A PC System for Data Acquisition and Recursive Parameter Estimation

Bergman, Sten; Persson, Per

1984

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Bergman, S., & Persson, P. (1984). *A PC System for Data Acquisition and Recursive Parameter Estimation*. (Technical Reports TFRT-7275). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
2

CODEN:LUTFD2/(TFRT-7275)/1-104/(1985)

A PC SYSTEM FOR DATA ACQUISITION AND RECURSIVE PARAMETER ESTIMATION

STEN BERGMAN
PER PERSSON

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
MAY 1985

LUND INSTITUTE OF TECHNOLOGY DEPARTMENT OF AUTOMATIC CONTROL Box 118 S 221 00 Lund Sweden	Document name Report
	Date of issue May 1985
	Document number CODEN:LUTFD2/(TFRT-7275)/1-104/(1985)
Author(s) BERGMAN Sten PERSSON Per	Supervisor
	Sponsoring organization

Title and subtitle

A PC SYSTEM FOR DATA ACQUISITION AND RECURSIVE PARAMETER ESTIMATION.

Abstract

The implementation of data acquisition and recursive parameter estimation algorithms in a personal computer (PC) is presented. The goal was to investigate the feasibility of using a cheap PC for BWR monitoring. The experience from program development, tests etc showed that the selected APPLE][+ PC was too slow to be used for real-time application, and by memory limitation only a limited model structure could be allowed. However from tests and modular programming in Pascal gives a good portability to other systems. By remote analysis a powerful VAX 11/780 could be used for estimation on an asynchronous basis. A commandprocedure for computer communication APPLE][/VAX was developed and tested with good success.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title	ISBN

Language English	Number of pages 104	Recipient's notes
Security classification		

DOKUMENTDATABLAD RT 3/81

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

A PC SYSTEM FOR DATA ACQUISITION AND RECURSIVE PARAMETER ESTIMATION

by

Sten Bergman

and

Per Persson

Department of Automatic Control

Lund Institute of Technology

BOX 118

S - 221 00 Lund, SWEDEN

- May 1985 -

TABLE OF CONTENTS:

ABSTRACT

1. INTRODUCTION

2. THE SYSTEM GOAL

3. SYSTEM DESIGN
System specification
Software design
Implementation experiences

4. IDENTIFICATION AND DETECTION

5. REMOTE ANALYSIS
Data communication
Data transmission

6. EXPERIMENTS

7. CONCLUSIONS

8. ACKNOWLEDGEMENTS

9. REFERENCES

APPENDIX 1 PROGRAM-USER INSTRUCTIONS

APPENDIX 2 PROGRAM CODE

ABSTRACT

The implementation of data acquisition and recursive parameter estimation algorithms in a personal computer (PC) is presented. The goal was to investigate the feasibility of using a cheap PC for BWR monitoring. The experience from program development, tests etc showed that the selected APPLE][+ PC was too slow to be used for real-time application, and by memory limitation only a limited model structure could be allowed. However from tests and modular programming in Pascal gives a good portability to other systems. By remote analysis a powerful VAX 11/780 could be used for estimation on an asynchronous basis. A commandprocedure for computer communication APPLE][/VAX was developed and tested with good success.

1 INTRODUCTION

Surveillance of the dynamic behaviour of systems and subsystems has become an important tool for detection of component failures and wear. One reason for this is the increased complexity of industrial systems together with the strict requirements on safe and steady operation. Another is the evolution of the micro-processor technology which have provided tools for efficient and cheap surveillance systems.

The dynamic surveillance problem can be approached in many different ways. See Willsky (1976) for a survey of methods. One approach is to monitor the process dynamics in real-time using recursive identification. The obtained estimates, or functions on them, can be displayed to detect sudden changes in the dynamic behaviour or to give indication when the system dynamics has drifted into an undesired region. See e.g. Hägglund (1983). Another approach which have been extensively applied for surveillance of nuclear reactors has been the detection of spectral changes in the reactor noise. In Gopal and Ciarimitaro (1977), Piety (1977), Piety and Robinson (1976) and Fry (1971) various aspects on algorithms for automatic signature analysis of power spectral density data are given. These applications mainly addresses the surveillance of of PWR reactors, using remote analysis methodology. In Andoh et al (1981), Tamaoki, Kawano and Sato (1982), Ohsawa and Kato (1974) and Andoh et al (1983) experiences from the surveillance of BWR reactors in Japan are presented.

The exterme safety requirements associated with power production has motivated several studies of dynamic surveillance. Different aspects on reactor noise and its relevance to safety are given in e.g. Thie (1976), Mayo and Ziegler (1979) and Currie (1980).

The experiences so far have been concentrated mainly to two different monitoring concepts. One concept related to plant based monitoring (on site) and another related to plant remote monitoring (off site). The latter concept has been applied in Holland, where raw data from the Borsele reactor is transferred more than 200 km to the research center in Petten, where analysis and fault detection are performed. A similar concept have been applied at the Sequoyah Nuclear Plant in the USA where data, (in this case computed spectra) are transferred to the Oak Ridge National Laborotories more than 250 km away from the plant.

While most experiences concerns spectral estimations on reactor noise, some applications have also been reported addressing autoregressive identification techniques. See e.g. Chow and Wu (1979) and Wu and Ouyang (1982).

In a feasibility study the main circulation process dynamics and its control were investigated, using recursive identification and autoregressive moving average structures. See e.g. Bergman and Gustafsson (1979). The goal of this study was to get a better knowledge of the process, to investigate if normal noise is sufficient for identification, compared to PRBS experiments, and to test various parametric model-structures for identification.

2 THE SYSTEM GOAL

Based on the experiences from reactor noise analysis from the Barsebäck plants, and application of recursive estimation methods, a research project was initiated in 1982, with the following goals:

1 Development of suitable methods for Boiling Water Reactor surveillance, identification and fault detection.
2 Implementation of a small test system in a Personal Computer (PC) environment.
3 Experimental validation.

The first goal addresses the problem of choosing an apropriate method for surveillance of a complex plant, like a nuclear reactor. While most surveillance systems so far, have been based on spectral analysis, using Fourier transform techniques (FFT), this system will address parametric identification methodology. That is, to fit parameters in a given linear input-output structure of the process. By letting the system discount old data the process may also be slowly timevarying. Compared to spectral estimation methods, using FFT, recursive estimation methods, in real time, should have some advantages, especially in computational simplicity. The surveillance of a BWR cat e.g. be made by stability monitoring. This requires measurement of transfer functions. In the following, BWR surveillance is supposed to involve model identification, related to the transfer function dynamics between the reactor dome pressure and the average neutron flux.

The second goal was to implement the identification routines or the interface between data acqusition and existing identification packages, like e.g. the IDPAC and RECID packages. The motivation for using a micro-processor front-end, was mainly because of other application areas than nuclear reactor noise analysis. Similar surveillance problems may be found in hydro-power plants, wind power plants etc.

The third goal was seen as essential for obtaining information on "real-life" problems. The experiments should concern both detection and surveillance as well as the implementation aspects, system structuring, software design etc.

3 SYSTEM DESIGN

In this section various aspects on the system design will be discussed. First a brief introduction to the system specification will be given, then different aspects on hardware requirements and software design will be discussed.

3.1 System specification

The noise monitoring system was based on the following general specification:

1. The monitor should be considered as an information system, working in parallell with other control, measurement and protection systems in the plant.
2. The main task is to track the process dynamics to estimate the inherent reactor stability and to detect anomalies.
3. It should be a cost-effective system in the sense that it should utilize high-level software languages and possibly existing packages for identification.
4. In the prototype phase, operator communication facilities could be kept to a minimum.
5. Possibility to integrate the system in an advanced diagnosis system should be foreseen in the design.

Existing prototype systems for reactor noise surveillance have mainly been based on moderate sized mini-computers like PDP, TOSBAC, NORD etc. equipped with A/D converters, 30-128 kbyte of primary memory, disc based operating systems, graphical devices for output etc. The prices for these surveillance systems varies typically between 40 000 and 80 000 $. The monitor in this concept study should if possible not exceed more than 10 000 $. A natural choice therefore fell on application of some existing and well proven PC-system. An APPLE II+ was selected for this prototype, mainly because of the following four reasons.

1. Good familarity with the APPLE existed.
2. Good support of peripherials existed.
3. High-level language support (PASCAL,FORTRAN etc) existed.
4. Low cost.

However, as the APPLE II+ is designed with a 8 bit 6502 central processing unit, running at only 1 MHz, problems were expected to show up in both addressing and speed. During the system development other systems have shown up in the market, but also accelerators for the APPLE exists today. The structure of the system is shown in Fig 3.1.

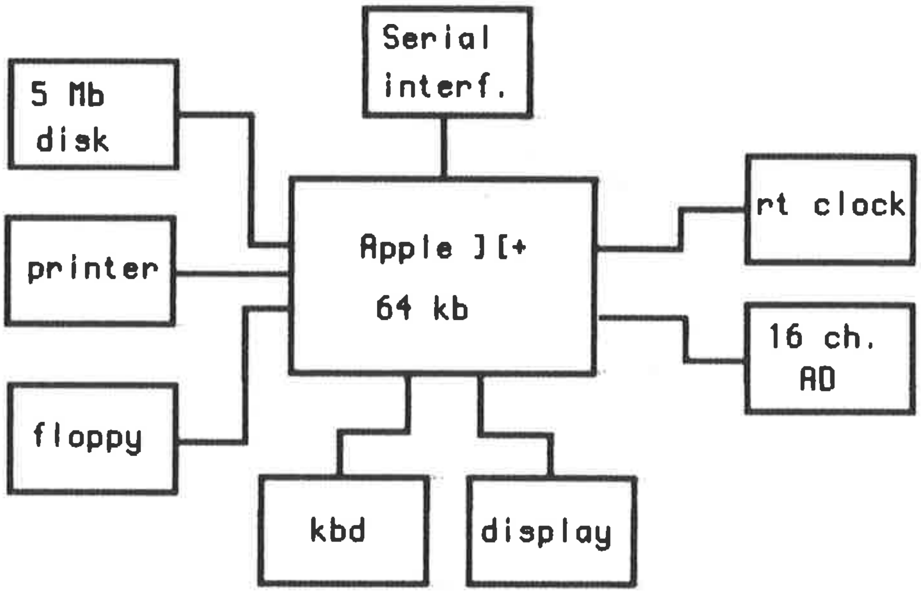

Fig 3.1 Hardware structure.

The system consists of the following parts: An APPLE II+ with 64 kbytes of memory, a 5 Megabyte Winchester disc, a 5 1/4" floppy disc, a real-time clock, 16 channels of 12 bit A/D converter, a serial communication card a printer and a CRT monitor.

3.2 Software design

A number of programs to take care of the data acquisition, parameter identification, data presentation and communication with the host computer (VAX 11/780) were developed using APPLE-PASCAL.

The sampling program (SAMPLE) can be run in two modes. One slow and one fast mode. In the slow mode the channels to be collected and the associated gaincodes can be chosen arbitrarily. The sampling throughput in this mode can be approximnately described by the minimum sampling interval T_s given by $T_s = 11 \cdot N_c + 32$ ms. A maximum sampling frequency of 5 Hz can thus be achived with 16 input channels. The fast mode simply scans the first N_c channels and assumes equal gain settings. The minimum sampling interval in this mode can be given by $T_s = 9 \cdot N_c + 4$ ms.

The identification program (IDENT) performs recursive identification according to the structure given in chapter 4.1. However, for a structure containing A,B,C,D and F polynomials, model orders exceeding 2 could not be estimated due to memory limitations in the APPLE. For an A,B,C structure model orders up to 3 could be handled. During the identification parameters are plotted on the CRT using graphic commands. Parameters are also logged on a parameter file on the disc, for later retrevial and analysis.

A number of utilities have also been developed, such as a parameter display program, a statistical program for parameter value analysis and a parameter calculation program including e.g. fault detection etc. A simple communication program for remote analysis at the VAX computer was also developed. This program (VAXCOM) allows the operator to use the APPLE as a simple terminal. The program accepts all simple VAX/VMS-commands as DIRECTORY, PRINT, SET DEFAULT etc. The program uses the $-sign from the VAX and interprets it as a VMS-prompt. Using the VAXCOM software,a small program for backup purpose was also developed (BACKUP). Another program also using VAXCOM is the VAX software which converts a "file-of-real" into ASCII characters and creates a VAX textfile.

In order to test the identification routine a program DATAGEN was also developed. This program simulates a system given on the form:

$$A(q^{-1})\ y(t) = B(q^{-1})\ u(t) + e(t) \tag{3.1}$$

where u(t) is a PRBS signal and e(t) is white noise. The program prompts the user for A-, and B-parameters and stores a {u(t),y(t)} sequence.

3.3 Implemementation experience

The APPLE-computer which has been used in this project has in many ways proved to be good. Its operating system and editor are efficient and easy to use. The APPLE graphics was sufficient for the presentation of data and calculated parameters. Especially when the computer is used together with a Winchester-disc it becomes very easy to handle. The CORVUS Winchester-disc of the system has been a problem. A couple of disc crashs with loss of some programs have delayed the work several weeks. After a disc crash the disc may be damaged in some places. The operating system can not cut a file in pieces and write it on the empty places on the disc, but must have a free area on the disc, with a size at least as big as the file. This means that when the disc has been damaged a couple of times and there are a number of "bad blocks" on the disc there may not be room enough to write a big file, although there is plenty of free room spread out on the disc. This is a great disadvantage of the operating system. Later (in 1984) it was found that the disc crashes probably was due to a bad internal grounding.

However the greatest disadvantage of the APPLE-computer is that it is slow. A identification of a file with 500 datapoints and a second order model takes about 15 minutes. This makes it impossible to use it for on-line identification, except for very slow systems. The program memory was not sufficently large. It was impossible to carry out identification with a model-order greater than 3. The realtime-clock is another problem. To decide the sampling period you must measure the time before and after the sampling sequence, compute the time difference and divide by the number of samples. The clock-card does not give the possibility of having a "wait-time"-function, i.e. a function which makes the program wait a specified time. This makes it necessary to have a delay-loop in the program to make it possible to change the sampling period.

4 IDENTIFICATION AND DETECTION

The test system was based on BWR stability properties reflected by the transfer function characteristics between reactor dome pressure and the average neutron flux density. Off-line estimation of parametric models would require an implementation of a minimization algorithm. However, using a recursive method, on-line processing of every new sample is possible. Given a number of pairs, {u(t),y(t)) t=1..N }, we recursively estimate a model represented by one of the following structures

$$A(q^{-1})\ y(k) = B(q^{-1})\ u(k-Td) + C(q^{-1})\ e(k) \tag{4.1}$$

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})}\ u(k - Td) + \frac{C(q^{-1})}{D(q^{-1})}\ e(k) \tag{4.2}$$

$$(1 + A(q^{-1}))\ y(k) = \frac{B(q^{-1})}{1 + F(q^{-1})}\ u(k - Td) + \frac{1 + C(q^{-1})}{1 + D(q^{-1})} \tag{4.3}$$

Introducing the parameter- and the measurement-vectors as

$$\varphi(t) = \begin{bmatrix} -y(t-1) \\ \cdot \\ \cdot \\ -y(t-n_a) \\ u(t-1) \\ \cdot \\ \cdot \\ \cdot \\ u(t-n_b) \\ e(t-1) \\ \cdot \\ \cdot \\ e(t-n_c) \end{bmatrix} \quad (4.4) \qquad \Theta(t) = \begin{bmatrix} a_1 \\ \cdot \\ \cdot \\ a_{na} \\ b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_{nb} \\ c_1 \\ \cdot \\ \cdot \\ c_{nc} \end{bmatrix} \quad (4.5)$$

The system model can now be represented by the simple relation:

$$y(t) = \Theta(t)^T \varphi(t) + e(t) \tag{4.6}$$

The covariance matrix P(t) is further defined by the relation

$$P(t) = \left[P(t-1) - \frac{P(t-1)\varphi(t-1)\varphi^T(t-1)P(t-1)}{\lambda(t) + \varphi^T(t)\ P(t-1)\ \varphi(t)}\right] / \lambda(t) \tag{4.7}$$

and the parameter update is given by

$$e(t) = y(t) - \Theta^T(t-1)\varphi(t) \tag{4.8}$$

$$\Theta(t) = \Theta(t-1) + P(t)\varphi(t)e(t) \tag{4.9}$$

Finally, $\lambda(t)$ is given by

$$\lambda(t) = \lambda_0 \cdot \lambda(t-1) + (1 - \lambda_0) \tag{4.10}$$

However, it is the computation of the matrix P which is the most time consuming part of the algorithm. The multiplication of two NxN matrixes requires N^3 multiplications and N^3 additions. Using the knowledge that P is symmetrical (i.e. $P=P^T$), we can write the ultiplication as follows

$$P\ \varphi\ \varphi^T P = (\varphi^T P^T)^T\ \varphi^T P = (\varphi^T P)^T \varphi^T P \tag{4.11}$$

We now multiply $\varphi^T P$, which requires N^2 multiplications and additions and obtain an Nx1 matrix, this matrix is transposed and multiplied by itself. This will give a symmetrical matrix so we just have to compute the superdiagonal part of the matrix which requires 0.5*N*(N-1) multiplications. To compute this matrix we now need $1.5*N^2+0.5*N$ multiplications and N^2 additions instead of N^3 operations of each. In Table 1 the number of different operations necessary to carry out the identfication algorithm are shown. N is the total number of parameters estimated.

Computational part	Operation			
	+	-	*	/
ε	0	N	N	N
P	N^2+2N+1	$0.5(N^2-N)$	$2.5N^2+0.5N$	N^2-N
λ	1	1	0	1
Θ	N	0	N	0
Relative weight	1	1	2	3

Table 4.1 Number of operations in the identification algorithm.

The relative weight means that in APPLE-Pascal a subtraction takes as long time as an addition, a multiplication takes twice and a division three times as long time as an addition. The total amount of operations, in equivalent additions, for one recursion in the identification is given by $N_{op} = 10.5 \cdot N^2 + 4.5 \cdot N - 1$

The absolute executiontimes for the four arithmetic operations in different languages and in different machines are presented in the following table. The executiontime also depends on the value of the operands.

Operation	Apple][+		VAX 11/780	
	Pascal	Basic	Pascal	FORTRAN
Addition	1.4 ms	48 ms	1.4 µs	0.6 µs
Multiplication	2.8 ms	45 ms	1.4 µs	0.6 µs
Subtraction	1.4 ms	40 ms	1.0 µs	0.6 µs
Division	4.2 ms	50 ms	1.0 µs	0.6 µs

Table 4.2 The executiontime of arithmetical operations on different computers.

The recursive algorithm has been implemented in both APPLE-Pascal and VAX-11 Pascal. Execution tests have been made with the following results for an ABC-model.

Time/recursion	C o m p u t e r	
Model order	APPLE][+	VAX 11/780
1	1.3 s	56 ms
2	2.5 s	66 ms
3	4.5 s	95 ms

Table 4.3 The time for one recursion of the identification algorithm, on different computers.

If the A,B,C,D,F model is used for the identification the APPLE can handle at most a second order model. This is of course a severe restriction.

5 REMOTE ANALYSIS

5.1 Data communication

By a modem the APPLE computer is connnected to the public telephone network. This permits data to be sent to other computers for efficient computation, remote supervision etc. As one of the drawbacks of the Apple was the computational speed, and limited memory capacity, the VAX 11/780 computer of the Department of Automatic control, at Lund Institute of Technology was chosen for the computational work. Existing packages for identification like IDPAC and RECID could then be used. Facilities for communication and remote execution of preselected macros was therefore implemented in the APPLE-programs.

The datacommunication consists of the following sequence, fist the signals are sampled with the program SAMPLE, and the values of the measurements are stored on a file on the CORVUS-disc. Then a VAX -textfile is created from the APPLE, with the VAX/VMS command CREATE. The values which were stored on the APPLE file are then converted to ASCII characters, which then are transmitted to the VAX-computer.

5.2 Data transformations

If we want to process the data further, we can execute a DCL commandprocedure from the APPLE. This procedure can call programs on the VAX, e.g. IDPAC, which can perform advanced signal processing. An example of such a procedure is given in Appendix 1.

For this prototype system, some examples on how to transform the raw data into specific parametric information will be given. In the first step the data quality is checked. This is made by blocking the data in windows, and computing the meanvalues, variances and maximum and minimum values for each window. If they are consistent the data will be accepted for further processing. In the next step Fourier spectra are computed for the input signals. In the third step Auto-Regressive identification is performed. In step four recursive estimation is performed using ARMAX structures.

6 EXPERIMENTS

In order to test the data acqusition and identification algorithm of the APPLE, a simple experiment was performed using an analog simulator. A second order system, given by the transfer function:

$$G(s) = \frac{1}{s^2 + p\,s + q} \tag{6.1}$$

was simulated. White noise, low-pass filtered by a first order system was used as input signal. The sampling rate was 10 Hz. The identification algorithm assumed a second order ABC-structure. In Table 6.1 the true and identified system parameters are shown.

True Parameters				Estimated Parameters			
Continuous System		Discrete System		Continuous System		Discrete System	
p	q	a_1	a_2	$\hat{p}$	$\hat{q}$	$\hat{a}_1$	$\hat{a}_2$
2	1	-1.626	0.661	2.80	0.87	-1.648	0.686
4	3	-1.350	0.436	1.40	1.20	-1.512	0.578
2	2	-1.591	0.661	1.86	1.00	-1.589	0.657
2	3	-1.557	0.661	1.60	1.00	-1.612	0.714
4	9	-1.183	0.437	0.69	1.60	-1.362	0.564
3	2	-1.474	0.537	1.50	1.20	-1.524	0.586

Table 6.1 Result of parameter estimation from analog simulator tests.

A comparision of the true and estimated parameters show a big discrepancy. The reason is not clear. However one explanation mat be that the input signal was not constant between the samplings. Another that the output was too corrupted by noise or that the A/D converter was bad. For estimation of dynamical changes, absolute parameter accuracy is of minor interest, especilally if parameter vectors are stored by learning procedures.

7 CONCLUSIONS

A reactor noise monitor consisting of a PC-system (APPLE) for data acquisition and a VAX-host computer for parameter estimation has been tested. Recursive estimation algorithms and a new fault detector was implemented in the PC-system (APPLE][+). However due to memory limitation only a third order model could be run in the PC. The computational speed was also found to be too low for on-line purposes for the PC. By remote connection the PC system can collect data, store them on disc files, establish communication with a VAX host computer, transfer data, and execute a command procedure in the VAX. Validation against analog simulator test, showed that the parameter estimation quality was poor. The reason is not clear, but one explanation could be that the input signal varied between the samplings.

By programming in Pascal the entire system is rather portable to other computers. Since 1982 many PC systems have been developed and are available. So are also efficient number crunchers in form of array processors (SKY, Systolic, Marinco etc). The capacity of these systems are far from what can be achieved by an APPLE. Still the price is not too frightening. A powerful PC system with an array processor, A/D converters etc could probably be achieved for a cost around 15000-20000 $, which still is about half the cost of existing systems.

8 ACKNOWLEDGEMENTS

Sydkraft Research Foundation is gratefully acknowledged for giving financial support for this project.

9 REFERENCES

Fry D (1971): "Experience in Reactor Malfunction Diagnosis Using On-line Analysis" Nucl. Techn., 10

Izumi and Iida (1973): "Application of On-line Digital Noise Analysis to Reactor Diagnosis in JMTR." J. Nucl. Sci. Technology, 10,4

Oksawa, Kato and Oyamada (1974): "Experiments of Anomaly Detection System of a Reactor Core" Nucl. Techn., 23

Tou and Gonzalez (1974): "Principles of Automatic Pattern Recognition" Addison-Wesley Publ. Co. Inc, Reading Massachusetts.

Kato et al (1976): "A New Monitoring Method for BWR Plant Equipment" Trans. Am. Nucl. Soc., 23

Piety and Robinson (1976): "An On-line Reactor Surveillance Algorithm Based on Multivariable Analysis of Noise" Nucl. Sci. Eng., 59

Willsky (1976): "A survey of Design Methods of Failure Detection in Dynamic Systems." Automatica 12

Gopal and Cirarimitaro (1977): "Experiences with Diagnostic Instrumentation in Nuclear Power Plants" Progr. Nucl. Energy, Vol 1

Piety K.R. (1977): "Statistical Algorithm for Automated Signature Analysis of Power Spectral Density Data." Progr. Nucl. Energy, Vol 1

Cow, Wu and Cain (1977): "Malfunction Detection of Nuclear Reactor by Dynamic Data System Methodology" Progr. Nucl. Energy, Vol 1.

Mayo C (1979): "Post Accident Reactor Diagnostics of TMI-2", paper presented at 12th Informal meeting on Reactor Noise, Studsvik, May 16-18, 1979.

Ziegler G (1980): "Post Accident Reactor Assesment by dynamic Measurements", Proc. ANS/ENS, Topl. Mtg Thermal Reactor Safety, Knoxville, TN, April 6-9, 1980.

Andoh et al (1981): "Operating Experience of a BWR Plant Diagnosis System" Prog. Nucl. Energy. No

Tamoki, Kawano and Sato (1982): "On-line Diagnosis Algorithm Based on Noise Analysis" Progr. Nucl. Eng. No. 9, 1982

Thie J (1982): "Power Reactor Noise", ANS Monograph.

Wu S. and Ouyang M. (1982): "A new approach to reactor noise analysis by the dynamic data system (DDS) methodology", Progr. in nucl. energy, Vol. 9, 1982.

Andoh et al (1983): "Development of BWR Plant Diagnosis System Using Noise Analysis" J. Nucl. Sci. and Techn. Vol 20, No 9.

Hägglund T (1983): "New Estimation Techniques for Adaptive Control" LUTFD2/TFRT-1025/

APPENDIX 1
USER INSTRUCTIONS FOR DATA ACQUISITION PROGRAMS

10.1 General remarks

These instructions describe some programs which were written for data acquisition and analysis. When the programs have been run the computersystem has had the following configuration. The computer was an Apple][+ with a Corvus Winchester disk in slot 6, an Epson Printer in slot 1, a 16 channel A/D converter in slot 5, a 300 baud communicationcard slot 2 and a clock-card in slot 3. The programs were written to be run with this configuration.

The reader of these instructions is assumed to have some familiarity with Apple][Pascal. He must know how to execute programs, how to make filespecifications etc. All data and parameterfiles are of the Pascal file type "file of real".
The programs SAMPLE, VAX, VAXCOM and BACKUP should be used together with a VAX computer. To be able to communicate between the Apple and the VAX you must login on the VAX with another terminal and give the VAX/VMS command

```
$ SET TERMINAL/SPEED=300/NOPARITY.
```

Then plug in the Apple in the communication card and run the desired program. It is recommended to run the programs with a slow communication card (300 baud) because the data transmission protocol is very simple and we must be sure that the VAX can take care of the incoming data without any problems.

10.2 Programs for data acqusition

DATAGEN

The program DATAGEN generates datafiles containing input and output signals from a discrete transfer function.

The program starts by asking for output-file name. Give the name without extension. The program then asks for the modelorder, the "noiselevel" n, the number of samples to be stored and A and B parameters in

$$A(q)\ y(k) = B(q)\ u(k) + n\ e(k).$$

The input signal is a PRBS signal generated by a shift register and the signal e consists of randomnumbers. Data is then stored on a file with the format:

```
NUMBER_OF_SAMPLES    ( = NOS)
NUMBER_OF_CHANNELS   ( = NOS, default = 2)
MODELORDER
CAHNNELNUMBER_1      (default = 0)
GAINCODE             (default = 4)
GAINCODE             (default = 4)
CHANNELNUMBER_2      (default = 1)
GAINCODE             (default = 4)
GAINCODE             (default = 4)
U(1)
Y(1)
.
.
U(NOS)
Y(NOS)
```

This program is mostly used to test other programs and identification algorithms.

SAMPLE

The program SAMPLE samples data and stores the values on a file. The program can collect data from 16 channels and store the data on files ready to be used by IDENT. The program can also transfer files to a VAX and can start a batchjob to take care of the transferred datafiles with more advanced programs e.g. IDPAC. The name of the VAX commandprocedure which does this is IDP.COM. (A little more about how to use this possibility can be found in section 10.3)

The program first asks for fast or slow scan. In fast scan the N first channels are scanned with the same gaincode, in slow scan different channels are allowed to be picked out and be scanned with different gaincodes.
Depending on which voltage range the signals which should be sampled lie in, different "gaincodes" must be chosen in this program. This table shows how to chose gaincode when the voltage interval is known. When in doubt chose gaincode 4, which corresponds to the largest voltage interval.

GAINCODE	VOLTAGE RANGE
0	0 .. 5 V
1	0 .. 1 V
2	0 .. 0.5 V
3	0 .. 0.1 V
4	-5 .. 5 V
5	-1 .. 1 V
6	-0.5 .. 0.5 V
7	-0.1 .. 0.1 V

Table 10.1 Gaincodes.

After the answer of this question the user has to fill in a table with two-letter commands.These commands are

```
AF 'applefile'          ! Outputfile on the Apple
                          given without extension
VF 'vaxfile'            ! Name of VAX-file without
                          extension if transmission
                          is to be done, not necessary
                          to specify if TR N is given
NS 'number_of_samples'
DE 'delay'              ! To control the sampling-rate
NF 'number_of_files'    ! The number of files which you want
                          to create
PR 'y' or 'n'           ! If you want a printed log of
                          the sampling
CI 'minutes'            ! If (T mod minutes)=0 the sampling
                          is started
                          (T is the time in minutes
                          since midnight)
ID 'y' or 'n'           ! If you want to start a batch-job
                          on the VAX, not necessary to
                          specify if TR N is given.
TR 'y' or 'n'           ! If you want to transmit files
                          to the VAX.
```

If you have chosen fast scan mode you must fill in

```
NC 'number_of_channels' ! The number of channels you want
                          to sample
GC 'gaincode'           ! Gaincode for the channels you want
                          to sample.
```

If you have chosen slow scan mode you must fill in

```
IC 'channelnumber' 'gaincode'
```

for the channels you want to sample. If you want to delete one entry write

```
IC 'channelnumber + 100'
```

The command EX starts the execution. Unfortunately no default values are assumed so you must fill in the whole table. After having sampled the values of one file you will get information of the sampling period, written on the screen and on the printer. Be sure that the printer is connected when you use the program! The sampling period can be altered by changing the delay parameter. The minimum sampling time in slowscan mode is $T_s = 11 \cdot N_c + 32$ ms, and in fastscan mode $T_s = 9 \cdot N_c + 4$ ms, where N_c is the number of channels.

The samples are stored on a file of real numbers with the format

```
NUMBER_OF_SAMPLES  (=NOS)
NUMBER_OF_CHANNELS (=NOC)
MODELORDER
CAHNNELNUMBER_1
GAINCODE
GAINCODE
.
.
CHANNELNUMBER_NOC
GAINCODE
GAINCODE
Y_1(1)
.
.
Y_NOC(1)
.
.
.
Y_1(NOS)
.
.
Y_NOC(NOS)
```

The Apple and VAX files will then be 'applefile'1.data .. 'applefile'number_of_files.data and 'vaxfile'1.t .. 'vaxfile'number_of_files.t. The files which are created on the VAX will be ASCII files with one column. The parameter

MODELORDER will always be set to 2 in this program.

10.3 Data displaying programs

DISP

The program DISP displays a datafile graphically on the screen.

The program first asks for the file you want to display. Write the filename, without extension. Then the program writes out the channels which are available in the file, and asks for which channel you want to display. The first 250 samples are now shown. Strike 'return' and the program prompts with '>'. Now you can give 4 commands

```
H       ! Print out help text
C       ! Continue, display the next 250 samples
E       ! Exit from the program
N       ! If you want to look at a new channel or a
          new file
```

If you print 'N' you get back to the mainprogram, which will ask for a new file to be displayed, if you answer '$' the last file used will be assumed. This is convenient when you want to look at another channel of the file you are displaying.

LIST

The program LIST lists the content of a file on the screen.

The commands in the program are

```
R 'filename'    ! List a file of real numbers
P 'filename'    ! List a parameterfile (generated in IDENT)
D 'filename'    ! List a datafile (generated in DATAGEN or
                  SAMPLE)
E               ! Exit from the program
H               ! Write out a help text
```

The filenames must be given without extension. The program stops the listing every 10 lines. To continue the listing type C to stop the listing type E or Q.

10.4 Calculation programs

IDENT

The program IDENT identifies a discrete model

$$A(q^{-1})\ y(k) = B(q^{-1})\ u(k - Td) + C(q^{-1})\ e(k) \qquad (10.1)$$

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})}\ u(k - Td) + \frac{C(q^{-1})}{D(q^{-1})}\ e(k) \qquad (10.2)$$

$$(1 + A(q^{-1}))\ y(k) = \frac{B(q^{-1})}{1 + F(q^{-1})}\ u(k - Td) + \frac{1 + C(q^{-1})}{1 + D(q^{-1})} \qquad (10.3)$$

from signals on a datafile and stores the estimations of A(q) for every sample on a parameterfile.

The program starts by asking which datafile you want to identify from, then the program asks for the name of the output parameterfile. Give the filenames without extensions. The next question will be 'MANUAL OR AUTOSTART ?' If you answer M (= manual) various parameters must be given to start the identification algorithm. You will be asked for these parameters by the questions

```
ID MODELORDER       !  The order of the identified model.
                       Must be < 4.
A 1                 !  Initial values for the A parameters
  .
  .
A modelorder

B 1                 !  Initial values for the B parameters
  .
  .
B modelorder

RUNBETA (Y/N)       !  Yes means a special identification
                       algorithm, a little different from
                       the LQ algorithm. Don't use it !
   gamma1           !  Parametervalues you must give if
   gamma2              you chose this algorithm.
   death

lambda0             !  Forgetting factor in the identification
lambda
delay Td            !  The delay in formulas (1) .. (3)
```

```
ABCD                 !  If you want to identify according to (2)
ABCDF                !  If you want to identify according to (3)

P for Y              !  Initial diagonal values of the P - matrix
PY
P for U
PU
P for EPS
PEPS
P for V and Z        ! Will be asked only if you identify model
PVZ                    (2) or (3)
```

Use manual only to alter the modelorder parameter and use the values given in the 'AUTO-case' for the rest of the parameters.

If you answer A (=auto) the identification will be started with the following parameters

```
MODELORDER = 2
RUNBETA = NO
ABCD = NO
ABCDF = NO
LAMBDA0 = 1
LAMBDA = 0.975
Td = 0
A1 = 0
A2 = 0
B1 = 0
B2 = 0
PY1 = 100
     .
     .
PEPS2 = 100
```

The program now writes out which channels are available and you chose which one you want as input channel (u in formula (1) .. (3)) and as output channel (y in formula (1) .. (3)). The identification starts and you can see its progress on the screen.

When the identification is finished you get the question 'Another identification?' if the identification has been successful answer N and you leave the program with the A parameters saved on a file. If you answer Y nothing is saved and you can redo the identification, probably with other parameters.

The result of the identification will be stored on a parameter file with the format

```
NUMBER_OF_SAMPLES
MODELORDER
A_1(1)
.
.
A_MODELORDER(1)
.
.
A_1(NUMBER_OF_SAMPLES)
.
.
A_MODELORDER(NUMBER_OF_SAMPLES)
```

Unfortunately this is not a very fast program, this table compares identification speed for similar algorithms on an Apple and a VAX.

Time/recursion	C o m p u t e r	
Model order	APPLE][+	VAX 11/780
1	1.3 s	56 ms
2	2.5 s	66 ms
3	4.5 s	95 ms

Table 10.2 The time for one recursion of the identification algorithm, on different computers.

The memory limitations of the Apple makes it impossible to identify according to model (2) or (3) with a modelorder greater than 2. If modelsstructure (1) is used the Apple can handle models of order 3 too.

PARCALC

The program carries out different calculations on the identified parameters, given from IDENT on a parameter file. The program is menu-driven. The program can plot parameterfiles in two formats, the user will be asked for the limits of the parametersizes. The program can also compute meanvalues of the parameters in a given time interval and the quadratic parameter-distance

$$d^2 = \Sigma\,(a_i - \hat{a}_i)^2$$

where a_i are given by the user and a@^$_i$ are read from the file. This can be used to detect changes in the estimated parameters.

<u>STAT</u>

The program STAT computes statistics from a datafile. Meanvalue, variance and a histogram plot is computed for every channel on a file.

The program starts by asking for the name of the datafile you want to examine. Give the name without extension. STAT then computes statistics and histograms of all channels of the file, and writes out which channels are available in the file. To display the results of these computations the following one-letter commands are available

```
H 'channelnumber' ! Display the histogram of the given channel
S 'channelnumber' ! Display statistics of the given channel
A                 ! Display statistics of all channels
E                 ! Exit from the program
```

10.5 Communication programs

BACKUP

The program BACKUP transfers textfiles from an APPLE memory device to a VAX.

First you must give the VAX/VMS command

```
$ SET TERMINAL/SPEED=300/NOPARITY
```

from another terminal, then plug in the Apple and execute the Apple program BACKUP. The program starts by asking if you want to read from a backupfile.

If you answer N (=no) the program prompts with ''' and you write the name of the textfile you want to transfer to the VAX. The program continues to prompt ''' until you write '$'. At the most you can transfer 30 files in one execution.

If you answer Y (=yes) the program will read a number of filespecifications from a textfile named 'SYS:BACKUP.DATA'. This file has the format

```
PREFIX_1
FILE_1
.
.
.
PREFIX_N
FILE_N
```

Maximum number of filenames in this 'backupfile' is 30. You can create this file with the Apple Pascal editor, if you rename the file after you have written it. (Output files from the editor will always be '.TEXT' files.) Then the program will write out these filenames one by one and ask if you want to transfer the file. Write 'Y' if you want transmission 'N' otherwise. After you have chosen the files the transmission takes place. It is very convenient to have the files you do backup on once a day on a file like this, so that the backup becomes easier. All filenames must be given without extension, '.TEXT' is always assumed.

This program was mainly used during the program development, to have backup of the programs on a more reliable memory device than the Apple devices. It was also possible to use the texthandling programs on the VAX on these files.

VAX

The program VAX transfers an Apple datafile to an ASCII-file on the VAX-computer.

First you must give the VAX/VMS command

```
$ SET TERMINAL/SPEED=300/NOPARITY
```

from another terminal, then plug in the Apple and execute the Apple program VAX. The program asks which file you want to transfer to the VAX. Give the filename of the Apple datafile, without extension. Then you will be asked for the name of the VAX file you will create. On the VAX this file will be created with the VMS command CREATE. After this the Apple file will be transferred to a VAX ASCII file. One real number is converted to 8 ASCII characters which are transferred to the VAX. This is a rather slow program, it takes about 7 minutes to transfer 1000 real numbers.

VAXCOM

The program VAXCOM enables the user to use the Apple as a very simple terminal to a VAX.

First you must give the VAX/VMS command

```
$ SET TERMINAL/SPEED=300/NOPARITY
```

from another terminal, then plug in the Apple and execute the program VAXCOM.

Be careful! This communication protocol is extremely simple. Don't try to do something advanced with this program. The program should only be used to check the communication between Apple and VAX. Only use 'simple' VAX/VMS commands to test this e.g. SHOW DEFAULT, SHOW USERS or DIRECTORY. The most important thing is that you must not produce a $-sign from the VAX to the Apple, the Apple will interpret this as a VAX/VMS prompt and consider the response of the last given command as terminated. You can terminate the program by typing 'SLUT'.

PRSET

The program PRSET sets the Epson printer in American mode.

10.6 IDPAC calls from Apple

If you give the command ID Y in the program SAMPLE, then SAMPLE will make the VAX/VMS commandprocedurecall @IDP 'vaxfile'.T and the following commandprocedure will be executed.

```
$ SET NOVERIFY                          ! Don't echo the commands
$ LEN='F$LOCATE(".",P1)'
$ S:='F$EXTRACT(0,LEN,P1)'              ! Find the string 'vaxfile'
$ SET DEFAULT DISK:[USERDIR]            ! Go to the desired directory
$ F:='S'".COM"                          ! Create unique filenames
$ SL:="SL"'S'                           ! for this procedure
$ !
$ ! Now write a file which, when executed, will
$ ! call IDPAC, and do some analysis on the
$ ! file 'vaxfile'.T. The logical name of this file
$ ! is COMFILE.
$ ! In this procedure it is supposed that we have sampled
$ ! 2 channels with 5 Hz and stored 1000 samples.
$ !
$ OPEN/WRITE COMFILE 'F'
$ !
$ ! Go to the directory where you want to place the
$ ! the result files.
$ !
$ WRITE COMFILE "$ SET DEFAULT DISK:[DATADIR]"
$ !
$ ! Call IDPAC
$ !
$ WRITE COMFILE "$ RUN DISK:[PROGDIR]IDPAC"
$ !
$ ! 5 Hz = 0.2 sec
$ !
$ WRITE COMFILE " LET TICK.=0.2"
$ WRITE COMFILE " LET DELTA.=0.2"
$ !
$ ! Give IDPAC commands which will convert the
$ ! input textfile to a IDPAC binary data-
$ ! file with 2 columns.
$ ! The input file is 'vaxfile' and the outputfile
$ ! will be SL'vaxfile'
$ !
$ WRITE COMFILE " CONV ''SL''''S' 2"    ! 2 channels
$ WRITE COMFILE "
$ WRITE COMFILE " ! Here should the rest of the
$ WRITE COMFILE " ! IDPAC commands be placed.
$ WRITE COMFILE " ! Be sure to have created unique
$ WRITE COMFILE " ! filenames for all files you
$ WRITE COMFILE " ! create in IDPAC.
$ WRITE COMFILE "
$ WRITE COMFILE " STOP"
$ WRITE COMFILE "$ EXIT"
$ CLOSE COMFILE
$ L:="DISK:[DATADIR]"'S'".LOG"
$ !
$ ! Start the file 'vaxfile'.COM  as a batch-job.
$ ! Delete the file 'vaxfile'.COM when the batch-job
$ ! is ready.
$ !
$ SUBMIT/NOPRINT/DELETE/NOIDENTIFY/LOG_FILE='L'  'F'
```

To be able to use this efficiently further knowledge in the art of writing commandprocedures is required.

APPENDIX 2

Program code

```
program Backup;

(*
  Transfers a number of apple-textfiles to a vax-computer.
  The filenames are given in the program or  read from
  a backup-file with the format:

  Prefix1
  File1
  Prefix2
  File2
  .
  .
  Prefixn
  Filen

  Author : Per Persson
  Date : 83-JULY-12    *)

var
  InFile   : file of char;
  Fil      : file of char;
  OutFile  : file of char;
  Back     : file of char;
  Sg       : string;
  Ch       : char;
  I        : integer;
  J        : integer;
  K        : integer;
  Vx1      : integer;
  Nof      : integer;
  FilArr   : array [1..30] of string;
  PrefArr  : array [1..30] of string;
  B        : array [1..30] of boolean;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 procedure Vx(Ch : char);

 (* Writes one character on the transmissionline.   *)

 begin
   OutFile^ := Ch;
   Put(OutFile);
 end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
function Gt : char;

(* Reads one character from the transmissionline.  *)

begin
  Get(InFile);
  Gt := InFile^;
```

```
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Init;

(* Initializes the program and reads filenames. *)

var
  St    : string;
  Colon : integer;
  Long  : integer;

begin
  for I := 1 to 30 do
    B[I] := true;
  rewrite(InFile,  'REMIN:');
  rewrite(OutFile, 'REMOUT:');
  writeln('Textfiles assumed.');
  write('Do you want to read from backupfile? ');
  readln(Ch);
  if Ch = 'N' then
  begin
    Nof := 0;
    repeat
      Nof := Nof + 1;
      write('>');
      readln(St);
      if St <> '$' then
      begin
        Colon := pos(':', St);
        Long := length(St);
        PrefArr[Nof] := copy(St, 1, Colon);
        FilArr[Nof] := copy(St, Colon + 1, Long - Colon);
      end;
    until St = '$';
    Nof := Nof - 1;
  end
  else
  begin
    reset(Back, 'SYS:BACKUP.DATA');
    Nof := 0;
    while not eof(Back) do
    begin
      Nof := Nof + 1;
      readln(Back, PrefArr[Nof]);
      readln(Back, FilArr[Nof]);
    end;
    writeln;
    writeln('Type Y for backup');
    writeln;
    for I := 1 to Nof do
    begin
      writeln(PrefArr[I], FilArr[I], '.TEXT');
      write('      - > ');
      readln(Ch);
      if Ch = 'Y' then
```

```
        B[I] := true
      else
        B[I] := false;
    end;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Away;

(* Sends one file on the transmissionline,
the escape-caracter is used as line terminator on the vax.  *)

var
  Ch : char;

begin
  while not eof(Fil) do
  begin
    if eoln(Fil) then
    begin
      readln(Fil);
      Vx(chr(27));
      Ch := Gt;
    end
    else
    begin
      read(Fil, Ch);
      Vx(Ch);
    end;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 begin
   Init;
   for I := 1 to Nof do
   if B[I] then
   begin
     write('Backup ', PrefArr[I], FilArr[I], '.TEXT');
     writeln;
     write('            - >');
     reset(Fil, concat(PrefArr[I], FilArr[I], '.TEXT'));

     (* Create a VAX/VMS file with the command create *)

     Vx('C');
     Vx('R');
     Vx('E');
     Vx('A');
     Vx('T');
     Vx('E');
     Vx(' ');
     for J := 1 to length(FilArr[I]) do
       Vx(FilArr[I][J]);
     Vx('.');
     Vx('A');
    Vx('P');
    Vx('L');
    Vx(chr(27));
    repeat
      Ch := Gt;
    until Ch = '$';
    Away;
    Vx(chr(90-64));
    repeat
      Ch := Gt;
    until Ch = '$';
    close(Fil);
    writeln('to vax ', FilArr[I], '.APL');
  end;
end.
```

```
program DataGen;
uses AppleStuff;

(* Generates a datafile from the model

a(q)y(t) = b(q)u(t) + e(t)

where u(t) is a prbs-signal and e(t) is noise.

Author : Per Persson
Date : 83-july-04 *)

type
  ObsType = record
              U : real;
              Y : real;
            end;
var
  Y          : array [0..50] of real;
  U          : array [0..50] of real;
  A          : array [0..50] of real;
  B          : array [0..50] of real;
  R          : real;
  Niv        : real;
  NoiseLev   : real;
  J          : integer;
  N          : integer;
  I          : integer;
  ModelOrder : integer;
  P1         : integer;
  P2         : integer;
  P3         : integer;
  P4         : integer;
  P5         : integer;
  P6         : integer;
  P7         : integer;
  P8         : integer;
  P9         : integer;
  P10        : integer;
  P10        : integer;
  P11        : integer;
  P12        : integer;
  Data       : file of real;
  Obs        : ObsType;
  St         : string;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure PRBS;

(* Generates a prbs-sequence. *)

var
  Ph : integer;

begin
  U[0] := P12;
```

```
    if ( P11 * P12 = 0 ) and ( P11 + P12 = 1 ) then
      Ph := 1
    else
      Ph := 0;
    P12 := P11;
    P11 := P10;
    P10 := P9;
    P9 := P8;
    P8 := P7;
    P7 := P6
    P6 := P5;
    P5 := P4;
    P4 := P3;
    P3 := P2;
    P2 := P1;
    P1 := Ph;
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure Start;

  (* Initializes the program. *)

  var
    I : integer;

  begin
    write('Output file:       ');
    readln(St);
    St := concat(St, '.DATA');
    write('Modelorder:        ');
    readln(ModelOrder);
    for I := 1 to ModelOrder do
    begin
      write('a', I, ':                 ');
      readln(A[I]);
    end;
    for I := 1 to ModelOrder do
    begin
      write('b', I, ':                 ');
      readln(B[I]);
    end;
    write('Noiselevel:        ');
    readln(NoiseLev);
    write('Number of samples:');
    readln(N);
    P1 := 1;
    P2 := 0;
    P3 := 1;
    P4 := 1;
    P5 := 1;
    P6 := 1;
    P7 := 1;
    P8 := 0;
    P9 := 1;
    P10 := 1;
    P11 := 1;
    P12 := 1;
    for I := 1 to 50 do
    begin
      Y[I] := 0;
      U[I] := 0;
    end;
    rewrite(Data, St);
    Data^ := N;
    Put(Data);
    Data^ := 2;
    Put(Data);
    Data^ := ModelOrder;
    Put(Data);
    Data^ := 0;
    Put(Data);
    Data^ := 4;
    Put(Data);
    Put(Data);
    Data^ := 1;
    Put(Data);
    Data^ := 4;
    Put(Data);
    Put(Data);
    Niv := 0.5;
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  begin
    Start;
    for I := 1 to N do
    begin
      PRBS;
      Y[0] := 0;
      for J := 1 to ModelOrder do
        Y[0] := Y[0] - A[j] * Y[j] + B[j] * U[j];
      Obs.U := U[0];
      R := random;
      R := R / 32767;
      Y[0] := Y[0] + NoiseLev * (R - Niv);
      Obs.Y := Y[0];
      Data^ := Obs.U;
      Put(Data);
      Data^ := Obs.Y;
      Put(Data);
      for J := ModelOrder downto 1 do
      begin
        Y[J] := Y[J - 1];
        U[J] := U[J - 1];
      end;
    end;
    close(Data, lock);
   end.
```

```
program Disp;
uses TurtleGraphics, AppleStuff;

(* Displays a datafile graphically on the screen.

   Author : Per Persson
   Date : 83-july-04 *)

type
  SetType = set of 'A'..'Z';

var
  Dat      : file of real;
  Command  : SetType;
  I        : integer;
  J        : integer;
  K        : integer;
  Mo       : integer;
  Ns       : integer;
  Kan      : integer;
  Chan     : integer;
  Noc      : integer;
  Inp      : integer;
  R1       : real;
  R2       : real;
  Oldfile  : string;
  Fil      : string;
  St       : string;
  Ch       : char;
  Chanv    : array [0..15] of
               record
                 I : integer;
                 G : integer;
               end;
  Range    : array [0..7,1..2] of real;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Help;

(* Is called to write out a Help text. *)

begin
  writeln;
  writeln('H Help');
  writeln('C Continue');
  writeln('E Exit');
  writeln('N New channel/file. Then $ for same file');
  writeln;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Dot(X, Y : integer);

(* Puts a dot on the screen. *)

var
  D : boolean;
```

```
begin
  DrawBlock(D, 1, 0, 0, 1, 1, X, Y, 3);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure StringIt(X : real; Int : boolean; var St : string);

(* Converts a real number to a text-string. *)

var
  I : integer;
  J : integer;
  S : string;

begin
  St := ' ';
  if X < 0 then
    St := concat(St, '-');
  if abs(X) < 1 then
    St := concat(St, '0');
  X := abs(X);
  I := trunc(X);
  str(I, S);
  if I > 0 then
    St := concat(St, S);
  if not Int then
  begin
    X := X - trunc(X);
    I := trunc(100 * X);
    str(I, S);
    if I < 1 then
      S := concat('00', S)
    else if I < 10 then
      S := concat('0', S);
    St := concat(St, '.', S);
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Rng;

(* Initializes the program. *)

var
  I : integer;

begin
  Fil := ' ';
  OldFile := ' ';
  Command := ['N','C','E'];
  Range[0,1] := 5;
  Range[0,2] := 0;
  Range[1,1] := 1;
  Range[1,2] := 0;
  Range[2,1] := 0.5;
  Range[2,2] := 0;
  Range[3,1] := 0.1;
  Range[3,2] := 0;
  Range[4,1] := 10;
  Range[4,2] := 5;
  Range[5,1] := 2;
  Range[5,2] := 1;
  Range[6,1] := 1;
  Range[6,2] := 0.5;
  Range[7,1] := 0.2;
  Range[7,2] := 0.1;
  for I := 0 to 15 do
  begin
    Chanv[i].I := 0;
    Chanv[i].G := 0;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure NewScr(K : integer);

(* Writes a new form on the screen
   with the time interval k*250 to (k + 1)*250.  *)

var
  I  : integer;
  St : string;

 procedure Ax(X, Y : integer; B : boolean; Val : real);
 begin
   PenColor(None);
   MoveTo(X, Y);
   StringIt(Val, B, St);
   PenColor(White);
   wstring(St);
 end;

begin
  InitTurtle;
  PenColor(White);
  FillScreen(Black);
  for I := 20 to 270 do
  begin
    Dot(I, 10);
    Dot(I, 170);
  end;
  for I := 11 to 169 do
  begin
    Dot(20, I);
    Dot(270, I);
  end;
  Ax(13, 0, true, K * 50);
  Ax(103, 0, true, ( K + 2 ) * 50);
  Ax(203, 0, true, ( K + 4 ) * 50);
  Ax(251, 0, true, ( K + 5 ) * 50);
  R1 := Range[Chanv[Kan].I,1];
  R2 := Range[Chanv[Kan].G,2];
  Ax(0, 6, true, -R2);
  Ax(0, 86, true, R1 / 2 - R2);
  Ax(0, 166, true, R1 - R2);
  Dot(18, 10);
  Dot(19, 10);
  Dot(18, 90);
  Dot(19, 90);
  Dot(18, 170);
  Dot(19, 170);
  Dot(20, 9);
  Dot(120, 9);
  Dot(220, 9);
  Dot(270, 9);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowChan;

(* Plots the content of a channel on the screen. *)

begin
  for J := 1 to Ns  do
  begin
    if ((J mod 250) = 0) or (J = 1) then
    begin
      if J > 1 then
      begin
        TextMode;
        repeat
          write('>');readln(Ch);
          case Ch of
            'E' : exit(program);
            'N' : exit(ShowChan);
            'C' : GrafMode;
            'H' : Help;
          end;
        until (Ch in Command );
      end;
      NewScr(J div 50);
      PenColor(None);
      MoveTo(20, 175);
      StringIt(Kan, true, St);
      St := concat('cannelnumber', St);
      PenColor(White);
      wstring(St);
      for I := 1 to Chan do
        Get(Dat);
      Inp := trunc(((Dat^ + R2) * 160) / R1);
      PenColor(None);
      MoveTo(20, Inp + 10);
      PenColor(White);
      for I := Chan + 1 to Noc do
        Get(Dat);
    end;
    for I := 1 to Chan do
      Get(Dat);
      Inp := trunc(((Dat^ + R2) * 160) / R1);
      for I := Chan + 1 to Noc do
        Get(Dat);
      MoveTo((J mod 250) + 20, Inp + 10);
      if ((J + 1) mod 250) = 0 then
        readln(Ch);
      PenColor(White);
    end;
    readln;
    TextMode;
    write('Type Q to quit. ');
    readln(Ch);
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
  begin
    Rng;
    repeat
      if Fil <> '$' then
        OldFile := Fil;
      write('Display file:       ');
      readln(Fil);
      if Fil = '$' then
        reset(Dat, concat(OldFile, '.DATA'))
      else
        reset(Dat, concat(Fil, '.DATA'));
      Ns := trunc(Dat^ + 0.2);
      Get(Dat);
      Noc := trunc(Dat^ + 0.2);
      Get(Dat);
      Mo := trunc(Dat^ + 0.2);
      writeln('Number of sampels: ', Ns);
      writeln('Systemorder:       ', Mo);
      writeln('Available channels:');
      for I := 1 to Noc do
      begin
        Get(Dat);
        J := trunc(Dat^ + 0.2);
        write(' ', J);
        Chanv[J].I := I;
        Get(Dat);
        K := trunc(Dat^ + 0.2);
        Chanv[J].G := K;
        Get(Dat);
      end;
      writeln;
      write('Display channel: ');
      readln(Chan);
      Kan := Chan;
      Chan := Chanv[Chan].I;
      ShowChan;
      close(Dat);
    until Ch = 'Q';
  end.
```

```
program Ident;

uses TurtleGraphics, AppleStuff;

(* Identifies a model
   B(q)y(t) = B(q)u(t) + C(q)e(t)
   from a given datafile.

   Author: Per Persson
   Date: 04 - JULY - 1983 *)

type
  FiType  = array [1..20] of real;
  MatType = array [1..10,1..10] of real;
  RecType = record
              u : real;
              y : real;
            end;
  FilType = File of real;

var
  Fi       : FiType;
  Theta    : FiType;
  W        : FiType;
  P1       : MatType;
  P        : MatType;
  Lambda   : real;
  Lambda0  : real;
  Gamma1   : real;
  Gamma2   : real;
  Beta     : real;
  Death    : real;
  R        : real;
  S        : real;
  Nlev     : real;
  Eps1     : real;
  V1       : real;
  Z1       : real;
  U1       : real;
  U        : real;
  Y1       : real;
  Y        : real;
  Td       : integer;
  Yl       : integer;
  Yh       : integer;
  Ul       : integer;
  Uh       : integer;
  El       : integer;
  Eh       : integer;
  Zl       : integer;
  Zh       : integer;
  Vl       : integer;
  Vh       : integer;
  T        : integer;
  G1       : integer;
```

```
   G2       : integer;
   Ja       : integer;
   Inp      : integer;
   Outp     : integer;
   Mo       : integer;
   Np       : integer;
   I        : integer;
   J        : integer;
   Noc      : integer;
   Na       : integer;
   Nb       : integer;
   Nc       : integer;
   Nd       : integer;
   Nf       : integer;
   Ua       : array [1..20] of real;
   A        : array [1..20] of real;
   B        : array [1..20] of real;
   Dt       : RecType;
   Ch       : char;
   Print    : char;
   Ch1      : char;
   Sgd      : string;
   Sgp      : string;
   St       : string;
   ABCD     : boolean;
   ABCDF    : boolean;
   RunBeta  : boolean;
   Pe       : boolean;
   ChanVec  : array [0..15] of boolean;
   Apar     : FilType;
   Data     : FilType;
   Pri      : file of char;
(* parameter file format:

                   number of sampel
                   modelorder
                   a1(1)
                   a2(1)
                   .
                   .
                   amo(1)
                   .
                   .
                   .
                   a1(nos)
                   a2(nos)
                   .
                   .
                   amo(nos)

         data file format:

                   number of samples
                   number of channels
                   modelorder
```

```
                    chnr(1)
                    gc(1)
                    gc(1)
                    .
                    .
                    chnr(noc)
                    gc(noc)
                    gc(noc)
                    y1(1)
                    .
                    .
                    ynoc(1)
                    .
                    .
                    .
                    y1(ns)
                    .
                    .
                    ynoc(ns)
  *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure PlotDot(X, Y : integer);

(* Plots a dot on the screen at x y. *)

var
  Dot : boolean;

begin
  DrawBlock(Dot, 1, 0, 0, 1, 1, X, Y, 3);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure InitAll(Bo : boolean);

(* Initailizes the program. *)

var
  I : integer;
  J : integer;

begin
  for I := 1 to 20 do
  begin
    Fi[I] := 0;
    W[I] := 0;
    Theta[I] := 0;
    Ua[I] := 0;
  end;
  for I := 0 to 15 do
    ChanVec[I] := false;
  rewrite(Apar, Sgp);
  reset(Data, Sgd);
  Np := trunc(Data^ + 0.5);
  Apar^ := Np;
  Put(Apar);
```

```
    if Bo then
      writeln(Np, ' sampel');
    Get(Data);
    Noc := trunc(Data^ + 0.5);
    Get(Data);
    Mo := trunc(Data^ + 0.5);
    if Bo then
    begin
      writeln('Default modelorder: ', Mo);
      write('Id modelorder      :');readln(Na);
    end
    else
      Na := Mo;
    Apar^ := Na;
    Put(Apar);
    Nb := Na;
    Nc := Na;
    Nd := Na;
    Nf := Na;
    Yl := 1;
    Yh := Na;
    Ul := Yh + 1;
    Uh := Na + Nb;
    El := Uh + 1;
    Eh := Na + Nb + Nc;
    Zl := Eh + 1;
    Zh := Na + Nb + Nc + Nf;
    Vl := Zh + 1;
    Vh := Na + Nb + Nc + Nd + Nf;
    if Bo then
    begin
      for J := 1 to Na do
      begin
        write('a', J:1, ':                    ');
        readln(Theta[Yl + J - 1]);
      end;
      for J := 1 to Na do
      begin
        write('b', J:1, ':                    ');
        readln(Theta[Ul + J - 1]);
      end;
    end;
    Beta := 0;
    Epsl := 0;
    T  := 0;
    R  := 0;
    S  := 0;
    Y  := 0;
    Y1 := 0;
    U  := 0;
    U1 := 0;
    Z1 := 0;
    V1 := 0;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Channel;

(* Initializes the channelvector. *)

var
  I : integer;

begin
  for I := 1 to Noc do
  begin
    Get(Data);
    ChanVec[trunc(Data^ + 0.5)] := true;
    Get(Data);
    Get(Data);
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure InitAll2(Bo : boolean);

(* Initializes the program.
   The initialization is too long to
   be made in one procedure.
   An Apple-Pascal limitation. *)

var
  I  : integer;
  J  : integer;
  Ph : real;
  Ch : char;

procedure SetUpp;

(* Questions and answers *)

begin
  writeln('P for y');
  write('py:        ');
  readln(Ph);
  for I := Yl to Yh do
    P[I,I] := Ph;
  writeln('P for u');
  write('pu:        ');
  readln(Ph);
  for I := Ul to Uh do
    P[I,I] := Ph;
  writeln('P for eps');
  write('peps:      ');
  readln(Ph);
  if Ph = 0 then
    Pe := false
  else
    Pe := true;
  for I := El to Eh do
    P[I,I] := Ph;
  if ABCD or ABCDF then
```

```
    begin
      writeln('P for v and z');
      write('pzv:    ');
      readln(Ph);
      for I := Zl to Vh do
        P[I,I] := Ph;
    end;
  end;

  begin
    for I := 1 to 10 do
      for J := 1 to 10 do
        P[I,J] := 0;
    if not Bo then
    begin
      RunBeta := false;
      Gamma1  := 0.95;
      Gamma2  := 0.95;
      Lambda0 := 1;
      Lambda  := 0.975;
      Death   := 0.5;
      Td := 0;
      for I := Yl to Eh do
        P[I,I] := 100;
      Pe := true;
      Vl := Eh;
      Vh := Eh;
      Zl := Eh;
      Zh := Eh;
      ABCD  := false;
      ABCDF := false;
    end
    else
    begin
      write('RunBeta (Y/N)   ');
      readln(Ch);
      if Ch = 'Y' then
        RunBeta := true
      else
        RunBeta := false;
      if RunBeta then
      begin
        write('Gamma1:             ');
        readln(Gamma1);
        write('Gamma2:             ');
        readln(Gamma2);
        write('Death:              ');
        readln(Death);
      end;
      write('Lambda0:            ');
      readln(Lambda0);
      write('Lambda:             ');
      readln(Lambda);
      write('Delay td:           ');
      readln(Td);
    write('ABCD model ?');
    readln(Ch);
    if Ch = 'Y' then
      ABCD := true
    else
    begin
      ABCD := false;
      write('ABCDF model?');
      readln(Ch);
      if Ch = 'Y' then
        ABCDF := true
      else
        ABCDF := false;
    end;
    if ABCD then
    begin
      Vl := Zh;
      Vh := Zh;
    end;
    if (not ABCD) and (not ABCDF) then
    begin
      Vl := Eh;
      Vh := Eh;
      Zl := Eh;
      Zh := Eh;
    end;
    SetUpp;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowCoeff;

(* Displays the current parameter estimate on a printer. *)

var
  I : integer;
  J : integer;

begin
  writeln(Pri, '- - - - - - - - - - - - - - - - - ');
  writeln(Pri, 'Datafile:       'Sgd);
  writeln(Pri, 'Parameterfile: ', Sgp);
  writeln(Pri, 'time ', T+1:4, ' sampel');
  write(Pri, 'a:');
  for I := Yl to Yh do
    write(Pri, Theta[I]:9:4);
  writeln(Pri);
  write(Pri, 'b:');
  for I := Ul to Uh do
    write(Pri, Theta[I]:9:4);
  writeln(Pri);
  write(Pri, 'c:');
  for I := El to Eh do
    write(Pri, Theta[I]:9:4);
  if ABCD Then
  begin
    write(Pri, 'd:');
    for I := Zl to Zh do
      write(Pri, Theta[I]:9:4);
    writeln(Pri);
  end;
  if ABCDF then
  begin
    write(Pri, 'f:');
    for I := Vl to Vh do
      write(Pri, Theta[I]:9:4);
  end;
  writeln(Pri);
  writeln(Pri);
  writeln(Pri);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Recid(Obs : RecType);

(* Carries out the recursive identiFication. Makes one recursion
   each time it is called *)

var
  ThetaIn : FiType;
  Ny      : real;
  Ny0     : real;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 procedure MultVM(Min : MatType; Vin : FiType; var Vut : FiType);

 var
   I      : integer;
   J      : integer;
   Vi     : real;
   LoInd  : integer;
   HiInd  : integer;

 (* Vut := Min * Vin *)

 begin
   if ABCD or ABCDF then
   begin
     LoInd := Yl;
     HiInd := Vh;
   end
   else
   begin
     LoInd := Yl;
     HiInd := Eh;
   end;
   for I := LoInd to HiInd do
   begin
     Vi := 0;
     for J := LoInd to HiInd do
       Vi := Vi + Min[I,J] * Vin[J];
     Vut[I] := Vi;
    end;
  end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure Mul(p : MatType; Fi : FiType; var Fih : MatType);

  (* Fih := P * Fi * transp(Fi) * p where p is symmetric *)

  var
    I      : integer;
    J      : integer;
    Sla    : FiType;
    H      : real;
    LoInd  : integer;
    HiInd  : integer;

  begin
    if ABCD or ABCDF then
    begin
      LoInd := Yl;
      HiInd := Vh;
    end
    else
    begin
      LoInd := Yl;
      HiInd := Eh;
    end;
    for I := LoInd to HiInd do
    begin
      H := 0;
      for J := LoInd to HiInd do
        H := H + Fi[J] * P[I,J];
      Sla[I] := H;
    end;
    for I := LoInd to HiInd do
      for J := I to HiInd do
      begin
        Fih[I,J] := Sla[I] * Sla[J];
        Fih[J,I] := Fih[I,J];
      end;
  end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure Measure;

  (* Updates the measurements *)

  var
    I   : integer;
    Uh  : real;
    UH1 : real;

  begin
    Y1 := Y;
    Y := Obs.Y;
    if Td = 0 then
    begin
```

```
      U1 := U;
      U := Obs.U;
    end
    else
    begin
      U1 := Ua[Td];
      for I := Td downto 2 do
        Ua[I] := Ua[I - 1];
      Ua[1] := Obs.U;
    end;
  end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure UpDateFi;

  var
    I  : integer;
    H  : real;
    H1 : real;

  begin
    for I := Yh downto Yl + 1 do
      Fi[I] := Fi[I - 1];
    Fi[Yl] := -Y1;
    for I := Uh downto Ul + 1 do
      Fi[I] := Fi[I - 1];
    Fi[Ul] := U1;
    for I := Eh downto El + 1 do
      Fi[I] := Fi[I - 1];
    Fi[El] := Eps1;
    if ABCDF then
    begin
      for I  := Vh downto Vl + 1 do
        Fi[I] := Fi[I - 1];
      Fi[Vl] := -V1;
    end;
    if ABCDF or ABCD then
    begin
      for Ii := Zh downto Zl + 1 do
        Fi[I] := Fi[I - 1];
      Fi[Zl] := -Z1;
    end;
  end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure ComputeZ;

  var
    I : integer;

  begin
    Z1 := 0;
    for I := Zl to Zh do
      Z1 := Z1 - Theta[I] * Fi[I];
    for I := Ul to Uh do
      Z1 := Z1 + Theta[I] * Fi[I];
  end;
```

```
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeV;

var
  I : integer;

begin
  V1 := 0;
  for I := Vl to Vh do
    V1 := V1 - Theta[I] * Fi[I];
  for I := El to Eh do
    V1 := V1 + Theta[I] * Fi[I];
  V1 := V1 + Eps1;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeEps;

var
  I : integer;

begin
  Eps1 := y;
  for I := Yl to Eh do
    Eps1 := Eps1 - Theta[I] * Fi[I];
  if ABCD or ABCDF Then
    for I := Zl to Vh do
      Eps1 := Eps1 - Theta[I] * Fi[I];
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeP;
var
  Fih : MatType;
  Vec : FiType;
  I   : integer;
  J   : integer;
  H   : real;
  Sla : FiType;

begin
  P1 := P;
  Mul(P, Fi, Fih);
  MultVM(P, Fi, Vec);
  H := 0;
  for I := Yl to Vh do
    H := H + Vec[I] * Fi[I];
  H := H + Lambda;
  for I := Yl to Vh do
    for J := I to Vh do
    begin
      P[I,J] := (P[I,J] - (Fih[I,J] / H)) / Lambda;
      P[J,I] := P[I,J];
    end;
  if RunBeta then
  begin
    for I := Yl to Uh do
```

```
      P[I,I] := P[I,I] + Beta;
    if ABCD then
      for I := Zl to Zh do
        P[I,I] := P[I,I] + Beta;
    if ABCDF then
      for I := Vl to Vh do
        P[I,I] := P[I,I] + Beta;
    if Pe then
      for I := El to Eh do
        P[I,I] := P[I,I] + Beta;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeLambda;
begin
  Lambda := Lambda0 * Lambda + (1 - Lambda0);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeTheta;

var
  I   : integer;
  Fi2 : real;
  H1  : real;
  H   : real;
  Vut : FiType;

begin
  MultVM(P, Fi, ThetaIn);
  for I := Yl to Eh do
    ThetaIn[I] := ThetaIn[I] * Eps1;
  if ABCD or ABCDF then
    for I := Zl to Vh do
      ThetaIn[I] := ThetaIn[I] * Eps1;
  for I := Yl to Eh do
    Theta[I] := Theta[I] + ThetaIn[I];
  if ABCD or ABCDF then
    for I := Zl to Vh do
      Theta[I] := Theta[I] + ThetaIn[I];
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ComputeBeta;

var
  I   : integer;
  Fi2 : real;
  H1  : real;
  H   : real;
  Vut : FiType;

(* Generally not used. *)

begin
  S := 0;
  for I := Yl to Eh do
```

```
        S := S + ThetaIn[I] * W[I];
      for I := Yl to Eh do
        W[I] := Gamma1 * W[I] + ThetaIn[I];
      if ABCD or ABCDF then
      begin
        for I := Zl to Vh do
          S := S + ThetaIn[I] * W[I];
        for I := Zl to Vh do
          W[I] := Gamma1 * W[I] + ThetaIn[I];
      end;
      if S > 0 then
        S := 1
      else
        S := -1.0;
      R := R * Gamma2 + (1 - Gamma2) * S;
      Fi2 := 0;
      for I := Yl to Eh do
        Fi2 := Fi2 + Fi[I] * Fi[I];
      if ABCD or ABCDF then
        for I := Zl to Vh do
          Fi2 := Fi2 + Fi[I] * Fi[I];
      MultVM(P1, Fi, Vut);
      H1 := 0;
      for I := Yl to Eh do
        H1 := H1 + Fi[I] * Vut[i];
      if ABCD or ABCDF then
        for I :=  Zl to Vh do
          H1 := H1 + Fi[I] * Vut[I];
      Ny0 := Lambda / (Lambda + H1);
      if R < 0.9 then
        Ny := 1
      else
        Ny := 8 * (1 - R);
      if R > 1 then
        Ny := 0;
      if Fi2 <> 0 then
        H := (Ny0 - Ny) / Fi2
      else
        H := 1000;
      if R > Death then
        Beta := H
      else
        Beta := 0;
    end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  begin
    Measure;
    UpDateFi;
    if ABCD or ABCDF then
      ComputeZ;
    ComputeEps;
    if ABCDF then
      ComputeV;
    ComputeLambda;
    ComputeP;
  ComputeTheta;
  if RunBeta then
    ComputeBeta;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure StringIt(X : real; Int : boolean; var St : string);

(* Converts a decimal number to a string. *)

var
  I : integer;
  J : integer;
  S : string;

begin
  St := ' ';
  if X < 0 then
    St := concat(St, '-');
  if abs(X) < 1 then
    St := concat(St, '0');
  X := abs(X);
  I := trunc(X);
  str(I, S);
  if I > 0 then
    St := concat(St, S);
  if not Int then
  begin
    X := X - trunc(X);
    I := trunc(10000 * X);
    str(I, S);
    if I < 1 then
      S := concat('000', S)
    else if I < 10 then
      S := concat('000', S)
    else if I < 100 then
      S := concat('00', S)
    else if I < 1000 then
      S := concat('0', S);
    St := concat(St, '.', S);
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Ax(X, Y : integer; B : boolean; Va : real);

(* Writes a number on the screen at a given position when the screen in
   in graphical mode *)

var
  St : string;

begin
  MoveTo(X, Y);
  StringIt(Va, B, St);
  wstring(St);
end;
```

```
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Form(Full : integer);

var
  J : integer;

begin
  GrafMode;
  FillScreen(Black);
  for J := 20 to 270 do
  begin
    PlotDot(J, 10);
    PlotDot(J, 170);
  end;
  for J := 11 to 169 do
  begin
    PlotDot(20, J);
    PlotDot(270, J);
  end;
  MoveTo(50, 180);
  wstring('Recursive identification');
  Ax(13, 0, true, Full * 50);
  Ax(103, 0, true, (Full + 2) * 50);
  Ax(203, 0, true, (Full + 4) * 50);
  Ax(251, 0, true, (Full + 5) * 50);
  Ax(0, 6, true, -2);
  Ax(0, 86, true, 0);
  Ax(0, 166, true, 2);
  PlotDot(18, 10);
  PlotDot(19, 10);
  PlotDot(18, 90);
  PlotDot(19, 90);
  PlotDot(18, 170);
  PlotDot(19, 170);
  PlotDot(20, 9);
  PlotDot(120, 9);
  PlotDot(220, 9);
  PlotDot(270, 9);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

begin

  (* Main program. Asks questions. You answer. *)

  Print  :=  'N';
  write('Printer ?');
  readln(Print);
  if ( Print = 'y' ) or ( Print = 'Y' ) then
  begin
    rewrite(Pri, 'PRINTER:');
    Print := 'Y';
  end;
  Ch := 'Y';
  while Ch = 'Y' do
```

```
begin
  write('Data file:        ');
  readln(Sgd);
  Sgd := concat(Sgd, '.DATA');
  write('Parameter file:  ');
  readln(Sgp);
  Sgp := concat(Sgp, '.DATA');
  write('Manual or autostart?(M/A)');
  readln(Ch1);
  InitAll(Ch1 = 'M');
  Channel;
  InitAll2(Ch1 = 'M');
  writeln('Available channels:');
  for I := 0 to 15 do
    if Chanvec[I] then
      writeln('Channel      ', I);
  writeln;
  write('Input  channel:      ');
  readln(Inp);
  write('Output channel:      ');
  readln(Outp);
  for J := 1 to Np do
  begin
    if (T mod 250) = 0 then
      Form(T div 50);
    for I := 0 to 15 do
      if ChanVec[I] then
      begin
        Get(Data);
        if I = Inp then
          Dt.U := Data^;
        if I = Outp then
          Dt.U := Data^;
      end;
    Recid(Dt);
    Ax(230, 180, true, T + 1);
    for I := 1 to Na do
    begin
      G1 := (16 - I)*10;
      Ax(30, G1, false, Theta[I]);
    end;
    if (((t + 1) mod 250) = 0) and ( Print = 'Y' ) then
      ShowCoeff;
    for I := 1 to Na do
    begin
      G1 := trunc((Theta[I] + 2) * 40) + 10;
      PlotDot((T mod 250) + 20, G1);
    end;
    for Ja := 1 to Na do
    begin
      Apar^ := Theta[Ja];
      Put(Apar);
    end;
    T := T + 1;
  end;
```

```
    if ((T mod 250) <> 0 ) and ( Print = 'Y' ) then
      ShowCoeff;
    readln(Ch);
    TextMode;
    write('Another identification? ');
    readln(Ch);
  end;
  close(Data);
  close(Apar, lock);
end.
```

```
program List;

(* Lists out files of real, as real numbers, parameters
   or data.

   Author: Per Persson
   Date: 15-AUGUST-1983 *)

var
  Fil : file of real;
  St  : string;
  Ch  : char;
  Ch1 : char;
  I   : integer;
  J   : integer;
  Ns  : integer;
  Mo  : integer;
  Noc : integer;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Help;

(* A help text is printed out at the command 'H'. *)

var
  Ch : char;

begin
  writeln;
  writeln('Type r, p or d followed by the filename. The printing');
  writeln('stops every 10 samples. To exit the procedure type');
  writeln('e or q, then you return to the commandline. To continue');
  writeln('the listing type <ret>. Exit help by typing <ret>');
  readln(Ch);
  writeln;
 end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Look;

(* Lists the file as a file of real numbers. *)

begin
  I := 1;
  reset(Fil, St);
  while not eof(Fil) do
  begin
    writeln(I, ' ', Fil^:9:4);
    I := I + 1;
    if (I mod 10) = 0 then
    begin
      write('>');
      readln(Ch);
      if (Ch = 'E') or (Ch = 'Q') then
        exit(Look);
    end;
    Get(Fil);
  end;
  readln;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ListPar;

(* Lists out the file as a parameterfile. *)

begin
  reset(Fil, St);
  Ns := trunc(Fil^ + 0.2);
  Get(Fil);
  Mo := trunc(Fil^ + 0.2);
  writeln('ns:  ', Ns);
  writeln('mo:  ', Mo);
  readln;
  for I := 1 to Ns do
  begin
    write(I, ':');
    for J := 1 to Mo do
    begin
      Get(Fil);
      write(Fil^:9:4);
    end;
    writeln;
    if (I mod 10) = 0 then
    begin
      write('>');
      readln(Ch);
      if (Ch = 'E') or (Ch = 'Q') then
        exit(ListPar);
    end;
  end;
  readln;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ListDat;

(* Lists out the file as a datafile. *)

begin
  reset(Fil, St);
  Ns := trunc(Fil^ + 0.2);
  Get(Fil);
  Noc := trunc(Fil^ + 0.2);
  Get(Fil);
  Mo := trunc(Fil^ + 0.2);
  writeln('ns:  ', Ns);
  writeln('noc: ', Noc);
  writeln('mo:  ', Mo);
  writeln;
  readln;
  for I := 1 to Noc do
  begin
    Get(Fil);
    writeln('channelrecord no ', I);
    writeln(trunc(Fil^ + 0.2));
    Get(Fil);
    writeln(trunc(Fil^ + 0.2));
    Get(Fil);
    writeln(trunc(Fil^ + 0.2));
    writeln;
    readln;
  end;
  for I := 1 to Ns do
  begin
    write(I, ':');
    for J := 1 to Noc do
    begin
      Get(Fil);
      write(Fil^:9:4);
    end;
    writeln;
    if (I mod 10) = 0 then
    begin
      write('>');
      readln(Ch);
      if (Ch = 'E') or (Ch = 'Q') then
        exit(ListDat);
    end;
  end;
  readln;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  while true do
  begin
    page(output);
    writeln('R   list file of real');
    writeln('P   list parameterfile');
    writeln('D   list datafle');
    writeln('E   exit');
    writeln('H   help');
    GoToXY(0, 10);
    write('> ');
    read(Ch);
    read(Ch1);
    if (Ch <> 'E') and (Ch <> 'H') then
      readln(St);
    St:=concat(St, '.DATA');
    case Ch of
      'R' : Look;
      'P' : ListPar;
      'D' : ListDat;
      'E' : exit(program);
      'H' : Help;
    end;
    close(Fil);
  end;
end.
```

```
program ParCalc;

uses AppleStuff, TurtleGraphics;

(* Shows the identified parameters from a file,
   and can also calculate meanvalue,
   parameter distance etc.

   Author : Per Persson
   Date : 04 - JULY - 1983 *)

var
  Apar : file of real;
  J    : integer;
  K    : integer;
  Ns   : integer;
  Mo   : integer;
  Cha  : char;
  Ch   : char;
  St   : string;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Dot(X, Y : integer);

(* Writes a dot on the screen at position x, y. *)

var D : boolean;
begin
  DrawBlock(D, 1, 0, 0, 1, 1, X, Y, 15);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure StringIt(X : real; Int : boolean; var St : string);

(* Converts a real number to a textstring. *)

var
  I : integer;
  J : integer;
  S : string;

begin
  St := ' ';
  if X < 0 then
    St := concat(St, '-');
  if abs(X) < 1 then
    St := concat(St, '0');
  X := abs(X);
  I := trunc(X);
  str(I, S);
  if I > 0 then
    St := concat(St, S);
  if not Int then
  begin
    X := X - trunc(X);
    I := trunc(10000 * X);
    str(I, S);
```

```
      if I < 1 then
        S := concat('000', S)
      else
        if I < 10 then
          S := concat('000', S)
        else if I < 100 then
          S := concat('00', S)
        else if I < 1000 then
          S := concat('0', S);
      St := concat(St, '.', S);
    end;
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure Ax(X, Y : integer; B : boolean; Val : real);

  (* Writes the number val at position x, y. This is
     necessary because APPLE in graphical Mode can not
     write a number on the screen. *)

  var
    St : string;

  begin
    MoveTo(X, Y);
    StringIt(Val, B, St);
    wstring(St);
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  procedure NewScr(K : integer);

  (* Writes a new form on the screen scaled from
     k*50 to (k + 5)*50 in x direction. The y -direction
     can be decided from the program. *)

  var
    I : integer;

  begin
    GrafMode;
    FillScreen(Black);
    for I := 20 to 270 do
    begin
      Dot(I, 10);
      Dot(I, 170);
    end;
    for I := 11 to 169 do
    begin
      Dot(20, I);
      Dot(270, I);
    end;
    Ax(13, 0, true, K * 50);
    Ax(103, 0, true, (K + 2) * 50);
    Ax(203, 0, true, (K + 4) * 50);
    Ax(251, 0, true, (K + 5) * 50);
    Dot(18, 10);
```

```
   Dot(19, 10);
   Dot(18, 90);;
   Dot(19, 90);
   Dot(18, 170);
   Dot(19, 170);
   Dot(20, 9);
   Dot(120, 9);
   Dot(220, 9);
   Dot(270, 9);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowAP;

(* Show A - parameters. *)

var
  I    : integer;
  J    : integer;
  L    : integer;
  Gain : integer;
  Yl   : real;
  Yh   : real;

begin
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.5);
  Get(Apar);
  Mo := trunc(Apar^ + 0.5);
  write('Lower y-axis limit: ');
  readln(Yl);
  write('Higher y-axis limit: ');
  readln(Yh);
  Gain := trunc(160 / (Yh - Yl));
  for I := 1 to Ns do
  begin
    if ((I Mod 250) = 0) or (I = 1) then
    begin
      if I <> 1 then
        readln(Ch);
      NewScr(I div 50);
      Moveto(30, 173);
      wstring('A-parameters');
      Ax(0, 10, false, Yl);
      Ax(0, 85, false, Yl + 0.5 *(Yh - Yl));
      Ax(0, 165, false, Yh);
    end;
    for J := 1 to Mo do
    begin
      Get(Apar);
      Dot((I Mod 250) + 20, trunc((Apar^ - Yl) * Gain) + 10);
    end;
  end;
  close(Apar);
  readln(Ch);
end;
```

```
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowPD;

(* Show parameter distance. *)

var
  Ar   : array [1..10] of real;
  I    : integer;
  Gain : integer;
  J    : integer;
  Pard : real;
  Yl   : real;
  Yh   : real;

begin
  page(output);
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.5);
  Get(Apar);
  Mo := trunc(Apar^ + 0.5);
  for I := 1 to Mo do
  begin
    write('ref a',I, ': ');
    readln(Ar[I]);
  end;
  write('Lower y axis limit:  ');readln(Yl);
  write('Higher y axis limit: ');readln(Yh);
  Gain := trunc(160 / (Yh - Yl));
  for I := 1 to Ns do
  begin
    if (I = 1) or ((I Mod 250)=0) then
    begin
      if I <> 1 then
        readln(Ch);
      NewScr(I div 50);
      Moveto(30, 173);
      wstring('Parameter distance');
      Ax(0, 10, false, Yl);
      Ax(0, 85, false, Yl + (Yh - Yl) / 2);
      Ax(0, 165, false, Yh);
    end;
    Pard := 0;
    for J := 1 to Mo do
    begin
      Get(Apar);
      Pard := Pard + (Apar^ - Ar[J]) * (Apar^ - Ar[J]);
    end;
    Dot((I Mod 250) + 20, trunc((Pard - Yl)*Gain) + 10);
  end;
  close(Apar);
  readln(Ch);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Mean;
```

```
(* Computes the mean value of the A - parameters in an interval. *)

var
 Ub : integer;
 Lb : integer;
 I  : integer;
 Mv      : array [1..10] of real;

begin
  for I := 1 to 10 do
    Mv[I] := 0;
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.2);
  Get(Apar);
  Mo := trunc(Apar^ + 0.2);
  writeln(Ns, ' points available');
  write('Lower time limit: ');
  readln(Lb);
  write('Higher time limit:');
  readln(Ub);
  for I := 1 to Lb - 1 do
    for J := 1 to Mo do
      Get(Apar);
  for I := Lb to Ub do
    for J := 1 to Mo do
    begin
      Get(Apar);
      Mv[J] := Mv[J] + Apar^;
    end;
  for I := 1 to Mo do
    writeln('mv a', I, ': ', Mv[I] / (Ub - Lb + 1));
  close(Apar);
  readln;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ListPar;

(* Lists the A - paramerters on the screen. *)

var
 I : integer;
 J : integer;

begin
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.2);
  Get(Apar);
  Mo := trunc(Apar^ + 0.2);
  writeln('Ns: ', Ns);
  writeln('Mo: ', Mo);
  readln;
  for I := 1 to Ns do
  begin
    writeln(I, ' :');
    for J := 1 to Mo do
    begin
      Get(Apar);
      write(Apar^:7:5);
    end;
    writeln;
    writeln;
  end;
  writeln;
  close(Apar);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowAPC;

(* Show A - parameters with a different x - axis scaling. *)

var
  I    : integer;
  J    : integer;
  L    : integer;
  Gain : integer;
  K    : real;
  Yh   : real;
  Yl   : real;

begin
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.5);
  Get(Apar);
  Mo := trunc(Apar^ + 0.5);
  if Ns < 250 then
  begin
    close(Apar);
    ShowAP;
  end
  else
  begin
    write('Lower y-axis limit: ');
    readln(Yl);
    write('Higher y-axis limit: ');
    readln(Yh);
    Gain := trunc(160 / (Yh - Yl));
    NewScr(0);
    Moveto(30, 173);
    wstring('A-parameters');
    Ax(0, 10, false, Yl);
    Ax(0, 85, false, Yl + 0.5 * (Yh - Yl));
    Ax(0, 165, false, Yh);
    Ax(13, 0, true, 0);
    Ax(103, 0, true, (Ns div 5) * 2);
    Ax(203, 0, true, (Ns div 5) * 4);
    Ax(251, 0, true, Ns);
    for I := 1 to Ns do
      for J := 1 to Mo do
      begin
        Get(Apar);
```

```
          K := I;
          K := K * 250.0;
          K := K / Ns;
          Dot(trunc(K) + 20, trunc((Apar^ - Yl) * Gain) + 10);
        end;
      close(Apar);
      readln(Ch);
    end;
  end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowPDC;

(* Show parameter distance with a different x - axis scaling. *)

var
  Ar   : array [1..10] of real;
  I    : integer;
  J    : integer;
  Gain : integer;
  Pard : real;
  K    : real;
  Yl   : real;
  Yh   : real;

begin
  page(output);
  reset(Apar, St);
  Ns := trunc(Apar^ + 0.5);
  Get(Apar);
  Mo := trunc(Apar^ + 0.5);
  if Ns > 250 then
  begin
    for I := 1 to Mo do
    begin
      write('ref a', I, ': ');
      readln(Ar[I]);
    end;
    write('Lower y Axis limit:  ');
    readln(Yl);
    write('Higher y axis limit: ');
    readln(Yh);
    Gain := trunc(160 / (Yh - Yl));
    NewScr(0);
    ax(13, 0, true, 0);
    Ax(103, 0, true, (Ns div 5) * 2);
    Ax(203, 0, true, (Ns div 5) * 4);
    Ax(241, 0, true, Ns);
    Moveto(30, 173);
    wstring('Parameter distance');
    Ax(0, 10, false, Yl);
    Ax(0, 85, false, Yl + (Yh - Yl) / 2);
    Ax(0, 165, false, Yh);
    for I := 1 to Ns do
    begin
      Pard := 0;
      for J := 1 to Mo do
      begin
        Get(Apar);
        Pard := Pard + (Apar^ - Ar[J]) * (Apar^ - Ar[J]);
      end;
      K := I;
      K := K * 250.0;
      K := K / Ns;
      Dot(trunc(K) + 20, trunc((Pard - Y1) * Gain) + 10);
    end;
    close(Apar);
    readln(Ch);
  end
  else
  begin
    close(Apar);
    ShowPD;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  Cha := 'B';
  write('Examine file: ');
  readln(St);
  St := concat(St, '.DATA');
  while Cha <> 'E' do
  begin
    TextMode;
    page(output);
    writeln('Available commands');
    writeln('1:  show a-parameters');
    writeln('2:  show parameter distance');
    writeln('3:  meanvalue of parameters');
    writeln('4:  list parameter values ');
    writeln('5:  show a-parameters compressed');
    writeln('6:  show parameter distance compressed');
    writeln('10: exit ');
    write('Command nr:');
    readln(K);
    case K of
      1 : ShowAP;
      2 : ShowPD;
      3 : Mean;
      4 : ListPar;
      5 : ShowAPC;
      6 : ShowPDC;
      10 : Cha := 'E';
    end;
  end;
end.
```

```
program PrSet;

(* Sets the epson printer in american mode.
   Author : Per Persson
   Date : 04 - JULY - 1984 *)

var
  Fil : file of char;

begin
  rewrite(Fil, 'PRINTER:');
  writeln(Fil, chr(27), chr(82), chr(0));
end.
```

```
program Sample;
uses AppleStuff, ClockStuff;

(*
  Collects data from max 16 channels.
  The sampletime is controlled with
  a delay loop and the variable m.

  Author : Per Persson

  Date : 83-july-04
*)


const
  Kilo = 1024;
  Slot = 5;

type
  Offsets = (a0,a1);
  AnVec   = packed array[a0..a1] of char;
  AdrType = (number,address);

var
  AdRec     : record
                case Select  : AdrType of
                  Number   : (NumAdr  : integer);
                  Address  : (PtrAdr  : ^AnVec) ;
                end;

  Conv      : record
                case integer of
                  1 : (Num  : integer);
                  2 : (Pac  : AnVec);
                end;
  SltPtr    : ^AnVec;
  SlotBas   : integer;
  Chnr      : integer;
  GainCode  : integer;
  Interv    : integer;
  I         : integer;
  M         : integer;
  Vxl       : integer;
  J         : integer;
  Nf        : integer;
  Ns        : integer;
  Ic        : integer;
  Noc       : integer;
  Mo        : integer;
  S1        : integer[10];
  S2        : integer[10];
  Re        : integer[10];
  Ti1       : integer[10];
  Mi1       : integer[10];
  Se1       : integer[10];
  Ti2       : integer[10];
  Mi2       : integer[10];
  Se2       : integer[10];
  Tid       : integer[12];
  Temp      : file of AnVec;
  Dat       : file of real;
  Pri       : file of char;
  InFile    : file of char;
  OutFile   : file of char;
  Chan      : array [0..15] of
                record
                  Collect  : boolean;
                  GainCode : integer;
                end;
  Range     : array [0..7,1..2] of real;
  Vxf       : string;
  Sg        : string;
  St        : string;
  Ch        : char;
  Idp       : boolean;
  Trans     : boolean;
  Print     : boolean;
  Fast      : boolean;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ADSlot;

(* Initializes variables and hardware AD conversion *)

var
 I : integer;

begin


  SlotBas := 48 * Kilo + 16 * (8 + Slot);   (* Tricky Pascal-programming *)
  AdRec.Select := Number;
  AdRec.NumAdr := SlotBas;
  AdRec.Select := Address;
  SltPtr := AdRec.PtrAdr;


  Range[0,1] := 5;
  Range[0,2] := 0;
  Range[1,1] := 1;
  Range[1,2] := 0;
  Range[2,1] := 0.5;
  Range[2,2] := 0;
  Range[3,1] := 0.1;
  Range[3,2] := 0;
  Range[4,1] := 10;
  Range[4,2] := 5;
  Range[5,1] := 2;
  Range[5,2] := 1;
  Range[6,1] := 1;
  Range[6,2] := 0.5;
```

```
   Range[7,1] := 0.2;
   Range[7,2] := 0.1;
   rewrite(Pri, 'PRINTER:');
   rewrite(InFile, 'REMIN:');
   rewrite(OutFile, 'REMOUT:');
   for I := 0 to 15 do
   begin
     Chan[I].Collect := false;
     Chan[I].GainCode := 0;
   end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure SlowScan;

(* For sampling in slow mode *)
(* The Bit-Pattern of the signal is saved on a  *)
(* temorary file Temp *)

var
 Val : integer;
 J   : integer;

begin
   for J := 0 to 15 do
     if Chan[J].Collect then
     begin
       Val := Chan[J].GainCode;
       SltPtr^[A0] := chr(J + 16 * Val);
       Temp^ := SltPtr^;
       Put(Temp);
     end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure FastScan;

(* For sampling in fast mode *)
(* The Bit-Pattern of the signal is saved on a  *)
(* temorary file Temp *)

var
   J : integer;

begin
   for J := 0 to Noc - 1 do
   begin
     SltPtr^[A0] := chr(J + 16 * GainCode);
     Temp^ := SltPtr^;
     Put(Temp);
   end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Pack(St : string);

(* Reads the temporary file Temp and converts it
(* a 'file of real' *)
(* this is done to speed up the sampling-rate *)

var
  I  : integer;
  J  : integer;
  K  : integer;
  R1 : real;
  R2 : real;

begin
  St := concat(St, '.DATA');
  if not Fast then
  begin
    Noc := 0;
    for I := 0 to 15 do
      if Chan[I].Collect then
        Noc := Noc + 1;
  end;
  rewrite(Dat, St);
  reset(Temp);
  I := 2;     (* default systemorder *)
  Dat^ := Ns;
  Put(Dat);
  Dat^ := Noc;
  Put(Dat);
  Dat^ := I;
  Put(Dat);
  if Fast then
  begin
    for J := 0 to Noc - 1 do
    begin
      Dat^ := J;
      Put(Dat);
      Dat^ := GainCode;
      Put(Dat);
      Put(Dat);
    end;
    R1 := Range[GainCode,1];
    R2 := Range[GainCode,2];
    for I := 1 to Ns do
      for J := 0 to Noc - 1 do
      begin
        Conv.Pac := Temp^;
        Dat^ := Conv.Num;
        Dat^ := (Dat^ * R1) / 4096;
        Dat^ := Dat^ - R2;
        Put(Dat);
        Get(Temp);
      end;
  end
  else
  begin
    for J := 0 to 15 do
      if Chan[J].Collect then
      begin
```

```
        Dat^ := J;
        Put(Dat);
        Dat^ := Chan[J].GainCode;
        Put(Dat);
        Put(Dat);
      end;
    for I := 1 to Ns do
      for J := 0 to 15 do
        if Chan[J].Collect then
        begin
          K := Chan[J].GainCode;
          Conv.Pac := Temp^;
          Dat^ := Conv.Num;
          Dat^ := (Dat^ * Range[K,1]) / 4096;
          Dat^ := Dat^ - Range[K,2];
          Put(Dat);
          Get(Temp);
        end;
  end;
  close(Dat, lock);
  close(Temp);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Vx(Ch : char);

(* Sends the character Ch to the VAX *)
(* and echoes it on the screen *)

begin
  OutFile^ := Ch;
  Put(OutFile);
  if Ch = chr(27) then
    writeln
  else
    write(Ch);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
function Gt : char;

(* Get one character from the VAX *)

begin
  Get(InFile);
  Gt := InFile^;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure SetUp;

var
  I   : integer;
  J   : integer;
  Ch1 : char;
  Ch2 : char;
  Out : boolean;
  A   : array [1..14] of
```

```
          packed array [1..2] of char;

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 procedure DeCode;

 procedure ReadChan;
 begin
   read(I);
   if (I < 16) and (i > -1) then
   begin
     Chan[I].Collect := true;
     readln(GainCode);
     Chan[I].GainCode := GainCode;
   end
     else if I >= 100 then
       Chan[I - 100].Collect := false;
   GoToXY(0, 10);
   writeln('                                                  ');
   GoToXY(0, 10);
   for I := 0 to 15 do
     if Chan[I].Collect then
       write(' ', I, ', ', Chan[I].GainCode);
   writeln;
   GoToXY(2, 21);
   writeln('              ');
 end;

 procedure ReadLog(var B : boolean; Pos : integer);
 begin
   read(Ch1);
   readln(Ch1);
   if Ch1 = 'Y' then
     B := true
   else
     B := false;
   GoToXY(23, Pos);
   if B then
     writeln('Yes')
   else
     writeln('No ');
 end;

 procedure ReDelWr(var N : integer; Pos : integer);
 begin
   readln(N);
   GoToXY(23, Pos);
   writeln('      ');
   GoToXY(23, Pos);
   writeln(N);
 end;

 procedure ReadStr(var S : string; Pos : integer);
 begin
   read(Ch1);
   readln(S);
```

```
    GoToXY(23, Pos);
    writeln('                  ');
    GoToXY(23, Pos);
    writeln(S, '*');
  end;

  begin
    case J of
       1  : ReadStr(St, 0);
       2  : begin
              ReadStr(Vxf, 1);
              Vxl := length(Vxf);
            end;
       3  : ReDelWr(Ns, 2);
       4  : ReDelWr(M, 3);
       5  : ReDelWr(Nf, 4);
       6  : ReadLog(Print, 5);
       7  : ReDelWr(Interv, 6);
       8  : ReadLog(Idp, 7);
       9  : ReadLog(Trans, 8);
       10 : ReadChan;
       11 : begin
              Out := false;
              GoToXY(23, 11);
              writeln('Executing');
            end;
       12 : ReDelWr(GainCode, 10);
       13 : exit(program);
       14 : begin
              readln(Noc);
              if (Noc < 17) and (Noc > 0) then
              begin
                GoToXY(23, 9);
                writeln('      ');
                GoToXY(23, 9);
                writeln(Noc);
              end;
            end;
       15 : begin end;
     end;
   end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Header;
begin
  if Print then
  begin
    writeln(Pri, '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    ReadTime;
    writeln(Pri, 'Start at: ', Month, '-', Date);
    writeln(Pri, '           ', Hours, ':', Minutes, ':', Seconds);
    writeln(Pri, 'Applefiles: ', St, '*');
    if Trans then
      writeln(Pri, 'Vaxfiles  : ', Vxf, '*');
    writeln(Pri, Nf, ' files will be created.');
    writeln(Pri, '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    end;
  end;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  begin
    page(outPut);
    writeln('AF applefile:         ');
    A2[1] := 'AF';
    writeln('VF vaxfile:           ');
    A2[2] := 'VF';
    writeln('NS number of samples:');
    A2[3] := 'NS';
    writeln('DE delay:             ');
    A2[4] := 'DE';
    writeln('NF number of files:   ');
    A2[5] := 'NF';
    writeln('PR printer:           ');
    A2[6] := 'PR';
    writeln('CI collectinterval:   ');
    A2[7] := 'CI';
    writeln('ID idpac:             ');
    A2[8] := 'ID';
    writeln('TR transmit:          ');
    A2[9] := 'TR';
    if Fast then
    begin
      writeln('NC number of channels:');
      A2[14] := 'NC';
      writeln('GC GainCode:          ');
      A2[12] := 'GC';
    end
    else
    begin
      writeln('IC inchannel:         ');
      A2[10] := 'IC';
      writeln;
    end;
    writeln('EX execute:           ');
    A2[11] := 'EX';
    A2[13] := 'QU';
    Out := true;
    while Out do
    begin
      GoToXY(0, 15);
      writeln('                           ');
      GoToXY(0, 15);
      write('> ');
      repeat
        read(Ch1);
      until Ch1<>' ';
      read(Ch2);
      J := 15;
      for I := 1 to 14 do
        if (Ch1 = A2[I][1]) and (Ch2 = A2[I][2]) then
          j := i;
      DeCode;
  end;
  Header;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Transmit;

(* Transmit a file to the VAX *)

var
  I  : integer;
  Ch : char;
  S  : packed array [1..80] of char;

(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

function Dig(I : integer) : char;

begin
  Dig := chr(ord('0') + I);
end;

(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure SendVax;

var
  I : integer;

begin
  for I := 1 to 8 do
  begin
    OutFile^ := S[I];
    Put(OutFile);
  end;
  OutFile^ := chr(27);
  Put(OutFile);
end;

(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

procedure Convert(R : real);

var
  L : real;
  P : real;
  I : integer;
  J : integer;
  K : integer;

begin
  I := 1;
  if R < 0 then
  begin
    S[I] := '-';
    I := I + 1;
    end;
    J := trunc(abs(R));
    if J < 1 then
    begin
    end
    else if J < 10 then
    begin
      S[I] := Dig(J);
      I := I + 1;
    end
    else if J < 100 then
    begin
      S[I] := Dig(J div 10);
      I := I + 1;
      S[I] := Dig(J mod 10);
      I := I + 1;
    end
    else
      writeln('Something is wrong.');
    S[I] := '.';
    I := I + 1;
    L := abs(R) - trunc(abs(R));
    while I < 9 do
    begin
      L := L * 10;
      K := trunc(L);
      S[I] := Dig(K);
      I := I + 1;
      P := K;
      L := L - P;
    end;
  end;

  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  begin
    reset(Dat, concat(St, Sg, '.DATA'));
    Get(Dat);
    Get(Dat);
    for I := 1 to Noc * 3 do
      Get(Dat);
    GoToXY(0, 17);
    writeln('                       ');
    ReadTime;
    if Print then
    begin
      writeln(Pri, Hours, ':', Minutes, ':', Seconds);
      writeln(Pri, '     Vaxfile created');
    end;
    GoToXY(0, 17);
    write('$ ');
    vx('C');
    vx('R');
    vx('E');
    vx('A');
  vx('T');
  vx('E');
  vx(' ');
  for I := 1 to vxl do
    Vx(Vxf[I]);
  if Ic < 10 then
   Vx(Dig(Ic))
  else
  begin
    Vx(Dig(Ic div 10));
    Vx(dig(Ic mod 10));
  end;
  Vx('.');
  Vx('T');
  Vx(chr(27));
  repeat
    Ch := Gt;
  until Ch='$';
  for I := 1 to Noc * Ns do
  begin
    Get(Dat);
    Convert(Dat^);
    GoToXY(0, 18);
    SendVax;
    repeat
      Ch := Gt;
    until Ch='$';
  end;
  Vx(chr(90 - 64));
  repeat
    Ch := Gt;
  until Ch='$';
  GoToXY(0, 17);
  writeln('                       ');
  GoToXY(0, 17);
  writeln('Control-Z');
  ReadTime;
  if Print then
  begin
    writeln(Pri, Hours, ':', Minutes, ':', Seconds);
    writeln(Pri, 'Applefile ', concat(St, Sg, '.data'), ' transmitted to VA
    write(Pri,   '     Vaxfile: ', Vxf);
    if Ic < 10 then
      write(Pri, Dig(Ic))
    else
    begin
      write(Pri, Dig(Ic div 10));
      write(Pri, Dig(Ic mod 10));
    end;
    writeln(Pri, '.T');
  end;
  close(Dat);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Idpac;
```

```
(* Starts a command-procedure on the Vax which starts IDPAC *)

var
  I  : integer;
  Ch : char;

 function Dig(I : integer) : char;

 begin
   Dig := chr(ord('0') + I);
 end;

begin
  write('$ ');
  Vx('');
  Vx('I');
  Vx('D');
  Vx('P');
  Vx(' ');
  for I := 1 to Vxl do
    Vx(Vxf[I]);
  if Ic < 10 then
    Vx(Dig(Ic))
  else
  begin
    Vx(Dig(Ic div 10));
    Vx(Dig(Ic mod 10));
  end;
  Vx('.');
  Vx('T');
  Vx(chr(27));
  repeat
    Ch := Gt;
  until Ch='$';
  writeln(Ch, Gt);
  ReadTime;
  if Print then
  begin
    writeln(Pri, Hours, ':', Minutes, ':', Seconds);
    writeln(Pri, '     Idpac called');
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure GetData;

(* Carries out the sampling *)

var
  I : integer;
  J : integer;

begin
  ReadTime;
  Ti1 := Hours;
  Mi1 := Minutes;
  Se1 := Seconds;
  Re := ReadMs;
  S1 := Re + Se1 * 1000 + Mi1 * 60000 + Ti1 * 3600000;
  for J := 1 to Ns do
  begin
    for I := 1 to M do
      begin end;
    if Fast then
      FastScan
     else
       SlowScan;
  end;
  ReadTime;
  Ti2 := Hours;
  Mi2 := Minutes;
  Se2 := Seconds;
  Re := ReadMs;
  S2 := Re + Se2 * 1000 + Mi2 * 60000 + Ti2 * 3600000;
  if Print then
  begin
    writeln(Pri);
    writeln(Pri, 'File number: ', Ic, ' of ', Nf);
    writeln(Pri, Ti1, ':', Mi1, ':', Se1);
    writeln(Pri, '     Sampling started');
    writeln(Pri, Ti2, ':', Mi2, ':', Se2);
    writeln(Pri, '     Sampling finished');
    writeln(Pri, '     Sample interval: ', (S2 - S1) div Ns, ' ms');
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  page(outPut);
  write('Fast or slow scan? ');
  readln(Ch);
  if Ch = 'F' then
    Fast := true
  else
    Fast := false;
  AdSlot;
  ClockSlot(3);
  SetUp;
  for Ic := 1 to Nf do
  begin
    rewrite(Temp, 'SIM:TEMP.DATA');
    repeat
      ReadTime;
      GoToXY(0, 16);
      writeln(Hours, ':', Minutes, ':', Seconds);
      Tid := Hours * 60 + Minutes;
    until ((Tid div Interv) * Interv - Tid) = 0;
    GetData;
    GoToXY(0, 20);
    writeln('Sample interval: ', (S2 - S1) div Ns, ' ms');
    str(Ic, Sg);
    Pack(concat(St, Sg));
    if Trans then
      Transmit;
    if Trans and Idp then
      Idpac;
    if Print then
    writeln(Pri, '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  end;
  if Print then
  begin
    writeln(Pri);
    writeln(Pri);
    writeln(Pri);
    writeln(Pri);
    writeln(Pri);
  end;
end.
```

```
program Stat;
uses TurtleGraphics;

(*  Computes statistics from a datafile
Plots histograms etc.

Author : Per Persson

Date : 04-JUL-1984 *)

var
  Data   : file of real;
  A      : array [0..15,0..19] of integer;
  Ns     : integer;
  Mo     : integer;
  Noc    : integer;
  Noc    : integer;
  ChVec  : array [0..15] of
             record
               Collect : boolean;
               Sd      : real;
               Mv      : real;
               Kvs     : real;
               Max     : real;
               Min     : real;
             end;
  Ch     : char;
  St     : string;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Dot(X, Y : integer);

(* Plots a dot at position x y. *)

var
  D : boolean;

begin
  DrawBlock(D, 1, 0, 0, 1, 1, X, Y, 15);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure StringIt(X : real; Int : boolean; var St : string);

(* Converts a decimal number to a textstring. *)

var
  J : integer;
  I : integer;
  S : string;

begin
  St := ' ';
  if X < 0 then
    St := concat(St, '-');
  if abs(X) < 1 then
    St := concat(St, '0');
```

```
   X := abs(X);
   I := trunc(X);
   str(I, S);
   if I > 0 then
     St := concat(St, S);
   if not Int then
   begin
     X := X - trunc(X);
     I := trunc(10000 * X);
     str(I, S);
     if I < 1 then
       S := concat('000', S)
     else if I < 10 then
        S := concat('000', S)
     else if I < 100 then
        S := concat('00', S)
     else if I < 1000 then
        S := concat('0', S);
     St := concat(St, '.', S);
   end;
 end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Ax(X, Y : integer; B : boolean; Val : real);

(* Writes a number at a given position on the screen when it is
graphic mode *)

var
  St : string;

begin
  MoveTo(X, Y);
  StringIt(Val, B, St);
  wstring(St);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure StartFile;

(* Opens a file rand reads its head. *)

var
  I : integer;

begin
  reset(Data, St);
  Ns := trunc(Data^ + 0.5);
  Get(Data);
  Noc := trunc(Data^ + 0.5);
  Get(Data);
  Mo := trunc(Data^ + 0.5);
  for I := 1 to Noc do
  begin
    Get(Data);
    ChVec[trunc(Data^ + 0.5)].Collect := true;
    Get(Data);
    Get(Data);
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Init;

(* Initializes variables used in the statistical routines. *)

var
  I : integer;
  J : integer;

begin
  write('Name of data file: ');readln(St);
  St := concat(St, '.DATA');
  for I := 0 to 15 do
  begin
    ChVec[I].Max := -1e10;
    ChVec[I].Min := 1e10;
    ChVec[I].Collect := false;
    ChVec[I].Mv := 0;
    ChVec[I].Kvs := 0;
  end;
  for I := 0 to 15 do
    for J := 0 to 19 do
      A[I,J] := 0;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Statistics;

(* Computes statistics. *)

var
  Hlp : real;
  I   : integer;
  J   : integer;

begin
  for I := 1 to Ns do
  begin
    for J := 0 to 15 do
      if ChVec[J].Collect then
      begin
        with ChVec[J] do
        begin
          Get(Data);
          if Data^ > Max then
            Max := Data^;
          if Data^ < Min then
            Min := Data^;
          Mv := Mv + Data^;
          Kvs := Kvs + Data^ * Data^;
        end;
      end;
  end;
```

```
   for I := 0 to 15 do
     if ChVec[I].Collect then
     begin
       ChVec[I].Mv := ChVec[I].Mv / Ns;
       Hlp := ChVec[I].Mv * ChVec[I].Mv / Ns;
       Hlp := ChVec[I].Kvs - Hlp;
       Hlp := Hlp / (Ns - 1);
       ChVec[I].Sd := Hlp;
     end;
 end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 procedure Count;

 var
   Delta : array [0..15] of real;
   I     : integer;
   J     : integer;
   H     : integer;

 begin
   for I := 0 to 15 do
     Delta[I] := (ChVec[I].Max - ChVec[I].Min) / 19;
   for I := 1 to Ns do
     for J := 0 to 15 do
       if ChVec[J].Collect then
       begin
         Get(Data);
         H := trunc((Data^ - ChVec[J].Min) / Delta[J]);
         A[J,H] := A[J,H] + 1;
       end;
 end;
 (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 procedure Histogram(I : integer);

 (* Computes and plots a histogram of channel i. *)

 var
   J   : integer;
   H   : integer;
   Mx  : integer;
   Pl  : integer;
   Ph  : integer;
   Px  : integer;
   Hlp : real;
   Sr  : string;
   Ch  : char;

 begin
   if ChVec[I].Collect then
   begin
     Mx := 0;
     GrafMode;
     FillScreen(Black);
     for J := 50 to 250 do
       Dot(J, 20);
```

```
    for J := 20 to 180 do
    begin
      Dot(50, J);
      Dot(250, J);
    end;
    str(I, Sr);
    Sr := concat('Channelnumber:  ', Sr);
    MoveTo(50, 182);
    wstring(Sr);
    for J := 0 to 19 do
    begin
      if A[I,J] > Mx then
        Mx := A[I,J];
      Dot(J * 10 + 50, 19);
      Dot(J * 10 + 50, 18);
    end;
    Dot(250, 19);
    Dot(250, 18);
    Ax(20, 20, true, 0);
    Ax(20, 175, true, Mx);
    Ax(30, 0, false, ChVec[I].Min);
    Ax(230, 0, false, ChVec[I].Max);
    for J := 0 to 10 do
    begin
      Dot(48, J * 16 + 20);
      Dot(49, J * 16 + 20);
    end;
    for J := 0 to 19 do
    begin
      if (J < 19) then
      begin
        Hlp := A[I, J];
        Hlp := Hlp * 160.0 / Mx;
        Pl := trunc(Hlp);
        Hlp := A[I, J+1];
        Hlp := Hlp * 160.0 / Mx;
        Ph := trunc(Hlp);
        if A[I,J] < A[I,J+1] then
          for Px := Pl to Ph do
            Dot((J + 1) * 10 + 50, Px + 20)
        else
          for Px := Pl downto Ph do
            Dot((J + 1) * 10 + 50, Px + 20);
      end;
      Pl := J * 10;
      Ph := Pl + 10;
      Hlp := A[I,J];
      Hlp := Hlp * 160.0 / Mx;
      Px := trunc(Hlp);
      for H := Pl to Ph do
        Dot(H + 50, Px + 20);
    end;
    readln(Ch);
    TextMode;
  end;
```

```
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure ShowStat(I : integer);

var
  Ch : char;

begin
  with ChVec[I] do
  begin
    if Collect then
    begin
      writeln;
      writeln('Channel:    ', I);
      writeln('Max-value:  ', Max:9:4);
      writeln('Min-value:  ', Min:9:4);
      writeln('Mean value:', Mv:9:4);
      writeln('Variance:   ', Sd:9:4);
      writeln;
      readln(Ch);
    end;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  Init;
  StartFile;
  Statistics;
  close(Data);
  StartFile;
  Count;
  close(Data);
  while true do
  begin
    page(output);
    writeln('H <chnr> histogram of channel chnr');
    writeln('S <chnr> statistics of channel chnr');
    writeln('A statistics of all channels');
    writeln('E exit');
    writeln('available channels:');
    for C := 0 to 15 do
      if ChVec[C].Collect then
        write(' ', C);
    writeln;
    writeln;
    write('>');
    read(Ch);
    if (Ch='H') or (Ch='S') then
      readln(C);
    case Ch of
     'H' : Histogram(C);
     'S' : ShowStat(C);
     'E' : exit(program);
     'A' : for C := 0 to 15 do
             if ChVec[C].Collect then
```

```
                  ShowStat(C);
        end;
      end;
    end.
```

```
program Vax;

uses AppleStuff;

(*  Transfers a datafile generated  by SAMPLE to VAX.
    Author : Per Persson
    Date : 83-july-04  *)

var
  Fil     : file of real;
  InFile  : file of char;
  OutFile : file of char;
  Ch      : char;
  St      : packed array [1..80] of char;
  L       : integer;
  I       : integer;
  Ns      : integer;
  Noc     : integer;
  S1      : string;
  S2      : string;
  S       : string;

(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Vx(Ch : char);

(* Writes a character on the port REMOUT: *)

begin
  OutFile^ := Ch;
  Put(OutFile);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
function Gt : char;

(* Reads a character from the port REMIN: *)

begin
  Get(InFile);
  Gt := InFile^;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Convert(R : real);

(* Converts a real number in the interval  ]-100,100[
   to a ASCII text string. Eight characters are used.  *)

var
  L : real;
  P : real;
  I : integer;
  J : integer;
  K : integer;

  function Dig(I : integer) : char;
```

```
   (* Convert a digit to its ASCII equivalent. *)

   begin
     Dig := chr(ord('0') + I);
   end;

begin
  I := 1;
  if R < 0 then
  begin
    St[I] := '-';
    I := I + 1;
  end;
  J := trunc(abs(R));
  if J < 1 then
  begin
  end
  else if J < 10 then
  begin
    St[I] := Dig(J);
    I := I + 1;
  end
  else if J < 100 then
  begin
    St[I] := Dig(J div 10);
    I := I + 1;
    St[I] := Dig(J mod 10);
    I := I + 1;
  end;
  St[I] := '.';
  L := abs(R) - trunc(abs(R));
  I := I + 1;
  while I < 9 do
  begin
    L := L * 10;
    K := trunc(L);
    St[I] := Dig(K);
    I := I + 1;
    P := K;
    L := L - P;
  end;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure SendVax;

(* Send the number string to the VAX from the port REMOUT: *)

var
  I : integer;

begin
  for I := 1 to 8 do
    Vx(St[I]);
  Vx(chr(27));  (* Use 'ESCAPE' as line terminator *)
end;
```

```
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  writeln('Transmission of datafiles.');
  writeln('APPLE - > VAX');
  rewrite(InFile, 'REMIN:');
  rewrite(OutFile, 'REMOUT:');
  while true do
  begin
    write('Transmit file without extension: ');
    readln(S1);
    write('VAX - file with extension        : ');
    readln(S2);
    S1 := concat(S1, '.DATA');
    reset(Fil, S1);
    Ns := trunc(Fil^ + 0.5);
    Get(Fil);
    Noc := trunc(Fil^ + 0.5);
    Get(Fil);
    write('$ ');  (* Use the VMS command CREATE to create a file on the VAX
    Vx('C');
    Vx('R');
    Vx('E');
    Vx('A');
    Vx('T');
    Vx('E');
    Vx(' ');
    for I := 1 to length(S2) do
      Vx(S2[I]);
    Vx(chr(27));  (* Use 'ESCAPE' as line terminator *)
    repeat
      Ch := Gt;
    until Ch='$';
    for I := 1 to 3 * Noc do
      Get(Fil);
    for I := 1 to Noc * Ns do
    begin
      Get(Fil);
      Convert(Fil^);
      SendVax;
      repeat
        Ch := Gt;
      until Ch='$';
    end;
    Vx(chr(90 - 64)); (* Close the VAX file with a control-z *)
    writeln('Control-Z');
    close(Fil);
  end;
  close(InFile);
  close(OutFile);
end.
```

```
program VaxCom;

(* Sets up communication between APPLE and VAX.
   This program allows the apple to be used as
   a simple terminal. The vax/vms command
      SET TERMINAL/SPEED=300/NOPARITY
   must be given before the program can be used.
   To terminate the program give the command 'slut'.
   Author : Per Persson
   Date : 04 - JULY - 1984 *)

var
  InFile  : file of char;
  OutFile : file of char;
  Esc     : char;
  Ch      : char;
  L       : integer;
  I       : integer;
  J       : integer;
  S       : string;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure Vx(Ch : char);

(* Puts a character on the port REMOUT: *)

begin
  OutFile^ := Ch;
  put(OutFile);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
function Gt : char;

(* Gets a character from the port REMIN: *)

begin
  Get(InFile);
  Gt := InFile^;
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure GetCom;

(* Reads a line from the keyboard and puts it in
   a string *)

begin
  readln(S);
  if S = 'SLUT' then
    exit(program);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
procedure SendVax;

(* Sends a string on the port REMOUT: *)

var
```

```
  I : integer;

begin
  for I := 1 to length(S) do
    Vx(S[I]);
  Vx(Esc);
end;
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
begin
  Esc := chr(27);
  rewrite(InFile, 'REMIN:');
  rewrite(OutFile, 'REMOUT:');
  Vx(Esc);                          (* Use 'ESCAPE as line terminator.   *)
                                    (* APPLE and VAX interprets CR       *)
                                    (* differiently.                     *)

  Ch := Gt;                         (* Search for a the VAX-prompt '$'   *)
  repeat
    Ch := Gt;
  until Ch = '$';
  write(Ch, Gt);

  while true do
  begin
    GetCom;                         (* Get a commandline       *)
    writeln;
    SendVax;                        (* Send it to VAX *)
    repeat                          (* Take care of the echo.  *)
      Ch := Gt;
    until Ch = '$';
    repeat                          (* Write the output from VAX on the termina
      Ch := Gt;
      if Ch = chr(10) then
        writeln
      else
        write(Ch);
    until Ch = '$';
    Ch := Gt;
    write(Ch);
  end;
  close(InFile);
  close(OutFile);
end.
```