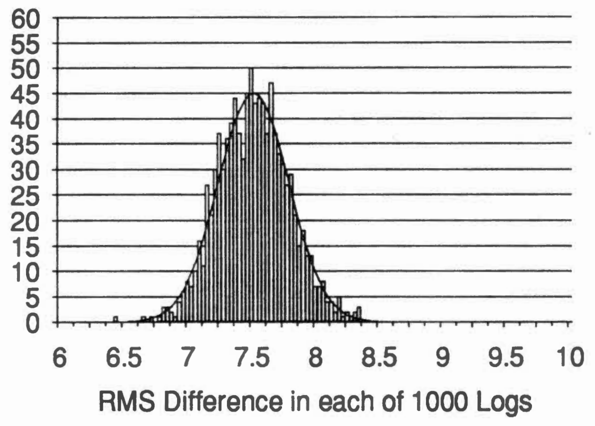
Histogram of Twin Window Average Filter (D3_62)



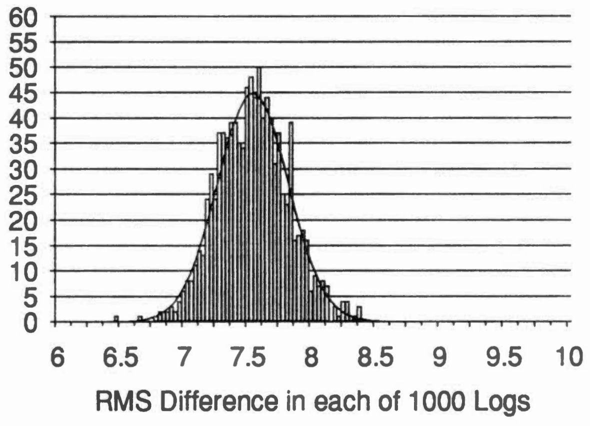
Histogram of Twin Window Average Filter (D3_65)



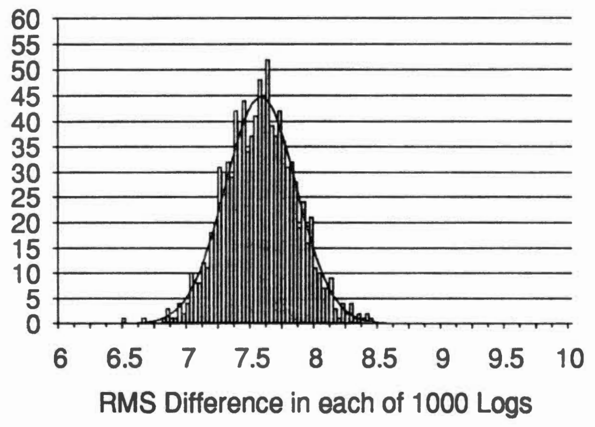
Histogram of Twin Window Average Filter (D3_68)



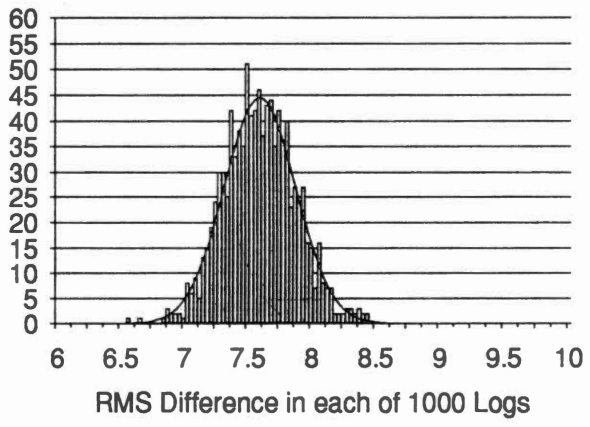
Histogram of Twin Window Average Filter (D3_71)



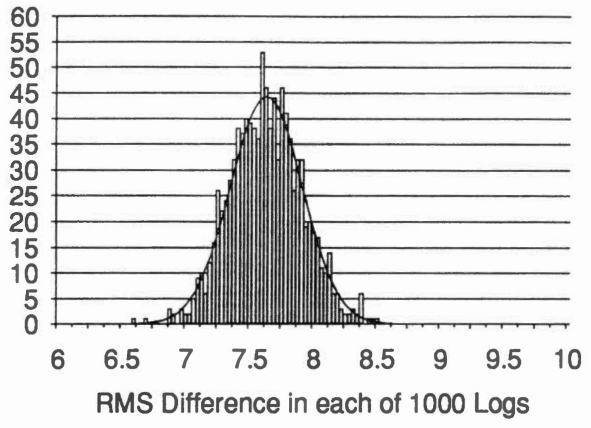
Histogram of Twin Window Average Filter (D3_75)



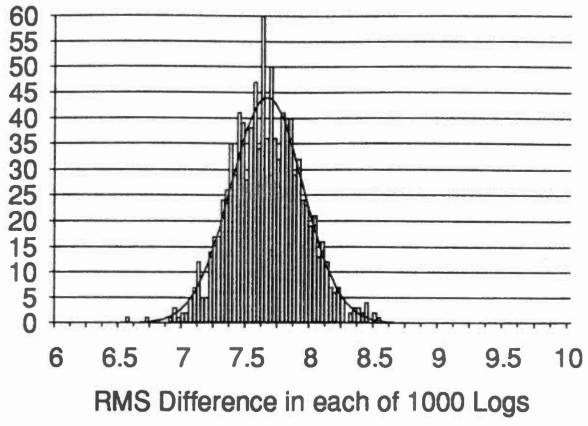
Histogram of Twin Window Average Filter (D3_78)



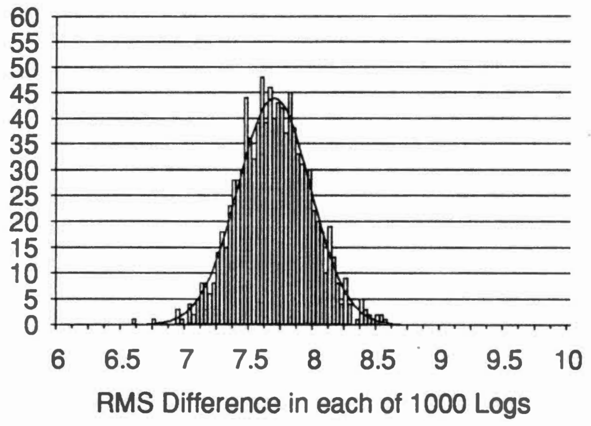
Histogram of Twin Window Average Filter (D3_81)



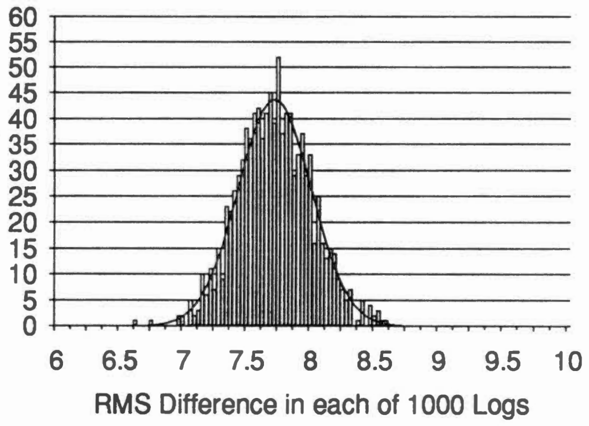
Histogram of Twin Window Average Filter (D3_84)



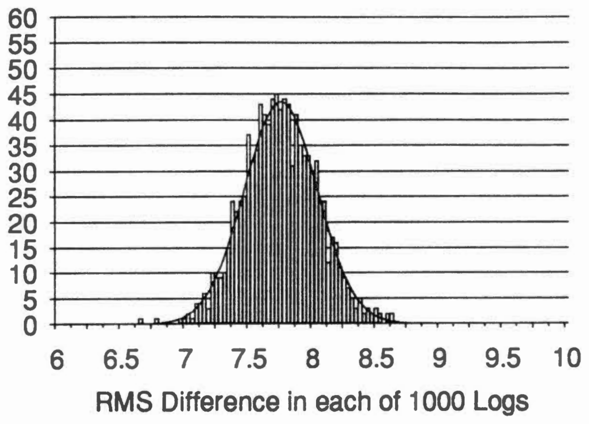
Histogram of Twin Window Average Filter (D3_87)



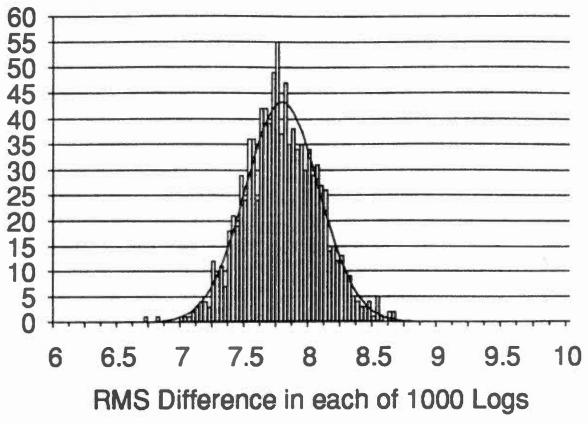
Histogram of Twin Window Average Filter (D3_90)



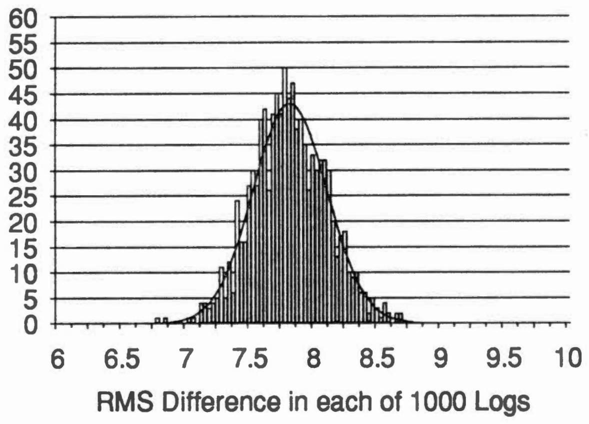
Histogram of Twin Window Average Filter (D3_93)



Histogram of Twin Window Average Filter (D3_96)



Histogram of Twin Window Average Filter (D4_00)

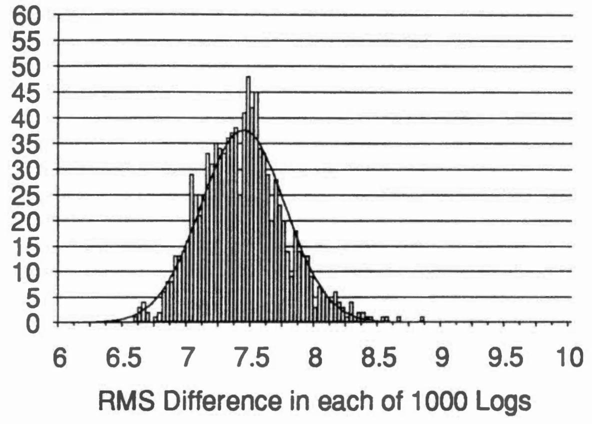


APPENDIX D

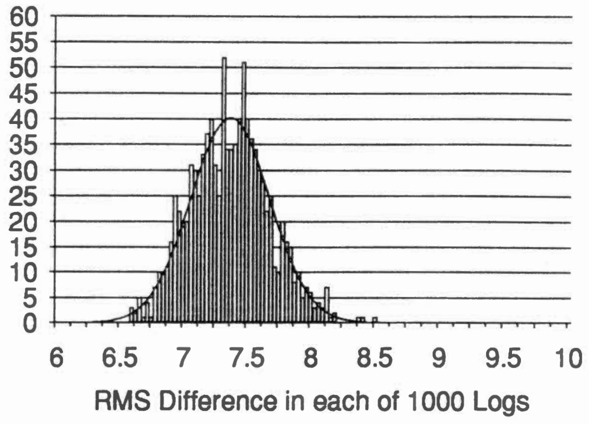
HISTOGRAMS OF A TWIN WINDOW AVERAGE FILTER FOLLOWED BY A RECURSIVE MEDIAN 3 FILTER

The main program given in Appendix A, `twaf.c`, has one subroutine commented out. The subroutine is called `rmfilt` and the parameter 3 means that it implements a recursive median filter of length 3 on the data. As can readily be determined from the program listing the RM3 is run after the TWAF. The resulting histograms have lower means than the TWAF alone, but do not match the corresponding normal curve as closely. The filter parameter for the TWAF was explained in the previous appendix. Its digits give the decimal number truncated to two places for an integral number of fractional parts whose size is $\frac{1}{32}$. For instance, D3_09 indicates 3.09 or $3\frac{3}{32}$.

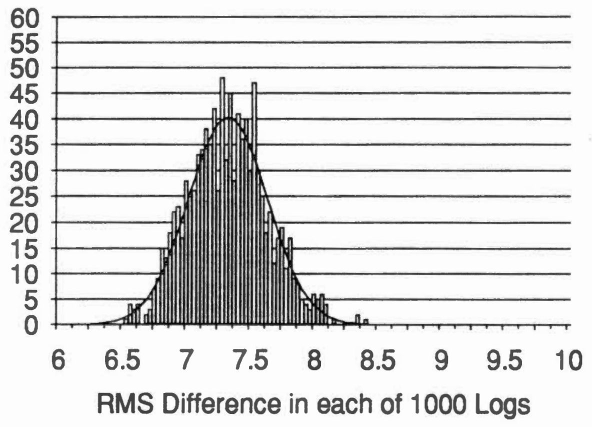
Histogram of TWAF followed by
Recursive Median 3 (D2_00)



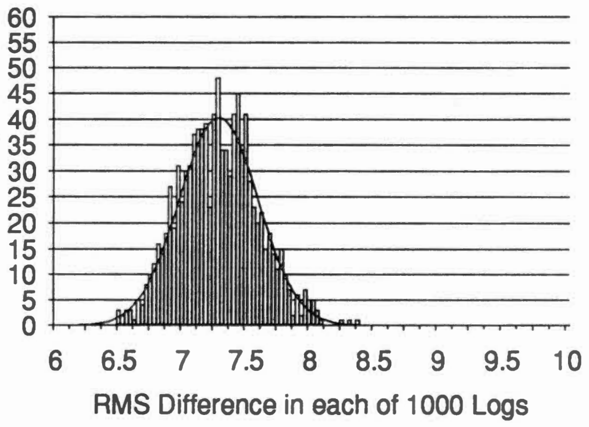
Histogram of TWAF followed by
Recursive Median 3 (D2_03)



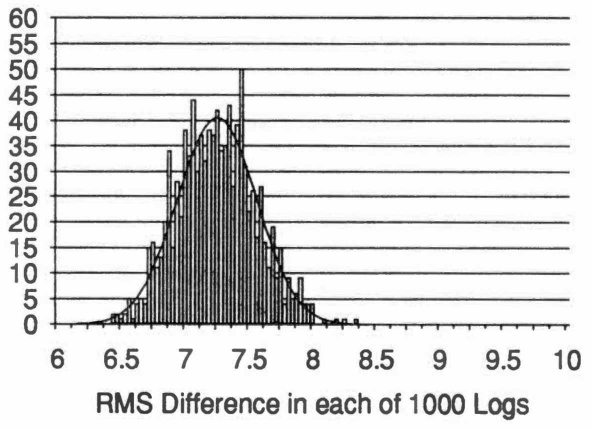
Histogram of TWAF followed by
Recursive Median 3 (D2_06)



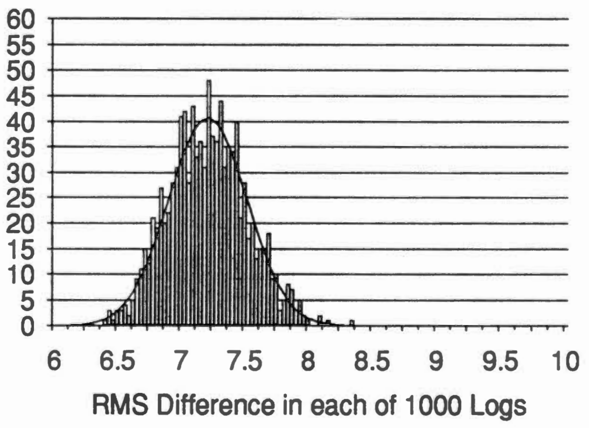
Histogram of TWAF followed by
Recursive Median 3 (D2_09)



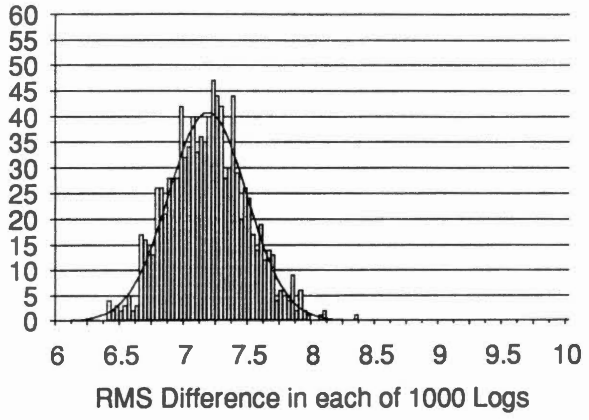
Histogram of TWAF followed by
Recursive Median 3 (D2_12)



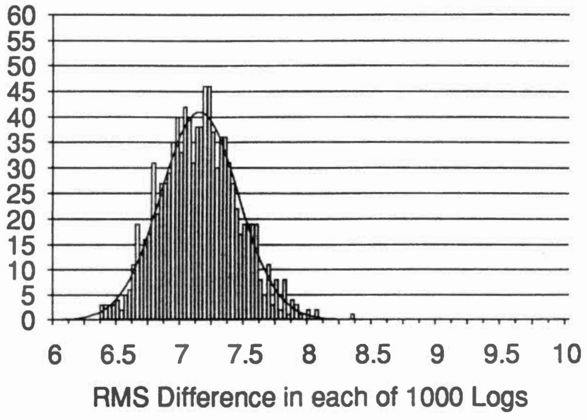
Histogram of TWAF followed by
Recursive Median 3 (D2_15)



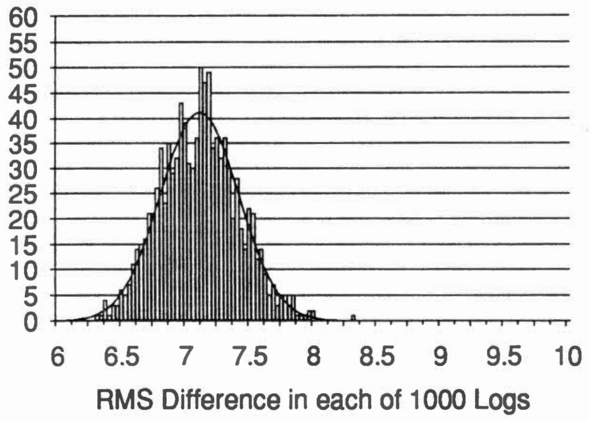
Histogram of TWAF followed by Recursive Median 3 (D2_18)



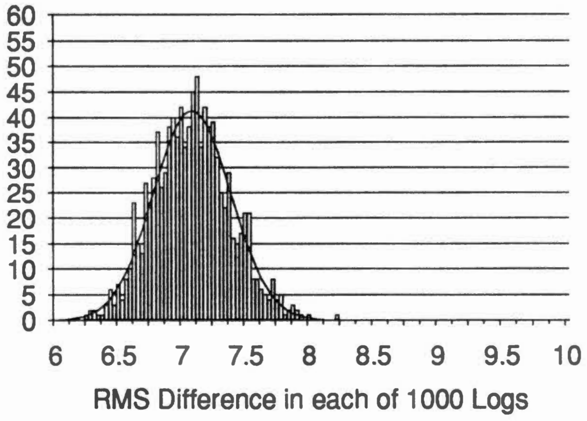
Histogram of TWAF followed by Recursive Median 3 (D2_21)



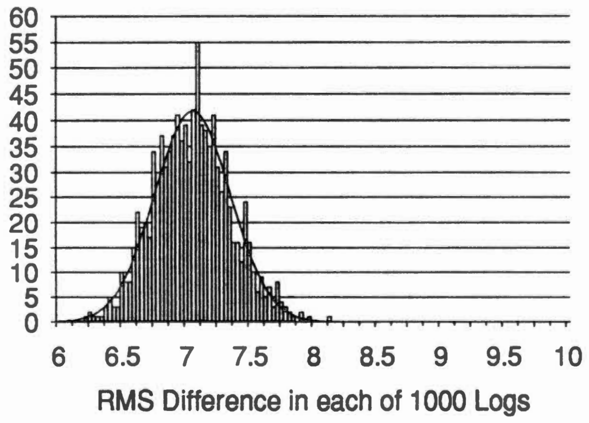
Histogram of TWAF followed by Recursive Median 3 (D2_25)



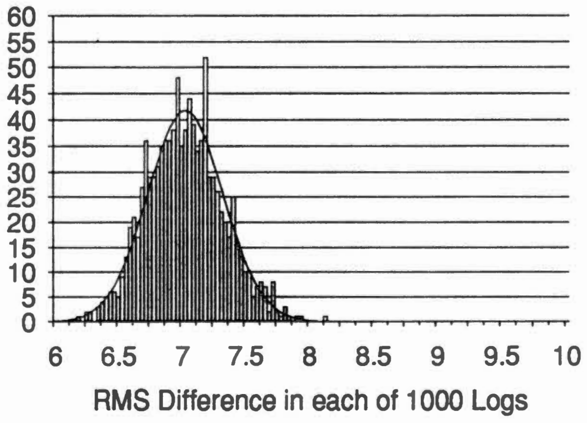
Histogram of TWAF followed by Recursive Median 3 (D2_28)



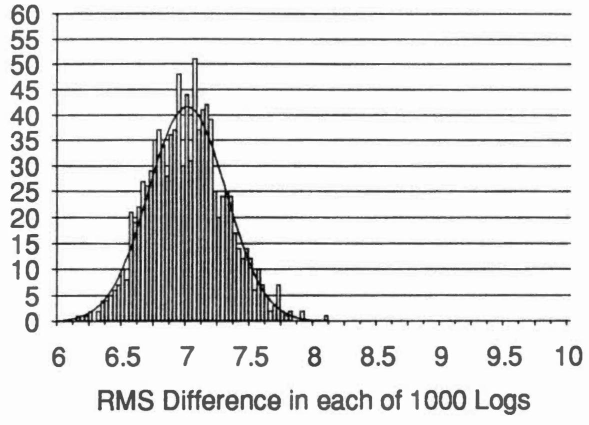
Histogram of TWAF followed by Recursive Median 3 (D2_31)



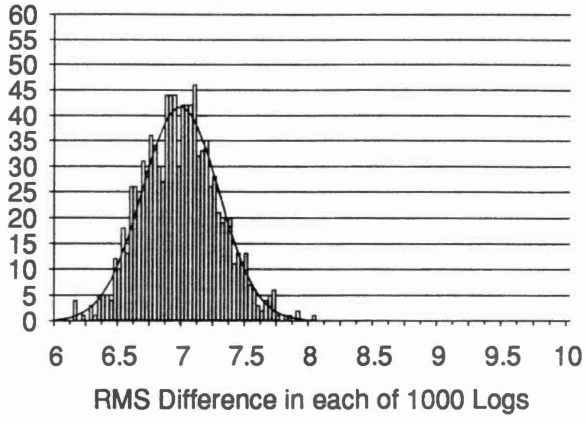
Histogram of TWAF followed by Recursive Median 3 (D2_34)



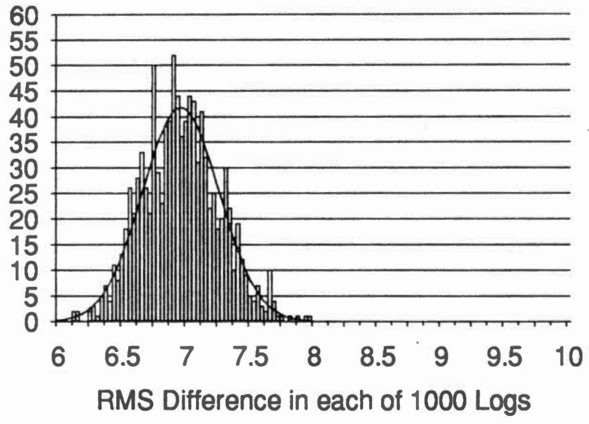
Histogram of TWAF followed by Recursive Median 3 (D2_37)



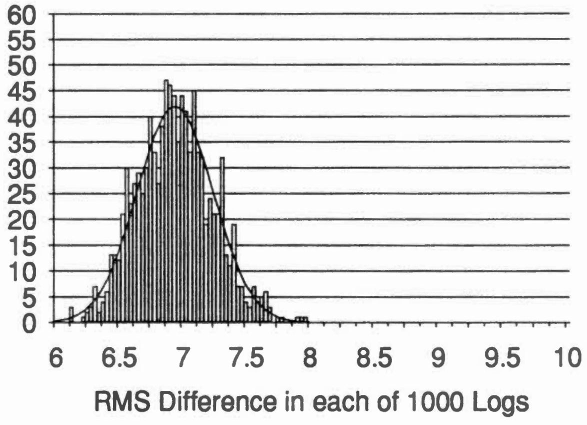
Histogram of TWAF followed by Recursive Median 3 (D2_40)



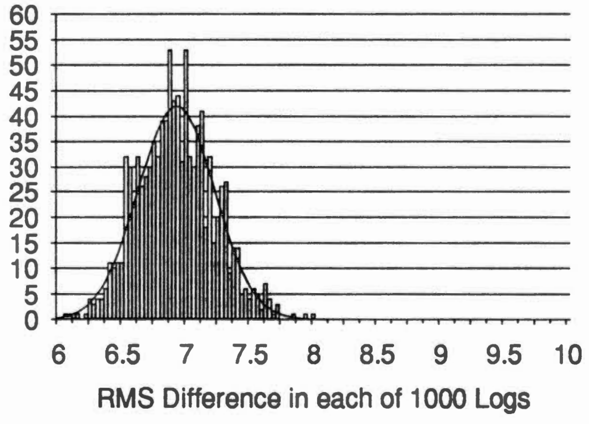
Histogram of TWAF followed by Recursive Median 3 (D2_43)



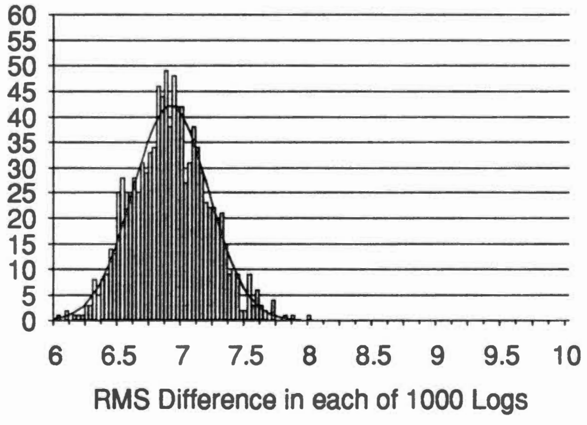
Histogram of TWAF followed by Recursive Median 3 (D2_46)



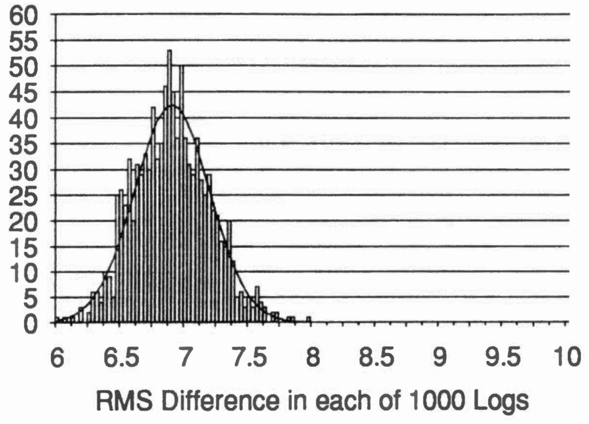
Histogram of TWAF followed by Recursive Median 3 (D2_50)



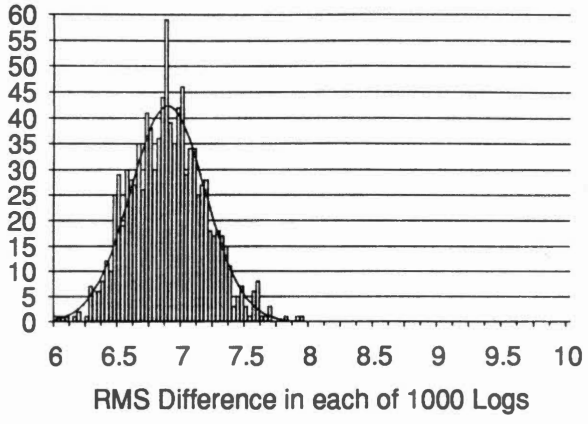
Histogram of TWAF followed by Recursive Median 3 (D2_53)



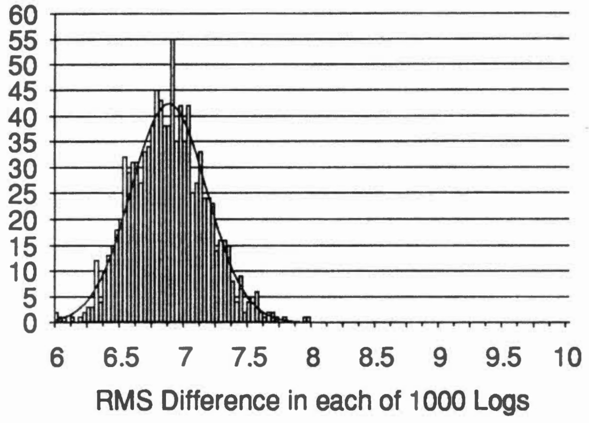
Histogram of TWAF followed by Recursive Median 3 (D2_56)



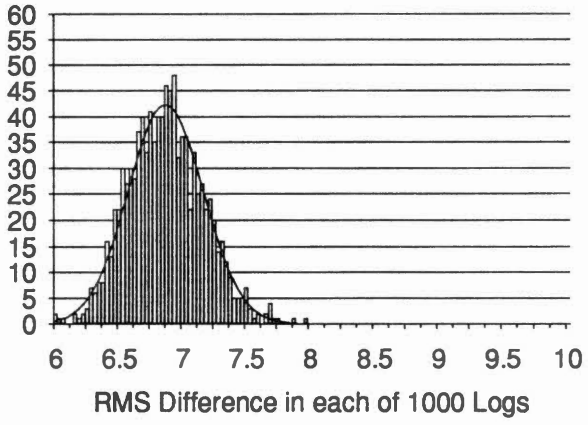
Histogram of TWAF followed by Recursive Median 3 (D2_59)



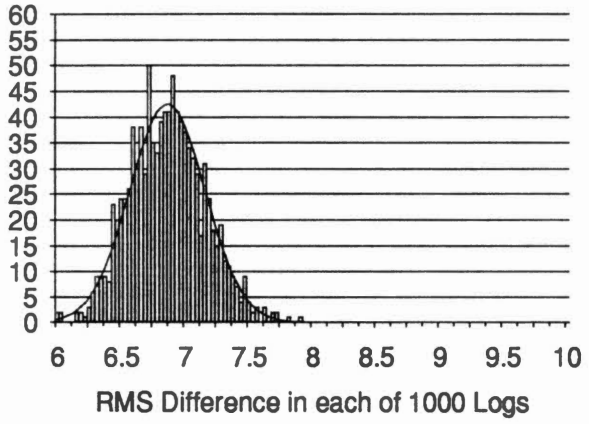
Histogram of TWAF followed by Recursive Median 3 (D2_62)



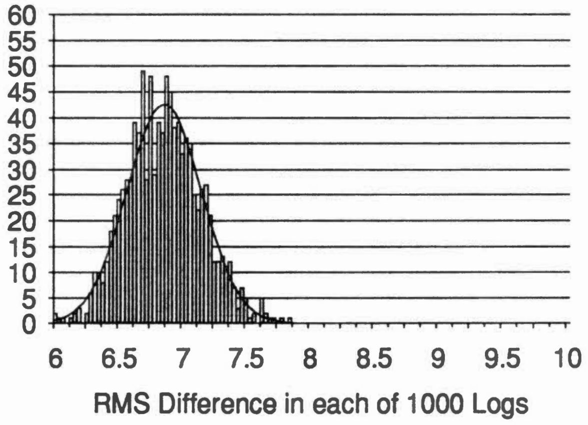
Histogram of TWAF followed by Recursive Median 3 (D2_65)



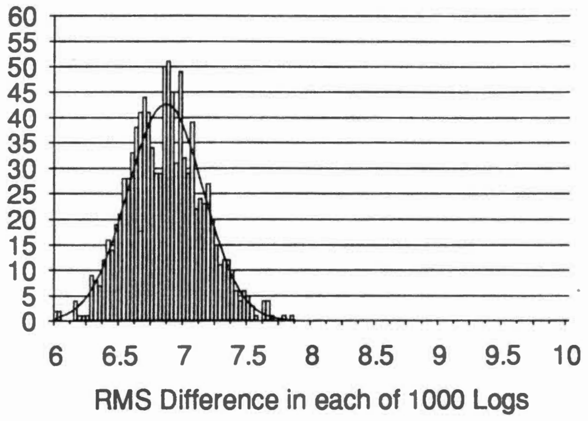
Histogram of TWAF followed by Recursive Median 3 (D2_68)



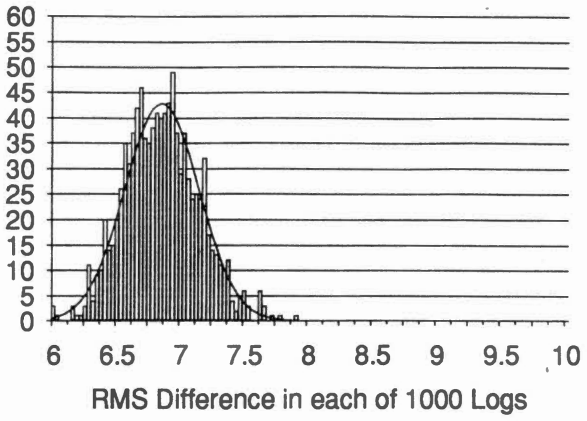
Histogram of TWAF followed by Recursive Median 3 (D2_71)



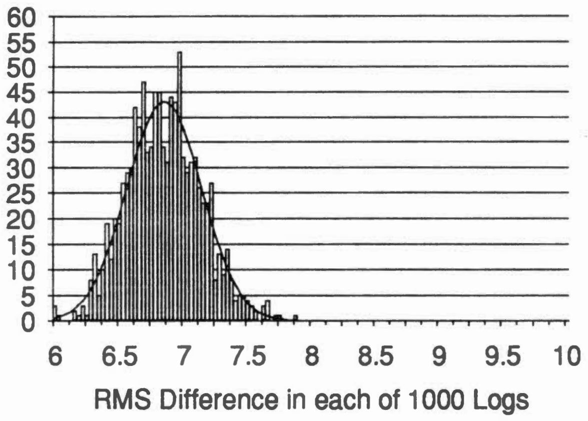
Histogram of TWAF followed by Recursive Median 3 (D2_75)



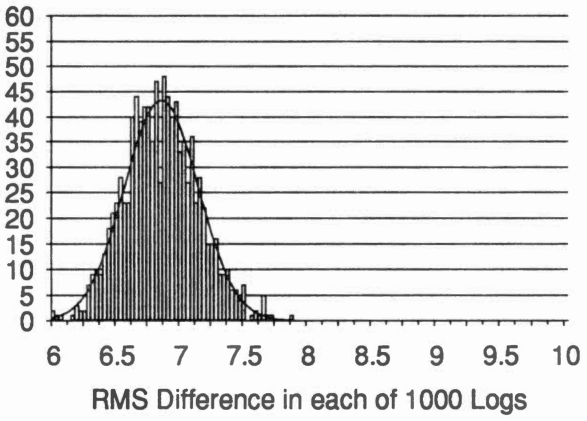
Histogram of TWAF followed by Recursive Median 3 (D2_78)



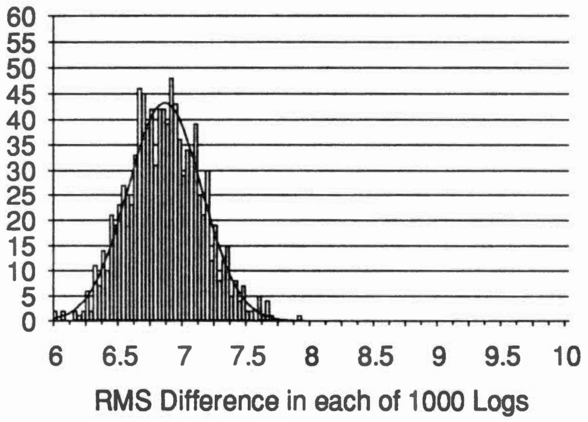
Histogram of TWAF followed by Recursive Median 3 (D2_81)



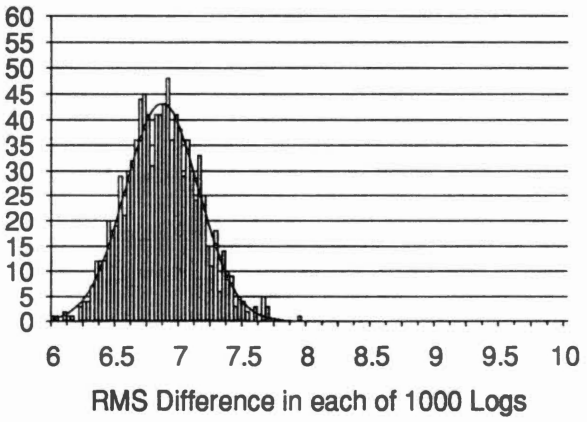
Histogram of TWAF followed by Recursive Median 3 (D2_84)



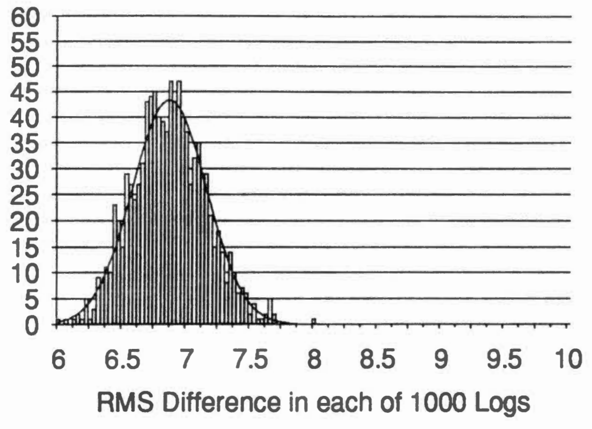
Histogram of TWAF followed by Recursive Median 3 (D2_87)



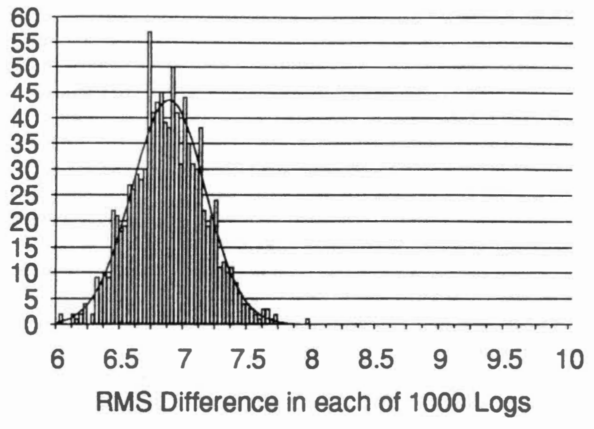
Histogram of TWAF followed by Recursive Median 3 (D2_90)



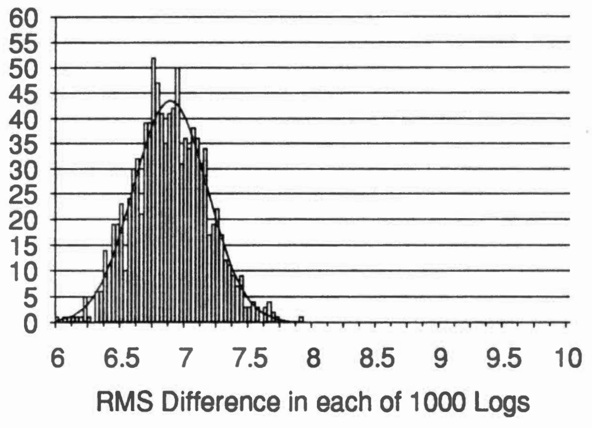
Histogram of TWAF followed by Recursive Median 3 (D2_93)



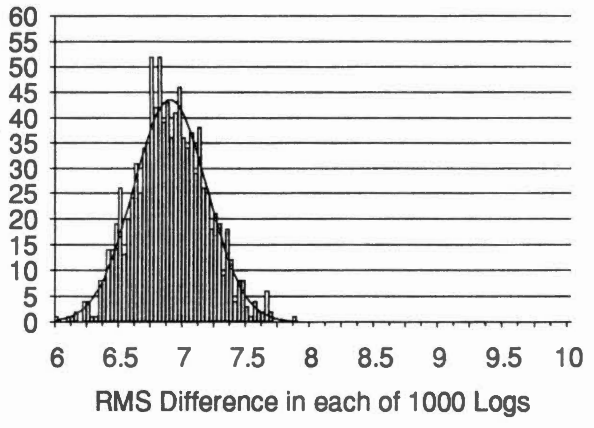
Histogram of TWAF followed by Recursive Median 3 (D2_96)



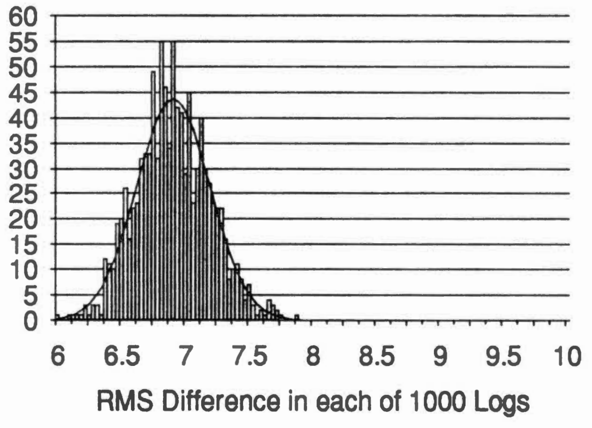
Histogram of TWAF followed by Recursive Median 3 (D3_00)



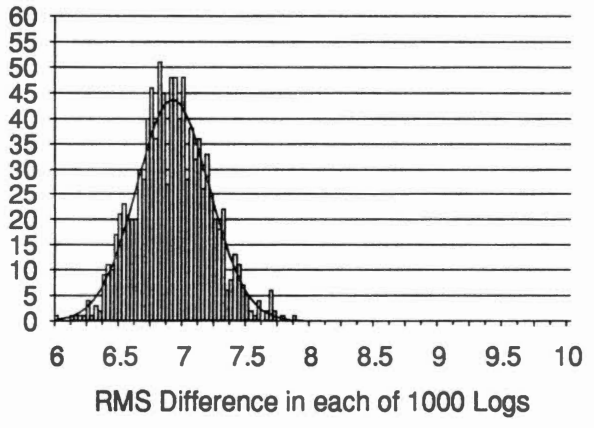
Histogram of TWAF followed by Recursive Median 3 (D3_03)



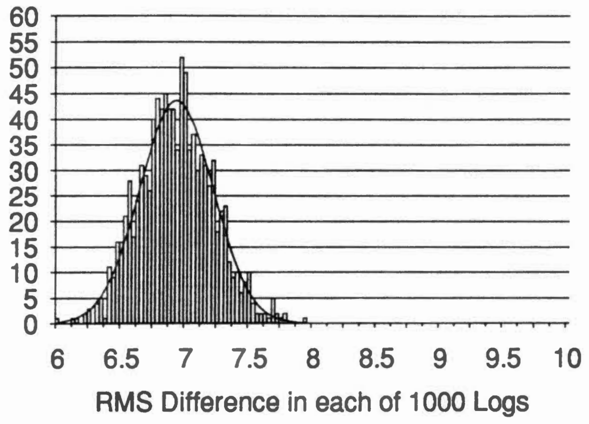
Histogram of TWAF followed by Recursive Median 3 (D3_06)



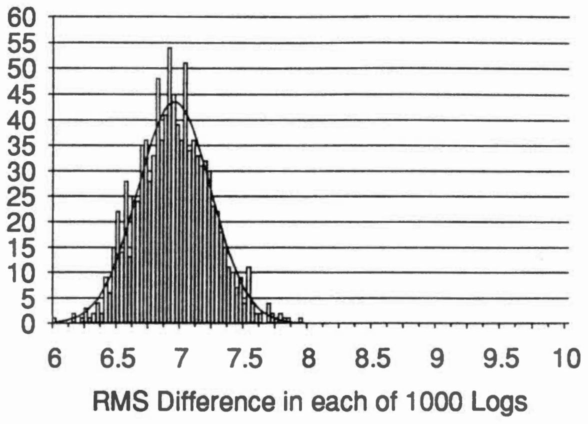
Histogram of TWAF followed by Recursive Median 3 (D3_09)



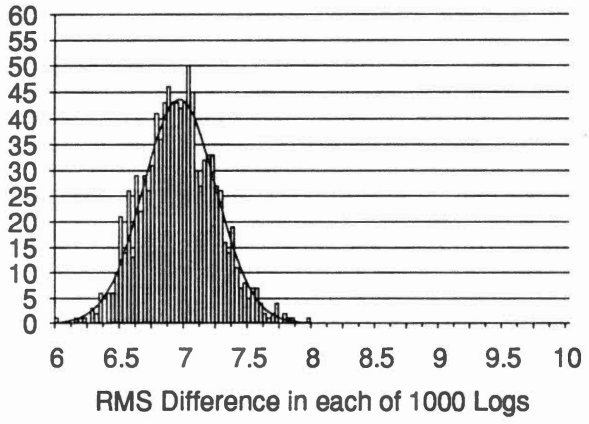
Histogram of TWAF followed by Recursive Median 3 (D3_12)



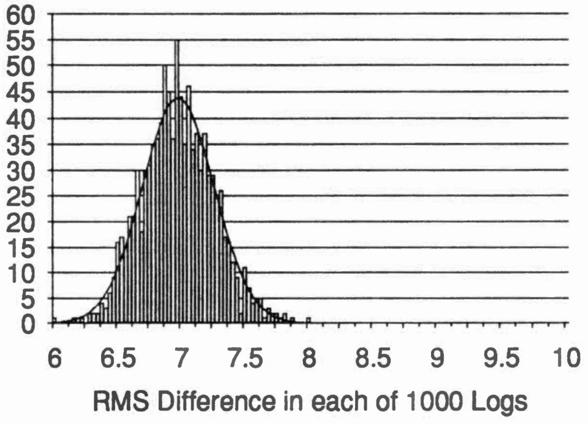
Histogram of TWAF followed by Recursive Median 3 (D3_15)



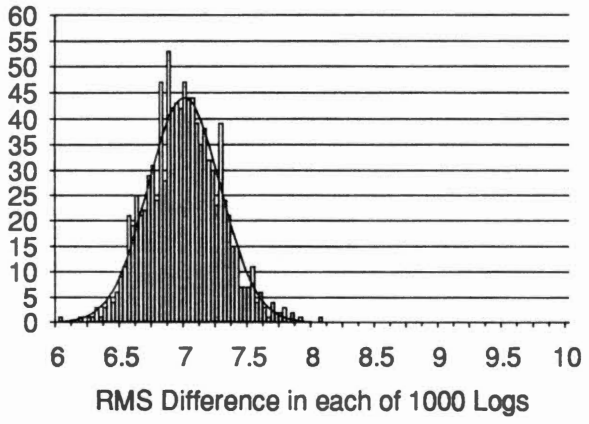
Histogram of TWAF followed by Recursive Median 3 (D3_18)



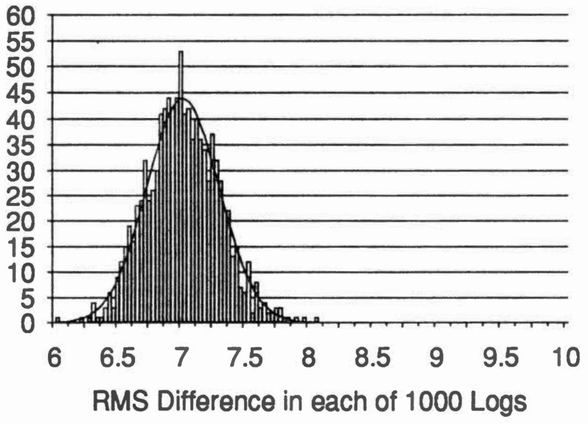
Histogram of TWAF followed by Recursive Median 3 (D3_21)



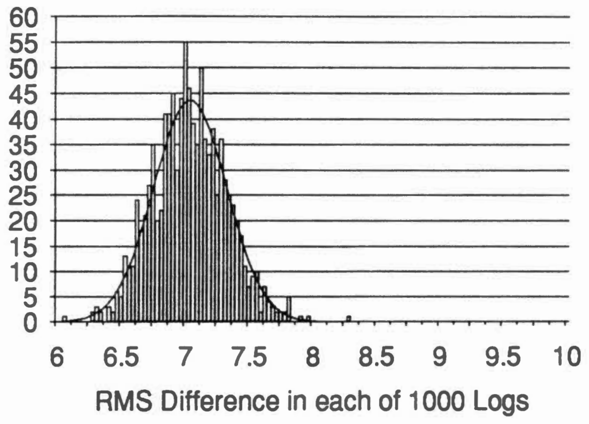
Histogram of TWAF followed by Recursive Median 3 (D3_25)



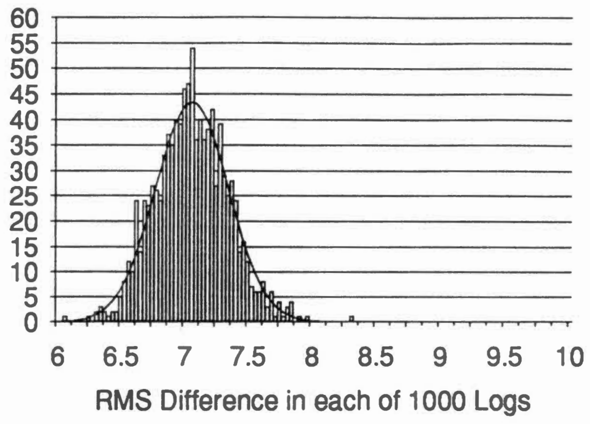
Histogram of TWAF followed by Recursive Median 3 (D3_28)



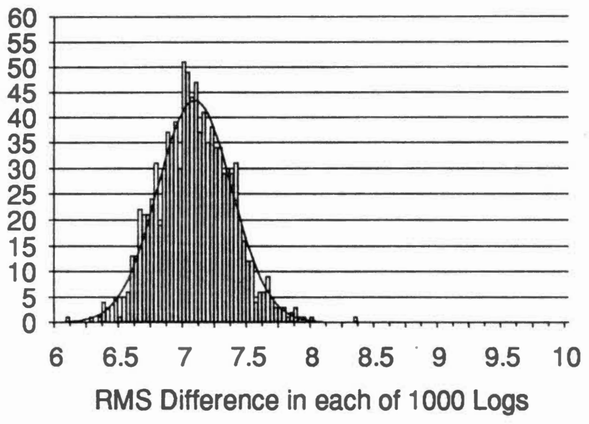
Histogram of TWAF followed by Recursive Median 3 (D3_31)



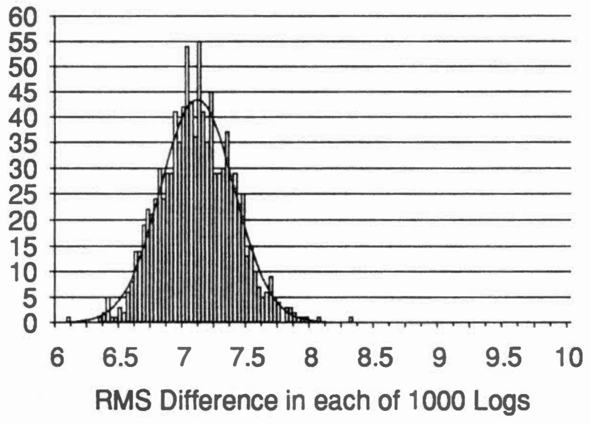
Histogram of TWAF followed by Recursive Median 3 (D3_34)



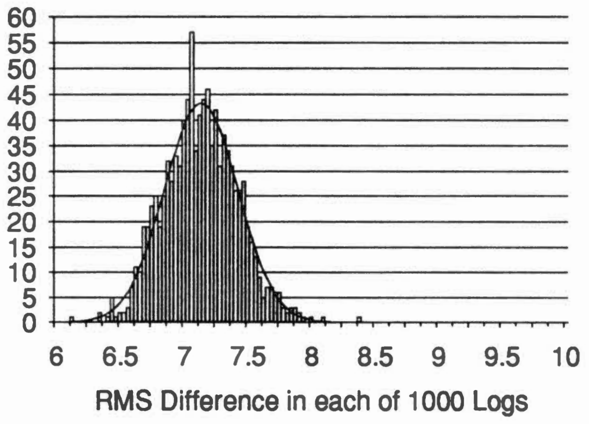
Histogram of TWAF followed by Recursive Median 3 (D3_37)



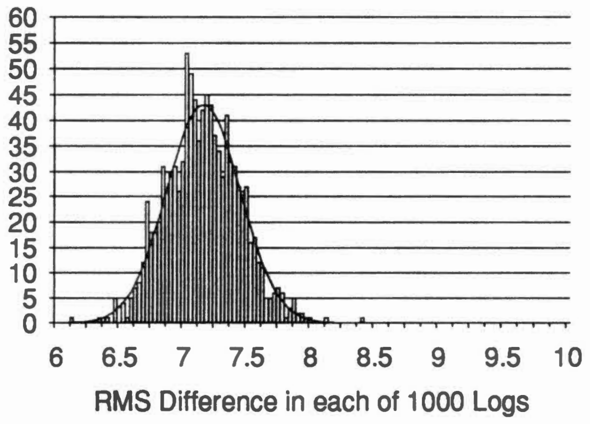
Histogram of TWAF followed by Recursive Median 3 (D3_40)



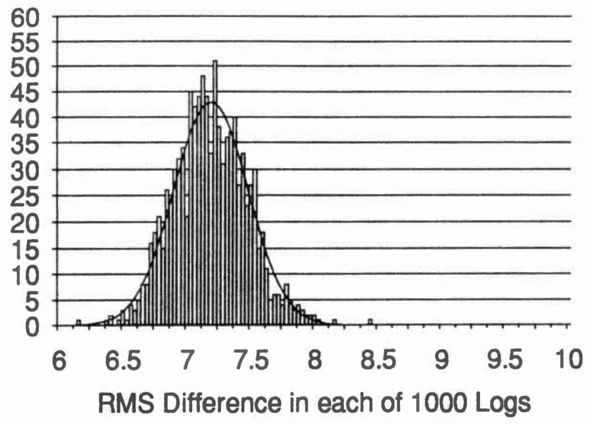
Histogram of TWAF followed by Recursive Median 3 (D3_43)



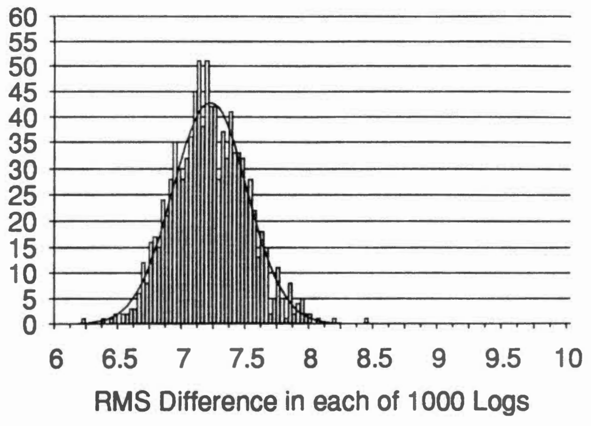
Histogram of TWAF followed by Recursive Median 3 (D3_46)



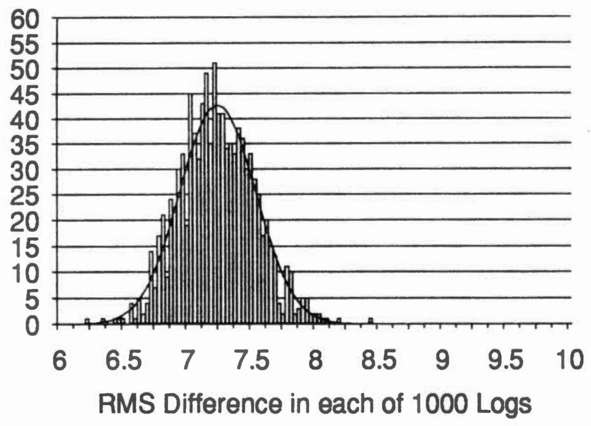
Histogram of TWAF followed by Recursive Median 3 (D3_50)



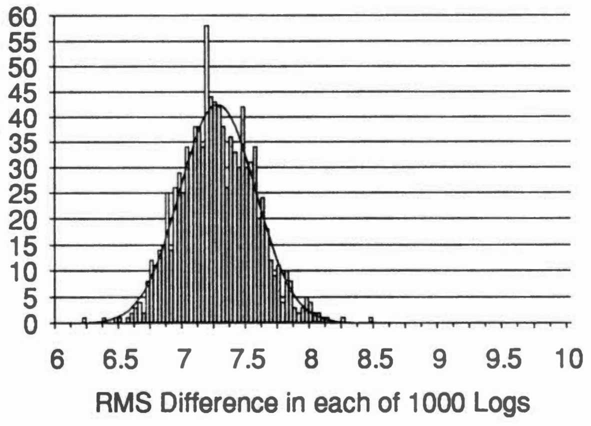
Histogram of TWAF followed by Recursive Median 3 (D3_53)



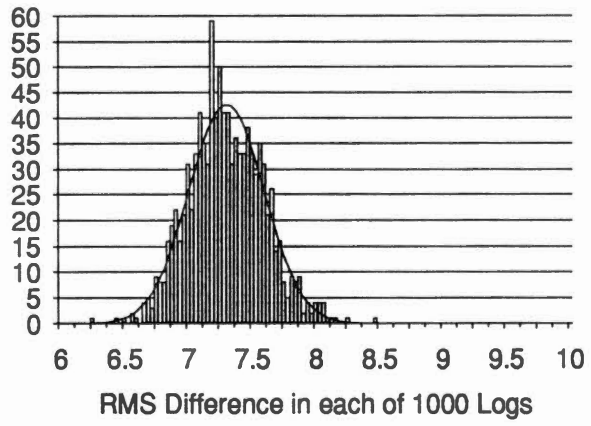
Histogram of TWAF followed by Recursive Median 3 (D3_56)



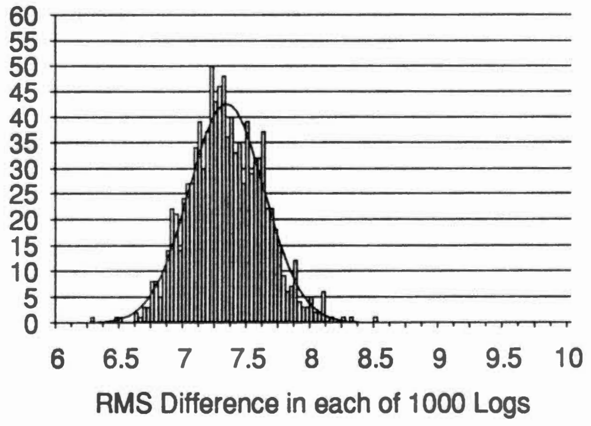
Histogram of TWAF followed by Recursive Median 3 (D3_59)

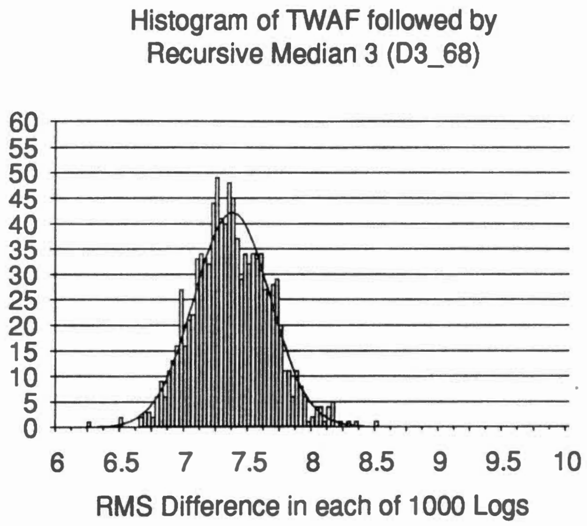


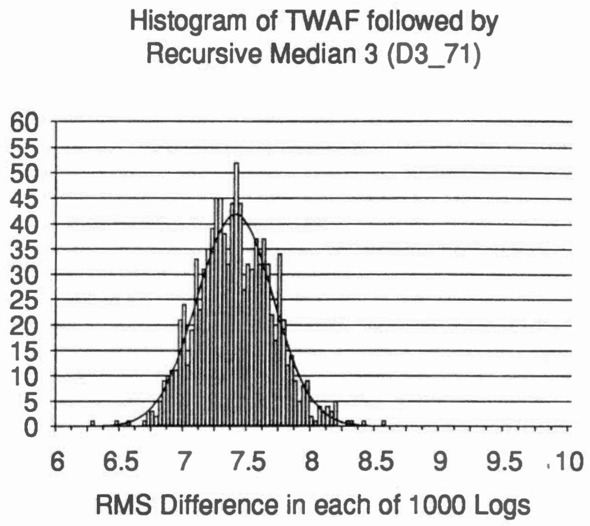
Histogram of TWAF followed by Recursive Median 3 (D3_62)

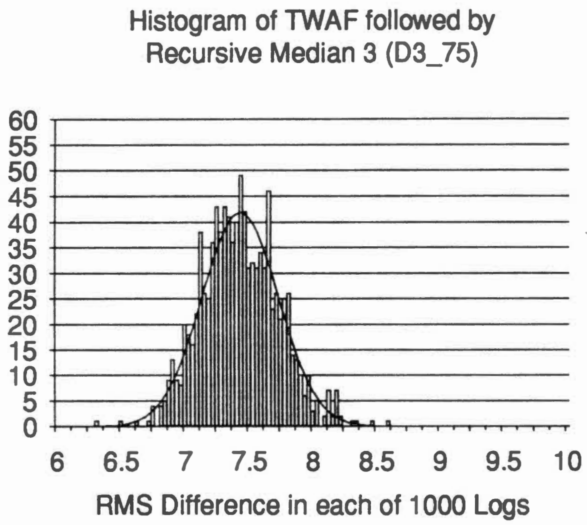


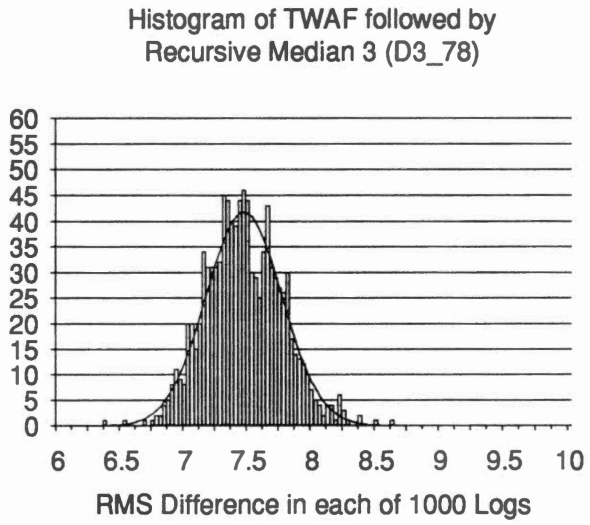
Histogram of TWAF followed by Recursive Median 3 (D3_65)

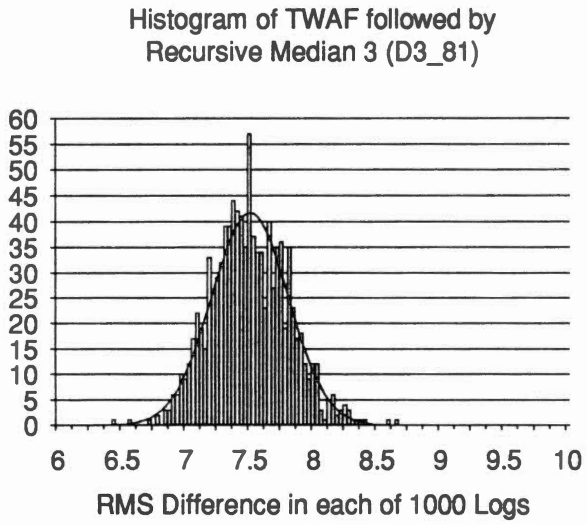


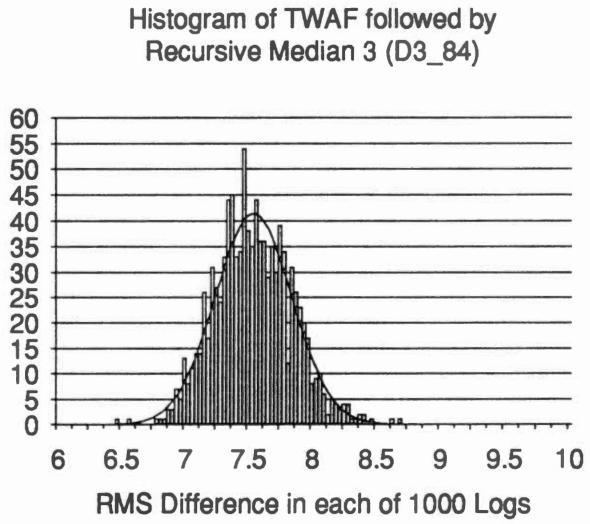

Histogram of TWAF followed by Recursive Median 3 (D3_68)
RMS Difference in each of 1000 Logs


Histogram of TWAF followed by Recursive Median 3 (D3_71)
RMS Difference in each of 1000 Logs

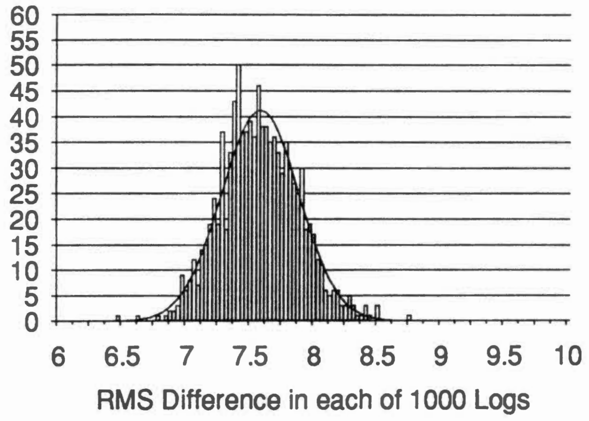

Histogram of TWAF followed by Recursive Median 3 (D3_75)
RMS Difference in each of 1000 Logs


Histogram of TWAF followed by Recursive Median 3 (D3_78)
RMS Difference in each of 1000 Logs

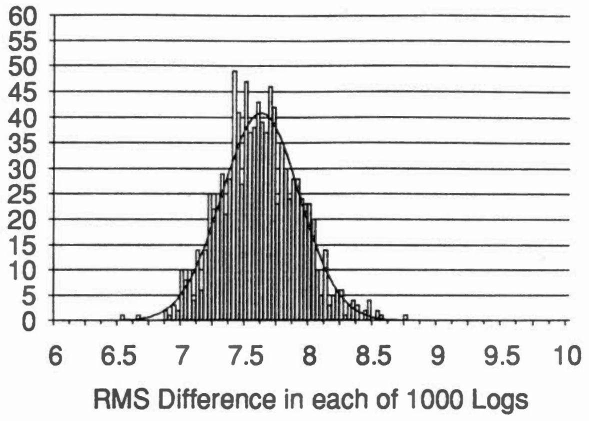

Histogram of TWAF followed by Recursive Median 3 (D3_81)
RMS Difference in each of 1000 Logs


Histogram of TWAF followed by Recursive Median 3 (D3_84)
RMS Difference in each of 1000 Logs

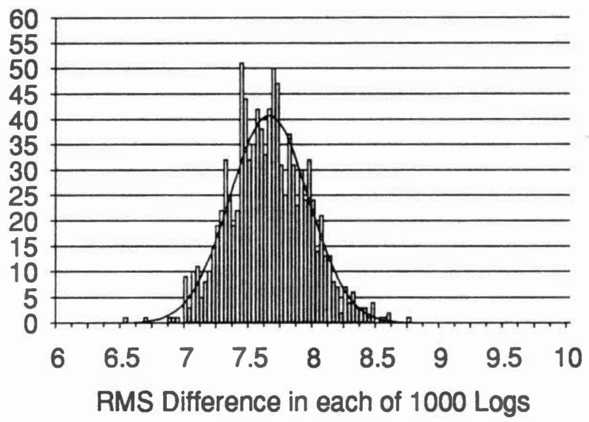
Histogram of TWAF followed by Recursive Median 3 (D3_87)



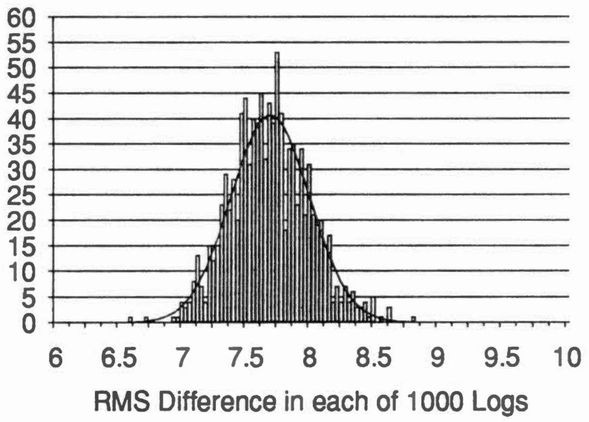
Histogram of TWAF followed by Recursive Median 3 (D3_90)



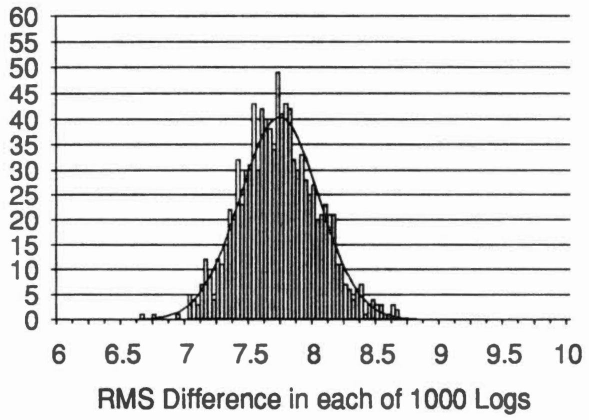
Histogram of TWAF followed by Recursive Median 3 (D3_93)



Histogram of TWAF followed by Recursive Median 3 (D3_96)



Histogram of TWAF followed by Recursive Median 3 (D4_00)



APPENDIX E

PROGRAM TO OPTIMIZE THE TWIN WINDOW AVERAGE FILTER

This is the program to optimize the Twin Window Average Filter with respect to the filter parameter. It requires the following routines from Appendix A:

`bubble.c, cputime.c, daytime.c, flush.c, gauss.c, ncproces.c, poisnois.c, randb.c,`

`reload.c, rmfilt.c, rmssumb.c, and twaffilt.c.`

In addition, to these, the main program `lvl.c`, requires the subroutines:

`appendel.c, gamv2lvl.c, rmslvl.c, savedata.c, and savemat.c.`

As in Appendix A, the source code was maintained in separate files with a Turbo C++ project file to compile each into an individual object file. These were then combined into an executable file.

The main program `lvl.c`:

```
/*  Compiles statistics for a particular level of signal.  Larry Paden 9/16/90.
Ideal gamma-ray log generator.
Makes multiple runs with different noise files.  Finds the RMS average of
the difference between the ideal log and a simple combination of recursive
median filters.  Larry Paden 3/30/84.
    Changed to print out the seed before each pass.  LJP 8/23/83.
    Synthetic log generator changed to produce small layers.  LJP 10/5/83.
    Prints the unformatted files directly.  LJP 8/30/84.
    Some subroutines are in [LARRY.PADEN]LIBRARY
    RELOAD subroutine added to pick up where it left off.  LJP 10/2/84.
    PHOENIX added.  LJP 10/4/84.
    Converted to C.  ljp 4/5/87.  Reload only protects old data under C.
    Reload removed from lvl version; it stopped the bedwid loop.  LJP 9/18/90.
    */
#include <stdio.h>
```

```
#include <sys\stat.h>
/* Level to examine. */                          #define LEVEL 288.0
/* Width of smallest bed. */                     /*#define WIDMIN 5*/
                                                 #define WIDMIN 5
/* Width of widest bed plus one. */              #define WIDMAX 11
/*/* RAM disk.  Root \ is limited, so use sub. */   #define DIR "D:\\lvl"  */
/* RAM disk.  Root \ is limited, so use sub. */ #define DIR "D:\\lv2"
/* Signal length in samples. */                  #define LENSIG 2048
/* Number of noisy logs. */                      #define RUNS 1000
/* Controls verbosity of text output. */         #define IRMSVERB 0
/* The number of points in the outer window. */ #define LARGE 9
/* Lowest filter parameter. */                   #define BOTTOM 2.0
/* Highest filter parameter. */                  #define TOP 4.0
/* Filter parameter increment. */                /*#define DELTA 0.03125*/
                                                 #define DELTA 0.125
/* Minimum seconds between hard disk writes. */ #define TIMEALL (600.0)
/* Minimum slope. */                             #define SLPMIN 0.0
/* Maximum slope. */                             #define SLPMAX 0.0
/*/* Name of file for text output. */              #define LUWNAME "dd\\text"*/
/* Name of file for text output. */              #define LUWNAME "d2\\text"
char tmpstr[128] = "";          /* Place to store a data file name. */
char dirname[128] = "";         /* Place to store a directory name. */
void main() {
    FILE *luwrite,              /* File descriptor for writing text. */
        *tempfile,              /* File descriptor for other files. */
        *hardfile;              /* File descriptor to copy to hard disk. */
    void flush();               /* For MS-DOS fflush only. */
    double cputime();           /* CPU (?) time used.  MS-DOS elapsed time. */
    double lasttime;            /* Save the last time files were transfered. */
    static double param,        /* Filter parameter. */
        rmsdiff[WIDMAX],        /* RMS difference between two logs. */
        rmsavg1[WIDMAX],
        rmsavg2[WIDMAX],        /* RMS average of the two logs. */
        signal[LENSIG],
        ideal[LENSIG],
        xplt[LENSIG];           /* Places to put logs. */
    float wrflt;                /* Convert to single precision. */
    long longtmp;               /* Temporary storage of a special length. */
    int limit,                  /* Length of a log in points. */
        irun,                   /* Log suite sequence number. */
        junk,                   /* Info returned, but not used. */
        kk,                     /* Loop counter through a log. */
        errtmp,                 /* Error message data. */
        place,                  /* Saves a place, typically in a string. */
        bedwid;                 /* Bed width under consideration. */
    unsigned short iseed[3];    /* Random number generator seed. */
/*  int sprflng, sprfint;       /* For checking sprintf results. */
    char *strrchr(),            /* Pointers to a string. */
        *errstr;            /* String to print in an error message. */
    limit = LENSIG;

    /* Write various input parameters. */
    luwrite = fopen (LUWNAME, "a+");
    daytime (luwrite);
    fprintf (luwrite, "Hard disk is updated every %g seconds from %s.\n",
        TIMEALL, DIR);
    fprintf (luwrite, "The outer window is length:  %d.\n", LARGE);
    fprintf (luwrite, "Bottom, top, delta: %g, %g, %g\n",
        BOTTOM, TOP, DELTA);
    fprintf (luwrite, "Files have %d points; transitions %g to %g.\n",
        LENSIG, SLPMIN, SLPMAX);
    flush (luwrite);

    /* Initialize various things.  (See discriptions above.) */
    iseed[0] = 0xe66d;
    iseed[1] = 0xdeec;
    iseed[2] = 0x5;
/*  if (stat (".\\dd", (struct stat *)ideal) != 0 &&
        (errtmp = mkdir (".\\dd")) != 0) {
        fprintf (luwrite, "Cannot make directory %s %d\n", ".\\dd", errtmp);
        exit (-9);
    }
*/
    if (stat (".\\d2", (struct stat *)ideal) != 0 &&
        (errtmp = mkdir (".\\d2")) != 0) {
        fprintf (luwrite, "Cannot make directory %s %d\n", ".\\d2", errtmp);
        exit (-9);
    }
    if (stat (DIR, (struct stat *)ideal) == 0) {
        fprintf (luwrite, "Directory %s already exists.  ", DIR);
        fprintf (luwrite, "Are other processes using it?\n");
        exit (-8);
    }
    if ((errtmp = mkdir (DIR)) != 0) {
        fprintf (luwrite, "Cannot make directory %s %d\n", DIR, errtmp);
        exit (-9);
    }
    sprintf (dirname, "%s\\*.*", DIR);
    lasttime = 0.0;
    irun = 0;
    ncprocessor();

    fprintf (luwrite, "Run and seed: %d %ux %ux %ux\n",
        irun, iseed[2], iseed[1], iseed[0]);
    daytime(luwrite); flush (luwrite);

    /* Do for each bed width.  Typically 5 to 10 inclusive. */
    for (bedwid=WIDMIN; bedwid<WIDMAX; bedwid++) {

    /* Do the number of times in RUNS. */
    for (irun=0; irun<RUNS; irun++) {

        /* Report on progress.  Make sure things are saved on hard disk. */
        printf ("%dL%d ", bedwid, irun); daytime (stdout);
        fprintf (luwrite, "%dL%d ", bedwid, irun);
        daytime (luwrite); flush (luwrite);
        if ((errtmp = sprintf (tmpstr, errstr="seed%.2d.",
            bedwid, (int)LEVEL)) != 7) {
```

```
        fprintf (luwrite,
            "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
        exit (-28);
    }
    savedata (luwrite, DIR, tmpstr, iseed, sizeof(short), 3);

    /* Check time.  If not written in last TIMEALL seconds, then write. */
    if (cputime()-lasttime > TIMEALL) {
        lasttime = cputime();
/*      errtmp = appendel (luwrite, dirname, ".\\dd", &longtmp);    */
        errtmp = appendel (luwrite, dirname, ".\\d2", &longtmp);
        fprintf (luwrite,
            "Appending %d files totalling %ld bytes (%g).\n",
            errtmp, longtmp, errtmp?(float)longtmp/errtmp:0.0);
        fprintf (luwrite, "%g %d %ux %ux %ux\n",
            cputime(), irun, iseed[2], iseed[1], iseed[0]);
        daytime(luwrite); flush (luwrite);
    }

    /* Create synthetic log; copy it; add noise; measure result; save. */
    gamv2lvl (ideal, limit, iseed, SLPMIN, SLPMAX,
        LEVEL, (double)bedwid, 24., 50.);
    for (kk=0; kk<limit; kk++) signal[kk] = ideal[kk];
    poisnois (signal, limit, iseed);
    rmssub (ideal, signal, 0, limit, IRMSVERB,
        &rmsdiff, &rmsavg1, &rmsavg2);

    /* Save the RMS difference between the ideal and noisy logs. */
    if ((errtmp = sprintf (tmpstr, errstr="diff%.2d%%.2d.%.3d",
        bedwid, (int)LEVEL)) != 14) {
        fprintf (luwrite,
            "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
        exit (-29);
    }
    savemat (luwrite, DIR, tmpstr, rmsdiff, bedwid+1, 0, 1);

    /* Save the ideal array average value. */
    if ((errtmp = sprintf (tmpstr, errstr="idyl%.2d%%.2d.%.3d",
        bedwid, (int)LEVEL)) != 14) {
        fprintf (luwrite,
            "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
        exit (-30);
    }
    savemat (luwrite, DIR, tmpstr, rmsavg1, bedwid+1, 0, 1);

    /* Save the noisy array average value. */
    if ((errtmp = sprintf (tmpstr, errstr="nois%.2d%%.2d.%.3d",
        bedwid, (int)LEVEL)) != 14) {
        fprintf (luwrite,
            "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
        exit (-31);
    }
    savemat (luwrite, DIR, tmpstr, rmsavg2, bedwid+1, 0, 1);
```

```
        /* Copy the data so that the copy can be filtered. */
        for (param=BOTTOM; param<=TOP; param+=DELTA) {
            for (kk=0; kk<limit; kk++) xplt[kk] = signal[kk];
            twaffilt (xplt, limit, LARGE, param, &junk, 0);
/*          rmfilt (xplt, limit, 3, &junk, 1);*/
            rmslvl (luwrite, ideal, xplt, 0, limit, IRMSVERB, bedwid, LEVEL,
                rmsdiff, rmsavg1, rmsavg2);

            /* Save postfiltering noisy average array. */
            if ((errtmp = sprintf (tmpstr, errstr="n%.3d%.2d%%.2d.%.3d",
                (int)(param*100.), bedwid, (int)LEVEL)) != 14) {
                fprintf (luwrite,
                    "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
                exit (-32);
            }
            savemat (luwrite, DIR, tmpstr, rmsavg2, bedwid+1, 0, 1);

            /* Save postfiltering RMS difference array. */
            if ((errtmp = sprintf (tmpstr, errstr="d%.3d%.2d%%.2d.%.3d",
                (int)(param*100.), bedwid, (int)LEVEL)) != 14) {
                fprintf (luwrite,
                    "Stopped by sprintf %d on format %s.\n", errtmp, errstr);
                exit (-33);
            }
            savemat (luwrite, DIR, tmpstr, rmsdiff, bedwid+1, 0, 1);
        }
    }

    /* Copy all the leftover pieces to hard disk. */
/*  errtmp = appendel (luwrite, dirname, ".\\dd", &longtmp); */
    errtmp = appendel (luwrite, dirname, ".\\d2", &longtmp);
    fprintf (luwrite, "Appending %d files totalling %ld bytes (%g).\n",
        errtmp, longtmp, errtmp?(float)longtmp/errtmp:0.0); flush (luwrite);
    }

    /* Copy all the leftover pieces to hard disk.  (Shouldn't be any now.) */
/*  errtmp = appendel (luwrite, dirname, ".\\dd", &longtmp);*/
    errtmp = appendel (luwrite, dirname, ".\\d2", &longtmp);
    fprintf (luwrite, "Appending %d files totalling %ld bytes (%g).\n",
    errtmp, longtmp, errtmp?(float)longtmp/errtmp:0.0);

    /* Clean up; run next job, if any. */
    daytime (luwrite);
    fprintf (luwrite, "Finished!!!");
    fclose (luwrite);
    system ("NEXT.BAT\n");
}
```

Subroutine `appendel.c`:

```
/* Takes a drive, path, and DOS file prototype; appends the files to those of
 * the same name in the output directory, or creates them if nonexistent; and
 * deletes the original file.  All the while doing extensive error checking.
 * If the output device is full, it prints a message and outputs a "." every
 * ten minutes hoping someone will correct the problem.  Larry Paden 9/14/90.
 */
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <string.h>
#define BUFLEN 128
int appendel (
    FILE *lu,
    char *indir,    /* Input path and file prototype. */
    char *outdir,   /* Output directory in which appended or created. */
    long *count)    /* Count of total bytes written. */
{
    struct ffblk fileblk;   /* Holds file information. */
    FILE *hardfile,         /* For writing. */
        *tempfile;          /* For reading. */
    int dirstat,            /* Zero if another file is found by findnext. */
        errnbr,             /* Error number return. */
        filecnt,        /* Count the files and return the value. */
        ii,                 /* Temporary counter. */
        nitems, nindx,      /* Number of items read in. */
        readdesc,
        writdesc;           /* File handles for reading and writing. */
    char infile [MAXPATH],  /* Actual individual input file name. */
        outfile [MAXPATH],  /* Actual individual output file name. */
        buffer [BUFLEN],    /* Transfer to this RAM way station. */
        *pnt;
    *count = 0;
    filecnt = 0;
    dirstat = findfirst (indir, &fileblk, 0);
    while (!dirstat) {
        filecnt++;

        /* Make true name and open the input file for reading. */
        strcpy (infile, indir);
        if ((pnt = strrchr (infile, '\\')) != 0) pnt[1] = '\0';
        strcat (infile, fileblk.ff_name);
        if ((tempfile = fopen (infile, "rb")) == NULL) {
        fprintf (lu, "Appendel cannot open input %s.\n", infile);
            exit (-17);
        }

        /* Make true name and open the output file for appending or create it. */
        strcpy (outfile, outdir);
        if (outfile[strlen(outfile)-1] != '\\') strcat (outfile, "\\");
        strcat (outfile, fileblk.ff_name);
        if ((hardfile = fopen (outfile, "a+b")) == NULL) {
            fprintf (lu, "Appendel cannot open output %s.\n", outfile);
            exit (-15);
        }

        /* Perform the copy operation with low-level reads and writes. */
        readdesc = fileno(tempfile);
        writdesc = fileno(hardfile);
        while ((nitems =
            read (readdesc, buffer, (unsigned)BUFLEN)) > 0) {
            (*count) += (errnbr = write (writdesc, buffer, (unsigned)nitems));
            if (errnbr != nitems) {
                fprintf (lu, "Bad write to %s %d<%d.\n",
                    outfile, errnbr, nitems); flush (lu);
                fprintf (stderr, "Bad write to %s %d<%d.\n",
                    outfile, errnbr, nitems);
                sleep (600);
                nindx = errnbr;
                nitems -= errnbr;

                /* While hard disk is full, give user a chance to fix. */
                (*count) += (errnbr =
                    write (writdesc, &buffer[nindx], (unsigned)nitems));
                while (errnbr != nitems) {
                    fprintf (lu, ".");
                    fprintf (stderr, ".");
                    nindx += errnbr;
                    nitems -= errnbr;
                    sleep (600U);
                    (*count) += (errnbr =
                        write (writdesc, &buffer[nindx], (unsigned)nitems));
                }
            }
        }
        if (fclose (tempfile) == EOF) {
            fprintf (lu, "Stopped by fclose tempfile.\n");
            exit (-18);
        }
        if (fclose (hardfile) == EOF) {
            fprintf (lu, "Stopped by fclose hardfile.\n");
            exit (-19);
        }
        unlink (infile);
        dirstat = findnext (&fileblk);
    }
    return (filecnt);
}
```

Subroutine `gamv2lvl.c`:

```
/* Generates random synthetic gamma-ray logs.  HIMIN and
 * HIMAX are chosen to make the average noise power of the
 * generated logs to be 13.  Larry Paden 10/5/83.
 * WIDMAX and initial width changed 5/17/84.  LJP
 * Gamge2 created to add random slopes between levels.  LJP 6/7/84.
 * Gamgevar to allow calling program to select width of slopes. LJP 8/5/84.
 * Gamgev2 to tidy up.  Parameters are the same, but fewer calls to erand()
 *  are made, so this will not generate the same synthetic log.  LJP 9/12/90.
 * Gamv2lvl makes logs with extra levels (lvl) every pnts points.  LJP 9/12/90.
 */
#include <math.h>
static double next=-1.0;    /* Next occurence of extra bed. */
```

```
void gamv2lvl (double xx[], /* Incoming ideal log. */
    int isize,              /* Length in samples of the ideal log. */
    unsigned short iseed[], /* Seed for the random number generator. */
    double slpmin,          /* Minimum transition between levels. */
    double slpmax,          /* Maximum transition between levels. */
    double lvl,             /* Level to use extra times in log. */
    double extent,  /* Width of lvl; if 0, then range from WIDMIN to WIDMAX-1. */
    double init,        /* Where to make the initial point.  If <0, set to pnts. */
    double pnts)                /* Attempt to produce lvl every pnts points. */
{
    double erand48b (unsigned short[]), slpwid, swidth, width, height, oldhi;
    int ii, jj;
    void newhw (double[], double[], int, unsigned short[],
        double, double, double, double);
    next = -1.0;
    slpwid=slpmax-slpmin;
    ii = 0;
    newhw (&height, &width, ii, iseed, lvl, extent, init, pnts);
/*  printf ("At %d W1, h1:  %g, %g\n", ii, width, height);*/
    while (ii < isize) {
        /* Generate width points on a level. */
        for (jj=ii; jj<=ii+width-1 && jj<isize; jj++) {xx[jj] = height;}
        ii = ii+width;
        oldhi = height;
        newhw (&height, &width, ii, iseed, lvl, extent, init, pnts);
/*      printf ("At %d W1, h1:  %g, %g\n", ii, width, height);*/
        if (slpmin+slpwid > 0.0) {
            swidth = slpmin + (slpwid==0.0 ? 0.0 : slpwid*erand48b(iseed));
/*          printf ("At %d W2, h2:  %g, %g\n", ii, swidth, height);*/
            /* Generate swidth points on a slope. */
            for (jj=ii; jj<=ii+swidth-1 && jj<isize; jj++) {
                xx[jj] = oldhi + (jj-ii+1)*(height-oldhi)/(int)(swidth+1);}
            ii = ii+swidth;
        }
    }
}
#define WIDMIN 5
#define WIDMAX 11
#define WIDE (WIDMAX-WIDMIN)
#define HIMIN 50
#define HIMAX 288
#define HIGH (HIMAX-HIMIN)
void newhw (
    double *height, /* New level to be generated. */
    double *width,      /* New number of points to be generated at that level. */
    int ii,         /* Counter to determine where in the log. */
    unsigned short iseed[], /* RV generator seed. */
    double lvl,     /* Make extra occurences of this bed. */
    double extent,  /* Width of lvl; if 0, then range from WIDMIN to WIDMAX-1. */
    double init,        /* Where to make the initial point.  If <0, set to pnts. */
    double pnts)    /* Average number of points apart for start of extra beds. */
{
    double erand48b (unsigned short[]);
    if (next == -1.0) {
        next = (init<0) ? pnts - WIDMIN - (WIDE-1)/2.0 : init;
    }
    if (ii < next) {    /* Generate as usual. */
        *height = HIMIN+HIGH*erand48b(iseed);
        *width = WIDMIN+WIDE*erand48b(iseed);
    }
    else {              /* Generate the special level. */
        *height = lvl;
        *width = (extent==0.0) ? WIDMIN+WIDE*erand48b(iseed) : extent;
        next += pnts;
    }
}
```

Subroutine `rmslvl.c:`

```
/* Given real arrays X1 and X2 and range ISTART to LIMIT, this calculates
 * the RMS average of the difference (RMSAVG), and the ordinary average
 * values of the two input files (AVG1 and AVG2.)  The results are labelled
 * and printed if VERBOSE > 0.  Larry Paden 6/22/83.
 * From Fortran.  ljp 4/5/87. */
#include <stdio.h>
#include <math.h>
#define TOTLEN 16
void rmslvl (
    FILE *lu,                   /* File pointer for messages. */
    double ideal[],
    double noisy[],             /* Incoming arrays. */
    int istart,                 /* First point in evaluation. */
    int limit,                  /* Last point NOT in evaluation. */
    int verbose,            /* Print if > 0. */
    int max,                    /* Maximum number of points in a run. */
    double lvl,                 /* Look for runs at this level. */
    double rmsavg[],
    double avg1[],
    double avg2[])              /* The three outputs. */
{
    int ii, cnt[TOTLEN], which;
    if (max > TOTLEN) {
        fprintf (lu, "Need more space in rmslvl %d>%d.\n", max, TOTLEN);
        exit (110);
    }
    for (ii=0; ii<=max; ii++) {
        rmsavg[ii] = 0.0;
        avg1[ii] = 0.0;
        avg2[ii] = 0.0;
        cnt[ii] = 0;
    }
    cnt[0] = limit-istart;
    which = 0;
    for (ii=istart; ii<limit; ii++) {
        rmsavg[0] += (ideal[ii]-noisy[ii]) * (ideal[ii]-noisy[ii]);
        avg1[0] += ideal[ii];
        avg2[0] += noisy[ii];
        if (ideal[ii] == lvl) {         /* Yes, it must be exactly equal! */
```

```
            cnt[++which]++;
            rmsavg[which] += (ideal[ii]-noisy[ii]) * (ideal[ii]-noisy[ii]);
            avg1[which] += ideal[ii];
            avg2[which] += noisy[ii];
        }
        else {
            which = 0;
        }
    }
    for (ii=0; ii<=max; ii++) {
        if (cnt[ii] > 0) {
            rmsavg[ii] = sqrt (rmsavg[ii]/cnt[ii]);
            avg1[ii] = avg1[ii]/cnt[ii];
            avg2[ii] = avg2[ii]/cnt[ii];
        }
    }
    if (verbose > 0) fprintf (lu, "RMS = %lg; averages = %lg %lg\n",
        *rmsavg, *avg1, *avg2);
    if (verbose > 1)
        for (ii=0; ii<max; ii++)
            fprintf (lu, "%.2d %g %g %g %d\n",
                ii, rmsavg[ii], avg1[ii], avg2[ii], cnt[ii]);
    flush (lu);
}
```

Subroutine `savedata.c`:

```
/* Lu should be a file descriptor open for writing (possible) error messages.
 * Creates or appends to file fname in directory dir the data pointed to by
 * wrflt which is size bytes long.  Larry J. Paden 9/14/90.
 * Example:  savedata (stderr, "D:\\dd", "datanoi2", &wrflt, sizeof(wrflt), 1);
 */
#include <stdio.h>
savedata (FILE *fd,       /* Send error messages to this file descriptor. */
    char *dir,            /* Open the target file in this directory. */
    char *fname,          /* Name of target file. */
    char *wrflt,          /* Pointer to data to write to target. */
    int size,             /* Size of single item of data. */
    int number)           /* Number of data items to be written. */
{
    FILE *tempfile;       /* File descriptor of the target file. */
    int place,            /* Holds the place in the string. */
        errtmp;           /* Holds error codes. */
    char tmpstr[256];     /* Place to build up true file name. */
    strcpy (tmpstr, dir);
    place=strlen(tmpstr);
    if (tmpstr[place] != '\\') {     /* If no \ on directory name, get one. */
        tmpstr[place++] = '\\';
        tmpstr[place] = '\0';
    }
    strcat (tmpstr, fname);
    if ((tempfile = fopen (tmpstr, "ab")) == NULL) {
        fprintf (fd, "Stopped by fopen %s.\n", tmpstr);
        exit (101);
    }
    if ((errtmp = fwrite (wrflt, size, number, tempfile)) != number) {
        fprintf (fd, "Stopped by fwrite %s. %d\n", tmpstr, errtmp);
        exit (102);
    }
    if (fclose (tempfile) == EOF) {
        fprintf (fd, "Stopped by fclose %s.\n", tmpstr);
        exit (103);
    }
    return (1);     /* The number of files written. */
}
```

Subroutine savemat.c:

```
/* Lu should be a file descriptor open for writing (possible) error messages.
 * Creates or appends to files in directory dir the data pointed to by
 * wrflt which is size bytes long.  The names of the (times) number of files
 * is created by format varied over the range from start to start+times*incr.
 * Larry J. Paden 9/14/90.
 * Example:  savemat (stderr, "D:\\dd", format, data_array, data_length, 0, 1);
 */
#include <stdio.h>
savemat (FILE *fd,          /* Send error messages to this file descriptor. */
    char dir[],             /* Open the target file in this directory. */
    char format[],          /* Printf style name of target files. */
    double data[],          /* Pointer to data to write to target. */
    int times,              /* Times to use format to create files. */
    int start,              /* Starting argument to printf. */
    int incr)               /* Increment to increase start. */
{
    FILE *tempfile;         /* File descriptor of the target file. */
    float wrflt;            /* Cast the incoming double into this float. */
    int ii,                 /* Loop counter. */
        place,              /* Holds the place in the string. */
        errtmp;             /* Holds error codes. */
    char tmpstr[256];       /* Place to build up true file name. */
    int value;              /* Current value of the start+=incr sequence. */
    value = start;
    strcpy (tmpstr, dir);
    place=strlen(tmpstr);
    if (tmpstr[place] != '\\') {    /* If no \ on directory name, get one. */
        tmpstr[place++] = '\\';
        tmpstr[place] = '\0';
    }
    for (ii=0; ii<times; ii++) {
        if ((errtmp = sprintf (&tmpstr[place], format, value)) < 3) {
            fprintf (fd, "Stopped by sprintf (\"%s\", \"%s\", %g).\n",
                tmpstr, format, value);
            exit (100);
        }
        if ((tempfile = fopen (tmpstr, "ab")) == NULL) {
            fprintf (fd, "Savemat cannot open %s.\n", tmpstr);
            exit (101);
        }
        wrflt = data[ii];
        if ((errtmp = fwrite (&wrflt, sizeof(wrflt), 1, tempfile)) != 1) {
            fprintf (fd, "Savemat cannot write %s. %d\n", tmpstr, errtmp);
            exit (102);
        }
        if (fclose (tempfile) == EOF) {
            fprintf (fd, "Savemat cannot close %s.\n", tmpstr);
            exit (103);
        }
        value += incr;
    }
    return (times);
}
```

2

VITA

Larry J. Paden

Candidate for the Degree of

Doctor of Philosophy

Thesis: NOISE REDUCTION IN THE GAMMA-RAY LOG BY MEANS OF NONLINEAR FILTERING

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Tulsa, Oklahoma, March 21, 1957 to Mary Lois and Jack Taylor Paden, Jr., the oldest of three children. Married Carol D. McAlister July 28, 1979. Has three children: John Lawrence born April 25, 1985; Zachary Taylor, September 3, 1987; and Katherine Elizabeth, July 6, 1990.

Education: Graduated valedictorian Charles Page High School at Sand Springs, Oklahoma in May 1975. Received the Bachelor of Science Degree in Electrical Engineering in July 1979 from Oklahoma State University. Received the Master of Electrical Engineering Degree in May 1980 from Oklahoma State University. Completed the requirements for the Doctor of Philosophy degree at Oklahoma State University in May 1991.

Professional Experience: National Science Foundation summer program at University of Texas at Arlington, 1977. Engineering Intern, Amoco Production Research Center, Tulsa, Oklahoma May 1978 to January 1979. Teaching Assistant, School of Electrical Engineering Oklahoma State University, January 1979 to May 1979. Graduate Research Assistant, School of Electrical Engineering Oklahoma State University, March 1979 to May 1980. Development Engineer, AT&T Western Electric, Oklahoma

City, Oklahoma May 1980 to January 1983. Took two-year leave of absence to return to Oklahoma State University. Graduate Research Assistant, School of Electrical Engineering Oklahoma State University, January 1983 to December 1984. Development Engineer, AT&T Technologies, Inc. Oklahoma City, Oklahoma January 1985 to August 1990, where he led the effort to fund, design, and build a multimillion dollar facility to test the electromagnetic compliance of the products produced. University of Oklahoma, School of Electrical Engineering and Computer Science, Visiting Instructor, lectured on Computer Architecture Spring 1989 and Spring 1991.

Awards/Affiliations: Vocational Industrial Clubs of America Oklahoma State Electronics Champion, 1975. Valedictorian of Charles Page High School Class, Sand Springs, Oklahoma, 1975. At Oklahoma State University, won several scholarships: Regents Distinguished Scholarship; Sigma Xi Scholarship; Naeter Scholarship. Served as Secretary 1976-1977, President 1977-1978 of the student branch of the Institute of Electrical and Electronics Engineers; where he led the drive that tripled the membership of that branch. Won local and district Student Paper contest of the Institute of Electrical and Electronics Engineers, and placed fourth in Region V in 1978. Currently a member of the Institute of Electrical and Electronics Engineers, Eta Kappa Nu, Past Master of Bethany Masonic Lodge #529, and Life Member of the American Association of Individual Investors.

Publications: Temperature Dependent Parameter Analysis of Thermoelectric Devices, Institute of Electrical and Electronics Engineers Region V Annual Conference Energy '78, Tulsa, Oklahoma, 1978, pp. 167-170, coauthor K.R. Rao. With Allan O. Steinhardt, Reduction of Poisson Noise in the Gamma-ray Log, 1983-84 Program Final Report, Oklahoma State University Research Consortium for Enhancement of Well Log Data via Signal Processing, Stillwater, Oklahoma, 1984, pp. 2.1-2.53. Reduction of Noise in the Gamma-Ray Log, SPWLA Symposium Record, Dallas, Texas, 1985, Volume I, paper JJ.