VARIANCE FRACTAL DIMENSION FOR SIGNAL FEATURE ENHANCEMENT AND SEGMENTATION FROM NOISE

by

Warren S. Grieder

A Thesis
presented to the University of Manitoba
in partial fulfillment of
the requirements of the degree of
Master of Science
in
the Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: W. Kinsner, Ph.D., P.Eng.

September1996

© Warren Grieder, 1996

(ix + 84 + A-134 + B-113 + C-29) = 369 pp.

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-16145-5

Canada

Name _____

Dissertation Abstracts International and *Masters Abstracts International* are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

SUBJECT TERM

Electronic and Electrical Engineering

SUBJECT CODE 0544

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Design and Decorative Arts 0389
Fine Arts 0357
Information Science 0723
Journalism 0391
Landscape Architecture 0390
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Educational Psychology 0525
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291
- Rhetoric and Composition 0681

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- Caribbean 0360
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Church 0330
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Fisheries and Aquaculture 0792
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Range Management 0777
- Soil Science 0481
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Animal Physiology 0433
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Plant Physiology 0817
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Dentistry 0567
- Education 0350
- Administration, Health Care 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Safety 0354
- Oncology 0992
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Rehabilitaion and Therapy 0382
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Condensed Matter 0611
- Electricity and Magnetism 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Environmental 0775
- Industrial 0546
- Marine and Ocean 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Cognitive 0633
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Nom _____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

_____ ☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS
Architecture 0729
Beaux-arts 0357
Bibliothéconomie 0399
Cinéma 0900
Communication verbale 0459
Communications 0708
Danse 0378
Histoire de l'art 0377
Journalisme 0391
Musique 0413
Sciences de l'information 0723
Théâtre 0465

ÉDUCATION
Généralités 515
Administration 0514
Art 0273
Collèges communautaires 0275
Commerce 0688
Économie domestique 0278
Éducation permanente 0516
Éducation préscolaire 0518
Éducation sanitaire 0680
Enseignement agricole 0517
Enseignement bilingue et multiculturel 0282
Enseignement industriel 0521
Enseignement primaire. 0524
Enseignement professionnel 0747
Enseignement religieux 0527
Enseignement secondaire 0533
Enseignement spécial 0529
Enseignement supérieur 0745
Évaluation 0288
Finances 0277
Formation des enseignants 0530
Histoire de l'éducation 0520
Langues et littérature 0279
Lecture 0535
Mathématiques 0280
Musique 0522
Orientation et consultation 0519
Philosophie de l'éducation 0998
Physique 0523
Programmes d'études et enseignement 0727
Psychologie 0525
Sciences 0714
Sciences sociales 0534
Sociologie de l'éducation 0340
Technologie 0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE
Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

Littérature
- Généralités 0401
- Anciennes 0294
- Comparée 0295
- Médiévale 0297
- Moderne 0298
- Africaine 0316
- Américaine 0591
- Anglaise 0593
- Asiatique 0305
- Canadienne (Anglaise) 0352
- Canadienne (Française) 0355
- Germanique 0311
- Latino-américaine 0312
- Moyen-orientale 0315
- Romane 0313
- Slave et est-européenne 0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE
Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES
Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

Droit 0398
Économie
- Généralités 0501
- Commerce-Affaires 0505
- Économie agricole 0503
- Économie du travail 0510
- Finances 0508
- Histoire 0509
- Théorie 0511

Études américaines 0323
Études canadiennes 0385
Études féministes 0453
Folklore 0358
Géographie 0366
Gérontologie 0351
Gestion des affaires
- Généralités 0310
- Administration 0454
- Banques 0770
- Comptabilité 0272
- Marketing 0338

Histoire
- Histoire générale 0578
- Ancienne 0579
- Médiévale 0581
- Moderne 0582
- Histoire des noirs 0328
- Africaine 0331
- Canadienne 0334
- États-Unis 0337
- Européenne 0335
- Moyen-orientale 0333
- Latino-américaine 0336
- Asie, Australie et Océanie 0332

Histoire des sciences 0585
Loisirs 0814
Planification urbaine et régionale 0999
Science politique
- Généralités 0615
- Administration publique 0617
- Droit et relations internationales 0616

Sociologie
- Généralités 0626
- Aide et bien-être social 0630
- Criminologie et établissements pénitentiaires 0627
- Démographie 0938
- Études de l' individu et de la famille 0628
- Études des relations interethniques et des relations raciales 0631
- Structure et développement social 0700
- Théorie et méthodes. 0344
- Travail et relations industrielles 0629

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES
Agriculture
- Généralités 0473
- Agronomie. 0285
- Alimentation et technologie alimentaire 0359
- Culture 0479
- Élevage et alimentation 0475
- Exploitation des pâturages 0777
- Pathologie animale 0476
- Pathologie végétale 0480
- Physiologie végétale 0817
- Sylviculture et faune 0478
- Technologie du bois 0746

Biologie
- Généralités 0306
- Anatomie 0287
- Biologie (Statistiques) 0308
- Biologie moléculaire 0307
- Botanique 0309
- Cellule 0379
- Écologie 0329
- Entomologie 0353
- Génétique 0369
- Limnologie 0793
- Microbiologie 0410
- Neurologie 0317
- Océanographie 0416
- Physiologie 0433
- Radiation 0821
- Science vétérinaire 0778
- Zoologie 0472

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE
Biogéochimie 0425
Géochimie 0996
Géodésie 0370
Géographie physique 0368
Géologie 0372
Géophysique 0373
Hydrologie 0388
Minéralogie 0411
Océanographie physique 0415
Paléobotanique 0345
Paléoécologie 0426
Paléontologie 0418
Paléozoologie 0985
Palynologie 0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT
Économie domestique 0386
Sciences de l'environnement 0768
Sciences de la santé
- Généralités 0566
- Administration des hôpitaux 0769
- Alimentation et nutrition 0570
- Audiologie 0300
- Chimiothérapie 0992
- Dentisterie 0567
- Développement humain 0758
- Enseignement 0350
- Immunologie 0982
- Loisirs 0575
- Médecine du travail et thérapie 0354
- Médecine et chirurgie 0564
- Obstétrique et gynécologie 0380
- Ophtalmologie 0381
- Orthophonie 0460
- Pathologie 0571
- Pharmacie 0572
- Pharmacologie 0419
- Physiothérapie 0382
- Radiologie 0574
- Santé mentale 0347
- Santé publique 0573
- Soins infirmiers 0569
- Toxicologie 0383

SCIENCES PHYSIQUES
Sciences Pures
Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÈres 0495
- Radiation 0754

Mathématiques 0405
Physique
- Généralités 0605
- Acoustique 0986
- Astronomie et astrophysique 0606
- Électronique et électricité 0607
- Fluides et plasma 0759
- Météorologie 0608
- Optique 0752
- Particules (Physique nucléaire) 0798
- Physique atomique 0748
- Physique de l'état solide 0611
- Physique moléculaire 0609
- Physique nucléaire 0610
- Radiation 0756

Statistiques 0463

Sciences Appliqués Et Technologie
Informatique 0984
Ingénierie
- Généralités 0537
- Agricole 0539
- Automobile 0540
- Biomédicale 0541
- Chaleur et thermodynamique 0348
- Conditionnement (Emballage) 0549
- Génie aérospatial 0538
- Génie chimique 0542
- Génie civil 0543
- Génie électronique et électrique 0544
- Génie industriel 0546
- Génie mécanique 0548
- Génie nucléaire 0552
- Ingénierie des systèmes 0790
- Mécanique navale 0547
- Métallurgie 0743
- Science des matériaux 0794
- Technique du pétrole 0765
- Technique minière 0551
- Techniques sanitaires et municipales 0554
- Technologie hydraulique 0545

Mécanique appliquée 0346
Géotechnologie 0428
Matières plastiques (Technologie) 0795
Recherche opérationnelle 0796
Textiles et tissus (Technologie) 0794

PSYCHOLOGIE
Généralités 0621
Personnalité 0625
Psychobiologie 0349
Psychologie clinique 0622
Psychologie du comportement 0384
Psychologie du développement 0620
Psychologie expérimentale 0623
Psychologie industrielle 0624
Psychologie physiologique 0989
Psychologie sociale 0451
Psychométrie 0632

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

VARIANCE FRACTAL DIMENSION FOR SIGNAL FEATURE ENHANCEMENT AND SEGMENTATION FROM NOISE

BY

WARREN S. GRIEDER

A Thesis/Practicum submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Warren S. Grieder © 1996

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis/practicum, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis/practicum and to lend or sell copies of the film, and to UNIVERSITY MICROFILMS INC. to publish an abstract of this thesis/practicum..

This reproduction or copy of this thesis has been made available by authority of the copyright owner solely for the purpose of private study and research, and may only be reproduced and copied as permitted by copyright laws or with express written authorization from the copyright owner.

Abstract

This thesis presents a study of the variance fractal dimension as it applies to temporal signals for distinguishing between signal and noise, as well as for finding boundaries between signal features. Speech signals and packet radio signals are studied, with the primary focus on speech. The variance fractal dimension is calculated on small windows within the complete signal, based on the variance of increments for selected time increments. The temporal sequence of the variance fractal dimension values from sequential, evenly spaced windows form a variance fractal dimension trajectory. The method of time increment selection can affect the trajectory value for signals with a high degree of correlation between adjacent samples. Two main settings for the selection of the time increments are studied: one, where the time increments are selected at a unit distance spacing, is affected greatly by the correlation between adjacent samples, the other is more immune to the influences of adjacent sample correlations since a dyadic selection of time increments is used. These selection methods are referred to as noise separation and fractal amplification, respectively. In the case of speech, the noise separation trajectories exhibit a distinctive change in value between the uncorrelated ambient noise that is recorded when there is no utterance present, which generally has a dimension greater than 1.85, and the more correlated utterances, which have dimension values less than 1.40 initially, and less than 1.70 in general. Initial stop consonants and affricatives have level changes between 3×10^{-4} and 1×10^{-3} seconds in duration, with slopes between –770 and –2217 dimension units per second. Initial fricatives, nasals, and approximants have level changes between 1×10^{-2} to 2.7×10^{-2} seconds, with slopes between –18.6 and –92.4 dimension units per second. There is also a level difference between the highly correlated vowels, which have dimensions between 1.0 and 1.1, and the less correlated consonants, most of which have dimensions between 1.1 and 1.7. The fractal amplification trajectories enhance features within the utterances themselves. These features contain characteristic patterns for several of the phonemes. When the variance fractal dimension is applied to packet radio transmissions, the trajectory obtained exhibits changes in level at the boundaries between the background noise, the packet radio signal, and the transmitter noise.

Acknowledgements

I would like to thank the people that have assisted me at various stages during the course of this thesis. First and foremost, I would like to thank my thesis advisor, Dr. W. Kinsner, who encouraged me to embark upon the Master's program, for his help and guidance throughout this study, and for his patience during the course of this work.

Thanks go to Armein Langi and Ken Ferens, who have both been very helpful with their advice and encouragement throughout the thesis process.

Special thanks to Geoff Stacey for use of his code for the creation of TIFF files, which was modified to save images of the experimental data collected in this thesis.

Thanks to the other members of our research group, past and present: Adi Indrayato, Larry Wall, Tom Tessier, Eric Jang, Epiphany Vera, Shamit Bal, Hongjin Chen, Richard Dansereau, Randy Allan, Jason Toonstra, and Lawrence Arendt, who have also encouraged me throughout this process.

Thanks to current roommates Sean Kalynuk, Bruce Mager, and Cheri Frazer for their assistance with the maintenance of my computer and for allowing me to use their computer equipment in the production of this thesis.

Special thanks to former roommate and C++ guru Don Teissen, who readily spotted problems I had during the development of my code that I would have taken much longer to track down.

Thanks also go the Paula Ehn for showing me how to operate the radio equipment for recording the packet signals.

Finally, I would like to thank my family and friends for their patience and support during this process, without which this thesis would not have been possible.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AFSK	Audio Frequency Shift Keying
CELP	Code Excited Linear Prediction
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
FSK	Frequency Shift Keying
LPC	Linear Predictive Coding
IPA	International Phonetic Alphabet
MRPA	Machine Readable Phonetic Alphabet
PSD	Power Spectral Density
SB16	Soundblaster 16
TIFF	Tagged Image File Format
TNC	Terminal Node Controller
VOC	Creative Voice

CHAPTER I
INTRODUCTION

1.1 Background

The detection of signals in the presence of noise is a difficult process. When the energy level of a signal is high in comparison to the energy level of the noise when no signal is present, energy based calculations are sufficient to detect the presence of a signal. However, when the energy level of the noise is comparable to the signal's energy level, the energy calculations become inadequate and other techniques must be used to distinguish signals from noise.

Even in circumstances when there is a significant difference between the energy levels of signal and noise, similarities in the energy levels of different signal features severely limits the ability of the energy based calculations to distinguish between individual features. Other techniques must be used in these situations as well.

Two major examples of signals that occur in the presence of noise are speech and packet radio transmissions. The detection of the onset of individual speech features is critical for speech recognition techniques. The detection of the onset of the packet radio transmission is critical to avoid data loss.

One major difficulty with the automatic recognition of speech by a machine is the unlimited number of possible utterances. This is further complicated by the non-stationary and variable nature of speech, where even two utterances of the same phrase, while perceptually identical to a human listener, contain several differences that make template matching of complete phrases impractical due to the extensive memory and search time requirements for a system with a large vocabulary. Thus, it is necessary to segment utterances into smaller units, such as words, syllables or phonemes. When the size of the segmentation units is reduced there are fewer possible matching patterns that exist, thus reducing both the size of the search for matching elements and the amount of storage space required for pattern templates. Although these elements are still large on the word and syllabic level, they become more manageable on the phonemic level.

Several techniques, which utilize various parameterization of speech, have been developed that can perform speech recognition with varying degrees of success. Among the parameterizations used are linear predictive coding (LPC), used in formant tracking

techniques, and fast Fourier transforms, used in the neural phonetic typewriter [Koho88]. With more accurate identification of the segmentation boundaries, further analysis using these and other techniques on the speech segments may result in increased performance.

In packet radio transmissions, it is possible for the beginning of the packet to be obscured by the noise. If the period of time between the actual onset of the packet and the time of detection of the packet is too great, the terminal node controller (TNC) is not activated in time, and the packet is lost, requiring a retransmission of the packet.

The use of fractal dimension calculations makes it possible to determine more accurately the locations of the boundaries between signals and the surrounding noise as well as the boundaries between adjacent features with the signal itself [Kins94a].

This thesis demonstrates how the variance fractal dimension of a signal can be used in the determination of boundaries within a signal.

1.2 Problem Definition

Thesis Statement

The object of this thesis is to study the variance fractal dimension trajectories of speech for the possibility of their use in a system that can perform segmentation of speech based on their calculation.

Problem Specifications

If the data contained within the variance fractal dimension trajectories is to be used in a system for the segmentation of speech, it must meet the following requirements:

- It must identify the boundaries between speech and the background noise,
- It must identify the boundaries between characteristic features within the speech itself.
- It should reduce the number of possible choices of corresponding phonemes for each segment if possible.

The ability to identify the boundaries between speech and surrounding noise is needed to identify the boundaries between larger units of speech, i.e. word and sentences. The ability to identify the boundaries between characteristic feature within the speech provides the starting and stopping points for the individual phonemes, thus increasing the accuracy of other analysis techniques that are applied to these segments. The ability to

provide partial identification for some of the phoneme further increases the accuracy of the other analysis techniques that are used.

The input for the system that produces the variance fractal dimension trajectories consists of speech sampled at 44.1 kilosamples per second. The sampling is performed using a Creative Labs Soundblaster 16 (SB16) sound card.

1.3 Thesis Organization

The process used in this study is reflected in the organization of this thesis. Chapter I provides the background and motivation of the thesis, the problem definition, and the thesis organization. These are necessary to introduce what is examined in this thesis.

Chapter II describes the production and techniques that can be used to analyze the types of signals that are examined in this thesis. The production of speech and more specifically the production of individual phonemes are described. The AX.25 protocol and the properties of the packet signals are also described.

Chapter III discusses the use of fractal dimension measurements and how they can be used to establish a trajectory waveform as a parameter of speech.

Chapter IV describes the architecture, organization, and implementation of the fractal dimension trajectory calculation system. The system analyzes an input signal and calculates a series of fractal dimension values based on signal segments spaced at regular intervals. This series forms the fractal dimension trajectory.

Chapter V describes the experiments performed, presents the results, and analyzes the results.

Chapter VI presents the conclusions for this thesis, and highlights its achievements and contributions. This chapter also gives the recommendations for this thesis.

References and Appendices, containing illustrations of all observed trajectories, including those not included elsewhere in this thesis, listings of the software used in the creation of these trajectories, and the AX.25 protocol are located at the end of this thesis.

CHAPTER II
PRODUCTION AND ANALYSIS OF NOISE BOUNDED SIGNALS

There are many types of signals that can be found in noisy environments. Speech and packet radio transmissions are two examples of signals that occur in such environments. The noise affecting a speech signal may be background noise from the environment in which the speech was uttered, or may be introduced within the channel over which the speech is being transmitted [Newe75]. In the case of packet radio, the noise can be background noise originating from sunspot activity or atmospheric conditions. In both these cases when no signal is being sent the background noise is still present.

2.1 The mechanics of speech production

The production of speech through the interactions within the system of speech organs, shown in Fig. 2.1 [Lang92]. This system of speech organs can be subdivided into three subsystems: the pulmonary tract, the larynx, and the vocal tract. The pulmonary tract consists of the lung and the trachea which provide the air flow required for speech. The larynx contains the vocal cords which convert the air flow into pulses. The vocal tract contains the pharynx, the oral cavity and the nasal cavity, in which the air pulses are resonated to produce the sounds that are recognized as speech.

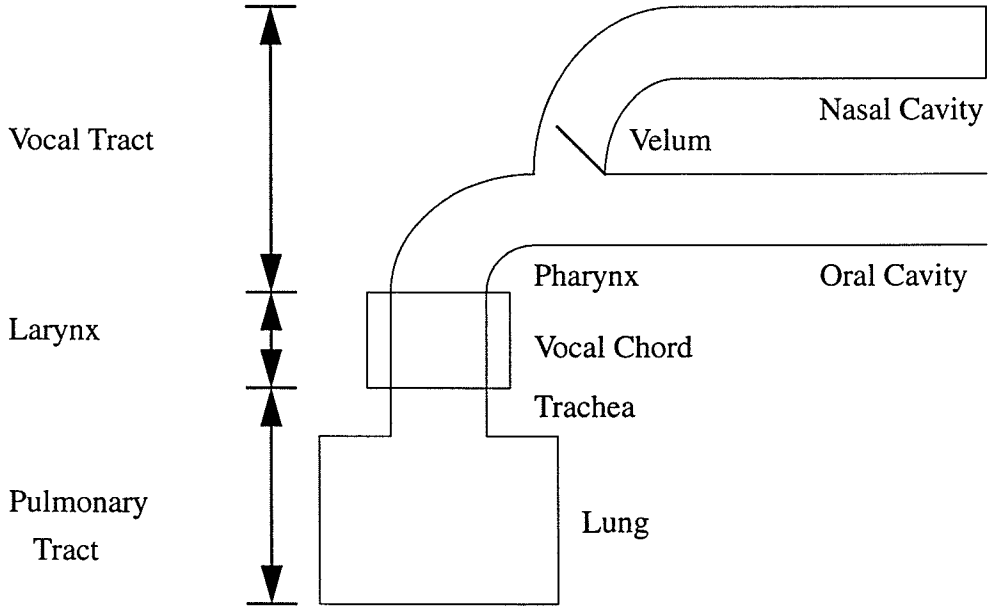
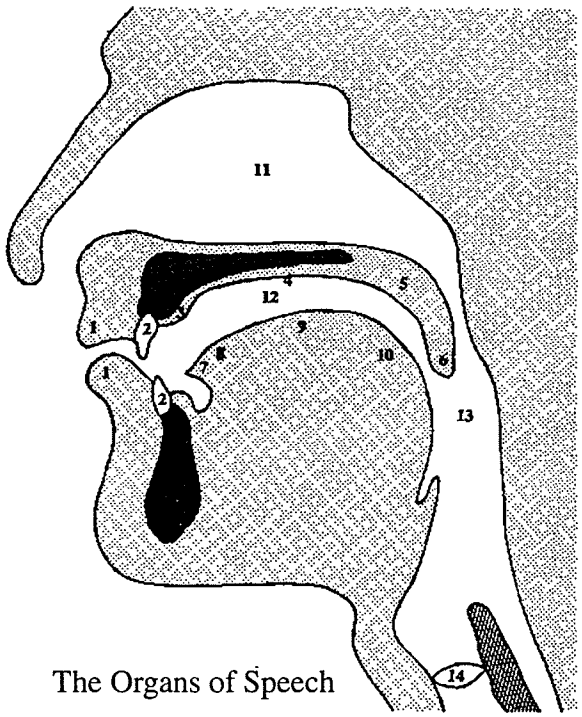


Fig. 2.1. A speech production model. (After [Lang92]).

The manner of the interactions of the larynx and vocal tract with the air flow from the lungs influences the type of sound produced. These interactions are known as articulations. The specific organs interacting with the air flow are known as the articulators. Figure 2.2 shows the location of the various parts of the speech system that can act as articulators. The differences between the differently articulated sounds make it possible to distinguish between the phonemes.



The Organs of Speech

1. Lips
2. Teeth
3. Alveolar ridge
4. Hard palate
5. Soft palate
6. Uvula
7. Tip of the tongue
8. Blade of the tongue.
9. Front of the tongue.
10. Back of the tongue.
11. Nasal cavity
12. Oral cavity
13. Pharynx
14. Larynx

Fig. 2.2. The speech organs. (From [Carr93]).

2.2 The Articulation of the English Phonemes

Phonemes are the members of the set of the smallest speech units that distinguish different utterances from each other in a language. Table 2.1 shows the Machine Readable Phonetic Alphabet (MRPA) and International Phonetic Alphabet (IPA) representations of the English phonemes and a corresponding keyword containing the phoneme. There are two categories of phonemes: consonants and vowels. The consonants are produced when the airflow from the lungs is obstructed in the middle of the vocal tract. The vowels are produced when this obstruction does not occur.

2.2.1 Consonants

The consonants can be divided into several type subcategories based on different descriptive parameters. The descriptive parameters that specify the consonants are the voicing state, the place of articulation, and manner of articulation [Carr93]. The voicing state for the consonants is determined by the presence or absence of vocal cord vibration. These states are called voiced and voiceless respectively.

The manner of articulation is specified by the degree to which the flow of air is impeded by the articulators. Stops and nasals result from complete closure of the articulators, with the nasal also accompanied by a lowering of the velum [Carr93]. Fricatives are produced by close approximation, in which a narrow opening between the articulators results in turbulent air flow [Bris86, Carr93]. Approximants are produced by open approximation, in which the opening between articulators is wide enough to avoid turbulent airflow [Carr93].

The place of articulation defines the part of the vocal tract that is acting as the articulator. Bilabial sounds are articulated by the lips. Labiodental sounds are articulated by the lower lip and upper teeth. Dental sounds are articulated with the tip of the tongue and the upper teeth. Alveolar sounds are articulated with the tip of the tongue and the alveolar ridge. Post-alveolar sounds are articulated behind the alveolar ridge. Palato-alveolar sounds are articulated with the blade of the tongue and the palato-alveolar region. Palatal sound are articulated with the front of the tongue and hard palate. Velar sounds are articulated by the back of the tongue and the soft palate. Uvular sound are articulated by

the back of the tongue and the uvula. Pharyngeal sound are articulated by the walls of the pharynx. Glottal sounds are articulated by the vocal cords.

The various combinations of these three parameters can be used to distinguish the phonemes as shown in Table 2.2.

Table 2.1. English Phoneme Representations and Keywords. (After [Bris86], [DoSh88]).

MRPA	IPA	word	MRPA	IPA	word
/p/	p	*pip*	/y/	j	*y*ear
/b/	b	bar*b*	/w/	w	*w*eal
/t/	t	*t*igh*t*	/ii/	i	b*ea*d
/d/	d	*d*ee*d*	/i/	ɪ	b*i*d
/k/	k	*k*ic*k*	/e/	ɛ	b*e*d
/g/	g	*g*a*g*	/a/	æ	b*a*d
/ch/	tʃ	*ch*ur*ch*	/aa/	ɑə	b*ar*d
/jh/	dʒ	*j*ud*g*e	/o/	ɑ	b*o*dy
/f/	f	*f*i*f*e	/oo/	oə	f*or*
/v/	v	*v*er*v*e	/u/	ʊ	b*oo*k
/th/	θ	*th*irtie*th*	/uu/	u	b*oo*t
/dh/	ð	o*th*er	/uh/	ʌ	b*u*d
/s/	s	*c*ea*s*e	/@@/	ʌə	b*ir*d
/z/	z	*z*oo*s*	/@/	ə	ba*n*an*a*
/sh/	ʃ	*sh*eepi*sh*	/ei/	eɪ	b*ay*
/zh/	ʒ	mea*s*ure	/ou/	o	b*oa*t
/h/	h	*h*and	/ai/	aɪ	b*uy*
/r/	r	*r*ear	/au/	aʊ	b*ou*gh
/l/	l	*l*oya*l*	/oi/	ɔɪ	b*oy*
/m/	m	*m*i*m*e	/i@/	ɪə	b*ee*r
/n/	n	*n*o*n*e	/e@/	ɛə	b*ea*r
/ng/	ŋ	ri*ng*i*ng*	/u@/	ʊə	p*oo*r

Table 2.2. Consonants of English. (After [Bris86]).

		bilabial	labio-dental	dental	alveolar	postalveolar	palatal	velar	uvular	pharyngeal	glottal
STOPS	voiceless	p			t			k			
	voiced	b			d			g			
NASALS		m			n			ŋ			
AFFRICATIVES	voiceless					tʃ					
	voiced					dʒ					
FRICATIVES	voiceless		f	θ	s	ʃ					
	voiced		v	ð	z	ʒ					
APPROXIMANTS	central	w				r	j	(w)			h
	lateral				l						

2.2.2 Vowels

Monophthongal vowels

The monophthongal vowels can also be described by a set of descriptive parameters. The articulatory position for these vowels is specified by the height of the tongue body, the front/back position of the tongue body, and the presence or absence of lip rounding.

The height of the tongue body is distinguished between its extremes, close and open. The tongue body is near the hard or soft palate for the close vowels, while it is as far as possible from the roof of the mouth for the open vowels. Two intermediate heights: half-open and half-close describe the remaining vowels.

The front-back dimension is divided into three categories: front, central, and back. The tongue is as far forward as it can go for the front vowels, as far back as it can go for the back vowels, and at an intermediate position for the central vowels.

Rounded vowels are produced by spread lips (lip rounding). Vowels produced without lip rounding are called unrounded vowels.

Figure 2.2 illustrates the monophthongal vowels in the English language. In this diagram the rounded vowels are /i/, /u/, /o/, and, /ɔ/.

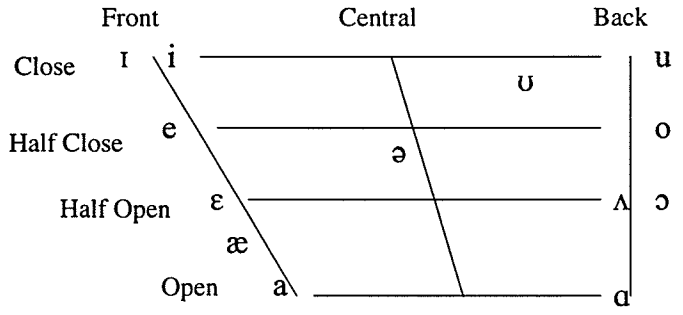


Fig. 2.3. The monophthongal English vowels. (After [Carr93]).

Diphthongal Vowels

The diphthongal vowels or diphthongs are articulated starting in the position of one monophthongal vowel and glide through a transition to the position of another monophthongal vowel. For example, the diphthong /aɪ/ begins its articulation as close, back, and unrounded and undergoes a transition to close, front, and unrounded.

2.3 Segmentation of Speech

Speech can be considered as a structure built up of many substructures. The identification of the boundaries of these substructures is necessary for the segmentation of speech [Dene75]. There are many levels of these substructures, as shown in Fig. 2.4.

Continuous Speech

Continuous speech is produced by a speaker as a means of communicating ideas to a listener. This unsegmented waveform is the main speech structure and is composed of the substructures at the various levels of segmentation.

Words

Words are the smallest segment of speech containing meaning to a listener. When combined together these words form the ideas contained within the phrases and sentences of continuous speech.

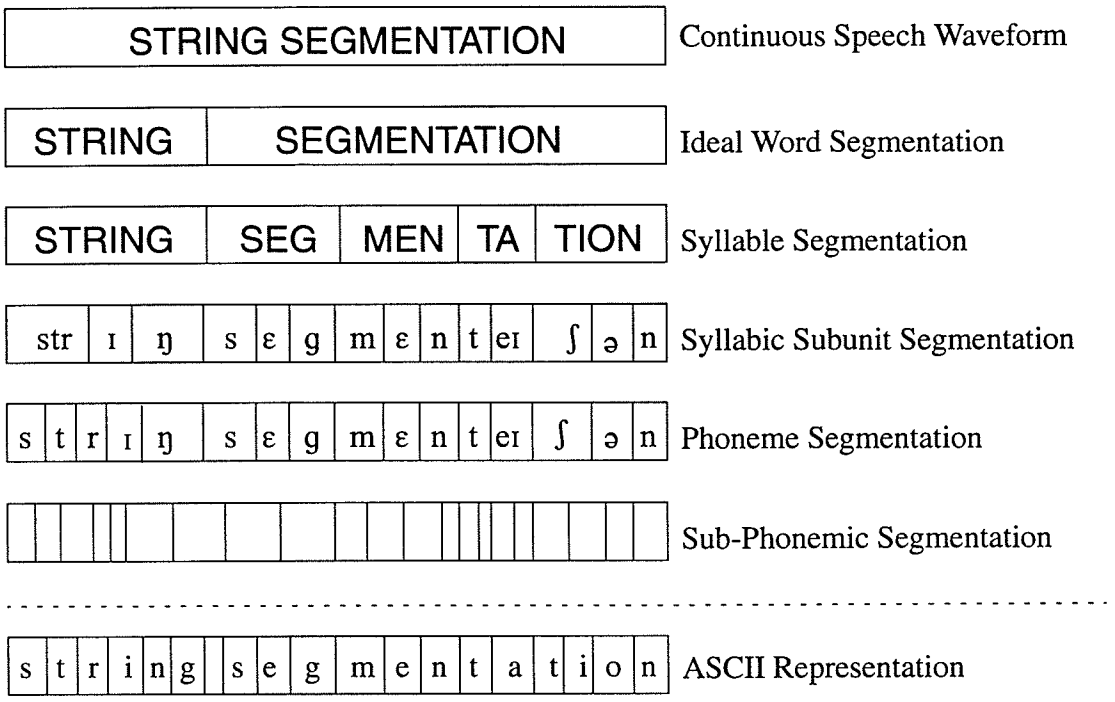


Fig. 2.4. Different levels of speech segmentation. (After [Bris86]).

Syllables

Individual words can be segmented into syllables. Syllables consist of a syllabic nucleus (a vowel or a syllabic consonant) either alone or preceded and/or followed by one or more constants.

Syllabic Subunits

The syllables can be further segmented into syllabic subunits. These subunits are either the syllabic nucleus of the syllable or a group of consonants that precede or follow the syllabic nucleus.

Phonemes

The phonemes are the smallest units of speech that can be used to distinguish utterances. Further segmentation of some phonemes into sub-phonemic units is possible. The stop gaps in the stop constants and the starting and ending articulatory positions of diphthongal vowels serve as examples of the sub-phonemic units.

2.4 Speech Analysis Techniques

There are many techniques that can be used in the analysis of speech. Some of the more popular techniques include formants, the fast Fourier transform (FFT), linear predictive coding (LPC), and energy based calculations.

2.4.1 Formants

One technique that has been used in the classification of the vowel sounds is the determination of the dominant frequencies in the power spectral density (PSD) of the speech waveform [Bris86, Furu89]. These dominant frequencies are called the formant frequencies. These frequencies are usually denoted as F_1, F_2, and F_3 in order of ascending frequency. The relationship of the first and second formants to the articulatory formation is shown in Fig. 2.5.

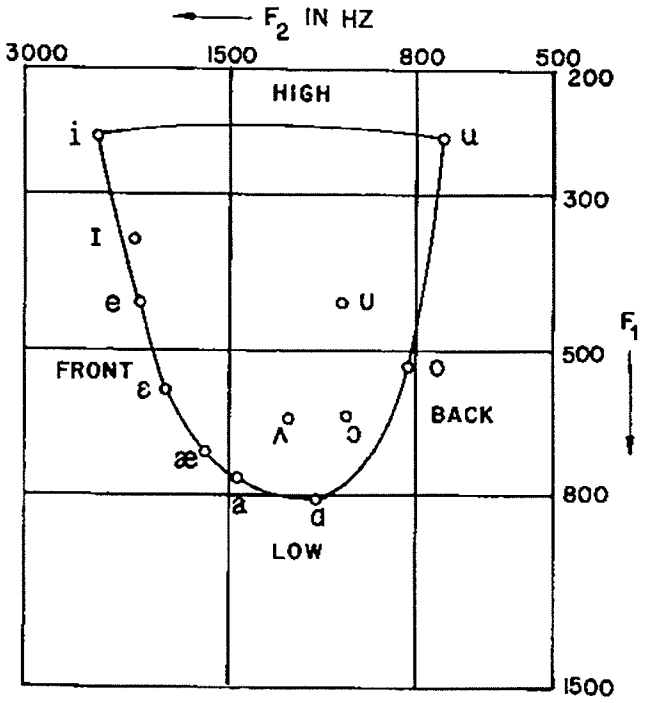


Fig. 2.5. Relation of formants to articulatory formations. (From [BrSh75]).

One major problem associated with the formants is the amount of variation that occurs from speaker to speaker or even within the speech of a single speaker depending on the adjacent phonemes. This is illustrated by Fig. 2.6, which shows the scatter diagram of 10 English vowels in the F_1-F_2 plane. Another problem with the formant frequencies,

which becomes apparent from the scatter diagram, is that the regions associated with the individual vowels are not distinct and in fact overlap.

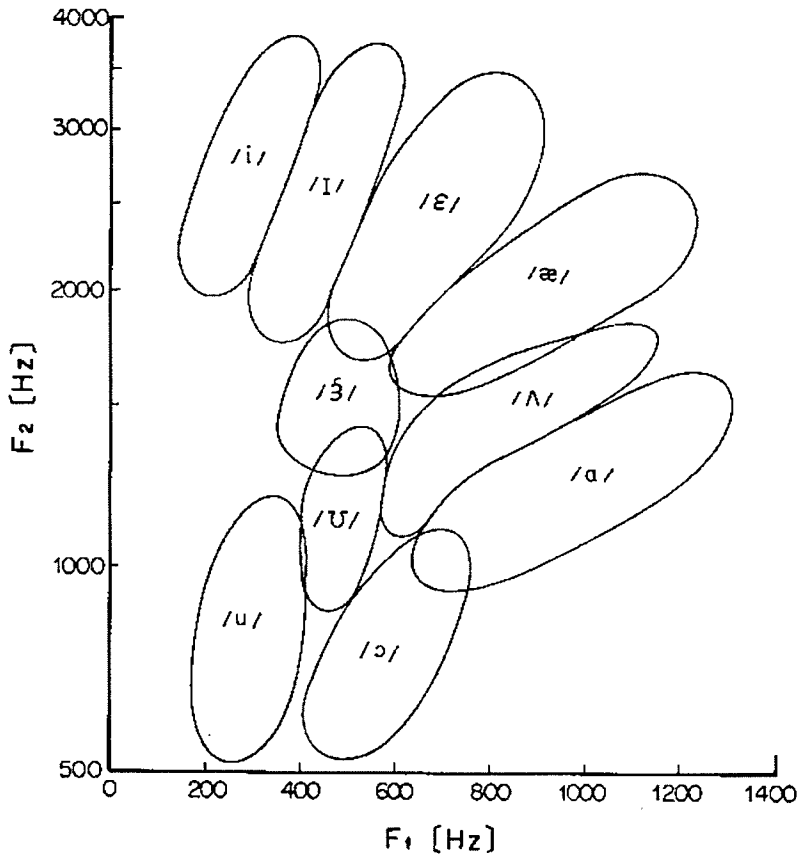


Fig. 2.6. Formant frequency scatter diagram. (From [Furu89]).

A further problem with formants is that they are either in transition or non-existent for consonants. The classification of consonants requires either different parameters such as the periodicity of waves, frequency spectrum, duration, and temporal variation [Furu89] or detailed analysis of the formant transitions [Harr88].

2.4.2 Discrete Fourier Transform

The values of the spectrum, $X(f)$, of a waveform, $x[n]$, at multiples of a frequency, f_0, can be calculated on a window of N samples using the discrete fourier transform (DFT) [Couc90] which is defined by

$$X(f) = \sum_{n=0}^{N-1} x[n]\, e^{-j(2\pi/N)fn} \tag{2.1}$$

where

$$f = cf_0, \qquad c \in I, -\frac{N}{2} < c < \frac{N}{2} \tag{2.2}$$

The frequency, f_0, is related the sampling frequency, f_s, by

$$f_s = Nf_0 \tag{2.3}$$

The DFT can be calculated more efficiently using FFT algorithms, which reduces the complexity of the problem from $O(N^2)$ to $O(N \log N)$. The spectrum obtained from the FFT has been used as the input the neural phonetic typewriter [Koho88].

The PSD of a waveform at multiples of the sampling frequency can be obtained by multiplying the spectrum values at these frequencies by their complex conjugates. The formant frequencies can be determined by locating the peaks in the PSD.

The DFT suffers from three types of error: leakage, aliasing and the picket-fence effect [Couc90]. Leakage is the result of considering a window within the entire signal which causes the components of the spectrum to spread. Aliasing results from a violation of the Nyquist criterion and does not occur in a properly sampled signal. The picket-fence effects results from the inability to determine spectral components intermediate to the multiples of f_0.

2.4.3 Neural Phonetic Typewriter

The neural phonetic typewriter, mentioned above, uses the FFT as the input to a neural net. This neural net performs a training process of vector quantization on this data to obtain a phonotopic map [Koho88]. After the training process is completed, an utterance presented to the neural network produces a sequence of responses from the phonotopic map. The path this sequence takes through the map forms a trajectory of the utterance within the map [Koho88].

2.4.4 Linear Predictive Coding

LPC analysis is based on an all-pole model of the vocal tract [Bris86]. This model is used in many applications, such as secure speech communication and speech recognition systems [Wein91]. The parameters obtained from LPC analysis are the coefficients α_i of the first order linear equation between the present sample $x[n]$ and the previous p samples as given by

$$x[n] + \sum_{i=1}^{p} \alpha_i x[n-p] = \varepsilon[n] \tag{2.4}$$

where $\varepsilon[n]$ is an uncorrelated statistical variable with a mean of 0 and a variance of σ^2.

The predicted value $\hat{x}[n]$ for $x[n]$ is given by

$$\hat{x}[n] = \sum_{i=1}^{p} \alpha_i x[n-p] \tag{2.5}$$

which combined with Eq. 2.4 gives

$$x[n] - \hat{x}[n] = \varepsilon[n] \tag{2.6}$$

where $\varepsilon[n]$ is designated as the residual error.

The LPC coefficient can be estimated by applying the least mean square error method to Eq. 2.6. The total square error β over the interval $[t_0, t_1]$ is

$$\beta = \sum_{t=t_0}^{t_1} \varepsilon^2[n] \tag{2.7a}$$

$$= \sum_{t=t_0}^{t_1} \left(\sum_{i=0}^{p} \alpha_i x[n-i] \right)^2 \tag{2.7b}$$

$$= \sum_{t=t_0}^{t_1} \sum_{i=0}^{p} \sum_{j=0}^{p} \alpha_i \alpha_j x[n-i]\, x[n-j] \tag{2.7c}$$

where $\alpha_0 = 1$. Defining

$$c_{ij} = \sum_{t=t_0}^{t_1} x[n-i]\, x[n-j] \tag{2.8}$$

gives

$$\beta = \sum_{i=0}^{p'} \sum_{j=0}^{p} \alpha_i c_{ij} \alpha_j \tag{2.9}$$

β is minimized by setting the partial derivative of β with respect to α_j for $j = 1, 2, \ldots, p$ and solving the p simultaneous linear equations

$$\frac{\partial \beta}{\partial \alpha_j} = 2 \sum_{i=0}^{p} \alpha_i c_{ij} \qquad (j = 1, 2, \ldots, p) \tag{2.10}$$

LPC has problems with nasals and many consonants that do not conform to the all-pole model [Atal75, Bris86].

2.4.5 Energy Calculations

The energy, in a sample signal is defined by

$$E = \sum_{n=-\infty}^{\infty} x^2(n) \tag{2.11}$$

For non-stationary signals like speech it is necessary to define the energy over a short window as

$$E = \sum_{m=0}^{N-1} [w(m)\,x(n-m)]^2 \qquad (2.12)$$

where $w(m)$ is a window function ($w(m) = 1$ in the simplest case) which selects a segment of $x[n]$, and N is the number of samples in the window [ScRa75]. The duration of this window must be of shorter duration than the shortest signal feature (typically 10 ms is sufficient for speech).

Energy calculations have difficulty differentiating between adjacent features that have a similar energy level. It is also difficult to distinguish between noise and signal at similar energy levels.

2.5 Packet Radio

Packet radio transmissions consist of computer data that has been converted into variable audio tones by a terminal node controller (TNC). The TNC acts as an interface between a computer and a transceiver, converting the data to tones for outgoing packets and from the tones for incoming packets.

Individual packets in a packet radio transmission become lost if the packet is not detected because of the background noise. This requires the retransmission of the lost packet. Packets are also lost through collisions that occur when two packets are being sent at the same time by different transmitters. One cause of these collisions is the failure of one transmitter to detect a packet already being transmitted by the other transmitter because it was obscured by the background noise. The accurate determination of the beginning of a packet from a noisy channel significantly reduces the number of packets that must be retransmitted.

2.5.1 AX.25 Protocol

The AX.25 protocol defines the data structure to be used to transport data reliably between two signalling terminals. The individual packets are small blocks of data, called frames. These frames are composed of smaller groups of data called fields.

There are three types of frames: supervisory, unnumbered, and information. The supervisory frames are responsible for providing link control, such as acknowledging received information frames or requesting the retransmission of information frames, and link-level window control. Unnumbered frames provide additional link control beyond that of the supervisory frames. These frames are also responsible for establishing and terminating links. Information frames contain the actual data to be transmitted between the two terminals. Information frames are numbered with the current sequence number and the next expected sequence number.

All frames are delimited by 8-bit flag fields having a value of 7E hex. A flag may be shared by two frames, which indicates the end of the first frame and the beginning of the second. All frames also include an address frame that identifies the source and destination of the transmission. An 8-bit control field identifies the frame type. Information frames contain an 8-bit protocol identifier field that identifies the layer 3 protocol that is used in the packet. The information frames also include information fields of up to 256*8 bits long containing the user data being transmitted. All frames contain a 16 bit frame check sequence that is calculated on both ends to ensure that the frame was not corrupted in transmission.

All fields are sent least significant bit first except the frame check sequence which is sent most significant bit first. If five 1 bits are sent in a row a zero bit is stuffed in the transmission and discarded by the receiver to prevent the identification of a false flag.

The AX.25 Protocol is listed in its entirety in Appendix C.

2.5.2 Packet Signal Properties

The most common packet signal is a 1200 baud signal with the 1's and 0's in the data stream represented by modulated audio frequency shift keying (AFSK) tones at 1200 and 2200 Hz. There are also 9600 baud packet signals using frequency shift keying (FSK) shifting between two radio frequency equidistant from a center frequency [Kurt94].

Although formant frequencies have no meaning in packet radio signals, the other techniques previously mentioned can be used in the analysis of the packet radio signals.

2.6 Summary

In this chapter, the production of both speech signals and packet radio signals are described. Both types of signals are structures built of small component elements, with each element having its own unique set of properties. The set of phonemes compose the set of component elements of speech, which can be combined to form larger structures such as words and sentences. Similarly the packet radio signals are composed of the set of bits (1 and 0), which form larger structures such as fields and frames. The speech signals are produced by the manipulation of the organs in the vocal tract. The packet radio signals are produced by the manipulation of the frequencies being transmitted.

Boundary detection plays an important role for both of these signal types. Missing the beginning of utterance can cause a loss of meaning and necessitate a repetition of the utterance. Missing the beginning of a packet transmission results in a lost packet which requires that the packet be retransmitted.

There are many types of analysis techniques that can be used on speech, several of which can be extended for the analysis of other temporal signals. In Chapter III, fractal based techniques that can examine the nature of the noise in a signal are discussed.

CHAPTER III
FRACTAL DIMENSION TRAJECTORIES OF SIGNALS

In Chapter II, the concept of a trajectory within a phononic map produced by a neural network was mentioned. In this chapter, the concept of a trajectory obtained from the fractal dimension of a temporal signal is introduced. These fractal dimension trajectories consist of a sequence of localized fractal dimensions calculated on a window within the temporal signal for which they are being calculated.

3.1 Fractal Dimension Measurements

The high degree of geometrical complexity of fractals allows them to model many natural phenomena [MaSu89, MaSu93]. One dimensional signals, such as speech, can be considered as fractals because their graph is a fractal set [MaSu89, MaSu93]. The fractal dimension of such a signal measures the degree of boundary fragmentation over multiple scales [MaSu93].

3.1.1 Hausdorff

The Hausdorff dimension provides the definition of the fractal dimension calculation. For sets A contained in Euclidean space,

$$\mathbf{R}^n = \{x | x = (x_1, \ldots, x_n), x \in \mathbf{R}\} \tag{3.1}$$

where n is a natural number, the Hausdorff dimension can be defined as follows [PeJS92].

Given the Euclidean distance function d(x,y) for x and y in $\mathbf{R}^n$,

$$d(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2} \tag{3.2}$$

and the infimum, which is the greatest lower bound on a subset of real numbers, and the supremum, which is the least upper bound on a subset of real numbers, of a subset X of real numbers

$$\inf\{x \in X\} = \text{largest lower bound of } X \tag{3.3a}$$
$$\sup\{x \in X\} = \text{smallest upper bound of } X \tag{3.3b}$$

the diameter of subset U of $\mathbf{R}^n$ can be defined by

$$\text{diam}(U) = \sup\{d(x, y) | x, y \in U\} . \tag{3.4}$$

A subset U of $\mathbf{R}^n$ is open if for any $x \in U$ there exists a ball $B_\varepsilon(x) = \{y \in \mathbf{R}^n \,|\, (d(x, y) < \varepsilon)\}$ of radius $\varepsilon > 0$ centered at x with $B_\varepsilon(x) \subseteq U$. A family of open subsets $\{U_1, U_2, U_3, \ldots\}$ is an open cover of a subset A of $\mathbf{R}^n$ if

$$A \subset \bigcup_{i=1}^{\infty} U_i \tag{3.5}$$

Let s and ε be positive real numbers. The Hausdorff dimension of A is defined by

$$h_\varepsilon^s(A) = \inf\left\{ \sum_{i=0}^{\infty} \text{diam}(U_i)^s \,\middle|\, \begin{array}{l} U_1, U_2, \ldots \text{is an open cover} \\ \text{of A with } \text{diam}(U_i) < \varepsilon \end{array} \right\} \tag{3.6}$$

and the s-dimensional Hausdorff measure of A is defined by

$$h^s(A) = \lim_{\varepsilon \to 0} h_\varepsilon^s(A) . \tag{3.7}$$

It was shown by Hausdorff that for any set A, there exists a number $D_H(A)$ such that

$$h^s(A) = \begin{cases} \infty \text{ for } s < D_H(A) \\ 0 \text{ for } s > D_H(A) \end{cases} \tag{3.8}$$

The number $D_H(A)$ is defined as the Hausdorff dimension

$$D_H(A) = \inf\{s|h^s(A) = 0\} = \sup\{s|h^s(A) = \infty\} \tag{3.9}$$

The calculation of the $\sum_{i=0}^{\infty} \text{diam}(U_i)^s$ terms makes the Hausdorff dimension difficult to calculate. Other techniques of calculating the fractal dimension can be used to simplify the calculation.

3.1.2 Choice of Dimension Calculation Technique

Many of the techniques that can be used involve the use of a covering set similar to balls used in the Hausdorff dimension, such as the boxes in the box dimension and the disks in the mass dimension [PeJS92]. These techniques can be used effectively on two dimensional images having horizontal and vertical axis with similar resolutions. However, when considering temporal signals where the dimension is calculated on a relatively small window (2^8 - 2^{10} samples in the horizontal resolution) compared to the possible range of the sample value (2^{16} values in the vertical resolution), their implementation can be quite tedious. It is much more efficient to use a dimension calculation technique that operates directly on the sample values themselves. Two such techniques are the variance fractal dimension and the spectral fractal dimension.

3.1.3 Variance Fractal Dimension

The variance fractal dimension is based on the properties of fractional Brownian motion [ZhBM90]. The calculations that are required for the variance fractal dimension are performed on samples separated by various time increments, making the variance fractal dimension a natural choice in the study of temporal signals.

Variance

The variance, σ^2, of a random variable is defined by the second moment taken about the mean

$$\sigma^2 = \mu_2 = E[(y-\mu_1)^2] = E(y^2) - \mu_1^2 \tag{3.10}$$

where E denotes the expectation function, y is the random variable and, μ_1 is the mean.

If this equation is applied to a finite number of samples, n, of the random variable, $y_1, y_2, \ldots, y_n$, an estimator of σ^2 is obtained

$$s^2 = \frac{\sum_{i=1}^{n} (y_i)^2}{n} - \left(\frac{\sum_{i=1}^{n} y_i}{n} \right) \tag{3.11a}$$

$$s^2 = \frac{\sum_{i=1}^{n} (y_i)^2}{n} - \frac{\left(\sum_{i=1}^{n} y_i \right)^2}{n^2} \tag{3.11b}$$

$$s^2 = \frac{\sum_{i=1}^{n} (y_i)^2 - \frac{\left(\sum_{i=1}^{n} y_i \right)^2}{n}}{n} \tag{3.11c}$$

where s^2 is the sample variance.

By manipulating Eq. 3.11c

$$s^2 = \frac{\sum_{i=1}^{n} (y_i)^2 - n\bar{y}^2}{n} \tag{3.12a}$$

$$s^2 = \frac{\sum_{i=1}^{n} y_i^2 - 2n\bar{y}^2 + n\bar{y}^2}{n} \tag{3.12b}$$

$$s^2 = \frac{\sum_{i=1}^{n} y_i^2 - 2\bar{y}n\frac{\sum_{i=1}^{n} y_i}{n} + \sum_{i=1}^{n} \bar{y}^2}{n} \tag{3.12c}$$

$$s^2 = \frac{\sum_{i=1}^{n} y_i^2 - 2\bar{y}\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \bar{y}^2}{n} \tag{3.12d}$$

$$s^2 = \frac{\sum_{i=1}^{n} (y_i^2 - 2\bar{y}y_i + \bar{y}^2)}{n} \tag{3.12e}$$

$$s^2 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n} \tag{3.12f}$$

is obtained where $\bar{y}$ is the mean of the set of samples.

By adding and subtracting μ_1, the mean of the random variable

$$s^2 = \frac{\sum_{i=1}^{n} ((y_i - \mu_1) - (\bar{y} - \mu_1))^2}{n} \tag{3.13a}$$

$$s^2 = \frac{\sum_{i=1}^{n} ((y_i - \mu_1)^2 - 2(y_i - \mu_1)(\bar{y} - \mu_1) + (\bar{y} - \mu_1)^2)}{n} \tag{3.13b}$$

$$s^2 = \frac{1}{n}\left[\sum_{i=1}^{n} (y_i - \mu_1)^2 - 2(\bar{y} - \mu_1)\sum_{i=1}^{n} (y_i - \mu_1) + n(\bar{y} - \mu_1)^2\right] \tag{3.13c}$$

$$s^2 = \frac{1}{n}\left[\sum_{i=1}^{n}(y_i-\mu_1)^2 - 2n(\bar{y}-\mu_1)\left(\frac{\sum_{i=1}^{n} y_i - n\mu_1}{n}\right) + n(\bar{y}-\mu_1)^2\right] \tag{3.13d}$$

$$s^2 = \frac{1}{n}\left[\sum_{i=1}^{n}(y_i-\mu_1)^2 - 2n(\bar{y}-\mu_1)^2 + n(\bar{y}-\mu_1)^2\right] \tag{3.13e}$$

$$s^2 = \frac{1}{n}\left[\sum_{i=1}^{n}(y_i-\mu_1)^2 - n(\bar{y}-\mu_1)^2\right] \tag{3.13f}$$

Since the samples $y_1, y_2, \ldots, y_n$ are selected from a population with a mean of μ_1 and a variance of σ^2

$$E[(y_i-\mu_1)^2] = \sigma^2 \tag{3.14a}$$

$$E[(\bar{y}-\mu_1)^2] = \frac{\sigma^2}{n} \tag{3.14b}$$

Taking the expectation value of s^2 gives

$$E(s^2) = E\left\{\frac{1}{n}\left[\sum_{i=1}^{n}(y_i-\mu_1)^2 - n(\bar{y}-\mu_1)^2\right]\right\} \tag{3.15a}$$

$$E(s^2) = \frac{1}{n}\left\{\sum_{i=1}^{n}E[(y_i-\mu_1)^2] - E[n(\bar{y}-\mu_1)^2]\right\} \tag{3.15b}$$

$$E(s^2) = \frac{1}{n}\left\{\sum_{i=1}^{n}E[(y_i-\mu_1)^2] - nE[(\bar{y}-\mu_1)^2]\right\} \tag{3.15c}$$

$$E(s^2) = \frac{1}{n}\left(\sum_{i=1}^{n}\sigma^2 - n\frac{\sigma^2}{n}\right) \tag{3.15d}$$

$$E(s^2) = \frac{1}{n}(n\sigma^2 - \sigma^2) \tag{3.15e}$$

$$E(s^2) = \frac{n-1}{n}(\sigma^2) \tag{3.15f}$$

Since the expected value of the estimation given in Eq. 3.11c is not equal to σ^2, it is biased, with the amount of bias being a factor of $\frac{n-1}{n}$. An unbiased estimator can be obtained by multiplying Eq. 3.11c, by a factor of $\frac{n}{n-1}$ to obtain

$$s^2 = \left(\frac{n}{n-1}\right)\frac{\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}}{n} \tag{3.16a}$$

$$s^2 = \frac{\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}}{n-1} \tag{3.16b}$$

Brownian Motion

The concept of Brownian motion was discovered in 1828 by botanist Robert Brown, who realized there was a physical explanation, rather that a biological one, for the erratic motion of small particles of matter suspended in a liquid [PeJS92]. The motion of the individual particles is altered by collisions with surrounding molecules.

One dimensional Brownian motion restricts the motion of the particles to a line, with the displacements being a given length, l, so that the displacements are either $+l$ or $-l$ with probabilities of 0.5. Since the individual displacements are of opposite signs with equal magnitudes and probabilities, the total expected displacement over a number, n, of individual displacements is 0, making it necessary to examine the square of displacement. The average of the square displacement, called the mean square displacement, measures the amount of spread the particles experience in n time steps [PeJS92].

Each path in a time step branches into two paths in the next time step. Thus, from the initial point at time zero ($n = 0$), there are two paths leading to first time step ($n = 1$).

From here a total of four branches lead to n = 2, and so on so that there are 2^n paths leading to the nth time step. Figure 3.1 shows how the number of these paths with different displacement lengths are distributed for the first five time displacements. It can be noted that the non-zero elements for each time step are the elements of Pascal's triangle.

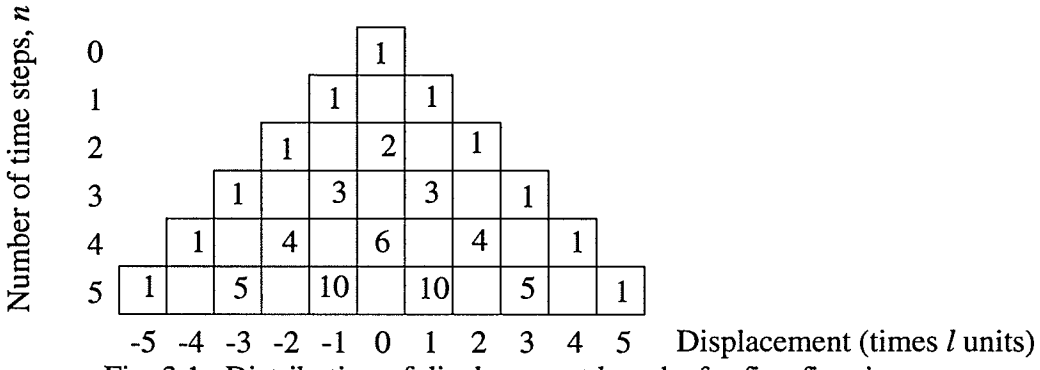


Fig. 3.1. Distribution of displacement lengths for first five time steps.

Calculating the mean square displacements for these time steps gives the values listed in Table 3.1.

Table 3.1. Mean square displacement for first five time steps.

n	mean square length
1	l^2
2	$2l^2$
3	$3l^2$
4	$4l^2$
5	$5l^2$

Examination of these values suggests that the mean square displacement in time step n is nl^2. Let's assume this is true for the nth time step. There are 2^n paths leading to the nth time step. Assign a number, i, to each path, where i is a number from 1 to 2^n.

Assume the displacement of the ith path is $k_i l$. This gives the mean square displacement of the nth time step as

$$\text{Mean square displacement of step } n = \frac{1}{2^n}\sum_{i=1}^{2^n} k_i^2 l^2 = nl^2 \tag{3.17}$$

Consider time step n+1. There are two paths leading from the ith path in time step n to step n+1: one of these paths has a displacement of $(k_i - 1)\,l$, while the other has a displacement of $(k_i + 1)\,l$. This results in

$$\text{Mean square displacement of step } n+1$$

$$= \frac{1}{2^{n+1}}\sum_{i=1}^{2^n} [\,(k_i-1)^2 l^2 + (k_i+1)^2 l^2] \tag{3.18a}$$

$$= \frac{1}{2^{n+1}}\sum_{i=1}^{2^n} [\,(k_i^2 - 2k_i + 1)\,l^2 + (k_i^2 + 2k_i + 1)\,l^2] \tag{3.18b}$$

$$= \frac{1}{2^{n+1}}\sum_{i=1}^{2^n} (2k_i^2 + 2)\,l^2 \tag{3.18c}$$

$$= \frac{1}{2^n}\sum_{i=1}^{2^n} (k_i^2 + 1)\,l^2 \tag{3.18d}$$

$$= \frac{1}{2^n}\sum_{i=1}^{2^n} k_i^2 l^2 + \frac{1}{2^n}\sum_{i=1}^{2^n} l^2 \tag{3.18e}$$

$$= \frac{1}{2^n}\sum_{i=1}^{2^n} k_i^2 l^2 + \frac{2^n l^2}{2^n} \tag{3.18f}$$

$$= \frac{1}{2^n}\sum_{i=1}^{2^n} k_i^2 l^2 + l^2 \tag{3.18g}$$

Substituting Eq. 3.17 into Eq. 3.18g gives

$$\text{Mean square displacement of step } n+1 \;=\; nl^2 + l^2 \;=\; (n+1)\,l^2 \tag{3.19a}$$

Thus, by induction, the mean square displacement of time step n is nl^2.

Assume that steps occur on average during a time interval of t. Denoting the average speed of the particle by v gives the relation

$$vt \;=\; nl \tag{3.20}$$

which leads to this formula for mean square displacement

$$nl^2 \;=\; vtl \tag{3.21}$$

Thus, the mean squared displacement is proportional to the time interval.

Table 3.2 shows the number of paths for the displacements, Δ, that occur in the 20th time step. These values are plotted in Fig. 3.2.

Table 3.2. Path distribution for 20 time steps

Δ	count	Δ	count	Δ	count
0	184756	±8	38760	±16	190
±2	167960	±10	15504	±18	20
±4	125970	±12	4845	±20	1
±6	77520	±14	1140		

The shape of this plot indicates the bell curve that is typical of a Gaussian random variable.

Thus the displacement over a time interval, t, for one dimensional Brownian motion can be modelled by a Gaussian random variable having a zero mean and a variance proportional to the length of the time interval.

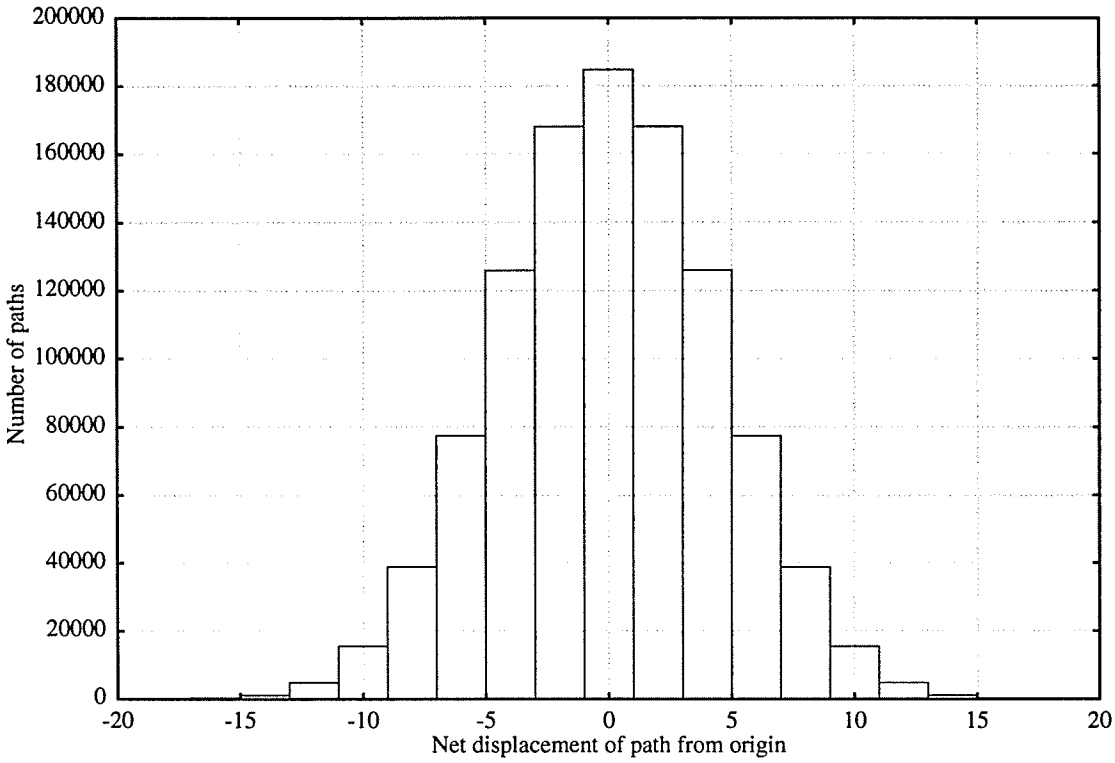


Fig. 3.2. Distribution of paths for n=20.

Scaling of Brownian Motion

One of the properties of a fractal is self similarity. Since Brownian motion has a random nature, exact affine self similarity is not possible [PeJS92]. However, by scaling the time axis and the amplitude of the graph of one dimensional Brownian motion by different factors in the horizontal and vertical directions, a new graph with similar properties is obtained.

The proper vertical scaling factor, r, can be obtained by considering the fact that the mean square displacements, Δ^2, are proportional to the time increments t.

$$\Delta^2 \propto t \tag{3.22}$$

If the time axis is multiplied by a factor of a, the vertical scaling factor has the property

$$(r\Delta)^2 \propto at \tag{3.23}$$

Since the mean square displacements and the time increments are proportional, there exists a constant, k, such that

$$\Delta^2 = kt \tag{3.24}$$

and

$$(r\Delta)^2 = kat \tag{3.25}$$

Rescaling Eq. 3.24 by a gives

$$a\Delta^2 = kat \tag{3.26a}$$

$$(\sqrt{a}\Delta)^2 = kat \tag{3.26b}$$

Thus, from Eq. 3.25 and Eq. 3.26b, the proper vertical rescaling factor is given by

$$r = \sqrt{a} \tag{3.27}$$

If a scaling factor other that this one is used, the statistical properties of the graph are changed.

Fractional Brownian Motion

Fractional Brownian motion exhibits scaling invariance at some vertical scaling factor between 1 and 2 when time is scaled by a factor of 2 [PeJS92]. This factor can be written in the form 2^H, where $0 < H < 1$. This exponent H is called the Hurst exponent, which is equal to 0.5 for ordinary Brownian motion.

Variance Fractal Dimension Calculation

The variance technique of calculating fractal dimension is derived from the properties of fractional Brownian motion. The fractal dimension, D_σ, of fractional Brownian motion is determined by the Hurst exponent, H, [Kins94b], [GaFG91], [ZhBM90], [PeJS92], [GrKi94] as given by

$$D_\sigma = 2 - H \tag{3.28}$$

The increments for fractional Brownian motion are given by

$$b(t_n, \Delta t) = B_H(t_n + \Delta t) - B_H(t_n) \tag{3.29}$$

where $B_H(t_n)$ is the sample of Brownian motion at time, t_n, and Δt is the time displacement over which the increment is being measured. These increments have the property

$$b(t_n, \Delta t) = k\xi|\Delta t|^H \tag{3.30}$$

where k is a constant and ξ is a normalized Gaussian random variable.

The increments will have a zero mean and a variance, $\mathrm{Var}\{b(t_n, \Delta t)\}$, given by

$$Var\{b(t_n, \Delta t)\} = \langle b^2(t_n, \Delta t)\rangle = k^2|\Delta t|^{2H} \tag{3.31}$$

Taking the logarithm of both sides gives

$$\log V\{b(t_n, \Delta t)\} = 2H\log\Delta t + C \tag{3.32}$$

where C is a constant. The value of H can be calculated by using least squares regression on a set of ordered pairs $(x_i, y_i) = \left(\log(\Delta t_i), \log\left(Var\{b(t_i, \Delta t_i)\}\right)\right)$ using

$$2H = \frac{N\sum_{i=1}^{N} x_i y_i - \left(\sum_{i=1}^{N} x_i\right)\left(\sum_{i=1}^{N} y_i\right)}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \tag{3.33}$$

where N is the number of ordered pairs used [Kins94c, MeSi88].

3.1.4 Spectral Fractal Dimension

The spectral fractal dimension is based on the PSD, $P(f)$, of fractional Brownian motion, which is given by

$$P(f) = kf^{-2H-1} \tag{3.34}$$

where k is a constant, H is the Hurst exponent, and the frequency, f, is

$$f = cf_0, \qquad c \in I, -\frac{N}{2} < c < \frac{N}{2} \tag{3.35}$$

Using an N point FFT on a window of N samples the spectrum, $X(f)$, can be obtained.

Since the signals being considered are real the PSD is given by

$$P(f) = |X(f)|^2 \tag{3.36}$$

Taking logarithms of Eq. 3.34 gives

$$\log\{P(f)\} = -(2H+1)\log\{f\} + C \tag{3.37}$$

which can be solved by least squares regression on pairs of positive frequencies and their corresponding PSD value to get H and thus D [ZhBM90].

The spectral dimension is limited to window sizes that are powers of 2, if FFT algorithms are used. In applications where other window sizes are required, either slower DFT algorithms must be used in calculating the spectrum, or a different dimension technique must be used.

3.2 Measuring Fractality of Signals with Variance Fractal Dimension

Many types of signals have a fractal nature. The variance fractal dimension can be used to determine how the fractal characteristic of the signal varies as the signal progresses through different features. Noticeable differences in the fractal dimension are expected at the boundaries between different signal features.

3.2.1 Obtaining Trajectories

The local fractal dimension [Mara91] is calculated for a localized portion of the original waveform to yield a fractal dimension sample. When considered in sequence, these samples form a fractal dimension waveform of the original signal. The path this waveform follows is the variance fractal dimension trajectory.

Settings of the Variance Fractal Dimension

The variance fractal dimension is calculated using ordered pairs consisting of a time increment and the variance of the signal increments. The set of time increments that is selected for use in the calculation can be altered so that the resultant trajectory represents different characteristics of the signal.

By choosing the set of time increments using a unit step selection, i.e. using $\{1\delta t, 2\delta t, 3\delta t, 4\delta t\}$ as the set of Δt's selected, with δt being the atomic time displacement, the correlation of adjacent samples has a large effect on the dimension trajectory obtained. In largely uncorrelated portions of the signal, such as background noise, a high dimension value is expected. In portions of the signal with higher correlation, such a speech, a much lower dimension value is obtained. Thus it is possible to separate a correlated signal from surrounding background noise. This is called the noise separation process.

In a situation where more detail is desired in the trajectory, it is necessary to use a dyadic selection of increments, i.e using $\{2\delta t, 4\delta t, 8\delta t, 16\delta t\}$ as the set of Δt's selected, to break up the correlation in the signal. The trajectory produced using the dyadic selection has a higher dimension than noise separation and shows greater variation in its level. This process is called fractal amplification.

3.3 Summary

In this chapter, the concept of fractal dimension has been discussed. There are many different techniques for calculating fractal dimension. The technique that seems the most versatile in term of simplicity of calculation and flexibility of parameters is the variance fractal dimension. This technique is used in the production of variance fractal dimension trajectories. Chapter IV describes a system that produces and displays these trajectories.

CHAPTER IV
PROGRAM DESIGN AND IMPLEMENTATION

This chapter describes a system that is designed to operate under DOS 6.2 and which generates and displays variance fractal dimension trajectories. The system receives a file stored in Creative Voice (VOC) file format as input. The data is extracted from the file and is passed to an implementation of the variance fractal dimension algorithm, which produces the data points for the variance fractal dimension trajectory. These trajectories are displayed on the screen. The user is able to control various settings at different points during this process using a keyboard driven command interface and can save the display as a tagged image file format (TIFF) file.

4.1 Creative Voice File Format

The VOC file format consists of a 26 byte header followed by a series of blocks, concluding with a terminator block [Ross94]. The two main versions of VOC files encountered are the older version 1.10, which contains only 8-bit data and the newer version 1.20, which has included provisions for 16-bit data.

VOC Header

The VOC file header contains information identifying the file type and version and a pointer to the first data block in the file. The organization of the data in the header is shown in Table 4.1. All multiple byte values in the VOC file are stored least significant byte (LSB) first.

Table 4.1. VOC Header Contents.

Byte	Contents	Description
00-12	43 72 65 61 74 69 76 65 20 56 6f 69 63 65 20 46 69 6c 65	"Creative Voice File"
13	1A	EOF to prevent printing
14-15	1a 00	Offset of first datablock
16-17	0A 01 (for Version 1.10) or 14 01 (for Version 1.20)	Version Number (minor, major)
18-19	29 11 (for Version 1.10) or 1f 11 (for Version 1.20)	2's Comp of Ver. # + 1234h

Data Blocks

The organization for most block types, shown in Table 4.2, in the VOC format consists of one byte for the block type, a three byte integer containing the number of bytes in the block information, and the block information. The terminator block is an exception, containing only its block type to signify the end of the VOC file. The formats of the block information for the more complex block types are given in Table 4.3 through Table 4.6.

Table 4.2. VOC Data Blocks.

Block Type	Description	Information Size	Information
00	Terminator	N/A	N/A
01	8-bit Sound Data	2 + length of data	See Table 4.3
02	8-bit Sound Continue	length of data	Voice Data
03	Silence	3	See Table 4.4
04	Marker	2	2 byte marker number
05	ASCII	length of string	null terminated string
06	Repeat	2	2 byte count number
07	End Repeat	0	N/A
08	Extended	4	See Table 4.5
09	16-bit Sound Data	12 + length of data	See Table 4.6

The marker number is kept in a status byte by the drivers for some VOC file players, recorders, and editors. The count number is one greater than the number of repetitions. If the counter number is #FFFF the number of repetitions is infinite.

Table 4.3. 8-bit Sound Information.

Byte	Contents
00	Sample Rate
01	Compression Type: 0 = 8 bits 1 = 4 bits 2 = 2.6 bits 3 = 2 bits 3+(number of channels) = Multi DAC
02+	Voice Data

The byte representing the sample rate is defined as

$$SRbyte = 256 - \left(\frac{1000000}{SR} \right) \tag{4.1}$$

where SR is the actual sampling rate.

Table 4.4. Silence Information.

Byte	Contents
00 - 01	Length on silence - 1
02	Sample Rate

The length of silence is the number of samples during the period of silence. The sampling rate is defined in Eq. 4.1.

Table 4.5. Extended Information.

Byte	Contents
00-01	Time Constant
02	Pack
03	Mode: 0 = Mono, 1 = Stereo

The time constant is defined by

$$\begin{aligned} TC &= 65536 - \left(\frac{256000000}{SR} \right) \text{ for mono} \\ TC &= 65536 - \left(\frac{256000000}{2SR} \right) \text{ for stereo} \end{aligned} \tag{4.2}$$

Table 4.6. 16-bit Sound Information.

Byte	Contents
00-01	Sample Rate
02-0B	Voice Data Format Codes
0C +	Voice Data

Unlike the 8-bit sound data and silence blocks, the sample rate is the actual sampling rate. The voice data code contains information on mono or stereo modes and compression type and some reserved bytes. For the mono data with no compression these bytes are: 00 00 10 01 04 00 00 00 00.

4.2 Variance Fractal Dimension Implementation

The variance dimension algorithm can be used to calculate the fractal dimension of a sequence of signal samples. The sequence contains N samples at times, t_n, given by

$$t_n = n\delta t, n = 0, 1, 2, \ldots, N-1 \tag{4.3}$$

where n is the time index, and δt is the time displacement between individual samples or the atomic displacement.

The time displacement used for the measurement of the increments, Δt, or the cumulative time displacement can be chosen as a multiple of the atomic time displacement with the multiples being chosen in either a unit-distance or a dyadic selection as shown respectively by

$$\Delta t = m\delta t, m = 1, 2, 3, \ldots, m_{max} \tag{4.4a}$$

$$\Delta t = m\delta t, m = 2^0, 2^1, 2^2, \ldots, m_{max} \tag{4.4b}$$

where m is the cumulative time displacement index and m_{max} is the largest displacement index desired. The unit displacement index (m=1) may be omitted to reduce the effect of correlation between adjacent signal samples.

The increment, $b(n\delta t, m\delta t)$, between speech samples at a specific starting time, $n\delta t$, and cumulative displacement, $m\delta t$, is determined by

$$b(n\delta t, m\delta t) = s(n\delta t + m\delta t) - s(n\delta t) \tag{4.5}$$

where $s(t)$ is the speech signal.

The variance of the increments, $V\{b(n\delta t, m\delta t)\}$, can be calculated using either a population of increments selected so that the time increments overlap, where the starting points of the increment intervals are selected at every sample, or so that the time increments do not nonoverlap,where the starting points of the increments are selected at the size of the cumulative displacement as shown respectively by

$$V\{b(n\delta t, m\delta t)\} = \frac{\sum_{n=0}^{N-m-1} b(n\delta t, m\delta t)^2}{N-m} \tag{4.6a}$$

$$V\{b(n\delta t, m\delta t)\} = \frac{\sum_{k=0}^{\left\lfloor \frac{N}{m} \right\rfloor - 1} b(km\delta t, m\delta t)^2}{\left\lfloor \frac{N}{m} \right\rfloor} \tag{4.6b}$$

The equation used to calculate the least square slope, $2H$, can be rewritten

$$\log V\{b(n\delta t, m\delta t)\} = 2H\log(m\delta t) + C \tag{4.7a}$$

$$\log V\{b(n\delta t, m\delta t)\} = 2H\log(m) + 2H\log(\delta t) + C \tag{4.7b}$$

$$\log V\{b(n\delta t, m\delta t)\} = 2H\log(m) + C^* \tag{4.7c}$$

where C^* is a constant. Thus it is sufficient to calculate the logarithms of the cumulative time displacement indices rather than the logarithms of the time displacements themselves.

The following algorithm, shown in the flowchart in Fig. 4.1, is used to calculate the variance dimension of a sequence of samples:

1. The cumulative time displacement index of m is set at 1 (or 2 if unit increments are being omitted.)
2.1 A summation variable is initialized to 0.
2.2 The time index, n, is set at 0.
2.3 While the value of n is less than $N - m$, steps 2.3.1 to 2.3.3 are performed. Otherwise processing continues at step 2.4.
2.3.1 The increment of the signal values from time index n to time index $n + m$ is calculated.
2.3.2 The square of the increment is added to the summation variable.
2.3.3 The value of n is incremented by 1 if the population of increments is overlapping. Otherwise, n is incremented by m.
2.4 The sum is divided by the number of increments, $N - m$ for overlapping increments or $\left\lfloor \frac{N}{m} \right\rfloor$ for nonoverlapping increments, in the window to give the variance of the increments.
2.5 The logarithms of the variance and the time displacement index are calculated and assigned to arrays corresponding to the two values.
2.6 The value of m is incremented by 1 for unit-distance multiples or multiple by 2 for dyadic multiples. If m is greater than the value of m_{max} processing continue with step 3. Otherwise, the entire process from step 2.1 is repeated with the new value of m.
3. Least squares regression is used on the values stored in the arrays containing the logarithms of the variances and the cumulative time increments to determine the slope of the best fit line.
4. The value of the slope is divided by 2 to obtain the Hurst exponent, H.
5. The variance dimension, D, is obtained by subtracting H from 2.

4.3 System for Production of Fractal Dimension Trajectories

A dimension trajectory of a speech waveform can be obtained using the variance dimension algorithm by calculating the dimension within a sequence of localized regions in the speech signal. Thus, it is necessary to divide the signal into a sequence of windows that start at regular intervals. Each window can be passed as a sequence of samples to the variance dimension algorithm to calculate a dimension value for that window. The sequence of dimension values corresponding to the sequence of windows provides the dimension trajectory.

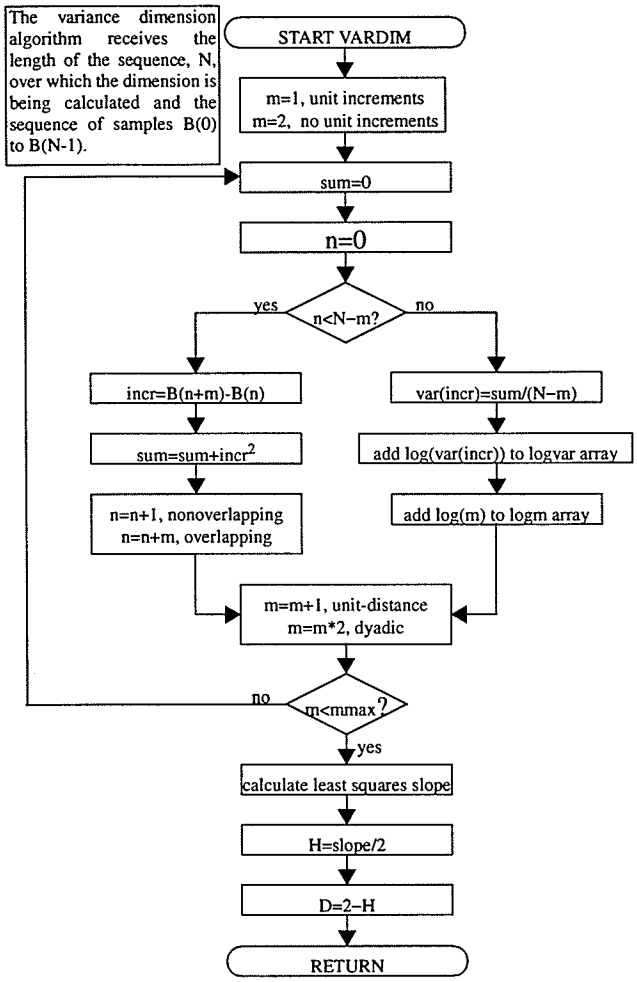


Fig. 4.1. Flowchart for variance dimension algorithm. (After [GrKi94].)

The spacing interval of the starting points for the window and the size of the window are important parameters to consider when calculating the dimension trajectory. The starting points of the windows can be spaced at intervals ranging from a single sample to the width of the window. When the spacing interval is small, many redundant calculations are made. When the spacing approaches the size of the window, critical transitions in the dimension may be missed. If the window size is chosen too small, the dimension is very erratic because the window is smaller than the self affine features within the speech waveforms. If the window size is chosen to be too large, distinct features within different segments of the speech waveform are blurred together because the windows are larger than the segments. Figure 4.2 illustrates the algorithm used to pass the windows of speech to the dimension calculation algorithm.

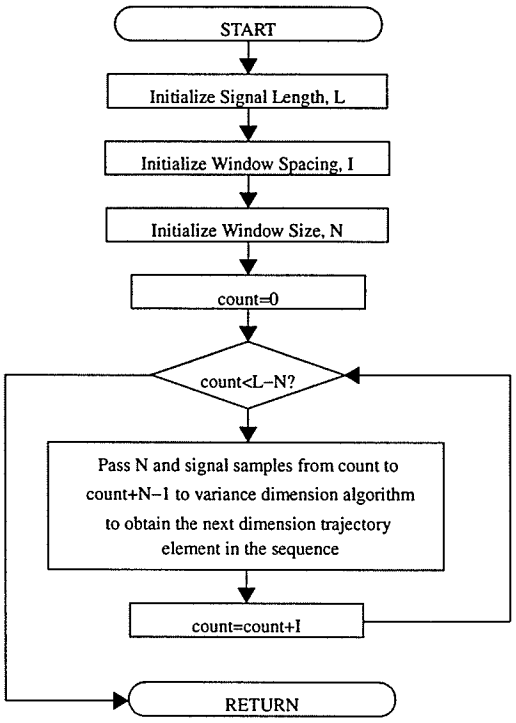


Fig. 4.2. Flowchart for obtaining trajectory. (After [GrKi94].)

4.4 Aligning Trajectory to Speech Waveform

It is possible to align the choice of the time index for a calculated sample in the trajectory waveform to any position in the window for which it is calculated. In order to choose the most appropriate position it is necessary to examine the effect on the trajectory of a transition between two periods of time with different dimension values that are constant within the individual time periods. The calculated dimension values for windows falling entirely within a given time period have the value associated for that window. The windows overlapping the two time periods have values which are intermediate to the value associated with the two time periods. The dimension values in these regions of overlap have an increasing slope, if the second time period has a higher dimension than the first time period, or a decreasing slope, if the opposite is true. If the time index of the calculated dimension value is assigned to the time corresponding to the beginning of the window for which it is calculated, the change in the level of the dimension trajectory begins one window length before the transition occurs and reaches the value for the second time period when the transition actually occurs. If the time index is assigned the time corresponding to the end of the window for which it is calculated, the change in the level of the

dimension trajectory begins at the start of the second time period and reaches the value for the second time period one window length into the second time period. Figure 4.3 illustrates these two cases.

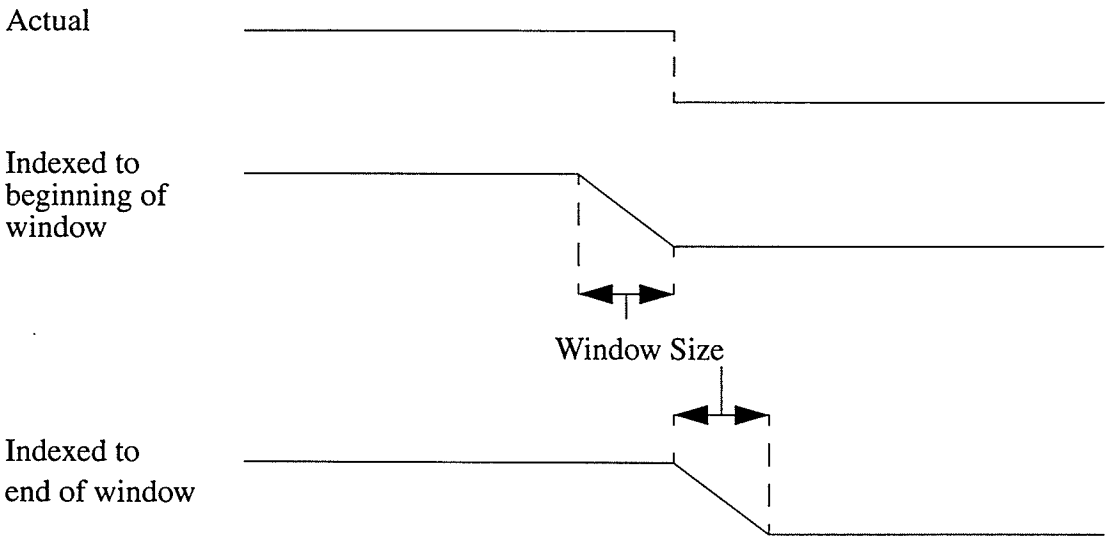


Fig. 4.3. Effect of time positioning on the dimension trajectory.

4.5 Graphical Display

The system's graphic display is divided into three areas: the input waveform display area, the fractal dimension trajectory display area, and the command interface area. When the display is saved to a TIFF file, it is redrawn, omitting the command interface area.

Input Waveform Display Area

The input waveform display area shows information pertaining to the input data.

The following information is located above the main waveform window, the input VOC file name, a time stamp, and the content of a description file associated with the VOC file. If the display is being saved to a TIFF file, the name of this file is displayed above the main waveform window as well.

The main waveform window contains a graph of the input data. The horizontal axis represents time in seconds. The vertical axis represents the amplitude measured as a

percentage of the clipping voltage. The ranges of the axes can be specified by the user to examine the details of the waveform.

Below the main waveform window, the start and stop times of the portion of the signal contained within the main display window are displayed. The sampling rates of the signal in hertz is displayed here as well.

Fractal Dimension Trajectory Display Area

The fractal dimension trajectory display area contains information pertaining to the fractal dimension trajectory calculated for the input.

Above the main trajectory window the type of dimension calculation used and the user supplied parameter settings for the calculation are displayed. The six parameters displayed for the variance fractal dimension are as follows. The selection type of the time increment that is chosen is displayed as "sequential" for a unit displacement selection, or as "dyadic" for a dyadic selection. "Overlapping" is displayed if the time increments are allowed to overlap, which provide a larger sample population than if the samples are not allowed to overlap; otherwise, "non-overlapping" is displayed. The smallest increment size in the sequence that starts from the atomic displacement is displayed as "smallest incr. size used" preceded by "2nd" if the atomic displacement is omitted, and "3rd" if the next increment size in the selection is omitted as well. (A previous version of the system, in which the inclusion of the atomic time interval is displayed as "unit increments", while the exclusion of the atomic time interval is displayed as "no unit increments", was used to generate many of trajectories used in this thesis, and images with these description are still included since the change did not affect the data that was generated.) The number time increments selected, N, is displayed as "N points". The number of samples used in calculating the individual samples in the trajectory, *wsize*, is displayed as "Window Size = *wsize*". The number of input signal samples between trajectory waveform samples, *wspacing*, is displayed as "Window Spacing = *wspacing*".

The main trajectory window contains a graph of the fractal dimension trajectory. The horizontal axis represents time in seconds. The vertical axis represents the fractal dimension. The ranges of the axes can be specified by the user.

Below the main trajectory window, some statistics for the portion of the fractal dimension trajectory that is displayed are given. These statistics include the minimum fractal dimension, the maximum fractal dimension, the average fractal dimension, and the standard deviation.

Command Interface Area

The command interface area display provides the user with menus containing which commands can be entered. These commands handle the input/output of the system, the entry of parameters for the dimension calculation, adjustments to the display parameters, and the activation of the analysis routine. Any on screen text entry also takes place in this area. This area is not displayed when the display is being saved to a TIFF file.

4.6 Generation of Test Data

Testing of the variance fractal dimension calculation algorithm is performed using sets of test signals with known fractal dimensions. Two such signal types are fractional Brownian motion and Weierstrass functions.

4.6.1 Fractional Brownian Motion

One dimensional fractional Brownian motion, $B_H(t)$, is defined by

$$Pr\left\{\frac{B_H(t_0+t)-B_H(t_0)}{|t|^H}<x\right\} = G(x) \tag{4.8}$$

where $G(x)$ is the cumulative distribution function of a Gaussian random variable with zero mean, and H is the Hurst exponent.

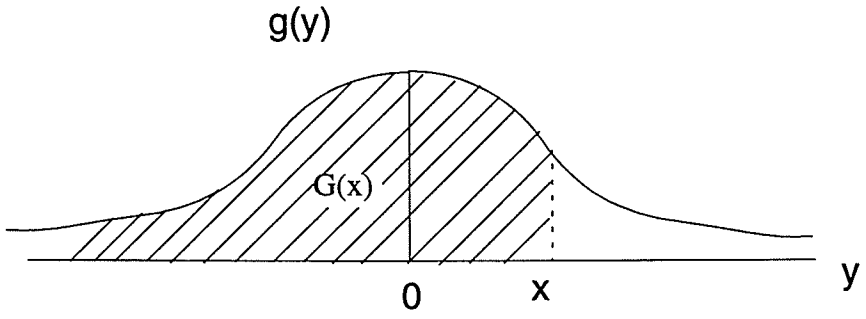


Fig. 4.4. Gaussian Distribution Function.

The fractal dimension, D, is related to H by

$$D = 2 - H \tag{4.9}$$

An equivalent representation of fractional brownian motion is given by

$$B_H(t_0 + t) - B_H(t_0) = k\xi|t|^H \tag{4.10}$$

where k is a constant and ξ is a normalized Gaussian random process.

The variance of increment V(t) is proportional to $|t|^{2H}$

$$V(t) = Var(B_H(t_0 + t) - B_H(t_0)) = k^2|t|^{2H} \tag{4.11}$$

Thus by generating $B_H(t)$ for $0 < H < 1$ we can obtain a waveform with fractal dimension D, such that $1 < D < 2$.

The simplest technique for generating fractional Brownian motion is random midpoint displacement.

Random Midpoint Displacement

Let $X(t) = B_H(t)$ be the fractional Brownian motion waveform to be generated.

Let $X(0)$ and $X(1)$ be the endpoints of the waveform.

Let $X(0) = 0$.

Let $X(1)$ be a sample of a Gaussian random variable with a mean of 0 and a variance of σ^2.

The value of the midpoint between $X(0)$ and $X(1)$, $X(0.5)$ is given by

$$X(0.5) = \frac{1}{2}(X(0) + X(1)) + D_1 \tag{4.12}$$

where the offset, D_1, is a Gaussian random variable with a mean of 0 and a variance of Δ_1^2.

$$Var\left(X(0.5) - X(0)\right) = Var\left(\frac{1}{2}\left(X(0) + X(1)\right) + D_1\right) \tag{4.13}$$

The variance between two samples of fractional Brownian motion is given by

$$Var\left(X(t_2) - X(t_1)\right) = \left|t_2 - t_1\right|^{2H}\sigma^2 \tag{4.14}$$

thus,

$$\frac{1}{2^{2H}}\sigma^2 = \frac{1}{4}\sigma^2 + \Delta_1^2 \tag{4.15a}$$

$$\Delta_1^2 = \left(\frac{1}{2^{2H}} - \frac{1}{4}\right)\sigma^2 \tag{4.15b}$$

$$\Delta_1^2 = \left(\frac{1 - 2^{2H-2}}{2^{2H}}\right)\sigma^2 \tag{4.15c}$$

Thus, D_1 can be obtained by scaling a Gaussian random variable with a mean of 0 and a variance of σ^2 by $\sqrt{\left(\frac{1 - 2^{2H-2}}{2^{2H}}\right)}$

Once $X(0.5)$ has been determined, $X(0.25)$ and $X(0.75)$ can be calculated.

$$X(0.25) = \frac{1}{2}\left(X(0) + X(0.5)\right) + D_{2,1} \tag{4.16a}$$

$$X(0.75) = \frac{1}{2}\left(X(0.5) + X(1)\right) + D_{2,2} \tag{4.16b}$$

The offsets $D_{2,1}$ and $D_{2,2}$ are Gaussian random samples with means of 0 and variances of Δ_2^2.

$$Var\left(X(0.25) - X(0)\right) = Var\left(\frac{1}{2}\left(X(0) + X(0.5)\right) + D_{2,1}\right) \tag{4.17}$$

$$\frac{1}{4^{2H}}\sigma^2 = \frac{1}{4}\left(\frac{1}{2^{2H}}\sigma^2\right) + \Delta_2^2 \tag{4.18a}$$

$$\Delta_2^2 = \frac{1}{2^{2H}}\left(\frac{1}{2^{2H}} - \frac{1}{4}\right)\sigma^2 \tag{4.18b}$$

$$\Delta_2^2 = \frac{1}{2^{2H}}(\Delta_1^2) \tag{4.18c}$$

Thus by reducing the previous scaling factor by a factor of $\frac{1}{2^H}$, $D_{2,1}$ and $D_{2,2}$ can be obtained from Gaussian random samples with zero mean and variance σ^2.

Further midpoints can be calculated in a similar manner, with the scaling factor being reduced by a factor of $\frac{1}{2^H}$ with each iteration.

4.6.2 Weierstrass Function

The Weierstrass function is given by

$$^{r}(x) = \sum_{i=1}^{\infty} \frac{\sin(\lambda^i x)}{\lambda^{\varepsilon i}} \tag{4.19}$$

where $\lambda < 1$ and $0 < \varepsilon < 1$ are constants [Harr89].

Although it has not be proven rigorously, it seems likely that

$$D = 2 - \varepsilon \tag{4.20}$$

is the fractal dimension of the Weierstrass function [Harr89].

4.7 Generation of Random Numbers

The proper generation of pseudo-random data is essential to maintain the statistical properties expected from random data. The algorithms used are described below.

4.7.1 Uniform Distribution

A pseudo-random sequence $\{I_j\}$ can be generated using the multiplicative congruential algorithm:

$$I_{j+1} = aI_j \bmod m \tag{4.21}$$

provided the multiplier, a, and the modulus, m, are chosen carefully.

Equation 4.21 cannot be implemented directly because of the limitation of 32-bit integers in a high level language. Schrage's algorithm provide a technique for multiplying two 32 bit number modulo a 32 bit constant without requiring an intermediate value of greater than 32 bits. Schrage's algorithm is given by

$$aI_j \bmod m = \begin{cases} a(I_j \bmod q) - r\lfloor I_j/q \rfloor, & a(I_j \bmod q) > r\lfloor I_j/q \rfloor \\ a(I_j \bmod q) - r\lfloor I_j/q \rfloor + m, & \text{otherwise} \end{cases} \tag{4.22a}$$

where the quotient, q, and the remainder, r, are given by

$$q = \lfloor m/a \rfloor \tag{4.22b}$$

$$r = m \bmod a \tag{4.22c}$$

(4.22d)

The multiplicative congruential algorithm is subject to low order serial correlation. This correlation can be removed by implementing the Bay and Durham shuffling algorithm.

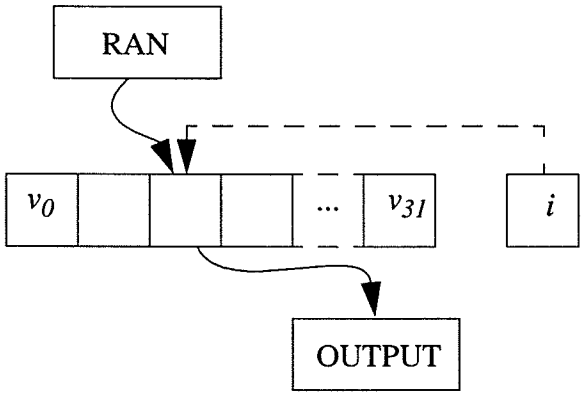


Fig. 4.5. Bay and Durham shuffling algorithm. (After [PTVF92]).

This algorithm require a table of values, v_i, and an index, i. The table is initialized with values obtained from the multiplicative congruential algorithm, and the index is given the value of the last element entered. When a random number is required a new entry will be generated. The index is scaled to the number of table entries and the corresponding entry is passed to the output and is replaced by the new entry. The output value becomes the index for the next random number. The sequence of number produced by the multiplicative congruential algorithm has a period of m-1. A longer period can be obtained by

combining one sequence with another sequence with a different period. The resulting period is the least common multiple of the length of the two original periods. The output random number is obtained by subtracting the value obtained from the second sequence from the value obtained from the shuffle table of the first sequence and adding the modulus of the first sequence if the result is negative.

A random value between 0 and 1 is obtained by dividing the integer result from the algorithm by the largest possible integer that the algorithm produces. This random value has uniform distribution on (0,1) and a probability distribution function of

$$p(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.23}$$

where x is the uniform random deviate.

4.7.2 Transformations

It is possible to use transformations to obtain random numbers corresponding to other types of probability distributions from a random number generated on a uniform distribution. The probability of generating a random number on the uniform distribution between x and $x + dx$, denoted by $p(x)dx$, is given by

$$p(x)\,dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

If $y(x)$ is a function of x the probability distribution of y, denoted by $p(y)dy$, as determined by the fundamental transformation law of probabilities is given by

$$p(y) = p(x)\left|\frac{dx}{dy}\right| \tag{4.25}$$

If the cumulative distribution function of the random variable y is $F(y)$ then

$$\frac{d}{dy}F(y) = p(y)$$

which has the solution

$$x = F(y) \tag{4.26}$$

Thus, the transform y(x) is

$$y(x) = F^{-1}(x) \tag{4.27}$$

If it is possible to calculate the inverse of the integral of the probability distribution function, the transform from the uniform distribution can be calculated.

For joint probability distributions, Eq. 4.25 generalizes to

$$p(y_1, y_2, \ldots)\, dy_1 dy_2 \ldots = p(x_1, x_2, \ldots) \left| \frac{\partial(x_1, x_2, \ldots)}{\partial(y_1, y_2, \ldots)} \right| dy_1 dy_2 \ldots \tag{4.28}$$

where $\left| \frac{\partial(x_1, x_2, \ldots)}{\partial(y_1, y_2, \ldots)} \right|$ is a the Jacobian determinant which becomes

$$\left| \frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} \right| = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} \tag{4.29}$$

for two variables.

4.7.3 Gaussian Distribution

The probability distribution function for a Gaussian distribution is given by

$$p(y) = \frac{1}{2\pi} e^{-\frac{y^2}{2}} \tag{4.30}$$

Since it is not possible to calculate the inverse of the integral for this distribution function, it is necessary to utilize the properties of the generalized formula for joint probabilities. Consider the transforms

$$y_1 = \sqrt{-2\ln x_1}\cos 2\pi x_2 \tag{4.31a}$$

$$y_2 = \sqrt{-2\ln x_1}\sin 2\pi x_2 \tag{4.31b}$$

where x_1 and x_2 are uniform deviates on (0,1).

These can be rewritten as

$$x_1 = \exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] \tag{4.32a}$$

$$x_2 = \frac{1}{2\pi}\operatorname{atan}\frac{y_2}{y_1} \tag{4.32b}$$

Calculating the Jacobian determinant

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \begin{vmatrix} \frac{\partial}{\partial y_1}\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] & \frac{\partial}{\partial y_2}\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] \\ \frac{\partial}{\partial y_1}\frac{1}{2\pi}\operatorname{atan}\frac{y_2}{y_1} & \frac{\partial}{\partial y_2}\frac{1}{2\pi}\operatorname{atan}\frac{y_2}{y_1} \end{vmatrix} \tag{4.33a}$$

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \begin{vmatrix} -y_1\left(\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]\right) & -y_2\left(\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]\right) \\ \frac{-y_2}{2\pi\left(1 + \left(\frac{y_2}{y_1}\right)^2\right)y_1^2} & \frac{1}{2\pi\left(1 + \left(\frac{y_2}{y_1}\right)^2\right)y_1} \end{vmatrix} \tag{4.33b}$$

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \begin{vmatrix} -y_1\left(\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]\right) & -y_2\left(\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]\right) \\ \frac{-y_2}{2\pi(y_1^2 + y_2^2)} & \frac{y_1}{2\pi(y_1^2 + y_2^2)} \end{vmatrix} \tag{4.33c}$$

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \frac{\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]}{2\pi(y_1^2 + y_2^2)}(-y_1^2 - y_2^2) \tag{4.33d}$$

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = \frac{-\exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right]}{2\pi} \tag{4.33e}$$

$$\left|\frac{\partial(x_1, x_2)}{\partial(y_1, y_2)}\right| = -\left(\frac{\exp\left[-\frac{y_1^2}{2}\right]}{2\pi}\right)\left(\frac{\exp\left[-\frac{y_2^2}{2}\right]}{2\pi}\right) \tag{4.33f}$$

which is the product of a function of y_1 alone and y_2 alone.

Thus, y_1 and y_2 are independent of each other, and both are distributed according to a normal distribution. Thus the transforms given in Eq. 4.31a and Eq. 4.31b can be used to transform two random deviates on a uniform distribution to two random deviates on a Gaussian distribution.

4.8 Acquiring Experimental Data

The manner in which data is gathered can help in the understanding of the results of experiments performed on this data. The data that was analyzed by the system described in this chapter for the experiments in Chapter V was obtained in the manner described below.

4.8.1 Acquiring Digitized Speech

The speech samples acquired for experimentation were recorded using a Realistic Highball II omnidirectional microphone. The input from the microphone was digitized as 16 bit samples using a Creative Labs Sound Blaster 16 (SB16) sound card.

4.8.1.1 Sampling rate choice

The standard sampling rates that can be used with the SB16 are 11025 Hz, 22050 Hz, and 44100 Hz. The highest sampling rate was chosen for speech signals to provide the greatest amount of information about the signal.

4.8.1.2 Selection of Test Words

The recorded speech consists of a set of test words that cover the range of English phonemes. This set of words was previously shown in Table 2.1.

4.8.2 Acquiring Packet Radio Transmissions

The packet radio transmissions were received using a Kenwood TW-4000 A radio. These transmissions were recorded with a Panasonic RX-FW20 tape recorder on BASF Chrome Maxima II tapes and later digitized using the SB16. Since the recording media

has a reduced frequency range, the lower sampling rate of 11025 Hz was used to avoid the inclusion of interpolated data. The inclusion of interpolated data artificially increases the correlation between adjacent and nearly adjacent samples thus decreasing the fractal dimension of the background noise and packet transmission alike.

4.9 Summary

The design of the system allows a recorded signal to be input for analysis. The analysis is conducted with user supplied parameters. The results of the analysis are displayed on screen and can be saved to a TIFF file.

The program is designed to allow the addition of other fractal dimension calculation techniques. This involves the addition of a menu item in the fractal dimension type selection menu routine, a menu routine for the selection of its parameters, and the routines that handle its calculation and display its results on the screen.

CHAPTER V
EXPERIMENTAL RESULTS AND ANALYSIS

Several experiments have been performed using the system described in Chapter IV. One set of experiments tests the accuracy of the variance fractal dimension algorithm by applying it to a set of waveform that were artificially generated to have a specific dimension. Further experiments have been performed on a set of test words to determine the behaviour of the fractal dimension during the speech process. The application of the process is also examined for packet radio transmissions.

5.1 Test Data

The test data used to verify the variance fractal dimension algorithm was generated by methods that produce waveforms with known fractal dimensions. Two sets of these generated waveforms are used: fractional Brownian motion and Weierstrass functions.

Fractional Brownian Motion

Nine waveforms of fractional Brownian motion were generated to have fractal dimensions from 1.1 to 1.9 spaced at equal intervals. The variance fractal dimension of each of these waveforms was calculated using both the noise separation and the fractal amplification settings. The results are given in Table 5.1 and plotted in Fig. 5.1.

Table 5.1. Dimension values for Fractional Brownian Motion.

Expected	Noise Separation	Fractal Amplification
1.1	1.15	1.13
1.2	1.23	1.22
1.3	1.33	1.32
1.4	1.42	1.41
1.5	1.50	1.51
1.6	1.57	1.59
1.7	1.63	1.67
1.8	1.69	1.73
1.9	1.73	1.78

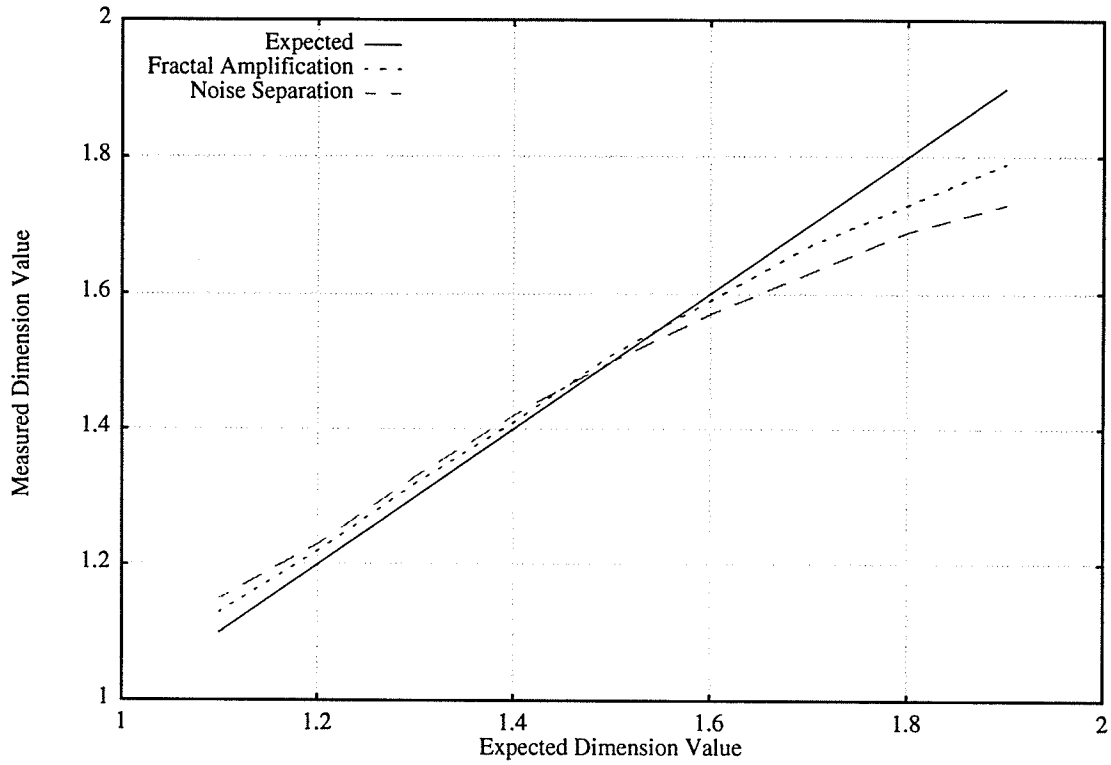


Fig. 5.1. Graph of dimension values for fractional Brownian motion.

These results indicate a tendency for the calculated dimension to be in the direction of 1.5 from the expected dimension. As the dimension value moves away from 1.5 the divergence of the calculated dimension increases with a greater error in the direction of a dimension value of 1.9 than in the direction of 1.1. The fractal amplification setting gives more accurate results the noise separation setting. However, both sets of calculated dimension are increasing functions, thus the dimension value for a signal portion that is calculated as being higher than the dimension value of another signal portion, would have the higher actual dimension value.

Weierstrass Function

Like the fractional Brownian motion test signals, a set of nine Weierstrass function test signals were generated to have fractal dimensions from 1.1 to 1.9 spaced at equal intervals. The variance fractal dimension of each of these waveforms was calculated using both the noise separation and the fractal amplification settings. The results are given in Table 5.2 and plotted in Fig. 5.2.

Table 5.2. Dimension values for Weierstrass function.

Expected	Noise Separation	Fractal Amplification
1.1	1.11	1.15
1.2	1.18	1.22
1.3	1.25	1.32
1.4	1.32	1.42
1.5	1.38	1.52
1.6	1.45	1.62
1.7	1.51	1.72
1.8	1.57	1.82
1.9	1.64	1.92

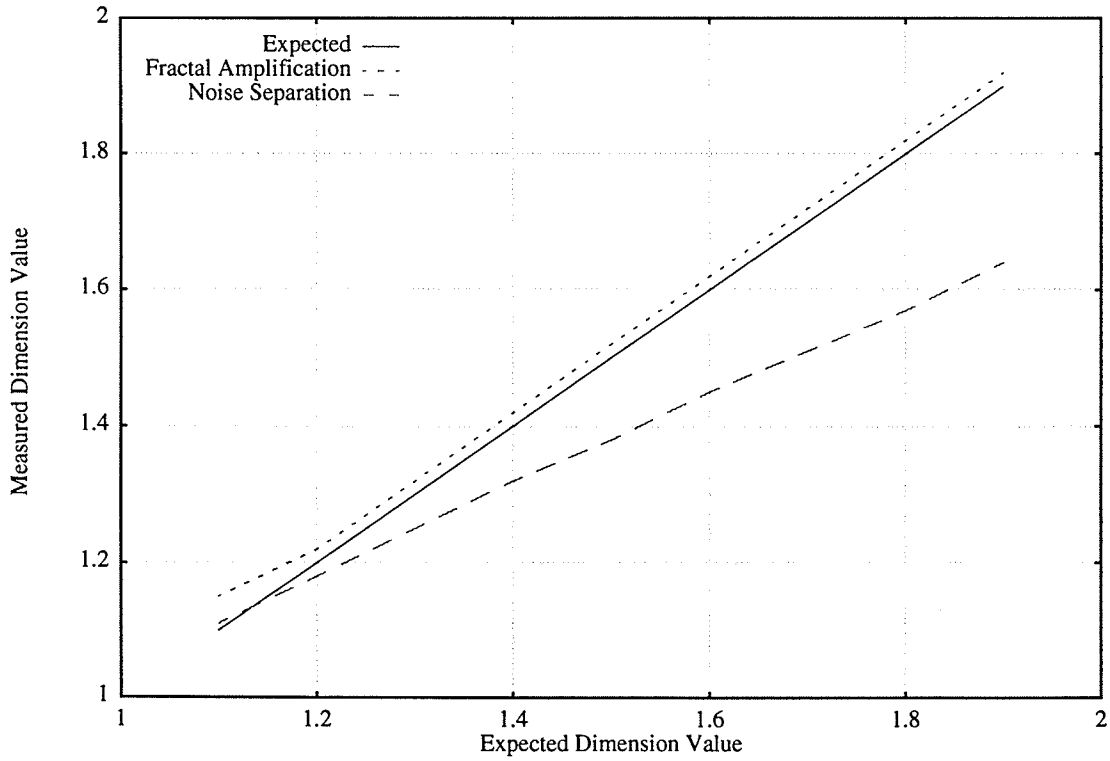


Fig. 5.2. Graph of dimension values for Weierstrass function.

Here the fractal dimension calculated using fractal amplification is very close to the expected dimension. However, the dimension calculated using the noise separation

setting is significantly lower. Both of these cases exhibit a monotonically increasing calculated dimension.

5.2 Speech Waveforms

5.2.1 Distinguishing Speech from Noise

The first set of experiments that involves the application of the variance fractal dimension to speech waveform examines the computed dimension trajectories of the recorded speech waveforms to distinguish the utterance from the preceding and trailing background noise.

Consider, as an example, the noise separation trajectory of the word tight, /taɪt/, shown in Fig. 5.3. Here, the utterance is clearly preceded and followed by trajectory levels with much higher values than the remainder of the signal. The short duration interval within the utterance, which has a dimension level similar the preceding and following noise, corresponds to the stop gap of the second occurrence of the phoneme /t/. This stop gap is a short period of silence which is recorded with the same noise characteristics as the preceding and following noise.

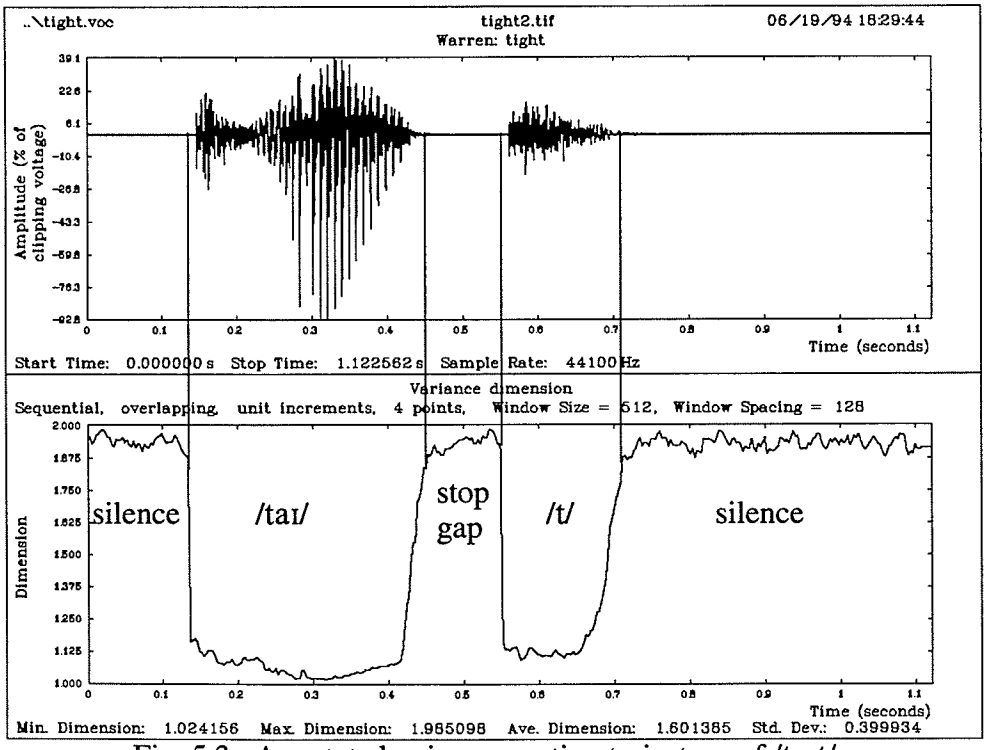


Fig. 5.3. Annotated noise separation trajectory of /taɪt/.

This type of behaviour can be observed with most of the trajectories calculated for the test words. In these cases a simple threshold check on the fractal dimension value is sufficient to separate the utterance from the surrounding noise; however, there at least two types of noises originating from the speaker have lower dimensions values that the ambient noise: breath clicks and audible breath releases.

The presence of a breath click can be seen in the trajectory of the word boy, /bɔɪ/, shown in Fig. 5.4. The breath click is seen in the trajectory as decrease in the dimension value having a very short duration. A check of the duration of lower level dimensions spans in the trajectory can separate breath clicks from normal utterances.

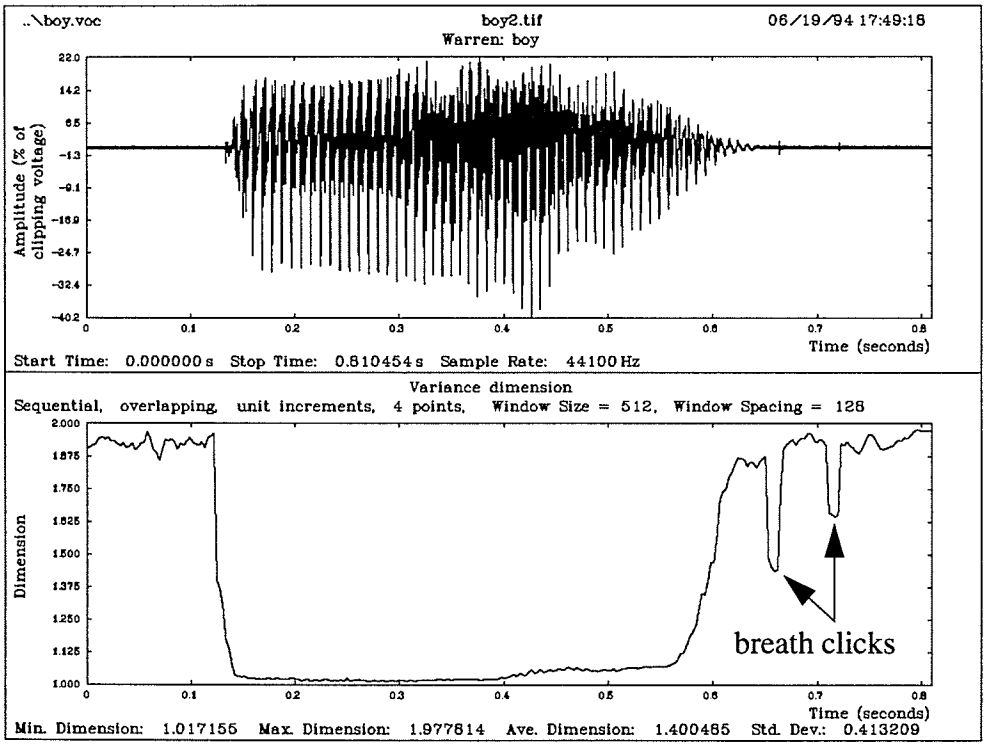


Fig. 5.4. Noise separation trajectory of /bɔɪ/ with breath clicks.

Audible breath releases, or aspirations, do not play a significant role in the structure of the English language; however, this is not the case in some other languages, such as Tai [Carr93]. In such languages, the aspiration is considered a feature of speech; thus, it is important for it to be distinguished from the ambient noise. Figure 5.5 shows the noise separation trajectory of an aspiration. This trajectory is characterized by a mid-range dimension value that rises to the dimension level of the background noise as the aspiration fades.

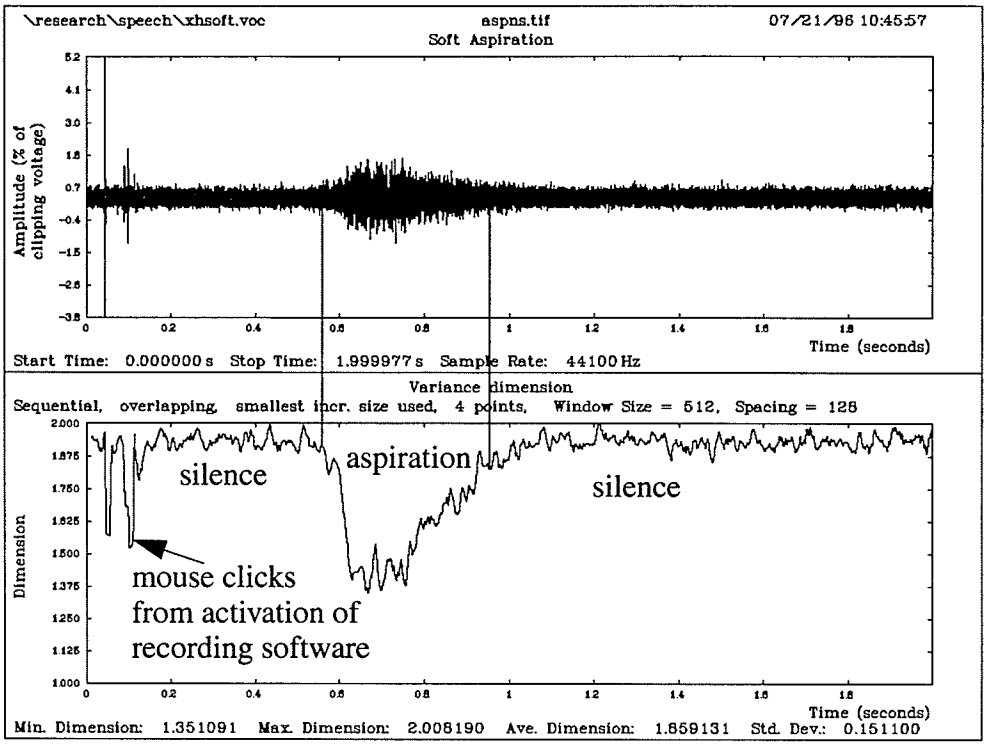


Fig. 5.5. Annotated noise separation trajectory of an aspiration.

The presence of aspirations accounts for the trailing trajectories that have lower dimensions than the leading trajectories in some of the test words. This is illustrated by the trajectory of the word pip, /pɪp/, shown in Figure 5.6, which shows a complete aspiration following the second /p/. In other recorded test words, the recording terminates before the aspiration is complete, thus the trajectory does not rise to dimension level of the ambient noise at the end of the recording. One such test word is zoos, /zuz/, which is shown in Fig. 5.7.

The set of noise separation trajectories calculated from the test words clearly illustrates distinct dimension values of the ambient noise within the silence, and the utterances and aspirations associated with speech.

5.2.2 Phoneme Trajectory Properties

The next set of experiments examines the calculated trajectories for properties that can be associated with specific phonemes. Both the noise separation and fractal amplification trajectories are examined in this experiment.

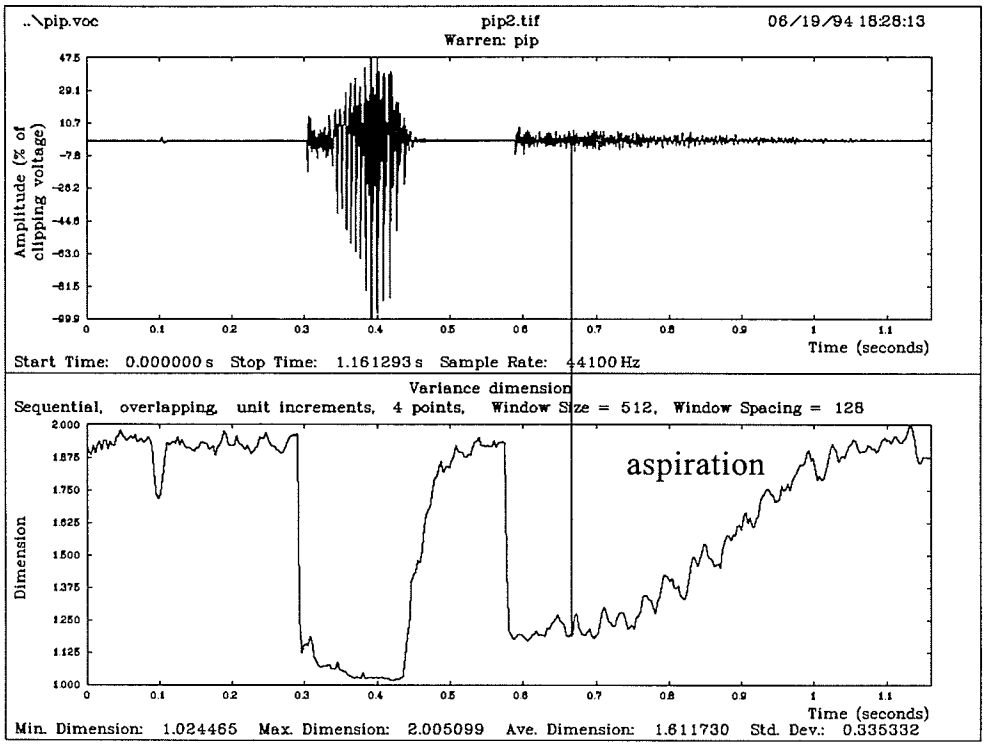


Fig. 5.6. Noise separation trajectory of /pɪp/ with aspiration.

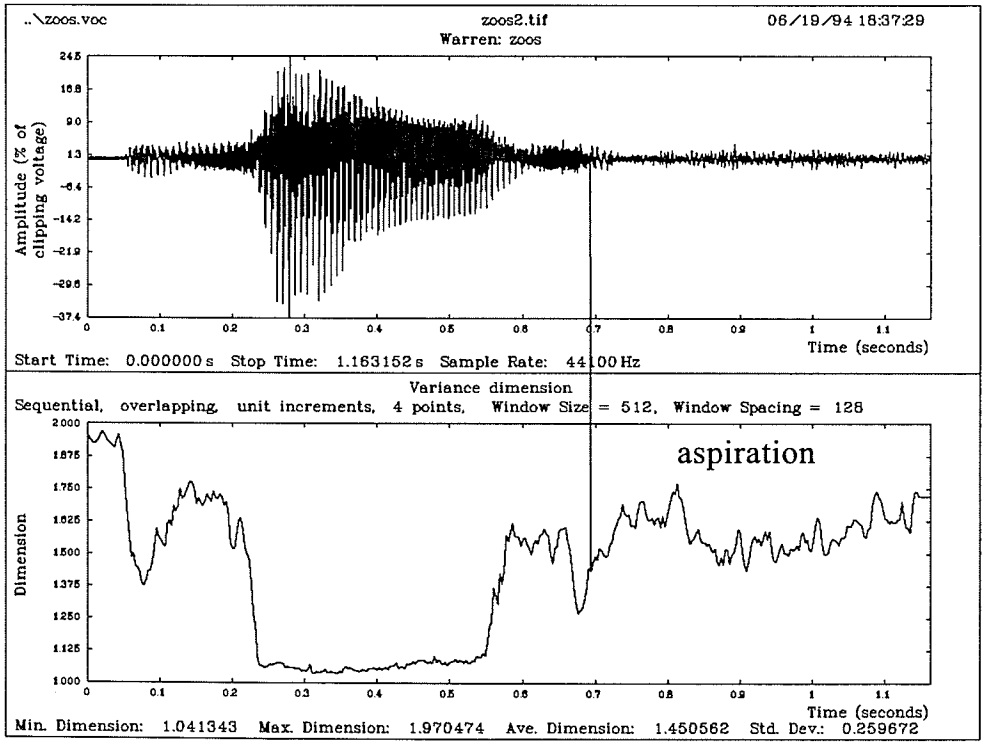


Fig. 5.7. Noise separation trajectory of /zuz/ with truncated aspiration.

5.2.2.1 Distinguishing Consonants from Vowels

Throughout the set of test words, the presence of a vowel is reflected in the noise separation trajectory by a very low and relatively constant dimension value between 1.0 and 1.1. Most of the consonants generally have a higher trajectory level with dimension values between 1.1 and 1.7, and show a greater amount of variation in the trajectory value. This type of behaviour is exhibited in the word banana, /bənænə/, shown in Fig. 5.8.

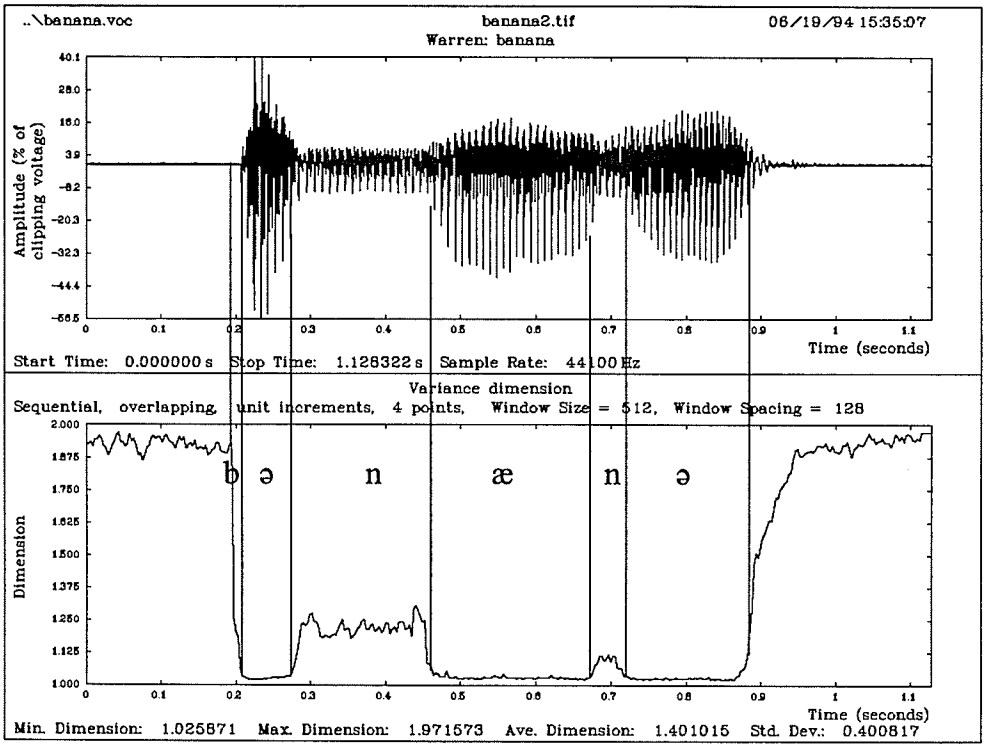


Fig. 5.8. Annotated noise separation trajectory of /bənænə/.

An exception is the behaviour of the trajectory for the nasal consonant, /ŋ/, which is almost indistinguishable from the trajectories for the vowels, as can be seen in the test word ringing, /rɪŋɪŋ/, shown in Fig. 5.9. However, the fractal amplification trajectory for /rɪŋɪŋ/ exhibits abrupt changes in behaviour between /ŋ/ and the vowels, as seen in Fig. 5.10.

5.2.2.2 Properties of Individual Phonemes

Although the properties of the noise separation trajectories distinguish between the utterance and the surrounding silence, as well as between the different classes of consonants and the vowels, the information present is not sufficient for distinguishing between

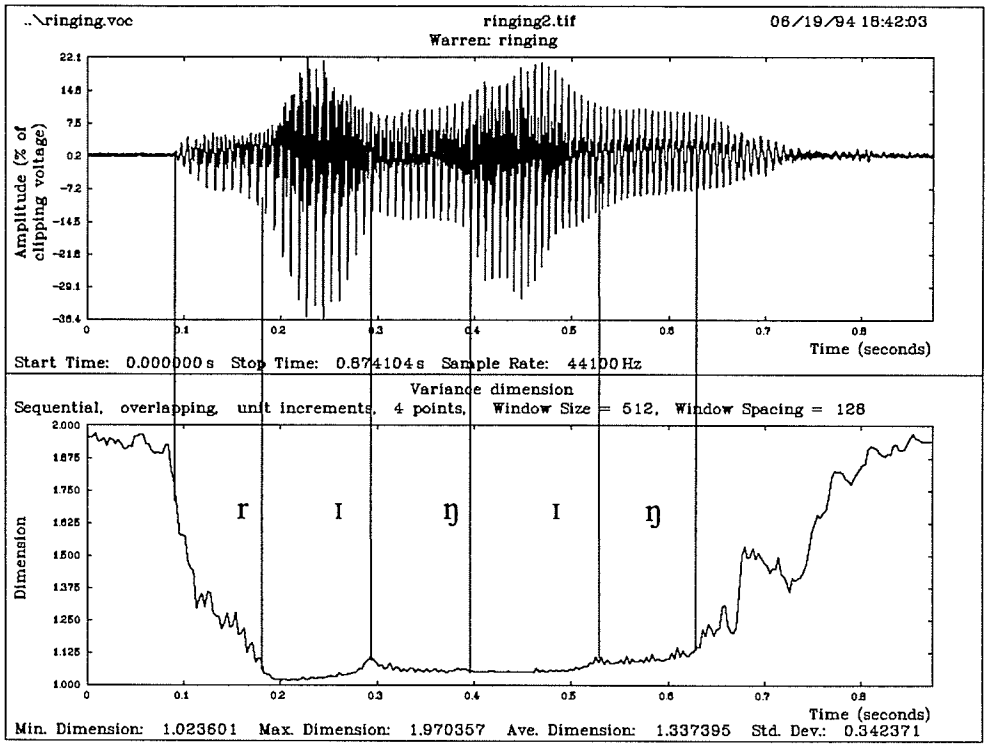


Fig. 5.9. Noise separation trajectory of /rɪŋɪŋ/.

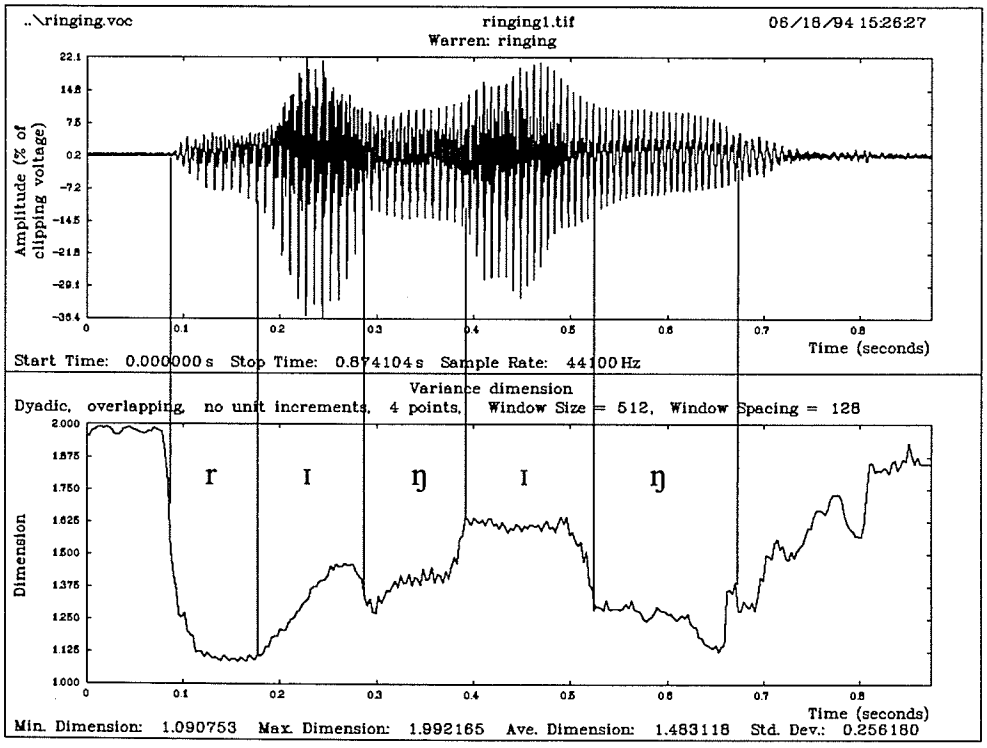


Fig. 5.10. Fractal amplification trajectory of /rɪŋɪŋ/.

different vowels, or between different consonants to a lesser degree. Examination of the fractal amplification trajectories of the test words indicates specific trajectory patterns occurring with many of the phonemes.

Consonant Properties

The trajectories of each consonant phoneme is observed to have properties that are characteristic of the class of the phoneme: stop consonants, fricatives, affricatives, nasals or approximants. As well, each phoneme's trajectories have their own unique properties.

The noise separation trajectories of stop consonants occurring in a position other than at the beginning of an utterance are characterized by the stop gap, for which the level of the trajectory rises to the level of the background noise, as previously seen in Fig. 5.3. In the case of stop consonants occurring at the beginning of an utterance, the vocal tract can start in the blocked position; thus, no stop gap is present in these cases. The drop in trajectory level from either the stop gap or initial silence is steeper than the drops for the fricatives, nasals, and approximants from the initial silence, with the dimension value falling from a value greater than 1.85 to a value less than 1.4 in a time span ranging from 3×10^{-4} to 1×10^{-3} seconds. The fall in dimension value has a slope between –770 and –2088 dimension units per second.

Fricatives have noise separation trajectories that exhibit a great degree of variability and irregularity, and are centered on a level between level of the ambient noise and the level of the vowels, the precise level varying from fricative to fricative. Trajectories of two occurrences of a typical fricative, /f/, are shown in Fig. 5.11. An exception to this tendency of the fricatives is the glottal fricative /h/, shown in Fig. 5.12, which exhibits little variability and irregularity. The drop in trajectory level from initial silence for fricatives is less steep than the stop consonants, and on the same order as the nasals and approximants. The amount of the drop in value is similar to the stop consonants in a time span ranging from 1×10^{-2} to 2.7×10^{-2} seconds. The fall in dimension value has a slope between -18.6 and –92.4 dimension units per second.

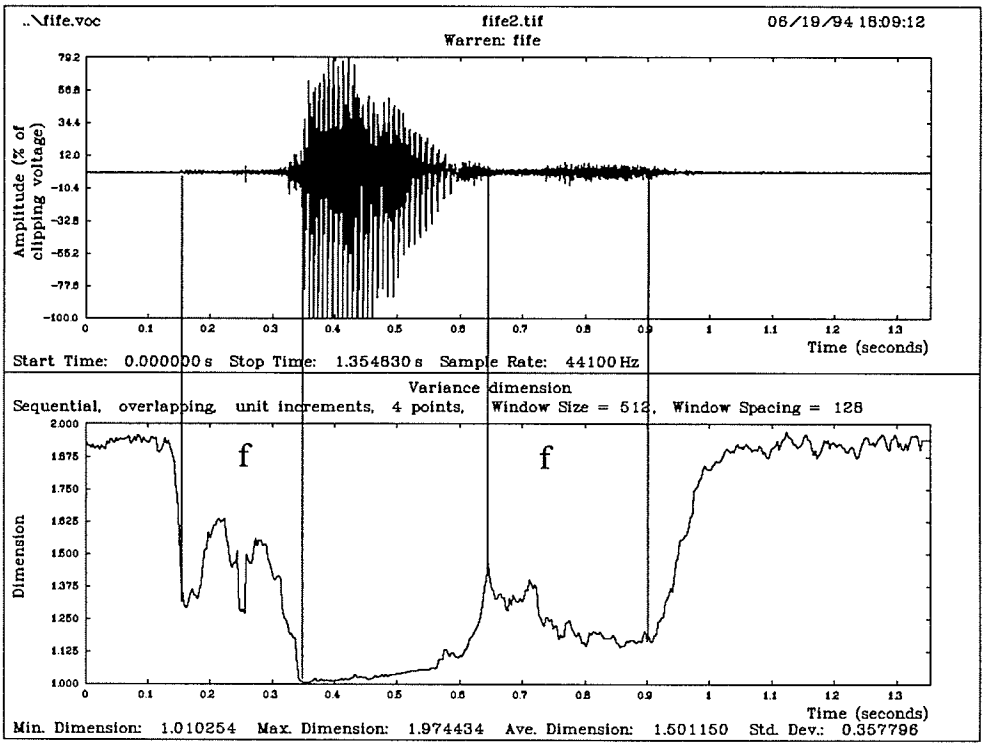


Fig. 5.11. Noise separation trajectory containing fricative /f/.

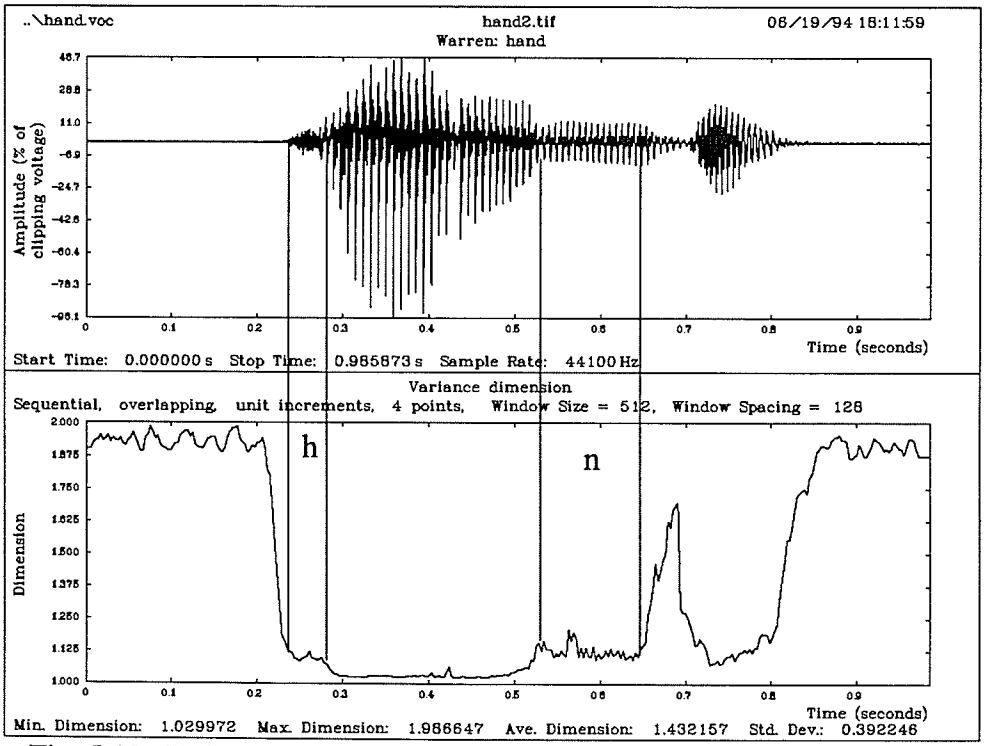


Fig. 5.12. Noise separation trajectory containing fricative /h/ and nasal /n/.

Affricatives, being the result of an audible fricative produced from a stop closure, have noise separation trajectories that exhibit a slope with the same properties as the stop consonants from either the initial noise or the stop gap followed by a period with an intermediate level trajectories similar to the fricatives. The fall in dimension value has a slope between –1635 and –2217 dimension units per second. Figure 5.13 shows the trajectory for an utterance containing two occurrences of the affricative /tʃ/.

The noise separation trajectories of the nasals, like the fricatives, occur at levels intermediate to the trajectory level of the background noise and the trajectory level of the vowels, with the exception of the phoneme /ŋ/, which is at the same level as the vowels as previously shown in Fig. 5.9. However, the degree of variability is much lower for the nasals, and the variations are much more regular. The drop in trajectory level from initial noise is much less steep than the stop consonants, and has duration similar to the fricatives and approximants. The fall in dimension value has a slope between –27.6 and –30.5 dimension units per second. An example trajectory for the nasal /n/ can be seen previously in Figure 5.12.

Approximants have noise separation trajectories sloping down to the level of a following vowel or up from a preceding vowel within the syllable. These trajectories exhibit little variation or irregularities. The slopes of the drop in trajectories levels for initial approximants have durations similar to the nasals and fricatives. The fall in dimension value has a slope between –19.3 and –45.2 dimension units per second. A trajectory containing an occurrence of the approximant /j/ is shown in Figure 5.14.

Examination of the fractal amplification trajectories shows phoneme specific pattern occurring for several of the phoneme. One such example is the phoneme /ʒ/ in the test word zoos, shown in Figure 5.15. The trajectory for /ʒ/ is characterized by a steep, jagged rise in level, follow by a plateau for most of the remainder of the phoneme's duration, concluding with a sharp drop.

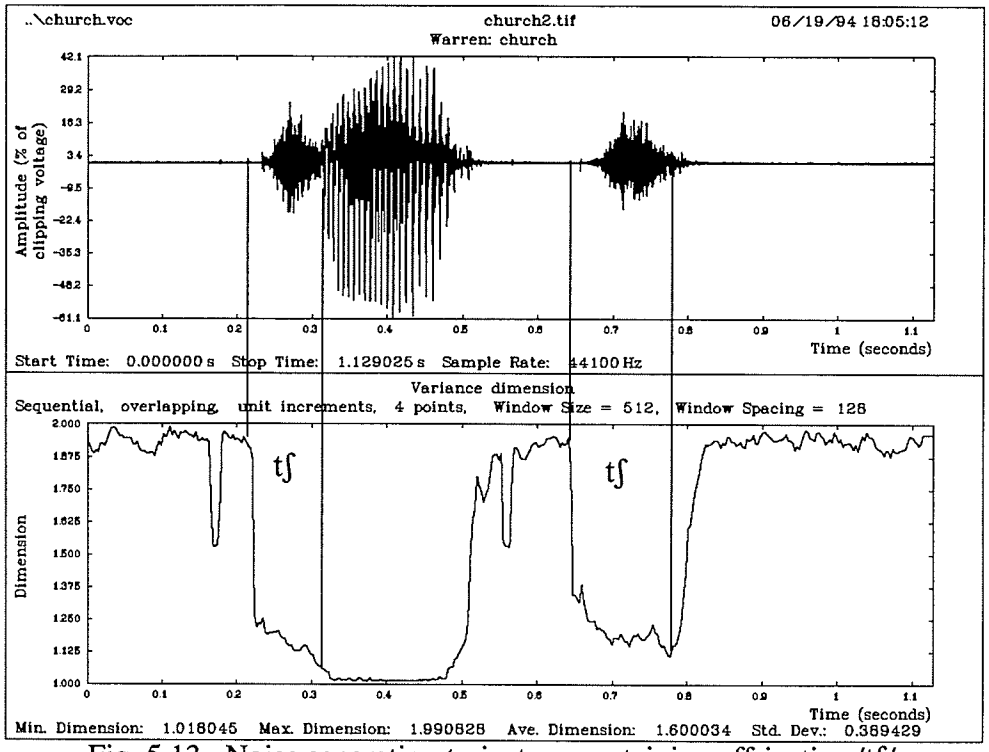


Fig. 5.13. Noise separation trajectory containing affricative /tʃ/.

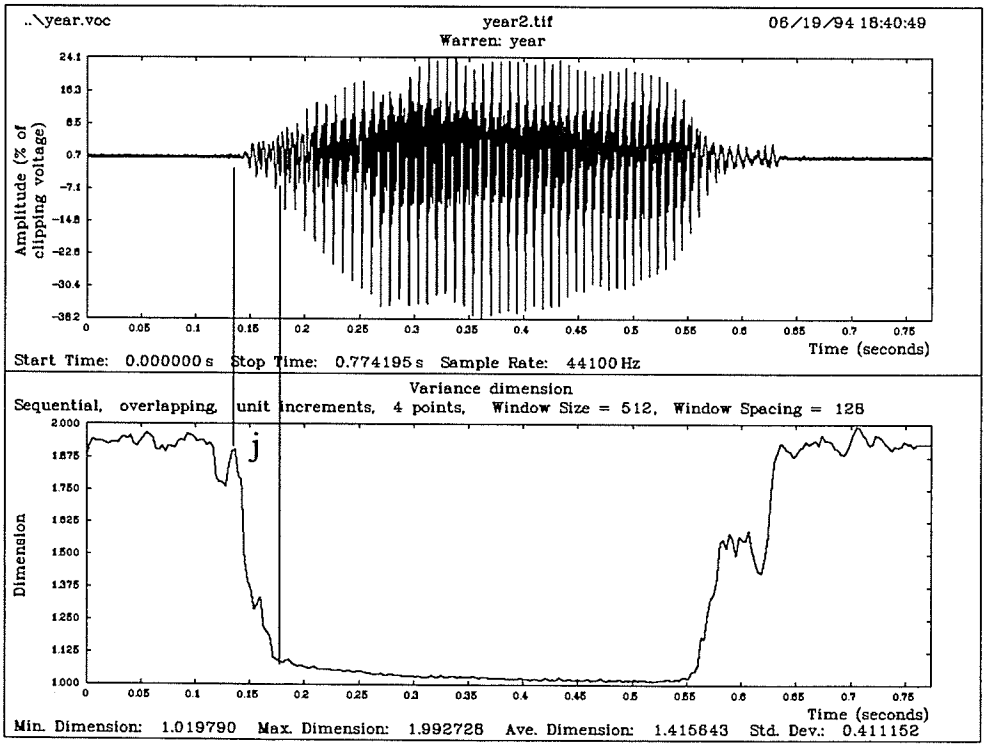


Fig. 5.14. Noise separation trajectory containing approximant /j/.

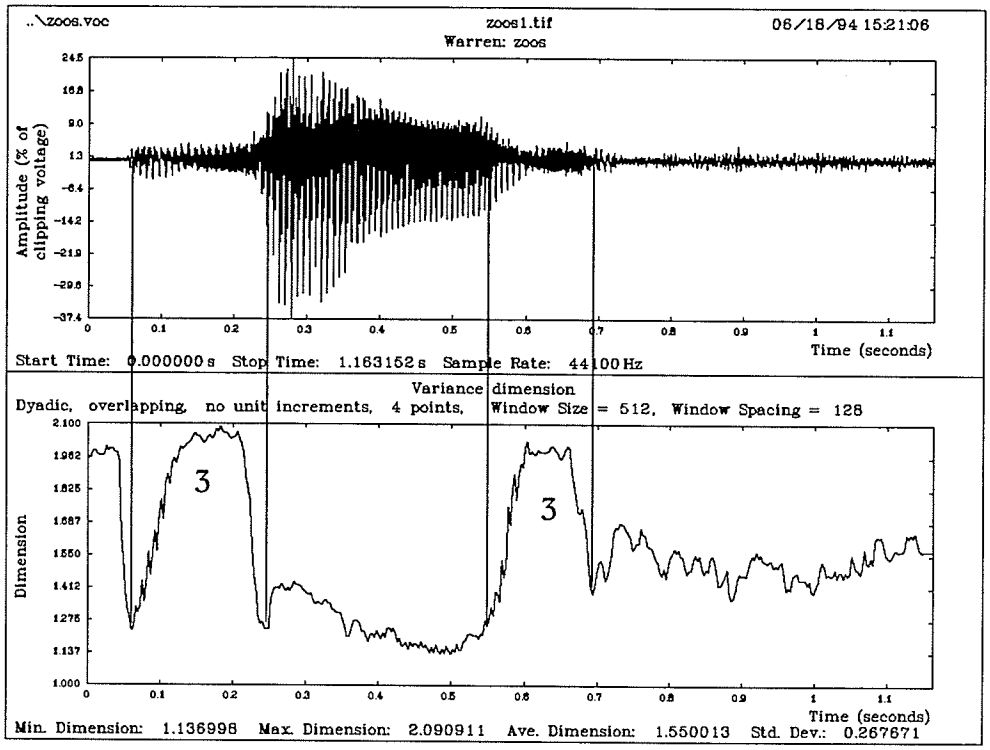


Fig. 5.15. Fractal amplification trajectory of zoos.

Brief descriptions of the fractal amplification trajectories for the consonants are given in Table 5.3. Since words alone are inadequate to fully describe the subtleties of the trajectory shapes, the complete set of trajectories of the test words is included in Appendix A. A more quantitative measure for the individual phoneme trajectories is presented by the measured values for occurrences of the phoneme at the beginning and within an utterance, which are given in Table 5.4a and Table 5.4b, respectively. These measured values are taken from the primary test words for the phoneme when it is available from the set of test words in the desired position. Phonemes marked with N/A are not available in the set of test words for the desired position. If more than one occurrence of the phoneme occurs in the test word in the desired position, i.e. the /ŋ/ in ringing, the values for the first occurrence are listed.

The fractal amplification dimension values for the consonant phonemes exhibit a high degree of overlap in range, as can be seen from their minimum and maximum dimensions. Thus, the dimension values themselves are inadequate for the identification of specific phonemes. Phoneme specific information is present only when the trajectory is considered as a whole.

Vowel Properties

The noise separation trajectories of the vowel show very little difference between the vowels, since they have very low dimension values with little variation in level. For example, the /æ/ in the test word bad has an average noise separation dimension of 1.035 with a standard deviation of 0.00348, and the /ʌ/ in the test word bud has an average noise separation dimension of 1.042 with a standard deviation of 0.00621.

With the fractal amplification trajectories, features characteristic to several of the vowel phonemes are noticed. For example, consider the occurrence of the phoneme /aɪ/ in the test words buy, fife, mime, and tight, shown in Fig. 5.16a, Fig. 5.16b, Fig. 5.16c, and Fig. 5.16d, respectively. In each of these words, the trajectory of the phoneme /aɪ/ rises for most of its duration followed by a sharp drop.

Descriptions of the fractal amplification trajectories for the vowels are given in Table 5.5. Like the consonants, the complete set of trajectories of the vowel test words is included in Appendix A as well. The measured values for occurrences of the vowel phonemes within an utterance are given in Table 5.6. These measured values are taken from the primary test words for the phoneme.

The fractal amplification dimension values of the vowels have a greater range of values than the noise separation dimension values. However, like the consonants, there is a good deal of overlap in the ranges of the individual phonemes. Once again it is the shape of the trajectory that contains the phoneme specific information.

Table 5.3. Fractal amplification descriptions for consonant phonemes.

Phoneme	Trajectory description
p	sharp initial drop followed by a small increase and a rapid drop
b	sharp initial drop followed by a very short plateau
t	sharp initial drop followed by a small increase and a gradual drop
d	sharp initial drop followed by a very short spike
k	sharp initial drop followed by a large spike
g	sharp initial drop followed by a spike
tʃ	short duration dip from ambient noise level followed by a noise level plateau and a smooth decline
dʒ	short duration dip followed by a high-level uneven plateau and a sharp drop
f	highly erratic mid-level plateau
v	uneven plateau
θ	low-level uneven plateau with a sharp rise to a higher-level plateau
ð	uneven plateau
s	steep smooth rise followed by an uneven plateau and a steep smooth fall
z	steep jagged rise followed by a short plateau and a steep smooth fall
ʃ	steep smooth rise followed by an uneven plateau and a steep smooth fall
ʒ	steep smooth rise followed by a short plateau and a steep smooth fall
h	decreasing slope with a few spikes
r	plateau with small variations
l	uneven plateau
m	uneven plateau
n	plateau with small variations
ŋ	plateau with small variations
j	short plateau with small variations
w	plateau with small variations

Table 5.4a. Fractal amplification data for consonant phonemes beginning utterances.

Phoneme	Minimum Dimension	Maximum Dimension	Average Dimension	Standard Deviation
p	1.350	1.554	1.481	0.0479
b	1.053	1.122	1.063	0.0150
t	1.235	1.434	1.352	0.0549
d	1.320	1.665	1.386	0.0925
k	1.181	1.858	1.383	0.174
g	1.217	1.447	1.363	0.0750
tʃ	1.265	1.992	1.783	0.203
dʒ	1.495	2.046	1.947	0.122
f	1.267	1.808	1.624	0.108
v	1.146	1.196	1.168	0.0140
θ	1.189	2.015	1.632	0.232
ð	N/A			
s	1.411	2.273	2.079	0.196
z	1.235	2.091	1.861	0.259
ʃ	1.509	2.130	2.036	0.0814
ʒ	N/A			
h	1.439	1.698	1.571	0.0623
r	1.059	1.174	1.096	0.0255
l	1.148	1.488	1.300	0.0827
m	1.190	1.396	1.302	0.0543
n	1.181	1.302	1.213	0.0280
ŋ	N/A			
j	1.310	1.544	1.379	0.0804
w	1.051	1.320	1.105	0.0576

Table 5.4b. Fractal amplification data for consonant phonemes within utterances.

Phoneme	Minimum Dimension	Maximum Dimension	Average Dimension	Standard Deviation
p	1.290	1.574	1.441	0.0943
b	1.364	1.638	1.468	0.0557
t	1.227	1.419	1.329	0.0502
d	1.423	1.616	1.531	0.0494
k	1.240	1.921	1.568	0.187
g	1.444	1.687	1.578	0.0686
tʃ	1.555	2.058	1.9982	0.161
dʒ	1.524	1.993	1.883	0.134
f	1.400	1.663	1.556	0.0509
v	1.419	1.939	1.686	0.158
θ	1.651	2.142	1.968	0.138
ð	1.169	1.450	1.264	0.0512
s	1.850	2.169	2.066	0.0603
z	1.392	2.028	1.796	0.212
ʃ	1.629	2.113	1.993	0.104
ʒ	1.428	2.118	1.943	0.127
h	N/A			
r	N/A			
l	1.046	1.477	1.124	0.0670
m	1.117	1.438	1.256	0.0645
n	1.127	1.303	1.189	0.0309
ŋ	1.299	1.644	1.434	0.0829
j	N/A			
w	N/A			

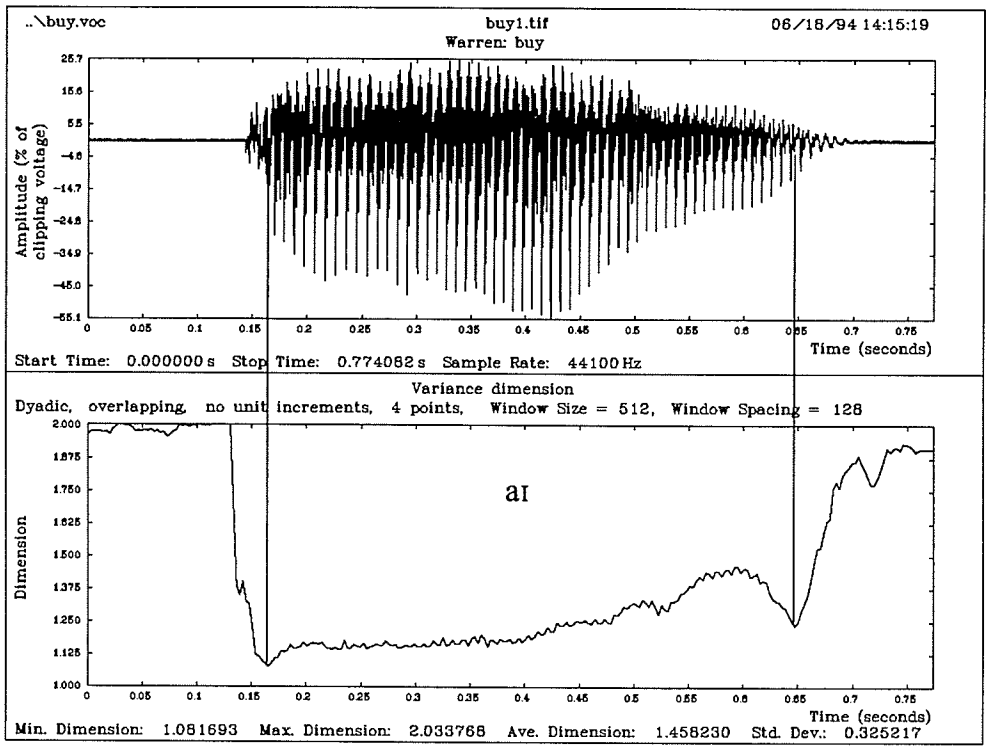


Fig. 5.16a. Fractal amplification trajectory of /baɪ/.

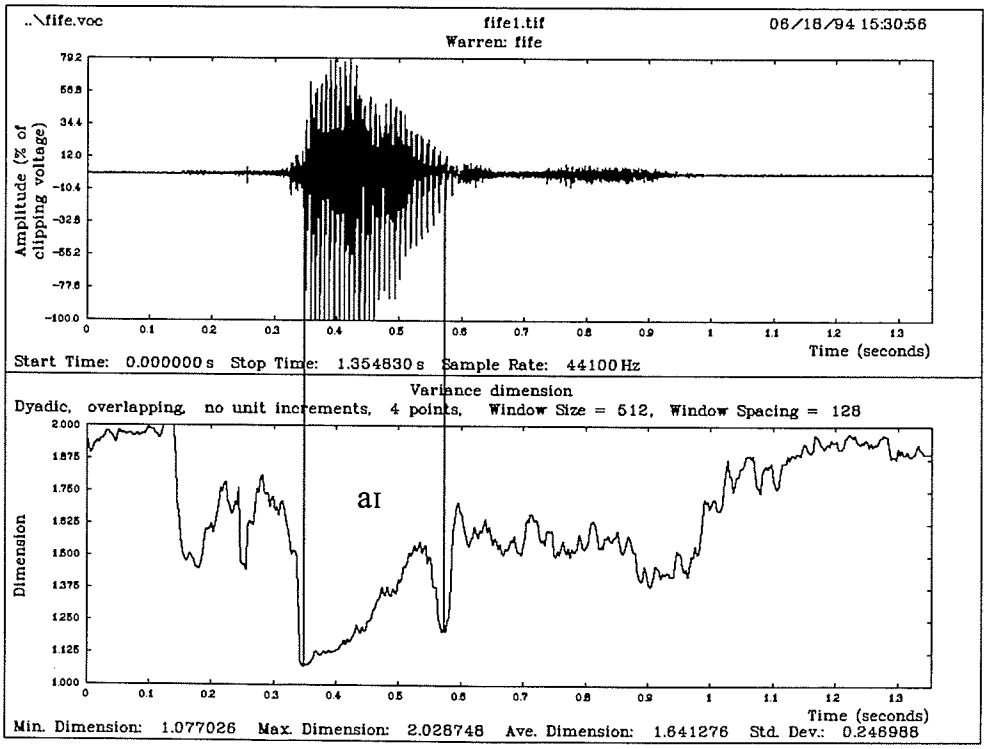


Fig. 5.16b. Fractal amplification trajectory of /faɪf/.

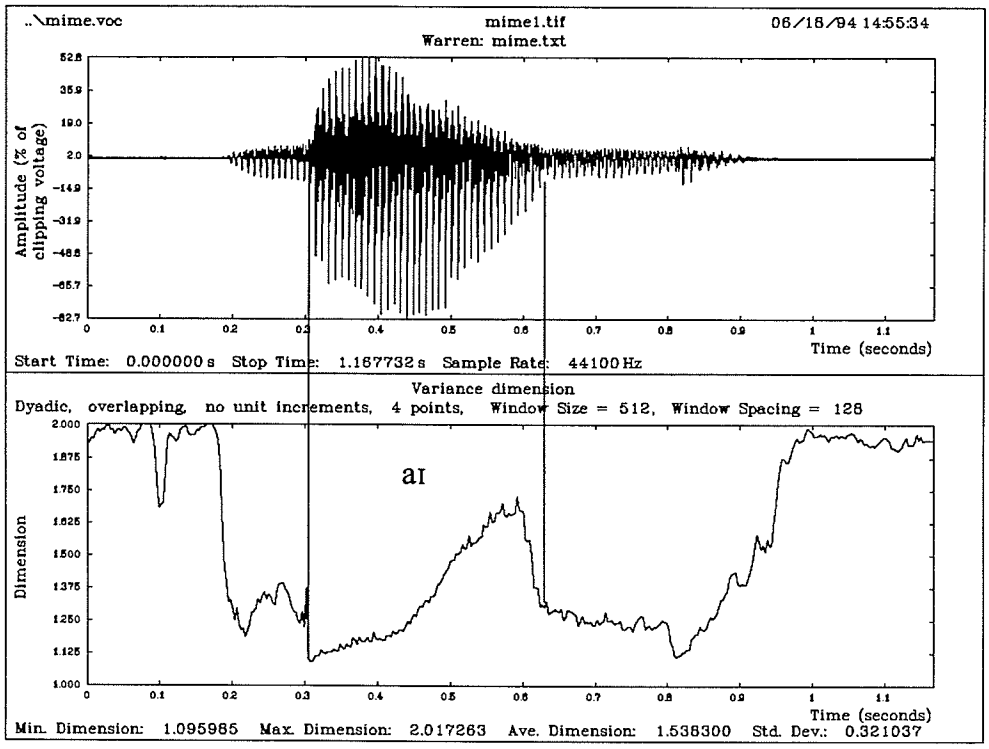


Fig. 5.16c. Fractal amplification trajectory of /maɪm/.

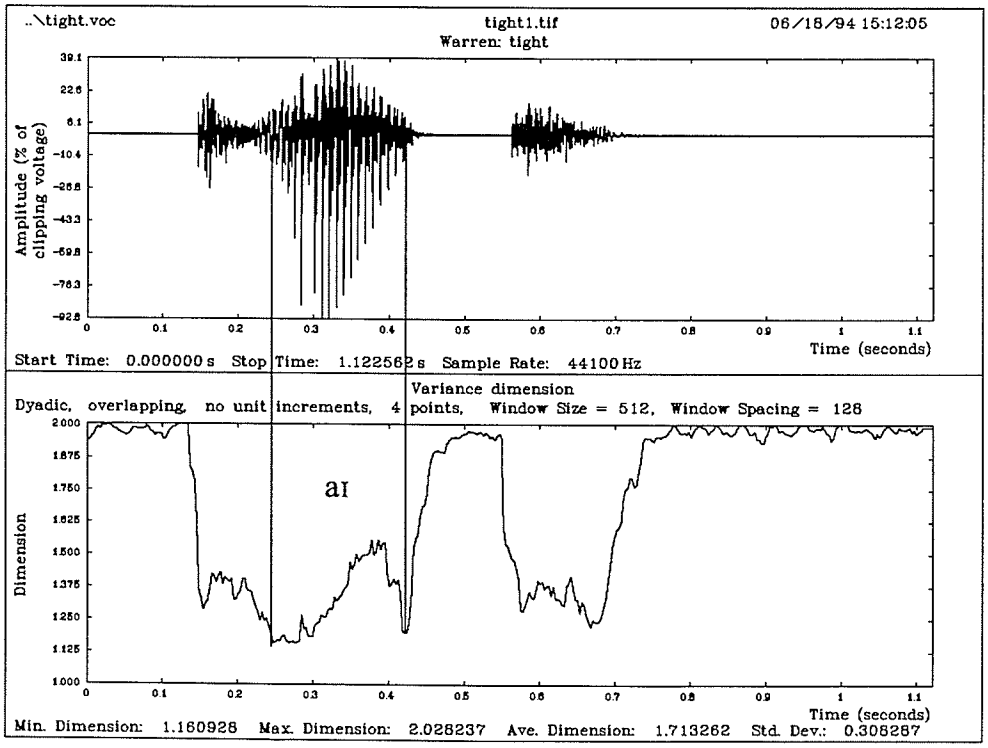


Fig. 5.16d. Fractal amplification trajectory of /taɪt/.

Table 5.5. Fractal amplification descriptions for vowel phonemes.

Phoneme	Trajectory description
i	rises to a very jagged plateau before descending
ɪ	jagged and descending
ɛ	slight rise and fall with small variations
æ	plateau with small variations
ɑə	plateau with very small variations
ɑ	slight rise with small variations
oə	slight rise
ʊ	plateau with very small variations
u	slight rise and fall with small variations followed by a smooth rise
ʌ	slight fall with very small variations
ʌə	slight fall and rise with very small variations
ə	slight rise and fall with very small variations
eɪ	plateau with slight variation an slight dip in center
o	quick rise followed by slow descent with very small variations.
aɪ	rises for most of its duration followed by a sharp drop
aʊ	slight fall
ɔɪ	gradually increasing rise to a plateau with drop with very small variations
ɪə	quick rise followed by slow descent with small variations.
ɛə	slight fall with small variations
ʊə	slight rise

Table 5.6. Fractal amplification data for vowel phonemes.

Phoneme	Minimum Dimension	Maximum Dimension	Average Dimension	Standard Deviation
i	1.589	1.846	1.753	0.0578
ɪ	1.283	1.545	1.429	0.0747
ɛ	1.212	1.454	1.354	0.0558
æ	1.223	1.361	1.287	0.0337
ɑə	1.139	1.265	1.182	0.0223
ɑ	1.135	1.255	1.193	0.0303
oə	1.038	1.165	1.082	0.0400
ʊ	1.071	1.137	1.106	0.0153
u	1.085	1.293	1.183	0.0643
ʌ	1.152	1.300	1.210	0.0223
ʌə	1.176	1.333	1.236	0.265
ə	1.191	1.301	1.242	0.248
eɪ	1.287	1.588	1.452	0.0612
o	1.039	1.236	1.100	0.580
aɪ	1.139	1.464	1.252	0.0970
aʊ	1.035	1.250	1.127	0.0567
ɔɪ	1.052	1.556	1.339	0.177
ɪə	1.168	1.562	1.317	0.122
ɛə	1.136	1.374	1.273	0.0615
ʊə	1.025	1.228	1.082	0.041

5.3 Packet Radio Transmissions

The variance fractal dimension can be applied to signal types other than speech. Packet radio transmission are chosen to illustrate this. The sampling rate has been reduced from the 44.1 kilosamples per seconds used for speech to 11.025 kilosamples per second to eliminate the artificial correlation introduced at higher sampling rates on the recording media. The duration of the window used in the calculation of fractal amplification trajectories is maintained by reducing the number of samples from 512 used for speech to 128. The packet transmission used in this experiment were received at the University of Manitoba Amateur Radio Society station on the date of July 14, 1995 from a packet BBS operating at a frequency of 144.310 Mhz. The packets were recorded at 3 hour intervals from 9:00 a.m. to 9:00 p.m.

Figure 5.17a shows the trajectory of a typical packet transmission from the 9:00 a.m. recording session. The trajectory exhibits changes in level corresponding to the transitions from background noise to the packet, shown in detail in Fig. 5.17b, from the packet to transmitter noise, shown in detail in Fig. 5.17c, and from transmitter noise to background noise, shown in detail in Fig. 5.17d. Trajectories of sample packets from 12:00 p.m., 3:00 p.m., 6:00 p.m. and 9:00 p.m. are included in Appendix A.

The most critical transition to detect is the transition from the background noise to the packet transmission, since this transition identified the beginning of the packet to be decoded by the TNC. This transition is very clear in the trajectories from all samples. The transition from the packet to the transmitter noise is less critical since the TNC detects the end of the packet from its content. This transition is less pronounced than the transition from background noise to packet transmission, especially in the 3:00 p.m. and 6:00 p.m. recordings. In general the trajectories of the packet and the noise types show a range of values that increasing in the following order: packet transmission, transmitter noise, and background noise.

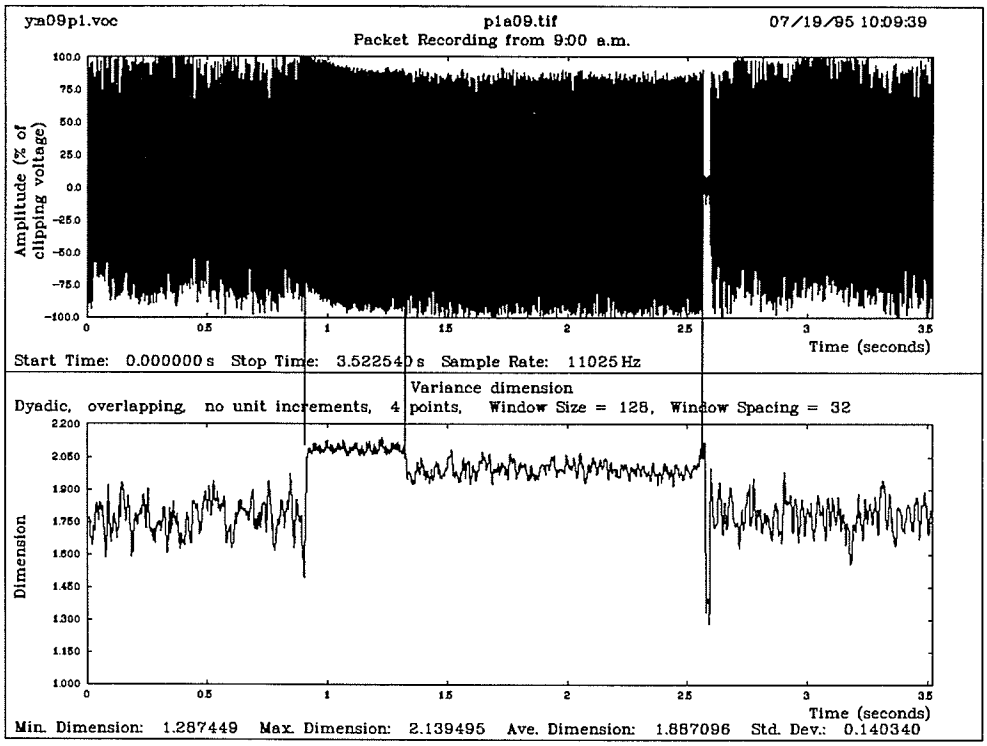


Fig. 5.17a. Variance fractal dimension of a packet radio transmission.

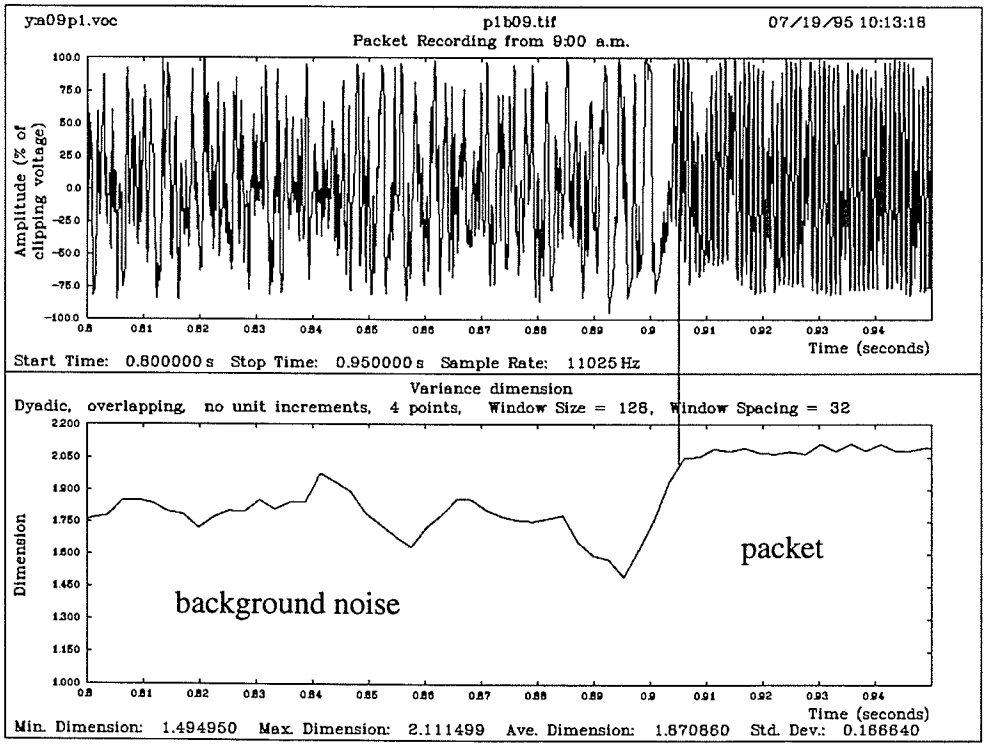


Fig. 5.17b. Transition between background noise and packet.

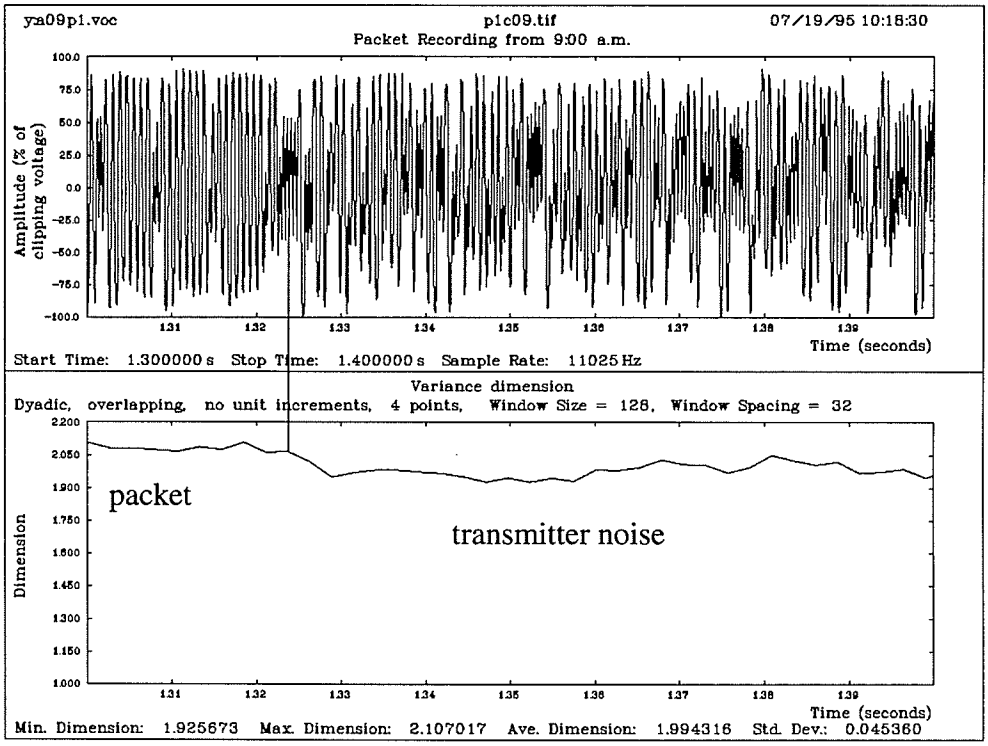


Fig. 5.17c. Transition between packet and transmitter noise.

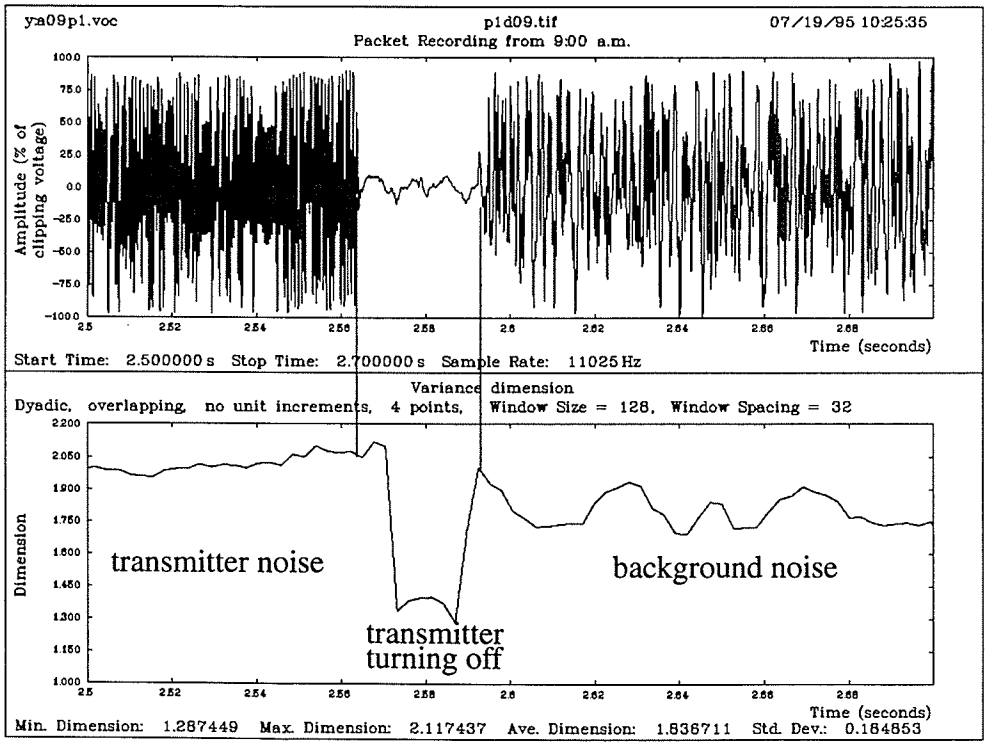


Fig. 5.17d. Transition between transmitter noise and background noise.

5.4 Summary

The experiments on generated test data with known dimensions demonstrate that the variance fractal dimension values calculated using the noise separation parameter setting exhibits a tendency to underestimate the actual dimension value, especially for the higher dimension values. A greater degree of accuracy is obtained for the variance fractal dimension calculated using the fractal amplification parameter setting.

The noise separation setting is not without its uses. The uncorrelated background noise produces has a higher noise separation dimension than the more correlated speech signals. This provides a method for identifying the presence of an utterance. In addition, the occurrence of vowels within an utterance can be identified by a noise separation trajectory with a very low value and exhibiting little variation. Most of the consonants, with the exception of the nasal /ŋ/ tend to have dimension values higher than the vowels. This offers the possibility to separate the consonants from the vowels, with the exception /ŋ/ being demonstrated to have a fractal amplification trajectory with significantly different behaviour from the surrounding vowels.

Fractal amplification trajectories exhibit a greater degree of detail than the noise separation trajectories. Characteristic shapes in the fractal amplification trajectories are noticeable for several of the phonemes.

The variance fractal dimension can be applied to temporal signals other than speech. The trajectories obtained for packet radio transmission exhibit changes in level corresponding to transitions between background noise, packet transmissions, and transmitter noise.

CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

The variance fractal dimension algorithm can be used to produce temporal trajectories of regularly spaced dimension values calculated on localized windows of a temporal signal. These trajectories exhibit characteristic features related to the content of the signal.

One setting of the parameters for the calculation of the parameters for the calculation of the variance fractal dimension takes advantage of the correlation between samples in close proximity to each other in a signal such as speech, and the lack of correlation in the background noise that is present during periods of silence, by calculating the dimension only on the smaller time increments. The calculated dimension value for this setting is higher for the uncorrelated noise, which generally has a noise separation dimension greater than 1.85, than for the more correlated speech, which has a noise separation dimension value less that 1.40 initially and less than 1.70 in general. The duration of the initial level changes is typically between 3×10^{-4} and 1×10^{-3} seconds for stop consonants and affricatives, and between 1×10^{-2} and 2.7×10^{-2} seconds for fricatives, nasals, and approximants. The level changes have slopes between –770 and –2217 dimension units per second for initial stop consonants and affricatives, and between –18.6 and –92.4 dimension units per second for initial fricatives, nasals, and approximants. These changes in level make it possible to identify the boundaries of the speech signal and the surrounding periods of silence. This setting of parameters is called noise separation because of its ability to distinguish the signal from noise.

Another setting of the parameters for calculation of the variance fractal dimension spaces out the time increments used in the calculation over a larger range in a dyadic selection to eliminate the effects of correlation between samples in close proximity. This setting enhances the features in the trajectory obtained, thus the setting is called fractal amplification.

In addition to detecting the boundaries between an utterance and surrounding silence, noise separation also detects the boundaries between the highly correlated vowel, which tend to have low dimension values between 1.0 and 1.1 and the majority of the consonants, which tend to be less correlated, and have higher dimension values in the range of

1.1 to 1.7. Also, the shape of the trajectory obtained from the consonants appears to be related to the consonant class. Most notable is the identification of the stop gaps which have dimensions at the level of the background noise.

Since the details brought out by fractal amplification exhibit a great variety of shapes, several of the phonemes can be seen to have distinctive trajectory shapes. This opens the possibility of using shape recognizing neural networks to process trajectories obtained from speech for the purposes of recognition.

The applicability of the variance fractal dimension to signal types other than speech is demonstrated by the trajectories obtained from packet radio transmissions. These trajectories indicate the boundaries of the various stages of the received signal: background noise, packet transmission, and transmitter noise. These boundaries are indicated by a change in the dimension value and standard deviation of the signal.

In summary, this thesis has contributed to technical knowledge through:

- An implementation of the variance fractal dimension algorithm which allows parameters to be set in different configurations for the extraction of different types of information.
- Noise separation is used for identification of the boundaries between speech and the surrounding silence. Internal boundaries between consonants and vowels are also determined.
- Potential for the use of the fractal amplification trajectory shapes for identification purposes is demonstrated.
- The beginning of packet radio transmissions are detected.

Future work could include the following extensions and improvements:

- Testing on a wider variety of test words to examine the behaviour of the trajectories over more complex phonemics structures such as consonant clusters, and to study the effects of different adjacent phonemes on a phoneme's trajectory.
- Examination of test words obtain from multiple speakers to determine the dependence/independence of characteristic features within trajectories on the speaker.
- The trajectories obtained from speech should be passed through a neural network for the purpose of recognition.

REFERENCES

[Atal75] B. Atal, "Linear prediction of speech -- Recent advances with applications to speech analysis," *Speech Recognition Invited Paper Presented at the 1974 IEEE Symposium*, New York: Academic Press, pp. 221-230, 1975.

[Bris86]G. Bristow, *Electronic Speech Recognition.* New York: MacGraw Hill, 395 pp., 1986.

[BrSh75] D. J. Broad and J. E. Shoup, "Concepts for Acoustic Phonetic Recognition," *Speech Recognition Invited Paper Presented at the 1974 IEEE Symposium*, New York: Academic Press, pp. 243-274, 1975.

[Carr93] Phillip Carr, *Phonology.* London: Macmillan, 324 pp., 1993.

[Couc90] Leon W. Couch II, *Digital and Analog Communication Systems.* New York: Macmillan, 766 pp., 1990.

[Dene75] P. Denes, "Automatic Speech Recognition: Old and New Ideas," *Speech Recognition Invited Paper Presented at the 1974 IEEE Symposium*, New York: Academic Press, pp. 73-82, 1975.

[Devr86] Luc Devroye, *Non-Uniform Random Variate Generation.* New York: Springer-Verlag, 843 pp., 1986.

[DoSh88] G. Docherty and L. Shockey, "Speech Synthesis," *Aspects of Speech Technology,* Edinburgh University Press, pp. 144-183, 1988.

[Furu89]S. Furui, *Digital Speech Processing, Synthesis, and Recognition.* New York: Marcel Dekker, Inc., 390 pp., 1989.

[GaFG91] N. Gache, P. Flandrin, and D. Garreau, "Fractal dimension estimators for fractional Brownian Motions," *Proc. IEEE Int. Conf. Acoust, Speech, Signal Processing,* (Toronto, ON), IEEE CH2977-7/91, pp. 3557-3560, 1991.

[GrKi94] W. Grieder and W.Kinsner, "Speech Segmentation by Variance Fractal Dimension," *Proc. Canadian Conference on Electrical and Computer Engineering,* (Halifax, NS; Sept. 25-28, 1994), pp. 481-485, 1994.

[Harr88] J. Harrington, "Automatic Recognition of English Consonants," *Aspects of Speech Technology,* Edinburgh University Press, pp. 69-143, 1988.

[Harr89] J. Harrison, "Introduction to Fractals," *Chaos and Fractals: The Mathematics Behind the Computer Graphics*, pp. 107-126, 1989.

[Kins94a] W. Kinsner, "Fractal and chaos engineering," *Lecture Notes*, Dept. Electrical & Computer Eng., University of Manitoba, Winnipeg, Manitoba, Canada, January 1994.

[Kins94b] W. Kinsner, "Fractal dimensions: Morphological, entropy, spectra, and variance classes," *Technical Report*, DEL94-4, Dept. Electrical & Computer Eng., University of Manitoba, Winnipeg, Manitoba, Canada, May 1994, 146 pp.

[Kins94c] W. Kinsner, "Batch and real-time computation of a fractal dimension based on variance of a time series," *Technical Report*, DEL94-6, ibid., June 15 1994, 22 pp.

[Koho88] T. Kohonen, "The neural phonetic typewriter," *Computer*, pp. 11-22, March 1988.

[Kurt94] Mike Curtis, *9600 Baud Packet Handbook.*, World Wide Web document, 1994.

[Lang92] A. Langi, "Code-excited linear predictive coding for high-quality and low bit-rate speech," M.Sc. thesis, The University of Manitoba, Winnipeg, MB, Canada, 138 pp., 1992.

[Mara91] P. Maragos, "Fractal Aspects of Speech Signals: dimension and interpolation," in Proc. IEEE ICASSP, CH2977/91, pp. 417-420, 1991.

[MaSu89] P. Maragos and F.-K. Sun, "Measuring fractal dimension: morphological estimates and iterative optimization," *SPIE Visual Communications and Image Processing IV*, pp. 416-430, Vol. 1199, 1989.

[MaSu93] P. Maragos and F.-K. Sun, "Measuring the fractal dimension of signals: morphological covers and iterative optimization," *IEEE Transactions on Signal Processing*, pp. 108-121, Vol. 41, No. 1, January 1993.

[MeSi88] W. Mendenhall and T. Sincich, *Statistics for the Engineering and Computer Sciences.* London: Collier Macmillan, 1036 pp., 1988.

[Newe75] A. Newell, "A tutorial on speech understanding systems," *Speech Recognition Invited Paper Presented at the 1974 IEEE Symposium*, New York: Academic Press, pp. 3-54, 1975.

[PeJS92] H. -O. Peitgen, H. Jurgens, and D. Saupe, *Chaos and Fractals.* New York: Springer-Verlag, 984 pp., 1992.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C*. New York: Cambridge University Press, 994 pp., 1992.

[Ross95] G. van Rossum, *FAQ: Audio File Formats*, Usenet FAQ, 1995.

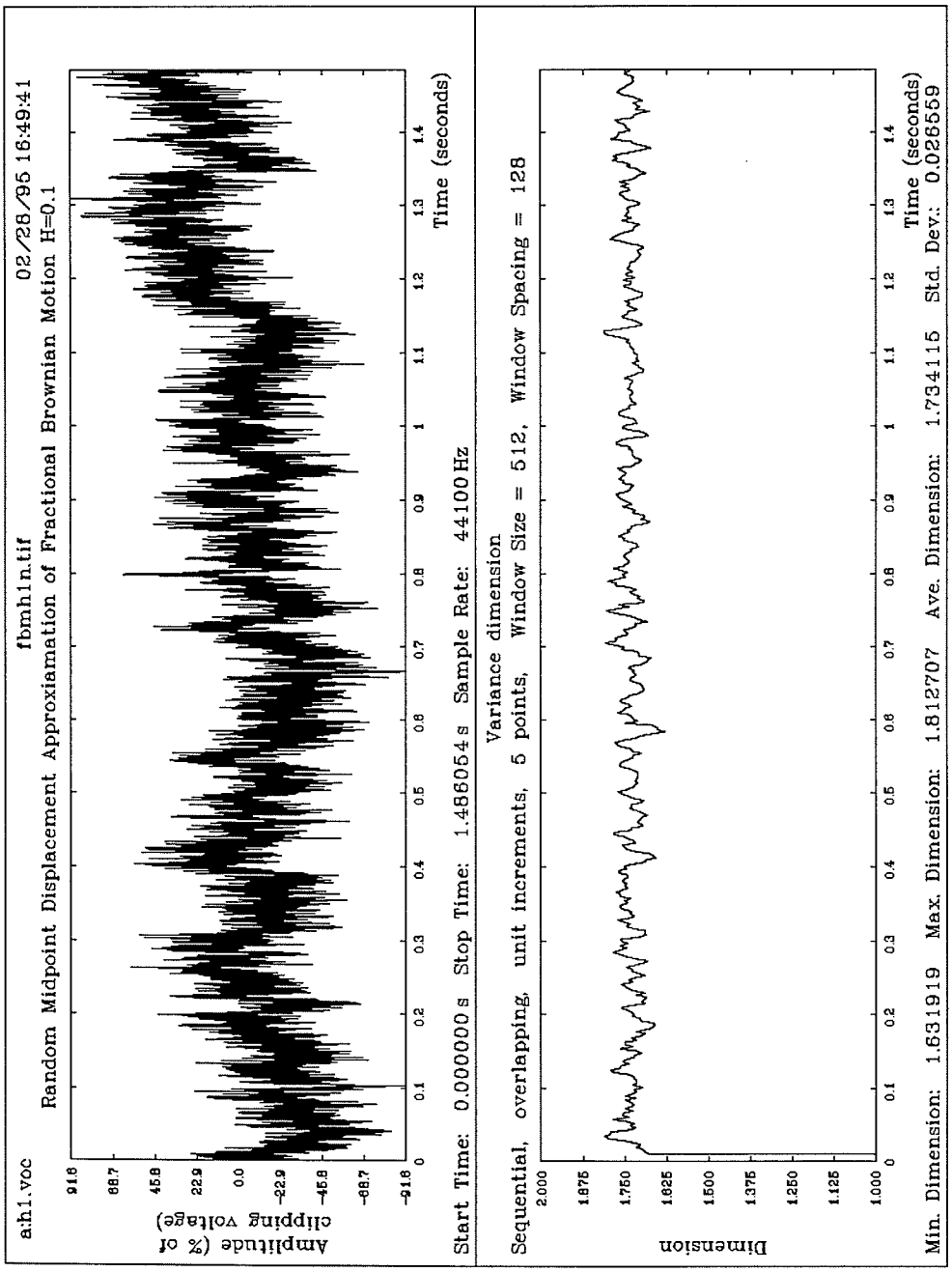
[ScRa75] R. W. Schafer and L. R. Rabiner, "Parametric representations of speech," *Speech Recognition Invited Paper Presented at the 1974 IEEE Symposium*, New York: Academic Press, pp. 99-150, 1975.

[Wein91] C. J. Weinstein, "Opportunities for advanced speech processing in military computer-based systems," in Proc. IEEE, vol. 79, IEEE 0018-9219/91, pp. 1626-1641, November 1991.

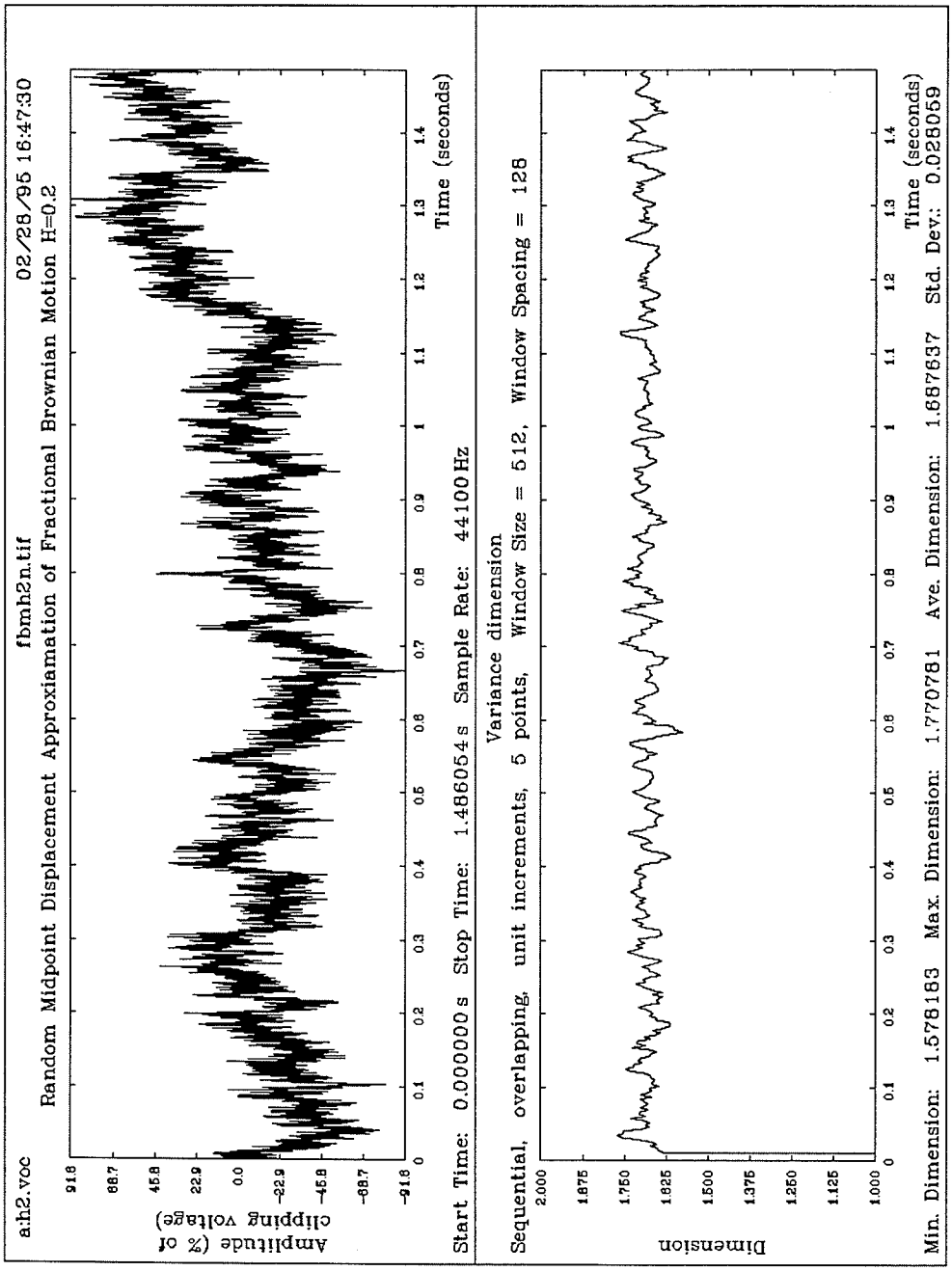
[ZhBM90] P. Zhang, H. Barad, and A. Martinez, "Fractal dimension estimation of fractional Brownian motion," *IEEE Southeastcon '90 Proceedings,* vol. 3, pp. 934-939, 1990.

Appendix A
Observed Trajectories

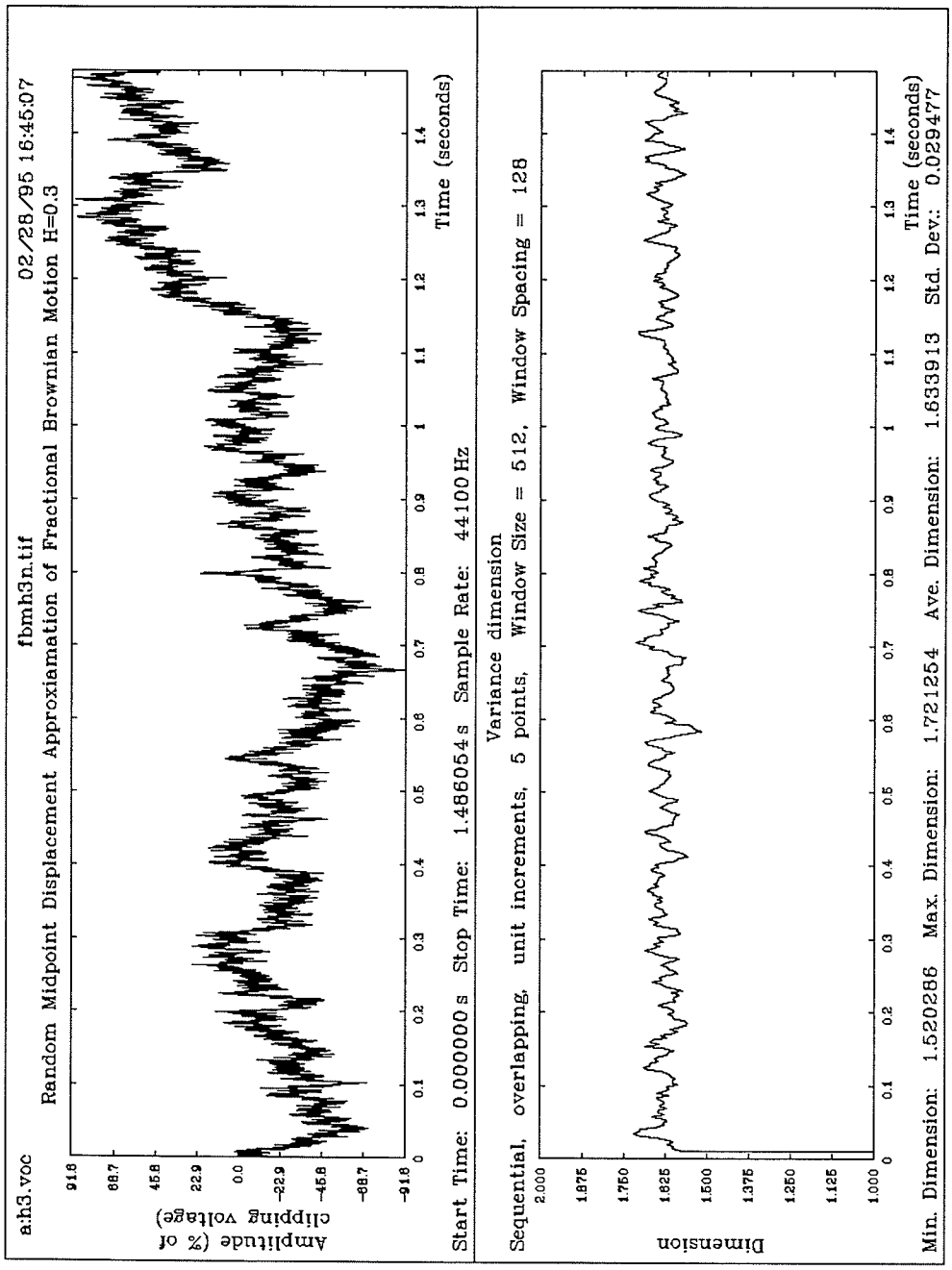
Fractal Brownian Motion, H = 0.1, Noise Separation Parameters



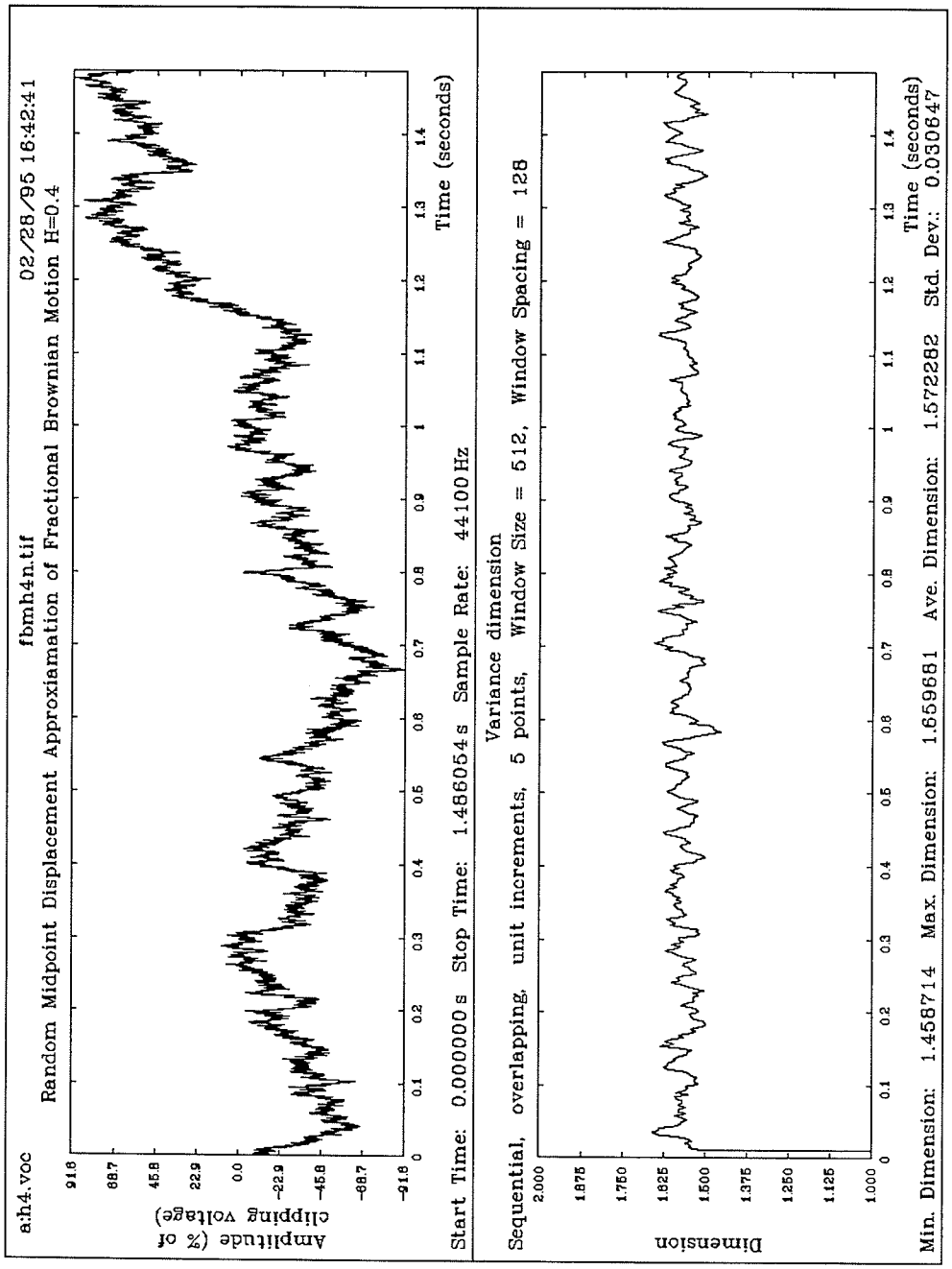
Fractal Brownian Motion, H = 0.2, Noise Separation Parameters



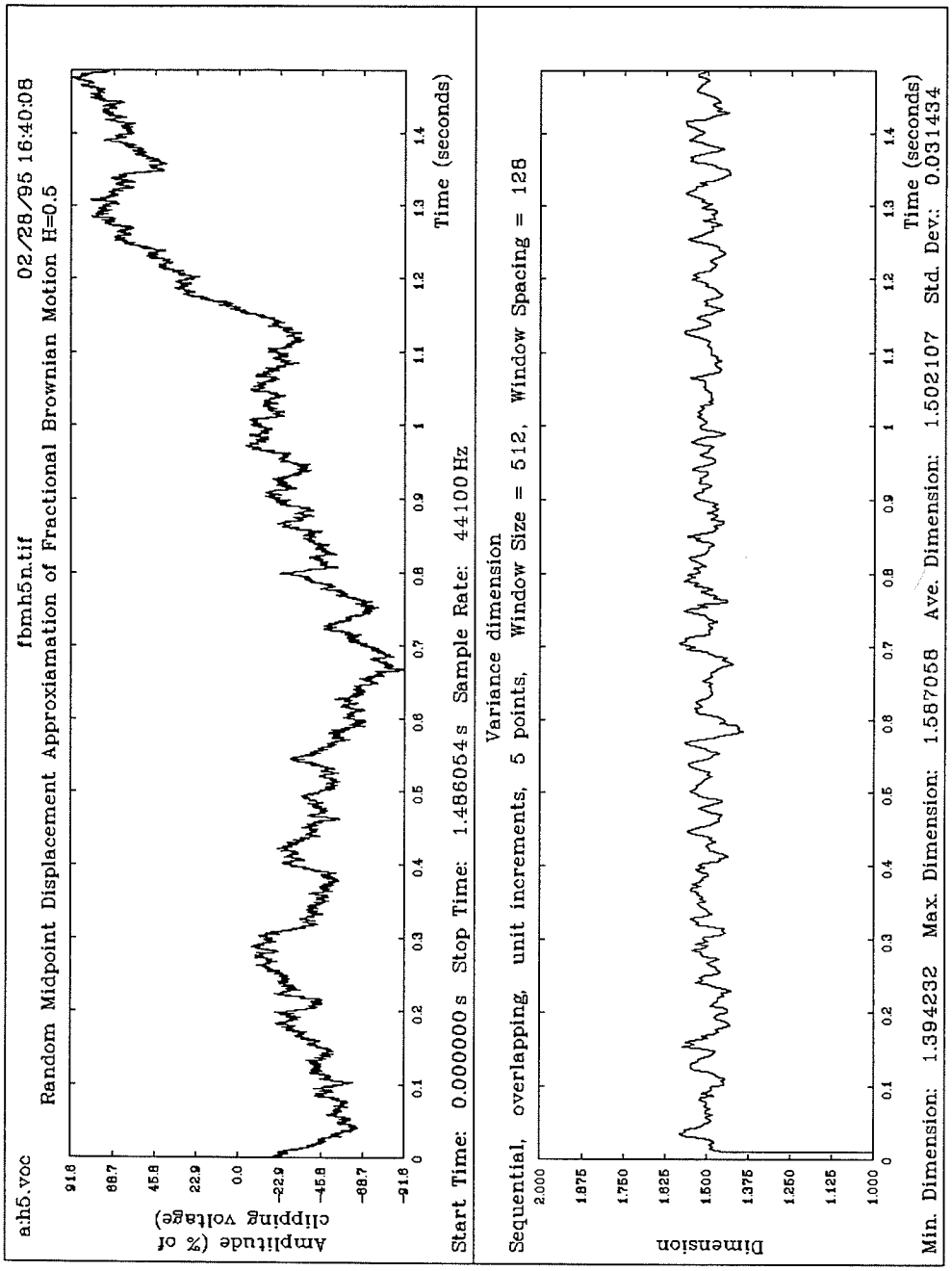
Fractal Brownian Motion, H = 0.3, Noise Separation Parameters



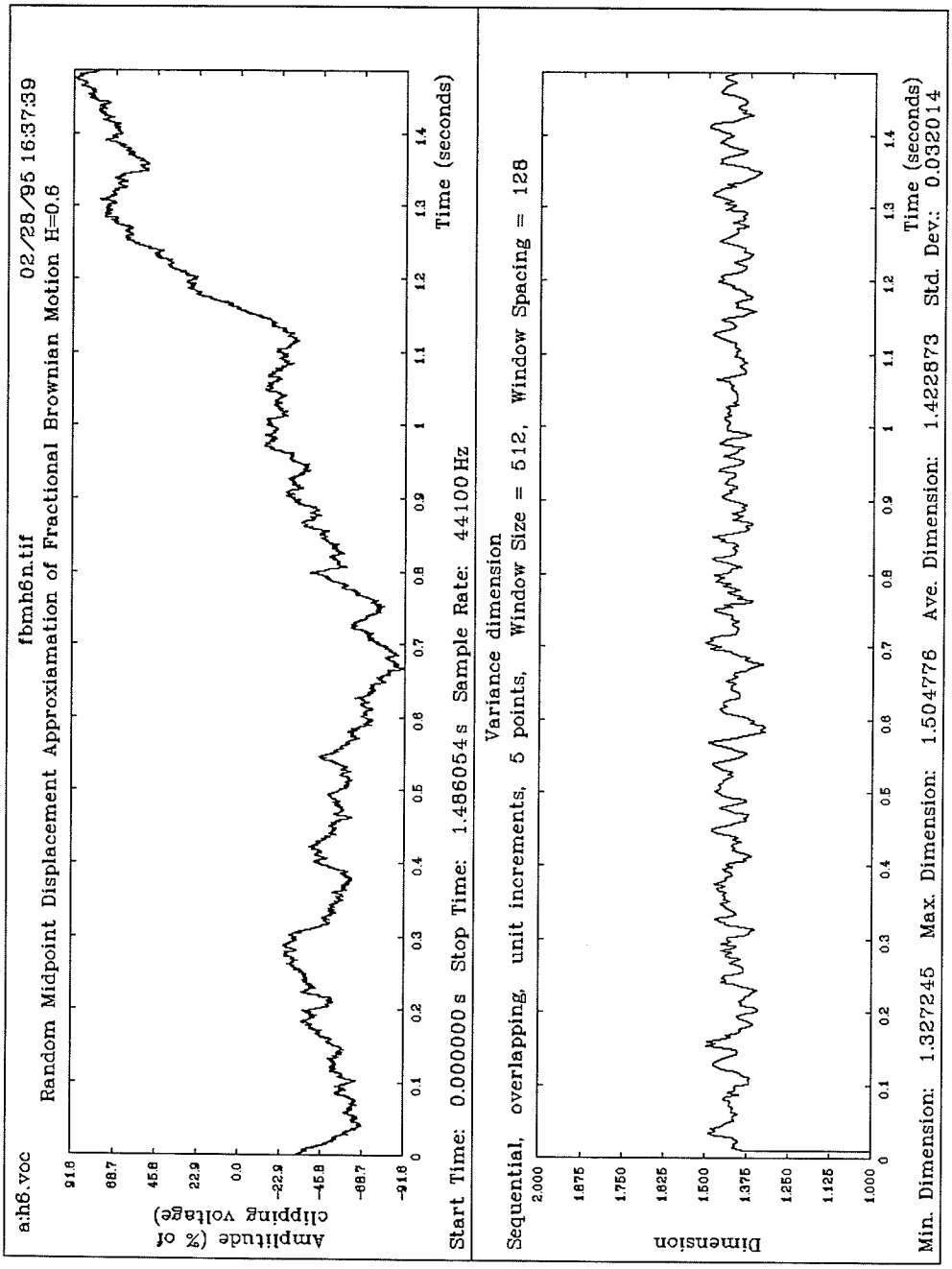
Fractal Brownian Motion, H = 0.4, Noise Separation Parameters



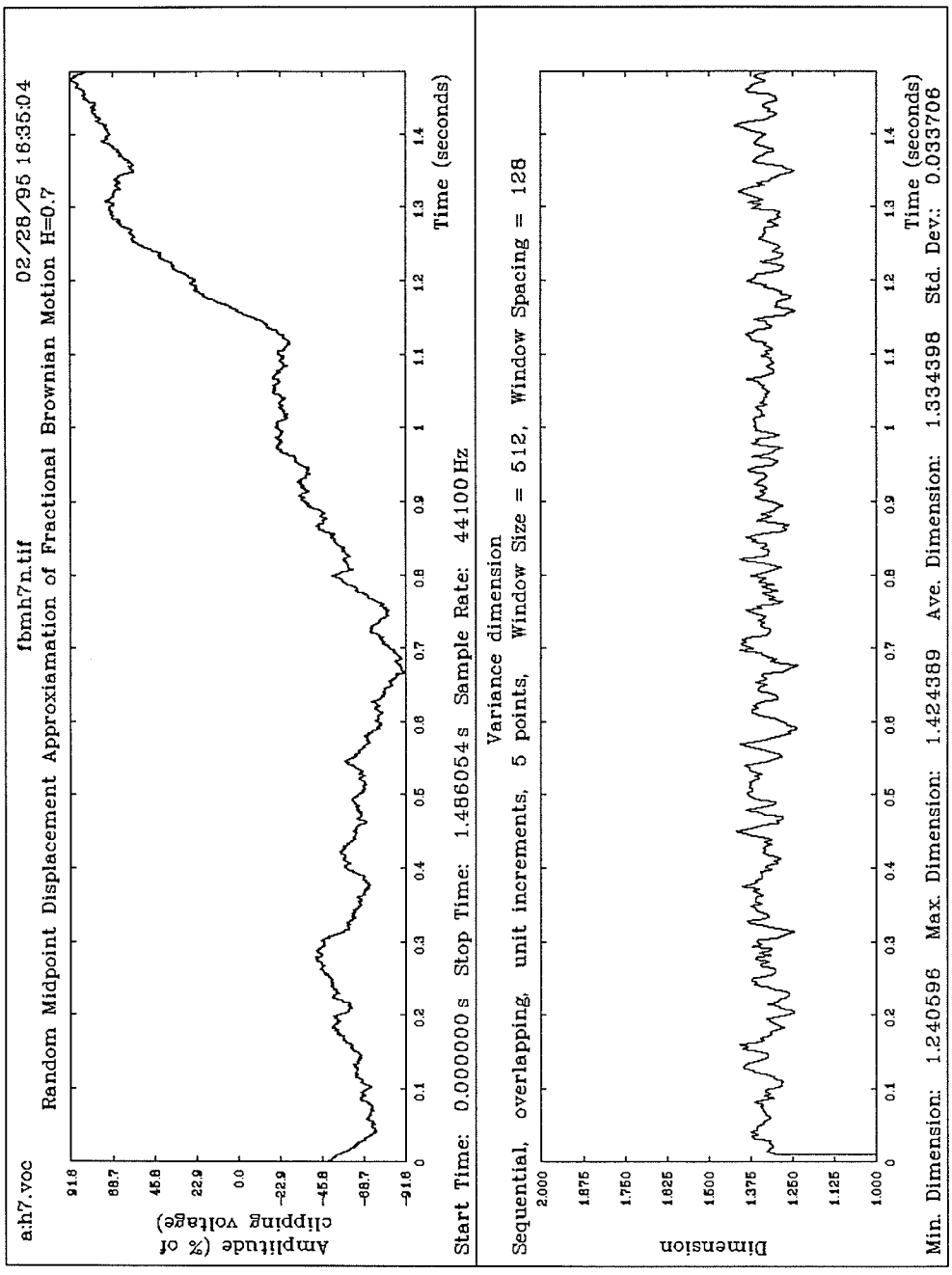
Fractal Brownian Motion, H = 0.5, Noise Separation Parameters



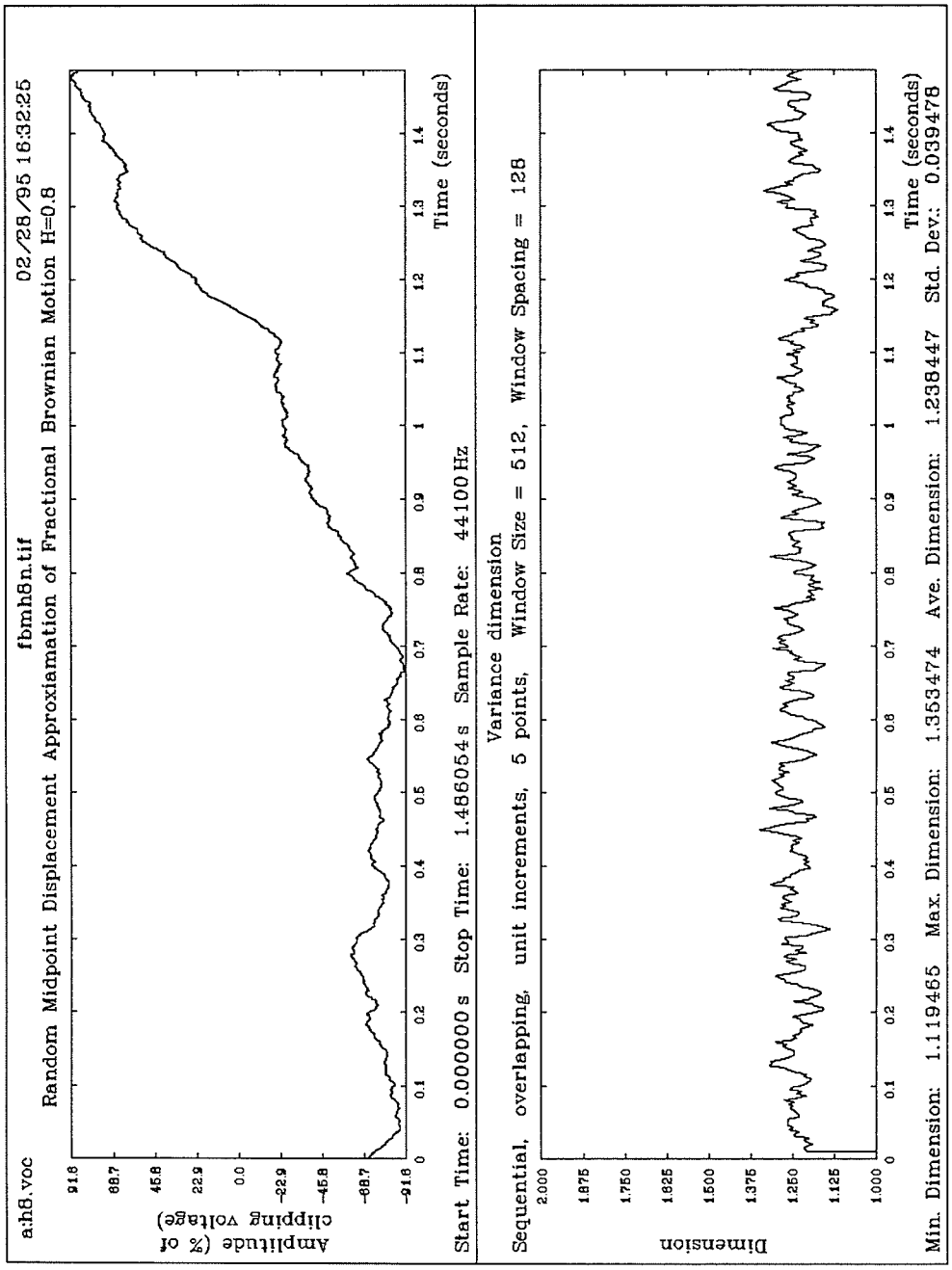
Fractal Brownian Motion, H = 0.6, Noise Separation Parameters



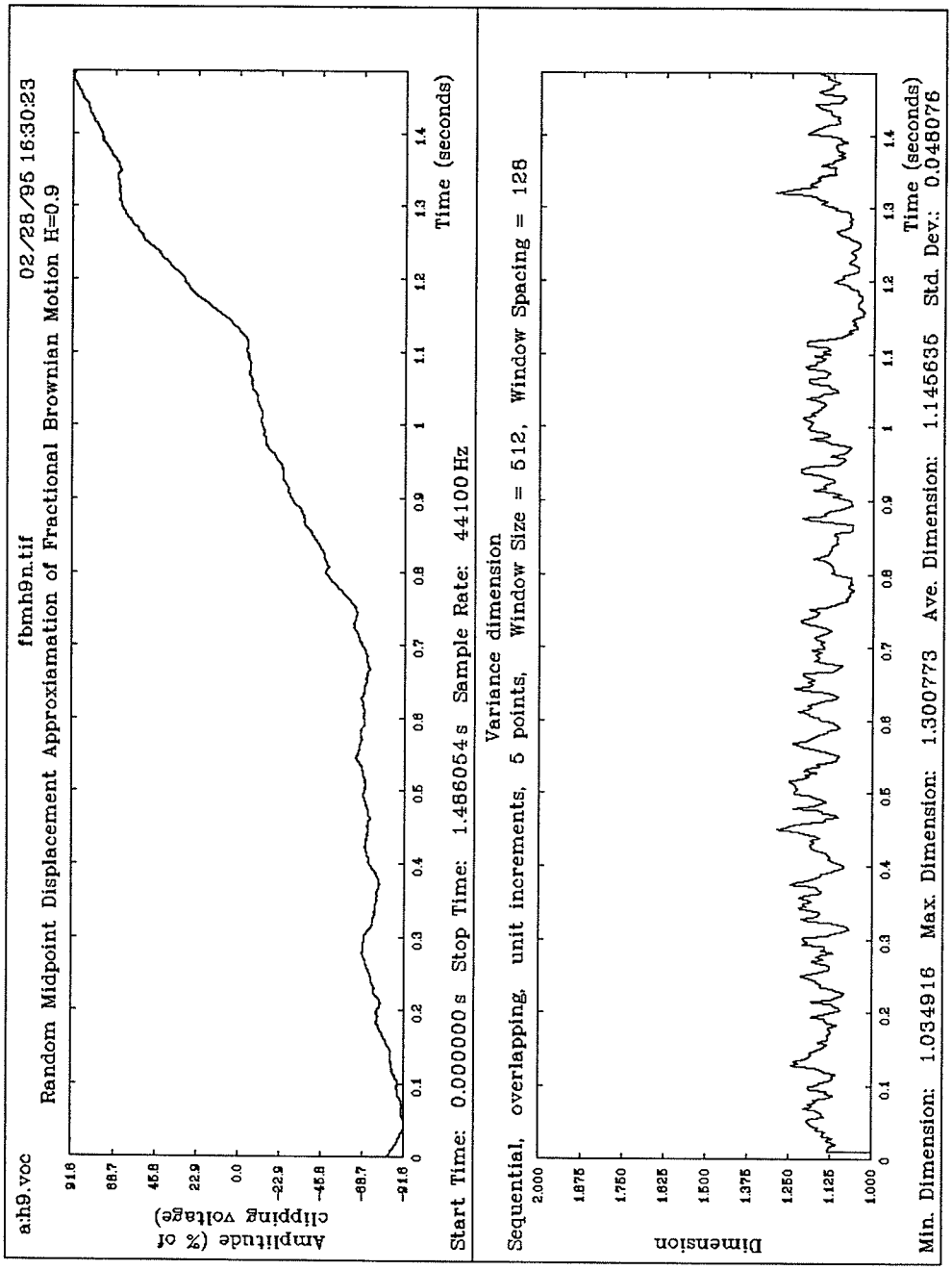
Fractal Brownian Motion, H = 0.7, Noise Separation Parameters



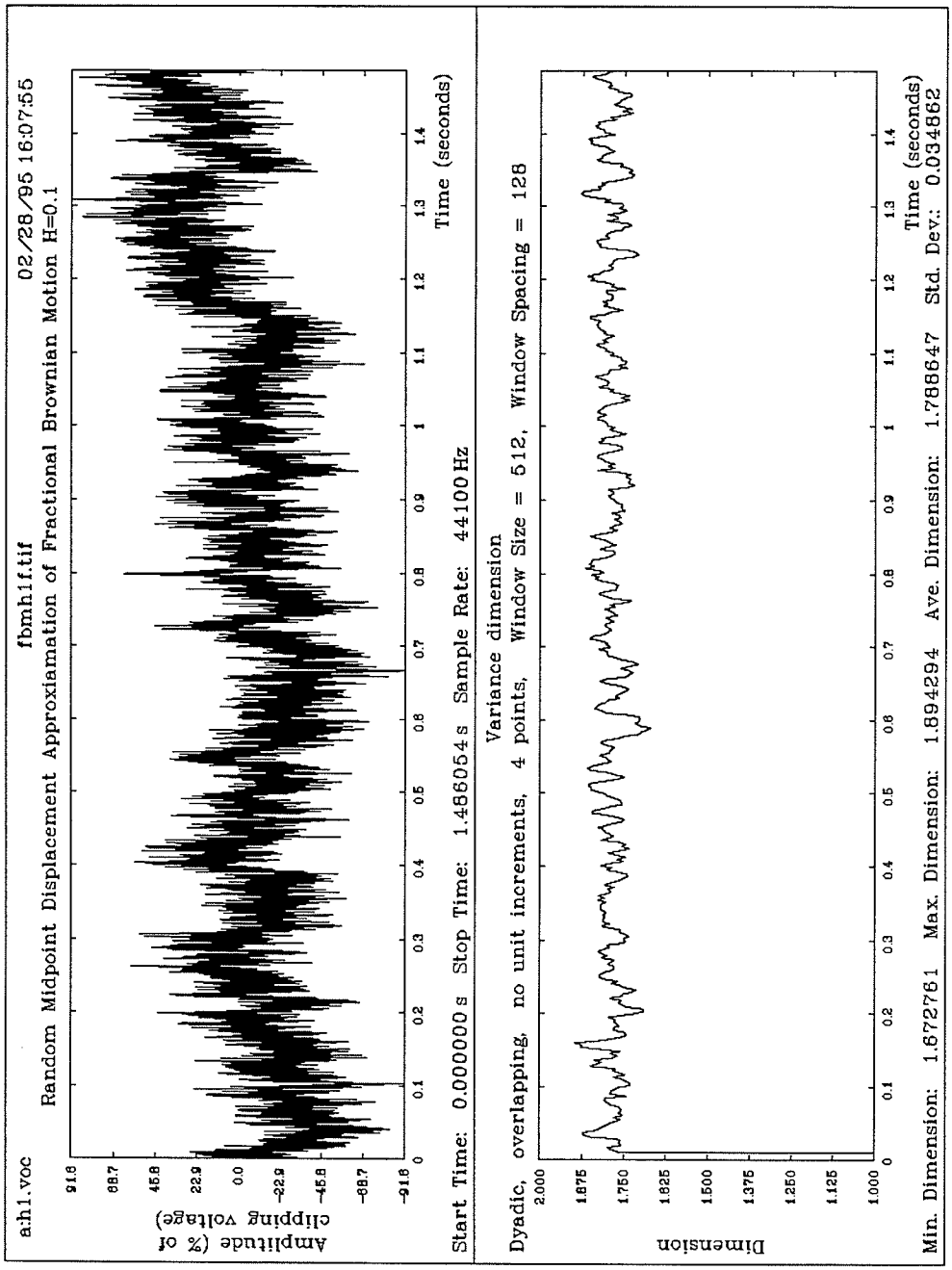
Fractal Brownian Motion, H = 0.8, Noise Separation Parameters



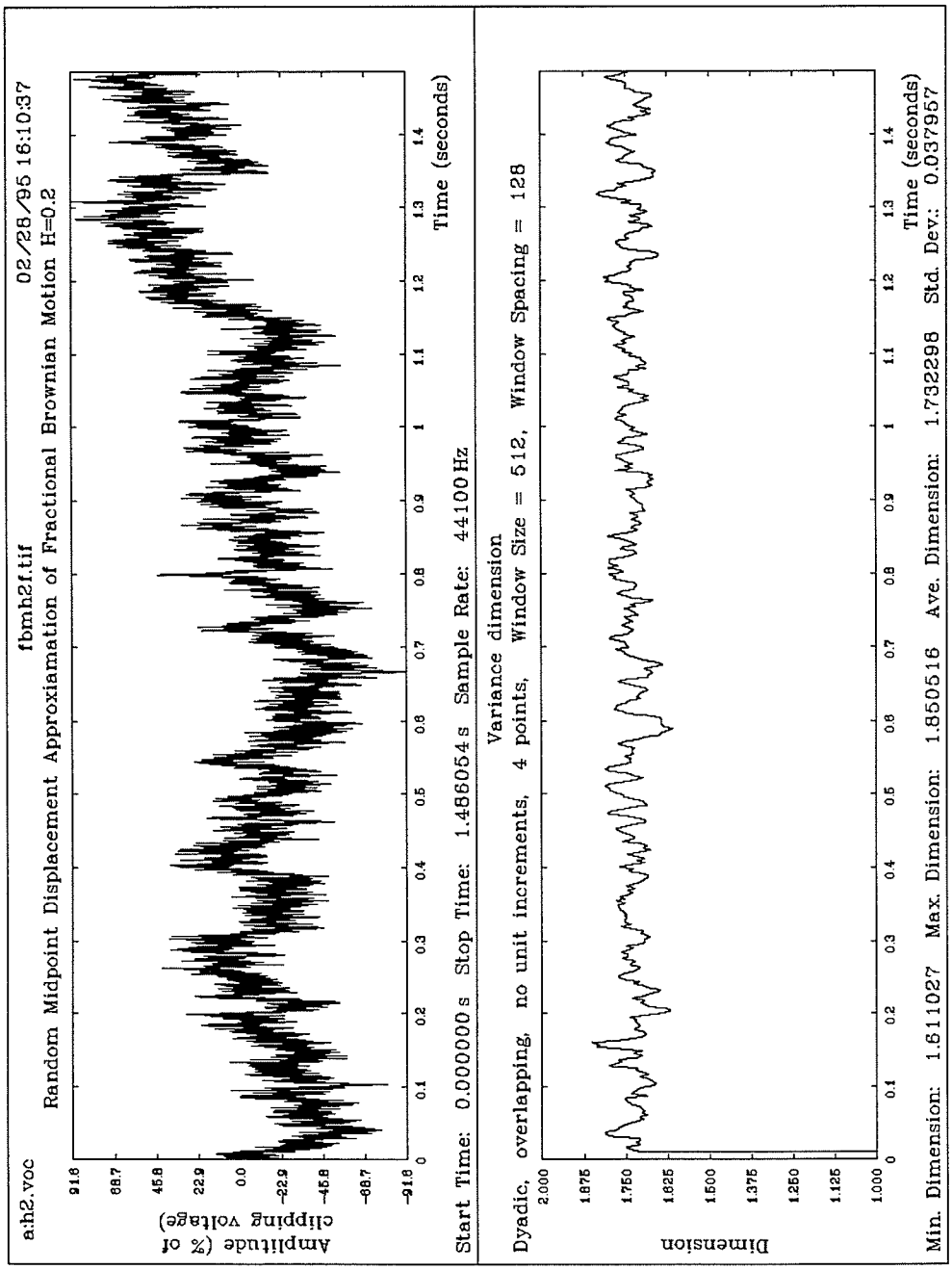
Fractal Brownian Motion, H = 0.9, Noise Separation Parameters



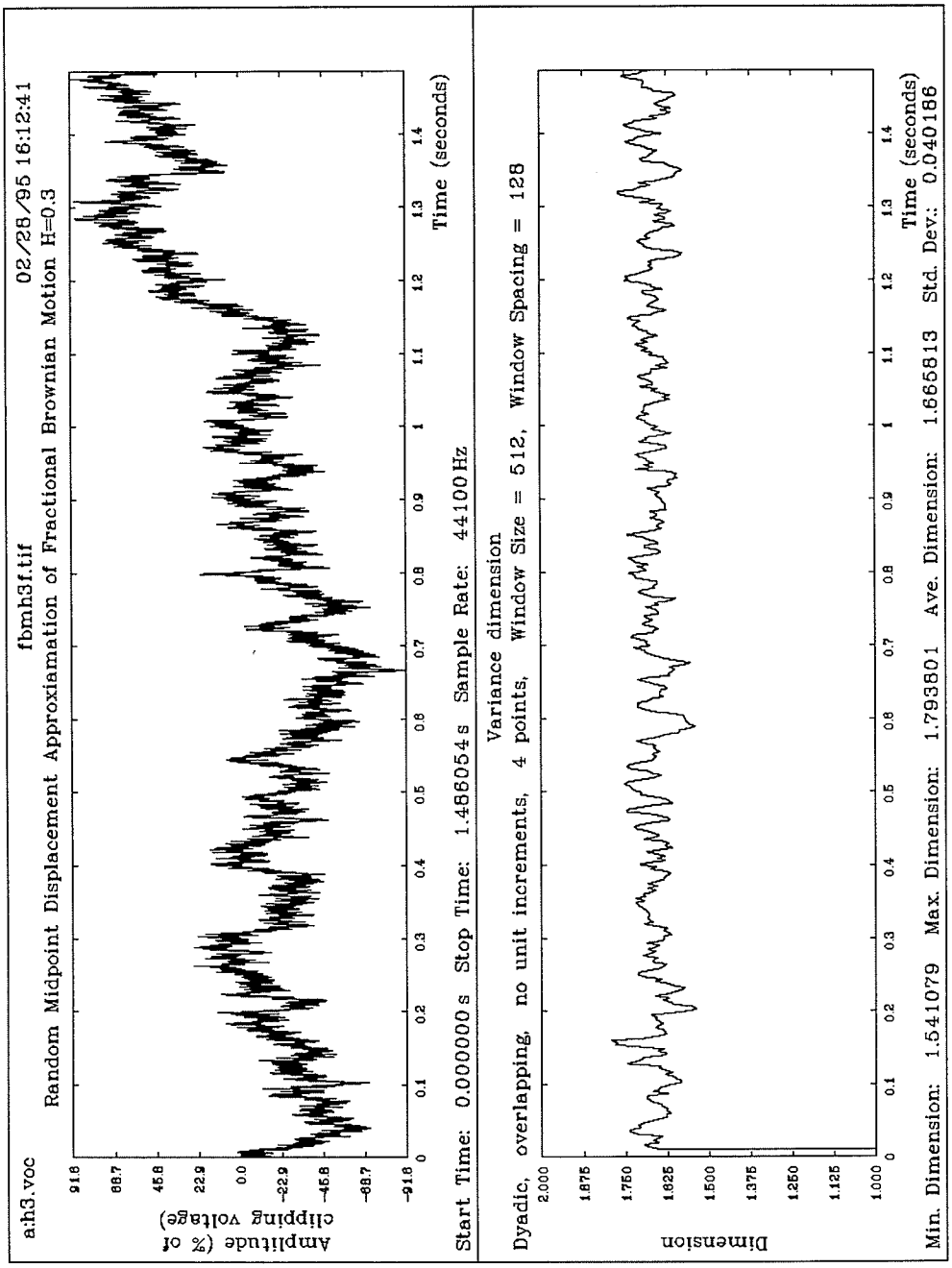
Fractal Brownian Motion, H = 0.1, Fractal Amplification Parameters



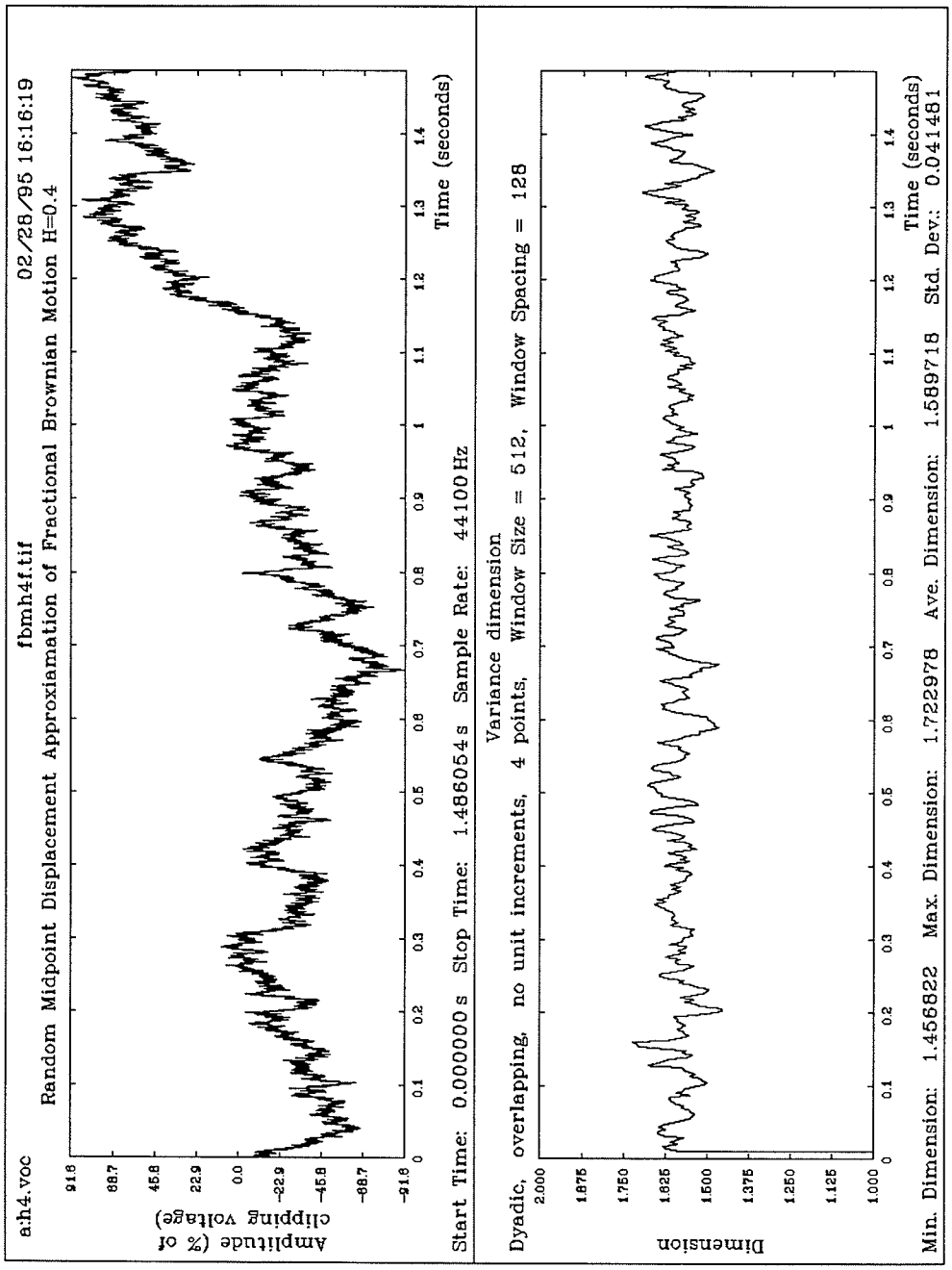
Fractal Brownian Motion, H = 0.2, Fractal Amplification Parameters



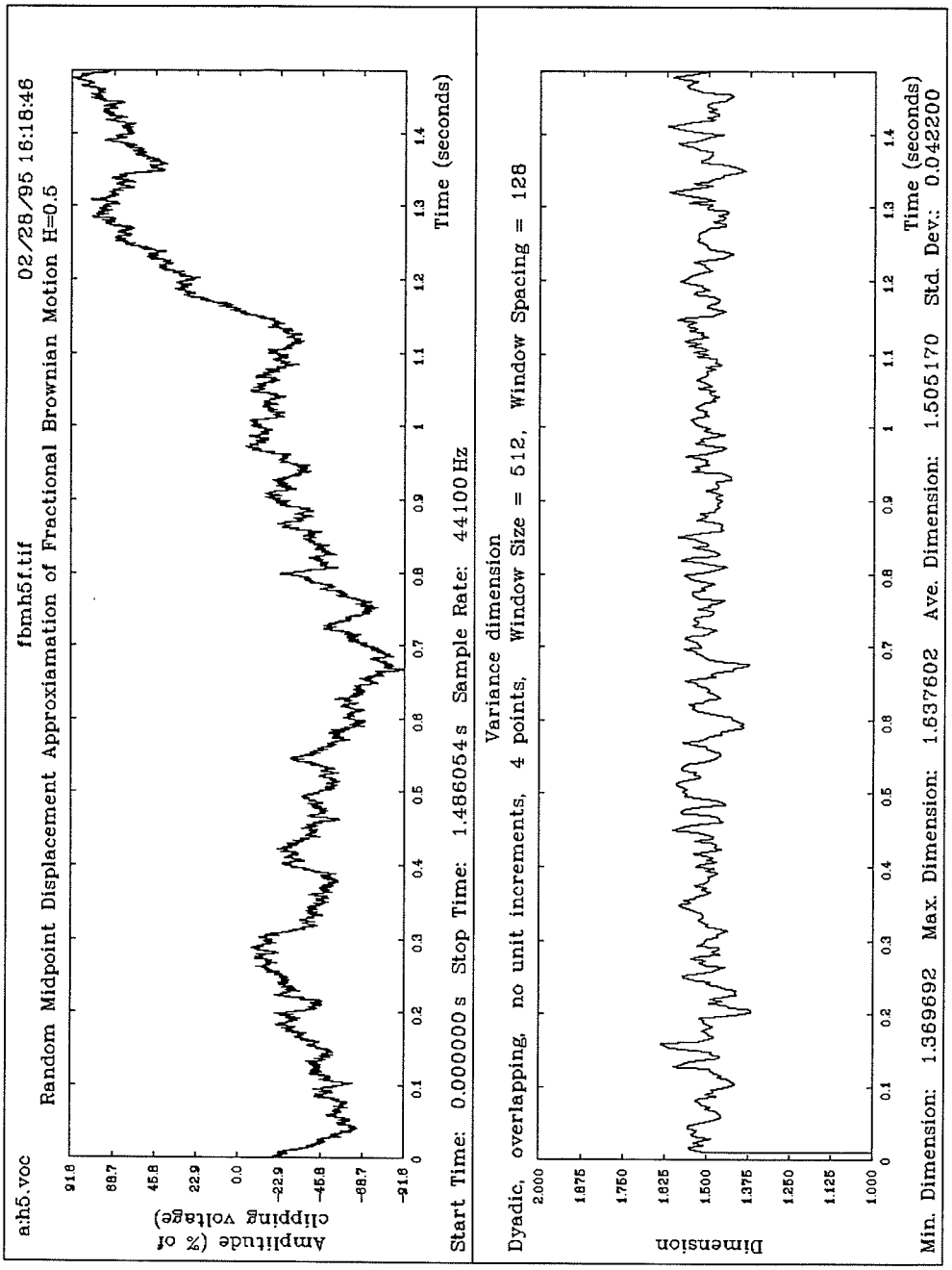
Fractal Brownian Motion, H = 0.3, Fractal Amplification Parameters



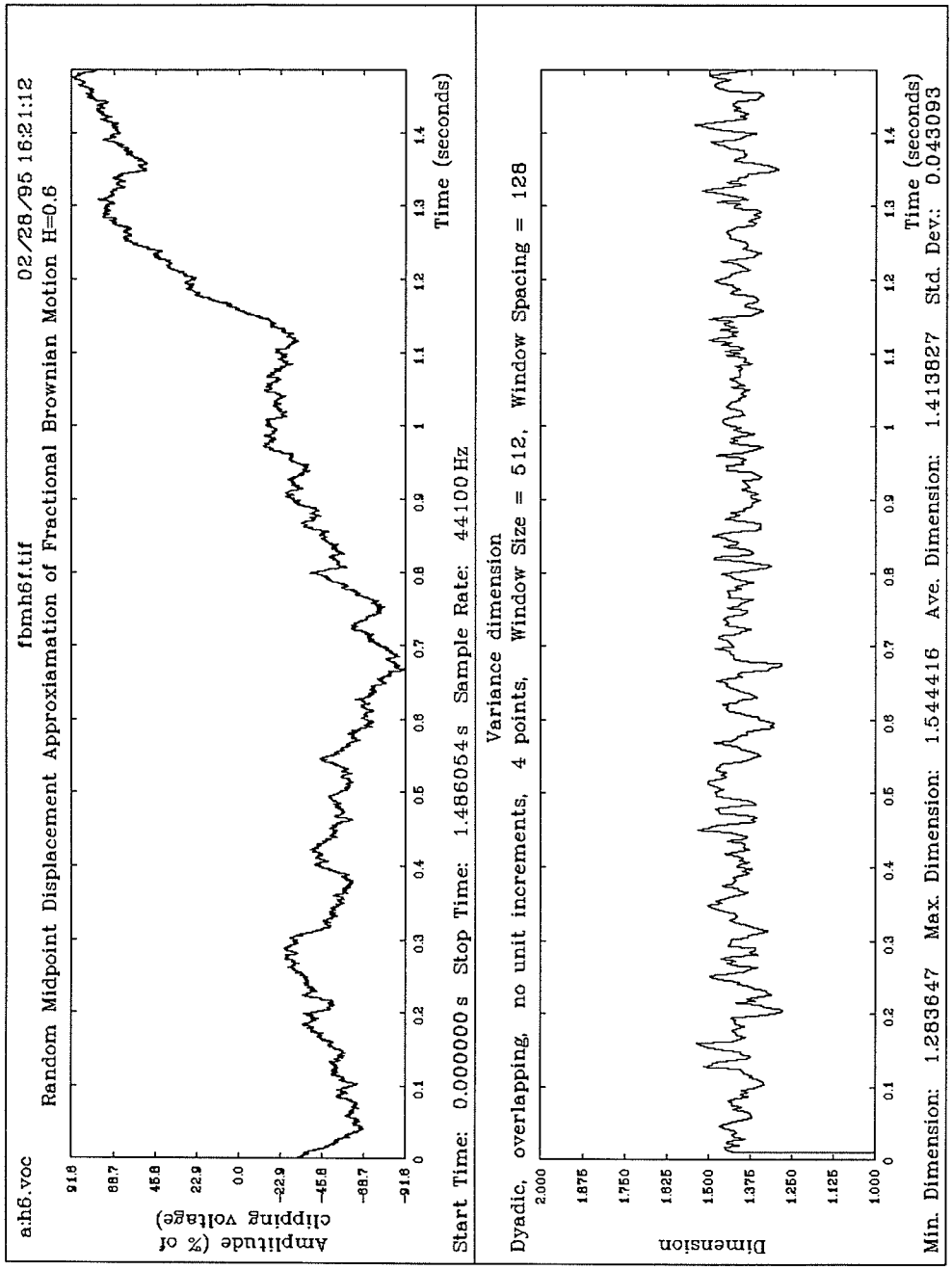
Fractal Brownian Motion, H = 0.4, Fractal Amplification Parameters



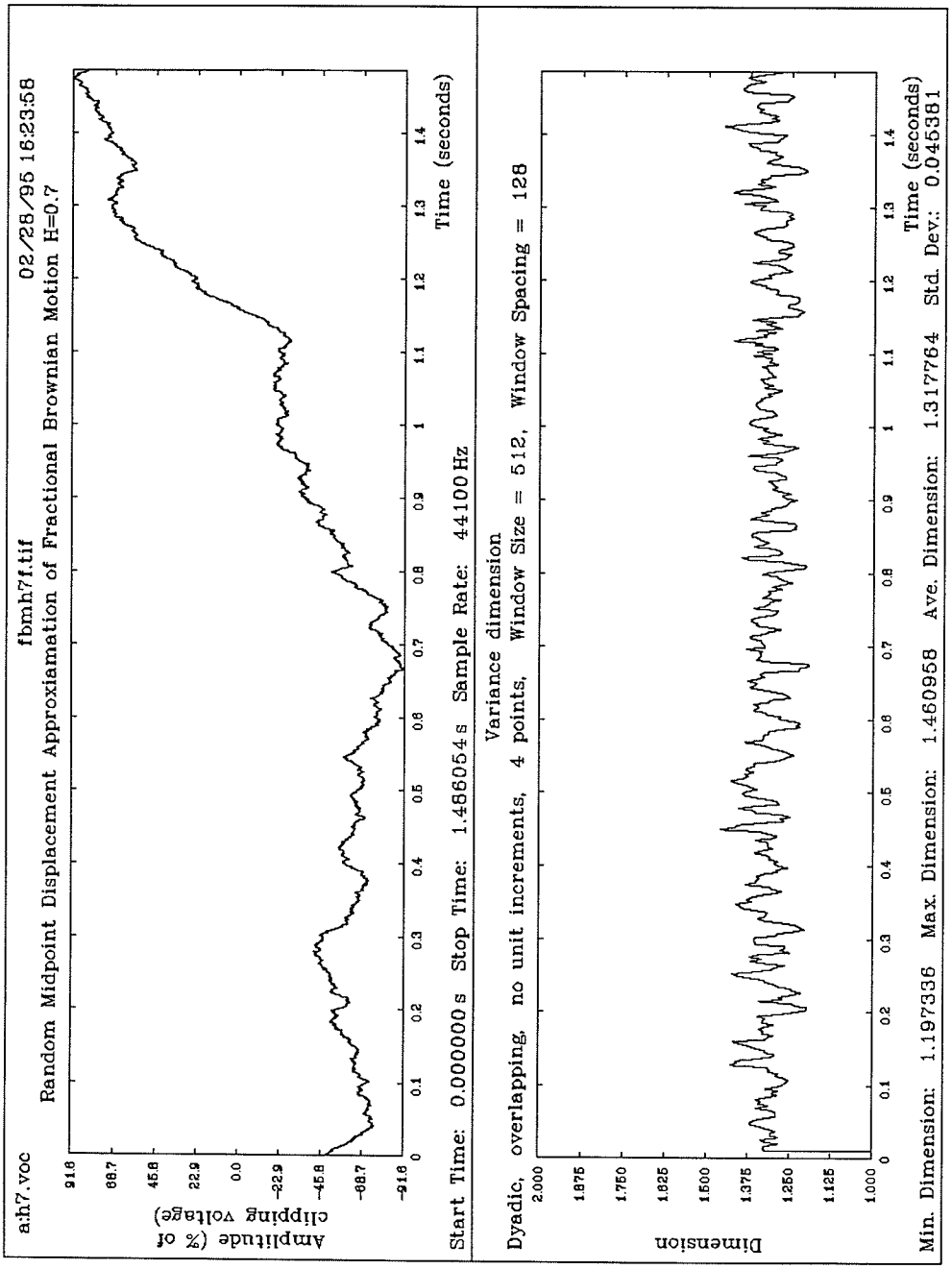
Fractal Brownian Motion, H = 0.5, Fractal Amplification Parameters



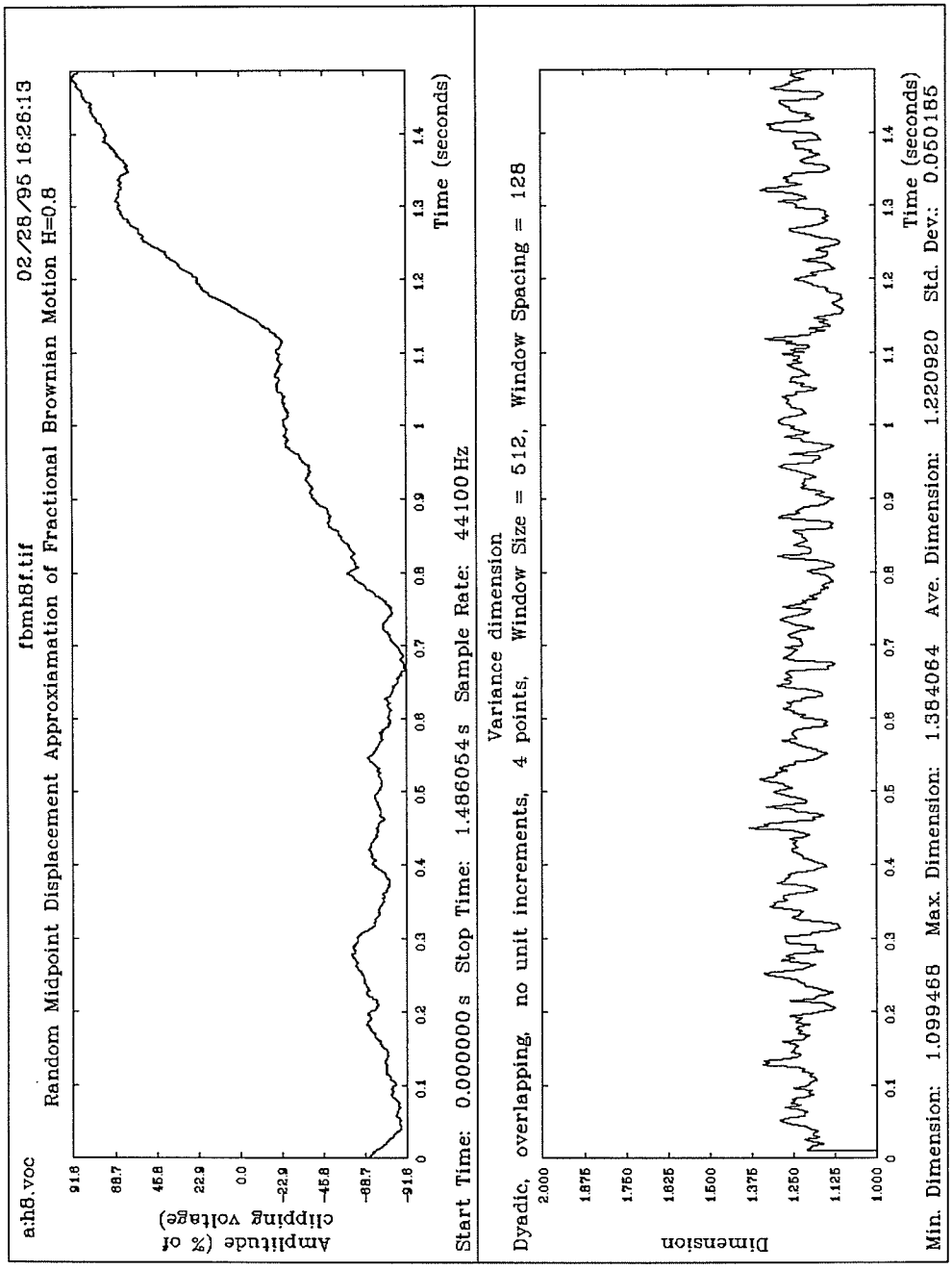
Fractal Brownian Motion, H = 0.6, Fractal Amplification Parameters



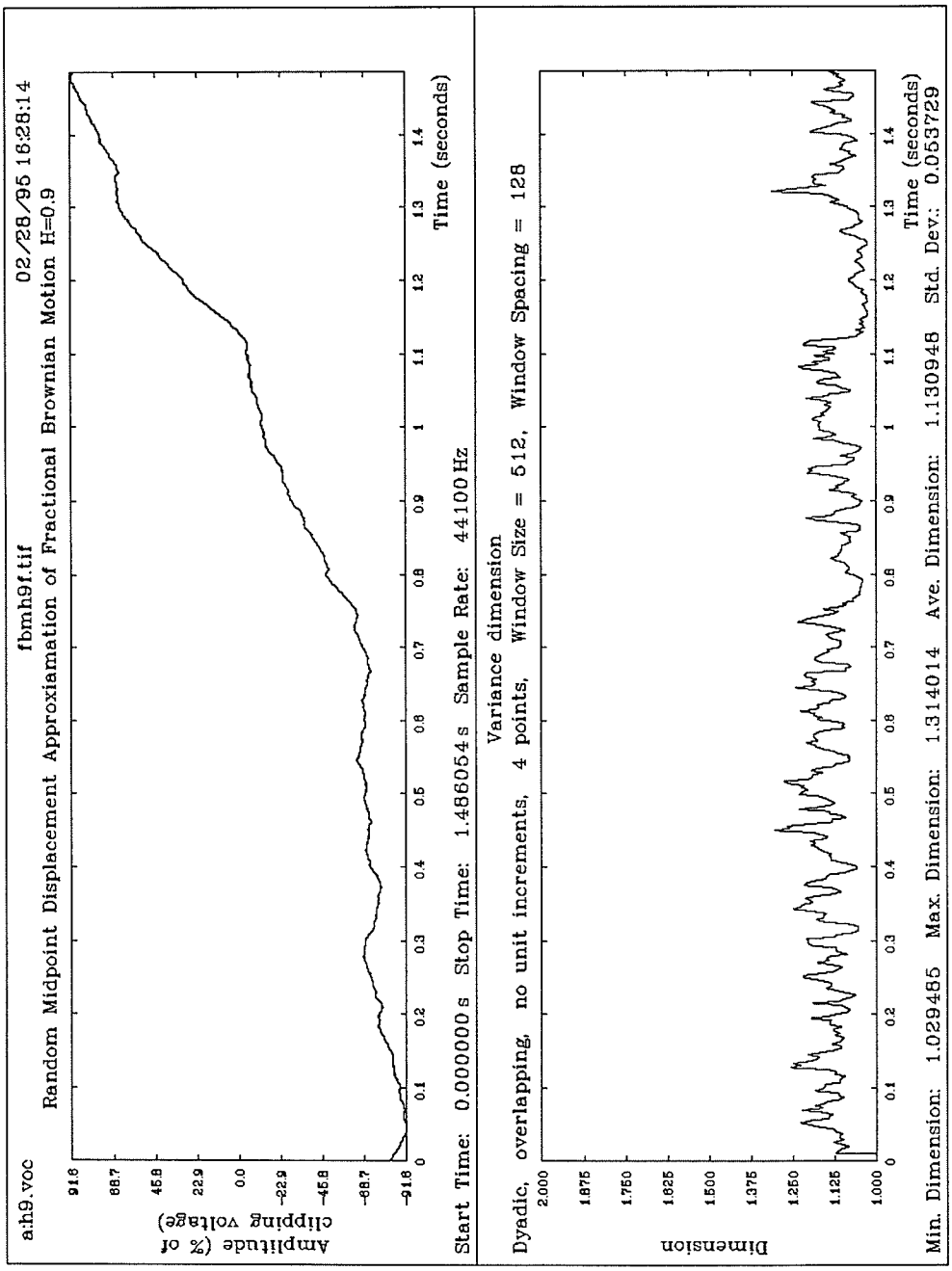
Fractal Brownian Motion, H = 0.7, Fractal Amplification Parameters



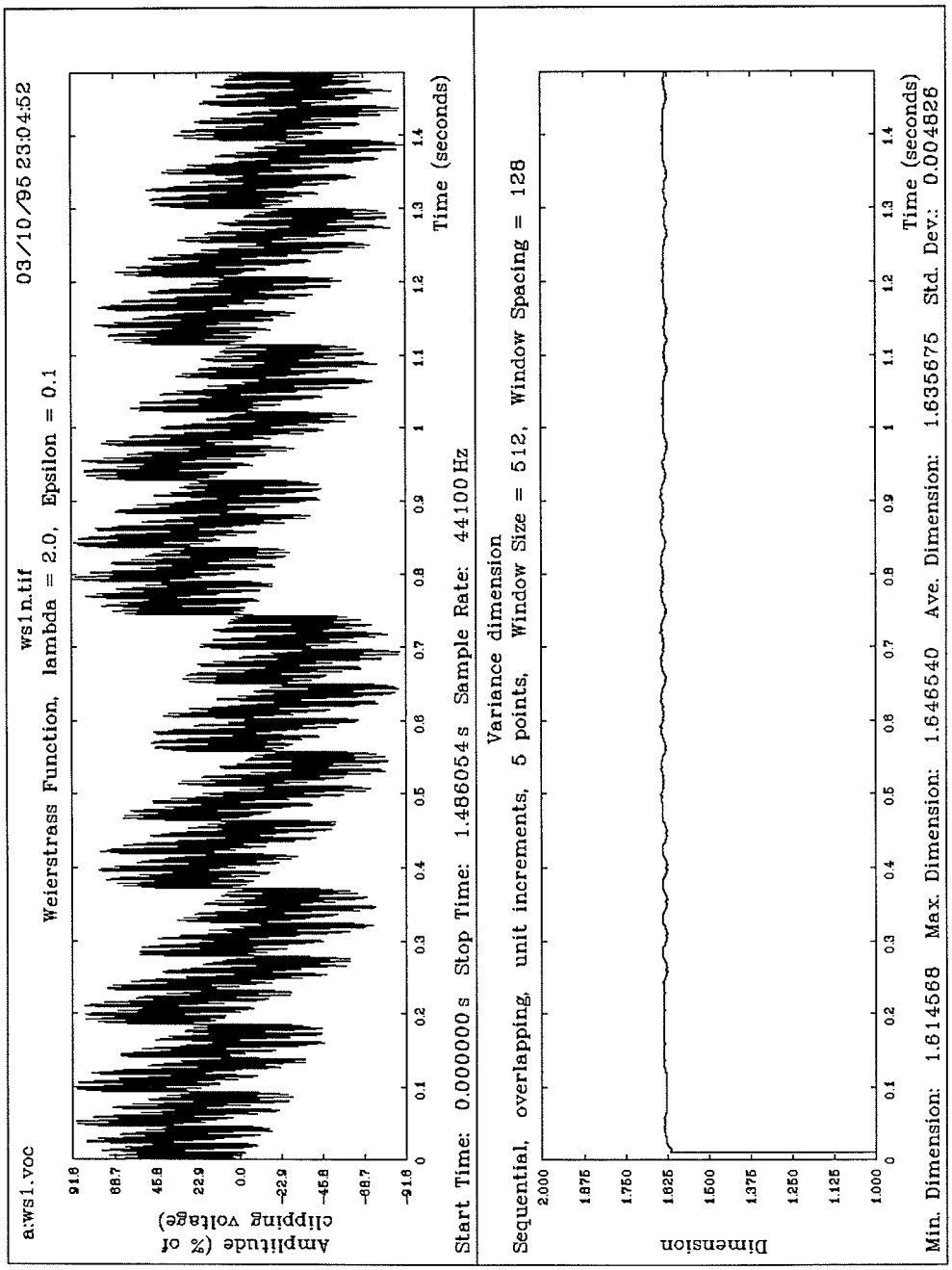
Fractal Brownian Motion, H = 0.8, Fractal Amplification Parameters



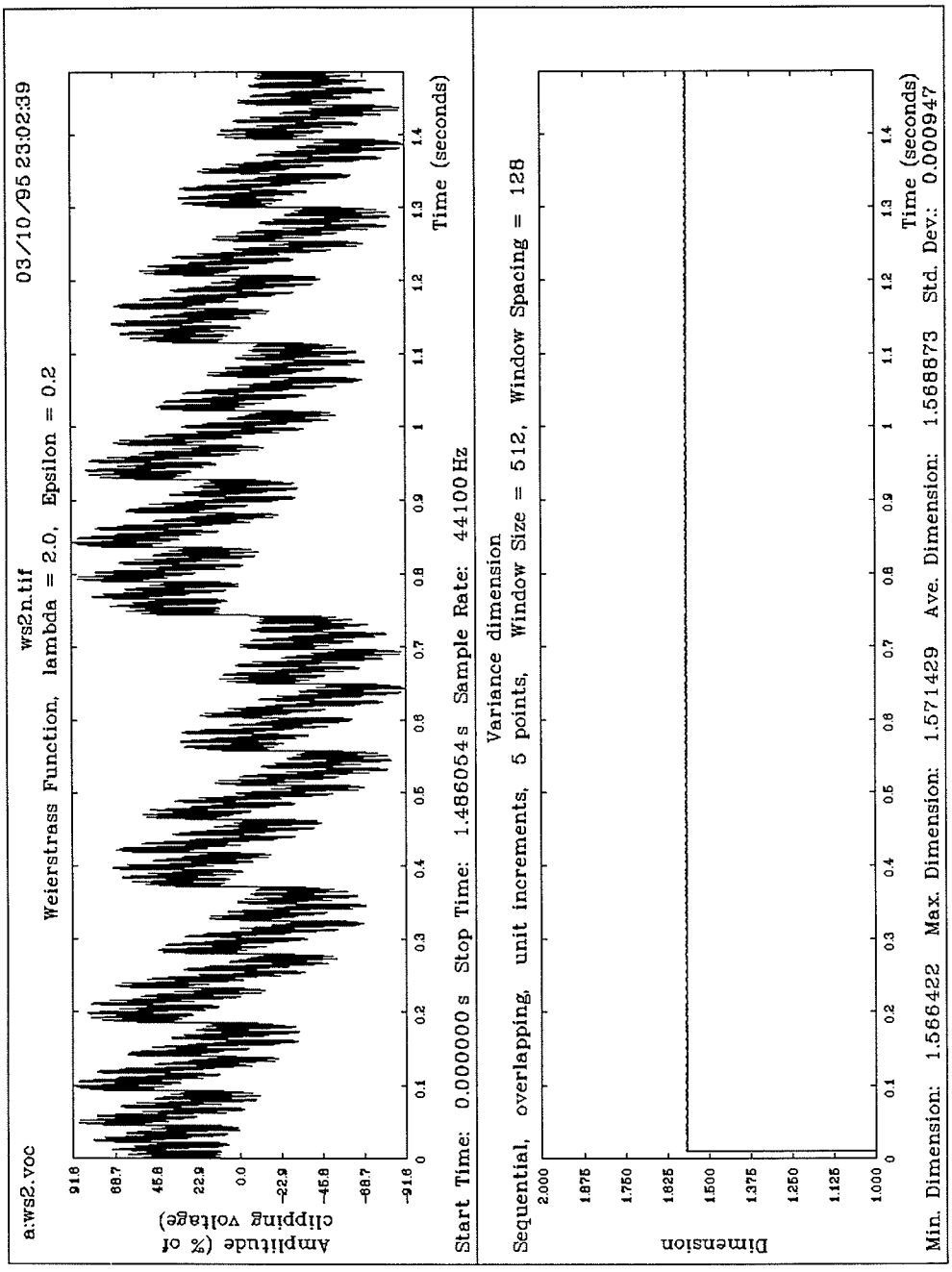
Fractal Brownian Motion, H = 0.9, Fractal Amplification Parameters



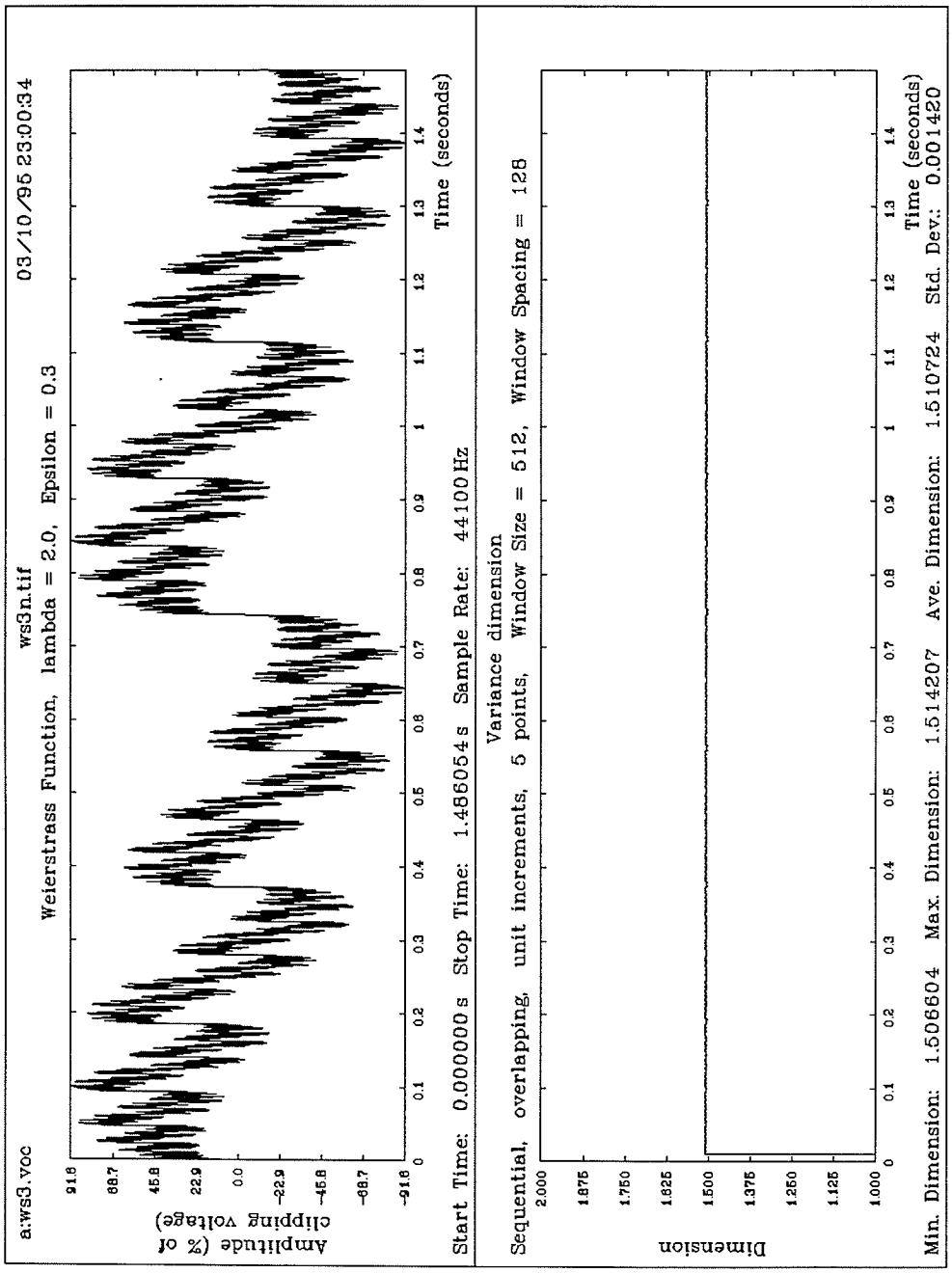
Weierstrass Function, $\varepsilon = 0.1$, Noise Separation Parameters



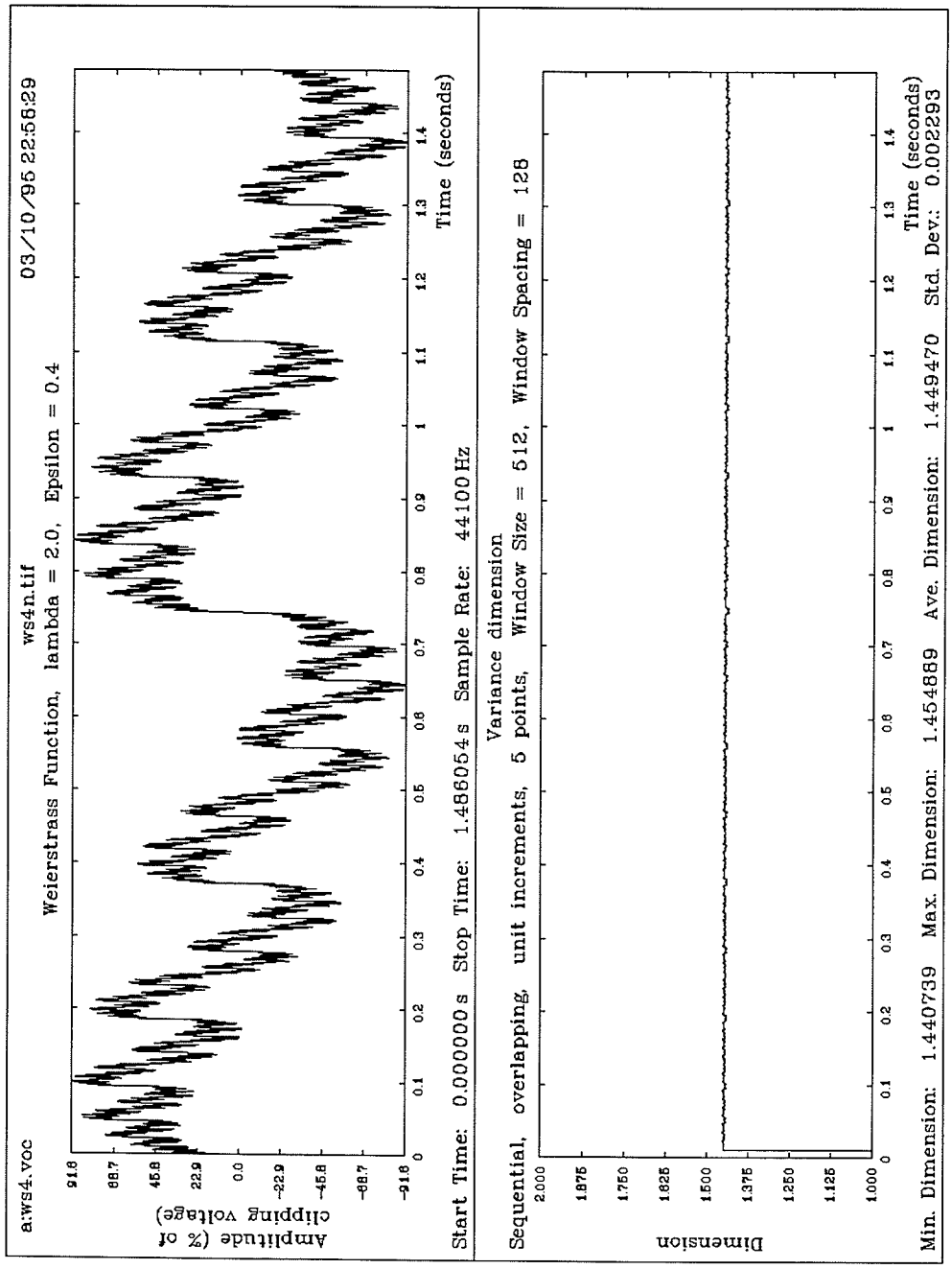
Weierstrass Function, $\varepsilon = 0.1$, Noise Separation Parameters



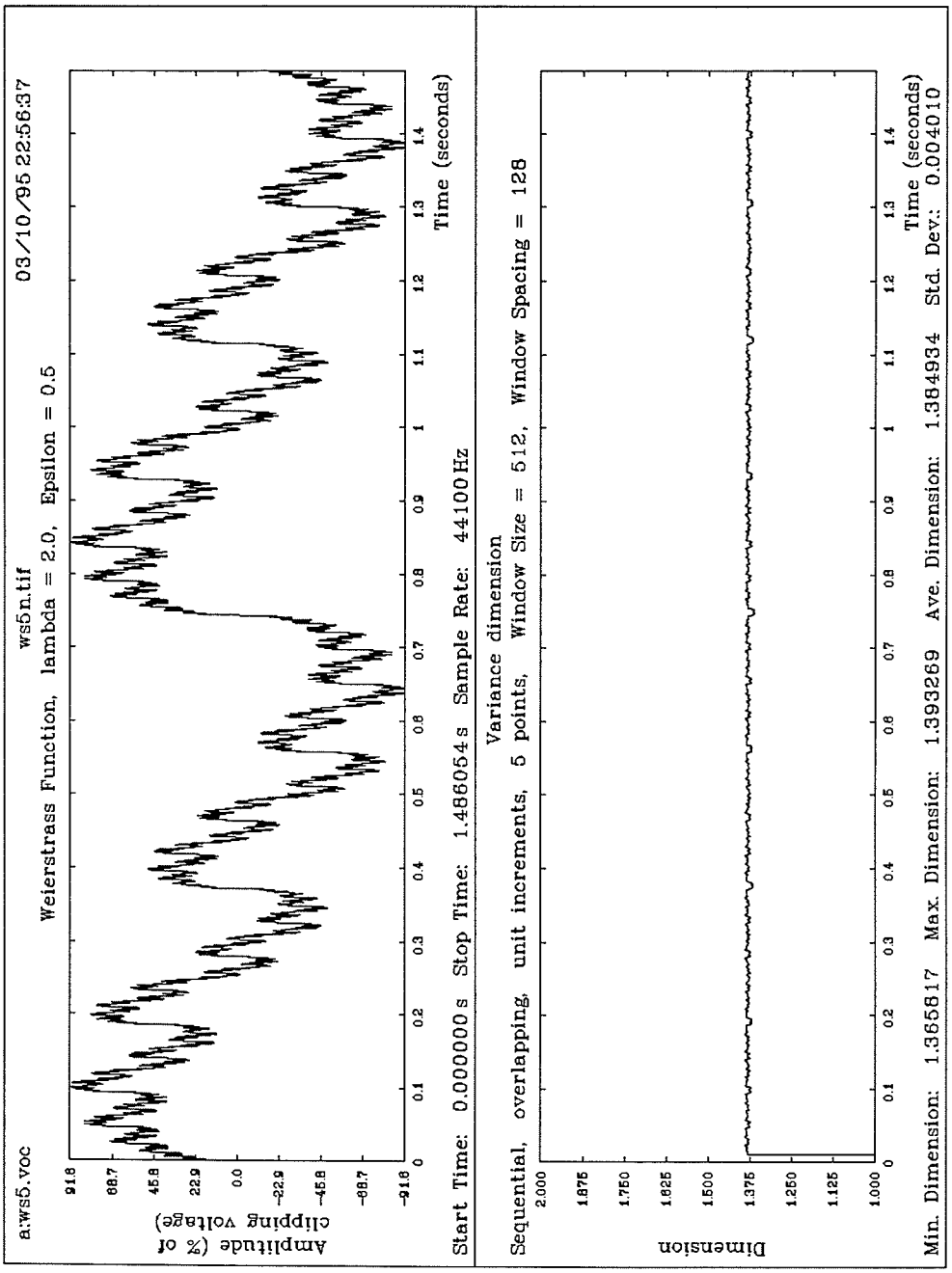
Weierstrass Function, $\varepsilon = 0.3$, Noise Separation Parameters



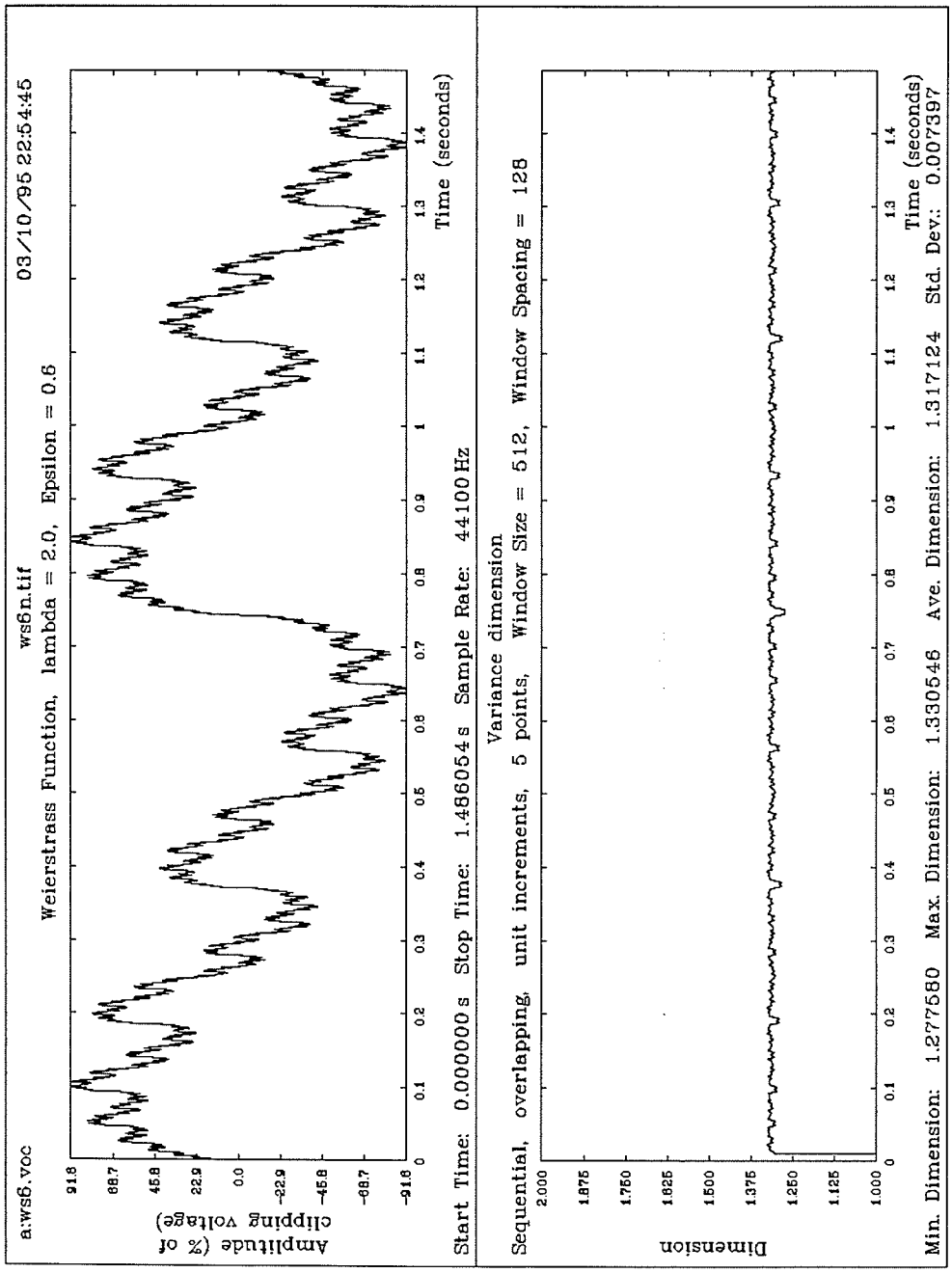
Weierstrass Function, $\varepsilon = 0.4$, Noise Separation Parameters



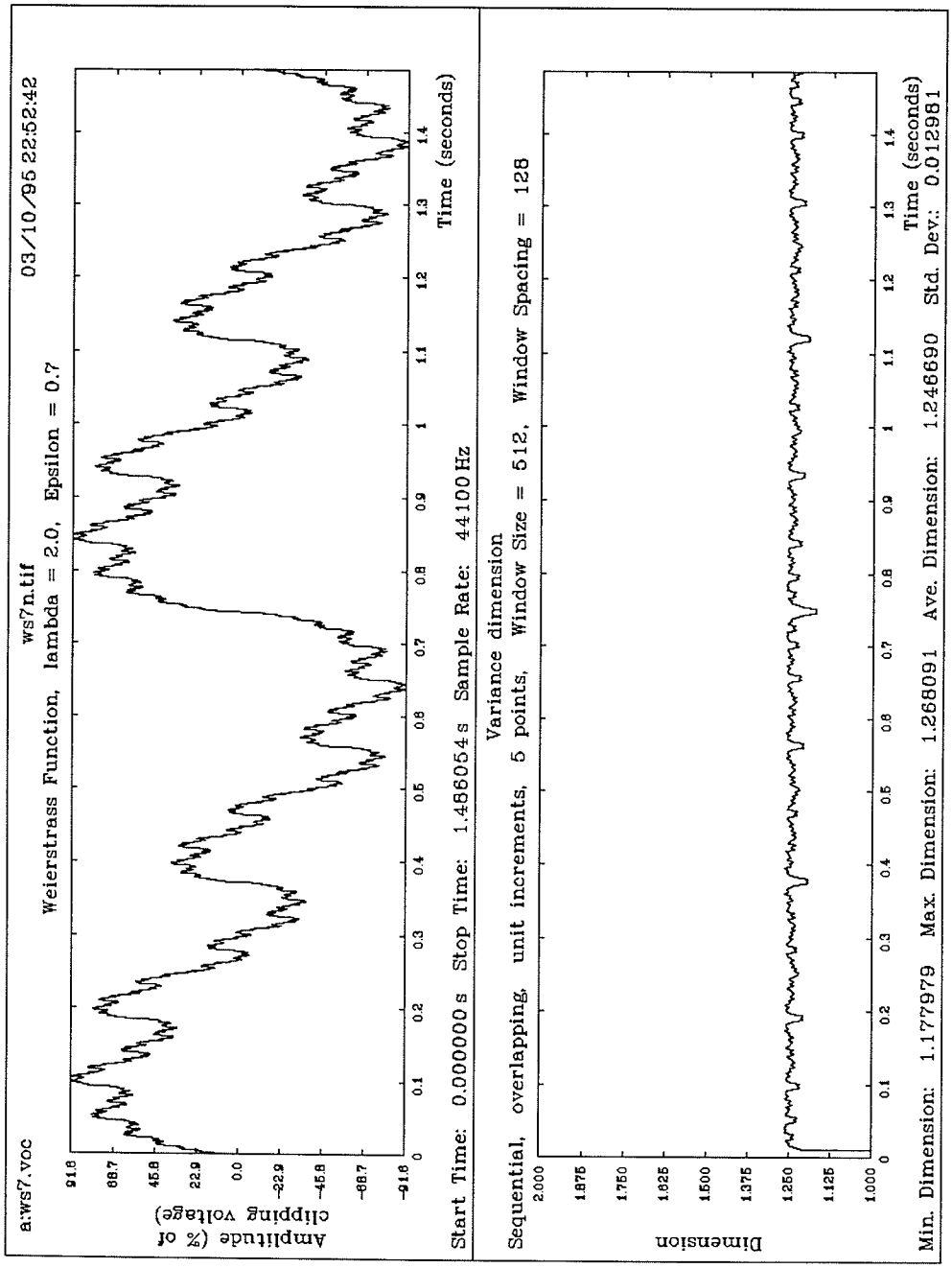
Weierstrass Function, $\varepsilon = 0.5$, Noise Separation Parameters



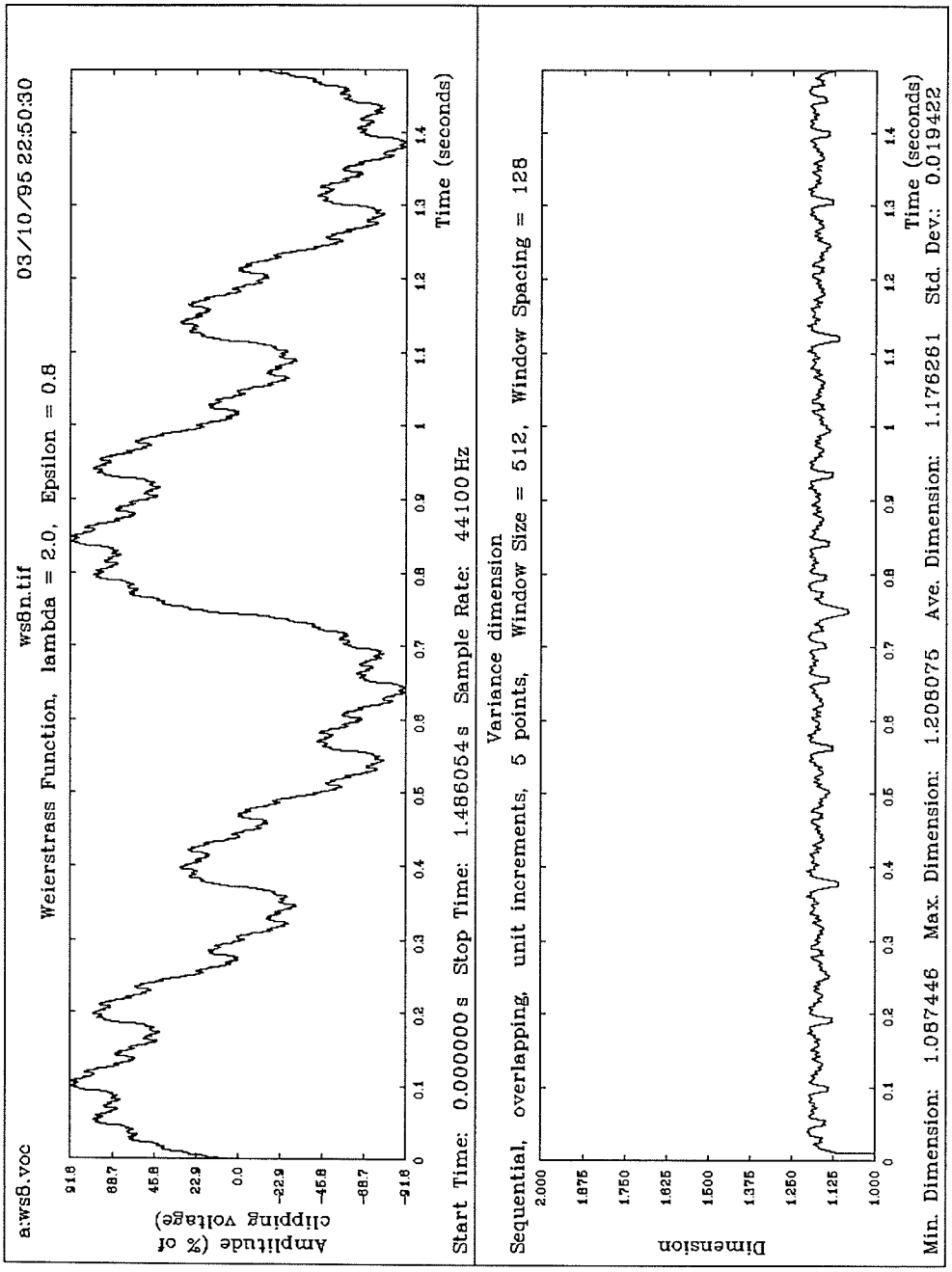
Weierstrass Function, $\varepsilon = 0.6$, Noise Separation Parameters



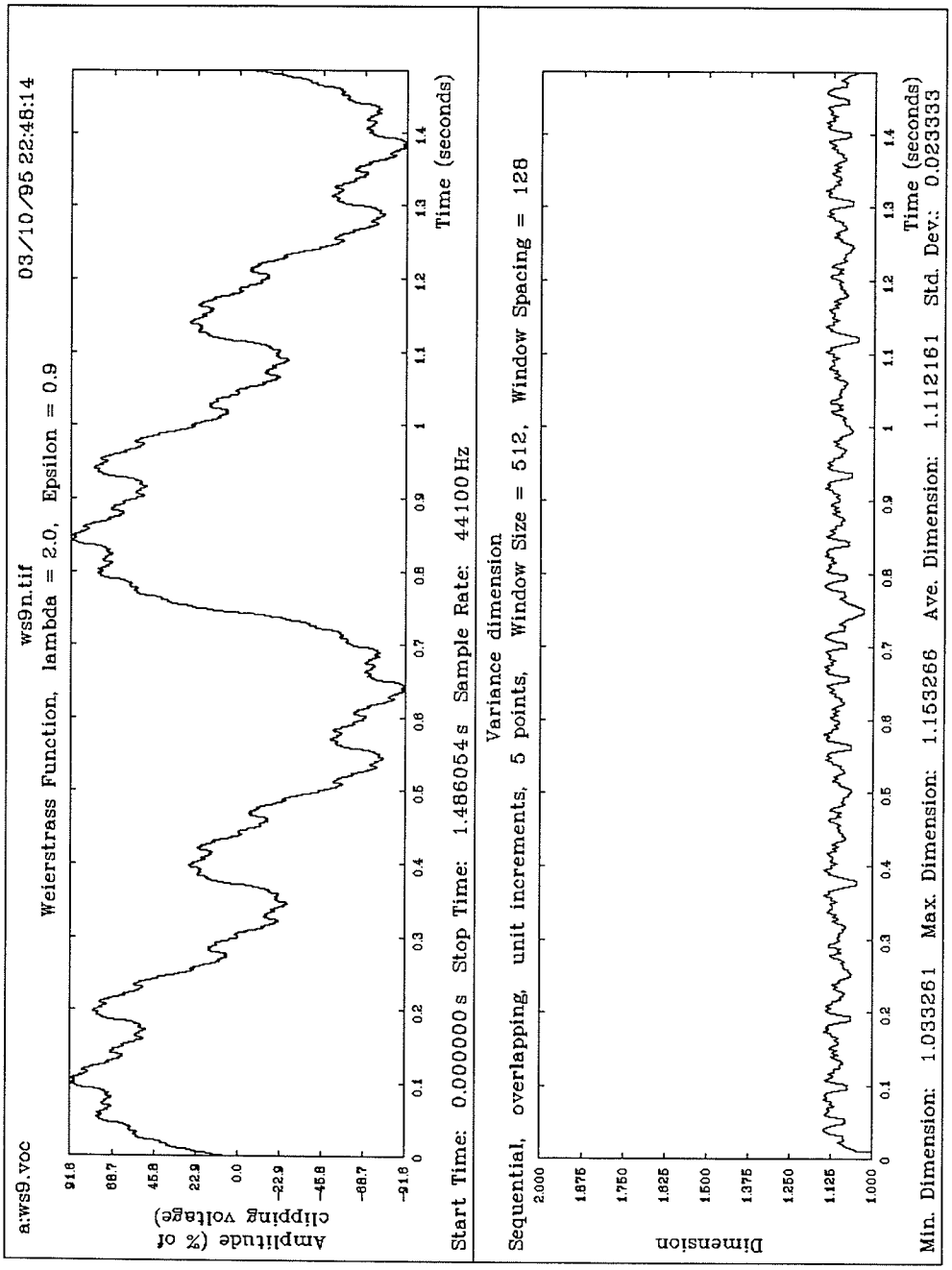
Weierstrass Function, $\varepsilon = 0.7$, Noise Separation Parameters



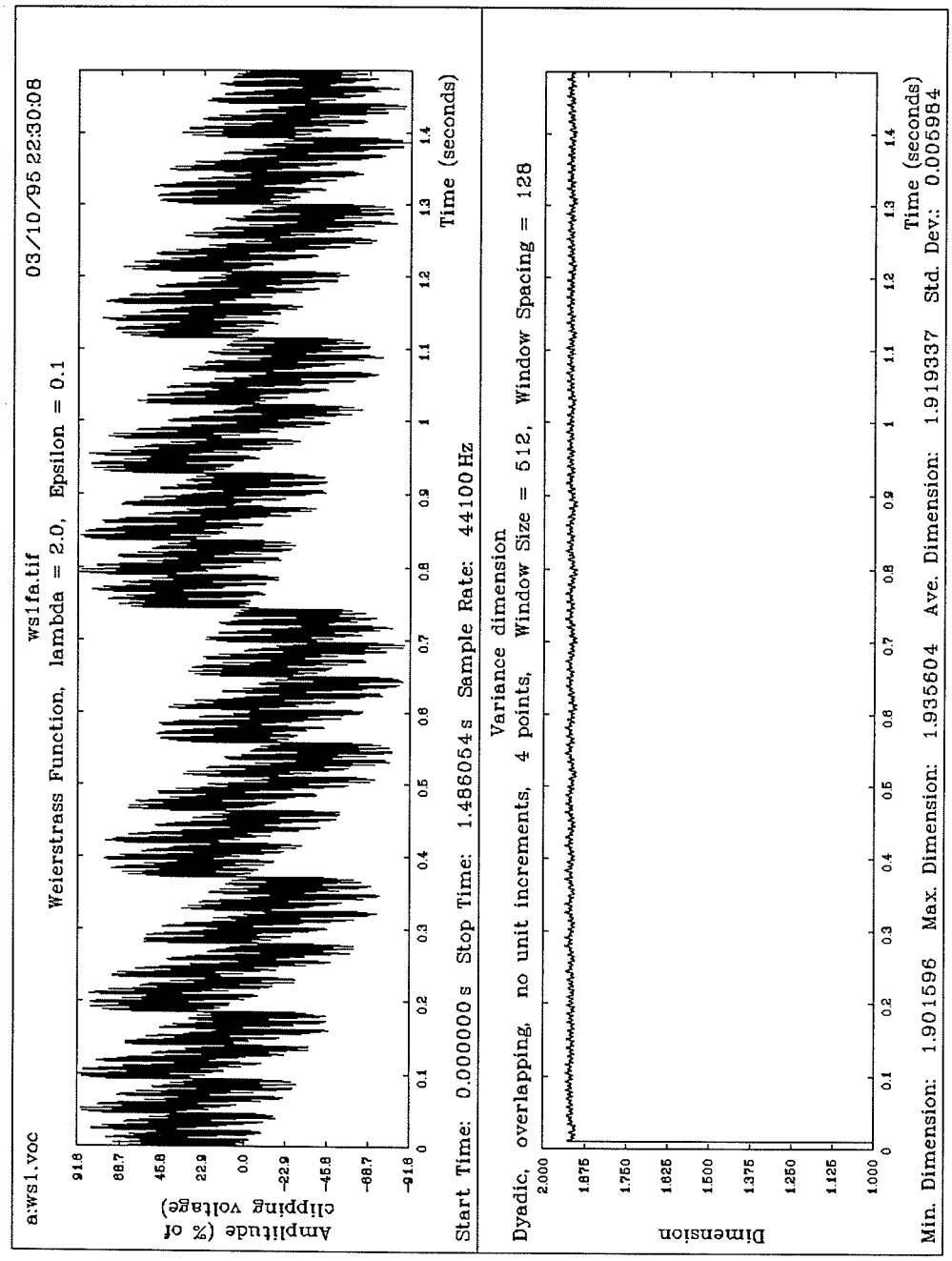
Weierstrass Function, $\varepsilon = 0.8$, Noise Separation Parameters



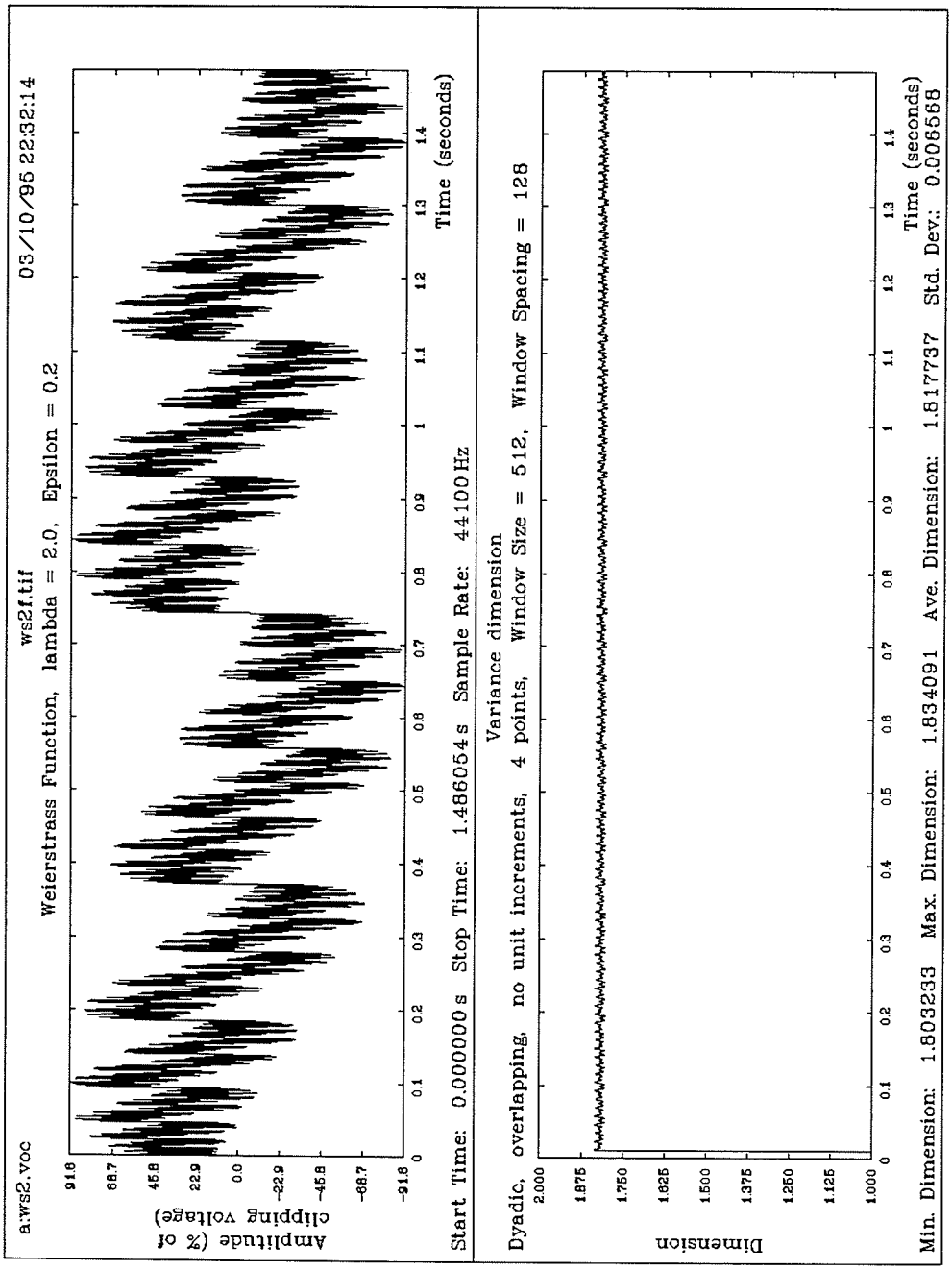
Weierstrass Function, $\varepsilon = 0.9$, Noise Separation Parameters



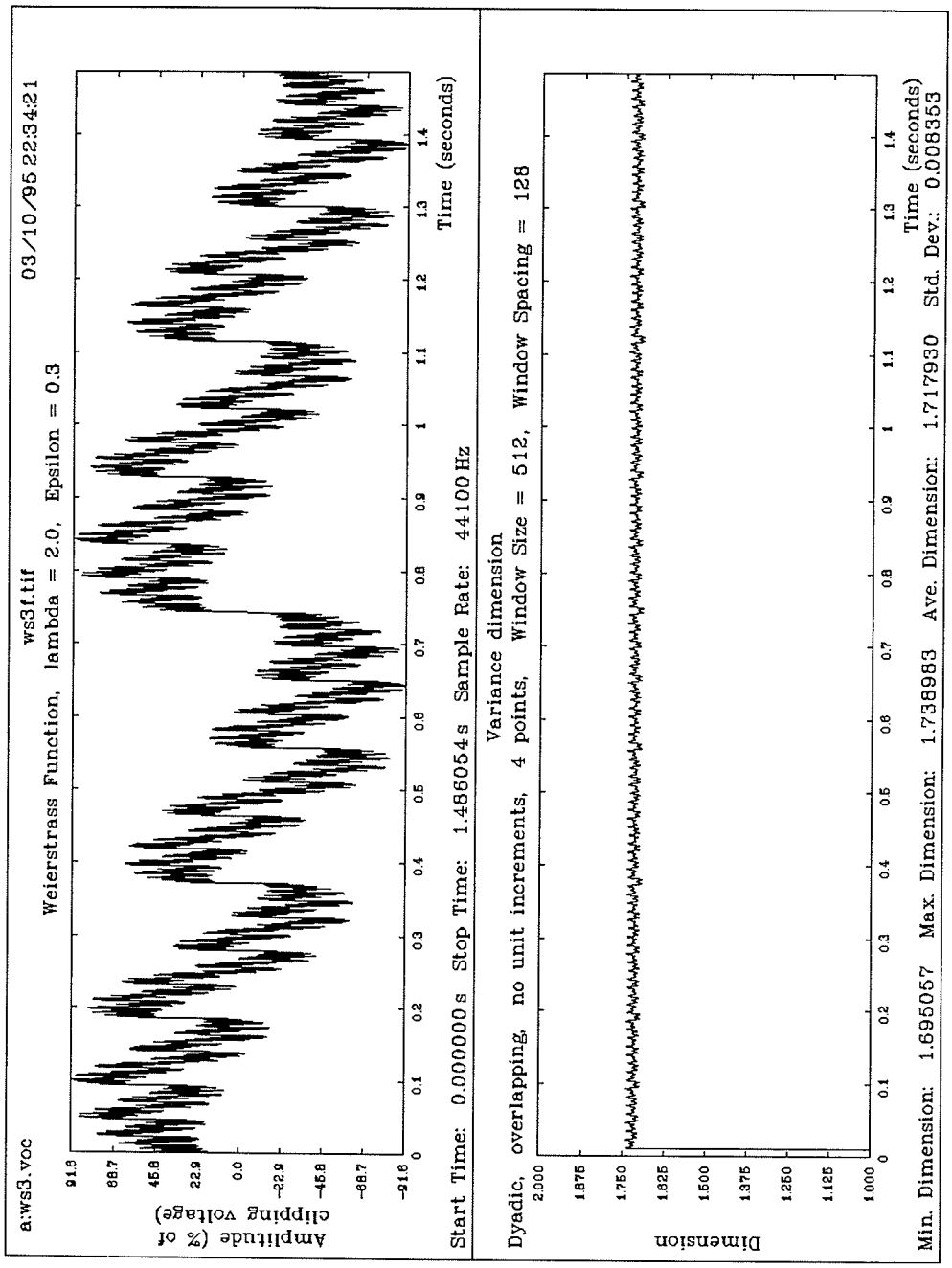
Weierstrass Function, $\varepsilon = 0.1$, Fractal Amplification Parameters



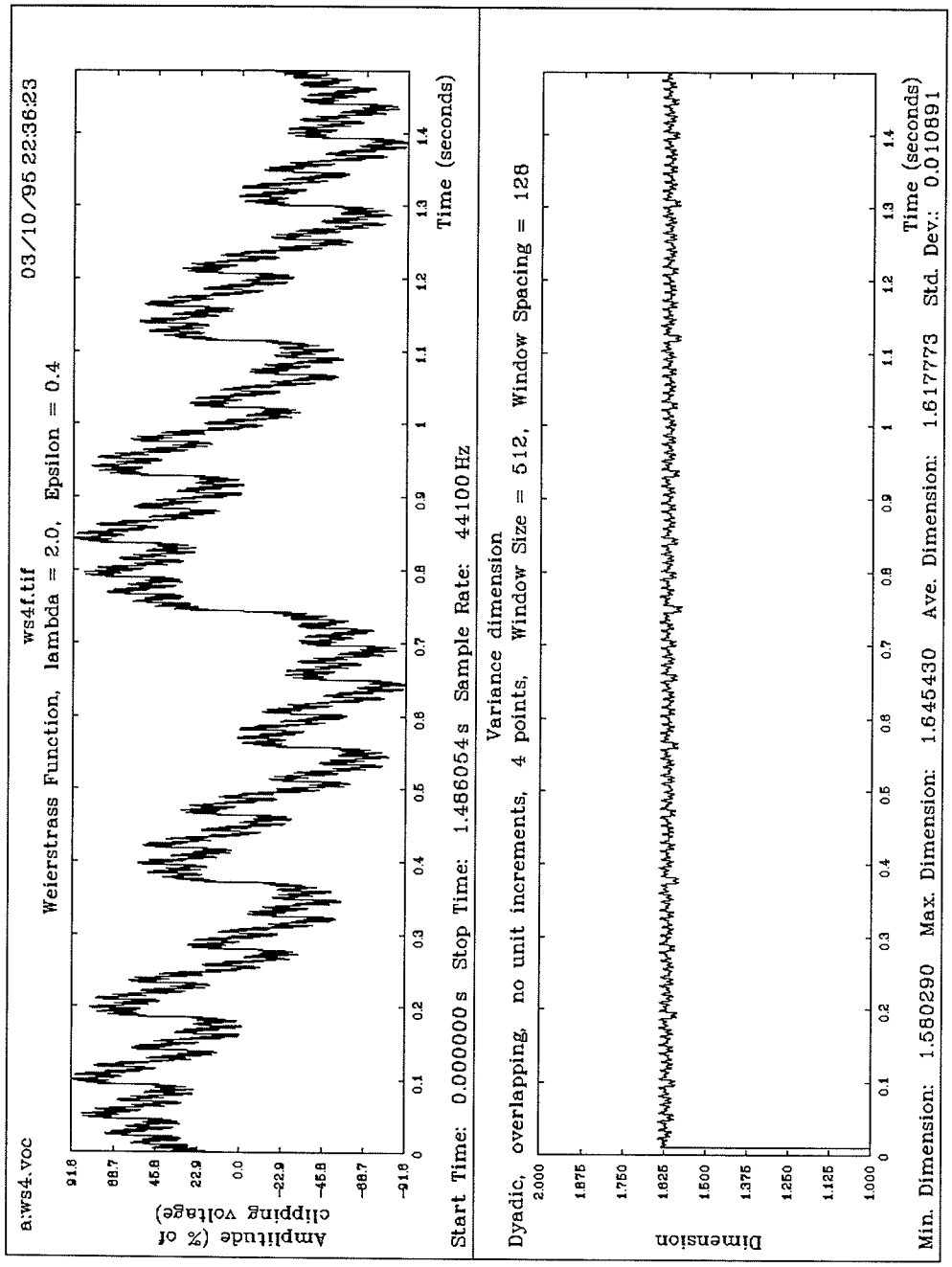
Weierstrass Function, ε = 0.2, Fractal Amplification Parameters



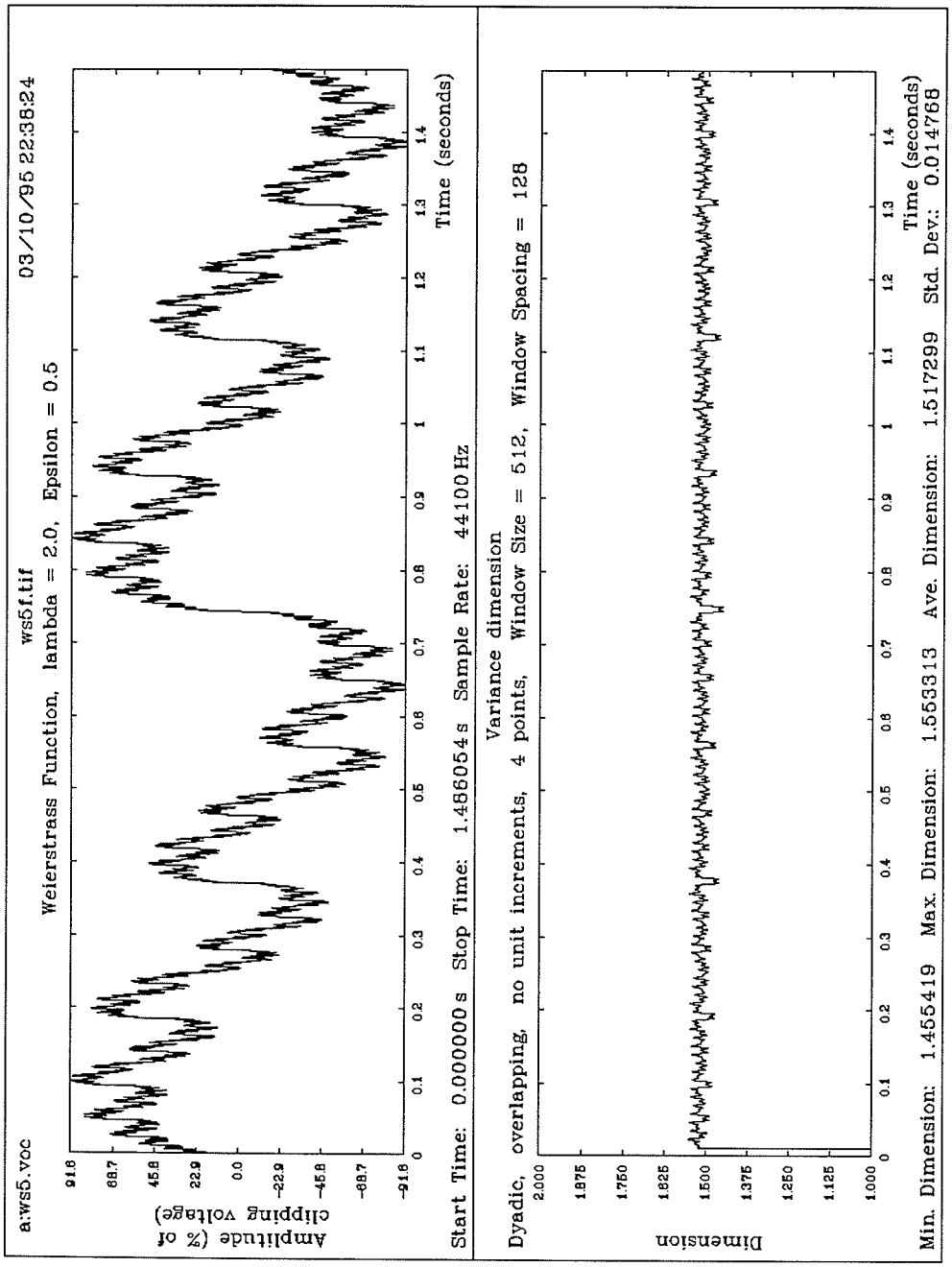
Weierstrass Function, $\varepsilon = 0.3$, Fractal Amplification Parameters



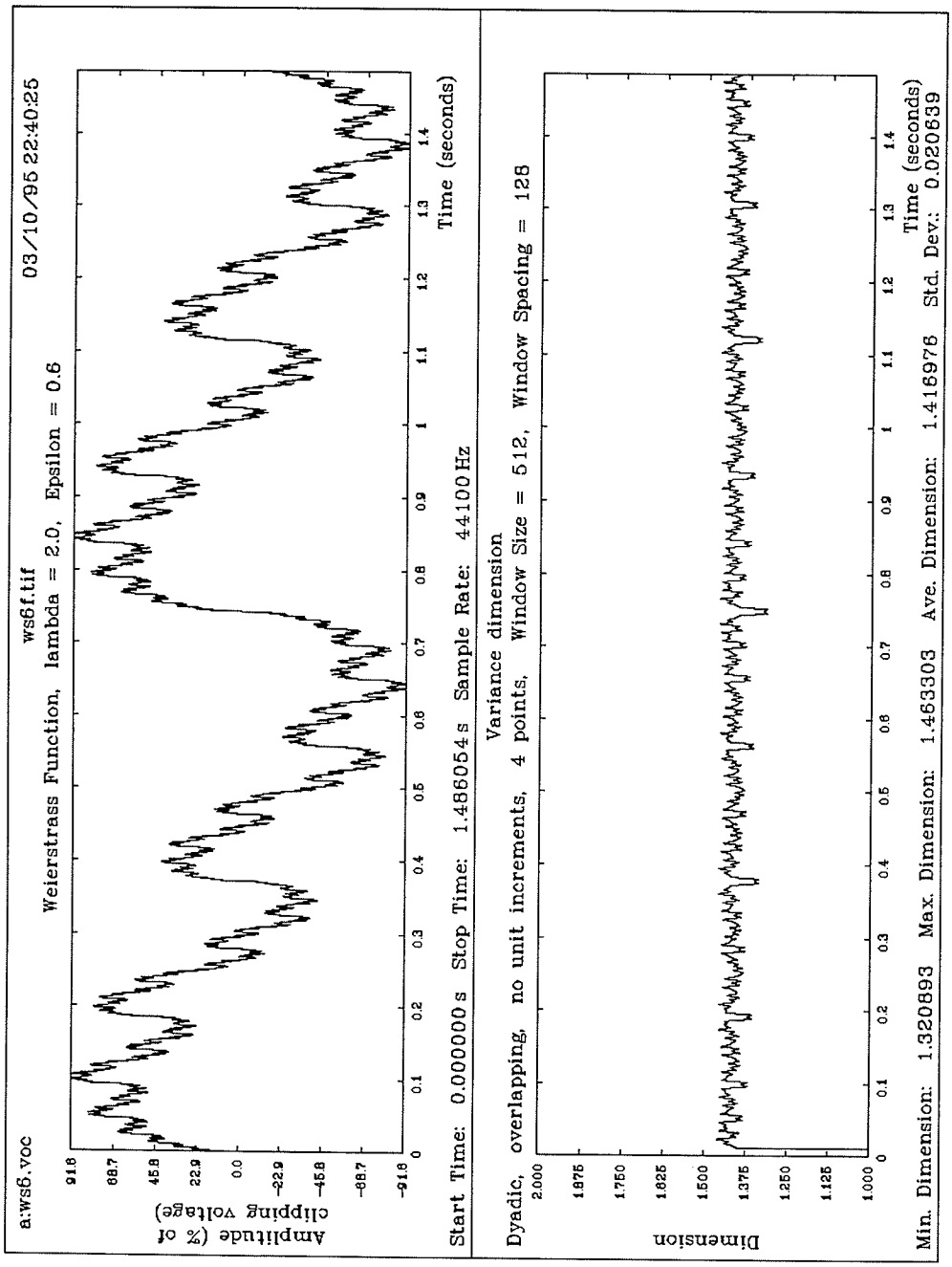
Weierstrass Function, $\varepsilon = 0.4$, Fractal Amplification Parameters



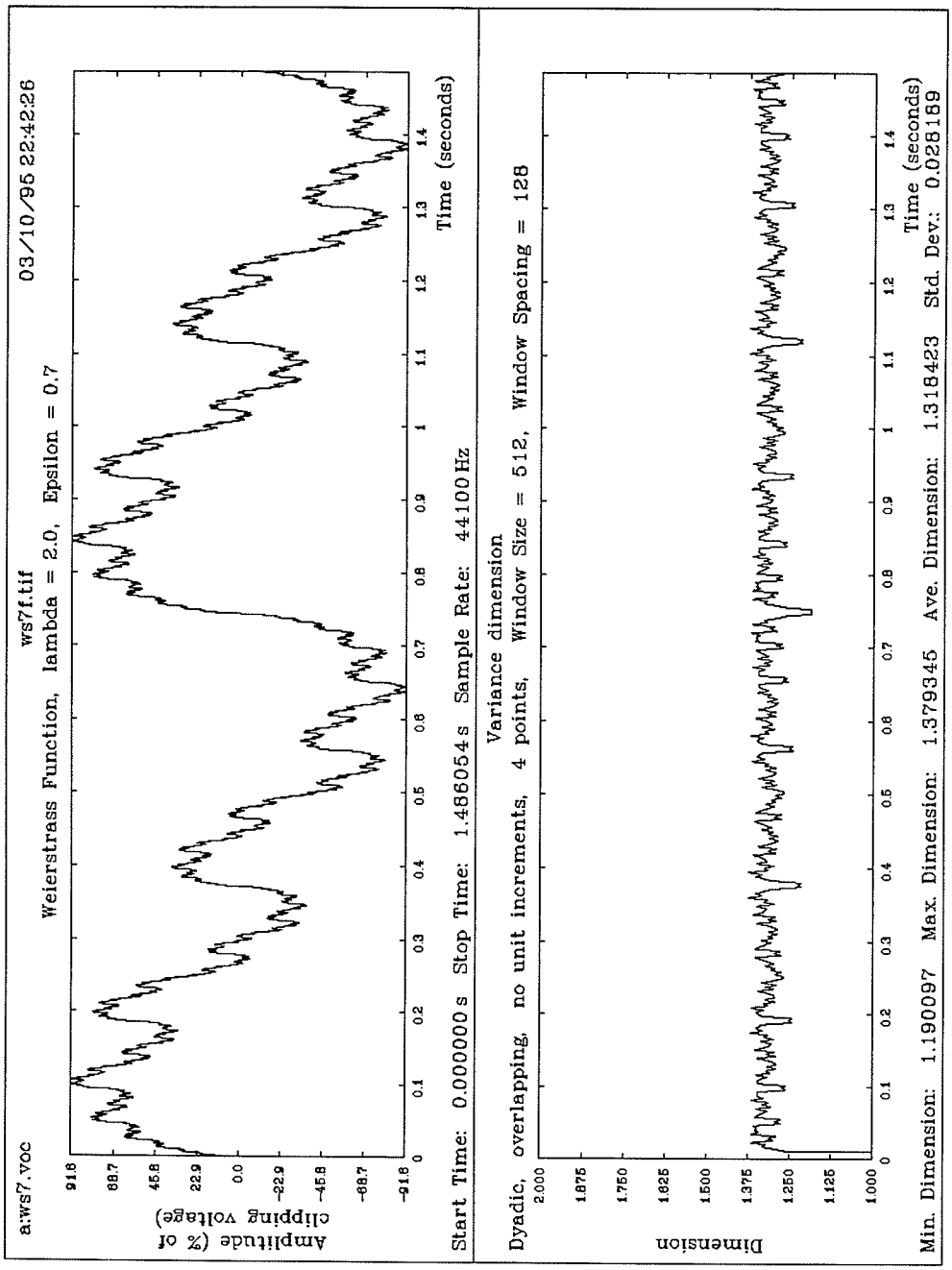
Weierstrass Function, $\varepsilon = 0.5$, Fractal Amplification Parameters



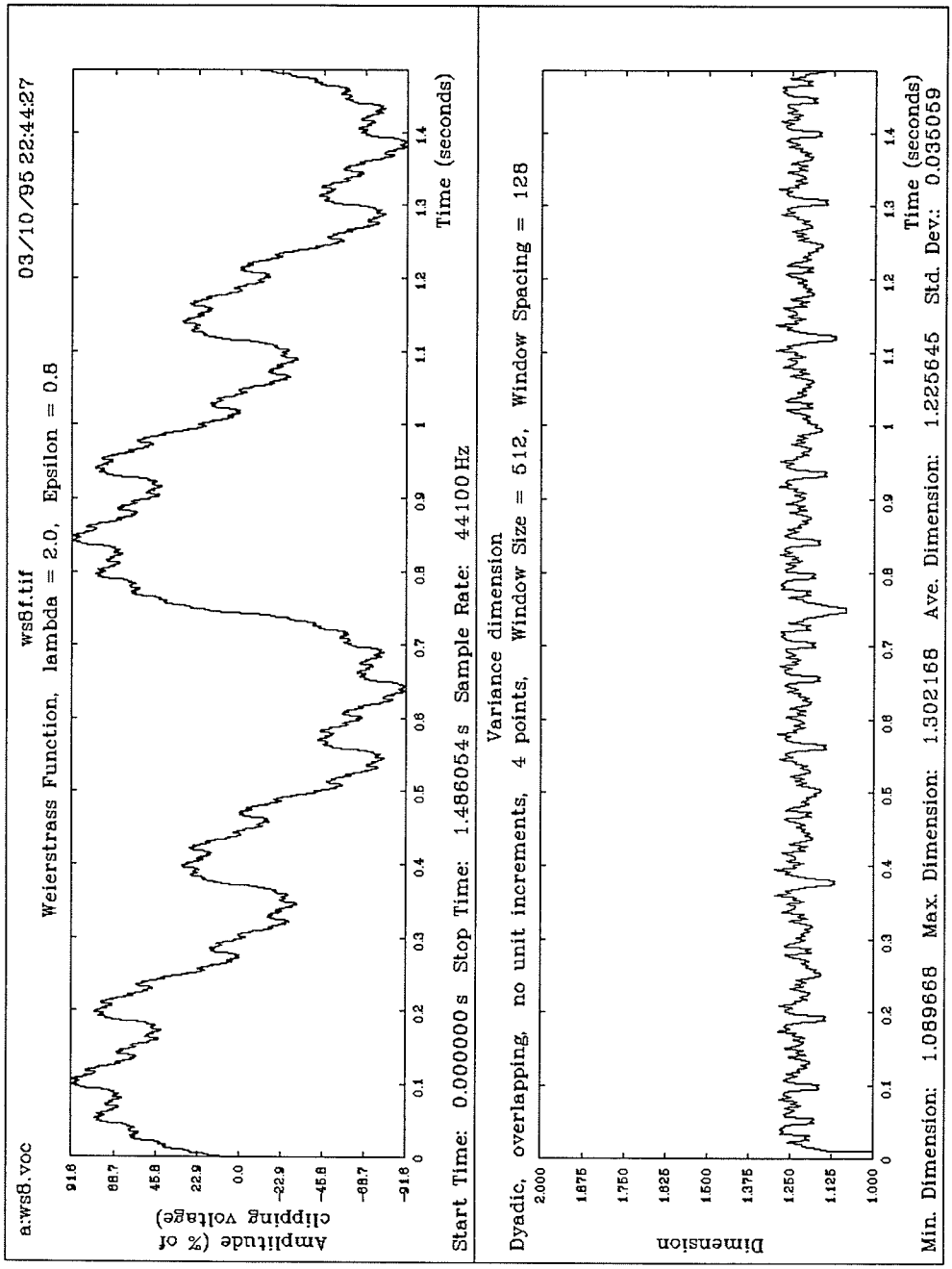
Weierstrass Function, $\varepsilon = 0.6$, Fractal Amplification Parameters



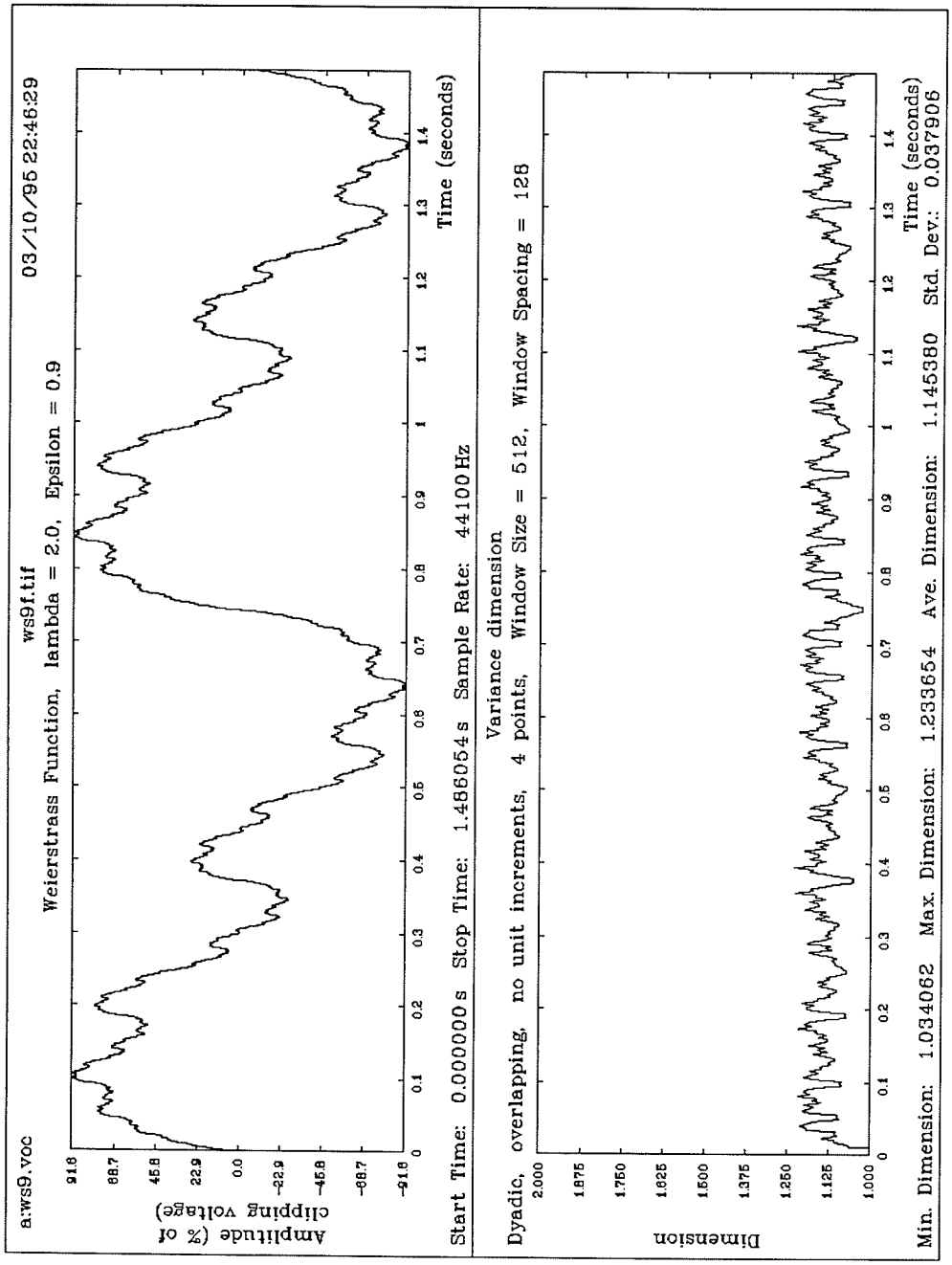
Weierstrass Function, $\varepsilon = 0.7$, Fractal Amplification Parameters



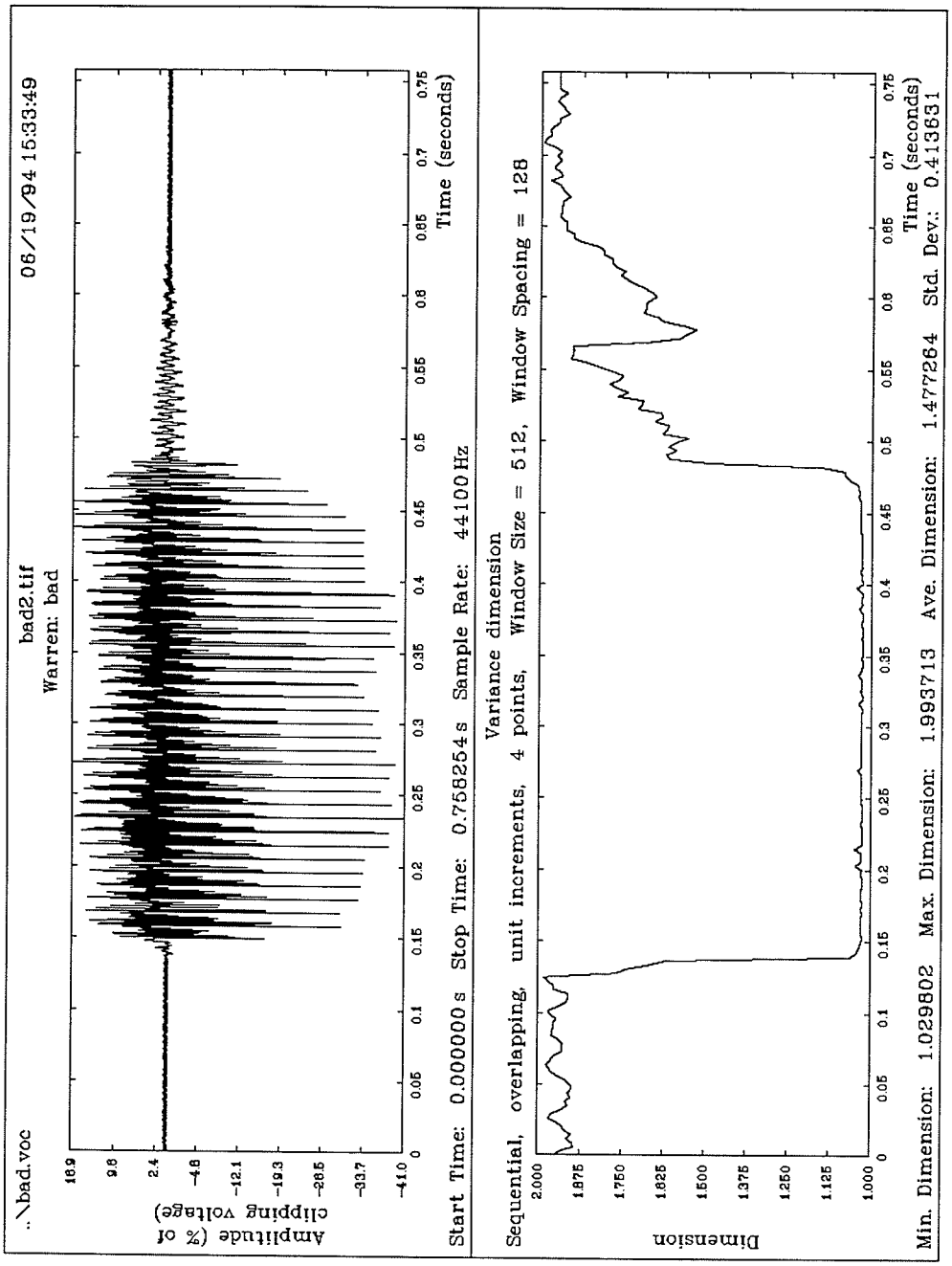
Weierstrass Function, $\varepsilon = 0.8$, Fractal Amplification Parameters



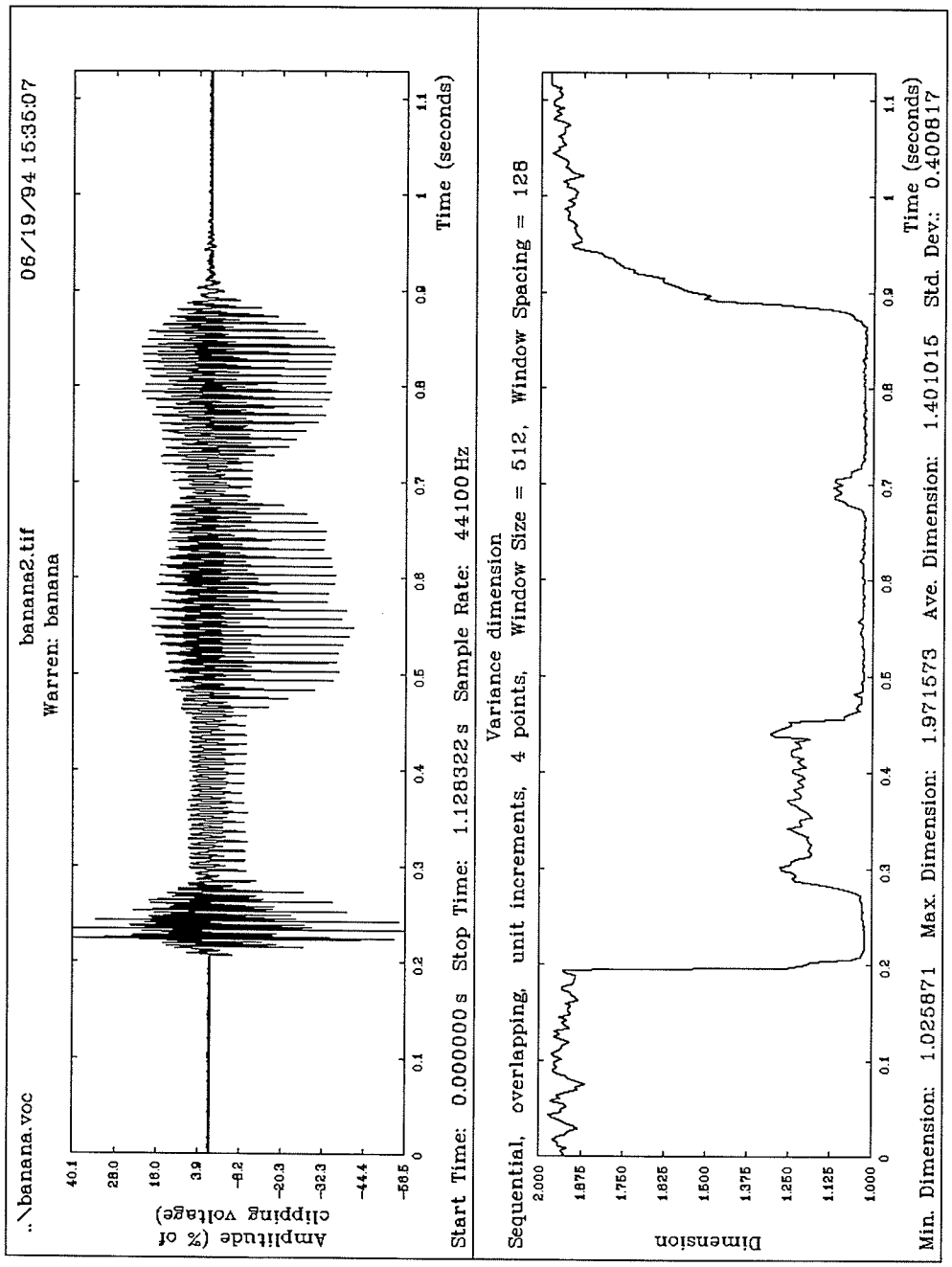
Weierstrass Function, $\varepsilon = 0.9$, Fractal Amplification Parameters



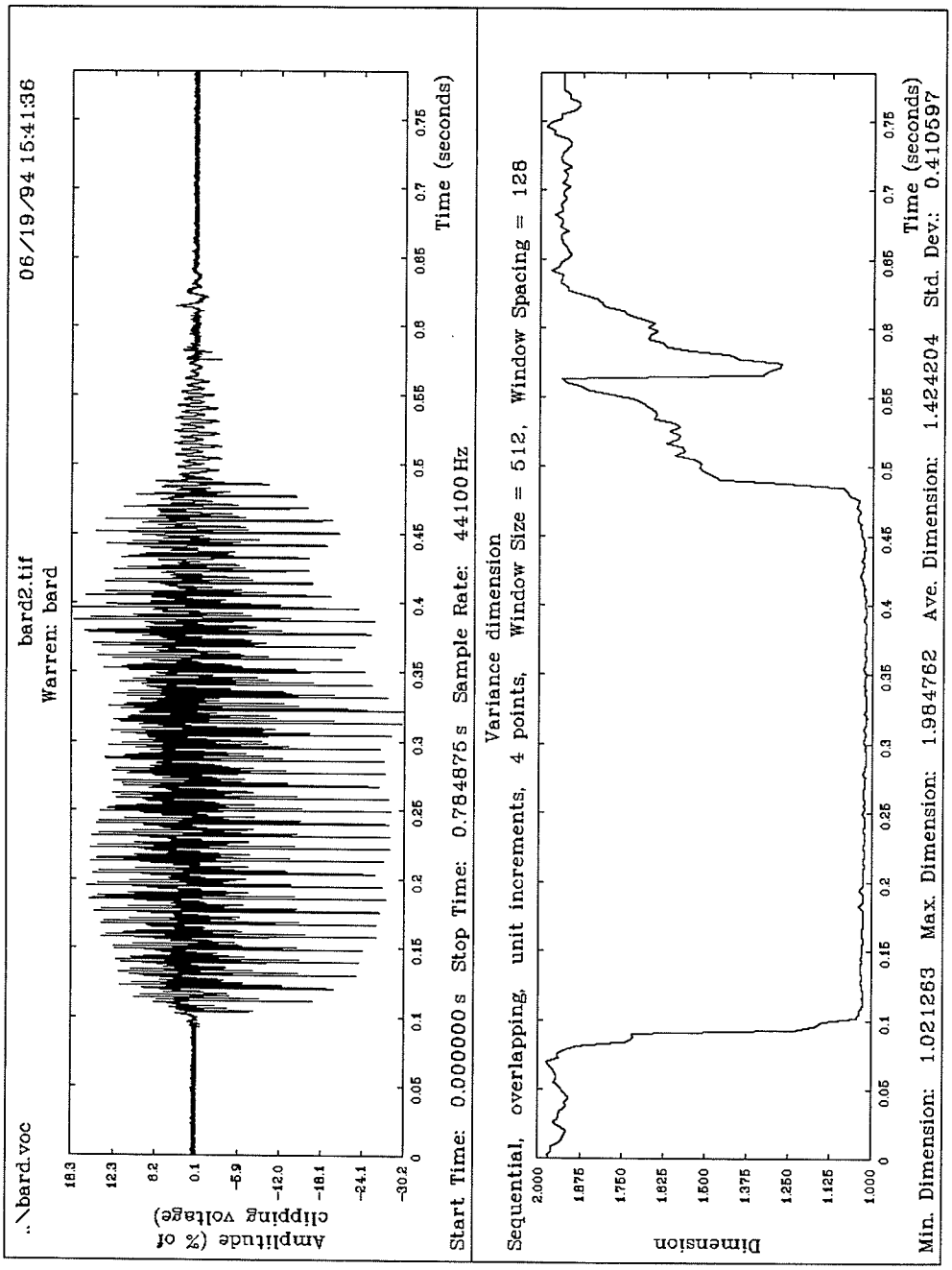
Bad, Noise Separation Parameters



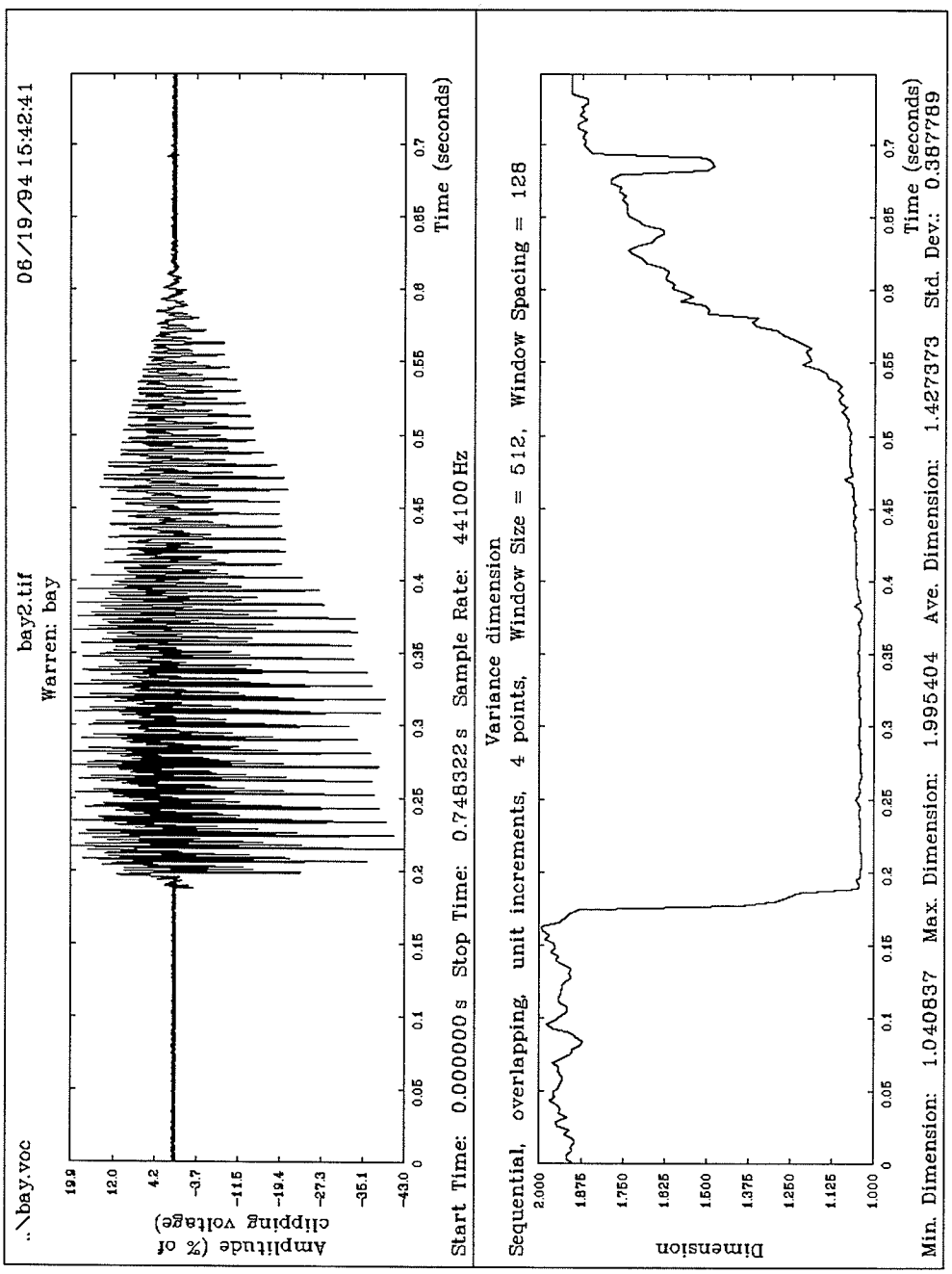
Banana, Noise Separation Parameters



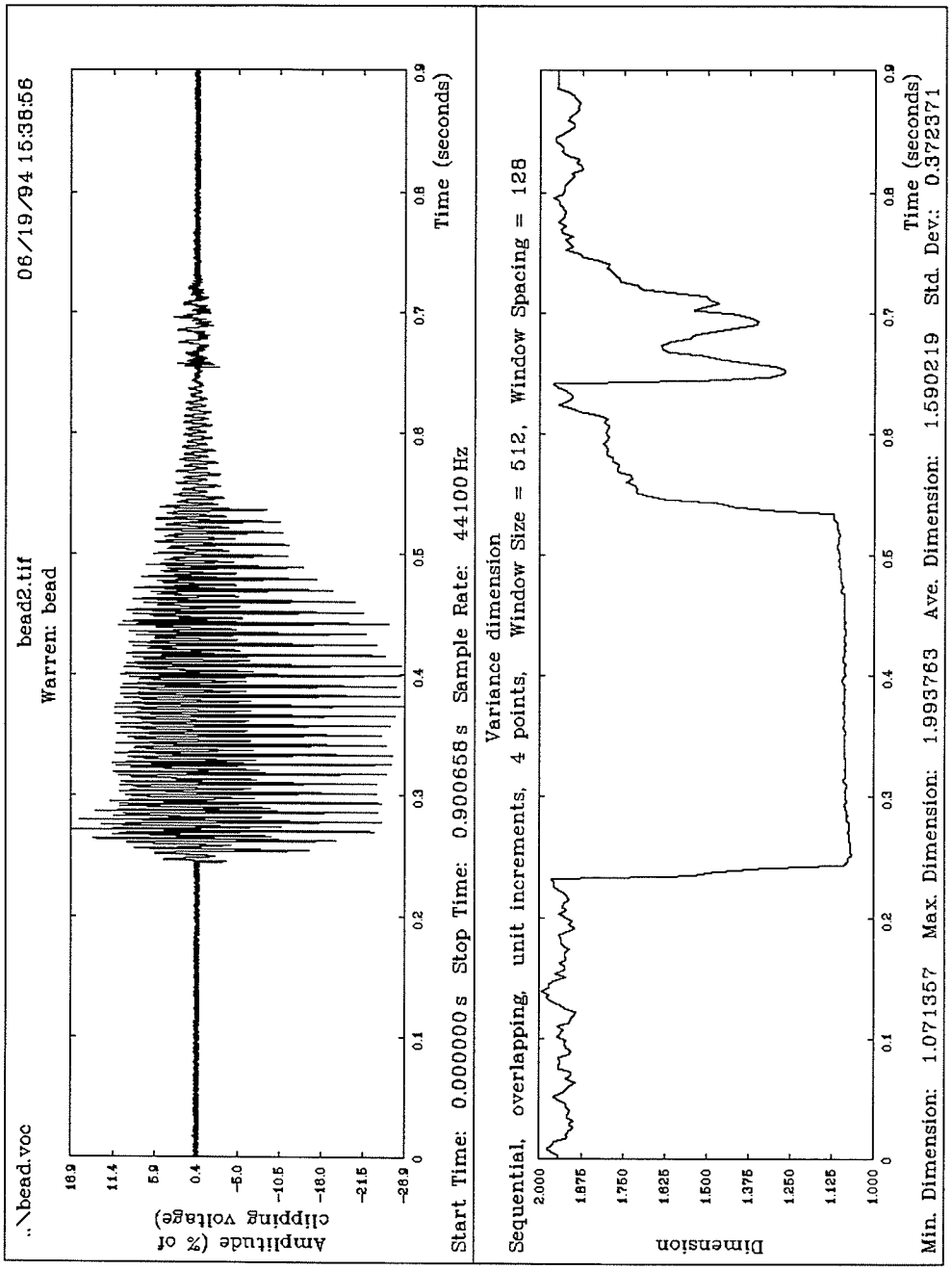
Bard, Noise Separation Parameters



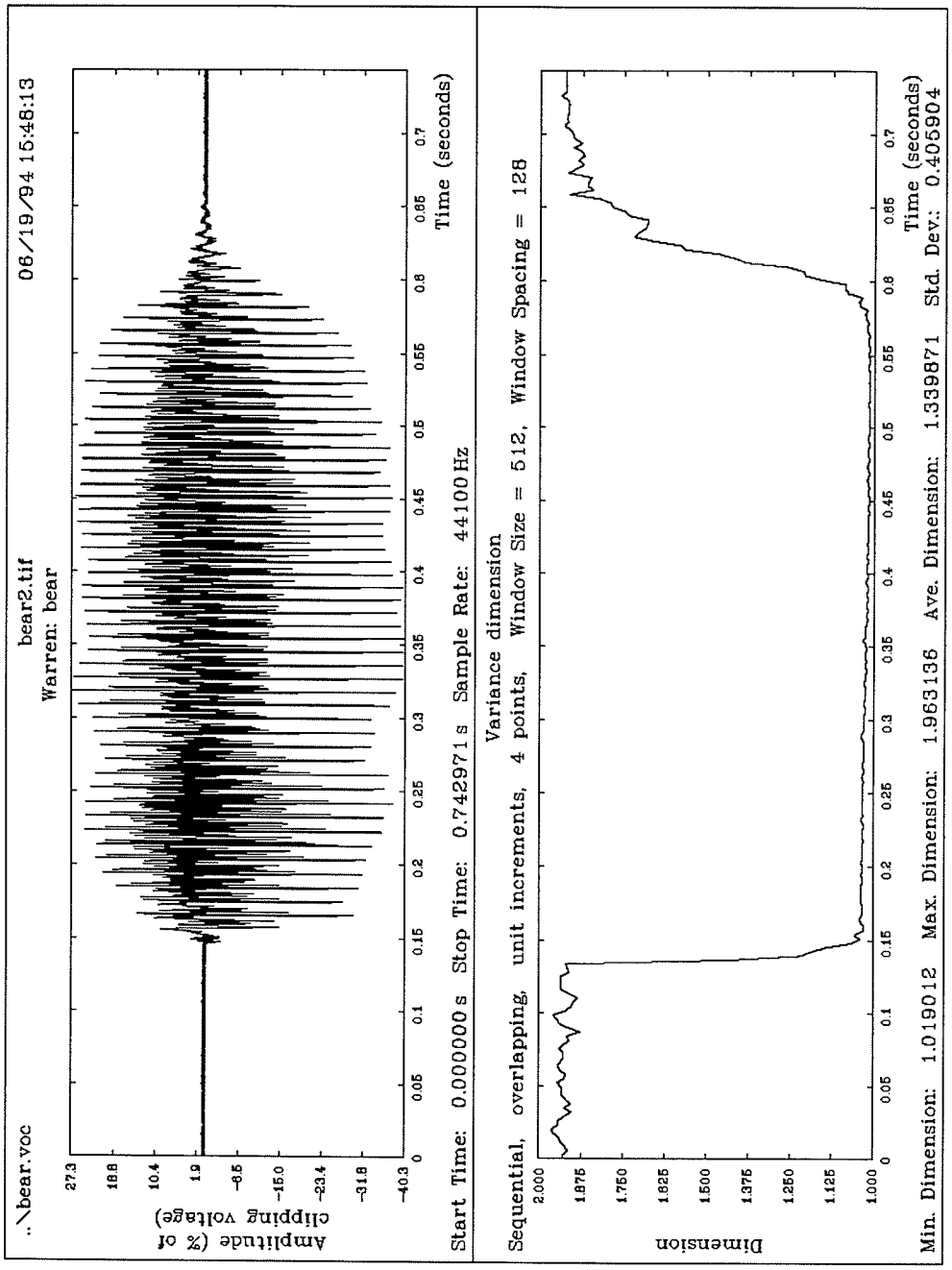
Bay, Noise Separation Parameters



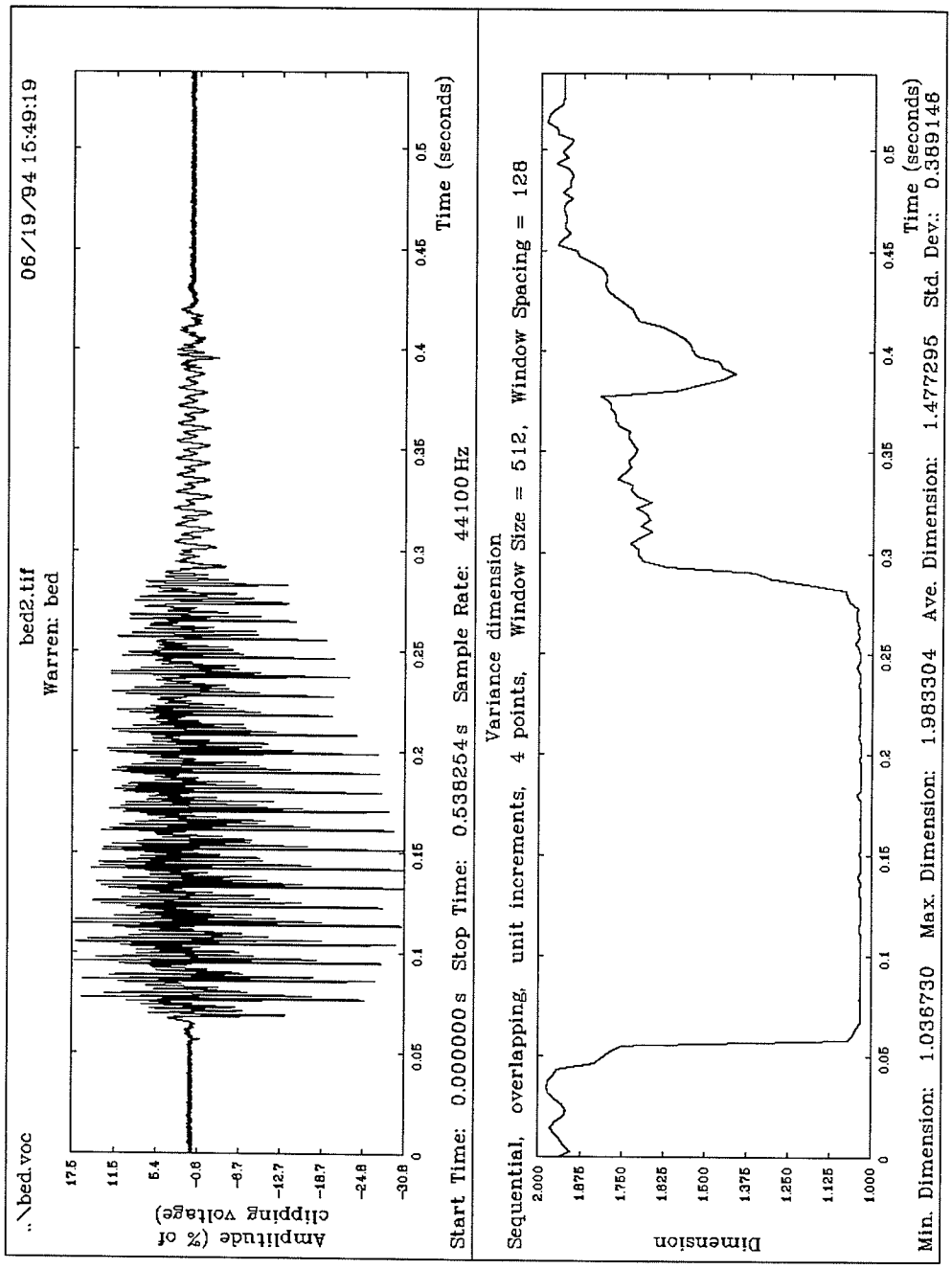
Bead, Noise Separation Parameters



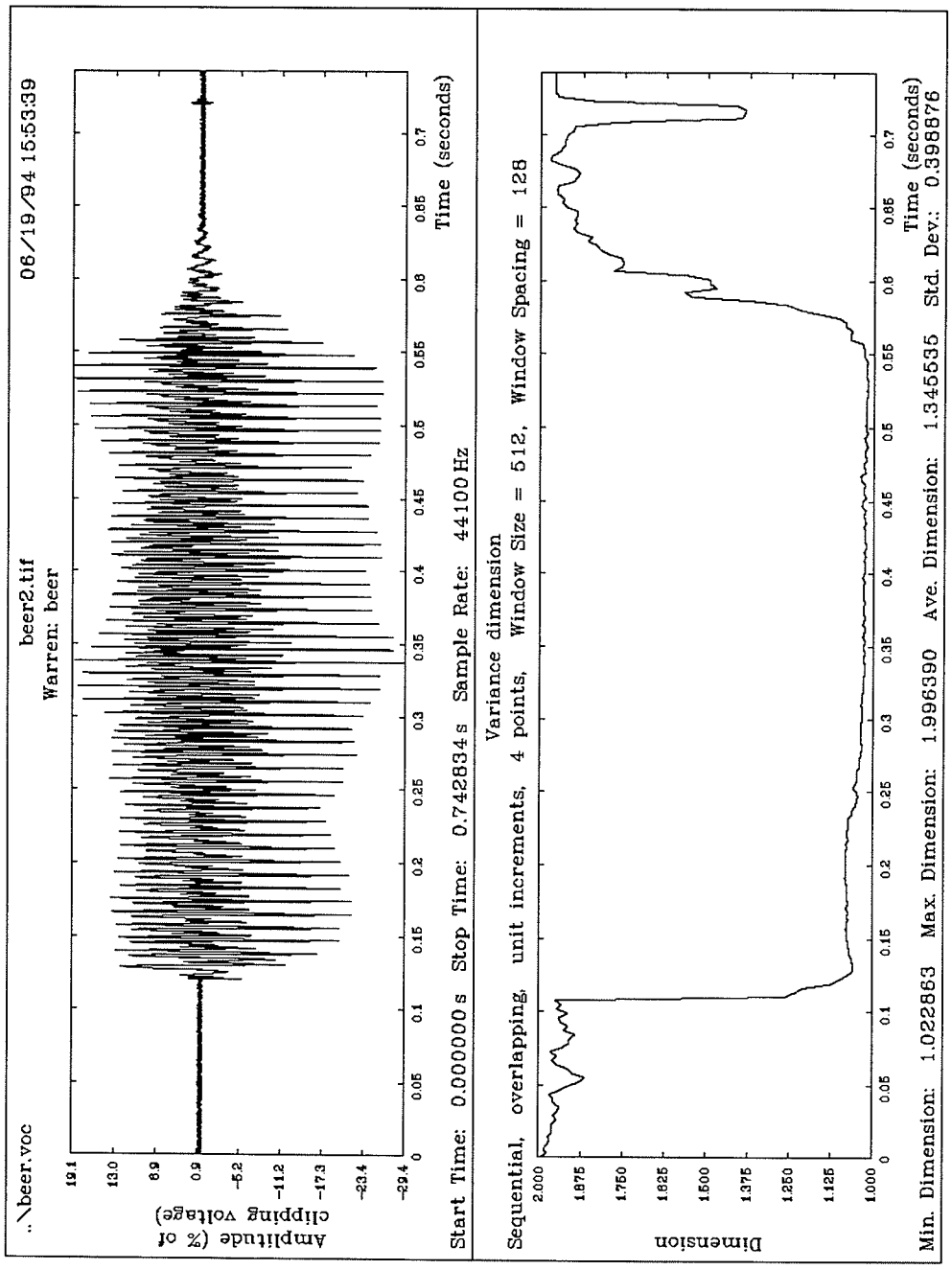
Bear, Noise Separation Parameters



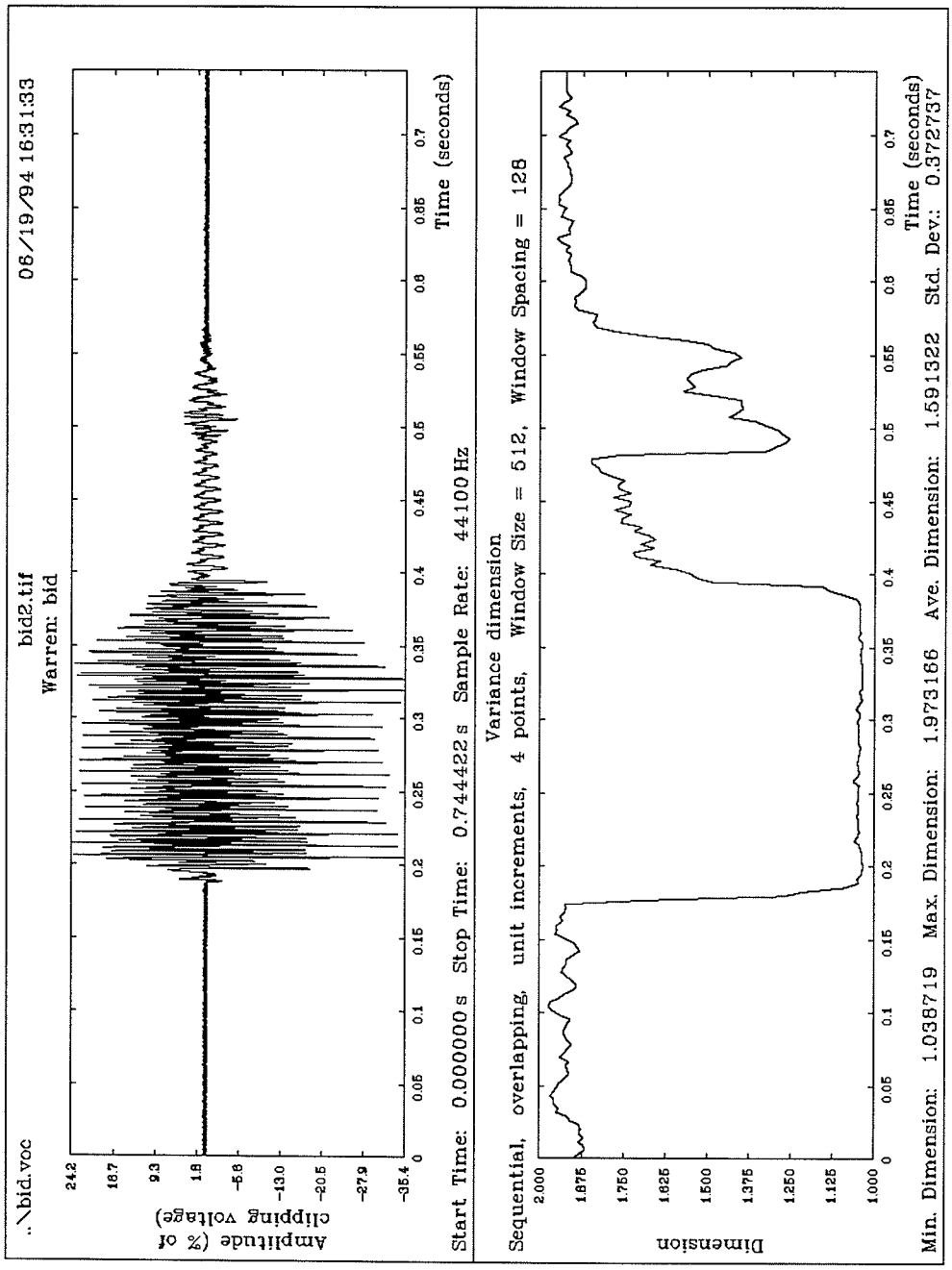
Bed, Noise Separation Parameters



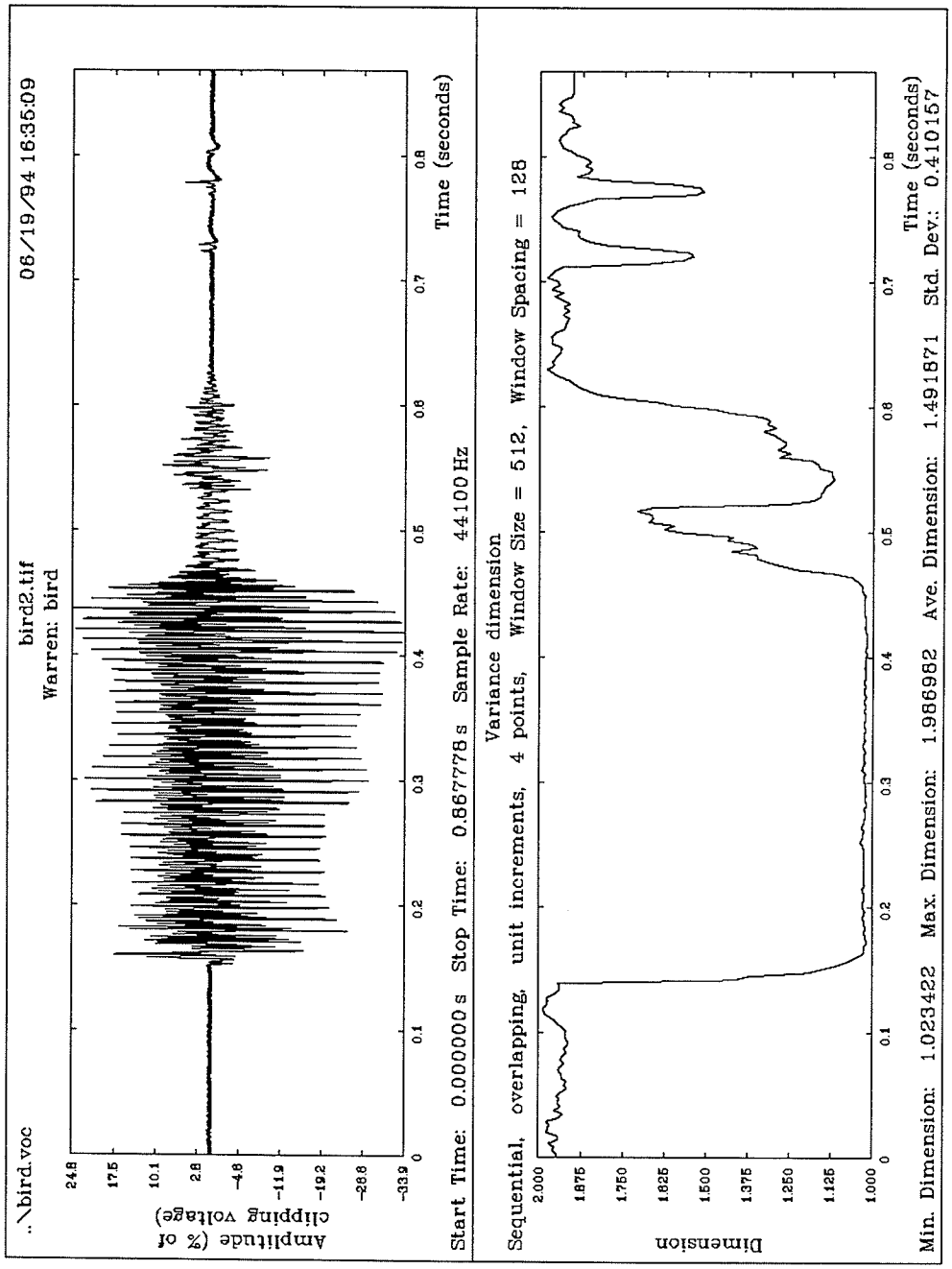
Beer, Noise Separation Parameters



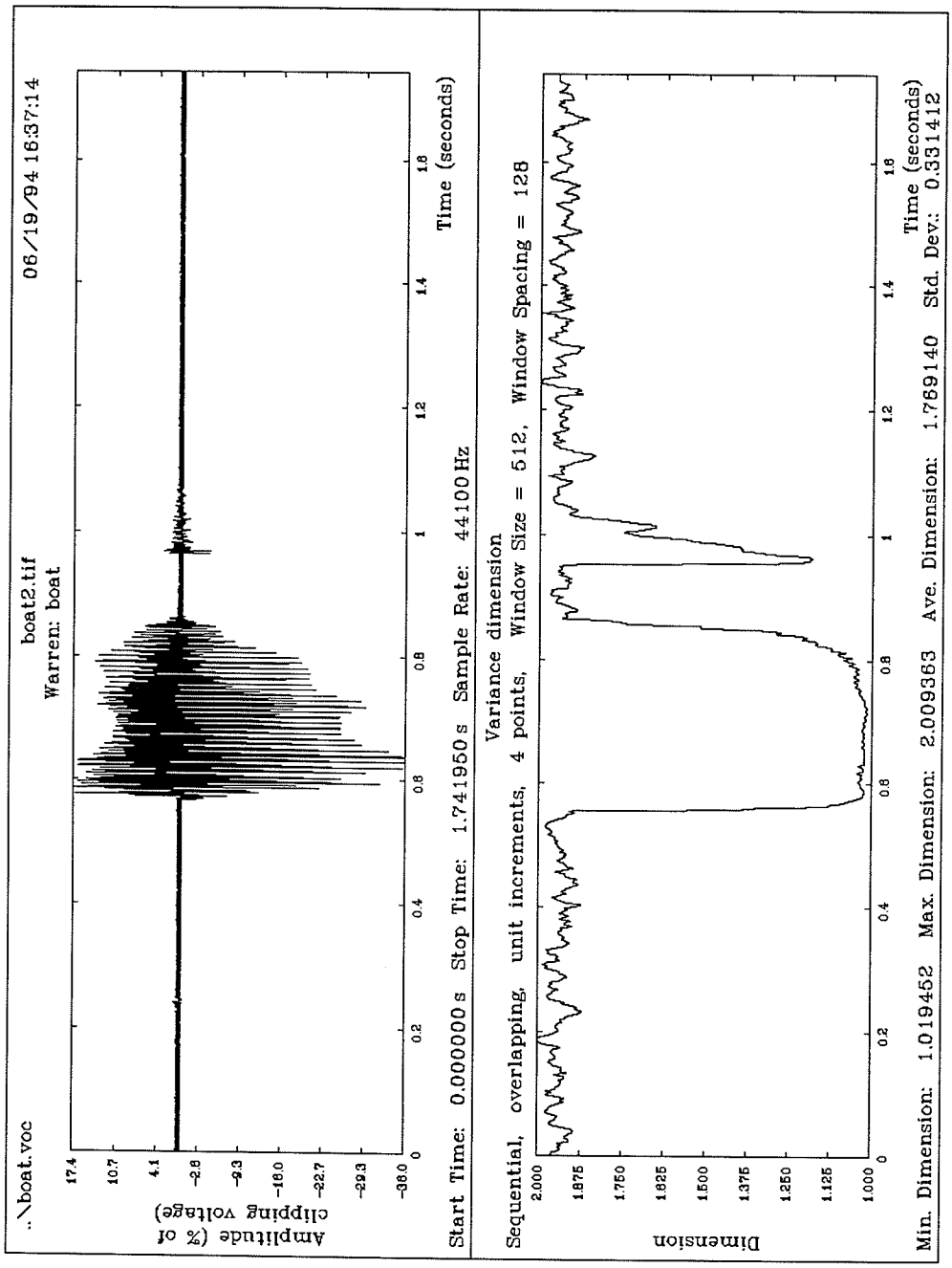
Bid, Noise Separation Parameters

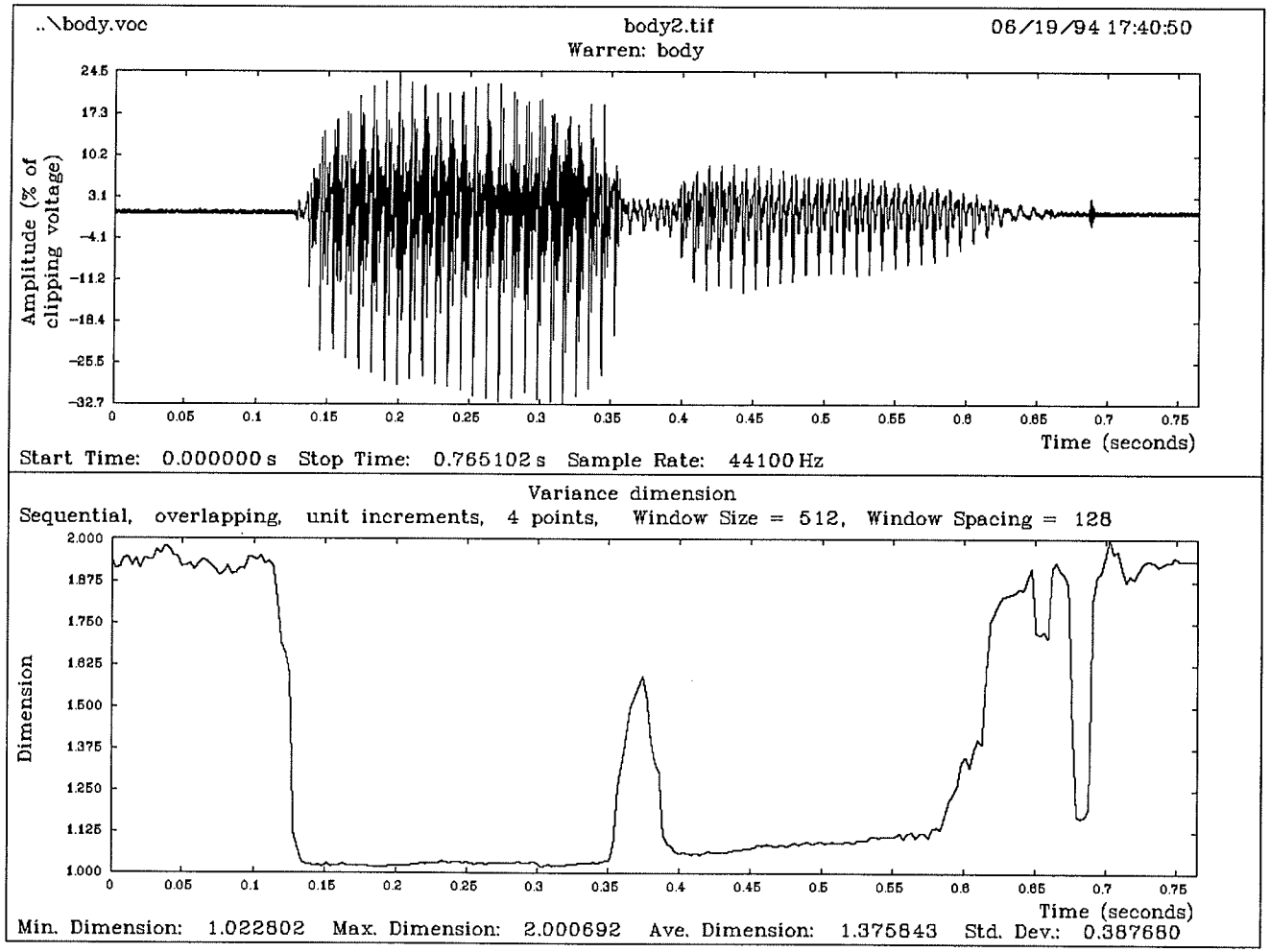


Bird, Noise Separation Parameters

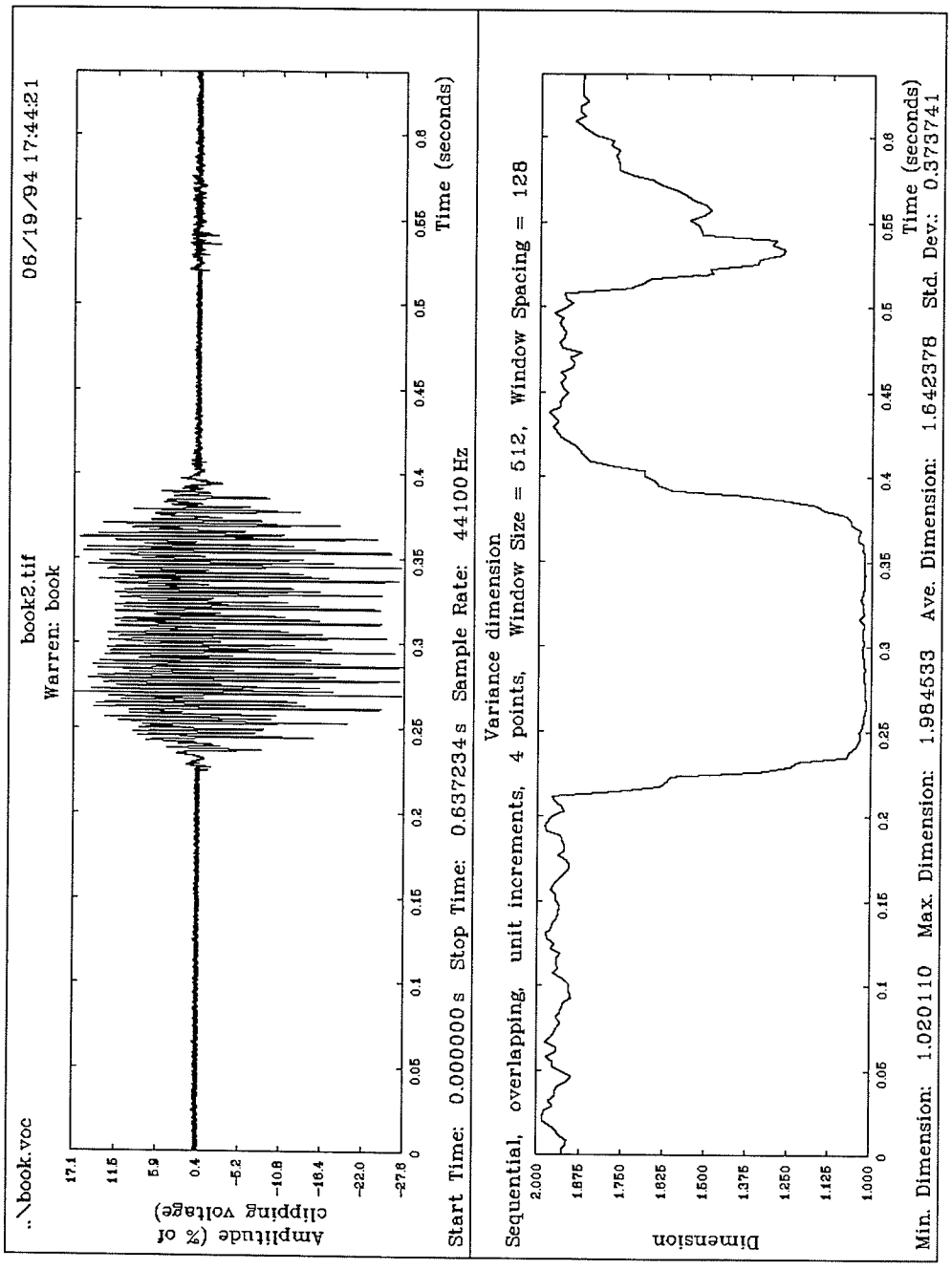


Boat, Noise Separation Parameters

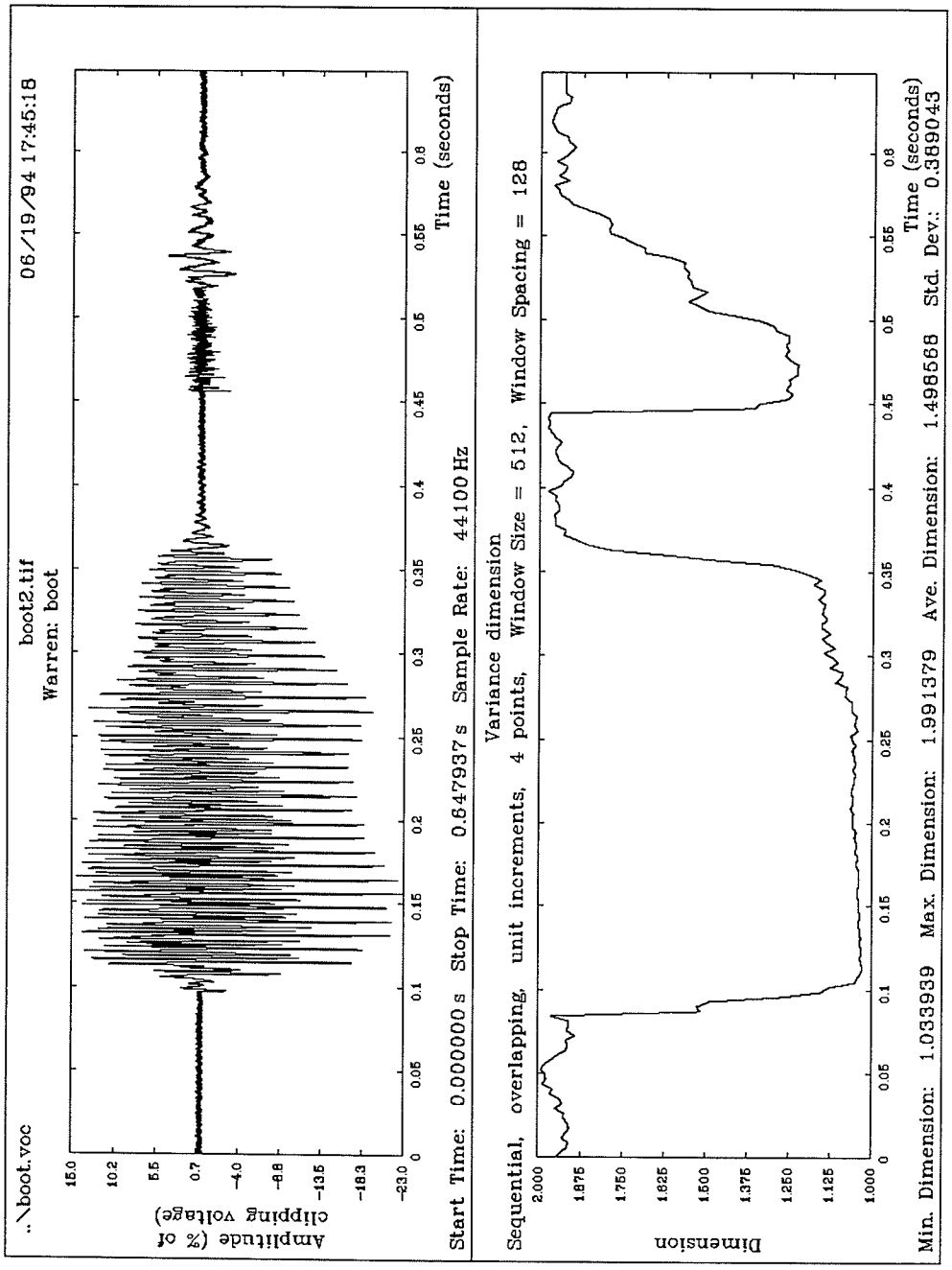



..\body.voc
body2.tif
06/19/94 17:40:50
Warren: body
Amplitude (% of clipping voltage)
24.5
17.3
10.2
3.1
-4.1
-11.2
-18.4
-25.5
-32.7
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
Time (seconds)
Start Time: 0.000000 s Stop Time: 0.765102 s Sample Rate: 44100 Hz
Variance dimension
Sequential, overlapping, unit increments, 4 points, Window Size = 512, Window Spacing = 128
Dimension
2.000
1.875
1.750
1.625
1.500
1.375
1.250
1.125
1.000
Time (seconds)
Min. Dimension: 1.022802 Max. Dimension: 2.000692 Ave. Dimension: 1.375843 Std. Dev.: 0.387660

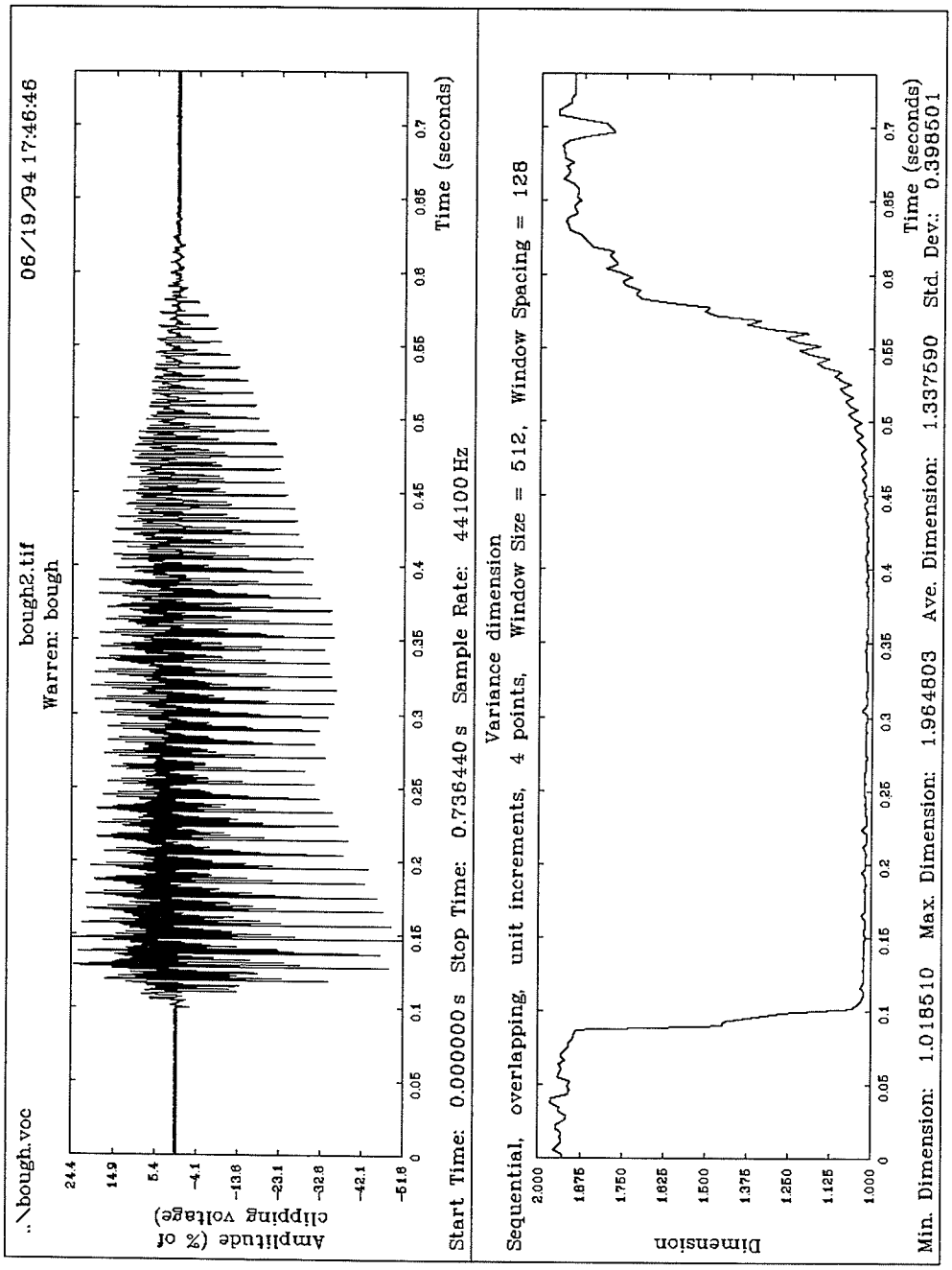
Book, Noise Separation Parameters



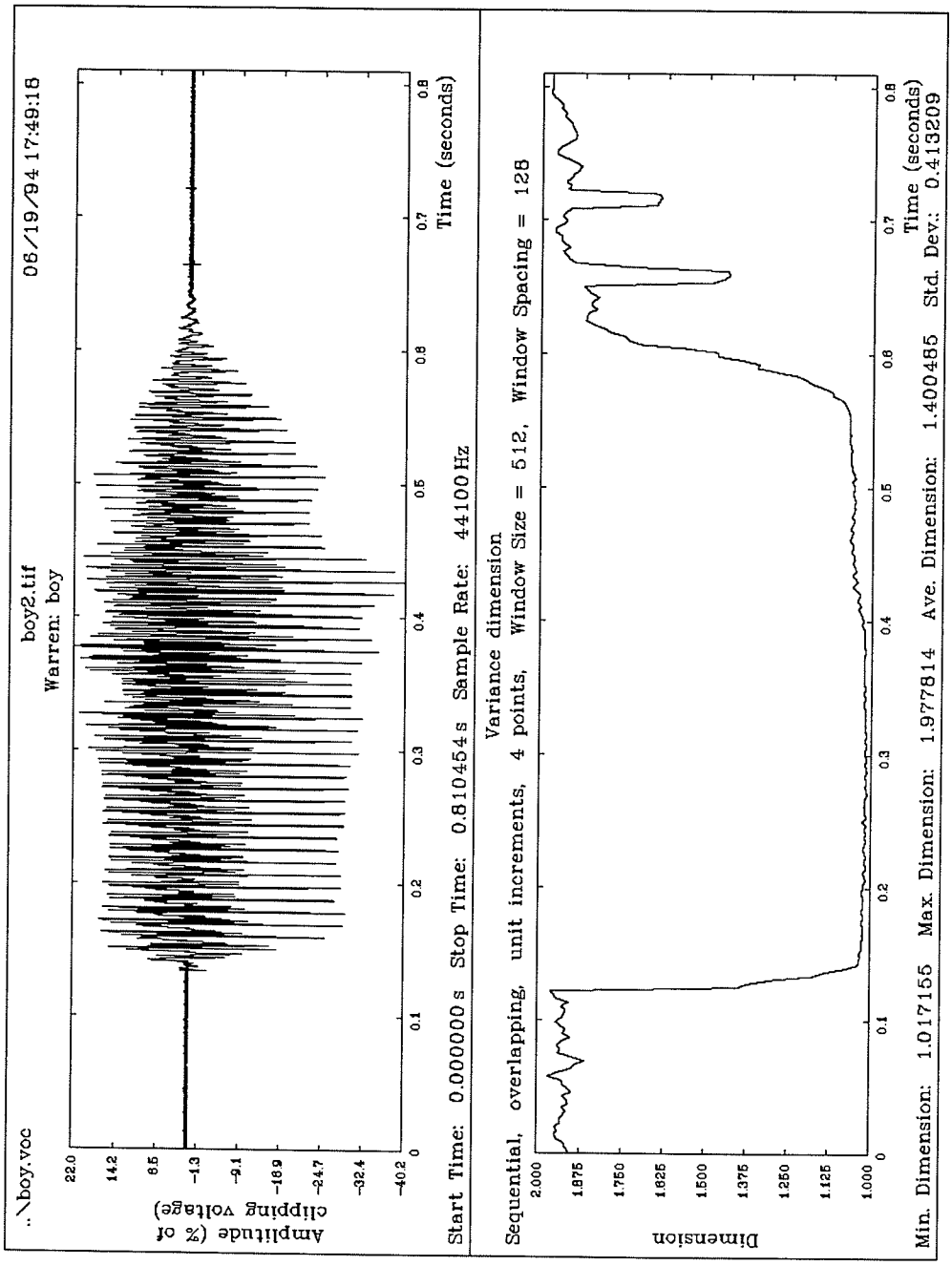
Boot, Noise Separation Parameters



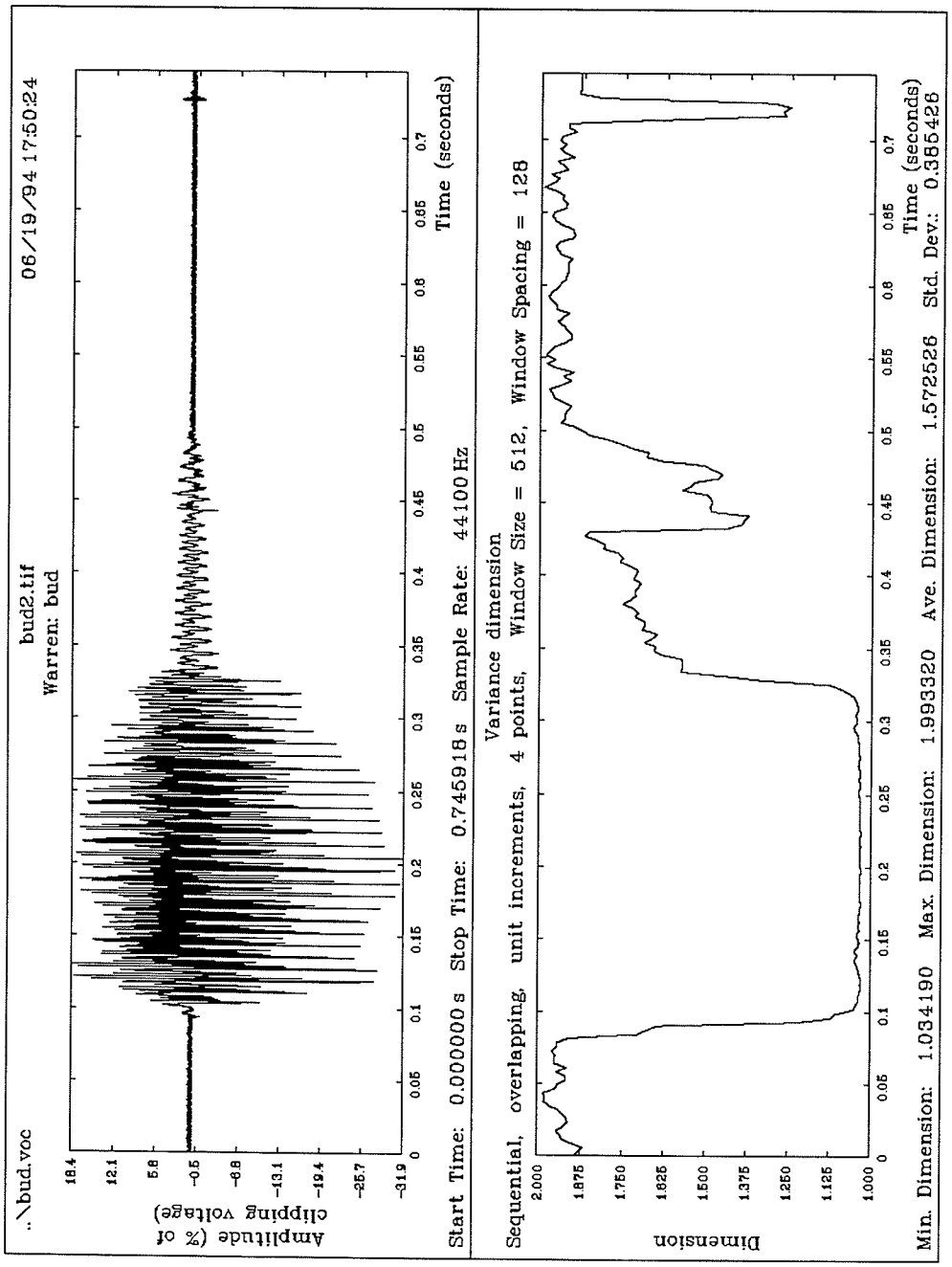
Bough, Noise Separation Parameters



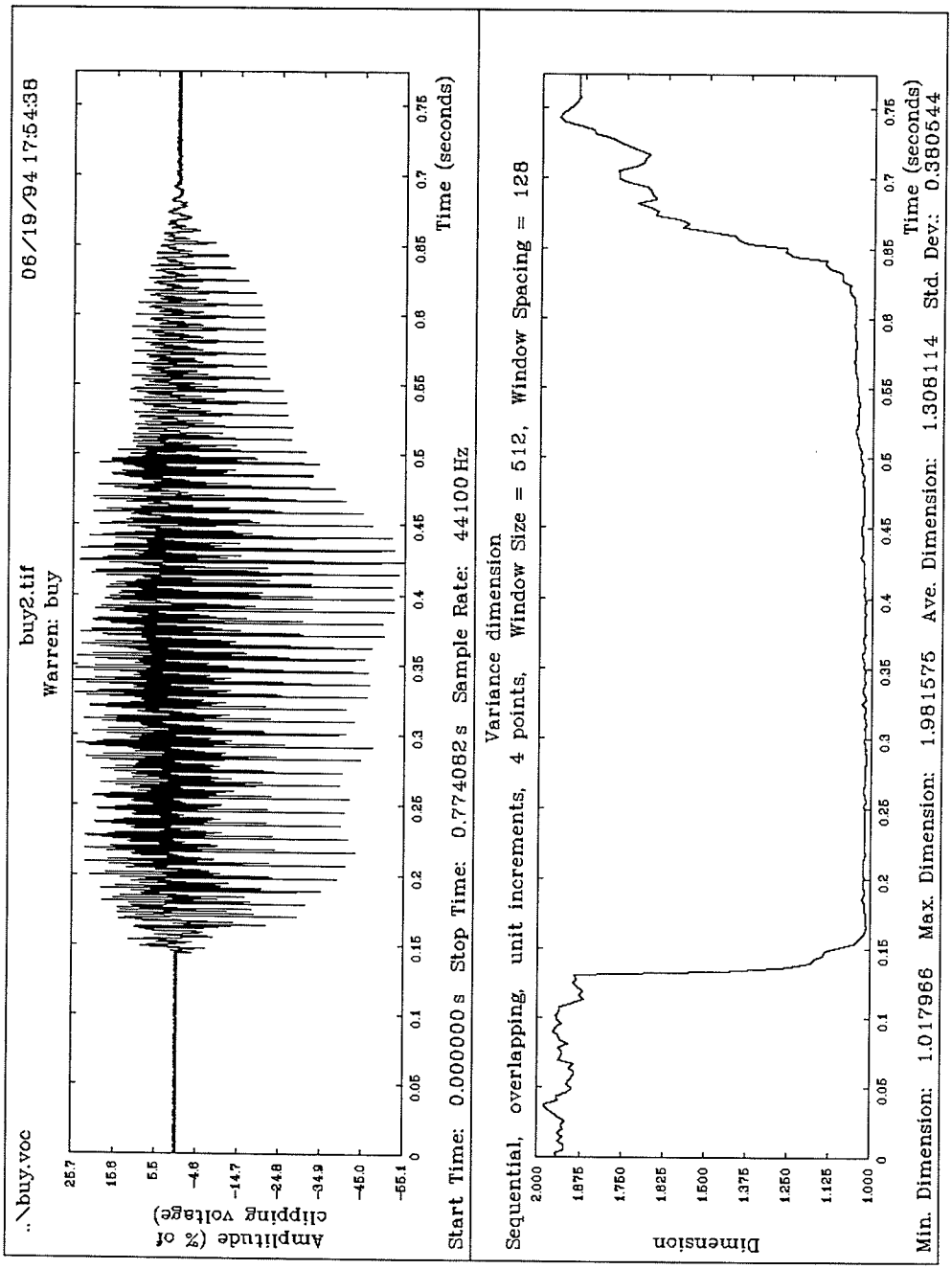
Boy, Noise Separation Parameters



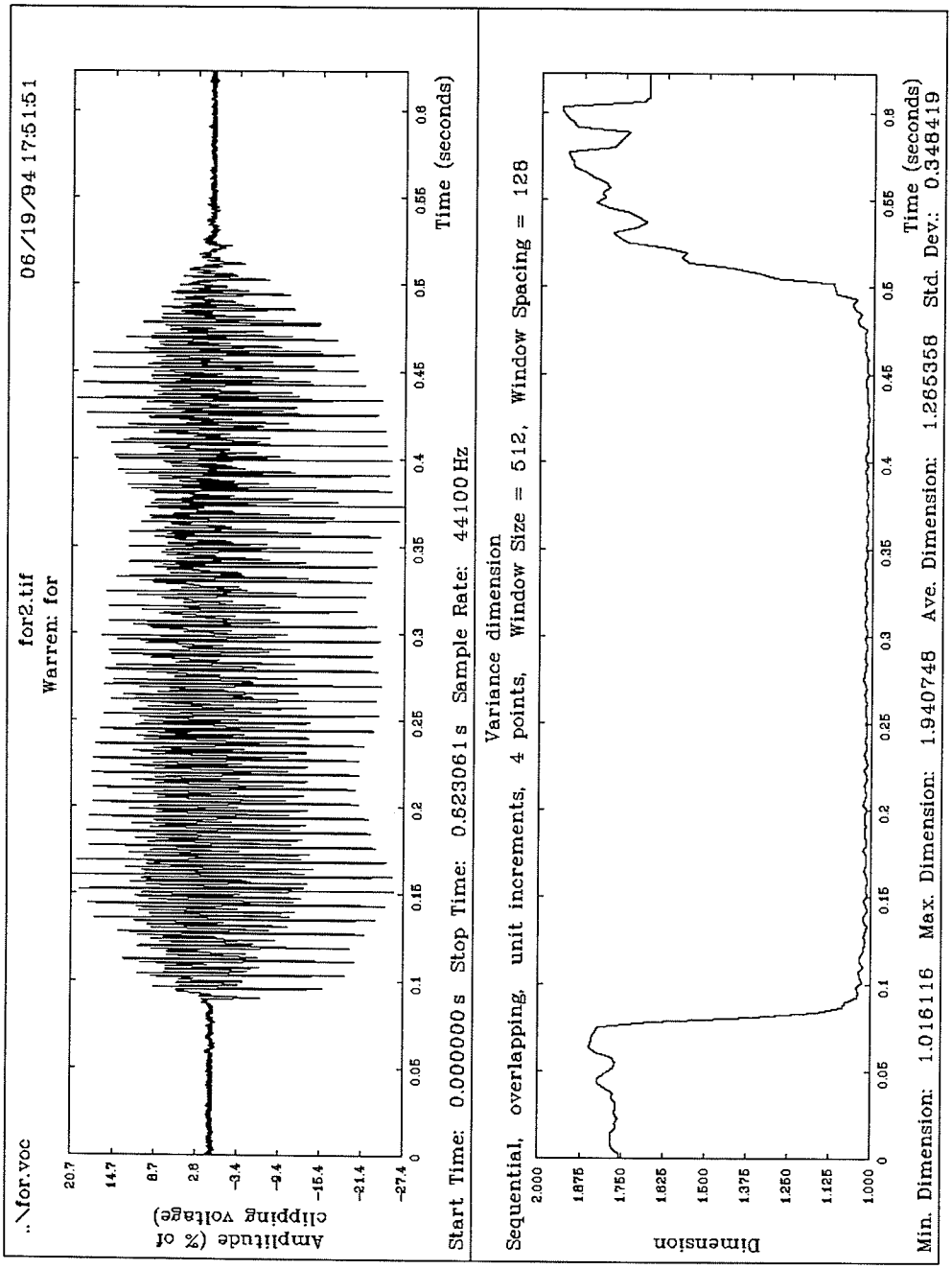
Bud, Noise Separation Parameters



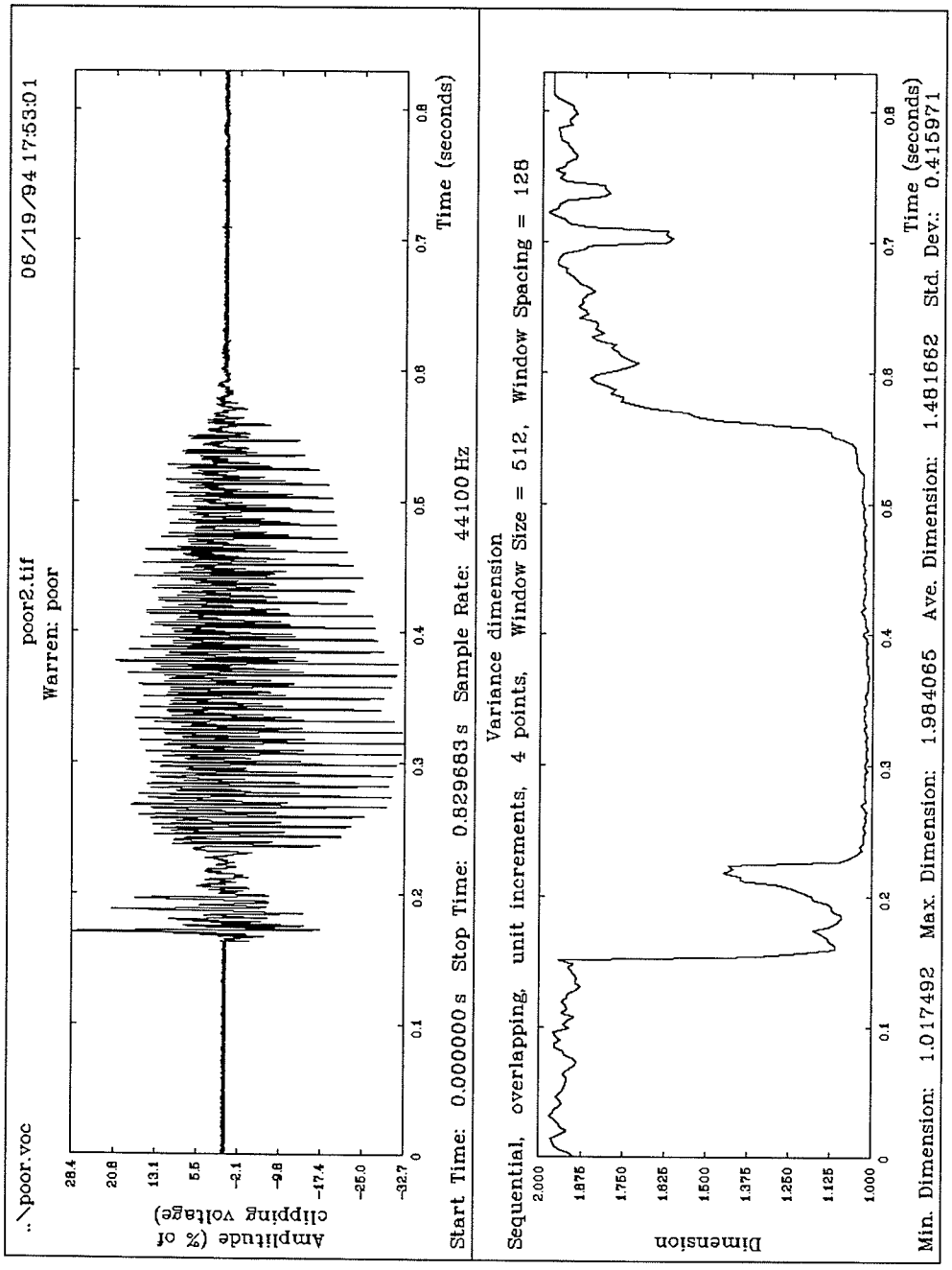
Buy, Noise Separation Parameters



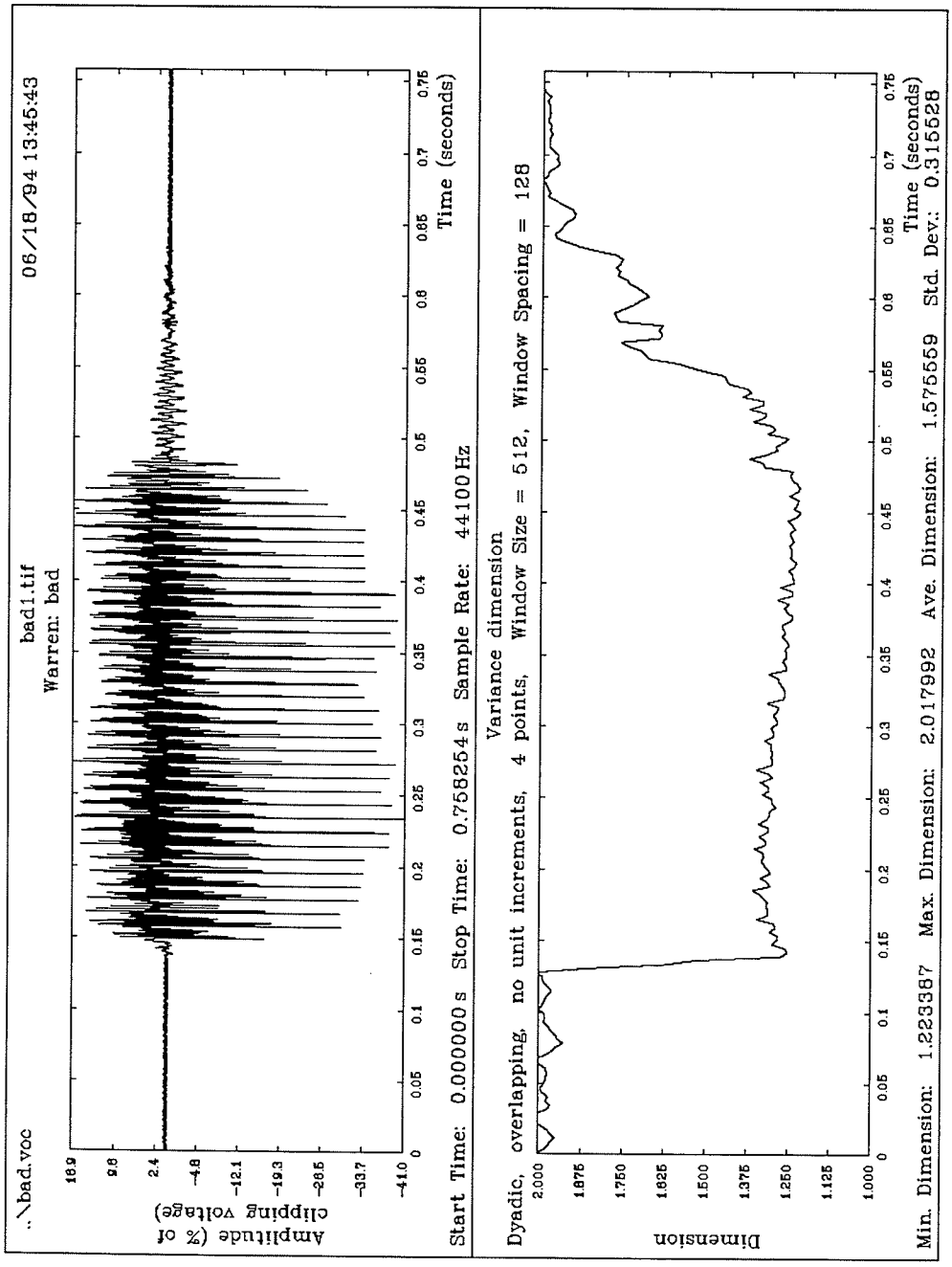
For, Noise Separation Parameters



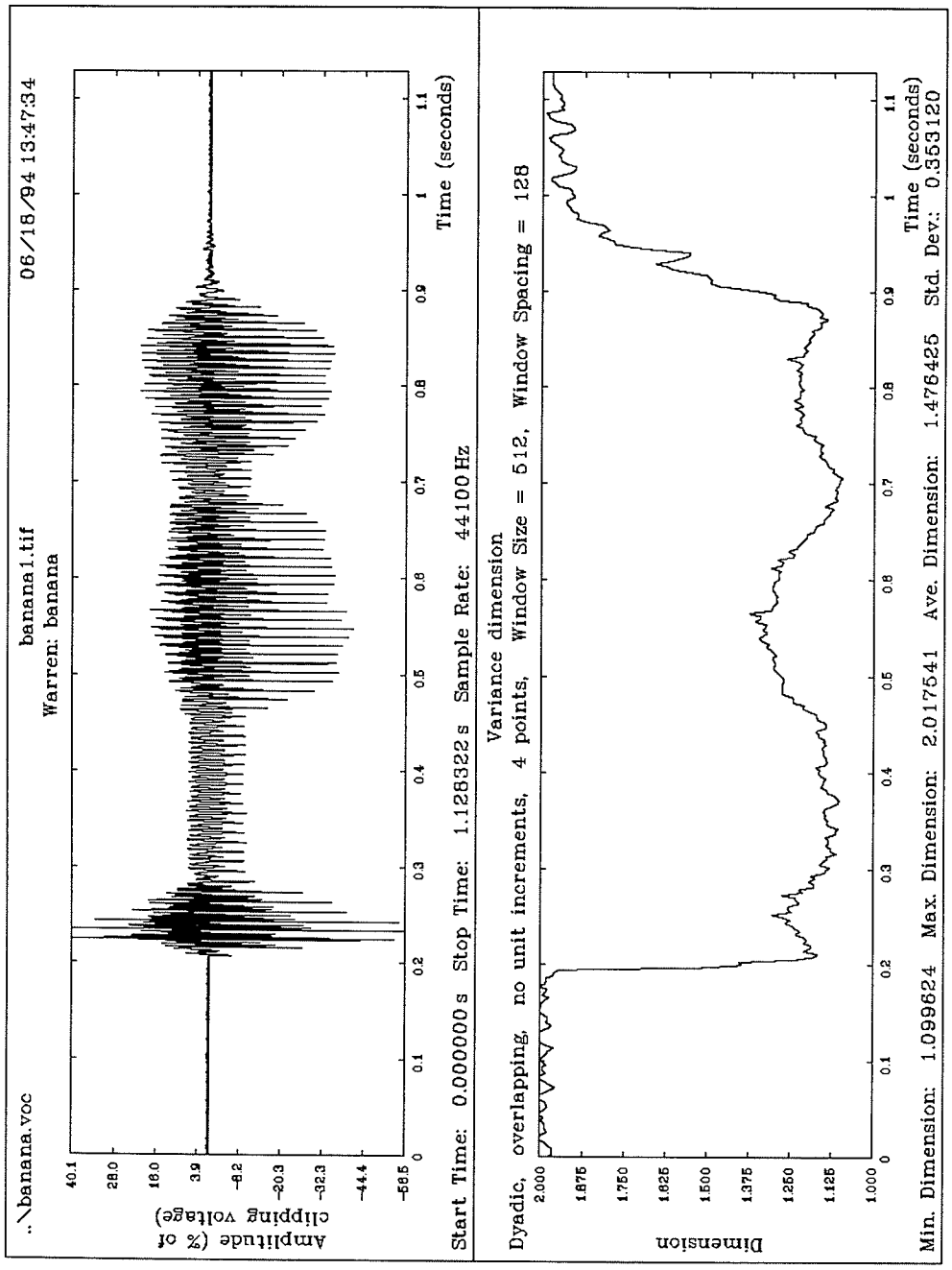
Poor, Noise Separation Parameters



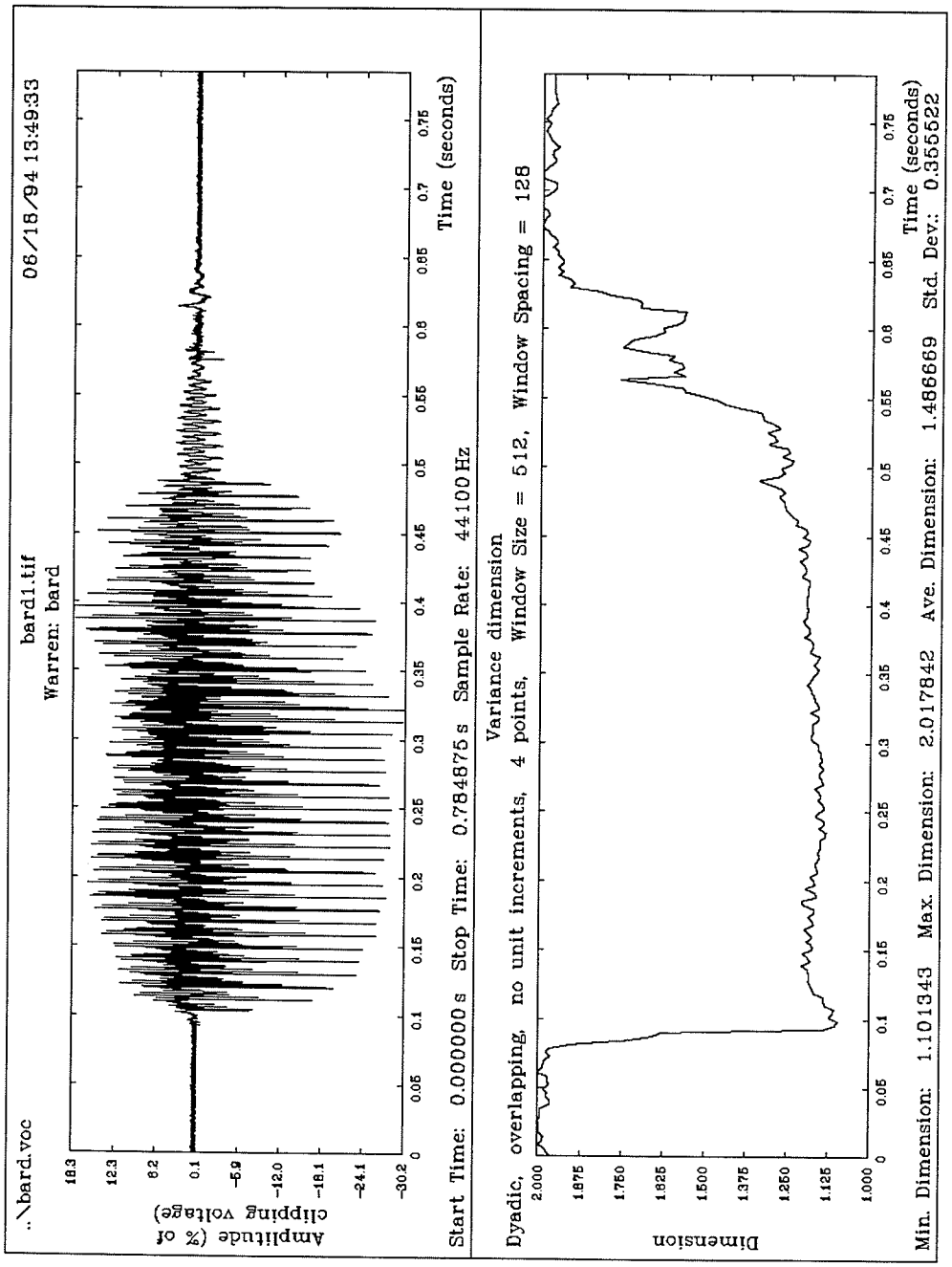
Bad, Fractal Amplification Parameters



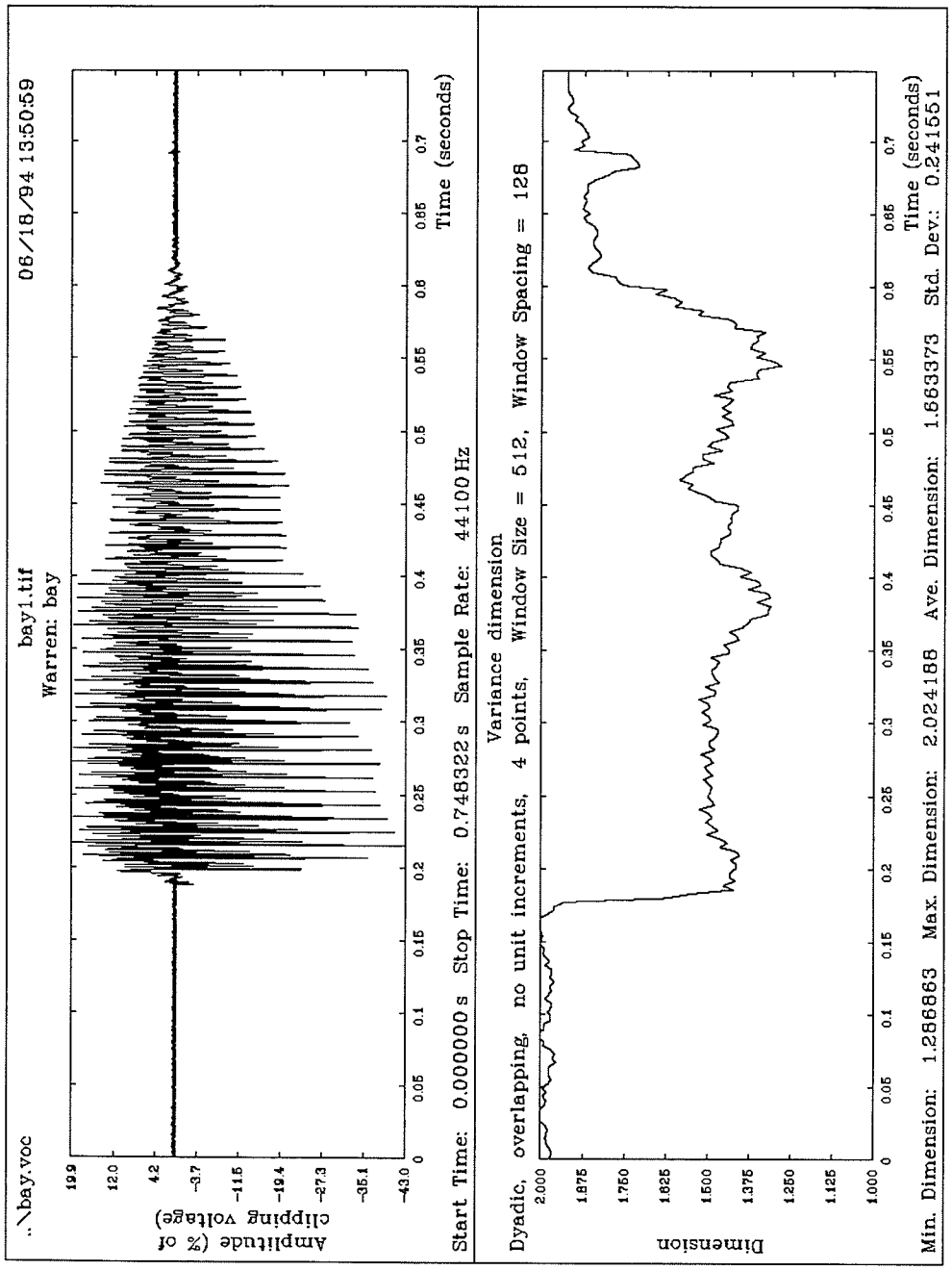
Banana, Fractal Amplification Parameters



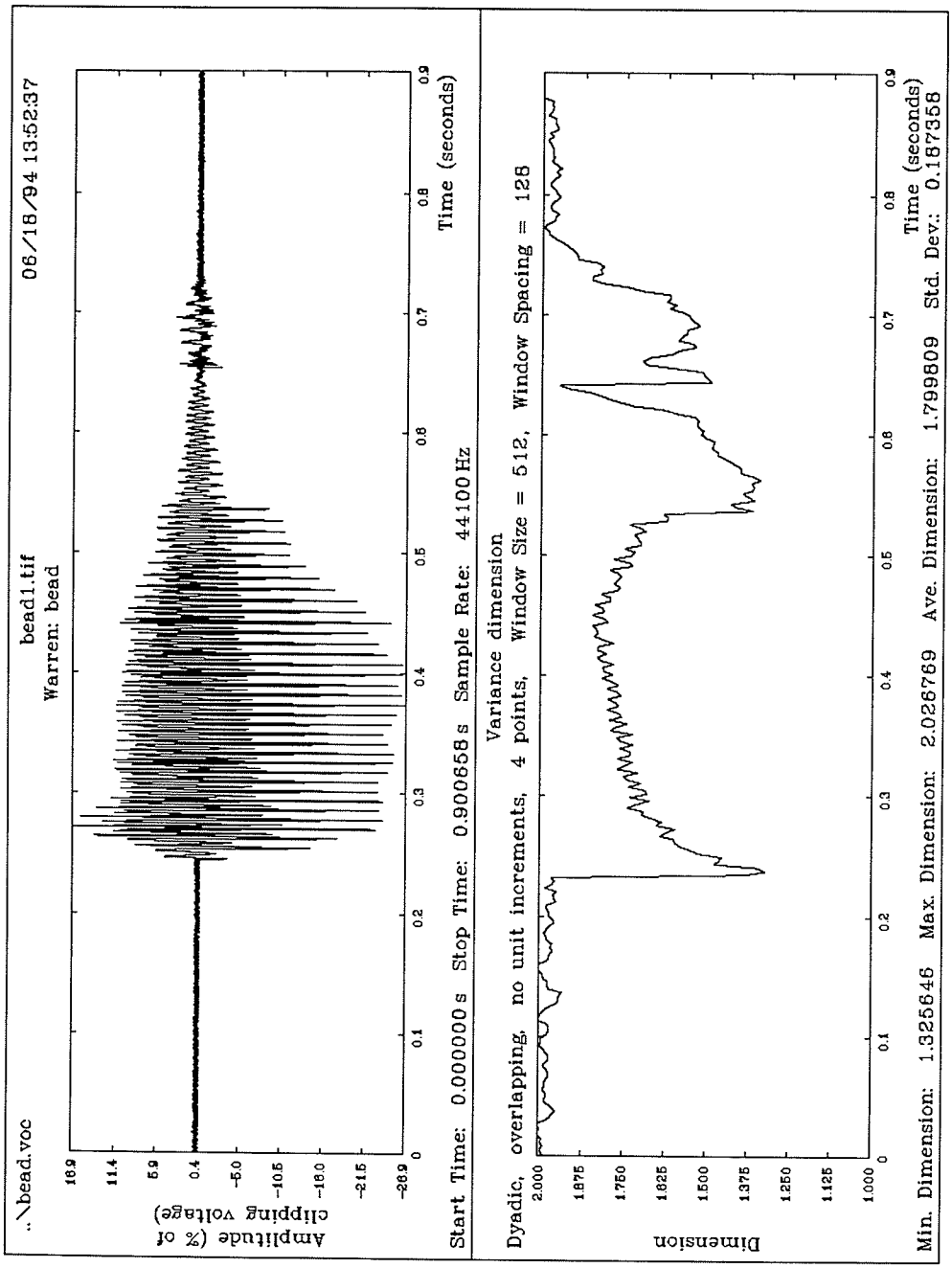
Bard, Fractal Amplification Parameters



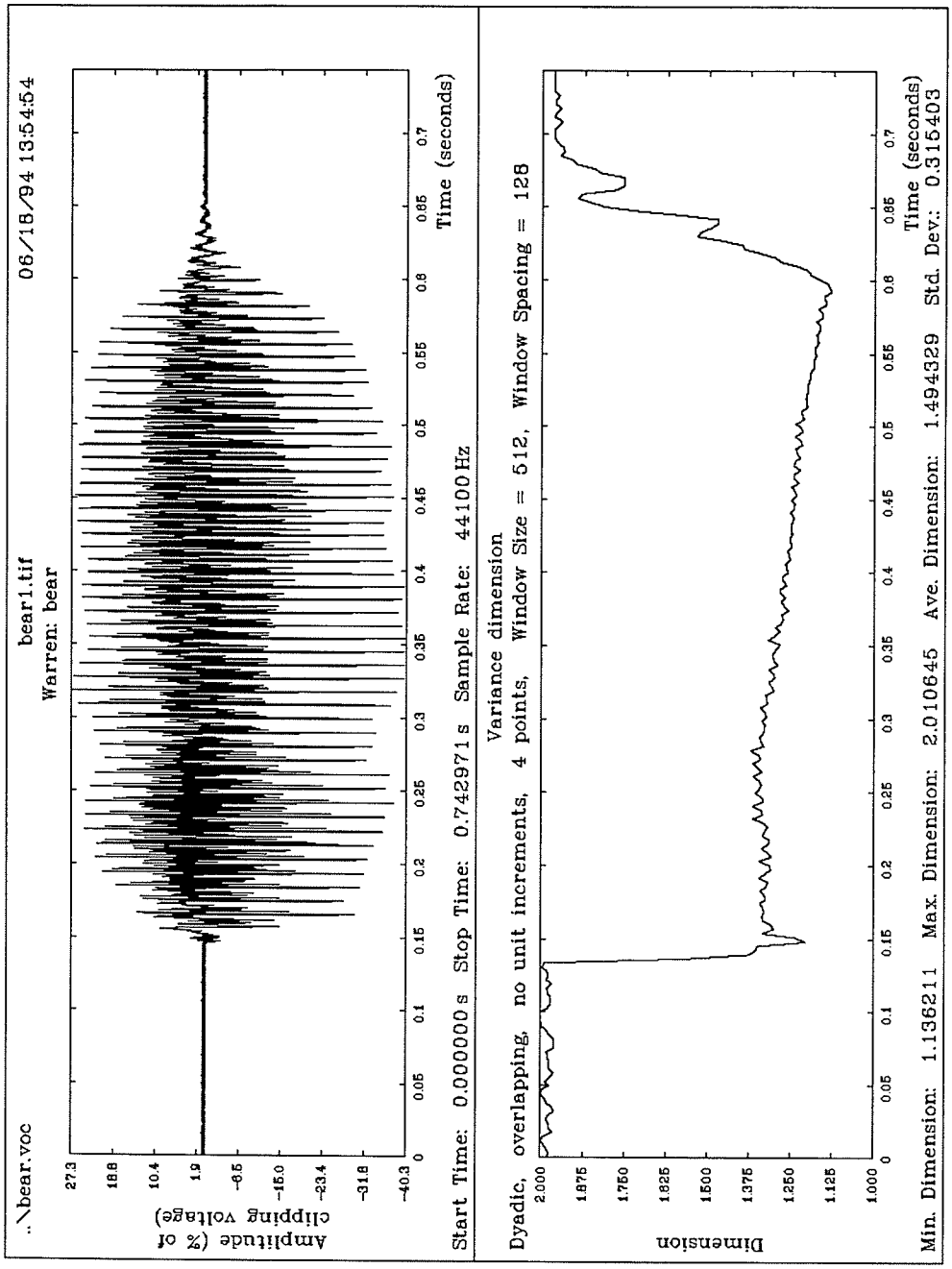
Bay, Fractal Amplification Parameters

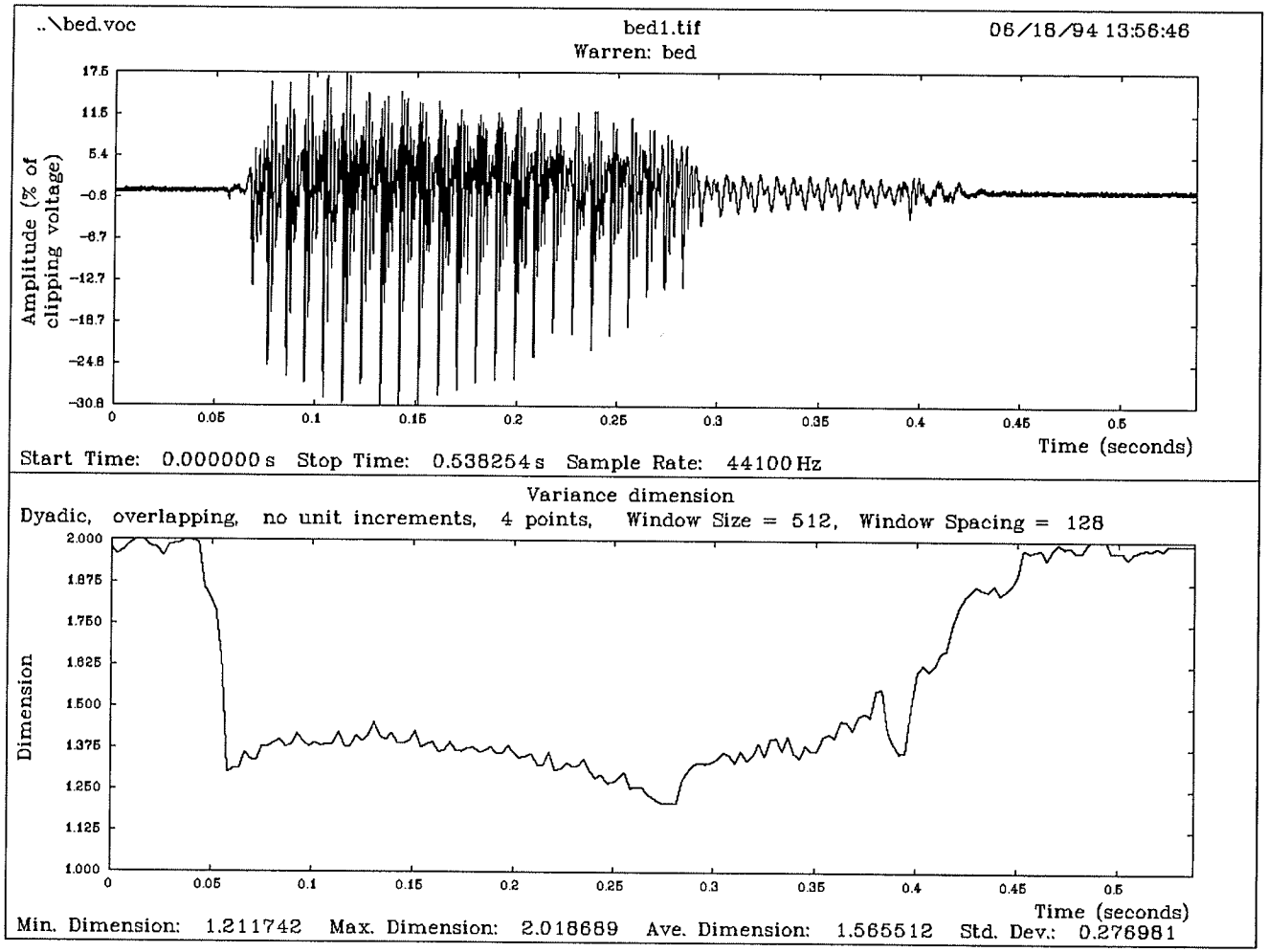


Bead, Fractal Amplification Parameters

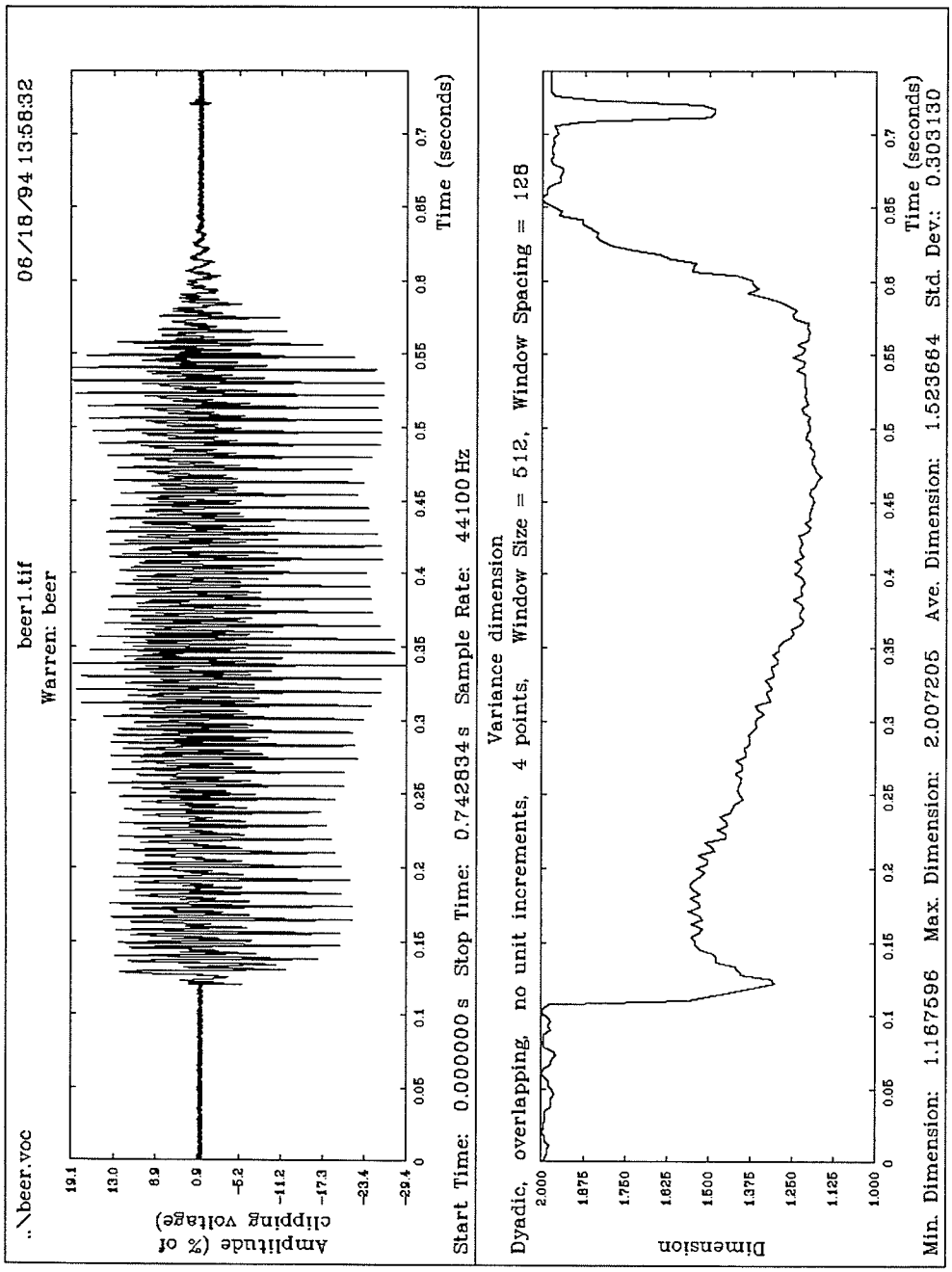


Bear, Fractal Amplification Parameters

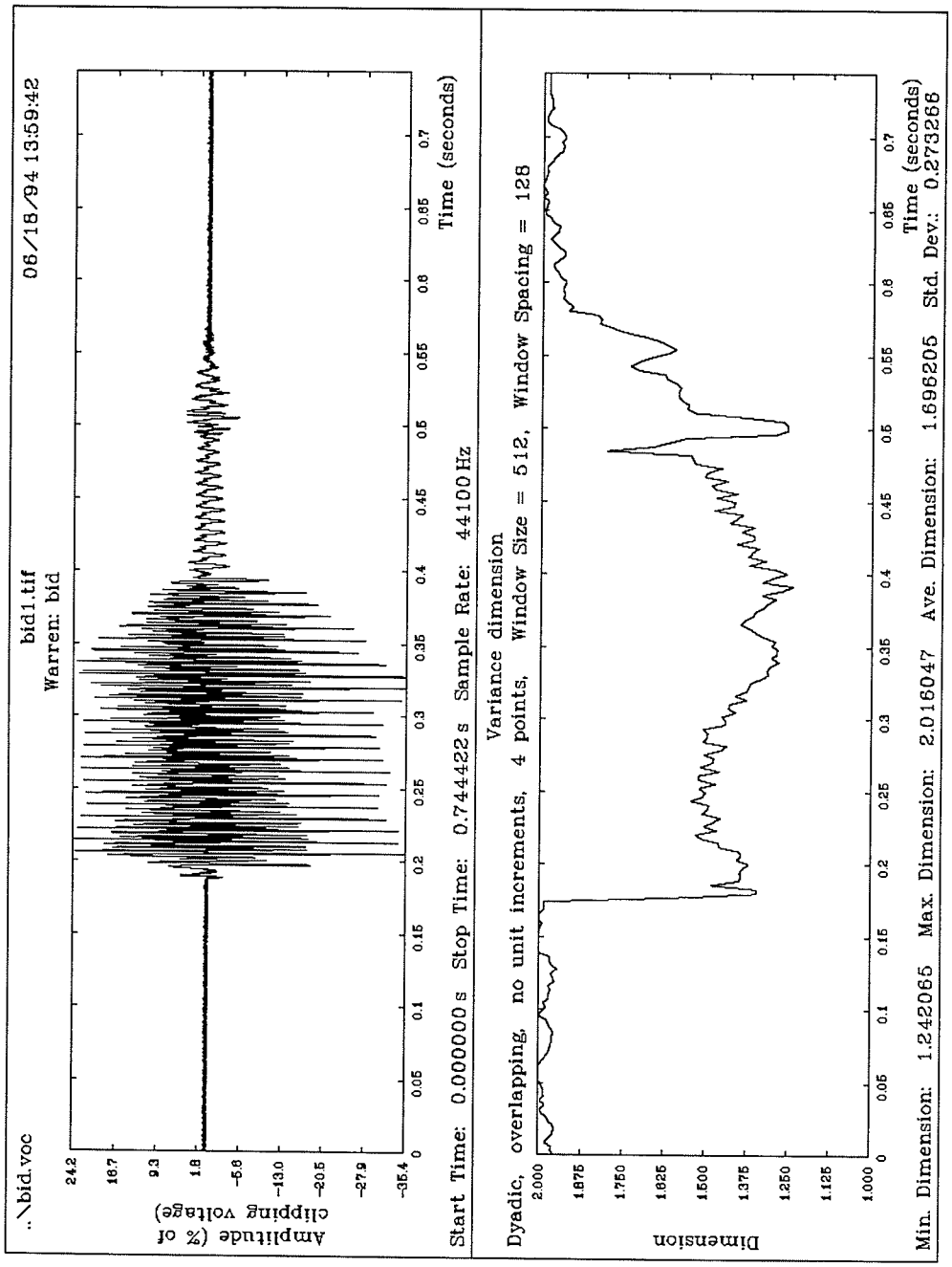



..\bed.voc
bed1.tif
06/18/94 13:56:46
Warren: bed
Amplitude (% of clipping voltage)
17.5
11.5
5.4
-0.6
-6.7
-12.7
-18.7
-24.8
-30.8
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Time (seconds)
Start Time: 0.000000 s Stop Time: 0.538254 s Sample Rate: 44100 Hz
Variance dimension
Dyadic, overlapping, no unit increments, 4 points, Window Size = 512, Window Spacing = 128
Dimension
2.000
1.875
1.750
1.625
1.500
1.375
1.250
1.125
1.000
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Time (seconds)
Min. Dimension: 1.211742 Max. Dimension: 2.018689 Ave. Dimension: 1.565512 Std. Dev.: 0.276981

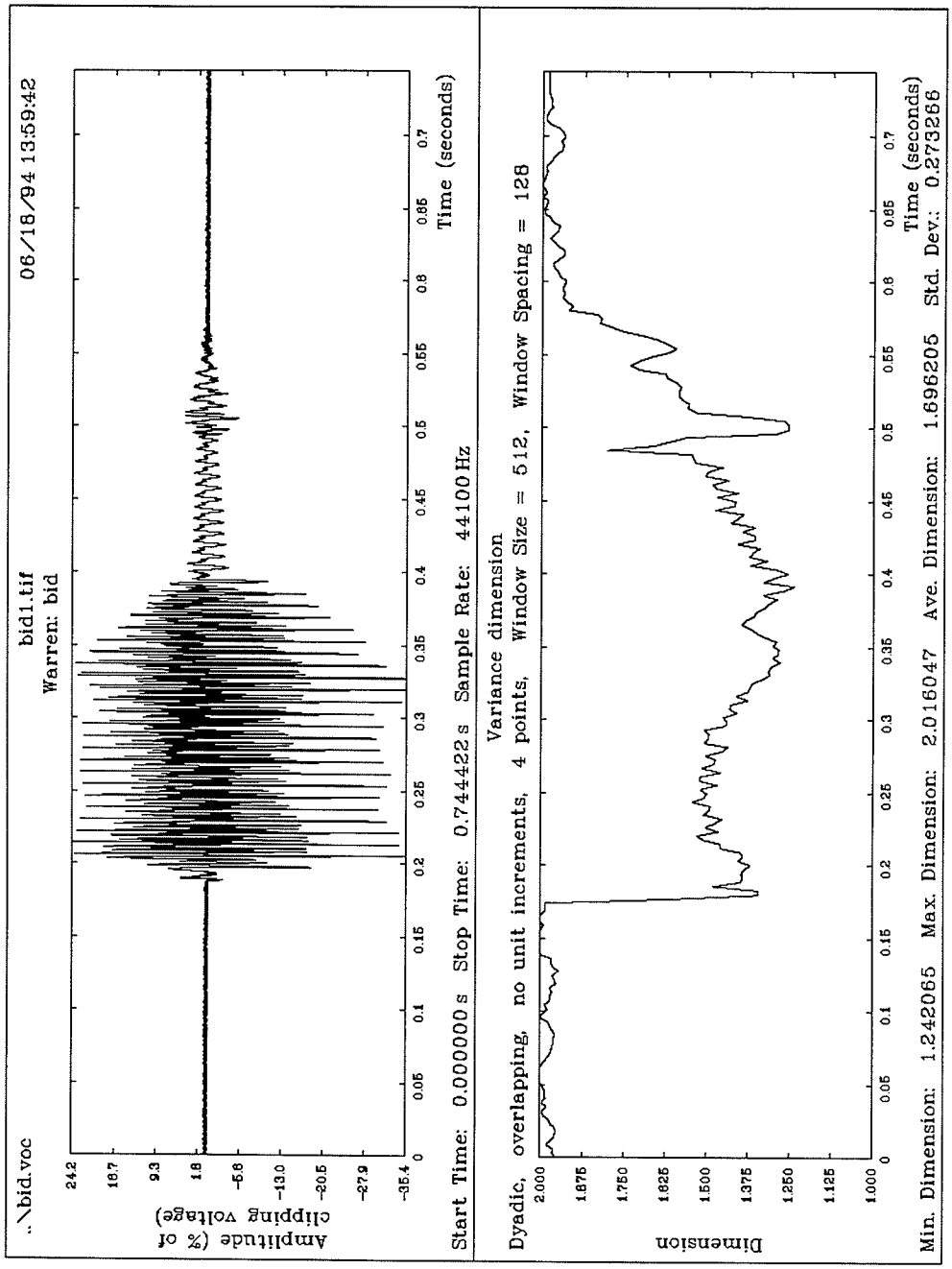
Beer, Fractal Amplification Parameters



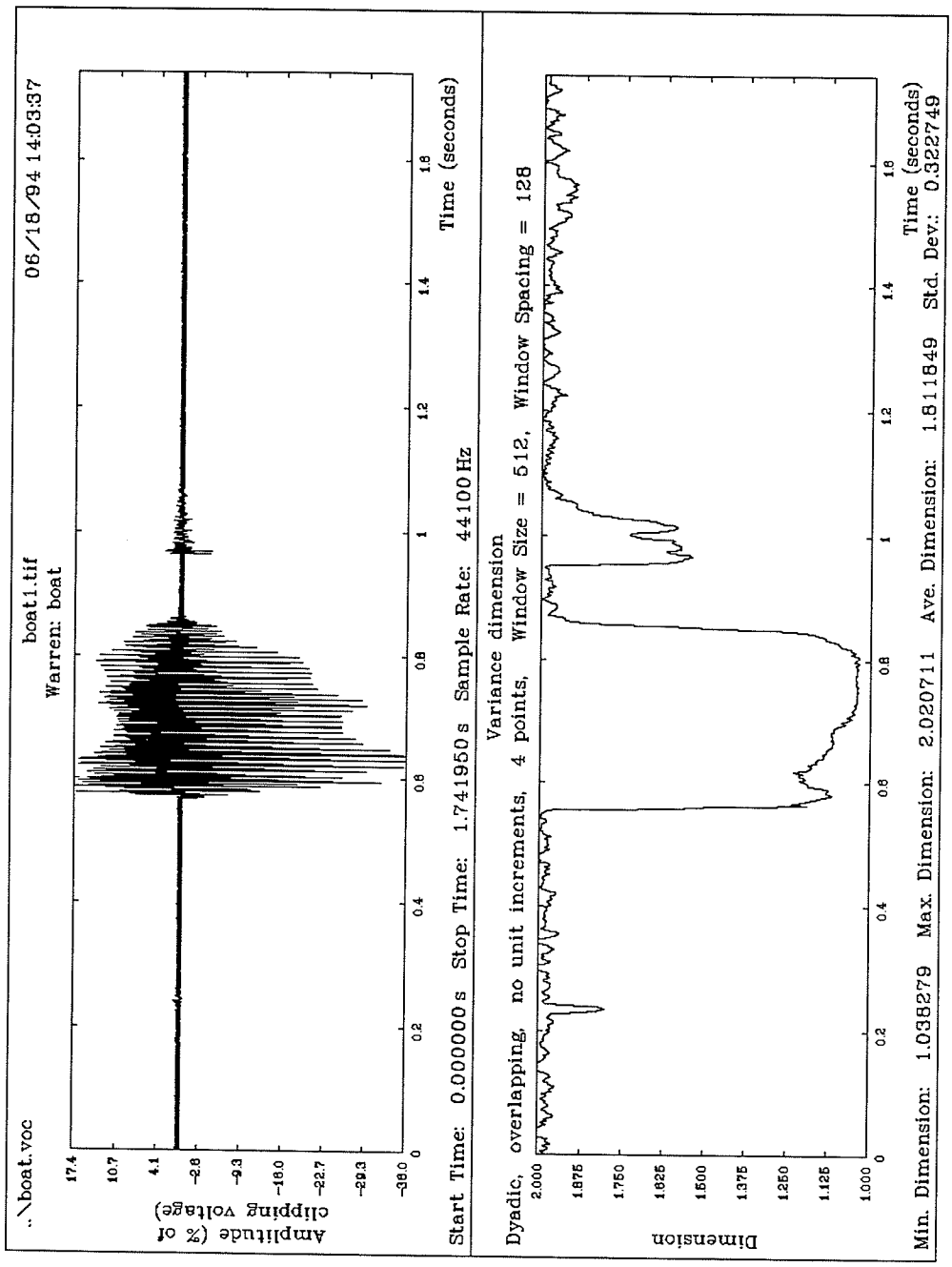
Bid, Fractal Amplification Parameters



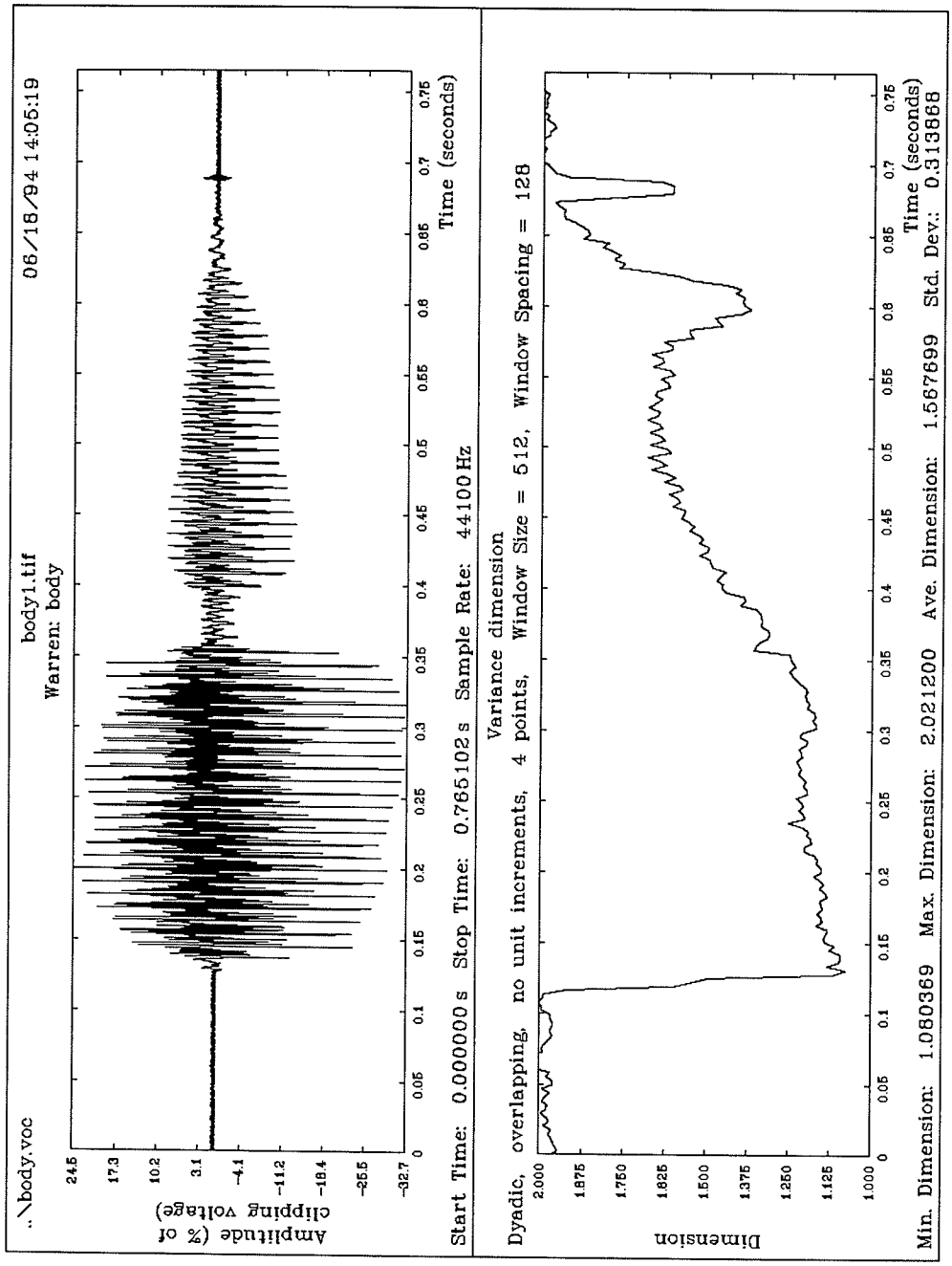
Bird, Fractal Amplification Parameters



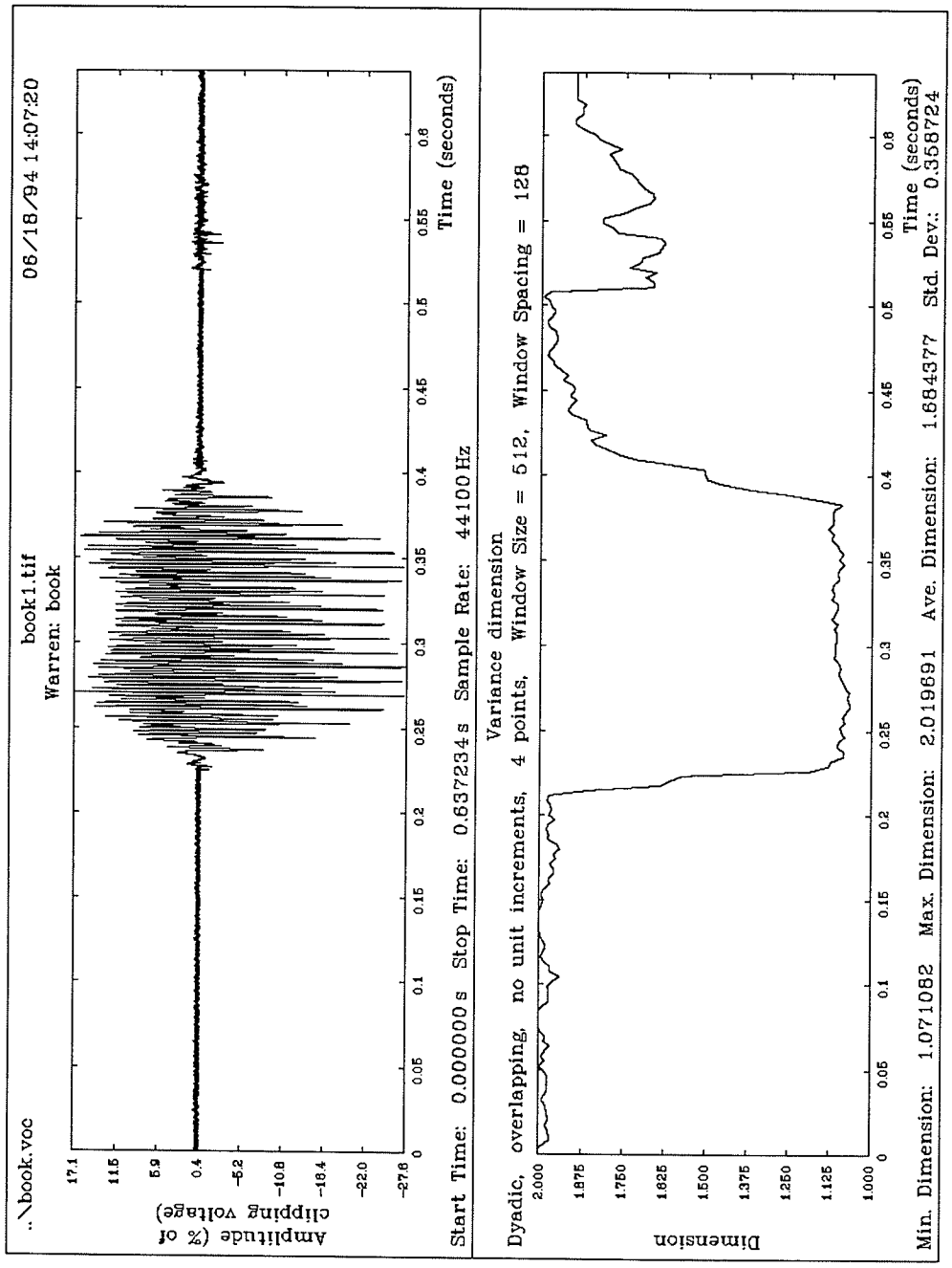
Boat, Fractal Amplification Parameters



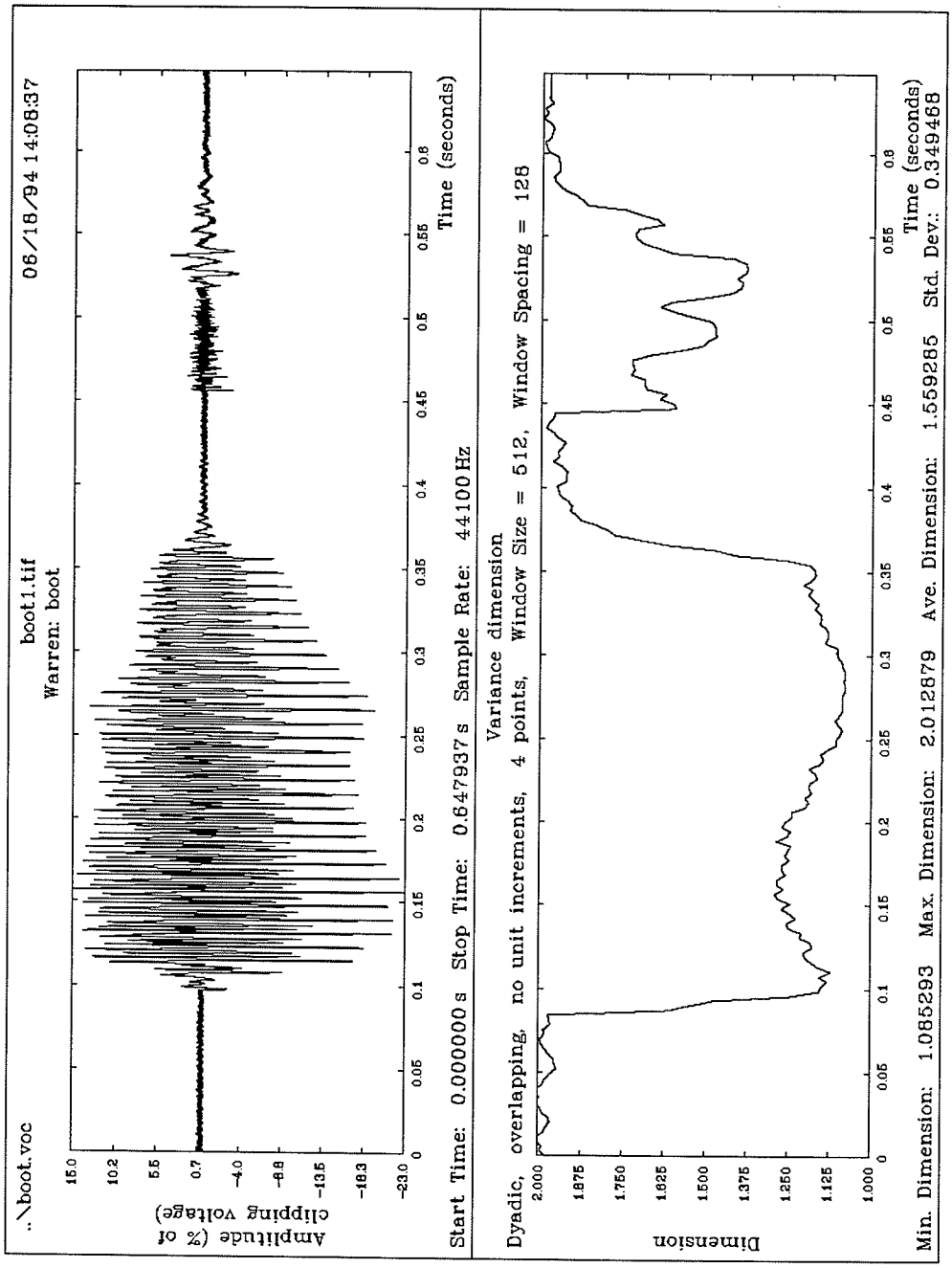
Body, Fractal Amplification Parameters



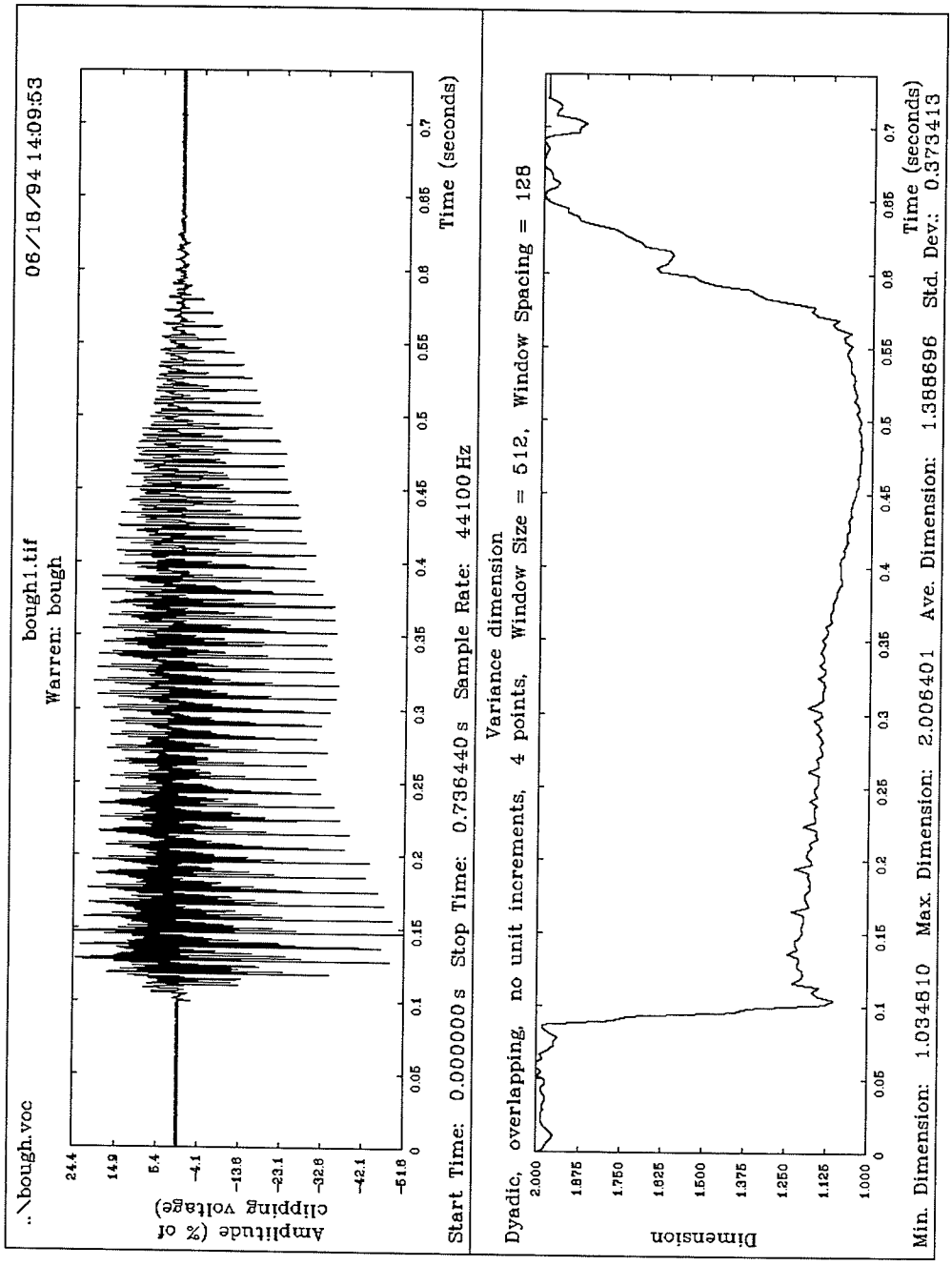
Book, Fractal Amplification Parameters



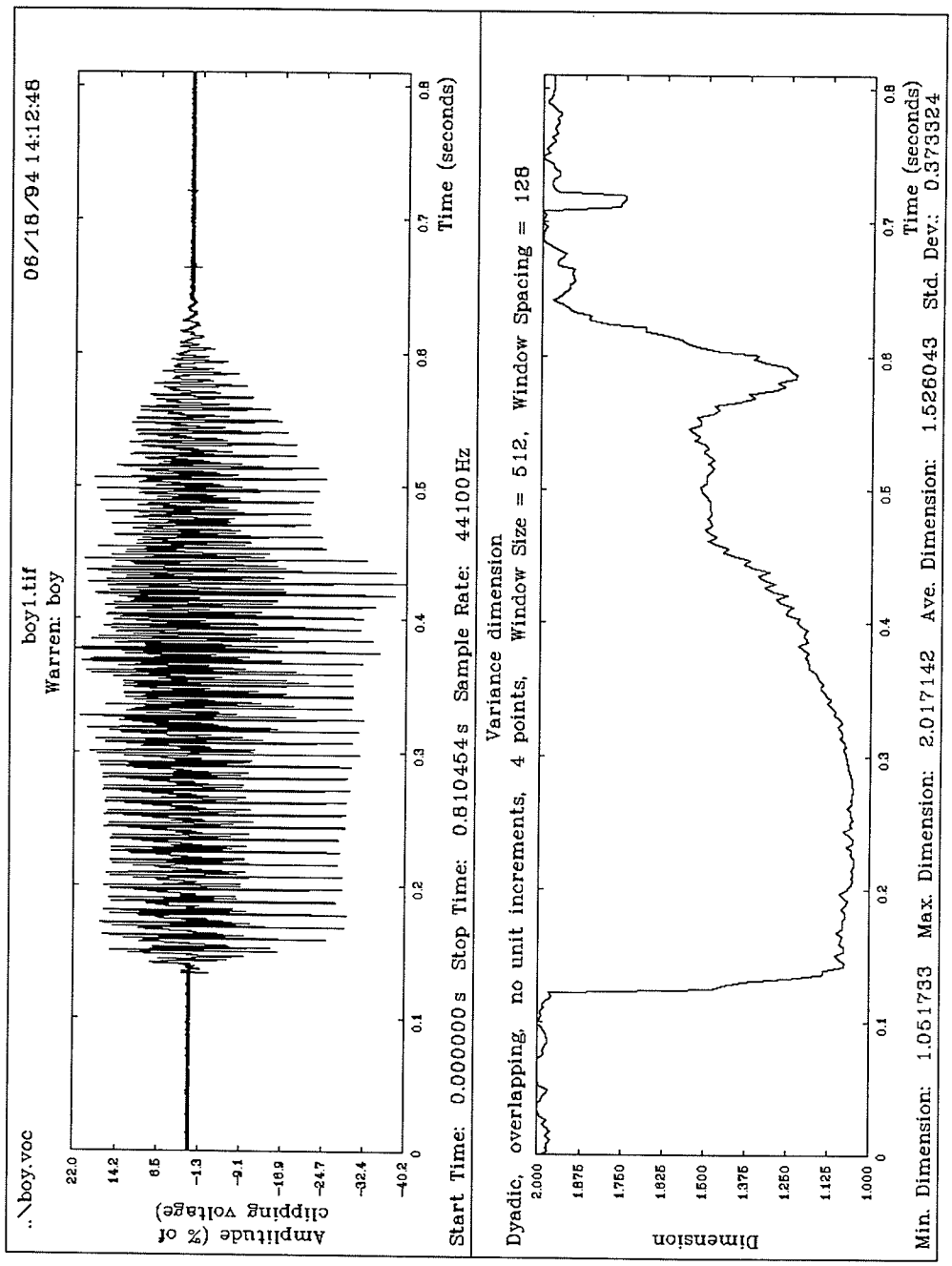
Boot, Fractal Amplification Parameters



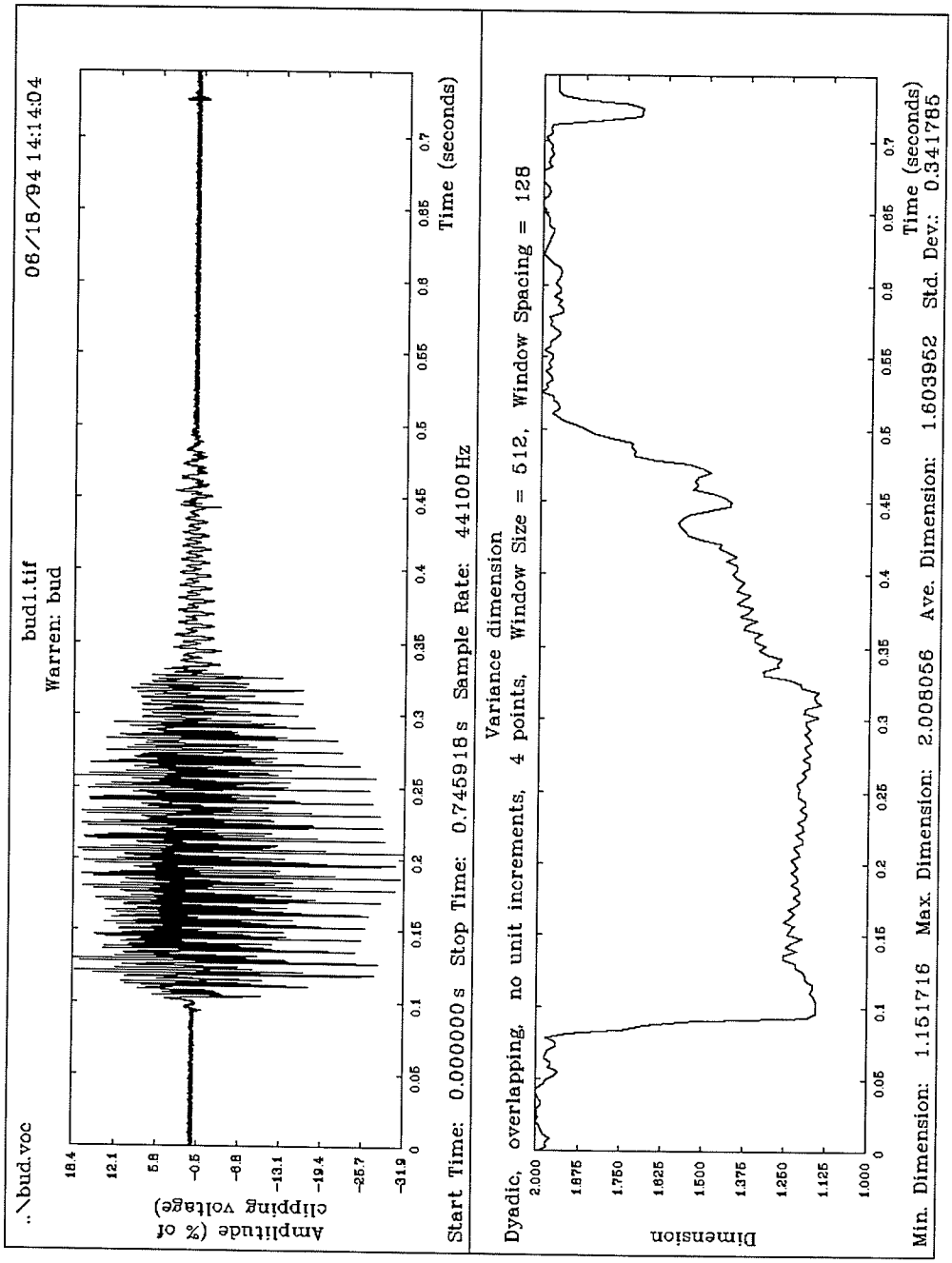
Bough, Fractal Amplification Parameters



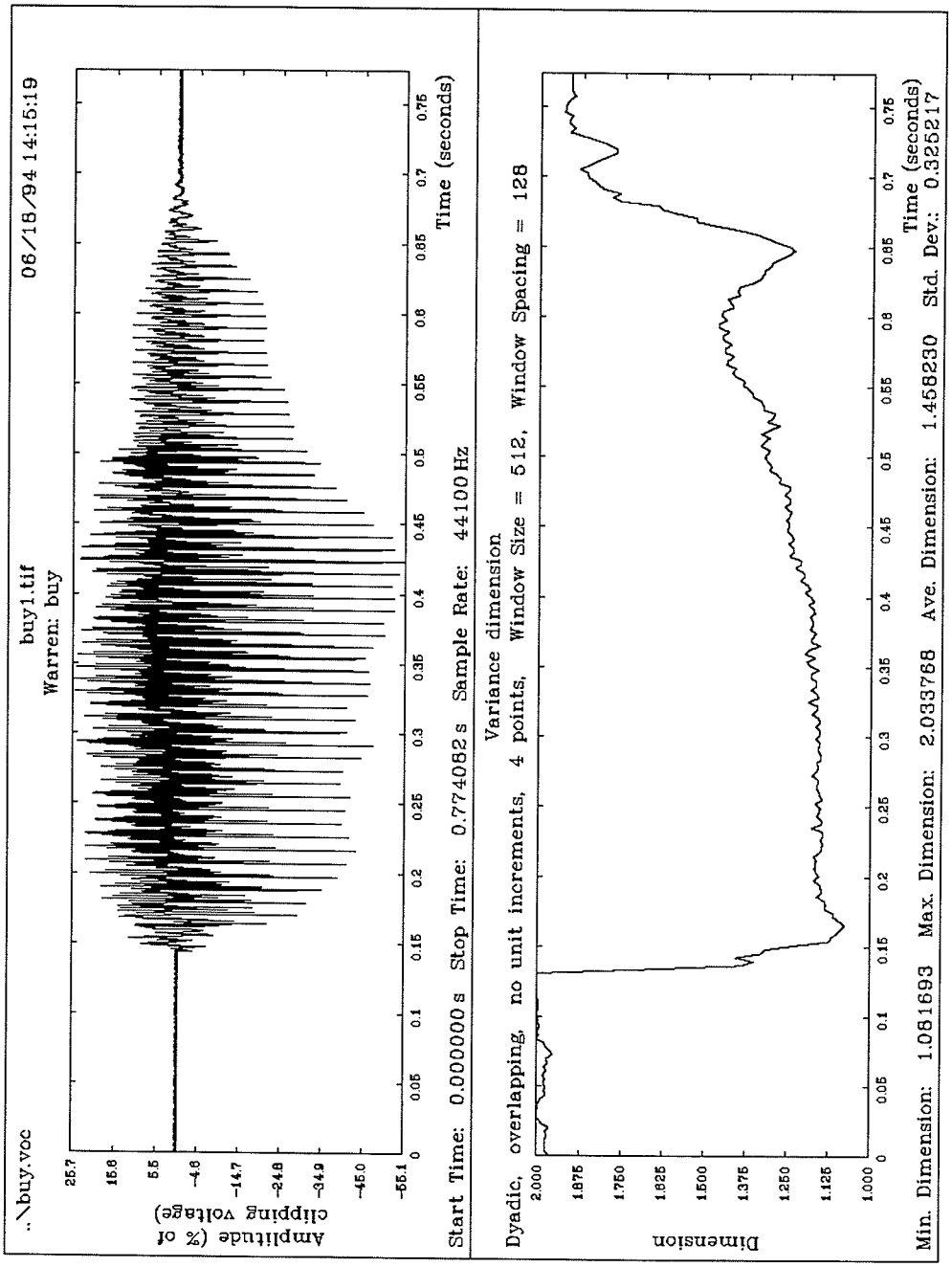
Boy, Fractal Amplification Parameters



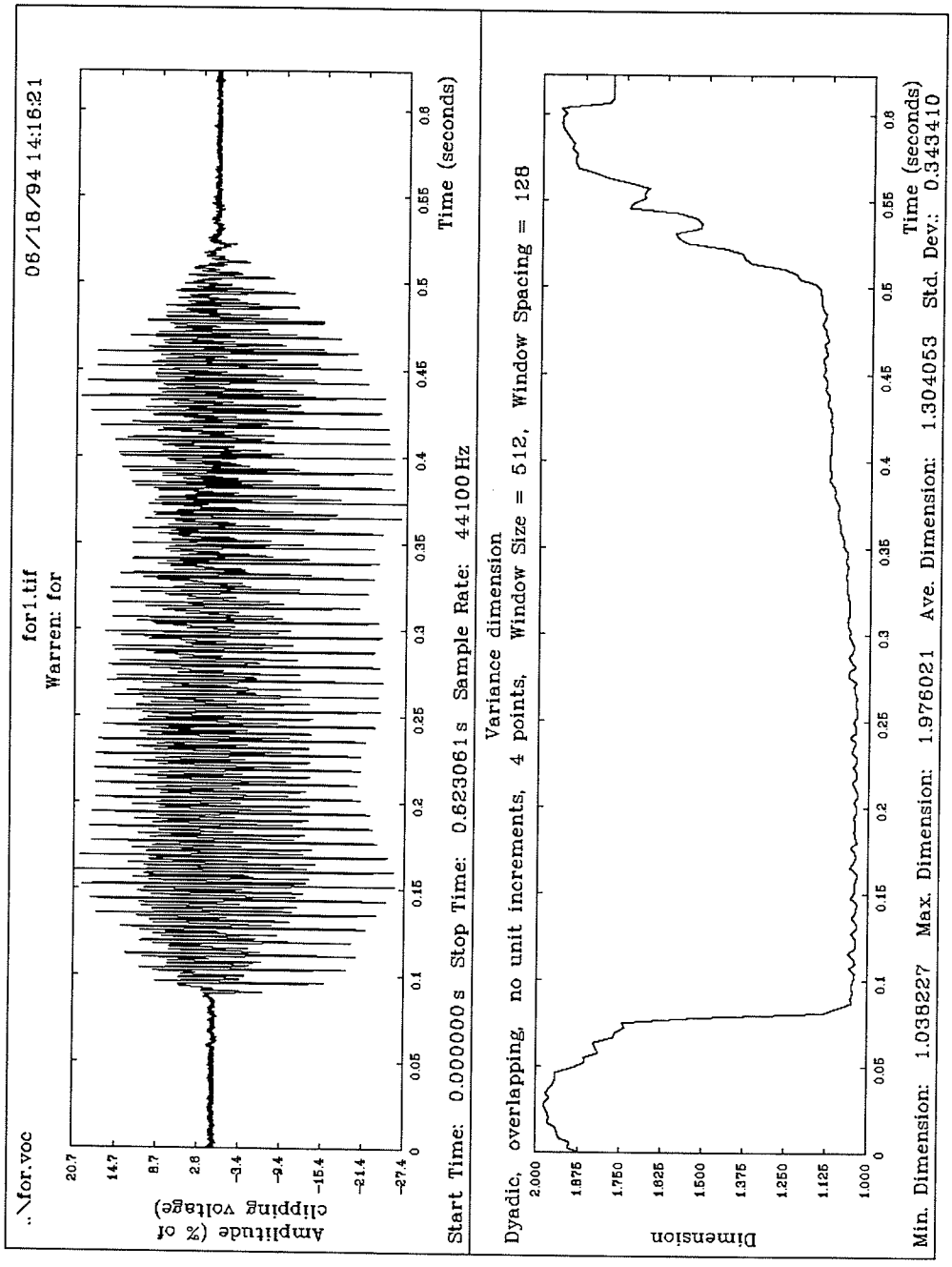
Bud, Fractal Amplification Parameters



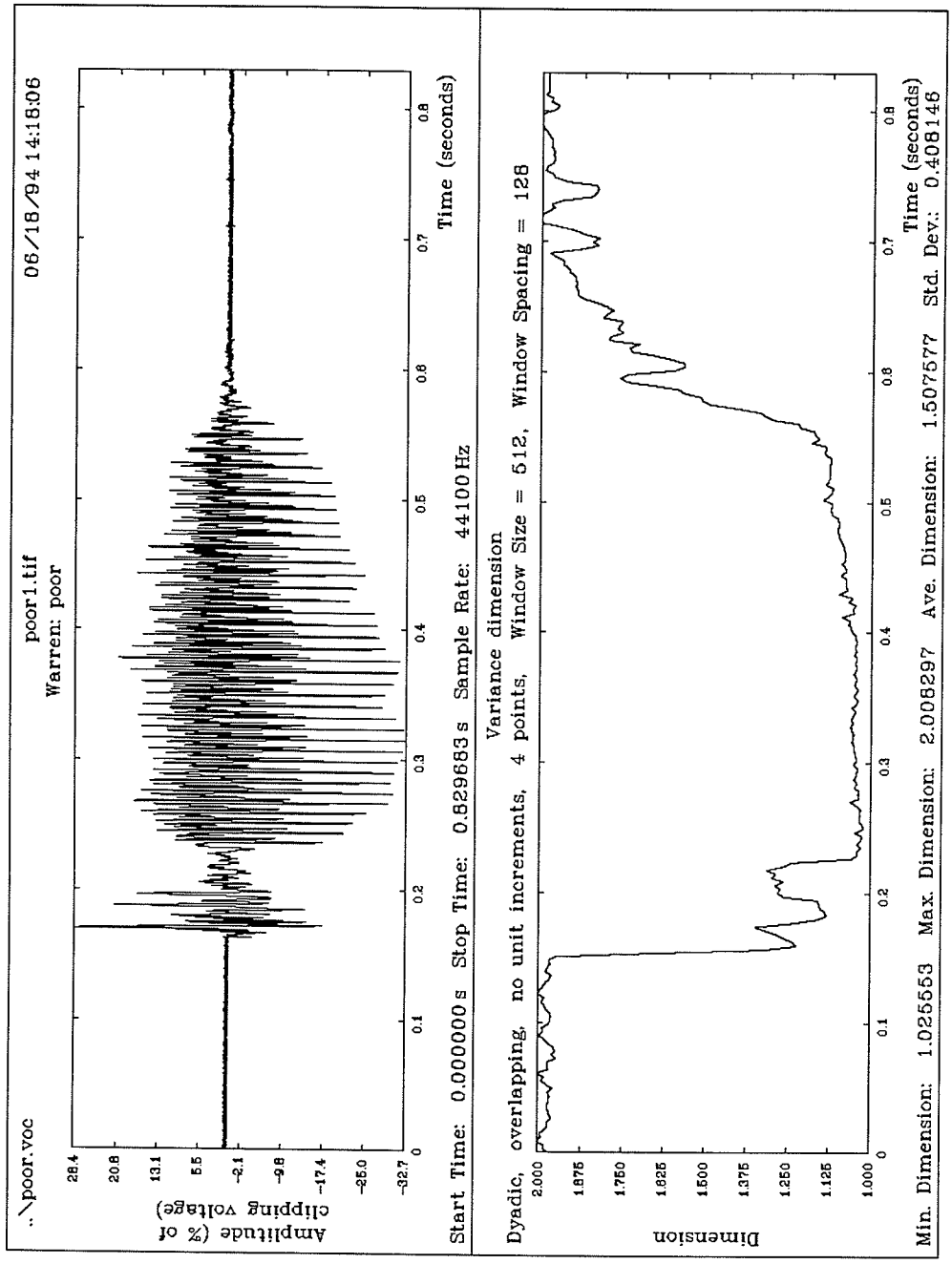
Buy, Fractal Amplification Parameters



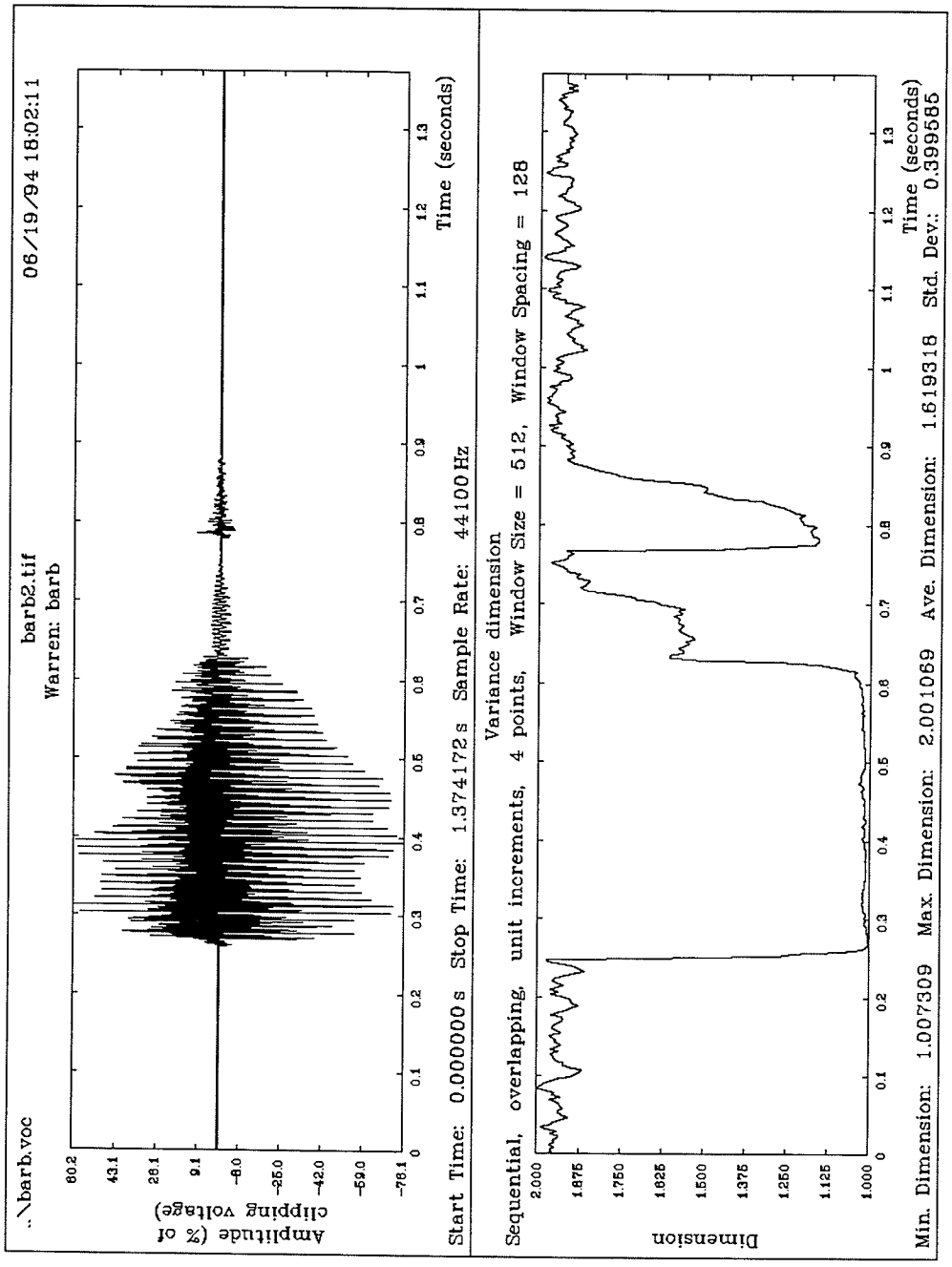
For, Fractal Amplification Parameters



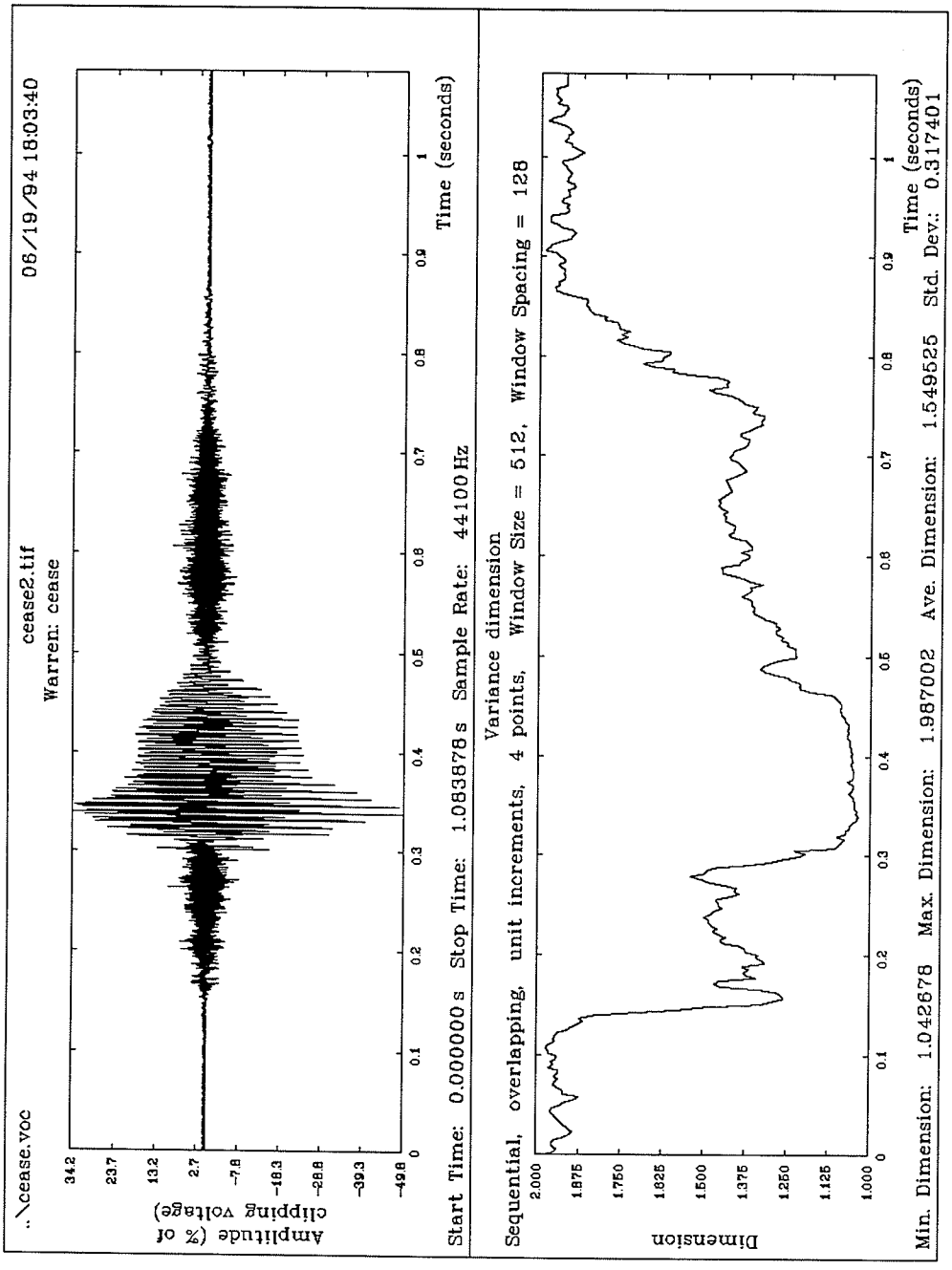
Poor, Fractal Amplification Parameters



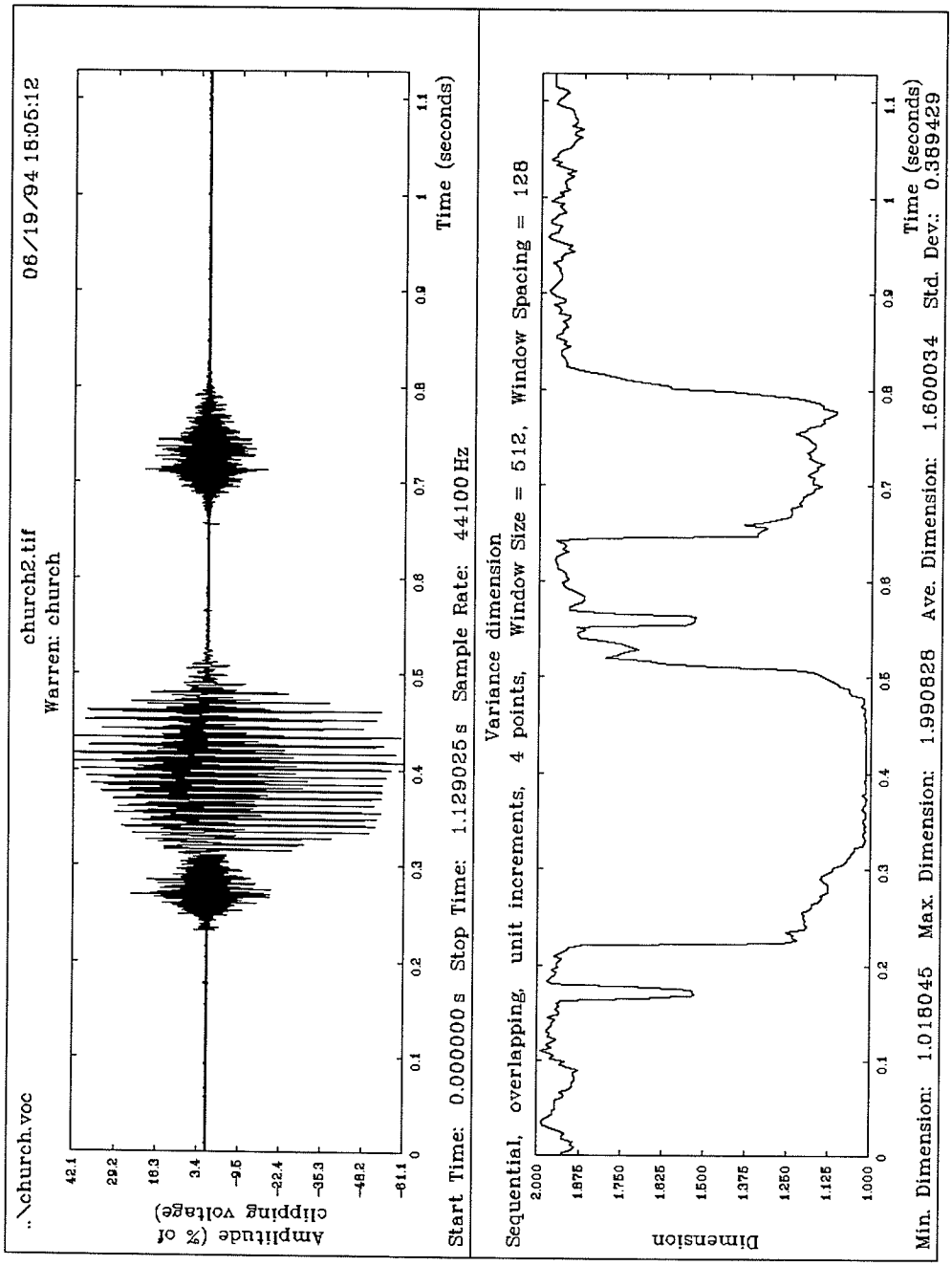
Barb, Noise Separation Parameters



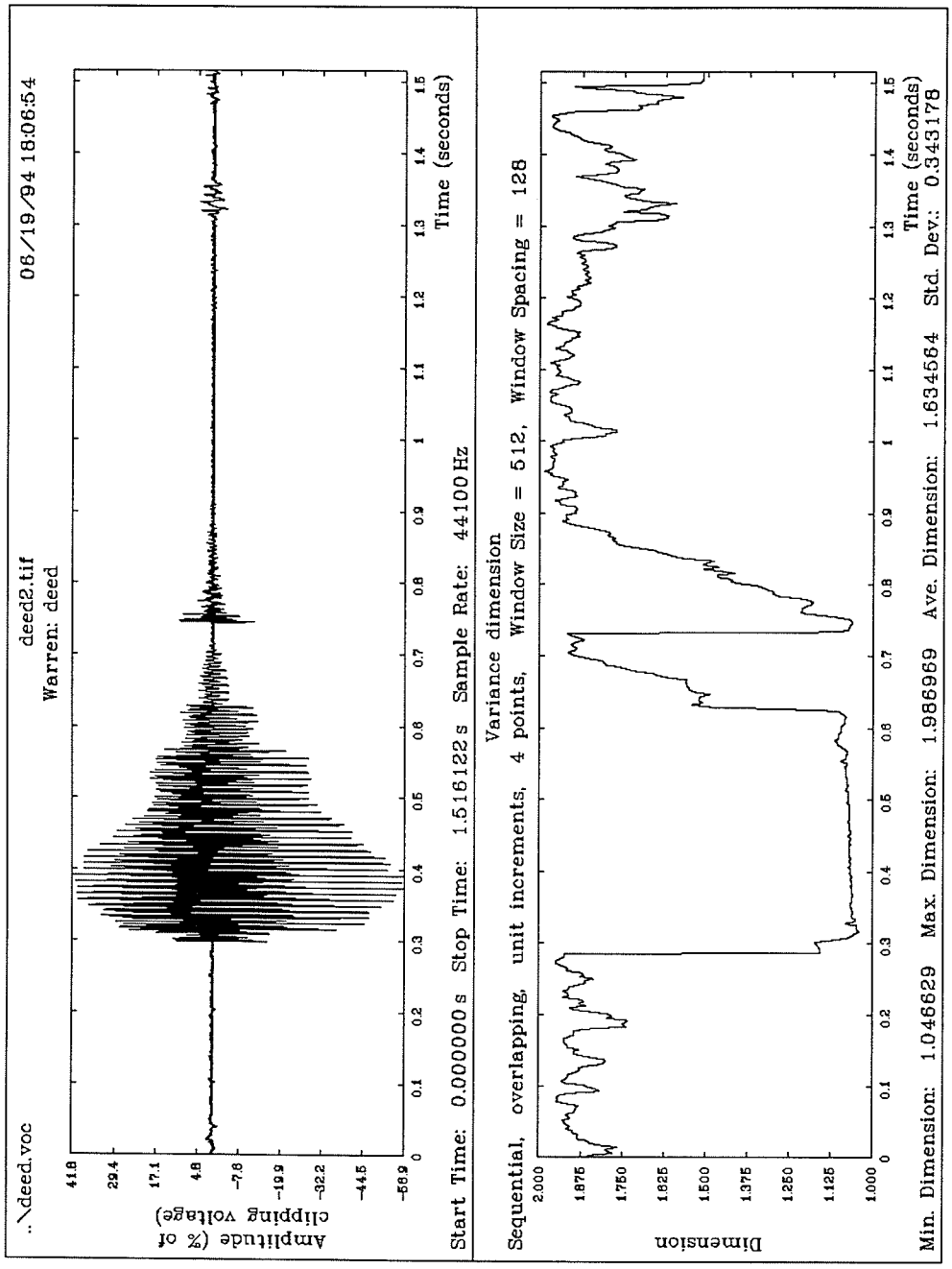
Cease, Noise Separation Parameters



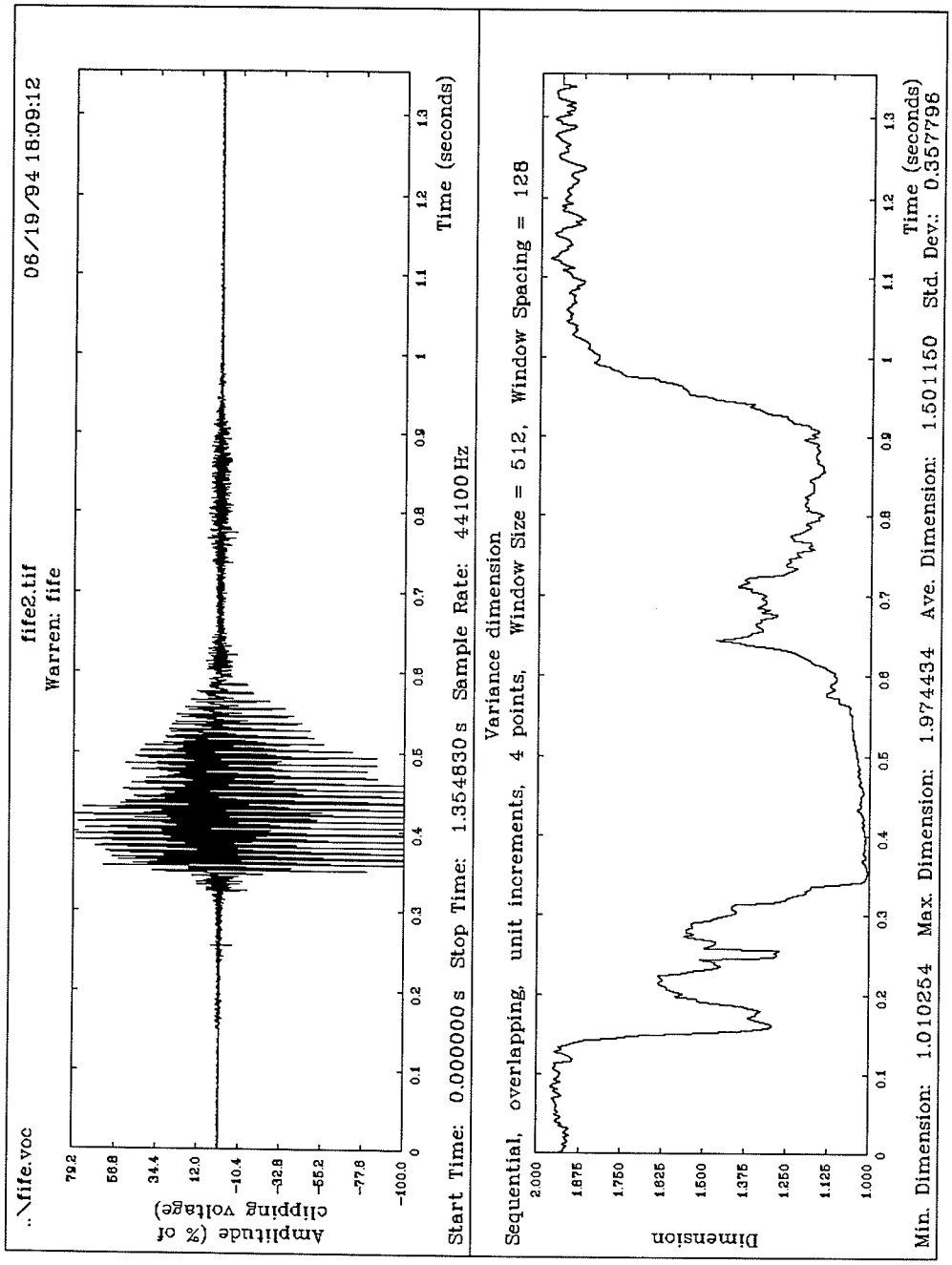
Church, Noise Separation Parameters



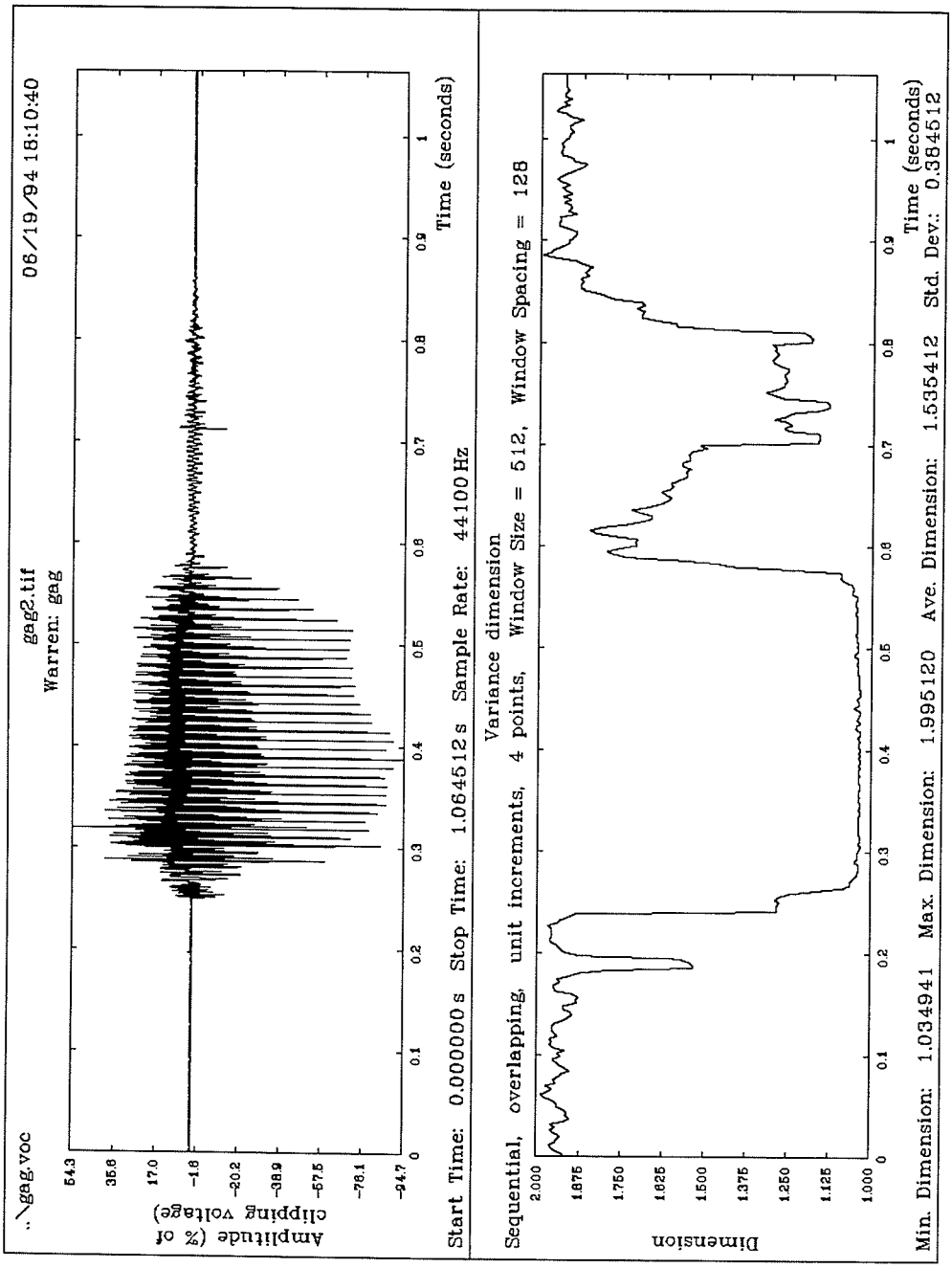
Deed, Noise Separation Parameters



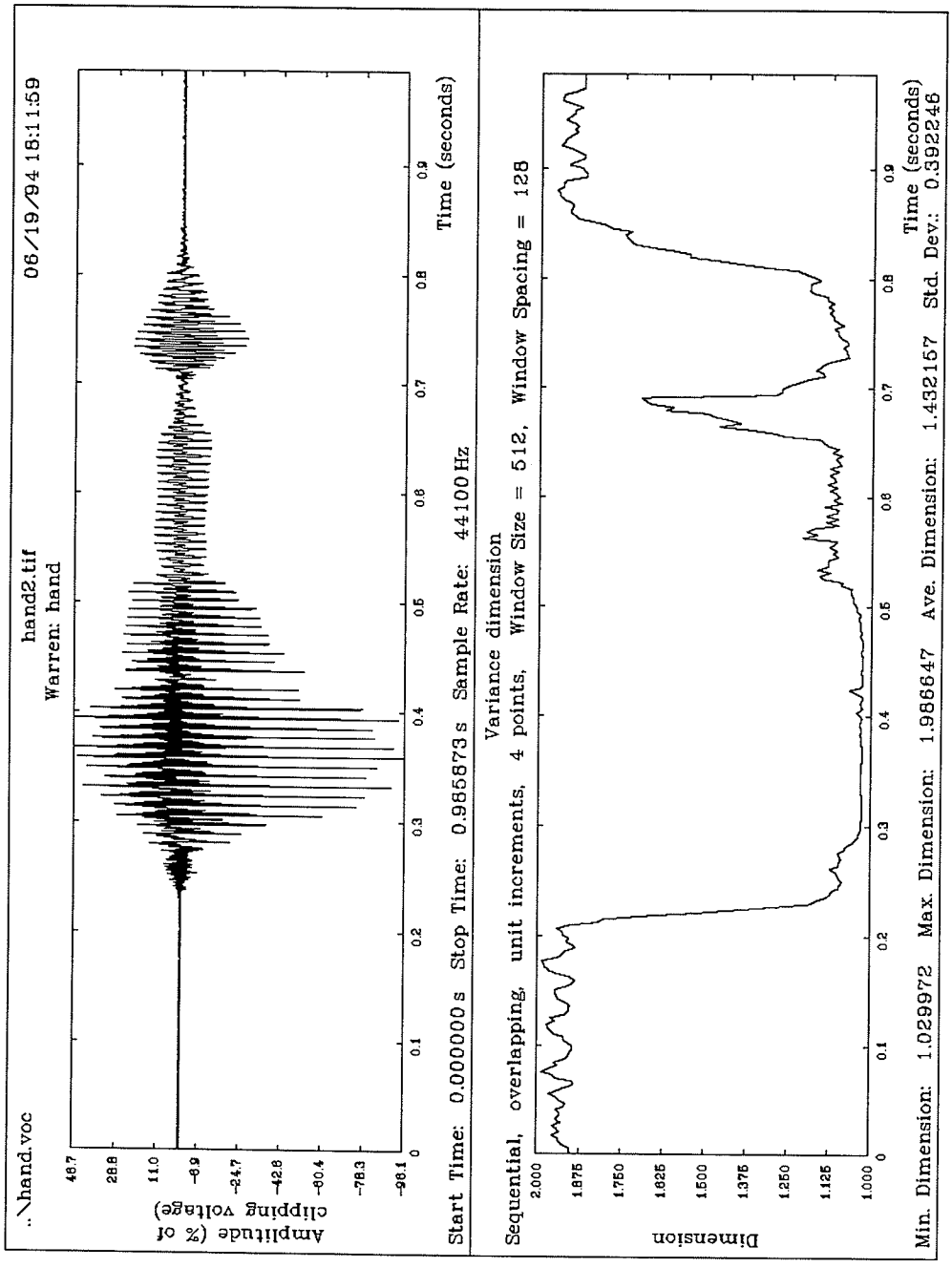
Fife, Noise Separation Parameters



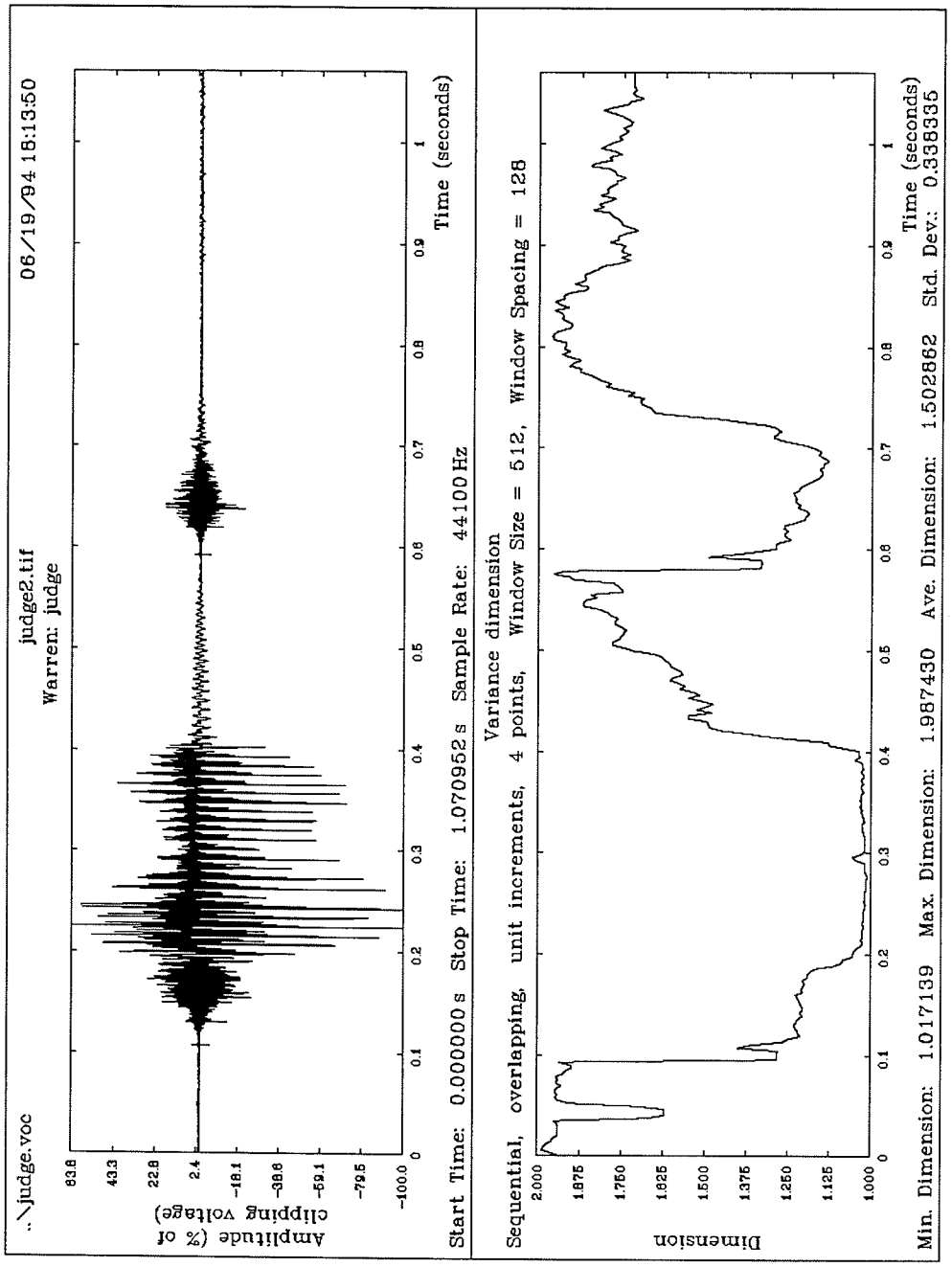
Gag, Noise Separation Parameters



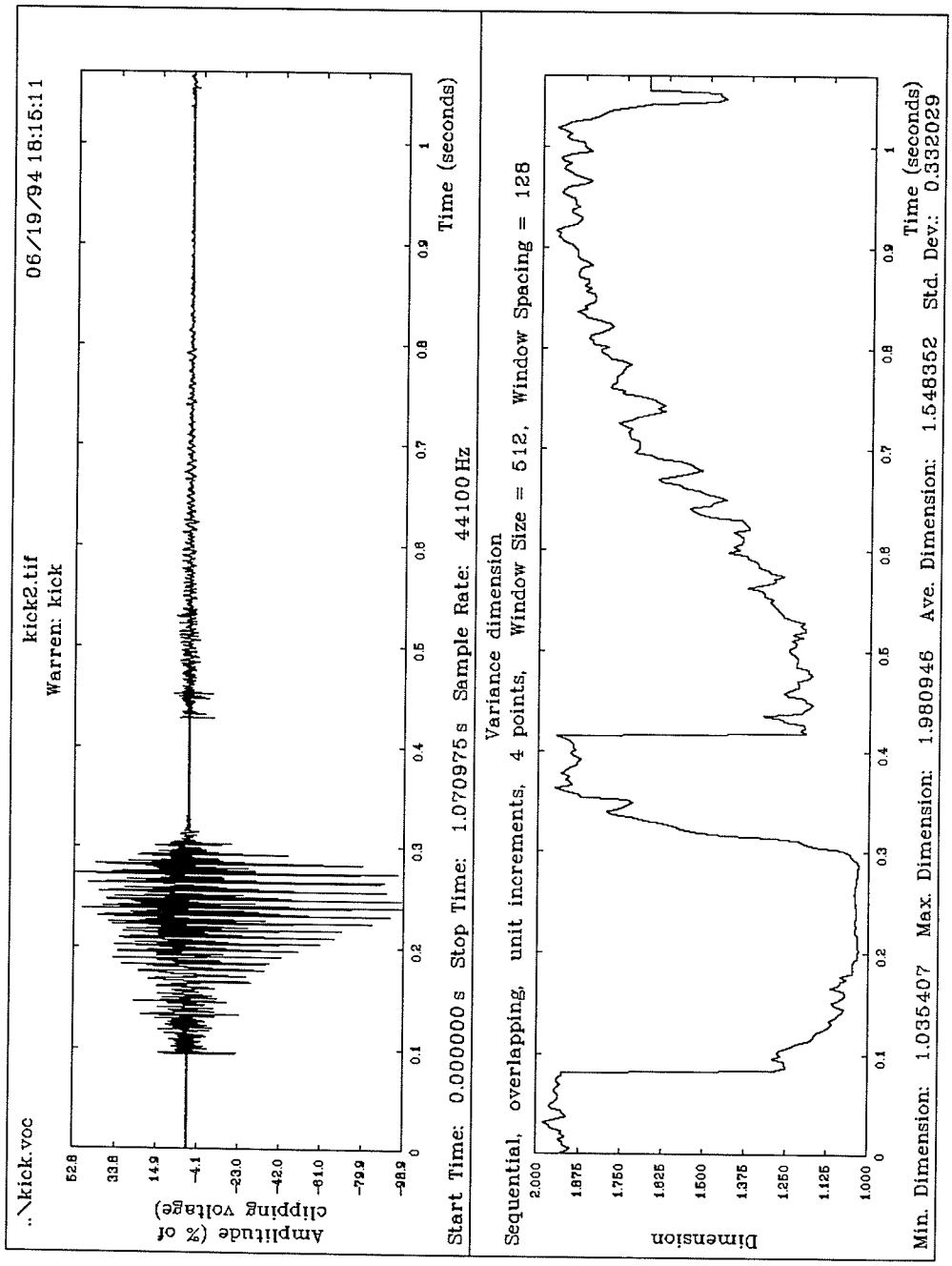
Hand, Noise Separation Parameters



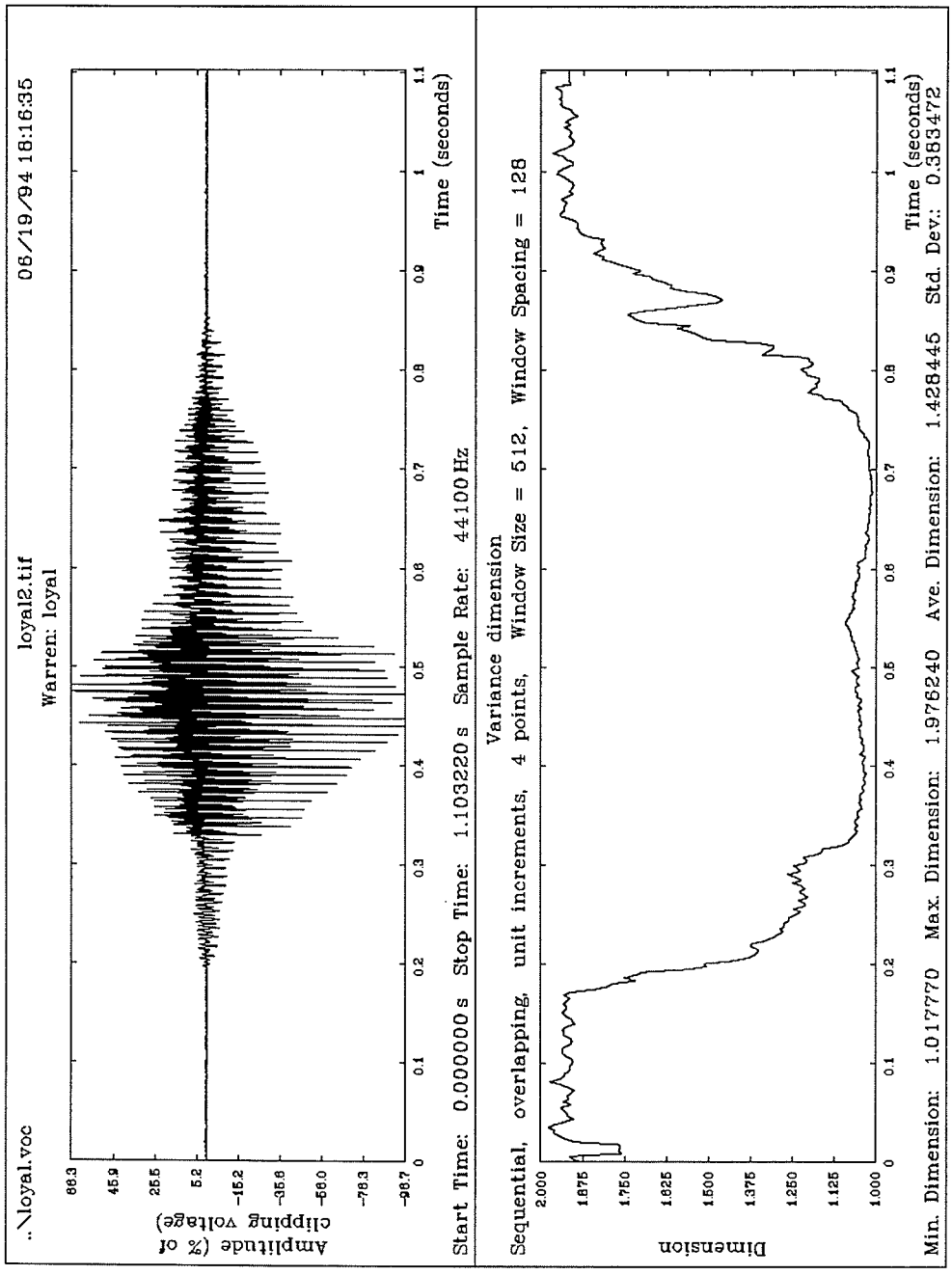
Judge, Noise Separation Parameters



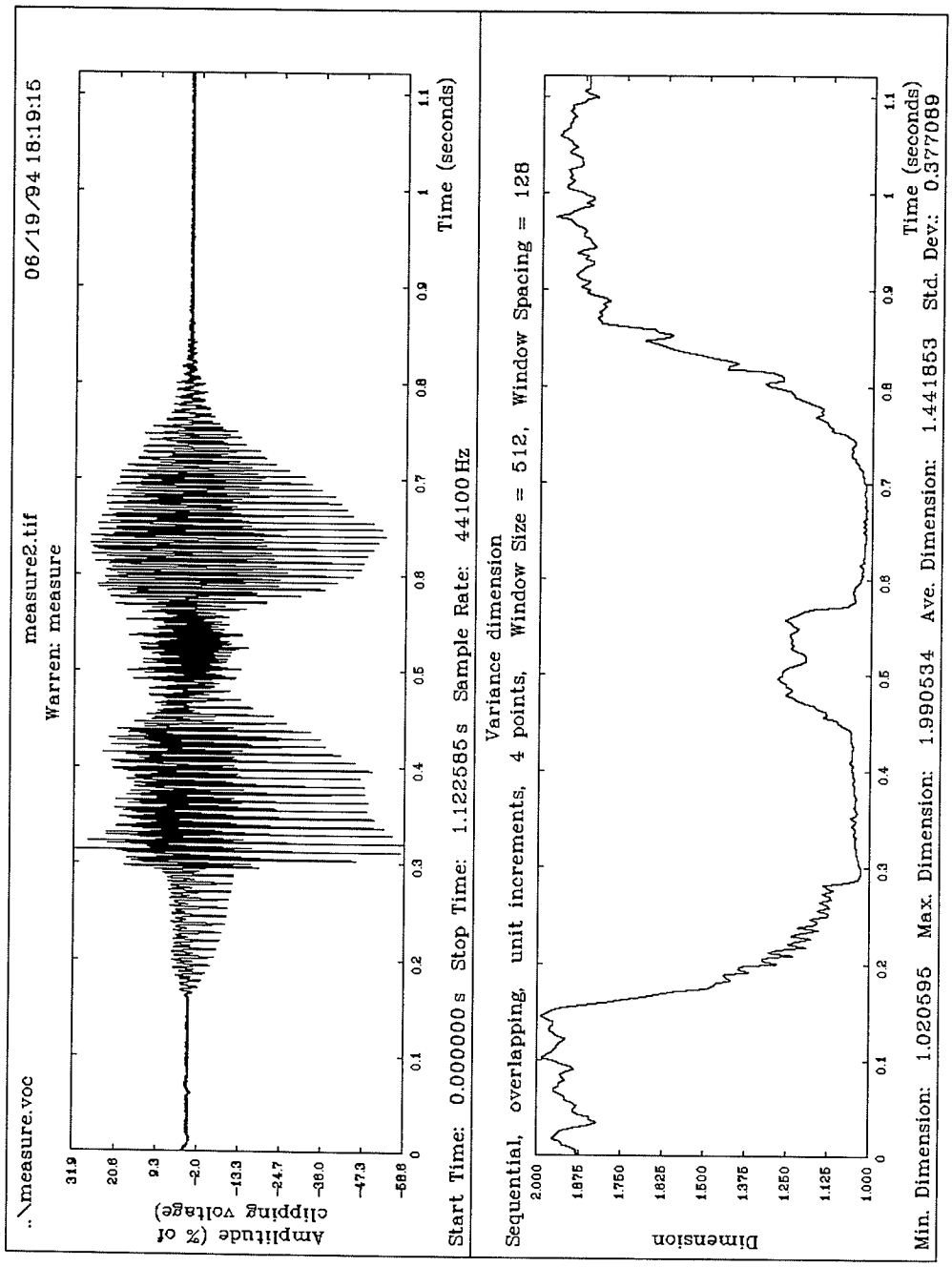
Kick, Noise Separation Parameters



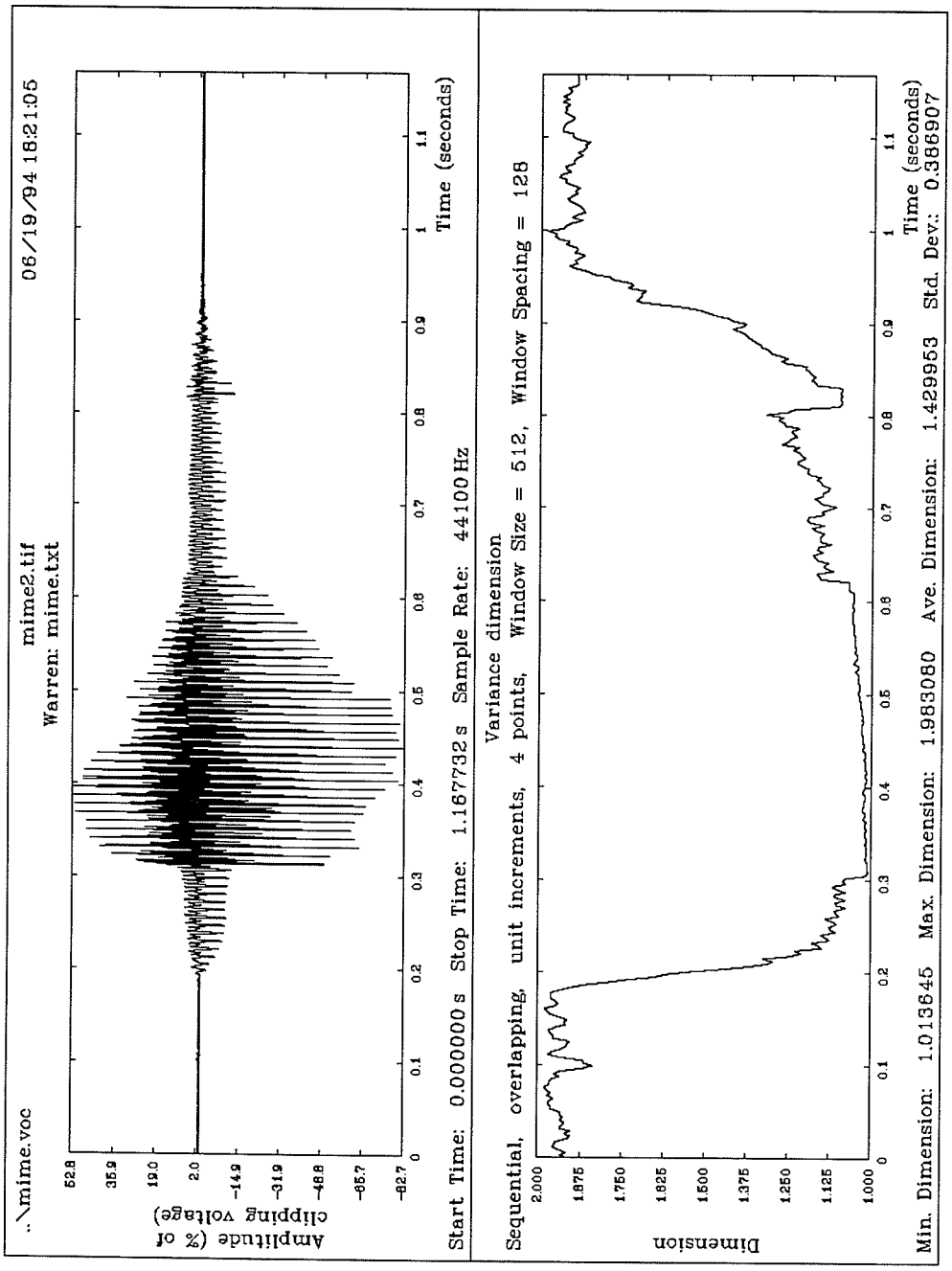
Loyal, Noise Separation Parameters



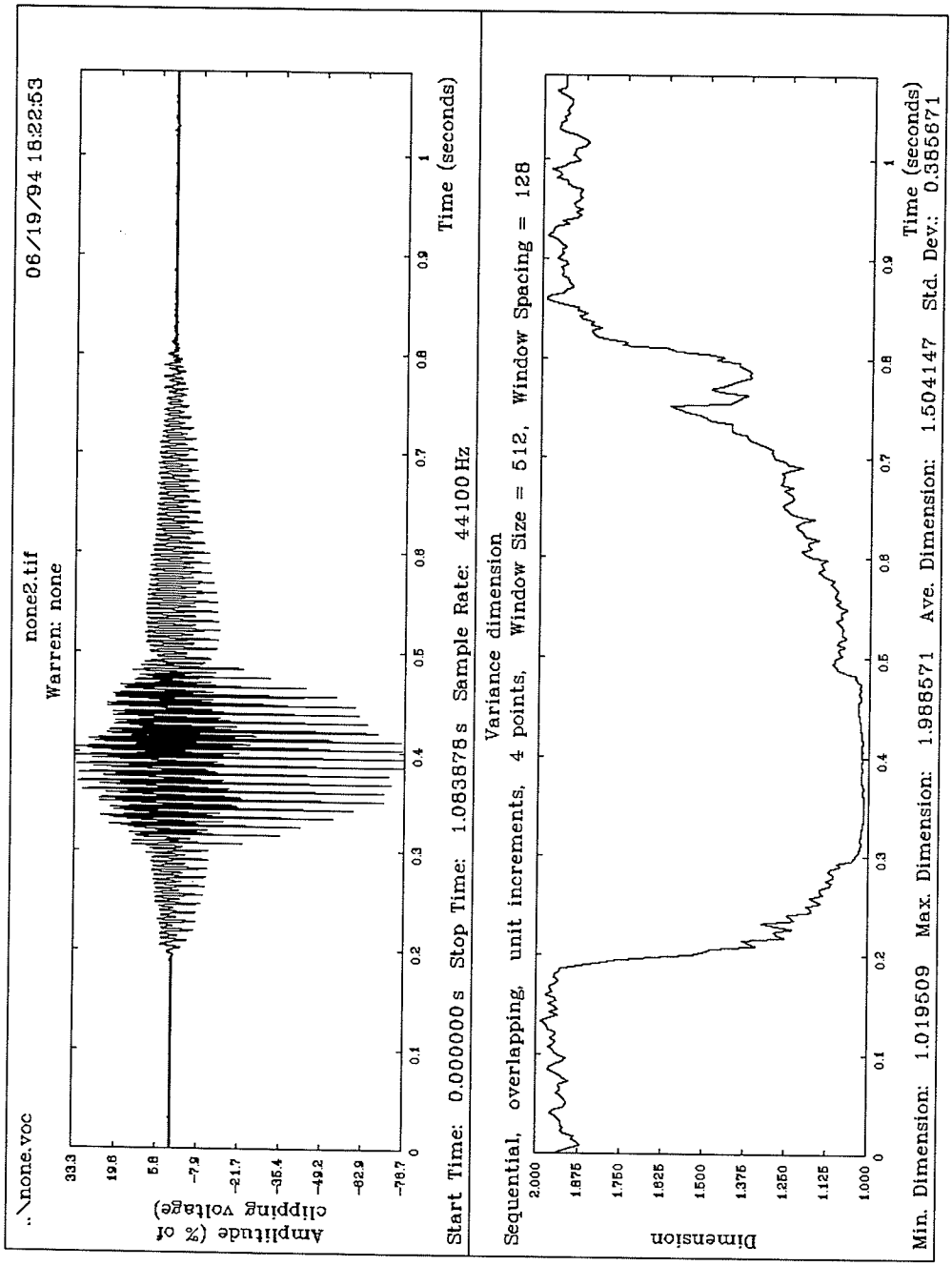
Measure, Noise Separation Parameters



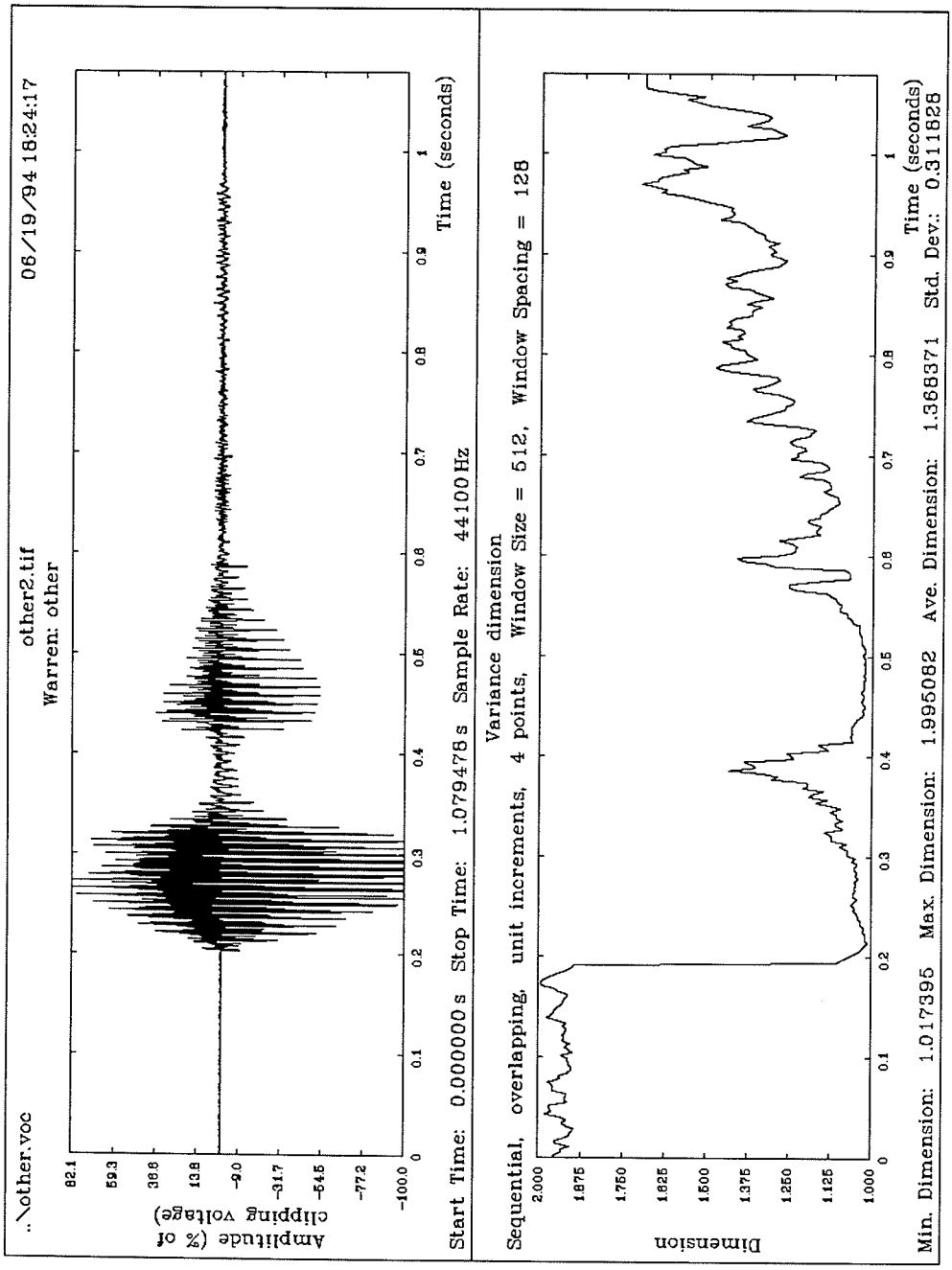
Mime, Noise Separation Parameters



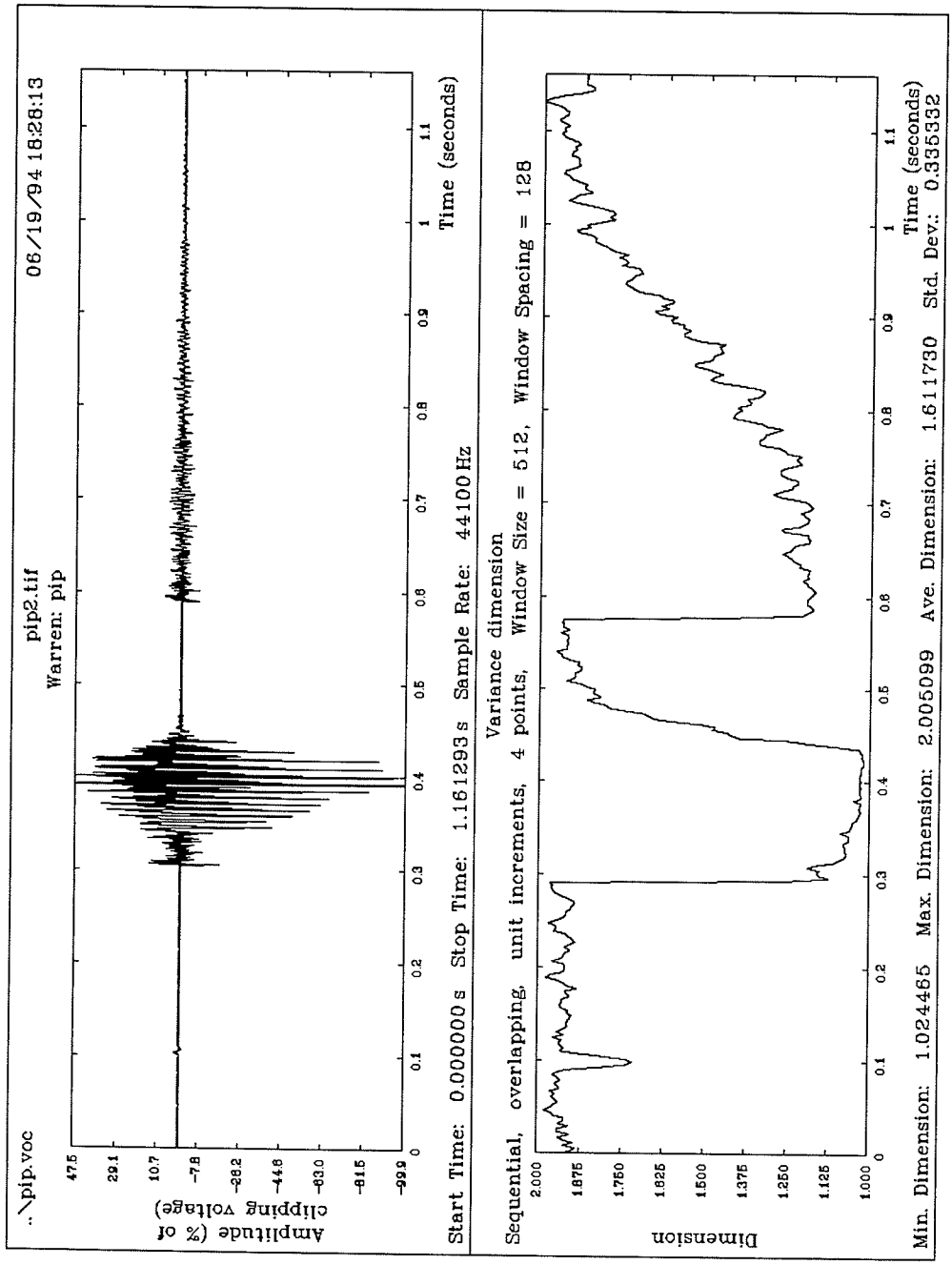
None, Noise Separation Parameters



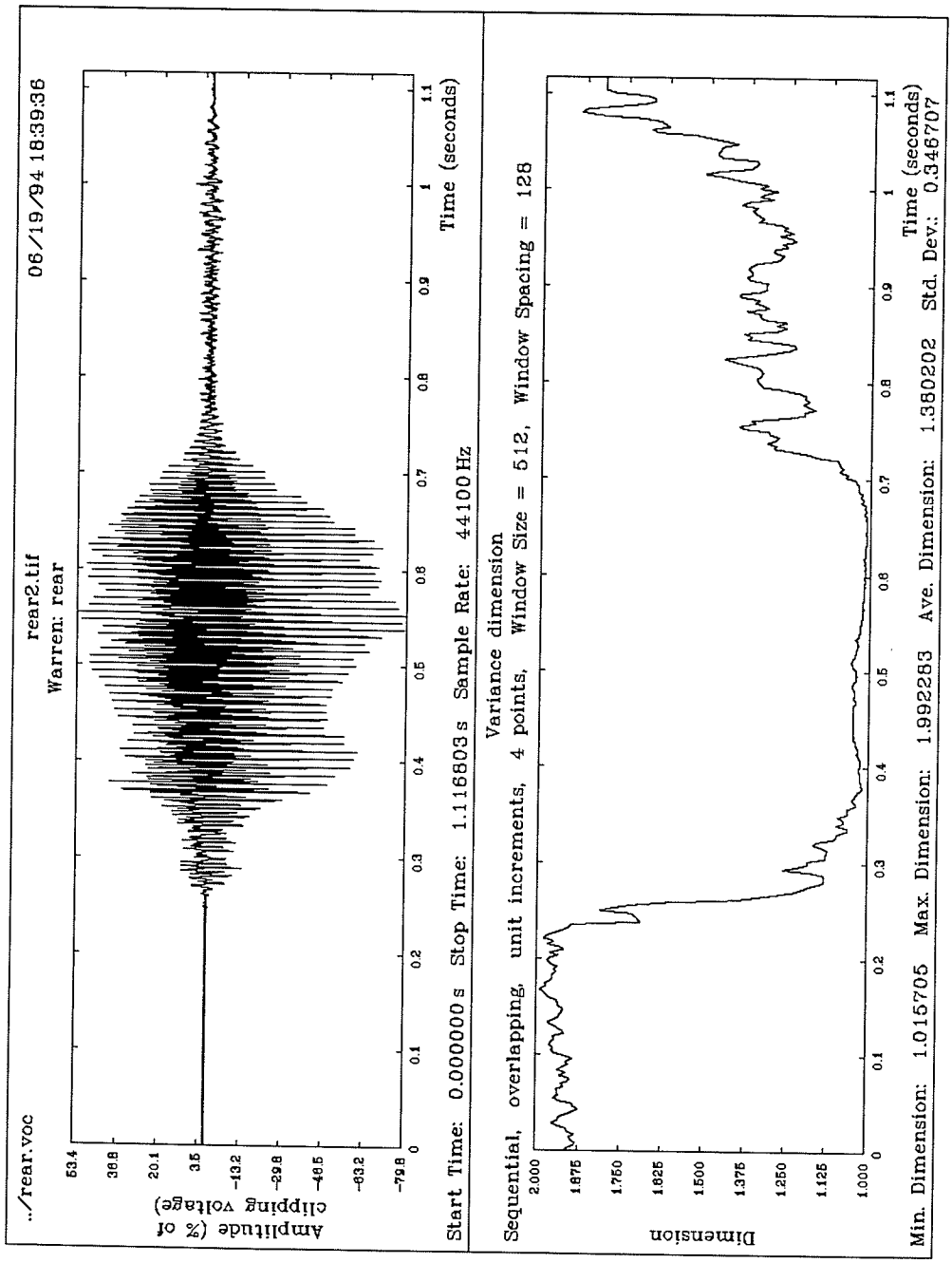
Other, Noise Separation Parameters



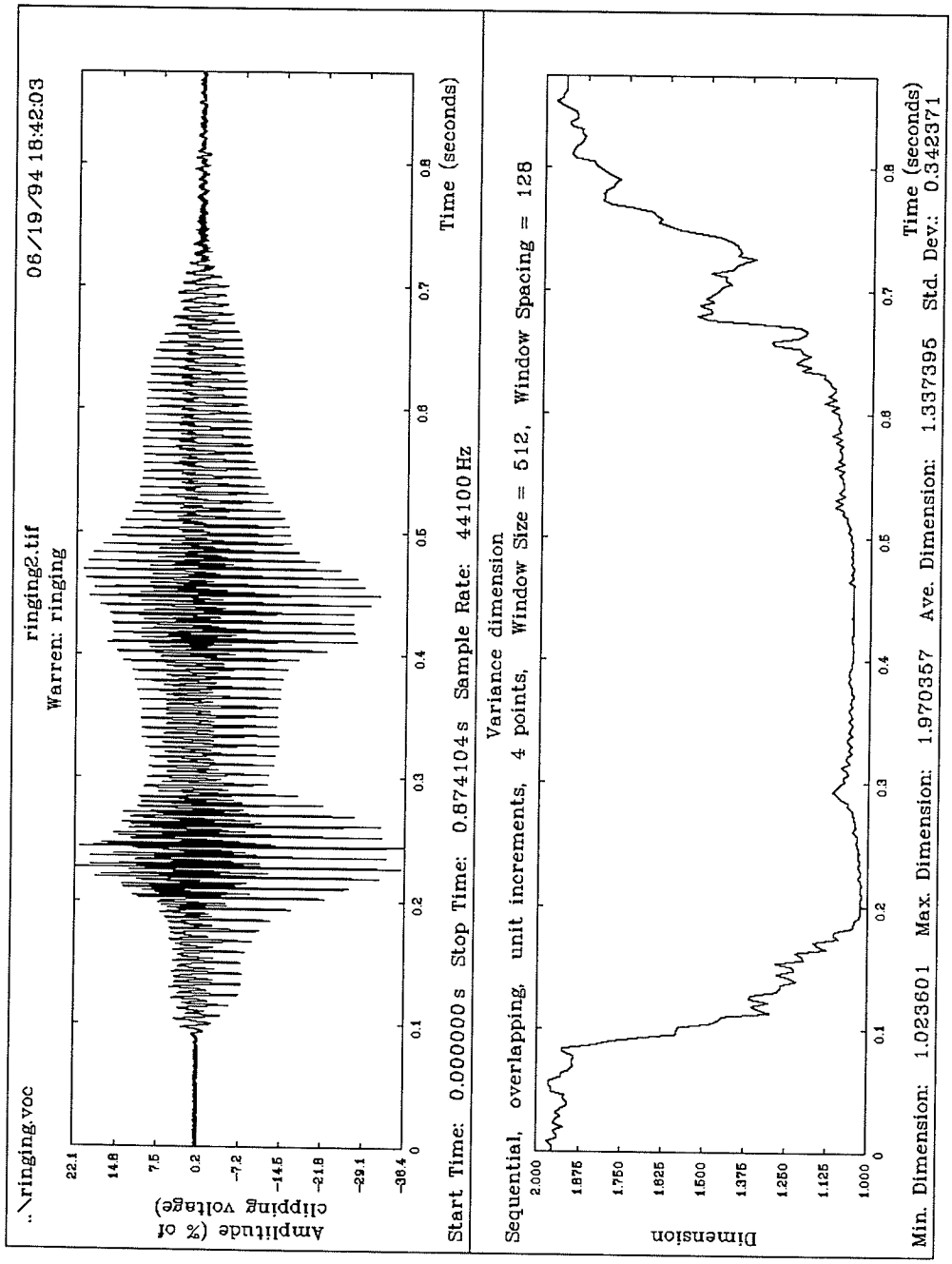
Pip, Noise Separation Parameters



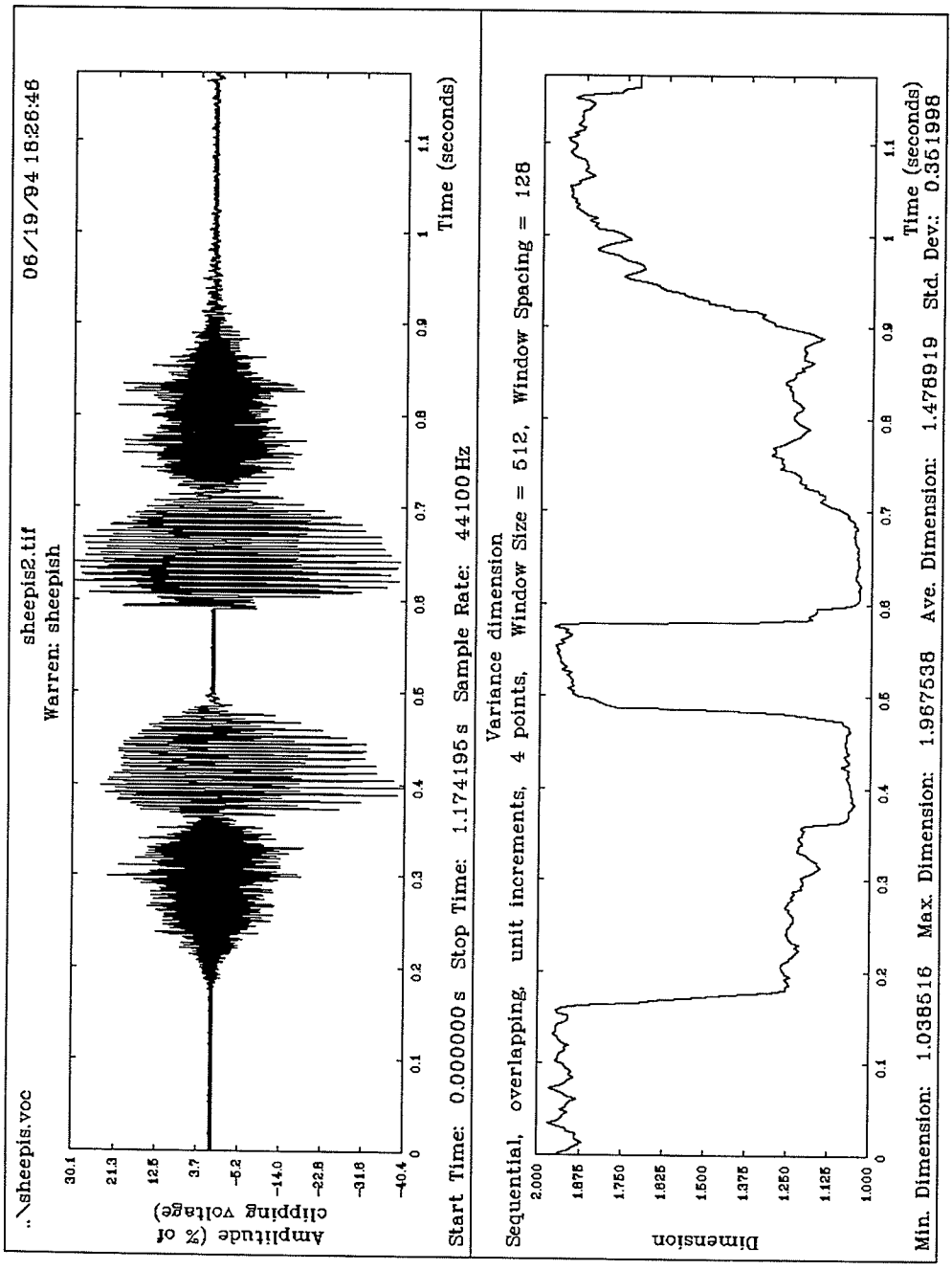
Rear, Noise Separation Parameters



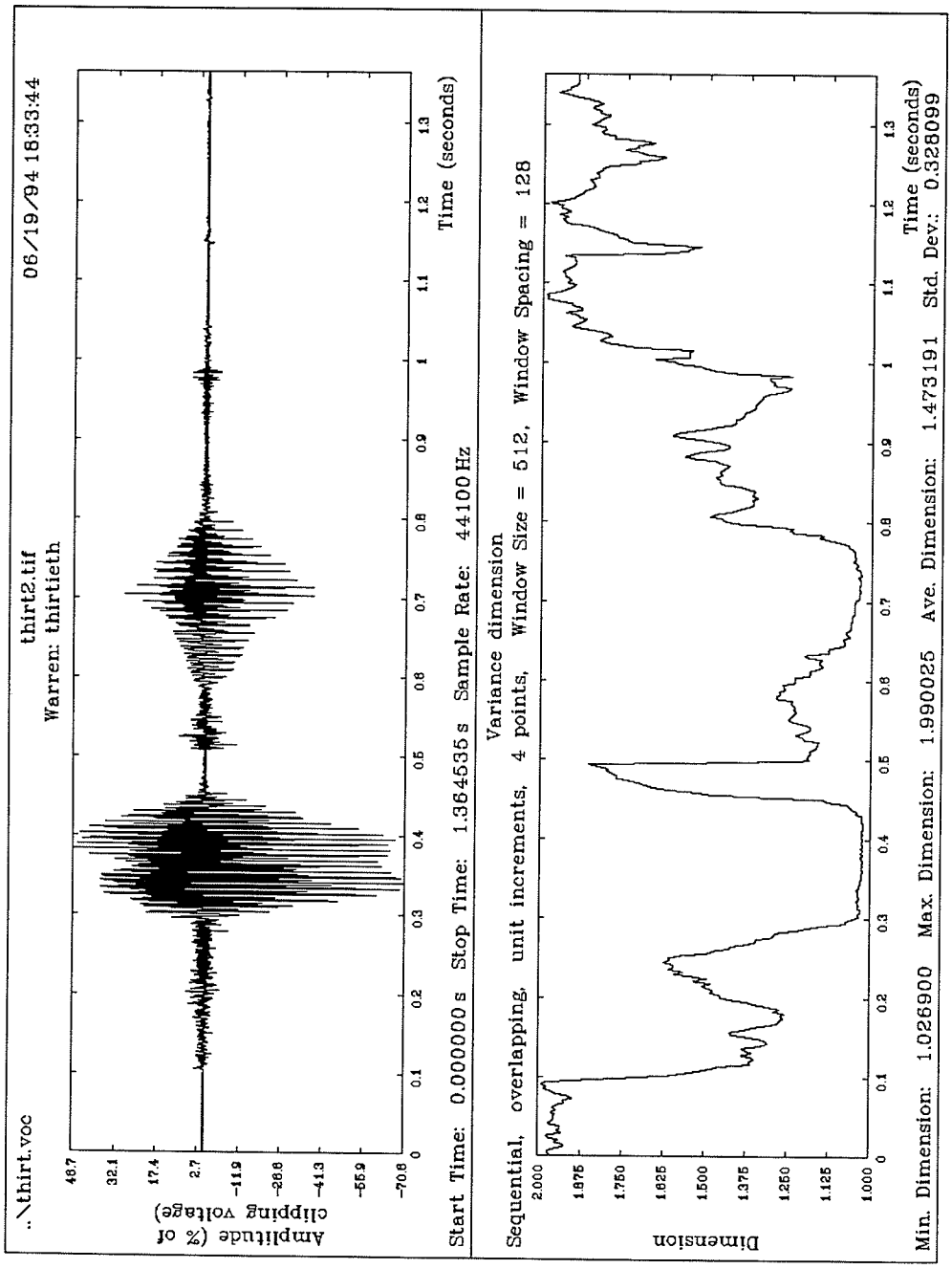
Ringing, Noise Separation Parameters



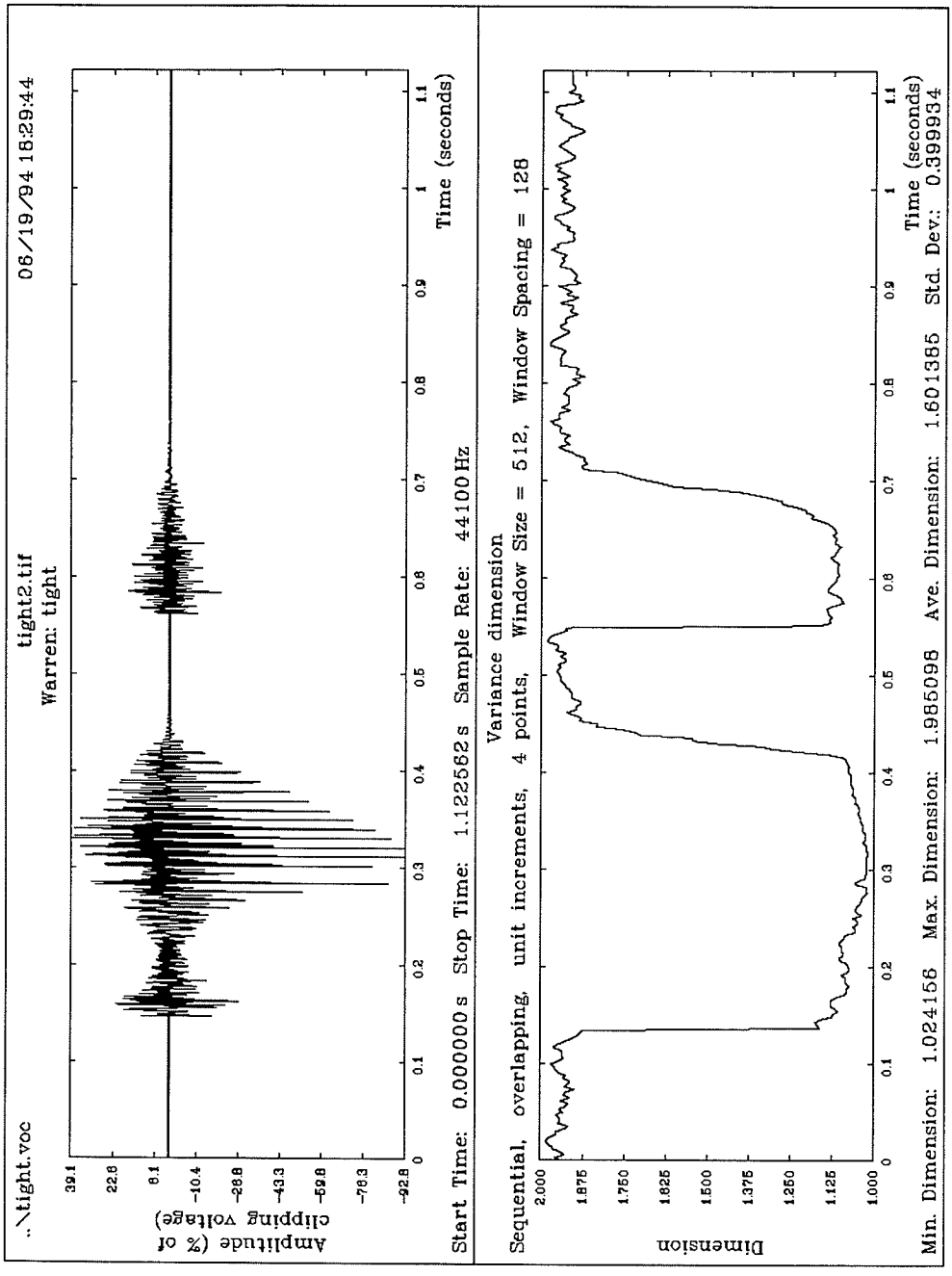
Sheepish, Noise Separation Parameters



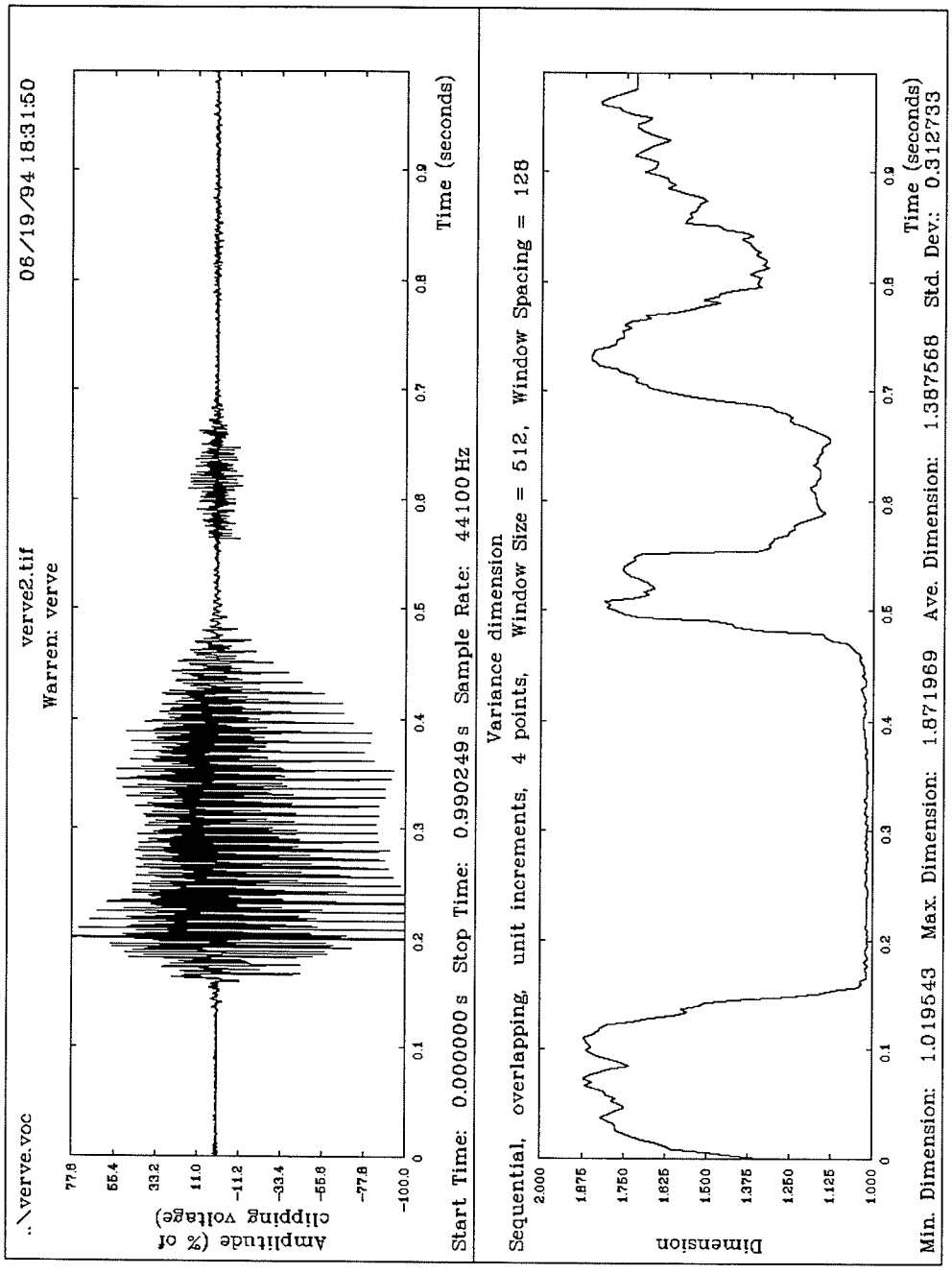
Thirtieth, Noise Separation Parameters



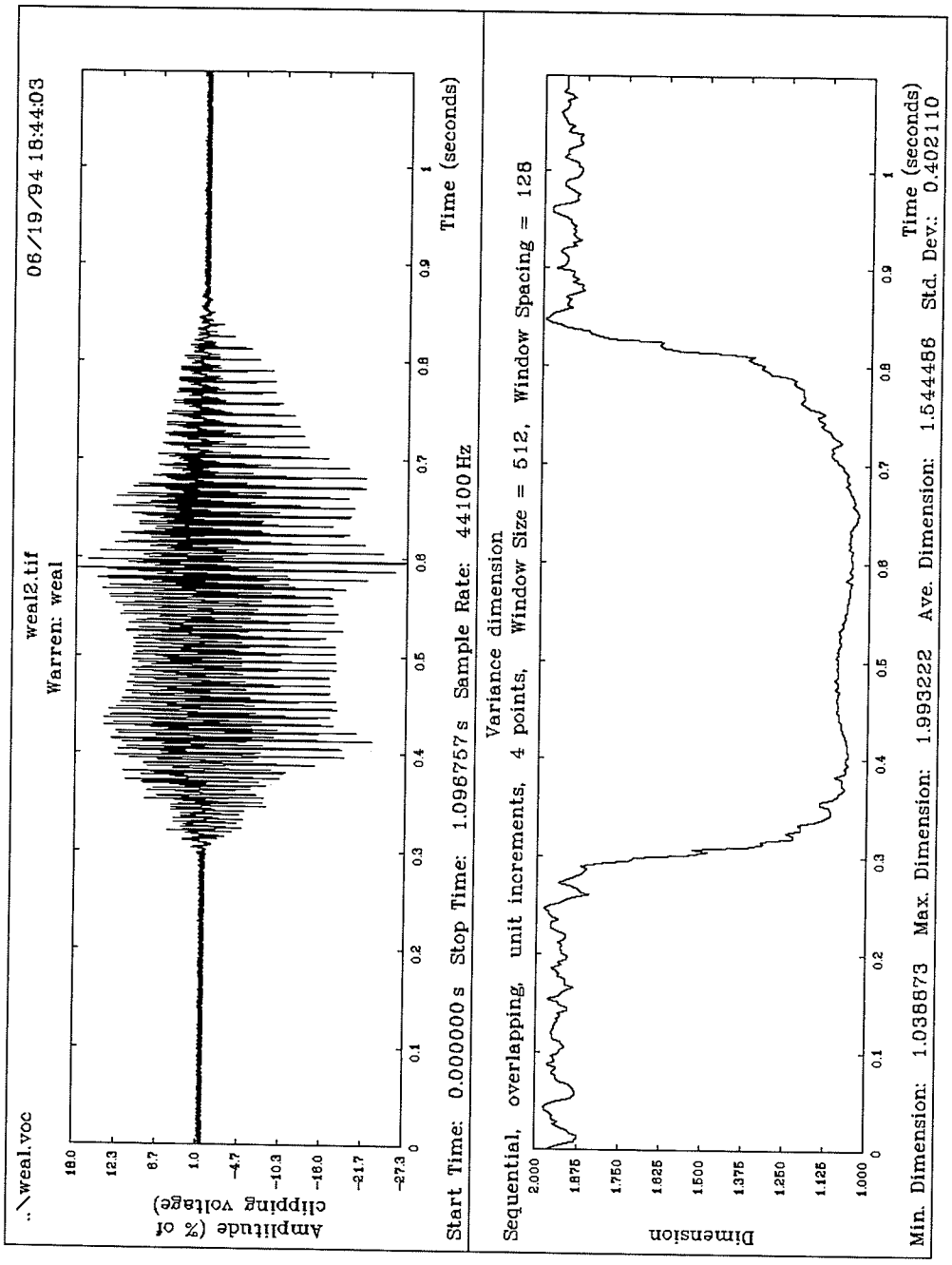
Tight, Noise Separation Parameters



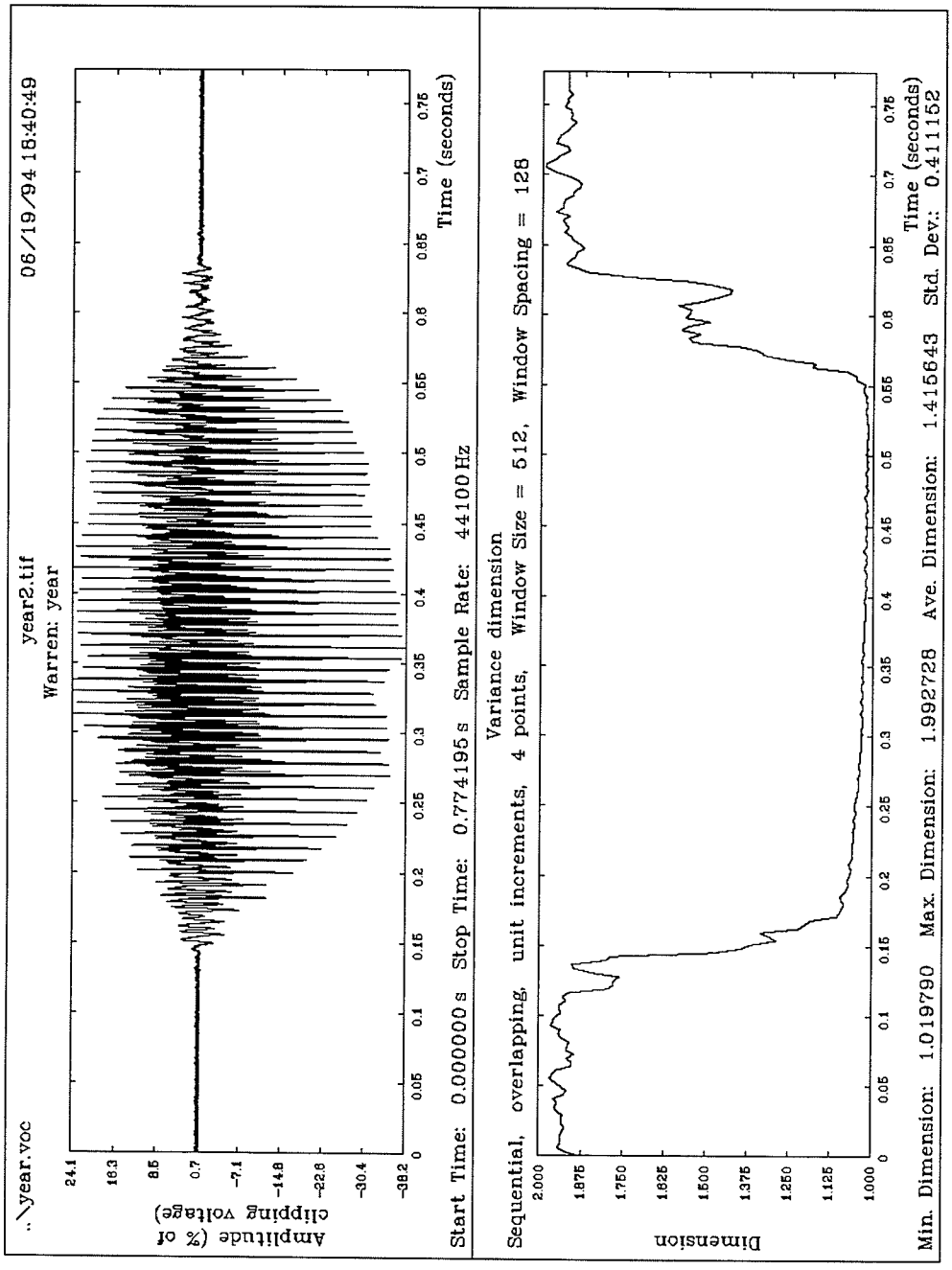
Verve, Noise Separation Parameters



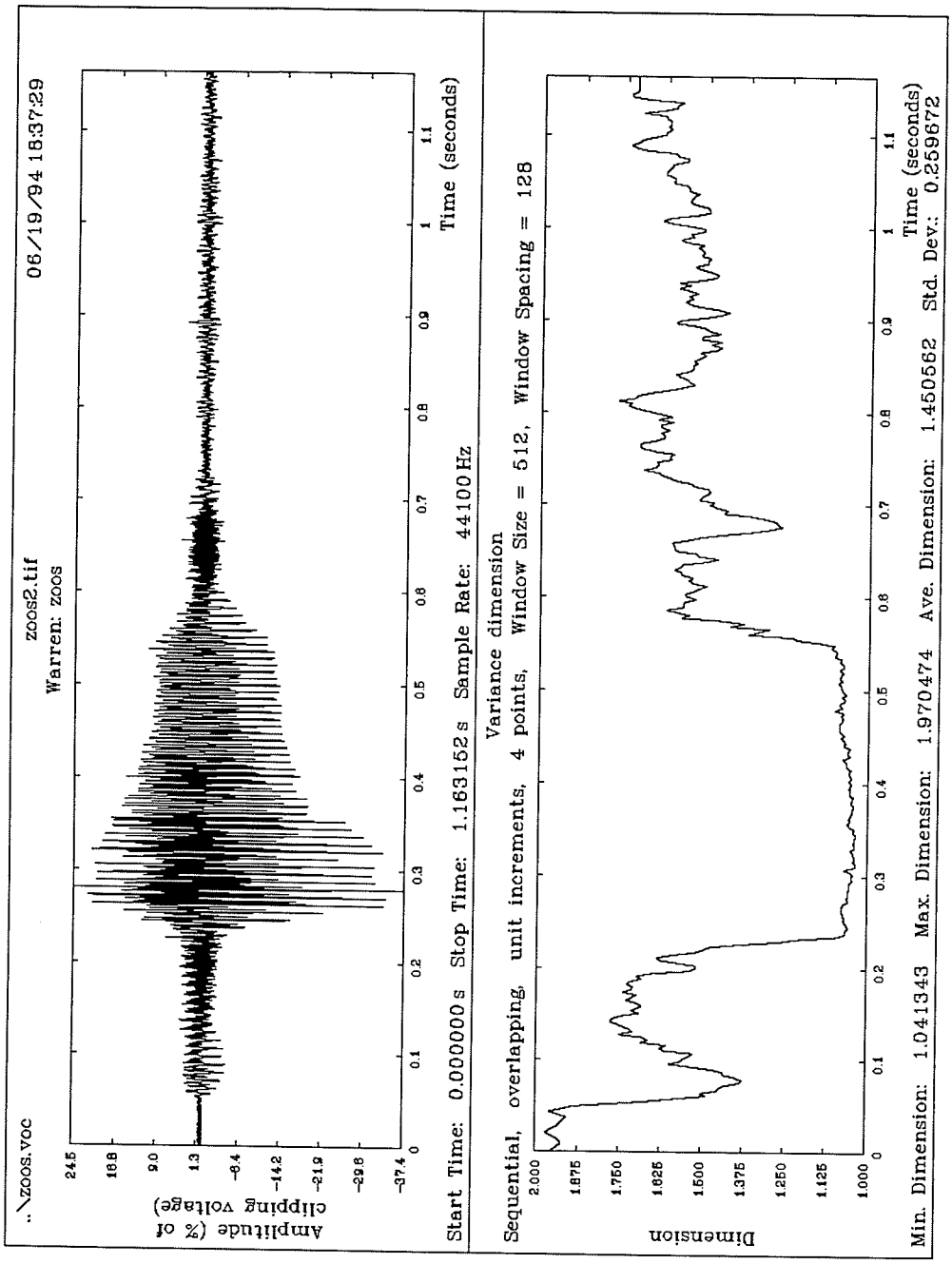
Weal, Noise Separation Parameters



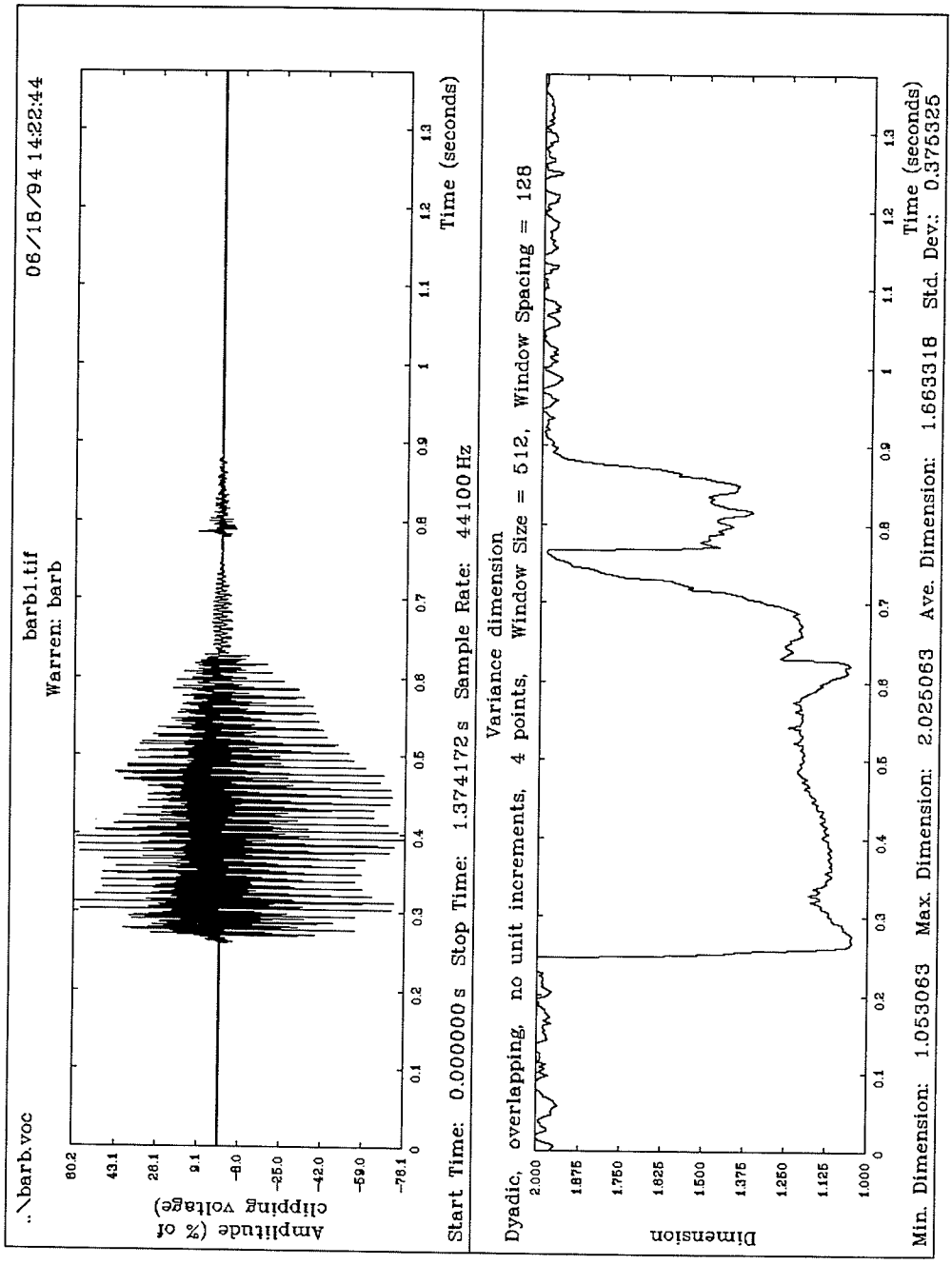
Year, Noise Separation Parameters

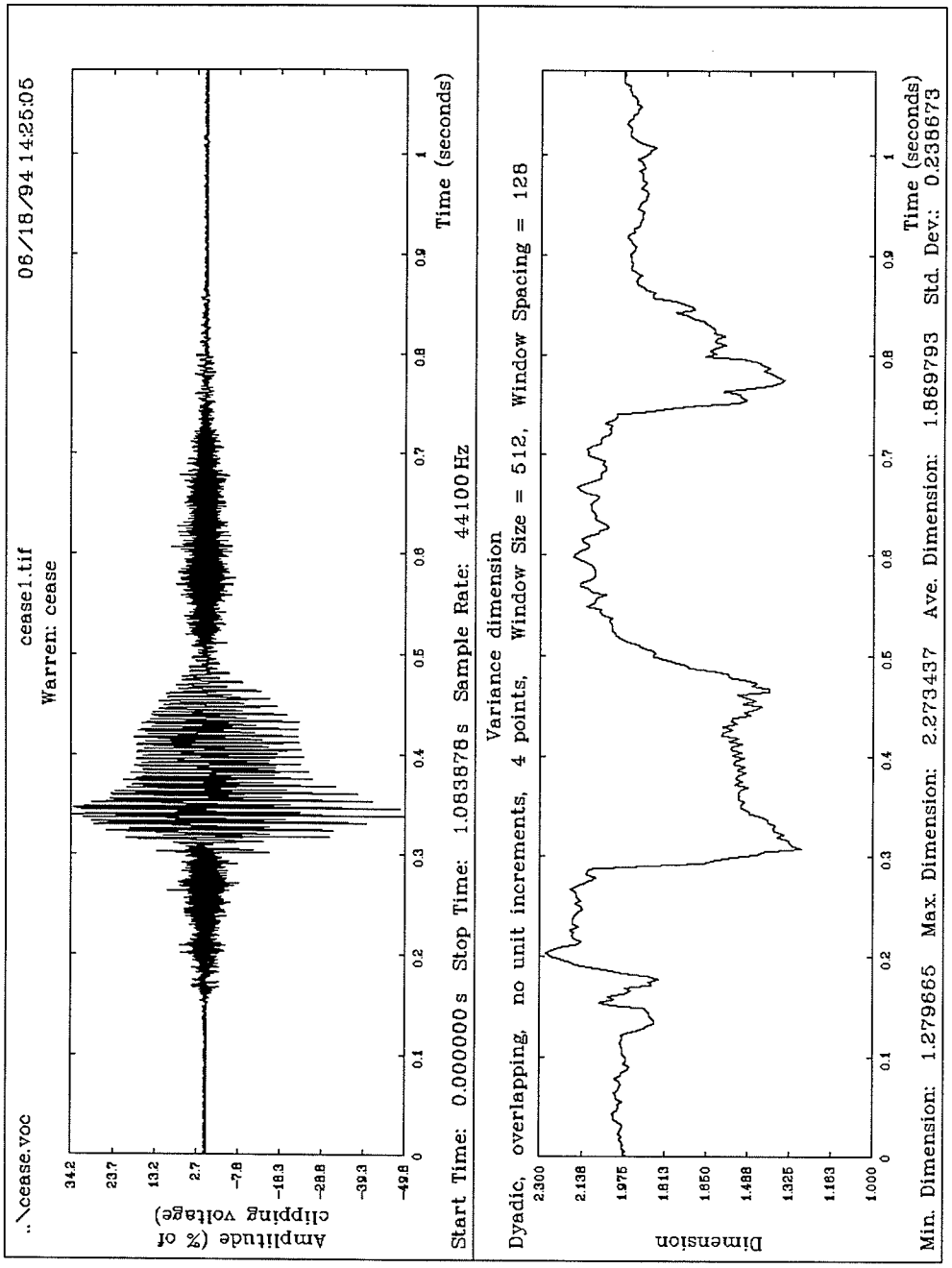


Zoos, Noise Separation Parameters

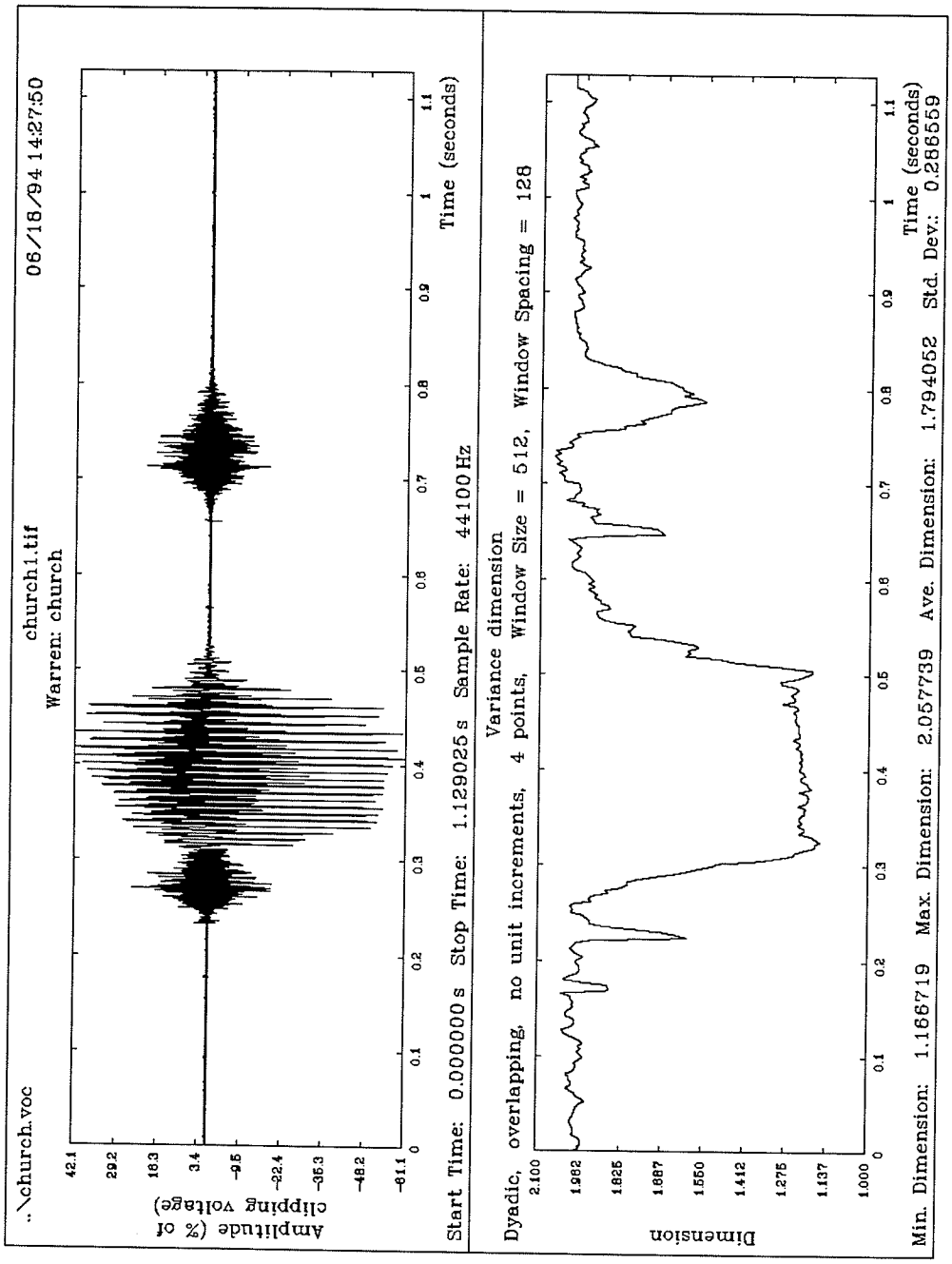


Barb, Fractal Amplification Parameters

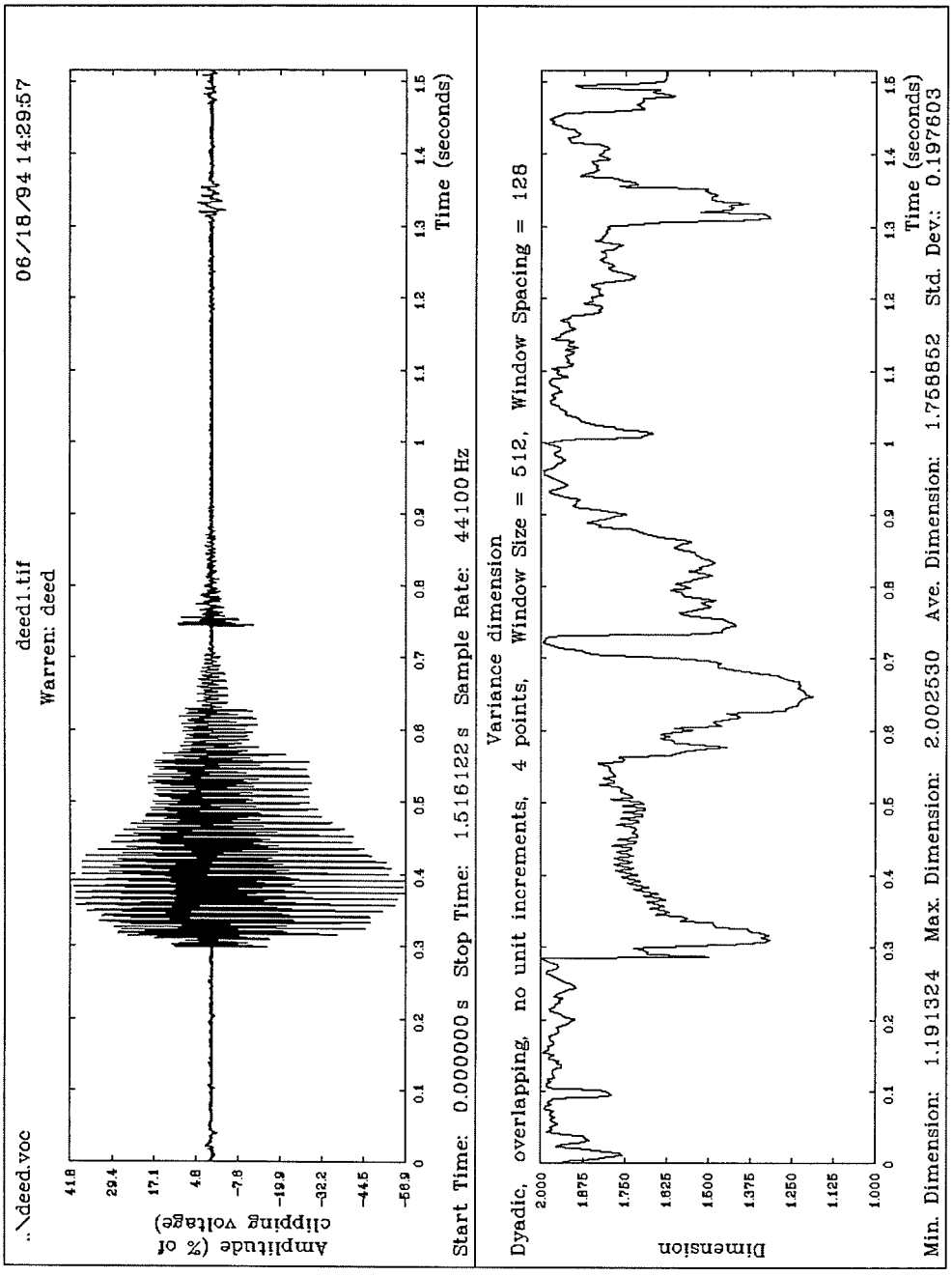



..\cease.voc
cease1.tif
Warren: cease
06/18/94 14:25:05
Amplitude (% of clipping voltage)
342
237
132
27
-78
-183
-288
-393
-498
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Time (seconds)
Start Time: 0.000000 s Stop Time: 1.083878 s Sample Rate: 44100 Hz
Variance dimension
Dyadic, overlapping, no unit increments, 4 points, Window Size = 512, Window Spacing = 128
Dimension
2.300
2.138
1.975
1.813
1.650
1.488
1.325
1.163
1.000
Time (seconds)
Min. Dimension: 1.279665 Max. Dimension: 2.273437 Ave. Dimension: 1.869793 Std. Dev.: 0.238673

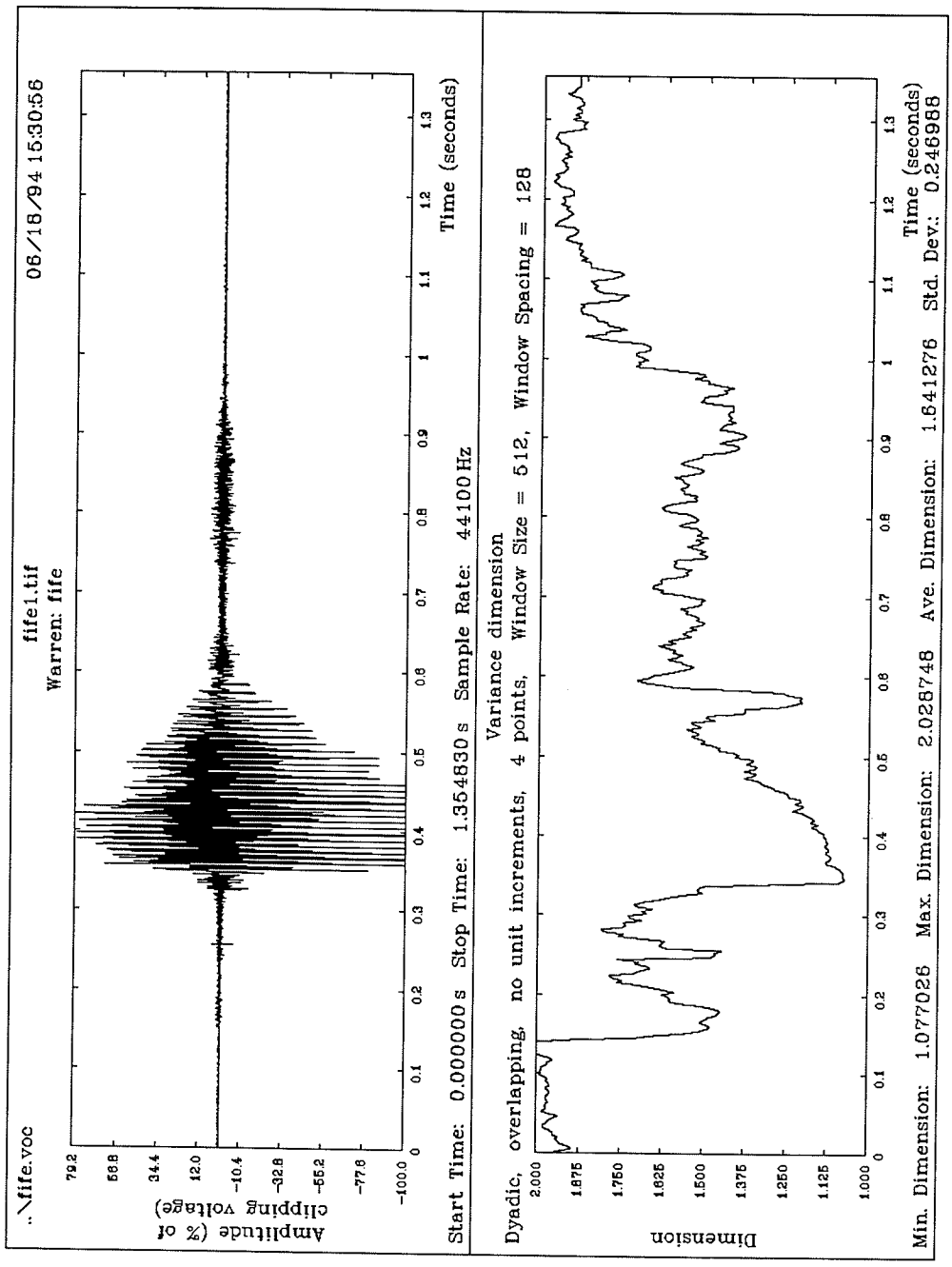
Church, Fractal Amplification Parameters



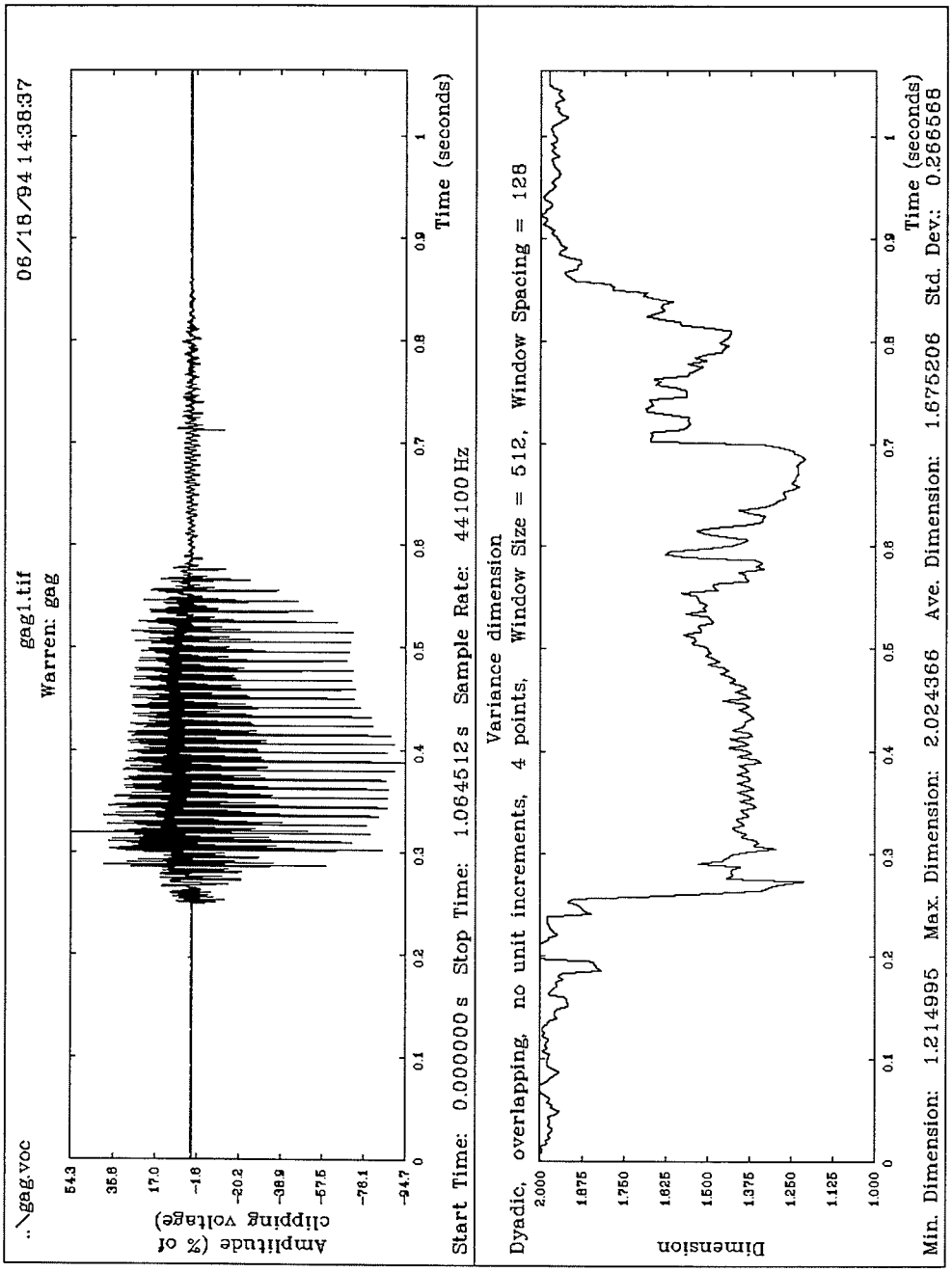
Deed, Fractal Amplification Parameters



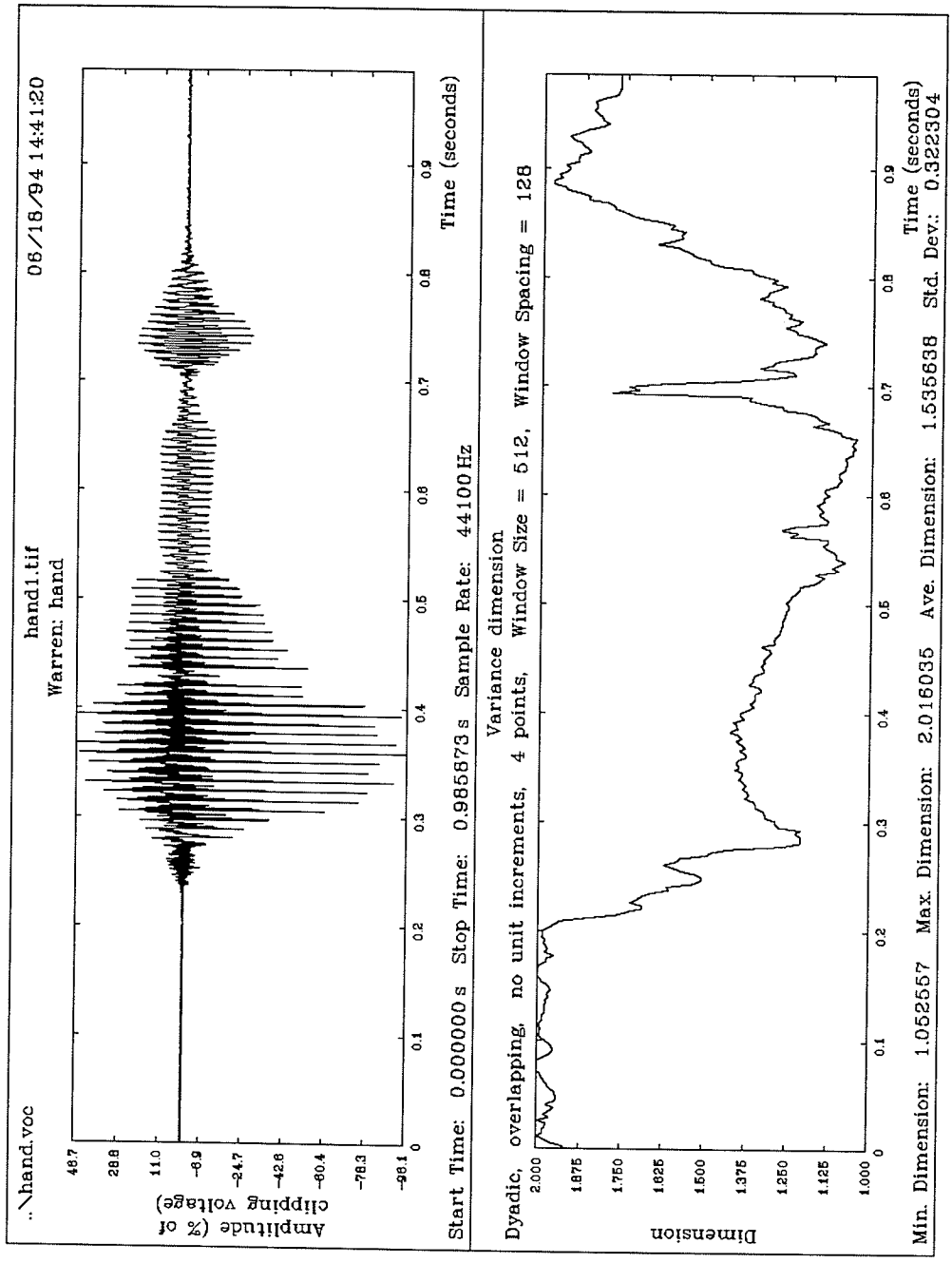
Fife, Fractal Amplification Parameters



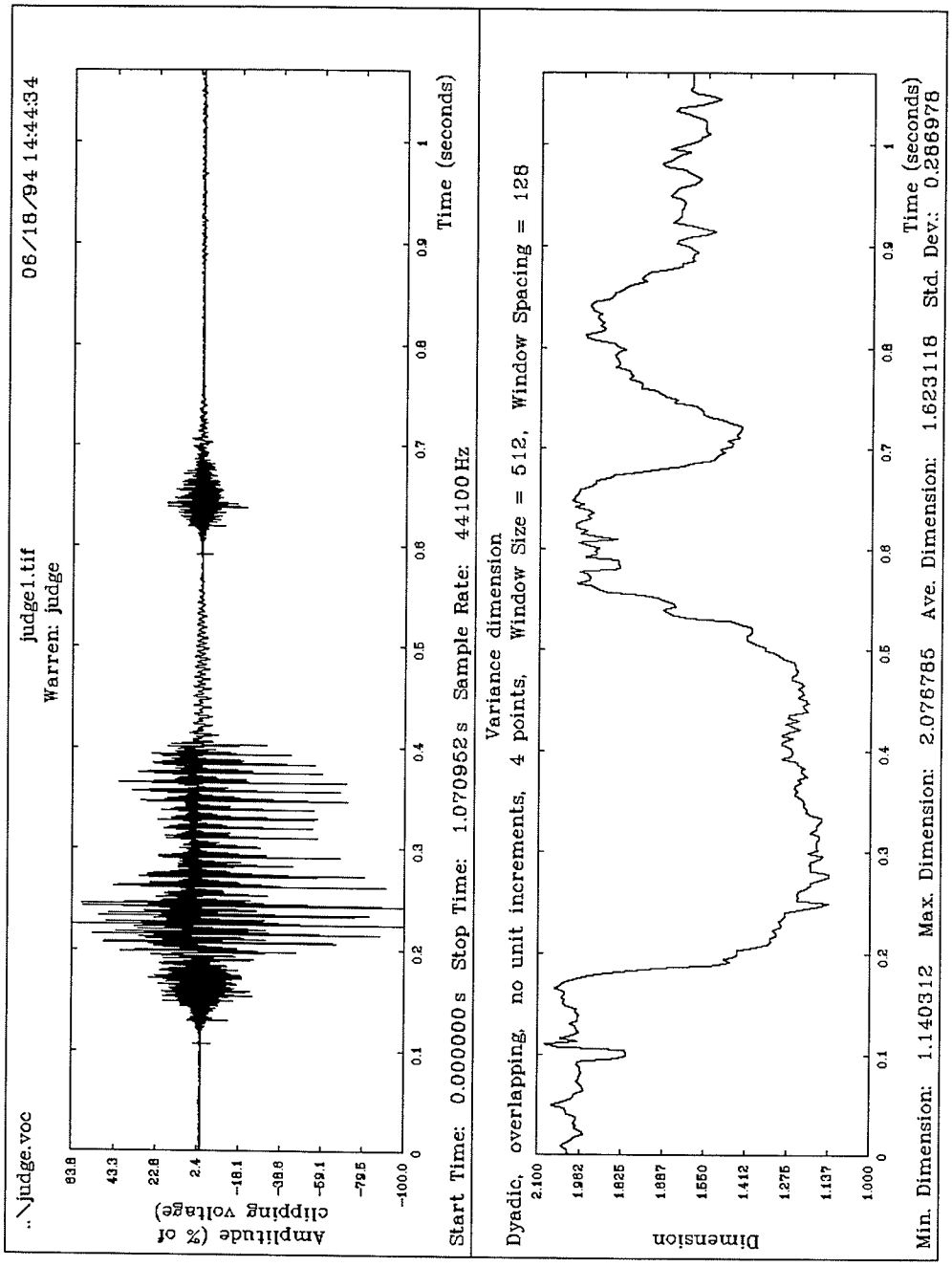
Gag, Fractal Amplification Parameters



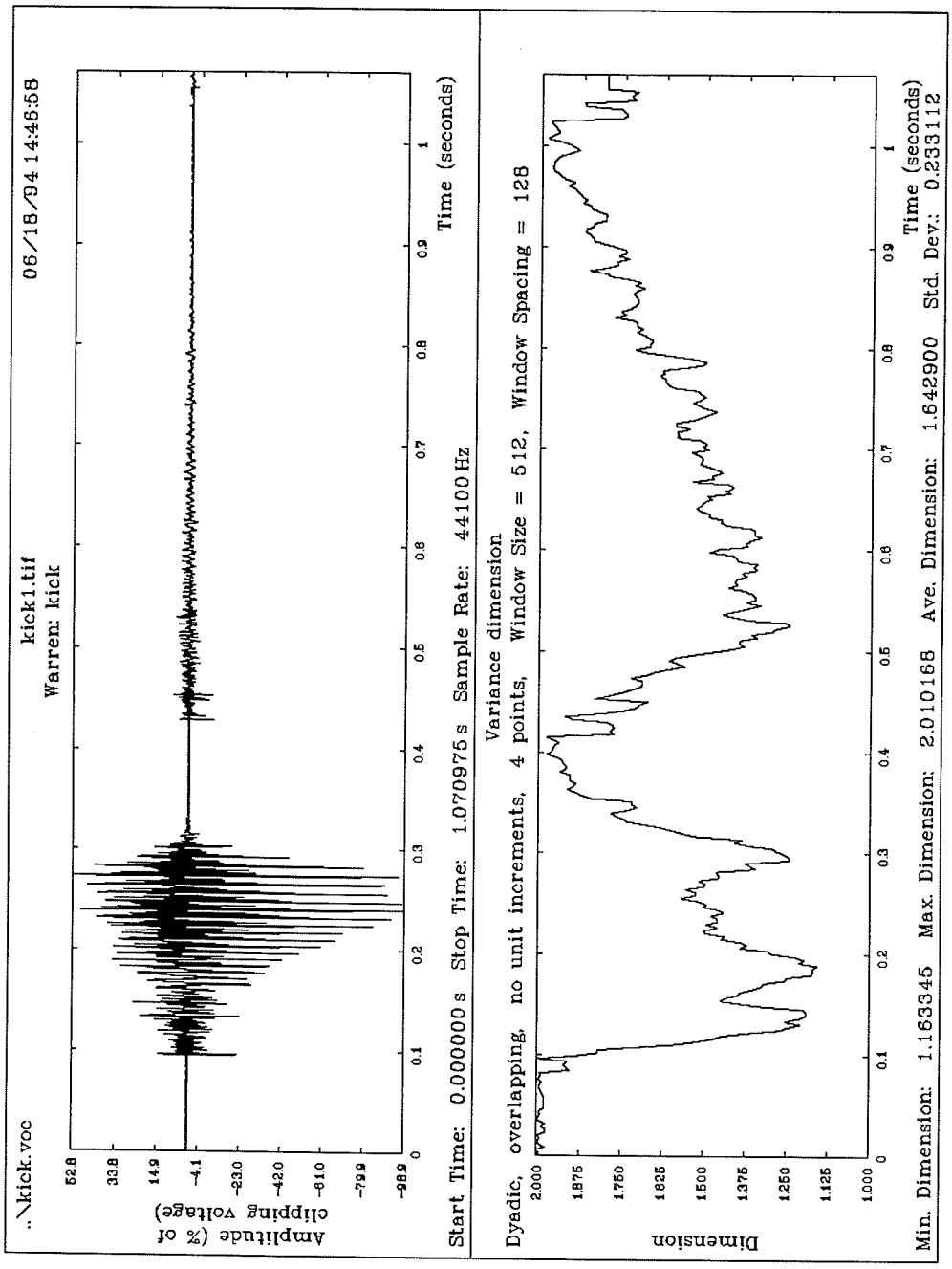
Hand, Fractal Amplification Parameters



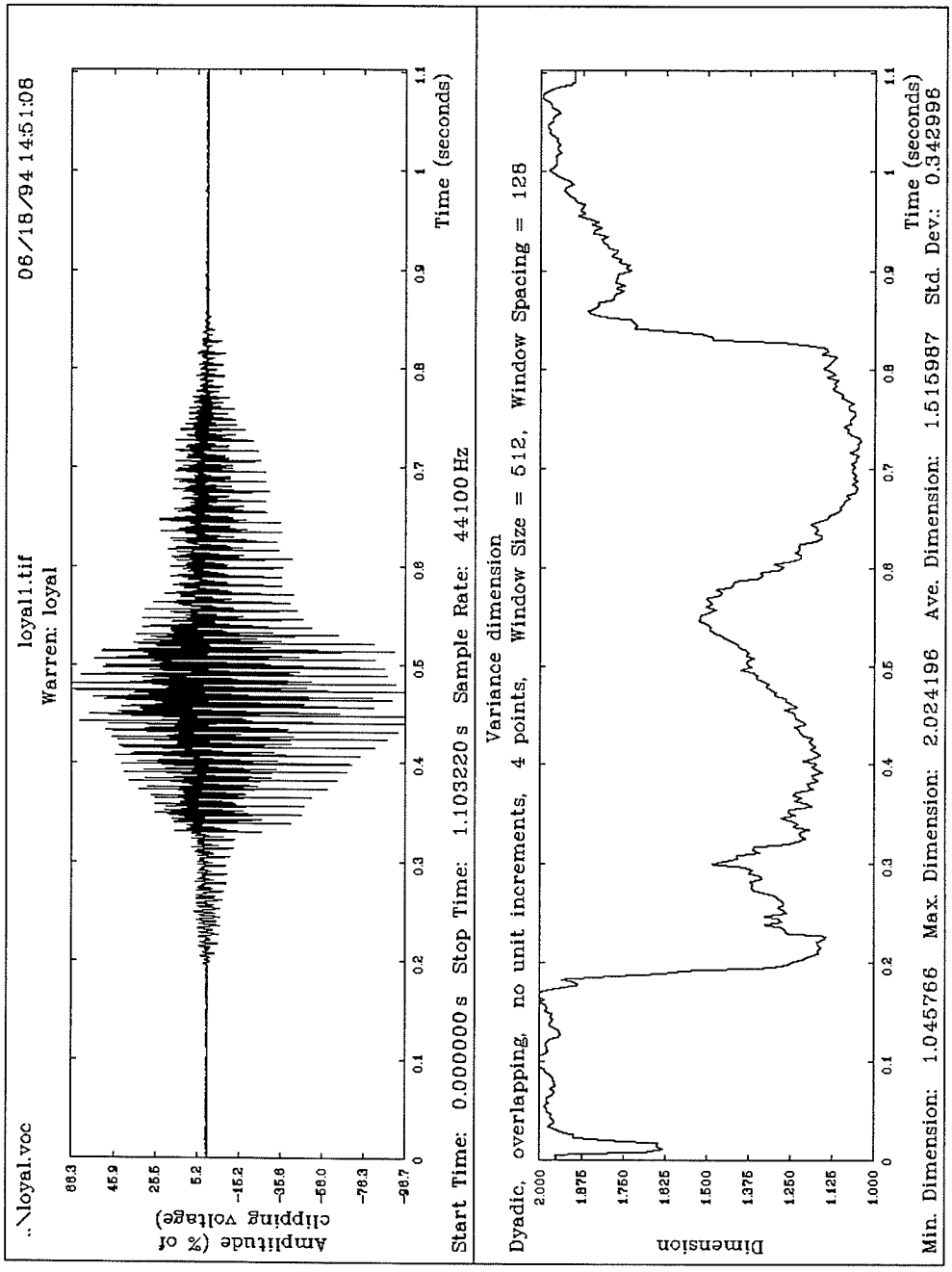
Judge, Fractal Amplification Parameters



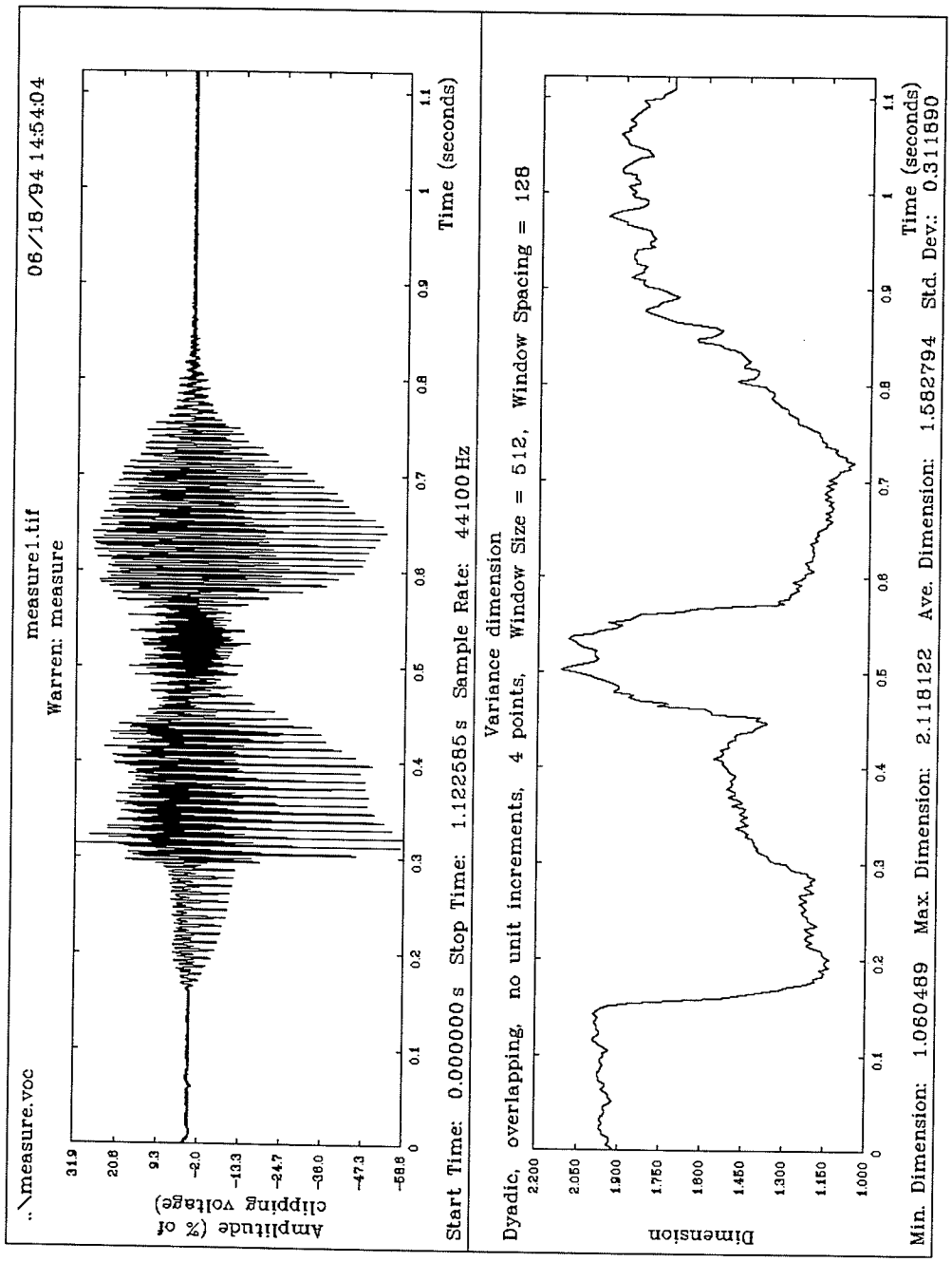
Kick, Fractal Amplification Parameters



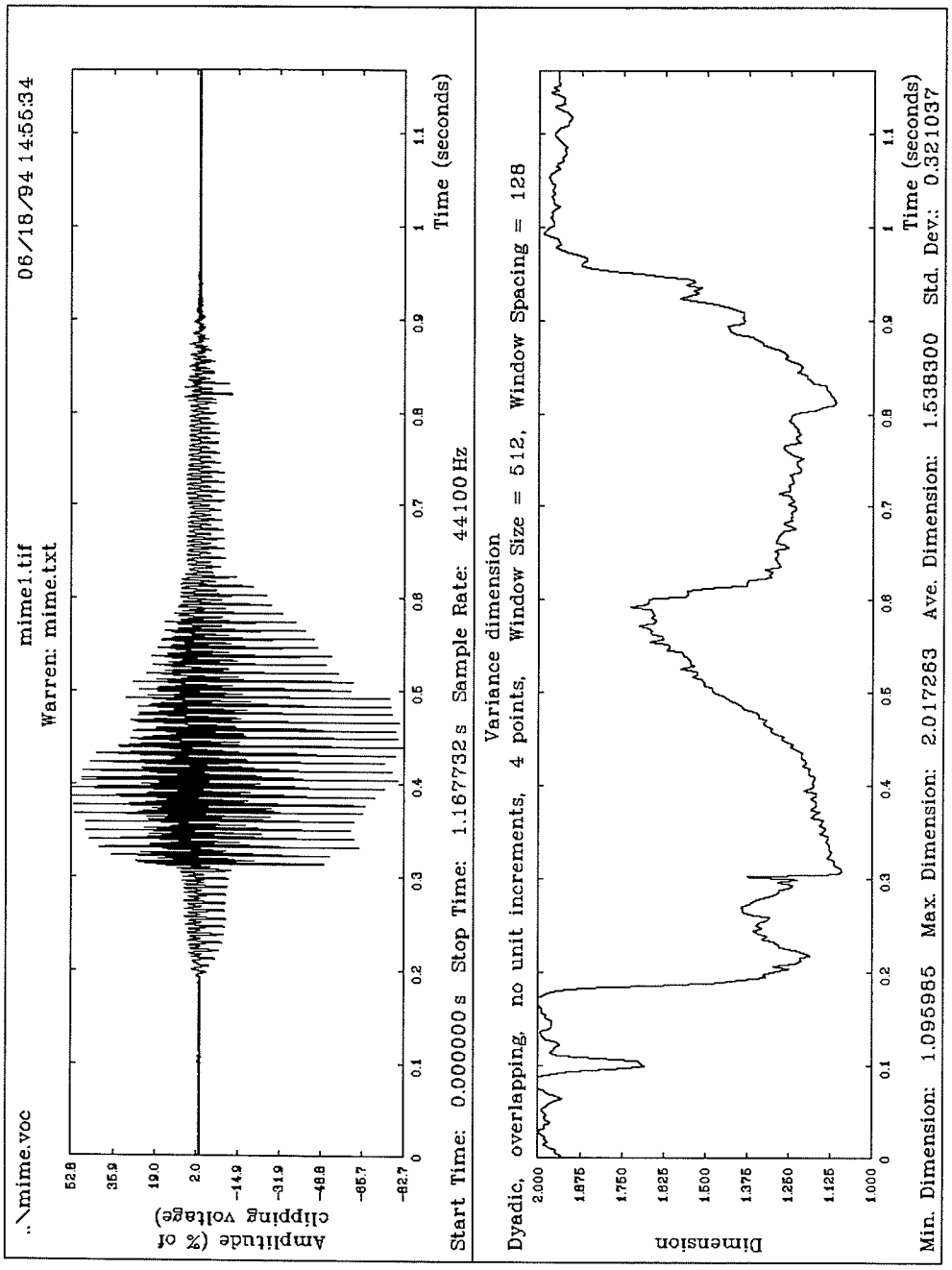
Loyal, Fractal Amplification Parameters



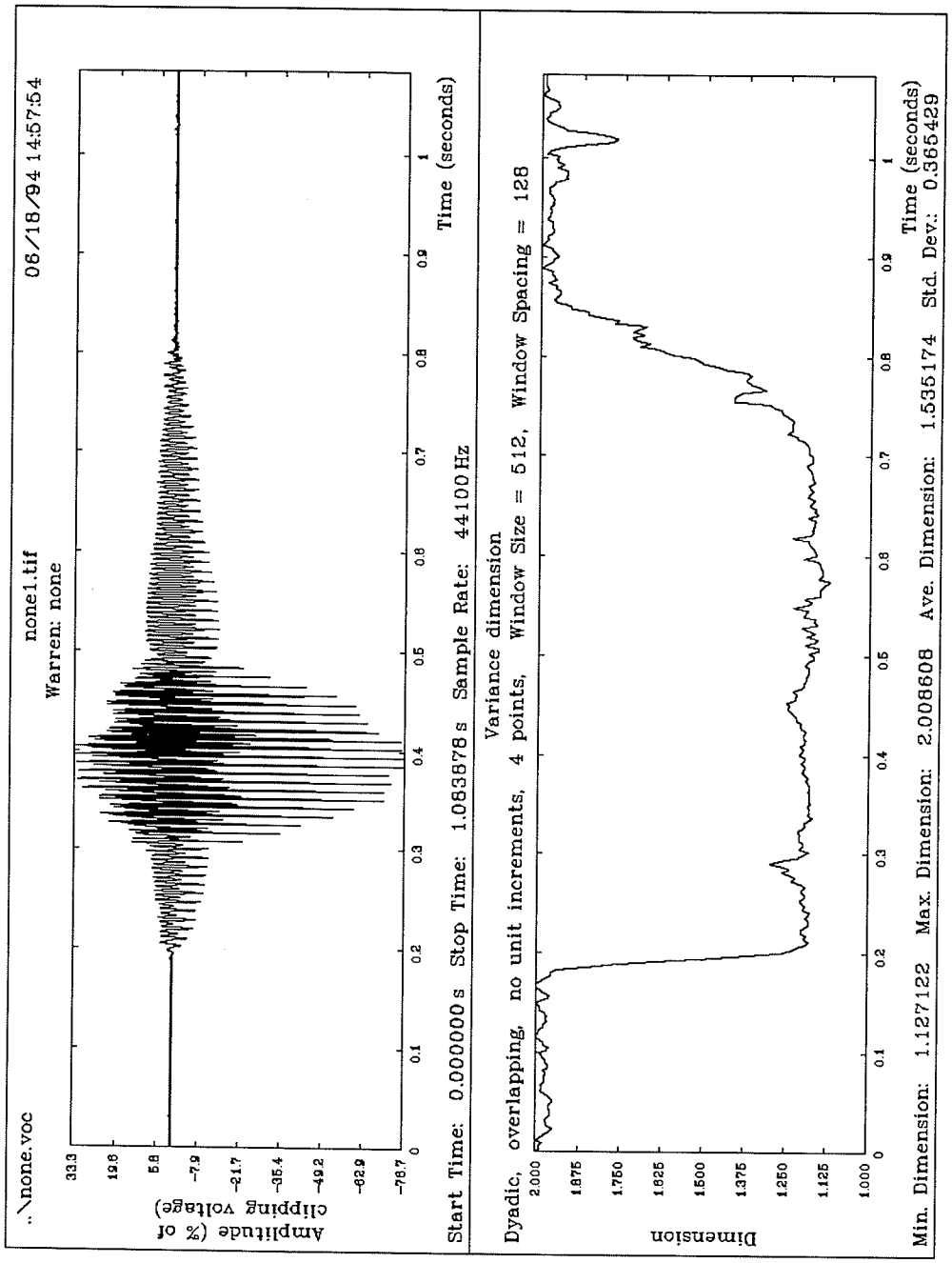
Measure, Fractal Amplification Parameters



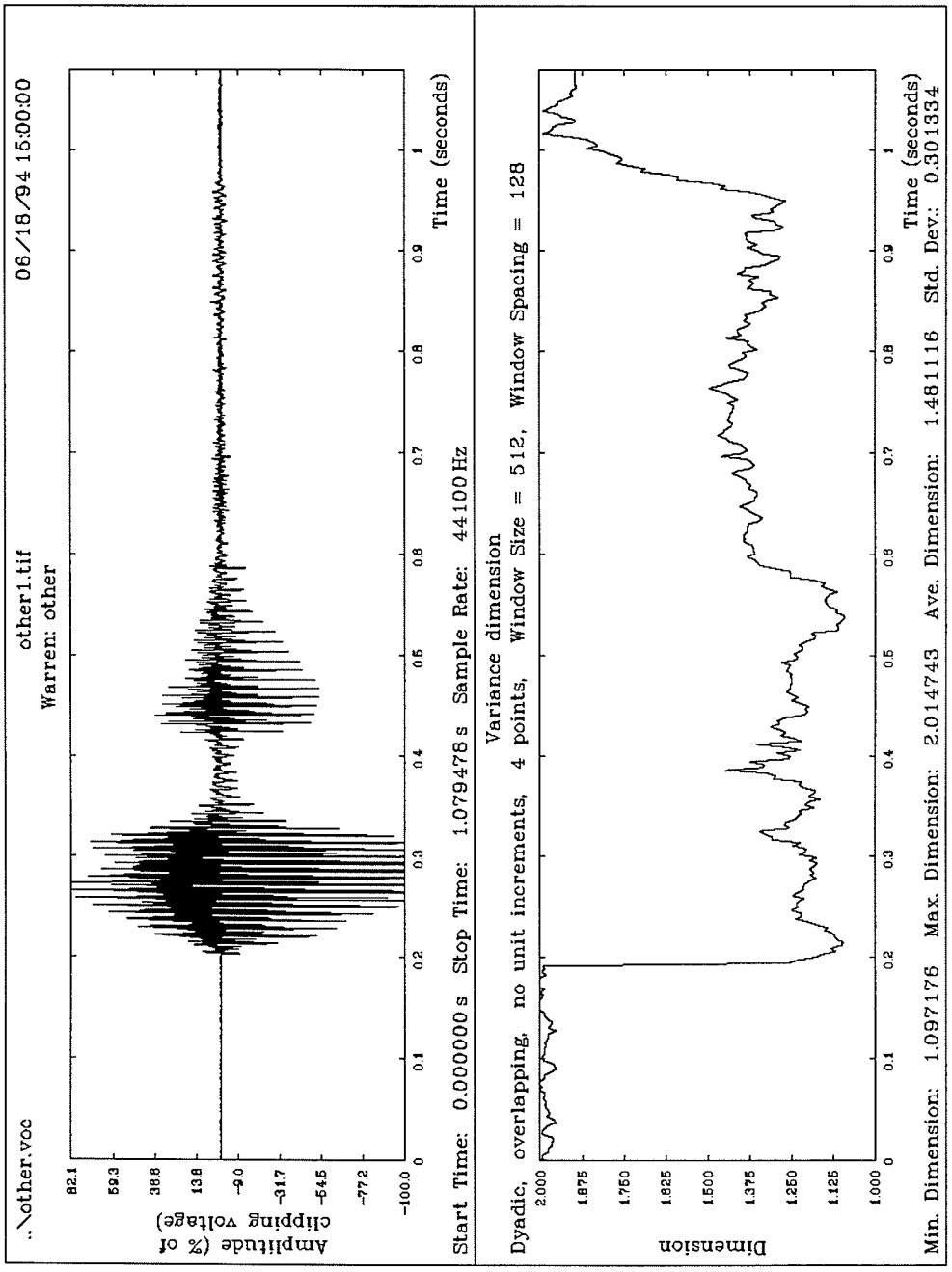
Mime, Fractal Amplification Parameters



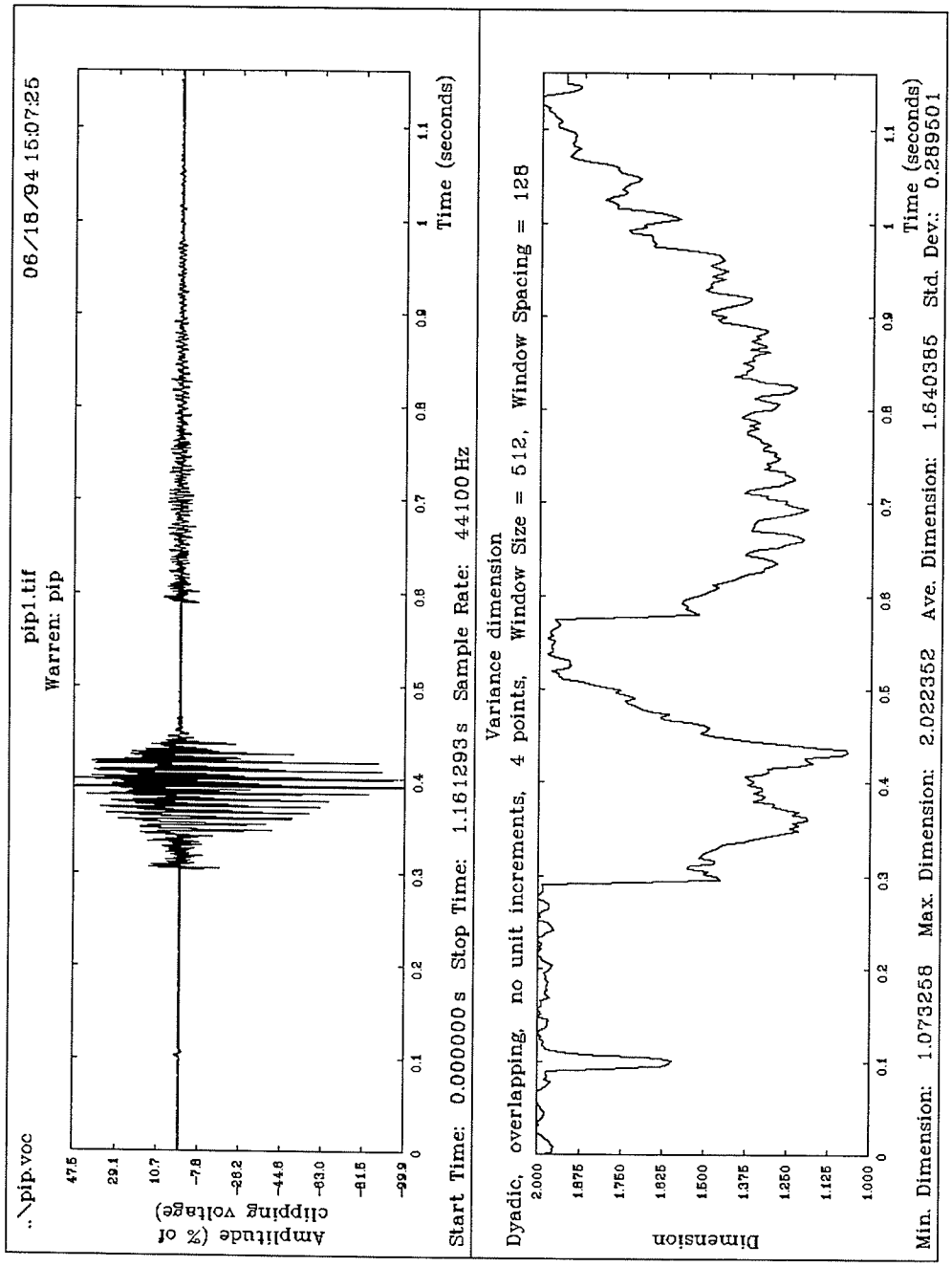
None, Fractal Amplification Parameters



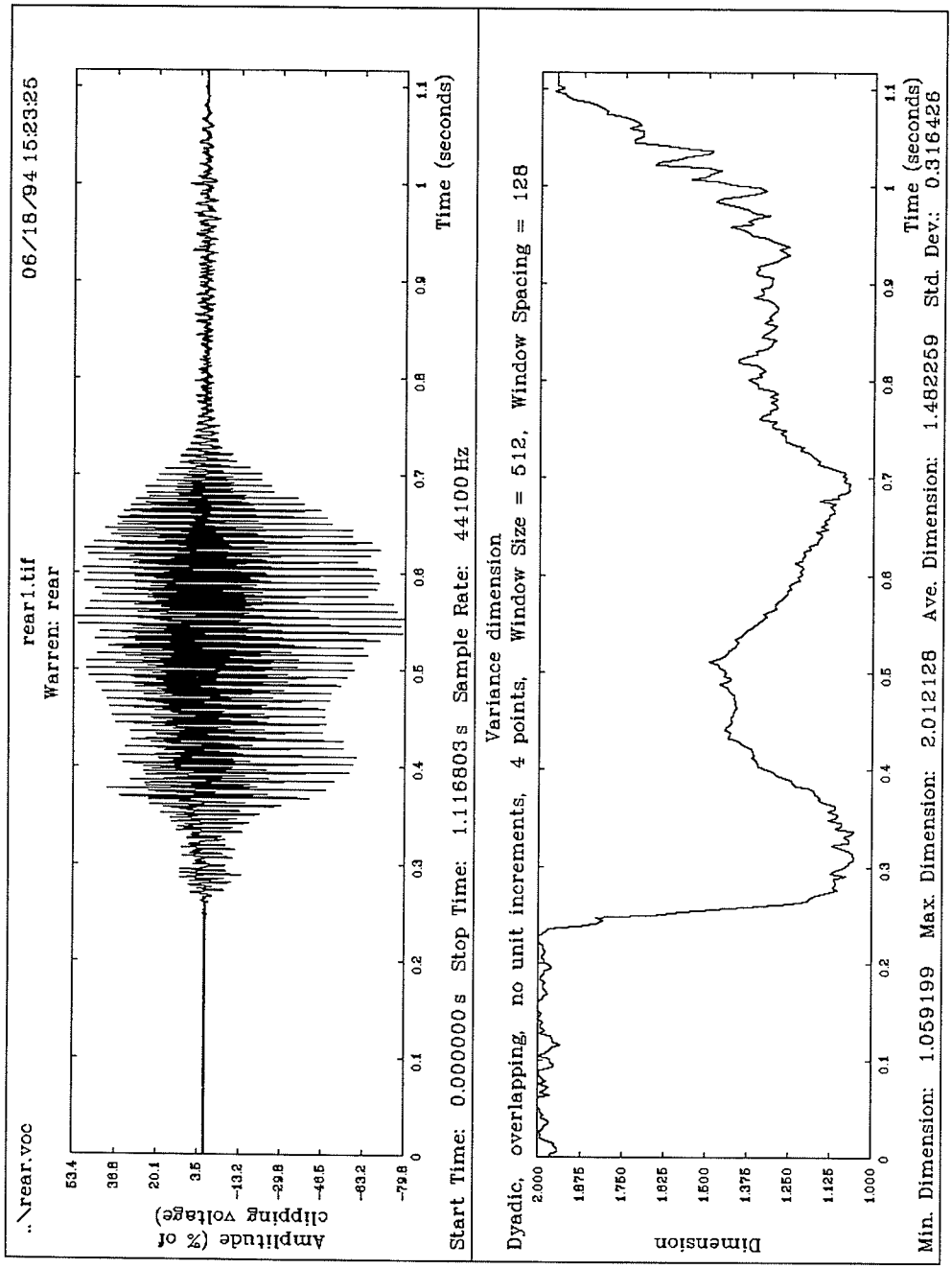
Other, Fractal Amplification Parameters



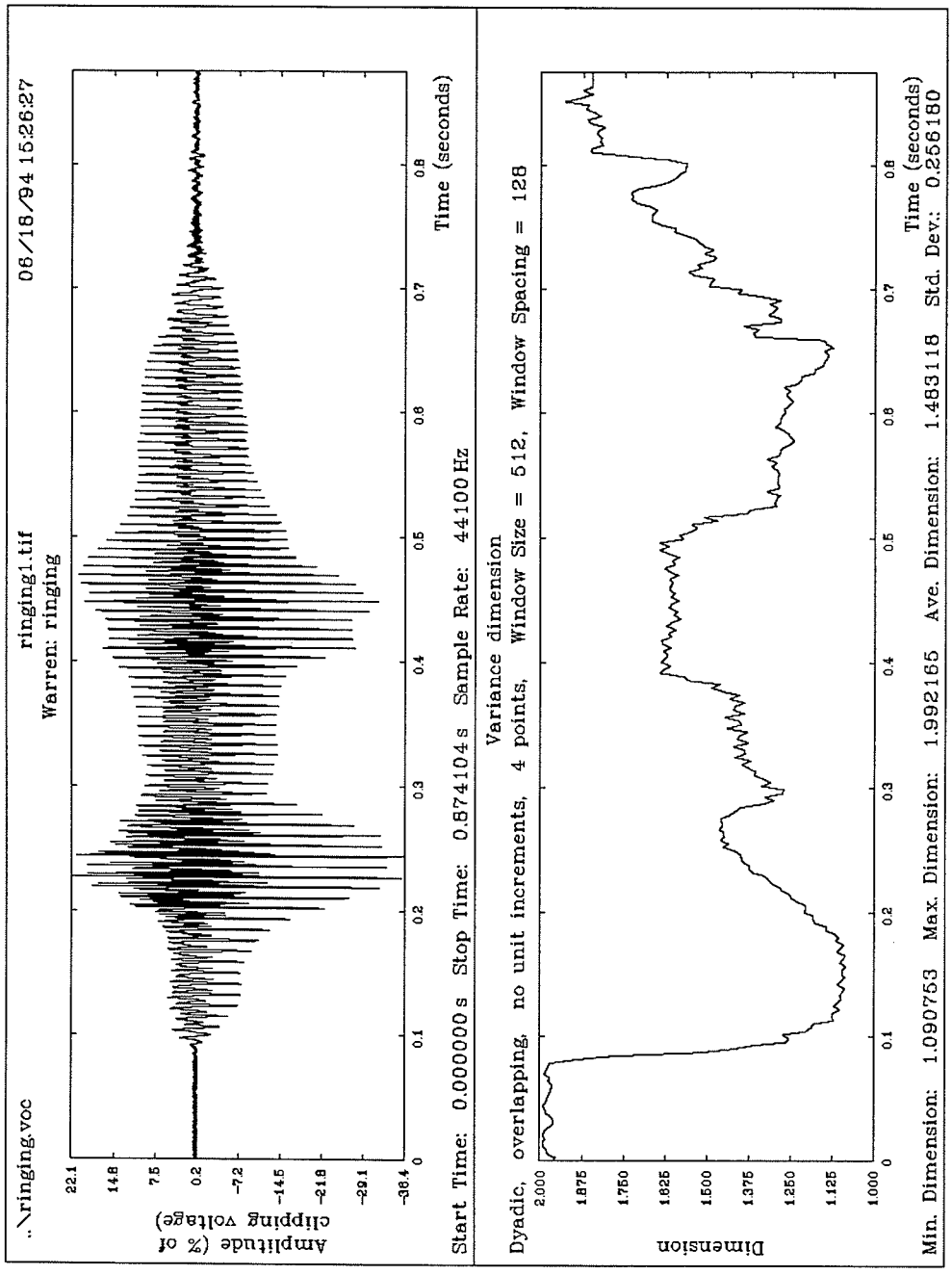
Pip, Fractal Amplification Parameters



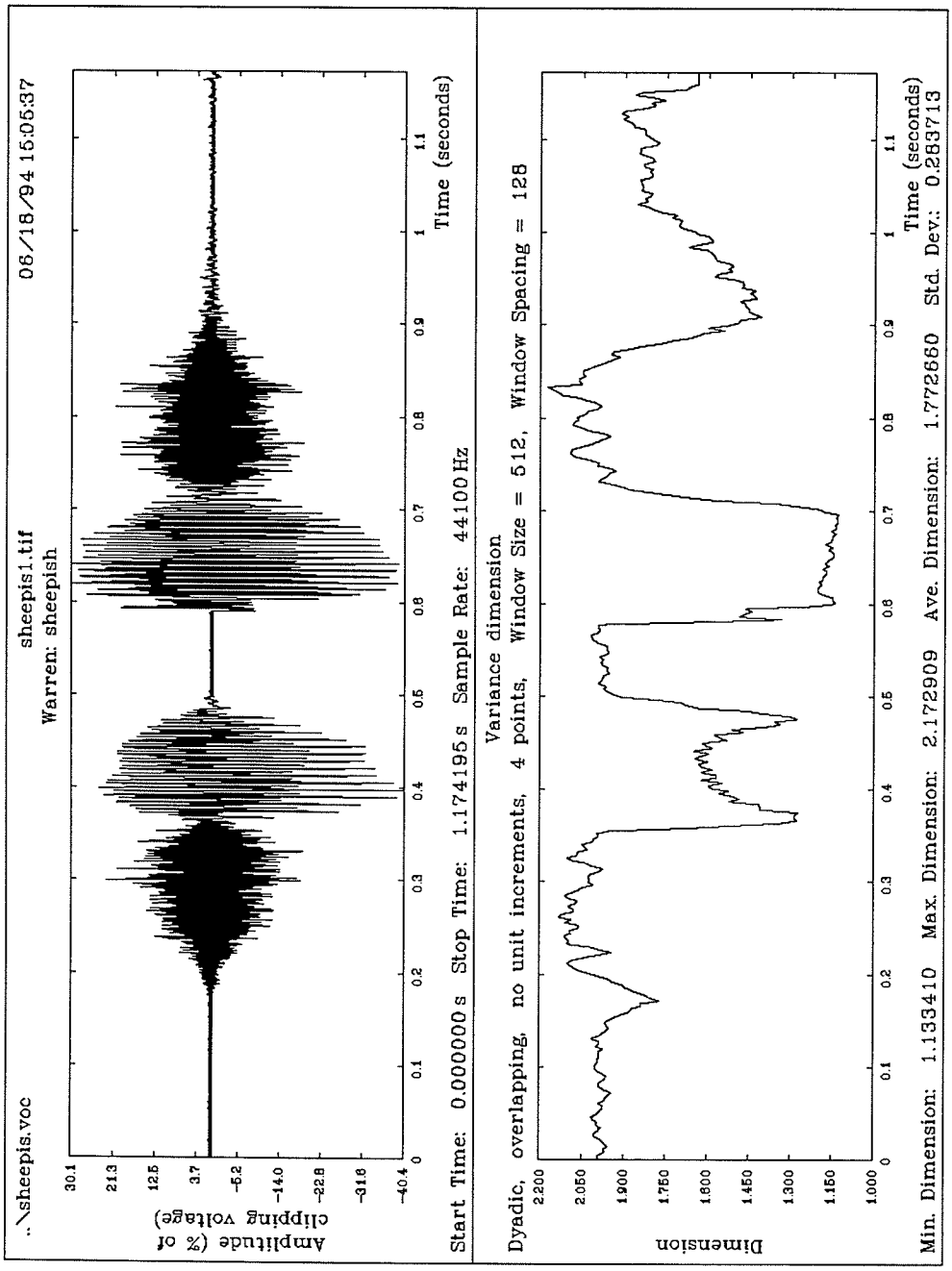
Rear, Fractal Amplification Parameters



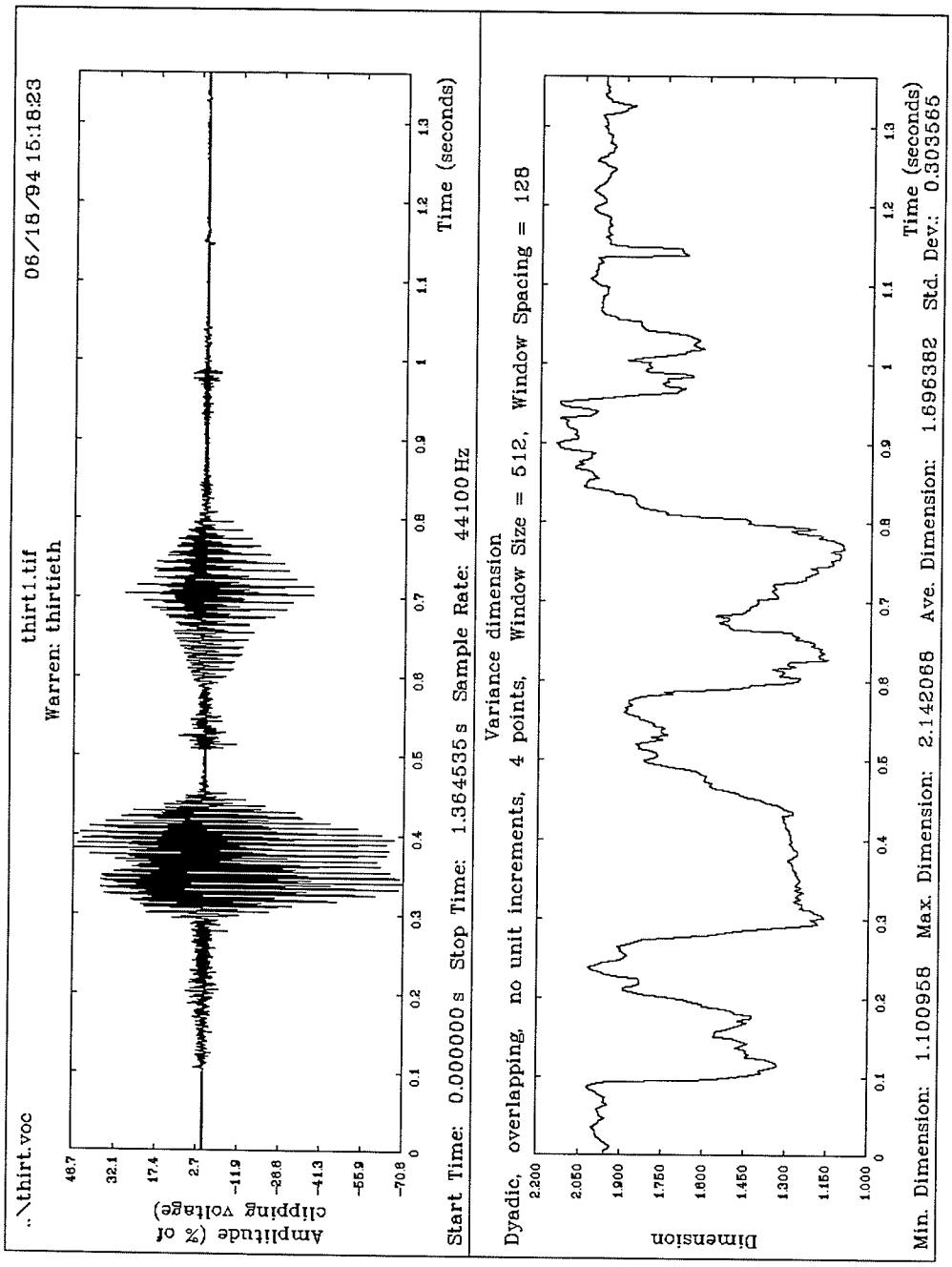
Ringing, Fractal Amplification Parameters



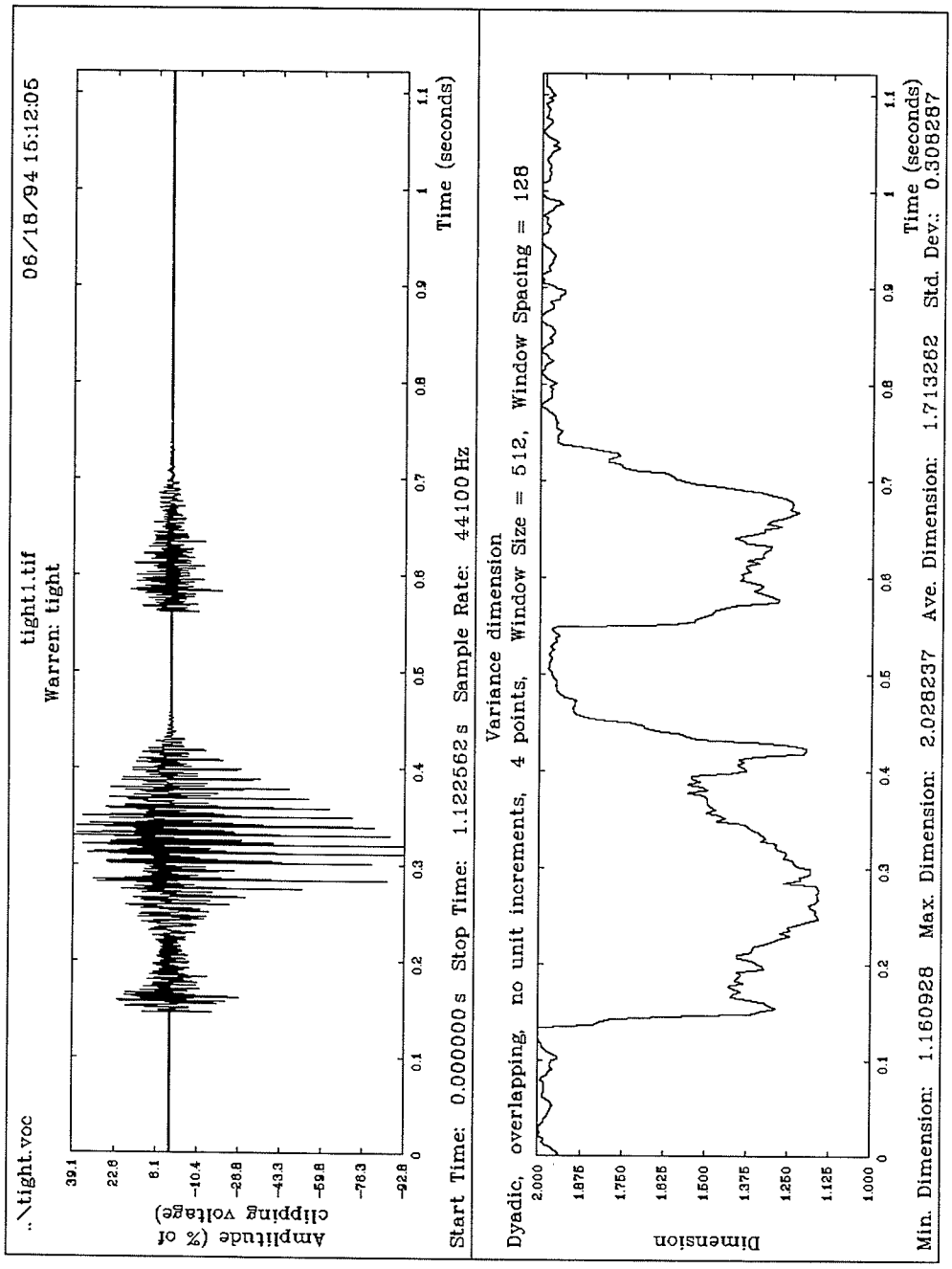
Sheepish, Fractal Amplification Parameters



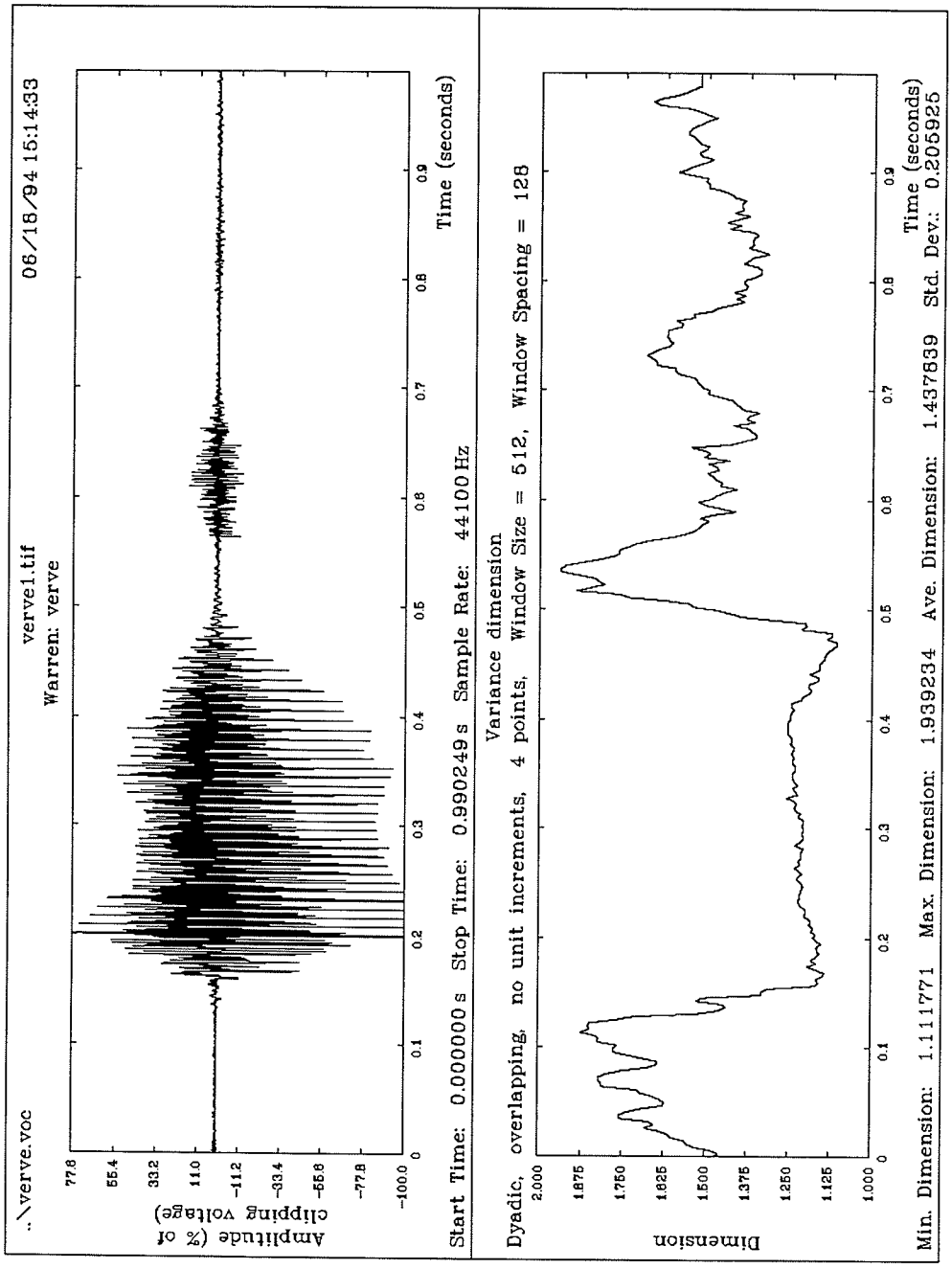
Thirtieth, Fractal Amplification Parameters



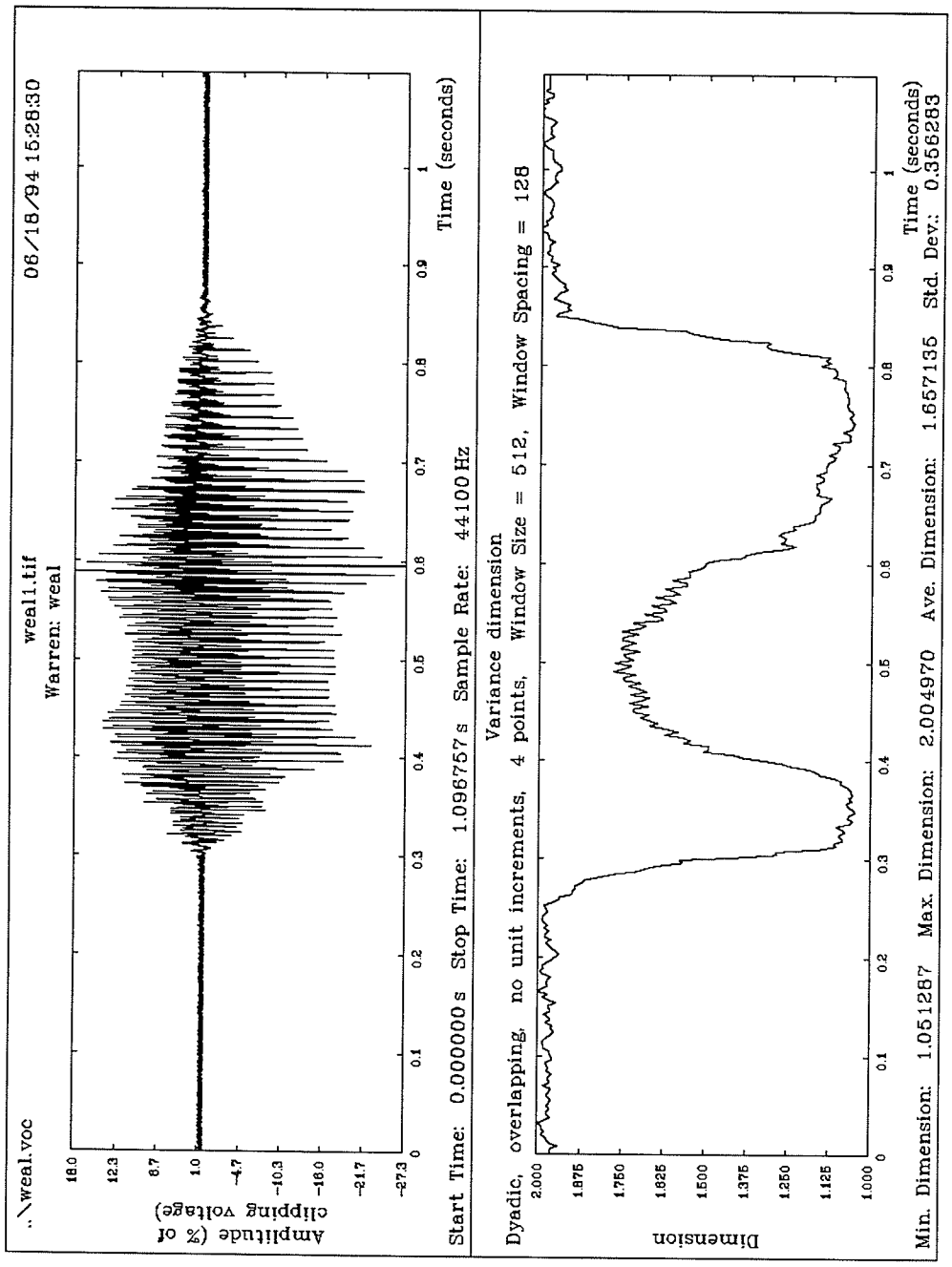
Tight, Fractal Amplification Parameters



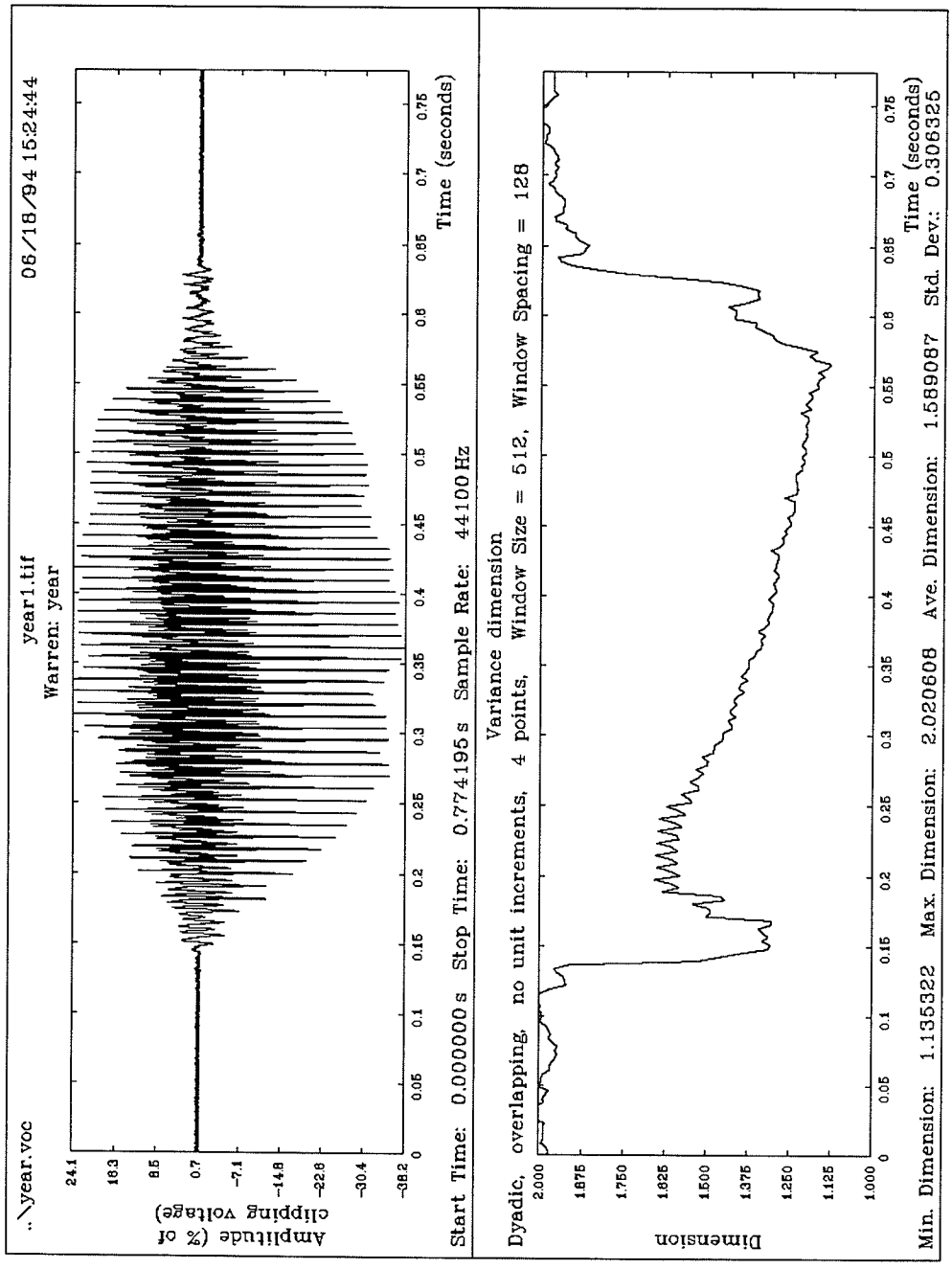
Verve, Fractal Amplification Parameters



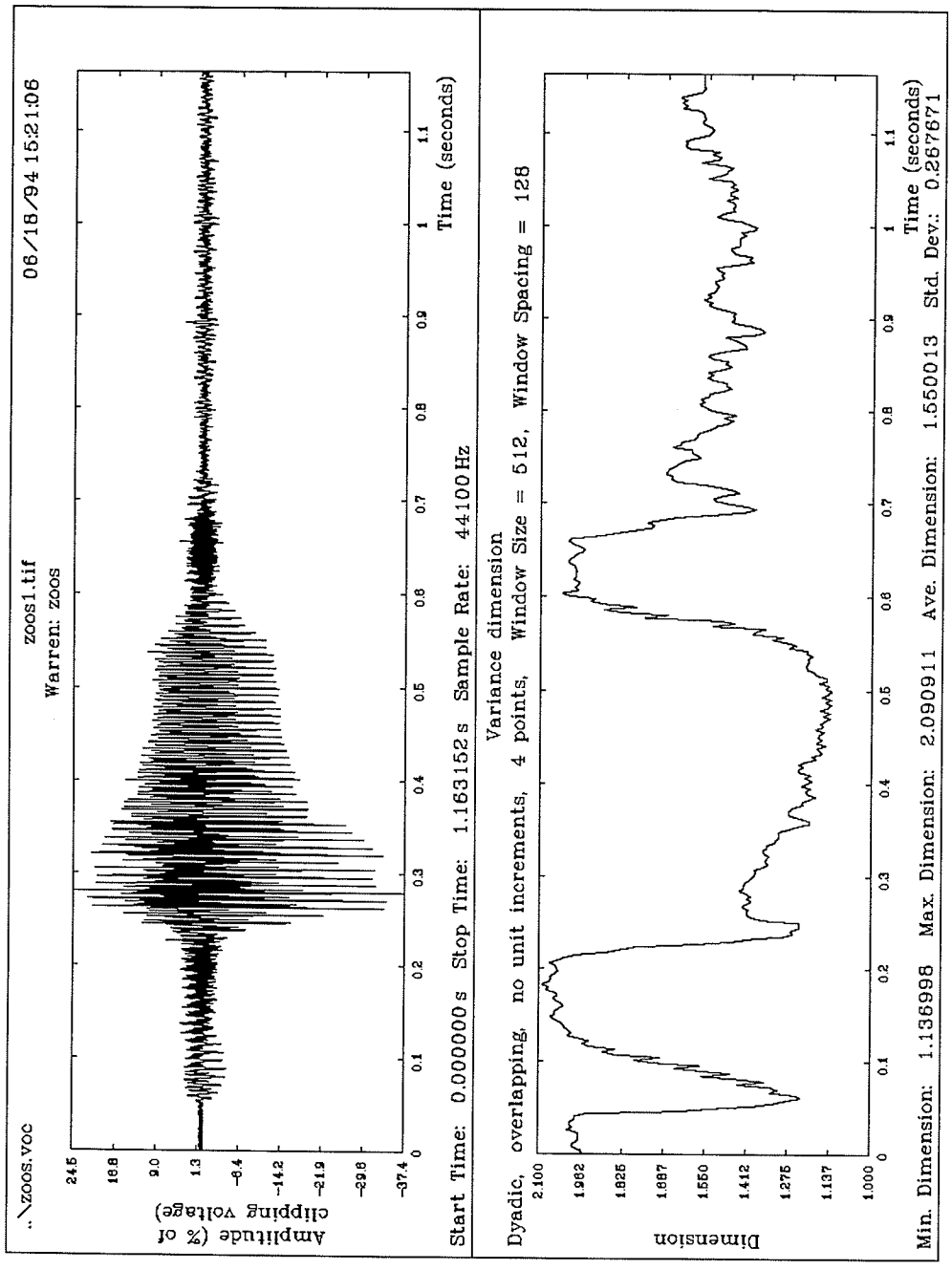
Weal, Fractal Amplification Parameters



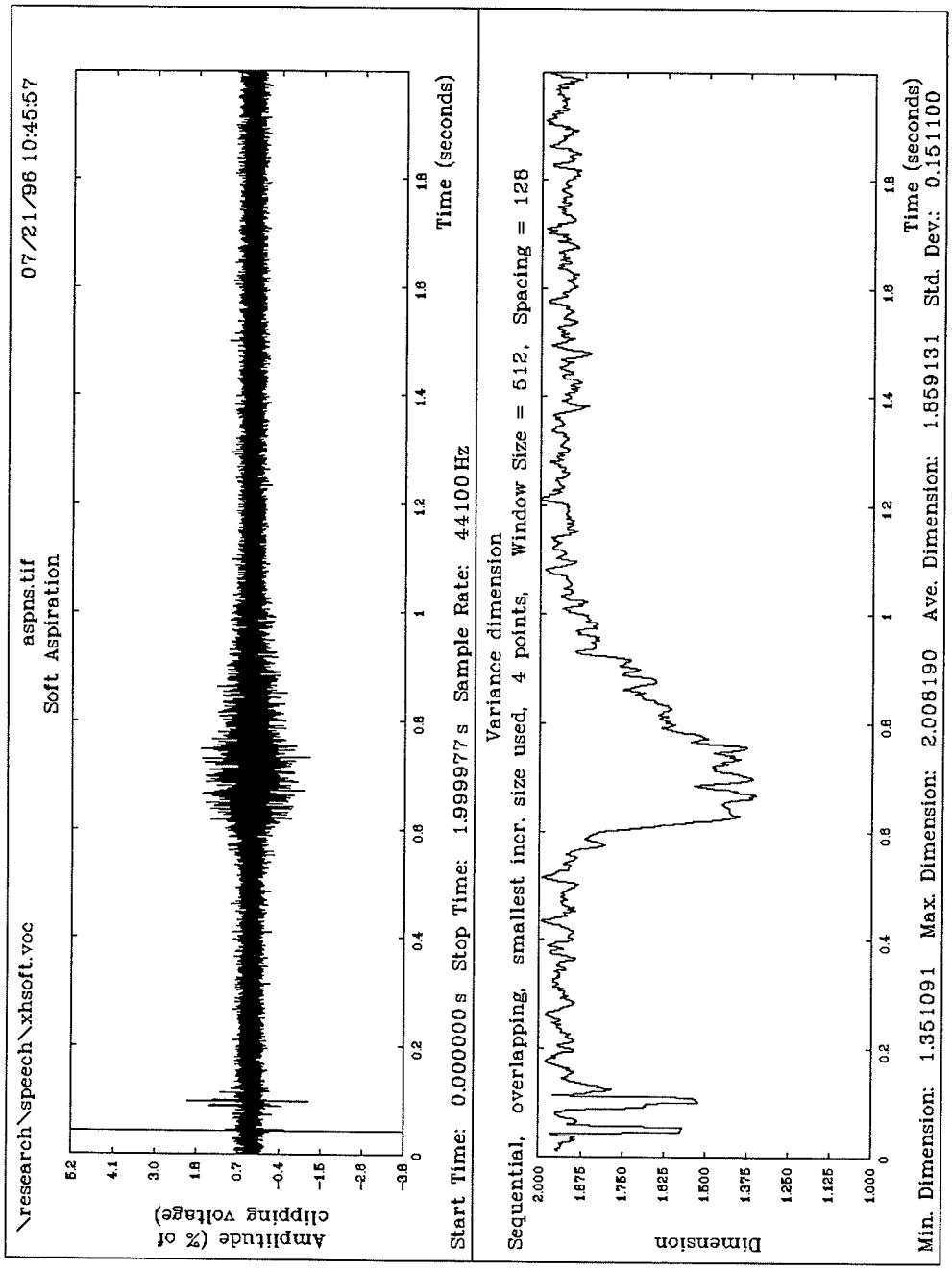
Year, Fractal Amplification Parameters



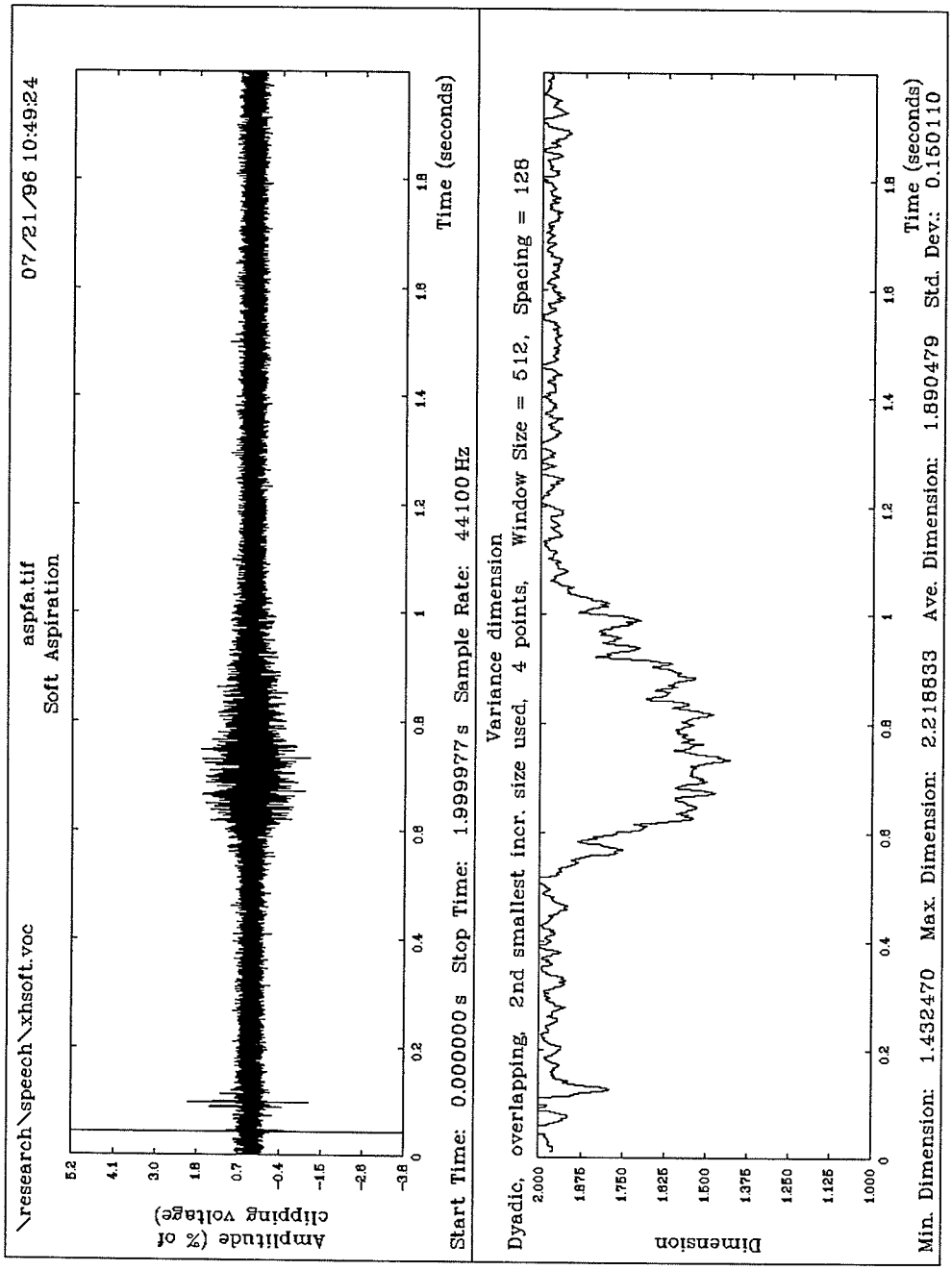
Zoos, Fractal Amplification Parameters



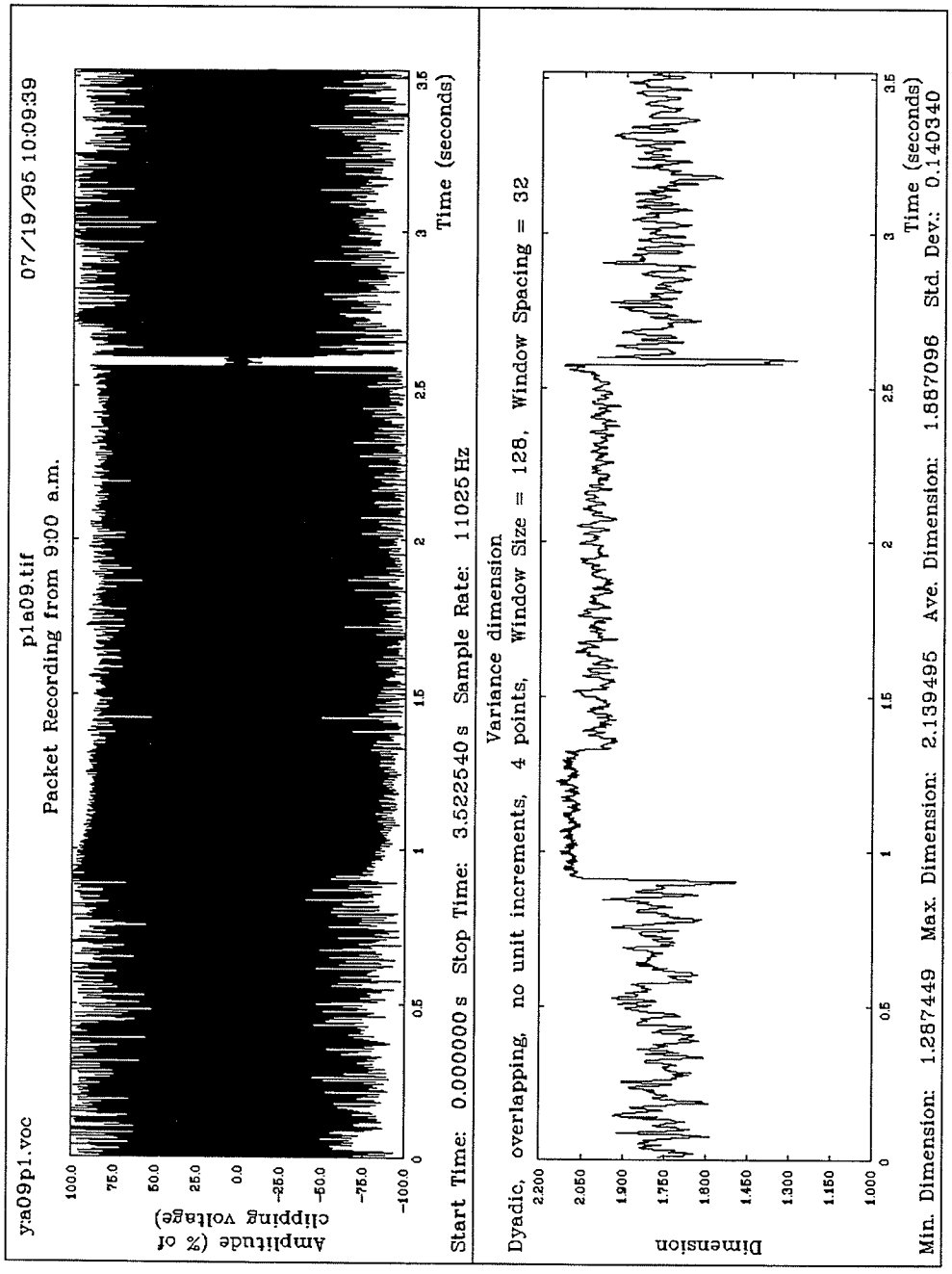
An aspiration, Noise Separation Parameters



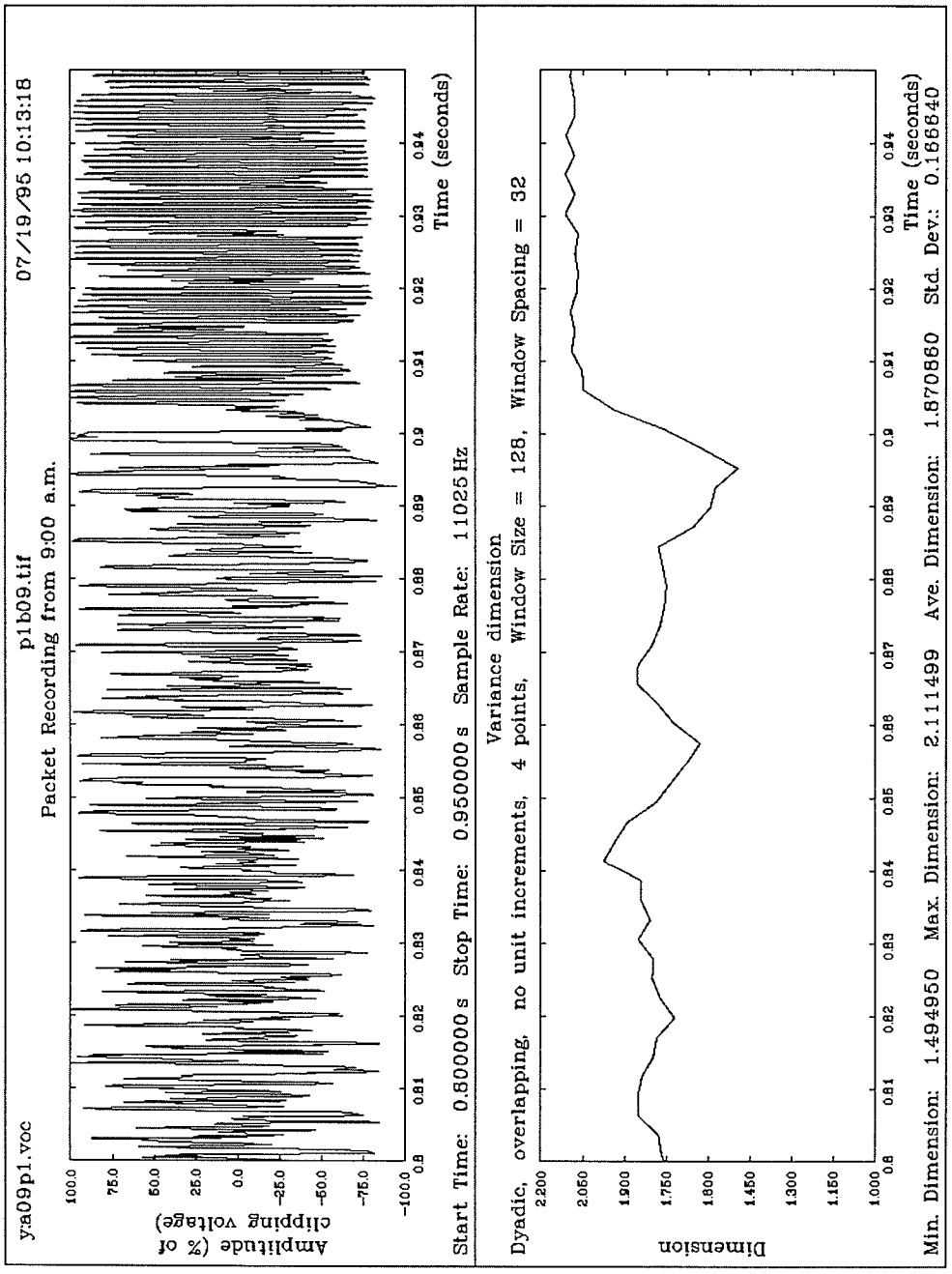
An aspiration, Fractal Amplification Parameters



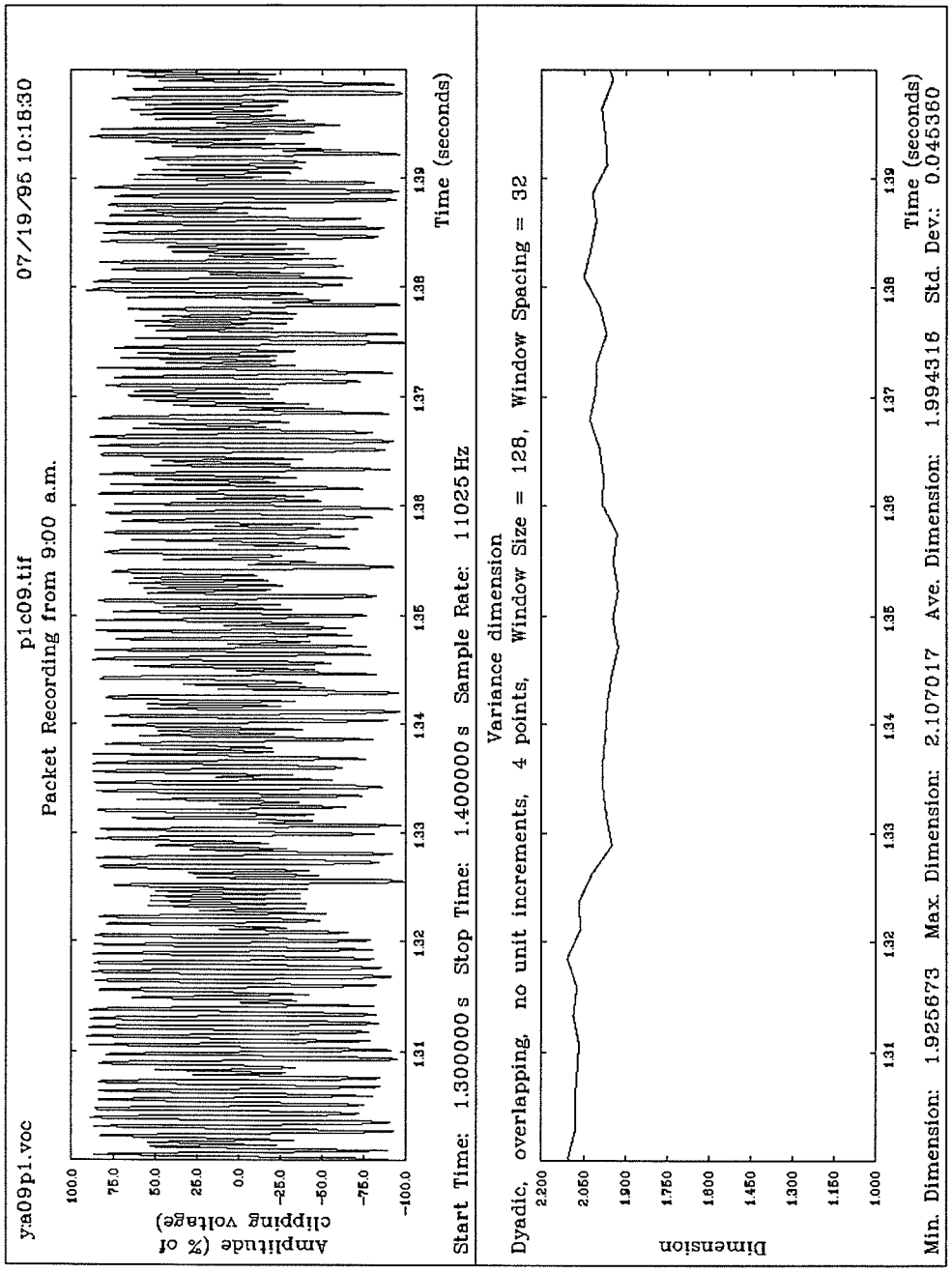
Sample 9:00 a.m. packet radio signal transmission, Fractal Amplification Parameters



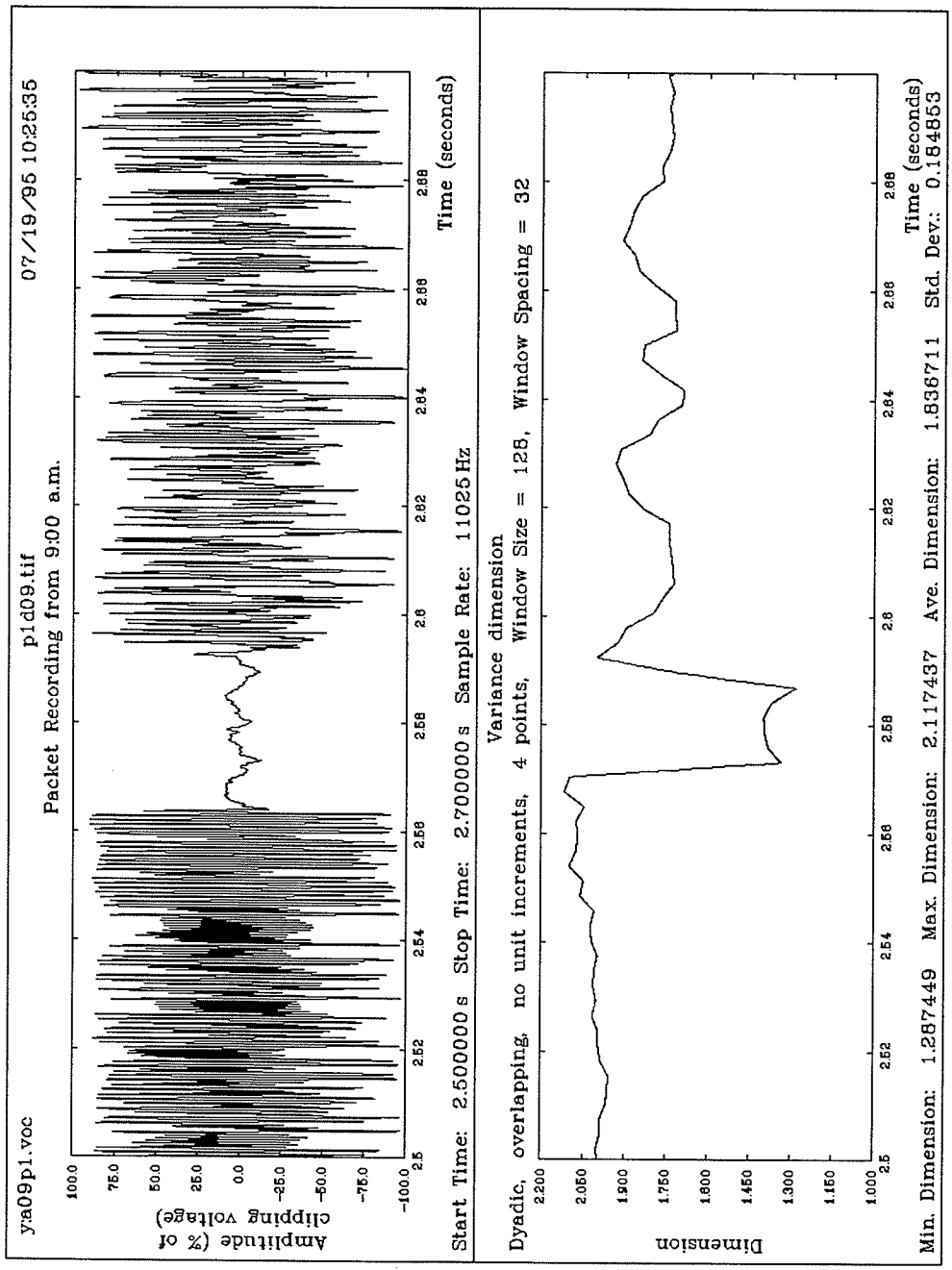
Transition between background noise and packet signal.



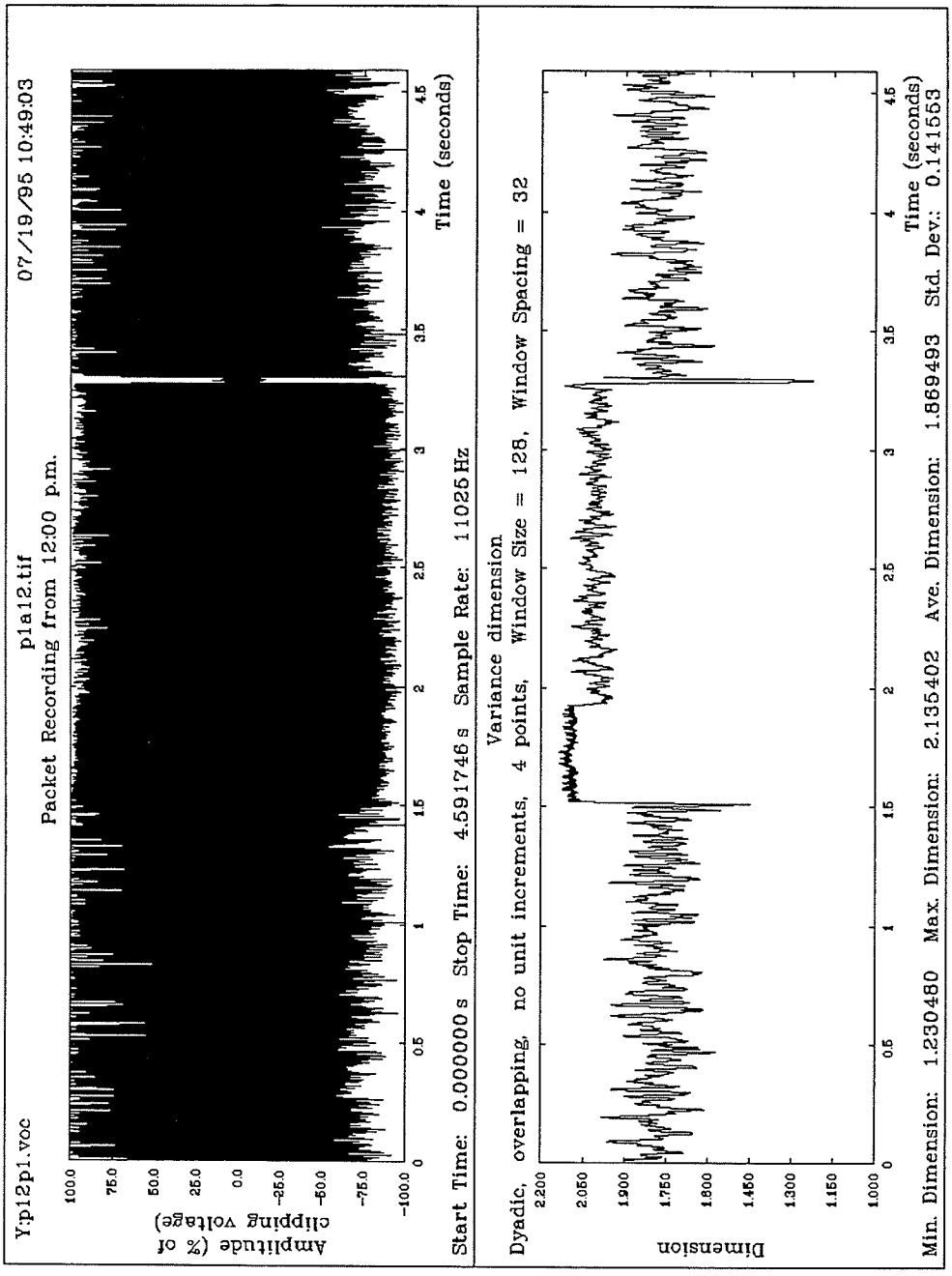
Transition between packet signal and transmitter noise.



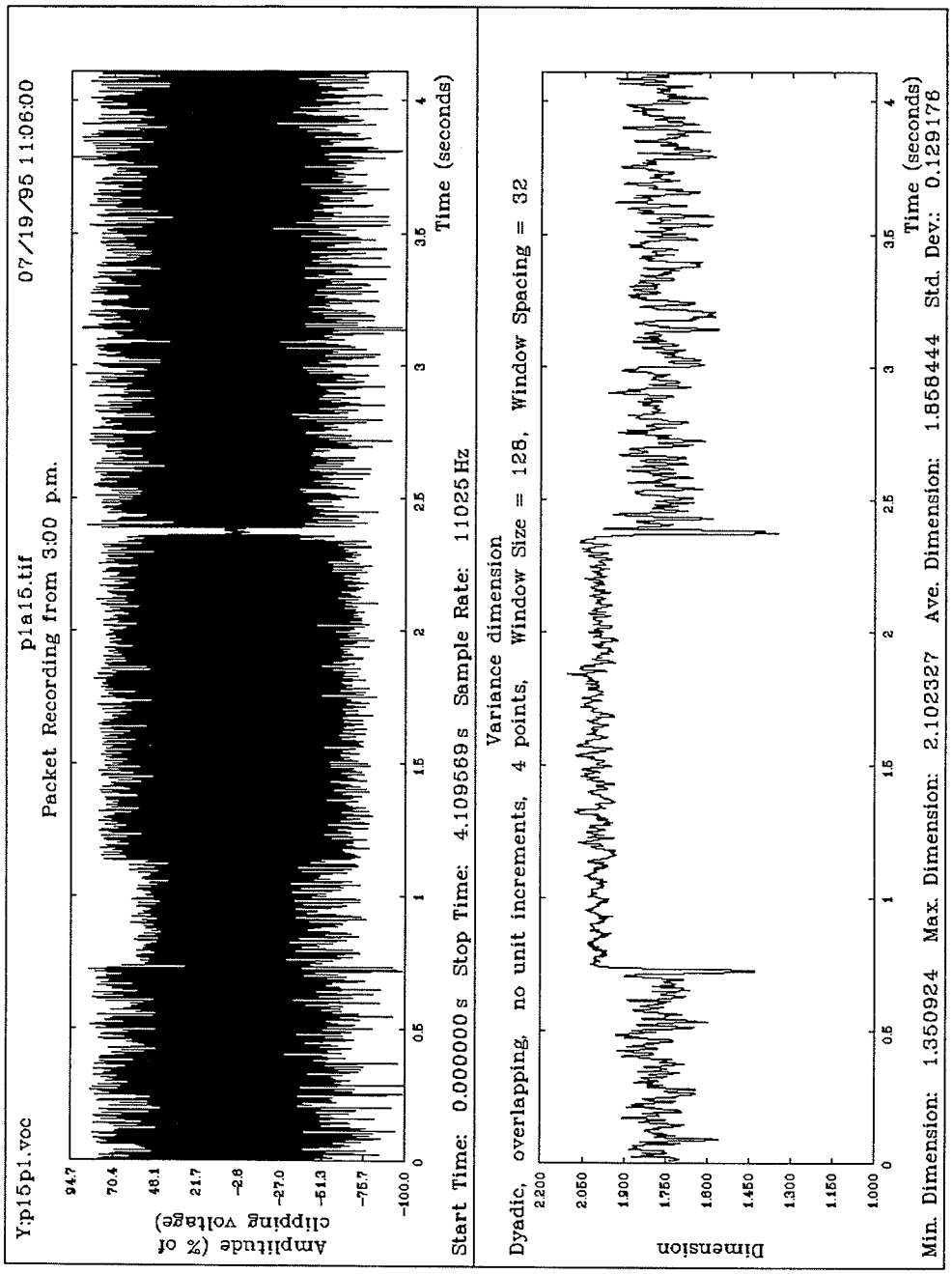
Transition between transmitter noise and background noise.



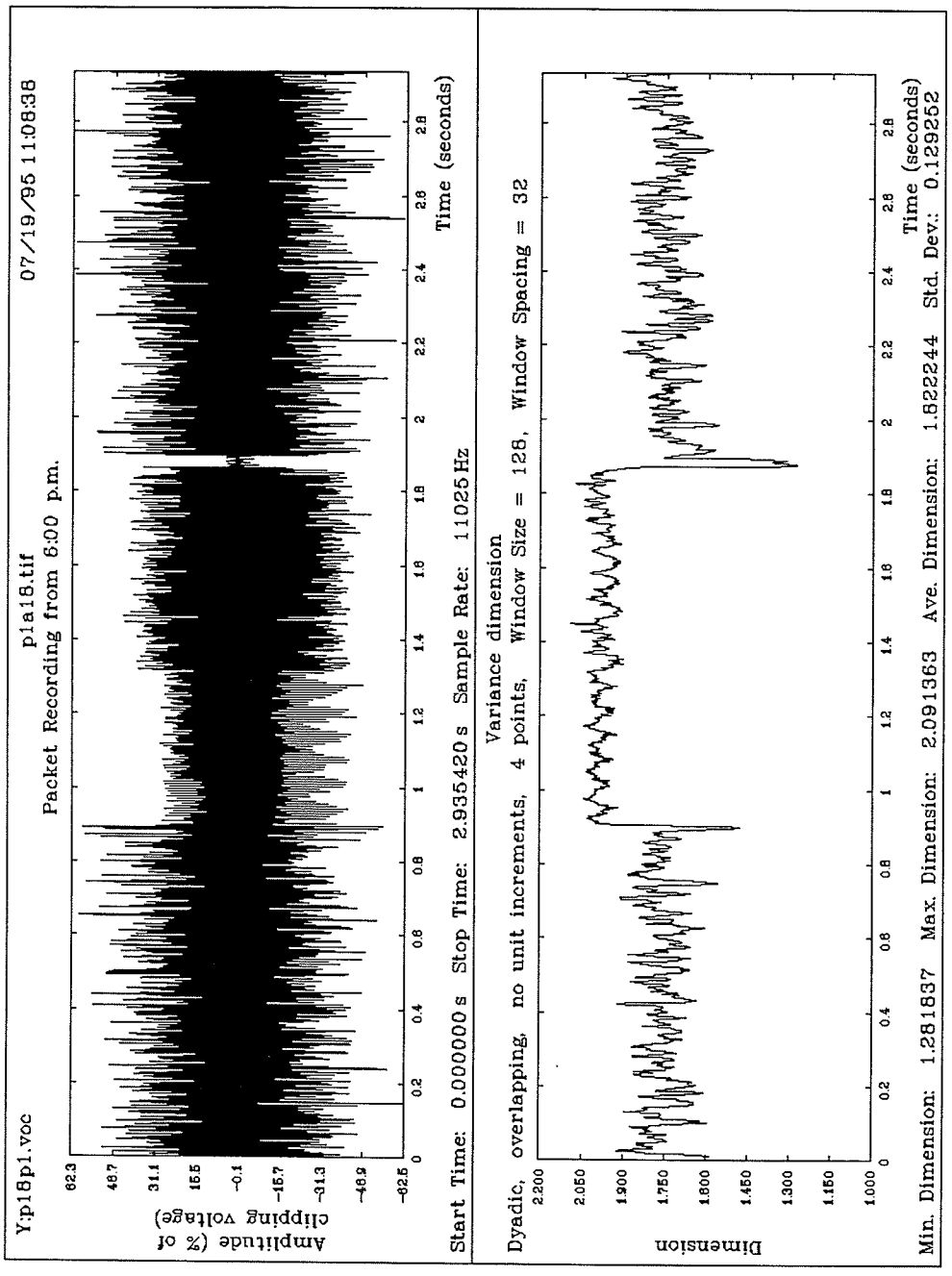
Sample 12:00 p.m. packet radio signal transmission, Fractal Amplification Parameters



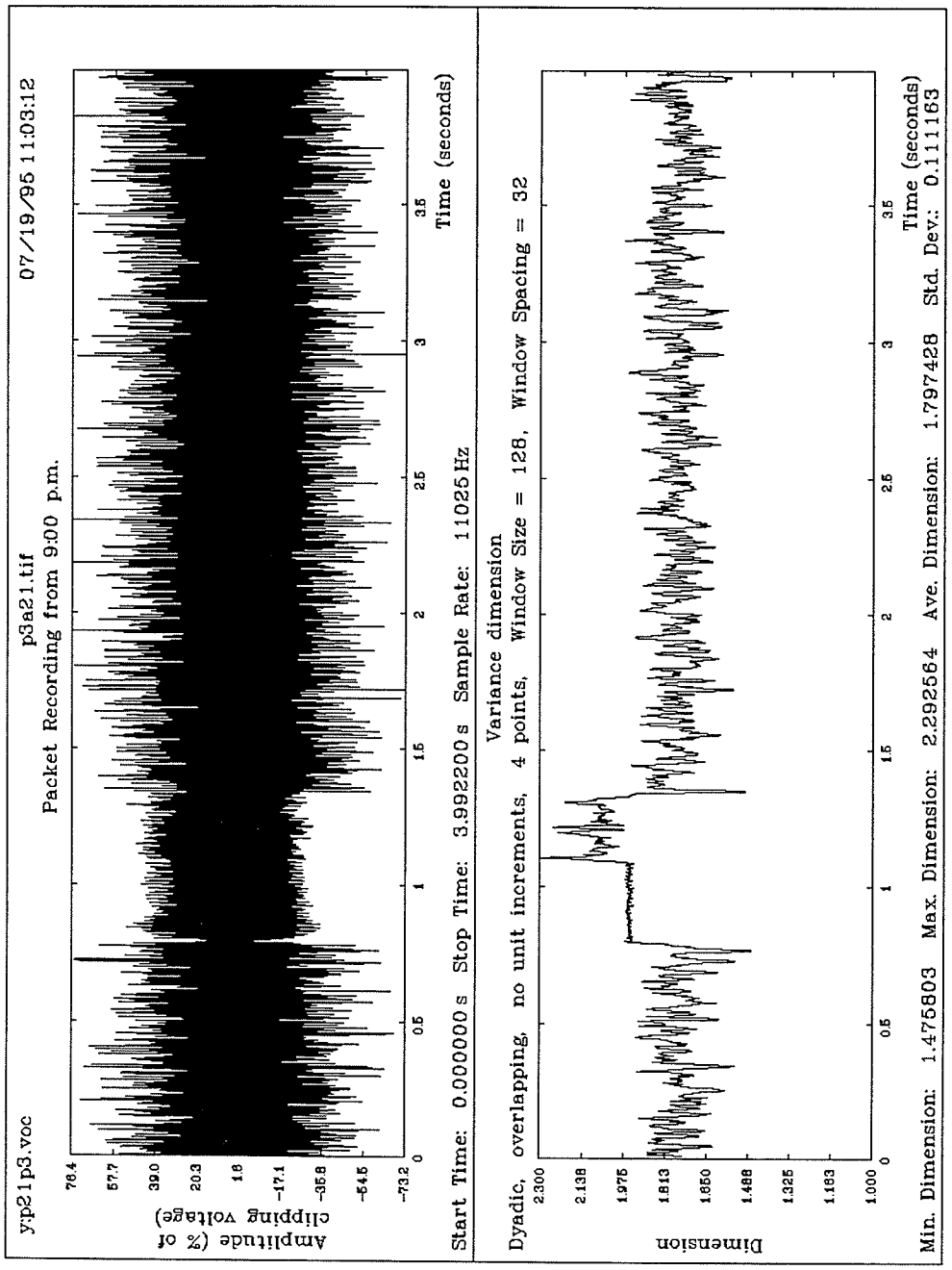
Sample 3:00 p.m. packet radio signal transmission, Fractal Amplification Parameters



Sample 6:00 p.m. packet radio signal transmission, Fractal Amplification Parameters



Sample 9:00 p.m. packet radio signal transmission, Fractal Amplification Parameters



Appendix B
Program Listings

B.1 Listing of std.h

```
//std.h

#include <stdio.h>
#include <io.h>
#include <math.h>
#include <time.h>
#include <graphics.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>

#include "xmsif.hpp"

#include "sgrdef.h"
#include "sndgrpro.h"
```

B.2 Listing of sndgrpro.h

```
//sndgrpro.h

//function prototypes

void SetPortAll(void);
void SetPortTop(void);
void SetPortMiddle(void);
void SetPortBottom(void);

void SetNormal(void);
void SetEnlarge(void);

void SaveTmpScreen(void);
void Enlarge(void);
void RedrawScreen(void);
void RestoreTmpScreen(void);

void MainWindow(void);
void ProcessData(void);
void CleanXMM(void);
int huge detect(void);
void SetDefaults(void);
void main(void);

void GetFileName(void);
void CouldNotOpen(void);
void InvalidBlock(void);
void NotVocFile(void);
```

```
void CouldNotAlloc(void);
void FileLoaded(void);

void ReadHeader(char *f_nvf);
void ReadBlockHeader(unsigned char *type, long int *size, unsigned\
    int *rate);

void LoadVocFile(char *f_voice, char *f_cno);
void LoadRaw1File(char *f_voice, char *f_cno);
void LoadRaw8File(char *f_voice, char *f_cno);
void LoadRaw16File(char *f_voice, char *f_cno);

void DrawSpeechFrame(void);
void WriteFileName(void);
void WriteFileDesc(void);

void WriteVert(void);
void WriteTime(void);
void DrawHash(double samp_inc);
void DrawSpeech(void);

void DisplayVocFile(void);
void SaveScreen(void);

void ShowMenu_1(void);
void ProcessMenu_1(char *f_quit, int *new_menu, char *f_voice);

void ShowMenu_2(void);
void ProcessMenu_2(char *f_quit, int *new_menu, char *f_voice);

void ShowMenu_3(void);
void ProcessMenu_3(char *f_quit, int *new_menu, char *f_voice);

void ShowFractalMenu_1(void);
void ShowAnalysisMenu_1(void);
void ShowRangesMenu_1(void);

void CalculateDim(int dim);
void DisplayDim(void);

void GetVarianceParameters(char *finished);
void GetBoxParameters(char *finished);

double vardim(short int huge *sp);
double boxdim(short int huge *sp);
double lsr(double *y, double *x, unsigned short z, unsigned short\
   maxpair);

void Buildf(unsigned short far *f);
void ProcessBox(void);
void ProcessVariance(void);

void DrawDimFrame(void);
void WriteDimTitle(void);
```

```
void WriteVertDim(void);
void WriteTimeDim(void);
void DrawHashDim(double samp_inc);
void DrawDim(void);

void GetNewInt(double *x);
void GetNewDouble(double *x);
void CheckVWinRange(double x, char * valid);
void CheckVIntRange(double x, char * valid);
void CheckVLSRRange(double x, char * valid);
void CheckBWinRange(double x, char * valid);
void CheckBIntRange(double x, char * valid);
void CheckBLSRRange(double x, char * valid);

void OutofRange(void);

void GetTimeRange(void);
void GetVoiceRange(void);
void autoscale(void);
void GetDimRange(void);
void ProcessRanges(void);

void WriteVoiceInfo(void);
void WriteDimInfo(void);
void GetMinMaxDim(void);

void FileError(void);
void GetVoiceName(char *name);
void WriteVocFile(char *outfile);
void SaveVoice(void);

double ran2(long *idum);
double gasdev(long *idum);

void FindPeak(long * peak);
void CalcWindowSignalPower(long peak, double * swp);
void GetSNR(double * snr);
void GetNoiseLevel(double * n_l);
void AddNoise(unsigned long size, double n_l, long *idum);
void LevelTooHigh(void);
void Noise(void);

void DimAbove(void);
void DimBelow(void);

void GetThreshold(double * th);
void DispAbove(long inum);
void DispBelow(long inum);
```

B.3 Listing of sgrdef.h

```
//sgrdef.h
```

```
#define LEFT 0          //left of screen
#define RIGHT 1023      //right of screen
#define TOP 0           //top of screen
#define BOTTOM 767      //bottom of screen

#define SWLn 85         //speech waveform box normal left
#define SWTn 52         //speech waveform box normal top
#define SWRn 970        //speech waveform box normal right
#define SWBn 212        //speech waveform box normal bottom

#define DWLn 85         //dimension waveform box normal left
#define DWTn 52         //dimension waveform box normal top
#define DWRn 970        //dimension waveform box normal right
#define DWBn 212        //dimension waveform box normal bottom

#define SWLe 85         //speech waveform box enlarged left
#define SWTe 52         //speech waveform box enlarged top
#define SWRe 970        //speech waveform box enlarged right
#define SWBe 325        //speech waveform box enlarged bottom

#define DWLe 85         //dimension waveform box enlarged left
#define DWTe 52         //dimension waveform box enlarged top
#define DWRe 970        //dimension waveform box enlarged right
#define DWBe 325        //dimension waveform box enlarged bottom

#define BOX1_Bn 270     //bottom of voice display box normal
#define BOX2_Bn 540     //bottom of dimension display box normal

#define BOX1_Be 383     //bottom of voice display box enlarged
#define BOX2_Be 767     //bottom of dimension display box enlarged


/*  for vga
#define LEFT 0
#define RIGHT 639
#define TOP 0
#define BOTTOM 479

#define SWL 66
#define SWT 33
#define SWR 608
#define SWB 133

#define BOX1_B 169
#define BOX2_B 339   */

struct var_struct
   {
      int winsize;          //window size
      int interval;         //interval between dimension calculations
      int lsrpoints;        //number of points to get for least sqrs
      int overlap;          //flag for overlaping intervals
      int dyadic;           //flag for dyadic intervals
      int unit;             //flag for unit displacement
```

```
   };

extern var_struct s_Var;  //Variance dimension parameters

struct box_struct
   {
      int winsize;         //window size
      int interval;        //interval between dimension calculations
      int lsrpoints;       //number of points to get for least sqrs
   };

extern box_struct s_Box;  //Variance dimension parameters

struct info_struct
   {
      double start_time;   //starting time displayed
      double stop_time;    //stopping time displayed
      double minvoice;     //minimum voice value displayed
      double maxvoice;     //maximum voice value displayed
      char autoscale;      //autoscaling flag
      double mindim;       //minimum dimension value displayed
      double maxdim;       //maximum dimension value displayed
   };

extern info_struct coord; //coordinate structure

extern double g_mindim;    //minimum dimension
extern double g_maxdim;    //maximum dimension
extern double g_avedim;    //average dimension
extern double g_stddev;    //standard deviation
extern long g_countd;      //dimension sample count

extern FILE *V_FP;         //voice file pointer

extern char far *vfn;      //voice file name
extern char far *bfn;      //backup voice file name
extern char far *tfn;      //tiff file name
extern char far *fdt;      //fractal dimension title

extern int SWL;            //speech waveform box left
extern int SWT;            //speech waveform box top
extern int SWR;            //speech waveform box right
extern int SWB;            //speech waveform box bottom

extern int DWL;            //dimension waveform box left
extern int DWT;            //dimension waveform box top
extern int DWR;            //dimension waveform box right
extern int DWB;            //dimension waveform box bottom

extern int BOX1_B;         //speech display box bottom
extern int BOX2_B;         //dimension display box bottom


extern short int far * voice;   //voice data
```

```
extern int voice_handle;         //voice handle
extern int dim_handle;           //dimension handle
extern long int g_voice_len;       //length of voice data
extern unsigned long g_voice_rate; //sampling rate
extern double g_tot_time;          //duration of voice data

extern int g_active_dim;           //active dimension measure
extern int menu_number;            //current menu number
extern int g_use_time;            //time / sample number flag
```

B.4 Listing of ran2def.h

```
//ran2def.h

#define IM1 2147483563L
#define IM2 2147483399L
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014L
#define IA2 40692L
#define IQ1 53668L
#define IQ2 52774L
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
```

B.5 Listing of normdef.h

```
//normdef.h

#define PI 3.14159265359
```

B.6 Listing of sndplot.cpp

```
//sndplot.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

//globals

var_struct s_Var; //structure containing Variance Dimension Parameters


box_struct s_Box; //structure containing Box Dimension Parameters
```

```
info_struct coord; //structure containing display coordinate info

// dimension statistics variables
double g_mindim;  //minimum dimension
double g_maxdim;  //maximumm dimension
double g_avedim;  //average dimension
double g_stddev;  //standard deviation of dimension
long g_countd;    //counter for number of dimension samples

//File infomation

FILE *V_FP;  //input voc file pointer

char far *vfn;  //voc file name
char far *bfn;  //backup of voc file name
char far *tfn;  //tiff file name
char far *fdt;  //fractal dimension title

//display coordinates

int SWL;  //coordinate of left side of speech window
int SWT;  //coordinate of top side of speech window
int SWR;  //coordinate of right side of speech window
int SWB;  //coordinate of bottom side of speech window

int DWL;  //coordinate of left side of dimension window
int DWT;  //coordinate of top side of dimension window
int DWR;  //coordinate of right side of dimension window
int DWB;  //coordinate of bottom side of dimension window

int BOX1_B; //bottom of speech display box
int BOX2_B; //bottom of dimension display box

short int far * voice; //voice pointer
int voice_handle;      //handle for voice data in extended memory
int dim_handle;        //handle for dimension data in extended memory
long int g_voice_len;  //length of voc file (samples)
unsigned long g_voice_rate;  //sampling rate of voc file
double g_tot_time;   //duration of voc file (seconds)

int g_active_dim; //dimension type selection parameter
int menu_number=1; //initial menu location
int g_use_time=1; //time / sample number axis flag

//Processdata
//This routine handles the main menu of the program calling
//routine that display the appropriate menu items base on the current
//valid data

void ProcessData(void)
{

   int new_menu_number; //next menu to display
   char f_quit = 0;  //flag to terminate routine
```

```
    char f_voice_valid = 0; //voice data valid flag



    while(f_quit==0)
    {
       //initialize next menu
       new_menu_number = menu_number;

       switch (menu_number)
       {
          case 1: //voice data not valid
          {
             SetPortBottom();
             clearviewport();
             ShowMenu_1();
             ProcessMenu_1(&f_quit,&new_menu_number,&f_voice_valid);
             break;
          }
          case 2: //voice data valid dimension not calculated
          {
             SetPortBottom();
             clearviewport();
             ShowMenu_2();
             ProcessMenu_2(&f_quit,&new_menu_number,&f_voice_valid);
             break;
          }
          case 3: //voice data valid dimension calculated
          {
             SetPortBottom();
             clearviewport();
             ShowMenu_3();
             ProcessMenu_3(&f_quit,&new_menu_number,&f_voice_valid);
             break;
          }
          default:
          {

          }
       }
       menu_number = new_menu_number; //set next menu
    }

}

//detect
//return autodetect value for SVGA driver

int huge detect(void)
{
   return(4);
}

//SetNormal
```

```
//This routine sets the size of the display boxes for normal operation

void SetNormal(void)
{
   SWL=SWLn;
   SWT=SWTn;
   SWR=SWRn;
   SWB=SWBn;

   DWL=DWLn;
   DWT=DWTn;
   DWR=DWRn;
   DWB=DWBn;

   BOX1_B=BOX1_Bn;
   BOX2_B=BOX2_Bn;
}

//SetEnlarge
//This routine set the display sizes for saving as tiff
//Menu box will not be visible

void SetEnlarge(void)
{
   SWL=SWLe;
   SWT=SWTe;
   SWR=SWRe;
   SWB=SWBe;

   DWL=DWLe;
   DWT=DWTe;
   DWR=DWRe;
   DWB=DWBe;

   BOX1_B=BOX1_Be;
   BOX2_B=BOX2_Be;
}

//SetDefaults
//The routine initialize the value of global variable and structure
//used in the program

void SetDefaults(void)
{
   voice_handle=0;
   dim_handle=0;
   g_active_dim=0;

   s_Var.winsize = 512;
   s_Var.interval =128;
   s_Var.lsrpoints = 4;
   s_Var.overlap=1;
   s_Var.dyadic=1;
   s_Var.unit=0;
```

```
    s_Box.winsize = 512;
    s_Box.interval = 128;
    s_Box.lsrpoints = 5;

    coord.autoscale = 1;
}


void main()
{
    char far *name;  //video driver name

    //allocate space for strings
    vfn = new char far [80];
    bfn = new char far [80];
    tfn = new char far [80];
    name = new char far [80];
    fdt = new char far [80];

    int gd, gm; //graphics driver and mode


    SetDefaults();
    SetNormal();
    name = "svga256";
    XMMlibinit(); //initialize extended memory manager
    gd=installuserdriver(name, detect); //install graphics driver
    gm=4;

    //start graphics mode
    initgraph(&gd, &gm,".");
    setcolor(15);
    MainWindow();

    //execute main program
    ProcessData();

    //shutdown program
    closegraph();
    CleanXMM();
    delete[] fdt;
    delete[] vfn;
    delete[] bfn;
    delete[] tfn;
    delete[] name;
}
```

B.7 Listing of sndgr.cpp

```
// sndgr.cpp

#ifndef MYGLOBAL
```

```
   #define MYGLOBAL
   #include "std.h"
#endif

//CleanXMM
//This routine releases allocated extended memory

void CleanXMM(void)
{
   if (voice_handle != 0)
   {
      XMMfree(voice_handle);
   }
   if (dim_handle != 0)
   {
      XMMfree(dim_handle);
   }
}

//SetPortAll
//This routine sets the viewport to the entire screen

void SetPortAll(void)
{
   setviewport(LEFT,TOP,RIGHT,BOTTOM,1);
}

//SetPortTop
//This routine sets the viewport to the top (voice display) box

void SetPortTop(void)
{
   setviewport(LEFT+1,TOP+1,RIGHT-1,BOX1_B-1,1);
}

//SetPortMiddle
//This routine sets the viewport to the middle (dimension display) box

void SetPortMiddle(void)
{
   setviewport(LEFT+1,BOX1_B+1,RIGHT-1,BOX2_B-1,1);
}

//SetPortBottom
//This routine sets the viewport to the Bottom (menu display) box

void SetPortBottom(void)
{
   setviewport(LEFT+1,BOX2_B+1,RIGHT-1,BOTTOM-1,1);
}

//MainWindow
//This routine draws the outline for the main display boxes
```

```
void MainWindow(void)
{
   cleardevice();
   SetPortAll();
   rectangle(LEFT,TOP,RIGHT,BOTTOM);
   line(LEFT,BOX1_B,RIGHT,BOX1_B);
   line(LEFT,BOX2_B,RIGHT,BOX2_B);
}

//GetFileName
//This routine prompts the uses to input a voice filename and
//stores the result in the voc filename string (vfn)

void GetFileName(void)
{
   int i=0;            //string length
   int j;              //counter variable
   char ch_in;         //input character
   char done=0;        //flag to indicate name entered
   char far *out_str; //string for output to screen
   char far *ch_str;  //typed character string
   char far *last_ch; //last character in filename (for screen update
                       //on backspace
   int tw;             //textwidth

   //allocate string space

   out_str = new char far [80];
   ch_str = new char far [80];
   last_ch = new char far [80];

   //set text display parameters
   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   //backup current filename in case invalid name entered
   for(j=0;j<80;j++)
   {
      bfn[j] = vfn[j];
   }

   //get filename

   out_str="Enter Name of Voc File to Open: ";
   moveto(2,202);
   outtext(out_str);

   while (done == 0)
   {
      ch_in = getch();
      switch (ch_in)
      {
         case '\r': //enter selected
         {
```

```
            done = 1;
            vfn[i]=0;
            break;
         }
         case '\b':  //backspace selected
         {
            if (i > 0)
            {
               i--;
               last_ch[0] = vfn[i];
               last_ch[1]=0;
               tw=textwidth(last_ch);
               moverel(-tw,0);
               setcolor(0);
               outtext(last_ch);
               setcolor(15);
               moverel(-tw,0);
            }
            break;
         }
         default:  //character entered
         {
            vfn[i]=ch_in;
            ch_str[0]=ch_in;
            ch_str[1]=0;
            outtext(ch_str);

            i++;
         }
      }

   }

   //release string allocations

   delete[] out_str;
   delete[] last_ch;
   delete[] ch_str;
}

//CouldNotOpen
//Error Message Routine

void CouldNotOpen(void)
{

   char far *out_str;

   out_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
```

```
    setusercharsize(1,2,1,2);

    out_str="Could Not Open File";
    moveto(2,22);
    outtext(out_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }
    delete[] out_str;

}

//NotVocFile
//Error Message Routine

void NotVocFile(void)
{

    char far *out_str;

    out_str=new char far [80];

    SetPortBottom();
    clearviewport();

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="Not a Voc File";
    moveto(2,22);
    outtext(out_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }
    delete[] out_str;
}

//InvalidBlock
//Error Message Routine

void InvalidBlock(void)
{
    char far *out_str;
```

```
    out_str = new char far [80];

    SetPortBottom();
    clearviewport();

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="Unsupported Block Type";
    moveto(2,22);
    outtext(out_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }
    delete[] out_str;

}

//CouldNotAlloc
//Error Message Routine

void CouldNotAlloc(void)
{
    char far *out_str;

    out_str=new char far [80];

    SetPortBottom();
    clearviewport();

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="File Too Large for Extended Memory";
    moveto(2,22);
    outtext(out_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }
    delete[] out_str;

}

//FileLoaded
```

```
//Successful File Loaded Message Routine

void FileLoaded(void)
{
   char far *out_str;

   out_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="File Loaded";
   moveto(2,22);
   outtext(out_str);

   out_str="Press Any Key to Continue";
   moveto(2,62);
   outtext(out_str);

   while (kbhit()==0)
   {
   }
   delete[] out_str;

}

//ReadHeader
//This routine reads the header of the voc file
//Checks first three byte for valid code

void ReadHeader(char *f_nvf)
{
   int i; //counter variable
   char temp; //read variable
   char h[26]; //header array

   for (i=0; i<26; i++)
   {
      fread(&temp,sizeof(temp),1,V_FP);
      h[i] = temp;
   }
   if (h[0]!='C'||h[1]!='r'||h[2]!='e')
   {
      *f_nvf = 1; //indicate file not valid
   }
}

//ReadBlockHeader
//This routine reads the header of an internal block in a voc file
//The type and size of the block and the sampling rate are determined
```

```
void ReadBlockHeader(unsigned char *type, long int *size,\
   unsigned long *rate)
{
   int i;                    //counter
   unsigned char temp;        //read variable
   unsigned char s1, s2, s3; //size variables
   unsigned char r1, r2;     //rate variables

   //read block type
   fread(&temp,sizeof(temp),1,V_FP);
   *type = temp;

   //read size
   fread(&s1, sizeof(s1), 1, V_FP);
   fread(&s2, sizeof(s2), 1, V_FP);
   fread(&s3, sizeof(s3), 1, V_FP);

   //read rate
   fread(&r1, sizeof(r1), 1, V_FP);
   fread(&r2, sizeof(r2), 1, V_FP);

   //calculate size
   *size = (unsigned long)s3*65536L+(unsigned long)s2*256+\
      (unsigned long)s1;
   *size = (*size - 12)/2;

   //calculate rate
   *rate = (unsigned long)r2*256+(unsigned long)r1;

   //read remaining data
   for (i=0;i<10;i++)
   {
      fread(&temp,sizeof(temp),1,V_FP);
   }
}

//LoadVocFile
//This routine reads the voc file data into extended memory

void LoadVocFile(char *f_voice,  char *f_cno)
{
   long int size;        //size of voice data
   unsigned long rate;   //sampling rate
   long int i;           //counter variable
   int j;                //counter variable
   short int far *data; //voice data pointer
   unsigned char type;   //block type
   char f_nvf=0;         //file not valid flag
   char f_allocfail=0;   //memory allocation failure flag
   unsigned long xmem;   //extended memory available
   unsigned long v_arysize;  //voice array size
   unsigned long d_arysize;  //dimension array size

   double total_time; //duration of data
```

```
//get file name

data = new short int far;
GetFileName();

//open file
V_FP = fopen(vfn,"rb");

//check if file found
if (V_FP == NULL)
{
   CouldNotOpen();
   *f_cno=1;

   for(j=0;j<80;j++)
   {
      vfn[j] = bfn[j];
   }

}
else
{
   //read file
   ReadHeader(&f_nvf);
   ReadBlockHeader(&type, &size, &rate);

   //check if file valid
   if (f_nvf==1)
   {
      NotVocFile();
      *f_cno=1;

      for(j=0;j<80;j++)
      {
         vfn[j] = bfn[j];
      }
   }

   //if if data is 16bit format
   else if (type != 9)
   {
      InvalidBlock();
      *f_cno=1;

      for(j=0;j<80;j++)
      {
         vfn[j] = bfn[j];
      }
   }
   else
   {
      if (*f_voice==1)
```

```
{
   //clear display
   XMMfree(voice_handle);
   *f_voice=0;
   SetPortTop();
   clearviewport();

   XMMfree(dim_handle);

   SetPortMiddle();
   clearviewport();
}

//set voice globals
g_voice_len = size;
g_voice_rate = rate;

//check extended memory
xmem = XMMcoreleft();

//calculate memory requirements
v_arysize=size*sizeof(short int);
d_arysize=size*sizeof(double);

//allocate extended memory for voice data
voice_handle = 0;
if (v_arysize < xmem)
{
   voice_handle = XMMalloc(v_arysize);
}
else
{
   f_allocfail = 1; //allocation failed
}

xmem = XMMcoreleft();

//allocate extended memory for dimension data
dim_handle = 0;
if (d_arysize < xmem)
{
   dim_handle = XMMalloc(d_arysize);
}
else
{
   f_allocfail = 1;  //allocation failed
}

//check if allocation failed
if (f_allocfail == 1)
{
   if (voice_handle != 0)
   {
      XMMfree(voice_handle);
```

```
            }
            if (dim_handle != 0)
            {
               XMMfree(dim_handle);
            }
            CouldNotAlloc();
            *f_cno = 1;
         }
         else
         {
            //store voice data in extended memory
            for (i=0; i<size; i++)
            {
               fread(&*data, sizeof(*data), 1, V_FP);
               XMMcopyto(sizeof(*data),data,voice_handle,\
                  i*sizeof(*data));
            }

            *f_voice = 1; //voice data valid

            //calculate total time
            total_time = (double)(g_voice_len-1)/(double)g_voice_rate;
            g_tot_time = total_time;

            coord.start_time = 0.0;    //start time for display
            coord.stop_time = g_tot_time;  //stop time for display

            //set scale coordinates
            if (coord.autoscale==0)
            {
               coord.minvoice = -100.0;
               coord.maxvoice = 100.0;
            }
            else
            {
               autoscale();
            }
            FileLoaded();
         }
      }

      fclose(V_FP);

   }
   delete data;
}

//DrawSpeechFrame
//This routine draws the frame for the speech waveform box

void DrawSpeechFrame(void)
{
   rectangle(SWL,SWT,SWR,SWB);
}
```

```
//WriteFileName
//This routine displays the filename of the voc file
//Current date and time are also displayed

void WriteFileName(void)
{
   char far *out_str; //output filename string
   char *date_str;    //date string
   char *time_str;    //time string
   //int tw;

   out_str=new char far [80];
   date_str=new char [80];
   time_str=new char [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   _strtime(time_str);
   _strdate(date_str);


   //tw = textwidth(vfn);

   moveto(20,2);
   outtext(vfn);

   moveto(500,2);
   if (BOX1_B==BOX1_Be)
   {
      outtext(tfn);
   }
   moveto(800,2);

   //display date and time
   outtext(date_str);
   outtext(time_str);

   //display file discription
   WriteFileDesc();

   delete[] out_str;
   delete[] time_str;
   delete[] date_str;

}

//WriteFileDesc
//This routine displays the content of the txt file containing the
//voc file description

void WriteFileDesc(void)
{
```

```
    char far *out_str;  //output string
    char far *filename; //text file name
    int tw;             //text width
    FILE * desc_file;   //text file pointer
    int i;              //counter variable for text file name length
    int len;            //length of text

    out_str=new char far [200];
    filename=new char far [80];

    //construct text file name
    i=0;
    filename[i]=vfn[i];
    while (vfn[i]!=0 && i<79)
    {
       i++;
       filename[i]=vfn[i];
    }
    i -= 3;
    filename[i++] = 't';
    filename[i++] = 'x';
    filename[i++] = 't';
    filename[i++] = 0;

    desc_file = fopen(filename, "rt");

    if (desc_file == NULL)
    {
       //text file does not exist
       out_str = "Description Unavailable";

    }
    else
    {
       len=fread(out_str,1,200,desc_file);
       out_str[len]=0;
       fclose(desc_file);

    }

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);


    tw = textwidth(out_str);

    moveto(RIGHT/2-tw/2,22);
    outtext(out_str);

    delete[] out_str;
    delete[] filename;

}
```

```
//WriteVert
//This routine displays the title for the verticle axis

void WriteVert(void)
{
   char far *out_str; //output string
   int th, tw; //textheight and width

   out_str=new char far [80];

   settextstyle(8,1,0);
   setusercharsize(1,2,1,2);

   out_str="Amplitude (% of";
   th = textheight(out_str);
   tw = textwidth(out_str);
   moveto(2,(BOX1_B/2-tw/2-1));
   outtext(out_str);

   out_str="clipping voltage)";
   tw = textwidth(out_str);
   moveto(4+th,(BOX1_B/2-tw/2-1));
   outtext(out_str);
   delete[] out_str;
}

//WriteTime
//This routine writes the title for the time axis
void WriteTime(void)
{
   char far *out_str; //output string
   int tw,th;         //text width and height

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   if (g_use_time == 1)
   {
      out_str="Time (seconds)";
   }
   else
   {
      out_str="Sample Number";
   }
   tw = textwidth(out_str);
   th = textheight(out_str);

   moveto(SWR-tw,SWB+th);
   outtext(out_str);

   delete[] out_str;
```

```
}

//DrawHash
//This routine draws the hash marks on the axes
void DrawHash(double samp_inc)
{
   double time;         //time interval
   double time_base;    //same order of magnitude unit time increment
   double time_log;     //log of time interval
   int time_logi;       //integer part of time log
   double time_mant;    //time/timebase
   double incr;         //time increment
   int n_inc;           //number of increments
   double s_inc;        //samples in increment
   int i;               //counter
   double tpos;         //time position
   int tabs;            //time coordinate
   int habs;            //waveform height coordinate
   char far *out_str;   //output string
   double hinc;         //waveform height increment
   double hpos;         //waveform height position
   int tw,th;           //text width and height
   double toff;         //time offset
   long int ist, iend; //first time increment, last time increment

   time = coord.stop_time - coord.start_time;


   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,3,1,3);

   ist=(long)(floor(coord.start_time*(double)g_voice_rate));
   iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));

   if (g_use_time == 0)
   {
      time = (double) (iend - ist);
      samp_inc = 1.0;
   }

   //determine increment between hash marks
   time_log = log10(time);
   time_logi = floor(time_log);

   time_base = pow10((double)time_logi);
   time_mant = time/time_base;

   if (1<=time_mant&&time_mant<1.6)
   {
      incr = 0.1 * time_base;
   }
   else if (time_mant<3.2)
```

```
{
   incr = 0.2 * time_base;
}
else if (time_mant<8.0)
{
   incr = 0.5 * time_base;
}
else
{
   incr = 1.0 * time_base;
}

//starting coordinate
tpos=coord.start_time;

if (g_use_time == 0)
{
   tpos = (double)ist;
}


//calculate offset to first hash
toff = incr - fmod(coord.start_time, incr);

if (g_use_time == 0)
{
   toff = incr - fmod((double)ist, incr);
}

if (toff == incr)
{
   toff = 0.0;
}

//caluculate initial hash position
tpos += toff;

n_inc = floor((time-toff)/incr);
s_inc = incr/samp_inc;

//draw time hash marks
for (i=0;i<=n_inc;i++)
{
   tabs = SWL +(int)((double)(SWR-SWL+1)/(double)(iend-\
      ist)*((double)i*s_inc+toff/samp_inc));
   moveto(tabs,SWT);
   linerel(0,4);
   moveto(tabs,SWB);
   linerel(0,-4);

   sprintf(out_str,"%g",tpos);
   tw=textwidth(out_str);
   moveto(tabs-tw/2,SWB+3);
```

```
      outtext(out_str);

      tpos += incr;
   }

   //set height hash mark parameters
   hinc = (coord.maxvoice-coord.minvoice)/8.0;
   hpos = coord.minvoice;

   //draw amplitude hash marks
   for (i=0;i<=8;i++)
   {
      habs=SWB-(int)(hinc*(double)i*(double)(SWB-SWT-1)/\
         (coord.maxvoice-coord.minvoice));

      if (i!=0 && i!=8)
      {
         moveto(SWL, habs);
         linerel(4,0);
         moveto(SWR, habs);
         linerel(-4,0);
      }

      sprintf(out_str,"%.1f",hpos);
      tw=textwidth(out_str);
      th=textheight(out_str);
      moveto(SWL-3-tw,habs-th/2-3);

      outtext(out_str);

      hpos += hinc;
   }
   delete[] out_str;

}

//DrawSpeech
//This routine draws the speech waveform

void DrawSpeech(void)
{
   long int i;  //counter
   int v_range, h_range; //vertical and horizontal coordinate ranges
   int v_zero; //vertical zero coordinate
   double h_inc; //horizontal coordinate increment
   double h_pos=0.0;    //horizontal coordinate
   int spx, spy;        //waveform data coordinate
   short int far *data; //waveform data value
   long int ist, iend;  //start and stop samples

   data = new short int far;

   v_range = SWB-SWT+1;
   h_range = SWR-SWL+1;
```

```
    v_zero = (int)((((coord.maxvoice)*(double)(SWB-SWT+1))-\
       (coord.minvoice)*(0.0))/(coord.maxvoice-coord.minvoice));
    setviewport(SWL+1,SWT+1,SWR+1,SWB+1,1);
    moveto(0,v_zero);

    ist=(long)(floor(coord.start_time*(double)g_voice_rate));
    iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));


    h_inc = (double)h_range/(double)(iend-ist+1);

    //draw waveform

    for (i=ist; i<iend+1; i++)
    {
       XMMcopyfrom(sizeof(*data),voice_handle,i*sizeof(*data),&*data);
       spy = v_zero-(int)(((double)(*data)/((coord.maxvoice-\
          coord.minvoice)*163.84))*(double)(v_range/2));
       spx = (int) h_pos;
       lineto(spx,spy);
       h_pos += h_inc;
    }
    delete data;
}

//DisplayVocFile
//This routine handles the creation of the sound file display

void DisplayVocFile(void)
{
    double samp_inc; //duration of sample


    samp_inc = 1.0/(double)g_voice_rate;

    SetPortTop();
    clearviewport();

    DrawSpeechFrame();
    WriteFileName();
    WriteVert();
    WriteTime();
    WriteVoiceInfo();

    DrawHash(samp_inc);
    DrawSpeech();
}

//WriteVoiceInfo
//This routine displays time and sampling rate information about the
//current voice file display

void WriteVoiceInfo(void)
{
```

```
    char far *out_str; //text output string
    char far *num_str; //numeric output string
    int th; //textheight

    out_str=new char far [80];
    num_str=new char far [80];


    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="Start Time:  ";

    th = textheight(out_str);
    moveto(10,BOX1_B-8-th);
    outtext(out_str);

    sprintf(num_str,"%f",coord.start_time);
    outtext(num_str);

    out_str="s  Stop Time:  ";
    outtext(out_str);

    sprintf(num_str,"%f",coord.stop_time);
    outtext(num_str);

    out_str="s  Sample Rate:  ";
    outtext(out_str);

    sprintf(num_str,"%lu",g_voice_rate);
    outtext(num_str);

    out_str="Hz";
    outtext(out_str);

    delete[] out_str;
    delete[] num_str;
}

//LoadRaw16File
//This routine reads the voc file data into extended memory

void LoadRaw16File(char *f_voice,  char *f_cno)
{
    long int size;        //size of voice data
    unsigned long rate;   //sampling rate
    long int i;           //counter variable
    int j;                //counter variable
    short int far *data; //voice data pointer
    unsigned char type;   //block type
    char f_nvf=0;         //file not valid flag
    char f_allocfail=0;   //memory allocation failure flag
    unsigned long xmem;   //extended memory available
    unsigned long v_arysize;  //voice array size
```

```
unsigned long d_arysize;  //dimension array size

double total_time; //duration of data


//get file name

data = new short int far;
GetFileName();

//open file
V_FP = fopen(vfn,"rb");

//check if file found
if (V_FP == NULL)
{
   CouldNotOpen();
   *f_cno=1;

   for(j=0;j<80;j++)
   {
      vfn[j] = bfn[j];
   }

}
else
{
   if (*f_voice==1)
   {
      //clear display
      XMMfree(voice_handle);
      *f_voice=0;
      SetPortTop();
      clearviewport();

      XMMfree(dim_handle);

      SetPortMiddle();
      clearviewport();
   }

   //set voice globals
   size = (long int)(floor((double)\
         filelength(fileno(V_FP))/2.0));
   g_voice_len = size;

   g_voice_rate = 44100L;

   //check extended memory
   xmem = XMMcoreleft();

   //calculate memory requirements
   v_arysize=size*sizeof(short int);
   d_arysize=size*sizeof(double);
```

```
//allocate extended memory for voice data
voice_handle = 0;
if (v_arysize < xmem)
{
   voice_handle = XMMalloc(v_arysize);
}
else
{
   f_allocfail = 1; //allocation failed
}

xmem = XMMcoreleft();

//allocate extended memory for dimension data
dim_handle = 0;
if (d_arysize < xmem)
{
   dim_handle = XMMalloc(d_arysize);
}
else
{
   f_allocfail = 1;  //allocation failed
}

//check if allocation failed
if (f_allocfail == 1)
{
   if (voice_handle != 0)
   {
      XMMfree(voice_handle);
   }
   if (dim_handle != 0)
   {
      XMMfree(dim_handle);
   }
   CouldNotAlloc();
   *f_cno = 1;
}
else
{
   //store voice data in extended memory
   for (i=0; i<size; i++)
   {
      fread(&*data, sizeof(*data), 1, V_FP);
      XMMcopyto(sizeof(*data),data,voice_handle,\
                     i*sizeof(*data));
   }

   *f_voice = 1; //voice data valid

   //calculate total time
   total_time = (double)(g_voice_len-1)/(double)g_voice_rate;
   g_tot_time = total_time;
```

```
            coord.start_time = 0.0;    //start time for display
            coord.stop_time = g_tot_time;  //stop time for display

            //set scale coordinates
            if (coord.autoscale==0)
            {
               coord.minvoice = -100.0;
               coord.maxvoice = 100.0;
            }
            else
            {
               autoscale();
            }
            FileLoaded();
         }

         fclose(V_FP);

      }
      delete data;
}

//LoadRaw8File
//This routine reads the voc file data into extended memory

void LoadRaw8File(char *f_voice,  char *f_cno)
{
   long int size;        //size of voice data
   unsigned long rate;   //sampling rate
   long int i;           //counter variable
   int j;                //counter variable
   char far *cdata; //voice data pointer
   short int far *data; //voice data pointer
   unsigned char type;   //block type
   char f_nvf=0;         //file not valid flag
   char f_allocfail=0;   //memory allocation failure flag
   unsigned long xmem;   //extended memory available
   unsigned long v_arysize;  //voice array size
   unsigned long d_arysize;  //dimension array size

   double total_time; //duration of data


   //get file name

   cdata = new char far;
   data = new short int far;
   GetFileName();

   //open file
   V_FP = fopen(vfn,"rb");

   //check if file found
```

```
if (V_FP == NULL)
{
   CouldNotOpen();
   *f_cno=1;

   for(j=0;j<80;j++)
   {
      vfn[j] = bfn[j];
   }

}
else
{
   if (*f_voice==1)
   {
      //clear display
      XMMfree(voice_handle);
      *f_voice=0;
      SetPortTop();
      clearviewport();

      XMMfree(dim_handle);

      SetPortMiddle();
      clearviewport();
   }

   //set voice globals
   size = filelength(fileno(V_FP));
   g_voice_len = size;

   g_voice_rate = 44100L;

   //check extended memory
   xmem = XMMcoreleft();

   //calculate memory requirements
   v_arysize=size*sizeof(short int);
   d_arysize=size*sizeof(double);

   //allocate extended memory for voice data
   voice_handle = 0;
   if (v_arysize < xmem)
   {
      voice_handle = XMMalloc(v_arysize);
   }
   else
   {
      f_allocfail = 1; //allocation failed
   }

   xmem = XMMcoreleft();

   //allocate extended memory for dimension data
```

```
dim_handle = 0;
if (d_arysize < xmem)
{
   dim_handle = XMMalloc(d_arysize);
}
else
{
   f_allocfail = 1;  //allocation failed
}

//check if allocation failed
if (f_allocfail == 1)
{
   if (voice_handle != 0)
   {
      XMMfree(voice_handle);
   }
   if (dim_handle != 0)
   {
      XMMfree(dim_handle);
   }
   CouldNotAlloc();
   *f_cno = 1;
}
else
{
   //store voice data in extended memory
   for (i=0; i<size; i++)
   {
      fread(&*cdata, sizeof(*cdata), 1, V_FP);
      *data = ((short int)*cdata-128)* 256;
      XMMcopyto(sizeof(*data),data,voice_handle,\
                     i*sizeof(*data));
   }

   *f_voice = 1; //voice data valid

   //calculate total time
   total_time = (double)(g_voice_len-1)/(double)g_voice_rate;
   g_tot_time = total_time;

   coord.start_time = 0.0;   //start time for display
   coord.stop_time = g_tot_time;  //stop time for display

   //set scale coordinates
   if (coord.autoscale==0)
   {
      coord.minvoice = -100.0;
      coord.maxvoice = 100.0;
   }
   else
   {
      autoscale();
   }
```

```
            FileLoaded();
         }

         fclose(V_FP);

      }
      delete data;
      delete cdata;
}

//LoadRaw1File
//This routine reads the voc file data into extended memory

void LoadRaw1File(char *f_voice,  char *f_cno)
{
   long int size;        //size of voice data
   unsigned long rate;   //sampling rate
   long int i;           //counter variable
   int j;                //counter variable
   int k;                //counter variable
   char far *cdata; //voice data pointer
   short int temp;
   short int far *data; //voice data pointer
   unsigned char type;  //block type
   char f_nvf=0;         //file not valid flag
   char f_allocfail=0;  //memory allocation failure flag
   unsigned long xmem;  //extended memory available
   unsigned long v_arysize;  //voice array size
   unsigned long d_arysize;  //dimension array size

   double total_time; //duration of data


   //get file name

   cdata = new char far;
   data = new short int far;
   GetFileName();

   //open file
   V_FP = fopen(vfn,"rb");

   //check if file found
   if (V_FP == NULL)
   {
      CouldNotOpen();
      *f_cno=1;

      for(j=0;j<80;j++)
      {
         vfn[j] = bfn[j];
      }

   }
```

```
else
{
   if (*f_voice==1)
   {
      //clear display
      XMMfree(voice_handle);
      *f_voice=0;
      SetPortTop();
      clearviewport();

      XMMfree(dim_handle);

      SetPortMiddle();
      clearviewport();
   }

   //set voice globals
   size = filelength(fileno(V_FP))*8;
   g_voice_len = size;

   g_voice_rate = 44100L;

   //check extended memory
   xmem = XMMcoreleft();

   //calculate memory requirements
   v_arysize=size*sizeof(short int);
   d_arysize=size*sizeof(double);

   //allocate extended memory for voice data
   voice_handle = 0;
   if (v_arysize < xmem)
   {
      voice_handle = XMMalloc(v_arysize);
   }
   else
   {
      f_allocfail = 1; //allocation failed
   }

   xmem = XMMcoreleft();

   //allocate extended memory for dimension data
   dim_handle = 0;
   if (d_arysize < xmem)
   {
      dim_handle = XMMalloc(d_arysize);
   }
   else
   {
      f_allocfail = 1;  //allocation failed
   }

   //check if allocation failed
```

```
if (f_allocfail == 1)
{
   if (voice_handle != 0)
   {
      XMMfree(voice_handle);
   }
   if (dim_handle != 0)
   {
      XMMfree(dim_handle);
   }
   CouldNotAlloc();
   *f_cno = 1;
}
else
{
   //store voice data in extended memory
   for (i=0; i<size; i+=8)
   {
      fread(&*cdata, sizeof(*cdata), 1, V_FP);
      temp = (short int)*cdata;
      for(k=0;k<8;k++)
      {
         if (temp >= (short int)pow(2.0,7.0 - (double)k))
         {
            *data = 30000;
            temp -= (short int)pow(2.0,7.0 - (double)k);
         }
         else
         {
            *data = 0;
         }
         XMMcopyto(sizeof(*data),data,voice_handle,\
                     (i+k)*sizeof(*data));
      }
   }

   *f_voice = 1; //voice data valid

   //calculate total time
   total_time = (double)(g_voice_len-1)/(double)g_voice_rate;
   g_tot_time = total_time;

   coord.start_time = 0.0;    //start time for display
   coord.stop_time = g_tot_time;  //stop time for display

   //set scale coordinates
   if (coord.autoscale==0)
   {
      coord.minvoice = -100.0;
      coord.maxvoice = 100.0;
   }
   else
   {
      autoscale();
```

```
            }
            FileLoaded();
         }

         fclose(V_FP);

      }
      delete data;
}
```

B.8 Listing of sgrfrac.cpp

```
//sgrfrac.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

// Routine lsr
//
// Calculates and returns the slope of best fitting line for the
// points located at the x-coordinates contained in the x array and
// the corresponding y-coordinates in the y array.  The number of
// points in the array is given by maxpair - z, where maxpair is the
// number of values calculated and z is the number of points where
// y would have equalled log(0) and have been ignored to prevent
// abnormal termination.  The slope is calculated using the least
// squares regression algorithm.

double lsr(double *y, double *x, unsigned short z,\
   unsigned short maxpair)
{
   double sxy, sx, sy, sx2, m;  //Summation variables and slope for
                                //least squares regression
   unsigned short k;            //for loop counter variable

 /* //Debugging Code

   FILE *llp;
   static int dcount=0;
   int q;

   if (dcount == 1)
   {
      dcount = 0;
   }

   if (dcount == 0)
   {
      llp = fopen("llptmp.txt", "at");

      for(q = 0; q<maxpair-z;q++)
```

```
        {
            fprintf(llp,"%lf ",y[q]);
        }

        fprintf(llp,"\n");
        fclose(llp);
    }

    dcount++;    // End Debugging Code */



    // initialize summation variables

    sxy=0.0;
    sx=0.0;
    sy=0.0;
    sx2=0.0;

    //calculate sums

    for (k = 0; k< (maxpair-z); k++)
    {
        sxy += y[k]*x[k];
        sy += y[k];
        sx += x[k];
        sx2 += x[k]*x[k];
    }

    //calculate slope

    m =  ((sxy - (sx*sy)/(double)maxpair)/(sx2-(sx*sx)/\
        (double)maxpair));

    return(m);
}

// routine Buildf
//
// sets size of boxs to be used for box counting dimension in array f.
// The number of points set is determined by the lsrpoints field of
// the s_Box structure.

void Buildf(unsigned short far *f)
{

/*
    switch (s_Box.lsrpoints)
    {

        case 10:
            f[9] = 30;
        case 9:
            f[8] = 20;
```

```
        case 8:
           f[7] = 15;
        case 7:
           f[6] = 12;
        case 6:
           f[5] = 10;
        case 5:
           f[4] = 6;
        case 4:
           f[3] = 5;
        case 3:
           f[2] = 4;
        default:
        {
           f[1] =3;
           f[0] =2;
        }
     } //*/
  ///*
     switch (s_Box.lsrpoints)
     {

        case 10:
           f[9] = 20;
        case 9:
           f[8] = 15;
        case 8:
           f[7] = 12;
        case 7:
           f[6] = 10;
        case 6:
           f[5] = 6;
        case 5:
           f[4] = 5;
        case 4:
           f[3] = 4;
        case 3:
           f[2] = 3;
        default:
        {
           f[1] =2;
           f[0] =1;
        }
     }//*/



}

// routine boxdim
//
// calculates the box dimension of speech window pointed to by sp.
// Returns the box counting fractal dimension.
```

```
double boxdim(short int huge *sp)
{
   double sum,s2;          //summation
   unsigned short l,k,t,z;//counters, z is number of zeros
   double dim;             //dimension value
   short min, max;         //minumum and maximum level in increment
   double *lvar;           //log of box count
   double *ldelt;          //log of delta t
   unsigned short far *f; //array of box sizes

   f = new unsigned short far [s_Box.lsrpoints];
   lvar = new double [s_Box.lsrpoints];
   ldelt = new double [s_Box.lsrpoints];


   Buildf(f);

   z = 0;
   for (t=1; t<=s_Box.lsrpoints; t++)
   {
      sum=0.0;
      s2=0.0;
      for(k=0; k< ((unsigned short)(s_Box.winsize)-f[t-1]); k+=f[t-1])
      {
         max = sp[k];
         min = sp[k];
         for(l=1; l<=f[t-1]; l++)
         {
            if ((unsigned short)(k+l) < (unsigned short)\
               (s_Box.winsize))
            {
               if (sp[k+l] < (short)min)
               {
                  min = sp[k+l];
               }
               if (sp[k+l] > (short)max)
               {
                  max = sp[k+l];
               }
            }
         }

      s2 = (double)(1.0 + ceil(max/(short)(f[t-1])) - \
         ceil(min/(short)(f[t-1])));
      sum += s2;

      }
         lvar[t-1] = log(sum);
         ldelt[t-1] = log(1.0/(double)f[t-1]);

   }
```

```
   dim = lsr(lvar, ldelt,z, (unsigned short)(s_Box.lsrpoints));

   delete[] f;
   delete[] lvar;
   delete[] ldelt;

   return(dim);

}

// routine vardim
//
// Calculates the fractal dimension of a speech window pointed to by
// sp using the variance dimension algorithm.  The variance fractal
// dimension is returned.

double vardim(short int far *sp)
{
   double sum,s2,sdiff;     //sumation variables
   unsigned short k,t,z;    //counters; z is number of zero points
   double dim;              //dimension
   double dpow;             //dimension power calculation variable

   double var, delt, m, H; //variance dimension variables
   double *lvar;           //log of variance
   double *ldelt;          //log of delta t
   double samps;           //samples

   lvar = new double [s_Var.lsrpoints];
   ldelt = new double [s_Var.lsrpoints];

   z = 0;
   if (s_Var.dyadic==0)
   {
      for (t=1; t<=(unsigned short)(s_Var.lsrpoints); t++)
      {
         sum=0.0;
         s2=0.0;
         if (s_Var.overlap==1)
         {
            for(k=0; k< ((unsigned short)(s_Var.winsize)-\
               (t+1-s_Var.unit)); k++)
            {
               sdiff = ((double)sp[k+(t+1-s_Var.unit)]-\
                  (double)sp[k]);
               sum += sdiff*sdiff;
               //s2 += sdiff;
            }
            samps = (double)(s_Var.winsize)-(double)(t+1-s_Var.unit);
            var =  (sum-(s2*s2)/((double)samps))/(samps-1.0);
 /*         var =  ((sum)-(s2*s2)/((double)(s_Var.winsize)-\
               (double)(t+(1-s_Var.unit))))/\
               ((double)(s_Var.winsize)-(double)(t+(1-s_Var.unit))\
                  -1.0); */
```

```
            delt =(double) (t+(1-s_Var.unit));
        }
        else
        {
            for(k=0; k< ((unsigned short)(s_Var.winsize)\
            -(t+(1-s_Var.unit))); k+=(t+(1-s_Var.unit)))
            {
                sdiff = ((double)sp[k+(t+(1-s_Var.unit))]-\
                    (double)sp[k]);
                sum += sdiff*sdiff;
                //s2 += sdiff;
            }
            samps = floor((double)(s_Var.winsize)/\
            (double)(t+1-s_Var.unit));
            var =  (sum-(s2*s2)/((double)samps))/(samps-1.0);
            delt =(double) (t+(1-s_Var.unit));
        }

        if (var == 0.0)
        {
            z++;
        }
        else
        {
            lvar[t-1-z] = log(var);
            ldelt[t-1-z] = log(delt);
        }
    }
}
else
    {
    for (t=1; t<=(unsigned short)(s_Var.lsrpoints); t++)
    {
        sum=0.0;
        s2=0.0;
        dpow = pow(2.0,(double)(t-s_Var.unit));
        if (s_Var.overlap==1)
        {
            for(k=0; k< ((unsigned short)(s_Var.winsize)-\
                dpow); k++)
            {
                sdiff = ((double)sp[k+(unsigned short int)dpow]-\
                    (double)sp[k]);
                sum += sdiff*sdiff;
                //s2 += sdiff;
            }
            samps = (double)(s_Var.winsize)-(double)dpow;
            var =  (sum-(s2*s2)/((double)samps))/(samps-1.0);

            delt =(double) dpow;
        }
        else
        {
            for(k=0; k< ((unsigned short)(s_Var.winsize)\
```

```
                -dpow); k+=dpow)
                {
                   sdiff = ((double)sp[k+(unsigned short)dpow]-\
                      (double)sp[k]);
                   sum += sdiff*sdiff;
                   //s2 += sdiff;
                }
                samps = floor((double)(s_Var.winsize)/\
                (double)(dpow));
                var =  (sum-(s2*s2)/((double)samps))/(samps-1.0);

                delt =(double) dpow;
             }

             if (var == 0.0)
             {
                z++;
             }
             else
             {
                lvar[t-1-z] = log(var);
                ldelt[t-1-z] = log(delt);
             }
          }
       }

       if (z<((unsigned short)(s_Var.lsrpoints)-1))
       {
          m=lsr(lvar, ldelt, z, (unsigned short)(s_Var.lsrpoints));
          H = m/2.0;
       }
       else
       {
          H=1.0;
       }
       dim = (2.0 - H);
       delete[] lvar;
       delete[] ldelt;

       return(dim);

}

// routine ProcessVariance
//
// Read windows of speech from extended memory into voice array
// Passes array to vardim routine.  Writes variance fractal dimensions
// into extended memory.  Size of windows determined by winsize field
// of s_Var structure.  Spacing of windows determined by interval
// field of s_Var structure.  Intermediate points of fractal dimension
// array calculated by linear interpolation.  End padded with final
// calculated value so dimension array is of same size as speech array

void ProcessVariance(void)
```

```
{
   int i;          //counter
   double fdold, fdnew, fdint; //interpolation variables
   long int k=0;     //array index for extended memory
   long int x;       //remaining samples at end
   long int diffx; //position to start for last dim pt
   long int j;       //counter


   XMMcopyfrom((unsigned long)(sizeof(short)*(s_Var.winsize)), \
      voice_handle, k*sizeof(short), voice);

   fdnew = vardim(voice);


   XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
   fdold = fdnew;
   k += s_Var.interval;

   while(k <= g_voice_len-s_Var.winsize)
   {
      XMMcopyfrom((unsigned long)(sizeof(short)*(s_Var.winsize)),\
          voice_handle, k*sizeof(short), voice);

      fdnew = vardim(voice);


      XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
      for (i=1; i<s_Var.interval; i++)
      {
         fdint = fdold + (double)i/(double)s_Var.interval*\
            (fdnew-fdold);


         XMMcopyto(sizeof(double), &fdint, dim_handle,\
            (k-s_Var.interval+i)*sizeof(double));
      }
      fdold = fdnew;
      k += s_Var.interval;
   }
   if (k<g_voice_len)
   {
      x = k -s_Var.interval;
      k = g_voice_len-s_Var.winsize;
      XMMcopyfrom((unsigned long)(sizeof(short)*(s_Var.winsize)),\
         voice_handle, x*sizeof(short), voice);

      diffx = k - x;

      fdnew = vardim(voice);

      XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
      for (i=1; i<diffx; i++)
      {
```

```
            fdint = fdold + (double)i/(double)s_Var.interval*\
               (fdnew-fdold);

            XMMcopyto(sizeof(double), &fdint, dim_handle, \
               (x+i)*sizeof(double));
         }
      }

      for (j=g_voice_len-s_Var.winsize; j<g_voice_len; j++)
      {

         XMMcopyto(sizeof(double), &fdnew, dim_handle, j*sizeof(double));
      }

      GetMinMaxDim();

      coord.mindim = 1.0;
      coord.maxdim = 2.0;
      coord.start_time=0.0;
      coord.stop_time=g_tot_time;

}

// routine ProcessBox
//
// Read windows of speech from extended memory into voice array
// Passes array to boxdim routine.  Writes box counting fractal
// dimensions into extended memory.  Size of windows determined by
// winsize field of s_Box structure.  Spacing of windows determined by
// interval field of s_Box structure.  Intermediate points of fractal
// dimension array calculated by linear interpolation.  End padded
// with final calculated value so dimension array is of same size as
// speech array

void ProcessBox(void)
{
   int i;                          //counter
   double fdold, fdnew, fdint;     //interpolation variables
   long int k=0;                   //array index for extended memory
   long int x;                     //remaining samples at end
   long int diffx;                 //position to start for last dim pt
   long int j;                     //counter


   XMMcopyfrom((unsigned long)(sizeof(short)*(s_Box.winsize)),\
      voice_handle, k*sizeof(short), voice);

   fdnew = boxdim(voice);

   XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
   fdold = fdnew;
   k += s_Box.interval;
```

```
    while(k < g_voice_len-s_Box.winsize)
    {
       XMMcopyfrom((unsigned long)(sizeof(short)*(s_Box.winsize)),\
       voice_handle, k*sizeof(short), voice);

       fdnew = boxdim(voice);

       XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
       for (i=1; i<s_Box.interval; i++)
       {
          fdint = fdold + (double)i/(double)s_Box.interval*\
             (fdnew-fdold);

          XMMcopyto(sizeof(double), &fdint, dim_handle,\
             (k-s_Box.interval+i)*sizeof(double));
       }
       fdold = fdnew;
       k += s_Box.interval;
    }
    if (k<g_voice_len)
    {
       x = k - s_Box.interval;
       k = g_voice_len-s_Box.winsize;
             XMMcopyfrom((unsigned long)(sizeof(short)*\
                (s_Box.winsize)), voice_handle, k*sizeof(short),\
                voice);
       diffx=k-x;

       fdnew = boxdim(voice);

       XMMcopyto(sizeof(double), &fdnew, dim_handle, k*sizeof(double));
       for (i=1; i<diffx; i++)
       {
          fdint = fdold + (double)i/(double)s_Box.interval*\
          (fdnew-fdold);

          XMMcopyto(sizeof(double), &fdint, dim_handle, \
             (x+i)*sizeof(double));
       }
    }
    for (j=g_voice_len-s_Box.winsize; j<g_voice_len; j++)
    {

       XMMcopyto(sizeof(double), &fdnew, dim_handle, j*sizeof(double));
    }

    GetMinMaxDim();
    coord.mindim = 1.0;
    coord.maxdim = 2.0;
    coord.start_time = 0.0;
    coord.stop_time = g_tot_time;
}

//routine GetNewInt
```

```
//get a integer from the user

void GetNewInt(double *x)
{
   int i=0;                 //character string length
   char ch_in;              //input character
   char done=0;             //value input flag
   char far *out_str;       //output string
   char far *ch_str;        //character string
   char far *last_ch;       //last character string
   char far *n_str;         //number string
   double y=0.7;            //initialize parameter variable
   int tw;                  //text width
   char **endptr=NULL;      //end of string

   out_str = new char far [80];
   n_str = new char far [80];
   ch_str = new char far [80];
   last_ch = new char far [80];


   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Enter New Value: ";
   moveto(2,182);
   outtext(out_str);

   while (done == 0)
   {
      ch_in = getch();
      switch (ch_in)
      {
         case '\r':
         {
            done = 1;
            n_str[i]=0;
            break;
         }
         case '\b':
         {
            if (i > 0)
            {
               i--;
               last_ch[0] = n_str[i];
               last_ch[1]=0;
               tw=textwidth(last_ch);
               moverel(-tw,0);
               setcolor(0);
               outtext(last_ch);
               setcolor(15);
               moverel(-tw,0);
            }
```

```
            break;
         }
         case '1':
         case '2':
         case '3':
         case '4':
         case '5':
         case '6':
         case '7':
         case '8':
         case '9':
         case '0':
         {
            n_str[i]=ch_in;
            ch_str[0]=ch_in;
            ch_str[1]=0;
            outtext(ch_str);

            i++;
         }
      }

   }
   n_str[i++]='.';
   n_str[i++]='0';
   n_str[i]=0;

   y=strtod(n_str, endptr);
   *x=y;
   delete[] out_str;
   delete[] ch_str;
   delete[] last_ch;
   delete[] n_str;
}

// routine OutofRange
//
// Error message warning user that an out of range value has been
// entered and the value has not been updated.

void OutofRange(void)
{
   char far * out_str; //output string

   out_str = new char far [80];
   out_str = "Out of Range:  Value Not Changed";
   moveto(2,202);
   outtext(out_str);

   out_str = "  Press Any Key to Continue";

   outtext(out_str);

   while (kbhit()==0)
```

```
    {
    }
    delete[] out_str;
}

// routine CheckVWinRange
//
// checks size of window for variance dimension
// valid set to 1 if ok

void CheckVWinRange(double x, char * valid)
{
    if (x<100.0||x>4000.0)
    {
        OutofRange();
    }
    else
    {
        *valid = 1;
    }
}

// routine CheckVIntRange
//
// checks size of interval for variance dimension
// valid set to 1 if ok

void CheckVIntRange(double x, char * valid)
{
    if (x<1.0||x>s_Var.winsize)
    {
        OutofRange();
    }
    else
    {
        *valid = 1;
    }
}

// routine CheckVLSRRange
//
// checks number of points to use for least squares regression for
// variance dimension.  valid set to 1 if ok

void CheckVLSRRange(double x, char * valid)
{
    if (x<2.0||x>10.0)
    {
        OutofRange();
    }
    else
    {
        *valid = 1;
    }
```

```
}

// routine CheckBWinRange
//
// checks size of window for box couunting dimension
// valid set to 1 if ok

void CheckBWinRange(double x, char * valid)
{
   if (x<100.0||x>4000.0)
   {
      OutofRange();
   }
   else
   {
      *valid = 1;
   }
}

// routine CheckBIntRange
//
// checks size of interval for box counting dimension
// valid set to 1 if ok

void CheckBIntRange(double x, char * valid)
{
   if (x<1.0||x>s_Box.winsize)
   {
      OutofRange();
   }
   else
   {
      *valid = 1;
   }
}

// routine CheckBLSRRange
//
// checks number of points to use for least squares regression for
// box counting dimension.  valid set to 1 if ok

void CheckBLSRRange(double x, char * valid)
{
   if (x<2.0||x>10.0)
   {
      OutofRange();
   }
   else
   {
      *valid = 1;
   }
}

// routine GetVarianceParameters
```

```
//
// allow user to set parameters for variance fractal dimension
// calculations.  finished set to 1 when paraments are accepted by
// the user.

void GetVarianceParameters(char *finished)
{
   char far *out_str; //output string
   char far *num_str; //numeric output string
   char done=0;       //menu exit flag
   char ch_in;        //input character
   double x;          //temporary parameter variable
   char v_range = 0;  //valid range flag

   out_str=new char far [80];
   num_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);
   SetPortBottom();
   clearviewport();

   out_str="Variance dimension parameters";
   moveto(2,2);
   outtext(out_str);

   out_str="<1> Window size (100 - 4000) = ";
   sprintf(num_str,"%i",s_Var.winsize);
   moveto(2,42);
   outtext(out_str);
   outtext(num_str);

   out_str="<2> Interval Between Samples (1 - Window Size) = ";
   sprintf(num_str,"%i",s_Var.interval);
   moveto(2,62);
   outtext(out_str);
   outtext(num_str);

   out_str="<3> Number of Data Pairs in Calculations (2 - 10) =";
   sprintf(num_str,"%i",s_Var.lsrpoints);
   moveto(2,82);
   outtext(out_str);
   outtext(num_str);


   out_str="<4> Toggle Overlap Increments = ";
   moveto(2,102);
   outtext(out_str);
   if (s_Var.overlap==1)
   {
      out_str = "on";
   }
   else
   {
```

```
   out_str = "off";
}
outtext(out_str);


out_str="<5> Toggle Sequence Type = ";
moveto(2,122);
outtext(out_str);
if (s_Var.dyadic==1)
{
   out_str = "Dyadic";
}
else
{
   out_str = "Linear";
}
outtext(out_str);


out_str="<6> Cycle Minimum Increment Size= ";
moveto(2,142);
outtext(out_str);
if (s_Var.unit==1)
{
   out_str = "smallest";
}
else if (s_Var.unit==0)
{
   out_str = "second smallest";
}
else
{
   out_str = "third smallest";
}
outtext(out_str);



out_str="<A> Accept Parameters";
moveto(2,162);
outtext(out_str);


while (done==0)
{
   ch_in = getch();
   switch ((char)ch_in)
   {
       case '1':
       {
         GetNewInt(&x);
         CheckVWinRange(x,&v_range);

         if (v_range != 0)
```

```
    {
       s_Var.winsize = (int) x;
       if (s_Var.interval > s_Var.winsize)
       {
          s_Var.interval = s_Var.winsize;
       }
    }
    v_range = 0;

    done=1;

    break;
  }
  case '2':
  {
    GetNewInt(&x);
    CheckVIntRange(x,&v_range);

    if (v_range != 0)
    {
       s_Var.interval = (int)x;
    }
    v_range = 0;

    done=1;
    break;
  }
  case '3':
  {
    GetNewInt(&x);
    CheckVLSRRange(x,&v_range);

    if (v_range != 0)
    {
       s_Var.lsrpoints = (int) x;
    }
    v_range = 0;

    done=1;
    break;
  }
  case '4':
  {

    if (s_Var.overlap == 1)
    {
       s_Var.overlap = 0;
    }
    else
    {
       s_Var.overlap = 1;
    }

    done=1;
```

```
            break;
          }
          case '5':
          {

            if (s_Var.dyadic == 1)
            {
               s_Var.dyadic = 0;
            }
            else
            {
               s_Var.dyadic = 1;
            }

            done=1;
            break;
          }
          case '6':
          {

            if (s_Var.unit == 1)
            {
               s_Var.unit = 0;
            }
            else if (s_Var.unit == 0)
            {
              s_Var.unit = -1;
            }
            else
            {
               s_Var.unit = 1;
            }

            done=1;
            break;
          }
          case 'a':
          case 'A':
          {
            *finished=1;
            done=1;
            break;
          }
      }
   }

   clearviewport();

   delete[] num_str;
   delete[] out_str;
}

// routine GetBoxParameters
//
```

```
// allow user to set parameters for boxcounting fractal dimension
// calculations.  finished set to 1 when paraments are accepted by
// the user.


void GetBoxParameters(char * finished)
{
   char far *out_str;     //output string
   char far *num_str;     //numeric output string
   char done=0;           //menu exit flag
   char ch_in;            //input character
   double x;              //parameter holding variable
   char v_range = 0;      //valid range flag

   out_str=new char far [80];
   num_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);
   SetPortBottom();
   clearviewport();

   out_str="Box counting dimension parameters";
   moveto(2,2);
   outtext(out_str);

   out_str="<1> Window size (100 - 4000)= ";
   sprintf(num_str,"%i",s_Box.winsize);
   moveto(2,42);
   outtext(out_str);
   outtext(num_str);

   out_str="<2> Interval Between Samples (1-Window Size)= ";
   sprintf(num_str,"%i",s_Box.interval);
   moveto(2,62);
   outtext(out_str);
   outtext(num_str);

   out_str="<3> Number of Data Pairs in Calculations (2-10)= ";
   sprintf(num_str,"%i",s_Box.lsrpoints);
   moveto(2,82);
   outtext(out_str);
   outtext(num_str);

   out_str="<A> Accept Parameters";
   moveto(2,102);
   outtext(out_str);


   while (done==0)
   {
      ch_in = getch();
      switch ((char)ch_in)
      {
```

```
case '1':
{
  GetNewInt(&x);
  CheckBWinRange(x,&v_range);

  if (v_range != 0)
  {
     s_Box.winsize = (int)x;
     if (s_Box.interval > s_Box.winsize)
     {
        s_Box.interval = s_Box.winsize;
     }
  }
  v_range = 0;

  done=1;

  break;
}
case '2':
{
  GetNewInt(&x);
  CheckBIntRange(x,&v_range);

  if (v_range != 0)
  {
     s_Box.interval = (int) x;
  }
  v_range = 0;

  done=1;
  break;
}
case '3':
{
  GetNewInt(&x);
  CheckVLSRRange(x,&v_range);

  if (v_range != 0)
  {
     s_Box.lsrpoints = (int) x;
  }
  v_range = 0;

  done=1;
  break;
}
case 'a':
case 'A':
{
  *finished=1;
  done=1;
  break;
}
```

```
      }
   }
   clearviewport();

   delete[] num_str;
   delete[] out_str;
}

// routine CalculateDim
//
// processes user's selection of technique for calculating fractal
// dimension

void CalculateDim(int dim)
{
   char finished = 0;  //flag for finished calculations
   switch (dim)
   {
      case 1:     //Variance technique
      {
         while (finished == 0)
         {
            GetVarianceParameters(&finished);
         }
         voice = new short int far [s_Var.winsize];
         ProcessVariance();
         fdt="Variance dimension";
         delete[] voice;
         break;
      }
      case 2://Box Counting technique
      {
         while (finished == 0)
         {
            GetBoxParameters(&finished);
         }
         voice = new short int far [s_Box.winsize];
         ProcessBox();
         fdt="Box Counting Dimension";
         delete[]voice;
         break;
      }
   }

}
```

B.9 Listing of sgrmenu.cpp

```
// sgrmenu.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
```

```
   #include "std.h"
#endif

//ShowFractalMenu_1
//This routine displays the selection of calculation type that can be
//performed and processes the users input

void ShowFractalMenu_1(void)
{
   char far *out_str;  //output string
   char done=0;        //menu exit flag
   char ch_in;         //input character

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);
   SetPortBottom();
   clearviewport();

   out_str="Choose Dimension to Calculate";
   moveto(2,2);
   outtext(out_str);

   out_str="<1> Variance";
   moveto(2,22);
   outtext(out_str);

   out_str="<2> Box Counting";
   moveto(2,42);
   outtext(out_str);

   while (done==0)
   {
      ch_in = getch();
      switch ((char)ch_in)
      {
         case '1':
         {
            g_active_dim=1;
            done=1;

            break;
         }
         case '2':
         {
            done=1;
            g_active_dim=2;
            break;
         }
      }
   }
   clearviewport();
   CalculateDim(g_active_dim);
```

```
   DisplayVocFile();
   DisplayDim();
   delete[] out_str;
}

//ShowAnalysisMenu_1
//This routine displays analysis options that can be performed on the
//dimension waveform and processes the users input
void ShowAnalysisMenu_1(void)
{
   char far *out_str; //output string
   char done=0;       //menu exit flag
   char ch_in;        //input character

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);
   SetPortBottom();
   clearviewport();

   out_str="Choose Analysis Process";
   moveto(2,2);
   outtext(out_str);

   out_str="<1> Amount of Signal with Dimension below threshold";
   moveto(2,22);
   outtext(out_str);

   out_str="<2> Amount of Signal with Dimension above threshold";
   moveto(2,42);
   outtext(out_str);

   while (done==0)
   {
      ch_in = getch();
      switch ((char)ch_in)
      {
         case '1':
         {
            DimBelow();
            done=1;

            break;
         }
         case '2':
         {
            done=1;
            DimAbove();
            break;
         }
      }
   }
   clearviewport();
```

```
   delete[] out_str;
}
//ShowRangesMenu_1
//This routine displays the display ranges that can be adjusted and
//processes the users input

void ShowRangesMenu_1(void)
{
   char far *out_str; //output string
   char done=0;       //interal menu exit flag
   char finished=0;   //routine exit flag
   char ch_in;        //input character

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   while (finished==0)
   {
      SetPortBottom();
      clearviewport();

      out_str="<V> Zoom on Voice";
      moveto(2,2);
      outtext(out_str);

      out_str="<D> Zoom on Dimension";
      moveto(2,22);
      outtext(out_str);

      out_str="<T> Adjust Time Span";
      moveto(2,42);
      outtext(out_str);

      out_str="<P> Process New Ranges";
      moveto(2,62);
      outtext(out_str);

      done=0;

      while (done==0)
      {
         ch_in = getch();
         switch ((char)ch_in)
         {
            case 'V':
            case 'v':
            {
               GetVoiceRange();
               done=1;

               break;
            }
```

```
            case 'D':
            case 'd':
            {
               GetDimRange();
               done=1;
               break;
            }
            case 'T':
            case 't':
            {
               GetTimeRange();
               done=1;
               break;
            }
            case 'P':
            case 'p':
            {
               ProcessRanges();
               done=1;
               finished=1;
               break;
            }
         }
      }

   }
   delete[] out_str;

}

//ShowMenu_1
//This routine displays the menu selection available when there is
//no valid voice data in memory

void ShowMenu_1(void)
{
   char far *out_str; //output string

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="<L> Load Voc File";
   moveto(2,2);
   outtext(out_str);

   out_str="<R> Load Raw File with 16 bit data <8> with 8 bit data";
   moveto(2,22);
   outtext(out_str);
   out_str=" <1> 1 bit data";
   outtext(out_str);

   out_str="<S> Save Screen";
```

```
   moveto(2,42);
   outtext(out_str);

   out_str="<Q> Quit";
   moveto(2,62);
   outtext(out_str);
   delete[] out_str;
}

//ProcessMenu_1
//This routine processes the users response when there is
//no valid voice data in memory

void ProcessMenu_1(char *f_quit, int *new_menu, char *f_voice)
{
   char done=0; //menu exit flag
   int ch_in;   //input character
   char f_cno=0;//could not open file flag

   while (done==0)
   {
      ch_in = getch();
      switch ((char)ch_in)
      {
          case 'q':
          case 'Q':
          {
            done=1;
            *f_quit=1;  //termination flag
            break;
          }
          case 'l':
          case 'L':
          {
            LoadVocFile(&*f_voice, &f_cno);
            if (f_cno == 0)
            {
               g_use_time = 1;
               DisplayVocFile();
               *new_menu = 2;
               f_cno = 0;
            }
            if(*f_voice == 0)
            {
               *new_menu=1;
            }
            done = 1;
            break;
          }
          case 'r':
          case 'R':
          {
            LoadRaw16File(&*f_voice, &f_cno);
            if (f_cno == 0)
```

```
    {
        g_use_time = 0;
        DisplayVocFile();
        *new_menu = 2;
        f_cno = 0;
    }
    if(*f_voice == 0)
    {
        *new_menu=1;
    }
    done = 1;

    break;
}
case '8':
case '*':
{
    LoadRaw8File(&*f_voice, &f_cno);
    if (f_cno == 0)
    {
        g_use_time = 0;
        DisplayVocFile();
        *new_menu = 2;
        f_cno = 0;
    }
    if(*f_voice == 0)
    {
        *new_menu=1;
    }
    done = 1;

    break;
}
case '1':
case '!':
{
    LoadRaw1File(&*f_voice, &f_cno);
    if (f_cno == 0)
    {
        g_use_time = 0;
        DisplayVocFile();
        *new_menu = 2;
        f_cno = 0;
    }
    if(*f_voice == 0)
    {
        *new_menu=1;
    }
    done = 1;

    break;
}
case 's':
case 'S':
```

```
            {
              SaveScreen();
              done = 1;
              break;
            }
        }
    }
}
//ShowMenu_2
//This routine displays the menu selection available when there is
//valid voice data in memory but the dimension has not been calculated

void ShowMenu_2(void)
{
   char far *out_str; //output string

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="<L> Load New Voc File";
   moveto(2,2);
   outtext(out_str);

   out_str="<R> Load Raw File with 16 bit data <8> with 8 bit data";
   moveto(2,22);
   outtext(out_str);
   out_str=" <1> 1 bit data";
   outtext(out_str);

   out_str="<N> Add Noise to Voice";
   moveto(2,42);
   outtext(out_str);

   out_str="<F> Calculate Fractal Dimension";
   moveto(2,62);
   outtext(out_str);

   out_str="<V> Save Voc File";
   moveto(2,82);
   outtext(out_str);

   out_str="<S> Save Screen";
   moveto(2,102);
   outtext(out_str);

   out_str="<Q> Quit";
   moveto(2,122);
   outtext(out_str);
   delete[] out_str;
}

//ProcessMenu_2
```

```
//This routine processes the users response when there is
//valid voice data in memory but the dimension has not been calculated

void ProcessMenu_2(char *f_quit, int *new_menu, char *f_voice)
{
   char done=0;  //menu exit
   int ch_in;    //input character
   char f_cno=0; //could not open file flag

   while (done==0)
   {
      ch_in = getch();
      switch ((char)ch_in)
      {
          case 'q':
          case 'Q':
          {
            done=1;
            *f_quit=1; //terminate
            break;
          }
          case 'f':
          case 'F':
          {
            done=1;
            ShowFractalMenu_1();
            *new_menu=3;
            break;
          }
          case 'l':
          case 'L':
          {
            LoadVocFile(&*f_voice, &f_cno);
            if (f_cno == 0)
            {
               g_use_time = 1;
               DisplayVocFile();
               *new_menu = 2;
               f_cno = 0;
            }
            if(*f_voice == 0)
            {
               *new_menu=1;
            }
            done = 1;

            break;
          }
          case 'r':
          case 'R':
          {
            LoadRaw16File(&*f_voice, &f_cno);
            if (f_cno == 0)
            {
```

```
      g_use_time = 0;
      DisplayVocFile();
      *new_menu = 2;
      f_cno = 0;
   }
   if(*f_voice == 0)
   {
      *new_menu=1;
   }
   done = 1;

   break;
}
case '8':
case '*':
{
   LoadRaw8File(&*f_voice, &f_cno);
   if (f_cno == 0)
   {
      g_use_time = 0;
      DisplayVocFile();
      *new_menu = 2;
      f_cno = 0;
   }
   if(*f_voice == 0)
   {
      *new_menu=1;
   }
   done = 1;

   break;
}
case '1':
case '!':
{
   LoadRaw1File(&*f_voice, &f_cno);
   if (f_cno == 0)
   {
      g_use_time = 0;
      DisplayVocFile();
      *new_menu = 2;
      f_cno = 0;
   }
   if(*f_voice == 0)
   {
      *new_menu=1;
   }
   done = 1;

   break;
}
case 'n':
case 'N':
{
```

```
            Noise();
            SetPortMiddle();
            clearviewport();
            DisplayVocFile();
            *new_menu = 2;

            done = 1;

            break;
          }
          case 's':
          case 'S':
          {
            SaveScreen();
            done=1;
            break;
          }
          case 'v':
          case 'V':
          {
            SaveVoice();
            done=1;
            break;
          }
      }
   }
}

//ShowMenu_3
//This routine displays the menu selection available when there is
//valid voice data in memory and the dimension has been calculated

void ShowMenu_3(void)
{
   char far *out_str; //output string

   out_str=new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="<L> Load New Voc File";
   moveto(2,2);
   outtext(out_str);

   out_str="<R> Load Raw File with 16 bit data <8> with 8 bit data";
   moveto(2,22);
   outtext(out_str);
   out_str=" <1> 1 bit data";
   outtext(out_str);

   out_str="<N> Add Noise to Voice";
   moveto(2,42);
   outtext(out_str);
```

```
    out_str="<F> Calculate Fractal Dimension";
    moveto(2,62);
    outtext(out_str);

    out_str="<A> Adjust Display Ranges";
    moveto(2,82);
    outtext(out_str);

    out_str="<I> Analysis";
    moveto(2,102);
    outtext(out_str);


    out_str="<V> Save Voc File";
    moveto(2,122);
    outtext(out_str);

    out_str="<S> Save Screen";
    moveto(2,142);
    outtext(out_str);

    out_str="<Q> Quit";
    moveto(2,162);
    outtext(out_str);
    delete[] out_str;
}

//ProcessMenu_3
//This routine processes the users response when there is
//valid voice data in memory and the dimension has been calculated

void ProcessMenu_3(char *f_quit, int *new_menu, char *f_voice)
{
    char done=0; //menu exit
    int ch_in;    //input character
    char f_cno=0; //could not open flag

    while (done==0)
    {
       ch_in = getch();
       switch ((char)ch_in)
       {
           case 'q':
           case 'Q':
           {
             done=1;
             *f_quit=1;
             break;
           }
           case 'f':
           case 'F':
           {
             done=1;
```

```
    ShowFractalMenu_1();
    *new_menu=3;
    break;
  }
  case 'i':
  case 'I':
  {
    done=1;
    ShowAnalysisMenu_1();
    *new_menu=3;
    break;
  }
  case 'a':
  case 'A':
  {
    done=1;
    ShowRangesMenu_1();
    break;
  }
  case 'l':
  case 'L':
  {
    LoadVocFile(&*f_voice, &f_cno);
    if (f_cno == 0)
    {
       g_use_time = 1;
       DisplayVocFile();
       *new_menu = 2;
       f_cno = 0;
    }
    if(*f_voice == 0)
    {
       *new_menu=1;
    }
    done = 1;

    break;
  }
  case 'r':
  case 'R':
  {
    LoadRaw16File(&*f_voice, &f_cno);
    if (f_cno == 0)
    {
       g_use_time = 0;
       DisplayVocFile();
       *new_menu = 2;
       f_cno = 0;
    }
    if(*f_voice == 0)
    {
       *new_menu=1;
    }
    done = 1;
```

```
  break;
}
case '8':
case '*':
{
  LoadRaw8File(&*f_voice, &f_cno);
  if (f_cno == 0)
  {
     g_use_time = 0;
     DisplayVocFile();
     *new_menu = 2;
     f_cno = 0;
  }
  if(*f_voice == 0)
  {
     *new_menu=1;
  }
  done = 1;

  break;
}
case '1':
case '!':
{
  LoadRaw1File(&*f_voice, &f_cno);
  if (f_cno == 0)
  {
     g_use_time = 0;
     DisplayVocFile();
     *new_menu = 2;
     f_cno = 0;
  }
  if(*f_voice == 0)
  {
     *new_menu=1;
  }
  done = 1;

  break;
}
case 'n':
case 'N':
{
  Noise();
  SetPortMiddle();
  clearviewport();
  DisplayVocFile();
  *new_menu = 2;

  done = 1;

  break;
}
```

```
            case 's':
            case 'S':
            {
              SaveScreen();
              done=1;
              break;
            }
            case 'v':
            case 'V':
            {
              SaveVoice();
              done=1;
              break;
            }
        }
    }
}
```

B.10 Listing of fracplot.cpp

```
//fracplot.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

//DrawDimFrame
//This routine draws the frame around the dimension waveform window

void DrawDimFrame(void)
{
   rectangle(DWL,DWT,DWR,DWB);
}

//WriteDimTitle
//This routine writes the title for the dimension display window
//and parameter settings

void WriteDimTitle(void)
{
   char far *out_str;    //output string
   char far *num_str;    //numeric output string
   int tw;               //text width

   out_str=new char far[80];
   num_str=new char far[80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str=fdt;
```

```
tw = textwidth(out_str);

moveto(RIGHT/2-tw/2,2);
outtext(out_str);

if (g_active_dim == 1)
{
   moveto(LEFT+10,22);
   if(s_Var.dyadic==1)
   {
   out_str="Dyadic, ";
   }
   else
   {
   out_str="Sequential, ";
   }
   outtext(out_str);

   if(s_Var.overlap==1)
   {
      out_str="overlapping, ";
   }
   else
   {
      out_str="non-overlapping, ";
   }
   outtext(out_str);

   if(s_Var.unit==1)
   {
      out_str="smallest incr. size used, ";
   }
   else if(s_Var.unit==-1)
   {
      out_str="3rd smallest incr. size used, ";
   }
   else
   {
      out_str="2nd smallest incr. size used, ";
   }
   outtext(out_str);


sprintf(num_str,"%i",s_Var.lsrpoints);
outtext(num_str);
out_str=" points, ";
outtext(out_str);



out_str=" Window Size = ";
outtext(out_str);
```

```
    sprintf(num_str,"%i",s_Var.winsize);
    outtext(num_str);

    out_str=", Spacing = ";
    outtext(out_str);

    sprintf(num_str,"%i",s_Var.interval);
    outtext(num_str);

    }

    delete[] num_str;
    delete[] out_str;

}

//WriteVertDim
//This routine writes the vertical axis title

void WriteVertDim(void)
{
    char far *out_str;  //output string
    int tw;             //text width

    out_str=new char far[80];

    settextstyle(8,1,0);
    setusercharsize(1,2,1,2);

    out_str="Dimension";

    tw = textwidth(out_str);
    moveto(2,(BOX1_B/2-tw/2-1));
    outtext(out_str);
    delete[] out_str;

}

//WriteTimeDim
//This routine writes the time axis title
void WriteTimeDim(void)
{
    char far *out_str; //output string
    int tw,th;         //text width and height

    out_str=new char far [80];
    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    if (g_use_time == 1)
    {
       out_str="Time (seconds)";
    }
```

```
    else
    {
       out_str="Sample Number";
    }
    tw = textwidth(out_str);
    th = textheight(out_str);

    moveto(DWR-tw,DWB+th);
    outtext(out_str);

    delete[] out_str;
}

//DrawHashDim
//This routine draws the hash marks for the dimension waveform window

void DrawHashDim(double samp_inc)
{
    double time;           //time range of window
    double time_base;      //order of magnitude for increment
    double time_log;       //log of time range
    int time_logi;         //integer part of log
    double time_mant;      //time/time_base
    double incr;           //increment size
    int n_inc;             //number of increments
    double s_inc;          //size of increment
    int i;                 //counter
    double tpos;           //time position
    int tabs;              //time coordinate
    int habs;              //waveform height coordinate
    char far *out_str;     //output string
    double hinc;           //height increment
    double hpos;           //height position
    int tw,th;             //text width and height
    double toff;           //time offset
    long int ist, iend;    //start and end sample

    time = coord.stop_time - coord.start_time;

    out_str = new char far [80];

    settextstyle(8,0,0);
    setusercharsize(1,3,1,3);

    ist=(long)(floor(coord.start_time*(double)g_voice_rate));
    iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));

    if (g_use_time == 0)
    {
       time = (double) (iend - ist);
       samp_inc = 1.0;
    }

    //calculate time increment
```

```
time_log = log10(time);
time_logi = floor(time_log);

time_base = pow10((double)time_logi);
time_mant = time/time_base;

if (1<=time_mant&&time_mant<1.6)
{
   incr = 0.1 * time_base;
}
else if (time_mant<3.2)
{
   incr = 0.2 * time_base;
}
else if (time_mant<8.0)
{
   incr = 0.5 * time_base;
}
else
{
   incr = 1.0 * time_base;
}

//get start time
tpos=coord.start_time;

if (g_use_time == 0)
{
   tpos = (double)ist;
}


//calculate position of first hash
toff = incr - fmod(coord.start_time, incr);

if (g_use_time == 0)
{
   toff = incr - fmod((double)ist, incr);
}

if (toff == incr)
{
   toff = 0.0;
}

tpos += toff;

//calculate number and samples per increment
n_inc = floor((time-toff)/incr);
s_inc = incr/samp_inc;

//draw time hash marks
for (i=0;i<=n_inc;i++)
```

```
    {
       tabs = DWL +(int)((double)(DWR-DWL+1)/(double)(iend-\
          ist)*((double)i*s_inc+toff/samp_inc));
       moveto(tabs,DWT);
       linerel(0,4);
       moveto(tabs,DWB);
       linerel(0,-4);


       sprintf(out_str,"%g",tpos);
       tw=textwidth(out_str);
       moveto(tabs-tw/2,DWB+3);

       outtext(out_str);

       tpos += incr;
    }

    //set height hash parameters
    hinc = (coord.maxdim-coord.mindim)/8.0;
    hpos = coord.mindim;

    //draw hash marks on vertical axis
    for (i=0;i<=8;i++)
    {
       habs=DWB-(int)((double)hinc*(double)i*(double)(DWB-DWT-1)/\
          (coord.maxdim- coord.mindim));

       if (i!=0 && i!=8)
       {
          moveto(DWL, habs);
          linerel(4,0);
          moveto(DWR, habs);
          linerel(-4,0);
       }

       sprintf(out_str,"%.3f",hpos);
       tw=textwidth(out_str);
       th=textheight(out_str);
       moveto(DWL-3-tw,habs-th/2-3);



       outtext(out_str);

       hpos += hinc;
    }
    delete[] out_str;
}

//DrawDim
//This routine draws the dimension waveform

void DrawDim(void)
```

```
{
   long int i;              //counter
   int v_range, h_range;    //vertical and horizontal range
   int v_zero;              //vertical zero coordinate
   int h_zero;     //horizontal initial coordinate
   long int h_off; //horizontal initial offset sample count
   double h_inc;            //horizontal increment
   double h_pos=0.0;        //current horizontal position
   int spx, spy;            //data coordinates
   double far *data;        //data value
   long int ist, iend;      //start and end samples

   data = new double far;

   //calculate range sizes
   v_range = DWB-DWT+1;
   h_range = DWR-DWL+1;

   //calculate zero coordinate
   v_zero = (int)((((coord.maxdim)*(double)(DWB-DWT+1))-\
      (coord.mindim)*(0.0))/(coord.maxdim-coord.mindim));
   setviewport(DWL+1,BOX1_B+DWT+1,DWR+1,BOX1_B+DWB+1,1);

   switch (g_active_dim)
      {
      case 1:
            h_off = s_Var.winsize;
            break;
      case 2:
            h_off = s_Box.winsize;
            break;
      default:
            ;
      }

   ist=(long)(floor(coord.start_time*(double)g_voice_rate));
   iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));

   if (ist > 0)
   {
      if (ist >= h_off)
      {
         ist -= h_off;
         h_off = 0;
      }
      else
      {
         h_off -= ist;
         ist = 0;
      }
   }

   h_inc = (double)h_range/(double)(iend-ist+1);
```

```
   h_pos = h_inc * h_off;

   h_zero = (int) h_pos;

   //start drawing
   moveto(h_zero,v_zero);

   //draw waveform

   for (i=ist; i<iend+1; i++)
   {
      XMMcopyfrom(sizeof(*data),dim_handle,i*sizeof(*data),&*data);
      spy = v_zero-(int)(((*data)/(coord.maxdim-coord.mindim))\
         *(double)(v_range));
      spx = (int) h_pos;
      if(i == ist)
      {
         moveto(spx,spy);
      }
      lineto(spx,spy);
      h_pos += h_inc;
   }
   delete data;
   SetPortMiddle();
}

/*
//DrawDim
//This routine draws the dimension waveform

void DrawDim(void)
{
   long int i;              //counter
   int v_range, h_range;    //vertical and horizontal range
   int v_zero;              //vertical zero coordinate
   double h_inc;            //horizontal increment
   double h_pos=0.0;        //current horizontal position
   int spx, spy;            //data coordinates
   double far *data;        //data value
   long int ist, iend;      //start and end samples

   data = new double far;

   //calculate range sizes
   v_range = DWB-DWT+1;
   h_range = DWR-DWL+1;

   //calculate zero coordinate
   v_zero = (int)((((coord.maxdim)*(double)(DWB-DWT+1))-\
      (coord.mindim)*(0.0))/(coord.maxdim-coord.mindim));
   setviewport(DWL+1,BOX1_B+DWT+1,DWR+1,BOX1_B+DWB+1,1);

   //start drawing
   moveto(0,v_zero);
```

```
    ist=(long)(floor(coord.start_time*(double)g_voice_rate));
    iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));


    h_inc = (double)h_range/(double)(iend-ist+1);

    //draw waveform

    for (i=ist; i<iend+1; i++)
    {
       XMMcopyfrom(sizeof(*data),dim_handle,i*sizeof(*data),&*data);
       spy = v_zero-(int)(((*data)/(coord.maxdim-coord.mindim))\
          *(double)(v_range));
       spx = (int) h_pos;
       lineto(spx,spy);
       h_pos += h_inc;
    }
    delete data;
    SetPortMiddle();
}
*/

//DisplayDim
//This routine handles the production of the dimension display

void DisplayDim(void)
{
    double total_time;    //total time displayed
    double samp_inc;      //time per sample

    SetPortMiddle();
    clearviewport();


    DrawDimFrame();
    WriteDimTitle();
    WriteVertDim();
    WriteTimeDim();
    WriteDimInfo();

    total_time = (double)(g_voice_len-1)/(double)g_voice_rate;
    samp_inc = 1.0/(double)g_voice_rate;
    g_tot_time = total_time;

    DrawHashDim(samp_inc);
    DrawDim();

}

//WriteDimInfo
//This route writes statistical information about dimension waveform
void WriteDimInfo(void)
{
```

```
    char far *out_str;    //output string
    char far *num_str;    //numeric output string
    int th;               //text height

    out_str=new char far [80];
    num_str=new char far [80];

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="Min. Dimension:  ";

    th = textheight(out_str);
    moveto(10,(BOX2_B-BOX1_B)-8-th);
    outtext(out_str);

    sprintf(num_str,"%lf",g_mindim);
    outtext(num_str);

    out_str="  Max. Dimension:  ";
    outtext(out_str);

    sprintf(num_str,"%lf",g_maxdim);
    outtext(num_str);

    out_str="  Ave. Dimension:  ";
    outtext(out_str);

    sprintf(num_str,"%lf",g_avedim);
    outtext(num_str);

    out_str="  Std. Dev.:  ";
    outtext(out_str);

    sprintf(num_str,"%lf",g_stddev);
    outtext(num_str);

    delete[] out_str;
    delete[] num_str;
}
```

B.11 Listing of range.cpp

```
// range.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

//GetNewDouble
//This routine get a doulbe value from the user
```

```
void GetNewDouble(double *x)
{
   int i=0;                    //build string length
   char ch_in;                 //input character
   char done=0;                //routine exit flag
   int dec=0;                  //decimal allowed
   int min=0;                  //minus allowed
   char far *out_str;          //output string
   char far *ch_str;           //input building string
   char far *last_ch;          //last character string
   char far *n_str;            //number string
   double y=0.7;               //initialized double
   int tw;                     //textwidth
   char **endptr=NULL;         //null pointer

   out_str = new char far [80];
   n_str = new char far [80];
   ch_str = new char far [80];
   last_ch = new char far [80];


   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Enter New Value: ";
   moveto(2,142);
   outtext(out_str);

   while (done == 0)
   {
      ch_in = getch();
      switch (ch_in)
      {
         case '\r':  //enter
         {
            done = 1;
            n_str[i]=0;
            break;
         }
         case '\b':  //backspace
         {
            if (i > 0)
            {
               i--;
               if (i < dec)
               {
                  dec = 0;
               }
               if (i==0)
               {
                  min = 0;
               }
               last_ch[0] = n_str[i];
               last_ch[1]=0;
```

```
            tw=textwidth(last_ch);
            moverel(-tw,0);
            setcolor(0);
            outtext(last_ch);
            setcolor(15);
            moverel(-tw,0);
        }
        break;
    }
    case '-':                   //minus
    {
        if (i==0)
        {
            n_str[i]=ch_in;
            ch_str[0]=ch_in;
            ch_str[1]=0;
            outtext(ch_str);

            i++;
            min = 1;
        }
        break;
    }
    case '.':                   //decimal
    {
        if ((dec == 0)&&((i-min)>0))
        {
        n_str[i]=ch_in;
        ch_str[0]=ch_in;
        ch_str[1]=0;
        outtext(ch_str);

        i++;
        dec = i;

        }
        break;
    }
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
    case '0':
    {
        n_str[i]=ch_in;
        ch_str[0]=ch_in;
        ch_str[1]=0;
        outtext(ch_str);
```

```
            i++;
          }
        }

      }

      if (dec==0)
      {
        n_str[i++]='.';
        n_str[i++]='0';
        n_str[i]=0;
      }
      else if (dec==i)
      {
        n_str[i++]='0';
        n_str[i]=0;
      }

      y=strtod(n_str, endptr);
      *x=y;
      delete[] out_str;
      delete[] ch_str;
      delete[] last_ch;
      delete[] n_str;
}

//GetTimeRange
//This routine gets the time range parameters from the user

void GetTimeRange(void)
{
   char far *out_str;  //output string
   char far *num_str;  //numeric output string

   char done=0;        //internal menu exit flag
   char finished=0;    //routine exit
   char ch_in;         //input character
   double x=0.0;       //temp parameter variable

   out_str=new char far [80];
   num_str=new char far [80];


   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   while (finished==0)
   {
      SetPortBottom();
      clearviewport();

      out_str="<1> Change Start Time = ";
      moveto(2,2);
      outtext(out_str);
```

```
sprintf(num_str,"%f",coord.start_time);
outtext(num_str);

out_str="<2> Change Stop Time = ";
moveto(2,22);
outtext(out_str);
sprintf(num_str,"%f",coord.stop_time);
outtext(num_str);

out_str="<3> Full Range";
moveto(2,42);
outtext(out_str);

out_str="<A> Accept New Range";
moveto(2,62);
outtext(out_str);

done=0;

while (done==0)
{
   ch_in = getch();
   switch ((char)ch_in)
   {
      case '1':
      {
         GetNewDouble(&x);
         if ((0.0<=x)&&(x<coord.stop_time))
         {
            coord.start_time = x;
         }
         else
         {
            OutofRange();
         }
         done=1;

         break;
      }
      case '2':
      {
         GetNewDouble(&x);
         if ((coord.start_time<x)&&(x<=g_tot_time))
         {
            coord.stop_time = x;
         }
         else
         {
            OutofRange();
         }
         done=1;
         break;
      }
      case '3':
```

```
            {
               coord.start_time = 0.0;
               coord.stop_time = g_tot_time;
               done=1;
               break;
            }
            case 'a':
            case 'A':
            {
               done=1;
               finished=1;
               break;
            }
         }
      }

   }
   delete[] out_str;
   delete[] num_str;

}

//GetVoiceRange
//This routine gets the voice level range parameters from the user

void GetVoiceRange(void)
{
   char far *out_str;    //output string
   char far *num_str;    //numeric output string

   char done=0;          //internal menu exit flag
   char finished=0;      //routine exit
   char ch_in;           //input character
   double x=0.0;         //temp parameter variable


   out_str=new char far [80];
   num_str=new char far [80];


   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   while (finished==0)
   {
      SetPortBottom();
      clearviewport();

      out_str="<1> Change Maximum % Threshold = ";
      moveto(2,2);
      outtext(out_str);
      sprintf(num_str,"%f",coord.maxvoice);
      outtext(num_str);
```

```
out_str="<2> Change Minimum % Threshold = ";
moveto(2,22);
outtext(out_str);
sprintf(num_str,"%f",coord.minvoice);
outtext(num_str);

out_str="<3> Fit to Window (Current Time Span only)";
moveto(2,42);
outtext(out_str);

out_str="<4> Full Range";
moveto(2,62);
outtext(out_str);

out_str="<A> Accept New Range";
moveto(2,82);
outtext(out_str);

done=0;

while (done==0)
{
   ch_in = getch();
   switch ((char)ch_in)
   {
      case '2':
      {
         GetNewDouble(&x);
         if ((-100.0<=x)&&(x<coord.maxvoice))
         {
            coord.autoscale=0;
            coord.minvoice = x;
         }
         else
         {
            OutofRange();
         }
         done=1;

         break;
      }
      case '1':
      {
         GetNewDouble(&x);
         if ((coord.minvoice<x)&&(x<=100.0))
         {
            coord.autoscale=0;
            coord.maxvoice = x;
         }
         else
         {
            OutofRange();
         }
         done=1;
```

```
                    break;
                }
                case '3':
                {
                    coord.autoscale=1;
                    autoscale();
                    done=1;
                    break;
                }
                case '4':
                {
                    coord.autoscale=0;
                    coord.minvoice = -100.0;
                    coord.maxvoice = 100.0;
                    done=1;
                    break;
                }
                case 'a':
                case 'A':
                {
                    done=1;
                    finished=1;
                    break;
                }
            }
        }

    }
    delete[] out_str;
    delete[] num_str;

}

//autoscale
//This routine scale a waveform to the maximum window display range
void autoscale(void)
{

    long ist, iend; //first and last samples
    long i;         //sample counter
    double minv, maxv; //min and max values

    short far *data;   //data value

    data = new short far;


    ist=(long)(floor(coord.start_time*\
        (double)g_voice_rate));
    iend=(long)(ceil(coord.stop_time*\
        (double)g_voice_rate));
    minv = 33000.0;
    maxv = -33000.0;
    for (i=ist; i<=iend; i++)
```

```
    {
       XMMcopyfrom(sizeof(*data),voice_handle,\
          i*sizeof(*data),&*data);

       if ((double)(*data)<minv)
       {
          minv=(double)(*data);
       }
       if ((double)(*data)>maxv)
       {
          maxv=(double)(*data);
       }
    }
    coord.minvoice = minv/327.68;
    coord.maxvoice = maxv/327.68;
    delete data;

}
//GetDimeRange
//This routine gets the dimension value range parameters from the user

void GetDimRange(void)
{
    char far *out_str;    //output string
    char far *num_str;    //numeric output string

    char done=0;          //internal menu exit
    char finished=0;      //routine exit
    char ch_in;           //input character
    double x=0.0;         //temp parameter variable

    out_str=new char far [80];
    num_str=new char far [80];


    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    while (finished==0)
    {
       SetPortBottom();
       clearviewport();

       out_str="<1> Change Minimum Dimension = ";
       moveto(2,2);
       outtext(out_str);
       sprintf(num_str,"%f",coord.mindim);
       outtext(num_str);

       out_str="<2> Change Maximum Dimension = ";
       moveto(2,22);
       outtext(out_str);
       sprintf(num_str,"%f",coord.maxdim);
       outtext(num_str);
```

```
out_str="<3> Normal Range";
moveto(2,42);
outtext(out_str);

out_str="<A> Accept New Range";
moveto(2,62);
outtext(out_str);

done=0;

while (done==0)
{
   ch_in = getch();
   switch ((char)ch_in)
   {
      case '1':
      {
         GetNewDouble(&x);
         if ((0.0<=x)&&(x<coord.maxdim))
         {
            coord.mindim = x;
         }
         else
         {
            OutofRange();
         }
         done=1;

         break;
      }
      case '2':
      {
         GetNewDouble(&x);
         if ((coord.mindim<x)&&(x<=3.0))
         {
            coord.maxdim = x;
         }
         else
         {
            OutofRange();
         }
         done=1;
         break;
      }
      case '3':
      {
         coord.mindim = 1.0;
         coord.maxdim = 2.0;
         done=1;
         break;
      }
      case 'a':
      case 'A':
```

```
            {
               done=1;
               finished=1;
               break;
            }
         }
      }

   }
   delete[] out_str;
   delete[] num_str;

}

//GetMinMaxDim
//This routine performs some statistical measures on the dimension
//waveform

void GetMinMaxDim(void)
{
   long int ist, iend;  //start and end samples
   long int i;          //sample counter
   double far *data;    //data value

   double sum, sum2;    //summation variables

   data = new double far;

   ist=(long)(floor(coord.start_time*(double)g_voice_rate));
   iend=(long)(ceil(coord.stop_time*(double)g_voice_rate));
   g_mindim=3.0;
   g_maxdim=0.0;
   sum=0.0;
   sum2=0.0;
   g_countd=0;

   //calculate minimum and maximum dimensions
   for (i=ist; i<=iend; i++)
   {
      XMMcopyfrom(sizeof(*data),dim_handle,i*sizeof(*data),&*data);

      if (*data<g_mindim)
      {
         g_mindim=*data;
      }
      if (*data>g_maxdim)
      {
         g_maxdim=*data;
      }
      sum += *data;
      sum2 += ((*data)*(*data));
      g_countd++;
   }
```

```
   //calculate average
   g_avedim = sum/(double)g_countd;

   //calculate standard deviation
   g_stddev = sqrt((sum2 - sum*sum/g_countd)/(g_countd - 1));
   delete data;
}

//ProcessRanges
//This routine handle user updates to the display ranges

void ProcessRanges(void)
{
   GetMinMaxDim();
   DisplayVocFile();
   DisplayDim();

}
```

B.12 Listing of analysis.cpp

```
//analysis.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

//DimBelow
//This routine counts the number of dimension values below a threshold
//value
void DimBelow(void)
{
   double th;           //threshold
   long i;              //counter
   long iend;           //end of data
   long inum;           //number of samples
   double far *data;    //data value

   data = new double far;

   switch (g_active_dim)
   {
      case 1:
      {
         iend = g_voice_len - s_Var.winsize;
         break;
      }
      case 2:
      {
         iend = g_voice_len - s_Box.winsize;
         break;
      }
```

```
    }

    GetThreshold(&th);

    inum=0;

    for (i=0;i<iend;i++)
    {
       XMMcopyfrom(sizeof(*data),dim_handle,i*sizeof(*data),data);
       if (*data < th)
       {
          inum++;
       }
    }
    DispBelow(inum);
    delete data;
}

//DimAbove
//This routine counts the number of samples above a threshold value

void DimAbove(void)
{
    double th;              //threshold
    long i;                 //counter
    long iend;              //ending sample
    long inum;              //number of samples
    double far *data;       //data value

    data = new double far;

    switch (g_active_dim)
    {
       case 1:
       {
          iend = g_voice_len - s_Var.winsize;
          break;
       }
       case 2:
       {
          iend = g_voice_len - s_Box.winsize;
          break;
       }
    }

    GetThreshold(&th);

    inum=0;

    for (i=0;i<iend;i++)
    {
       XMMcopyfrom(sizeof(*data),dim_handle,i*sizeof(*data),data);
       if (*data > th)
       {
```

```
            inum++;
         }
      }
      DispAbove(inum);
      delete data;
}

//GetThreshold
//this routine get a threshold value from the user

void GetThreshold(double * th)
{
   char far *out_str;    //output string

   out_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Enter Threshold Value:  ";
   moveto(2,2);
   outtext(out_str);

   GetNewDouble(th);

   delete[] out_str;
}

//DispBelow
//This routine displays the number of dimension sample below a
//threshold

void DispBelow(long inum)
{
   char far *out_str;  //output string
   char far *num_str;  //numeric output string

   out_str=new char far [80];
   num_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Dimension samples below threshold  ";
   moveto(2,2);
   outtext(out_str);

   sprintf(num_str,"%li",inum);
```

```
    outtext(num_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }

    delete[] out_str;
    delete[] num_str;
}

//DispAbove
//This routine displays the number of dimension sample above a
//threshold

void DispAbove(long inum)
{
    char far *out_str;    //output string
    char far *num_str;    //input string

    out_str=new char far [80];
    num_str=new char far [80];

    SetPortBottom();
    clearviewport();

    settextstyle(8,0,0);
    setusercharsize(1,2,1,2);

    out_str="Dimension samples above threshold  ";
    moveto(2,2);
    outtext(out_str);

    sprintf(num_str,"%li",inum);
    outtext(num_str);

    out_str="Press Any Key to Continue";
    moveto(2,62);
    outtext(out_str);

    while (kbhit()==0)
    {
    }

    delete[] out_str;
    delete[] num_str;
}
```

B.13 Listing of screensv.cpp

```
//screensv.cpp

#ifndef MYGLOBAL
   #define MYGLOBAL
   #include "std.h"
#endif

//GetScreenName
//This routine gets the tif file name to save the screen to

void GetScreenName(char *name)
{
   int i=0;            //filename length
   char ch_in;         //input character
   char done=0;        //exit flag
   char far *out_str; //output string
   char far *ch_str;  //filename building string
   char far *last_ch; //last enter character string
   int tw;             //textwidth

   out_str = new char far [80];
   ch_str = new char far [80];
   last_ch = new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Enter Name of TIFF File to Save: ";
   moveto(2,202);
   outtext(out_str);

   while (done == 0)
   {
      ch_in = getch();
      switch (ch_in)
      {
         case '\r':
         {
            done = 1;
            name[i]=0;
            tfn[i]=0;
            break;
         }
         case '\b':
         {
            if (i > 0)
            {
               i--;
               last_ch[0] = name[i];
               last_ch[1]=0;
               tw=textwidth(last_ch);
               moverel(-tw,0);
```

```
                setcolor(0);
                outtext(last_ch);
                setcolor(15);
                moverel(-tw,0);
             }
             break;
          }
          default:
          {
             name[i]=ch_in;
             tfn[i]=ch_in;
             ch_str[0]=ch_in;
             ch_str[1]=0;
             outtext(ch_str);

             i++;
          }
       }

    }
    delete[] last_ch;
    delete[] ch_str;
    delete[] out_str;
}

//FileError
//Error Message Routine

void FileError(void)
{
   char far *out_str;     //output string

   out_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="File Error: Not Saved";
   moveto(2,22);
   outtext(out_str);

   out_str="Press Any Key to Continue";
   moveto(2,62);
   outtext(out_str);

   while (kbhit()==0)
   {
   }
   delete[] out_str;

}
```

```
//TIFF_WRITE
//This routine writes a tiff image file containing the current screen
//infomation in a bilevel format.

void TIFF_WRITE(char *out_file)
{
   int i,j;                        //couuters
   int  c,d;                       //pixel colour(individual, group of 8)
   int k;                          //pixel counter mod 8
   int dp;                         //binary pixel colour 0=black,1=white
   unsigned char buf[220];         // fixed header size for TIF file
   short int *ibuf;                //header pointer short integer size
   long int  *lbuf;                //header pointer long integer size
   FILE *output;                   //tiff file pointer

   ibuf=(short int*) &buf[10];
   lbuf=(long int*) &buf[10];

   if ((output = fopen(out_file, "wb")) == NULL) //open file as binary
   {
    FileError();
    return;
   }

//              Create TIFF file header for writing

   buf[0]='I';
   buf[1]='I';                                  // byte order identifier
   buf[2]= 42;                                  // 2 TIF file identifier
   buf[3]= 0;
   buf[4]= 8;
   buf[5]= 0;
   buf[6]= 0;
   buf[7]= 0;                                   // 6 first IFD offset
//                                  IFD's
   buf[8]= 12;
   buf[9]= 0;                                   // 8 number of IFD's
//      IFD 1
   ibuf[0]= 256;                          // 10 image width identifier
   ibuf[1]= 3;                            // 12 short field type
   ibuf[2]= 1;
   ibuf[3]= 0;                            // 14 one entry immediate
   ibuf[4]= (short int) 1024;             // 18 image width entry
   ibuf[5]= 0;

//      IFD 2

   ibuf[6]= 257;                          // 22 image length identifier
   ibuf[7]= 3;                            // 24 short field type
   ibuf[8]= 1;
   ibuf[9]= 0;                            // 26 one entry immediate
   ibuf[10]= (short int) 768;
   ibuf[11]= 0;                           // 30 image length entry
```

```
//      IFD 3
   ibuf[12]= 259;                    // 34 image compression identifier
   ibuf[13]= 3;                      // 36 long field type
   ibuf[14]= 1;
   ibuf[15]= 0;                      // 38 one entry immediate
   ibuf[16]= 1;
   ibuf[17]= 0;                      // 42 no compression used
//      IFD 4
   ibuf[18]= 262;                    // 46 photometric interpretation
   ibuf[19]= 3;                      // 48 long field type
   ibuf[20]= 1;
   ibuf[21]= 0;                      // 50 one entry immediate
   ibuf[22]= 1;
   ibuf[23]= 0;                      // 54 photometric interpretation
//      IFD 5
   ibuf[24]= 266;                    // 58 Fill Order
   ibuf[25]= 3;                      // 60 short field type
   ibuf[26]= 1;
   ibuf[27]= 0;                      // 62 one entry immediate
   ibuf[28]= 1;
   ibuf[29]= 0;                      // 66 photometric interpretation
//      IFD 6
   ibuf[30]= 270;                    // 70 strip offset
   ibuf[31]= 2;                      // 72 ascii field type
   lbuf[16]= 46;                     // 74 46 characters
   lbuf[17]= 174;                    // 78 strip offset
//      IFD 7
   ibuf[36]= 273;                    // 82 strip offset
   ibuf[37]= 4;                      // 84 long field type
   lbuf[19]= 1;                      // 86 one entry immediate
   lbuf[20]= 220;                    // 90 strip offset
//      IFD 8

   ibuf[42]= 278;                    // 94 Rows per strip
   ibuf[43]= 3;                      // 96 short field type
   ibuf[44]= 1;
   ibuf[45]= 0;                      // 98 one entry immediate
   ibuf[46]= (short int) 768;
   ibuf[47]= 0;                      // 102 Rows per strip

//      IFD 9

   ibuf[48]= 279;                    // 106 Strip byte count
   ibuf[49]= 4;                      // 108 long field type
   lbuf[25]= 1;                      // 110 one entry immediate
   lbuf[26]= (long)ceil((double)1024/(double)8)*768;
                                     // 114 Strip byte count

//      IFD 10
   ibuf[54]= 282;                    // 118 XResolution
   ibuf[55]= 5;                      // 120 long field type
   lbuf[28]= 1;                      // 122 one offset
   lbuf[29]= 158;                    // 126 XResolution location
```

```
//      IFD 11
   ibuf[60]= 283;                    // 130 YResolution
   ibuf[61]= 5;                      // 132 long field type
   lbuf[31]= 1;                      // 134 one offset
   lbuf[32]= 166;                    // 138 YResolution location
//      IFD 12
   ibuf[66]= 296;                    // 142 Resolution unit
   ibuf[67]= 3;                      // 144 long field type
   ibuf[68]= 1;
   ibuf[69]= 0;                      // 146 one entry immediate
   ibuf[70]= 2;
   ibuf[71]= 0;                      // 150 Resolution unit
//      End of IFD flag
   lbuf[36]= 0;                      // 154
//      Resolution arrays
   lbuf[37]=100;                     // 158
   lbuf[38]=1;                       // 162
   lbuf[39]=100;                     // 166
   lbuf[40]=1;                       // 170
//      Save message string
   strncpy((char *)&lbuf[41],\
      "A Fine TIFF image brought to you by Sndplot.",46);

   fwrite(buf,1,220,output);            // write header to file

   setviewport(0,0,1023,767,1);     // set viewport
   for(i=0;i<768;i++)
   {
      for(j=0;j<1024;j++)
      {
         k=j%8;
         if (k==0)
         {
            d=0;
         }
         c=getpixel(j,i);        // get screen pixel

         if (c==0) dp=1;    // convert black to white
         else dp=0;               // all else to black

         d += dp*(int)pow(2.0,7.0 - (double)k);
         if (k==7||j==1023)
         {
            fputc(d,output);
         }
      }
   }
   fclose(output);               // close file after done
   return;
}

//SaveScreen
//This routine handles the screen saving process
```

```
void SaveScreen(void)
{
   char * filename;     //tiff file name

   filename = new char [80];

// SaveTmpScreen();
   GetScreenName(filename);
   SetEnlarge();
   Enlarge();
   TIFF_WRITE(filename);
   SetNormal();
// RestoreTmpScreen();
   RedrawScreen();
   delete[] filename;
}


voidSaveTmpScreen(void)
{
   int i, j;
   int pix;
   FILE * tmpscr;

   tmpscr = fopen("scrnswap.tmp", "wb");

   for (i=TOP; i<=BOTTOM; i++)
   {
      for(j=LEFT; j<=RIGHT; j++)
      {
         pix = getpixel(j,i);
         fputc(pix, tmpscr);
      }
   }
   fclose(tmpscr);

}

//Enlarge
//This routine redraws the screen without the menu box, enlarging the
//waveform displays
voidEnlarge(void)
{
   SetPortAll();
   clearviewport();

   MainWindow();

   if (menu_number!=1)
   {
      DisplayVocFile();
      if (menu_number!=2)
      {
         DisplayDim();
```

```
      }
   }

}
//RedrawScreen
//This routine redraws the original screen
void RedrawScreen(void)
{
   SetPortAll();
   clearviewport();

   MainWindow();

   if (menu_number!=1)
   {
      DisplayVocFile();
      if (menu_number!=2)
      {
         DisplayDim();
      }
   }

}

voidRestoreTmpScreen(void)
{
   int i, j;
   int pix;
   FILE * tmpscr;

   tmpscr = fopen("scrnswap.tmp", "rb");

   for (i=TOP; i<=BOTTOM; i++)
   {
      for(j=LEFT; j<=RIGHT; j++)
      {
         pix = fgetc(tmpscr);
         putpixel(j,i,pix);
      }
   }

   fclose(tmpscr);

}
```

B.14 Listing of svoc.cpp

```
//svoc.cpp

#ifndef MYGLOBALS
   #define MYGLOBALS
   #include "std.h"
#endif
```

```
//GetVoiceName
//This routine gets the name of a voc file to save to from the user
void GetVoiceName(char *name)
{
   int i=0;           //length of file name
   char ch_in;        //input character
   char done=0;       //done flag
   char far *out_str; //output string
   char far *ch_str;  //file name build string
   char far *last_ch; //last character entered
   int tw;            //text width

   out_str = new char far [80];
   ch_str = new char far [80];
   last_ch = new char far [80];

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Enter Name of VOC File to Save: ";
   moveto(2,202);
   outtext(out_str);

   while (done == 0)
   {
      ch_in = getch();
      switch (ch_in)
      {
         case '\r':
         {
            done = 1;
            name[i]=0;
            break;
         }
         case '\b':
         {
            if (i > 0)
            {
               i--;
               last_ch[0] = name[i];
               last_ch[1]=0;
               tw=textwidth(last_ch);
               moverel(-tw,0);
               setcolor(0);
               outtext(last_ch);
               setcolor(15);
               moverel(-tw,0);
            }
            break;
         }
         default:
         {
            name[i]=ch_in;
```

```
            ch_str[0]=ch_in;
            ch_str[1]=0;
            outtext(ch_str);

            i++;
          }
        }

      }
      delete[] last_ch;
      delete[] ch_str;
      delete[] out_str;
}

//WriteVocFile
//This routine write the voc data in extended memory to file

void WriteVocFile(char *outfile)
{
   char  b[42];  //header and block header
   int i;        //counter
   long int j;   //counter
   long fsize;   //file size
   char bt1,bt2,bt3; //file size component bytes
   char tb;          //termination block (for playback)
   short int far * sp; //data value
   FILE   *PCMFile;    //voc file pointer

   sp = new short int far;
   if ((PCMFile = fopen(outfile, "wb")) == NULL) //open file as binary
   {
    FileError();
    return;
   }

   //write header info
   b[0]=0x43;
   b[1]=0x72;
   b[2]=0x65;
   b[3]=0x61;
   b[4]=0x74;
   b[5]=0x69;
   b[6]=0x76;
   b[7]=0x65;
   b[8]=0x20;
   b[9]=0x56;
   b[10]=0x6f;
   b[11]=0x69;
   b[12]=0x63;
   b[13]=0x65;
   b[14]=0x20;
   b[15]=0x46;
   b[16]=0x69;
   b[17]=0x6c;
```

```
b[18]=0x65;
b[19]=0x1a;
b[20]=0x1a;
b[21]=0x00;
b[22]=0x14;
b[23]=0x01;
b[24]=0x1f;
b[25]=0x11;

b[26]=0x09;

//calculate file size component bytes
fsize = (2*g_voice_len)+12;

bt3 =(char) (fsize%256);

fsize = (fsize/256);

bt2 =(char)(fsize%256);

bt1 = (char)(fsize/256);

b[27]=bt3;
b[28]=bt2;
b[29]=bt1;

b[30]=0x44;
b[31]=0xac;

b[32]=0x00;
b[33]=0x00;
b[34]=0x10;
b[35]=0x01;
b[36]=0x04;
b[37]=0x00;
b[38]=0x00;
b[39]=0x00;
b[40]=0x00;
b[41]=0x00;

for (i=0;i<42;i++)
{
   fwrite(&b[i], sizeof(b[i]), 1, PCMFile);
}

for (j=0;j<g_voice_len;j++)
{
   XMMcopyfrom(sizeof(short),voice_handle,j*sizeof(short),sp);
   fwrite(sp, sizeof(short), 1, PCMFile);
}
tb=0;
fwrite(&tb, sizeof(tb), 1, PCMFile);
fclose(PCMFile);
delete sp;
```

```
}

//SaveVoice
//This routine handles the saving of a voc file

void SaveVoice(void)
{
   char * filename; //voc filename

   filename = new char [80];

   GetVoiceName(filename);
   WriteVocFile(filename);
   delete[] filename;
}
```

B.15 Listing of noise.cpp

```
//noise.cpp

#ifndef MYGLOBALS
   #define MYGLOBALS
   #include "std.h"
#endif


//FindPeak
//This routine locates the highest amplitude in a voice waveform


void FindPeak(long * peak)
{
   short max=0;    //maximum amplitude
   long i;         //counter
   short * data;   //voice data value


   data = new short;


   for (i=0;i<g_voice_len;i++)
   {
      XMMcopyfrom(sizeof(*data),voice_handle,i*sizeof(*data),data);
      if (max < abs(*data))
      {
         max = abs(*data);
         *peak = i;
      }
   }


   delete data;
```

```
}


//CalcWindowSignalPower
//This routine calculates the signal power within a the peak value
void CalcWindowSignalPower(long peak, double * swp)
{
   long i;         //counter
   short * data;   //voice data value


   data = new short;
   *swp =0.0;


   if (peak < 256)
   {
      peak = 256;
   }
   if (peak > (g_voice_len - 256))
   {
      peak = g_voice_len - 256;
   }


   for (i=(peak-256);i<(peak+256);i++)
   {
      XMMcopyfrom(sizeof(*data),voice_handle,i*sizeof(*data),data);
      *swp += (*data)*(*data);
   }


   *swp /= 512.0;


   delete data;
}


//GetSNR
//This routine gets the desired SNR level from the user
void GetSNR(double * x)
{
   int i=0;             //length of input string
   char ch_in;          //input character
   char done=0;         //exit routine flag
   int dec=0;           //decimal point allowed
   int min=0;           //minus sign allowed
   char far *out_str;   //output string
   char far *ch_str;    //input string
   char far *last_ch;   //last character entered
   char far *n_str;     //numeric string
   double y=0.7;        //initialed double
```

```
int tw;              //text width
char **endptr=NULL; //null pointer

out_str = new char far [80];
n_str = new char far [80];
ch_str = new char far [80];
last_ch = new char far [80];


settextstyle(8,0,0);
setusercharsize(1,2,1,2);
SetPortBottom();
clearviewport();

out_str="Enter SNR (dB): ";
moveto(2,142);
outtext(out_str);

while (done == 0)
{
   ch_in = getch();
   switch (ch_in)
   {
      case '\r':
      {
         done = 1;
         n_str[i]=0;
         break;
      }
      case '\b':
      {
         if (i > 0)
         {
            i--;
            if (i < dec)
            {
               dec = 0;
            }
            if (i==0)
            {
               min = 0;
            }
            last_ch[0] = n_str[i];
            last_ch[1]=0;
            tw=textwidth(last_ch);
            moverel(-tw,0);
            setcolor(0);
            outtext(last_ch);
            setcolor(15);
            moverel(-tw,0);
         }
         break;
      }
      case '-':
```

```
        {
           if (i==0)
           {
              n_str[i]=ch_in;
              ch_str[0]=ch_in;
              ch_str[1]=0;
              outtext(ch_str);

              i++;
              min = 1;
           }
           break;
        }
        case '.':
        {
           if ((dec == 0)&&((i-min)>0))
           {
           n_str[i]=ch_in;
           ch_str[0]=ch_in;
           ch_str[1]=0;
           outtext(ch_str);

           i++;
           dec = i;

           }
           break;
        }
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
        case '0':
        {
           n_str[i]=ch_in;
           ch_str[0]=ch_in;
           ch_str[1]=0;
           outtext(ch_str);

           i++;
        }
     }

  }

  if (dec==0)
  {
    n_str[i++]='.';
    n_str[i++]='0';
```

```
      n_str[i]=0;
    }
    else if (dec==i)
    {
      n_str[i++]='0';
      n_str[i]=0;
    }

    y=strtod(n_str, endptr);
    *x=y;
    delete[] out_str;
    delete[] ch_str;
    delete[] last_ch;
    delete[] n_str;
}


//GetNoiseLevel
//This routine calculate the amplitude of noise needed for given SNR


void GetNoiseLevel(double *n_l)
{
   double snr=0.0;  //Signal to noise ratio
   double swp=0.0;  //Power of signal within window around peak
   long peak=0;     //Peak signal value location


   FindPeak(&peak);
   CalcWindowSignalPower(peak, &swp);
   GetSNR(&snr);


   *n_l = swp/exp(snr/10.0);
}


//Noise
//This routine handles the addition of white gaussian noise to a voice
// waveform


void Noise(void)
{
   long idum = -432;    //pseudorandom seed
   double n_l=0.0;      //noise amplitude


   GetNoiseLevel(&n_l);
   if (n_l < 32000.0 && n_l > -32000.0)
   {
      AddNoise(g_voice_len,n_l, &idum);
   }
   else
```

```
    {
       LevelTooHigh();
    }

}

//ProcessBlock routine add wgn to data block

void AddNoise(unsigned long size, double n_l, long *idum)
{
    short int far * newval;    //new data value
    short int far * temp;      //temp data value
    long int orlon;            //old data value
    long int newlon;           //new data value

    unsigned long ns;          //number of samples
    unsigned long i;           //counter
    double wgs;                //white gaussian double
    long wgsamp;               //white gaussian long

    newval = new short int far;
    temp = new short int far;

    ns = size;

    for (i=0; i<ns; i++)
    {

       XMMcopyfrom(sizeof(short),voice_handle,i*sizeof(short),temp);

       orlon = (long) *temp;

       wgs = gasdev(idum);
       wgsamp = (long) (n_l * wgs);

       newlon = orlon + wgsamp;

       if (newlon< -32767L)
       {
          newlon = -32767;
       }
       if (newlon > 32767L)
       {
          newlon = 32767;
       }

       *newval = (short) newlon;



       XMMcopyto(sizeof(short), newval, voice_handle, i*sizeof(short));

    }
    delete newval;
```

```
   delete temp;
}

//LevelTooHigh
//Error Message Routine

void LevelTooHigh(void)
{
   char far *out_str; //output string

   out_str=new char far [80];

   SetPortBottom();
   clearviewport();

   settextstyle(8,0,0);
   setusercharsize(1,2,1,2);

   out_str="Noise Level Too High";
   moveto(2,22);
   outtext(out_str);

   out_str="Press Any Key to Continue";
   moveto(2,62);
   outtext(out_str);

   while (kbhit()==0)
   {
   }
   delete[] out_str;

}
```

B.16 Listing of normal.cpp

```
// normal.cpp
#include "normdef.h"

#ifndef MYGLOB
   #define MYGLOB
   #include "std.h"
#endif

//gasdev
//This routine generates a white gaussian sample using
//transformational methods
double gasdev(long *idum)

{

   double fac,v1,x1,x2;  //transformation variables
```

```
   do
   {
      x1=ran2(idum);
      x2=ran2(idum);
   }
   while (x1 == 0.0);

   fac=sqrt(-2.0*log(x1));
   v1=cos(2*PI*x2);

   return v1*fac;


}
```

B.17 Listing of ran2.cpp

```
// ran2.cpp

#include "ran2def.h"

#ifndef MYGLOB
   #define MYGLOB
   #include "std.h"
#endif


//ran2 from crecipes
//This routine provides a pseudorandom long integer

double ran2(long *idum)
{
   long int j;
   long k;
   static long idum2=123456789L;
   static long iy=0;
   static long huge iv[NTAB];
   double temp;

   if (*idum <=0 || !iy)
   {
      if (-(*idum) <1)
      {
         *idum =1;
      }
      else
      {
         *idum = -(*idum);
      }
      idum2=(*idum);
      for (j=NTAB+7;j>=0;j--)
      {
         k=(*idum)/IQ1;
```

```
            *idum = IA1*(*idum-k*IQ1)-IR1*k;
            if (*idum <0)
            {
                *idum += IM1;
            }
            if (j<NTAB)
            {
               iv[j] = *idum;
            }
         }
         iy=iv[0];
      }
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-IR1*k;
      if (*idum < 0) *idum += IM1;
      k=idum2/IQ2;
      idum2=IA2*(idum2-k*IQ2)-IR2*k;
      if (idum2 < 0) idum2 += IM2;
      j=iy/NDIV;
      iy=iv[j]-idum2;
      iv[j]=*idum;
      if (iy < 1) iy += IMM1;
      if ((temp=AM*iy)>RNMX) return RNMX;
      else return temp;
   }
```

Appendix C
AX.25 Protocol

AX.25 Amateur Packet-Radio Link-Layer Protocol
Version 2.0 October 1984

2. AX.25 Link-Layer Protocol Specification

2.1 Scope and Field of Operation

In order to provide a mechanism for the reliable transport of data between two signaling terminals, it is necessary to define a protocol that can accept and deliver data over a variety of types of communications links. The AX.25 Link-Layer Protocol is designed to provide this service, independent of any other level that may or may not exist.

This protocol conforms to ISO Recommendations 3309, 4335 (including DAD 1&2) and 6256 high-level data link control (HDLC) and uses some terminology found in these documents. It also conforms with ANSI X3.66, which describes ADCCP, balanced mode.

This protocol follows, in principle, the CCITT X.25 Recommendation, with the exception of an extended address field and the addition of the Unnumbered Information (UI) frame. It also follows the principles of CCITT Recommendation Q.921 (LAPD) in the use of multiple links, distinguished by the address field, on a single shared channel.

As defined, this protocol will work equally well in either half- or full-duplex Amateur Radio environments.

This protocol has been designed to work equally well for direct connections between two individual amateur packet-radio stations or an individual station and a multiport controller.

This protocol allows for the establishment of more than one link-layer connection per device, if the device is so capable.

This protocol does not prohibit self-connections. A self-connection is considered to be when a device establishes a link to itself using its own address for both the source and destination of the frame.

Most link-layer protocols assume that one primary (or master) device (generally called a DCE, or data circuit-terminating equipment) is connected to one or more secondary (or slave) device(s) (usually called a DTE, or data terminating equipment). This type of unbalanced operation is not practical in a shared-RF Amateur Radio environment. Instead, AX.25 assumes

that both ends of the link are of the same class, thereby eliminating the two different classes of devices. The term DXE is used in this protocol specification to describe the balanced type of device found in amateur packet radio.

2.2 Frame Structure

Link layer packet radio transmissions are sent in small blocks of data, called frames. Each frame is made up of several smaller groups, called fields. Fig.1 shows the three basic types of frames. Note that the first bit to be transmitted is on the left side.

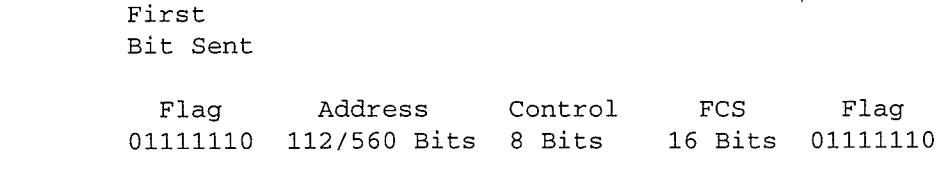


Fig. 1A -- U and S frame construction

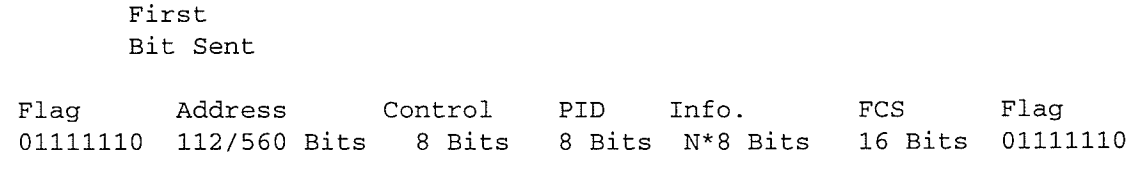


Fig. 1B -- Information frame construction

Each field is made up of an integral number of octets (or bytes), and serves a specific function as outlined below.

2.2.1 Flag Field

The flag field is one octet long. Since the flag is used to delimit frames, it occurs at both the beginning and end of each frame. Two frames may share one flag, which would denote the end of the first frame, and the start of the next frame. A flag consists of a zero followed by six ones followed by another zero, or 01111110 (7E hex). As a result of bit stuffing (see 2.2.6, below), this sequence is not allowed to occur anywhere else inside a complete frame.

2.2.2 Address Field

The address field is used to identify both the source of the frame and its destination. In addition, the address field contains the command/response information and facilities for level 2 repeater operation.

The encoding of the address field is described in 2.2.13.

2.2.3 Control Field

The control field is used to identify the type of frame being passed and control several attributes of the level 2 connection. It is one octet in length, and its encoding is discussed in 2.3.2.1, below.

2.2.4 PID Field

The Protocol Identifier (PID) field shall appear in information frames (I and UI) only. It identifies what kind of layer 3 protocol, if any, is in use.

The PID itself is not included as part of the octet count of the information field. The encoding of the PID is as follows:

```
M       L
S       S
B       B
yy01yyyy AX.25 layer 3 implemented.
yy10yyyy AX.25 layer 3 implemented.
11001100 Internet Protocol datagram layer 3 implemented.
11001101 Address resolution protocol layer 3 implemented.
11110000 No layer 3 implemented.
11111111 Escape character.  Next octet contains more Level 3
         protocol information.
```

Where:

A y indicates all combinations used.

Note:

All forms of yy11yyyy and yy00yyyy other than those listed above are reserved at this time for future level 3 protocols. The assignment of these formats is up to amateur agreement. It is recommended that the creators of level 3 protocols contact the ARRL Ad Hoc Committee on Digital Communications for suggested encodings.

2.2.5 Information Field

The information field is used to convey user data from one end of the link to the other. I fields are allowed in only three types of frames: the I frame, the UI frame, and the FRMR frame. The I field can be up to 256 octets long, and shall contain an integral number of octets. These constraints apply prior to the insertion of zero bits as specified in 2.2.6, below. Any information in the I field shall be passed along the link transparently, except for the zero-bit insertion (see 2.2.6) necessary to prevent flags from accidentally appearing in the I field.

2.2.6 Bit Stuffing

In order to assure that the flag bit sequence mentioned above doesn't appear accidentally anywhere else in a frame, the sending station shall monitor the bit sequence for a group of five or more contiguous one bits. Any time five contiguous one bits are sent the sending station shall insert a zero bit after the first one bit. During frame reception, any time five contiguous one bits are received, a zero bit immediately following five one bits shall be discarded.

2.2.7 Frame-Check Sequence

The frame-check sequence (FCS) is a sixteen-bit number calculated by both the sender and receiver of a frame. It is used to insure that the frame was not corrupted by the medium used to get the frame from the sender to the receiver. It shall be calculated in accordance with ISO 3309 (HDLC) Recommendations.

2.2.8 Order of Bit Transmission

With the exception of the FCS field, all fields of an AX.25 frame shall be sent with each octet's least-significant bit first. The FCS shall be sent most-significant bit first.

2.2.9 Invalid Frames

Any frame consisting of less than 136 bits (including the opening and closing flags), not bounded by opening and closing flags, or not octet aligned (an integral number of octets), shall be considered an invalid frame by the link layer. See also 2.4.4.4, below.

2.2.10 Frame Abort

If a frame must be prematurely aborted, at least fifteen contiguous ones shall be sent with no bit stuffing added.

2.2.11 Interframe Time Fill

Whenever it is necessary for a DXE to keep its transmitter on while not actually sending frames, the time between frames should be filled with contiguous flags.

2.2.12 Link Channel States

Not applicable.

2.2.13 Address-Field Encoding

The address field of all frames shall be encoded with both the destination and source amateur call signs for the frame. Except for the Secondary Station Identifier (SSID), the address field should be made up of upper-case alpha and numeric ASCII

characters only. If level 2 amateur "repeaters" are to be used, their call signs shall also be in the address field.

The HDLC address field is extended beyond one octet by assigning the least-significant bit of each octet to be an "extension bit". The extension bit of each octet is set to zero, to indicate the next octet contains more address information, or one, to indicate this is the last octet of the HDLC address field. To make room for this extension bit, the amateur Radio call sign information is shifted one bit left.

2.2.13.1 Nonrepeater Address-Field Encoding

If level 2 repeaters are not being used, the address field is encoded as shown in Fig. 2. The destination address is the call sign and SSID of the amateur radio station to which the frame is addressed, while the source address contains the amateur call sign and SSID of the station that sent the frame. These call signs are the call signs of the two ends of a level 2 AX.25 link only.

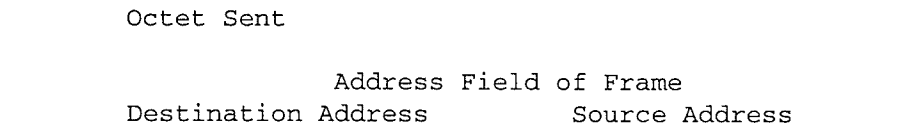


Fig. 2 -- Nonrepeater Address-Field Encoding

A1 through A14 are the fourteen octets that make up the two address subfields of the address field. The destination subaddress is seven octets long (A1 thru A7), and is sent first. This address sequence provides the receivers of frames time to check the destination address subfield to see if the frame is addressed to them while the rest of the frame is being received. The source address subfield is then sent in octets A8 through A14. Both of these subfields are encoded in the same manner, except that the last octet of the address field has the HDLC address extension bit set.

There is an octet at the end of each address subfield that contains the Secondary Station Identifier (SSID). The SSID subfield allows an Amateur Radio operator to have more than one packet-radio station operating under the same call sign. This is useful when an amateur wants to put up a repeater in addition to a regular station, for example. The C bits (see 2.4.1.2, below) and H bit (see 2.2.13.2, below) are also contained in this octet, along with two bits which are reserved for future use.

Fig. 3A shows a typical AX.25 frame in the nonrepeater mode of operation.

Octet	ASCII	Bin.Data	Hex Data
Flag		01111110	7E
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	O	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01100100	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100001	61
Control	I	00111110	3E
PID	none	11110000	F0
FCS	part 1	XXXXXXXX	HH
FCS	part 2	XXXXXXXX	HH
Flag		01111110	7E
Bit position		76543210	

Fig. 3A -- Nonrepeater AX.25 frame

The frame shown is an I frame, not going through a level 2 repeater, from WB4JFI (SSID=0) to K8MMO (SSID=0), with no level 3 protocol. The P/F bit is set; the receive sequence number (N(R)) is 1; the send sequence number (N(S)) is 7.

2.2.13.1.1 Destination Subfield Encoding

Fig. 3 shows how an amateur call sign is placed in the destination address subfield, occupying octets A1 thru A7.

Octet	ASCII	Bin.Data	Hex Data
A1	W	10101110	AE
A2	B	10000100	84
A3	4	01101000	68
A4	J	10010100	94
A5	F	10001100	8C
A6	I	10010010	92
A7	SSID	CRRSSID0	
Bit Position-->		76543210	

Fig. 3 -- Destination Field Encoding

Where:

1. The top octet (A1) is the first octet sent, with bit

0 of each octet being the first bit sent, and bit 7 being the last bit sent.

2. The first (low-order or bit 0) bit of each octet is the HDLC address extension bit, which is set to zero on all but the last octet in the address field, where it is set to one.

3. The bits marked "r" are reserved bits. They may be used in an agreed-upon manner in individual networks. When not implemented, they should be set to one.

4. The bit marked "C" is used as the command/response bit of an AX.25 frame, as outlined in 2.4.1.2 below.

5. The characters of the call sign should be standard seven-bit ASCII (upper case only) placed in the leftmost seven bits of the octet to make room for the address extension bit. If the call sign contains fewer than six characters, it should be padded with ASCII spaces between the last call sign character and the SSID octet.

6. The 0000 SSID is reserved for the first personal AX.25 station. This establishes one standard SSID for "normal" stations to use for the first station.

2.2.13.2 Level 2 Repeater-Address Encoding

If a frame is to go through level 2 amateur packet repeater(s), there is an additional address subfield appended to the end of the address field. This additional subfield contains the call sign(s) of the repeater(s) to be used. This allows more than one repeater to share the same RF channel. If this subfield exists, the last octet of the source subfield has its address extension bit set to zero, indicating that more address-field data follows. The repeater-address subfield is encoded in the same manner as the destination and source address subfields, except for the most-significant bit in the last octet, called the "H" bit. The H bit is used to indicate whether a frame has been repeated or not.

In order to provide some method of indicating when a frame has been repeated, the H bit is set to zero on frames going to a repeater. The repeater will set the H bit to one when the frame is retransmitted. Stations should monitor the H bit, and discard any frames going to the repeater (uplink frames), while operating through a repeater. Fig. 4 shows how the repeater-address subfield is encoded. Fig. 4A is an example of a complete frame after being repeated.

Octet	ASCII	Bin.Data	Hex Data
A15	W	10101110	AE
A16	B	10000100	84
A17	4	01101000	68
A18	J	10010100	94
A19	F	10001100	8C
A20	I	10010010	92
A21	SSID	HRRSSID1	
Bit Order	-->	76543210	

Fig. 4 -- Repeater-address encoding

Where:

1. The top octet is the first octet sent, with bit 0 being sent first and bit 7 sent last of each octet.

2. As with the source and destination address subfields discussed above, bit 0 of each octet is the HDLC address extension bit, which is set to zero on all but the last address octet, where it is set to one.

3. The "R" bits are reserved in the same manner as in the source and destination subfields.

4. The "H" bit is the has-been-repeated bit. It is set to zero whenever a frame has not been repeated, and set to one by the repeater when the frame has been repeated.

Octet	ASCII	Bin.Data	Hex Data
Flag		01111110	7E
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	O	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01101000	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100000	60
A15	W	10101110	AE
A16	B	10000100	84
A17	4	01101000	68
A18	J	10010100	94
A19	F	10001100	8C
A20	I	10010010	92

A21	SSID	11100011	E3
Control	I	00111110	3F
PID	none	11110000	F0
FCS	part 1	XXXXXXXX	HH
FCS	part 2	XXXXXXXX	HH
Flag		01111110	7E
Bit position		76543210	

Fig. 4A -- AX.25 frame in repeater mode

The above frame is the same as Fig. 3A, except for the addition of a repeater-address subfield (WB4JFI, SSID=1). The H bit is set, indicating this is from the output of the repeater.

2.2.13.3 Multiple Repeater Operation

The link-layer AX.25 protocol allows operation through more than one repeater, creating a primitive frame routing mechanism. Up to eight repeaters may be used by extending the repeater-address subfield. When there is more than one repeater address, the repeater address immediately following the source address subfield will be considered the address of the first repeater of a multiple-repeater chain. As a frame progresses through a chain of repeaters, each successive repeater will set the H bit (has-been-repeated bit) in its SSID octet, indicating that the frame has been successfully repeated through it. No other changes to the frame are made (except for the necessary recalculation of the FCS). The destination station can determine the route the frame took to each it by examining the address field.

The number of repeater addresses is variable. All but the last repeater address will have the address extension bits of all octets set to zero, as will all but the last octet (SSID octet) of the last repeater address. The last octet of the last repeater address will have the address extension bit set to one, indicating the end of the address field.

It should be noted that various timers (see 2.4.7, below) may have to be adjusted to accommodate the additional delays encountered when a frame must pass through a multiple-repeater chain, and the return acknowledgement must travel through the same path before reaching the source device.

It is anticipated that multiple-repeater operation is a temporary method of interconnecting stations over large distances until such time that a layer 3 protocol is in use. Once this layer 3 protocol becomes operational, repeater chaining should be phased out.

2.3 Elements of Procedure

2.3.1

The elements of procedure are defined in terms of actions that occur on receipt of frames.

2.3.2 Control-Field Formats and State Variables

2.3.2.1 Control-Field Formats

The control field is responsible for identifying the type of frame being sent, and is also used to convey commands and responses from one end of the link to the other in order to maintain proper link control.

The control fields used in AX.25 use the CCITT X.25 control fields for balanced operation (LAPB), with an additional control field taken from ADCCP to allow connectionless and round-table operation.

There are three general types of AX.25 frames. They are the Information frame (I frame), the Supervisory frame (S frame), and the Unnumbered frame (U frame). Fig. 5 shows the basic format of the control field associated with these types of frames.

Control-Field Type	Control-Field Bits 7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	M	M	1	1

Fig. 5 -- Control-field formats

Where:

1. Bit 0 is the first bit sent and bit 7 is the last bit sent of the control field.

2. N(S) is the send sequence number (bit 1 is the LSB).

3. N(R) is the receive sequence number (bit 5 is the LSB).

4. The "S" bits are the supervisory function bits, and their encoding is discussed in 2.3.4.2.

5. The "M" bits are the unnumbered frame modifier bits and their encoding is discussed in 2.3.4.3.

6. The P/F bit is the Poll/Final bit. Its function is described in 2.3.3. The distinction between command and response, and therefore the distinction between P

bit and F bit, is made by addressing rules discussed in 2.4.1.2.

2.3.2.1.1 Information-Transfer Format

All I frames have bit 0 of the control field set to zero. N(S) is the sender's send sequence number (the send sequence number of this frame). N(R) is the sender's receive sequence number (the sequence number of the next expected received frame). These numbers are described in 2.3.2.4. In addition, the P/F bit is to be used as described in 2.4.2.

2.3.2.1.2 Supervisory Format

Supervisory frames are denoted by having bit 0 of the control field set to one, and bit 1 of the control field set to zero. S frames provide supervisory link control such as acknowledging or requesting retransmission of I frames, and link-level window control. Since S frames do not have an information field, the sender's send variable and the receiver's receive variable are not incremented for S frames. In addition, the P/F bit is used as described in 2.4.2.

2.3.2.1.3 Unnumbered Format

Unnumbered frames are distinguished by having both bits 0 and 1 of the control field set to one. U frames are responsible for maintaining additional control over the link beyond what is accomplished with S frames. They are also responsible for establishing and terminating link connections. U frames also allow for the transmission and reception of information outside of the normal flow control. Some U frames may contain information and PID fields. The P/F bit is used as described in 2.4.2.

2.3.2.2 Control-Field Parameters

2.3.2.3 Sequence Numbers

Every AX.25 I frame shall be assigned, modulo 8, a sequential number from 0 to 7. This will allow up to seven outstanding I frames per level 2 connection at a time.

2.3.2.4 Frame Variables and Sequence Numbers

2.3.2.4.1 Send State Variable V(S)

The send state variable is a variable that is internal to the DXE and is never sent. It contains the next sequential number to be assigned to the next transmitted I frame. This variable is updated upon the transmission of each I frame.

2.3.2.4.2 Send Sequence Number N(S)

The send sequence number is found in the control field of all I frames. It contains the sequence number of the I frame being sent. Just prior to the transmission of the I frame, N(S) is updated to equal the send state variable.

2.3.2.4.3 Receive State Variable V(R)

The receive state variable is a variable that is internal to the DXE. It contains the sequence number of the next expected received I frame. This variable is updated upon the reception of an error-free I frame whose send sequence number equals the present received state variable value.

2.3.2.4.4 Received Sequence Number N(R)

The received sequence number is in both I and S frames. Prior to sending an I or S frame, this variable is updated to equal that of the received state variable, thus implicitly acknowledging the proper reception of all frames up to and including N(R)-1.

2.3.3 Functions of Poll/Final (P/F) Bit

The P/F bit is used in all types of frames. It is used in a command (poll) mode to request an immediate reply to a frame. The reply to this poll is indicated by setting the response (final) bit in the appropriate frame. Only one outstanding poll condition per direction is allowed at a time. The procedure for P/F bit operation is described in 2.4.2.

2.3.4 Control Field Coding for Commands and Responses

The following commands and responses, indicated by their control field encoding, are to be use by the DXE:

2.3.4.1 Information Command Frame Control Field

The function of the information (I) command is to transfer across a data link sequentially numbered frames containing an information field.

The information-frame control field is encoded as shown in Fig. 6. These frames are sequentially numbered by the N(S) subfield to maintain control of their passage over the link-layer connection.

Control Field Bits
7 6 5 4 3 2 1 0
N(R) P N(S) 0

Fig. 6 -- I frame control field

2.3.4.2 Supervisory Frame Control Field

The supervisory frame control fields are encoded as shown in Fig. 7.

Control Field Bits		7	6	5	4	3	2	1	0
Receive Ready	RR		N(R)		P/F	0	0	0	1
Receive Not Ready	RNR		N(R)		P/F	0	1	0	1
Reject	REJ		N(R)		P/F	1	0	0	1

Fig. 7 -- S frame control fields

The Frame identifiers:

C or SABM	Layer 2 Connect Request
D or DISC	Layer 2 Disconnect Request
I	Information Frame
RR	Receive Ready. System Ready To Receive
RNR or NR	Receive Not Ready. TNC Buffer Full
RJ or REJ	Reject Frame. Out of Sequence or Duplicate
FRMR	Frame Reject. Fatal Error
UI	Unnumbered Information Frame. "Unproto"
DM	Disconnect Mode. System Busy or Disconnected.

2.3.4.2.1 Receive Ready (RR) Command and Response

Receive Ready is used to do the following:

1. to indicate that the sender of the RR is now able to receive more I frames.

2. to acknowledge properly received I frames up to, and including N(R)-1, and

3. to clear a previously set busy condition created by an RNR command having been sent.

The status of the DXE at the other end of the link can be requested by sending a RR command frame with the P-bit set to one.

2.3.4.2.2 Receive Not Ready (RNR) Command and Response

Receive Not Ready is used to indicate to the sender of I frames that the receiving DXE is temporarily busy and cannot accept any more I frames. Frames up to N(R)-1 are acknowledged. Any I frames numbered N(R) and higher that might have been caught between states and not acknowledged when the RNR command was sent are not acknowledged.

The RNR condition can be cleared by the sending of a UA, RR, REJ, or SABM frame.

The status of the DXE at the other end of the link can be requested by sending a RNR command frame with the P bit set to one.

2.3.4.2.3 Reject (REJ) Command and Response

The reject frame is used to request retransmission of I frames starting with N(R). Any frames that were sent with a sequence number of N(R)-1 or less are acknowledged. Additional I frames may be appended to the retransmission of the N(R) frame if there are any.

Only one reject frame condition is allowed in each direction at a time. The reject condition is cleared by the proper reception of I frames up to the I frame that caused the reject condition to be initiated.

The status of the DXE at the other end of the link can be requested by sending a REJ command frame with the P bit set to one.

2.3.4.3 Unnumbered Frame Control Fields

Unnumbered frame control fields are either commands or responses.

Fig. 8 shows the layout of U frames implemented within this protocol.

Control Field	Type	Control Field Bits							
		7	6	5	4	3	2	1	0
Set Asynchronous Balanced Mode-SABM	Cmd	0	0	1	P	1	1	1	1
Disconnect-DISC	Cmd	0	1	0	P	0	0	1	1
Disconnected Mode-DM	Res	0	0	0	F	1	1	1	1
Unnumbered Acknowledge-UA	Res	0	1	1	F	0	0	1	1
Frame Reject-FRMR	Res	1	0	0	F	0	1	1	1
Unnumbered Information-UI	Either	0	0	0	P/F	0	0	1	1

Fig. 8 -- U frame control fields

2.3.4.3.1 Set Asynchronous Balanced Mode (SABM) Command

The SABM command is used to place 2 DXEs in the asynchronous balanced mode. This is a balanced mode of operation known as LAPB where both devices are treated as equals.

Information fields are not allowed in SABM commands. Any outstanding I frames left when the SABM command is issued will remain unacknowledged.

The DXE confirms reception and acceptance of a SABM command by sending a UA response frame at the earliest opportunity. If the DXE is not capable of accepting a SABM

command, it should respond with a DM frame if possible.

2.3.4.3.2 Disconnect (DISC) Command

The DISC command is used to terminate a link session between two stations. No information field is permitted in a DISC command frame.

Prior to acting on the DISC frame, the receiving DXE confirms acceptance of the DISC by issuing a UA response frame at its earliest opportunity. The DXE sending the DISC enters the disconnected state when it receives the UA response.

Any unacknowledged I frames left when this command is acted upon will remain unacknowledged.

2.3.4.3.3 Frame Reject (FRMR) Response

2.3.4.3.3.1
The FRMR response frame is sent to report that the receiver of a frame cannot successfully process that frame and that the error condition is not correctable by sending the offending frame again. Typically this condition will appear when a frame without an FCS error has been received with one of the following conditions:

1. The reception of an invalid or not implemented command or response frame.

2. The reception of an I frame whose information field exceeds the agreed-upon length. (See 2.4.7.3, below.)

3. The reception of an improper N(R). This usually happens when the N(R) frame has already been sent and acknowledged, or when N(R) is out of sequence with what was expected.

4. The reception of a frame with an information field where one is not allowed, or the reception of a U or S frame whose length is incorrect. Bits W and Y described in 2.3.4.3.3.2 should both be set to one to indicate this condition.

5. The reception of a supervisory frame with the F bit set to one, except during a timer recovery condition (see 2.4.4.9), or except as a reply to a command frame sent with the P bit set to one. Bit W (described in 2.3.4.3.3.2) should be set to one.

6. The reception of an unexpected UA or DM response frame. Bit W should be set to one.

7. The reception of a frame with an invalid N(S). Bit W should be set to one.

An invalid N(R) is defined as one which points to an I

frame that previously has been transmitted and acknowledged, or an I frame which has not been transmitted and is not the next sequential I frame pending transmission.

An invalid N(S) is defined as an N(S) that is equal to the last transmitted N(R)+k and is equal to the received state variable V(R), where k is the maximum number of outstanding information frames as defined in 2.4.7.4 below.

An invalid or not implemented command or response is defined as a frame with a control field that is unknown to the receiver of this frame.

2.3.4.3.3.2

When a FRMR frame is sent, an information field is added to the frame that contains additional information indicating where the problem occurred. This information field is three octets long and is shown in Fig. 9.

Information Field Bits

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Z	Y	X	W	V(R)			C R	V(S)			0	Rejected Frame Control Field							

Fig. 9 -- FRMR frame information field

Where:

1. The rejected frame control field carries the control field of the frame that caused the reject condition. It is in bits 0-7 of the information field.

2. V(S) is the current send state variable of the device reporting the rejection (bit 9 is the low bit).

3. The CR bit is set to zero to indicate the rejected frame was a command, or one if it was a response.

4. V(R) is the current receive state variable of the device reporting rejection (bit 13 is the low bit).

5. If W is set to 1, the control field received was invalid or not implemented.

6. If X is set to 1, the frame that caused the reject condition was considered invalid because it was a U or S frame that had an information field that is not allowed. Bit W must be set to 1 in addition to the X bit.

7. If Y is set to 1, the control field received and returned in bits exceeded the maximum allowed under this recommendation in 2.4.7.3, below.

8. If A is set to 1, the control field received and returned

in bits 1 to 8 contained an invalid N(R).

9. Bits 8, and 20 to 23 are set to 0.

2.3.4.3.4 Unnumbered Acknowledge (UA) Response

The UA response frame is sent to acknowledge the reception and acceptance of a SABM or DISC command frame. A received command is not actually processed until the UA response frame is sent. Information fields are not permitted in a UA frame.

2.3.4.3.5 Disconnected Mode (DM) Response

The disconnected mode response is sent whenever a DXE receives a frame other than a SABM or UI frame while in a disconnected mode. It is also sent to request a set mode command, or to indicate it cannot accept a connection at the moment. The DM response does not have an information field.

Whenever a SABM frame is a received, and it is determined that a connection is not possible, a DM frame shall be sent. This will indicate that the called station cannot accept a connection at that time.

While a DXE is in the disconnected mode, it will respond to any command other than a SABM or UI frame with a DM response with the P/F bit set to 1.

2.3.4.3.6 Unnumbered Information (UI) Frame

The Unnumbered Information frame contains PID and information fields and is used to pass information along the link outside the normal information controls. This allows information fields to go back and forth on the link bypassing flow control. Since these frames are not acknowledgeable, if one gets obliterated, there is no way to recover it. A received UI frame with the P bit set shall cause a response to be transmitted. This response shall be a DM frame when in the disconnected state or a RR (or RNR, if appropriate) frame in the information transfer state.

2.3.5 Link Error Reporting and Recovery

There are several link-layer errors that are recoverable without terminating the connection. These error situations may occur as a result of malfunctions within the DXE, or if transmission errors occur.

2.3.5.1 DXE Busy Condition

When a DXE becomes temporarily unable to receive I frames, such as when receive buffers are full, it will send a Receive Not Ready (RNR) frame. This informs the other DXE that

this DXE cannot handle any more I frames at the moment. This condition is usually cleared by the sending of a UA, RR, REJ, or SABM command frame.

2.3.5.2 Send Sequence Number Error

If the send sequence number, N(S), of an otherwise error-free received frame does not match the receive state variable, V(R), a send sequence error has occurred, and the information field will be discarded. The receiver will not acknowledge this frame, or any other I frames, until N(S) matches V(R).

The control field of the erroneous I frame(s) will be accepted so that link supervisory functions such as checking the P/F bit can still be performed. Because of this updating, the retransmitted I frame may have an updated P bit and N(R).

2.3.5.3 Reject (REJ) Recovery

REJ is used to request a retransmission of I frames following the detection of a N(S) sequence error. Only one outstanding "sent REJ" condition is allowed at a time. This condition is cleared when the requested I frame has been received.

A DXE receiving the REJ command will clear the condition by resending all outstanding I frames (up to the window size), starting with the one indicated in N(R) of the REJ frame.

2.3.5.4 Time-out Error Recovery

2.3.5.4.1 T1 Timer Recovery

If a DXE, due to a transmission error, does not receive (or receives and discards) a single I frame or the last I frame in a sequence of I frames, it will not detect a send-sequence-number error, and therefore will not transmit a REJ. The DXE which transmitted the unacknowledged I frame(s) shall, following the completion of time-out period T1, take appropriate recovery action to determine when I frame retransmission should begin as described in 2.4.4.9, below. This condition is cleared by the reception of an acknowledgement for the sent frame(s), or by the link being reset. See 2.4.6.

2.3.5.4.2 Timer T3 Recovery

Timer T3 is used to assure the link is still functional during periods of low information transfer. Whenever T1 is not running (no outstanding I frames), T3 is used to periodically poll the other DXE of a link. When T3 times out, a RR or RNR frame is transmitted as a command and with the P bit set. The waiting acknowledgement procedure (2.4.4.9, below) is then executed.

2.3.5.5 Invalid Frame or FCS Error

If an invalid frame is received, or a frame is received with an FCS error, that frame will be discarded with no action taken.

2.3.5.6 Frame Rejection Condition

A frame rejection condition occurs when an otherwise error-free frame has been received with one of the conditions listed in 2.3.4.3.3 above.

Once a rejection error occurs, no more I frames are accepted (except for the examination of the P/F bit) until the error is resolved. The error condition is reported to the other DXE by sending a FRMR response frame. See 2.4.5.

2.4 Description of AX.25 Procedures

The following describes the procedures used to setup, use, and disconnect a balanced link between two DXE stations.

2.4.1 Address Field Operation

2.4.1.1 Address Information

All transmitted frames shall have address fields conforming to 2.2.13, above. All frames shall have both the destination device and the source device addresses in the address field, with the destination address coming first. This allows many links to share the same RF channel. The destination address is always the address of the station(s) to receive the frame, while the source address contains the address of the device that sent the frame.

The destination address can be a group name or club call sign if the point-to-multipoint operation is allowed. Operation with destination addresses other than actual amateur call signs is a subject for further study.

2.4.1.2 Command/Response Procedure

AX.25 Version 2.0 has implemented the command/response information in the address field. In order to maintain compatibility with previous versions of AX.25, the command/response information is conveyed using two bits.

An upward-compatible AX.25 DXE can determine whether it is communicating with a DXE using an older version of this protocol by testing the command/response bit information located in bit 7 of the SSID octets of both the destination and source address subfields. If both C bits are set to zero, the device is using the older protocol. The newer version of the protocol always has one of these two bits set to one and the other set to

zero, depending on whether the frame is a command or a response.

The command/response information is encoded into the address field as shown in Fig. 10.

Frame Type	Dest. SSID C-Bit	Source SSID C-Bit
Previous versions	0	0
Command (V.2.0)	1	0
Response (V.2.0)	0	1
Previous versions	1	1

Fig. 10 -- Command/Response encoding

Since all frames are considered either commands or responses, a device shall always have one of the bits set to one, and the other bit set to zero.

The use of the command/response information in AX.25 allows S frames to be either commands or responses. This aids maintenance of proper control over the link during the information transfer state.

2.4.2 P/F Bit Procedures

The next response frame returned by the DXE to a SABM or DISC command with the P bit set to 1 will be a UA or DM response with the F bit set to 1.

The next response frame returned to an I frame with the P bit set to 1, received during the information transfer state, will be a RR, RNR, or REJ response with the F bit set to 1.

The next response frame returned to a supervisory command frame with the P bit set to 1, received during the information transfer state, will be a RR, RNR, or REJ response frame with the F bit set to 1.

The next response frame returned to a S or I command frame with the P bit set to 1, received in the disconnected state, will be a DM response frame with the F bit set to 1.

The P bit is used in conjunction with the time-out recovery condition discussed in 2.3.5.4, above.

When not used, the P/F bit is set to zero.

2.4.3 Procedures For Link Set-Up and Disconnection

2.4.3.1 LAPB Link Connection Establishment

When one DXE wishes to connect to another DXE, it will send a SABM command frame to that device and start timer (T1). If the other DXE is there and able to connect, it will respond

with a UA response frame, and reset both of its internal state variables (V(S) and V(R)). The reception of the UA response frame at the other end will cause the DXE requesting the connection to cancel the T1 timer and set its internal state variables to 0.

If the other DXE doesn't respond before T1 times out, the device requesting the connection will re-send the SABM frame, and start T1 running again. The DXE should continue to try to establish a connection until it has tried unsuccessfully N2 times. N2 is defined in 2.4.7.2, below.

If, upon reception of a SABM command, the DXE decides that it cannot enter the indicated state, it should send a DM frame.

When receiving a DM response, the DXE sending the SABM should cancel its T1 timer, and not enter the information-transfer state.

The DXE sending a SABM command will ignore and discard any frames except SABM, DISC, UA, and DM frames from the other DXE.

Frames other than UA and DM in response to a received SABM will be sent only after the link is set up and if no outstanding SABM exists.

2.4.3.2 Information-Transfer Phase

After establishing a link connection, the DXE will enter the information-transfer state. In this state, the DXE will accept and transmit I and S frames according to the procedure outlined in 2.4.4, below.

When receiving a SABM command while in the information-transfer state, the DXE will follow the resetting procedure outlined in 2.4.6 below.

2.4.3.3 Link Disconnection

2.4.3.3.1
While in the information-transfer state, either DXE may indicate a request to disconnect the link by transmitting a DISC command frame and starting timer T1 (see 2.4.7).

2.4.3.3.2
A DXE, upon receiving a valid DISC command, shall send a UA response frame and enter the disconnected state. A DXE, upon receiving a UA or DM response to a sent DISC command, shall cancel timer T1, and enter the disconnected state.

2.4.3.3.3
If a UA or DM response is not correctly received before T1

times out, the DISC frame should be sent again and T1 restarted. If this happens N2 times, the DXE should enter the disconnected state.

2.4.3.4 Disconnected State

2.4.3.4.1

A DXE in the disconnected state shall monitor received commands and react upon the reception of a SABM as described in 2.4.3.1 above and will transmit a DM frame in response to a DISC command.

2.4.3.4.2

In the disconnected state, a DXE may initiate a link set-up as outlined in connection establishment above (2.4.3.1). It may also respond to the reception of a SABM and establish a connection, or it may ignore the SABM and send a DM instead.

2.4.3.4.3

Any DXE receiving a command frame other than a SABM or UI frame with the P bit set to one should respond with a DM frame with the F bit set to one. The offending frame should be ignored.

2.4.3.4.4

When the DXE enters the disconnected state after an error condition or if an internal error has resulted in the DXE being in the disconnected state, the DXE should indicate this by sending a DM response rather than a DISC frame and follow the link disconnection procedure outlined in 2.4.3.3.3, above. The DXE may then try to re-establish the link using the link set-up procedure outlined in 2.4.3.1, above.

2.4.3.5 Collision Recovery

2.4.3.5.1 Collisions in a Half-Duplex Environment

Collisions of frames in a half-duplex environment are taken care of by the retry nature of the T1 timer and retransmission count variable. No other special action needs to be taken.

2.4.3.5.2 Collisions of Unnumbered Commands

If sent and received SABM or DISC command frames are the same, both DXEs should send a UA response at the earliest opportunity, and both devices should enter the indicated state.

If sent and received SABM or DISC commands are different, both DXEs should enter the disconnected state and transmit a DM frame at the earliest opportunity.

2.4.3.5.3 Collision of a DM with a SABM or DISC

When an unsolicited DM response frame is sent, a collision between it and a SABM or DISC may occur. In order to prevent this DM from being misinterpreted, all unsolicited DM frames should be transmitted with the F bit set to zero. All SABM and DISC frames should be sent with the P bit set to one. This will prevent any confusion when a DM frame is received.

2.4.3.6 Connectionless Operation

In Amateur Radio, there is an additional type of operation that is not feasible using level 2 connections. This operation is the round table, where several amateurs may be engaged in one conversation. This type of operation cannot be accommodated by AX.25 link-layer connections.

The way round-table activity is implemented is technically outside the AX.25 connection, but still using the AX.25 frame structure.

AX.25 uses a special frame for this operation, called the Unnumbered Information (UI) frame. When this type of operation is used, the destination address should have a code word installed in it to prevent the users of that particular round table from seeing all frames going through the shared RF medium. An example of this is if a group of amateurs are in a round-table discussion about packet radio, they could put PACKET in the destination address, so they would receive frames only from others in the same discussion. An added advantage of the use of AX.25 in this manner is that the source of each frame is in the source address subfield, so software could be written to automatically display who is making what comments.

Since this mode is connectionless, there will be no requests for retransmissions of bad frames. Collisions will also occur, with the potential of losing the frames that collided.

2.4.4 Procedures for Information Transfer

Once a connection has been established, as outlined above, both devices are able to accept I, S, and U frames.

2.4.4.1 Sending I Frames

Whenever a DXE has an I frame to transmit, it will send the I frame with N(S) of the control field equal to its current send state variable V(S). Once the I frame is sent, the send state variable is incremented by one. If timer T1 is not running, it should be started. If timer T1 is running, it should be restarted.

The DXE should not transmit any more I frames if its send state variable equals the last received N(R) from the other side

of the link plus seven. If it were to send more I frames, the flow control window would be exceed, and errors could result.

If a DXE is in a busy condition, it may still send I frames as long as the other device is not also busy.

If a DXE is in the frame-rejection mode, it will stop sending I frames.

2.4.4.2 Receiving I Frames

2.4.4.2.1

If a DXE receives a valid I frame (one with a correct FCS and whose send sequence number equals the receiver's receive state variable) and is not in the busy condition, it will accept the received I frame, increment its receive state variable, and act in one of the following manners:

1. If it has an I frame to send, that I frame may be sent with the transmitted N(R) equal to its receive state variable V(R) (thus acknowledging the received frame). Alternately, the device may send a RR frame with N(R) equal to V(R), and then send the I frame.

2. If there are no outstanding I frames, the receiving device will send a RR frame with N(R) equal to V(R). The receiving DXE may wait a small period of time before sending the RR frame to be sure additional I frames are not being transmitted.

2.4.4.2.2

If the DXE is in a busy condition, it may ignore any received I frames without reporting this condition other than repeating the indication of the busy condition.

If a busy condition exists, the DXE receiving the busy condition indication should poll the sender of the busy indication periodically until the busy condition disappears.

A DXE may poll the busy DXE periodically with RR or RNR frames with the P bit set to one.

The reception of I frames that contain zero-length information fields shall be reported to the next level but no information field will be transferred.

2.4.4.3 Reception of Out of Sequence Frames

When an I frame is received with a correct FCS, but its send sequence number, N(S), does not match the current receiver's receive state variable, the frame should be discarded. A REJ frame shall be sent with a receive sequence number equal to one higher (modulo 8) than the last correctly received I frame if an uncleared N(S) sequence error condition has not been previously established. The received state variable and poll bit of the

discarded frame should be checked and acted upon, if necessary, before discarding the frame.

2.4.4.4 Reception of Incorrect Frames

When a DXE receives a frame with an incorrect FCS, an invalid frame, or a frame with an improper address, that frame shall be discarded.

2.4.4.5 Receiving Acknowledgement

Whenever an I or S frame is correctly received, even in a busy condition, the N(R) of the received frame should be checked to see if it includes an acknowledgement of outstanding sent I frames. The T1 timer should be cancelled if the received frame actually acknowledges previously unacknowledged frames. If the T1 timer is cancelled and there are still some frames that have been sent that are not acknowledged, T1 should be started again. If the T1 timer runs out before an acknowledgement is received, the device should proceed to the retransmission procedure in 2.4.4.9.

2.4.4.6 Receiving Reject

Upon receiving a REJ frame, the transmitting DXE will set its send state variable to the same value as the REJ frame's received sequence number in the control field. The DXE will then retransmit any I frame(s) outstanding at the next available opportunity conforming to the following:

1. If the DXE is not transmitting at the time, and the channel is open, the device may commence to retransmit the I frame(s) immediately.

2. If the DXE is operating on a full-duplex channel transmitting a UI or S frame when it receives a REJ frame, it may finish sending the UI or S frame and then retransmit the I frame(s).

3. If the DXE is operating in a full-duplex channel transmitting another I frame when it receives a REJ frame, it may abort the I frame it was sending and start retransmission of the requested I frames immediately.

4. The DXE may send just the one I frame outstanding, or it may send more than the one indicated if more I frames followed the first one not acknowledged, provided the total to be sent does not exceed the flow-control window (7 frames).

If the DXE receives a REJ frame with the poll bit set, it should respond with either a RR or RNR frame with the final bit set before retransmitting the outstanding I frame(s).

2.4.4.7 Receiving a RNR Frame

Whenever a DXE receives a RNR frame, it shall stop transmission of I frames until the busy condition has been cleared. If timer T1 runs out after the RNR was received, the waiting acknowledgement procedure listed in 2.4.4.9, below, should be performed. The poll bit may be used in conjunction with S frames to test for a change in the condition of the busied-out DXE.

2.4.4.8 Sending a Busy Indication

Whenever a DXE enters a busy condition, it will indicate this by sending a RNR response at the next opportunity. While the DXE is in the busy condition, it may receive and process S frames, and if a received S frame has the P bit set to one, the DXE should send a RNR frame with the F bit set to one at the next possible opportunity. To clear the busy condition, the DXE should send either a RR or REJ frame with the received sequence number equal to the current receive state variable, depending on whether the last received I frame was properly received or not.

2.4.4.9 Waiting Acknowledgement

If timer T1 runs out waiting the acknowledgement from the other DXE for an I frame transmitted, the DXE will restart timer T1 and transmit an appropriate supervisory command frame (RR or RNR) with the P bit set. If the DXE receives correctly a supervisory response frame with the F bit set and with an N(R) within the range from the last N(R) received to the last N(S) sent plus one, the DXE will restart timer T1 and will set its send state variable V(S) to the received N(R). It may then resume with I frame transmission or retransmission, as appropriate. If, on the other hand, the DXE receives correctly a supervisory response frame with the F bit not set, or an I frame or supervisory command frame, and with an N(R) within the range from the last N(R) received to the last N(S) sent plus one, the DXE will not restart timer T1, but will use the received N(R) as an indication of acknowledgement of transmitted I frames up to and including I frame numbered N(R)-1.

If timer T1 runs out before a supervisory response frame with the F bit set is received, the DXE will retransmit an appropriate supervisory command frame (RR or RNR) with the P bit set. After N2 attempts to get a supervisory response frame with the F bit set from the other DXE, the DXE will initiate a link resetting procedure as described in 2.4.6, below.

2.4.5 Frame Rejection Conditions

A DXE shall initiate the frame-reset procedure when a frame is received with the correct FCS and address field during the information-transfer state with one or more of the conditions in 2.3.4.3.3, above.

Under these conditions, the DXE will ask the other DXE to reset the link by transmitting a FRMR response as outlined in 2.4.6.3, below.

After sending the FRMR frame, the sending DXE will enter the frame reject condition. This condition is cleared when the DXE that sent the FRMR frame receives a SABM or DISC command, or a DM response frame. Any other command received while the DXE is in the frame reject state will cause another FRMR to be sent out with the same information field as originally sent.

In the frame rejection condition, additional I frames will not be transmitted, and received I frames and S frames will be discarded by the DXE.

The DXE that sent the FRMR frame shall start the T1 timer when the FRMR is sent. If no SABM or DISC frame is received before the timer runs out, the FRMR frame shall be retransmitted, and the T1 timer restarted as described in the waiting acknowledgement section (2.4.4.9) above. If the FRMR is sent N2 times without success, the link shall be reset.

2.4.6 Resetting Procedure

2.4.6.1
The resetting procedure is used to initialize both directions of data flow after a nonrecoverable error has occurred. This resetting procedure is used in the information-transfer state of an AX.25 link only.

2.4.6.2
A DXE shall initiate a reset procedure whenever it receives an unexpected UA response frame or an unsolicited response frame with the F bit set to one. A DXE may also initiate the reset procedure upon receipt of a FRMR frame. Alternatively, the DXE may respond to a FRMR by terminating the connection with a DISC frame.

2.4.6.3
A DXE shall reset the link by sending a SABM frame and starting timer T1. Upon receiving a SABM frame from the DXE previously connected to, the receiver of a SABM frame should send a UA frame back at the earliest opportunity, set its send and receive state variables, V(S) and V(R), to zero and stop T1 unless it has sent a SABM or DISC itself. If the UA is correctly received by the initial DXE, it resets its send and receive state variables, V(S) and V(R), and stops timer T1. Any busy condition that previously existed will also be cleared.

If a DM response is received, the DXE will enter the disconnected state and stop timer T1. If timer T1 runs out before a UA or DM response frame is received, the SABM will be retransmitted and timer T1 restarted. If timer T1 runs out N2

times, the DXE will enter the disconnected state, and any previously existing link conditions will be cleared.

Other commands or responses received by the DXE before completion of the reset procedure will be discarded.

2.4.6.4

One DXE may request that the other DXE reset the link by sending a DM response frame. After the DM frame is sent, the sending DXE will then enter the disconnected state.

2.4.7 List of System Defined Parameters

2.4.7.1 Timers

To maintain the integrity of the AX.25 level 2 connection, use of these timers is recommended.

2.4.7.1.1 Acknowledgement Timer T1

The first timer, T1, is used to make sure a DXE doesn't wait forever for a response to a frame it sends. This timer cannot be expressed in absolute time, since the time required to send frames varies greatly with the signaling rate used at level 1. T1 should take at least twice the amount of time it would take to send maximum length frame to the other DXE, and get the proper response frame back from the other DXE. This would allow time for the other DXE to do some processing before responding.

If level 2 repeaters are to be used, the value of T1 should be adjusted according to the number of repeaters the frame is being transferred through.

2.4.7.1.2 Response Delay Timer T2

The second timer, T2, may be implemented by the DXE to specify a maximum amount of delay to be introduced between the time an I frame is received, and the time the resulting response frame is sent. This delay may be introduced to allow a receiving DXE to wait a short period of time to determine if there is more than one frame being sent to it. If more frames are received, the DXE can acknowledge them at once (up to seven), rather than acknowledge each individual frame. The use of timer T2 is not mandatory, but is recommended to improve channel efficiency. Note that, on full-duplex channels, acknowledgements should not be delayed beyond k/2 frames to achieve maximum throughput. The k parameter is defined in 2.4.7.4, below.

2.4.7.1.3 Inactive Link Timer T3

The third timer, T3, is used whenever T1 isn't running to maintain link integrity. It is recommended that whenever there are no outstanding unacknowledged I frames or P-bit frames (during the information-transfer state), a RR or RNR frame with

the P bit set to one be sent every T3 time units to query the status of the other DXE. The period of T3 is locally defined, and depends greatly on level 1 operation. T3 should be greater than T1, and may be very large on channels of high integrity.

2.4.7.2 Maximum Number of Retries (N2)

The maximum number of retries is used in conjunction with the T1 timer.

2.4.7.3 Maximum Number of Octets in an I Field (N1)

The maximum number of octets allowed in the I field will be 256. There shall also be an integral number of octets.

2.4.7.4 Maximum Number of I Frames Outstanding (k)

The maximum number of outstanding I frames at a time is seven.