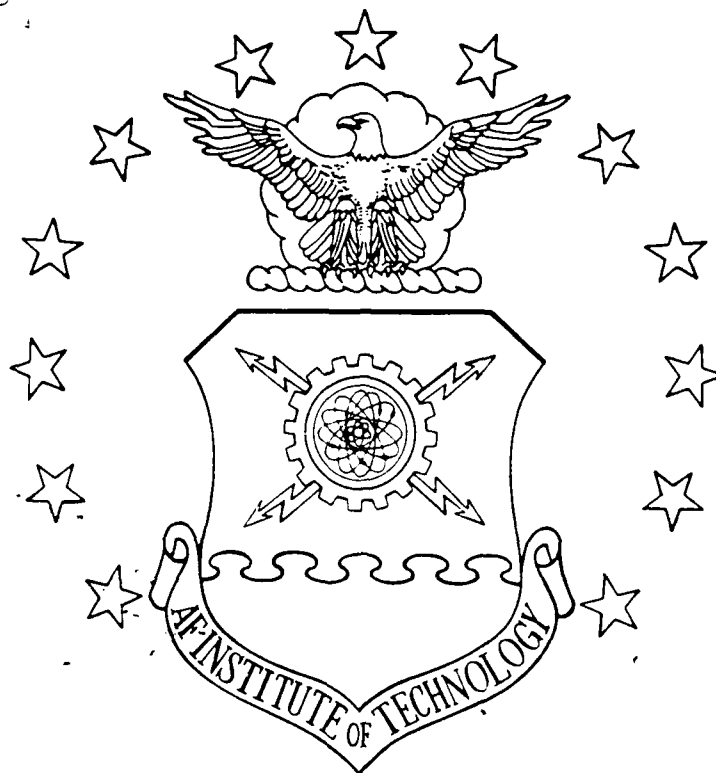


DTIC FILE COPY

1

AD-A220 472



AN INTERACTIVE LIFE CYCLE COST
FORECASTING TOOL

THESIS

David L. Sumner
First Lieutenant, USAF

AFIT/GOR/ENS/90M-17

SD DTIC
ELECTE
APR 16 1990
Op **B** **D**

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

90 04 13 201

AFIT/GOR/ENS/90M-17

AN INTERACTIVE LIFE CYCLE COST FORECASTING TOOL

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

David L. Sumner, B.S.

First Lieutenant, USAF



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

March 1990

Approved for public release; distribution unlimited

Preface

I chose this research topic because of my interest in Monte Carlo simulation and statistical description. I am indebted to Dr. J.P. Cain for presenting such an interesting opportunity, and for giving me the economics background to deal with it.

Also, sincerest thanks are in order to Maj Ken Bauer for helping me to ensure statistical integrity in all the sampling and descriptive procedures.

I am forever indebted to wife, Tammy, and my family for their love and support.

David L. Sumner

Table of Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vii
List of Abbreviations	viii
Abstract	ix
I. Introduction	1
Background	1
Objective	4
Subobjectives	5
II. Literature Review	6
Introduction	6
Chapter Overview	6
LCC Modeling	7
Parametric CER	7
Engineering	8
Accounting	11
Random Number Generation	12
General Sampling Techniques	13
Specific Generation Techniques	17
III. Methodology	22
Model Definition	22
CER's	22
Single Variables	24
Monte Carlo Methodology	24
CER's	24
Single Variable	25
Time Phase Spending	25
Output Analysis	28
Frequency Histogram	28
Moment Estimation	29
Quantile Estimation	30
Non-Parametric Probabilities	30
T-Tests	31
Non-Parametric Testing	32

IV. Implementation	35
Random Variables	35
Beta Sampling	35
Normal Sampling	36
Uniform Sampling	37
Multivariate Normal Sampling	37
Cost Estimation	38
Power Transformations	40
Time Phased Spending	42
Software Production	44
Documentation	45
V. Conclusions and Recommendations	46
Appendix A: Users' Manual	49
Appendix B: Random deviate generator output	80
Appendix C: Turbo Pascal 4.0 source code.	93
Bibliography	163
Vita	165

List of Figures

Figure	Page
1. Department of Defense 1979 Budget	4
2. Parametric CER model example	9
3. Engineering model example	10
4. Accounting model example	12
5. Spending Timelines for cost components	27
6. Example of a Histogram	29
7. Input uncertainty probability distributions . . .	36
8. Present value for time phased spending calculations	43
9. Main menu screen	54
10. Menu layout tree	55
11. Available distributions	59
12. Example of MOMENTS output	61
13. Example of HISTOGRAM output	63
14. TIME SERIES plot output	64
15. Example of XYPLOT output	65
16. Spending timeline for Cost 1	73
17. Spending timeline for Cost 2	73
18. Spending timeline for Cost 3	74
19. Spending timeline for Cost 4	74
20. Uniform deviate histogram	82
21. Uniform moments	83
22. Uniform deviate serial correlations	84
23. Normal deviate histogram	85

24.	Normal deviate moments	86
25.	Normal deviate serial correlations	87
26.	Beta deviate histogram	89
27.	Beta deviate moments	90
28.	Beta deviate serial correlations	91

List of Tables

Table	Page
1. Example time parameters	75

List of Abbreviations

APR - Annual percentage rate
CDF - Cumulative Density Function.
CER - Cost Estimating Relationship
CI - Confidence Interval
CON - Length of constant cost period
CR - Carriage return
GLD - Generalized Lambda Distribution
LCC - Life Cycle Cost
LRU - Line replaceable unit
MSE - Mean Square Error
PC - Personal Computer
PDF - Probability Density Function
PI - Length of Phase in period
PO - Length of Phase out period
PV - Present value

Abstract

A tool was developed for Monte Carlo simulation of life cycle costs using parametric cost modeling. Additionally, the analysis of the performance of parametric CER cost estimation has been cut down to a more manageable task. Models can be built and tested quickly and easily.

Random deviate generators were researched and built. Several applicable statistical description routines were also implemented. Statistical integrity and great accuracy has been maintained, while made accessible through an intuitive, user friendly interface.

AN INTERACTIVE LIFE CYCLE COST FORECASTING TOOL

I. Introduction

Background

Each year the United States government spends billions of dollars for manpower and equipment to protect and preserve the nation. As important as national defense is, it is only one of many ways the government serves the people. Programs such as education, transportation, and human assistance are also necessary parts of government spending.

Unfortunately the government is not endowed with unlimited resources. Each program is in constant competition with others for funding. Indeed a large part of our political system is dedicated to the parsing and distribution of tax dollars. Since the government is tasked with performing many services with limited resources, it is compelled to get the most from each tax dollar spent. Since many of the dollars spent on national defense go toward acquiring new weapon systems, the government should buy the least expensive piece of hardware capable of doing the job, or buy the best piece of hardware while staying within the budgetary constraints. The latter case, maximizing

efficiency can be demonstrated by the simple optimization problem below:

$$\text{Max } E = F(A,B) \quad (1)$$

$$\text{st. } C = C_A + C_B \quad (2)$$

Where C_A and C_B are cost functions, linear or nonlinear, for Systems A and B. These cost functions may be based on quantity purchased. C is a budgetary constraint, and E is some production function representing the combined performance of systems A and B. Note that this is for a given configuration. The E represents effectiveness and is equated to some function of the quantity of systems A and B purchased.

Either strategy, maximizing effectiveness or minimizing cost, leads to some type of cost comparison among the proposed systems. The problem is that new weapon systems do not come off the shelf with clear cut price tags. The cost of each program must be estimated, then compared with the other programs in question. It is easy to see that realized efficiency depends greatly on the quality of the cost estimation.

Fisher, as well as others, has suggested that to properly estimate the budgetary impact of a particular system, all phases of the program must be examined (Fisher:66). Purchase price alone does not constitute the

system cost. There are research and development, testing, procurement, operation, and maintenance phases that must be considered. All spending associated with a program is called the life cycle cost (LCC). The DoD Life Cycle Costing Guide For System Acquisitions defines LCC.

The LCC system is the total cost to the Government of acquisition and ownership of that system over its full life. It includes the cost of development, acquisition, operation, support and where applicable, disposal. (DoD:1-1)

Life cycle costing has several advantages over simple purchase price estimation. Since life cycle costing includes all the phases of the program's life, a more realistic look at the budgetary effect is achieved. Figure 1 demonstrates the large portion of DoD funds used for operation and support, costs not included in the purchase price. It is very possible that system A may have a lower purchase price than system B, but have such a large manning and support requirement that these costs overwhelm the purchase price and make B a more economical choice.

Since the stages of the program occur chronologically, LCC also allows for the "timing" of the money spent. Banks exist and flourish all over the world profiting almost exclusively from the time value of money. So the timing of money spent by competing programs can be a very significant factor.

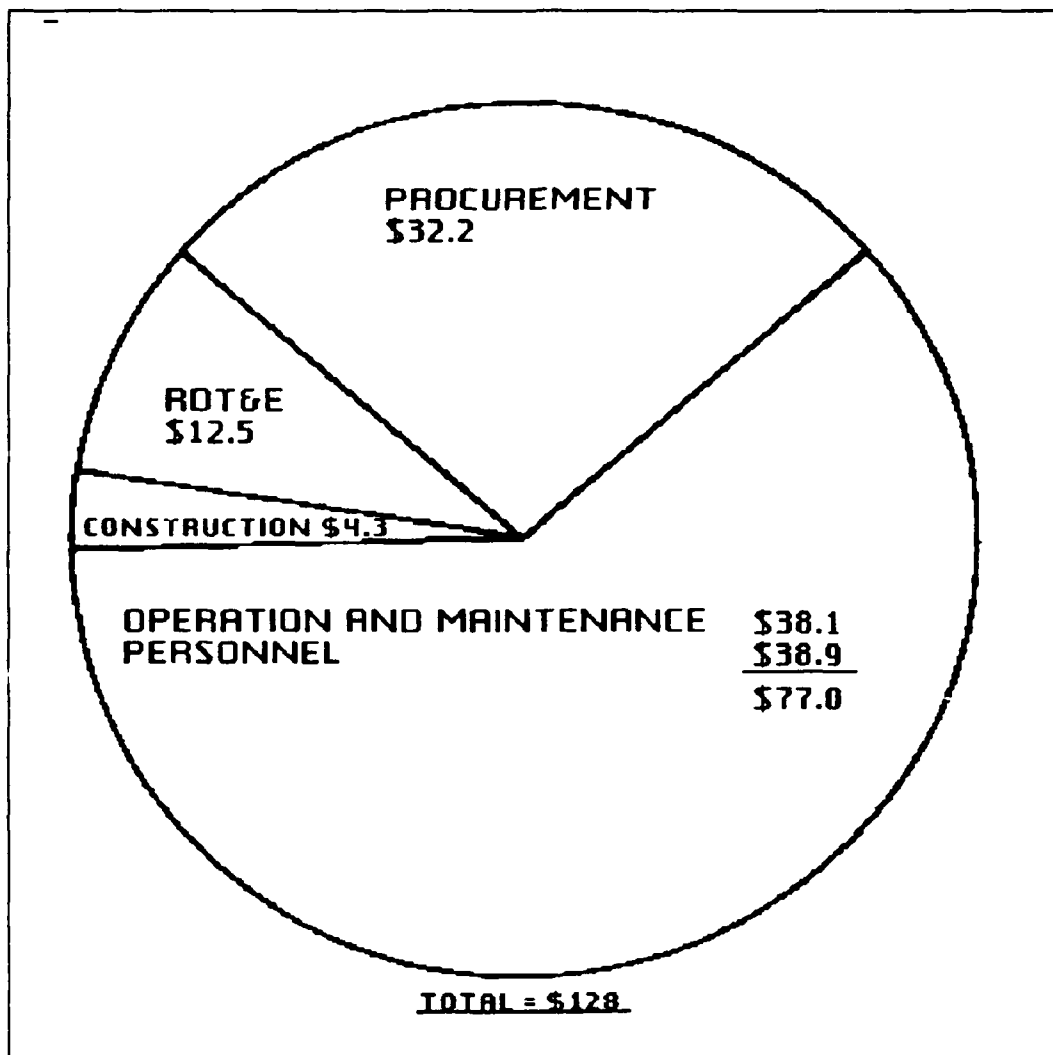


Figure 1 Department of Defense 1979 Budget
Billions of Dollars (Seldon:2)

Objective

The objective of this research is to develop an easy to use, flexible, PC-based Monte Carlo simulation for the preliminary estimation or forecasting, of life cycle costs.

Subobjectives

Model Definition. The first step is to survey life cycle cost models by reviewing literature. Identification of the proper model must precede all other steps.

Model Construction. Construction of the mathematical model, which means creation of the source code, is the next step in using the LCC cost model chosen. The actual mathematical calculations must be coded in this step.

It will be necessary to develop random number generators, the basis of the Monte Carlo simulation. Development of fast, competent generators is crucial to achieving the overall objective.

Software for the analysis of the LCC outputs must also be developed. The next section of code is the LCC distribution identification routines. Quantile estimation is closely linked to distribution identification, and it must also be coded.

Software Production. After verification of the various mathematical software routines, these routines must be integrated into a single program with a logical, flexible control structure. The integration of the routines is no more important than the development of a user friendly interface. The program must be easy to use.

Documentation. The documentation must be polished up and formatted. This documentation must include a User's Manual.

..

II. LITERATURE REVIEW

Introduction

The goal of this research is to design and build a flexible, easy to use tool for Monte Carlo simulation of the life cycle costs of weapon systems, including estimation of the distribution of these costs. Monte Carlo simulation is simply a compilation of random generations aggregated to infer something about the real world. The tool must generate samples from a cost distribution according to historic data about past weapon system attributes and costs. The areas of literature that must be reviewed for this effort include theoretical life cycle costing models, statistical estimation, and random number generation.

Chapter Overview

This chapter will present a brief overview of the types of LCC models available in the literature, providing the characteristics of each model, good and bad.

Since Monte Carlo simulation involves the representation of random events, the random number generator used by the modeling program must generate numbers that actually appear to be random. This chapter will review promising methods for generating the random numbers needed

by the life cycle cost model. Sample output from the random deviate generators actually used can be found in Appendix B.

LCC Modeling

The literature suggests that there are basically three types of LCC models (Collins:55; Krisch:1527; Seldon:161; DoD:3-3):

- 1) Parametric cost factor models
- 2) Engineering cost factor models
- 3) Accounting models

Parametric. The parametric cost factor model is named so "because the physical and performance measures are commonly called parameters in the estimating equations" used to forecast costs (DoD:3-3). The weight or number of engines on an aircraft may be used, for instance, to estimate the production and operating costs. Krisch explains that curve fitting can be used to derive Cost Estimating Relationships (CER) between cost and production schedule and system characteristics. These CER's can be adjusted to make the best estimate possible from past data (Krisch:1527).

Several advantages are associated with using this type of cost estimation, as outlined by the DoD Life Cycle Costing Guide For System Acquisitions:

- 1) Cost estimates are based on general system characteristics, no detailed information is necessary;
- 2) Model is very fast and easy to use;
- 3) Model is resistant to user bias;
- 4) Since parametric statistics are used in generating the forecasts, confidence intervals (CI's) can be placed on the forecasts (DoD:3-6).

It is in this last advantage that lies the real power of this type of modeling.

Figure 2 shows an example of a parametric CER model. Notice that the costs are being calculated from general characteristics, and that there is some error associated with each equation.

Engineering. Many authors ignore this model or group it with the model above in the "cost factor" category (Krisch:1527; Collins:54). This is because the engineering model is very similar to the parametric CER model mechanically. The system is broken down into cost components like above, and cost relationships are used to determine the cost of each component.

The difference in the CER and engineering models lies in the type estimating relationships used. The model above uses relationships of convenience, which may or may not capture a great deal of the cost variance. The engineering model uses specific hardware-to-cost relationships to

COST REPRESENTATION FOR 1 C-130		
<u>CER's</u>		
AIRFRAME	=	$200,000 + 75 X_1 + e$
ENGINES	=	$2,000 + 63 X_2 + e$
ELECTRONICS	=	$530 + 200 X_3 + e$
MANPOWER	=	$300,000 + 400,000 X_4 + e$
OPS	=	$500,000 + 12,000 X_5 + e$
<u>WHERE</u>		
X_1 = airframe weight		
X_2 = thrust		
X_3 = number radios		
X_4 = crewmembers		
X_5 = yearly flying hours		
e = error, iid $N(0, MSE)$		

Figure 2 Parametric CER cost model example.

determine cost. Obviously this takes more information, and indeed DoD does not recommend this as a method for preliminary work since the level of detail needed is usually obtained after many crucial decisions (based on cost) have been made (DoD:3-12). The main advantages of this method are increased accuracy and hence more detailed sensitivity

analysis of differing configurations, and ease of transition from a CER model (DoD:3-11).

Figure 3 is an example of an engineering model. Notice that the cost equations are more detail than in the CER model. Also the cost equations do not have the same error

COST REPRESENTATION FOR 1 C-130	
<u>cost equations</u>	
AIRFRAME	$= 13 X_1 + 109 X_2 + 12 X_3 + 2.5 X_4 + .03 X_5 + 750 X_6$
ENGINES	$= 150,000 X_7 + 520 X_8 + 3700 X_9 + 6200 X_{10}$
WHERE	
	X_1 = ribs in fuselage
	X_2 = windows
	X_3 = aluminum sq. ft.
	X_4 = pipes ft.
	X_5 = rivets
	X_7 = hydraulic pumps
	X_7 = compressors
	X_8 = fuel pumps
	X_9 = propellers
	X_{10} = fuel filtration systems

Figure 3 Engineering cost model example.

associated with the CER's because these cost equations are

the real world relationships between physical construction and cost.

Accounting. This seems to be the most detailed of the models types, summing costs over system components at a very low level, taking into account such needs as personnel, training, etc. This methods takes an enormous amount of information, such as lists of "contractor supplied LRU's ... flying hour programs and development scenarios ... Labor rates, inventory costs and repair cycles times, for example" (Collins:55).

Figure 4 shows an example of a portion of an accounting model. Notice that subsystems must be accounted for at a very low level. This would continue until virtually every part in the aircraft, and all the service costs, have been accounted for.

The reader may notice that the different types of models seem to be a progression of more and more detailed cost models. For this reason there seems to be a consensus among authors that the less specific models are more useful early in the acquisition cycle when little is known about the proposed system (Collins:56; DoD:3-10; Krisch:1527).

COST REPRESENTATION FOR 1 C-130		
<u>AIRFRAME</u>		
rivets	door adjustment	door seals
screws	hydraulic pumps	carpet
windows	hydraulic lines	boosters
floor board	interior lights	light bulbs
hinges	throttle cables	yoke
seats	hydraulic valves	insulation
wire	manual switches	
:	:	:
:	:	:
:	:	:
AIRFRAME		
:		
:		
:		
+		
total cost		

Random Number Generation

This portion of the literature review is concerned with the generation of random numbers, the simulation of samples conforming to a given cumulative density function. A cumulative density function (CDF), or distribution, is simply a function that makes a generalization about a population of values. Given an initial value, the CDF will identify the probability that a number drawn at random from

the specified distribution (or population of numbers) will be smaller than the initial value:

$$\text{CDF}(X) = P(x \leq X) \quad (3)$$

A number drawn at random from the population is called a random deviate since we don't know exactly what its value will be, or how far it will deviate from the expected value. Random deviates are useful because they allow modelers to sample from real-world processes.

Computer programming is by nature very structured, making the generation of random numbers no trivial matter. Winchmann and Hill offer the following from von Neumann, "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin" (Winchmann:127). Since any numbers generated by the methods to follow are reproducible by rerunning the same code again, they are not truly random. These numbers are called pseudo random deviates. These pseudo random deviates can be just as effective as truly random deviates if the generation method is designed with care, and the reproducibility they allow can be an aid in experimentation.

General Sampling Techniques. There are various methods for generating samples from some specified distribution.

The following methods were drawn from Pritsker, but they are commonly found in the literature (Pritsker:707):

- 1) Inverse transformation
- 2) Rejection
- 3) Composition

Ross adds another general technique, also common in the literature (Ross:442):

- 4) Hazard rate

Inverse Transformation. Pritsker makes it clear that the inverse transformation, or inverse CDF method, is by far the easiest method to use (Pritsker:708). This method entails using a function that is the inverse of the CDF function to generate properly distributed variables. Since the CDF returns a probability (between zero and one) associated with some value, the inverse CDF begins with a number between zero and one to produce the value associated with the given level of probability. Since the CDF of the distribution in question must be invertible, this method is not applicable for some of the commonly used distributions. Like all other methods discussed in this section the inverse transform method requires the generation of numbers uniformly distributed between zero and one. A simple uniform random number generator ($U[0,1]$) may be found in Winchmann et al (not used here), but the construction of

$U(0,1)$ generators is not the subject of this section (Winchmann:127-128).

Rejection. Tadikamalla gives a very good summary of the rejection method, also known as acceptance-rejection, as developed by Von Neumann (Tadikamalla:925-928). Rather than using the CDF described above, rejection makes use of the PDF associated with the distribution being modeled. The PDF is the first algebraic derivative of the CDF. The area under the PDF curve, taken between two points, gives the probability that a randomly drawn value will lie between the two points. The key is to find some other function, whose value returned for a given zero-one number will always be larger the value of the PDF at the same point. This is known as a majorizing function. This majorizing function must be easy to sample from (by inverse CDF or some other method).

A uniform zero-one variate and a variate from the majorizing function are drawn. If the uniform variate is smaller than the ratio of majorizing function value to PDF value, keep the variate as a sample from the designated distribution, otherwise draw new variates and try again. It stands to reason that the closer the majorizing function and the real PDF are, the fewer variates are rejected; so the smaller the difference between PDF and majorizing function, the more efficiently random numbers are generated (Tadikamalla:925-928).

Composition. This method may be used when the density function can be written as a weighted sum of other distributions (with the sum of weighting factors totaling one [Pritsker:710]). To sample from the designated distribution, first sample from the component distributions and sum according to the weighting factors to create one random variable.

Hazard Rate. The hazard function (H) is defined as the ratio of the PDF to the CDF (Ross:442):

$$H = \text{PDF/CDF} = P(t < x < t + dt | x > t) \quad (4)$$

This specifies the probability of a random variable x being greater than some value $t + dt$ given that x is greater than t . Leemis (Leemis:892-894) as well as others recognize that the hazard function for any probability distribution has unit exponential distribution, a known probability structure. The inverse of this hazard rate can be used to generate random variables similar to the inverse transformation method.

There are other methods for generating random variables that make use of the special properties of particular distributions. Special algorithms have been developed for sampling from most of the commonly used distributions which are faster than the general techniques. While Pritsker (Pritsker:710-711) gives a tidy outline of these, only the

methods applicable to the distributions needed for this research will be discussed in the next section. Also outlined by Pritsker is the idea that there may be useful working relationships among the various methods for generating random variables.

... the composition method may employ the inverse transformation method to select a subdistribution and then any sampling procedure to obtain a random sample from the subdistribution. The acceptance/rejection method is frequently used where majorizing functions are defined for portions of the distribution function. Thus, it should be clear that the methods for generating random samples are not necessarily used independently. (Pritsker:710)

Specific Generation Techniques. Since the majority of the random numbers needed for life cycle cost estimating must be distributed according to the beta distribution, this section will focus on generating beta deviates. The beta is useful for modeling symmetric or skewed unimodal data. The beta differs from the normal by the lack of long tails that extend out to positive and negative infinity, enabling the representation of populations having discrete upper and lower limits. Unfortunately the CDF of the beta distribution cannot be inverted, so the inverse CDF method is of no use. Furthermore, the combination of the first four moments of the beta make it very difficult to sample from by any method. For that reason the literature is mainly concerned

with the generation of two-parameter beta variates as the ratio of two-parameter gamma variates. Ross offers the following formula for obtaining a beta variable with parameters n and m (Ross:452):

$$\text{Beta}(n,m) = \text{Gamma}(n,1) / (\text{Gamma}(n,1) + \text{Gamma}(m,1)) \quad (5)$$

The parameters are often noted as (α, β) elsewhere in the literature.

Fishman provides a comparison of his improved method for generating gamma deviates to the algorithm of Wallace, developed earlier. The following is Wallace's algorithm as reported by Fishman:

- For integral α method 2 uses (7) [see paragraph below]. For non-integral α the six steps are:
1. Generate a uniform deviate U .
 2. If $U \leq 1 - \alpha + \langle \alpha \rangle$ then generate X' from (7) using $b = \langle \alpha \rangle$.
 3. Otherwise, generate X' from (7) using $b = \langle \alpha + 1 \rangle$.
 4. Generate a uniform deviate U .
 5. If $U \leq (X'/t') t' / (1 - t' + t'X'/t)$ then X' has the pdf in (2).
 6. Otherwise, go to step 1. (Fishman:408)

where t is $\langle \alpha \rangle$, the largest integer contained in α , and t' is $\alpha - t$. Fishman's (7) refers to the negative natural log of a multiplicative sum of α uniform deviates, the standard

method for calculating integral parameter gammas

(Fishman:408;Tadikamalla:925;Wallace:693):

$$-\text{Ln} \prod_{i=1}^{<\alpha>} U(0,1)_i \quad (6)$$

while (2) refers to (Fishman:407):

$$f_X(X, \alpha, \beta) = c(\alpha, \beta) a(\alpha, \beta) g(X, \alpha, \beta) h(X, \alpha, \beta) \quad 0 \leq X \leq \infty$$

$$= 0 \quad \text{elsewhere} \quad (7)$$

$$0 \leq h(X, \alpha, \beta), \quad (8)$$

$$\int_0^{\infty} h(X, \alpha, \beta) dx = 1 \quad (9)$$

$$0 < g(X, \alpha, \beta) < \infty, \quad (10)$$

$$a(\alpha, \beta) < 1/g(X, \alpha, \beta), \quad (11)$$

$$1/c(\alpha, \beta) = a(\alpha, \beta) \int_0^{\infty} g(X, \alpha, \beta) h(\alpha, \beta) dx \quad (12)$$

The method proposed by Fishman is computationally easier and proves faster as implemented on the IBM 360/75.

The steps for Fishman's algorithm are:

- 1) draw an exponential variable X with unit mean
- 2) draw a U(0,1) number,
- 3) if $U(0,1) \leq (X/\exp(X+1)) \alpha^{-1}$ then multiply X by α , then keeping this product as a gamma random variable other wise return to step 1. (Fishman:408).

Although the two methods may seem fundamentally different, they differ only in the choice of the majorizing function $h(X, \alpha, B)$, which does not even clearly appear in Fishman's method. Both these methods employ the rejection technique outlined in the general techniques section.

Another similar method is proposed by Tadikamalla, and compares favorably with Fishman for small values of α and is considerable better for larger values of α . This method uses the generation of Laplace variates (Tadikamalla:925-928).

Since many random beta variates will be generated in the course of modeling life cycle costs, efficiency and speed is very important. Kronmal and Peterson propose a modified Rejection method with Acceptance-Complement methodology that avoids repeating steps and has useful design flexibilities (Kronmal:271-281).

Greenberg has implemented a new density function able to approximate the type of data generally modeled by the non-integral gamma distribution through the mixture of integral gamma deviates which are much more quickly calculated (Greenberg:32-33). The method is quite simple and fast, but is not a true gamma, and using a ratio to generate beta variable is not recommended by Greenberg; however, it is possible that this density function could be

used to directly model the data used to estimate costs, rather than to mimic a beta distribution.

Along the same lines , Ramberg, Dudewicz, Tadikamalla, and Mykytka have proposed a Generalized Lambda Distribution (GLD) that can take on the shape of virtually any of the commonly used distributions (Ramberg:210-214). Once again, the beta cannot be simulated directly but the authors have confidence in betas generated via ratios of gamma approximations. Although this would be slower than using one of the more direct gamma generators, the power of the GLD lies in its uses for exploring sensitivity analysis. Since the distribution can be shaped virtually any way, it can be tweaked to test the sensitivity of a model to any certain input distribution assumption. Indeed such a study was undertaken by the authors (Ramberg:210-214).

Also mentioned in the general techniques section was the use of the hazard function. Recall that the hazard function is the ratio of CDF to PDF and is unit exponential, and known density structure. A proof is offered by Devroye (Devroye:281). Devroye offers as many hazard rate based generation methods as have been noted already in this section, but applications to the beta distribution do not seem to be available in the literature.

III. Methodology

Model Definition

The first step in identifying the proper model is to research the models available. Of the various types of LCC modeling found in the literature, the methodology that is most appropriate for preliminary analysis is the parametric CER method, or cost factor model.

The cost factor model uses CER's and single variables as cost components to define the total cost of the system. For example, for some aircraft the radio cost may be represented by one single variable, the navigation system by a CER, and the engines by more CER's. Each of these cost factors also has some specific time frame (R&D costs are up front whereas operating costs occur later in system life).

CER'S - Recall that a CER is a cost estimating relationship (a regression equation). For example, the cost of a radio may be estimated by three dollars per channel it receives, twelve dollars per watt of transmitting power, plus fifty dollars for the basic chassis:

$$C = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \quad (13)$$

where

C = radio cost
 β_0 = \$50.00
 β_1 = \$3.00
 β_2 = \$12.00
 X_1 = channels
 X_2 = watts transmitting power

CER's attempt to estimate costs for future products based on characteristics shared with past products. The future products must have characteristics in common with the past products, to allow forecasting according to regression equations (based on those characteristics). Forecasting the cost of new technologies must be performed with great care. Obviously the cost of the B-2 engines cannot be estimated by comparison with the engines from the F-84, because the two do not share many characteristics.

The characteristics are the explanatory variables for the CER's, or regression equations. The distributions and parameters of each input variable must be specified. Additionally the β parameter estimates and their covariances must be known, along with the MSE of the regression.

Three basic types of CER's will be available in the model; natural logarithm, learning curve, and linear (regular) with explanatory variable power transformations allowed (Box:531). The overall system cost may be composed

of up to twenty CER's, with each CER allowed up to ten components, or explanatory variables.

SINGLE VARIABLES - Some costs can be adequately represented by a single variable from one of the eleven available distributions. For example, the cost of a tire may be normally distributed by $N(150,12)$ due to fluctuations in availability. Again the distribution and parameters for each variable must be specified when the system is defined. A limit of twenty single variable cost components imposed.

MONTE CARLO METHODOLOGY

The cost simulation methodology will be to predict the cost associated with each CER cost component and each single variable cost component (through random number generation), summing all these up to get one overall estimate of system cost:

$$\text{TOTAL COST ESTIMATE} = \sum_{j=1}^n \text{CER}_j + \sum_{k=1}^m \text{single variable} \quad (14)$$

where

n = number of CER's

m = number of single variable cost components.

By viewing many repetitions of this estimate the analyst may get an idea of the true cost distribution underlying these sample costs.

CER'S - Predicting the cost associated with one CER can be a very complicated task. The following steps capture the

essence of the process. These steps are detailed more thoroughly in Chapter IV.

- 1) draw a set of dependent β parameters from a multivariate normal distribution according to the β covariance matrix.
- 2) sample each explanatory variable from the proper distribution.
- 3) transform each variable by the appropriate power.
- 4) multiply each transformed variable by the appropriate β value.
- 5) sum the products.
- 6) add a normally distributed $N(0, \text{MSE})$ value to the sum.
- 7) distribute the value of the sum according to the time phase specifications, then find the present values.

SINGLE VARIABLES - Single variable cost components are much simpler to simulate. Each variable is sampled from the appropriate distribution, time phased, and converted to a present value.

TIME PHASED SPENDING

After each cost component has been estimated (CER's and single variable cost components), it must be adjusted to a Present Value (PV) before it is added to the overall system cost. Obviously money spent during different phases will have different PV's due to the discount rate (interest).

For example: Farmer Brown owes \$100.00 to the Farm Equipment Corporation and \$100.00 to the General Feed Store.

However, the equipment bill is not due until next year. If the current interest rate at the local bank is 10%, Farmer Brown may invest \$90.91 and allow the money to collect interest, or "grow" to \$100.00 over the next year:

$$PV(\$100.00) = 100/(1 + r)^N = \$90.91 \quad (15)$$

where

r = periodic interest rate, 10% here
N = number of periods into the future, 1 here

The feed bill however, must be paid now, costing Farmer Brown the entire \$100.00. This is the concept behind present values; "How much will I have to invest now to have \$XXX.XX at some point in the future?" Obviously this depends on how far into the future the money is due and the prevailing interest (or discount) rate.

This creates a need for the user to specify the timing of the money to be spent and the current interest rate. The spending process is broken into four periods, called the NOCOST period, Phase In period (PI), Constant period (CON), and Phase Out period (PO). The user must specify the length of each of the time periods (which may be zero) for each cost component.

Figure 5 shows examples of various time phase configurations. The total area associated with each year (ie. under the curve between 0 and 1 for the first year) will determine how much of the money from the cost component

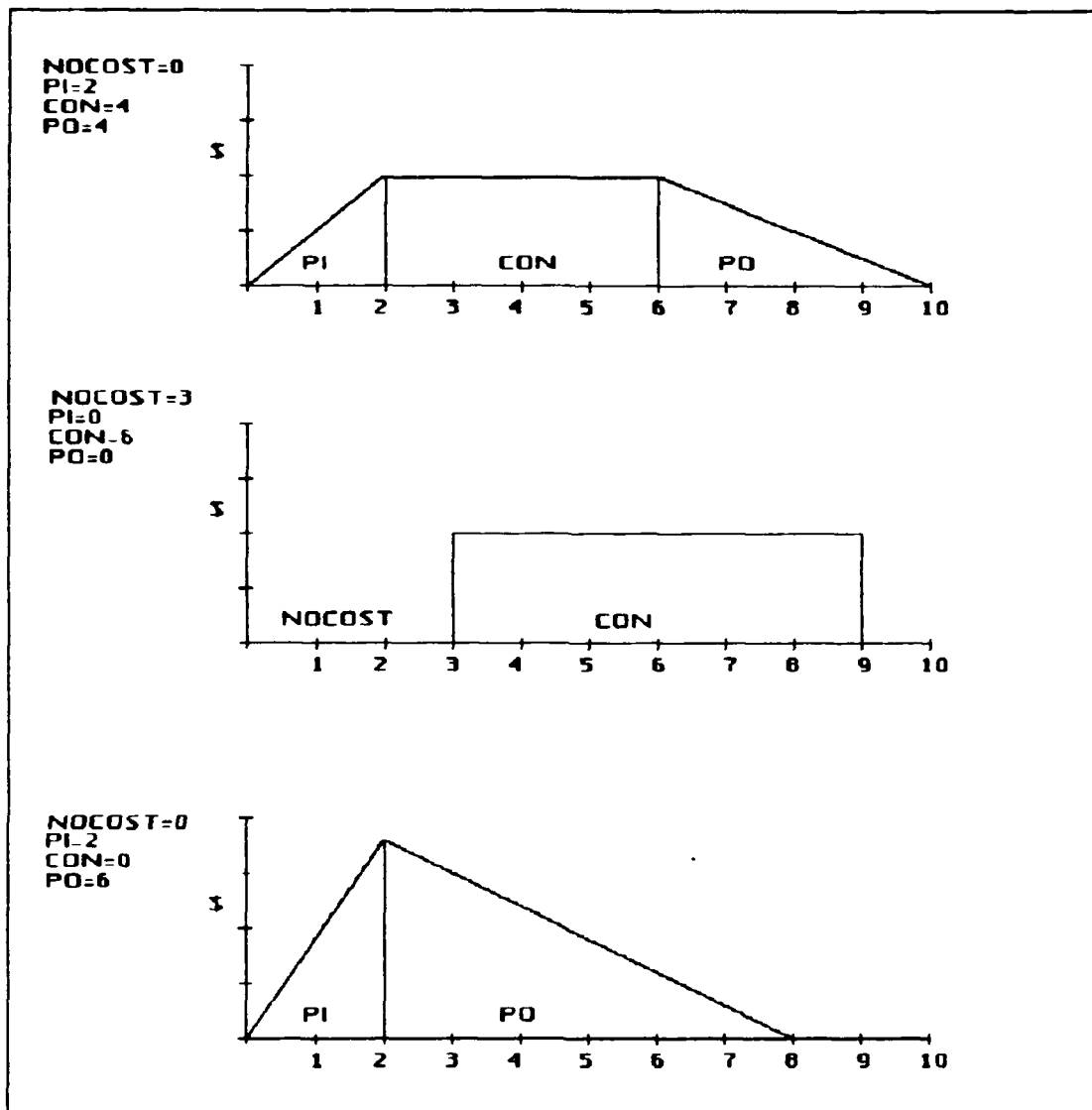


Figure 5 Spending timelines for cost components.

in question will be spent that year. Note that the second spending timeline has no NOCOST, PI, or PO period, and the last has only PI and PO periods. Remember that each separate cost component has its own timeline, and that the sum of these timelines would give an overall \$/year timeline for the system.

Although this method does not nail down the exact amount of money to be spent each year, it requires only four time parameters from the user, and it will suffice for preliminary cost analysis.

OUTPUT ANALYSIS

Now that a set of cost estimates for the system in question has been produced, through Monte Carlo simulation, the set must be examined. It is wise to gather as much information as possible about the simulation and its output since the experiment cannot be run with the real world system. The object of this section is to guide the user through descriptive tests and procedures designed to explore the underlying distribution of possible costs for the system in question.

FREQUENCY HISTOGRAM - A frequency histogram is a graphical representation of the number of data points that fall into each of a set of numerical ranges, or classes. Viewing a histogram can lend insight to how the population underlying the data sample is distributed. Figure 6 is an example of a histogram (produced by the Frequency Histogram option of this software and converted to black and white). This figure shows the number of data points that fall into each of the numerical classes. The upper class bounds are written at the bottom of the graph under the boundaries of the bars. The X axis is in the units of the data set. The

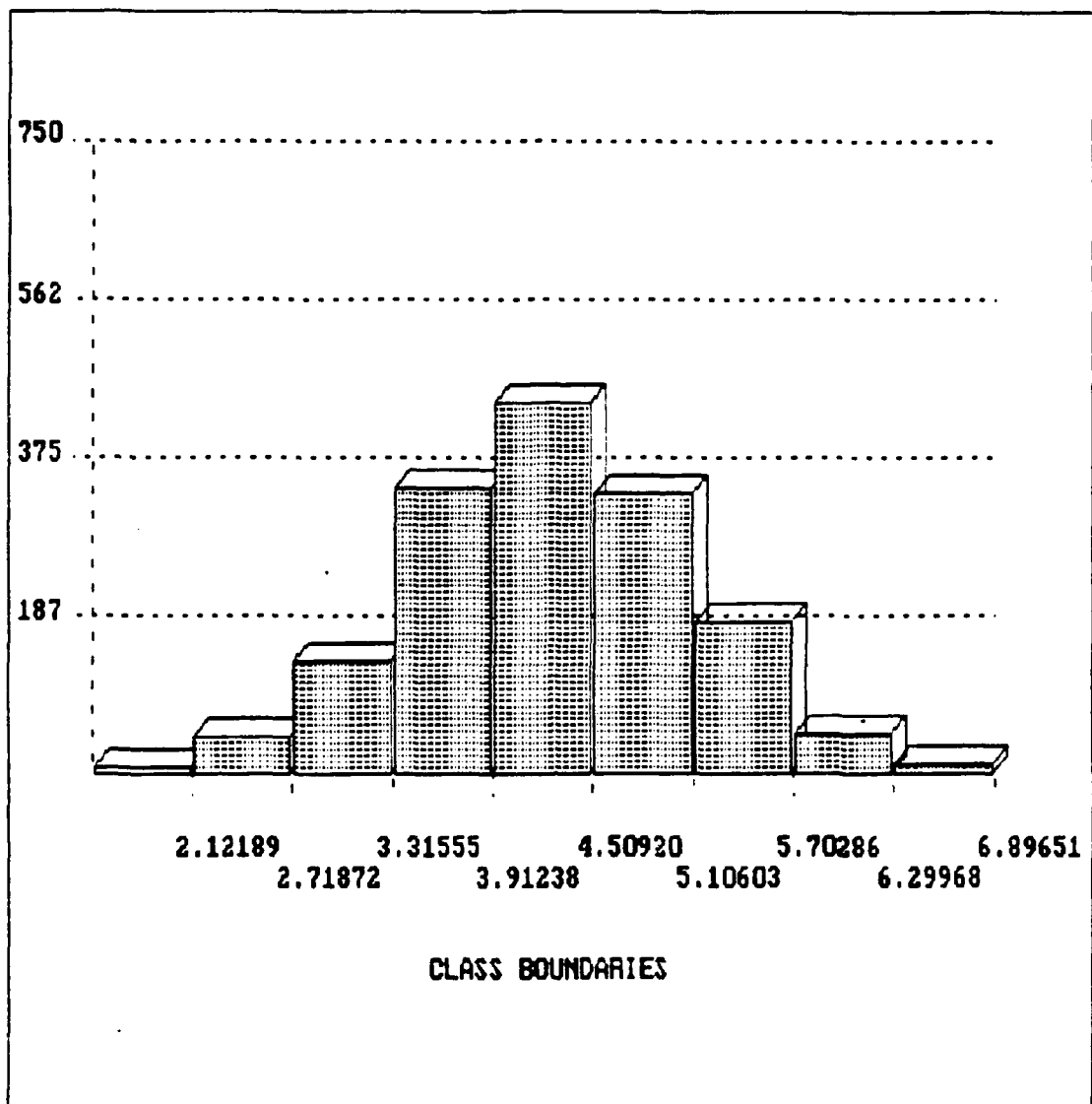


Figure 6 Example of a histogram.

Y axis is the frequency of observations.

MOMENTS - The first two moments of a sample, the mean and variance, give some idea of the data range and spread. This information can facilitate comparisons among data sets. Also useful are the extreme high and low values, the standard error of the variance estimator, and the median, or

the fiftieth percentile of the data (see the next heading for more percentile information).

QUANTILE ESTIMATION - Quantiles, or percentiles, deal with the order, or rank, of data points. The fiftieth percentile is the middle point of the data after the set has been sorted (ie. of 3, 4, and 5, 4 is the fiftieth percentile). The fiftieth percentile is also called the median. Unless the data of interest is normally or uniformly distributed, the median is often more useful than the mean in describing the typical value of the data points.

The value of any quantile, from anywhere above 0 to anywhere below 100, can be estimated. Thus at the ninetieth percentile value, X , there is a ninety percent chance that subsequent random values (from the same distribution) will be lower than X . Another way to state this is that ninety percent of the values drawn from this distribution will be less than X .

NON-PARAMETRIC PROBABILITY - At times it would be very useful to know the probability of drawing a number smaller than some reference value from a give population. This is a very simple problem should you know the true distribution of the population and its parameters (mean, variance, high, low, α , β etc.). However, the analyst will rarely know the distribution type of the simulation output data. Rather

than performing some lengthy set of weak tests to ascertain which distribution is at hand, it is possible to use nonparametric order statistics to estimate the probability introduced above.

The problem of finding a probability associated with some value, is the opposite of quantile estimation, finding the value associated with some level of probability. Both quantiles and probabilities are yet another way to conduct comparisons among the cost distribution of competing systems.

Note that the descriptive methods described so far are non-parametric, meaning they do not require any assumptions about the distribution underlying the sample data.

T-TESTS FOR SAMPLE MEANS - Through T-tests the analyst may test the hypothesis that the means of two systems' cost distributions are the same. These tests are very important because point estimates of means, as well as other parameters, can be very misleading. The fact that the mean of one sample is higher than the mean of a sample drawn from a different distribution, is not enough information to conclude that the underlying distribution means share the same relationship.

T-TEST 1 - This T-test is for two independent populations with the same variance (σ^2). Although the variance need not be known, the hypothesis of $\sigma_1^2 = \sigma_2^2$ must be tested (with failure to reject) before this T-test

can be used. This T-test assumes normally distributed samples are used, which is a good assumption for large samples since X approaches normality as the sample size approaches infinity.

T-TEST 2 - This T-test is for two populations with the unequal variances (σ^2). The variances need not be known, and need not be tested for equality. However, this test should not be used unless the test above is invalid due to differing variances. The test above is a much stronger test, meaning it will be able to reject the null hypothesis ($\mu_1 = \mu_2$) more often, without more error. This T-test assumes normally distributed samples and independent populations, like the test above.

T-TEST 3 - This test is for paired data. It is not appropriate for analyzing the output from this Monte Carlo simulation, but it is offered for use with other simulations or observations. The populations in question should be dependent. Elements within pairs of observations can and should be correlated, being observed from similar scenarios. For a more complete discussion of paired data see Johnson (374) or any other elementary statistics text.

NON-PARAMETRIC TESTING - The following section will outline how to perform tests similar to the T-tests above, but without making assumptions about the underlying distributions. Additionally these tests will allow

hypothesis testing for any quantile, or probability, for any two distributions.

TEST FOR EQUAL QUANTILES - Quantile estimation can be used to test the hypothesis that the N^{th} quantiles of two distributions are equal, provided confidence intervals (CI's) have been also estimated. Simply stated, if the CI for the N^{th} quantile of sample set A intersects with the CI for the N^{th} quantile of sample set B, then fail to reject the null hypothesis. Should the two CI's not intersect, reject the null hypothesis.

It should be noted that such joint use of the two confidence intervals (each encompassing α amount of risk) compounds the risk associated with the α values used to create the confidence intervals.

For example, if the two confidence intervals are created with $\alpha = 0.1$ (as in the program) then the total confidence in the outcome of the test is a product of the separate confidences:

$$C = \prod_{i=1}^N (1 - \alpha_i) \quad (16)$$

or

$$C = (1 - \alpha_1) * (1 - \alpha_2) = 0.81 \quad (17)$$

providing the separate confidence intervals are independent (they are when produced by this Monte Carlo simulation program). If the two intervals were not independent (ie.

produced from the same run of some other simulation program)
the α 's are added to get the overall amount of risk
associated with the test:

$$C \geq 1 - \sum_{i=1}^N \alpha_i \quad (18)$$

or

$$C \geq 1 - (\alpha_1 + \alpha_2) = 0.8 \quad (19)$$

would describe the confidence in the test above, if the
confidence intervals were not independent. This is known as
the Bonferroni inequality (Kleijnen:41). The actual
confidence is greater than or equal to the resulting
confidence factor, thus the name evolved.

IV. IMPLEMENTATION

RANDOM VARIABLES

The heart of a Monte Carlo simulation is the set of random number generators, and the heart of all the random number generators is the uniform(0,1) generator used to feed them. This U(0,1) generator was taken from Numerical Recipes: The Art of Scientific Computing (Press:199). This generator was tested for mean, variance, and serial correlation (see Appendix B). There were no apparent problems.

Although it is possible to speed up the random number generators by programming them in assembly language, this hampers flexibility in support of the code. So assembly language was not used. All the univariate distributions below are available for single variable cost components as well as CER input variables.

BETA SAMPLING - The nine beta distributions shown in Figure 7, taken from Dienemann, provide a wide range of characteristics. The beta deviates are generated using the ratio of gammas discussed in Chapter II, Eq(3). The gamma variables used for the ratios are generated using either Fishman's or Wallace's method (one is much faster for $\alpha \leq 1.0$). Both are discussed in Chapter II. The beta

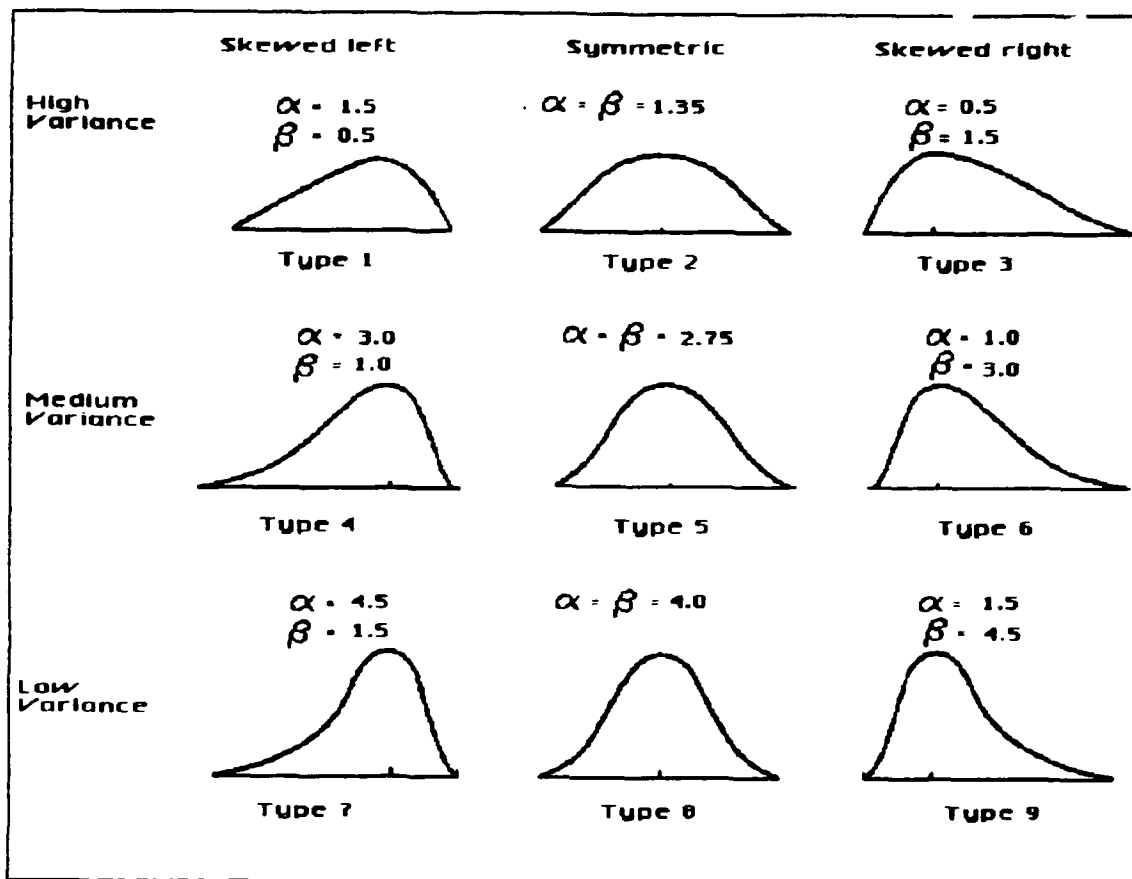


Figure 7 Input uncertainty probability distributions (Dienemann:14).

deviate is generated on the (0,1) interval and multiplied by (hi-low) and then added to the low value specified by the user:

$$\text{Beta}(\alpha, \beta)_{(hi, low)} = \text{Beta}(\alpha, \beta)_{(0,1)} * (hi-low) + low \quad (20)$$

NORMAL SAMPLING - In addition, the normal distribution is also available for CER explanatory variables and single variable cost components. The analyst should exercise

caution when specifying a cost as a normally distributed deviate since normally distributed variables can easily have negative values (which make no sense in terms of cost and cannot be used in power transformations). The normal deviate is calculate by the following equation (Ross:447):

$$N(0,1) = (-2 \ln(U_1(0,1)))^{1/2} * \cos(2\pi U_2(0,1)) \quad (21)$$

This produces a standard normal deviate. To achieve a deviate from the proper distribution, the value must be multiplied by the standard deviation and added to the mean:

$$N(\mu, \sigma^2) = N(0,1) * (\sigma) + \mu \quad (22)$$

UNIFORM SAMPLING - [The easiest of them all.] A uniform sample is create by multiplying the $U(0,1)$ deviate by the (hi-low) and adding it to the low:

$$U(\text{low}, \text{hi}) = U(0,1) * (\text{hi-low}) + \text{low} \quad (23)$$

MULTIVARIATE NORMAL SAMPLING - This is the most direct and technically pleasing way to account for the variance of the β parameters of the CER's. The β 's must be distributed according to their means and the covariance matrix, $\beta_{\text{est.}} \sim (\beta, (X'X)^{-1} \sigma^2)$. The following set of steps captures the essence of the process:

- 1) reduce the covariance matrix to its Cholesky square root (Maindonald:17)
- 2) draw a vector of independent, standard normal deviates
- 3) multiply the vector by the Cholesky matrix
- 4) add the estimated means to the remaining vector of deviates.

or

$$\underline{\beta}_{\text{sample}} = \underline{C} * \underline{N} + \underline{\beta}_{\text{HAT}} \quad (24)$$

where

$\underline{C}$ = Cholesky square root of covariance matrix
 $\underline{N}$ = vector of independent standard normal deviates
 $\underline{\beta}_{\text{HAT}}$ = estimated means
 $\underline{\beta}_{\text{sample}}$ = one sampled vector of dependent normals

This is the same type of algorithm used by the IMSL library routine.

COST ESTIMATION

Each CER is evaluated according to its type. Linear CER's are the simplest (described in Chapter III). The natural logarithm CER's are handled much like the linear CER's with two exceptions:

- 1) after each explanatory variable is sampled, its natural log is taken
- 2) after the rolling sum for that CER is totalled its exponent is taken (inverse natural log, ie. e^{total}).

Let it be clear that the X variable must be specified in its actual form for Ln CER's; the program will take the Ln of the explanatory variables during the calculations. The output variable is return in its standard form as well, the exponentiation is taken care of by the program.

For example, suppose

$$Y = X_1^{\beta_1} X_2^{\beta_2} \quad (24)$$

and the X variables are uniformly distributed between 5 and 10. The β parameters are estimated by:

$$\ln(Y) = \beta_1 \ln(X_1) + \beta_2 \ln(X_2). \quad (25)$$

with linear regression software. For the purposes of this program, X_1 and X_2 should be described as uniform with low=5 and hi=10. The program will handle the natural logarithms and output Y (not $\ln[Y]$).

The learning curve CER adds one more twist. It too is a natural log process, but after the final CER value (e^{Total}) is calculated, it is multiplied by X_1 , the first explanatory variable. This is because learning curve CER's apply to individual parts, needing to be multiplied by the total number of parts purchased (X_1) to find the total cost for that component. Like the Ln CER's, the variables for

the learning curve CER's should be specified in standard form (non-log).

POWER TRANSFORMATIONS

Power transformations on the input variables can lend extra explanatory power to linear equations. In essence, these transformations allow nonlinear fitting with linear software and methodology.

The process for finding the optimal power transformations (Box:) has been implemented along with a basic regression tool in the limited version of MATRIX (also written by the author). This implementation of the Box algorithm accepts up to 39 input variables and up to 80 observations, which is usually adequate in light of the small data sets used to produce CER's. It should be noted that power transformations can only be estimated by using the original data. The user must first estimate the power transformation, then transform the data and generate his own new CER.

Note: MATRIX cannot take a power transformation on a variable with a negative value. Ensure that no power transformation is specified for any variable that can possibly be negative. Be especially wary of normally distributed variables.

Use of this program will provide the user with the optimal power transformation value (α) for each explanatory

variable (one CER at a time). Since this is accomplished through successive regressions, the original data used to generate the CER must also be used to find the correct transformation.

For example, if the fixed part of the model is actually:

$$Y = X_1^4 + X_2^{1/3} \quad (26)$$

the algorithm will return α values of 4.0 and 0.33 (or very near there) for X_1 and X_2 respectively when given the X matrix and Y vector. The analyst must then make the transformations to the X data and estimate a new CER (MATRIX can do both). The α values and the new β values (along with covariances and MSE) that are estimated using the MATRIX program are then entered during the CER definition phase of the model builder program.

Box and Draper (Box:296) offer a lengthy discussion on process of estimating power transformations which boils down to the following steps:

- 1) fit a the model $Y_u = \sum_i \beta_i X_i + \text{error}_u$
- 2) form $Z_{iu} = \beta_i * X_{iu} * \ln(X_{iu})$
- 3) fit a model with the Z's as new input variables,
 $Y_u = \sum_i \beta_i X_i + A_i Z_i + \text{error}_u$
- 4) for A_i that are significant (using t-test), set
 $\alpha_i = A_i + 1$

- 5) raise X_i to the α_i power for all α_i that are significant, across all observations.
- 6) repeat steps 1-5 until no Z's are significant, keeping track of the cumulative effect of each α .

TIME PHASED SPENDING

Remember that each CER of single variable cost component has its own spending timeline defined by NOCOST, PI, CON, and PO. Figure 5 shows the spending time line for a CER. The total length of this spending process is seven years (total of NOCOST, PI, CON, and PO). This means that the money associated with the CER (the output variable) will be spent in seven lump sums, one payment at the end of each year (the model assumes spending at the end of each period).

NOTE: The actual spending profile should be based upon the likely pattern of payments to the constructor (progress payments are often embodied in the contract). This may or may not correspond to the expected delivery date(s) of the hardware.

The goal is to find how much of the money is spent each year, as specified by the four time period values. The sum to be paid at the end of each year is proportional to the amount of area under the curve in the block for that year. In Figure 8, the area of C, associated with year 1, is just over 3% of the total area:

$$\text{proportion} = C / (A + B) = 0.1666 / 5.5 = .03 \quad (27)$$

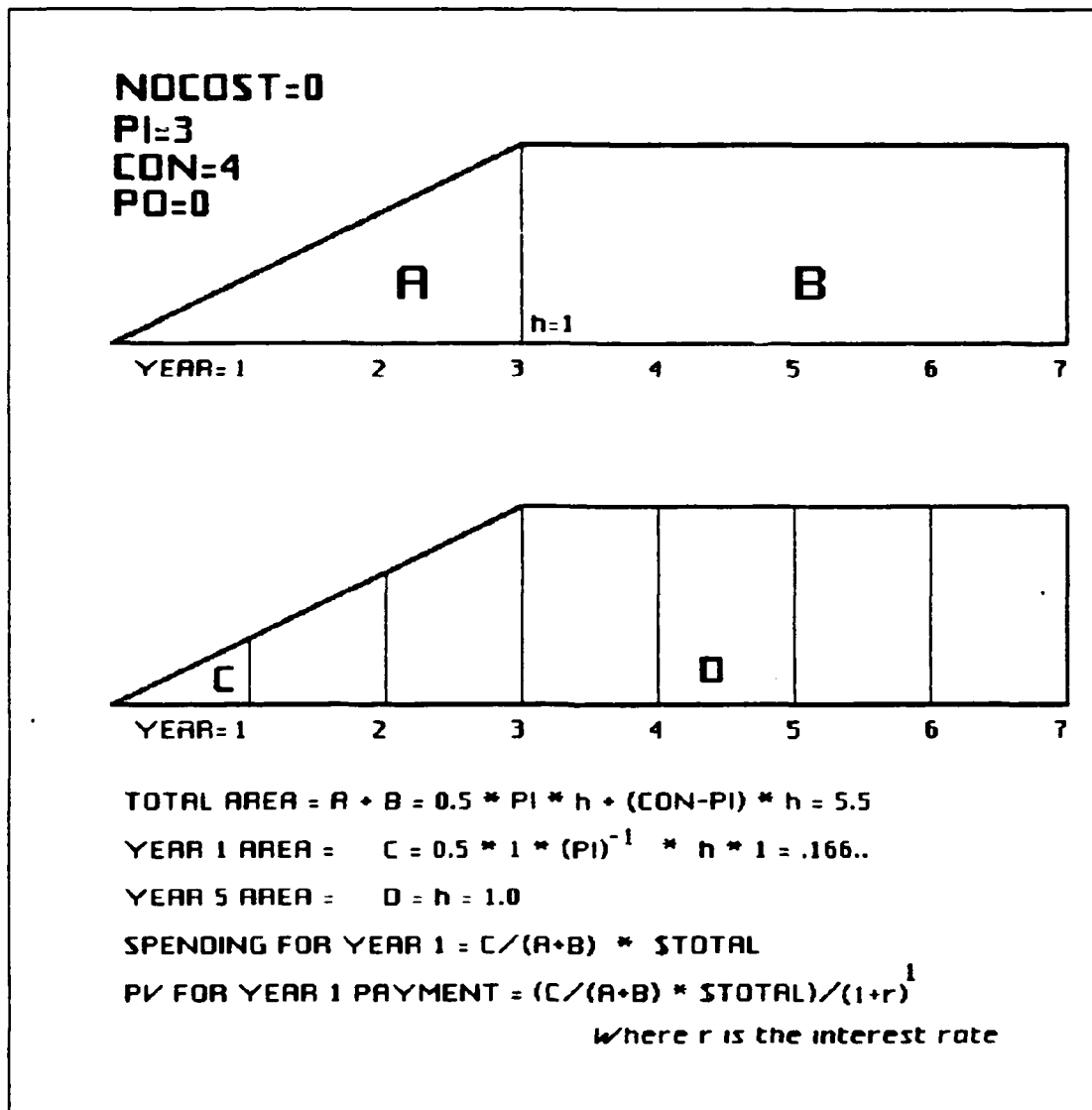


Figure 8 Present value calculations for time phased spending.

That means that just over 3% of the total cost associated with this CER will be incurred at the end of the first year. The sum due at the end of each year is calculated in the same basic geometric fashion.

Once the total cost associated with one cost element has been broken down into yearly expenditures, each expenditure must be adjusted to a present value. The adjustment is made with Eq (13) from Chapter III. The sum of all these present values:

$$\text{Component cost} = \sum_{i=1}^N PV_i \quad (28)$$

where

$N = \text{nocost} + \text{pi} + \text{con} + \text{po}$ (years over which to spend money)

is the final cost associated with this one cost element (CER in this case). The total system cost (life cycle cost) will be the sum of the present values of all the cost components:

$$\text{TOTAL COST} = \sum_{J=1}^M PV(\text{VAR}_J) + \sum_{P=1}^K PV(\text{CER}_P) \quad (29)$$

WHERE

M = number of variable cost components
 K = number of CER's

Software Production

The routines have been be consolidated into a single program with a homogenous control structure. The user interface is a system of pop-up menus and windows with cursor selection for novice users, as well as first letter selection to facilitate "command language" use by experienced users.

Documentation

The User's Manual (Appendix A) will provide every bit of information needed to operate this software. The function of each menu item is discussed, and a one example tutorial is provided.

Appendix C provides all the source code for the Monte Carlo simulation program and the MATRIX program. Inline comments and the readable style of Pascal code make it quite understandable.

V. CONCLUSIONS AND RECOMMENDATIONS

THE GOAL

The goal of this effort was to develop a tool to enable quick preliminary analysis and comparison of weapon system costs. That goal was realized. The PC environment is quite adequate for the simulation of LCC costs. Parametric LCC estimation is both quick and easy.

VALIDATION/VERIFICATION

Verification was performed on each procedure and the system as a whole. Remember that verification is checking to see if the code performs as intended. Validation, on the other hand, is checking to see that the methodology that has been implemented in the code actually model the real world closely enough.

Validation is difficult because this software is a model building tool. The user builds his own model of the real world with the CER's and single variable cost components he uses to describe the costs.

This program will accurately and consistently produce forecasts of the costs, as they are described by the user. Whether or not the user's model accurately captures the essence of his part of reality is beyond the control of the author. Bottom line: the user must validate his own model.

RECOMMENDATIONS FOR FUTURE RESEARCH

VARIANCE REDUCTION - Each output parameter generated by any simulation is an estimate. It is only natural to want the best possible estimate of each parameter. Variance reduction offers the possibility of increasing the power of this simulation software. Whether or not the common techniques of variance reduction can tighten the confidence intervals generated by this software without biasing the estimates of the output distribution's spread is not clear to the author. Due to time constraints sufficient research and experimentation was not performed in this area.

Since all source code has been documented and provided herein, the software should not cease to develop. In particular, the use of variance reduction could be studied through experimentation. Several runs of a model using various types of variance reduction, and none, could provide adequate data for an analysis of variance describing the effect of the different variance reduction techniques.

PARAMETRIC LCC ESTIMATION VALIDATION - Now that a powerful and easy to use tool has been provided, the validation of parametric cost modeling can be undertaken much more swiftly. Essentially, a follow-on to researcher could probe that actual value of the modeling methodology without having to sweat the details of implementation.

RANDOM DEVIATE GENERATION - It has been suggested that triangular distributions can be used in place of beta distributions with little loss in accuracy. If this is the case, then a great deal of time may be saved by replacing the acceptance/rejection generators used for beta deviates.

APPENDIX A: INTERACTIVE LIFE CYCLE COST FORECASTING TOOL

USERS' MANUAL

APPENDIX OVERVIEW

The purpose of this users' guide is to assist with the actual keyboard entries necessary to use the software. The three sections of this manual do not contain sufficient information to use the software properly. Theory, methodology, and assumptions should be reviewed in the preceding chapters before the software is used.

The first section deals with the user environment; the menu system and the entry conventions required by the language. The second section is a listing and description of the functions available. Provided in the third section is a numerical example with a keystroke by keystroke tutorial.

THE USER ENVIRONMENT

RESPONDING TO PROMPTS - There are certain conventions that must be observed when using this software, most of them deal with data entry. Following is a list of the most commonly violated conventions.

REAL NUMBERS - real numbers (eg. 2321.234 or 0.345) must be entered with at least one digit left of the decimal and no commas. Reals less than one may not be entered without a leading 0. For example, .5 is not allowed, nor are fractions such as 1/2; 0.5 is the only way to enter the value. The value 10,567 may only be entered as 10567 or 10567.00 or 1.0567E4. -0.03329 can only be entered as such or as -3.329E-2. Notice that scientific notation is allowed, as used in the preceding examples.

INTEGERS - integers must be entered much like reals except that decimals and digits left of them are not allowed. Like with reals, commas are not allowed. For integers scientific notation is not allowed.

FILE NAMES - When the user is prompted for a filename, a string of up to eight characters may be used. These characters may be letters, numbers or symbols. No extension is required or allowed. Extensions are assigned automatically depending on the file type. Data set files are given .SET and the LCC system files are given the .STM extension. Following each file name prompt will be a default name. To accept the default name simply press CR.

EXCEPTIONS - The above rule concerning file extensions does not apply when the user is prompted for the

file to read in the **READ ASCII** option. Nor does it apply when the user is prompted for the filename to write in the **WRITE ASCII** option. These filenames should be given the appropriate extensions since they apply to ASCII files, not data set file used by this software. The data set files will still retain their .SET extensions.

THE SCREEN PRINTING PROMPT - In some of the routines, the user has the option of getting a printout of the graphics that will appear on the screen (FREQUENCY HISTOGRAM, TIME SERIES PLOT, MOMENTS, XY PLOTS). Simply answering yes to the prompt will not ensure that a hardcopy of the screen will be produced. This option works only if a screen dump utility has been installed.

With some systems the DOS command GRAPHICS is sufficient (this DOS feature does not seem to work for all computer-monitor-printer combinations). Some printer manufacturers provide custom screen dump routines for their printers. Another possibility is using a public domain dump utility such as EGADMP that is provided with the CHART package.

Once one of these utilities has been installed, graphics may be printed by requesting a dump from the routine, or simply pressing SHIFT and PRTSCR simultaneously

while the graphics are on the screen. The same action will dump a text screen to the printer. If lines and boxes do not appear on the hard copy as they do on the text screen, exit the program and issue the DOS command GRAFTABL. This enables DOS to send the extended ASCII characters that define lines and corners to the printer. Some printers require that the IBM mode be activated to print these symbols. Check your printer manual.

MENU OPERATION - This sub-section deals with the specific operation of the menus. This is a custom, pop-up menu system written by the author in Turbo Pascal 4.0. Figure 9 is a black and white reproduction of the main menu. This menu offers six choices; the data menu, the statistics menu, the T-testing menu, the Files menu, the Random deviates menu, and setting a new seed for the random number generators.

A menu choice is selected by pressing the key corresponding to the first letter of the choice. Note that all the first letters are in bold type to remind you of this selection method. To select the statistics menu press the S key.

Another way to select a menu item is to use the arrow keys to highlight the desired item and press the carriage return (or enter) key. Notice that the Data choice is underlined in Figure 9. This indicates that this item would

be highlighted in a different color on the computer screen. Pushing the down arrow once would cause the Data item to return to normal colors and the Statistics item to become highlighted. This is much easier to see with the software actually running on the color screen.

If a mouse is installed with cursor key emulation, the mouse may be used rather than the cursor keys. One of the mouse buttons must emulate the carriage return key for actually selecting one of the menu items. A mouse driver is provided with the package (for the Genius family of serial mice).

The QUIT option on the main menu is replaced by a BACKUP option on each subordinate menu. The BACKUP option takes you back up one level to the previous menu. For example, selecting the BACKUP choice from the data menu will make the program exit the data menu and return to the main menu.

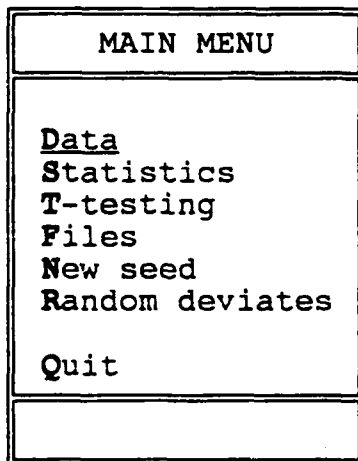


Figure 9

MENU LAYOUT - Figure 10 shows the layout of the menus. This will make it easier to find a function without roaming through the entire menu system. Unlike some commercial packages that claim to enable their users (pun intended), these routines are laid out in logical groupings.

Some menu choices call another menu, while others perform functions. The thick boxes with capital captions in Figure 10 represent menus while the thin boxes with lower case captions are procedures. For example, the DEFINE option on the DATA menu is actually a subordinate menu offering several choices, while the ENTER option is simply a keyboard data entry procedure.

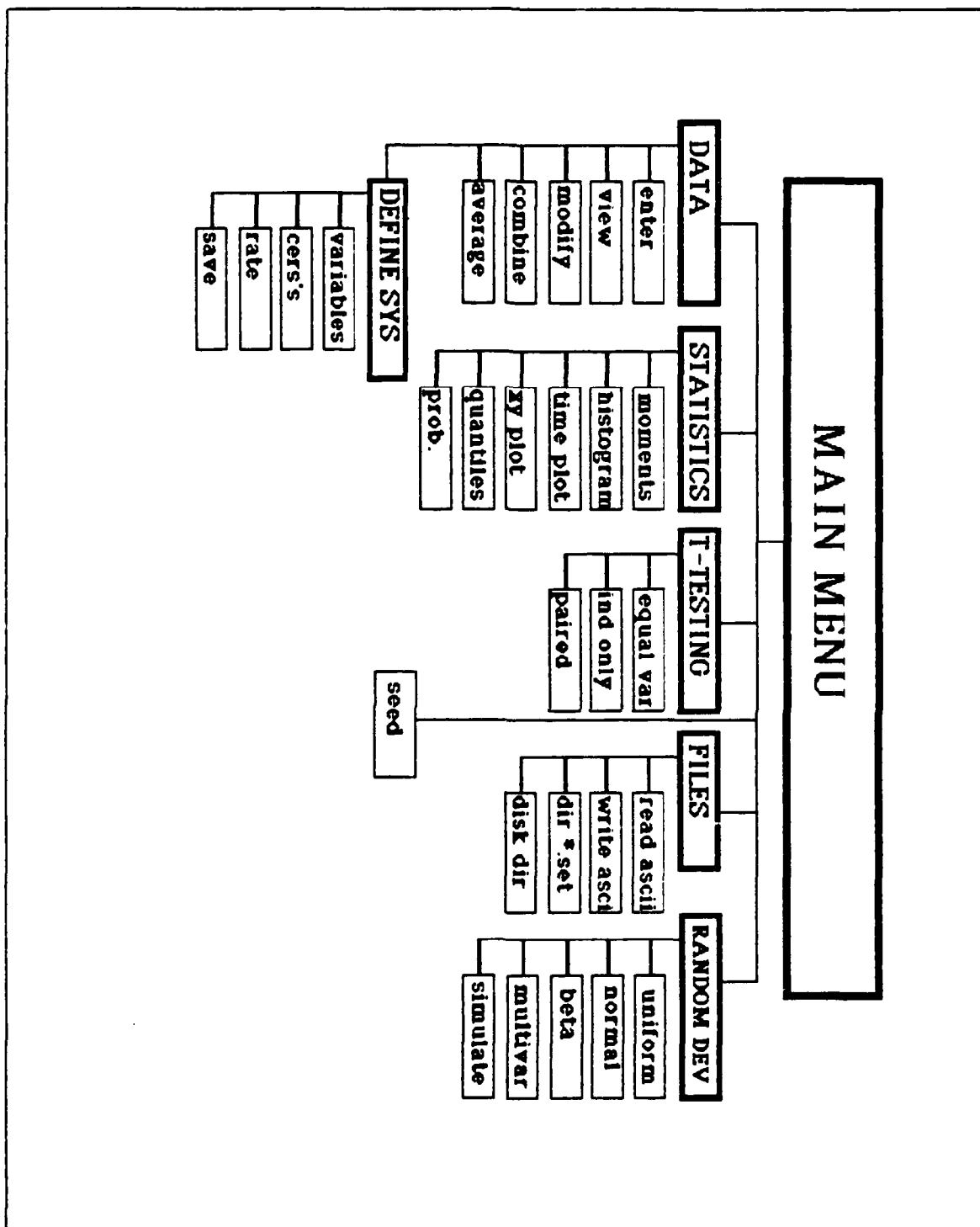


Figure 10 Menu layout for Model Builder.

AVAILABLE FUNCTIONS

DATA MENU - The data menu handles the 'administrative' duties such as data entry, correction, combination, and viewing. One mathematical routine, averaging, is offered.

ENTER - Allows for data entry from the keyboard. Saves the data set to a .SET file for use with the statistical functions. See READ ASCII under the file menu. After selecting this item a work window will be opened and the user will be prompted for a file name under which to store the new data set. If the file name entered already appears on disk, the user will be notified and asked if he wishes to proceed (overwrite). Answering yes will not overwrite the file yet. Enter one data value at a time, pressing the carriage return (CR) after each value. After typing in all the values, enter a "q" at the next prompt. The user will get a chance to change the name of the new data set, or overwrite the old data set if a data set with the same name already exists.

VIEW - This option writes the data values to the screen. It will scroll without pausing if there are more values than space on the screen. The <control S> keystroke can be used to stop the scrolling, but a better idea is to write the data to an ASCII file (see FILE menu) and view it with your favorite editor.

MODIFY - Allows the user to change any data point in a set providing the user knows the number of the point that

needs to be changed. Not very useful for large data sets. It is easier to write the data to an ASCII file, edit with your own editor, then read the ASCII file back in (see READ ASCII and WRITE ASCII at the FILE menu).

COMBINE - Prompts the user for two data set names. Reads the sets from disk, concatenates the second file onto the end of the first and saves the new data set to disk.

AVERAGE - The only mathematical routine on the DATA menu. Will prompt the user for a data set name. After reading the set from disk the routine will ask for another data set name. The values of the two sets will be averaged, ie. the mean of the first value of each set, then the mean of the second value from each set. The routine will continue to prompt the user until the user enters "q" to indicate that no more files should be averaged. This is useful for averaging the observations between simulations runs without averaging within the runs, for tasks such as beginning of steady state identification.

DEFINE SYSTEM - This item calls another menu, the LCC system definition menu. This is where cost simulation begins. Here a system is defined, with variables and CER's, so a Monte Carlo simulation can be run to forecast the cost distribution of the system in question.

VARIABLES - This item allows the user to define the single variable cost components. The user will be prompted for a filename. If the file already exists, the

"-- File read --" message will indicate that the user is updating a file that already exists. If the name is new and unique, the "-- New file --" message will indicate so. Now enter the number of single variable cost components for this system. For each variable enter the variable type, an integer from Figure 11, along with the appropriate parameters as they are prompted for. Be sure to know the NOCOST, PI, CON, and PO values before sitting down to define the variables.

CER'S - Prompts for information defining CER's. File name prompt and message are exactly like above. Enter the number of CER's, and the number of the CER to be defined now (each CER must be defined with a separate pass through this option). As the prompts suggest, enter all the information associated with the CER. First the number of input variables for the CER. Next define each variable and its power transformation. If no transformation is desired enter "1.0".

After the input variables are defined, enter the estimated β parameters. Next enter the covariances, paying careful attention to which covariance the routine is asking for (eg. $\text{cov}[1,1]=\text{var}[\beta_1]$, $\text{cov}[1,2]=\text{cov}[\beta_1, \beta_2]$). All that remains is to enter the MSE from the regression used to determine the CER.

As noted above this routine must be called three times to define three CER's. It is a good idea to save the file

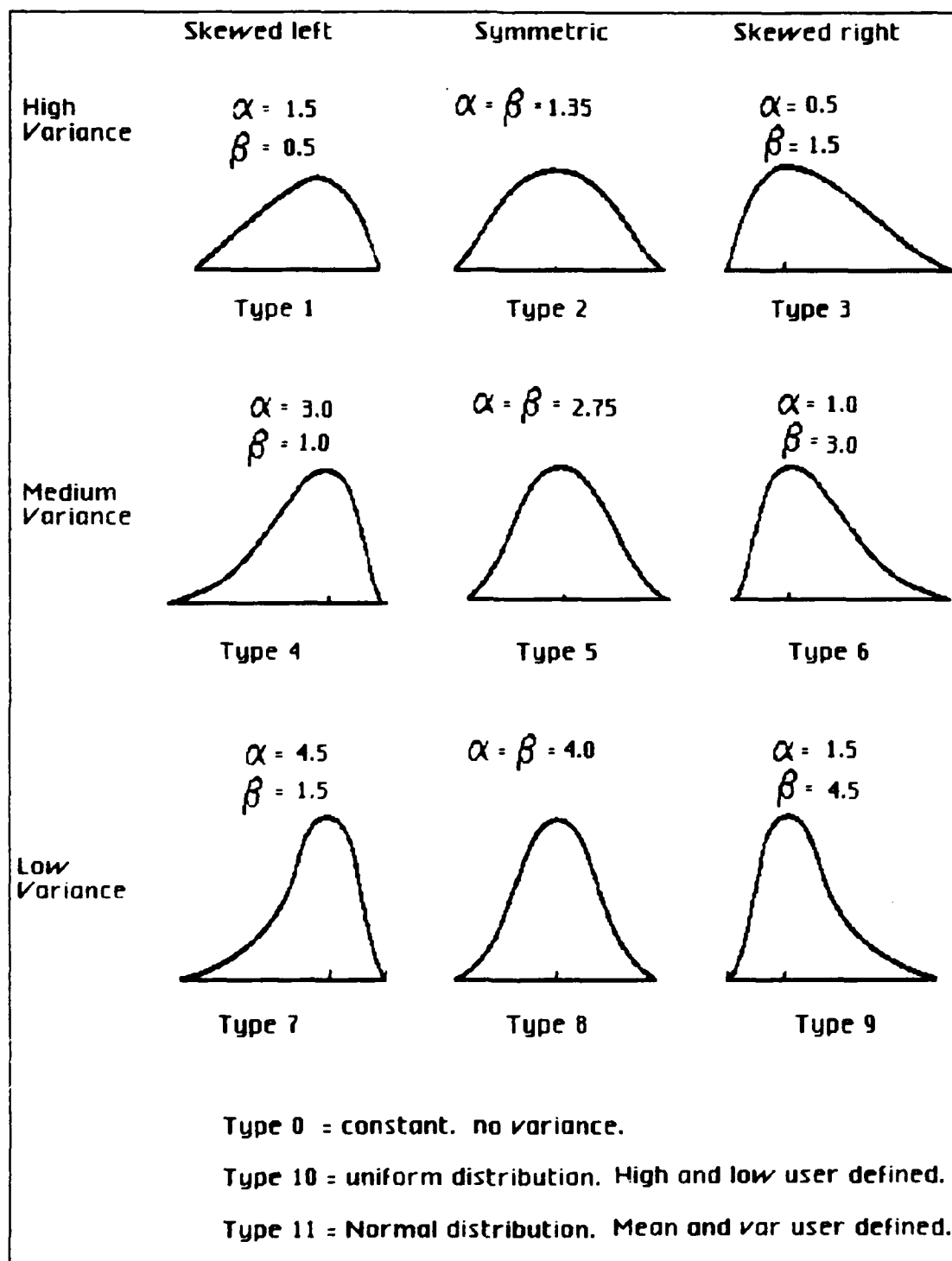


Figure 11 Available random number distributions (modified from Dienemann:14)

each time a CER is added, just in case.

RATE - Prompts the user for the interest rate to use for the present values. This interest rate should be entered as an Annual Percentage Rate (APR). For example if the current interest rate is 10% annually, enter 10.0. If no interest rate is desired enter 0.0. There must be a value for interest rate.

SAVE - Saves the information entered to a system definition file with a .STM extension for use with the Monte Carlo simulation module. It is prudent to save the system file between each phase of definition (variables, CER's, and interest rate).

STATISTICS MENU - This is the section of the program that does most of the actual work. Nearly all the numerical processing routines are handled here.

MOMENTS - Prompts user for a data set name. Calculates the mean, median, variance, standard deviation, low, hi, and number of data points for the set. Additionally a 1 axis plot of the data is created to facilitate identification of outliers, or clusters.

Figure 12 shows the display generated by this routine. The two tick marks near the center of the number line are the mean and the median (color coded with the words in the table no the actual screen). Note that the data points are lined up along a number line. Note also that if more than

mean. The jackknife statistics are calculated only at the user's request (by selecting "J" while viewing the plot).

FREQUENCY HISTOGRAM - After prompting the user for a data set name, this routine will ask a series of questions. listed below are the options corresponding to the questions.

1) Printout - Answer by selecting "y" or "n". This forces a screen dump, making a hardcopy of the graph. See the conventions section for more information.

2) Number of classes - Answer by entering an integer between 5 and 20, inclusive. Sets the number of classes, or ranges, and the number of bars on the graph.

3) Automatic classing - Answer by selecting "y" or "n". If "y" is chosen the range of data will be separated into N equal width ranges, N being the chosen number of classes. Otherwise, the user will be prompted for the upper limit for each class. This is useful for viewing two distributions under the class structure.

Figure 13 offers a sample histogram with nine classes. The number of classes will vary with number of samples and type of data, but nine is usually a good starting place.

TIME PLOT - After prompting user for a data set name, this routine will offer the option of using a moving average. If the user enters "y", he will then be prompted for any odd integer, specifying the size of the moving average window. This will result in the plotting of the moving average points, rather than the actual data. This

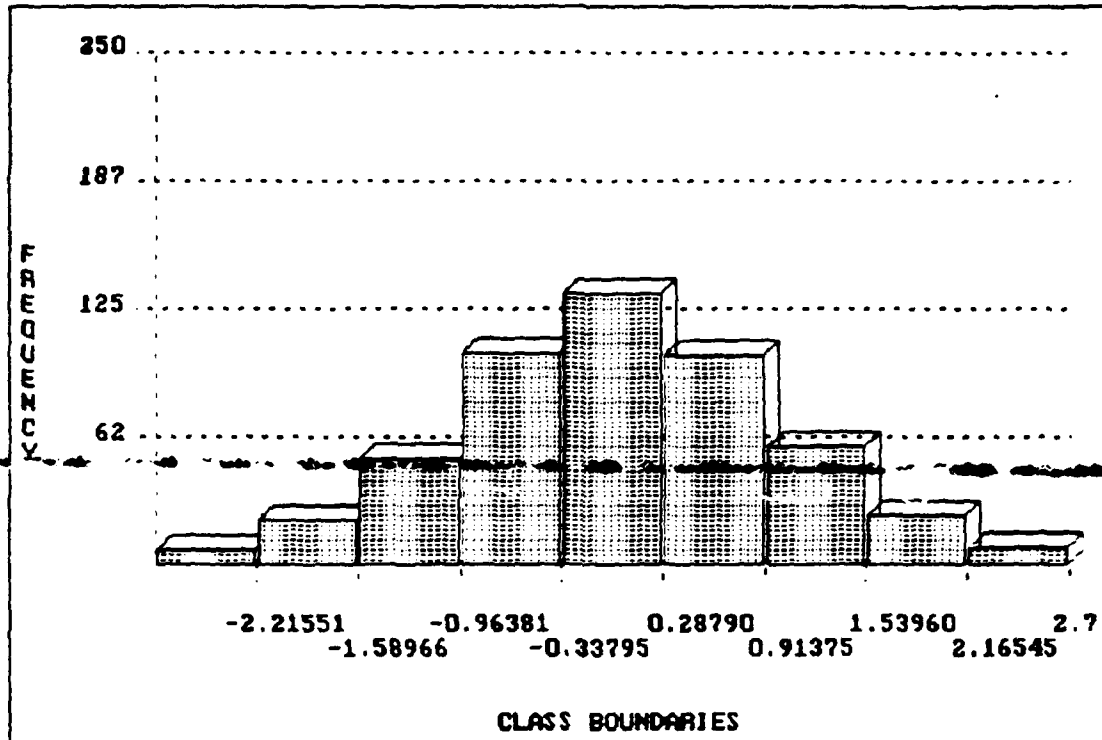


Figure 13 Example of a histogram routine output.

option is good for checking the data for serial correlation. This is not useful for the output of this model building program.

Figure 14 shows an example of a time series data plot. Note that the text displays the size of the moving average window (0 if moving average is not used). The line through the data represents the mean (color coded to the word mean and the number on the actual screen). Only the first 500 data points will be plotted.

XY PLOT - Provides an X vs Y plot of two data sets. The user is prompted for two data set names. One axis is scaled to one data set, the other axis is scaled to the

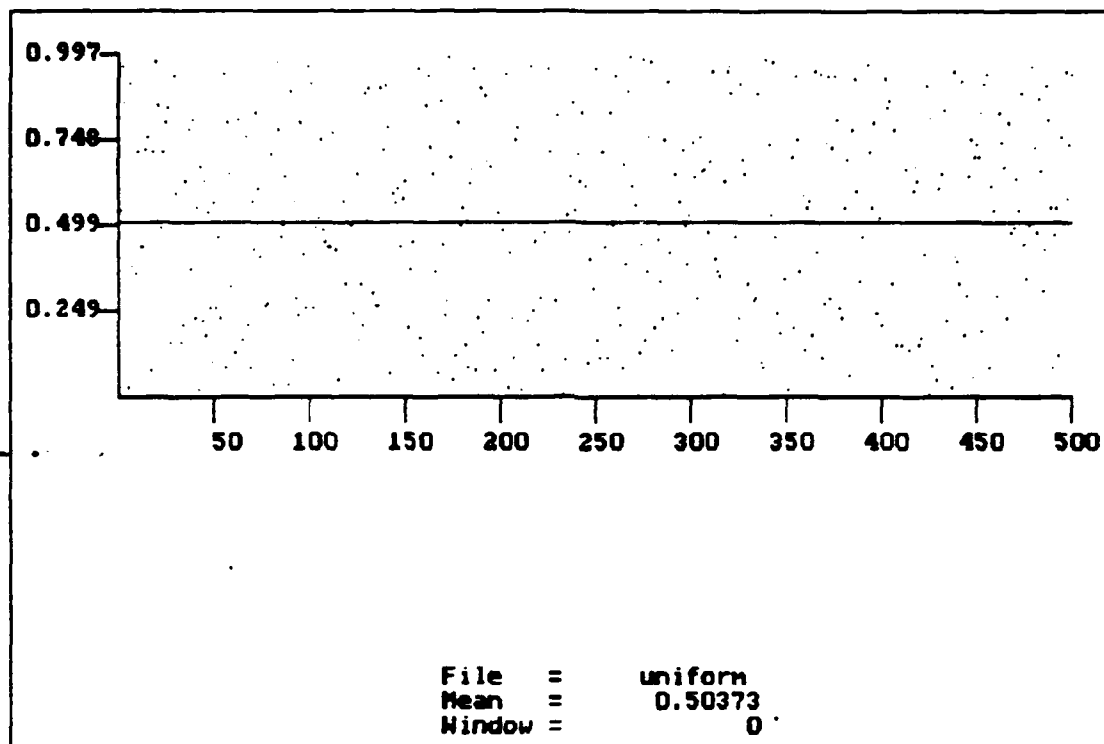


Figure 14 Time series option output.

other data set. The data points are plotted as pairs such as (dataA[1],dataB[1]), (dataA[2],dataB[2]), and so on. The means of the two data sets are drawn as lines through the appropriate axis (these are color coded with the file names on the screen). This option is useful for identifying correlation between two data sets. A discernable pattern, suggests that there may be some dependence or relationship between the two variables. Figure 15 shows a graph made by the XYplot option.

QUANTILE ESTIMATION - After the user is prompted for the data set name, the quantile estimates will be displayed on the screen. Point estimates and 90% confidence intervals

will be provided for 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 25%, and 75%.

NON-PARAMETRIC PROBABILITY - This routine will prompt the user for a filename, like the QUANTILE routine. Then the routine will prompt for a reference value. The

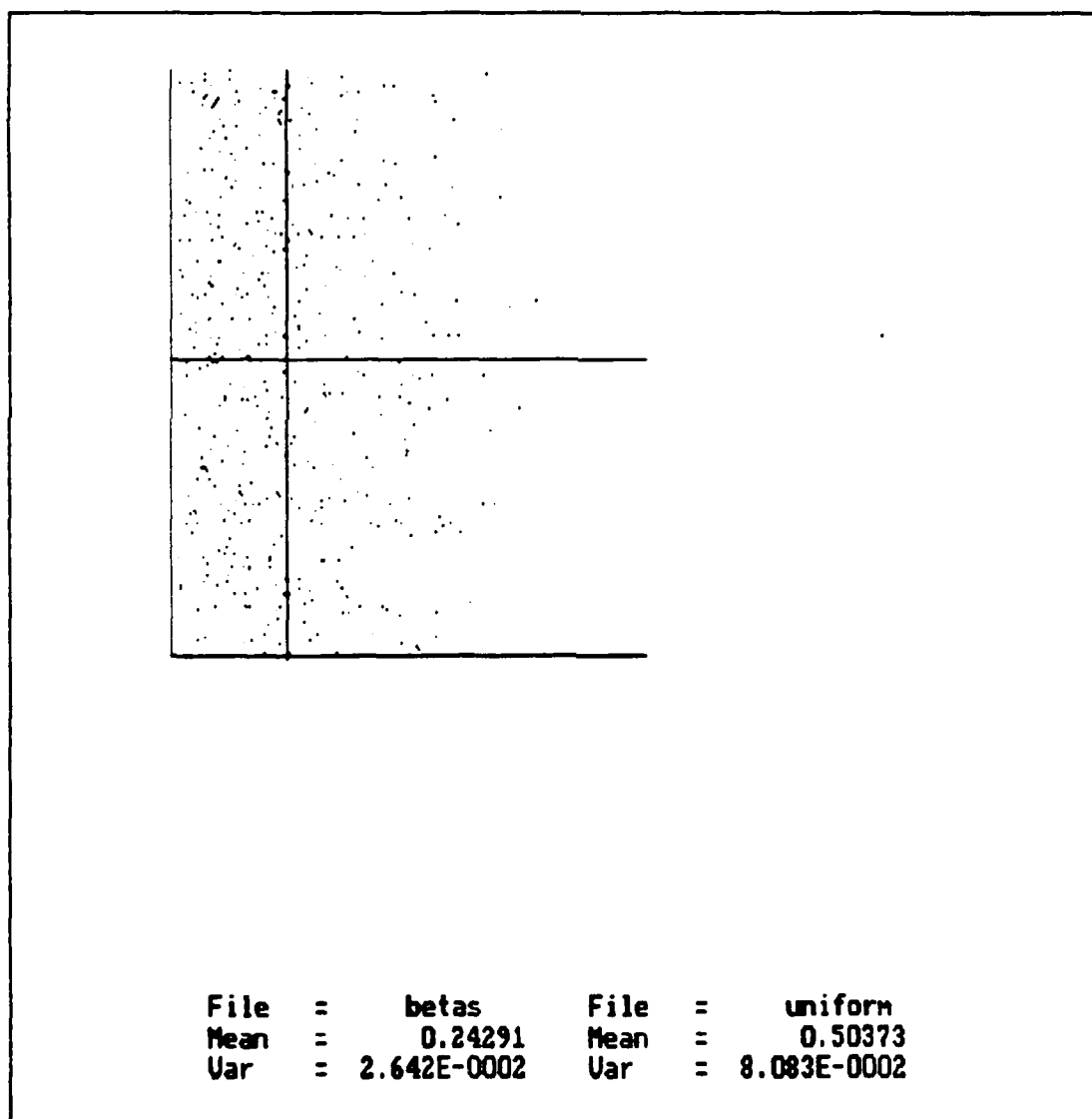


Figure 15 Example of an XY plot.

routine will display the probability that the next value drawn from the data set will be less than the reference value. A 90% confidence interval is provided.

T-TESTING MENU - This menu offers the three T-tests described in Chapter III. For more information concerning the assumptions associated with each test, see Chapter III.

T-TEST 1 - The user will be asked for the names of the two data sets to compare. Next the user will be prompted for the F-test α value. After the F-test is performed with failure to reject, the user will be prompted for the α value for the T-test. The results will be displayed.

T-TESTS 2&3 - The user will be asked for the names of the two data sets to compare. Next the user will be prompted for the T-test α value. The results will be displayed.

FILES MENU - This menu handles importing and exporting ASCII text files and checking the contents of the current disk drive.

READ ASCII - Reads a single column of textual numbers into a data set file. Multiple columns and nonnumeric characters are not allowed. The user is prompted for the name of the ASCII file to read (include the

extension when answering this prompt) and then the name under which to store the data set.

WRITE ASCII - Writes a single column of textual numbers into an ASCII file. The user is prompted for the name of the data set file holding the information and then the name of the ASCII file to write (include the extension when answering this prompt).

SET DIRECTORY - Provides a list of the files on the current disk drive that have the .SET extension.

DISK DIRECTORY - Provides a list of the files on the current disk drive according to the file specification entered by the user. For example, to see all the files that have the .DAT extension, enter *.DAT at the file specification prompt.

RANDOM DEVIATES MENU - This menu handles the routines that generate random deviates. The user may write deviates to a file or use them to simulate Life Cycle Costs.

RANDOM DEVIATE FILES - The first three options, Uniform Random, Normal Random, and Generate Betas prompt the user for the appropriate distribution parameters, how many deviates to generate, and the name of the .SET file in which to save the deviates.

MULTI-NORMAL - Generates vectors of dependent normal variates. First the user is prompted for the number of variables in the dependent set, ten is the maximum allowed.

Next the user is prompted for the mean of each variable. After the means are entered, the user is prompted to enter the covariances of the variables. The user is also asked how many vectors to generate. Each set of samples is written as a row vector to the file MULTNORM.DAT.

SIMULATE COSTS - This is the routine that generates Monte Carlo samples of LCC's. The user will be prompted for the name of the system (.STM) file in which the costs are described. See DESCRIBE SYS under the DATA menu for more information. The user will then be prompted for the number of runs, 2500 is the maximum. Finally the user is asked for a file name for the resulting cost estimates.

TUTORIAL

This section of the users' manual is a walk through of the software with a numerical example. A system will be described, entered into the program, and its cost simulated. Let it be clear that this example is grossly over simplified, but it will serve to exercise all the routines of the program. The statistical description procedures are not walked through; only data organization, entry, and simulation is covered here.

THE SYSTEM - John is starting a flying club. He needs to estimate the cost of purchasing and operating a fleet of 6 small aircraft with a life expectancy of ten years. From data about past purchases and operation of similar aircraft John has developed a set of equations representing the costs associated with this fleet of aircraft.

John will have two years of setup and aircraft acquisition, after which the fleet of aircraft will be put into service for ten years. So the entire process will stretch out for twelve years. The current interest rate is 9.5%, and is expected to hold steady for at least twelve years. Following are the equations and their explanations.

The first cost incurred is a setup fee, money needed up front for maintenance equipment and facility renovation. John knows how much this will cost as he has contracted this

out, having already negotiated a package price of \$32,000. This cost is incurred in the first year:

$$C_{\text{SETUP}} = X_1$$

where

$$X_1 = 32,000$$

Next John must purchase the aircraft. The cost of the aircraft fleet has been estimated by the following equation (CER):

$$C_{\text{AIRCRAFT}} = X_1^{\beta_1} * X_2^{\beta_2}$$

where

X_1 = personnel capacity of aircraft

X_2 = thickness of aluminum skin

β_1 = estimated at 5.2

β_2 = estimated at 0.5

$\text{cov}(\beta) = \text{estimated at } 1 = 1.7, 2 = 0.05, 1, 2 = 0.02$

$\text{MSE} = 1.2$

John has decided that X_1 will be 4 people. The aircraft manufacturer has notified John that the thickness of the aluminum sheeting (X_2) is usually uniformly distributed between 0.25" and 0.37". Since the aircraft must be purchased after the first year of setup, this cost will be incurred in year 2.

The next cost is the maintenance of the aircraft. It has been shown that the maintenance and operation cost for

six aircraft of this type can be forecasted by the following equation:

$$C_{M\&O} = \beta_1 X_1 + \beta_2 X_2^{\alpha_2}$$

where

X_1 = flying hours per plane per year
 X_2 = average humidity over aircraft life
 β_1 = estimated at 1,380
 β_2 = estimated at 22,000
 $\text{cov}(\underline{\beta})$ = estimated at 1= 205, 2= 1307, 1,2= 125.9
 α_2 = 0.5
 MSE = 15,700

Having researched flying clubs in other areas of similar size and demographic composition, John has estimated the flying hours per aircraft per year to be beta distributed between 1,000 and 1,500 hours, with $\alpha = 0.5$ and $\beta = 1.5$ (beta type 3, see figure 11). The local weather archives reveal that the area's average relative humidity over a ten year span is beta distributed between 0.5 and 0.8, with parameters $\alpha = 4.5$ and $\beta = 1.5$ (beta type 7, see figure 11). Since the aircraft will not fly for the first two years, the maintenance and operation costs will be incurred from year three through year twelve.

The last cost John has to worry about is a set of replacement engines. The FAA has made a ruling that small aircraft engines and their propeller speed reduction gears must be replaced after each five years of duty. John has negotiated with the local aircraft parts retailer and has settled on a price scale for the replacement engines. The

scale depends on the number of engines purchase. The more you buy, the cheaper they are:

$$C_{1 \text{ ENGINE}} = X_1^{\beta_1} * X_2^{\beta_2}$$

where

$$\begin{aligned} C_{1 \text{ ENGINE}} &= \text{average cost per unit} \\ X_1 &= \text{number of engines purchased (lot size)} \\ X_2 &= \text{number reduction gears per engine} \\ \beta_1 &= \text{estimated at 0.95} \\ \beta_2 &= \text{estimated at 0.3} \\ \text{cov}(\beta) &= \text{estimated at 1= 0, 2= 0.05, 1,2= 0} \\ \text{MSE} &= 1,200 \end{aligned}$$

This is a learning curve CER. John will be buying 6 engines (X_1) each having 4 reduction gears (X_2). Since the original engines will be replaced five years after start of service, these new engines must be bought in year 7. Note that C is the average cost per unit. The cost of this cost component will be figured by multiplying that average cost per unit by the number of units to be purchaes, X_1 .

SETTING UP SPENDING TIMELINES

Now that all the costs are broken out, the value of NOCOST, PI, CON, and PO must be determined for each cost.

COST 1: SETUP - Figure 16 shows the spending timeline for this cost. The setup will occur in year 1, so the bill will be paid at the end of that year. Since this bill will be paid in its entirety the first year, there are no periods of NOCOST, PI or PO. There will simply be constant spending

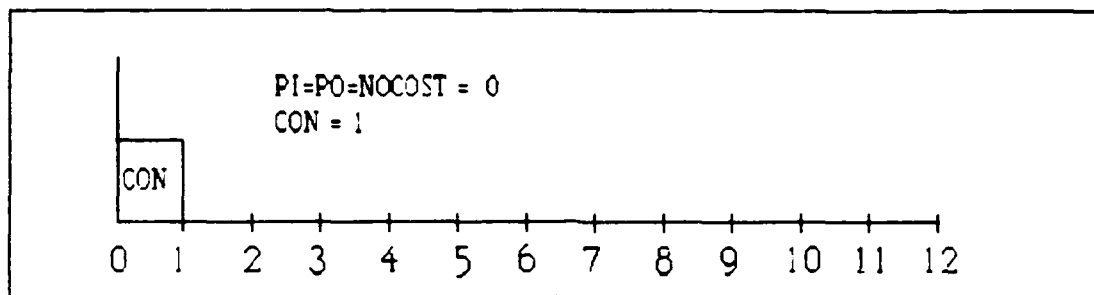


Figure 16 Spending timeline for Cost 1.

in year 1. So the value of CON is 1.

COST 2: AIRCRAFT PURCHASE - Figure 17 shows the spending timeline for this cost. The purchase will occur in year 2, so the bill will be paid at the end of that year.

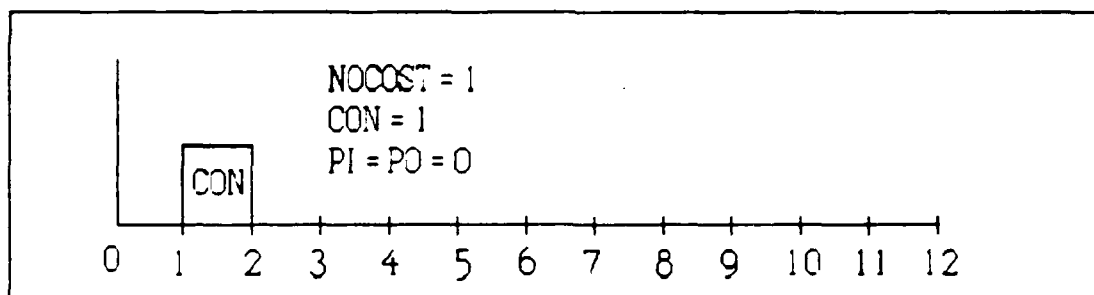


Figure 17 Spending timeline for Cost 2.

This is just like the case of setup above, except the lump sum is paid in year two. Since no money is due in year 1, the NOCOST period is 1 year. The entire cost will be paid in the single year following the NOCOST period (year two) so the CON period is 1 year long.

COST 3: MAINTENANCE AND OPERATION - Figure 18 shows the spending timeline for this cost. The aircraft will

begin to fly in year three. Since no money will be spent

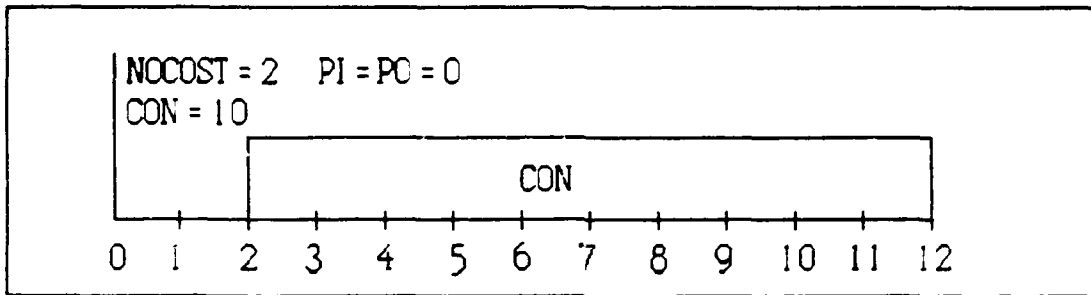


Figure 18 Spending timeline for Cost 3.

for M&O in years 1 and 2, the NOCOST period is 2 years. The maintenance costs will be incurred evenly over the operating life of the aircraft, years 3 through 12, or for 10 years. So the CON period is 10 years. Again there are no PI and PO periods.

COST 4: ADDITIONAL ENGINE PURCHASE - Figure 19 shows the spending timeline for this cost. The purchase of all the additional engines will occur in the fifth year of

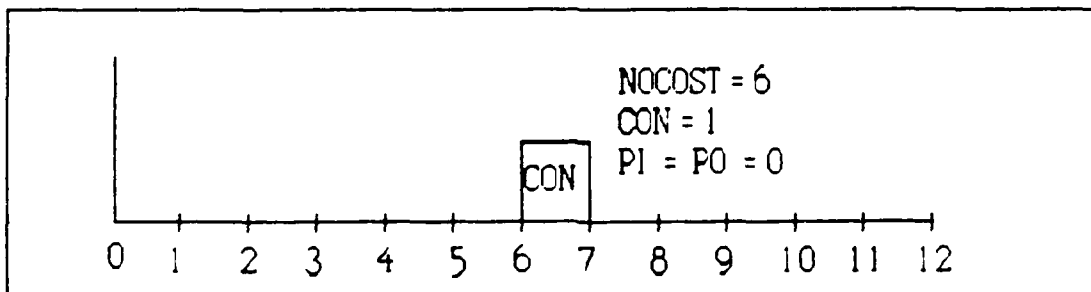


Figure 19 Spending timeline for Cost 4.

aircraft operation, year 7. So the bill will be paid at the end of that year. Since no money is due in years 1 through

6, the NOCOST period is 6 years. The entire cost will be paid in the single year following the NOCOST period (year two) so once again the CON period is 1 year long.

Table 1 lists the values of NOCOST, PI, CON, and PO for each of the costs described above:

Table 1

<u>Cost</u>	<u>NOCOST</u>	<u>PI</u>	<u>CON</u>	<u>PO</u>
SETUP	0	0	1	0
AIRCRAFT PURCHASE	1	0	1	0
MAINTENANCE & OPS	2	0	10	0
REPLACEMENT ENGINES	6	0	1	0

DATA ENTRY

To enter the data describing the system cost components, start the program by typing its name, THESIS. Select DATA from the opening menu by pressing D, or by using the down arrow key to highlight the DATA option, and pressing CR. Now select the DEFINE SYS option from the DATA menu by pressing D. Now you are ready to begin entering costs.

Begin by entering the only single variable cost component, setup costs. Remember that this cost was a

constant. Select VARs by pressing V. When prompted for a file name, enter JOHNFLY. Next you are prompted for the number of variables. Respond to the prompt by entering the number 1, because there is only 1 single variable cost component (the others are CER's).

A new screen appears and you are prompted for the type of the variable. Since our variable is a constant, enter a 0 (see Figure 11). When prompted for the constant value, enter 32000 (no commas). Next you are prompted for NOCOST, PI, CON, and PO in that order. Enter the values 0, 0, 1, and 0 in that order, pressing CR after each (see Table 1). Since there are no more single variable cost components, the program returns to the DEFINE SYS menu.

To save the information you have entered so far, press the S key to select SAVE. You are again prompted for a file name, but this time the name JOHNFLY is offered in parentheses as the default. Since JOHNFLY is the name you want, simply press the CR to accept it.

Next you need to enter the CER's. Select CER from the menu by pressing C. Again you are prompted for a file name, press CR to accept the default. When prompted for the number of CER's, enter the total, 3. Next enter the number of the CER you want to describe. Start with 1, which will be the MAINTENANCE AND OPERATIONS CER.

In response to the next prompt, enter the number of explanatory variables (number of β parameters), 2. Next is

the sequence of time parameter prompts for this CER. Enter the NOCOST, PI, CON, and PO values as they are asked for, just like in the single variable case. Next the screen clears and you are prompted for the type of the CER; regular (linear), natural logarithm, or learning curve. Select regular by pressing R.

The next group of prompts pertains to the explanatory variables of the CER. For each variable you are prompted for the type, high, and low (or mean and variance for normals) just like with the single variables discussed above. Since X_1 is distributed beta ($\alpha=0.5, \beta=1.5$) enter 3 for the type (see Figure 11). Next enter 1000 and 1500 as the low and high values. Since no power transformation is desired, enter 1.0 at the α prompt. For the next explanatory variable, follow the same sequence, except enter 0.5 for α value since that power transformation is indicated by the CER describing the maintenance costs.

After you have described the explanatory variables, you will be prompted for the estimates of the β parameters. Enter 1380 for the first β and 22000 for the second. Next you are prompted for the covariances. Be careful to note which covariance you are being prompted for. First the cov[1,1] appears. Enter 205 since it is the variance of β_1 . Next the cov[1,2] prompt appears. Enter 125.7. Finally the cov[2,2] prompt appears, so enter 1307, the

variance of β_2 . The last prompt is for the MSE. Enter 15700.

To enter the next two CER's follow the same steps. When entering the aircraft cost CER be sure to select natural logarithm as the CER type. Describe the X variables as the beta distributed variables they are. The program will handle the logarithms and exponentiation. Be sure to enter 1.0 for the transformation α 's.

For the replacement engines CER, select learning curve as the CER type. Be sure X_1 , the first explanatory variable, is the number of units to purchase. The CER outcome, $C_{1 \text{ ENGINE}}$ must be multiplied by X_1 to get the total cost of the entire lot of engines, since the CER describes the average cost per unit (engine).

The last step in data entry is to enter the current interest rate. Select RATE from the DEFINE SYS menu. Enter the APR, 9.5. Now save the system file one last time by pressing S and backup to the main menu using the B key.

RUNNING THE MONTE CARLO SIMULATION

To run the simulation, proceed to the RANDOM DEVIATES menu by selecting R and the main menu. Now select S to simulate costs. When prompted for the file name containing the system information (CER's, rate, etc.) enter JOHNFLY. Next you are prompted for the number of runs to perform, or the number of estimates to make. Enter any integer between

1 and 2500. You must also enter the name of the data set in which to store the cost estimates. You may choose any name, but it is best to keep the same name as the .STM file containing the CER information. Enter JOHNFLY. This will not erase the system information file.

SEEDS

The seed is set to a default of 12345 when the program is initiated. If you run an LCC simulation today with the default seed, and simulate the same system next week with the default seed, the output numbers will be exactly the same. You may change the seed upon initiating the program to give different random deviates. Changing the seed is not a dangerous procedure. The seed may be set to any integer between 1 and 32000.

OUTPUT ANALYSIS

You are now ready to use the statistical description routines provided by the program. For information concerning the significance of each routine see Chapter III. For specific program operation information see the functions section of this appendix.

APPENDIX B: RANDOM DEVIATE GENERATOR OUTPUT

OVERVIEW

The purpose of this appendix is to demonstrate the capabilities of the deviate generators used in the model building program. Checks of mean, variance, and serial autocorrelation are performed on each generator. There are more powerful tests for specific distributions available but they have not been performed. The purpose of this appendix is not to prove the theoretical accuracy of the random deviate sampling procedures, but to demonstrate that they have been coded according to the references provided, and that they have no major flaws. For more rigorous validation of the sampling procedures see the appropriate references.

The following sample statistics are provided via STATISTIX II. Additional statistics and graphs are provided from the model builder statistical functions, helping to validate by comparison those statistical functions that have been repeated).

UNIFORM(0,1)

Source: Press.

Type:

<u>500 element test set: Observed</u>		<u>Expected</u>
mean	0.5037	0.5000
var	0.0808	0.0833
low	0.0004	0.0000
high	0.9970	1.0000

Figures 20 and 21 are graphical depictions of the output deviates, created by the model builder program. Figure 18 shows the first 22 serial autocorrelations for the 500 element test set, as calculated by STATISTIX II.

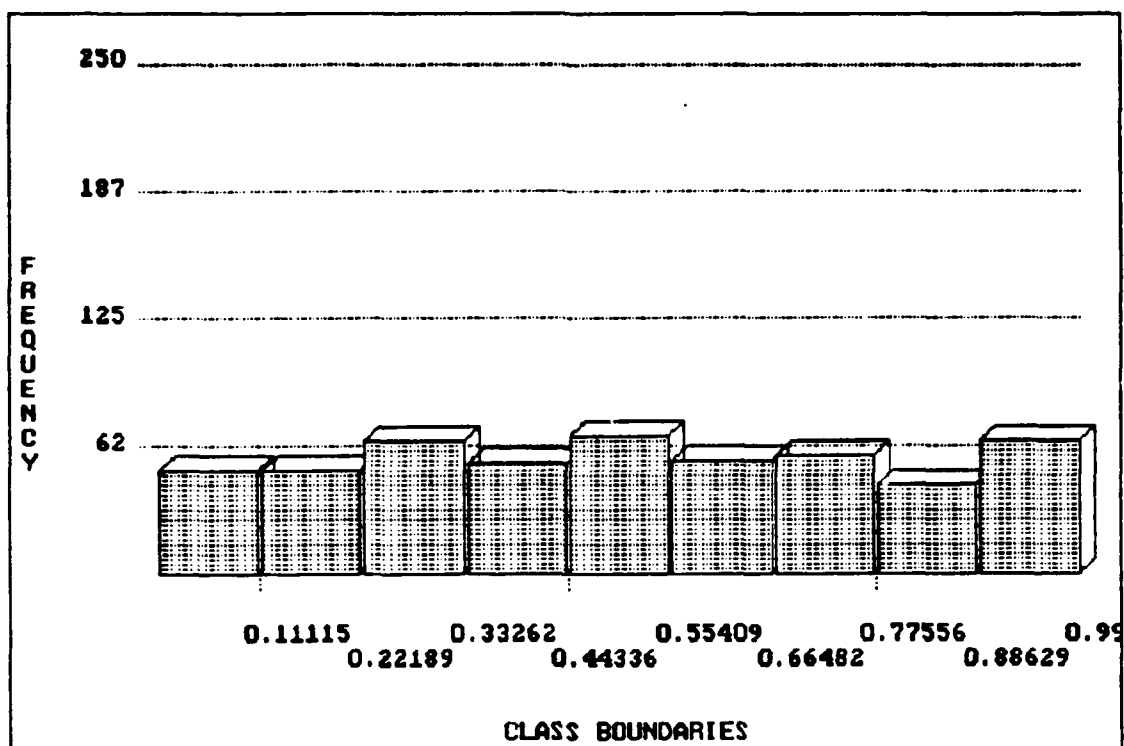


Figure 20 Uniform histogram

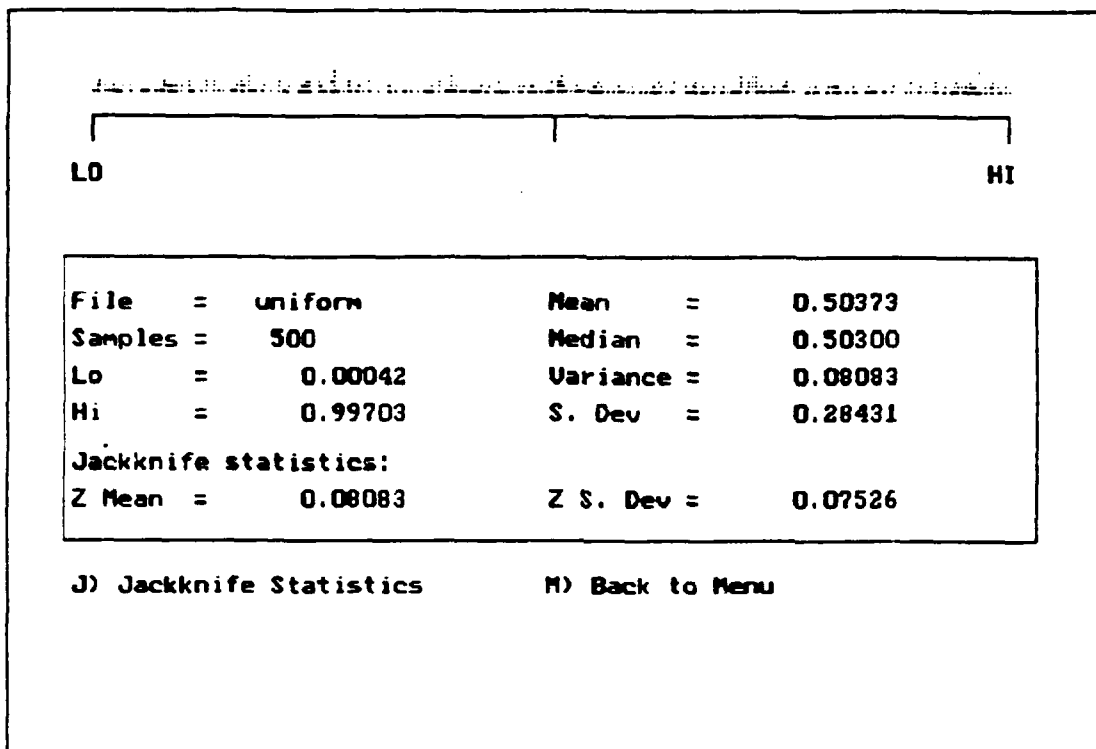


Figure 21 Uniform moments.



AUTOCORRELATION PLOT FOR UNIFORM

		-1.0	-0.8	-0.6	-0.4	-0.2	0.0	0.2	0.4	0.6
LAG	CORR.									
		+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
1	-0.021						> ** <			
2	0.022						> ** <			
3	-0.017						> * <			
4	0.063						> ** <			
5	-0.028						> ** <			
6	0.043						> ** <			
7	0.005						> * <			
8	-0.087						> ** <			
9	0.051						> ** <			
10	0.051						> ** <			
11	0.084						> ** <			
12	0.006						> * <			
13	0.021						> ** <			
14	0.035						> ** <			
15	0.008						> * <			
16	-0.038						> ** <			
17	-0.011						> * <			
18	-0.026						> ** <			
19	-0.089						> ** <			
20	0.096						> ** >			
21	-0.014						> * <			
22	0.008						> * <			
23	0.004						> * <			
24	-0.048						> ** <			
25	-0.091						> ** <			

MEAN OF THE SERIES	4.883E-01
STD. DEV. OF SERIES	2.924E-01
NUMBER OF CASES	500



Figure 22

NORMAL(0,1)

Source: Ross.

Type:

<u>500 element test set: Observed</u>		<u>Expected</u>
mean	0.0052	0.0000
var	0.9820	1.0000

Figures 23 and 24 are graphical depictions of the output deviates, created by the model builder program. Figure 25 shows the first 25 serial autocorrelations for the 500 element test set, as calculated by STATISTIX II.

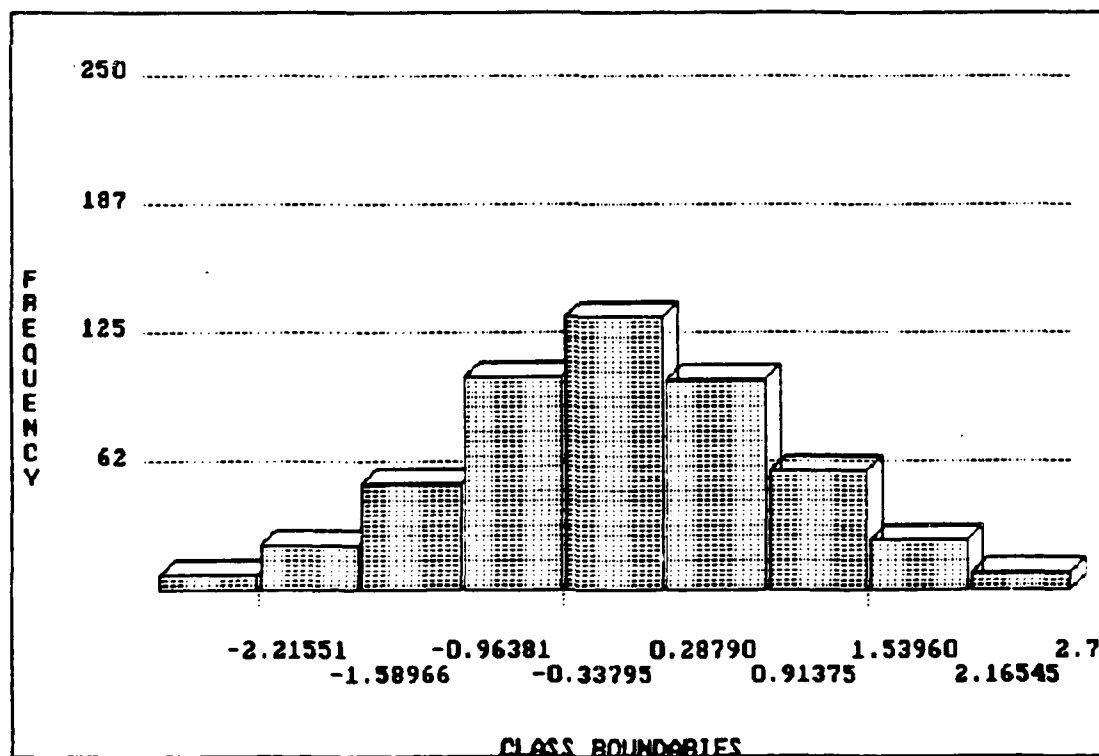


Figure 23 Normal histogram.

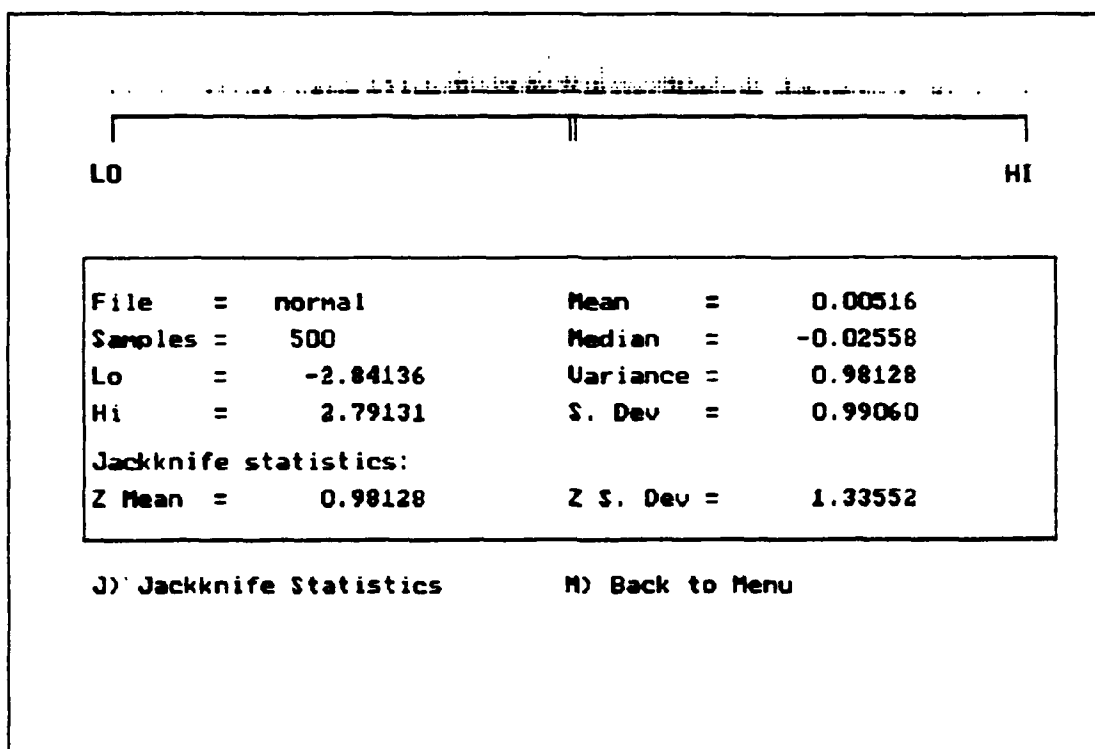


Figure 24 Normal moments.



AUTOCORRELATION PLOT FOR NORMAL

		-1.0	-0.8	-0.6	-0.4	-0.2	0.0	0.2	0.4	0.6
LAG	CORR.									
		+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
1	-0.083						> ** <			
2	0.022						> ** <			
3	0.090						> ** >			
4	0.003						> * <			
5	-0.011						> * <			
6	0.034						> ** <			
7	-0.020						> ** <			
8	0.066						> ** <			
9	-0.023						> ** <			
10	-0.031						> ** <			
11	0.028						> ** <			
12	0.015						> * <			
13	-0.038						> ** <			
14	0.025						> ** <			
15	0.074						> ** <			
16	-0.034						> ** <			
17	0.005						> * <			
18	-0.013						> * <			
19	-0.015						> * <			
20	-0.019						> * <			
21	0.006						> * <			
22	-0.069						> ** <			
23	0.004						> * <			
24	-0.059						> ** <			
25	0.008						> * <			

MEAN OF THE SERIES	5.160E-03
STD. DEV. OF SERIES	0.990
NUMBER OF CASES	500



Figure 25

BETA($\alpha=1.5, \beta=4.5$) ON (0,1) INTERVAL

Source: Ross, Fishman.

Type: Acceptance-Rejection.

<u>500 element test set: Observed</u>		<u>Expected</u>
mean	0.2447	0.2500
var	0.0258	0.0268
low	0.0048	0.0000
high	0.7109	1.0000

Figures 26 and 27 are graphical depictions of the output deviates, created by the model builder program. Figure 28 shows the first 25 serial autocorrelations for the 500 element test set, as calculated by STATISTIX II. Note that the 20th autocorrelation appears significant. Remembering the way risk builds when more than one confidence interval is considered simultaneously we could calculate that the probability of at least one autocorrelation being out of its 90% CI when no actual autocorrelations exist is over 0.95!

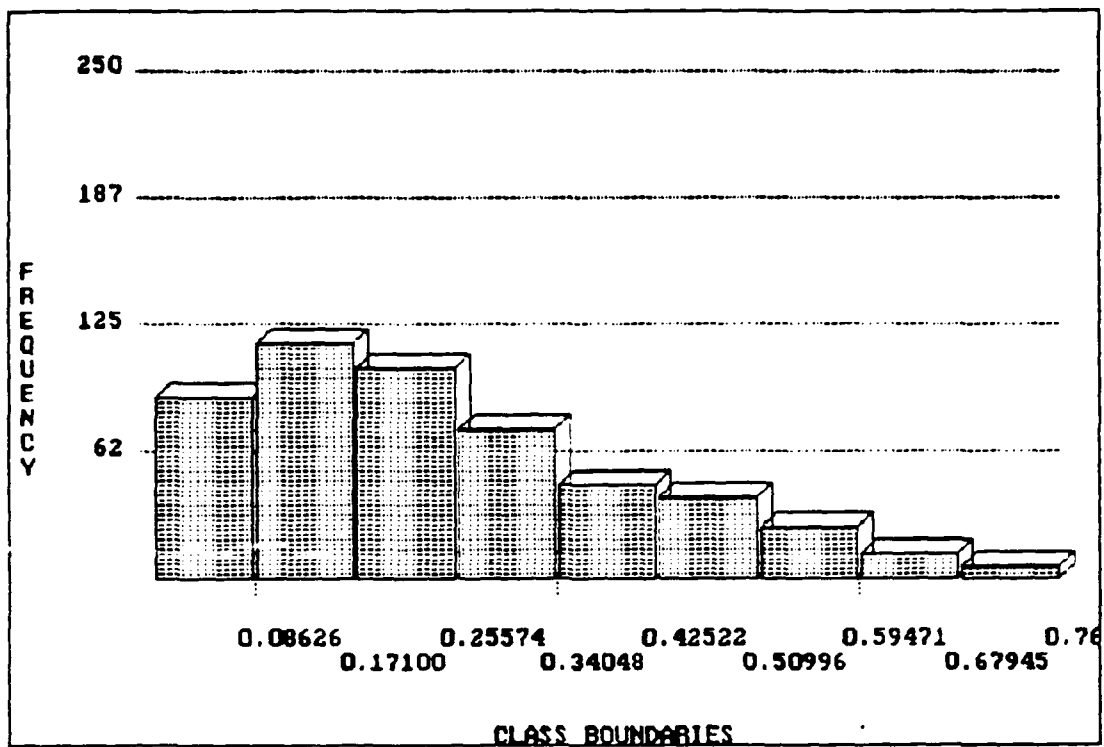


Figure 26 Beta histogram.

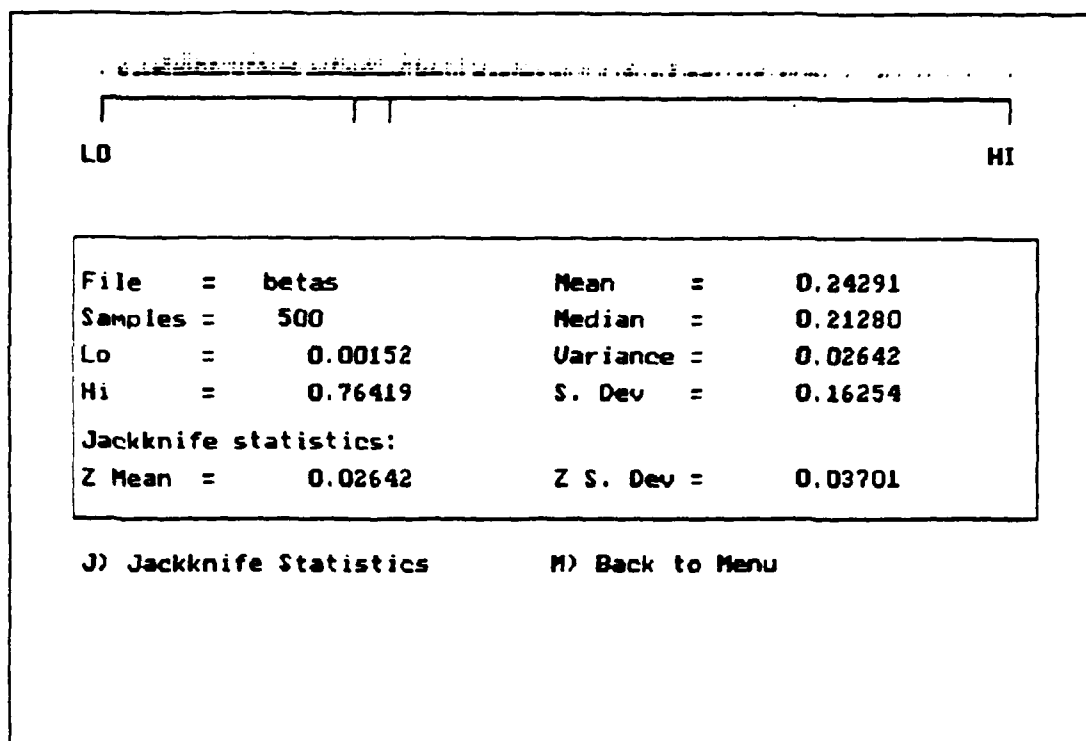


Figure 27 Beta moments.

AUTOCORRELATION PLOT FOR BETA

LAG	CORR.	-1.0	-0.8	-0.6	-0.4	-0.2	0.0	0.2	0.4	0.6
1	0.053						> **<			
2	-0.022						> **<			
3	-0.018						> *<			
4	-0.028						> **<			
5	0.050						> **<			
6	0.028						> **<			
7	0.052						> **<			
8	-0.088						> **<			
9	-0.064						> **<			
10	0.081						> **<			
11	-0.018						> *<			
12	-0.012						> *<			
13	0.063						> **<			
14	0.001						> *<			
15	0.039						> **<			
16	0.007						> *<			
17	0.026						> **<			
18	0.023						> **<			
19	0.018						> *<			
20	0.102						> **>*			
21	-0.025						> **<			
22	-0.036						> **<			
23	-0.067						> **<			
24	-0.004						> *<			
25	0.056						> **<			

MEAN OF THE SERIES 2.447E-01
STD. DEV. OF SERIES 1.605E-01
NUMBER OF CASES 500

Figure 28

MULTIVARIATE NORMAL

Source: IMSL User Guide

Type:

500 element test set: Observed

Expected

mean	x1	4.985	5.000
	x2	10.09	10.00
	x3	15.03	15.00

Covariances

Observed

Expected

2.957	0.578	1.994	3.0	1.0	2.0
0.578	8.338	2.413	1.0	9.0	3.0
1.994	2.413	4.912	2.0	3.0	5.0

APPENDIX C: TURBO PASCAL 4.0 SOURCE CODE

```
{S+}    {Stack checking on}
{N+}    {numeric coprocessor yes}
{$= 64000,0,65536}
```

PROGRAM thesis;

Uses

```
Crt,
Dos,
Printer,
Graph,
dave-menu, (* original menuing unit written by me *)
dave-stat; (* original unit written by me, houses many statistical routines *)
```

```
(* **** *)
(* Note: the type and variable declarations, along with many of the *)
(* statistical and file handling routines used in this program *)
(* are in the unit DAVESTAT. Some other routines along with all *)
(* the homemade scrolling menu control routines are in the DAVEMENU *)
(* unit. Since those units have been included with the USES *)
(* statement, the program acts as if they were listed here. *)
(* **** *)
```

```
procedure multinormal(var betas:doubvect;chol:tensq);
(* This is an original procedure written for multinormal sampling using the *)
(* cholesky square root of the covariance matrix. See IMSL for algorithm. *)
(* This routine is used for generating dependent betas for cers evaluation. *)
var
  n,x          :integer;
  temp2,temp    :double;
begin
  n:=trunc(betas[0]); (* This is the number of betas in this CER *)
  mat2.matrix:=chol;  (* PUT the cholesky square root of the cov matrix in MAT2 *)
  mat2.rowsize:=n;    (* These next 4 lines setup the sizes of working matrices *)
  mat2.colsize:=n;
  mat1.colsize:=n;
  mat1.rowsize:=1;
  (* draw the independent normals and put them in a vector *)
  for x:= 1 to n do begin
    temp      := ran3(seed);
    temp2     := ran3(seed);
    mat1.matrix[1,x] := sqrt(-2*ln(temp))*sin(2*pi*temp2);
  end;
  (* Postmultiply independent normals by cholesky square root of the covariance matrix *)
  MULTmat;
  (* Now add in the means to complete the dependent multinormal samples *)
  for x:=1 to n do
    betas[x] := mat3.matrix[1,x] + betas[x];
  end;
```

```
PROCEDURE MANUAL(SETG:INFO;VAR NUMARRAY:VECT;NUMG:INTEGER;VAR BOUNDARIES:twenvect);
(* this allows the user to enter his own class boundaries for the *)
(* frequency histogram routine. *)
VAR X,V      :INTEGER;
    MIN,WIDTH : double;
    go       :boolean;
```

```

(* original *)
BEGIN
  FOR X:= 1 TO NUMG DO
    BEGIN
      WRITE('Upper boundary for class ',X,' ');double read(BOUNDARIES[X]);
      writeln('      (inclusive)');
      writeln;
    END;

    for x:=1 to 20 do (* initialize the observed frequency vector *)
      numarray[x]:=0;
    numarray[0]:=1;

    for x:=1 to numg do begin (* Now count the observed frequencies, on class at a time *)
      go:=true; (* The data will be sorted before it *)
      for y:= numarray[0] to setg.size do (* is sent to this routine, so using *)
        if (setg.data[y]<=boundaries[x]) and go then (* the go flag allows me to make only *)
          numarray[x]:=numarray[x]+1 (* 1 complete pass through the data to *)
        else go:=false; (* count class frequencies. *)
      numarray[0]:=numarray[0]+numarray[x];
    end;
  END;

```

```

(* original *)
PROCEDURE autoclass(SETG:INFO;VAR NUMARRAY:VECT;NUMG:INTEGER;VAR BOUNDARIES:twenvect);
(* this automatically sets the class boundaries for the histogram *)
(* it divides the data range into equal class widths *)
(* The data is sorted before it comes to this routine, like the routine above. *)
VAR X,Y :INTEGER;
    MIN,WIDTH : double;
    cummfreq :twenvect;
    stuff :filename;
    go :boolean;
BEGIN
  MIN := SETG.DATA[1];
  WIDTH := (SETG.DATA[SETG.SIZE] - MIN)/NUMG;

  FOR X := 1 TO (NUMG) DO
    BOUNDARIES[X]:= MIN+WIDTH*X;
  for x:=1 to 20 do
    numarray[x]:=0;
  numarray[0]:=1;

  for x:=1 to numg do begin (* counting class frequencies just like procedure above *)
    go:=true;
    for y:= numarray[0] to setg.size do
      if (setg.data[y]<=boundaries[x]) and go then
        numarray[x]:=numarray[x]+1
      else go:=false;
    numarray[0]:=numarray[0]+numarray[x];
  end;
END;

```

```

(* deQue *)
PROCEDURE ANNAXIS(XORI,YORI,XLEN,NUMX:INTEGER;XTXTARR:ANNARRAYTYPE);
(* annotates the x axis for graphs *)
VAR y1,TEXTLENGTH,realxg,J,NUMTODD:INTEGER;
    SPACING :double;
BEGIN
  SPACING:= XLEN/NUMX;
  NUMTODD := NUMX +1;
  FOR j:= 1 TO NUMTODD DO
    BEGIN
      TEXTLENGTH := LENGTH(XTXTARR[J]);
      REALXG := round(XORI+(J-1)*(SPACING)-(8*textlength/2));
      y1:=280;
    END;
  END;

```

```

    if odd(j) then y1 := 290;
    outtextxy(realxg,y1,xtextarr[j]);
END;
XTEXTARR[0] := 'CLASS BOUNDARIES';
outtextxy(280,320,xtextarr[0]);
END;

```

```

(* deQue *)
PROCEDURE ANNYXIS(XORI, YORI, VLEN, NUMY: INTEGER; YTEXTARR: ANNARRAYTYPE);
(* annotates the y axis for graphs *)
VAR TEXTLENGTH, XTEXT, YTEXT, J, NUMTODD: INTEGER;
    SPACING, REALXG, REALYG : double;
BEGIN
    SPACING := VLEN/NUMY;
    NUMTODD := NUMY + 1;
    FOR J := 1 TO NUMTODD DO
        BEGIN
            TEXTLENGTH := LENGTH(YTEXTARR[J]);
            REALXG := XORI - (TEXTLENGTH * 8.0) - 15;
            REALYG := YORI - SPACING*(J-1) - 4;
            outtextxy(round(realxg),round(realyg),ytextarr[j]);
        END;
        YTEXTARR[0] := 'FREQUENCY';
        TEXTLENGTH := LENGTH(YTEXTARR[0]);
        FOR j := 1 TO TEXTLENGTH DO
            outtextxy(30,120+j*10,ytextarr[0][j]);
        END;
    END;

```

```

(* deQue *)
PROCEDURE AXISTEXT(VAR GTEXTARR: ANNARRAYTYPE; NUMG: INTEGER; LOWVAL, HIVAL: double;
    REALFORMAT: BOOLEAN);
(* writes the text at the x axis *)
VAR INTERVAL, REALVAL: double;
    INTVAL, J : INTEGER;
BEGIN
    gtextarr[1] := ' ';
    INTERVAL := (HIVAL - LOWVAL)/NUMG;
    IF REALFORMAT THEN
        FOR J := 1 TO NUMG DO
            BEGIN
                REALVAL := LOWVAL + j * INTERVAL;
                STR(REALVAL:12:5,GTEXTARR[J+1]);
            END
        ELSE
            FOR J := 1 TO NUMG DO
                BEGIN
                    INTVAL := TRUNC(0.5*(LOWVAL + TRUNC(INTERVAL)*J));
                    STR(INTVAL:3,GTEXTARR[J+1]);
                END;
            END;
    END;

```

```

(* deQue *)
PROCEDURE MANXAXIS(VAR GTEXTARR: ANNARRAYTYPE; NUMG: INTEGER; BOUNDARIES: twenvect;
(* writes the x axis text when manual class boundaries are used *)
    REALFORMAT: BOOLEAN);
VAR INTERVAL, REALVAL: double;
    INTVAL, J : INTEGER;
BEGIN
    gtextarr[1] := ' ';
    IF REALFORMAT THEN
        FOR J := 1 TO NUMG DO
            BEGIN

```

```

        STR(BOUNDARIES[J]:12:5,GTEXTARR[J+1])
    END
ELSE
    FOR J:= 1 TO NUMG DO
        BEGIN
            INTVAL := TRUNC(BOUNDARIES[J]);
            STR(INTVAL:3,GTEXTARR[J+1]);
        END;
    END;
END;

```

```

(* deQue *)
PROCEDURE DOXAXIS(XORI,YORI,XLEN,NUMX:INTEGER);
(* this actually draws the lines for the x axis itself *)
VAR X,J,K :INTEGER;
    TEMP : double;
BEGIN
    LINE(XORI,YORI,XORI+XLEN,YORI);
    FOR j:= 1 TO NUMX DO
        BEGIN
            TEMP := XLEN/NUMX*j;
            X := XORI + TRUNC(TEMP);
            LINE(X,YORI,X,YORI+7);
        END
    END;
END;

```

```

(* deQue *)
PROCEDURE DOYAXIS(XORI,YORI,YLEN,NUMY,XLEN:INTEGER);
(* just like doxaxis except for the y axis *)
VAR Y,J,K :INTEGER;
    TEMP : double;
BEGIN
    LINE(XORI,YORI,XORI,YORI-YLEN);
    FOR j:= 1 TO NUMY DO
        BEGIN
            TEMP :=round(YLEN/NUMY)*j;
            Y := YORI - ROUND(TEMP);
            LINE(XORI-10,Y,XORI+XLEN,Y);
        END
    END;
END;

```

```

procedure graphix;
(* sets up the computer for the EGA 640x480 color mode *)
var
    GraphDriver,Graphmode,ErrorCode      :integer; { The Graphics device driver }
begin
    (* INITIALIZE THE EGA 4 COLOR GRAPHICS SCREEN 600h X 350v *)
    graphdriver:=4;
    graphmode:=1;
    InitGraph(graphdriver,graphmode,''); { activate graphics }
    ErrorCode := GraphResult;           { error? }
    if ErrorCode <> grOk then begin
        writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
        Halt(1);
    end;
end;

```

```

(* highly modified deQue *)
PROCEDURE HISTOGRAM;
(* draws a frequency histogram based on a data set *)
VAR NUMARRAY          :VECT;
    BOUNDARIES         :twenvect;

```

```

GTEXTARR                                :ANNARRAYTYPE;
XORI, YORI, XLEN, YLEN, X, Y            :INTEGER;
NUMX, NUMY, ULX, ULY, LEX, LRY, J, MAXCLASS :INTEGER;
LOX, HIX, LOY, HIY, SCALEX, SCALEY, BARWIDTH :double;
REPLY, REPLY2, AUTO, PRYNT              :CHAR;
REALFORMATX, REALFORMATY               :BOOLEAN;
setg                                     :infopt;
p                                       :pointer;
BEGIN
  mark(p);
  new(setg);
  setwindow(36,10,78,18);
  READLN(setg);
  IF (not EXIST(setg.name)) or (setg.SIZE<=2) THEN begin
    release(p);
    exit;
  end;
  MAXCLASS:=0;
  writeln;
  repeat
    write( 'How many classes? (5-20): ');intread(numx);writeln;
  until (numx>4) and (numx<21);

  writeln;
  WRITE( 'Do you want a printout? : ');PRYNT:=READKEY;writeln(' ',prynt);
  writeln;
  IF NUMX=0 THEN NUMX:=1;
  write( 'Automatic classing? : ');AUTO:=READKEY;writeln(' ',auto);
  writeln;

  NUMY:= 4;
  FOR X:= 0 TO NUMX DO NUMARRAY[X]:=0;
  WRITE( ' -- Please wait -- ');
  SORTAR(setg);
  IF UPCASE(AUTO)= 'Y' THEN autoclass(setg,NUMARRAY,NUMX,BOUNDARIES)
  ELSE begin
    MANUAL(setg,NUMARRAY,NUMX,BOUNDARIES);
    WRITE( ' -- Please wait -- ');
  end;
  FOR Y:= 1 TO NUMX DO
    IF NUMARRAY[X] > MAXCLASS THEN MAXCLASS := NUMARRAY[X];
  XORI := 100; (* left edge of graph *)
  YORI := 259; (* top edge of graph *)
  XLEN := 480; (* length of x axis *)
  YLEN := 210; (* length of y axis *)
  REALFORMATX:= TRUE; (* says to show x axis label numbers as reals *)
  REALFORMATY:= FALSE; (* says to show y axis label numbers as integers *)
  graphx; (* initialize the graphics card mode *)
  SETCOLOR(RED); (* these three lines draw the x and y axes in red *)
  DOXAXIS(XORI,YORI,XLEN,NUMX); (* draw x axis *)
  DOYAXIS(XORI,YORI,YLEN,NUMY,XLEN); (* draw y axis *)
  LOX:=setg.DATA[1]; (* these two lines get hi and low values, data is sorted *)
  HIX:=setg.DATA[setg.SIZE];
  SETCOLOR(WHITE);

  (* this next block figures class boundaries, observed frequencies, and writes the axis text*)
  IF UPCASE(AUTO)= 'Y' THEN AXISTEXT(GTEXTARR,NUMX,LOX,HIX,REALFORMATX)
  ELSE MANXAXIS(GTEXTARR,NUMX,BOUNDARIES,REALFORMATX);
  ANNAXIS(XORI,YORI,XLEN,NUMX,GTEXTARR);
  AXISTEXT(GTEXTARR,NUMY,0,setg.SIZE,REALFORMATY);
  ANNAXIS(XORI,YORI,YLEN,NUMY,GTEXTARR);
  SCALEY := 2.0*YLEN/setg.SIZE;
  BARWIDTH := XLEN/NUMX;

  (* now draw the bars for the histogram based on observed frequencies per class *)
  FOR j:= 1 TO NUMX DO BEGIN
    (* this code calculates the proper height for each *)
    (* bar of the histogram and draws each bar in 3-dimensional form *)
    ULX := XORI + TRUNC(BARWIDTH * (J-1));

```

```

    ULV := TRUNC(YORI - NUMARRAY[J] * SCALEY);
    LRX := XORI + TRUNC(BARWIDTH*J);
    bar3D(ULX+1,ULV,LRX-1,YORI,B,TRUE);
    (*BAR (ULX+1,ULV,LRX-1,YORI): TO HAVE 2-D RATHER THAN 3-D *)
END;

IF UPCASE(PRVNT)='V' THEN PRVTC;
PAUSE:=READKEY;
RESTORECROMODE;
release(p);
END;

```

```

(* original *)
PROCEDURE MOMENTS;
(* calculates mean,var,median,and jackknifing for a data set *)
(* along with a horizontal plot for identifying outliers *)
(* and checking the spread of the data *)
VAR PRVNT, _ac :CHAR;
    I,J,L,R,X,MID,XORI,YORI,XLEN,x1,y1 :integer;
    GraphDriver, GraphMode, ErrorCode :INTEGER;
    Num, MEDIAN, LOX, HIX, XSCALE, MEAN, VARIANCE :double;
    STUFF :FILENAME;
    pixelcolor :word;
    setg :infopt;
    p :pointer;
BEGIN
    mark(p);
    ne:=setg;
    setwindow(36,10,76,18);
    READIN(setg);
    IF (not EXIST(setg.name)) or (setg.SIZE <= 2) THEN begin
        release(p);
        exit;
    end;
    writeln;
    WRITE('Do you want a printout? : ');PRVNT:=READKEY;
    writeln;
    write(' -- Please wait -- ');
    MEANVAR(setg,MEAN,VARIANCE);
    SORTHR(setg);
    LOX :=setg.DATA[1];
    HIX :=setg.DATA[setg.SIZE];
    XORI := 60;
    YORI :=148;
    XLEN := 480;
    if hix-lox < 0.00001 then begin
        writeln('No spread in the data, all = ',lox);
        pause:=readkey;
        release(p);
        exit;
    end;
    XSCALE:=XLEN/(HIX-LOX);

    (* find the median *)
    IF ODD(setg.SIZE) THEN MEDIAN := setg.DATA[(setg.SIZE + 1) DIV 2]
    ELSE BEGIN
        M10 := setg.SIZE DIV 2;
        MEDIAN := (setg.DATA[M10] + setg.DATA[M10 + 1]) / 2.0
    END;

    (* DRAW THE NUMBER LINE *)
    graphix;
    SETCOLOR(WHITE);
    LINE(XORI,YORI,XORI+XLEN,YORI);
    LINE(XORI,YORI,XORI,YORI+10);
    LINE(XORI+XLEN,YORI,XORI+XLEN,YORI+10);
    OUTTEXTXY(XORI-10,YORI+20,'LO');

```

```

OUTTEXTXY(XORI+XLEN-10,YORI+20,'HI');

(* PLOT THE MEAN TICK MARK and the Median tick mark *)
SETCOLOR(RED);
LINE(ROUND((MEAN-LOX)*XSCALE)+XORI,YORI+10,ROUND((MEAN-LOX)*XSCALE)+XORI,YORI);
setcolor(lightgreen);
LINE(ROUND((Median-LOX)*XSCALE)+XORI,YORI+10,ROUND((Median-LOX)*XSCALE)+XORI,YORI);

(* NOW PLOT EACH OF THE DATA POINTS *)
SETCOLOR(YELLOW);
FOR j:= 1 TO setg^.SIZE DO begin
  x1:=round((setg^.DATA[j]-LOX)*XSCALE)+XORI;
  y1:=YORI-10;
  pixelcolor:=getpixel(x1,y1);
  while pixelcolor=yellow do begin (* IF THIS POINT IS ALREADY PLOTTED, *)
    y1:=y1-2; (* THEN GO UP TWO DOTS TO INDICATE THAT *)
    pixelcolor:=getpixel(x1,y1); (* MORE THAN ONE POINT IS PLOTTED HERE *)
  end;
  PUTPIXEL(x1,y1,yellow);
END;

(* Now draw the tabular data box at the bottom of the screen *)
SETCOLOR(WHITE);
LINE(45,205,45,320);
LINE(45,205,555,205);
LINE(555,205,555,320);
LINE(45,320,555,320);

(* Now convert all the data to strings and write the strings *)
outtextxy(50,220,'File = '+setg^.NAME);
STR(setg^.SIZE:6,STUFF); outtextxy(50,235,'Samples = '+STUFF);
STR(LOX:12:5,stuff); outtextxy(50,250,'Lo = '+stuff);
STR(HIX:12:5,stuff); outtextxy(50,265,'Hi = '+stuff);
SETCOLOR(RED);
STR(mean:12:5,stuff); outtextxy(300,220,'Mean = '+stuff);
setcolor(lightgreen);
STR(MEDIAN:12:5,STUFF); outtextxy(300,235,'Median = '+stuff);
setcolor(white);
STR(VARIANCE:12:5,STUFF); outtextxy(300,250,'Variance = '+STUFF);
STR(SQR(VARIANCE):12:5,STUFF); outtextxy(300,265,'S. Dev = '+STUFF);
outtextxy(50,335,'J) Jackknife Statistics M) Back to Menu');

jac:=readkey;
IF UPCASE(JAC)='J' THEN BEGIN
  setcolor(white+blink);
  outtextxy(50,285,'Jackknife statistics:');
  JACK(setg,MEAN,VARIANCE);
  STR(MEAN:12:5,STUFF); outtextxy(50,300,'Z Mean = '+stuff);
  str(sqrt(variance):12:5,stuff);outtextxy(300,300,'Z S. Dev = '+stuff);
  pause:=readkey;
end;
IF UPCASE(PRVNT)='V' THEN PRNTSC;
restorecrtmode;
release(p);
END;

(* original *)
PROCEDURE PLOTit;
(* this plots one data set against another, basically an x y plot *)
VAR stuff :filename;
tempset,setg,seth :infopt;
PRVNT :CHAR;
V,J,XLEN,L,R,X,MID,XORI,YORI,XLEN,x1,y1 :INTEGER;
xscale,YSCALE,NUM,MEDIAN,LOX,HIX,MEAN,VARIANCE :double;

```

```

        hi,hmean,hvariance,loh,hih                :double;
        p                                           :pointer;
BEGIN
    mark(p);
    new(tempset);new(setg);new(seth);
    setwindow(36,10,78,18);
    READIN(setg);
    if (not EXIST(setg^.name)) or (setg^.SIZE <= 2) THEN begin
        release(p);
        exit;
    end;
    readin(seth);
    if (not EXIST(seth^.name)) or (seth^.SIZE <= 2) THEN begin
        release(p);
        exit;
    end;
    write'n;
    WRITE('Do you want a printout? : ');PRVNT:=READKEY;write(prvnt);
    write'n;
    write'n(  -- PLEASE WAIT --');
    WRITELN;
    MEANVAR(setg^,MEAN,VARIANCE);
    tempset^:=setg^;
    SORTHR(setg^);
    LOX:=setg^.DATA[1];
    HIX:=setg^.DATA[setg^.SIZE];
    setg^:=tempset^;
    MEANVAR(seth^,hmean,hvariance);
    tempset^:=seth^;
    SORTHR(seth^);
    LOh:=seth^.DATA[1];
    HIh:=seth^.DATA[seth^.SIZE];
    seth^:=tempset^;
    XORI  := 80;
    YORI  :=210;
    XLEN  :=250;
    YLEN:=190;
    if hix>hih then hih:=hix;
    if hih>hix then hix:=hih;
    YSCALE:=YLEN/(HIh);
    xscale:=xlen/(hih);

    (* draw the axes *)
    graphix;
    SETCOLOR(WHITE);
    LINE(XORI,YORI,XORI+XLEN,YORI);
    LINE(XORI,YORI,XORI,YORI-YLEN);

    (* PLOT THE MEAN LINE and WRITE THE MEAN AND FILE NAME *)
    SETCOLOR(RED);
    LINE(XORI+1,ROUND(YORI-MEAN*YSCALE),XORI+XLEN,ROUND(YORI-MEAN*YSCALE));
    str(mean:12:5,stuff);outtextxy(300,330,'Mean  = '+stuff);
    OUTTEXTXY(300,320,'File   =   '+setg^.name);
    str(variance:12,stuff);outtextxy(300,340,'Var    = '+stuff);

    SETCOLOR(green);
    LINE(round(xORI+hmean*xSCALE),yori+1,round(xORI+hmean*xscale),YORI-ylen);
    str(hmean:12:5,stuff);outtextxy(100,330,'Mean  = '+stuff);
    OUTTEXTXY(100,320,'File   =   '+seth^.name);
    str(hvariance:12,stuff);outtextxy(100,340,'Var    = '+stuff);
    setcolor(white);

    (* NOW PLOT EACH OF THE DATA POINTS *)
    SETCOLOR(YELLOW);
    FOR j:= 1 TO setg^.size DO BEGIN
        x1:=round(XORI+seth^.data[j]*XSCALE);
        y1:=YORI-TRUNC(setg^.DATA[j]*YSCALE);
        PUTPIXEL(x1,y1,yellow);
    
```

```

end;
IF UPCASE(PRVNT)='Y' THEN PRTSC;
pause:=readkey;
restorecrtmode;
release(p);
END;

```

```

(* original *)
(* this plots the data in a time series fashion *)
(* it can plot a moving average of any odd size window *)
PROCEDURE TIMEPLOT;
VAR AVERAGES, setg          :INFOptr;
    MOVING, PRVNT, jac      :CHAR;
    XSCALE, WINDOW, Y, J, YLEN, L, R, X, MID, XORI, YORI, XLEN, x1, y1 :INTEGER;
    SUM, YSCALE, NUM, MEDIAN, LOX, HIX, MEAN, VARIANCE :double;
    STUFF                  :FILENAME;
    point1, point2         :integer;
    pixelcolor             :word;
    p                      :pointer;
BEGIN
    mark(p);
    new(setg); new(averages);
    setwindow(36, 10, 78, 18);
    READIN(setg);
    IF (not EXIST(setg^.name)) or (setg^.SIZE <= 2) THEN begin
        release(p);
        exit;
    end;
    window:=0;
    averages:=setg;
    writeln;
    WRITE('Do you want a printout? : '); PRVNT:=READKEY;
    writeln;
    WRITE('Use moving averages? : '); MOVING:=READKEY; writeln(' ', moving);
    writeln;
    IF UPCASE(MOVING)='Y' THEN BEGIN
        WINDOW:=2;
        while not odd(window) do begin
            WRITE('window size for averages (ODD): '); intREAD(WINDOW);
            writeln;
        end;
        writeln(' -- Please Wait -- ');
        for x:= 1 to (setg^.size-window) do begin
            (* now calculate the moving average values *)
            sum:=0;
            for y:= 1 to window do
                sum:= sum+setg^.data[x+y];
            averages^.data[x+(trunc(window/2)+1)]:=sum/window;
        end;
    END;
    ELSE writeln(' -- Please Wait -- ');
    MEANVAR(setg, MEAN, VARIANCE);
    SORTHR(setg);
    LOX:=setg^.DATA[1];
    HIX:=setg^.DATA[setg^.SIZE];
    XORI := 80;
    YORI := 210;
    XLEN := 500;
    YLEN:=140;
    YSCALE:=YLEN/(HIX);
    graphix;
    (* DRAW THE NUMBER LINE *)
    XSCALE:=TRUNC(500/setg^.SIZE);
    if xscale=0 then xscale:=1;
    SETCOLOR(WHITE);
    LINE(XORI, YORI, XORI+XLEN, YORI);
    LINE(XORI, YORI, XORI, YORI-YLEN);

```

```

FOR X:= 1 TO TRUNC(10/XSCALE) DO BEGIN
  LINE(XORI+X*50*XSCALE, VORI, XORI+X*50*XSCALE, VORI+10);
  STR(X*50:3, STUFF);
  OUTTEXTXY(XORI+X*50*XSCALE-8, VORI+15, STUFF);
END;
FOR X:= 1 TO 4 DO BEGIN
  LINE(XORI, VORI-X*35, XORI-10, VORI-X*35);
  STR((HIX/4)*X:9:3, STUFF);
  OUTTEXTXY(XORI-80, VORI-(X*35)-4, STUFF);
END;

(* PLOT THE MEAN LINE and WRITE THE MEAN AND FILE NAME *)
SETCOLOR(RED);
LINE(XORI+1, ROUND(VORI-MEAN*YSCALE), XORI+XLEN, ROUND(VORI-MEAN*YSCALE));
str(mean:12:5, stuff); outtextxy(250, 330, 'Mean = '+stuff);
setcolor(white);
OUTTEXTXY(250, 320, 'File = '+setg^.name);
str(window:12, stuff); outtextxy(250, 340, 'window = '+stuff);

(* NOW PLOT EACH OF THE DATA POINTS *)
SETCOLOR(YELLOW);
point1:=1;
POINT2:=setg^.SIZE;
IF WINDOW<> 0 THEN begin
  (* calculate the first and last points who will *)
  (* be plotted depending on the window size, *)
  (* since some data points do not have a moving average *)
  point1:=POINT1+1+TRUNC(WINDOW/2);
  point2:=point2-trunc(window/2);
end;
FOR j:= point1 TO POINT2 DO BEGIN
  x1:=round(XORI+J*XSCALE);
  y1:=VORI-TRUNC(averages^.DATA[J]*YSCALE);
  PUTPIXEL(x1, y1, yellow);
end;
IF UPCASE(PRINT)='V' THEN PRTSC;
pause:=readkey;
restorecrtmode;
release(p);
END;

```

```

(* original *)
PROCEDURE COMBINE;
(* this appends a second data set to the end of a data set *)
VAR X      : INTEGER;
    set1, set2 : infoPtr;
    p       : pointer;
BEGIN
  mark(p);
  new(set1); new(set2);
  setwindow(36, 10, 78, 18);
  TWOSETS(set1, set2);
  IF (not EXIST(set1^.name)) or (not EXIST(set2^.name)) THEN exit;
  WRITE('Name the new set : '); READLN(set2^.NAME);
  writeln;
  FOR X:= 1 TO set1^.SIZE DO
    set2^.DATA[X+set1^.SIZE] := set1^.DATA[X];
  set2^.SIZE := set1^.SIZE + set2^.SIZE;
  SAVE(set2);
  release(p);
END;

```

```

(* original *)

```

```

Procedure ftestvar(seta,setb:info;var equal:boolean);
(* this performs an f-test for equal variances between two data sets *)
var pvalue,alpha,fstat,vara,varb,meana,meanb :double;
    temp                                     :string[15];
    df1,df2,code                            :integer;
begin
equal:=true;
writeln;
writeln('Performing F test for Equal variances ...');
writeln;
meanvar(seta,meana,vara);
meanvar(setb,meanb,varb);
(* calculate the test statistic and set the degrees of freedom *)
if vara>varb then begin
    fstat:=vara/varb;
    df1:=seta.size-1;
    df2:=setb.size-1;
end
else begin
    fstat:=varb/vara;
    df1:=setb.size-1;
    df2:=seta.size-1;
end;
(* now calculate the p value *)
pvalue:=fvalue(fstat,df1,df2);
write('Enter alpha level : ');double read(alpha);
writeln;
if pvalue > alpha then
    writeln('F test passed ...')
else begin
    equal:=false;
    writeln('F test failed ...');
end;
writeln('var A = ',vara:12:5);
writeln('var B = ',varb:12:5);
writeln('Any key continues ...');
pause:=readkey;
end;

```

```

(* original *)
procedure ttest;
(* a t-test for two sets with equal variances, independent sets *)
var equal                               :boolean;
    alpha,pvalue,tstat,spooled,diff,vara,varb,meana,meanb :double;
    df                                   :integer;
    seta,setb                           :infoptr;
    p                                    :pointer;

```

```

begin
mark(p);
new(seta);new(setb);
twoSETS(seta,setb);
IF (not EXIST(seta^.name)) or (not EXIST(setb^.name)) then begin
    release(p);
    exit;
end;
(* first check to see if variances are equal *)
ftestvar(seta,setb,equal);
if not equal then begin
    release(p);
    exit;
end;
meanvar(seta,meana,vara);
meanvar(setb,meanb,varb);
diff:=meana-meanb;
df:=seta.size+setb.size-2;
(* find pooled standard dev *)

```

```

spooled:=sqrt(((seta^.size-1)*vara + (setb^.size-1)*varb)/df);
if spooled < 0.0000001 then begin
  writeln('s pooled = ',spooled:12:5);
  writeln;
  writeln('s pooled < 0.0000001 ...');
  writeln('leaving the test routine ...');
  pause:=readkey;
  release(p);
  exit
end;
(* calculate the test statistic *)
tstat:=abs(diff/spooled*sqrt(1/seta^.size+1/setb^.size)); (* use this to integrate t and get p value *)
writeln;
writeln('Now performing T test. ');
write('Enter the alpha level : ');doubleread(alpha);
pvalue:=tvalue(tstat,df);
if pvalue > alpha/2 then begin
  writeln('No evidence to reject hypothesis ...');
end
else begin
  writeln;
  writeln('Reject the null hypothesis (A=B)');
  writeln(' means are statistically different ...');
end;
writeln('mean A = ',meanA:12:5);
writeln('mean B = ',meanB:12:5);
writeln;
pause:=readkey;
release(p);
end;

(* original *)
procedure t2test;
(* A t-test for independent sets, equal variances not needed *)
(* This works like the procedure above with different variance formula *)
var equal
stanerr,pvalue,alpha,tstat
spooled,diff,vara,varb,meanA,meanB
df
seta,setb
p
: boolean;
: double;
: double;
: integer;
: infoptr;
: pointer;

begin
mark(p);
new(seta);new(setb);
twosets(seta,setb);
IF not EXIST(seta.name) then exit;
if not exist(setb.name) then exit;
meanvar(seta,meanA,vara);
meanvar(setb,meanB,varb);
diff:=meanA-meanB;
stanerr:= sqrt(vara/seta^.size+varb/setb^.size);
if stanerr < 0.0000001 then begin
  writeln('standard error = ',stanerr:12:5);
  writeln;
  writeln('standard error < 0.0000001 ...');
  writeln('leaving test routine ...');
  pause:=readkey;
  exit;
end;
tstat:=ABS(diff/sqrt(vara/seta^.size+varb/setb^.size)); (* use this to integrate t and get p value *)
df:=seta^.size-1;
if df > setb^.size-1 then df:=setb^.size-1;
pvalue:=tvalue(tstat,df);
writeln;
write('Enter the alpha: ');doubleread(alpha);
if pvalue > alpha then begin

```

```

        writeln;
        writeln('No evidence to reject hypothesis ...');
    end
else begin
    writeln;
    writeln('Reject the null hypothesis (A=B)');
    writeln(' means are statistically different ...');
end;
writeln('mean A = ',meana:12:5);
writeln('mean B = ',meanb:12:5);
writeln;
pause:=readkey;
release(p);
end;

(* original *)
procedure t3test;
(* weakest test, independence not needed *)
(* Works basically like the above tests but requires paired data *)
var equal :boolean;
    standev,sum,sumsq,alpha,pvalue,tstat,meana :double;
    x,samples :integer;
    seta,setb :infoPtr;
    p :pointer;
begin
    mark(p);
    new(seta);new(setb);
    twosets(seta,setb);
    IF not EXIST(seta.name) then exit;
    if not exist(setb.name) THEN exit;
    samples:=seta.size;
    if samples > setb.size then samples:=setb.size;
    if samples < 2 then exit;
    sum:=0;
    sumsq:=0;
    for x:=1 to samples do begin
        seta.data[x]:=seta.data[x]-setb.data[x];
        sum:=sum+seta.data[x];
        sumsq:=sumsq+sqr(seta.data[x]);
    end;
    meana:=sum/samples;
    standev:=sqrt((samples*sumsq-sqr(meana))/(samples*(samples-1)));
    if standev< 0.0000001 then begin
        writeln('standev = ',standev:12:5);
        writeln;
        writeln('standev < 0.0000001 ...');
        writeln('leaving the test routine ...');
        pause:=readkey;
        exit;
    end;
    tstat:=ABS(meana/(standev * sqrt(samples)));
    pvalue:=tvalue(tstat,samples-1);
    writeln;
    write('Enter the alpha : ');doubleRead(alpha);
    if pvalue > alpha then begin
        writeln;
        writeln('No evidence to reject hypothesis ...');
    end
else begin
    writeln;
    writeln('Reject the null hypothesis (A=B)');
    writeln(' means are statistically different ...');
end;
writeln('mean = ',meana:12:5);
writeln;
pause:=readkey;
release(p);
end;

```

```

PROCEDURE uRANDOM;
(* generates a set of uniform samples *)
VAR x      : INTEGER;
    seta   : infoPtr;
    p      : pointer;
BEGIN
    mark(p);
    new(seta);
    setwindow(36,10,78,18);
    write('how many?      : ');
    intread(seta^.size);
    if seta^.size>1500 then begin
        writeln('Number set to max of 2500');
        seta^.size:=2500;
    end;
    WRITELN;
    WRITE('Name for data file : ');
    readln(seta^.name);
    setname:=seta^.name;
    writeLn;
    for x:= 1 to seta^.size do
        seta^.data[x]:=(ran3(seed));
    save(seta^);
END;

function uniform(hi,low:double):double;
(* returns one uniform random variable *)
BEGIN
    uniform:=ran3(seed)*(hi-low)+low;
END;

function power(number,exponent:double):double;
(* handles exponentiation for positive numbers with any power *)
label 10;
begin
    if exponent=0 then
        power:=1
    else if number=0 then
        power:=0
    else if number > 0.0 then
        power := exp(exponent*ln(number));
    else power:=number;
10:end;

function wgrandom(alpha,beta:double):double;
(* Wallaces procedure for generating gamma vars *)
(* This is called by the beta generating function *)
VAR temp3,temp4,temp2,temp5 :double;
label 10;
BEGIN
10: temp2:=ran3(seed);
    temp3:=int(alpha);
    temp4:=alpha-temp3;
    count:=trunc(temp3);
    temp5:=1;
    for y:= 1 to count do
        temp5:=temp5*ran3(seed);
    if (temp2 <= 1-alpha+temp3) then temp5:=temp5*ran3(seed);

```

```

temp5:=-1*ln(temp5);
temp2:=ran3(seed);
if (temp2 <= power(temp5/temp3,temp4)/(1-temp4+temp4*temp5/temp3)) then
  wgrandom:=temp5*beta
else goto 10;
END;

```

```

function fgrandom(alpha,beta:double):double;
(* fishman's method of generating gamma rv's *)
(* This is called by the beta generating function *)
var temp,temp2:double;
label 10;
begin;
10: temp:=-1*ln(ran3(seed));
   temp2:=ran3(seed);
   if (temp2 <= power(temp/exp(temp+1),alpha-1)) then
     fgrandom:=alpha*temp*beta
   else goto 10;
END;

```

```

function betavar(alpha,beta:double):double;
(* This generates beta vars by generating a ratio of gamma vars *)
var temp:double;
begin
  if alpha<1 then temp:=fgrandom(alpha,1)
  else temp:=wgrandom(alpha,1);
  if abs(temp)>0.001 then
    if beta<1 then
      betavar:=temp/(temp+fgrandom(beta,1))
    else betavar:=temp/(temp+wgrandom(beta,1))
  ELSE BETAVAR:=0;
END;

```

```

procedure lotsofbetas;
(* generates A SETOF beta deviates *)
VAR x          :INTEGER;
    seta       :infoptr;
    p         :pointer;
    alpha,beta :double;
BEGIN
  mark(p);
  new(seta);
  setwindow(36,10,78,18);
  WRITELN;
  write('enter the alpha : ');
  doubleread(alpha);
  writeln;
  write('enter the beta : ');
  doubleread(beta);
  WRITELN;
  write('how many? : ');
  intread(seta^.size);
  if seta^.size > 2500 then begin
    writeln('Number set to max of 2500');
    seta^.size := 2500;
  end;
  count:=seta^.size;
  WRITELN;
  WRITE('Name for data file : ');
  readln(seta^.name);
  setname:=seta^.name;
  writeln;

```

```

for x:= 1 to trunc(seta^.size) do begin
  seta^.data[x] := betavar(alpha,beta);
end;
save(seta^);
release(p);
end;

```

```

procedure Nrandom;
(* generates A SETOF normal deviates *)
VAR x : INTEGER;
    temp,temp2,mean,variance :double;
    seta :infoptr;
    p :pointer;
BEGIN
  mark(p);
  new(seta);
  set^.indo=(36,10,78,18);
  WRITELN;
  write('enter the mean : ');
  double^read(mean);
  writeLn;
  write('enter the variance: ');
  double^read(variance);
  WRITELN;
  write('how many? : ');
  int^read(seta^.size);
  if seta^.size > 2500 then begin
    writeLn('Number set to max of 2500');
    seta^.size := 2500;
  end;
  count:=seta^.size;
  WRITELN;
  WRITE('Name for data file : ');
  readLn(seta^.name);
  setname:=seta^.name;
  writeLn;

```

```

for x:= 1 to trunc(seta^.size/2) do begin
  writeLn(x);
  temp :=ran3(seed);
  temp2 :=ran3(seed);
  seta^.data[x+(trunc(seta^.size/2))]:= sqrt(-2*ln(temp))*cos(2*pi*temp2);
  seta^.data[x] := sqrt(-2*ln(temp))*sin(2*pi*temp2);
end;
if odd(seta^.size) then begin
  temp:=ran3(seed);
  temp2:=ran3(seed);
  seta^.data[seta^.size]:=sqrt(-2*ln(temp))*sin(2*pi*temp2);
end;

for x:= 1 to seta^.size do begin
  seta^.data[x]:= (seta^.data[x]*sqrt(variance))+mean;
end;
save(seta^);
release(p);
end;

```

```

function normal(mean,variance:double):double;
(* returns one normal deviate *)
var temp,temp2,temp3 :double;
BEGIN
  temp := ran3(seed);
  temp2 := ran3(seed);
  temp3 := sqrt(-2*ln(temp))*cos(2*pi*temp2);

```

```

    normal:= (temp3*sqrt(variance))*mean;
end;

```

```

procedure directory;
(* makes the call to check the disk directory for any file specification *)
var filespec:filename;
begin
    setwindow(36,10,78,18);
    write('Enter Dir mask : ');readln(filespec);
    WINDOW(1,1,80,25);
    showdir(filespec);
end;

```

```

(* original *)
procedure average;
(* this procedure averages each term of any number of data sets into one *)
var seta,setb      :infoPtr;
    done,numfiles,x :integer;
    p               :pointer;
begin
    mark(p);
    new(seta);new(setb);
    setwindow(36,10,78,18);
    numfiles:=1;
    writeLn;
    writeLn('First file:');
    writeLn('*****');
    readln(seta^);
    if not exist(seta^.name) and (seta^.name[1]<>'Q') then BEGIN
        writeLn;
        write('Data file does not exist ...');
        PAUSE:=READKEY;
        exit;
    END;
    repeat
        done:=0;
        writeLn;
        writeLn('Next file:');
        writeLn('*****');
        readln(setb^);
        if exist(setb^.name) then begin
            for x:= 1 to seta^.size do
                seta^.data[x]:=seta^.data[x]+setb^.data[x];
            numfiles:=numfiles+1;
        end
        else if (setb^.name<>'Q') and (setb^.name<>'q') then begin
            writeLn;
            write('Data file does not exist ...');
            PAUSE:=READKEY;
        END
        else if (setb^.name='Q') or (setb^.name='q') then done:=1;
    until done=1;
    for x:=1 to seta^.size do seta^.data[x]:= seta^.data[x]/numfiles;
    clrscr;
    write('Name for new set: ');readln(seta^.name);
    save(seta^);
    release(p);
end;

```

```

procedure enterrate(var sys:systemrec);

```

```

(* This procedure allows the user to enter or change the interest rate *)
(* associated with a LCC system. *)
var tempname : word;
    fil      : file of systemrec;
begin
    setwindow(50,15,78,22);
    WRITEln;
    write('File name? (',sysname,') : ');readln(tempname);
    IF TEMPNAME <> '' THEN
        SysNAME:=TEMPNAME;
        tempname:=SysNAME;
        assign(fil,tempname+'.stm');
        if exist2(tempname+'.stm') then begin
            reset(fil);
            read(fil,sys);
            close(fil);
            writeln;
            writeln(' -- file read -- ');
        end
        else writeln(' -- new file -- ');
            writeln;
            write('interest reate(apr): ');double read(sys.rate);
    end;

procedure entvars(var sys:systemrec);
(* This routine allows the user to specify what types of single variable *)
(* cost components contribute to the LCC of the system. *)
var countvar : integer;
    tempname : word;
    fil      : file of systemrec;
begin
    setwindow(50,15,78,22);
    WRITEln;
    write('File name? (',sysname,') : ');readln(tempname);
    IF TEMPNAME <> '' THEN
        SysNAME:=TEMPNAME;
        tempname:=SysNAME;
        assign(fil,tempname+'.stm');
        if exist2(tempname+'.stm') then begin
            reset(fil);
            read(fil,sys);
            close(fil);
            writeln;
            writeln(' -- file read -- ');
        end
        else writeln(' -- new file -- ');
            writeln;
            write(' How many Variables : ');int read(countvar);
            window(1,1,80,25);clrscr;
            setwindow(3,2,78,22);
            writeln(' Single Variable Components');
            writeln('*****');
            writeln;
            FOR X:= 1 TO countvar DO BEGIN
                WRITE(X:2,' TYPE: ');DOUBLEREAD(sys.VARS[1,X]);
                if SYS.VARS[1,X] = 0 then BEGIN
                    WRITE('constant : ');
                    DOUBLEREAD(SYS.VARS[2,X])
                END
                ELSE IF SYS.VARS[1,X]<11 THEN BEGIN
                    WRITE(' LOW: ');DOUBLEREAD(sys.VARS[2,X]);
                    write(' HI: ');DOUBLEREAD(sys.VARS[3,X]);
                END
                ELSE BEGIN
                    WRITE(' MEAN: ');DOUBLEREAD(sys.VARS[2,X]);
                    write(' VAR: ');DOUBLEREAD(sys.VARS[3,X]);
                END;
                write(' nocost : ');double read(sys.varstime[4,x]);
            end;

```

```

        write(' phase in: ');double read(sys.varstime[1,x]);
        write(' constant: ');double read(sys.varstime[2,x]);
        write(' phase out: ');double read(sys.varstime[3,x]);
    END;
    sys.vars[1,0]:=countvar;
end;

```

```

procedure savesys(sys:systemrec);
(* This procedure saves an LCC system specification to disk *)
VAR OVER      :CHAR;
    tempname   :word;
    TEMP2      :FILENAME;
    sysfile    :file of systemrec;
BEGIN
    setwindow(50,15,78,22);
    writeLn;
    write(' File name? (',sysname,'): ');readln(tempname);
    IF TEMPNAME <> '' THEN
        SysNAME:=TEMPNAME;
        tempname:=SysNAME;
        TEMP2:=CONCAT(tempname+'.stm');
        IF EXIST2(TEMP2) THEN
            BEGIN
                WRITE('File Exists, Over-write? ');
                OVER:=READLN;
                writeLn;
            END
        ELSE
            OVER:=N;
        IF UPCASE(OVER) = 'Y' THEN
            BEGIN
                ASSIGN(sysfile,TEMP2);
                REWRITE(sysfile);
                WRITE(sysfile,Sys);
                CLOSE(sysfile);
            END;
        END;
    END;
END;

```

```

procedure entercer(var sys:systemrec);
(* This reads in the Beta Cov matrix, and finds the cholesky square root. *)
(* This is an original procedure written for cholesky decomposition using *)
(* the method as explained in J.H. Maindonald's STATISTICAL COMPUTATION *)
(* It also reads in all other information needed for the CER's of the LCC *)
(* system to be simulated *)
var
    cer,z,n,x,y,j,k      :integer;
    temp2,temp,sum,sum2,sum3 :double;
    tempname              :word;
    fil                   :file of systemrec;
    intercept,certype     :char;
begin
    setwindow(50,15,78,22);
    WRITELn;

    write('File name? (',sysname,'): ');readln(tempname);
    IF TEMPNAME <> '' THEN
        SysNAME:=TEMPNAME;
        tempname:=SysNAME;
        assign(fil,tempname+'.stm');
        if exist2(tempname+'.stm') then begin
            reset(fil);
            read(fil,sys);
            close(fil);
            writeLn;
            writeLn(' -- file read -- ');
        end;
    end;

```

```

end
else writeln(' -- new file -- ');
writeln;
write('How many CERs : ');intread(x);
if x>20 then begin
    write('X set at upper limit of 20. ');
    writeln;
    x:=20;
end;
sys.cervars[0,1,1]:=x;
writeln;
write('Which CER : ');intread(cer);
if cer>20 then begin
    write('CER# set at upper limit of 20. ');
    writeln;
    cer:=20;
end;
windo=(1,1,80,25);clnscr;
setwindo=(3,2,78,22);
write(' CER Information CER( ,cer, ) ');
write('***** ');
writeln;
WRITELN;
repeat
    write('Number of  $\beta$ 's (max=9): ');intread(n);
    writeln;
until n<10;
write(' Time period lengths ... ');
writeln;
write(' Inocost : ');double read(sys.certime[4,cer]);
write(' Phase in : ');double read(sys.certime[1,cer]);
write(' Constant : ');double read(sys.certime[2,cer]);
write(' Phase out : ');double read(sys.certime[3,cer]);
clnscr;
writeln;
write(' ( R ) Regular ');
write(' ( N ) Natural log ');
write(' ( L ) Learning curve ');
writeln;
repeat
    write('Which type of CER ? ');certype:=readkey;
    writeln(certype);
    sys.certypes[cer]:=upcase(certype);
until (sys.certypes[cer]='R') or (sys.certypes[cer]='N') or (sys.certypes[cer]='L');
writeln;
write('Is there an intercept term? ');intercept:=readkey;
writeln;
if (intercept='y') or (intercept='Y') then begin
    writeln;
    write('The intercept will be called  $\beta',n+1,'.$  ');
    write('Enter the intercept value ( $\beta',n+1,'.$  ');double read(sys.cerbetas[cer,n+1]);
    case certype of
        'R':begin
            sys.cervars[cer,n+1,1]:=0;
            sys.cervars[cer,n+1,2]:=1.0;
            sys.cervars[cer,n+1,3]:=-1.0;
            sys.cervars[cer,n+1,4]:=1.0;
        end;
        'N':begin
            sys.cervars[cer,n+1,1]:=0;
            sys.cervars[cer,n+1,2]:=exp(1.0);
            sys.cervars[cer,n+1,3]:=exp(1.0);
            sys.cervars[cer,n+1,4]:=1.0;
        end;
        'L':begin
            sys.cervars[cer,n+1,1]:=0;
            sys.cervars[cer,n+1,2]:=exp(1.0);
            sys.cervars[cer,n+1,3]:=exp(1.0);
            sys.cervars[cer,n+1,4]:=1.0;
        end;
    end;
end;

```

```

        end;
    end; (* end of case *)
end; (* end of if *)
clrscr;
writeln;
writeln('Explanatory Variables CER(',cer,',')');
writeln('*****');
writeln;
FOR X:= 1 TO n DO BEGIN
    WRITE(X:2,' TYPE: ');DOUBLEREAD(sys.cerVARs[cer,X,1]);
    WRITE(' LOW: ');DOUBLEREAD(sys.cerVARs[cer,X,2]);
    WRITE(' HI : ');DOUBLEREAD(sys.cerVARs[cer,X,3]);
    write('   $\alpha$  : ');doubleread(sys.cervars[cer,x,4]);
END;
window=(1,1,80,25);clrscr;
setwindow=(3,2,78,22);
writeln('Beta estimates CER(',cer,',')');
writeln('*****');
writeln;
For x:= 1 to n do begin
    write('Enter  $\beta$  ',x,' : ');doubleread(sys.cerbetas[cer,x]);
end;
writeln;
if (intercept='y') or (intercept='V') then
    n:=n+1;
sys.cervars[cer,0,1]:=n;
(* initialize cov matrix *)
for x:= 1 to n do
    for y:= 1 to n do
        sys.cercovs[cer,x,y]:=0;
(* read in the cov matrix in triangular form *)
writeln('Now the Covariances ...');
if (intercept='y') or (intercept='V') then
    writeln('Remember, the intercept is called  $\beta$  ',n,'. ');
writeln;
for x:= 1 to n do
    for y:= 1 to x do begin
        write(' cov[',x,',',y,'] = ');doubleread(sys.cercovs[cer,y,x]);
    end;
writeln;
write('Enter the MSE: ');doubleread(sys.cermse[cer]);
(* construct the cholesky square root *)
for x:= 1 to n do begin
    if x>1 then
        for y:= 1 to x-1 do
            for z:= x to n do
                sys.cercovs[cer,x,z]:=sys.cercovs[cer,x,z]-sys.cercovs[cer,y,x]*sys.cercovs[cer,y,z];
            temp:=sqrt(abs(sys.cercovs[cer,x,x]));
            if abs(temp)<=0.001 then temp:=1;
            for y:= 1 to n do
                sys.cercovs[cer,x,y]:=sys.cercovs[cer,x,y]/temp;
        end;
    end;
end;
end;

```

```

procedure multirandom;
var
    counter,z,n,x,y,j,k      :integer;
    temp2,temp,sum,sum2,sum3 :double;
    betas                    :doubvect;
    tempname                  :wordrd;

```

```

(* type as specified by "type" which is passed in. *)
(* NOTE: hi and low contain the mean and var for normal variables. *)
var type2,low2,hi2      : integer;
    tempval             : double;
begin
type2:=trunc(type);
case type2 of
    0:sample:=low;

    1:begin
        tempval:=betavar(1.5,0.5);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    2:begin
        tempval:=betavar(1.35,1.35);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    3:begin
        tempval:=betavar(0.5,1.5);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    4:begin
        tempval:=betavar(3.0,1.0);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    5:begin
        tempval:=betavar(2.75,2.75);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    6:begin
        tempval:=betavar(1.0,3.0);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    7:begin
        tempval:=betavar(4.5,1.5);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    8:begin
        tempval:=betavar(4.0,4.0);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    9:begin
        tempval:=betavar(1.5,4.5);
        tempval:=tempval*(hi-low) + low;
        sample :=power(tempval,transform);
    end;

    10:begin
        tempval:=uniform(low,hi);
        sample :=power(tempval,transform);
    end;
end;

```

```

11:begin
    tempval:=normal(low,hi);
    sample :=power(tempval,transform);
end;

end; (* of case statement *)
end;

```

```

procedure pval(var amount:double;nocostt,phasein,cons,phaseout,rate:double);
(* This is the present value routine for finding the present value *)
(* of a future cash stream. The amount of money, lenght of phasein, *)
(* phaseout, and constant periods are passed in. *)
var areapi,areapo,areaco,totalarea      : double;
    lastcumarea,height,value,nocost      : double;
    x,le,phi,po,co                       : integer;
    portion                               : largevect;
begin
    writeln('amount=',amount);
    writeln('nocostt=',nocostt);
    writeln('phasein=',phasein);
    writeln('cons=',cons);
    writeln('phaseout=',phaseout);
    writeln('rate=',rate);
    pause:=readkey;
    areapi:=0;
    areapo:=0;
    areaco:=0;
    totalarea:=0;
    phi:=trunc(phasein);
    po:=trunc(phaseout);
    co:=trunc(cons);
    nocost:=trunc(nocostt);
    height:=1;
    (* calculate the total area *)
    if phi>0 then
        totalarea:=phi*0.5*height;
    else phi:=0;
    if co>0 then
        totalarea:=totalarea + co*height;
    else co:=0;
    if po>0 then
        totalarea:= totalarea + po*0.5*height;
    else po:=0;
    le:=phi+co+po;
    (* calculate the portions *)
    if phi<>0 then begin
        lastcumarea:=0;
        for x:= 1 to phi do begin
            portion[x]:=((x*height/phi)*x*0.5-lastcumarea)/totalarea;
            lastcumarea:=portion[x]*totalarea;
        end;
    end;
    if co<>0 then
        for x:= phi+1 to phi+co do begin
            portion[x]:=height/totalarea;
        end;
    if po<>0 then begin
        lastcumarea:=0;
        for x:= le downto le-po+1 do begin
            portion[x]:=((le-x+1)*(le-x+1)*1/po*0.5-lastcumarea)/totalarea;
            lastcumarea:=portion[x]*totalarea;
        end;
    end;
end;

```

```

(* calculate the overall pv number *)
value:=0;
for x:= 1 to 1e do
    value:=value + portion[x]*amount/power((1+rate/100),x+NOCOST);
amount:=value;
end;

```

```

procedure montecarlo;
(* This actually runs the show, it calls the routines to simulate the *)
(* system cost as many times as requested. *)
var totalcost,tempcost,tempcost2,firstx : double;
    run,numruns,counter,counter2,x,y,z : integer;
    name : wordrd;
    filename : filename;
    sysfile : file of systemrec; (* the system file store to disk *)
    system : systemrec; (* record to hold cost components *)
    betas : doublevec;
    tempset : ^info;
    tempmat : tensq;
    p : pointer;

begin
    mark(0);
    (*new(system);*)
    new(tempset);
    setwindow(30,10,78,20);
    writeLn;
    write('System file? ');readln(name);
    filename:=concat(name,'.stm');
    if not exist2(filename) then begin
        writeLn('That file is not here ...');
        pause:=readkey;
        release(p);
        exit;
    end;
    writeLn;
    write('How many runs? ');intread(numruns);
    if numruns>1500 then begin
        numruns:=1500;
        writeLn('Runs set at max of 1500.');
```

```

    end;
    assign(sysfile,filename);
    reset(sysfile);
    read(sysfile,system); (* read the disk file into the system record *)
    for run:= 1 to numruns do begin
        write('.');
        totalcost:=0;

        (* add variable costs to the total cost *)
        counter:=trunc(system.vars[1,0]); (* number of variable components *)
        for x:= 1 to counter do begin
            tempcost:=sample(system.vars[1,x],system.vars[2,x],system.vars[3,x],1);
            pval(tempcost,system.varstime[4,x],system.varstime[1,x],system.varstime[2,x],system.varstime[3,x],system.rate);
            totalcost:=totalcost + tempcost;
        end;

        (* now add cers to the total cost *)
        counter:=trunc(system.cervars[0,1]); (* number of CERs *)
        for x:= 1 to counter do begin
            counter2:=trunc(system.cervars[x,0,1]); (* number of input vars for this cer *)
            betas[0]:=counter2; (* a counter used in multinomial *)
        end;
    end;
end;

```

```

    for y:= 1 to counter2 do begin (* puts betas and cov in temp variables for use *)
      betas[y]:=system.cerbetas[x,y];
      for z:= 1 to counter2 do
        tempmat[y,z]:=system.cercovs[x,y,z];
      end;
      multinormal(betas,tempmat);
      tempcost2:=0;
      for V:= 1 to counter2 do begin
        tempcost:=sample(system.cervars[x,y,1],system.cervars[x,y,2],system.cervars[x,y,3],system.cervars[x,y,4]);
        if y=1 then firstx:=tempcost; (* store for use in lc cers *)
        if (system.certypes[x]='L') or (system.certypes[x]='N') then
          tempcost:=Ln(tempcost);
          tempcost2:=tempcost2+tempcost*betas[y]; (* running total within cer *)
        end;
        tempcost2:=tempcost2+sample(11,0,system.cermse[x],1.0); (* account for MSE variance of prediction*)
        if (system.certypes[x]='L') or (system.certypes[x]='N') then
          tempcost2:=exp(tempcost2);
          if (system.certypes[x]='L') then
            tempcost2:=tempcost2 * firstx;
          end;
        pval:=tempcost2,system.certime[4,x],system.certime[1,x],system.certime[2,x],system.certime[3,x],system.rate);
        ;
        totalcost:=totalcost+tempcost2;
      end;

      tempset^.data[run]:=totalcost; (* store this observation of total cost *)
    end;
    writeln;
    writeln('Enter name for output set : ');readln(tempset^.name);
    tempset^.size:=numruns;
    save(tempset^);
    release(p);
  end;
end;

```

```

(* **** *)
(* *)
(* Everything from here down is menus. These are *)
(* the heart of the user interface. What you *)
(* see here are the menu test entries and the *)
(* resulting procedures called for each menu *)
(* choice. The actual scrolling of the cursor, *)
(* first letter selection, and popup windows *)
(* are handled in the DAVEMENU unit. *)
(* *)
(* **** *)

```

```

PROCEDURE enterMENU;

```

```

var
  GO,go2      :char;
  wich,temp   :integer;
  p           :pointer;
  TEMPNAME    :WORRD;

```

```

BEGIN

```

```

  mark(p);
  new(sys);
  sys^.vars[1,0]:=0;
  sys^.cervars[0,1,1]:=0;
  (* **** *)
  wich:=6;
  menu^.left[wich]:=34;

```

```

menu^.max[wich]:=4;
menu^.top[wich]:=10;
menu^.text[1,wich]:='Vars (single) ';
menu^.text[2,wich]:='Define CERs  ';
menu^.text[3,wich]:='Interest rate';
menu^.text[4,wich]:='Save file    ';
menu^.text[0,wich]:='Backup      ';
menu^.text[menu^.max[wich]+1,wich]:=' DEFINE SYSTEM ';
(*****)
go:=menu^.text[1,wich][1];
GO2:=menu^.text[1,wich][1];
temp:=1;
WHILE upcase(GO) <> 'B' DO BEGIN
  windo=(1,1,80,25);
  menucontrol(wich,go,go2,temp);
  GO:=UPCASE(GO);
  (*****)
  CASE GO OF
    'V' : entvars(sys^);
    'D' : entercer(sys^);
    'I' : enterate(sys^);
    'S' : savesys(sys^);
  END; (* END OF CASE AND ELSE*)
  (*****)
end; (* WHILE LOOP END *)
release(p);
END; (* PROCEDURE END *)

PROCEDURE testMENU;
var
  GO,go2:char;
  wich,temp:integer;
  tempstring:string15;

begin
  wich:=5;
  (***** CHANGE THESE PARAMETERS TO MAKE A NEW MENU SCREEN *****)
  menu^.left[wich]:=18;
  menu^.top[wich]:=6;
  menu^.max[wich]:=3;
  menu^.text[1,wich]:='Equal var,Ind';
  menu^.text[2,wich]:='Ind. only  ';
  menu^.text[3,wich]:='Samples paired';
  menu^.text[0,wich]:='Backup      ';
  menu^.text[menu^.max[wich]+1,wich]:=' Tests      ';
  (*****)
  go:=menu^.text[1,wich][1];
  GO2:=menu^.text[1,wich][1];
  temp:=1;
  WHILE upcase(GO) <> 'B' DO BEGIN
    windo=(1,1,80,25);
    menucontrol(wich,go,go2,temp);
    GO:=UPCASE(GO);
    (*****)
    CASE GO OF
      'E':t1test;
      'I':t2test;
      'S':t3test;
      'B':exit;
    end; (* of case *)
    (*****)
  end; (* while *)
end; (* procedure *)

```

```

PROCEDURE workMENU;
var
    GO,go2:char;
    wich,temp:integer;
    tempstring:string15;

begin
    wich:=4;
    (***** CHANGE THESE PARAMETERS TO MAKE A NEW MENU SCREEN *****)
    menu^.left[wich]:=18;
    menu^.top[wich]:=6;
    menu^.max[wich]:=6;
    menu^.text[1,wich]:='Moments      ';
    menu^.text[2,wich]:='Freq Histogram';
    menu^.text[3,wich]:='Time plot   ';
    menu^.text[0,wich]:='Backup      ';
    menu^.text[4,wich]:='X vs Y Plot  ';
    menu^.text[5,wich]:='Quantiles    ';
    menu^.text[6,wich]:='Nonpar. Prob  ';
    menu^.text[menu^.max[wich]+1,wich]:=' Statistics  ';
    (*****
    go:=menu^.text[1,wich][1];
    GO2:=menu^.text[1,wich][1];
    temp:=1;
    WHILE upcase(GO) <> 'B' DO BEGIN
        window(1,1,80,25);
        menucontrol(wich,go,go2,temp);
        GO:=UPCASE(GO);
    (*****
        CASE GO OF
            'M':moments;
            'F':histogram;
            'T':timeplot;
            'X':plotit;
            'Q':quantiles;
            'N':probability;
        END; (* END OF CASE *)
    (*****
        end; (* while *)
    end; (* procedure *)

```

```

PROCEDURE fileMENU;
var
    GO,go2:char;
    wich,temp:integer;
    tempstring:string15;

begin
    wich:=3;
    (***** CHANGE THESE PARAMETERS TO MAKE A NEW MENU SCREEN *****)
    menu^.left[wich]:=18;
    menu^.top[wich]:=6;
    menu^.max[wich]:=4;
    menu^.text[1,wich]:='Read ASCII   ';
    menu^.text[2,wich]:='Write ASCII  ';
    menu^.text[3,wich]:='Set Directory';
    menu^.text[4,wich]:='Disk Directory';
    menu^.text[0,wich]:='Backup      ';
    menu^.text[menu^.max[wich]+1,wich]:=' Files      ';
    (*****
    go:=menu^.text[1,wich][1];

```

```

GO2:=menu^.text[1,wich][1];
temp:=1;
WHILE upcase(GO) <> 'B' DO BEGIN
    window(1,1,80,25);
    menucontrol(wich,go,go2,temp);
    GO:=UPCASE(GO);
    (*****
    CASE GO OF
        'S':SHOWDIR('*.*set');
        'R':READASCII;
        'W':WRITASCII;
        'D':directory;
    END; (* END OF CASE *)
    (*****
end; (* end of while *)
END; (* end of procedure *)

PROCEDURE randommenu;
var
    GO,go2:char;
    wich,temp:integer;
    tempstring:string15;

begin
    wich:=7;
    (***** CHANGE THESE PARAMETERS TO MAKE A NEW MENU SCREEN *****)
    menu^.left[wich]:=18;
    menu^.top[wich]:=6;
    menu^.max[wich]:=5;
    menu^.text[1,wich]:='Uniform Random';
    menu^.text[2,wich]:='Normal random';
    menu^.text[3,wich]:='Generate betas';
    menu^.text[4,wich]:='Multi-normal';
    menu^.text[5,wich]:='Simulate costs';
    menu^.text[0,wich]:='Backup';
    menu^.text[menu^.max[wich]+1,wich]:='Files';
    (*****
    go:=menu^.text[1,wich][1];
    GO2:=menu^.text[1,wich][1];
    temp:=1;
    WHILE upcase(GO) <> 'B' DO BEGIN
        window(1,1,80,25);
        menucontrol(wich,go,go2,temp);
        GO:=UPCASE(GO);
        (*****
        CASE GO OF
            'S':montecarlo;
            'U':uRANDOM;
            'N':nrandom;
            'G':lotsofbetas;
            'M':multirandom;
        END; (* END OF CASE *)
        (*****
end; (* end of while *)
END; (* end of procedure *)

PROCEDURE datamenu;
var
    GO,go2:char;
    wich,temp:integer;

begin
    wich:=2;

```

```

(***** CHANGE THESE PARAMETERS TO MAKE A NEW MENU SCREEN *****)
menu^.left[wich]:=18;
menu^.top[wich] :=6;
menu^.max[wich]:=6;
menu^.text[1,wich]:='Enter      ';
menu^.text[2,wich]:='View      ';
menu^.text[3,wich]:='Modify     ';
menu^.text[4,wich]:='Combine     ';
menu^.text[5,wich]:='Average     ';
menu^.text[6,wich]:='Define system';
menu^.text[0,wich]:='Backup     ';
menu^.text[menu^.max[wich]+1,wich]:='      Data      ';
(*****)
go:=menu^.text[1,wich][1];
GO2:=menu^.text[1,wich][1];
temp:=1;
WHILE upcase(GO) <> 'B' DO BEGIN
    window(1,1,80,25);
    menucontrol(wich,go,go2,temp);
    GO:=UPCASE(GO);
    (*****)
    CASE GO OF
        'E':REEDKEY;
        'V':see;
        'M':change;
        'C':combine;
        'A':average;
        'D':entermenu;
    end; (* end of case *)
    (*****)
end;
end;

PROCEDURE NEWSEED;
begin
    set-window(18,8,43,11);
    writeLn;
    write('New seed: ');intread(seed);
    if seed>0 then seed:=seed* -1;
end;

PROCEDURE mainMENU;
var
    GO,go2:char;
    wich,temp:integer;
begin
    (*****)
    wich:=1;
    menu^.left[wich]:=1;
    menu^.max[wich]:=6;
    menu^.top[wich]:=3;
    menu^.text[1,wich]:='Data      ';
    menu^.text[2,wich]:='Statistics  ';
    menu^.text[3,wich]:='T-testing   ';
    menu^.text[4,wich]:='Files       ';
    menu^.text[5,wich]:='Random deviates';
    menu^.text[6,wich]:='New seed    ';
    menu^.text[0,wich]:='Quit       ';
    menu^.text[menu^.max[wich]+1,wich]:='MAIN MENU  ';
    (*****)
    go:=menu^.text[1,wich][1];
    GO2:=menu^.text[1,wich][1];
    temp:=1;
    WHILE upcase(GO) <> 'Q' DO BEGIN
        window(1,1,80,25);
        menucontrol(wich,go,go2,temp);
        GO:=UPCASE(GO);
        (*****)

```

```

CASE 60 OF
  'D' : DATAMENU;
  'S' : WORKMENU;
  'T' : TESTMENU;
  'F' : filemenu;
  'N' : NEWSSEED;
  'R' : randommenu;
END; (* END OF CASE AND ELSE*)
(*****)
end; (* WHILE LOOP END *)
END; (* PROCEDURE END *)

```

```

procedure open;
type   fy:= file;
var p                                     :pointer;
    x                                     :integer;
    val                                  :real;
    textfil                             :text;
    fil                                  :fy;
    result,size                          :word;

```

```

begin
  graphix;
  setcolor(white);
  line(50,100,210,100);
  line(50,100,50,20);
  setcolor(red);
  line(51,60,210,60);
  for x:=1 to 62 do begin
    val:=sin(x/10)/1.5;
    putpixel(51+x,trunc(60-val*50),yellow);
  end;
  for x:=63 to 160 do begin
    val:=sin(x/10)/4;
    putpixel(51+x,trunc(60-val*50),yellow);
  end;
  setcolor(white);
  line(390,100,550,100);
  line(390,100,390,20);
  bar3d(391,100-40,421,99,8,true);
  bar3d(423,100-60,453,99,8,true);
  bar3d(455,100-50,485,99,8,true);
  bar3d(487,100-35,517,99,8,true);
  bar3d(519,100-15,549,99,8,true);
  settxtstyle(1,0,5);
  outtextxy(60,200,'GENERIC MODEL BUILDER');
  settxtstyle(1,0,1);
  outtextxy(280,240,'by 1lt David Summer');
  setcolor(white);
  pause:=readkey;
  restorecrtmode;
end;

```

```
(* MAIN PROGRAM*)
```

```
BEGIN
```

```

  open;
  seed:=-12345;
  SETNAME:='NEW';
  sysname:='NEW';
  coloroncolor(white,blue);
  write('          GENERIC MODEL -BUILDER    -- by Dave Summer    ');
  gotoxy(1,25);write('');
  coloroncolor(white,black);
  mainMENU;
END.

```

```

{$R-}    {Range checking off}
{$B+}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N+}    {8087 required}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

```

```
PROGRAM MATWORKS;
```

```
Uses
```

```

  Crt,
  Dos,
  PRINTER,
  davedemo;

```

```
TYPE
```

```

FILENAME = STRING[12];
TENBYTEN = ARRAY [1..80,1..80] OF double;
doubleARR = ARRAY [1..80] OF double;
INTARR = ARRAY [1..80] OF INTEGER;
WORRO = STRING[8];
INFOPTR = ^INFO;
INFO = RECORD
    MATRIX : TENBYTEN;
    ROWSIZE : INTEGER;
    COLSIZE : INTEGER;
    NAME : WORRO;
END;

```

```
VAR
```

```

MATA,MATB : INFOptr;
pause : char;
ADDSUB : INTEGER;
FIL : FILE OF INFO;
MATNAME : WORRO;
TINY : double;
X,V,Z : INTEGER;
textfile : text;

```

```

function power(number,exponent:double):double;
label 10;
begin
  if exponent=0 then begin
    power:=1;
    goto 10;end;
  if number=0 then begin
    power:=0;
    goto 10;end;
  if number > 0.0 then
    power := exp(exponent*ln(number))
  else begin
    writeln(number,' number must be positive for power function');
    pause:=readkey;
  end;
10:end;

```

```

PROCEDURE ludcmp(VAR a:INFOptr; n,np: integer;
  VAR indx: INTARR; VAR d: double);

```

```
CONST
```

```
  tiny=1.0e-20;
```

```
VAR
```

```

  k,j,imax,i: integer;
  sum,dum,big: double;
  vv: doubleARR;

```

```

BEGIN
  d := 1.0;
  FOR i := 1 to n DO BEGIN
    big := 0.0;
    FOR j := 1 to n DO IF (abs(a^.MATRIX[i,j]) > big) THEN big := abs(a^.MATRIX[i,j]);
    IF (big = 0.0) THEN BEGIN
      writeln('pause in LUOCMP - singular matrix'); readln
    END;
    vv[i] := 1.0/big
  END;
  FOR j := 1 to n DO BEGIN
    IF (j > 1) THEN BEGIN
      FOR i := 1 to j-1 DO BEGIN
        sum := a^.MATRIX[i,j];
        IF (i > 1) THEN BEGIN
          FOR k := 1 to i-1 DO BEGIN
            sum := sum - a^.MATRIX[i,k]*a^.MATRIX[k,j]
          END;
          a^.MATRIX[i,j] := sum
        END
      END
    END;
    big := 0.0;
    FOR i := j to n DO BEGIN
      sum := a^.MATRIX[i,j];
      IF (j > 1) THEN BEGIN
        FOR k := 1 to j-1 DO BEGIN
          sum := sum - a^.MATRIX[i,k]*a^.MATRIX[k,j]
        END;
        a^.MATRIX[i,j] := sum
      END;
      dum := vv[i]*abs(sum);
      IF (dum > big) THEN BEGIN
        big := dum;
        imax := i
      END
    END;
    IF (j <> imax) THEN BEGIN
      FOR k := 1 to n DO BEGIN
        dum := a^.MATRIX[imax,k];
        a^.MATRIX[imax,k] := a^.MATRIX[j,k];
        a^.MATRIX[j,k] := dum
      END;
      d := -d;
      vv[imax] := vv[j]
    END;
    indx[j] := imax;
    IF (j <> n) THEN BEGIN
      IF (a^.MATRIX[j,j] = 0.0) THEN a^.MATRIX[j,j] := tiny;
      dum := 1.0/a^.MATRIX[j,j];
      FOR i := j+1 to n DO BEGIN
        a^.MATRIX[i,j] := a^.MATRIX[i,j]*dum
      END
    END
  END;
  IF (a^.MATRIX[n,n] = 0.0) THEN a^.MATRIX[n,n] := tiny
END;

```

```

FUNCTION EXIST(temp:WORRD) : BOOLEAN;
VAR FIL:FILE;
    OK :BOOLEAN;
    TEMPP:FILENAME;
GO:CHAR;
BEGIN
  TEMPP:=CONCAT(TEMP, '.MAT');
  ASSIGN(fil,TEMPP);
  {$I-}

```

```

RESET(FIL);
{$!+}
OK:=IORESULT = 0;
IF NOT OK THEN
  EXIST := FALSE
ELSE
  BEGIN
    CLOSE(FIL);
    EXIST:=TRUE;
  END;
END;

```

```

FUNCTION EXIST2(temp:WORRD) : BOOLEAN;
VAR FIL:FILE;
    OK :BOOLEAN;
    TEMPP:FILENAME;
GO:CHAR;
BEGIN
  TEMPP:=TEMP;
  ASSIGN(FIL, TEMPP);
  {$!-}
  RESET(FIL);
  {$!+}
  OK:=IORESULT = 0;
  IF NOT OK THEN
    EXIST2 := FALSE
  ELSE
    BEGIN
      CLOSE(FIL);
      EXIST2:=TRUE;
    END;
END;

```

```

PROCEDURE SAVE(MATG:INFOptr;AUTO:INTEGER);
VAR OVER:CHAR;
    TEMP:FILENAME;
BEGIN
  OVER:='y';
  IF AUTO=0 THEN
    BEGIN
      IF EXIST(MATG^.NAME) THEN
        BEGIN
          writeLn;WRITE('FILE EXISTS, OVERWRITE?: ');
          OVER:=READKEY;writeLn;writeLn;
        END;
      END;
      IF UPCASE(OVER) = 'Y' THEN
        BEGIN
          TEMP:=CONCAT(MATG^.NAME, '.MAT ');
          ASSIGN(FIL, TEMP);
          REWRITE(FIL);
          WRITE(FIL, MATG^);
          CLOSE(FIL);
        END;
      END;
    END;

```

```

PROCEDURE READASCII;
VAR
  X,Y,Z      :INTEGER;
  FIL        :text;
  GO,NAME    :WORRD;
  mat        :infoptr;
  p          :pointer;

```

```

BEGIN
mark(p);
new(mat);
setwindow(35,10,75,18);
writeln; WRITE('What ASCII file?');
READLN(NAME);
writeln;
IF NOT EXIST2(NAME) THEN
  BEGIN
    WRITEln('FILE DOES NOT EXIST ');
    PAUSE:=READKEY;
    release(p);
    EXIT;
  END
ELSE BEGIN
  ASSIGN(FIL,NAME);
  RESET(FIL);
  MAT^.ROWSIZE := 0;
  WHILE NOT EOF(FIL) DO
    BEGIN
      MAT^.ROWSIZE := MAT^.ROWSIZE+1;
      MAT^.COLSIZE:=0;
      WHILE NOT EOLN(FIL) DO
        BEGIN
          MAT^.COLSIZE := MAT^.COLSIZE+1;
          READ(FIL,mat^.MATRIX[mat^.ROWSIZE,mat^.COLSIZE]);
        END;
      READLN(FIL);
    END;
  CLOSE(FIL);
  write'n;WRITE('Name for the MATRIX file?');READLN(mat^.NAME);
  SAVE(MAT,0);

END;
release(p);
END;

```

```

PROCEDURE READIN(VAR MAT:INFOpt);
VAR GO:CHAR;
    TEMP:FILENAME;
BEGIN
  if exist(mat^.NAME)
  then
    begin
      TEMP:=CONCAT(mat^.NAME, '.MAT');
      assign(fil,TEMP);
      RESET(FIL);
      READ(FIL,mat);
      CLOSE(FIL);
    end
  else
    BEGIN
      GOTOXY(17,19);
      write('Sorry, that file is not here ...');
      GO:=READKEY;
    END;
  END;

```

```

FUNCTION LOG(X:double;BASE:double):double;
BEGIN
  IF (X<0)OR(BASE<2) THEN BEGIN
    WRITE('SORRY, NO NEGATIVE X OR BASE < 2');
    LOG:=1
  END
  ELSE LOG:=(LN(X)/LN(BASE));

```

END;

PROCEDURE WRITASCII;

VAR

 tempname :filename;
 max :double;
 DIG,DEC,X,Y,Z :INTEGER;
 FIL :text;
 GO,NAME :WORRO;
 mata :infoptr;
 p :pointer;

BEGIN

mark(p);

new(mata);

setwindow(35,10,75,18);

WRITE('WHICH MATRIX: ');READLN(mata^.NAME);

IF NOT EXIST(mata^.NAME) THEN BEGIN

 writeln;WRITE('FILE DOES NOT EXIST');

 PAUSE:=READKEY;

 release(p);

 EXIT;

END;

READLN(MATA);

write('New data file name? ');readln(tempname);

ASSIGN(FIL,tempname);

REWRITE(FIL);

(*max:=0;

for x:= 1 to mata^.rowSize do

 for y:= 1 to mata^.colSize do

 if abs(mata^.matrix[x,y])>max then max:=mata^.matrix[x,y];

dig:=trunc('log(max,10))+2;

dec:=10-dig;*)

dec:=0;

dig:=3;

FOR X:= 1 TO mata^.ROWSIZE DO BEGIN

 FOR Y:= 1 TO mata^.COLSIZE DO

 WRITE(FIL,mata^.MATRIX[X,Y]:DIG:DEC);

 WRITELN(FIL);

END;

CLOSE(FIL);

release(p);

END;

procedure onemat(var mata:infoptr);

begin

writeln;

write('Which matrix?: ');readln(mata^.name);

writeln;

end;

PROCEDURE VIEW(MAT:INFOptr;AUTO:INTEGER);

VAR X,Y :INTEGER;

 saiv :char;

BEGIN

window(1,1,80,25);

CLRSCL;

FOR X := 1 TO mat^.ROWSIZE DO

 BEGIN

 FOR Y:= 1 TO mat^.COLSIZE DO

 WRITE(mat^.MATRIX[X,Y]:8:4,' ');

 WRITELN;

 END;

PAUSE:=READKEY;

IF AUTO = 0 THEN BEGIN

```

writeln('Do you want to save ',mat^.name,'? (Y/N)');
SAIV:=READKEY;
if upcase(saiv)='Y' then
    save(mat,0);
END;
END;

```

```

PROCEDURE REEDKEY;
(* read a matrix in from the keyboard*)
VAR
    X,Y,FVLE : INTEGER;
    OVER      : CHAR;
    mat       : infoptr;
    p         : pointer;
BEGIN
    mark(p);
    new(mat);
    setwindow(35,10,75,18);
    mat^.name:='null';
    mat^.colsize:=1;
    mat^.rowsize:=1;
    WRITE('Name for matrix?: ');
    READLN(mat^.NAME);
    IF EXIST(mat^.NAME) THEN
        BEGIN
            WRITELN;WRITE('FILE EXISTS, OVERWRITE? (Y/N): ');
            OVER := READKEY;
            writeLn;
        END
    ELSE
        OVER:='Y';
        IF UPCASE(OVER)='Y' THEN
            BEGIN
                WRITELN;WRITE('How many rows?: ');
                READLN(mat^.ROWSIZE);
                WRITELN;WRITE('How many cols?: ');
                READLN(mat^.COLSIZE);
                FOR X := 1 TO mat^.ROWSIZE DO
                    FOR Y := 1 TO mat^.COLSIZE DO
                        BEGIN
                            WRITELN;WRITE('Enter ELEMENT ['X,',',',Y,']: ');
                            READLN(mat^.MATRIX[X,Y]);
                        END;
                    END;
                END;
                SAVE(MAT,0);
                release(p);
            END;
        END;
END;

```

```

PROCEDURE CHANGE;
VAR
    X,Y      : INTEGER;
    SAIV,chanj : CHAR;
    mata     : infoptr;
    p        : pointer;
BEGIN
    mark(p);
    new(mata);
    setwindow(35,10,75,18);
    onemat(mata);
    READLN(MATA);
    IF not EXIST(MATA^.NAME) THEN begin
        release(p);
        exit;
    END;
END;

```

```

CHANJ:='V';
WHILE CHANJ='V' DO
BEGIN
    writeln;
    WRITE('Change something? ');
    chanj:=READKEY;
    IF upcase(chanj)='V' THEN
    BEGIN
        writeln;WRITE('Change which element?(row):');
        READLN(X);
        writeln;WRITE('Change which element?(col):');
        READLN(Y);
        writeln;WRITE('OLD VALUE=',MATA^.MATRIX[X,Y], ' NEW=');
        READLN(MATA^.MATRIX[X,Y]);
        writeln;
        vie=(mata,C)
    END;
    END;
    release(p);
END;

```

```

PROCEDURE ADDMAT (var MATA,MATB:INFOptr;ADDSUB:INTEGER);
VAR X,Y      :INTEGER;
    SAIV,PAUSE : CHAR;

BEGIN
    IF (MATA^.ROWSIZE = MATB^.ROWSIZE) AND (MATA^.COLSIZE = MATB^.COLSIZE) THEN
    BEGIN
        WRITE('Name the result: ');
        READLN(MATB^.NAME);
        FOR X := 1 TO MATA^.ROWSIZE DO
            FOR Y := 1 TO MATA^.COLSIZE DO
                MATB^.MATRIX[X,Y] := MATA^.MATRIX[X,Y] + MATB^.MATRIX[X,Y]*ADDSUB;
            VIEW(MATB,C);
        END
    END
    ELSE
    BEGIN
        WRITE('Matrices are not same size ...');
        PAUSE:=READKEY;
    END;
END;

```

```

PROCEDURE MULT (MAT1,MAT2:INFOptr; VAR result:INFOptr;AUTO:INTEGER);
VAR X,Y,Z      :INTEGER;
    SUM         :double;
    mat3        :info;

BEGIN
    IF (MAT2^.ROWSIZE=MAT1^.COLSIZE) THEN
    BEGIN
        IF AUTO = 0 THEN
        BEGIN
            GOTOXY(30,6); WRITE('MULTIPLY ');
            GOTOXY(15,19); WRITE(' ');
            GOTOXY(15,19); WRITE('What do you want to call the result?:');
            READLN(MAT3.NAME);
        END;
        MAT3.ROWSIZE:=MAT1^.ROWSIZE;
        MAT3.COLSIZE:=MAT2^.COLSIZE;
        FOR X := 1 TO MAT1^.ROWSIZE DO
            FOR Y := 1 TO MAT2^.COLSIZE DO

```

```

    BEGIN
        SUM:=0;
        FOR Z := 1 TO MAT1^.COLSIZE DO
            SUM := SUM + MAT1^.MATRIX[X,Z]*MAT2^.MATRIX[Z,Y];
        MAT3.MATRIX[X,Y]:=SUM;
    END;
    IF AUTO = 0 THEN begin
        result:=mat3;
        VIEW(result,0);
        save(result,0);
    end;
    result:=mat3;
END
ELSE
    BEGIN
        WRITELN;
        WRITELN(' ',MAT1^.NAME,' X ',MAT2^.NAME,' UNDEFINED DUE TO SIZES. ');
        PAUSE:=READKEY;
    END;
END;

```

```

PROCEDURE COPY2 (VAR SOURCE:INFOptr); (* ADD A COL OF 1'S TO LEFT OF MATRIX TO GET INTERCEPT *)
VAR X,Y,tempsize : INTEGER;
BEGIN
    source^.colsize := source^.colsize + 1;
    for x:= source^.colsize downto 2 do
        for y:= 1 to source^.rowsize do
            source^.matrix[y,x]:=source^.matrix[y,x-1];
        for x:= 1 to source^.rowsize do
            source^.matrix[x,1]:=1;
        end;
    END;

```

```

PROCEDURE COPY3(VAR SOURCE:INFOptr); (*build x matrix for poly fitting *)
VAR X,Y,Z : INTEGER;
BEGIN
    repeat
        write('To what order?: ');READLN(Z);
        IF Z > SOURCE^.ROWSIZE THEN
            WRITELN('Too few observ's. Max order = ',SOURCE^.ROWSIZE-2);
        until z<source^.rowsize;
        for x:= 1 to source^.rowsize do
            source^.matrix[x,2]:=source^.matrix[x,1];
        for x:= 1 to source^.rowsize do
            source^.matrix[x,1]:=1;
        for y:= 3 to z do
            for x:= 1 to source^.rowsize do
                source^.matrix[x,y]:=power(source^.matrix[x,2],y-1);
            end;
        end;
    end;

```

```

PROCEDURE TRANSPMAT (var MAT1:INFOptr;auto:integer);
VAR X,Y : INTEGER;
    SAV : CHAR;
    tempval : double;
BEGIN
    IF AUTO = 0 THEN
.. BEGIN

```

```

    write('name the result :');READLN(MAT1^.NAME);
END;
FOR X:= 2 TO MAT1^.ROWSIZE DO
  FOR Y:= 1 TO x-1 DO begin
    tempval:=mat1^.matrix[x,y];
    MAT1^.MATRIX[x,y] := MAT1^.MATRIX[y,x];
    mat1^.matrix[y,x]:=tempval;
  end;
x:=mat1^.rowSize;
mat1^.rowSize:=mat1^.colSize;
mat1^.colSize:=x;
IF AUTO = 0 THEN begin
  VIEW(MAT1,0);
  save(mat1,0);
end;
END;

```

```

PROCEDURE MANIPMAT(VAR MATA:INFOptr);

```

```

VAR X,Y:INTEGER;
    sa,v,CHOICE : CHAR;
    KONSTANT:doub'e;

```

```

BEGIN
  writeln('Name the result?:');READLN(MATA^.NAME);
  writeln('How will you change the matrix?');
  writeln('    1) square the elements');
  writeln('    2) square root of the elements');
  writeln('    3) add a constant to the elements');
  writeln('    4) multiply by a constant');
  writeln('    5) divide by a constant');
  writeln;
  CHOICE:=READKEY;
  CASE CHOICE OF
    '1': FOR X := 1 TO mata^.ROWSIZE DO
          FOR Y := 1 TO mata^.COLSIZE DO
            mata^.MATRIX[X,Y]:= SQR(mata^.MATRIX[X,Y]);
        '2': FOR X := 1 TO mata^.ROWSIZE DO
          FOR Y := 1 TO mata^.COLSIZE DO
            mata^.MATRIX[X,Y] := SQR(mata^.MATRIX[X,Y]);
        '3': BEGIN
            WRITELN;
            write('Enter your constant: ');READLN(konstant);
            FOR X := 1 TO mata^.ROWSIZE DO
              FOR Y := 1 TO mata^.COLSIZE DO
                mata^.MATRIX[X,Y] := mata^.MATRIX[X,Y] + konstant;
            end;
        '4': BEGIN
            WRITELN;
            write('Enter your constant: ');READLN(konstant);
            FOR X := 1 TO mata^.ROWSIZE DO
              FOR Y := 1 TO mata^.COLSIZE DO
                mata^.MATRIX[X,Y] := mata^.MATRIX[X,Y] * konstant;
            end;
        '5': BEGIN
            WRITELN;
            write('Enter your constant: ');READLN(konstant);
            FOR X := 1 TO mata^.ROWSIZE DO
              FOR Y := 1 TO mata^.COLSIZE DO
                mata^.MATRIX[X,Y] := mata^.MATRIX[X,Y] / konstant;
            end;
  END;
  view(mata,0);
END;

```

```

PROCEDURE TWOMATS(VAR MATA,MATB:INFOptr);
BEGIN
  writeln;
  WRITE('FIRST MAT:');
  READLN(MATA^.NAME);
  READLN(MATA);
  if not exist(mata^.name) then exit;
  writeln;
  WRITE('SECOND:');
  READLN(MATB^.NAME);
  READLN(MATB);
END;

```

```

procedure logit(var mata:infoptr);
var x,y :integer;
begin
  for x:= 1 to mata^.rowSize do
    for y:= 1 to mata^.colSize do
      mata^.matrix[x,y]:=ln(mata^.matrix[x,y]);
    end;
  end;

```

```

PROCEDURE lubksb(a: infoptr; n,np: integer; indx: intarr; VAR b:doublearr);
(* Programs Using LUBKSB must define the types
TYPE
  g'narray = ARRAY [1..n] OF real;
  g'indx = ARRAY [1..n] OF integer;
  g'npbnp = ARRAY [1..np,1..np] OF real;
in the main routine *)
VAR
  j,ip,ii,i: integer;
  sum: real;
BEGIN
  ii := 0;
  FOR i := 1 to n DO BEGIN
    ip := indx[i];
    sum := b[ip];
    b[ip] := b[i];
    IF (ii <> 0) THEN BEGIN
      FOR j := ii to i-1 DO BEGIN
        sum := sum-a^.matrix[i,j]*b[j]
      END
    END ELSE IF (sum <> 0.0) THEN BEGIN
      ii := i
    END;
    b[i] := sum
  END;
  FOR i := n DOWNT0 1 DO BEGIN
    sum := b[i];
    IF (i < n) THEN BEGIN
      FOR j := i+1 to n DO BEGIN
        sum := sum-a^.matrix[i,j]*b[j]
      END
    END;
    b[i] := sum/a^.matrix[i,i]
  END
END;

```

```

procedure inv(mata:infoptr);
(* takes a matrix inverse, see Press, et al,p.38 *)
var colvect      : doublearr;
    indx         : intarr;

```

```

D           : double;
N,NP,x,y,z,j : integer;
tempmat     : info;
begin
N:= mata^.rowsize;
NP:=30;
for x := 1 to N do begin
  for y:= 1 to N do
    tempmat.matrix[x,y]:=0;
    tempmat.matrix[x,x]:=1;
  end;
ludcmp(mata,N,NP,indx,D);
for j:= 1 to N do begin
  for y:= 1 to np do colvect[y]:=tempmat.matrix[y,j];
  lubksb(mata,N,NP,indx,colvect);
  for y:= 1 to np do tempmat.matrix[y,j]:=colvect[y];
end;
for x:= 1 to N do
  for y:= 1 to N do
    mata^.matrix[x,y]:=tempmat.matrix[x,y];
end;
end;

```

```

PROCEDURE DETERMAT(MATG:INFOptr;auto:integer);
VAR X      :INTEGER;
D          :double;
INDX       :INTARR;
good       :boolean;
math       :infoptr;
p          :pointer;
BEGIN
  mark(p);
  new(math);
  LUDCMP(MATG,MATG^.ROWSIZE,MATG^.ROWSIZE,INDX,D);
  FOR X := 1 TO MATG^.ROWSIZE DO
    D:=D*MATG^.MATRIX[X,X];
  if auto=1 then begin
    write'n;
    writeln('Determinant = ',d:11:4);
    PAUSE:=READKEY;
  end;
  release(p);
END;

```

```

procedure direct;
var tempspec:filename;
begin
set=indow(35,10,75,18);
write('Enter the Filespec : ');readln(tempspec);
windo=(1,1,80,25);
showdi=(tempspec);
end;

```

```

PROCEDURE rsm(linsys:integer);
VAR SSR,SST,nybarsq,vary,meany,sumy,SUMYSQ,RSQ,d,sse,ssy :double;
logs          :char;
good          :boolean;
n,k,x,v       :INTEGER;
MATC,MATO     :INFOPTR;
yresid,pred   :doublearr;
xname,yname   :wordr;

```

```

        menu                :char;
        p                    :pointer;
BEGIN
    mark(p);
    setwindow(19,7,50,16);
    NEW(MATC);NEW(MATD);
    (* read in and work on the x matrix *)
    writeln('X matrix ...');
    ONEMAT(MATA);
    READIN(MATA);
    IF NOT EXIST(MATA^.NAME) THEN begin
        release(p);
        EXIT;
    end;
    if (mata^.rowsize<=mata^.colsize) then begin
        writeln('Too few observations ...');
        pause:=readkey;
        release(p);
        exit;
    end;
    n:=mata^.rowsize;
    write('Natural logarithms of X? : ');
    LOGS:=READKEY;
    writeln;
    if (logs='y') or (logs='Y') then
        logit(mata);
    COPY2(MATA);                (* add constant column of ones *)
    k:=mata^.colsize;
    view(mata,0);
    matd:=mata;
    matd:=matd;
    TRANSPOMAT(MATB,1);          (* matb returns xt *)
    MULT(MATB,MATA,MATA,1);      (* mata returns xtx *)
    INV(MATA);                   (* mata returns xtx inv *)
    (* now read in and work on the y matrix *)
    writeln;
    writeln('Y matrix ...');
    ONEMAT(MATC);
    READIN(MATC);
    IF NOT EXIST(MATC^.NAME) THEN begin
        release(p);
        EXIT;
    end;
    yname:=matc^.name;
    write('Natural logarithms of Y? : ');LOGS:=READKEY;
    writeln;
    if (logs='y') or (logs='Y') then logit(matc);
    for x:= 1 to matc^.rowsize do
        yresid[x]:=matc^.matrix[x,1];
    MULT(MATB,MATC,MATB,1);      (* matd returns xty *)
    MULT(MATA,MATB,MATC,1);      (* matc returns estimated  $\beta$ 's *)
    view(matc,0);
    (* find mean and sum of sq of y *)
    sumy:=0;
    sumysq:=0;
    for x:= 1 to n do begin
        sumy:= sumy+yresid[x];
        sumysq:=sumysq+SQR(yresid[x]);
    end;
    meany:= (sumy/n);

    (* calculate the yhat vector *)
    (* note: here resid holds the x matrix *)
    mult(matd,matc,matd,1);      (* matd returns yhat *)
    view(matd,0);
    for x:= 1 to n do
        pred[x]:=matd^.matrix[x,1];
    (* calculate the residual vector *)
    for x:= 1 to matd^.rowsize do

```

```

    yresid[x]:=yresid[x]-pred[x];
    (* calculate n*sqr(y mean) *)
    nybarsq:=matd^.rowsize*sqr(meany);
    (* calculate the sum of residuals squared *)
    sse:=0;
    for x:= 1 to n do
        sse:=sse + sqr(yresid[x]);
    transpmat(matc,1); (* transpose the beta hat vector *)
    mult(matc,matb,matb,1); (* matb returns b'x'y *)
    SSR:=matb^.matrix[1,1]-nybarsq;
    SST:=sumysq-nybarsq;
    writeln('beta covariance matrix ...');
    for x:= 1 to mata^.rowsize do
        for y:= 1 to mata^.colsize do
            mata^.matrix[x,y]:=mata^.matrix[x,y]*(sse/(n-k));
        end;
    end;
    window(1,1,80,25);clrscr;
    writeln(' Estimator      Estimate      Std Error      T value');
    writeln('-----');
    writeln;
    for x:= 1 to matc^.colsize do begin
        writeln('      beta',x-1,'      ',matc^.matrix[1,x]:11:5,'      ',sqrt(mata^.matrix[x,x]):11:5,
            '      ',matc^.matrix[1,x]/sqrt(mata^.matrix[x,x]):11:5);
    end;
    writeln;
    writeln('N      = ',n);
    writeln('R^2      = ',SSR/SST:8:5);
    writeln('MSE      = ',sse/(n-k));
    writeln;
    writeln('      P) Predictions and residuals');
    writeln('      B) Beta covariance matrix ');
    writeln('      M) Menu ');
    menu:=readkey;
    if upcase(menu)='P' then begin
        clrscr;
        writeln('Predictions      Residuals');
        writeln('-----');
        for x:= 1 to n do
            writeln(pred[x]:11:5,'      ',yresid[x]:11:5);
        end;
        PAUSE:=READKEY;
    end;
    else if upcase(menu)='B' then begin
        for x:= 1 to mata^.rowsize do begin
            writeln;
            for y:= 1 to mata^.colsize do
                write(mata^.matrix[x,y]:11:5);
            end;
            writeln;
        end;
        pause:=readkey;
    end;
    release(p);
end;

```

```

PROCEDURE xtransform(linsys:integer);
VAR SSR,SST,nybarsq,vary,meany :double;
    sumy,SUMYSQ,RSQ,d,sse,ssy,old :double;
    logs :char;
    good :boolean;
    pass,n,k,numx,X,Y :INTEGER;
    MATC,MATD :INFOPTR;
    yresid,pred,yvect :doublearr;
    xname,yname :word;
    menu :char;
    p :pointer;

```

```

        alphasig                :boolean;
        alpha                    :doublearr;
BEGIN
    mark(p);
    pass:=1;
    setwindow(19,7,50,16);
    NEW(MATC);NEW(MATD);
    (* read in and work on the x matrix *)
    writeln('X matrix ...');
    ONEMAT(MATA);
    xname:=mataf.name;
    READIN(MATA);
    IF NOT EXIST(MATAf.NAME) THEN begin
        release(p);
        EXIT;
    end;
    if (mataf.rowsize<=mataf.colsize) then begin
        writeln('Too few observations ...');
        pause:=readkey;
        release(p);
        exit;
    end;
    n:=mataf.rowsize;
    numx:=mataf.colsize;
    copy2(mata);
    for x:= 1 to n do
        alpha[x]:=1;
        writeln;
        writeln('Y matrix ...');
        ONEMAT(MATC);
        READIN(MATC);
        IF NOT EXIST(MATCf.NAME) THEN begin
            release(p);
            EXIT;
        end;
        yname:=matcf.name;
        FOR X:= 1 TO MATCf.ROWSIZE DO
            YVECT[X]:=MATCf.MATRIX[X,1];
        write('Natural logarithms of Y? : ');LOGS:=READKEY;
        writeln;
        if (logs='y') or (logs='Y') then logit(matc);

    repeat

        matdf:=mataf;
        matbf:=mataf;
        TRANSPMAT(MATB,1);          (* matb returns xt *)
        MULT(MATB,MATA,MATa,1);      (* mataf returns xtx *)
        INV(MATA);
        (* now work on the y matrix *)
        MULT(MATB,MATC,MATB,1);      (* matbf returns xty *)
        MULT(MATA,MATB,MATC,1);      (* matc returns estimated beta's *)
        writeln('Pass = ',pass);
        (* find mean and sum of sq of y *)
        sumy:=0;
        sumysq:=0;
        for x:= 1 to matcf.rowsize do begin
            sumy:= sumy+matcf.matrix[x,1];
            sumysq:=sumysq+sqr(matcf.matrix[x,1]);
        end;
        many:= (sumy/x);
        (* calculate the yhat vector *)
        (* note: here MATD holds the x matrix *)
        mult(MATD,MATC,MATD,1);
        (* calculate the residual vector *)
        for x:= 1 to matDf.rowsize do
            VreSid[x]:=YVECT[X]-matDf.matrix[x,1];
        (* calculate n*sqr(y mean) *)
        nybarsq:=n*sqr(many);

```

```

(* calculate the sum of residuals squared *)
sse:=0;
for x:= 1 to MATD^.rowsize do
  sse:=sse + sqr(VRESID[x]);
transpmat(matC,1); (* transpose the beta hat vector *)
mult(matC,matB,matB,1);

SSR:=matB^.matrix[1,1]-nybarsq;
SST:=sumysq-nybarsq;
for x:= 1 to mata^.rowsize do
  for y:= 1 to mata^.colsize do
    mata^.matrix[x,y]:=mata^.matrix[x,y]*(sse/(n-numx-1));

(* now create the Z values *)
MATA^.NAME:=XNAME;
READIN(MATA);
for x:= 1 to n do
  for y:= 1 to numx do
    mata^.matrix[x,y]:=power(mata^.matrix[x,y],alpha[y]);
for x:= 1 to mata^.rowsize do
  for y:= 1 to mata^.colsize do
    mata^.matrix[x,y+numx]:=matC^.matrix[1,y+1]*mata^.matrix[x,y]*LN(mata^.matrix[x,y]);
(* now regress again, this time on X|Z *)
mata^.colsize:=numx * 2;
copy2(mata);
MATB^:=MATA^;
MATC^:=MATA^;
k:=mata^.colsize;
TRANSPOMAT(MATB,1); (* matb returns xt *)
MULT(MATB,MATA,mata,1); (* matA returns xtx *)
INV(mata); (* matA returns xix inv *)
(* now work on the y matrix *)
matC^.name:=yname;
readin(matC);
MULT(MATB,MATC,MATB,1); (* matB^ returns xty *)
MULT(MATA,MATB,MATC,1); (* matC returns estimated beta's *)
(* note: on next line matd will hold the x matrix *)
mult(MATD,matC,MATD,1); (* calculate the yhat vector, put it in matd *)
FOR X:= 1 TO n DO
  PRED[X]:=MATD^.MATRIX[X,1];
(* calculate the residual vector *)
for x:= 1 to n do
  VRESID[x]:=VVECT[X]-pred[x];
(* calculate the sum of residuals squared *)
sse:=0;
for x:= 1 to n do
  sse:=sse + sqr(VRESID[x]);
transpmat(matC,1); (* transpose the beta hat vector *)
mult(matC,matB,matB,1);
SSR:=matB^.matrix[1,1]-nybarsq;
SST:=sumysq-nybarsq;
for x:= 1 to mata^.rowsize do (* modify xtxinv to be var-cov matrix *)
  for y:= 1 to mata^.colsize do
    mata^.matrix[x,y]:=mata^.matrix[x,y]*(sse/(n-(numx*2+1)));
alphasig:=false;
(* test to see if any alphas are significant *)
for x:= 1 to numx do
  if abs(matC^.matrix[1,x+numx+1]) > abs(2*sqr(mata^.matrix[x+numx+1,x+numx+1])) then begin
    alpha[x]:=alpha[x]*(matC^.matrix[1,x+numx+1]+1);
    alphasig:=true;
  end;
mata^.name:=xname; (* recall the name of the original x matrix *)
if alphasig then begin (* transformation for subsequent passes if any significant *)
  readin(mata);
  for x:= 1 to mata^.rowsize do
    for y:=1 to mata^.colsize do

```

```

        mataf.matrix[x,y]:=power(mataf.matrix[x,y],alpha[y]);
    copy2(mata);
    matcf.rowsize:=n;
    matcf.colsize:=1;
    for xi:= 1 to n do
        matcf.matrix[x,1]:=yvect[x];
    end;
    (* show betas and cov matrix if nothing significant on first pass *)
    if not(alphasig) and (pass=1) then begin
        writeln('pass = ',pass);
        for xi:=1 to numx*2+1 do
            writeln(matc.matrix[1,x]:11:5, ' ',sqrt(mataf.matrix[x,x]):11:5);
        pause:=readkey;
    end;
    pass:=pass+1;

until not(alphasig);

window(1,1,80,25);clrscr;
writeln(' Estimator      Estimate ');
writeln(' _____ ');
writeln;
for xi:= 1 to numx do
    writeln('      a[x, '      ',alpha[x]:11:5);
writeln;
writeln(' Perform the transformations? ');pause:=readkey;
writeln;
if (pause='y') or (pause='Y') then begin
    mataf.name:=xname;      (* recall the name of the original x matrix *)
    readln(mata);
    writeln(' Name for transformed matrix : ');readln(mataf.name);
    for xi:= 1 to mataf.rowsize do
        for y:=1 to mataf.colsize do
            mataf.matrix[x,y]:=power(mataf.matrix[x,y],alpha[y]);
        save(mata,1);
    end;
    release(b);
end;
end;

```

```

PROCEDURE datamenu;
var
    GO,GO2:char;
    wich,temp:integer;
begin
    (*****);
    wich:=3;
    menuf.left[wich]:=18;
    menuf.max[wich]:=7;
    menuf.top[wich]:=6;
    menuf.text[1,wich]:='Enter      ';
    menuf.text[2,wich]:='View      ';
    menuf.text[3,wich]:='Import ASCII  ';
    menuf.text[4,wich]:='Write ASCII  ';
    menuf.text[5,wich]:='Disk Dir   ';
    menuf.text[6,wich]:='List MAT files';
    menuf.text[7,wich]:=' Modify      ';
    menuf.text[0,wich]:='Backup      ';
    menuf.text[menuf.max[wich]+1,wich]:=' DATA MENU  ';
    (*****);
    go:=menuf.text[1,wich][1];
    GO2:=menuf.text[1,wich][1];
    temp:=1;

```

```

WHILE upcase(GO) <> 'B' DO BEGIN
  window(1,1,80,25);
  menucontrol(wich,go,go2,temp);

  GO:=UPCASE(GO);
  (*****)
  CASE GO OF
    'E':REEDKEY;
    'V':BEGIN
      set=indo=(35,10,75,18);
      onemat(mata);
      READLN(MATA);
      IF EXIST(MATA^.NAME) THEN
        VIEW(MATA,0);
      END;
    'I':READASCII;
    'W':WRITASCII;
    'L':SHOWDIR('*.MAT');
    'D':direct;
    'M':BEGIN
      set=indo=(35,10,75,18);
      onemat(mata);
      READLN(MATA);
      IF EXIST(MATA^.NAME) THEN
        CHANGE;
      END;
    END;
  end;
end;
END;

```

```

PROCEDURE workmenu;
var
  GO,go2:char;
  wich,temp:integer;
  sing      :boolean;
  inda      : INTARR;
  d         : double;
  good      :boolean;
BEGIN
  (*****)
  wich:=2;
  menu^.left[wich]:=18;
  menu^.max[wich]:=7;
  menu^.top[wich]:=6;
  menu^.text[1,wich]:= 'Add      ';
  menu^.text[2,wich]:= 'Subtract  ';
  menu^.text[3,wich]:= 'Multiply   ';
  menu^.text[4,wich]:= 'Invert     ';
  menu^.text[5,wich]:= 'Transpose  ';
  menu^.text[6,wich]:= 'Change Matrix';
  menu^.text[7,wich]:= 'Determinant';
  menu^.text[0,wich]:= 'Backup     ';
  menu^.text[menu^.max[wich]+1,wich]:= 'MATRIX MENU ';
  (*****)
  go:=menu^.text[1,wich][1];
  GO2:=menu^.text[1,wich][1];
  temp:=1;
  WHILE upcase(GO) <> 'B' DO BEGIN
    window(1,1,80,25);
    menucontrol(wich,go,go2,temp);

    GO:=UPCASE(GO);
    (*****)
    CASE GO OF

```

```

'A':BEGIN
  SETWINDOW(35,9,79,18);
  TWOMATS(MATA,MATB);
  IF EXIST(MATA^.NAME) AND EXIST(MATB^.NAME) THEN
    ADDMAT(MATA,MATB,1);
  END;
'S': BEGIN
  SETWINDOW(35,9,79,18);
  TWOMATS(MATA,MATB);
  IF EXIST(MATA^.NAME) AND EXIST(MATB^.NAME) THEN
    ADDMAT(MATA,MATB,-1);
  END;
'M': BEGIN
  SETWINDOW(35,9,73,18);
  TWOMATS(MATA,MATB);
  IF EXIST(MATA^.NAME) AND EXIST(MATB^.NAME) THEN
    MULT(MATA,MATB,MATD,0);
  END;
'I':BEGIN
  SETWINDOW(35,9,79,18);
  onemat(mata);
  READIN(MATA);
  IF EXIST(MATA^.NAME) THEN begin
    INV(MATA);
    WRITE('NAME THE INVERSE: ');READLN(MATA^.NAME);
    VIEW(MATA,0);
  end;
  END;
'T':BEGIN
  SETWINDOW(35,9,79,18);
  onemat(mata);
  READIN(MATA);
  IF EXIST(MATA^.NAME) THEN
    TRANSPMAT(MATA,0);
  END;
'C':BEGIN
  SETWINDOW(35,9,79,18);
  onemat(mata);
  READIN(MATA);
  IF EXIST(MATA^.NAME) THEN
    MANIPMAT(MATA);
  end;
'D':BEGIN
  SETWINDOW(35,9,79,18);
  onemat(mata);
  READIN(MATA);
  IF EXIST(MATA^.NAME) THEN
    DETERMAT(MATA,1);
  end;
  END; (* END OF CASE AND ELSE*)
  (*****);
end; (* WHILE LOOP END *)
END; (* PROCEDURE END *)

```

```

PROCEDURE mainMENU;
var
  GO,go2:char;
  wich,temp:integer;
begin
  (*****);
  wich:=1;
  menu^.left[wich]:=1;
  menu^.max[wich]:=4;
  menu^.top[wich]:=3;
  menu^.text[1,wich]:='Data';

```

```

menu^.text[2,wich]:='Matrix ops';
menu^.text[3,wich]:='Regression';
menu^.text[4,wich]:='X transform';
menu^.text[0,wich]:='Quit';
menu^.text[menu^.max[wich]-1,wich]:='MAIN MENU';
(*****);
go:=menu^.text[1,wich][1];
GO2:=menu^.text[1,wich][1];
temp:=1;
WHILE upcase(GO) <> 'Q' DO BEGIN
  window(1,1,80,25);
  menucontrol(wich,go,go2,temp);

  GO:=UPCASE(GO);
  (*****);
  CASE GO OF
    'D' : DATAMENU;
    'M' : WORKMENU;
    'R' : rsm(0);
    'X' : xtransform(0);
  END; (* END OF CASE AND ELSE*)
  (*****);
end; (* WHILE LOOP END *)
END; (* PROCEDURE END *)

```

```

(* MAIN PROGRAM *)
BEGIN
  ne:=mata);new(matb);
  TINY:=0.000001;
  clrscr;
  textbackground(blue);
  write(
gotoxy(1,25);write('
TEXTBACKGROUND(BLACK);
MAINMENU;
END.

```

MATRIX -- by Dave Summer

);

);

```
unit davestat;
```

```
(*****
  This unit contains many of the statistical procedures called in the
  probstat program. Some of them are original, and others are taken from
  Numerical Recipes or Turbo Pascal Programmer's Library.
  *****)
```

```
INTERFACE
uses crt, Gavemenu;
```

```
TYPE
```

```
  vect      = array[0..20] of integer;
  largevect = array[1..100] of double;
  doubvect  = array[0..10] of double;
  twenvect  = array[0..20] of double;
  charvect  = array[1..20] of char;
  threebytew = array[1..3,0..20] of double;
  fourbytew = array[1..4,0..20] of double;
  tensq     = array[1..10,1..10] of double;
  twensq    = array[1..20,1..20] of double;
  twencube  = array[1..20,1..10,1..10] of double;
  fourtwensq = array[0..20,0..20,1..4] of double;
```

```
  systemrec = record (* Holds the system cost definitions *)
    vars      : threebytew; (* Single var type, hi, low *)
    varstime  : fourbytew; (* Single var nocost, PI, CON, PC lengths *)
    cervars   : fourtwensq; (* Cer explanatory var type, hi, low, alpha *)
    certime   : fourbytew; (* Cer nocost, PI, CON, PC lengths *)
    cerbetas  : twensq; (* Cer beta estimates *)
    certypes  : charvect; (* Cer type *)
    cermse    : twenvect; (* Cer MSE *)
    cercovs   : twencube; (* Cer cov matrices (cholesky sqrt) *)
    rate      : double; (* Associated interest rate for PV *)
  end;
```

```
  sysptr = ^systemrec;
  AXISANNSIZE = STRING[80];
  ANNARRAYTYPE = ARRAY [0..21] OF AXISANNSIZE;
  FILENAME = STRING[12];
  WORRD = STRING[8];
  infomatptr = ^info;
  INFOmat = RECORD
    MATRIX : tensq;
    ROWSIZE : INTEGER;
    COLSIZE : INTEGER;
    NAME : WORRD;
  END;
```

```
  DATARAY = ARRAY [1..2500] OF double;
  INFO = RECORD
    DATA : DATARAY;
    SIZE : INTEGER;
    NAME : WORRD;
  END;
  infoptr = ^info;
```

```
CONST
```

```
  XORI=1;
  VORI=12;
```

```
var
```

```
  FIL : FILE OF INFO;
  sysname, setname : WORRD;
```

```

MAT1,MAT2,mat3           :INFOmat;
pause                   :char;
sys                     :sysptr;
seed                    :integer;

x,y,z,count,glix1,glix2,glix3,seed1,seed2: integer;
glr                     : ARRAY [1..97] OF double;
glinext,glinextp        : integer;
gima                    : ARRAY [1..55] OF double;

FUNCTION ran3(VAR idum: integer): double; (* from numerical recipes *)
PROCEDURE MEANVAR(SETG:INFO;VAR MEAN,VARIANCE:double); (* original *)
PROCEDURE JACK(SETG:INFO;VAR ZMEAN,ZVAR:double); (* original *)
FUNCTION gammin(xx: double): double;
FUNCTION beta(a,b,x: double): double;
function tvalue(tstat:double;df:integer):double;
function fvalue(fstat:double;df1,df2:integer):double;
FUNCTION EXIST(temp:WORRD) : BOOLEAN;

FUNCTION EXIST2(temp:filename) : BOOLEAN;
PROCEDURE SAVE(SETG:INFO);
PROCEDURE READIN(VAR SETG:INFO);
PROCEDURE TWOSETS(VAR SETA,SETB:INFO);
PROCEDURE MULTmat;
PROCEDURE READASCII;
PROCEDURE WRITASCII;
PROCEDURE REEDKEY;
PROCEDURE CHANGE;
PROCEDURE SEE;
PROCEDURE SORTPR (VAR SETG:INFO);
PROCEDURE QUANTILES;
PROCEDURE PROBABILITY;

IMPLEMENTATION

FUNCTION ran3(VAR idum: integer): double; (* from numerical recipes *)

(*****
This procedure was taken from Numerical recipes. It generates uniform(0,1)
variables.
*****)

(* Programs using RAN3 must declare the following variables
in the main routine. Machines with 4-byte integers can use the integer
implementation of this routine, substituting gima of type integer, the
commented CONST and VAR declarations, and the MOD function in the third
line after the BEGIN. *)
(* CONST
  mbig=1000000000;
  mseed=161803398;
  mz=0;
  fac=1.0e-9;
VAR
  i,ii,k,mj,mk: integer; *)
CONST
  mbig=4.0e6;
  mseed=1618033.0;
  mz=0.0;
  fac=2.5e-7; (* 1/mbig *)
VAR
  i,ii,k: integer;
  mj,mk: double;
BEGIN
  IF (idum < 0) THEN BEGIN
    mj := mseed+idum;

```

```

(* The following IF block is  $m_j := m_j \bmod m_{big}$ ; for real variables. *)
IF  $m_j > 0.0$  THEN  $m_j := m_j - m_{big} * \text{trunc}(m_j / m_{big})$ ;
ELSE  $m_j := m_{big} - \text{abs}(m_j) + m_{big} * \text{trunc}(\text{abs}(m_j) / m_{big})$ ;
g[ma][55] :=  $m_j$ ;
mk := 1;
FOR i := 1 TO 54 DO BEGIN
  ii :=  $21 * i \bmod 55$ ;
  g[ma][ii] := mk;
  mk :=  $m_j - mk$ ;
  IF (mk < mz) THEN mk :=  $mk + m_{big}$ ;
  mj := g[ma][ii];
END;
FOR k := 1 TO 4 DO BEGIN
  FOR i := 1 TO 55 DO BEGIN
    g[ma][i] :=  $g[ma][i] - g[ma][1 + ((i-30) \bmod 55)]$ ;
    IF ( $g[ma][i] < mz$ ) THEN  $g[ma][i] := g[ma][i] + m_{big}$ ;
  END;
END;
g[linext] := 0;
g[linextp] := 31;
idum := 1;
END;
g[linext] := g[linext] + 1;
IF (g[linext] = 56) THEN g[linext] := 1;
g[linextp] := g[linextp] + 1;
IF (g[linextp] = 56) THEN g[linextp] := 1;
mj := g[ma][g[linext]] - g[ma][g[linextp]];
IF (mj < mz) THEN mj :=  $m_j + m_{big}$ ;
g[ma][g[linext]] := mj;
ran3 := mj * fac;
END;

```

```

PROCEDURE MEANVAR(SETG:INFO;VAR MEAN,VARIANCE:double); (* original *)

```

```

(*****
This procedure is all original. It calculates the mean and variance of
a data set using n-1 weighting.
*****)

```

```

VAR SUM,SUMSQ:double;
X:INTEGER;
BEGIN
  IF SETG.SIZE <= 2 THEN EXIT;
  SUMSQ := 0;
  SUM := 0;
  FOR X := 1 TO SETG.SIZE DO
    BEGIN
      SUMSQ := SUMSQ + SQR(SETG.DATA[X]);
      SUM := SUM + SETG.DATA[X];
    END;
  MEAN := SUM/X;
  VARIANCE := (SUMSQ - X*SQR(mean))/(X-1);
END;

```

```

PROCEDURE JACK(SETG:INFO;VAR ZMEAN,ZVAR:double); (* original *)

```

```

(*****
This procedure calculates the "jackknife" estimate of the sample variance.
The standard error of this estimate allows a confidence interval (symmetric)
to be drawn on the sample variance.
*****)

```

```

VAR PSUEDOJ
  SUMZ,SUMZSQ,SUM,SUMSQ,MEAN,MEANJ,VARIANCE :double;
  X,Y,M :INTEGER;

```

```
(*****
In calculating the pseudovalues and z values, it is necessary to sum
the entire sample set and their squares n times, each time leaving out
the i'th value. This procedure gets around that. First the entire set
and their squares are summed. Then, the first value is subtracted from the
sum, and its square from the sum of squares, and the appropriate psuedovalue
and z value are generated. Then the value is added back in and the next
value is subtracted out.
```

```
*****)
```

```
BEGIN
  IF SETG.SIZE<=2 THEN EXIT;
  MEANVAR(SETG,MEAN,VARIANCE);
  M :=SETG.SIZE;
  SUMZ :=0;
  SUMZSQ :=0;
  SUM :=0;
  SUMSQ :=0;
  FOR X := 1 TO SETG.SIZE DO
    BEGIN
      SUMSQ := SUMSQ + SQR(SETG.DATA[X]);
      SUM := SUM + SETG.DATA[X];
    END;
  FOR Y:= 1 TO SETG.SIZE DO
    BEGIN
      SUM :=SUM-SETG.DATA[Y];
      SUMSQ :=SUMSQ-SQR(SETG.DATA[Y]);
      MEANJ := SUM/(M-1);
      psuedoj[Y]:= (SUMSQ - (M-1)*SQR(meanJ))/(M-2);
      PSUEDOJ[Y]:= M*(VARIANCE)-(M-1)*psuedoj[Y];
      SUMZ :=SUMZ+PSUEDOJ[Y];
      SUMZSQ :=SUMZSQ+SQR(PSUEDOJ[Y]);
      SUM :=SUM+SETG.DATA[Y];
      SUMSQ :=SUMSQ+SQR(SETG.DATA[Y]);
    END;
  ZMEAN :=SUMZ/Y;
  ZVAR :=(SUMZSQ-(M*SQR(ZMEAN)))/(M-1);
END;
```

```
FUNCTION gammaln(xx: double): double;
```

```
(*****
This is an incomplete gamma function pulled from Numerical Recipes. It is
used to find T values and F values.
*****)
```

```
CONST
  stp = 2.50662827465;
  half = 0.5;
  one = 1.0;
  fpf = 5.5;
VAR
  x,tmp,ser: double;
  j: integer;
  cof: ARRAY [1..6] OF double;
BEGIN
  cof[1] := 76.18009173;
  cof[2] := -86.50532033;
  cof[3] := 24.01409822;
  cof[4] := -1.231739516;
  cof[5] := 0.120858003e-2;
  cof[6] := -0.536382e-5;
  x := xx-one;
  tmp := x*fpf;
  tmp := (x+half)*ln(tmp)-tmp;
  ser := one;
  FOR j := 1 to 6 DO BEGIN
```

```

        x := x+one;
        ser := ser+cof[j]/x
    END;
    gammln := (tmp+ln(stp*ser))
END;
FUNCTION betacf(a,b,x: double): double;
LABEL 1;
CONST
    itmax=100;
    eps=3.0e-7;
VAR
    tem,qap,qam,qab,em,d: double;
    bz,bpp,bp,bm,az,app: double;
    am,aold,ap: double;
    m: integer;
BEGIN
    am := 1.0;
    bm := 1.0;
    az := 1.0;
    qab := a+b;
    qap := a+1.0;
    qam := a-1.0;
    bz := 1.0-qab*x/qap;
    FOR m := 1 to itmax DO BEGIN
        em := m;
        tem := em+em;
        d := em*(b-m)*x/((qam+tem)*(a+tem));
        ap := az+d*am;
        bp := bz+d*bm;
        d := -(a+em)*(qab+em)*x/((a+tem)*(qap+tem));
        app := ap+d*az;
        bpp := bp+d*bz;
        aold := az;
        am := ap/bpp;
        bm := bp/bpp;
        az := app/bpp;
        bz := 1.0;
        IF ((abs(az-aold)) < (eps*abs(az))) THEN GOTO 1
    END;
    writeln('pause in BETACF');
    writeln('a or b too big, or itmax too small'); readln;
1: betacf := az
END;

```

```

FUNCTION betai(a,b,x: double): double;
(*****
This is an incomplete beta function used to generate T and Z values. See
Numerical Recipes.
*****)
VAR
    bt: double;
BEGIN
    IF ((x < 0.0) OR (x > 1.0)) THEN BEGIN
        writeln('pause in routine BETAI'); readln
    END;
    IF ((x = 0.0) OR (x = 1.0)) THEN bt := 0.0
    ELSE bt := exp(gammln(a+b)-gammln(a)-gammln(b)
        +a*ln(x)+b*ln(1.0-x));
    IF (x < ((a+1.0)/(a+b+2.0))) THEN
        betai := bt*betacf(a,b,x)/a
    ELSE betai := 1.0-bt*betacf(b,a,1.0-x)/b
END;

```

```

function tvalue(tstat:double;df:integer):double;

```

```

(*****
This is original, using a formula developed in Numerical Recipes.

```

note: this returns the p value of t statistic and df, for one
tailed test.

*****)

```
begin
tvalue:=betai(df/2,1/2,df/(df+tstat*tstat))/2;
end;
```

```
function fvalue(fstat:double;df1,df2:integer):double;
```

This returns the p value for fstat and the two degrees of freedom.
*****)

```
begin
fvalue:=betai(df2/2,df1/2,df2/(df2+df1*fstat));
end;
```

```
FUNCTION EXIST(temp:WORRO) : BOOLEAN;
(* this checks to see if the given ".set" file is on the disk *)
var
```

```
    OK      :BOOLEAN;
    TEMPP   :FILENAME;
    GO      :CHAR;
```

```
BEGIN
    TEMPP:=CONCAT(TEMP, '.SET');
    ASSIGN(fil,TEMPP);
    {$I-}
    RESET(FIL);
    {$I+}
    OK:=IORESULT = 0;
    IF NOT OK THEN
        EXIST := FALSE
    ELSE
        BEGIN
            CLOSE(FIL);
            EXIST:=TRUE;
        END;
    END;
END;
```

```
FUNCTION EXIST2(temp:filename) : BOOLEAN;
(* this checks to see if the file name passed in is on the disk *)
```

```
VAR FIL      :FILE;
    OK      :BOOLEAN;
    TEMPP   :FILENAME;
```

```
GO:CHAR;
BEGIN
    TEMPP:=TEMP;
    ASSIGN(fil,TEMPP);
    {$I-}
    RESET(FIL);
    {$I+}
    OK:=IORESULT = 0;
    IF NOT OK THEN
        EXIST2 := FALSE
    ELSE
        BEGIN
            CLOSE(FIL);
            EXIST2:=TRUE;
        END;
    END;
```

```

END;
END;

```

```

PROCEDURE SAVE(SETG:INFO);
(* this procedure writes the data set to a '.set' disk file *)
VAR OVER      :CHAR;
    TEMP      :FILENAME;
label 10;
BEGIN
10: TEMP:=CONCAT(SETG.NAME, '.SET');
   IF EXIST(SETG.NAME) THEN
       BEGIN
           WRITE('File Exists, Overwrite?: ');
           OVER:=READKEY;
           writeLn;
       END
   ELSE
       OVER:='Y';
   IF UPCASE(OVER) = 'Y' THEN
       BEGIN
           ASSIGN(FIL,TEMP);
           REWRITE(FIL);
           WRITE(FIL,SETg);
           CLOSE(FIL);
       END
   ELSE begin
       WRITE('New name ("q" to quit): ');
       readLn(setg.name);
       if (setg.name='q') or (setg.name='Q') then exit;
       setname:=setg.name;
       goto 10;
   end;
END;

```

```

PROCEDURE READIN(VAR SETG:INFO);
(* this procedure reads a ".set" file from disk into the SETG variable *)
VAR PAUSE      :CHAR;
    TEMP      :FILENAME;
    TEMPNAME   :WORD;
BEGIN
    writeLn;
    WRITE('Data file name? (',SETNAME,'): ');READLN(TEMPNAME);
    IF TEMPNAME <> '' THEN
        SETNAME:=TEMPNAME;
        SETG.NAME:=SETNAME;
        if exist(setg.name) then begin
            TEMP:=CONCAT(SETG.NAME, '.SET');
            ASSIGN(FIL,TEMP);
            RESET(FIL);
            READ(FIL,SETG);
            CLOSE(FIL);
        end
    else BEGIN
        writeLn;
        write('Data file does not exist ...');
        PAUSE:=READKEY;
    END;
END;

```

```

(* original *)
PROCEDURE TWOSETS(VAR SETA,SETB:INFO);
(* this provides the prompts for reading two '.set' data files from disk *)
BEGIN
    set=indo=(36,10,78,18);

```

```

writeln;
WRITEln('FIRST SET: ');
writeln('*****');
READln(SETA);
writeln;
writeln;
WRITEln('SECOND SET: ');
writeln('*****');
READln(SETB);
writeln;
END;

```

```

PROCEDURE MULTmat;
(* this procedure multiplies matrices, it is used by the MULTINORMAL *)
(* variate generator, which is used to sample dependent  $\beta$  parameters *)
(* **** note that it uses the matrices that are global variables to avoid *)
(* **** passing parameters and filling up the stack. *)
VAR X,V,Z :INTEGER;
    SUM :double;
BEGIN
    MAT3.ROWSIZE:=MAT1.ROWSIZE;
    MAT3.COLSIZ:=MAT2.COLSIZ;
    FOR X := 1 TO MAT1.ROWSIZE DO
        FOR V := 1 TO MAT2.COLSIZ DO BEGIN
            SUM:=0;
            FOR Z := 1 TO MAT1.COLSIZ DO
                SUM := SUM + MAT1.MATRIX[X,Z]*MAT2.MATRIX[Z,V];
            MAT3.MATRIX[X,V]:=SUM;
        END;
    end;

```

```

PROCEDURE READASCII;
(* this reads a single column ascii file and stores data in a ".set" file *)
TYPE
    FVL = TEXT;
VAR
    X,V,Z :INTEGER;
    FIL :FVL;
    GO,NAME :WORRD;
    SETG :INFOPTR;
    p :pointer;

BEGIN
    mark(p);
    new(setg);
    setwindow(36,10,78,18);
    WRITEln;
    WRITE('What ASCII file? : ');READln(NAME);
    writeln;
    IF NOT EXIST2(NAME) THEN BEGIN
        WRITE('Data file does not exist ...');
        PAUSE:=READKEY;
        writeln;
        EXIT;
    END
    ELSE BEGIN (* if the ascii file exists *)
        ASSIGN(FIL,NAME);
        RESET(FIL);
        setg^.SIZE := 0;
        WHILE NOT EOF(FIL) DO BEGIN
            setg^.SIZE:=setg^.SIZE+1; (* increment number of data points *)
            READln(FIL,setg^.DATA[setg^.SIZE]);
        END;
        CLOSE(FIL);
        WRITE('New file name? : ');READln(setg^.NAME);
        SAVE(setg); (* save the data to a disk file of ".set" type *)
    END;

```

```

window(1,1,80,25);
release(p);
END;

```

```

PROCEDURE WRITASCII;
(* this writes a data ".set" file to a single column ascii file *)
TYPE
    FVL = TEXT;

```

```

VAR
    X,Y,Z : INTEGER;
    FIL    : FVL;
    GO     : WORRD;
    name   : filename;
    SETG   : infoptr;
    p      : pointer;
BEGIN
    mark(p);
    new(setg);
    setwindow(36,10,78,18);
    READLN(setg);
    IF NOT EXIST(setg.NAME) THEN EXIT;
    WRITE('Name of new file : ');READLN(NAME);
    writeln;
    ASSIGN(FIL,NAME);
    REWRITE(FIL);
    for x:= 1 to setg.SIZE do
        WRITELN(FIL,setg.DATA[X]:12:5);
    CLOSE(FIL);
    window(1,1,80,25);
    release(p);
END;

```

```

PROCEDURE REEDKEY;
(* this procedure allow the user to enter data from the keyboard *)
(* into a ".set" file *)

```

```

VAR
    code,QUIT,X,Y,FVLE : INTEGER;
    OVER                : char;
    TEMPNAME            : WORRD;
    temp                : string15;
    temp2               : double;
    setg                : infoptr;
    p                   : pointer;
label 10;
BEGIN
    mark(p);
    new(setg);
    setwindow(36,10,78,18);
    writeln;
    WRITE('Name your data (',SETNAME,') : ');READLN(TEMPNAME);
    IF TEMPNAME <> '' THEN (* if a name was entered use it , otherwise use the default name *)
        SETNAME:=TEMPNAME;
    setg.NAME:=SETNAME;
    IF EXIST(setg.name) THEN begin
        writeln;
        WRITE('File Exists, Overwrite?: ');
        OVER := READKEY;
        writeln;
    end
    ELSE
        OVER:='Y';
    IF UPCASE(OVER)='Y' THEN
        BEGIN
            writeln;

```

```

WRITE('Enter "q" when done. ');
QUIT:=0;
X:=0;
WHILE QUIT=0 DO
BEGIN
  X:=X+1;
10:WRITEln;
  WRITE('Enter ELEMENT ['X,']: ');READLN(TEMP);
  (* reading a character variable and converting to a real number *)
  (* keeps the program from bombing if a data entry error is made *)
  val(TEMP,TEMP2,code);
  if code=0 then begin
    (* if the text was successfully converted to a number *)
    setg^.DATA[X]:=TEMP2;
    setg^.SIZE:=X;
  END
  else if upcase(temp[1])='Q' then quit:=1
    (* when the user enters a "q" the program will stop *)
    (* prompting for data points and save the data file *)
  else begin
    beep;
    goto 10;
  end;
END;
save(setg);
END;
release(p);
END;

```

```

PROCEDURE CHANGE;
(* This procedure allows you to change a value in the data set. *)
(* It is much easier to write and ascii file, edit it, and read it back in *)
VAR
  code,X,V : INTEGER;
  temp : string16;
  temp2 : double;
  SAIV,CHANJ : char;
  setg : infostr;
  p : pointer;
BEGIN
  mark(p);
  ne:=setg;
  set:=info(36,10,78,18);
  READLN(setg);
  IF not EXIST(setg.name) THEN exit;
  CHANJ:='Y';
  WHILE upcase(CHANJ)='Y' DO
  BEGIN
    writeLn;
    WRITE('Change something?: ');CHANJ:=READKEY;
    writeLn;
    writeLn;
    IF upcase(chanj)='Y' THEN
      BEGIN
        write('Which element? ');intREAD(X);
        writeLn;
        WRITE('Old value = ',setg^.DATA[X]);
        writeLn;
        write('New value = ');doubleRead(setg^.data[X]);
        writeLn;
        write('Save the data?: ');SAIV:=READKEY;
        writeLn;
        writeLn;
        IF upcase(SAIV)='Y' THEN
          BEGIN

```

```

        WRITE('Name the data : '); READLN(setg^.NAME);
        writeln;
        SAVE(setg^);
    END;
END;
END;
release(p);
END;

```

```

PROCEDURE SEE;
(* This writes a data set to the screen *)
VAR X,Y : INTEGER;
    setg : ^info;
    p : ^pointer;
BEGIN
    mark(p);
    new(setg);
    set:=info=(36,10,78,18);
    readln(setg^);
    IF not EXIST(setg^.name) THEN exit;
    window(1,1,80,25);
    CURSOR;
    FOR X := 1 TO setg^.size DO
        WRITE(SETG^.DATA[X]:B:4);
    PAUSE:=READKEY;
    release(p);
END;

```

```

PROCEDURE SORTHR (VAR SETG:INFO);
(* This is a heapsort for sorting values within a data set *)
(* This was lifted verbatim from Pascal Programmers Library *)
(* With the exception of the line before the "L9:" label, I *)
(* had to add this to stop the procedure from trying to access *)
(* a data set element at the zero index (causes an error) *)
LABEL L1,L2,L3,L4,L5,L7,L8,L9;
VAR I,U,L,R,PAUSE : INTEGER;
    NUM : double;
    case : char;
BEGIN
    IF SETG.SIZE <= 1 THEN EXIT;
L1: L:=SETG.SIZE DIV 2 + 1;
    R:=SETG.SIZE;
L2: IF L > 1 THEN
        BEGIN
            L := L - 1;
            NUM := SETG.DATA[L];
        END
    ELSE
        BEGIN
            NUM := SETG.DATA[R];
            SETG.DATA[R]:=SETG.DATA[1];
            R:= R - 1;
            IF R = 1 THEN
                BEGIN
                    SETG.DATA[1]:=NUM;
                    EXIT
                END
            END
        END;
L3: J := L;
L4: I:= J;
    J:=J+J;
    IF J>R THEN GOTO L8;
    IF J=R THEN GOTO L7;
L5: IF SETG.DATA[J] < SETG.DATA[J+1] THEN
        J:=J+1;

```

```

L7: SETG.DATA[I]:=SETG.DATA[J];
   GOTO L4;
L8: J := I;
   I:= J DIV 2;
   if I<1 then I:=1;
L9: IF (NUM <= SETG.DATA[I]) OR (J = L) THEN
   BEGIN
     SETG.DATA[J] := NUM;
     GOTO L2
   END
ELSE
   BEGIN
     SETG.DATA[J] := SETG.DATA[I];
     GOTO L8
   END;
END;

```

```

procedure quantiles;
(* This routine gets point estimates and 90% two-sided confidence intervals *)
(* for the quantiles with multiples of 10 ie. .1,.2,.3 etc. See Kleijnen. *)
var setg : tinfoptr;
    T : tpointer;
    p,n,q : double;

```

```

begin
  mark('');
  new(setg);
  set:=indow(36,10,78,18);
  READLN(setg);
  if (not EXIST(setg.name)) or (setg.SIZE <= 20) THEN begin
    write(' Insufficient sample size (<20) ); (* ASSUMPTIONS DEPEND ON N>20 *)
    pause:=readkey;
    release(T);
    exit;
  end;
  n:=setg.size;
  writeLn;
  write(' Please wait -- Sorting ...');
  sortrn(setg);
  window(1,1,80,25);
  clrscr;
  set:=indow(10,5,70,22);
  writeLn(' Quantile Estimation and 90% Confidence Intervals');
  writeLn;
  coloroncolor(blue,black);
  writeLn(' % Point Lo Hi');
  writeLn(' -----');
  coloroncolor(white,black);
  writeLn;
  for x:= 1 to 9 do begin (* SEE KLEIJNEN PAGE 36 FOR ALGORITHM DISCUSSION *)
    p:=x/10;
    q:=1-p;
    write(' ',x*10:3,'%',setg.data[trunc(x/10*n)]:13:2);
    write(setg.data[trunc(n*p-1.645*sqrt(n*p*(1-p)))+1]:13:2,' ');
    writeLn(setg.data[trunc(n*p+1.645*sqrt(n*p*(1-p)))+1]:13:2);
  end;
  writeLn;
  p:=0.25;
  repeat
    q:=1-p;
    write(' ',p*100:3:0,'%',setg.data[trunc(p*n)]:13:2);
    write(setg.data[trunc(n*p-1.645*sqrt(n*p*(1-p)))+1]:13:2,' ');
    writeLn(setg.data[trunc(n*p+1.645*sqrt(n*p*(1-p)))+1]:13:2);
    p:=p+0.5;
  until p>0.75;
  pause:=readkey;
  release(T);

```

end;

Procedure Probability;

(* This routine uses the estimator for binomial probabilities to get a *)
 (* nonparametric estimation of the probability of a value drawn from the *)
 (* data set's underlying population being less than X. The estimator *)
 (* and CI formulas can be found in any basic stats book. I used Elementary *)
 (* Statistics by Duxbury. *)

var setg :infopt;
 T :pointer;
 Lo,Hi,sum,number,p,q :double;
 n :integer;

begin

mark(T);
 new(setg);
 setg.ndo=(36,10,70,10);
 READLN(setg);
 if (not EXISTS(setg.name)) or (setg.SIZE <= 20) THEN begin
 write('Insufficient sample size (<20)); (* assumption of N>20 *)
 pause:=readkey;
 release(T);
 exit;
 end;
 writeLn('Function returns P(X<X));
 write('Enter X: ');double read(number);
 n:=setg.size;
 writeLn;
 write('Please Wait -- Sorting ...');
 writeLn;
 sortLn(setg);
 sum:=0;
 for x:= 1 to n do
 if setg.data[x]<number then sum:=sum+1;
 p:=sum/(n-1);
 q:=1-p;
 Lo:=p-1.645*sqrt(p*q/n);
 Hi:=p+1.645*sqrt(p*q/n);
 if (p*n < 5) or (q*n < 5) then begin (* q*n and P*n must be > 5 *)
 write('Insufficient data spread ...); (* for CI to hold water *)
 pause:=readkey;
 release(T);
 exit;
 end;
 writeLn;
 writeLn('Point estimate: ',p:7:5);
 writeLn('90% CI: Hi: ',Hi:7:5);
 writeLn('Lo: ',Lo:7:5);
 pause:=readkey;
 release(T);

end;

end.

```
{M 64000,0,655000}
{N+} (* sets the coprocessor on *)
Unit daveMenu;
```

```
(*****)
```

This unit contains all the Procedures necessary to support the scrolling menus in the main program. The menu choices and attributes are stored in MENU which is a record. Notice that this is a pointer variable, meaning that this variable resides in the HEAP, leaving the whole 64K of main data memory free for the application (main program). Also note that this is a global variable, eliminating the need to pass it as a parameter among the various Procedures, which would overload the stack section of memory.

```
(*****)
```

```
INTERFACE
```

```
Uses
```

```
  Crt;
```

```
  Dos;
```

```
type
```

```
String30=string[30];
```

```
String15=string[15];
```

```
ScreenPtr=ScreenType;
```

```
ScreenType=Record
```

```
  Pos : ARRAY[1..25,1..80] OF RECORD
```

```
    Ch : CHAR;
```

```
    At : BYTE;
```

```
  End;
```

```
  CursX,CursY : integer;
```

```
End;
```

```
(*****)
```

```
(* This enables storing *)
```

```
(* a text Screen to *)
```

```
(* memory for later use. *)
```

```
(* *)
```

```
(*****)
```

```
MenText=array [0..11,1..10] of string30;
```

```
MenPos=array [1..2,0..9,1..10] of integer;
```

```
MenFirst=array [1..10] of boolean;
```

```
MenScreen=array [1..10] of ScreenType;
```

```
MenInfoPtr = MenInfo;
```

```
MenInfo = record
```

```
  text:menText;
```

```
  pos :menPos;
```

```
  first:menFirst;
```

```
  Screen:menScreen;
```

```
  max:array [1..10] of integer;
```

```
  top:array [1..10] of integer;
```

```
  left:array [1..10] of integer;
```

```
End;
```

```
Filename = STRING[12];
```

```
Word = STRING[8];
```

```
VAR
```

```
  x : integer;
```

```
  Screen : ScreenPtr;
```

```
  Menu : MenInfoPtr;
```

```
  pause : Char;
```

```
Procedure ColorOnColor(letters,back:word);
```

```
Procedure Beep; (* original *)
```

```
Procedure DoubleRead(var value:double);
```

```
Procedure IntRead(var value:integer);
```

```
Procedure GenericBox(x1,y1,x2,x3,x4,y4,LineType:integer);
```

```
Procedure SetWindow(y1,x1,y2,x2:integer);
```

```

Procedure HorzSetMenu(wich:integer);
Procedure ScrollMenu(wich:integer;var go,go2:char;var temp:integer);
Procedure Highlight(xpos,ypos,choice,wich:integer);
Procedure MenuControl(wich:integer;var go,go2:char;var temp:integer);
Procedure ShowDir(FileSpec:FileName); (* turbo 4.0 or pascal programmers library *)
Procedure Prtsc;

```

IMPLEMENTATION

```

Procedure Prtsc; (* from Pascal Programmers Library, By Que *)

```

```

(*****
This Procedure forces a screen dump to the printer when in the
text mode, and will work in the graphic mode if a graphic screen
dump utility such as egadmp of the CHART package, has been
installed.
*****)

```

```

Var Reg:Registers;
Begin
  Intr($5,Dos.Registers(REG));
End;

```

```

Procedure ShowDir(FileSpec:FileName);

```

```

(*****
This Procedure displays a directory of the current disk using the
file specification given, such as *.* or *.pas. Taken from the
Que book Turbo Pascal Programmers Library.
*****)

```

```

type
  String80 =string[80];
const Columns = 5;
VAR  J,ColSize :integer;
      Reg      :registers;
      Ota      :ARRAY[1..43] OF BYTE;
      Attr     :BYTE;
      Pause    :CHAR;
Begin
  CLRSR;
  ATTR:=0;
  COLSIZE := 80 DIV COLUMNS;
  REG.OX :=OFS(OTA);
  REG.OS := SEG(OTA);
  REG.AX := $1A00;
  MSDOS(Dos.Registers(REG));
  ColorOnColor(yellow,blue);
  gotoxy(24,1);writeln('***** DIRECTORY *****');
  TextBackground(black);
  writeln;
  FileSpec := FileSpec + CHR(0);
  REG.OX := OFS(FileSpec[1]);
  REG.OS:= SEG(FileSpec[1]);
  REG.CX:=ATTR;
  REG.AX:=$4E00;
  MSDOS(Dos.Registers(REG));
  IF LO(REG.AX) <> 0 THEN
    Begin
      writeln('NO SET FILES FOUND');
      PAUSE:=READKEY;
      EXIT;
    End;

```

```

IF DTA[22] AND $10 <> 0 THEN WRITE ('[D]');
J := 31;
WHILE DTA[J] <> 0 DO
  Begin
    WRITE(CHR(DTA[J]));
    J := J+1;
  End;
REPEAT
  REG.OX := OFS(DTA);
  REG.OS := SEG(DTA);
  REG.AX := $4F00;
  MSDOS(Dos.Registers(REG));
  IF LO(REG.AX) = 0 THEN
    Begin
      IF WHEREX > (COLUMNS -1 ) * COLSIZE +1 THEN writeln;
      WHILE (WHEREX MOD COLSIZE) <> 1 DO WRITE (' ');
      IF DTA[22] AND $10 <> 0 THEN WRITE ('[D]');
      J := 31;
      WHILE DTA[J] <> 0 DO
        Begin
          WRITE (CHR(DTA[J]));
          J := J+1;
        End;
      End;
      UNTIL LO(REG.AX) <> 0;
      writeln;
      PAUSE:=READKEY;
    End;

```

```

Procedure ColorOnColor(letters,back:word);

```

```

(*****
this allows the programmer to change the text color and the background
color with one statement.
*****)

```

```

Begin
textcolor(letters);
TextBackground(back);
End;

```

```

Procedure Beep;
Begin
sound(500);
delay(250);
nosound;
End;

```

```

Procedure DoubleRead(var value:double);

```

```

(*****
This procedure does error checking while reading in double precision numbers.
It keeps the program from aborting if the number is entered improperly.
*****)

```

```

var temp : FileName;
    code : integer;
label 10;
Begin
10:readln(temp);
val(temp,value,code);
if (code<>0) then Begin
  Beep;
  write('Re-enter the value: ');

```

```

    goto 10;
End;
End;

```

```

Procedure IntRead(var value:integer);

```

```

(*****
This procedure does error checking while reading in integer numbers.
It keeps the program from aborting if the number is entered improperly.
*****)

```

```

var temp : FileName;
    code : integer;
label 10;
Begin
10:readln(temp);
write'n';
val(temp,value,code);
if (code<>0) then Begin
    Beep;
    write('Re-enter the value: ');
    goto 10;
End;
End;

```

```

Procedure GenericBox(x1,y1,x2,x3,y4,LineType:integer);

```

```

(*****
This procedure draws a box starting at the x1,y1 position, ending at the
x4,y4 position. Horizontal dividers are drawn at positions x2 and x3 if they
are set to something other than zero. Allows for drawing with single or double
lines, or a mix of both.
*****)

```

```

type
    bar = string[79];
var
    lyne,lyne2 : bar;
    hl,h12,vl,ul,ur,ll,lr,li,ri,ti,bi,milchar;
    X:integer;
Begin
window=(y1,x1,y4,x4);
clrscr;
windo=(1,1,80,25);
if LineType=1 then Begin
    ul:=#218;
    hl:=#196;
    ur:=#191;
    vl:=#179;
    ll:=#192;
    lr:=#217;
    ri:=#147;
    li:=#147;
End
else if LineType=3 then Begin
    HL:=#205;
    h12:=#196;
    UR:=#187;
    UL:=#201;
    LR:=#188;
    LL:=#200;
    ri:=#182;
    li:=#199;
    VL:=#186;

```

```

End
else if LineType=2 then Begin
  UL:=CHAR(201);
  HL:=CHAR(205);
  UR:=CHAR(187);
  LR:=CHAR(188);
  LL:=CHAR(200);
  rI:=CHAR(185);
  lI:=CHAR(204);
  VL:=CHAR(186);
End;
LYNE:='';
lyne2:='';
FOR X:=1 TO V4-V1-1 DO
  LYNE:=LYNE+HL;
if LineType=3 then for x:=1 to y4-y1-1 do
  lyne2:=lyne2+rI2
else lyne2:=lyne;
GOTOXY(Y1,X1);WRITE(UL);WRITE(LYNE);
GOTOXY(V4,X1);WRITE(UR);
FOR X:= X1+1 TO X4 DO
  Begin
    GOTOXY(V1,X); WRITE(VL);
    GOTOXY(V4,X); WRITE(VL);
  End;
IF X2<=0 THEN Begin
  GOTOXY(V1,X2);WRITE(LI);WRITE(LYNE2);
  GOTOXY(V4,X2);WRITE(RI);
End;
IF X3<=0 THEN Begin
  GOTOXY(V1,X3);WRITE(LI);WRITE(LYNE2);
  GOTOXY(V4,X3);WRITE(RI);
End;
GOTOXY(V1,X4);WRITE(LL);WRITE(LYNE);
GOTOXY(V4,X4);WRITE(LR);
End;

```

Procedure SetWindow(y1,x1,y2,x2:integer); (* turbo 4.0 documentatin *)

```

(*****
This procedure draws a box in the specified region and clears it for a pop-
up window.
*****)

```

```

var i:integer;
Begin
  window:= (y1-1,x1-1,y2+1,x2+1);
  clrscr;
  window:= (1,1,80,25);
  textcolor(blue);
  GenericBox(x1-1,y1-1,0,x2+1,y2+1,2);
  window:= (y1,x1,y2,x2);
  ColorOnColor:= (white,black);
End;

```

Procedure HorzSetMenu(wich:integer);

```

(*****
This procedure sets the positions for each line of text for a menu screen.
The coordinates are delimited by the top of menu, left edge of menu, and
how many choices are on the menu.
*****)

```

```

var x:integer;

```

```

Begin
  for x:= 1 to Menu^.max[wich] do Begin
    Menu^.pos[2,x,wich]:=x+3+Menu^.top[wich];
    Menu^.pos[1,x,wich]:=Menu^.left[wich]+2;
  End;
  Menu^.pos[2,0,wich]:=Menu^.pos[2,Menu^.max[wich],wich]+2;
  Menu^.pos[1,0,wich]:=Menu^.left[wich]+2;
End;

Procedure horzdrawMenu(wich:integer);

(*****
This procedures uses the others to draw a menu screen.
*****)

var x:integer;
Begin
  ColorOnColor(yellow,black);
  GenericBox(Menu^.top[wich],Menu^.left[wich],Menu^.top[wich]+2,
    Menu^.max[wich]+7+Menu^.top[wich]-1,Menu^.max[wich]+9+
    Menu^.top[wich]-1,Menu^.left[wich]+18,2);
  ColorOnColor(white,black);
  TextBackground(red);
  GOTOXY(Menu^.left[wich]+2,Menu^.top[wich]+1); WRITE(Menu^.text[Menu^.max[wich]+1,wich]);
  TextBackground(black);
  for x:= 1 to Menu^.max[wich]+1 do Begin (* use x-1 to get around not letting x be zero*)
    ColorOnColor(WHITE,BLACK);
    GOTOXY(Menu^.pos[1,x-1,wich],Menu^.pos[2,x-1,wich]);WRITE(Menu^.text[x-1,wich]);
    ColorOnColor(CYAN,BLACK);
    GOTOXY(Menu^.pos[1,x-1,wich],Menu^.pos[2,x-1,wich]);WRITE(Menu^.text[x-1,wich][1]);
  End;
End;

```

```

Procedure ScrollMenu(wich:integer;var go,go2:char;var temp:integer);

(*****
This procedure moves the highlighting bar up and down the menu as the arrow
keys are used. It also sets the menu choice each time it moves the bar
so that when the enter key is hit the proper menu choice will be selected.
*****)

```

```

Begin
  go:=readkey;
  if go='P' then
    if temp<Menu^.max[wich] then temp:=temp+1
    else temp:=0;
  if go='N' then
    if temp>0 then temp:=temp-1
    else temp:=Menu^.max[wich];
  go2:=Menu^.text[temp,wich][1];
End;

```

```

Procedure Highlight(xpos,ypos,choice,wich:integer);

(*****
This procedure writes a string of text in yellow on cyan highlighting.
*****)

```

```

Begin
  ColorOnColor(WHITE,cyan);
  gotoxy(xpos,ypos);write(Menu^.text[choice,wich]);
  ColorOnColor(white,black);
End;

```

```

Procedure MenuControl(wich:integer;var go,go2:char;var temp:integer);

(*****
This procedure is the master control module that takes over any time the
program enters a menu procedure. It handles the screen drawing, scrolling,
and returns the proper choice the user selected to the individual menu
procedure in the main menu, so that choice can be executed. Note that if this
particular menu has been called once before, the screen need not be drawn
again since it was stored in memory. The menu screen is simply recalled from
memory with the index WICH.
*****)

Begin
  IF (Menu^.First[wich]) THEN Begin
    Menu^.First[wich]:=FALSE;
    HorzSetMenu(wich);
    horzdrawMenu(wich);
    Menu^.Screen[wich]:=Screen^;
    Highlight(Menu^.pos[1,temp,wich],Menu^.pos[2,temp,wich],temp,wich);
  End
  ELSE if (go<>'P') and (go<>'H') then Begin
    Screen^:=Menu^.Screen[wich];
    go:=Menu^.text[1,wich][1];
    GO2:=Menu^.text[1,wich][1];
    temp:=1;
    Highlight(Menu^.pos[1,temp,wich],Menu^.pos[2,temp,wich],temp,wich);
  End;
  GOTOXY(Menu^.left[wich]+10,Menu^.pos[2,0,wich]+2);
  GO:=READKEY;
  if go=#0 then Begin
    Screen^:=Menu^.Screen[wich];
    ScrollMenu(wich,go,go2,temp);
    Highlight(Menu^.pos[1,temp,wich],Menu^.pos[2,temp,wich],temp,wich);
  End;
  if go=chr($00) then
    go:=go2;
  for x:= 1 to Menu^.max[wich]+1 do
    if upcase(go)=Menu^.text[x,wich][1] then Begin
      temp:=x;
      Screen^:=Menu^.Screen[wich];
      Highlight(Menu^.pos[1,temp,wich],Menu^.pos[2,temp,wich],temp,wich);
    End;
  End;
End;

(*****
This initializes the pointer heap variables and sets their initial values.
*****)

Begin
  new(Menu);
  new(Screen);
  Screen := PTR($8800,$0000);
  for x:= 1 to 10 do
    Menu^.First[x]:=true;
  End.

```

Bibliography

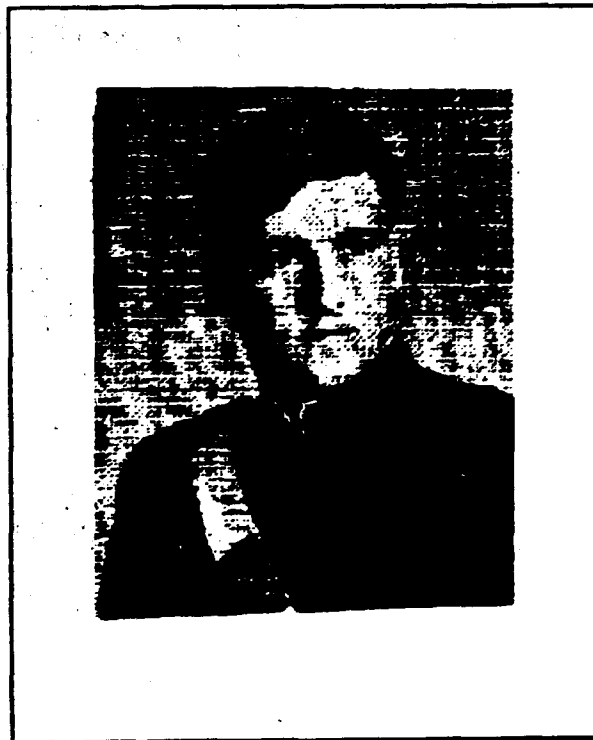
- Box, G.E.P. and Norman R. Draper, Empirical Model-Building and Response Surfaces. New York: John Wiley & Sons, 1987.
- Box, G.E.P. and P.W. Tidwell, "Transformation of the independent variables," Technometrics, 4:531-550 (1962).
- Devroye, Luc "The Analysis of Some Algorithms for Generating Random Variates with a Given Hazard Rate," Naval Research Logistics Quarterly, 33: 281-292 (1986).
- Department of Defense, Life Cycle Costing Guide For System Acquisitions (Interim). LCC-3, Washington: Government Printing Office, January 1973.
- Dienemann, Paul F. "Estimating Cost Uncertainty Using Monte Carlo Techniques," Memorandum RM-4854-PR, January 1966.
- Fisher, Gene H. Cost Considerations in Systems Analysis, New York: American Elsevier Publishing Company, 1972.
- Fishman, George S. "Sampling from the Gamma Distribution on a Computer," Communications of the ACM, 19:407-409 (July 1976).
- Greenberg, Irwin "A simple approximation for the simulation of continuous random variables," Simulation, 49:32-33 (July 1987).
- Johnson, Robert Elementary Statistics, Boston: Duxbury Press, 1980.
- Kleijnen, Jack P.C. Statistical Tools for Simulation Practitioners, New York: Marcell Dekker Inc., 1987.

- Kronmal, Richard A. and Arthur V. Peterson Jr. "An Acceptance-Complement Analogue of the Mixture-Plus-Acceptance-Rejection Method for Generating Random Variables," ACM Transactions on Mathematical Software, 10: 271-281 (September 1984).
- Leemis, Lawrence M. "Variate Generation for Accelerated Life and Proportional Hazard Models," Operations Research, 35: 892-894 (November-December 1987).
- Maindonald, J.H. Statistical Computation, New York: John Wiley and Sons, 1984.
- Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. Numerical Recipes: The Art of Scientific Computing, New York: Cambridge University Press, 1986.
- Pritsker, A. Alan B., Introduction to Simulation and Slam II, (third edition). New York: Systems Publishing Corporation, 1986.
- Ramberg, John s. and others "A Probability Distribution and Its Uses in Fitting Data," Technometrics, 21:210-214 (May 1979).
- Ross, Sheldon M., Introduction to Probability Models (third edition) New York: Academic Press, 1985.
- Rugg, T. and Feldman, P. Turbo Pascal Programmers Library, Indianapolis: Que Corporation, 1986.
- Seldon, M. Robert, Life Cycle Costing: A Better Method of Government Procurement, Boulder: Westview Press, Inc., 1979.
- Tadikamalla, Pandu R. "Computer Generation of Gamma Random Variables II," Communications of the ACM, 21:925-928 (November 1978).
- Wallace, N.D. "Computer Generation of Gamma Random Variates with Non-integral Shape Parameters," Communications of the ACM, 17:691-695 (December 1974).
- Winchmann, Brian and David Hill "Building a Random Number Generator," Byte, 127-128 (March 1987).

VITA

First Lieutenant David L. Sumner [REDACTED]

[REDACTED] He graduated from high school in Beulaville, NC in 1982. He entered the United States Air Force Academy on 28 June 1982 and majored in Operations Research. He graduated and was commissioned 28 May 1986. Prior to entering the School of Engineering, Air Force Institute of Technology in August 1988, he was an operations research analyst at the 28th Air Division, Tinker AFB, OK.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION / AVAILABILITY OF REPORT		
2b DECLASSIFICATION DOWNGRADING SCHEDULE			Approved for public release; distribution unlimited.		
4 PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GOR/ENS/90M-17			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION School of Engineering		6b OFFICE SYMBOL (If applicable) AFIT/ENS		7a NAME OF MONITORING ORGANIZATION	
6c ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433			7b ADDRESS (City, State, and ZIP Code)		
3a NAME OF FUNDING SPONSORING ORGANIZATION		8b OFFICE SYMBOL (If applicable)		9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO
			WORK UNIT ACCESSION NO		
11 TITLE (Include Security Classification) AN INTERACTIVE LIFE CYCLE COST FORECASTING TOOL (unclassified)					
12 PERSONAL AUTHOR(S) David L. Sumner, First Lieutenant, USAF					
13a TYPE OF REPORT MS Thesis		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 1990 March	
15 PAGE COUNT 175					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
05	03		LIFE CYCLE COSTS		
12	03		MONTE CARLO METHOD, RANDOM NUMBER GENERATION		
19 ABSTRACT (Continue on reverse if necessary and identify by block number)					
A tool was developed for Monte Carlo simulation of life cycle costs using parametric cost modeling. Additionally, the analysis of the performance of parametric CER cost estimation has been cut down to a more manageable task. Models can be built and tested quickly and easily. Random deviate generators were researched and built. Several applicable statistical description routines were also implemented. Statistical integrity and great accuracy has been maintained, while made accessible through an intuitive, user friendly interface.					
20 DISTRIBUTION AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION		
22a NAME OF RESPONSIBLE INDIVIDUAL David L. Sumner, First Lieutenant, USAF			22b TELEPHONE (Include Area Code) 513 255 3362		22c OFFICE SYMBOL AFIT/ENS