AECL EACL

AECL Research EACL Recherche

AECL-10983, COG-93-423

SYVAC3 Parameter Distribution Package

Progiciel à distribution de paramètres SYVAC3

Terry Andres
with contributions from / avec la collaboration d'Annette Skeet

 January 1995 janvier

AECL RESEARCH

SYVAC3 PARAMETER DISTRIBUTION PACKAGE

by

Terry Andres

with contributions from Annette Skeet

The Canadian Nuclear Fuel Waste Management Program is funded jointly by AECL and Ontario Hydro under the auspices of the CANDU Owners Group.

Whiteshell Laboratories
Pinawa, Manitoba, R0E 1L0
1995

AECL-10983
COG-93-423

PROGICIEL À DISTRIBUTION DE PARAMÈTRES SYVAC3

par

Terry Andres

avec la collaboration d'Annette Skeet

RÉSUMÉ

SYVAC3 (Systems Variability Analysis Code, generation 3) (Code d'analyse de variabilité des systèmes, 3^{e} génération) est un programme de calcul qui permet d'appliquer une méthode appelée Analyse variabilité des systèmes pour analyser le comportement probable d'un système devant l'incertitude. La méthode est basée sur la simulation répétée du système pour déterminer la variation de comportement qu'il peut manifester. SYVAC3 est utilisé spécialement pour les systèmes représentant la migration des contaminants, et comport plusieurs éléments permettant de simplifier la modélisation de ces systèmes. Il constitue un outil général de prédiction des impacts de la dispersion de contaminants sur l'environnement.

Dans le présent rapport, on décrit un type d'objets pour logiciel appelé Distribution de paramètres. On peut se servir de ce type d'objets dans SYVAC3 et aussi s'en servir indépendamment. Distribution de paramètres comporte les sous-types d'objets suivants : (1) Distribution bêta, (2) Distribution binomiale, (3) Distribution constante, (4) Distribution log-normale, (5) Distribution log-uniforme, (6) Distribution normale, (7) Distribution normale par morceaux, (8) Distribution triangulaire, et (9) Distribution uniforme. On peut transformer certaines de ces distributions en établissant la relation entre deux objets (paramètres de distribution).

On y donne les spécifications des distributions des paramètres et y explique comment s'en servir. Il devrait répondre aux besoins des utilisateurs occasionnels, examinateurs et programmeurs désirant ajouter leurs propres sous-types d'objets.

EACL Recherche
Laboratoires de Whiteshell
Pinawa (Manitoba) R0E 1L0
1995

AECL-10983
COG-93-423

SYVAC3 PARAMETER DISTRIBUTION PACKAGE

by

Terry Andres

with contributions from Annette Skeet

ABSTRACT

SYVAC3 (Systems Variability Analysis Code, generation 3) is a computer program that implements a method called systems variability analysis to analyze the behaviour of a system in the presence of uncertainty. This method is based on simulating the system many times to determine the variation in behaviour it can exhibit. SYVAC3 specializes in systems representing the transport of contaminants, and has several features to simplify the modelling of such systems. It provides a general tool for estimating environmental impacts from the dispersal of contaminants.

This report describes a software object type (a generalization of a data type) called Parameter Distribution. This object type is used in SYVAC3, and can also be used independently. Parameter Distribution has the following subtypes: (1) Beta Distribution, (2) Binomial Distribution, (3) Constant Distribution, (4) Lognormal Distribution, (5) Loguniform Distribution, (6) Normal Distribution, (7) Piecewise Uniform Distribution, (8) Triangular Distribution, and (9) Uniform Distribution. Some of these distributions can be altered by correlating two Parameter Distribution objects.

This report provides complete specifications for Parameter Distributions, and also explains how to use them. It should meet the needs of casual users, reviewers, and programmers who wish to add their own subtypes.

AECL Research
Whiteshell Laboratories
Pinawa, Manitoba R0E 1L0
1995

AECL-10983
COG-93-423

ACKNOWLEDGEMENTS

Annette Skeet made significant contributions to an earlier version of this document, *The SYVAC3 Computer Program—Volume 3: Parameter Sampling Package*. She also coauthored several other unpublished documents on SYVAC3 in the same series. These documents have been widely distributed in draft form, and have been used as a source of material for this manual. Annette Skeet has left AECL, and did not participate directly in writing this manual. However, her work lives on.

Bruce Goodwin, Garry Sherman, Anne Wills and Laverne Wojciechowski took the time and effort to review this document. The author very much appreciates their efforts and the many helpful suggestions they made.

Ted Iwanowski made many revisions to this document to correct errors and to improve its appearance and readability. In addition, he created the tables at the beginning and an early version of the index at the end. The author is very grateful for his assistance.

This work was carried out at AECL for the Canadian Nuclear Fuel Waste Management Program. This program is funded jointly by AECL and Ontario Hydro under the auspices of the CANDU Owners Group.

How to Use This Document

This document is a manual for the use and further development of the Parameter Distribution object type implemented in the Parameter Sampling Package (PSP) of the SYVAC3 computer program. It has three main parts. The first part, in chapters 1 to 3, is a user manual. These chapters describe SYVAC3 and Parameter Distributions, and they provide information so that programmers can use Parameter Distributions with programs other than SYVAC3. The second part, in chapters 4 to 6, specifies Parameter Distributions, Probability Distributions and Pseudorandom Generators in an object-oriented way. The third part, in chapters 7 to 12, presents design information stating how the current implementation was developed, and suggests alternatives. This preface helps the reader find the section most appropriate for his immediate needs.

This document is a manual. Its purpose is to provide a clear description of the PSP for users of the package, programmers who modify the package, and reviewers. It is intended that most readers will go immediately to the section that is of interest to them. Few readers will read the entire document from cover to cover. The manual contains several features to facilitate the intended use. First, there are three ways for the reader to find the relevant section: the flowchart below; extensive tables of contents, algorithms, tables and figures; and an index at the back. Second, the manual is divided into three parts. The first part (chapters 1 to 3) is a user manual, which provides information in the form needed by a programmer using the PSP. The second part (chapters 4 to 6) provides a formal specification of the software, and the third part (chapters 7 to 12) describes the design of the software. The last two parts are to help the developer understand and modify the PSP. The third feature of the manual to aid in quick reference is the division of the manual into two-page sections that tend to stand alone.

One side-effect of the design of the manual is a certain amount of redundancy. The intent is to avoid as much as possible making the reader search back and forth. This intent was not completely achievable. One way this intent was realized was to define abbreviations in each two-page section in which they occur. Two technical terms are used so frequently that they are not defined in each section. They are the terms CDF and PDF which are used throughout this report to refer to the cumulative distribution function and the probability density function respectively. In addition, SV309 is used throughout to refer to version 3.09 of SYVAC3.

The following flowchart can be used to find a particular section of the document:

[1] To learn about Parameter Distributions in general terms, or to use them inside or outside of SYVAC3, go to [2]; for more detailed information go to [5].

[2] To learn more about SYVAC3, or to acquire SYVAC3, including the code for Parameter Distributions, read Chapter 1; otherwise go to [3].

[3] To learn more about Parameter Distributions, including the subtypes and the operations provided, read Chapter 2; otherwise go to [4].

[4] To apply Parameter Distributions outside SYVAC3, read Chapter 3.

[5] To examine in detail the essential requirements for the Parameter Distribution object type and any related object types, go to [6]; for design information go to ?.

[6] To understand the essential requirements for the Parameter Distribution, Conditional Distribution, Truncation Interval, or Probability Distribution object types, read Chapter 4; for other object types go to [7].

[7] To understand the essential requirements for the subtypes of the Probability Distribution object type (e.g., Beta Distribution, Normal Distribution), read Chapter 5; for information on Pseudorandom Generators go to [8].

[8] To understand the essential requirements for the Pseudorandom Generator object type, read Chapter 6.

[9] To learn more about the design for the current implementation (SV309) of Parameter Distribution, go to [10]; otherwise look up related topics in the index and seek out the relevant passages.

[10] To learn more about the major design decisions that shaped the Parameter Sampling Package, including the choice of data structures used in the current implementation, read Chapter 7; otherwise go to [12].

[11] To obtain general information about the Parameter Sampling Package operations on Parameter Distributions and Probability Distributions, read Chapter 8; otherwise go to [12].

[12] To learn more about the algorithms used for routines associated with the Beta Distribution and Binomial Distribution subtypes, read Chapter 9; otherwise go to [13].

[13] To learn more about the algorithms used for routines associated with the Constant, Triangular and Uniform family of Probability Distribution subtypes, read Chapter 10; otherwise go to [14].

[14] To learn more about the algorithms used for routines associated with the Normal and Lognormal Distribution subtypes, read Chapter 11; otherwise go to [15].

[15] To learn more about the algorithms used for Pseudorandom Generators, read Chapter 12.

CONTENTS

continued...

CONTENTS (continued)

continued...

CONTENTS (continued)

continued...

CONTENTS (concluded)

LIST OF ALGORITHMS

LIST OF FIGURES

continued...

LIST OF FIGURES (concluded)

LIST OF TABLES

continued...

LIST OF TABLES (concluded)

1 SYSTEMS VARIABILITY ANALYSIS

1.1 What is Systems Variability Analysis?

Systems Variability Analysis (SVA) is a procedure for analyzing the probable behaviour of a system in the presence of uncertainty (Dormuth and Quick 1981). SVA is based on simulating the system many times to determine the variation in behaviour it can exhibit. SYVAC3 is a computer program shell used to perform Systems Variability Analysis.

SVA was developed to assess the environmental impacts of nuclear fuel waste disposal. In the disposal concept being considered, the wastes are placed in a vault deep underground in plutonic rock. The vault is then sealed, and all shafts and tunnels are filled to isolate the vault from the ground surface. Without actually building a disposal vault, researchers can use mathematical models of the disposal system to estimate its environmental impacts. The modelling process introduces uncertainty because the natural environment around the vault cannot be investigated in great detail without destroying its integrity. SVA is a procedure by which researchers can analyze the probable behaviour of the disposal system in the presence of this uncertainty.

Figure 1.1/1 shows how SVA works. As shown on the left, researchers identify both a system model and data appropriate for the assessment task. The system model is generally deterministic, that is, it takes single values for model parameters and derives single values for consequence variables. The system model may have internal structure, such as the vault, geosphere and biosphere components shown in the figure. The data inputs are not deterministic; instead, each parameter has a probability distribution that shows the relative likelihood of its taking particular values.

Specific values for the model parameters must be selected before the system can be simulated using the system model and parameter distribution. In SVA the selection for each simulation is done by random sampling. Many independent simulations are performed. Each one is a "what-if" experiment. Individually, simulations estimate the environmental impacts that would occur under specific sets of conditions. Taken collectively, a statistical analysis of the simulation results can produce frequency distributions showing the relative likelihood of different consequences.

There are other procedures that could generate data sets for simulations, but none is as representative as random sampling. For example, it would not be appropriate to assume the worst about every uncertain aspect of the disposal system, for that approach would quickly lead to irrational behaviour. Imagine a motorist driving to work who assumes the worst about every aspect of his trip. Instead of carrying a single spare tire, he fills the back of his van with them since it is possible that he could drive over not just one, but several nails and pieces of glass on his way to work. Instead of wearing one seatbelt, he wears at least two, since the first one may give way in an accident. In addition he has two airbags to protect the driver. He carries several tanks of fuel, in case some of them spring a leak or have faulty gauges. And he travels at 30 kilometres per hour on the highway, since greater speed can cause greater damage and injury in an accident.

A driver who behaves this way has gone far beyond the point of diminishing returns. And so it is with a waste disposal concept. An analyst should be realistic in his assessment, and conservative where there is doubt. If he wants to recommend a rational approach to waste disposal, he may not be able to assume the worst (or the best) about every situation. Instead he can employ available information to its fullest by applying SVA.

The box marked SYVAC in the figure represents the computer program used to conduct simulations of the target system. The labels on top of the box represent the respective roles of SYVAC and the models. SYVAC is the executive, controlling the operation of the system model, which comprises the submodels and the procedural glue to bind them together.

The dashed arrows represent the simulation loop. Each time they pass through the loop, parameter values sampled from the parameter distributions drive the models, producing consequence estimates that are stored for later analysis. Iteration translates the uncertainty in the inputs into a distribution of possible results in the output.

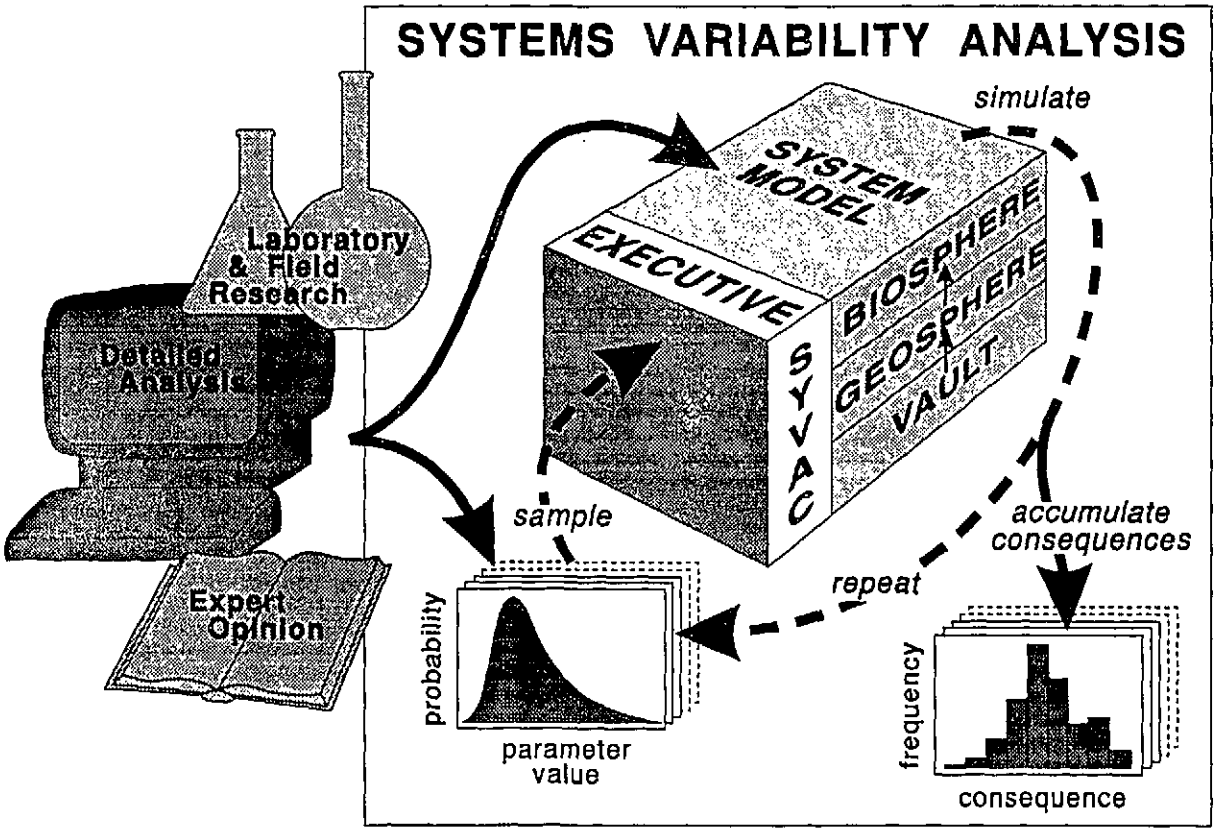

FIGURE 1.1/1: SYVAC3 Invokes a System Model Many Times to Determine the Distribution of Environmental Consequences

1.2 Four-Step Procedure for Systems Variability Analysis: An Example

Systems Variability Analysis (SVA) consists of a four-step procedure. Its essence can be illustrated by looking at an example where the amount of paint required to paint a fence is calulated (see Figure 1.2/1).

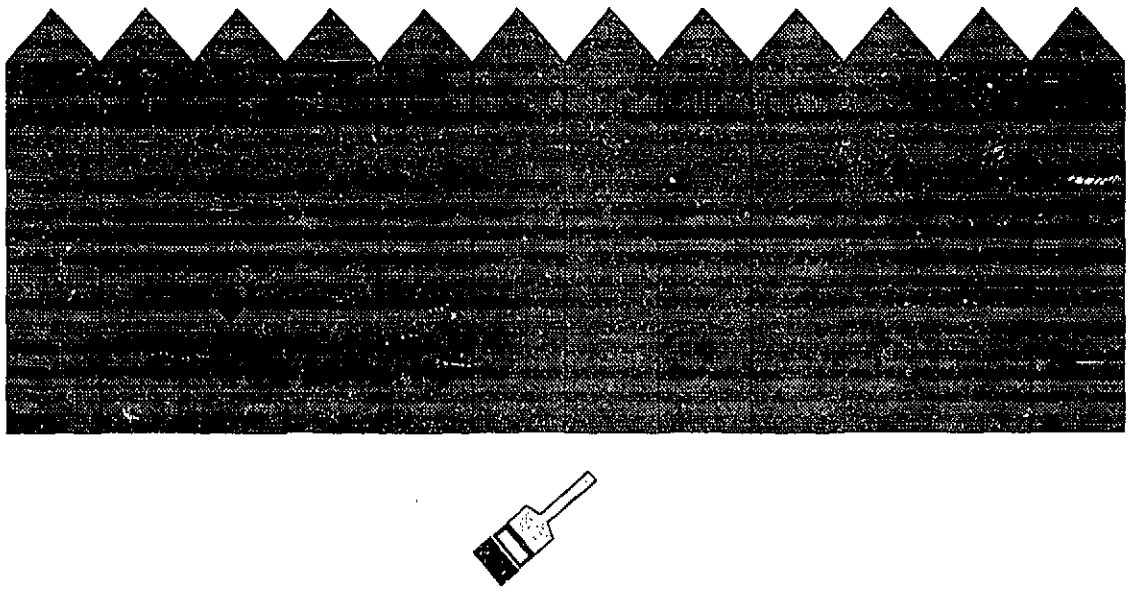

FIGURE 1.2/1: Paint the Fence

SVA applies to more than just assessment projects. It can be applied fruitfully in situations where

- partial information about a problematic situation is available–the situation concerns a system of related components (e.g., wasteform, barriers, groundwater);
- the general behaviour of the system is understood to some degree–for example, scientists may know the mechanisms by which contaminants can disperse in the environment; and
- the objective is to make a decision based on the value of a specified consequence, which is a quantitative measure of system behaviour (e.g., the total amount of contaminant released).

In these cases, a system model represents the actual system. By performing experiments on the system model, researchers find out how the model behaves. By extension, researchers can extrapolate this behaviour, when appropriate, to the actual system. As with an airplane cockpit simulator, researchers can try experiments that would be unacceptable or impossible in

real life. The validity of the results from such experiments depends on the validity and applicability of the system model and the parameter distributions.

SVA consists of the following four steps:

(1) Construct a mathematical model that allows you to calculate the specified consequence from the available information. Models must represent the system as it is understood to work. And of course the models must lead to precisely the required consequence.

(2) Fit probability distributions to parameters of the model using available numeric information that is consistent with your understanding of the system.

(3) Simulate the system many times by executing the model with random data sampled from the probability distributions. Store the simulation results.

(4) Analyze simulation results to determine the behaviour of the specified consequence.

To show the general nature of SVA, it is applied here to the simple problem of determining how much paint is needed to paint a fence. The mathematical model is shown in Table 1.2/1.

TABLE 1.2/1
SYSTEM MODEL FOR THE FENCE EXAMPLE

SAMPLED PARAMETERS

C	=	number of coats of paint to be applied [ ]
H	=	height of the fence [m]
L	=	length of the fence [m]
P	=	paint coverage [m^2/can]
S	=	number of sides to the fence to be painted [ ]

CONSEQUENCES

N	=	number of cans of paint required [can]

MATHEMATICAL MODEL

$$N = (L \cdot H \cdot S \cdot C)/P$$

Assume that the amount of paint required can be calculated using a simple formula based on the length of the fence, its height, and the paint coverage. If precise values are given for all the parameters of the model, the mathematical model can be used to calculate exactly the amount of paint required. If we are uncertain about the parameter values, and assign probability distributions to their values, then the amount of paint required will be a random variable. But can these parameter values be stated precisely? SVA uses probability distributions.

1.3 Parameter Distributions for the Fence Example

A typical model has some parameters with values that are known quite accurately, and others with values that are uncertain. The former are treated as constants in SVA, and the latter are assigned probability distributions. In the fence example, the number of sides of the fence and the number of coats of paint are considered constant. The other parameters are assigned normal distributions that are truncated below to prevent selection of negative values.

Usually the dimensions of a fence can be measured accurately, but in this case let us assume that the fence is inaccessible, perhaps because it is at a distant cottage. The dimensions must be estimated from memory, and so there is a considerable amount of uncertainty in the estimates. The paint coverage is naturally uncertain, since it depends on temperature, humidity, condition of the wood, and skill of the painter.

Normal distributions are used in this example because they are commonly known. They are shown in Table 1.3/1 and in Figure 1.3/1.

TABLE 1.3/1

DISTRIBUTIONS OF FENCE PARAMETERS

PARAMETER NAME	*SYMBOL*	*DISTRIBUTION TYPE*	*ATTRIBUTES*	*VALUES*
Number of coats	C	Constant	constant value	2 [ ]
Height of the fence	H	Normal	mean	1 [m]
			sigma	0.1 [m]
			lower bound	0 [m]
			upper bound	∞ [m]
Length of the fence	L	Normal	mean	25 [m]
			sigma	2 [m]
			lower bound	0 [m]
			upper bound	∞ [m]
Paint coverage	P	Normal	mean	60 [m^2/can]
			sigma	10 [m^2/can]
			lower bound	0 [m^2/can]
			upper bound	∞ [m^2/can]
Number of sides	S	Constant	constant value	2 [ ]

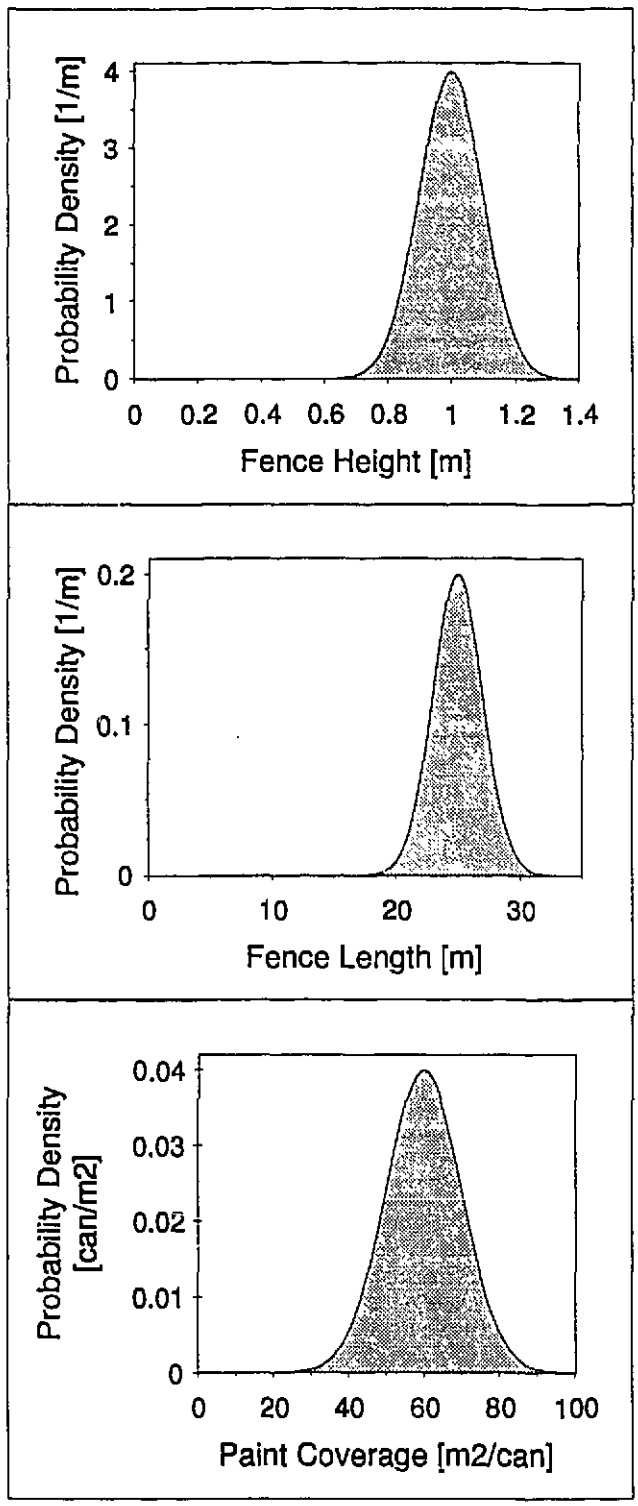

FIGURE 1.3/1: Parameter Distributions

1.4 Applying Systems Variability Analysis to the Fence Example

SVA operates by repeatedly simulating an uncertain system. In the fence-painting case, many sets of random values are generated for fence length, fence height and paint coverage. Each set is placed in a row of a table, along with the computed paint requirement, in cans. For 500 simulations, the table has 500 rows. This data set can be analyzed statistically to assess paint requirements.

Table 1.4/1 shows part of a table containing 500 simulations of the fence-painting example. Each row of the table corresponds to one simulation. In each row, the sampled parameters were randomly sampled from their distributions, and the number of cans of paint required was calculated from the mathematical model. Values for different simulations were chosen independently.

The constants (number of sides, number of coats) had the same values in every simulation. The other parameters took values that matched their distributions. For example, the theoretical distribution and the observed frequencies for fence length are plotted together in Figure 1.4/1. They show good agreement.

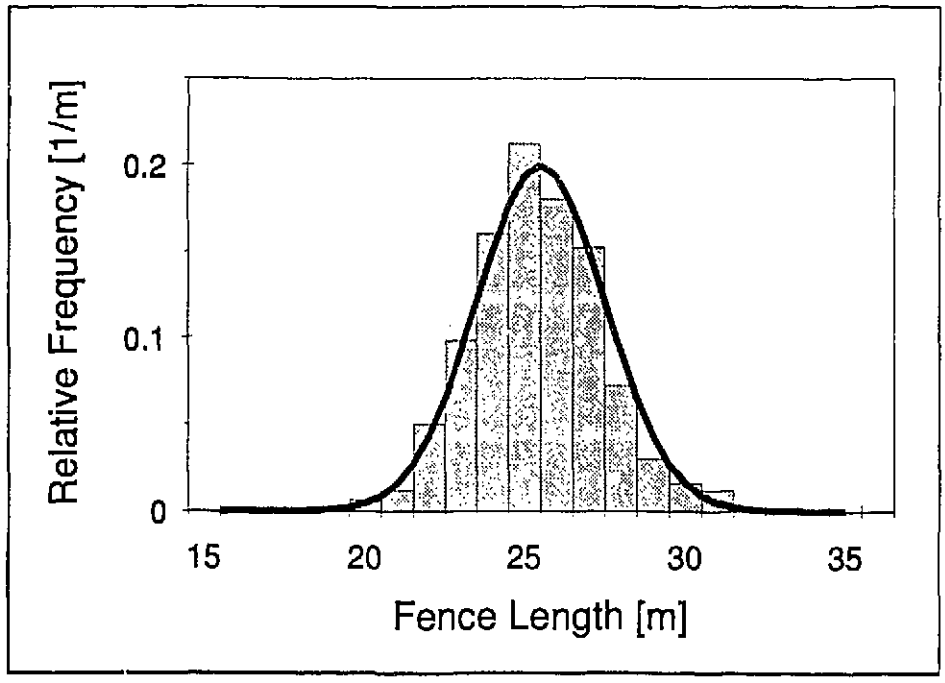

FIGURE 1.4/1: Theoretical Distribution and Relative Frequencies for Fence Length

TABLE 1.4/1

SOME SIMULATIONS OUT OF 500

	Number of Sides	Number of Coats	Fence Height	Fence Length	Paint Coverage	Cans of Paint
Min	2	2	0.76	19.1	32.4	0.9
Avg	2	2	0.99	24.9	60.4	1.7
Max	2	2	1.29	30.7	91.2	3.1
Std Dev	0	0	0.10	2.0	9.5	0.3
1	2	2	1.08	23.5	52.3	1.9
2	2	2	1.06	25.8	64.6	1.7
3	2	2	0.97	25.4	57.8	1.7
4	2	2	1.11	27.0	56.2	2.1
5	2	2	1.13	21.2	53.1	1.8
6	2	2	1.18	27.8	61.6	2.1
7	2	2	0.94	24.3	54.5	1.7
8	2	2	1.03	23.4	45.4	2.1
9	2	2	1.01	24.9	47.5	2.1
10	2	2	0.92	23.3	38.4	2.2
11	2	2	0.86	23.7	54.1	1.5
12	2	2	0.86	27.2	73.8	1.3
13	2	2	1.22	23.4	56.7	2.0
14	2	2	0.92	24.1	50.2	1.8
15	2	2	0.99	22.0	49.8	1.7
16	2	2	1.04	25.3	69.7	1.5
17	2	2	0.95	25.0	79.4	1.2
18	2	2	0.80	28.2	64.2	1.4
19	2	2	0.88	22.7	45.9	1.7
20	2	2	1.13	24.3	44.8	2.5
21	2	2	1.24	25.1	66.7	1.9
22	2	2	1.15	24.3	57.3	1.9
23	2	2	1.01	26.0	66.2	1.6
24	2	2	0.84	24.7	65.9	1.3
25	2	2	0.93	25.8	78.7	1.2
26	2	2	0.91	23.9	59.4	1.5
27	2	2	0.78	24.2	55.9	1.4
28	2	2	0.93	23.7	58.9	1.5
29	2	2	1.10	29.9	49.4	2.6
30	2	2	1.09	21.5	64.6	1.5
31	2	2	0.96	25.9	57.7	1.7
32	2	2	0.92	23.3	69.0	1.2
33	2	2	0.91	24.4	50.0	1.8
34	2	2	0.89	23.1	61.9	1.3
35	2	2	0.87	24.6	65.6	1.3
36	2	2	1.07	24.7	69.8	1.5
37	2	2	0.92	26.6	62.8	1.6
38	2	2	0.80	22.4	52.9	1.4
39	2	2	0.98	26.8	57.2	1.8
40	2	2	1.12	28.6	61.3	2.1

1.5 Results from the Fence Example

Figure 1.5/1 shows a histogram and a cumulative distribution plot obtained from 500 simulations of the fence painting example. From one to four cans of paint were required in the simulations. Table 1.5/1 shows the cumulative probabilities for whole cans of paint. An average of 1.7 cans is required. The painter can choose a solution that matches his level of risk-taking.

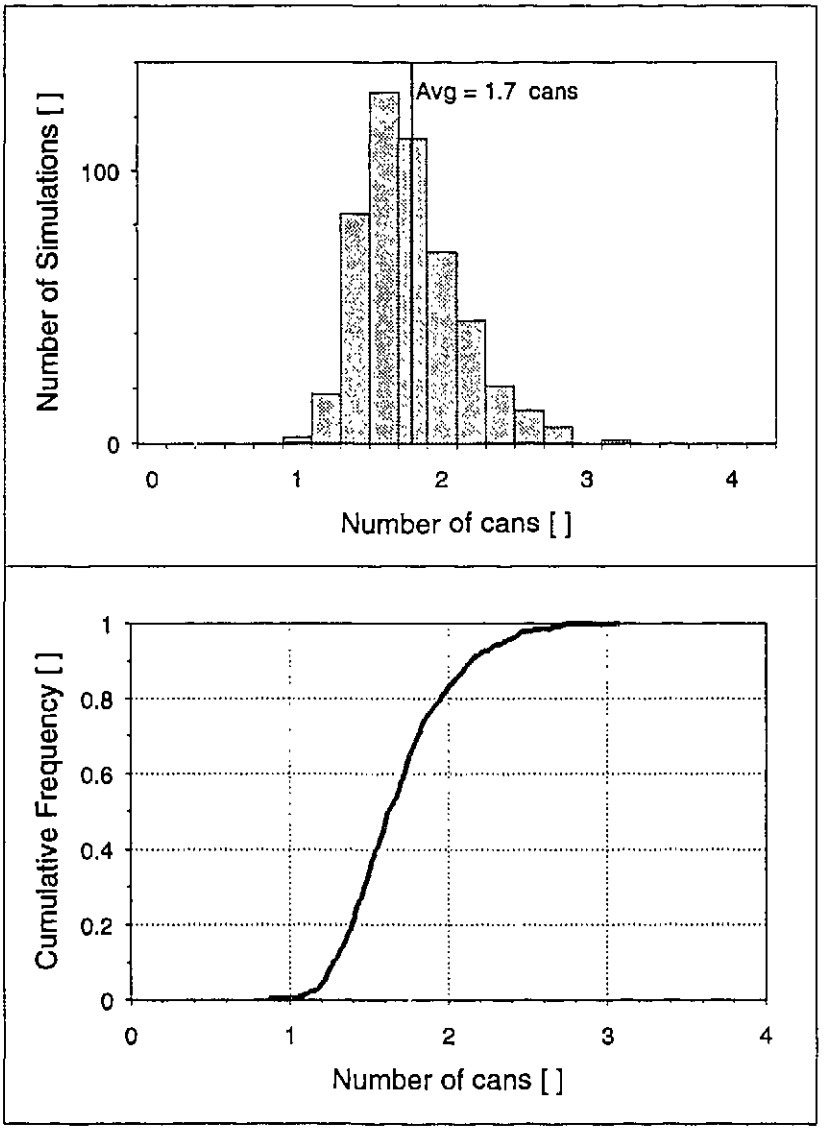

FIGURE 1.5/1: Histogram and Cumulative Frequencies of Cans of Paint Required

TABLE 1.5/1

CUMULATIVE FREQUENCIES FOR WHOLE CANS OF PAINT

Number of Cans	*Fraction of Simulations*
0	0.000
1	0.004
2	0.830
3	0.998
4	1.000

There are many ways of exploring the data in Table 1.4/1. One can plot histograms and cumulative distributions of computed (or sampled) quantities, as shown in Figure 1.5/1. Means, standard deviations, and other statistics can be estimated. Sensitivity analysis can be performed to identify particularly influential parameters. All of this information can be used by an analyst to understand the model and how it performs when parameters vary according to their distributions.

In the case of the fence example, the histogram shown in Figure 1.5/1 indicates a skewed distribution. That is, the peak of the distribution is to the left of the average value of 1.7 cans, and the tail to the right of the mean is longer than the tail to the left. As a result, there is a high probability that a small number of cans (i.e., one or two) will do. There is a low probability that more paint will be required, but in no simulation was there a need for more than four cans.

Decision makers can use this type of information as an aid in making sound decisions. But SVA does not tell a decision maker which decision to make. Considering Table 1.5/1, it is clear that one can of paint is unlikely to be sufficient. Two cans will be sufficient most of the time. Three cans will almost certainly provide enough paint. Four cans of paint will be too much in almost all cases. A risk-taking decision maker (or one who does not like to paint) may still choose one can as his initial purchase. A decision maker who is highly risk-averse may buy three or even four cans. Clearly other pieces of information, such as whether one can buy more matching paint later, could affect the decision.

In applying SVA, an analyst strives to obtain relevant and reliable information to help the decision maker reach his decision. The use of this method helps the decision maker deal rationally with situations fraught with uncertainty.

1.6 The Role of SYVAC3

SYVAC3 is a tool to assist analysts in carrying out the third step in SVA, i.e. simulating the system to yield an estimate of the specified consequence. SYVAC3 must be provided with the results of the first two steps in system variability analysis: models and data. SYVAC3 has five primary functions, shown in Figure 1.6/1. It links together six major object types: Case Simulation, Case Variable, Model Version, Parameter Distribution, SYVAC3 Case, and Variable Value.

SYVAC3 is a computer program shell; when linked with a system model it forms a computer program that performs simulations. Figure 1.6/1 shows five primary functions of SYVAC3:

(1) Input Case Data, including attributes of Case Variables and Parameter Distributions, and prepare to simulate.

(2) Sample Parameter Values for every Case Sampled Parameter in each Case Simulation.

(3) Simulate the System, that is, compute Variable Values for all Case Variables that depend on the Variable Values assigned to Case Sampled Parameters.

(4) Save Simulation Results–store Variable Values for every Case Variable in a Case Simulation so that they can be retrieved and analyzed later.

(5) Output Case Summary–wind up the simulations, and write a summary of what happened to the relevant output files.

The first and last steps in this procedure are carried out once for every case; i.e., once for every set of simulations performed as a group. The middle three steps are performed once for every simulation in the group. The looping arrow in the figure represents this iteration carried out within a case.

The capitalized phrases in this procedure represent major object types that play a role in SVA. SYVAC3 links these all together. In alphabetical order, they are:

(1) Case Sampled Parameter–a subtype of Case Variable that takes a sampled value in each Case Simulation in a SYVAC3 Case.

(2) Case Simulation–within a SYVAC3 Case, the operation of invoking a Model Version with a set of input Variable Values, thus generating a set of output Variable Values.

(3) Case Time Series Variable–a subtype of Case Variable that takes a value consisting of a single-valued function of time from time zero to the end time of a Case Simulation.

(4) Case Variable–a variable belonging to any of various subtypes that takes a unique value for each Case Simulation in a SYVAC3 Case.

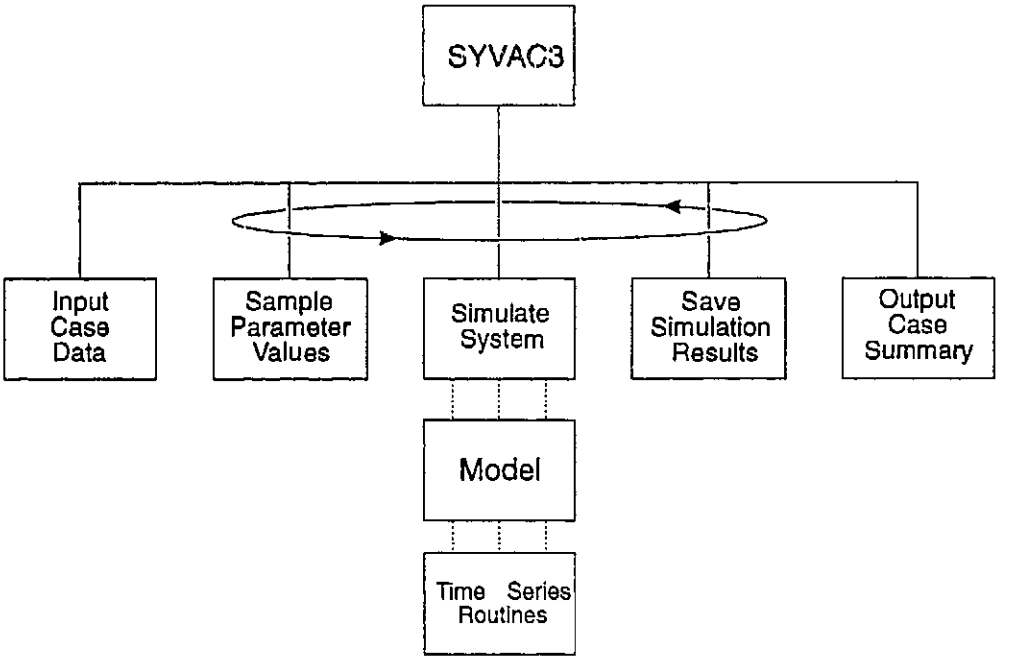

FIGURE 1.6/1: Primary Functions of SYVAC3

(5) Model Version–consists of a set of input Case Variables, a set of output Case Variables, and rules for computing Variable Values for the outputs, given the inputs.

(6) Parameter Distribution–one of several types of probability distributions (e.g., Normal Distribution or Uniform Distribution) assigned to a Case Sampled Parameter. It describes the relative likelihood of sampling Variable Values from different intervals.

(7) SYVAC3 Case–a set of Case Simulations performed as a group that all use the same Model Version and the same Case Variables with the same attributes. They differ in the Variable Values assigned to Case Variables.

(8) Variable Value–a value assigned to a Case Variable in a Case Simulation. A Variable Value can take a number as a value, or it can belong to a subtype with a structured value, such as a Time Series.

The Time Series Routines highlighted in Figure 1.6/1 perform a variety of operations on the subtype of Variable Value called Time Series. SYVAC3 is typically used with system models that represent the transport of contaminants. Time Series operations have greatly simplified the calculations in this type of model. As a result of their usefulness, they have grown to become a significant fraction of the SYVAC3 code.

1.7 SYVAC3 and Related Software

A modelling group using SYVAC3 codes a model and several interface routines, and compiles and links them together with SYVAC3. Programmers require a significant amount of time to train, and a considerable amount of effort is needed to prepare each application. Once prepared, the code is relatively efficient to run, and large simulations can be carried out. With somewhat less training and preparation, it is possible to use packages from SYVAC3 (e.g., the Parameter Sampling Package, and the Time Series Package) as libraries of routines. Other products that may be of value include the SAMPLE program, which generates input files based on a variety of statistical experimental designs, and the Mathematical Algorithm library that contains routines to model the one-dimensional transport of contaminants in the presence of advection, diffusion/dispersion, sorption, decay (n-member decay chains) and a variety of boundary conditions. The CC3 model is a very large system model representing the potential radioactive dose from nuclides in a high-level waste disposal vault. Many of the features of SYVAC3 were designed to support this model.

SYVAC3 is a computer program shell that is distributed as Fortran source code, so that users can compile and link their code directly with it. SYVAC3 comprises about 20 000 lines of Fortran code arranged in 170 modules. SYVAC3 has been coded in Fortran 77 for portability and execution efficiency. Typical users of SYVAC3 are familiar with Fortran and software development, and are prepared to code their models in Fortran.

Users can expect to go through a training period before they become expert in developing models to link with SYVAC3. Experience has shown that the greatest productivity comes from modifying existing simple models to create new ones. There are many details to learn, and so potential users should be willing to invest a significant amount in training and development if they choose to use SYVAC3.

A series of manuals provides support for users. These manuals typically combine functional specifications, design specifications, and user documentation in each volume. They are:

- SYVAC3 Manual (Andres in preparation): explains how to use the entire SYVAC3 system to develop simulation applications. Unlike the other manuals, it does not include specifications, but has only user documentation.

- Parameter Distribution Manual (this document): describes the Parameter Distribution software object, and how it may be used both within and without SYVAC3.

- File Reading Manual (Andres in preparation): explains how to use the File Reading Package to read text files and Fortran code. The operations provided by this package allow free format input that skips over comments and allows "logical lines" to span many physical lines in a file.

- Time Series Manual (Andres in preparation): describes the Time Series software object, and how it may be used both within and without SYVAC3. The Time Series

Package (TSP) comprises over half of SYVAC3. It is used in modelling to represent time-varying quantities and to solve systems of differential equations.

- SYVAC3 Specifications (Andres in preparation): specifies the parts of SYVAC3 not specified in the other manuals, showing how all the parts fit together.

Potential users with simple applications may prefer to use one of the available spreadsheet applications, such as @RISK[1] or Crystal Ball[2]. In these products the system model and parameter distributions reside in a spreadsheet, and the results of simulations are deposited there. Unlike SYVAC3, these packages have neither the capacity nor the speed to handle models with thousands of variables, thousands of simulations, and solutions to differential equations. However, they would be quite appropriate for the fence-painting example.

Certain parts of SYVAC3 can be used independently of the main code. For example, the Parameter Sampling Package could be used fairly easily to add random sampling to any Fortran program. The TSP handles functions of a single time variable in solving transport equations and compartment models for flows and concentrations of contaminants. The routines in this package could be used in any model, not just one linked to SYVAC3.

The following software has been developed in conjunction with SYVAC3:

- SAMPLE–This menu-driven program generates an input file for SYVAC3 that contains a statistical experimental design. A wide variety of possible designs are supported, including simple random sampling, fractional factorial, and latin hypercube, with or without discretization and importance sampling (Andres 1987).

- ML3–This Mathematical Algorithm Library provides response functions (i.e., Green's functions) to use with the Time Series Management Package in solving systems of one-dimensional transport equations. These equations describe advection, diffusion/dispersion, sorption, and radionuclide decay for any length of decay chain. Semi-infinite and mass transfer coefficient boundary conditions are supported.

- CC3–This model represents a hypothetical disposal system compatible with the Canadian concept for nuclear fuel waste disposal. The model describes the release of radionuclides from containers in the vault, transport through a network of one-dimensional flows in the geosphere, and estimation of dose by a wide variety of pathways in the biosphere. The model has thousands of parameters, and yields thousands of output variables, yet takes only a few minutes per simulation on a modern personal computer (PC).

---

[1] At this writing, @RISK works with Lotus 1-2-3 for MS-DOS and with Microsoft Excel for Windows; it is produced by Palisade Corp., 31 Decker Rd., Newfield, NY 14867.

[2] Crystal Ball for the Macintosh or for Microsoft Excel comes from Decisioneering, Inc., 1380 Lawrence St., #610, Denver, CO 80204.

2 PARAMETER DISTRIBUTIONS

2.1 What is a Parameter Distribution?

A Parameter Distribution is a software object that represents one of several types of possibly truncated probability distributions (e.g., Normal Distribution or Uniform Distribution). It describes the relative likelihood of randomly sampling a Variable Value for a Case Sampled Parameter at different locations. The Parameter Distribution object type supports several operations. The most important are the evaluation of a cumulative distribution function (CDF) and inverse CDF. These operations make it possible for SYVAC3 to perform random sampling of Case Sampled Parameters with different Parameter Distribution subtypes. The Parameter Sampling Package (PSP) implements these operations. It can be used independently of SYVAC3.

A Parameter Distribution is a software object. That is, it exists in software, and like any software object, it has attributes and operations. A software object type is an abstraction of a family of similar objects, all with the same attributes and operations. The concept of a software object type is a generalization of the traditional concept of data type. In Fortran 77 (the language of SYVAC3), the simplest software objects are variables and constants that have either character, integer, logical, real, complex or double precision values. They have attributes such as precision and value. The Fortran language defines operations such as addition or multiplication for these data types. Fortran 77 data types together with their attributes (e.g., numeric value) and operations (e.g., +) form simple numeric object types.

Unlike languages such as C and Pascal, Fortran 77 has very limited support for programmer extensions to the set of data objects. The only structuring mechanisms are to define arrays of simple values or common blocks containing variables. It is not possible in Fortran 77 to define new data types. It is also not possible to define operations for existing data types, although these can be emulated with subroutines and function calls.

Some other languages like Pascal and C do permit the programmer to define new data types; these types can have attributes that are stored in named fields like a common block. But whereas a Fortran common block is unique, in Pascal and C the programmer can define multiple objects, all with the same data type. In this way these languages provide support for more complicated software objects. Nevertheless, data types defined in these languages do not have associated operations. Some newer languages like C++ allow programmers to define object types with specified attributes (fields) and also operations. These object types can exhibit encapsulation (the storage representation is hidden, or encapsulated), inheritance (new object types are based on old ones) and polymorphism (operators like "+" work differently for different object types).

Several authors (e.g., Coad and Yourdon 1990; Rumbaugh et al 1991) have shown that software object types can be specified even if the final implementation occurs in a language such as Fortran 77 that does not directly support object types. This manual describes several types of software objects using the object-oriented paradigm. Their names are capitalized wherever they appear. Later chapters show how these object types can be implemented in Fortran 77. The Parameter Distribution object type in particular is the focus of the manual.

Informally speaking, a Parameter Distribution is an object consisting of a probability distribution that is possibly truncated and confined to a finite interval. Each Case Sampled Parameter in a SYVAC3 Case has an associated Parameter Distribution object that describes the relative likelihood of randomly sampling Variable Values from different locations in the permitted interval. More formally, the Parameter Distribution object type consists of a Probability Distribution object type combined with a Truncation Interval object type. This formal definition is discussed more fully in Chapter 4.

As an example, suppose that a model of waste migration has a Case Sampled Parameter called "Path Length," or *PATHL* for short, and that *PATHL* is normally distributed, i.e., it is linked to a Parameter Distribution containing a Normal Distribution. That Normal Distribution has two attributes, a Mean Value and a Standard Deviation. Suppose that the Mean Value is 7000 m and the Standard Deviation is 1000 m. The PDF and CDF of *PATHL* appear in Figure 2.1/1.

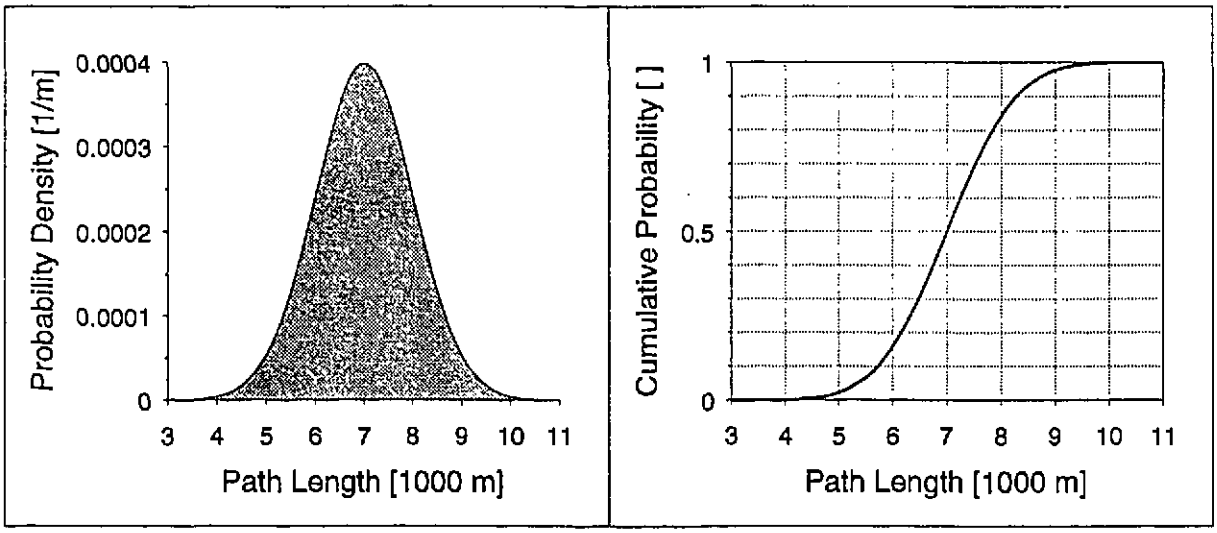

FIGURE 2.1/1: Parameter Distribution for a Normal Variate *PATHL*

The PDF plot shows the well-known bell-shaped curve of a normal PDF. Values of *PATHL* are shown across the horizontal axis in the CDF plot, and the corresponding probabilities are plotted on the vertical axis. For example, the plot shows there is a probability of 0.5 that *PATHL* would have a value less than 7000 m, and a probability of 0.9 that *PATHL* would have a value less than about 8300 m.

The PSP converts back and forth between values of Case Sampled Parameters and the corresponding cumulative probabilities, in accordance with Parameter Distributions. These conversions occur during the input of parameter distributions into SYVAC3, and during the sampling of parameter values for each simulation. In this example, the PSP converts back and forth between the values of *PATHL* and the corresponding cumulative probabilities. Mathematically, it evaluates two functions: the CDF and the inverse CDF. These evaluations are available not just for a Normal Distribution, as in this example, but for all the different Parameter Distribution subtypes.

2.2 How SYVAC3 Uses the Parameter Sampling Package

SYVAC3 uses the Parameter Sampling Package (PSP) to convert between probabilities and quantiles of a Parameter Distribution. The most important application is sampling Variable Values for Sampled Parameters. The cumulative probability associated with each sampled value comes from either a Pseudorandom Generator or from an external Sampling Method. The PSP inverts the CDF of the Parameter Distribution to convert the cumulative probability to a quantile, which becomes the new sampled value. SYVAC3 also uses the PSP to convert Truncation Limits on a Parameter Distribution into both probability and quantile forms.

Figure 2.2/1 shows the main operations of the object type Parameter Distribution: (1) to evaluate the CDF at a specified point in the distribution to yield a cumulative probability, and (2) to invert the CDF to find a quantile of the distribution, given a cumulative probability. These operations are implemented in the PSP so that SYVAC3 can use them in random sampling.

SYVAC3 uses a technique called the probability transform method (Rubinstein 1981) to sample random values for Sampled Parameters. If X is a Sampled Parameter with a CDF $F(x)$, the random variable $Y = F(X)$ is distributed uniformly between zero and one. Conversely, if Y is distributed uniformly between zero and one, then $X = F^{-1}(Y)$ is distributed according to the CDF $F(x)$. Variates like Y with a uniform distribution between zero and one can be generated in many different ways. A typical way, as in SYVAC3, is to use a Pseudorandom Generator, i.e., an object that uses a deterministic formula to generate a sequence of values that resembles a random sequence. Then values can be generated for any parameter X by inverting its CDF at a value y generated by the Pseudorandom Generator.

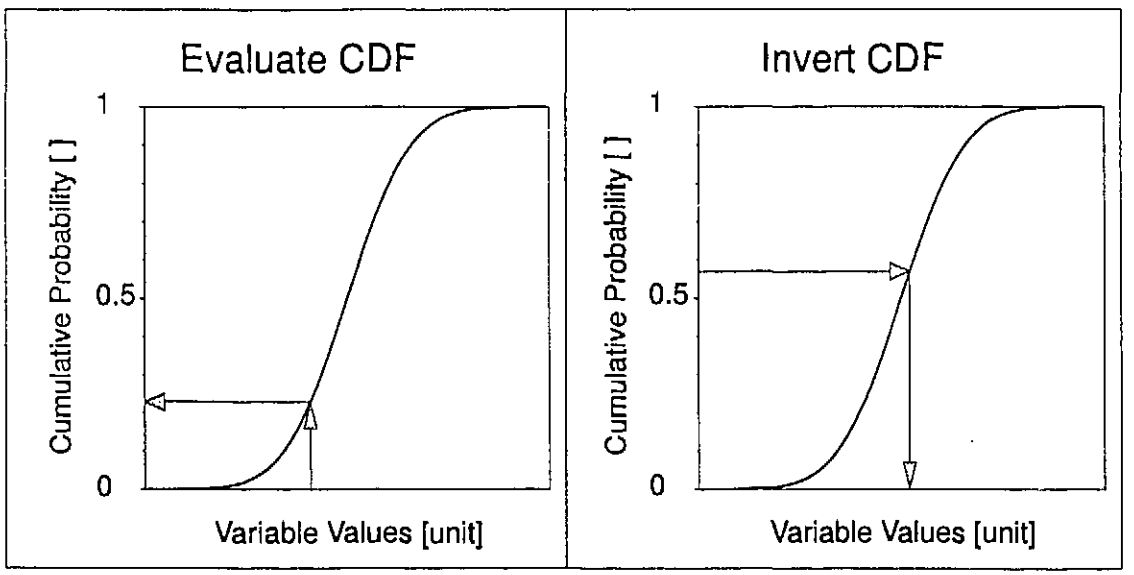

FIGURE 2.2/1: Transformations Through the CDF

Truncation Limits arise when the expert providing a Parameter Distribution stipulates lower and/or upper limits on the possible values of a parameter. For example, without limits, the distribution mentioned in Section 2.1 for the parameter *PATHL* (normal with mean 7000 m and standard deviation 1000 m) can take on any value from negative infinity to positive infinity. There are no intrinsic limits on a Normal Distribution. Values that are further than three or four standard deviations from the mean are extremely unlikely to arise, but they are possible. Values of *PATHL* less than zero are meaningless. An expert may therefore stipulate a lower Truncation Limit on *PATHL*, perhaps at zero, or perhaps at some larger value. Alternatively, the expert could specify a lower "quantile bound" of 0.005. This means that the lowest acceptable value of the parameter is the quantile having the associated cumulative probability of 0.005.

When a Parameter Distribution has Truncation Limits, the sampling process changes. Cumulative probabilities between zero and one must be mapped to a smaller interval that corresponds to the Truncation Limits, as shown in Figure 2.2/2. Variable Values generated in this way will follow the appropriate truncated distribution.

SYVAC3 uses the PSP to transform Truncation Limits. The expert providing a Parameter Distribution can specify limits as either cumulative probabilities or as Variable Values. Whichever is provided, SYVAC3 calculates both values and reports them in an output file. The transformations again involve evaluating or inverting a CDF.

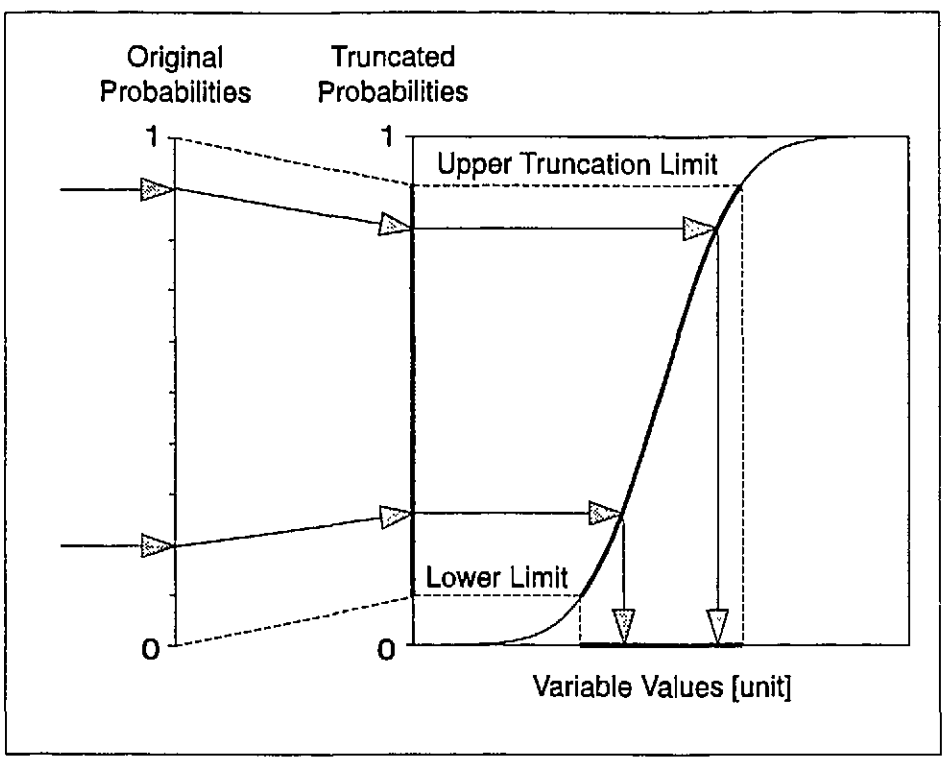

FIGURE 2.2/2: Effect of Truncation Limits

2.3 Features of the Parameter Sampling Package

The Parameter Sampling Package (PSP) currently supports the Parameter Distribution type and 10 different subtypes. It also contains a Pseudorandom Generator instance called SUPRAN (Walker 1985). The PSP is part of SYVAC3, but other programs can use it too. The PSP can evaluate and invert the CDF for each Parameter Distribution subtype. It can sample a Variable Value for any Sampled Parameter, whatever its Parameter Distribution subtype, from a single random number. Conversions are done in double precision, with an accuracy of at least eight significant figures, and usually more. Each Parameter Distribution is checked for internal consistency on each operation.

The PSP implements 10 subtypes of the Parameter Distribution object type, and one instance of a Pseudorandom Generator object. The following are the Parameter Distribution subtypes:

(1) Beta Distribution–has a variety of shapes of PDF on a finite domain.

(2) Uniform Distribution–any value in the domain is equally likely to be sampled.

(3) Constant Distribution–the same constant value is always sampled.

(4) Lognormal Distribution–when the PDF is plotted on a logarithmic scale, the PDF looks exactly like a normal distribution.

(5) Loguniform Distribution–when viewed on a logarithmic scale, any value in the domain is equally likely to be sampled.

(6) Normal Distribution–the normal or gaussian distribution has a PDF with the familiar bell shape..

(7) Piecewise Uniform Distribution–converts a histogram into a distribution; it can also represent finite discrete distributions.

(8) Triangular Distribution–the triangular PDF has a single mode in a finite domain.

(9) Correlated Normal Distribution–sampled values depend upon values sampled from an independent Normal Distribution or Lognormal Distribution; the marginal PDF is normal.

(10) Correlated Lognormal Distribution–sampled values depend upon values sampled from an independent Normal Distribution or Lognormal Distribution; the marginal PDF is lognormal.

PDFs of these distributions appear in Section 4.8. Stephens et al. (1989) provide additional information on how and when to use each Parameter Distribution subtype. The Binomial Distribution is also a Parameter Distribution subtype, but it is not completely integrated into the PSP yet. It is described in Sections 5.3, 5.4 and 9.5.

By design, the PSP is independent of SYVAC3; other programs can use it. Such programs should be written in Fortran 77, or should be able to call Fortran subroutines. Calls to the package appear in the code, which must be linked with the library containing the PSP.

A modeller writing a model to be linked with SYVAC3 can also call the PSP directly. For example, a model might use a random process to represent rainfall over successive years. To implement the random process, the model could issue calls to the Pseudorandom Generator, SUPRAN, in the PSP to sample rainfall for each year. This approach is appropriate when the number of values to be sampled is not known in advance. When a specific set of variables is to be assigned single values, it is easier for the modeller to make them into Case Sampled Parameters, because then SYVAC3 takes care of sampling and storing of values. Since SUPRAN supports generating multiple pseudorandom sequences of values at the same time, calls to SUPRAN from within a model would not affect the regular sampling of parameters.

When used by SYVAC3 or another program for random sampling, the PSP guarantees that each value sampled from a Parameter Distribution requires only one random seed. (In contrast, some procedures for generating normal variates require 12 random seeds.) The probability transform method of sampling, combined with support for Truncation Limits, makes single-seed sampling possible. Each value is sampled by generating a uniform variate between zero and one with SUPRAN (or another generator) and transforming that value by inverting the CDF of the appropriate Parameter Distribution (see Section 2.2). If the same sequence of parameters is sampled more than once from the same sequence of random seeds, the same values will always be obtained. If two SYVAC3 Cases are identical apart from some Parameter Distributions, and if the two use the same sequence of seeds, the values of all parameters with unchanged distributions will stay the same. This feature permits the use of sensitive statistical tests to determine the effect of changing a Parameter Distribution.

Transformations carried out by the PSP use high precision. All routines in the PSP operate in double precision (i.e. with about 16 decimal digits precision on a computer with a 32-bit word size). All CDF and inverse CDF routines are accurate to at least eight significant figures, and most are more accurate. The Pseudorandom Generator in the package, SUPRAN, uses all 32 bits of precision in the underlying random seed. It generates a double precision uniform variate between zero and one with about nine digits of precision. An emphasis is placed on high precision because the probability transform method is used for parameter sampling in SYVAC3. One potential disadvantage of this technique is loss of precision in distribution tails because of a coarse spacing of possible sampled values. Maintaining high precision in the computations makes the spacing finer and avoids this problem.

External code can treat routines in the PSP like black boxes that perform well-defined functions. If the call to such a routine is not correct, however, it may not be obvious to the calling program when it gets unreliable numbers back. To increase the likelihood of correct usage, conversions done by the main subroutines in the package (TRAVAL and TRAQUA) begin with consistency checks. For example, the standard deviation of a normal distribution should be positive, and the left range end of a uniform distribution should not be greater than the right range end. TRAVAL and TRAQUA test such properties and return an error flag to the calling routine. In SYVAC3, these checks are applied to Parameter Distributions as they are read from an input file, so that erroneous inputs can be flagged to the user.

3 CALLING ROUTINES IN THE PARAMETER SAMPLING PACKAGE

3.1 Structure of the Package

Most calls to the Parameter Sampling Package (PSP) invoke operations on Parameter Distribution or Pseudorandom Generator objects. CKDIST, TRAQUA and TRAVAL invoke operations on Parameter Distributions, and SUPRAN generates a random value from a Pseudorandom Generator. These routines in turn call other subroutines in the package. The lower level routines perform useful mathematical functions, such as evaluation of the error function erf(x). Users of the PSP can invoke the mathematical routines directly. This chapter provides the argument lists for all routines in the PSP that could have value to users. The main subroutines and their interconnections are shown in Figures 3.1/1 and 3.1/2.

TRAQUA converts from a probability to a Variable Value by inverting the cumulative distribution function (CDF) of a Parameter Distribution. TRAQUA performs the calculations internally for a simple distribution like the Uniform Distribution. For more complicated distributions, like the Normal Distribution, TRAQUA calls an external routine. Figure 3.1/1 shows the routines called by TRAQUA. A dashed line to INVBIN indicates that INVBIN exists but TRAQUA does not yet call it in SV309. Double bars along the sides of a box indicate a subroutine; single lines indicate a function.

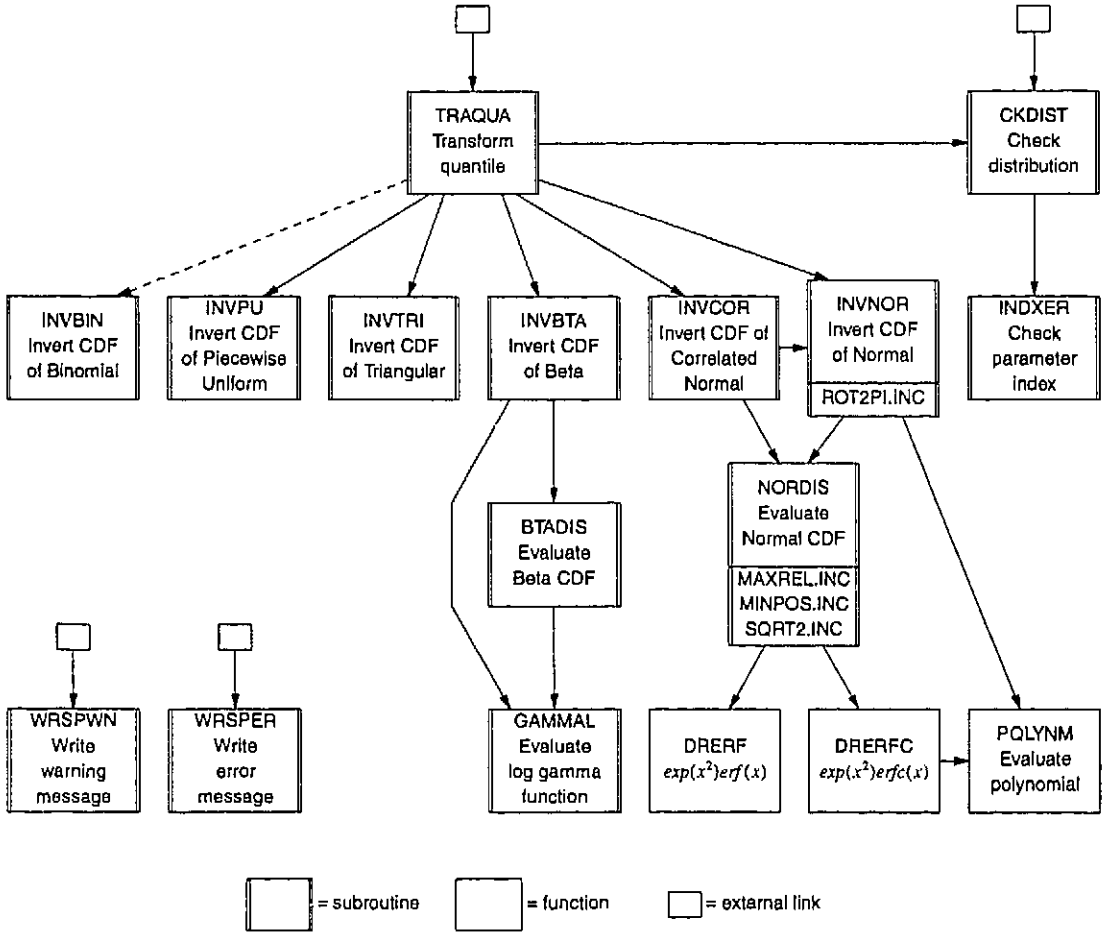

FIGURE 3.1/1: Structure Chart for TRAQUA

TRAVAL evaluates the CDF of a Parameter Distribution to convert from a Variable Value to a cumulative probability. As with TRAQUA, TRAVAL calls an external routine only for the more complicated distributions. Figure 3.1/2 shows the routines called by TRAVAL. TRAVAL does not yet call BINDIS in SV309. Some routines appear in both Figures 3.1/1 and 3.1/2.

Both TRAQUA and TRAVAL call CKDIST to check the validity of a Parameter Distribution before performing their computations. CKDIST may also be called independently to check a Parameter Distribution, as shown in Figures 3.1/1 and 3.1/2.

The routines WRSPER and WRSPWN shown Figures 3.1/1 and 3.1/2 are general error-handling routines that may be called from anywhere in the PSP.

SUPRAN implements a Pseudorandom Generator. Each call to SUPRAN generates a new uniform random number between zero and one. SUPRAN does not appear in the structure charts because it is a single routine that does not call any other routines.

Also not shown in the diagrams are the functions DERFC and DERF and the subroutine EXERFC. These routines evaluate the mathematical functions erfc(x), erf(x) and exp(x)(erfc(y) - erfc(z)) respectively. They have been placed in this package as general library routines because their evaluation is similar to the evaluation of NORDIS.

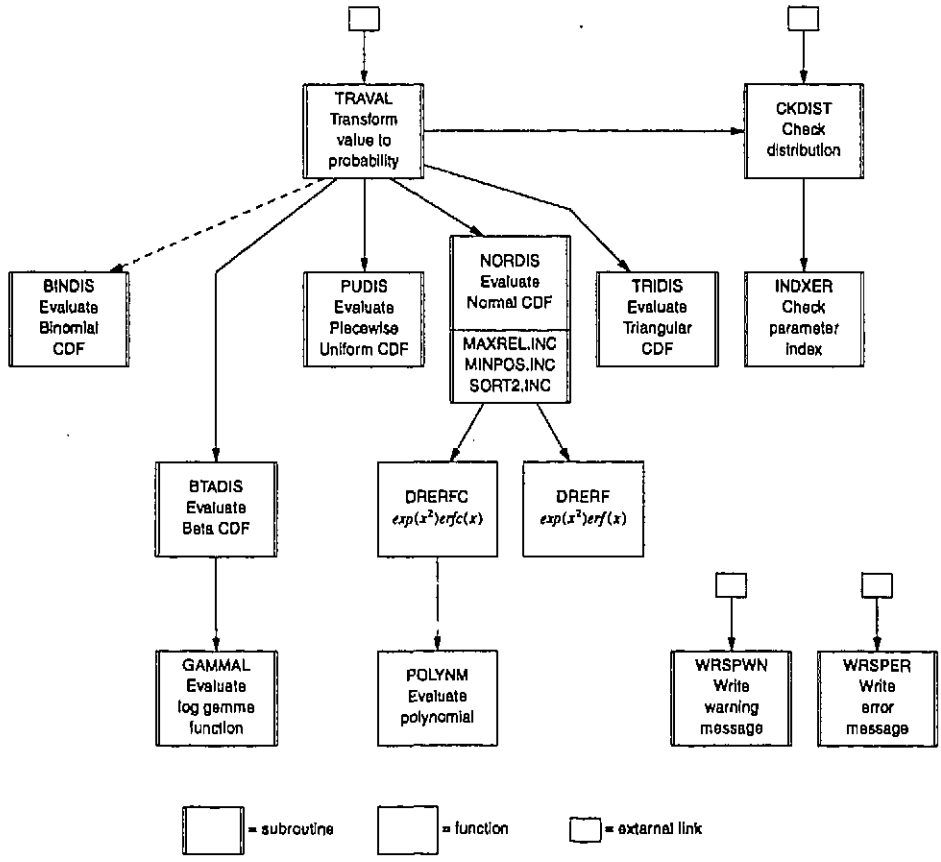

FIGURE 3.1/2: Structure Chart for TRAVAL

3.2 How SYVAC3 Calls the Parameter Sampling Package

SYVAC3 calls subroutines in the Parameter Sampling Package (PSP) from only three places: the INPDIS, SKIP, and ASSVAL subroutines. The PSP routines call two SYVAC3 routines, RETMES and PEXP. PSP routines use three include files from the rest of SYVAC3: MAXREL.INC and MINPOS.INC from the Time Series Package (TSP) and MXPAR.INC from the general SYVAC3 Package (SVP). These are the only interfaces between this package and the rest of SYVAC3.

Figure 3.2/1 shows how the PSP fits into SYVAC3. The top of the hierarchy shows three SYVAC3 routines that call routines in the PSP. The bottom two routines are SYVAC3 routines that are called by the PSP.

The SYVAC3 subroutine INPDIS reads Parameter Distributions from an input file. It calls TRAVAL and TRAQUA in the Parameter Sampling Package to transform truncation limits either from Variable Values to cumulative probabilities or vice versa. The user can specify bounds in either form. Both forms are printed in an output file. By calling these routines, INPDIS also checks the newly read distribution, since both TRAVAL and TRAQUA call CKDIST. The only distribution type not checked by CKDIST is the constant distribution, which has no truncation limits and no error conditions to check.

SYVAC3 calls the subroutine SKIP to pass over some simulations not needed for a SYVAC3 Case. It would call SKIP for simulations 1 to 49, for example, if the simulations in a SYVAC3 case started at simulation 50. SYVAC3 would also call SKIP if a SYVAC3 Case specified simulations in two or more ranges, such as 1 to 50, and 101 to 150. SKIP calls the Pseudorandom Generator, SUPRAN, enough times to update the random seed to the value it should have at the beginning of the next simulation to be performed. Since SUPRAN is a short routine with no loops, these calls do not take much time compared to the time it would take to rerun all the simulations being skipped. The ability to perform exactly the right number of calls to SUPRAN depends on the fact that SYVAC3 uses exactly one random seed in each simulation to generate a value for each parameter.

The SYVAC3 subroutine ASSVAL samples parameter values. It calls different routines in the sampling package, depending on which sampling method is being used:

(1) Random sampling. For each parameter, ASSVAL calls SUPRAN to generate a uniform variate between zero and one. This value is treated as a cumulative probability. ASSVAL calls TRAQUA to transform the cumulative probability to a parameter value.

(2) Quantile sampling. For each parameter, ASSVAL reads a number between zero and one from an input file, and treats it as a cumulative probability. ASSVAL calls TRAQUA to transform the cumulative probability to the corresponding parameter value.

(3) "On file" sampling. For each parameter, ASSVAL reads a parameter value from an input file. It calls TRAVAL to determine the corresponding cumulative probability.

The cumulative probability is checked against the quantile parameter bounds, and if it lies outside the bounds, WRSPWN prints a warning message to an output file.

The include files MAXREL.INC, MINPOS.INC, and MXPAR.INC all define constants that are used in the PSP. MAXREL is the largest representable floating-point number. MINPOS is the smallest positive floating-point number. These two constants depend on the floating-point hardware of the computer on which the code runs. MXPAR sets a limit on the number of parameters in a SYVAC3 simulation.

The SYVAC3 subroutines RETMES and PEXP provide needed functions to the PSP. RETMES retrieves a list of message files where messages should be written. PEXP evaluates a protected exponential function that gracefully handles numeric underflow conditions.

Two of the files that logically belong to the PSP reside in the SVP in SV309. These are BINDIS and INVBIN. They will be moved to the PSP in a later version of the code that fully installs the Binomial Distribution.

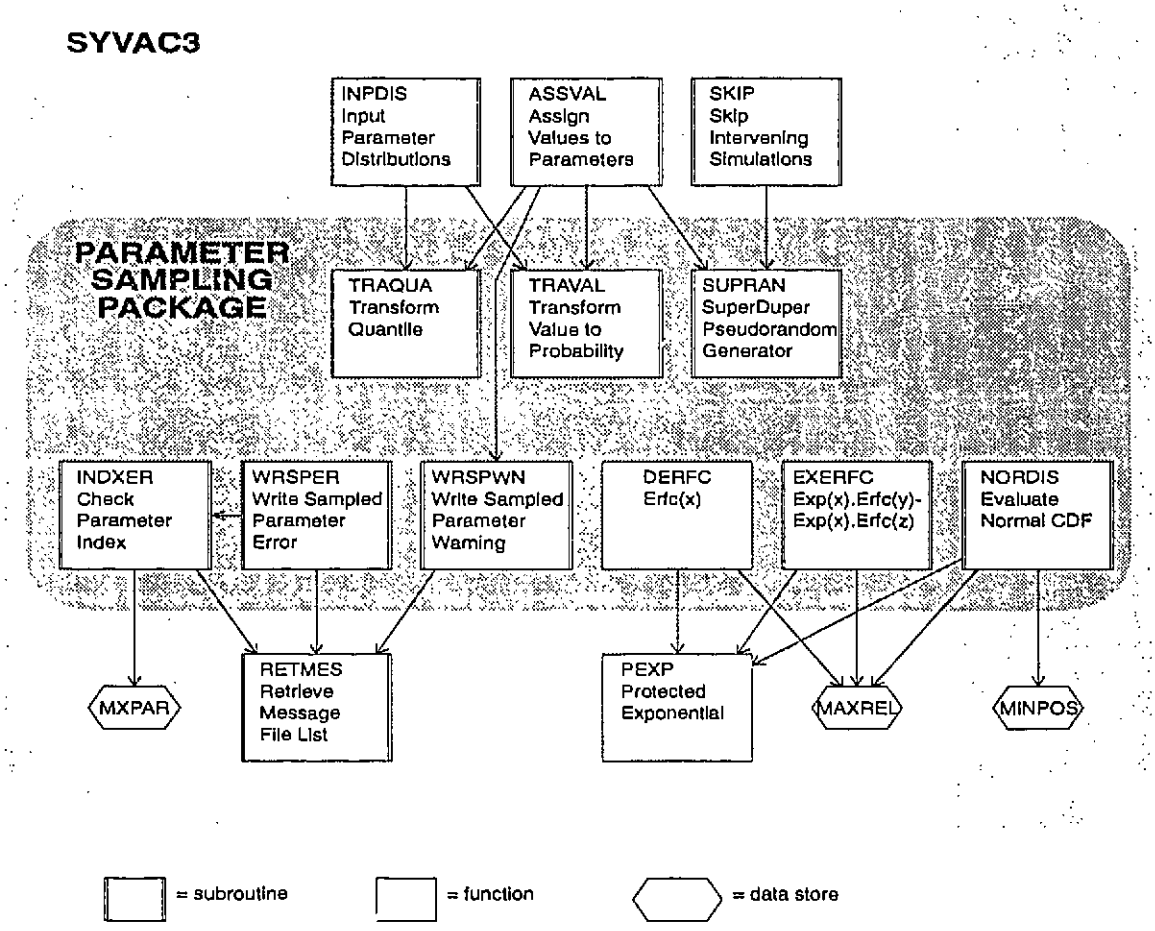

FIGURE 3.2/1: Links Between the PSP and the Rest of SYVAC3

3.3 How Any Program Can Call the Parameter Sampling Package

Any other program can call routines in the Parameter Sampling Package (PSP) in much the same way that SYVAC3 does. Code from that program would call specific routines in the PSP. The most common targets of those calls are likely to be TRAQUA, TRAVAL, CKDIST, and SUPRAN, although other routines can also be called. In turn, the PSP routines call two routines that the calling program must provide, RETMES and PEXP. PSP routines must have access to three include files from the calling program: MAXREL.INC, MINPOS.INC, and MXPAR.INC. These are the only interfaces needed between the PSP and the calling program.

Figure 3.3/1 shows how the PSP fits into another program other than SYVAC3. The available calls to routines in the PSP are at the top of the hierarchy. Routines that include files needed by the PSP are at the bottom of the hierarchy.

SYVAC3 appears in Figure 3.3/1 because the PSP is not entirely independent of the rest of SYVAC3. The two routines BINDIS and INVBIN, which evaluate and invert the CDF for a Binomial Distribution, should be in the PSP. In SYVAC3 Version 3.09 (SV309), they reside in the general SYVAC3 package (SVP). The mathematical routine PEXP can also be found in SVP, and that is the most convenient place to get a copy to use with the PSP. The include files MAXREL.INC and MINPOS.INC can be borrowed from the Time Series Package (TSP) of SYVAC3.

In contrast, the routine RETMES and the include file MXPAR.INC are likely to be adapted to each calling program. RETMES retrieves a list of message files on which to write error and warning messages. That list will depend on the error handling approach of the calling program. MXPAR puts a limit on the number of Parameter Distributions, and that limit is specific to the calling program.

The three include files MAXREL.INC, MINPOS.INC and MXPAR.INC each define a single constant in a Fortran PARAMETER statement. These constants are:

MAXREL.INC:	MAXREL =	an estimate of the maximum double-precision value that can be represented in the floating-point system being used.
MINPOS.INC:	MINPOS =	an estimate of the minimum positive double-precision value that can be represented in the floating-point system being used, usually close to 1/MAXREL.
MXPAR.INC:	MXPAR =	the maximum permitted value of a Parameter Distribution index.

Not all routines in the PSP appear in Figure 3.3/1. The missing routines would clutter the diagram without adding significantly to the links shown. In principle, all routines in the package are visible externally and, in practice, all but a few have some possible use by a calling program. Programming interfaces for all routines in the PSP can be found in Sections 3.7 to 3.10.

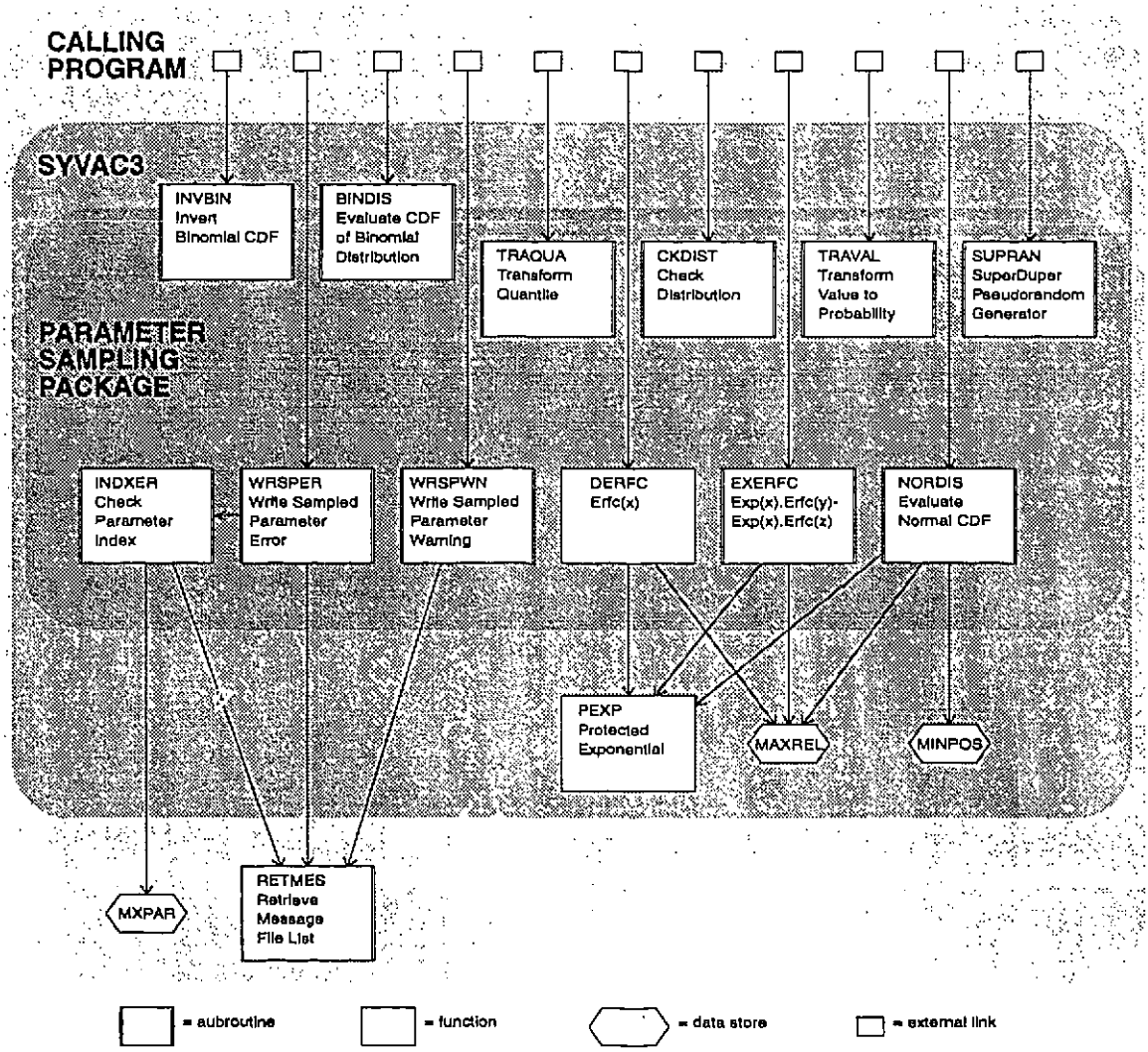

FIGURE 3.3/1: Links Between the PSP and a Calling Program

3.4 Data Structure for Parameter Distributions

When calling TRAQUA, TRAVAL, or CKDIST, the calling routine must pass a Parameter Distribution object down in the argument list. The data structure that defines a Parameter Distribution object is shown in Figure 3.4/1. In addition, there are some special fields for correlated distributions and for the Piecewise Uniform Distribution. Since the Parameter Sampling Package (PSP) routines were programmed in Fortran 77, it is not possible just to define a new data type with these fields. Instead, a group of Parameter Distribution objects is stored in a set of arrays, with an entry or a column in each array for each object. A program can retrieve an individual Parameter Distribution object by indexing into these arrays.

A routine must initialize the seven arrays in Table 3.4/1 before calling TRAQUA, TRAVAL, or CKDIST. These arrays constitute the data structure to store a set of Parameter Distribution objects. They are one-dimensional arrays, except for the two with entries in the column "Number of Rows," which are two-dimensional arrays. An "*" in the "Number of Columns" column indicates that the variables passed down have an "assumed size," to use Fortran jargon. That is, any variable with the right rank (i.e., the right number of subscripts) and the right first dimension (if there are two) can be passed down. A constraint on all the variables except PUDPAR is that the dimensions represented by "*" be no more than the value of MXPAR, available from the include file MXPAR.INC.

TABLE 3.4/1

STORAGE STRUCTURE FOR A SET OF PARAMETER DISTRIBUTIONS

Name of Array	*Number of Rows*	*Number of Columns*	*Fortran Data Type*	*Description*
PSNAME		*	Character*(*)	Name of variable in Fortran code
DSTTYP		*	Character*(*)	Name of distribution type
DSTPAR	4	*	Double Precision	Values of distribution attributes
IDXCOR		*	Integer	Index of correlated parameter
LOBNDD		*	Double Precision	Lower truncation limit for probabilities
HIBNDD		*	Double Precision	Upper truncation limit for probabilities
PUDPAR	3	*	Double Precision	Attributes of a Piecewise Uniform Distribution

The Parameter Distribution data structure consists of a name, a distribution type, a set of up to four numerical attributes (depending on distribution type), and a set of lower and upper truncation limits. These are shown in Figure 3.4/1, where each column represents a single variable. For example, column two describes a parameter *PRECIP* associated with a Normal Distribution having mean 78.0 and standard deviation 11.0, and quantile bounds 0.0001 and 1.

The numbers in the DSTPAR array mean different things for different distributions. Table 3.5/4 defines their use. The arrows in Figure 3.4/1 show the effects of pointers. The first attribute of a Piecewise Uniform Distribution is a pointer into the PUDPAR array–it points to the position just before the first location related to that distribution. The IDXCOR array points to a correlated parameter for each Correlated Normal or Correlated Lognormal Distribution.

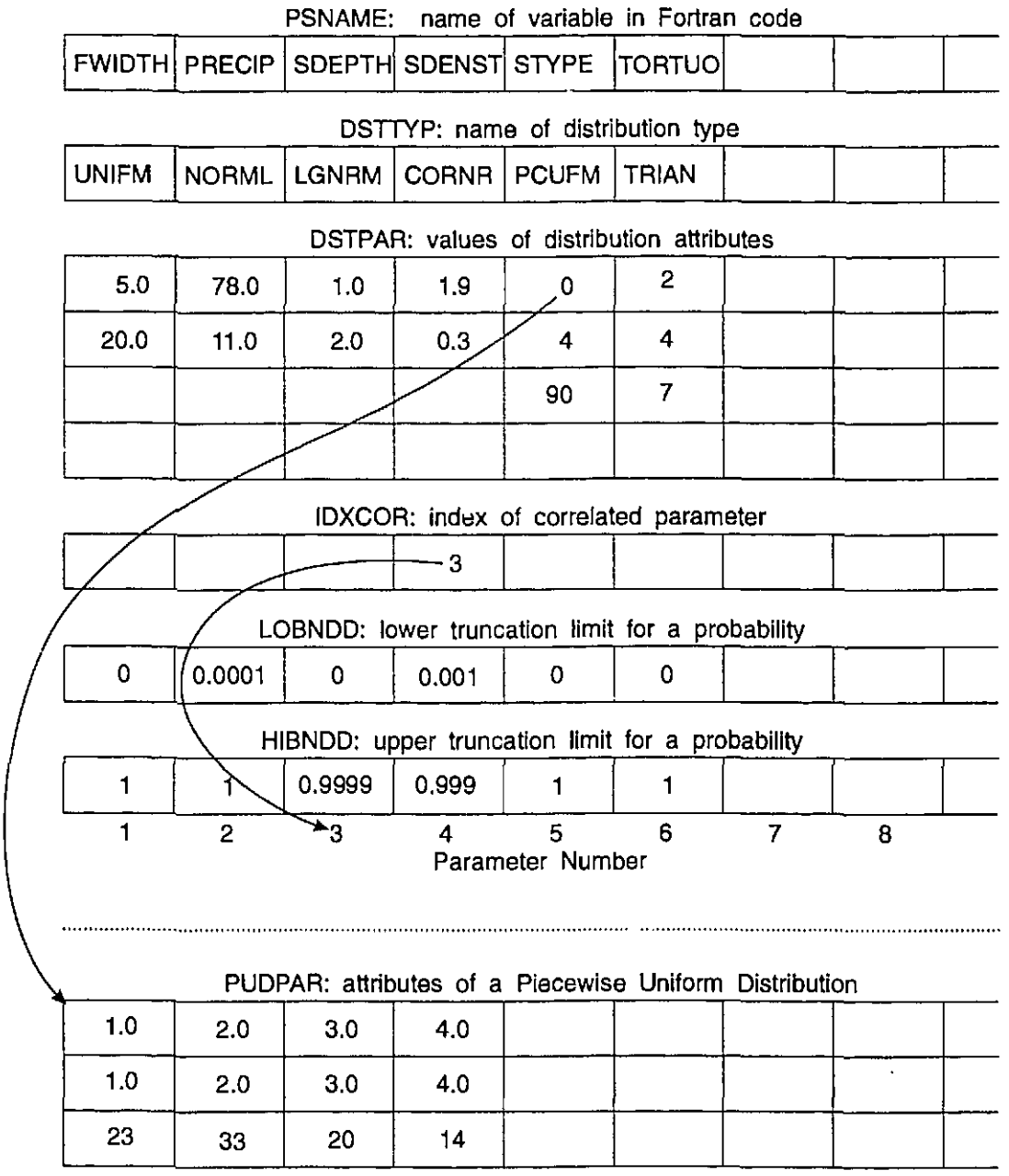

FIGURE 3.4/1: Storage Structure for a Set of Parameter Distributions

3.5 Contents of the Parameter Distribution Data Structure

The data arrays shown on the previous page contain data about Parameter Distributions in a coded format. The tables shown here define the possible entries in these arrays acceptable to the Parameter Sampling Package (PSP).

TABLE 3.5/1
VALID ENTRIES FOR PSNAME

Keep the declared length of character strings in PSNAME to no more than 80 characters.

Each variable name can be any character string, since the PSP uses this data only in formatting warning and error messages.

TABLE 3.5/2
VALID ENTRIES FOR IDXCOR

Rules for IDXCOR[i], $0 < i \leq$ length(IDXCOR):

If (DSTTYP[i] = 'CORLN' or DSTTYP[i] = 'CORNR') then
IDXCOR[i] = j, where:
1) $0 < j < i$, and
2) DSTTYP[j] = 'LGNRM' or DSTTYP[j] = 'NORML'
Else
IDXCOR[i] is undefined
End if

TABLE 3.5/3
VALID ENTRIES FOR PUDPAR

If (DSTTYP[i] = 'PCUFM', for $0 < i \leq$ length(DSTTYP)) then
1) Parameter i has a distribution with attributes $a1$, $a2$, and $a3$ that is formed as a mixture of $a2$ Uniform Distributions.
2) Columns $a1$+1 to $a1$+$a2$ of PUDPAR contain data on these Uniform Distributions.
3) For column $a1+j$ such that $1 \leq j \leq a2$,
   a) Ordered endpoints: PUDPAR[1,$a1+j$] ≤ PUDPAR[2,$a1+j$], where these two values define the endpoints of the j'th Uniform Distribution.
   b) Positive weights: 0 < PUDPAR[3,$a1+j$], which is the relative weight assigned to the j'th Uniform Distribution.
   c) Nonoverlapping intervals: If ($1 < j \leq a2$) then:
      PUDPAR[2,$a1+j-1$] ≤ PUDPAR[1,$a1+j$]
4) $$a3 = \sum_{j=1}^{a2} \text{PUDPAR}[3,a1+j]$$

Else
PUDPAR is not used by Parameter Distribution i
End if

TABLE 3.5/4
VALID ENTRIES FOR DSTTYP AND DSTPAR

Distribution	*DSTTYP*	*DSTPAR*	*Checks*
Beta	BETA	(*a1*) left range end	$a1 \le a2$
		(*a2*) right range end	$a1 \le a2$
		(*a3*) exponent *a*	$0 < a3$
		(*a4*) exponent *b*	$0 < a4$
Binomial*	BINOM	(*a1*) number of Bernoulli trials	$0 \le a1$
		(*a2*) probability of success	$0 \le a2 \le 1$
Constant	CONST	(*a1*) constant value	–
Correlated Lognormal**	CORLN	(*a1*) geometric mean	$0 < a1$
		(*a2*) geometric standard deviation	$1 < a2$
		(*a3*) correlation coefficient	$-1 \le a3 \le 1$
Correlated Normal**	CORNR	(*a1*) mean	–
		(*a2*) standard deviation	$0 < a2$
		(*a3*) correlation coefficient	$-1 \le a3 \le 1$
Lognormal	LGNRM	(*a1*) geometric mean	$0 < a1$
		(*a2*) geometric standard deviation	$1 < a2$
Loguniform	LGUFM	(*a1*) left range end	$0 < a1$
		(*a2*) right range end	$a1 < a2$
Normal	NORML	(*a1*) mean	–
		(*a2*) standard deviation	$0 < a2$
Piecewise Uniform***	PCUFM	(*a1*) pointer to PUDPAR	$0 \le a1$
		(*a2*) number of ranges	$0 < a2$
		(*a3*) total weight	$0 < a3$
Triangular	TRIAN	(*a1*) left range end	–
		(*a2*) mode	$a1 \le a2 \le a3$
		(*a3*) right range end	$a1 < a3$
Uniform	UNIFM	(*a1*) left range end	–
		(*a2*) right range end	$a1 < a2$

*Not yet implemented in SV309. **Uses data array IDXCOR. ***Uses data array PUDPAR.

TABLE 3.5/5
VALID ENTRIES FOR LOBNDD AND HIBNDD

Rules for LOBNDD[*i*] and HIBNDD[*i*], for $0 < i \le$ length(LOBNDD) = length(HIBNDD):

If (DSTTYP[*i*] = 'CONST') then
    LOBNDD[*i*] and HIBNDD[*i*] are undefined
Else
    $0 \le$ LOBNDD[*i*] $\le$ HIBNDD[*i*] ≤ 1
End if

3.6 Setting Up a RETMES Routine

A RETMES routine is required for the error checking in CKDIST, TRAVAL and TRAQUA to work. RETMES allows the main program to customize the error handling of the Parameter Sampling Package (PSP). RETMES returns one or two output unit numbers. Each unit number designates an output file where error and warning messages are to be written. RETMES also returns a character string to be written at the beginning of the error or warning message.

The subroutines TRAVAL, TRAQUA, and CKDIST perform extensive error checking on incoming data. TRAVAL and TRAQUA each call CKDIST, and each do additional checking. CKDIST checks the validity of the parameter index INDX, which identifies the distribution of interest. It then checks the indicated distribution for internal consistency. For example, a normal distribution is checked to ensure that the standard deviation is positive, and a uniform distribution is checked to ensure that the range ends are in the correct order (lower value, then higher value). In addition to these checks, TRAVAL tests whether the parameter value being transformed is outside the range of the distribution, and prints a warning if it is. TRAQUA tests that the cumulative probability being transformed is outside the range zero to one, and prints an error message if it is.

To print warning and error messages correctly, the calling program must provide a RETMES subroutine (see Table 3.6/1). The code for the RETMES subroutine on the following page is that used in SYVAC3. It uses a MESSGE.INC include file where values of output file unit numbers, IUMESG(1:NIUMSG), have been stored by code not shown here. The number of the current run (simulation) is in a variable called RNMSG, also from the MESSGE include file. SYVAC3 starts its value at zero and increases it by one at the beginning of each run.

Inspection of the code shows that this version of RETMES can handle two specific values for MSGTYP, namely "ERROR" and "WARNING." Other values are treated as messages that are not errors, and a suitable prefix containing just the run number is placed in BEGIN. The value of BEGIN is not entirely fixed in this version of RETMES, but depends on the state of the program, as well as on MSGTYP. For example, if RNMSG has the value zero, BEGIN will return the character string "**ERROR**" if MSGTYP is "ERROR." Once the simulations have started, and RNMSG is greater than 0, the value that BEGIN returns changes to "**ERROR IN RUN # nnn **" .

TABLE 3.6/1
RETRIEVE ERROR MESSAGE INFORMATION

SUBROUTINE **RETMES**(MSGTYP,BEGIN,NMSGFL,IUMESG)					**RETMES**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
MSGTYP	character*(*)	scalar	in	type of message to write	WARNING, ERROR, other
BEGIN	character*(*)	scalar	out	prefix to message	keep it short
NMSGFL	integer	scalar	out	number of files to receive message	in {0, 1, 2}
IUMESG	integer	2	out	unit numbers of files	implementation-dependent

TABLE 3.6/2

EXAMPLE OF A RETMES ROUTINE

```
      SUBROUTINE RETMES(MSGTYP,BEGIN,NMSGFL,IUMESG)
C
C************************************************************** RETMES
C
C Retrieve error message information - the beginning of the message
C and the unit numbersof the message files.
C
C  87-FEB-08  VERSION 01B  A. SKEET
C
C*****************************************************************
C
      IMPLICIT NONE
C
      INCLUDE 'MESSGE.INC'
C
      CHARACTER
     1 BEGIN*(*),              ! beginning of an error message, in
                               ! standard format []
     1 CHVAL*5,                ! character representation of a value []
     1 MSGTYP*(*)              ! message type []
C
      INTEGER
     1 IUMESG(2),              ! Fortran unit numbers of message files
                               ! []
     1 J,                      ! general index []
     1 NMSGFL                  ! number of message files
C
      IF (MSGTYP .EQ. 'ERROR' ) THEN
C.....create the beginning of the error message.
         IF (RNMSG .GT. 0) THEN
            WRITE(CHVAL,10)RNMSG
10          FORMAT(I5)
            BEGIN = '**ERROR IN RUN # ' // CHVAL // ' **'
         ELSE
            BEGIN = '**ERROR**'
         END IF
      ELSE IF (MSGTYP .EQ. 'WARNING') THEN
C.....create the beginning of a warning message
         IF (RNMSG .GT. 0) THEN
            WRITE(CHVAL,10)RNMSG
            BEGIN = '--WARNING IN RUN # ' // CHVAL // ' --'
         ELSE
            BEGIN = '--WARNING--'
         END IF
      ELSE
C.....create the beginning of a message that is not an error
C.....or warning
         IF (RNMSG .GT. 0) THEN
            WRITE(CHVAL,10)RNMSG
            BEGIN = 'RUN # ' // CHVAL
         ELSE
            BEGIN = ' '
         END IF
      END IF
C.....set message file unit numbers.
      NMSGFL = NIUMSG
      DO 20 J = 1,NIUMSG
         IUMESG(J) = IUMSG(J)
20    CONTINUE
      RETURN
      END
```

3.7 Argument Lists for Parameter Sampling Routines, BINDIS–EXERFC

The boxes in this section show the arguments in the order needed to call each routine in the Parameter Sampling Package (PSP). The arguments that define Parameter Distributions (e.g., in CKDIST) were described in Section 3.4. The others are defined here briefly. For more detail, see the specifications for these routines in Chapters 8 to 12.

TABLE 3.7/1
BINOMIAL DISTRIBUTION CDF
(from the SYVAC3 general code (SV) package, not the Parameter Sampling Package)

SUBROUTINE **BINDIS**(VARVAL, NTRIAL, SPROB, CPROB)					**BINDIS**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
VARVAL	double precision	scalar	in	value of binomial variate	integer in [0,NTRIAL]
NTRIAL	double precision	scalar	in	number of Bernoulli trials	integer > 0
SPROB	double precision	scalar	in	probability of success	in [0,1]
CPROB	double precision	scalar	out	probability $X \leq$ VARVAL	in [0,1]

TABLE 3.7/2
STANDARD BETA DISTRIBUTION CDF (INCOMPLETE BETA RATIO FUNCTION)

SUBROUTINE **BTADIS**(SHPBT1, SHPBT2, STDVAL, BTAPRB)					**BTADIS**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
SHPBT1	double precision	scalar	in	first shape attribute	> 0
SHPBT2	double precision	scalar	in	second shape attribute	> 0
STDVAL	double precision	scalar	in	standardized value	in [0,1]
BTAPRB	double precision	scalar	out	beta CDF F(STDVAL)	in [0,1]

TABLE 3.7/3
CHECK PARAMETER DISTRIBUTION

SUBROUTINE **CKDIST**(INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, DSTOK)					**CKDIST**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
INDX	integer	scalar	in	index of Parameter Distribution	> 0, valid index
PSNAME	character*(*)	*	in	name of variable in code	see Table 3.5/1
DSTTYP	character*(*)	*	in	name of distribution type	see Table 3.5/4
DSTPAR	double precision	4,*	in	values of distribution attributes	see Table 3.5/4
IDXCOR	integer	*	in	index of correlated parameter	see Table 3.5/2
PUDPAR	double precision	3,*	in	attributes of Uniform Distributions	see Table 3.5/3
LOBNDD	double precision	*	in	lower truncation limit	see Table 3.5/5
HIBNDD	double precision	*	in	upper truncation limit	see Table 3.5/5
DSTOK	logical	scalar	out	input distribution is OK	in [TRUE, FALSE]

TABLE 3.7/4
ERROR FUNCTION erf(x)

DOUBLE PRECISION FUNCTION **DERF**(XND)					**DERF**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any value	–
DERF	double precision	scalar	out	erf(XND)	in [-1,1]

TABLE 3.7/5
COMPLEMENTARY ERROR FUNCTION erfc(x)

DOUBLE PRECISION FUNCTION **DERFC**(XND)					**DERFC**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any value	–
DERFC	double precision	scalar	out	erfc(XND)	in [0,2]

TABLE 3.7/6
EVALUATE $\exp(x^2)\mathrm{erf}(x)$ OVER A RESTRICTED DOMAIN

DOUBLE PRECISION FUNCTION **DRERF**(XND)					**DRERF**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any value in domain	in [0,0.5]
DRERF	double precision	scalar	out	$\exp(\mathrm{XND}^2)\mathrm{erf}(\mathrm{XND})$	in [0,1]

TABLE 3.7/7
EVALUATE $\exp(x^2)\mathrm{erfc}(x)$ FOR POSITIVE x

DOUBLE PRECISION FUNCTION **DRERFC**(XND)					**DRERFC**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any positive value	> 0
DRERFC	double precision	scalar	out	$\exp(\mathrm{XND}^2)\mathrm{erf}(\mathrm{XND})$	in [0,1]

TABLE 3.7/8
EVALUATE $\exp(x)[\mathrm{erfc}(y) - \mathrm{erfc}(z)]$

SUBROUTINE **EXERFC**(XND, YND, ZND, RESULT)					**EXERFC**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any value	no result overflow
YND	double precision	scalar	in	any value	–
ZND	double precision	scalar	in	any value	–
RESULT	double precision	scalar	out	$\exp(\mathrm{XND}^2)$[erfc(YND)-erfc(ZND)]	–

3.8 Argument Lists for Parameter Sampling Routines, GAMMAL–INVTRI

This section continues the list of subroutines and functions used in the Parameter Sampling Package (PSP). The boxes show the arguments in order needed to call each routine. For more detail, see the specifications for these routines in Chapters 8 to 12.

TABLE 3.8/1
LOG GAMMA FUNCTION [ln Γ(x)]

SUBROUTINE **GAMMAL**(X, LOGGAM)					**GAMMAL**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
X	double precision	scalar	in	any positive value	> 0, no result overflow
LOGGAM	double precision	scalar	out	ln Γ(X)	> -0.13

TABLE 3.8/2
CHECK PARAMETER INDEX FOR ERROR

SUBROUTINE **INDXER**(MODULE, INDX, INDXOK)					**INDXER**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
MODULE	character*(*)	scalar	in	name of calling module	correct name
INDX	integer	scalar	in	index of a parameter	in [1,MXPAR]
INDXOK	logical	scalar	out	index lies in the acceptable range	TRUE or FALSE

TABLE 3.8/3
INVERT BINOMIAL CDF
(from the SYVAC3 general code (SV) package, not the Parameter Sampling Package)

SUBROUTINE **INVBIN**(QUANTL, NTRIAL, SPROB, BINVLU)					**INVBIN**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
NTRIAL	double precision	scalar	in	number of Bernoulli trials	integer > 0
SPROB	double precision	scalar	in	probability of success	in [0,1]
BINVLU	double precision	scalar	out	least n where CDF $F(n) \geq$ QUANTL	integer in [0,NTRIAL]

TABLE 3.8/4
INVERT STANDARD BETA CDF

SUBROUTINE **INVBTA**(QUANTL, SHPBT1, SHPBT2, BTAVLU)					**INVBTA**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
SHPBT1	double precision	scalar	in	first shape attribute	> 0
SHPBT2	double precision	scalar	in	second shape attribute	> 0
BTAVLU	double precision	scalar	out	inverse beta CDF F^{-1}(QUANTL)	in [0,1]

TABLE 3.8/5
INVERT CDF OF CORRELATED NORMAL

SUBROUTINE INVCOR(QUANTL, MEAN, STDDEV, LOBND, HIBND, CORCOF, CORMU, CORSIG, CORVAL, CORVLU)					**INVCOR**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
MEAN	double precision	scalar	in	mean of CORNR variate	–
STDDEV	double precision	scalar	in	standard deviation of CORNR variate	> 0
LOBND	double precision	scalar	in	lower probability bound of CORNR variate	in [0,1]
HIBND	double precision	scalar	in	upper probability bound of CORNR variate	in [0,1]
CORCOF	double precision	scalar	in	correlation coefficient	in [-1,1]
CORMU	double precision	scalar	in	mean of correlated variate	–
CORSIG	double precision	scalar	in	standard deviation of correlated variate	> 0
CORVAL	double precision	scalar	in	value of correlated variable	–
CORVLU	double precision	scalar	out	inverse conditional CDF F^{-1}(QUANTL|CORVAL)	–

TABLE 3.8/6
INVERT STANDARD NORMAL CDF

SUBROUTINE **INVNOR**(QUANLT, NORVLU)					**INVNOR**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANLT	double precision	scalar	in	cumulative probability	in [0,1]
NORVLU	double precision	scalar	out	inverse normal CDF F^{-1}(QUANLT)	–

TABLE 3.8/7
INVERT PIECEWISE UNIFORM CDF

SUBROUTINE **INVPU**(QUANTL, PUPTRS, NUMCLS, TOTWGT, PUDPAR, PUVLU)					**INVPU**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
PUPTRS	integer	scalar	in	pointer to attributes in PUDPAR	see Table 3.5/4
NUMCLS	integer	scalar	in	number of Uniform Distributions	see Table 3.5/4
TOTWGT	double precision	scalar	in	total of weights for each distribution	> 0
PUDPAR	double precision	3, *	in	attributes of Uniform Distributions	see Table 3.5/3
BINVLU	double precision	scalar	out	inverse Piecewise Uniform CDF F^{-1}(QUANTL)	inside a bin

TABLE 3.8/8
INVERT STANDARD TRIANGULAR CDF

SUBROUTINE INVTRI(QUANTL, MODE, TRIVLU)					**INVTRI**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
MODE	double precision	scalar	in	location of mode	in [0,1]
TRIVLU	double precision	scalar	out	inverse triangular CDF F^{-1}(QUANTL)	in [0,1]

3.9 Argument Lists for Parameter Sampling Routines, NORDIS–TRAVAL

This section continues the list of subroutines and functions used in the Parameter Sampling Package (PSP). The boxes show the arguments in the order needed to call each routine. For more detail, see the specifications for these routines in Chapters 8 to 12.

TABLE 3.9/1
EVALUATE STANDARD NORMAL CDF

SUBROUTINE NORDIS(XND, CDF)					NORDIS
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
XND	double precision	scalar	in	any value	–
CDF	double precision	scalar	out	normal CDF F(XND)	in [0,1]

TABLE 3.9/2
EVALUATE PROTECTED EXPONENTIAL FUNCTION
(from the SYVAC3 general code (SV) package, not in the Parameter Sampling Package)

DOUBLE PRECISION FUNCTION **PEXP**(X)					PEXP
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
X	double precision	scalar	in	any value	< ln MAXREAL
PEXP	double precision	scalar	out	exp(XND), or 0 to prevent underflow	≥ 0

TABLE 3.9/3
EVALUATE POLYNOMIAL

DOUBLE PRECISION FUNCTION **POLYNM**(DEGREE,COEF,XND)					POLYNM
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
DEGREE	integer	scalar	in	degree of polynomial	≥ 0
COEF	double precision	0:DEGREE	in	polynomial coefficients	no overflow
XND	double precision	scalar	in	argument of polynomial	no overflow
POLYNM	double precision	scalar	out	polynomial p(XND)	–

TABLE 3.9/4
EVALUATE PIECEWISE UNIFORM CDF

SUBROUTINE **PUDIS**(STDVAL, PUPTRS, NUMCLS, TOTWGT, PUDPAR, PUPRB)					PUDIS
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
STDVAL	double precision	scalar	in	any value	–
PUPTRS	integer	scalar	in	pointer to attributes in PUDPAR	see Table 3.5/4
NUMCLS	integer	scalar	in	number of Uniform Distributions	see Table 3.5/4
TOTWGT	double precision	scalar	in	total of weights for each distribution	> 0
PUDPAR	double precision	3, *	in	attributes of Uniform Distributions	see Table 3.5/3
PUPRB	double precision	scalar	out	Piecewise Uniform CDF F(XND)	in [0,1]

TABLE 3.9/5
"SUPER-DUPER" PSEUDORANDOM NUMBER GENERATOR

SUBROUTINE **SUPRAN**(SEED, RNDNUM)					**SUPRAN**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
SEED	integer	scalar	both	random seed	–
RNDNUM	double precision	scalar	out	pseudorandom number	in [0,1]

TABLE 3.9/6
TRANSLATE QUANTILE FROM PROBABILITY TO VALUE

SUBROUTINE **TRAQUA**(INDX, QUANTL, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, PARVAL, TRANOK)					**TRAQUA**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
INDX	integer	scalar	in	index of Parameter Distribution	> 0, valid index
QUANTL	double precision	scalar	in	cumulative probability	in [0,1]
PSNAME	character*(*)	*	in	name of variable in code	see Table 3.5/1
DSTTYP	character*(*)	*	in	name of distribution type	see Table 34/4
DSTPAR	double precision	4, *	in	values of distribution attributes	see Table 3.5/4
IDXCOR	integer	*	in	index of correlated parameter	see Table 3.5/2
PUDPAR	double precision	3, *	in	attributes of Piecewise Uniform	see Table 3.5/3
LOBNDD	double precision	*	in	lower truncation limit	see Table 3.5/5
HIBNDD	double precision	*	in	upper truncation limit	see Table 3.5/5
PARVAL	double precision	scalar	out	inverse CDF F^{-1}(INDX; QUANTL)	in distribution range
TRANOK	logical	scalar	out	transformed OK	TRUE or FALSE

TABLE 3.9/7
TRANSLATE QUANTILE FROM VALUE TO PROBABILITY

SUBROUTINE **TRAVAL**(INDX, PARVAL, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, QUANTL, TRANOK)					**TRAVAL**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
INDX	integer	scalar	in	index of Parameter Distribution	> 0, valid index
PARVAL	double precision	scalar	in	parameter value	–
PSNAME	character*(*)	*	in	name of variable in code	see Table 3.5/1
DSTTYP	character*(*)	*	in	name of distribution type	see Table 34/4
DSTPAR	double precision	4, *	in	values of distribution attributes	see Table 3.5/4
IDXCOR	integer	*	in	index of correlated parameter	see Table 3.5/2
PUDPAR	double precision	3, *	in	attributes of Piecewise Uniform	see Table 3.5/3
LOBNDD	double precision	*	in	lower truncation limit	see Table 3.5/5
HIBNDD	double precision	*	in	upper truncation limit	see Table 3.5/5
QUANTL	double precision	scalar	out	CDF F(INDX; PARVAL)	in [0,1]
TRANOK	logical	scalar	out	transformed OK	TRUE or FALSE

3.10 Argument Lists for Parameter Sampling Routines, TRIDIS-WRSPWN

This section continues the list of subroutines and functions used in the Parameter Sampling Package (PSP). The boxes show the arguments in the order needed to call each routine. For more detail, see the specifications for these routines in Chapters 8 to 12.

TABLE 3.10/1
EVALUATE STANDARD TRIANGULAR CDF

SUBROUTINE **TRIDIS**(VARVAL, MODE, TRIPRB)					**TRIDIS**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
VARVAL	double precision	scalar	in	variable value	–
MODE	double precision	scalar	in	location of mode	in [0,1]
TRIPRB	double precision	scalar	in	triangular CDF *F*(VARVAL)	in [0,1]

TABLE 3.10/2
WRITE ERROR MESSAGE

SUBROUTINE **WRSPER**(MODULE, ERROR, INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, STOPP)					**WRSPER**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
MODULE	character*(*)	scalar	in	name of module with error	valid module
ERROR	character*(*)	scalar	in	error message	–
INDX	integer	scalar	in	index of Parameter Distribution	> 0, valid index
PARVAL	double precision	scalar	in	parameter value	–
PSNAME	character*(*)	*	in	name of variable in code	see Table 3.5/1
DSTTYP	character*(*)	*	in	name of distribution type	see Table 34/4
DSTPAR	double precision	4, *	in	values of distribution attributes	see Table 3.5/4
IDXCOR	integer	*	in	index of correlated parameter	see Table 3.5/2
PUDPAR	double precision	3, *	in	attributes of Piecewise Uniform	see Table 3.5/3
LOBNDD	double precision	*	in	lower truncation limit	see Table 3.5/5
HIBNDD	double precision	*	in	upper truncation limit	see Table 3.5/5
STOPP	logical	scalar	out	stop execution because of error	in [TRUE,FALSE]

TABLE 3.10/3
WRITE WARNING MESSAGE

SUBROUTINE **WRSPWN**(MODULE, WARNIN)					**WRSPWN**
Name	*Declared Type*	*Rank*	*In/Out*	*Definition*	*Constraints*
MODULE	character*(*)	scalar	in	name of module with error	valid module
WARNIN	character*(*)	scalar	in	warning message	–

4 ESSENTIAL REQUIREMENTS FOR PARAMETER DISTRIBUTIONS

4.1 Parameter Distribution and Associated Object Types

The Parameter Sampling Package (PSP) provides a Parameter Distribution object type so that software developers can create and use Parameter Distribution objects. An object of this type has two parts: a Probability Distribution object and a Truncation Interval object. The Parameter Distribution object type has one subtype, the Conditional Distribution, tied by a correlation operation to another independent Parameter Distribution. A Probability Distribution object has many subtypes, each corresponding to a different family of Probability Distributions. A Truncation Interval has two subtypes, corresponding to bounds based on parameter values and bounds based on probabilities respectively. Figure 4.1/1 illustrates these object types and their subtypes. Chapters 4 and 5 discuss the attributes and operations of these object types.

Section 2.1 introduced the concept of software objects and software object types. The PSP implements several software object types related to the basic Parameter Distribution object type. Figure 4.1/1 shows the associations among these types. It is an object diagram based on (but not identical to) the object diagram notation of Rumbaugh et al. (1991)[1].

A Parameter Distribution object represents the distribution of values that could be associated with a physical parameter of a model. Each such object consists of a Probability Distribution and a Truncation Interval. The Probability Distribution object represents a statistical probability distribution, such as a normal distribution or a uniform distribution. The Truncation Interval represents an allowed range of values in which the physical parameter must lie. Essentially a Parameter Distribution is a Probability Distribution truncated so that there is zero possibility of a value occurring outside the Truncation Interval.

The PSP also implements subtypes of these major object types. A Conditional Distribution is a subtype of the Parameter Distribution. It represents the distribution of values for a physical parameter when that parameter is known to be correlated with another parameter. A Conditional Distribution has a correlation association with an independent Parameter Distribution.

The Truncation Interval object type has two subtypes, Value Interval and Probability Interval. A Value Interval represents an allowed range of values for a parameter, expressed in the same

---

[1] The squares in the diagram represent software object types. The name of each type appears in the top part of the square. The bottom part of each square is to list attributes and operations of the object type, which are not needed here.

Three types of associations are indicated by lines joining the object types. First there are simple associations like *correlation* and *mixture of* that are shown by lines with optional arrowheads on each end. No arrowhead on the end of a line near an object type means that there is a one-to-one correspondence between objects of that type and instances of the association. A single arrowhead means that only one object is involved, and that the association is optional. A double arrowhead means that any number of such objects can be involved in the association. The second type of association is an aggregation association like *composed of*, indicated by a diamond shape, which shows the parts of an object type. In this case, each Parameter Distribution object is composed of one Truncation Interval object and one Probability Distribution object. The third type of association, with a bar across the connection, shows that an object type is a generalization of several other object types. For example, the *subtypes* association indicates that Truncation Interval and Probability Distribution are each generalizations of multiple subtypes.

units as the parameter (e.g., 30 cm to 50 cm). A Probability Interval also represents an allowed range of values, but these are expressed as quantiles of a distribution (e.g., from the 0.01 quantile to the 0.99 quantile of the untruncated distribution, where 0.01 and 0.99 are probabilities). An unconstrained parameter has the Probability Interval from the zero quantile to the one quantile.

The Probability Distribution object type has many subtypes (Figure 4.1/1), which are described in Chapter 5.

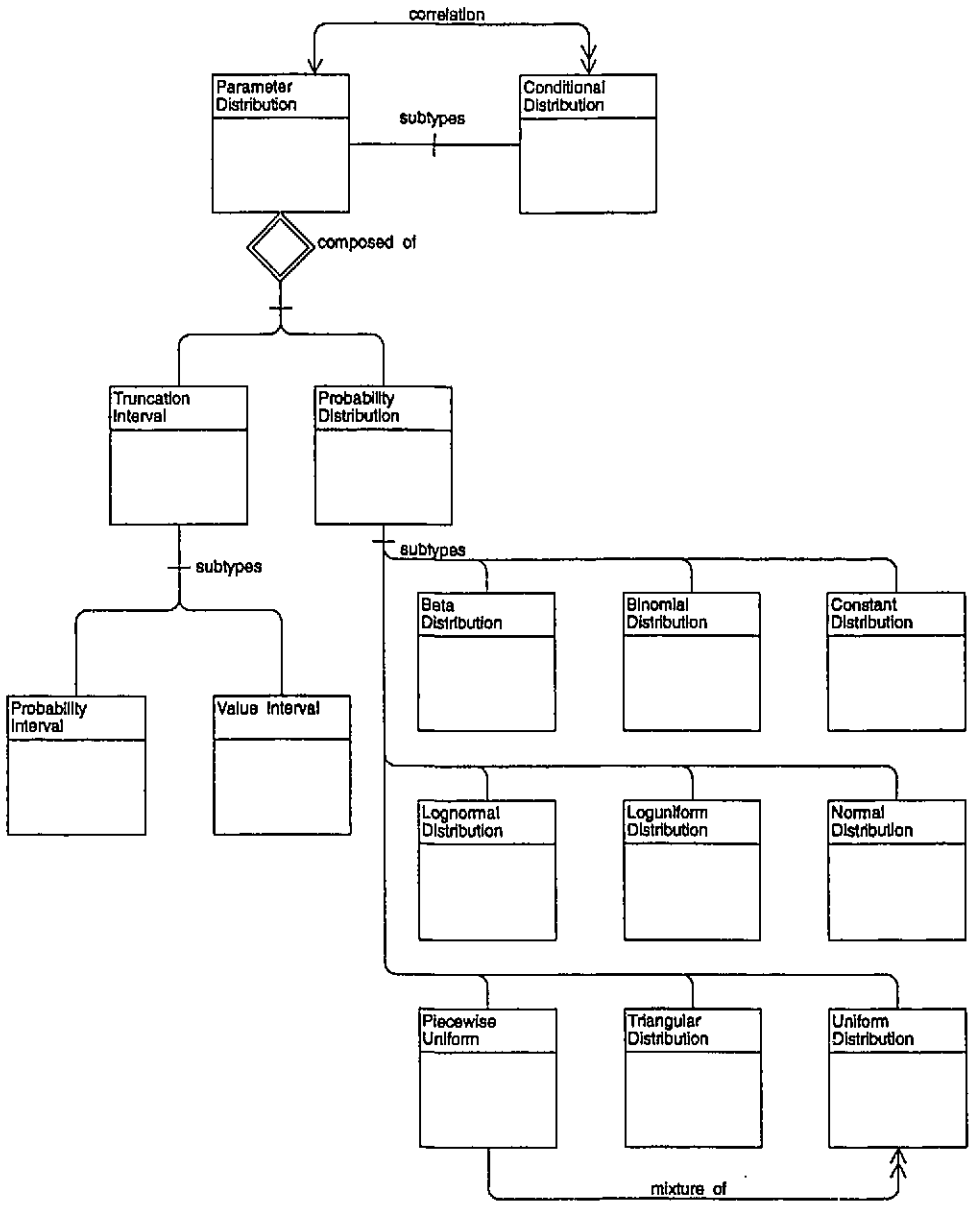

FIGURE 4.1/1: Parameter Distribution and Associated Object Types, Including Subtypes

4.2 Mathematica Notation for Specifying Objects

Diagrams like Figure 4.1/1 can show the associations between object types, but another technique is used to specify unambiguously the details of an object type. In this report, object specifications are written in a subset of the Mathematica language, a general-purpose programming language with strong mathematical features. This section summarizes the Mathematica conventions used in this document.

CRITERIA

Specifications for object types can be written in several different ways, such as with data-flow diagrams, structured English, pseudocode, or decision tables. The following criteria were used to select among the candidate methods for documenting the Parameter Sampling Package (PSP):

(1) Use a formal language that can be checked for correct syntax. Informal specifications for SYVAC3 models have had problems with high defect rates, since there was no effective way to check them.
(2) Use an executable language that can produce results. Having an "oracle" that can generate correct results simplifies testing of the final implementation.
(3) Use a language that can support object-oriented programming. The specifications are object-oriented, so they should be expressed in a language that supports inheritance, encapsulation and overloading.
(4) Use a language with strong support for mathematical functions. The PSP evaluates some classical nonlinear functions, and the specifications are easier to express if these functions can be called directly without further definition in the programming language.

CONVENTIONS

One language that met all these criteria was Mathematica (Wolfram 1991). It is an interactive symbolic manipulation language with arbitrary-precision arithmetic that is available on a variety of computers. Object attributes and operations are described here in Mathematica notation. The final implementation does not need to follow the same syntax as the Mathematica code, but it should produce the same results. The following conventions apply:

(1) Each object type is represented by an expression of the form "*object type name*[*attribute list*]." The *attribute list* contains not just attributes, but also component objects and associated objects. For example, Table 4.2/1 lists the representations for the major object types shown in Figure 4.1/1.
(2) A definition consists of an expression, followed by ":=" and the expansion to be used.
(3) A definition may introduce a variable to represent an expression by using the following syntax: "*name_object type name*". The three parts to this form are
    name: used to refer to the expression in the definition. It may be omitted if the expression is not referenced again.
    _: (underscore) represents a single expression; "__" (two underscores)

may be used instead to represent a non-empty sequence of expressions (e.g., definition of ProbDis in Table 4.2/1), and "___" (three underscores) represent a possibly empty sequence (e.g., last entry in `ParDis[...]` in Table 4.2/1).

object type name: identifies the type of object matched (e.g., `x_Real`, `j_Integer`, `q_ParDis`). Each object in a sequence must be of the same type. This part may be omitted if any type of object will do.

(4) A variable that must match a particular pattern has the syntax "*name*`:`*pattern*". For example, "`q:ParDis[d_ProbDis, t_TruncInt]`" means a variable `q` that is not only a Parameter Distribution, but one containing Probability Distribution object `d` and one Truncation Interval object `t`, and no other components.

(5) A variable that must satisfy a logical function (see the list in Table 4.2/1) has the syntax "*name*`?`*logical function*". For example, "`p_?NumberQ`" means any expression `p` such that `NumberQ[p]==True`.

(6) The numerical comparison operators for equality and relative size are "<", "<=", "==", ">=", ">", and "!=", where the last operator means "is not equal to."

(7) Each operation on an object type is represented by an expression of the form "*operation name*`[`*argument list*`]`". For example, the probability density function (PDF) of a Parameter Distribution object `q` at point `x` is represented by `PDF[q,x]`.

(8) Parts of an expression "*name*`[`*a1, a2, . . .* `]`" are denoted by double brackets, so that *a1* == *name*`[[1]]`, *a2* == *name*`[[2]]`, and so on.

TABLE 4.2/1
MATHEMATICA DEFINITIONS

Object Definitions

`ProbDis[st__]` a Probability Distribution with a subtype and attributes given by the non-empty sequence *st* (see Section 4.4)

`TruncInt[__]` a Truncation Interval with a subtype and attributes given by an unnamed non-empty sequence (see Section 4.5)

`ParDis[d_ProbDis, t_TruncInt, st___]`
a Parameter Distribution composed of Probability Distribution *d* and Truncation Interval *t*, with an optional sequence *st* to specify a subtype and its attributes.

`ConDis[d_ProbDis, t_TruncInt, c_?CorrCoefQ, q_ParDis] := ParDis[d, t, ConDis, c, q]`
a Conditional Distribution composed of Probability Distribution *d* and Truncation Interval *t*, along with a Correlation Coefficient *c*, and an independent Parameter Distribution *q*. The expression on the right shows a Conditional Distribution expressed as a subtype of Parameter Distribution.

Logical Tests of Numerical Values

`CorrCoefQ[x_]:= NumberQ[x] && (-1 <= x) && (x <= 1)`
test for a valid correlation coefficient (&& means "and")

`NumberQ[x_]` gives True if x is a number, and False otherwise

`ProbabilityQ[x_]:= NumberQ[x] && (0 <= x) && (x <= 1)`
test for a valid probability

4.3 Parameter Distribution Objects

A Parameter Distribution is essentially a Probability Distribution restricted to lie in a Truncation Interval, as shown in Figure 4.3/1. The composition and subtypes of the Parameter Distribution object type have already been discussed in Section 4.1, and they reappear here in Figure 4.3/2. The main requirement on the Parameter Distribution object type is to provide the operations CDF and quantile.

ASSOCIATIONS

Correlation–Each Conditional Distribution object (CCDF is the conditional CDF) is correlated with an independent Parameter Distribution object, in the sense that knowing a value chosen from the latter influences the distribution of the former. This association is strictly one-way, even though, statistically, values sampled from the two Parameter Distributions are mutually correlated. While a Conditional Distribution is associated with precisely one independent Parameter Distribution, each Parameter Distribution can be the target of 0, 1 or several correlations. Section 4.5 discusses the correlation association.

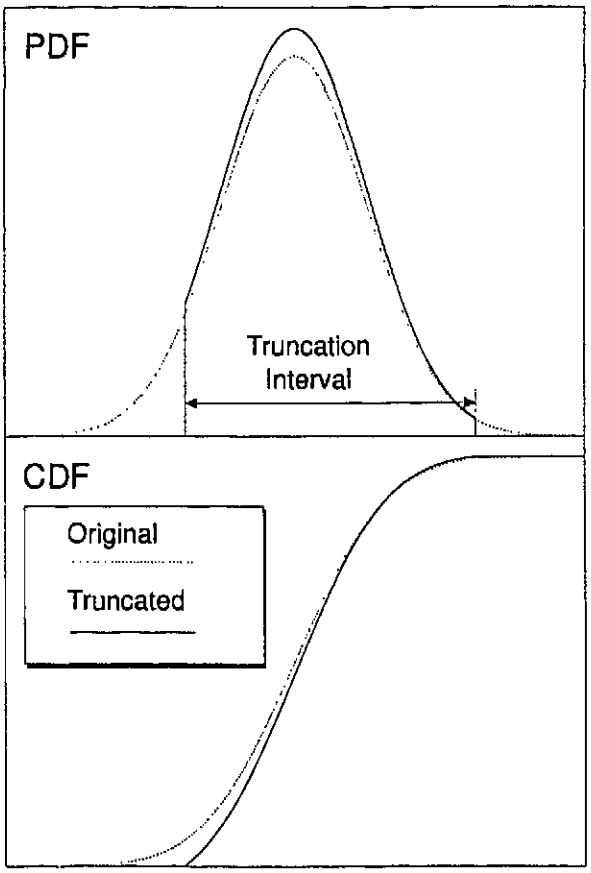

FIGURE 4.3/1: Comparing a Truncated Distribution to the Original Distribution

ATTRIBUTES AND OPERATIONS

The full set of Parameter Distribution operations is shown in Table 4.3/1. Of these, some are optional parts of a conforming implementation. In particular, PDF, mean, median and Sigma are optional (and not implemented in SV309). Section 4.4 gives full specifications for each operation in the table.

The Parameter Distribution operations shown in Figure 4.3/2 are abbreviated. "*Statistics*" refers to mean, median and Sigma; "*bounds*" refers to LowValue, HighValue, LowProbability and HighProbability. This figure also shows the attributes (preceded by ":") and operations (preceded by "-") of the Parameter Distribution object types. The operations differ from those of the same name applied to a Probability Distribution in that they allow for the Truncation Interval. For example, in Figure 4.3/1, the truncated PDF, shown by a solid line, is clearly

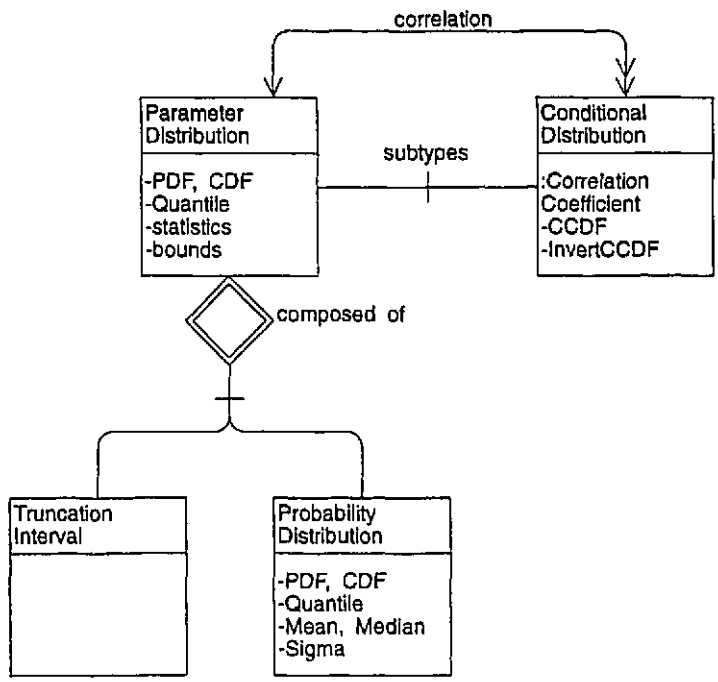

FIGURE 4.3/2: Parameter Distributions and Associated Object Types
(CCDF is the conditional CDF)

TABLE 4.3/1

OPERATIONS ON A PARAMETER DISTRIBUTION

`CDF[q_ParDis, x_?NumberQ]`	evaluate CDF at *x*
`HighProbability[q_ParDis]`	evaluate the cumulative probability of the Probability Distribution component at the upper limit of the Parameter Distribution
`HighValue[q_ParDis]`	find the upper limit of the Parameter Distribution
`LowProbability[q_ParDis]`	evaluate the cumulative probability of the internal Probability Distribution at the lower limit of the Parameter Distribution
`LowValue[q_ParDis]`	find the lower limit of the Parameter Distribution
`mean[q_ParDis]`	compute average value
`median[q_ParDis]`	compute 50th percentile
`PDF[q_ParDis, x_?NumberQ]`	evaluate PDF at *x*
`quantile[q_ParDis, p_?ProbabilityQ]`	invert CDF at probability *p*
`Sigma[q_ParDis]`	compute standard deviation

narrower and higher than the untruncated PDF, shown with a dotted line. To be valid PDFs, both must have an area of one. Similarly, the truncated CDF is narrower, but still ranges from zero to one.

4.4 Operations on Parameter Distributions

The Parameter Distribution operations defined in Section 4.3 are based on applying operations of the same name to the Probability Distribution part of the Parameter Distribution. The Truncation Interval modifies these operations the same way, whatever the subtype of Probability Distribution involved.

The CDF of a Parameter Distribution at a point x gives the probability of sampling a value less than x and within the Truncation Interval. The PDF gives the corresponding density, which for continuous functions is the derivative of the CDF. The PDF must be zero outside the bounds of the Truncation Interval. Therefore, the CDF must be constant outside the Truncation Interval. The CDF takes the constant value zero to the left of the Truncation Interval, and it takes the constant value one to the right of the Truncation Interval.

Truncation also affects statistics of a Parameter Distribution. If an infinite tail of a distribution is truncated on the left, then the mean and median shift to the right, and the standard deviation becomes smaller. If truncated on the right, the mean and median shift left and the standard deviation becomes smaller. If both tails are truncated, as shown in Figure 4.3/1, the mean and median may move in either direction, but the standard deviation can only diminish.

The Mathematica expressions in Table 4.4/1 define the Parameter Distribution operations. The conventions are explained in Section 4.2. Any new aspects of Mathematica used here are explained where they occur. Notice that CDF of a `ParDis` object is defined in terms of CDF of a `ProbDis` object, and similarly for PDF and quantile.

TABLE 4.4/1
PARDIS OPERATIONS

CDF

```
CDF[q:ParDis[d_ProbDis, t_TruncInt], x_?NumberQ] :=
   Which[
      x < LowValue[q],    0,
      x > HighValue[q],   1,
      True,          (CDF[d,x] - CDF[d,LowValue[q]]) /
                        (CDF[d,HighValue[q]] - CDF[d,LowValue[q]]) ]
```

where `Which[`$test_1$, $value_1$, $test_2$, $value_2$, ...`]` returns the first $value_i$ associated with $test_i$ that evaluates to `True`.

HighProbability

```
HighProbability[q:ParDis[d_ProbDis, t_TruncInt]] :=
   Which[
      MatchQ[t, TruncInt[ProbInt, lp_?NumberQ, hp_?NumberQ]],  t[[3]],
      MatchQ[t, TruncInt[ValInt, lv_?NumberQ, hv_?NumberQ]],   CDF[d,
                                                               t[[3]] ],
      True,                                                    1 ]
```

where `MatchQ[`*exp*, *pat*`]` takes the value `True` if the expression *exp* matches the pattern *pat*, and `False` otherwise.

TABLE 4.4/1 (concluded)

HighValue

```
HighValue[q:ParDis[d_ProbDis, t_TruncInt]] :=
   Which[
      MatchQ[t, TruncInt[ProbInt, lp_?NumberQ, hp_?NumberQ]],  quantile[d,
                                                               t[[3]]],
      MatchQ[t, TruncInt[ValInt, lv_?NumberQ, hv_?NumberQ]],   t[[3]],
      True,                                                    Infinity ]
```

LowProbability

```
LowProbability[q:ParDis[d_ProbDis, t_TruncInt]] :=
   Which[
      MatchQ[t, TruncInt[ProbInt, lp_?NumberQ, hp_?NumberQ]],  t[[2]],
      MatchQ[t, TruncInt[ValInt, lv_?NumberQ, hv_?NumberQ]],   CDF[d,
                                                               t[[2]] ],
      True,                                                    0 ]
```

LowValue

```
LowValue[q:ParDis[d_ProbDis, t_TruncInt]] :=
   Which[
      MatchQ[t, TruncInt[ProbInt, lp_?NumberQ, hp_?NumberQ]],  quantile[d,
                                                               t[[2]]],
      MatchQ[t, TruncInt[ValInt, lv_?NumberQ, hv_?NumberQ] ],  t[[2]],
      True,                                                    -Infinity ]
```

mean

```
mean[q:ParDis[d_ProbDis, t_TruncInt]] :=
   Integrate[(x PDF[q,x]), {x, LowValue[q], HighValue[q]} ]
```

median

```
median[q_ParDis] := quantile[q, 0.5]
```

PDF

```
PDF[q:ParDis[d_ProbDis, t_TruncInt], x_?NumberQ ] :=
   Which[
      x < LowValue[q],   0,
      x > HighValue[q],  0,
      True,              PDF[d,x] /
                         (CDF[d,HighValue[q]] - CDF[d,LowValue[q]]) ]
```

quantile

```
quantile[q:ParDis[d_ProbDis, t_TruncInt], p_?ProbabilityQ] :=
   quantile[d, LowProbability[q] + p (HighProbability[q]-LowProbability[q]) ]
```

Sigma

```
Sigma[q:ParDis[d_ProbDis, t_TruncInt]] :=
Sqrt[ Integrate[(x-mean[q])^2 PDF[q, x], {x, LowValue[q],HighValue[q]}] ]
```

4.5 Conditional Distribution Objects

The Conditional Distribution object type is a subtype of Parameter Distribution having a correlation association with another independent Parameter Distribution. A conforming implementation of the Parameter Sampling Package must support Conditional Distributions where the Probability Distribution components of both correlated distributions are Normal or Log-normal Distributions.

Let X and Y be two correlated normal random values, with the X values coming from a Parameter Distribution, and the Y values coming from a Conditional Distribution. Assume the Correlation Coefficient is negative. Figure 4.5/1 shows three examples of the conditional distributions for Y when low (dashed), medium (solid) and high (dotted) values of X are selected. The Conditional Distribution operations CPDF (conditional PDF) and CCDF (conditional CDF) evaluate these conditional distributions. Figure 4.5/2 shows a scatter of points randomly sampled from X and Y. Histograms and theoretical PDFs plotted below and to the left of the figure illustrate the marginal PDFs of X and Y (i.e., the PDFs when each is examined independently). The PDF operation for the Conditional Distribution object Y is inherited from the Parameter Distribution object type; it evaluates the marginal distribution for Y just as PDF does for X. It is clear from both Figure 4.5/1 and from the scatterplot that the variation of Y-values given any particular value of X is less than the variation allowed by the marginal distribution. Table 4.5/1 lists the additional Parameter Distribution operations for a Conditional Distribution.

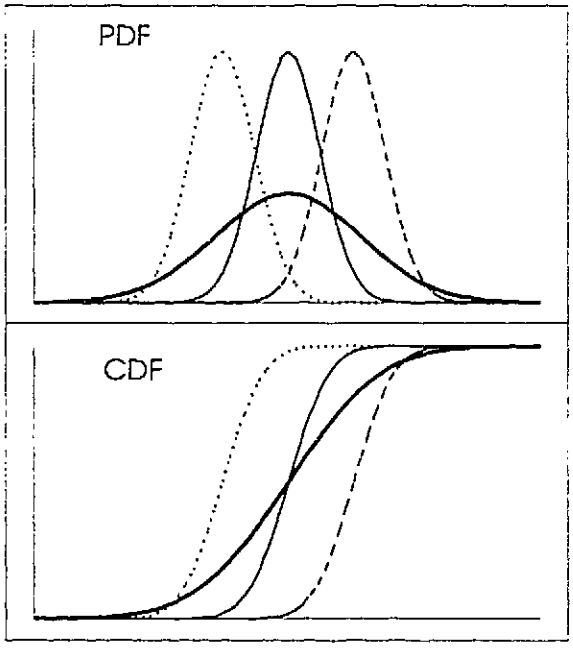

FIGURE 4.5/1: Marginal and Conditional PDFs and CDFs for a Correlated Normal

ASSOCIATIONS

Correlation–Each Conditional Distribution object is correlated with an independent Parameter Distribution object, in the sense that knowing a value chosen from the latter influences the distribution of the former. This association of object types is strictly one-way, even though, statistically, values sampled from the two Parameter Distributions are mutually correlated. While a Conditional Distribution is associated with precisely one independent Parameter Distribution, each Parameter Distribution can be the target of 0, 1 or several correlations.

ATTRIBUTES

Correlation Coefficient–The correlation coefficient establishes the strength of the correlation. A value of zero means no correlation, whereas a value of 1 (-1) means a direct linear relation with a positive (negative) slope. (The correlation coefficient in the two figures is -0.9.)

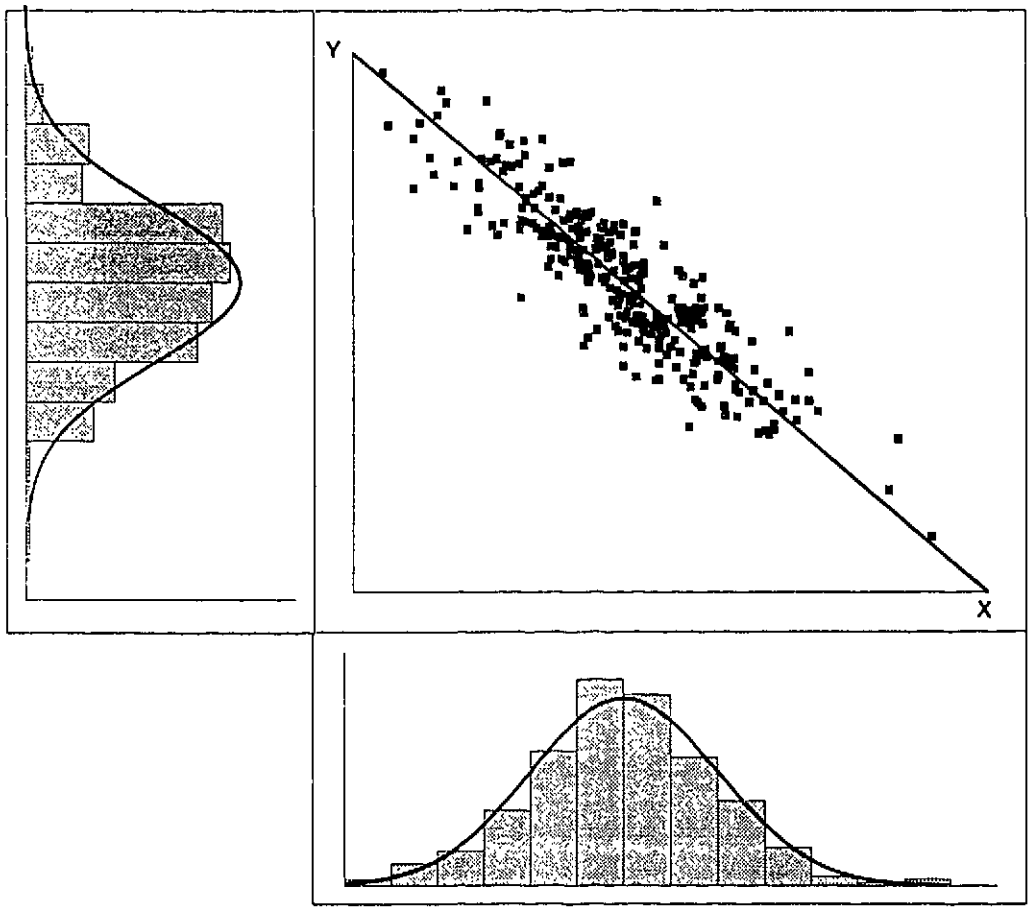

FIGURE 4.5/2: Scatterplot of Correlated Values with Marginal Distributions

TABLE 4.5/1

EXTRA PARAMETER DISTRIBUTION OPERATIONS FOR A CONDITIONAL DISTRIBUTION

`CCDF[q:ParDis[__, ConDis, __], y_?NumberQ, x_?NumberQ]`	evaluate conditional CDF at *x* given independent value *y*
`InvertCCDF[q:ParDis[__, ConDis, __], y_?NumberQ, p_?ProbabilityQ]`	invert conditional CDF at probability *p* given independent value *y*
`PDF[q:ParDis[__, ConDis, __], y_?NumberQ, x_?NumberQ]`	evaluate PDF at *x* given independent value *y*

4.6 Operations on Conditional Distributions

A Conditional Distribution has three parts: a Probability Distribution, a Truncation Interval, and a correlation association with another Parameter Distribution. In general, this information is not sufficient to uniquely identify a Conditional Distribution. There is one major exception: if the Conditional Distribution object has an untruncated Normal Distribution, and if the correlated Parameter Distribution object has an untruncated Normal Distribution, then the conditional distribution is also normally distributed, whatever value is chosen for the independent Parameter Distribution. It is essential that the implementation of a Conditional Distribution handle this case correctly. The implementation must also handle untruncated Lognormal Distributions correctly, using the assumption that it is the logarithm of the parameter that is correlated, not the parameter itself. Support for correlations of truncated Normal and Lognormal Distributions, and for other subtypes of Parameter Distribution, is optional. If support is given, it is desirable but not necessary that the achieved correlation match exactly the specified Correlation Coefficient. It is necessary, however, that the actual correlation be a monotonic function of the specified Correlation Coefficient.

Assume that X is a normally distributed variate with mean μ_X and standard deviation σ_X, written $X \sim N(\mu_X,\sigma_X)$. Assume that $Y \sim N(\mu_Y,\sigma_Y)$: that is, Y is also normally distributed. If X and Y are correlated, with correlation coefficient ρ, then the conditional distribution of X given a specific value of Y (i.e., $Y = y$) is also normal. More specifically, the conditional mean and variance of X are

$$E(X \mid Y=y) = \mu_X + \rho\sigma_X (y-\mu_Y) / \sigma_Y \quad \text{and} \tag{4.6-1}$$

$$\text{var}(X \mid Y=y) = \sigma_X^2 (1-\rho^2) \quad . \tag{4.6-2}$$

Together, the marginal distributions for both X and Y, the correlation coefficient ρ, and the value of Y uniquely specify the conditional distribution of X.

If either X or Y is distributed lognormally, then the logarithm of that variable is distributed normally. If we assume that the correlation coefficient applies to the logarithm, and not to the variable itself, then this case reduces to the previous case. Therefore, the case where either X or Y (or both) is distributed lognormally also has a unique solution.

In general, however, knowing the distributions for X and Y, the correlation coefficient , and the value of Y is not sufficient to uniquely define a conditional distribution for X. That is, there can be an infinite number of different conditional distributions for X, all of which satisfy the requirements.

This multiplicity of conditional distributions will arise even if X and Y have normal distributions, provided that at least one of the distributions is truncated. If the truncation is far out in the tails of the distribution, then all the possible conditional distributions will be similar to one another. They will also be similar to the conditional distribution for the untruncated case. If one of the distributions is severely truncated, however, the possible conditional distributions will be quite different from the untruncated case.

The Mathematica algorithms below first map each distribution onto a standard normal distribution N(0,1), and then apply the correlation association. They work correctly for untruncated Normal and Lognormal Distribution objects. They also give results when applied to other Parameter and Conditional Distributions, but in general the correlation achieved will not exactly match the target Correlation Coefficient. Implementation in these cases is optional. Implementors may also use some other scheme to induce correlation among sampled parameters. It is not necessary to have the actual correlation match the target Correlation Coefficient exactly since, in general, achieving that end is extremely difficult. The actual correlation must, however, satisfy these conditions:

(1) The actual correlation must be a monotonic function of the target Correlation Coefficient.
(2) If the Correlation Coefficient has a value of zero, then the conditional CDF (CCDF) and PDF must be the same as the marginal CDF and PDF.
(3) If the Correlation Coefficient has a value of ±1, then the range of the conditional distribution must degenerate to a single point.

Table 4.6/1 lists the Conditional Distribution operations.

TABLE 4.6/1
CONDITIONAL DISTRIBUTION OPERATIONS

CCDF

```
CCDF[q:ParDis[d_ProbDis, t_TruncInt, ConDis, c_?CorrCoefQ, q1_ParDis],
y_?NumberQ, x_?NumberQ] :=
   CDF[
      NormalDistribution[
         c Quantile[ NormalDistribution[0,1], CDF[q1,y] ],
         Sqrt[ 1 - c^2 ] ],
      Quantile[
         NormalDistribution[0, 1],
         CDF[ParDis[d, t], x ] ] ]
```

where `NormalDistribution[`*mu*`, `*sigma*`]` is a Normal Distribution object provided in the standard Mathematica package Statistics "ContinuousDistributions," and `CDF` and `Quantile` applied to a Normal Distribution object are operations from the same package.

InvertCCDF

```
InvertCCDF[q:ParDis[d_ProbDis, t_TruncInt, ConDis, c_?CorrCoefQ, q1_ParDis],
y_?NumberQ, p_?ProbabilityQ] :=
   quantile[
      ParDis[d, t],
      CDF[
         NormalDistribution[0,1],
         Quantile[
            NormalDistribution[
               c Quantile[ NormalDistribution[0,1], CDF[q1,y] ],
               Sqrt[ 1 - c^2 ] ],
            p ] ] ]
```

4.7 Truncation Interval Objects

Figure 4.7/1 shows the Truncation Interval and associated data types. A Truncation Interval object describes a continuous interval within which the value of a Sampled Parameter must lie. Ordinarily it would simply be considered a pair of attributes of a Parameter Distribution (i.e., Low Value and High Value). It is treated as an object with two subtypes because the interval can be specified either in the value or the probability domain (Figure 4.7/2). Every Parameter Distribution must have a single Truncation Interval component of one of the two subtypes. A conforming implementation may support only the Probability Interval subtype, in which case a Truncation Interval will be a Probability Interval.

ASSOCIATIONS

- Composed of–A Truncation Interval is a required component of every Parameter Distribution.
- Subtypes–A Truncation Interval can be of one of two subtypes, either a Probability Interval, or a Value Interval. It must be one or the other.

ATTRIBUTES

Figure 4.7/2 shows the (nonlinear) correspondence between values of a variable and cumulative probabilities associated with variable values. The set of attributes of a Truncation Interval object depends on the subtype:

(1) Probability Interval: the attributes are specified as lower and upper bounds in the probability domain:
   - Low Probability: the cumulative probability at the lowest value in the Truncation Interval, calculated using the Probability Distribution component of the Parameter Distribution;
   - High Probability: the cumulative probability at the highest value in the Truncation Interval calculated using the Probability Distribution component of the Parameter Distribution.

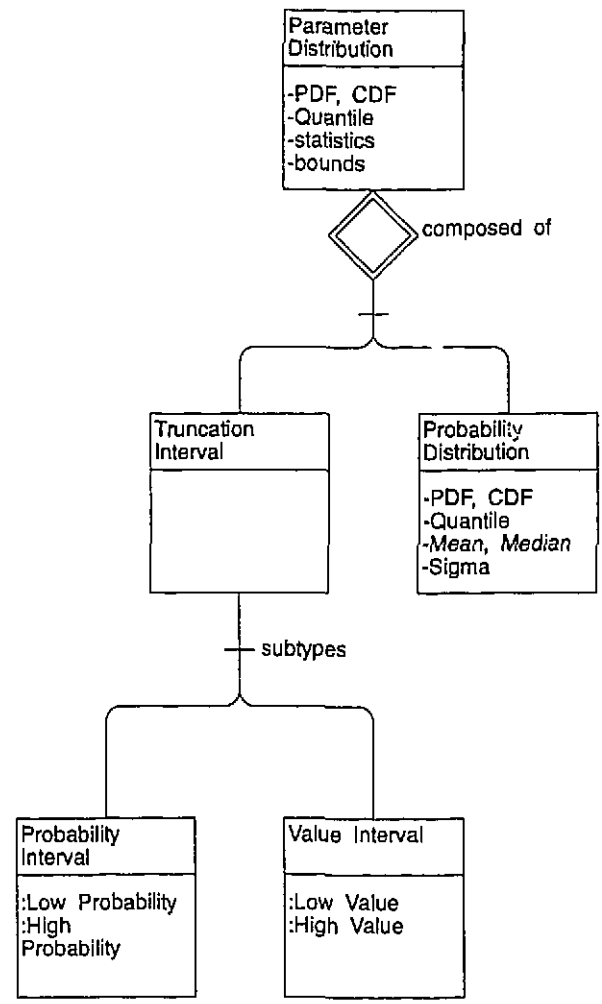

FIGURE 4.7/1: Truncation Interval and Associated Data Types

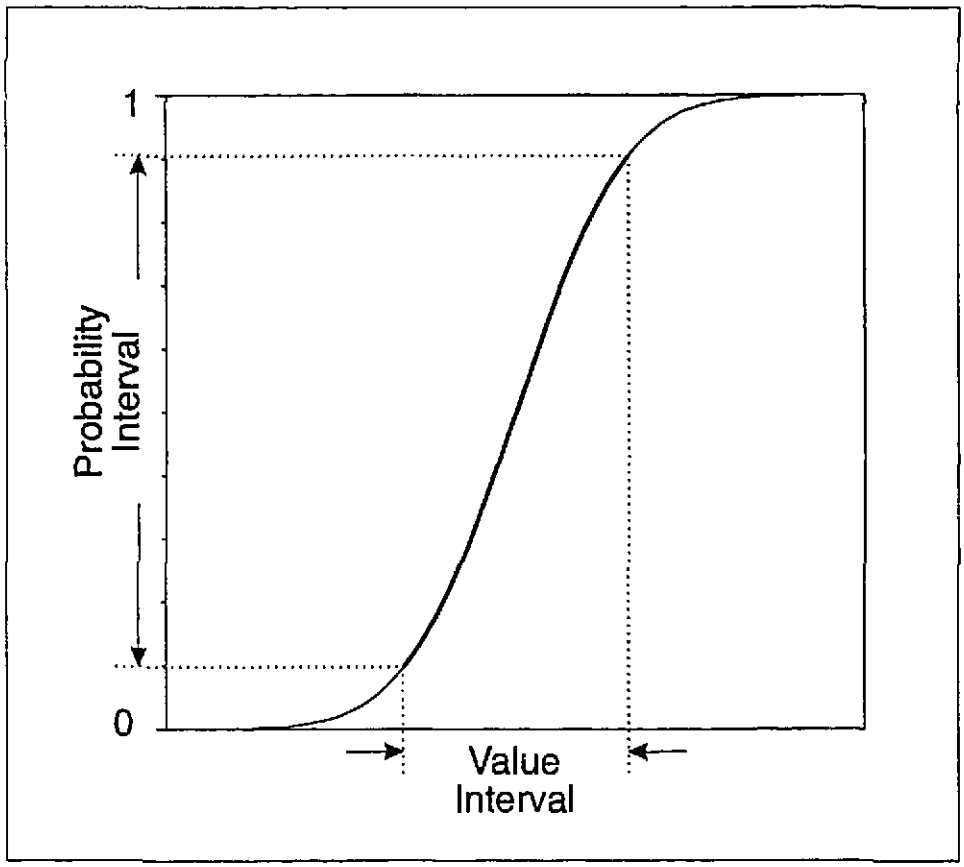

FIGURE 4.7/2: Relationship Between Probability Interval and Value Interval on a CDF Plot of a Probability Distribution

(2) Value Interval: the attributes are specified as lower and upper bounds in the value domain.
- Low Value: the greatest lower bound on the interval having a non-zero PDF.
- High Value: the least upper bound on the interval having a non-zero PDF.

Table 4.7/1 lists the Truncation Interval subtypes.

TABLE 4.7/1
SUBTYPES OF TRUNCATION INTERVAL

`ProbInt[lp_?ProbabilityQ, hp_?ProbabilityQ] /; lp < hp`	
`:= TruncInt[ProbInt, lp, hp]`	Probability Interval with low and high probabilities *lp* and *hp*
`ValInt[lv_?NumberQ, hv_?NumberQ] /; lv < hv`	
`:= TruncInt[ValInt, lv, hv]`	Value Interval with low and high values *lv* and *hv*

4.8 Probability Distribution Objects

The Probability Distribution object type encapsulates the characteristics of a mathematical probability distribution. That is, it describes how values for a random variable are distributed along a number line. A Parameter Distribution also performs this function; the difference between a Probability Distribution and a Parameter Distribution is the constraint that the latter lie within a Truncation Interval. Having the two different object types simplifies the discussion of operations such as PDF and CDF. These operations apply to the subtypes of Probability Distribution (Figure 4.8/1) without truncation. Since truncation of a distribution has a common effect on these operations, whatever the subtype of Probability Distribution involved, the effects of truncation need be discussed only for Parameter Distribution objects.

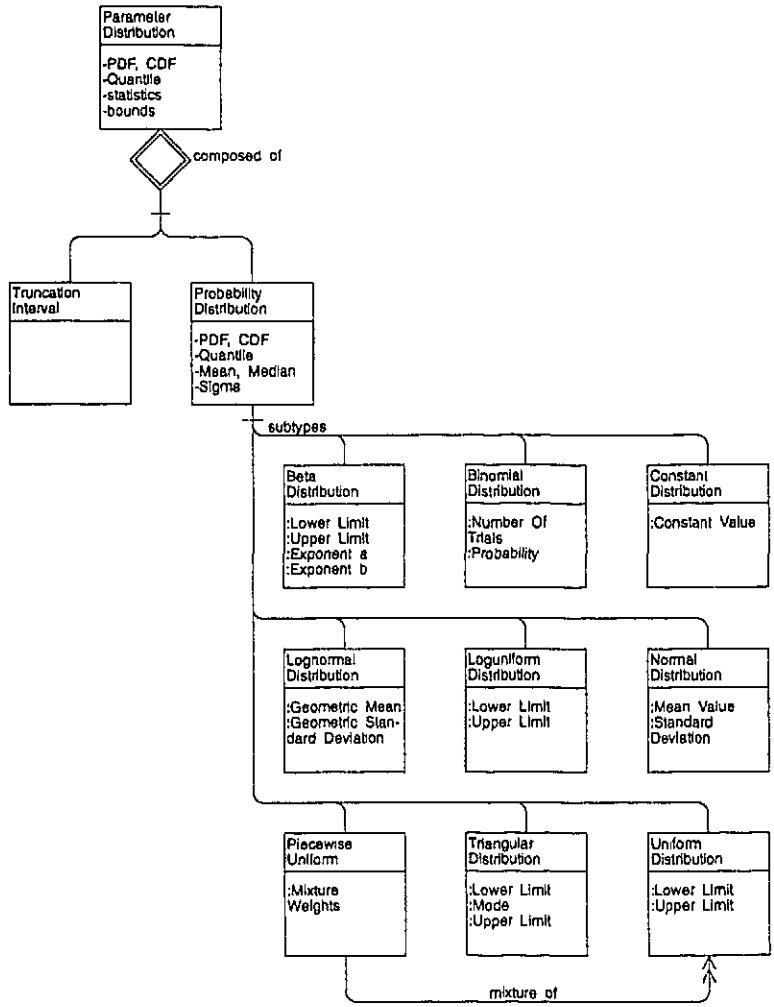

FIGURE 4.8/1: Probability Distribution and Associated Object Types

ASSOCIATIONS

The only associations that affect the Probability Distribution type are those that expand higher level objects into lower level objects (Figure 4.8/1).

- Composed of--A Probability Distribution is a required component of every Parameter Distribution.
- Subtypes--A Probability Distribution can be of one of 9 subtypes, with general shapes as shown in Figure 4.8/2. It must be one of these.

The operations on a Probability Distribution are listed in Table 4.8/1 on the next page.

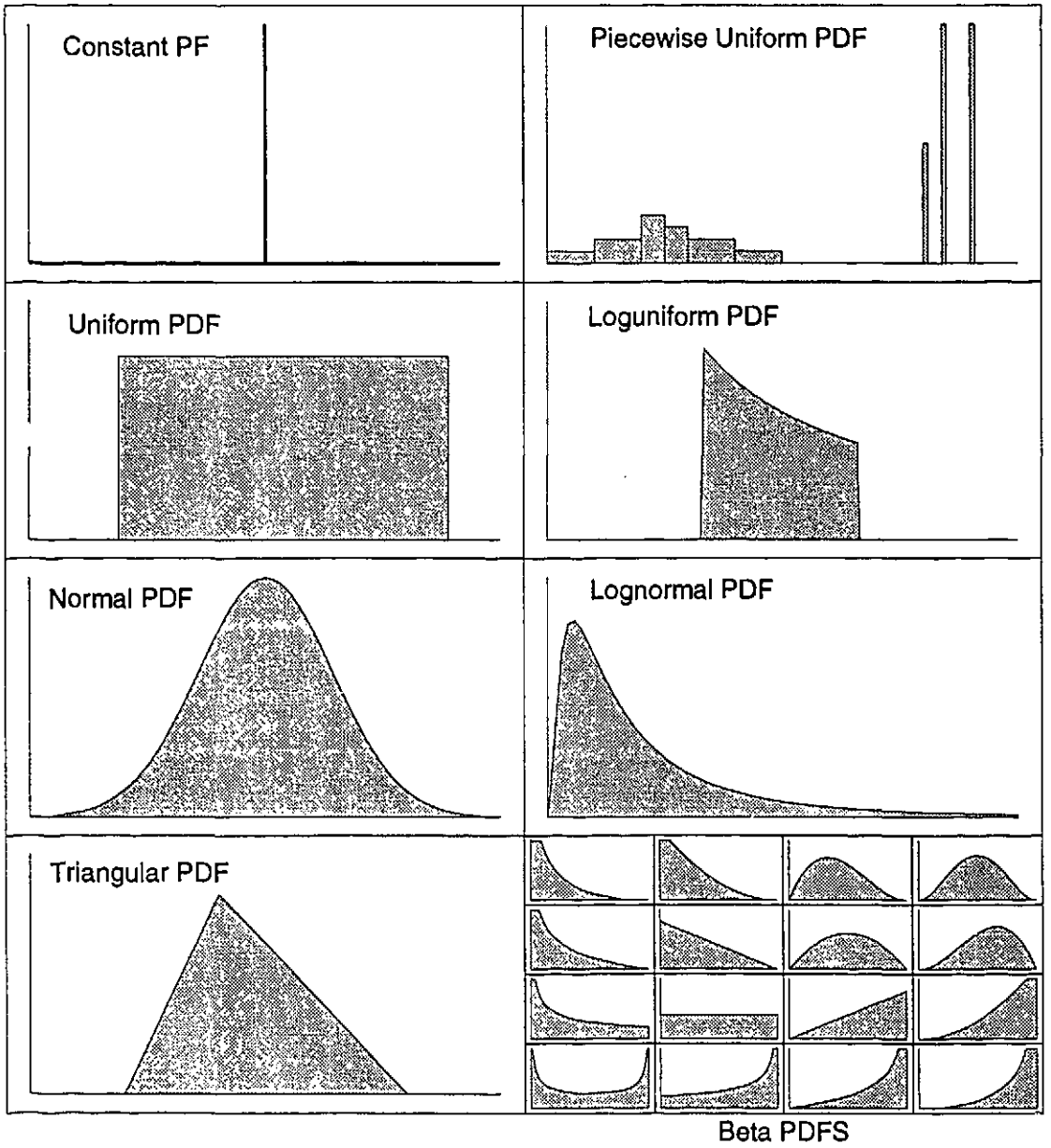

FIGURE 4.8/2: Shapes of SYVAC3 Probability Distributions

TABLE 4.8/1

OPERATIONS ON A PROBABILITY DISTRIBUTION

`CDF[d_ProbDis, x_?NumberQ]`	evaluate CDF at x
`mean[d_ProbDis]`	compute average value
`median[d_ProbDis]`	compute 50th percentile
`PDF[d_ProbDis, x_?NumberQ]`	evaluate PDF at x
`quantile[d_ProbDis, p_?ProbabilityQ]`	invert CDF at probability p
`Sigma[d_ProbDis]`	compute standard deviation

5 OPERATIONS ON PROBABILITY DISTRIBUTION OBJECTS

5.1 Probability Distribution: Beta Distribution

Figure 5.1/1, an excerpt from Figure 4.8/1, shows the object type Beta Distribution as a subtype of Probability Distribution. A Beta Distribution covers a finite interval specified by Lower and Upper Limits. It has two shape attributes that give it a variety of possible PDF shapes, as shown in Figure 5.1/2.

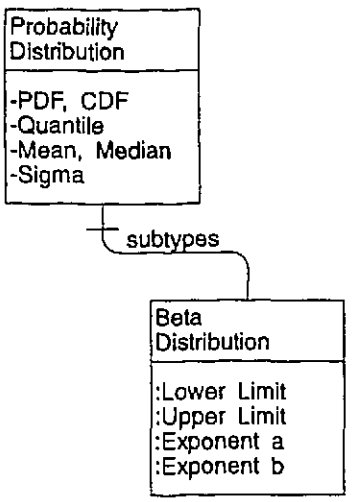

FIGURE 5.1/1: Beta Distribution as a Subtype of Probability Distribution

ASSOCIATIONS

- Subtypes–The Beta Distribution is a subtype of the Probability Distribution (Figure 5.1/1).

ATTRIBUTES

- Lower Limit–lower end of the interval with non-zero PDF.
- Upper Limit–upper end of the interval with non-zero PDF.
- Exponent *a*–the exponent of the independent variable X in the PDF for a Beta Distribution.
- Exponent *b*–the exponent of the expression $(1-X)$, where X is the independent variable, in the PDF for a Beta Distribution.

The attributes must satisfy the following constraints:

- Lower Limit < Upper Limit,
- $a > 0$, and
- $b > 0$.

OPERATIONS

The only operations are those inherited from the Probability Distribution. They are specified in detail in Section 5.2. The Mathematica representation of a Beta Distribution is shown in Table 5.1/1.

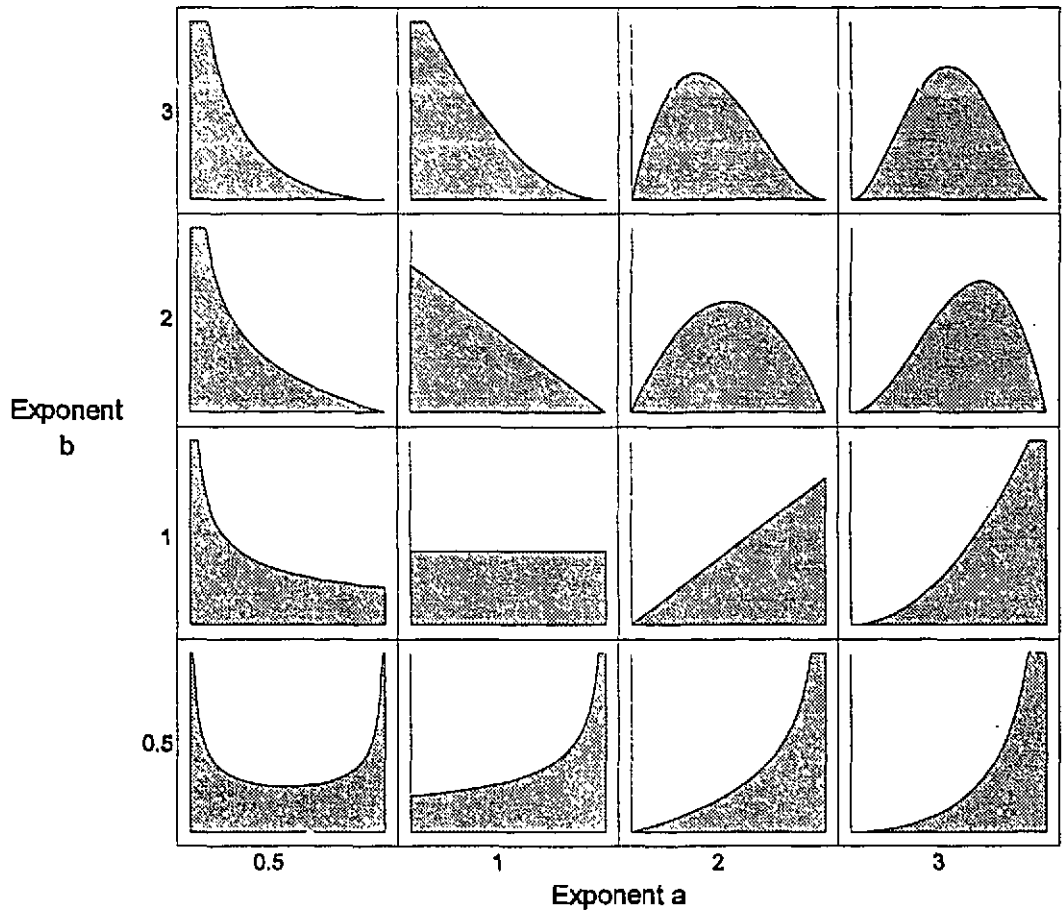

FIGURE 5.1/2: Sample Shapes for the Beta PDF

TABLE 5.1/1
OPERATIONS ON A BETA DISTRIBUTION

`BetaDis[low_?NumberQ, high_?NumberQ, a_?PositiveQ, b_?PositiveQ] /; low < high :=` `ProbDis[BetaDis, low, high, a, b]`	Beta Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[BetaDis, __], x_?NumberQ]`	evaluate cumulative distribution function at *x*
`mean[d:ProbDis[BetaDis, __]]`	compute average value
`median[d:ProbDis[BetaDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[BetaDis, __], x_?NumberQ]`	evaluate probability density function at *x*
`quantile[d:ProbDis[BetaDis, __], p_?ProbabilityQ]`	invert cumulative distribution function at probability *p*
`Sigma[d:ProbDis[BetaDis, __]]`	compute standard deviation

5.2 Beta Distribution Operations

Mathematica provides operations for a BetaDistribution object type, which is a subtype of Beta Distribution with Lower Limit = 0 and Upper Limit = 1. This section defines operations for a more general Beta Distribution in terms of the Mathematica operations. The definitions are also provided algebraically.

Mathematically, a standard beta distribution is defined over the interval from zero to one. It has two shape attributes a and b, which must both be positive. The PDF of a standard beta distribution is given by

$$f_x(a,b) = \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)} \qquad 0 \le x \le 1,\ 0 < a,\ 0 < b \tag{5.2-1}$$

where $B(a,b)$ is the complete beta function, defined as

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \qquad 0 < a,\ 0 < b \tag{5.2-2}$$

and $\Gamma(a)$ is the gamma function

$$\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt \qquad 0 < a\ . \tag{5.2-3}$$

The CDF of the standard beta distribution, also known as the incomplete beta ratio function, is

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)} \qquad 0 \le x \le 1,\ 0 < a,\ 0 < b \tag{5.2-4}$$

where $B_x(a,b)$ is the incomplete beta function

$$B_x(a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt \qquad 0 \le x \le 1,\ 0 < a,\ 0 < b. \tag{5.2-5}$$

These functions are all discussed by Spanier and Oldham (1987).

The mean value of a variate X with a standard beta distribution with shape attributes a and b is

$$\mathrm{E}(X) = \frac{a}{a+b} \qquad 0 < a,\ 0 < b. \tag{5.2-6}$$

The variance of such a variate is

$$\text{var}(X) = \frac{ab}{(a+b)^2(a+b+1)} \qquad 0 < a, 0 < b. \qquad (5.2\text{-}7)$$

Mathematica already implements all these equations, so the formal definitions in Table 5.2/1 give a succinct description of Beta Distribution operations. The definitions there allow a Lower Limit and Upper Limit that differ from zero and one, generalizing the standard beta distribution described here algebraically.

TABLE 5.2/1

BETA DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[BetaDis, low_?NumberQ, high_?NumberQ, a_?PositiveQ,
b_?PositiveQ] /; low < high, x_?NumberQ] :=
   Which[
      x < low,     0,
      x > high,    1,
      True,        CDF[BetaDistribution[a, b], (x-low)/(high-low)] ]
```

mean

```
mean[d:ProbDis[BetaDis, a_?PositiveQ, b_?PositiveQ, low_?NumberQ,
high_?NumberQ]] :=
   (b low + a high) / (a + b)
```

median

```
median[d:ProbDis[BetaDis, ___]] := quantile[d, 0.5]
```

PDF

```
PDF[d:ProbDis[BetaDis, low_?NumberQ, high_?NumberQ, a_?PositiveQ,
b_?PositiveQ] /; low < high, x_?NumberQ ] :=
   Which[
      x < low,     0,
      x > high,    1,
      True,        PDF[BetaDistribution[a,b], (x-low)/(high-low)] /
                   (high-low) ]
```

quantile

```
quantile[d:ProbDis[BetaDis, low_?NumberQ, high_?NumberQ, a_?PositiveQ,
b_?PositiveQ], p_?ProbabilityQ] :=
   low + Quantile[BetaDistribution[a, b], p] (high-low)
```

Sigma

```
Sigma[d:ProbDis[BetaDis, low_?NumberQ, high_?NumberQ, a_?PositiveQ,
b_?PositiveQ] /; low < high] :=
   (high-low) Sqrt[a b/(a+b+1)] / (a+b)
```

5.3 Probability Distribution: Binomial Distribution

Figure 5.3/1, an excerpt from Figure 4.8/1, shows the object type Binomial Distribution as a subtype of Probability Distribution. A Binomial Distribution is discrete; only a finite number of equally spaced values belong to its domain.

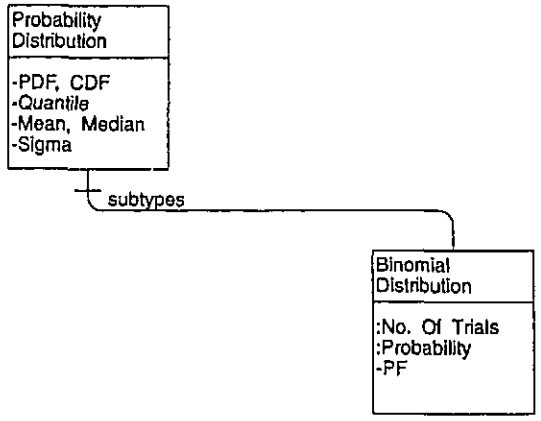

FIGURE 5.3/1: Binomial Distribution as a Subtype of Probability Distribution

A binomial variate X represents the number of successes in a sequence of N Bernoulli trials, where each trial independently has a probability p of success. For example, if a fair coin is tossed 10 times, the number of times it comes up "heads" is a binomial variate with 10 trials and probability 0.5 of success. As N gets to be very large, X/N falls on an ever denser scattering of possible values in the interval from 0 to 1. The limiting distribution of these points is a normal distribution with mean p and standard deviation $[Np(1-p)]^{0.5}$. Figure 5.3/2 shows a plot of the *probability function* (PF) (not the probability density function) for a binomial variate for N = 20 and p = 0.2. A probability function associates a finite probability, here represented by a narrow bar, with each possible value.

ASSOCIATIONS

- Subtypes–The Binomial Distribution is a subtype of Probability Distribution (Figure 5.3/1).

ATTRIBUTES

- Number of Trials–The number of Bernoulli trials, n, used to generate a binomial variate; the variate can take integer values from 0 to n.
- Probability–The probability p of success in any Bernoulli trial.

The attributes must satisfy the following constraints:

- $n \geq 0$, n an integer, and
- $0 \leq p \leq 1$.

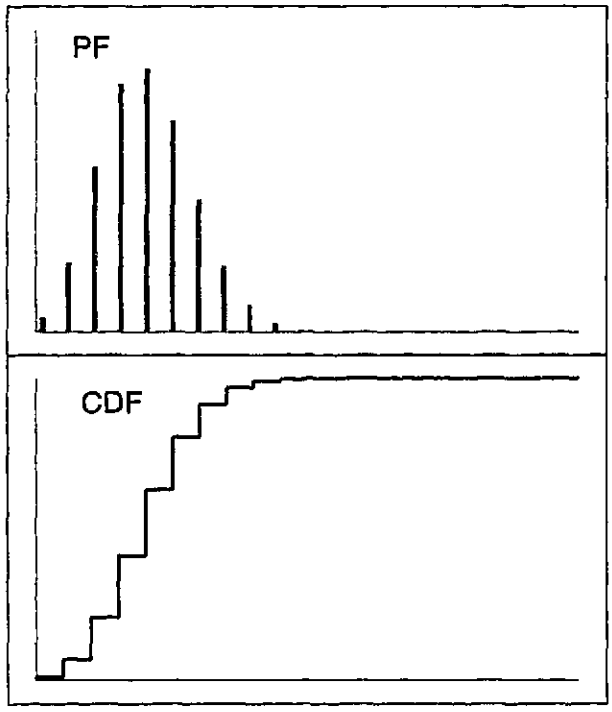

FIGURE 5.3/2: Probability Function and CDF for a Binomial Distribution

OPERATIONS

The Binomial Distribution inherits most operations from the Probability Distribution. In addition, there is PF, the probability function operation. Section 5.4 specifies all operations in detail. The Mathematica representation of a Binomial Distribution is shown in Table 5.3/1.

TABLE 5.3/1

OPERATIONS ON A BINOMIAL DISTRIBUTION

`BinoDis[n_?NonNegIntegerQ, p_?ProbabilityQ] := ProbDis[BinoDis, n, p]`	Binomial Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[BinoDis, __], x_?NumberQ]`	evaluate cumulative distribution function at *x*
`mean[d:ProbDis[BinoDis, __]]`	compute average value
`median[d:ProbDis[BinoDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[BinoDis, __], x_?NumberQ]`	evaluate probability density function at *x*
`PF[d:ProbDis[BinoDis, __], x_?NumberQ]`	evaluate probability function at *x*
`quantile[d:ProbDis[BinoDis, __], p1_?ProbabilityQ]`	invert cumulative distribution function at probability *p1*
`Sigma[d:ProbDis[BinoDis, __]]`	compute standard deviation

5.4 Binomial Distribution Operations

Mathematica provides operations for a BinomialDistribution object, which is identical to a BinoDis object in having two attributes: n, the Number of Trials, and p, the Probability of success. This section defines operations in terms of the Mathematica operations. The definitions are also provided algebraically.

Mathematically, a binomial distribution is defined over the set of positive integers from 0 to n. It gives the distribution of values for a variate that represents the number of successes in n independent Bernoulli trials. In a single Bernoulli trial, "success" or "failure" is chosen randomly, with "success" having a probability p of occurring. The maximum number of successes possible in n trials is n, and the minimum number is 0.

The PDF for a discrete variable involves a "Dirac comb," or a sum of multipliers times Dirac delta functions. The BinoDis PDF is

$$f_x(n,p) = \sum_{j=0}^{n} g_j(n,p)\delta(x-j) \qquad 0 \le n,\ 0 \le p \le 1, \tag{5.4-1}$$

where $g_j(n,p)$ is the probability function (PF), which gives the probability of j successes out of n trials. A sample PF is plotted in Figure 5.3/2. The definition of $g_j(n,p)$ is

$$g_j(n,p) = \binom{n}{j} p^j (1-p)^{n-j} \qquad 0 \le j \le n,\ 0 \le p \le 1. \tag{5.4-2}$$

The CDF of the binomial distribution can be computed directly from the PDF or the PF:

$$F_x(n,p) = \int_{-\infty}^{x} f_x(n,p)dx = \sum_{j=0}^{k} g_j(n,p) \qquad 0 \le n,\ 0 \le p \le 1, \tag{5.4-3}$$

where k is chosen to satisfy the following conditions:

- if $x < 0$, then $k = -1$; in this case the summation is empty, and $F_x(n,p) = 0$.
- if $0 \le x \le n$, then k is the largest integer not greater than x; in this case the summation holds over all integer values from 0 up to x.
- if $n < x$, then $k = n$; in this case $F_x(n,p) = 1$.

These conditions all arise because of the definition of the Dirac delta function and of its behaviour in integrals. The behaviour when x is an integer between 0 and n is subject to convention; since we have defined the CDF $F_x(n,p)$ to be the probability of sampling a value less than or equal to x, it includes the probability of precisely x successes.

When n is large, these calculations of cumulative probability can be computationally intensive, and so another method of computing the same result is of interest. Press et al. (1986) give the following equivalence:

$$F_x(n,p) = \sum_{j=0}^{k} g_j(n,p) = I_{1-p}(n-k,k+1) \qquad 0 \le k \le n,\ 0 \le p \le 1, \tag{5.4-4}$$

where $I_x(a,b)$ is the incomplete beta ratio function (but called the incomplete beta function by Press et al. (1986)) defined in the discussion of the Beta Distribution in Section 5.2.

The mean and variance of a variate X with a binomial distribution having attributes n and p are

$$\mathrm{E}(X) = np \qquad 0 \le n,\ 0 \le p \le 1, \tag{5.4-5}$$

$$\mathrm{var}(X) = np(1-p) \qquad 0 \le n,\ 0 \le p \le 1. \tag{5.4-6}$$

Table 5.4/1 lists the Binomial Distribution operations.

TABLE 5.4/1

BINOMIAL DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ], x_?NumberQ] :=
   CDF[BinomialDistribution[n, p], x]
```

mean

```
mean[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ] ] := n p
```

median

```
median[d:ProbDis[BinoDis, ___]:=quantile[d, 0.5]
```

PDF

```
PDF[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ], x_?NumberQ ] :=
   Which[
      x < 0,          0,
      x > n,          0,
      True,           PDF[BinomialDistribution[n, p], x] DiracDelta[0] ]
```

PF

```
PF[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ], x_?NumberQ ] :=
   Which[
      x < 0,          0,
      x > n,          0,
      True,           PDF[BinomialDistribution[n, p], x] ]
```

quantile

```
quantile[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ],
p1_?ProbabilityQ] :=
   Quantile[BinomialDistribution[n, p], p1]
```

Sigma

```
Sigma[d:ProbDis[BinoDis, n_?NonNegIntegerQ, p_?ProbabilityQ]] :=
   Sqrt[n p (1-p)]
```

5.5 Probability Distribution: Constant Distribution

Figure 5.5/1, an excerpt from Figure 4.8/1, shows the object type Constant Distribution as a subtype of Probability Distribution. A Constant Distribution permits only a single value for a random variable. It is a discrete distribution, but it can be thought of as the limiting case of a Uniform Distribution as the Lower and Upper Limits approach each other, or as the limiting case of a Normal Distribution as the Standard Deviation goes to 0. The PDF for a Constant Distribution is a Dirac delta function, and the cumulative distribution function is a Heaviside step function. Figure 5.5/2 shows the probability function (PF) and CDF.

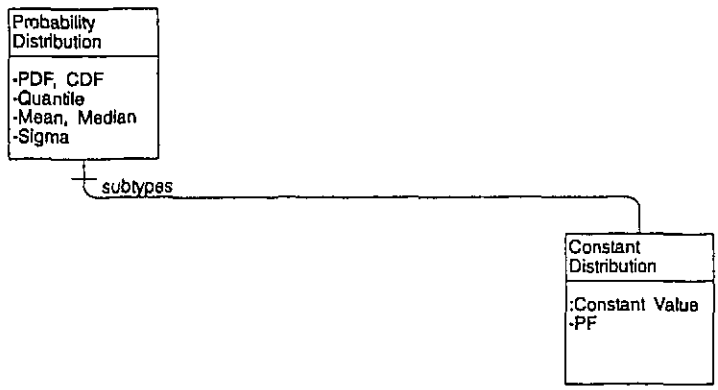

Figure 5.5/1: Constant Distribution as a Subtype of Probability Distribution

ASSOCIATIONS

- Subtypes–The Constant Distribution is a subtype of Parameter Distribution (Figure 5.5/1).

ATTRIBUTES

- Constant Value–The only permitted value for a random variable with this distribution. There is no constraint on the Constant Value.

OPERATIONS

The Constant Distribution inherits most operations from Probability Distribution. In addition, there is PF, the probability function operation. All operations are trivial because the distribution is so simple. They are specified in detail using Mathematica notation in Table 5.5/1.

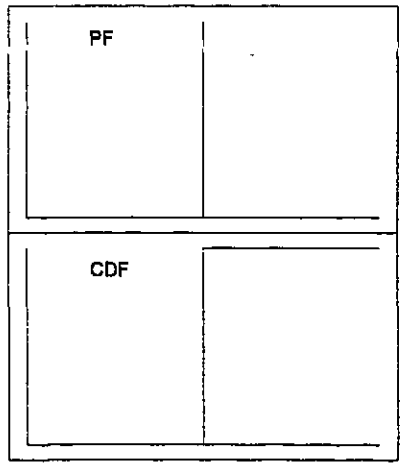

FIGURE 5.5/2: Probability Function and CDF for a Constant Distribution

TABLE 5.5/1
OPERATIONS ON A CONSTANT DISTRIBUTION

`ConstDis[v_?NumberQ] :=`	`ProbDis[ConstDis, v]` Constant Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[ConstDis, v_?NumberQ], x_?NumberQ] :=`	`UnitStep[x-v]` evaluate cumulative distribution function at x using the Heaviside step function, *UnitStep*(x-v) which is 0 for $x < v$, and 1 for $x \geq v$.
`mean[d:ProbDis[ConstDis, v_?NumberQ]] :=`	`v` compute average value
`median[d:ProbDis[ConstDis, v_?NumberQ]] :=`	`v` compute 50th percentile
`PDF[d:ProbDis[ConstDis, v_?NumberQ], x_?NumberQ] :=`	`DiracDelta[x-v]` evaluate probability density function at x using the Dirac delta function.
`PF[d:ProbDis[ConstDis, v_?NumberQ], x_?NumberQ] :=`	`If[x==v, 1, 0]` evaluate probability function at x using the *If* operation, which in this case takes the value 1 if $x = v$, and 0 otherwise.
`quantile[d:ProbDis[ConstDis, v_?NumberQ], p1_?ProbabilityQ] :=`	`v` invert cumulative distribution function at probability *p1*
`Sigma[d:ProbDis[ConstDis, v_?NumberQ]] :=`	`0` compute standard deviation

5.6 Probability Distribution: Lognormal Distribution

Figure 5.6/1, an excerpt from Figure 4.8/1, shows the object type Lognormal Distribution as a subtype of Probability Distribution. A Lognormal Distribution has a domain stretching from zero to infinity; all parameter values sampled from this distribution must be positive. A Lognormal Distribution has an asymmetric PDF skewed to the left (see Figure 5.6/2). In the limiting case where the Geometric Standard Deviation is very close to one, a Lognormal Distribution can resemble a Normal Distribution.

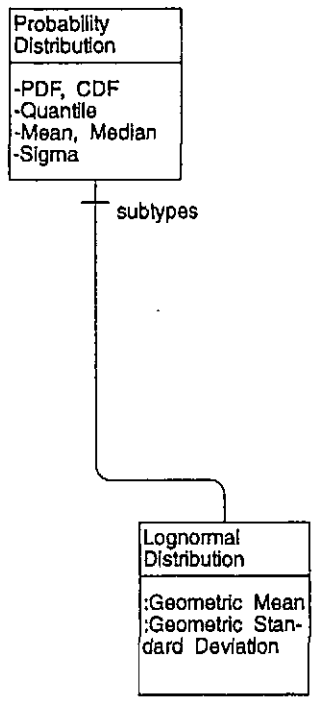

FIGURE 5.6/1: Lognormal Distribution as a Subtype of Probability Distribution

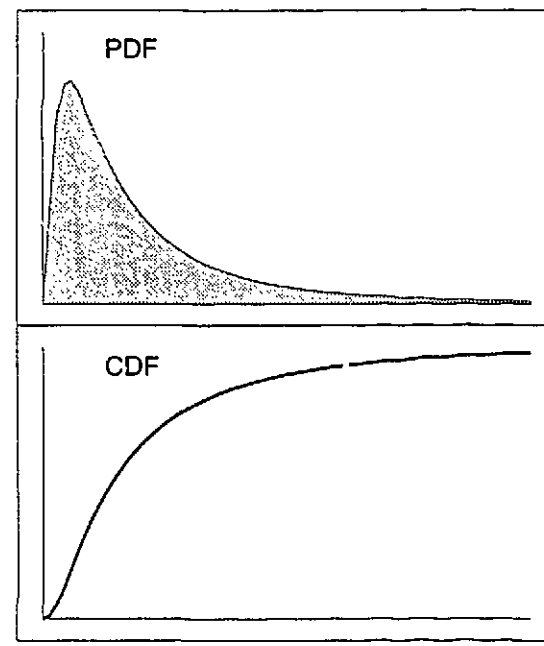

FIGURE 5.6/2: PDF and CDF for a Lognormal Distribution

The logarithm of a lognormal variate has a Normal Distribution. According to the central limit theorem of statistics, the Lognormal Distribution occurs naturally when one computes the geometric mean of the first N variates from an infinite sequence of positive random variables, all having the same variance.

ASSOCIATIONS

- Subtypes–The Lognormal Distribution is a subtype of Probability Distribution (Figure 5.6/1).

ATTRIBUTES

- Geometric Mean *gm*–the exponential of the mean of the natural logarithm of the Lognormal Distribution, with the same physical units as the random variable having this distribution.
- Geometric Standard Deviation *gsd*–the exponential of the standard deviation of the natural logarithm of the current Probability Distribution, having no physical units.

The attributes must satisfy the following constraints:

- $gm > 0$, and
- $gsd > 1$.

OPERATIONS

The only operations are those inherited from the Probability Distribution. They are specified in detail in Section 5.7. The Mathematica representation of a Lognormal Distribution is shown in Table 5.6/1.

TABLE 5.6/1

OPERATIONS ON A LOGNORMAL DISTRIBUTION

`LogNorDis[gm_?PositiveQ, gsd_?MoreThan1Q]`	
`:= ProbDis[LogNorDis, gm, gsd]`	Lognormal Distribution as a subtype of Probability Distribution, where
`MoreThan1Q[x_] := NumberQ[x] && (1 < x)`	test for a number greater than 1
`PositiveQ[x_] := NumberQ[x] && (0 < x)`	test for a positive number
`CDF[d:ProbDis[LogNorDis, __], x_?NumberQ]`	evaluate cumulative distribution function at *x*
`mean[d:ProbDis[LogNorDis, __]]`	compute average value
`median[d:ProbDis[LogNorDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[LogNorDis, __], x_?NumberQ]`	evaluate probability density function at *x*
`quantile[d:ProbDis[LogNorDis, __], p1_?ProbabilityQ]`	invert cumulative distribution function at probability *p1*
`Sigma[d:ProbDis[LogNorDis, __]]`	compute standard deviation

5.7 Lognormal Distribution Operations

Operations are provided by Mathematica for a LogNormalDistribution object having parameters mu and sigma to represent the mean and standard deviation. This section defines operations for a Lognormal Distribution in terms of the Mathematica operations. The definitions are also provided algebraically.

A random variable X has a lognormal distribution when $Y = \ln X$ has a normal distribution. If μ_Y and σ_Y represent the mean and standard deviation of Y, then the geometric mean and geometric standard deviation of X are defined by the following expressions:

$$gm_X = \exp(\mu_Y) \tag{5.7-1}$$

and

$$gsd_X = \exp(\sigma_Y) \ . \tag{5.7-2}$$

The PDF of a lognormal distribution is given by the following function:

$$f_x(gm,gsd) = \frac{1}{\sqrt{2\pi}x\log(gsd)}\exp\left[-\frac{(\log x - \log gm)^2}{2(\log gsd)^2}\right] \qquad 0 < x,\ 0 < gm,\ 1 < gsd \tag{5.7-3}$$

The CDF of a lognormal distribution is the same as that for a normal distribution, applied to the log of the variate:

$$F_x(gm,gsd) = \Phi_{\log x}(\log gm, \log gsd) \qquad 0 < x,\ 0 < gm,\ 1 < gsd \tag{5.7-4}$$

where $\Phi_x(\mu,\sigma)$ is the CDF at x of a normal variate with mean μ and standard deviation σ.

The mean value of a variate X with a lognormal distribution with geometric mean gm and geometric standard deviation gsd is

$$E(X) = gm\sqrt{gv} \qquad 0 < gm,\ 1 < gsd, \tag{5.7-5}$$

where gv is the geometric variance, defined to be

$$gv = \exp(\sigma_Y^2) \tag{5.7-6}$$

using the same definition for σ_Y as in Equation (5.7-2).

The variance of such a variate is

$$\text{var}(X) = gm^2 gv(gv-1) \qquad 0 < gm,\ 1 < gsd. \qquad (5.7\text{-}7)$$

Mathematica already contains equivalents to all these equations, given a LogNormalDistribution object with specified mean and standard deviation *for the logarithm*. Here, however, the simple equations above have been coded directly into Mathematica. Table 5.7/1 lists the Lognormal Distribution operations.

TABLE 5.7/1

LOGNORMAL DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[LogNorDis, gm_?PositiveQ, gsd_?MoreThan1Q], x_?NumberQ] :=
   Which[
      x <= 0,      0,
      x > 0,       CDF[NormalDistribution[Log[gm], Log[gsd]], Log[x] ] ]
```

mean

```
mean[d:ProbDis[LogNorDis, gm_?PositiveQ, gsd_?MoreThan1Q]] :=
   Module[
      {gv = Exp[ Log[gsd]^2 ]},
      gm Sqrt[gv] ]
```

where Module[{*x* = x_0, *y* = y_0, ... }, *body*] defines a code structure with local variables *x*, *y*, etc. The value of the module is given by the last line of the body.

median

```
median[d:ProbDis[LogNorDis, ___] := quantile[d, 0.5]
```

PDF

```
PDF[d:ProbDis[LogNorDis, gm_?PositiveQ, gsd_?MoreThan1Q], x_?NumberQ ] :=
   Which[
      x <= 0,      0,
      x > 0,       PDF[NormalDistribution[Log[gm], Log[gsd], Log[x] ] / x ]
```

quantile

```
quantile[d:ProbDis[LogNorDis, gm_?PositiveQ, gsd_?MoreThan1Q],
p_?ProbabilityQ] :=
   Exp[
      Quantile[NormalDistribution[Log[gm], Log[gsd]], p] ]
```

Sigma

```
Sigma[d:ProbDis[LogNorDis, gm_?PositiveQ, gsd_?MoreThan1Q]] :=
   Module[
      {gv = Exp[ Log[gsd]^2 ]},
      gm Sqrt[gv (gv-1)] ]
```

5.8 Probability Distribution: Loguniform Distribution

Figure 5.8/1, an excerpt from Figure 4.8/1, shows the Loguniform Distribution as a subtype of Probability Distribution. A Loguniform Distribution has a domain stretching over a positive finite interval; all parameter values sampled from this distribution must be positive. A Loguniform Distribution has an asymmetric PDF skewed to the left (see Figure 5.8/2). In the limiting case where the interval is very narrow compared to the Lower Limit, a Loguniform Distribution can resemble a Uniform Distribution.

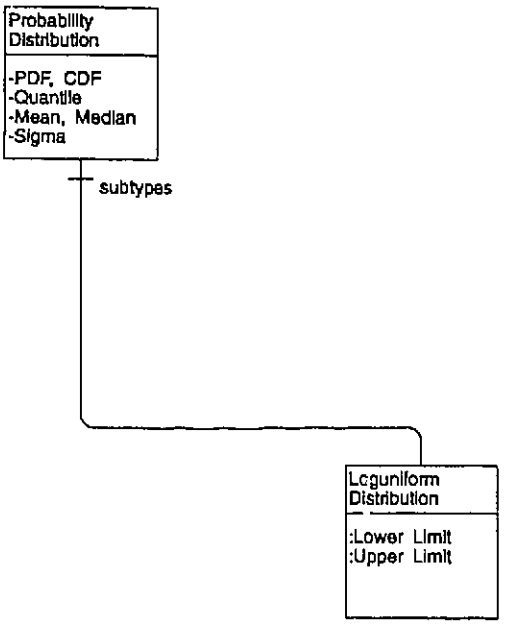

FIGURE 5.8/1: Loguniform Distribution as a Subtype of Probability Distribution

The logarithm of a loguniform variate has a Uniform Distribution. The Loguniform Distribution is appropriate where a positive value is usually measured on a log scale (e.g., pH or loudness), and where the value is completely unknown apart from sharp upper and lower bounds.

ASSOCIATIONS

- Subtypes–The Loguniform Distribution is a subtype of the Probability Distribution (Figure 5.8/1).

ATTRIBUTES

- Lower Limit–lower end of the interval with non-zero PDF.
- Upper Limit–upper end of the interval with non-zero PDF.

The attributes must satisfy the following constraint:

- Lower Limit < Upper Limit.

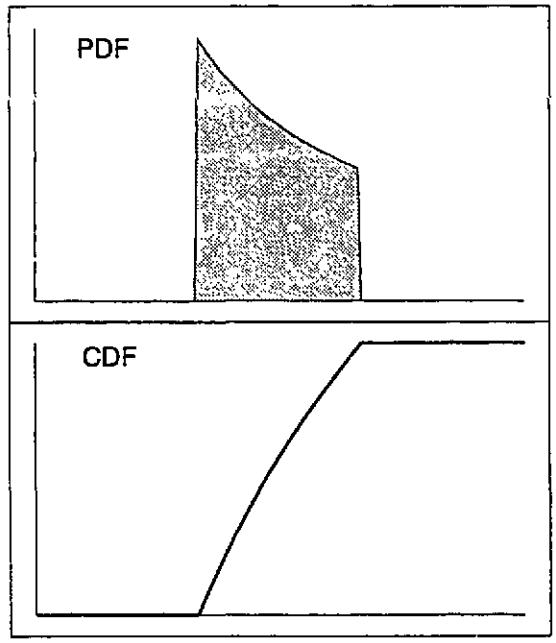

FIGURE 5.8/2: PDF and CDF for a Loguniform Distribution

OPERATIONS

The only operations are those inherited from the Probability Distribution. Section 5.9 specifies these operations in detail. The Mathematica representation of a Loguniform Distribution is shown in Table 5.8/1.

TABLE 5.8/1

OPERATIONS ON A LOGUNIFORM DISTRIBUTION

`LogUniDis[low_?PositiveQ, high_?PositiveQ] /; low < high`	
`:= ProbDis[LogUniDis, low, high]`	Loguniform Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[LogUniDis, __], x_?NumberQ]`	evaluate cumulative distribution function at x
`mean[d:ProbDis[LogUniDis, __]]`	compute average value
`median[d:ProbDis[LogUniDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[LogUniDis, __], x_?NumberQ]`	evaluate probability density function at x
`quantile[d:ProbDis[LogUniDis, __], p_?ProbabilityQ]`	invert cumulative distribution function at probability p
`Sigma[d:ProbDis[LogUniDis, __]]`	compute standard deviation

5.9 Loguniform Distribution Operations

Mathematica does not provide operations for a LogUniformDistribution object, but the equations are simple. Operations for a Loguniform Distribution are defined algebraically and by the equivalent Mathematica expressions.

Statistically, a random variable X has a loguniform distribution between two positive values, X_L and X_H, when $Y = \log X$ has a uniform distribution between $\log X_L$ and $\log X_H$. The PDF of X, $f_x(X_L,X_H)$, is just the PDF of Y at the same point, divided by the value of X

$$f_x(X_L,X_H) = \frac{U(x-X_L) - U(x-X_H)}{x(\log X_H - \log X_L)} \qquad 0 < x,\ 0 < X_L < X_H\ , \tag{5.9-1}$$

where $U(x)$ is the Heaviside step function, used to ensure that the PDF is zero outside the domain $[X_L,X_H]$.

The CDF of X is the same as that of Y at the corresponding point:

$$\begin{aligned} F_x(X_L,X_H) &= U(x-X_H) + \left[U(x-X_L) - U(x-X_H)\right]\left(\frac{\log x - \log X_L}{\log X_H - \log X_L}\right) \\ &= U(x-X_H) + \left[U(x-X_L) - U(x-X_H)\right]\left(\frac{\log\frac{x}{X_L}}{\log\frac{X_H}{X_L}}\right) \end{aligned} \qquad 0 < x,\ 0 < X_L < X_H\ . \tag{5.9-2}$$

Solving this equation for x when the CDF is 0.5 gives a simple expression for the median:

$$\text{median}(X) = \sqrt{X_L X_H}\ . \tag{5.9-3}$$

Integration of $xf_x(X_L,X_H)$ from X_L to X_H shows that the mean value of the variate X is

$$\text{E}(X) = \frac{X_H - X_L}{\log X_H - \log X_L} \qquad 0 < x,\ 0 < X_L < X_H\ . \tag{5.9-4}$$

Integration of $(x - E(X))^2 f_x(X_L, X_H)$ gives the variance of the variate X, which can be simplified to:

$$\text{var}(X) = E(X)\left[\frac{1}{2}(X_L + X_H) - E(X)\right] \qquad 0 < x,\ 0 < X_L < X_H. \quad (5.9\text{-}5)$$

The formal definitions of these operations in Mathematica notation appear in Table 5.9/1.

TABLE 5.9/1

LOGUNIFORM DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ] /; low < high,
x_?NumberQ] :=
   Which[
      x < low,     0,
      x > high,    1,
      True,        Log[x/low] / Log[high/low] ]
```

mean

```
mean[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ]
/; low < high ] :=
   (high-low) / Log[high/low]
```

median

```
median[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ]
/; low < high ] :=
   Sqrt[low high]
```

PDF

```
PDF[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ] /; low < high,
x_?NumberQ ] :=
   Which[
      x < low,     0,
      x > high,    0,
      True,        1 / (x Log[high/low]) ]
```

quantile

```
quantile[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ]
/; low < high, p_?ProbabilityQ] :=
   low (high/low)^p
```

Sigma

```
Sigma[d:ProbDis[LogUniDis, low_?PositiveQ, high_?PositiveQ]
/; low < high ] :=
   Sqrt[mean[d] ( (low+high)/2 - mean[d] )]
```

5.10 Probability Distribution: Normal Distribution

Figure 5.10/1, an excerpt from Figure 4.8/1, shows the object type Normal Distribution as a subtype of Probability Distribution. A Normal Distribution has an infinite domain stretching from negative infinity to positive infinity. Its PDF is symmetric about the mean value, which is also the median and mode. The PDF has the form of a bell-shaped curve (Figure 5.10/2).

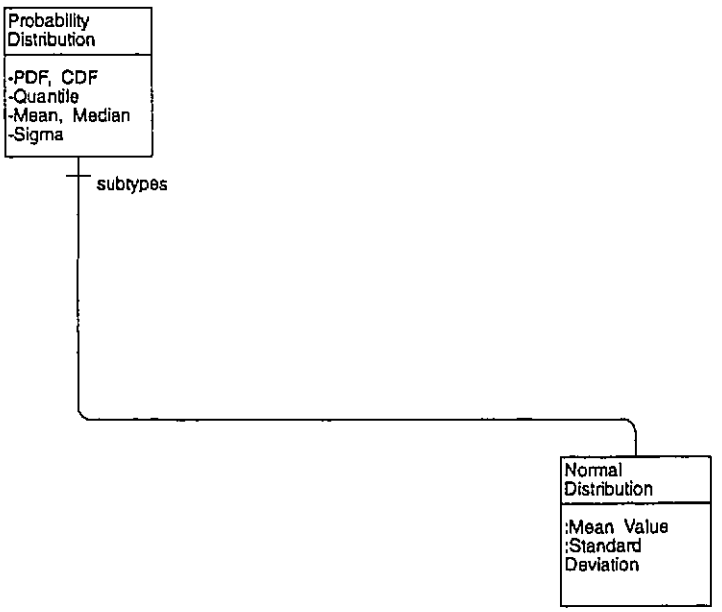

FIGURE 5.10/1: Normal Distribution as a Subtype of Probability Distribution

By the Central Limit Theorem of statistics, the probability distribution of the average of a large number of variates with similar distributions approaches a Normal Distribution as the number of variates gets very large. As a result, this distribution is often used to represent sums and averages.

ASSOCIATIONS

- Subtypes–The Normal Distribution is a subtype of the Probability Distribution (Figure 5.10/1).

ATTRIBUTES

- Mean Value μ–the position of the centre of symmetry of the distribution.
- Standard Deviation σ–square root of the variance.

The attributes must satisfy the following constraint:

- $\sigma > 0$.

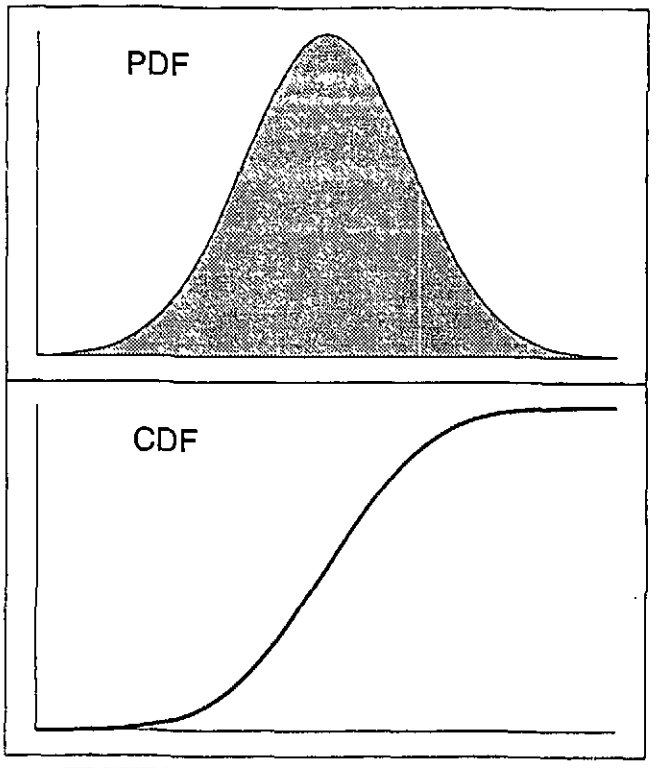

FIGURE 5.10/2: PDF and CDF for a Normal Distribution

OPERATIONS

The only operations are those inherited from the Probability Distribution. They are specified in detail in Section 5.11. The Mathematica representation of a Normal Distribution is shown in Table 5.10/1.

TABLE 5.10/1

OPERATIONS ON A NORMAL DISTRIBUTION

`NormDis[mu_?NumberQ, sigma_?PositiveQ] :=`	`ProbDis[NormDis, mu, sigma]` Normal Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[NormDis, __], x_?NumberQ]`	evaluate cumulative distribution function at x
`mean[d:ProbDis[NormDis, __]]`	compute average value
`median[d:ProbDis[NormDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[NormDis, __], x_?NumberQ]`	evaluate probability density function at x
`quantile[d:ProbDis[NormDis, __], p_?ProbabilityQ]`	invert cumulative distribution function at probability p
`Sigma[d:ProbDis[NormDis, __]]`	compute standard deviation

5.11 Normal Distribution Operations

Mathematica provides operations for a NormalDistribution object having parameters μ and σ to represent the mean and standard deviation. This section defines operations for a Normal Distribution in terms of Mathematica operations. The definitions are also provided algebraically.

The PDF of a normal distribution with mean μ and standard deviation σ is given by

$$\phi_x(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \qquad 0 < \sigma. \tag{5.11-1}$$

The CDF of a normal distribution is defined by the integral of the exponential function, $\phi_x(\mu,\sigma)$, which has no closed form in terms of elementary functions. It can be expressed in terms of other functions representing exponential integrals, however, and specifically in terms of the error function:

$$\begin{aligned}\Phi_x(\mu,\sigma) &\equiv \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right] \\ &\equiv \frac{1}{2}\left[2 - \text{erfc}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]\end{aligned} \qquad 0 < \sigma, \tag{5.11-2}$$

where erf(x) is the error function and erfc(x) is the complementary error function.

By definition, the mean and standard deviation of a variate X with a normal distribution are just μ and σ respectively. The median is also μ because of the symmetry of the distribution.

Mathematica already contains equivalents to all these equations, given a NormalDistribution object with specified mean and standard deviation. The specifications in Table 5.11/1 use those definitions.

TABLE 5.11/1

NORMAL DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ], x_?NumberQ] :=
   CDF[NormalDistribution[mu, sigma], x ]
```

mean

```
mean[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ]] := mu
```

median

```
median[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ] := mu
```

PDF

```
PDF[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ], x_?NumberQ ] :=
   PDF[NormalDistribution[mu, sigma], x]
```

quantile

```
quantile[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ],
p_?ProbabilityQ] :=
  Quantile[NormalDistribution[mu, sigma], p]
```

Sigma

```
Sigma[d:ProbDis[NormDis, mu_?NumberQ, sigma_?PositiveQ]] := sigma
```

5.12 Probability Distribution: Piecewise Uniform Distribution

Figure 5.12/1, an excerpt from Figure 4.8/1, shows the object type Piecewise Uniform Distribution as a subtype of Probability Distribution. A Piecewise Uniform Distribution is defined as a mixture of nonoverlapping Uniform Distributions. That is, the distribution is composed of a set of Uniform Distributions, each of which has a certain probability of being chosen. The PDF of a Piecewise Uniform Distribution is a histogram. As a result, it can approximate any other continuous distribution. But it is more flexible than that. By reducing the widths of the bins in the histogram to zero, the Piecewise Uniform Distribution can also represent a discrete distribution with a finite number of possible values.

Figure 5.12/2 shows a mixture of discrete components (where the bins have been kept wide enough to see) and continuous components to illustrate the flexibility of this Probability Distribution subtype.

ASSOCIATIONS

- Mixture of–Each bin in the PDF of a Piecewise Uniform Distribution represents a Uniform Distribution (Figure 5.12/1); the set of Uniform Distributions forms an ordered sequence associated with the integers from 1 to J.
- Subtypes–The Piecewise Uniform Distribution is a subtype of Probability Distribution (Figure 5.12/1).

OPERATIONS

The only operations are those inherited from the Probability Distribution (see Section 5.13). The Mathematica representation of a Piecewise Uniform Distribution is shown in Table 5.12/1.

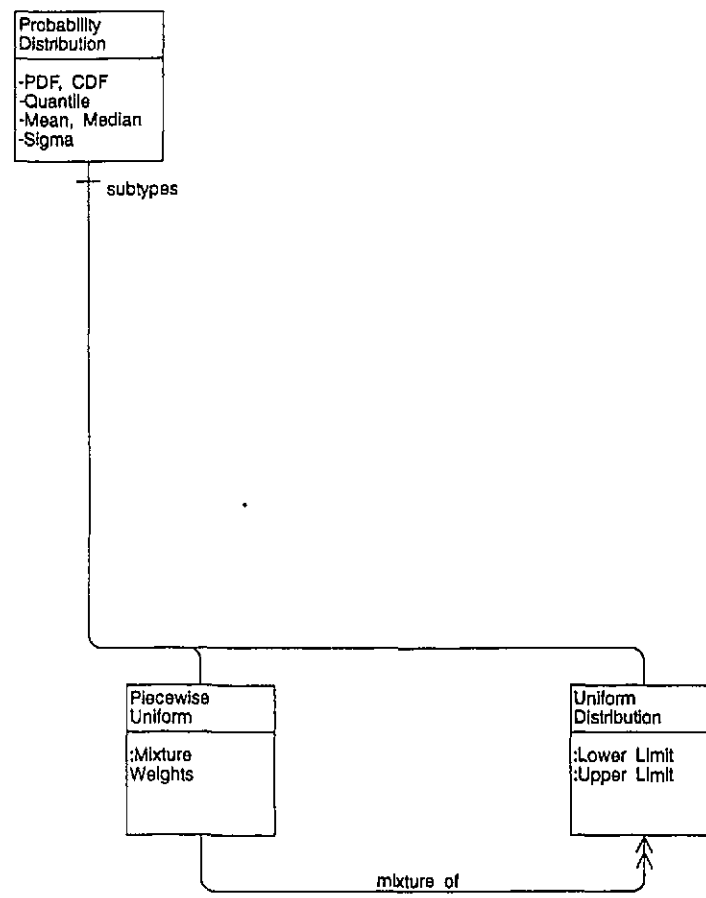

FIGURE 5.12/1: Piecewise Uniform Distribution as a Subtype of Probability Distribution

ATTRIBUTES

- Mixture Weights W_j—relative weights (not normalized to sum to 1), for j from 1 to J, of sampling from each of the Uniform Distributions that make up a Piecewise Uniform Distribution.

The attributes must satisfy the following constraints:

- $W_j \geq 0$ for each j from 1 to J, with

$$\sum_{j=1}^{J} W_j > 0 \ .$$

- If the j'th Uniform Distribution is defined on the interval $[L_j, H_j]$, for all Uniform Distributions from 1 to J, then $L_j \leq H_j$, for j from 1 to J; and $H_j \leq L_{j+1}$, for j from 1 to J-1.

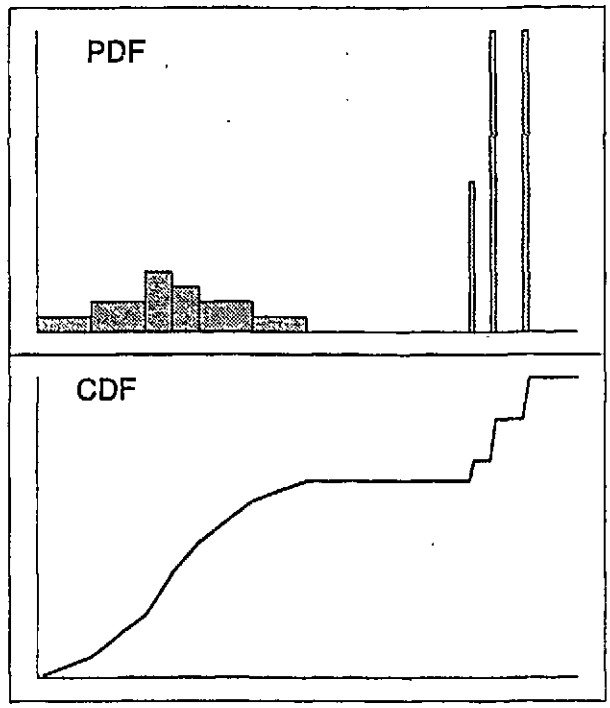

FIGURE 5.12/2: PDF and CDF for a Piecewise Uniform Distribution

TABLE 5.12/1

OPERATIONS ON A PIECEWISE UNIFORM DISTRIBUTION

`PcwUniDis[ws_List, ds_List] /; VectorQ[ws, PositiveQ] && VectorQ[ds, UniformQ] && (Length[ws] == Length[ds]) :=`	`ProbDis[PcwUniDis, ws, ds]` Piecewise Uniform Distribution as a subtype of Probability Distribution, where:
`Length[x_]`	Mathematica operation to find the length of a list x
`UniformQ[x_] :=`	`MatchQ[x, ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high]` tests for a valid Uniform Distribution (see Section 5.16)
`VectorQ[v_List, test]`	Mathematica operation to test each entry of a list v using operation *test*, and return *True* only if all entries pass
`CDF[d:ProbDis[PcwUniDis, __], x_?NumberQ]`	evaluate cumulative distribution function at x
`mean[d:ProbDis[PcwUniDis, __]]`	compute average value
`median[d:ProbDis[PcwUniDis, __]]`	compute 50th percentile
`PDF[d:ProbDis[PcwUniDis, __], x_?NumberQ]`	evaluate probability density function at x
`quantile[d:ProbDis[PcwUniDis, __], p_?ProbabilityQ]`	invert cumulative distribution function at probability p
`Sigma[d:ProbDis[PcwUniDis, __]]`	compute standard deviation

5.13 Piecewise Uniform Distribution Operations

The Piecewise Uniform Distribution is a mixture of Uniform Distributions. Most operations have definitions based on that fact. Since Mathematica does not have built-in operations for a Piecewise Uniform Distribution, the operational definitions in this section are expressed both in algebraic and Mathematica notation.

The PDF of a mixture is the weighted sum of the PDFs of the probability distributions that are mixed together. The weights for the PDF are the same as the weights that define the mixture. Define the probability of selecting the j'th distribution from a mixture by P_j, and define it to be

$$P_j = \frac{W_j}{\sum_{k=1}^{J} W_k} \qquad 1 \le j \le J. \tag{5.13-1}$$

Then the PDF $f(x)$ for Piecewise Uniform variate X is

$$f(x) = \sum_{j=1}^{J} P_j f_j(x) \ , \tag{5.13-2}$$

where $f_j(x)$ is the PDF of the j'th Uniform Distribution making up the mixture (see Section 5.17).

The CDF is the integral of Equation (5.13-2):

$$F(x) = \sum_{j=1}^{J} P_j F_j(x) \tag{5.13-3}$$

where $F_j(x)$ is the CDF of the j'th Uniform Distribution making up the mixture.

Similarly, the mean of a Piecewise Uniform Distribution is the weighted average of the means of the contributing Uniform Distributions:

$$\mathrm{E}(X) = \sum_{j=1}^{J} P_j \frac{L_j + H_j}{2} \ . \tag{5.13-4}$$

The median is more difficult to find – it can be evaluated as the inverse CDF at 0.5.

The variance of a Piecewise Uniform Distribution is made up of two parts: the average variance of the Uniform Distributions, plus variance of the expected values of the Uniform Distributions from $E(x)$:

$$\mathrm{var}(X) = \frac{1}{J} \sum_{j=1}^{J} \frac{P_j}{12} (H_j - L_j)^2 + \frac{1}{J} \sum_{j=1}^{J} P_j \left[\frac{L_j + H_j}{2} - \mathrm{E}(X) \right]^2 \ . \tag{5.13-5}$$

TABLE 5.13/1
PIECEWISE UNIFORM DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[PcwUniDis, ws_List, ds_List], x_?NumberQ] /;
VectorQ[ws, PositiveQ] && VectorQ[ds, UniformQ] && (Length[ws] == Length[ds])
:=
   Module[
      { arglist, cumprob },
      arglist := Thread[List[ds, x]];
      cumprob := Apply[CDF, arglist, {1} ];
      Dot[ws, cumprob] / Fold[Plus, 0, ws] ]
```

mean

```
mean[d:ProbDis[PcwUniDis, ws_List, ds_List] ] /; VectorQ[ws, PositiveQ] &&
VectorQ[ds, UniformQ] && (Length[ws] == Length[ds]) :=
   Dot[ws, Map[mean, ds]] / Fold[Plus, 0, ws]
```

median

```
median[d:ProbDis[PcwUniDis, ws_List, ds_List] ] /; VectorQ[ws, PositiveQ] &&
VectorQ[ds, UniformQ] && (Length[ws] == Length[ds]) :=
   quantile[d, 0.5]
```

PDF

```
PDF[d:ProbDis[PcwUniDis, ws_List, ds_List], x_?NumberQ] /;
VectorQ[ws, PositiveQ] && VectorQ[ds, UniformQ] && (Length[ws] == Length[ds])
:=
   Module[
      { arglist, localPDF },
      arglist := Thread[List[ds, x]];
      localPDF := Apply[PDF, arglist, {1} ];
      Dot[ws, localPDF] / Fold[Plus, 0, ws] ]
```

quantile

```
quantile[d:ProbDis[PcwUniDis, ws_List, ds_List], p_?ProbabilityQ] /;
VectorQ[ws, PositiveQ] && VectorQ[ds, UniformQ] && (Length[ws] == Length[ds])
:=
   Module[
      { wsloc, cumweight, udist, wtlist },
      wsloc := ws / Fold[Plus, 0, ws];
      cumweight := Drop[ FoldList[Plus, 0, wsloc], 1];
      wtlist := Thread[List[wsloc, cumweight, ds]];
      udist := Flatten[ Select[wtlist, #[[2]] >= p &, 1] ];
      quantile[udist[[3]], 1 - (udist[[2]]-p) / udist[[1]] ] ]
```

Sigma

```
Sigma[d:ProbDis[PcwUniDis, ws_List, ds_List] ] /;
VectorQ[ws, PositiveQ] && VectorQ[ds, UniformQ] && (Length[ws] == Length[ds])
:=
   Module[
      { lmean, lvar, wsloc },
      wsloc := ws / Fold[Plus, 0, ws];
      lmean := mean[d];
      lvar := Dot[wsloc,
         Map[Sigma, ds, {1}]^2 + (lmean - Map[mean, ds, {1}])^2 ];
      Sqrt[lvar] ]
```

NOTE: Consult a Mathematica manual for an explanation of the functions.

5.14 Probability Distribution: Triangular Distribution

Figure 5.14/1, an excerpt from Figure 4.8/1, shows the object type Triangular Distribution as a subtype of the Probability Distribution. A Triangular Distribution has a domain stretching over a finite interval (see Figure 5.14/2). It has a possibly asymmetric PDF with a single mode (peak) inside the interval. The PDF diminishes to zero at the endpoints of the interval unless the mode is at an endpoint. The Triangular Distribution is appropriate when all that is known about a parameter's value is the upper and lower bounds, and the approximate location of the mode.

ASSOCIATIONS

- Subtypes–The Triangular Distribution is a subtype of Probability Distribution (Figure 5.14/1).

ATTRIBUTES

- Lower Limit–lower end of the interval with non-zero PDF.
- Mode–location of the peak of the PDF.
- Upper Limit–upper end of the interval with non-zero PDF.

The attributes must satisfy the following constraints:

- Lower Limit ≤ Mode,
- Mode ≤ Upper Limit, and
- Lower Limit < Upper Limit.

OPERATIONS

The only operations are those inherited from the Probability Distribution (see Section 5.15). The Mathematica representation of a Uniform Distribution is shown in Table 5.14/1.

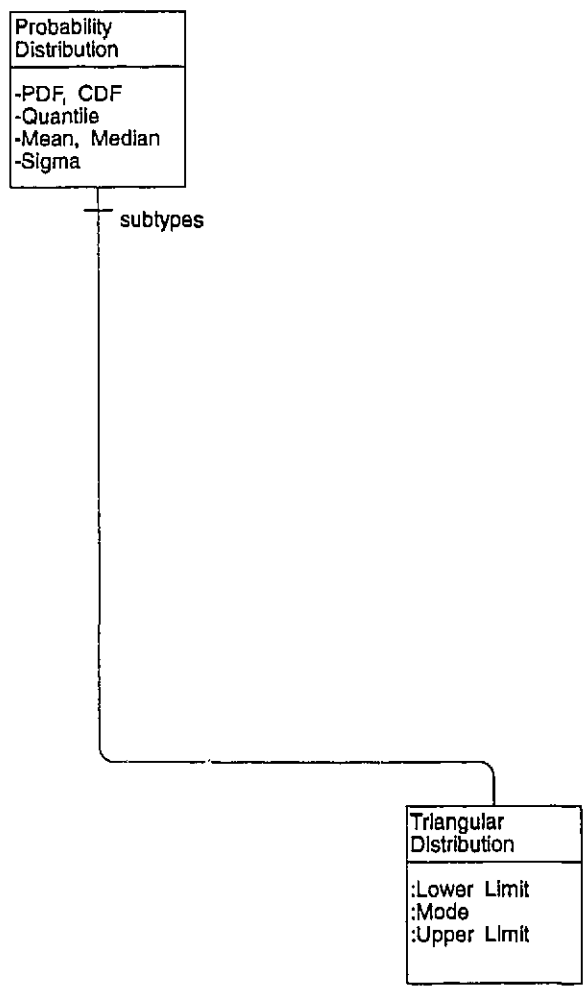

FIGURE 5.14/1: Triangular Distribution as a Subtype of Probability Distribution

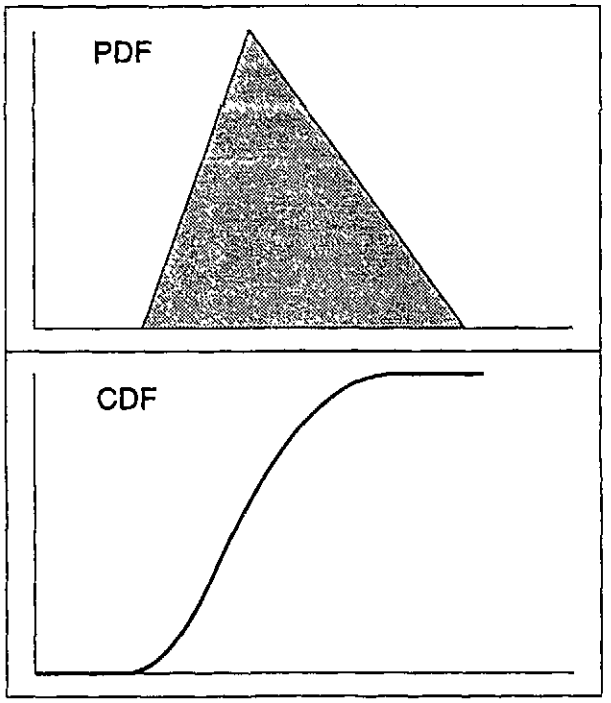

FIGURE 5.14/2: PDF and CDF for a Triangular Distribution

TABLE 5.14/1
OPERATIONS ON A TRIANGULAR DISTRIBUTION

```
TriDis[low_?NumberQ, mode_?NumberQ, high_?NumberQ] /; (low<=mode) &&
(mode<=high) && (low<high)  :=  ProbDis[TriDis, low, mode, high]
```
Triangular Distribution as a subtype of Probability Distribution

```
CDF[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), x_?NumberQ]
```
evaluate cumulative distribution function at *x*

```
mean[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ]
```
compute average value

```
median[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ]
```
compute 50^{th} percentile

```
PDF[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), x_?NumberQ]
```
evaluate probability density function at *x*

```
quantile[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), p_?ProbabilityQ]
```
invert cumulative distribution function at probability *p*

```
Sigma[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ]
```
compute standard deviation

5.15 Triangular Distribution Operations

Mathematica does not provide operations for a Triangular Distribution. They are simple, however, and the formulas are given in this section, both mathematically and in Mathematica notation.

Mathematically, a triangular distribution is defined over the interval from the Lower Limit a to the Upper Limit c. The mode is located at a point b between a and c. The PDF $f(x)$ is continuous; it goes to 0 at a and c and takes the value 0 for all points below a and greater than c. The value of the PDF at b is determined by the fact that the area under the PDF (i.e., the area of a triangle) must be 1; therefore $f(b) = 2/(c-a)$. These constraints define the PDF as a piecewise linear function:

$$f(x) = \frac{2(x-a)[U(x-a)-U(x-b)]}{(b-a)(c-a)} + \frac{2(c-x)[U(x-b)-U(x-c)]}{(c-b)(c-a)} \quad a < b < c, \qquad (5.15\text{-}1)$$

where $U(x)$ is the Heaviside step function, and $U(x-a)$-$U(x-b)$ is a top-hat function with the value 1 between a and b, and 0 elsewhere. In the limiting cases, where $b = a$ or $b = c$, the PDF is discontinuous, and it equals the limiting form of Equation (5.15-1) as b approaches an extreme value.

The CDF of the triangular distribution is a piecewise quadratic function that has the value 0 up to a, and the value 1 for $x > c$. By considering the areas of triangles, one can show that $F(x)$ takes the value $(b-a)/(c-a)$ at b. It can readily be found by integrating Equation (5.15-1) that

$$F(x) = \frac{(x-a)^2[U(x-a)-U(x-b)]}{(b-a)(c-a)} + U(x-b) - \frac{(c-x)^2[U(x-b)-U(x-c)]}{(c-b)(c-a)} . \qquad (5.15\text{-}2)$$

The mean value of a variate X with a triangular distribution is given by a very simple expression:

$$\mathrm{E}(X) = \frac{1}{3}(a + b + c) . \qquad (5.15\text{-}3)$$

The median, in contrast, is more complicated and is best evaluated by inverting the CDF above at 0.5.

The variance also has a simple and symmetric expression:

$$\mathrm{var}(x) = \frac{a^2 + b^2 + c^2 - ab - ac - bc}{18} \quad . \tag{5.15-4}$$

The Mathematica version of these expressions is given in Table 5.15/1.

TABLE 5.15/1
TRIANGULAR DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), x_?NumberQ] :=
   Which[
      x < low,     0,
      x > high,    1,
      x < mode,    (x-low)^2  / ((high-low) (mode-low)),
      x == mode,   (mode-low) / (high-low),
      x > mode,    1 - (high-x)^2 / ((high-low) (high-mode)) ]
```

mean

```
mean[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ] :=
   (low + mode + high) / 3
```

median

```
median[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ] :=
   quantile[d, 0.5]
```

PDF

```
PDF[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), x_?NumberQ] :=
   Which[
      x < low,     0,
      x > high,    0,
      x < mode,    2 (x-low) / ((mode-low) (high-low)),
      x == mode,   2 / (high-low),
      x > mode,    2 (high-x) / ((high-mode) (high-low)) ]
```

quantile

```
quantile[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high), p_?ProbabilityQ] :=
   Which[
      p <= 0,                      low,
      p <= (mode-low)/(high-low),  low + Sqrt[p (high-low) (mode-low)],
      p <  high,                   high - Sqrt[(1-p) (high-low) (high-mode)],
      True,                        high ]
```

Sigma

```
Sigma[d:ProbDis[TriDis, low_?NumberQ, mode_?NumberQ, high_?NumberQ] /;
(low<=mode) && (mode<=high) && (low<high) ] :=
   Sqrt[ (low^2 + mode^2 + high^2 - low mode - low high - mode high) / 18 ]
```

5.16 Probability Distribution: Uniform Distribution

Figure 5.16/1, an excerpt from Figure 4.8/1, shows the object type Uniform Distribution as a subtype of Probability Distribution. A Uniform Distribution has a domain stretching over a finite interval. A Uniform Distribution has a symmetric PDF in which every location between the Lower Limit and the Upper Limit has the same value (Figure 5.16/2). The Uniform Distribution is appropriate where a value is completely unknown apart from sharp upper and lower bounds.

ASSOCIATIONS

- Mixture of–Each bin in the PDF of a Piecewise Uniform Distribution represents a Uniform Distribution (Figure 5.16/1); the set of Uniform Distributions forms an ordered sequence associated with the integers from 1 to J.
- Subtypes–TheUniformDistribution is a subtype of Probability Distribution (Figure 5.16/1).

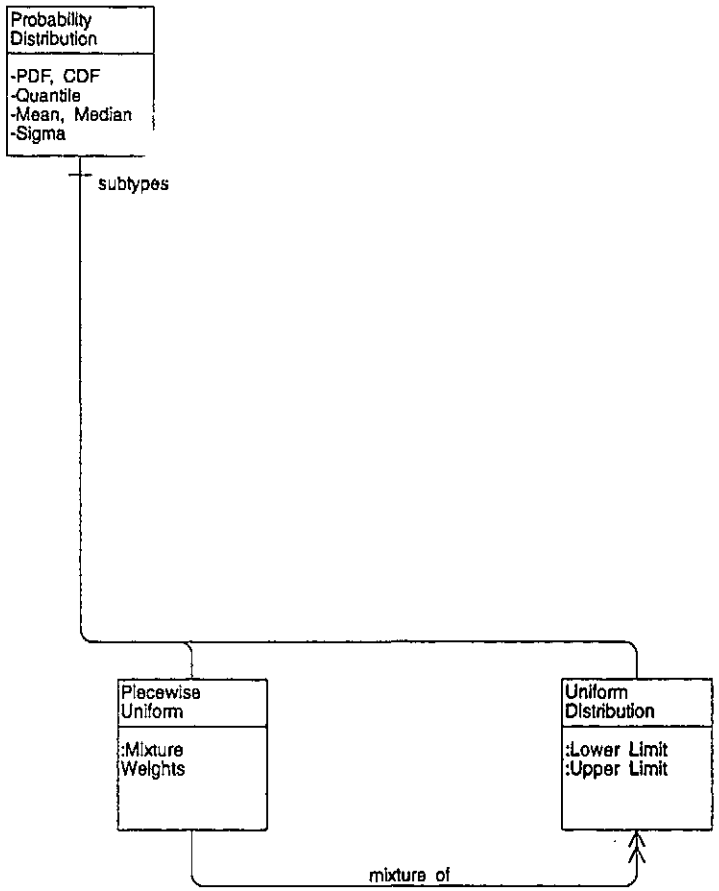

FIGURE 5.16/1: Uniform Distribution as a Subtype of Probability Distribution

ATTRIBUTES

- Lower Limit–lower end of the interval with non-zero PDF.
- Upper Limit–upper end of the interval with non-zero PDF.

The attributes must satisfy the following constraint:

- Lower Limit ≤ Upper Limit.

OPERATIONS

The only operations are those inherited from Probability Distribution (see Section 5.17). The Mathematica representation of a Uniform Distribution is shown in Table 5.16/1.

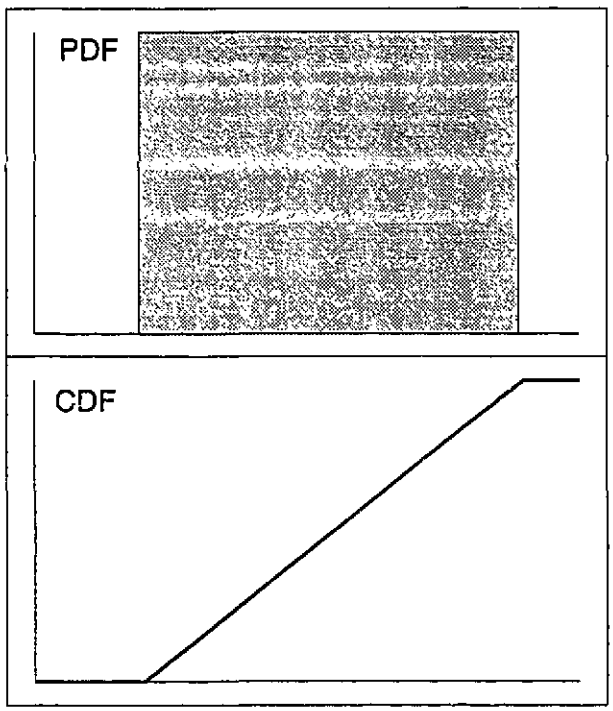

FIGURE 5.16/2: PDF and CDF for a Uniform Distribution

TABLE 5.16/1

OPERATIONS ON A UNIFORM DISTRIBUTION

`UnifDis[low_?NumberQ, high_?NumberQ] /; low <= high := ProbDis[UnifDis, low, high]`	Uniform Distribution as a subtype of Probability Distribution
`CDF[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high, x_?NumberQ]`	evaluate cumulative distribution function at *x*
`mean[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ]`	compute average value
`median[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ]`	compute 50th percentile
`PDF[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high, x_?NumberQ]`	evaluate probability density function at *x*
`quantile[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high, p_?ProbabilityQ]`	invert cumulative distribution function at probability *p*
`Sigma[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ]`	compute standard deviation

5.17 Uniform Distribution Operations

Mathematica provides operations for a UniformDistribution[min,max] object, where min and max represent the Lower Limit and Upper Limit. The expressions to be evaluated are all simple, and so they are shown here algebraically as well.

The PDF of Uniform Distribution is constant over the domain of the distribution, and zero outside that domain. The value of the PDF is determined by the requirement that it integrate to one. The PDF $f(x)$ for a Uniform variate X is

$$f(x) = \begin{cases} \delta(x - X_L) & \text{if } X_L = X_H \\ \dfrac{U(x - X_L) - U(x - X_H)}{X_H - X_L} & \text{otherwise} \end{cases} \tag{5.17-1}$$

where X_L and X_H represent the Lower Limit and Upper Limit respectively; $U(x)$ is the Heaviside step function that takes the value one for $x > 0$ and the value zero otherwise; and $\delta(x)$ is the Dirac delta function that represents an infinitesimally narrow spike with unit area.

The CDF is the integral of Equation (5.17-1):

$$F(x) = \begin{cases} U(x - X_H) & \text{if } X_L = X_H \\ U(x - X_H) + [U(x - X_L) - U(x - X_H)]\dfrac{x - X_L}{X_H - X_L} & \text{otherwise} \end{cases} \qquad 1 \le j \le J. \tag{5.17-2}$$

The mean of a Uniform Distribution is the midpoint of the domain, by symmetry:

$$\mathrm{E}(X) = \frac{1}{2}(X_L + X_H) \quad . \tag{5.17-3}$$

The median is the same as the mean, again by symmetry.

The standard deviation of a Uniform Distribution is proportional to the width of the interval that defines its domain.

$$\sigma(X) = \frac{1}{\sqrt{12}}(X_H - X_L) \quad . \tag{5.17-4}$$

The following table uses these explicit definitions instead of the built-in Mathematica routines so that it is clear what transformation is taking place.

TABLE 5.17/1

UNIFORM DISTRIBUTION OPERATIONS

CDF

```
CDF[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high,
x_?NumberQ] :=
   Which[
      low == high,    UnitStep[x-low],
      x < low,        0,
      x > high,       1,
      True,           (x-low) / (high-low) ]
```

mean

```
mean[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ] :=
   (low + high) / 2
```

median

```
median[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ] :=
   (low + high) / 2
```

PDF

```
PDF[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high,
x_?NumberQ]  :=
   Which[
      low == high,    DiracDelta[x-low],
      x < low,        0,
      x >= high,      0,
      True,           1 / (high-low) ]
```

quantile

```
quantile[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high,
p_?ProbabilityQ] :=
   low + p (high-low)
```

Sigma

```
Sigma[d:ProbDis[UnifDis, low_?NumberQ, high_?NumberQ] /; low <= high ] :=
   (high - low) / Sqrt[12]
```

6 PSEUDORANDOM NUMBER GENERATORS

6.1 Role of the Pseudorandom Number Generator

Figure 6.1/1 shows the relationship between the SYVAC3 objects Pseudorandom Generator and Parameter Distribution. During SYVAC3 simulations, a Sampling Method maintains a slot for each Case-Sampled Parameter assigned to it. If the Sampling Method implements random sampling, it uses a Pseudorandom Generator as a source of random numbers uniformly distributed between zero and one. In each simulation, each Case Sampled Parameter gets such a number. SYVAC3 converts the value between zero and one to a suitable Variable Value by invoking the operations of the appropriate Parameter Distribution. In non-SYVAC3 applications, a similar relationship is established between a Pseudorandom Generator and a Parameter Distribution: the former generates uniform values between zero and one that are converted to quantiles of the Parameter Distribution. The names of the other object classes involved may be different outside of SYVAC3.

Section 2.2 showed how SYVAC3 converts uniformly distributed random numbers between zero and one (hereafter called *standard random numbers*) to Variable Values from any Parameter Distribution by inverting the cumulative distribution function (CDF). Figure 2.2/1 is reproduced here as Figure 6.1/2. There are many other ways of generating Variable Values from particular distributions (Knuth 1969), but the CDF inversion technique has many advantages:

- A predictable number of standard random numbers. We know in advance of any sampling that each variable will require precisely one standard random number for

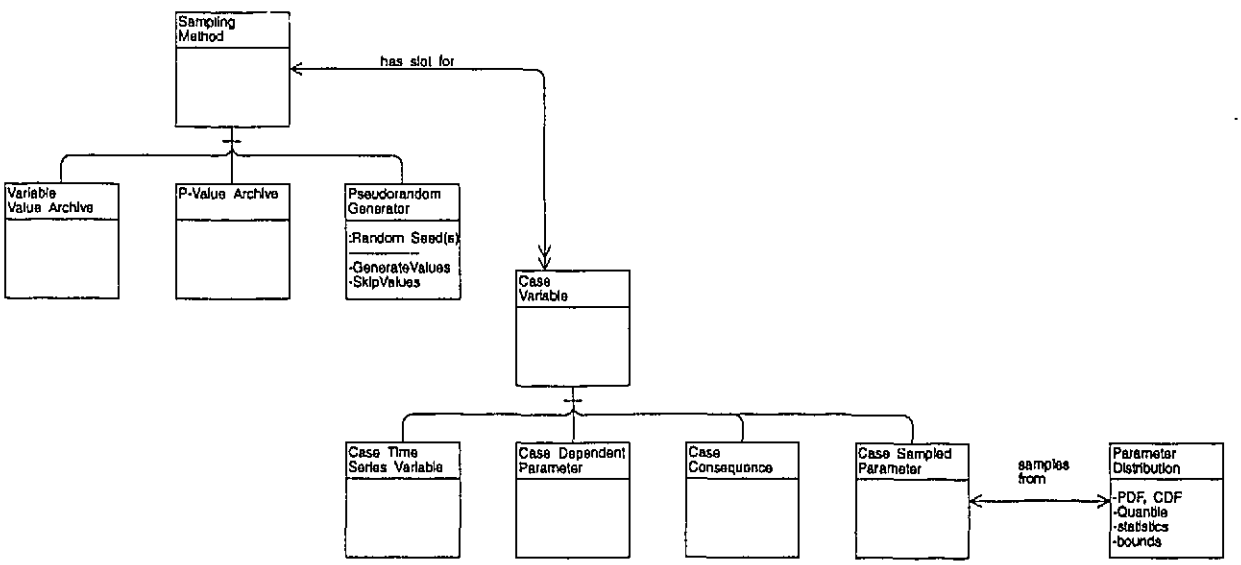

FIGURE 6.1/1: Association between a Pseudorandom Generator and Parameter Distributions in SYVAC3

every Variable Value to be generated. As a result, we know how many standard random numbers are needed. We can say with certainty which pseudorandom number is associated with which Variable Value.

- Monotonic relationship. If two standard random values X_1 and X_2 are used to generate two Variable Values V_1 and V_2 respectively, then $V_1 \leq V_2$ if and only if $X_1 \leq X_2$. This relationship simplifies implementation of sophisticated sampling methods other than simple random sampling. For example, to get extreme values for any Case Sampled Parameter, whatever its distribution, it is only necessary to assign it standard values near zero or one.

- Addition of new distribution types. The CDF for a distribution must be inverted before a sampling procedure for that new distribution type can be developed. This inversion is straightforward and can be performed without reference to other elements in this report. It is often easier to evaluate a CDF than to invert it, and in such cases a general approach to CDF inversion works quite well (see Section 7.7).

SYVAC3 can assign Case Sampled Parameters to different Sampling Methods, each one associated with its own Pseudorandom Generator. Different objects of the Pseudorandom Generator object type behave similarly—each one generates a sequence of standard "random" numbers. They differ in the sequences that they generate. In general, the values produced by one generator should be statistically independent of the values produced by any other generator.

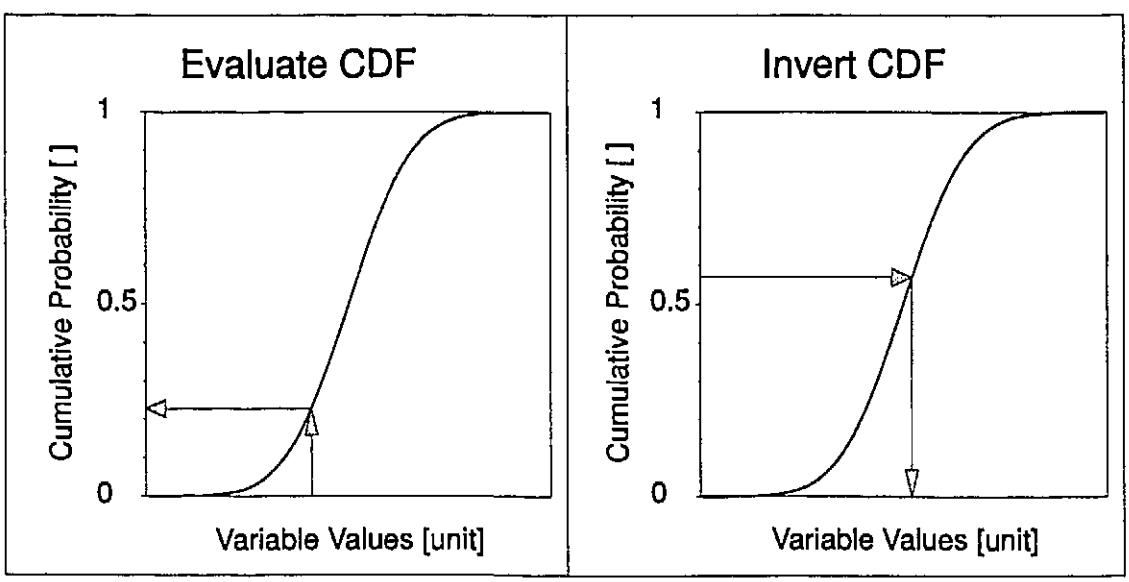

FIGURE 6.1/2: Transformations Through the CDF

6.2 Requirements for a Pseudorandom Generator

Each Pseudorandom Generator object is associated with a long sequence of standard "random" numbers (Figure 6.2/1). This object type has three attributes: the Initial Random Seed(s) that define the sequence, the Current Position in the sequence, and the Sequence Length. This object type has two main operations: Generate Value and Skip Values. The first releases a single standard "random" number from the sequence and advances the Current Position by one. The second (optional) operation advances the Current Position in the sequence by a specified number of places.

ASSOCIATIONS

There are no associations internally within the Parameter Sampling Package (PSP) between the Pseudorandom Generator object type and other object types. SYVAC3 does establish associations between the Pseudorandom Generator object type and other SYVAC3 object types, as was shown in Figure 6.1/1.

ATTRIBUTES

- Initial Random Seed(s)–Different instances of the Pseudorandom Generator object type are virtually indistinguishable in their performance; they all generate sequences of standard "random" numbers. They can be distinguished, however, by the Initial Random Seed(s) used to initialize them. These are one or more integer values. Since most common Pseudorandom Generators use a linear congruential or multiplicative congruential algorithm, an Initial Random Seed typically plays a functional role in the performance of the generator by providing a number on which the algorithm works. But this functional role is not essential. Instead, the Initial Random Seed(s) can be thought of as a set of integer "keys" that identify a specific pseudorandom sequence in the space of all possible sequences.

- Current Position–When a Pseudorandom Generator is created, the Initial Random Seed(s) uniquely identifies an underlying pseudorandom sequence of standard values. By convention, a generator will start releasing values from the beginning of this sequence. After some number have been generated or skipped, the generator must remember where it is in the sequence; that location is its Current Position. It may be stored in a coded form. For example, the Current Position for a multiplicative congruential generator may actually be the current random seed the generator will use in its next iteration.

- Sequence Length–This number describes the length of the underlying pseudorandom sequence of standard values released one by one by the generator. It may be a hard limit as in circumstances where the underlying algorithm can generate only a finite number of values. It may also be an artificial limit. For example, a single long

pseudorandom sequence may be broken up into shorter subsequences to create multiple independent Pseudorandom Generator objects. Then the Sequence Length would define the number of values that can be generated without overlapping those released by another Pseudorandom Generator object.

OPERATIONS

- Generate Value releases the next standard value (i.e., a number between zero and one) in the underlying pseudorandom sequence, and advances the Current Position to the next item in the sequence.

- Skip Values (optional) discards a specified number of standard values from the underlying pseudorandom sequence, and advances the Current Position accordingly. This operation is optional because its function can be achieved by invoking Generate Value the requisite number of times and discarding the standard values released by these invocations. The designer of an implementation may choose to implement this operation for efficiency reasons, especially where hundreds of thousands or millions of values are commonly skipped.

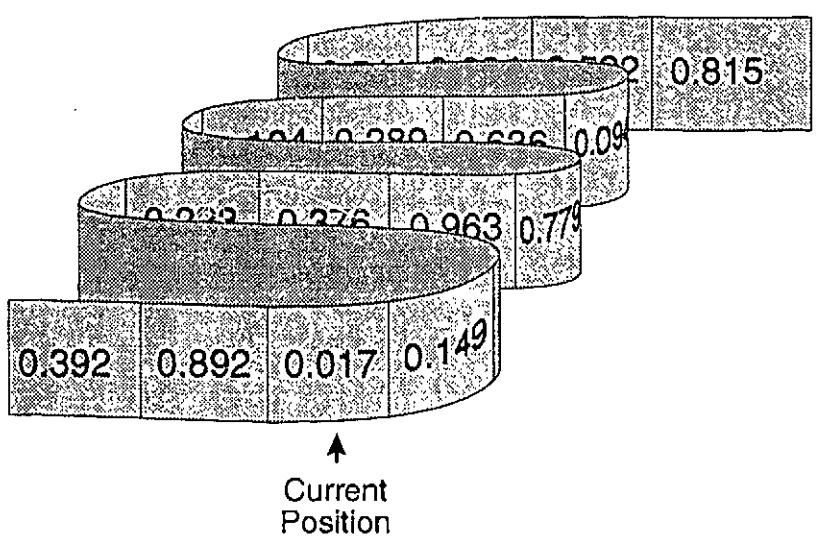

FIGURE 6.2/1: A Pseudorandom Generator is Based on a Sequence of Standard Random Numbers

6.3 Quality of a Pseudorandom Sequence

A truly random sequence of numbers uniformly distributed between zero and one is characterized by unpredictability—knowing the values of some entries in the sequence does not help in predicting other entries. Pseudorandom sequences, in contrast, are generated repeatably by a deterministic algorithm. Nevertheless, a high-quality pseudorandom sequence appears to be random when statistical tests are applied. A sequence used for a Pseudorandom Generator is required to have a long period, a large set of possible outcomes, uniform probabilities, and documented performance against statistical tests.

What is a random number? Is 0.6931472783 a random number? These questions are difficult to answer because they raise deep philosophical issues that have not yet been resolved to everyone's satisfaction.

Richard von Mises (1919) introduced the concept of the "irregular collective," which forms the basis of some frequentist theories of probability. Durand (1971) describes Mises' collective as ". . . an infinite sequence E, say of elements e_1, e_2, e_3, ..., each of which bears one of a set of labels E_1, E_2, ..., E_k representing events. In addition, if E is to pass as a collective, it must meet two requirements, as follows: first, that the relative frequency n_i/n of the event E_i in the first n elements e_1, e_2, ..., e_n shall approach a limit as n approaches infinity; second, that the arrangement of events within E shall be random in terms of criteria developed by Mises. Assuming both requirements are met, the limiting relative frequency, say $\lim(n_i/n) = p_i$, is the probability of the event E_i within the collective E."

In this context, Durand (1971) reports a definition of relative randomness: "An infinite sequence E in which the labels (events) E_1, E_2, ..., E_k recur with relative frequencies $Rf(E_i) = n_i/n$ that approach limits as n approaches infinity will be regarded as random with respect to a systematic rule S for selecting an infinite subsequence $E|S$ if the relative frequencies in $E|S$ approach limits equal to those in E—that is, if $\lim Rf(E_i|S) = \lim Rf(E_i)$."

These definitions introduce several ideas associated with randomness:

- Infinity required—Randomness in an infinite sequence can be defined, but randomness in a finite sequence is an elusive concept. The number 0.6931472783 quoted above might very well be random in an infinite sequence, but taken by itself it is not.

- Finite set of outcomes—While the sequence of events should be infinite, the number k of possible outcomes in a sequence can be analyzed most easily when finite.

- Nonpredictability—If you can find a selection rule S that forms a subsequence $E|S$ of E in which the probabilities of the possible outcomes E_1, E_2, ..., E_k differ from those in E as a whole, then you have found nonrandomness. Randomness is associated with having no such predictable pattern.

In contrast to the foregoing, a pseudorandom sequence is a finite sequence R of elements r_1, r_2, ... r_N, each of which bears a label from E_1, E_2, ... E_k. The elements of the sequence are generated by a deterministic algorithm, so that the same sequence can be generated at different times or by different people or computer programs. Nevertheless, such a sequence superficially appears to be random in the sense given above. The probability p_i of each event E_i is defined to be n_i/N. The following criteria can be used to compare one pseudorandom sequence with another for use in simulation:

- Long sequence—The larger N is, the better. On an absolute scale, $N > 10^9$ is preferred to allow some room for expansion, since some applications of SYVAC3 have already used 50 000 simulations with about 4000 parameters each, for a total of about 2×10^8. Much larger values of N are desirable. Note that when practical (as opposed to theoretical) statistical tests are applied to a sequence, it matters little whether the sequence is actually infinite, or just very long, since only a finite number of entries are tested.

- Uniformity—A sequence can be considered random whatever probability p_i is assigned to the selection of label E_i, but the role of a Pseudorandom Generator described in Section 6.1 requires uniformly distributed variates between zero and one. Consequently, the closer each p_i is to $1/k$, the better. We can in fact require that $p_i = 1/k$, since this goal is commonly achieved.

- Large set of outcomes—The larger k is, the better. The labels E_1, E_2, ... E_k are mapped onto rational numbers between zero and one. Applications of SYVAC3 are typically concerned with probabilities of events on the order of 10^{-6}. The value of k should be at least 10^8 to make the p_j's small with respect to 10^{-6}.

- "Randomness"—A pseudorandom sequence should be free of obvious patterns. If an observer can construct a selection rule S such that probabilities of outcomes in $R|S$ differ appreciably from $1/k$, then the randomness of the sequence is compromised. Of course, it is always possible in a deterministic sequence that a selection rule based on the generation algorithm could select a nonrandom subsequence. But selection rules unrelated to the generation algorithm should be random with respect to R. Knuth (1969) proposed a series of statistical tests that check for nonrandom behaviour. In addition, several subsequent authors have documented their tests of well-known pseudorandom number generators (e.g., Park and Miller 1988, Fishman and Moore 1983), and a sequence that fails one of these tests is considered suspect.

7 PARAMETER SAMPLING PACKAGE: MAJOR DESIGN DECISIONS

7.1 Design Goals Affecting the Parameter Sampling Package (PSP)

A variety of package designs could achieve the requirements for the PSP expressed so far. The actual design achieves design goals that include interoperability with SYVAC3, portability between platforms and applications, accurate results expressed with adequate precision, and efficiency of operation.

INTEROPERABILITY WITH SYVAC3

While the PSP has a variety of uses, typically it provides services to SYVAC3. The same people who developed the PSP also developed the rest of SYVAC3 in the same software environment. Interoperability with SYVAC3 was a key goal for design of the PSP. This goal affected the following aspects of the design:

- Language–Like the rest of SYVAC3, the PSP used Fortran 77 for source code, so that its development could use the same hardware, software, procedures and staff as the rest of SYVAC3.

- Software Library–The PSP took the form of a library of subroutines so that it could link with SYVAC3 and a model, allowing the entire system to execute as one program.

- Data Interface–SYVAC3 maintains descriptions of all the Parameter Distributions needed for a model. The PSP uses SYVAC3's data structures as an interface with SYVAC3 (see Section 3.4).

- Error Handling–SYVAC3 conducts many simulations in a single invocation. When it detects errors, SYVAC3 puts the current simulation on hold to await later examination, and continues with subsequent simulations. The PSP must therefore trap errors and report them without stopping.

PORTABILITY

While the PSP typically functions as part of SYVAC3, it can also operate independently of SYVAC3. It must be portable from one application to another, and not depend too much on SYVAC3 services. Furthermore, SYVAC3 itself operates on several hardware platforms, among them Digital Equipment Corporation's VAX computer and the IBM PC. The PSP must therefore share SYVAC3's requirements for portability. The goal of portability affected several parts of the design:

- Language–Fortran 77 is available universally, with only minor variations between platforms. Its use promotes portability. To maintain this advantage, SYVAC3 developers used only a few well-documented extensions to this standard.
- Data Interface–As much as possible, the data interface between the PSP and calling code consists only of the essentials, specifically a set of arrays required to transmit

details of Parameter Distributions. Any program that can set up these arrays can call the PSP.

- Error Handling—When the PSP detects error conditions, it uses the RETMES routine (see Section 3.6) to obtain the minimum information needed from the host program to report errors. The PSP does not need information about the hardware or software set-up to communicate error reports back to the user.

ACCURACY AND PRECISION

The design of the PSP supports sampling of parameter values for simulations. The design goal was to meet the quality requirements for pseudorandom sequences already stated in Section 6.3:

- Long Sequences—Pseudorandom Generators provided by the PSP must generate pseudorandom sequences with at least 10^9 entries. Since most algorithms for generating pseudorandom sequences operate mostly on integers, this requirement implies the need for at least 32-bit integer arithmetic (see Section 6.5).

- Probability Precision—To resolve probabilities of events down to 10^{-6} (one in a million), a goal was set to make probability-related calculations maintain accuracy to at least eight significant figures. Since 32-bit integers cover the range from -2147483648 to 2147483647, they meet the precision part of that requirement. Hence 32-bit integer arithmetic is adequate for the Pseudorandom Generators. In contrast, the precision possible with floating-point arithmetic on 32-bit computers is limited to about seven significant figures. All floating-point calculations on such computers must use "double precision" arithmetic to meet the precision part of the goal. To ensure accuracy as well as precision, all cumulative distribution function (CDF) and inverse CDF calculations must calculate results that maintain an accuracy of at least eight significant figures.

EFFICIENCY

The final design goal of the PSP was efficiency. When used with a model having N parameters, a single simulation requires N invocations of a Pseudorandom Generator, and N CDF inversions. When N is large, say 4000, these calculations can be time-consuming. Meeting an efficiency goal particularly affected the design of CDF evaluation and inversion code. While many algorithms to evaluate a CDF may work reliably over a large domain, some are faster than others in different parts of the domain. Several CDF routines in the PSP contain code for a variety of algorithms. The argument values determine at runtime which algorithm to invoke.

Several other design goals, including simplicity, reliability, maintainability, correctness, testability, useability and completeness, played a role in constraining the design of the PSP. The following sections discuss the implications of all these goals for the design of the PSP.

7.2 The Choice of Fortran 77

For compatibility with SYVAC3, developers used Fortran 77 to program the Parameter Sampling Package (PSP). Doing so promoted portability and ensured compatibility with available skills and computing equipment. Using Fortran 77 constrained the implementation of the object-oriented features present in the specification. Programming standards filled the gap, providing a means of implementing an object-oriented design in Fortran 77.

The team that developed SYVAC3 was working on a project to assess the environmental impacts of nuclear fuel waste disposal. They adopted the Fortran 77 language for SYVAC3, since that was the standard programming language in that project. As part of SYVAC3, the PSP also used Fortran 77.

There were both advantages and disadvantages to the choice of Fortran 77, as shown in Table 7.2/1. It greatly affected the design of the PSP, particularly with respect to the data structures used.

To illustrate the impact, consider what would have happened if the developers had chosen other languages. Today, many application developers use the language C++, which supports object-oriented programming. C++ was not widely available in 1985 when SYVAC3 was first designed. If it had been, developers could have mapped the object types described in the specification directly onto object classes in the code. They would map operations in the specifications onto methods in the code. Specifications and code would have been very similar in structure.

Pascal and C were widely available in 1985. If developers had chosen one of these languages, they could have mapped the object types onto structured data types in the code. They would implement operations as procedure calls. The gap between specification and code would be larger, but the similarities would still be clear.

Fortran 77 has no object classes and no structured data types. There are still a variety of ways to implement the data storage, as discussed in Section 7.3. But there cannot be a direct mapping from the object type in the specification to a single construct in the code. Here a limitation of Fortran 77 clearly constrains the design of the code.

Another limitation of Fortran 77 affects the choice of algorithms. Since Fortran 77 does not support recursion, it was not possible to use recursion in any of the CDF and inverse CDF routines. Where a recursive algorithm was used, as in the log $\Gamma(x)$ function GAMMAL, it had to be explicitly converted to an iterative algorithm.

One potential benefit of using Fortran 77 on the design of the PSP was the availability of a variety of mathematical and statistical code libraries. For example, it would have been easy and quick to develop the PSP by using commercial statistical libraries for CDFs and inverse CDFs of beta, normal, and other distributions. Instead the development team created new routines so that they would be more portable. That decision allowed AECL to sell SYVAC3

TABLE 7.2/1

ADVANTAGES AND DISADVANTAGES TO THE SELECTION OF Fortran 77 FOR SYVAC3

ADVANTAGES	
1.	Available staff was familiar with Fortran 77.
2.	Software tools and libraries were available for Fortran 77.
3.	Fortran 77 produced efficient code that executed fast on available computing hardware.
4	Fortran 77 compilers were available for all available computers.
5.	Programs written to conform to the Fortran 77 ANSI standard would run with little modification on all available computers.
6.	Fortran 77 had features (e.g., complex number support) to ease scientific programming.
DISADVANTAGES	
1.	Fortran 77 did not support many constructs of structured programming (e.g., WHILE loops).
2.	Fortran 77 did not support user-defined object or data types, apart from arrays.
3.	Fortran 77 did not support any of the characteristics of object-oriented programming (inheritance, polymorphism, operator overloading)
4.	Fortran 77 had primitive interactive input/output operations that did not lend themselves to user-friendly interfaces.
5.	Fortran 77 had primitive file-accessing operations that did not lend themselves to database management.
6.	Fortran 77 did not support dynamic allocation of memory.
7.	Fortran 77 had a complex syntax that did not lend itself easily to creation of tools to manage software.
8.	Fortran 77 did not support recursion.

to customers using a variety of computers, without getting software licences for code libraries for every computer type. Such licences would be necessary because we distribute SYVAC3 in source code form, so that it can be linked with models the customer develops.

In future implementations it may be possible to distribute SYVAC3 as an executable module that can be linked with a model in other ways. Then the PSP could employ commercial libraries at the cost of a runtime distribution fee. The disadvantage of that approach, of course, is that the developers would need access to every type of computer on which the code is to run in order to develop the executable files.

The choice of Fortran 77 also affected error handling in the PSP. Fortran 77 has no intrinsic error handling, but the PSP can catch errors by explicit checks. It then reports them by printing error messages on user-defined files and devices. It also passes flags back to the calling routines when errors are encountered.

7.3 Design of Data Structures for Parameter Distribution

As Figure 7.3/1 shows, each Parameter Distribution object contains a Truncation Interval object and a Probability Distribution object. In Fortran 77, designers must use array data structures to represent these object types. A 2-element vector can represent a Truncation Interval object. In the majority of cases, a 4-element vector and a character string can represent a Probability Distribution. Conditional Distributions and Piecewise Uniform Distributions require special treatment.

The simple data structures shown in Figure 7.3/1 can represent most types of Parameter Distributions. There are two components to this data structure:

(1) A truncation interval—A 2-element double-precision vector can store upper and lower bounds for either Probability Interval or Value Interval object types. Supporting both subtypes of Truncation Interval as two alternatives would require a way of discriminating between them. Support for the Value Interval subtype is optional, however, as stated in Section 4.7. The current design supports only Probability Interval subtypes, and so the 2-element vector contains all the required data.

(2) A probability distribution—A 4-element double-precision vector can store the attributes for the majority of subtypes. Figure 7.3/1 shows how the attributes fit in this vector. All attributes can be stored as Fortran 77 DOUBLE PRECISION data types (although the INTEGER data type is more appropriate for some, such as the Number of Trials in a Binomial Distribution). A character string indicates clearly the Parameter Distribution subtype to permit correct interpretation of the attributes. Unfortunately, there are some exceptional subtypes of Parameter Distribution that cannot be handled by this simple structure:

- A Parameter Distribution that is also a Conditional Distribution needs an extra field to store a Correlation Coefficient. In addition, the design must preserve the "correlation" association, by retaining a link between the Conditional Distribution and another Parameter Distribution. Section 7.4 addresses Conditional Distributions.

- Each Piecewise Uniform Distribution has several components that are Uniform Distributions. There is no room in the structure shown in Figure 7.3/1 for these distributions. Section 7.5 deals with Piecewise Uniform Distributions.

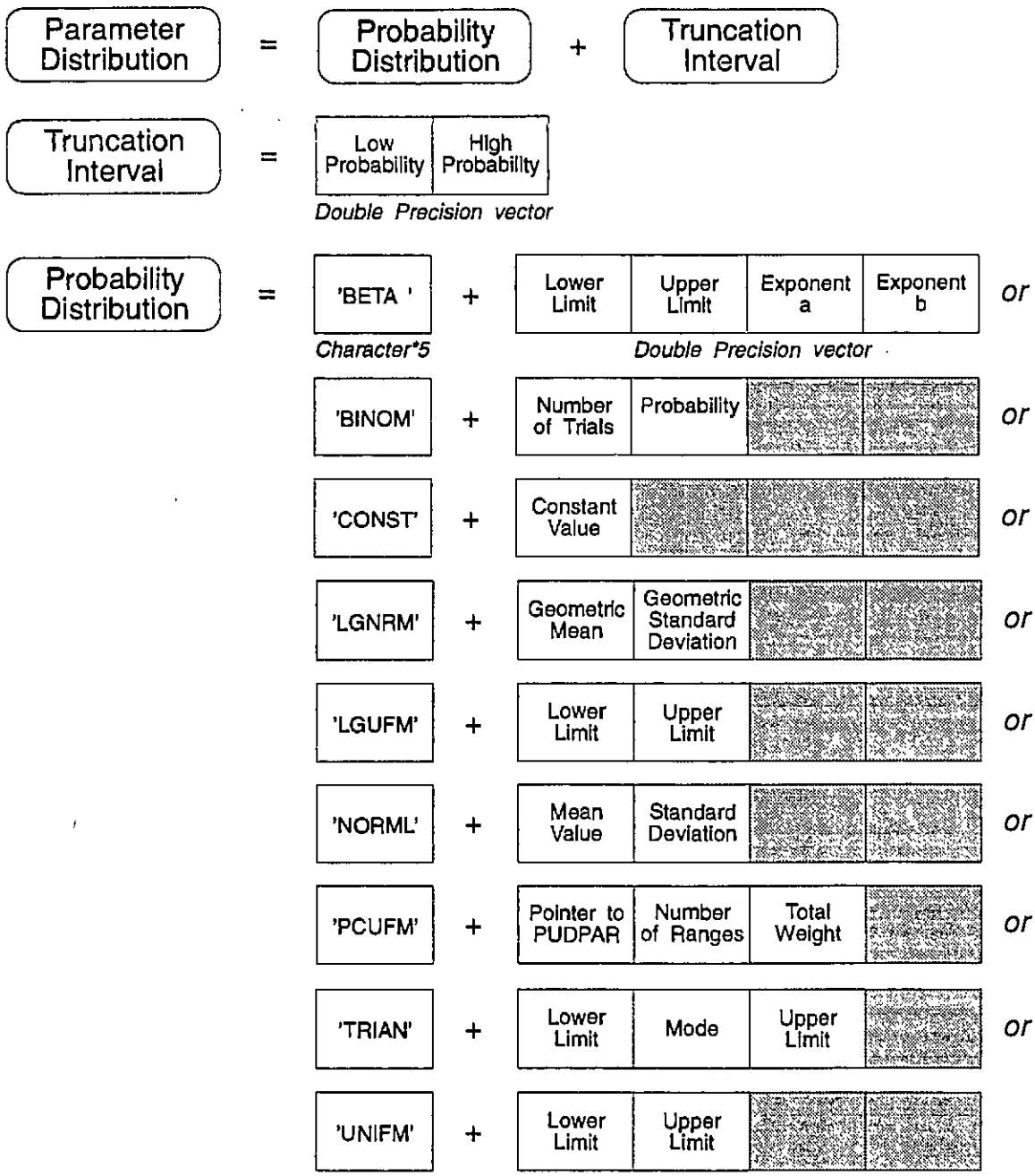

FIGURE 7.3/1: Primary Data Structure of the Parameter Distribution Object Type

7.4 Representation of Conditional Distributions

Section 4.5 points out that a marginal distribution and a Correlation Coefficient uniquely define a Conditional Distribution only when the marginal distributions for both the Conditional Distribution and the independent Parameter Distribution are untruncated Normal Distributions. Lognormal Distributions also have this uniqueness property if we take the correlation with respect to the logarithm of the variable rather than the variable itself. The Parameter Sampling Package (PSP) must support these two cases, according to Section 4.5, and may or may not support other non-unique Conditional Distributions. The SV309 design supports only these two types of Conditional Distribution, fulfilling the essential requirement. In this design, the two new Parameter Distribution subtypes correspond to two new Probability Distribution subtypes called Correlated Normal and Correlated Lognormal Distributions. These object subtypes require minor extensions to the data structure introduced in Section 7.3.

SV309 supports only two types of Conditional Distribution objects. These appear in the design as standard Parameter Distribution objects, with Probability Distribution subtypes that are either Correlated Normal or Correlated Lognormal objects. The "correlation" association establishes a link between each such object and an independent Parameter Distribution that must have a Normal or Lognormal Distribution for its Probability Distribution component. Figure 7.4/1 shows the attributes of the two new Probability Distribution subtypes. The Correlation Coefficient appears as an extra attribute for these subtypes. Implementing the link to another distribution raises important design issues.

Most languages that permit the definition of user-defined types (e.g., C or Pascal) also support pointers to refer to instances of those types. In such a language, the correlation link required for correlated distributions would be implemented as a pointer. However, Fortran 77 does not support pointers. This leaves the designer with two options. The design could store multiple Parameter Distributions in a single set of arrays, so that integer indices could point to individual Parameter Distributions. Or, it could keep a full copy of the independent distribution as part of the correlated distribution. SV309 uses the former alternative.

More specifically, to represent multiple objects of the same object type, the design takes each array (or scalar) required to implement a single Parameter Distribution object, and adds another dimension to it. For example, a single Parameter Distribution object requires a 4-element attribute vector. To store multiple Parameter Distribution objects, it is necessary to provide such a vector for each object. SV309 accomplishes this feat by making the original vector into an array with four rows. Then each Parameter Distribution object gets a single column with four entries in which to store its attributes. By convention, the new dimension is always the last dimension, and so the last index of an array reference identifies the specific object being referenced. This convention promotes efficient data handling on computers with virtual memory, since Fortran 77 stores data in column-major order. In this type of structure, the correlation link in Figure 7.4/1 is an integer pointer that identifies the location of the correlated Parameter Distribution object.

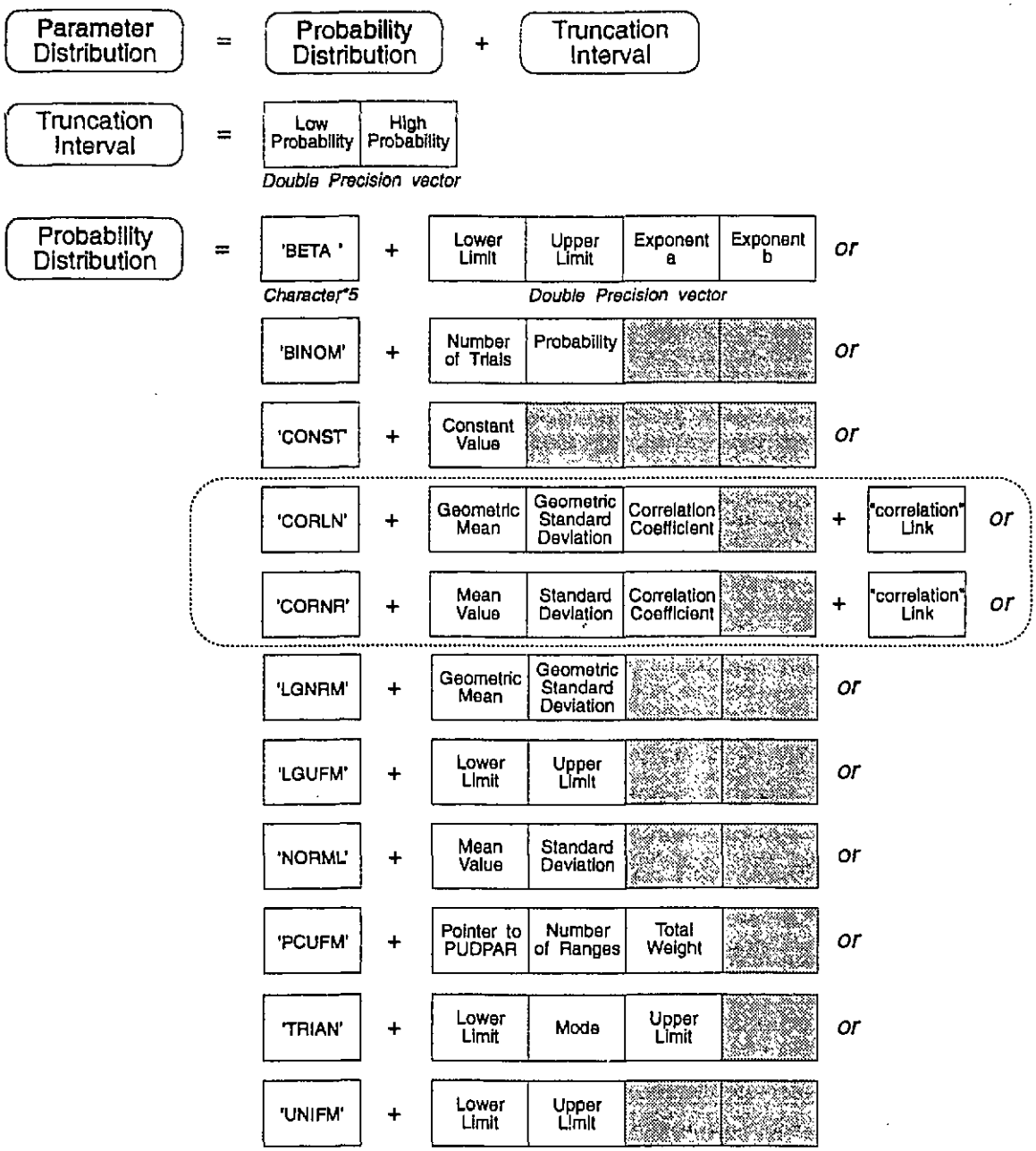

FIGURE 7.4/1: Parameter Distribution Data Structure with Conditional Distributions

7.5 Representation of Piecewise Uniform Distributions

Figure 7.5/1 shows the Parameter Distribution data structure with storage for Piecewise Uniform Distribution objects. The data placed in the auxiliary array PUDPAR describe the Uniform Distribution objects that make up the Piecewise Uniform Distribution object, and the relative weights that apply to each. By placing them in PUDPAR, this design distinguishes between Uniform Distribution objects that stand alone and those that belong to a Piecewise Uniform Distribution.

The designer faces awkward design choices when implementing the Piecewise Uniform Distribution because it is the only Probability Distribution subtype that requires a nondeterministic amount of storage. In a computer language with user-defined types and dynamic allocation of memory, the Uniform Distribution objects that make up a Piecewise Uniform Distribution could be stored as a linked list of dynamically allocated data records linked by pointers. In Fortran 77, they must be stored in an array.

Section 7.4 established the requirement to store multiple Parameter Distribution objects in arrays to permit linking of correlated Parameter Distributions. Uniform Distributions from Piecewise Uniform Distribution objects could be stored as additional Probability Distribution records in these arrays, but this approach faces two criticisms. First, it violates the simplicity of the design in Figure 7.5/1, where there are precisely one Probability Distribution object and one Truncation Interval object for every Parameter Distribution object. If there are extra Uniform Distribution objects in the storage array, the structure would need to be more complex. It would need to distinguish between those entries containing Parameter Distribution objects and those containing extra Probability Distribution objects. Second, the normal storage for a Uniform Distribution provides no place to record the relative weights needed for an association with a Piecewise Uniform Distribution. To avoid these difficulties, the SV309 design uses an auxiliary array called PUDPAR to store Uniform Distribution objects that belong to a Piecewise Uniform Distribution.

Figure 7.5/1 shows the Pointer to PUDPAR as an attribute of a Piecewise Uniform Distribution. The curve connects this pointer to the column of PUDPAR just before the storage for the current distribution. That means that the columns of PUDPAR used for this distribution have indices $P+1$ to $P+N$, where P is the pointer value, and N is the Number of Ranges.

The design has a minor inconsistency. Pointers for Correlated Normal and Correlated Lognormal Distributions have a separate pointer vector, whereas pointers for Piecewise Uniform Distributions reside in attribute storage. This inconsistency comes from the original design. The inconsistency will likely remain since any changes to make the two consistent would break old code calling the Parameter Sampling Package. Inflexibility in changing implementations is a disadvantage of letting the internal data structure become public.

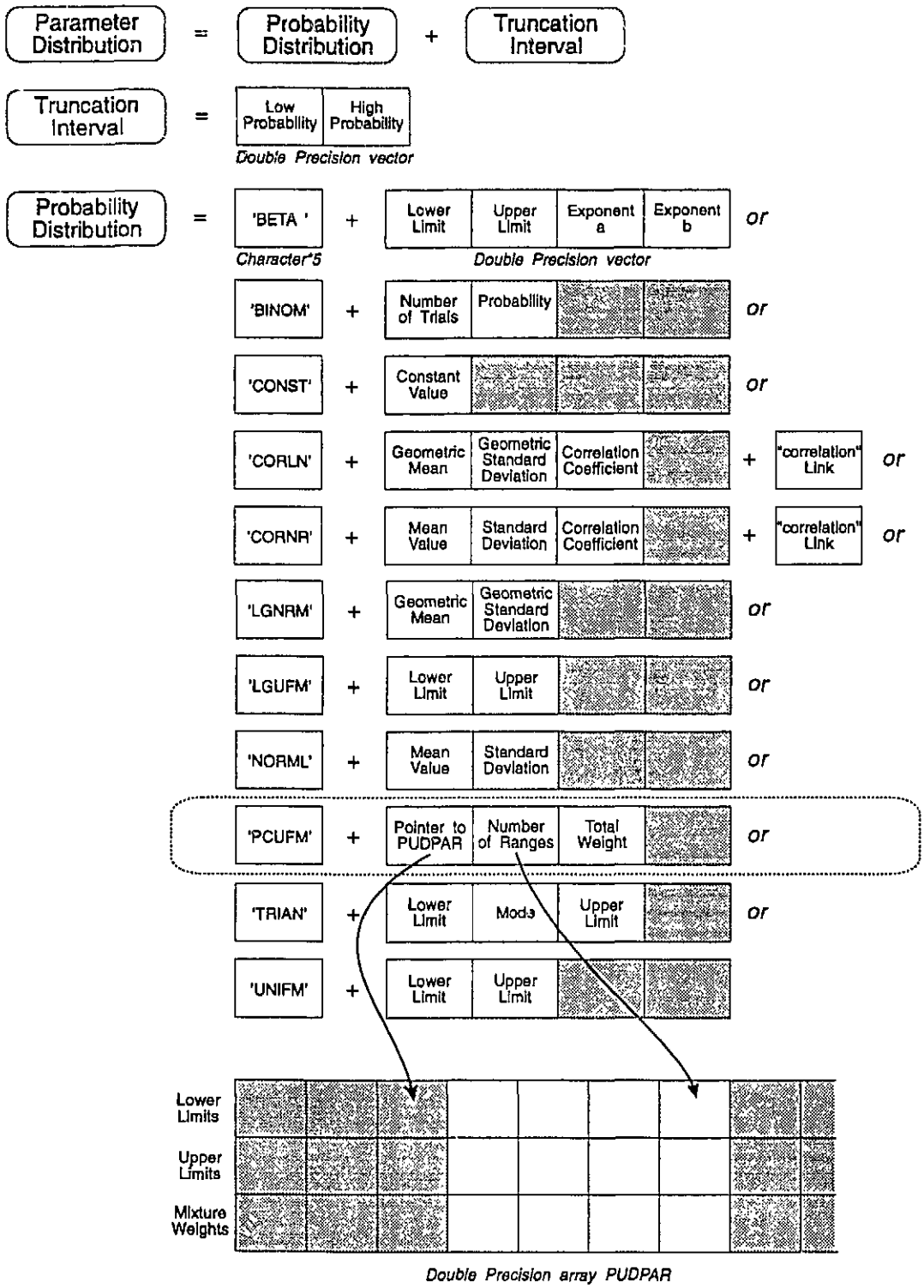

FIGURE 7.5/1: Parameter Distribution Data Structure with Piecewise Uniform Distributions

7.6 Data Storage for Multiple Instances of Parameter Distribution

The preceding sections laid out a set of data structures for Parameter Distribution object type. As those sections pointed out, the Parameter Sampling Package (PSP) extends the data structure arrays with an extra dimension to store multiple objects. This section reviews the data storage alternatives in more detail.

Fortran 77 lacks support for objects, structured data types (except arrays), dynamic allocation of memory, and pointers. Nevertheless, a Fortran 77 program can implement an object-oriented specification. Suppose an instance of an object type requires five named fields. To be specific, suppose these fields are A, B, C, D and E. There are four main ways of implementing instances of such an object type in Fortran 77:

(1) Separate declarations—To generate two instances of the object type, define five discrete variables for each instance. A naming convention that links the five variable names together makes the code clearer. For example, to store the components of an object instance known as OBJ1, call the variables OBJ1A, OBJ1B, OBJ1C, OBJ1D and OBJ1E. A macro preprocessor can generate the declarations for the variables automatically, simplifying the process at the expense of using a nonportable software tool. This approach stretches the variable name space of a program, especially when representing many object instances.

(2) Increase variable rank—Suppose A, B, C and D are scalars, while E is a vector of length four. To implement up to 100 instances of the object type, add another dimension to each variable's rank. For example, A becomes a vector of length 100, and E becomes a 4 x 100 array. Then the set {A(i), B(i), C(i), D(i), E(*,i)} represents the i^{th} object instance, with E(*,i) designating the column consisting of {E(1,i), E(2,i), E(3,i), E(4,i)}. Fortran 77 stores arrays in column order. On a computer with virtual memory, the new dimension should always come at the end so that all fields of an object instance reside close together in memory, hopefully on the same page of virtual memory.

(3) Simulate dynamic allocation—Although Fortran 77 does not support the dynamic allocation of memory for new object instances, the code can simulate dynamic allocation in a long preallocated vector. Suppose V is the storage vector. Let OBJ1 be a variable designating an object instance. An elegant implementation of this mechanism uses statement function subprograms to implement the fields. That is, A(OBJ1) represents the value of the A field for the object OBJ1, and E(3,OBJ1) represents the third entry in the E vector for OBJ1. The code must call special routines to create object instances and to assign values to them. Once created, OBJ1 contains an integer pointer. Statement functions A(N) and E(N,J) have the following definitions:

```
A(N) = V(N+1)
E(J,N) = V(N+5+J) .
```

That means that A(OBJ1) becomes shorthand for V(OBJ1+1). This method can work well if implemented fully. It can be complex, however. Unless the code functions flawlessly, there is a risk of overwriting values, causing mysterious errors. Because CHARACTER values are incompatible with other data types in Fortran, they need a separate array. The SYVAC3 Time Series Package uses a variation on this scheme.

(4) File storage–While Fortran 77 does not support dynamic allocation of variables in memory, it does support dynamic allocation of files. Each object instance could correspond to a file. The file structure could be as complicated as necessary. However, this mechanism suffers from performance problems since reference to a file is much, much slower than access to memory on most machines. S (Becker et al. 1988) is one software system that uses this mechanism. Furthermore it is difficult to standardize, since it depends on the file management capabilities of the underlying operating system.

The PSP uses primarily the second method listed here. To store multiple Parameter Distributions, it extends the rank of a data structure for a single Parameter Distribution. There is one exception. A simple variation of method three maintains the array PUDPAR used to store Uniform Distribution data from Piecewise Uniform Distributions.

Given that rank extension is the main mechanism for storing multiple Parameter Distributions, two other design strategies complete the description of the storage design:

(1) Hidden or open storage–The PSP could have an internal data structure accessible to outside routines only through subroutine calls. This "information cluster" (Page-Jones 1980) approach permits occasional changes to the data structure without affecting calling programs as long as the calling interface remains unchanged. The PSP uses an open data structure instead. SYVAC3 already accesses all the fields stored in the data structure when it reads in the Parameter Distribution descriptions, and so there is no point in hiding them. Another alternative would be to use PSP routines to read Parameter Distribution data, but that choice would create interdependencies between the PSP and the SYVAC3 File Reading Package, limiting the portability of the former.

(2) COMMON block or argument list–The arrays that provide storage for Parameter Distributions can be located in a Fortran 77 COMMON block defined in the PSP, or they can be passed in argument lists when other routines call the PSP. PSP uses the argument list option, for one main reason. If PSP defines a COMMON block, it must declare the maximum number of Parameter Distributions to be stored. If AECL were to distribute the PSP as a compiled library, this declared maximum would constrain the use of the package. When the calling program passes arrays through the argument list, in contrast, it can declare their dimensions, making the PSP more portable.

7.7 Selection of Algorithms

The specifications call for PDF and CDF calculations, along with a variety of statistics, for all Probability Distribution subtypes. However, Section 4.3 clearly states that PDF, mean, median, and Sigma calculations are not essential for a conforming implementation. Current implementations of the Parameter Sampling Package do not support these operations. Section 7.1 identifies the accuracy and performance requirements for those calculations that are performed. The current design uses algorithms that exceed the 8-significant-figure accuracy requirement where possible and practical. It also provides access to extra mathematical functions (e.g., log $\Gamma(x)$, erf(x)) needed for the required calculations, or closely related to the required calculations. Finally, certain generic algorithms, such as inverting a CDF with Newton's method, or scaling standard distributions, appear frequently.

REQUIRED VERSUS OPTIONAL OPERATIONS

The first implementations of the PSP preceded writing of the detailed specifications presented in this document. Informal early designs called for the PSP to be a software library that implemented the minimal set of routines needed explicitly by SYVAC3. The popularity of new object-oriented methods in the early 1990's (particularly books by Coad and Yourdon (1989) and Rumbaugh et al. (1992)) led to a reconsideration of the PSP. The package clearly supported object types, but it did not provide a full range of common operations on these object types. Operations like evaluating the PDF or the standard deviation were missing, even though authors like Stephens et al. (1989) emphasized the importance of these aspects of probability distributions. To rationalize the object-oriented requirements and the actual implementation, this report includes these extra operations in the specifications, but deems them optional requirements.

The Binomial Distribution falls into the same category. It did not appear in early versions of the code, but SV309 contains routines to perform both CDF and inverse CDF operations for this distribution. However, the integration of the Binomial Distribution and the PSP is still not complete. The specifications in this document describe the integration as it should be, but clearly identify this distribution as an optional component of an implementation.

GUIDELINES FOR ALGORITHMS

While the PSP in SV309 does not implement some optional operations, it far exceeds some requirements mentioned in the specifications. For example, Normal Distribution CDF and inverse CDF computations usually provide 16-significant-figure accuracy, rather than the required eight significant figures. Furthermore, the PSP provides apparently unrelated functions such as the error function erf(x). The following guidelines were used to determine what algorithm to use and what routines to implement.

- Strive for machine accuracy—Algorithms should provide a degree of accuracy that matches the machine precision. For example, "double precision" floating-point arithmetic on a 32-bit computer gives 16 (IBM PC) or 17 (DEC VAX) digits of precision. If only half of those digits are correct, there is a risk of unknowing users placing excessive confidence on the results. Furthermore, future revisions to the package will likely require better accuracy. Where possible and where improvements would not degrade performance too much, accurate algorithms were used.

- Maximize the number of useful products—The CDF of a normal distribution has a direct linkage to the error function erf(x) and complementary error function erfc(x) (see Equation (?-

2) in Section 5.11). To implement one function without the other would be wasteful, especially since models used with SYVAC3 commonly employ the error function and complementary error function in solving differential equations for diffusive transport. The PSP includes several such routines, even though they may not be needed by SYVAC3.

GENERIC ALGORITHMS

Several distributions use the following generic algorithms in their CDF or inverse CDF calculations.

- Scaling and shifting to reach standard distribution–Beta, Normal, Uniform and other Distributions have one or more attributes that share the same physical units as the random variable represented by the distribution. These parameters are either location attributes (e.g., Mean of a Normal Distribution, Lower Limit of a Uniform Distribution) or scale attributes (e.g., Standard Deviation of a Normal Distribution). Where such attributes exist, a standard distribution can be defined with standardized values for the location and scale attributes. Where possible, code in the PSP performs CDF and inverse CDF calculations on the standard distributions using transformed arguments. Designs for TRAQUA and TRAVAL in the following sections identify the location and scale attributes, and specify how to transform them.

- Use Newton's method to invert a CDF–It is often easier to evaluate the CDF than the inverse CDF for a given distribution, if only because more methods have been published. Newton's method works very well at refining a crude estimate of an inverse CDF, given an accurate CDF. If $F(x)$ is the CDF being inverted at a point p, then Newton's method gives a way of finding a zero y_p of the function $G(x) = F(x)-p$ such that $G(y_p) = 0$. (Any elementary numerical analysis text may be consulted for details on Newton's method.) If x_i is the i^{th} approximation to y_p, then a much better approximation is given by

$$x_{i+1} = x_i - \frac{G(x_i)}{G'(x_i)} = x_i - \frac{F(x_i) - p}{f(x_i)} \quad , \tag{7.7-1}$$

where $f(x)$ is the PDF for the distribution. Typically this iterative scheme doubles the number of correct figures in x with every iteration.

8 DESIGN OF PARAMETER DISTRIBUTION AND PROBABILITY DISTRIBUTION OPERATIONS

8.1 Parameter Distribution Operations

Table 8.1/1 shows the Parameter Distribution operations identified in the requirements. Some of these are optional, and are not implemented in SYVAC3 version 3.09 (SV309). Table 8.1/2 shows additional maintenance operations that are implicit in the definition of an object. Because of the decision to expose the data structure for Parameter Distributions, not all of these are required either. Table 8.1/3 shows the operations that have been implemented, and identifies which routines perform these operations.

Table 8.1/1 (derived from Table 4.3/1) uses Mathematica notation to identify several operations for the Parameter Distribution object type. These operations should apply to any Parameter Distribution object, whatever the details of its internal structure (i.e., independently of the Probability Distribution subtype). The operations identified as "optional" are desirable for a complete Parameter Sampling Package (PSP), but not necessary for use with SYVAC3. None of the operations so marked have been implemented in SV309. They appear here as guidelines for future improvements to the PSP.

TABLE 8.1/1
EXPLICIT OPERATIONS FOR PARAMETER DISTRIBUTIONS

Operation	*Definition*	*Optional (Yes/No)*
`CDF[q_ParDis, x_?NumberQ]`	evaluate cumulative distribution function at x	No
`HighProbability[q_ParDis]`	evaluate the cumulative probability of the internal Probability Distribution at the upper limit of the Parameter Distribution	No
`HighValue[q_ParDis]`	find the upper limit of the Parameter Distribution	No
`LowProbability[q_ParDis]`	evaluate the cumulative probability of the internal Probability Distribution at the lower limit of the Parameter Distribution	No
`LowValue[q_ParDis]`	find the lower limit of the Parameter Distribution	No
`mean[q_ParDis]`	compute average value	Yes
`median[q_ParDis]`	compute 50th percentile	No
`PDF[q_ParDis, x_?NumberQ]`	evaluate probability density function at x	Yes
`quantile[q_ParDis, p_?ProbabilityQ]`	invert cumulative distribution function at probability p	No
`Sigma[q_ParDis]`	compute standard deviation	Yes

In addition to the specialized operations that apply to Parameter Distributions, there are several maintenance operations that apply to most object types. They were not shown in the requirements since they arise automatically when trying to implement a new object type. Table 8.1/2 lists these operations. Those that are labelled "optional" do not need to be implemented at all. The rest are needed, but the PSP does not necessarily have to implement them as packaged routines. Because the data structure of the Parameter Distribution object type is open and available to calling routines, it suffices to let calling routines perform these opera-

tions by manipulating the data structure directly. For example, a calling routine could read Parameter Distribution attributes from a data file and store them into the data arrays directly. This option is a valid implementation of the "read" operation. Table 8.1/3 summarizes the required operations and how the PSP deals with them.

TABLE 8.1/2
MAINTENANCE OPERATIONS FOR PARAMETER DISTRIBUTIONS

Operation	*Definition*	*Optional (Yes/No)*
`check[pd_ParDis]`	check *pd* for validity and internal integrity	No
`copy[pd_ParDis]`	creates new object identical to an old one	Yes
`create[ParDis, d_ProbDis, t_TruncInt, st___]`	creates new object: *ParDis[d, t, st]*	No
`delete[pd_ParDis]`	deletes existing object *pd*	Yes
`read[ParDis,inputdevice]`	read data from inputdevice and create new *ParDis* object	No
`write[pd_ParDis, outputdevice]`	write contents of *ParDis* object to *outputdevice*	No

TABLE 8.1/3
METHODS OF IMPLEMENTING ESSENTIAL OPERATIONS

Operation	*Method of Implementation*	*Routine*
`CDF`	evaluate Probability Distribution CDF and scale to Truncation Interval	TRAVAL
`check`	explicit routine	CKDIST
`create`	direct manipulation of data structures	
`HighProbability`	calling routine looks up result in data structure	
`HighValue`	transform HighProbability in calling routine	TRAQUA
`LowProbability`	calling routine looks up result in data structure	
`LowValue`	transform LowProbability in calling routine	TRAQUA
`median`	explicit routine with special argument value	TRAQUA
`quantile`	explicit routine	TRAQUA
`read`	operation in calling routine with direct manipulation of data structures	
`write`	operation in calling routine with direct manipulation of data structures	

8.2 Direct Manipulation of Data Structures by Calling Routines

The operations **create**, **HighProbability**, *and* **LowProbability**, *identified in Section 8.1, require the calling routine to directly manipulate the data arrays that store Parameter Distribution objects. This section shows how to perform these manipulations, and shows why they should be performed by the calling routines.*

The PSP does not directly support "HighProbability" and "LowProbability" operations because these operations are trivially simple to implement in the calling routine. To demonstrate this simplicity, Algorithm 8.2/1 shows the algorithm for HighProbability. The algorithm for LowProbability is strictly analogous. Both are just table lookups.

ALGORITHM 8.2/1: Algorithm for the Operation "HighProbability"

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
INDX	Integer	In	Index i of Parameter Distribution	$0 < i$	1: lower bound
HIBNDD	Double Precision (*)	In	Upper probability bounds p_H	$0 \le p_H(i) \le 1$	0: lower bound 1: upper bound
HIGHP	Double Precision	Out	Probabilistic upper bound	$= p_H(i)$	

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
HIGHP $\leftarrow p_H(i)$	Simple table lookup.

The PSP does not support the "create" operation by providing a "Create Parameter Distribution" operation to generate new Parameter Distribution objects because the data interface would be excessively cumbersome. Table 8.2/1 provides a possible argument list for such a routine. The routine would need all the arrays used in the Parameter Distribution data structure, including pointers and counts. It would also require arguments containing all the fields to be inserted into those arrays, including an array for Piecewise Uniform Distribution data. In short, it would be about as much work for the calling routine to set up arguments for the call to "create Parameter Distribution" as it is to perform the "create" operation itself. In a language with structured data types, there would still be a point in implementing this routine, because it would create a new instance of an object type. In this Fortran 77 design, however, implementing a new object essentially means filling in a new row in each existing table. No new data objects (i.e., no new arrays) are created.

Calling routines implement "create" by performing the following actions: (1) copying the incoming fields into the permanent data structures, and (2) setting all pointers appropriately. The only source of complexity is in handling the variety of Probability Distribution types.

TABLE 8.2/1
EXAMPLE: EXCESSIVELY LONG ARGUMENT LIST FOR A "CREATE PARAMETER DISTRIBUTION" ROUTINE

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
PSNAME	Character*(*)	In	Name of new parameter (useful for error messages)	Unique	None
DTYPE	Character*(*)	In	Distribution type for new parameter	See Table 3.5/4	Derive from Table 3.5/4
ATTRIB	Double Precision (4)	In	Distribution attributes for new parameter	See Table 3.5/4	Derive from Table 3.5/4
ICORR	Integer	In	Pointer k to correlated parameter (for DTYPE in {"CORLN", "CORNR"})	$0 < k \leq i$	1: lower bound i: upper bound
NPUCMP	Integer	In	Number of ranges in Piecewise Uniform	0 < NPUCMP	1:lower bound
PUCOMP	Double Precision (3, NPUCMP)	In	Piecewise Uniform ranges $R(k,l)$	See Table 3.5/3	$R(1,l) = R(2,l)$ $R(3,l) = 0$ $R(2,l) = R(1,l+1)$
LOWP	Double Precision	In	Lower probability bound q_L for new parameter	$0 \leq q_L < q_H \leq 1$	0: lower bound q_H: upper bound
HIGHP	Double Precision	In	Upper probability bound q_H for new parameter	$0 \leq q_L < q_H \leq 1$	q_L : lower bound 1: upper bound
MXPAR	Integer	In	Maximum number of Parameter Distributions	$i <$ MXPAR	i+1: lower bound
MXPUCL	Integer	In	Maximum number of components in all Piecewise Uniform Distributions	$j <$ MXPUCL	j+1: lower bound
LINDX	Integer	In/Out	Index i of last Parameter Distribution	$0 \leq i <$ MXPAR	0: lower bound
LPUPTR	Integer	In/Out	Index j of last Piecewise Uniform component	$0 \leq j <$ MXPUCL	0: lower bound
DSTTYP	Character*(*) (MXPAR)	In/Out	Distribution types	Valid	See Table 3.5/4
DSTPAR	Double Precision (4, MXPAR)	In/Out	Probability Distribution attributes $A(k,l)$	See Table 3.5/4	Derive from Table 3.5/4
IDXCOR	Integer (MXPAR)	In/Out	Indices $K(l)$ of correlated parameters for DSTTYP(l) in {"CORLN", "CORNR"} only	$0 < K(i) < i$ See Table 3.5/4	1: lower bound i-1: upper bound
PUDPAR	Double Precision (3, MXPUCL)	In/Out	Piecewise Uniform detailed attributes $Q(k,l)$	See Table 3.5/3	$Q(1,l) = Q(2,l)$ $Q(3,l) = 0$ $Q(2,l) = Q(1,l+1)$
LOBNDD	Double Precision (MXPAR)	In/Out	Lower probability bounds $p_L(i)$	$0 \leq p_L(i) \leq p_H(i)$	0: lower bound $p_H(i)$: upper bound
HIBNDD	Double Precision (MXPAR)	In/Out	Upper probability bounds $p_H(i)$	$p_L(i) \leq p_H(i) \leq 1$	$p_L(i)$: lower bound 1: upper bound
OK	Logical	Out	Created OK	True or False	

8.3 Reading and Writing Parameter Distributions

The operations **read** *and* **write** *(identified in Section 8.1) perform input and output of Parameter Distribution objects in a standard format. In an application like SYVAC3, the data used to create new Parameter Distribution objects come from a data file. A read operation would be a more practical addition to the Parameter Sampling Package (PSP) than a create operation, since the new data does not come through the argument list (see Section 8.2). In SYVAC3, the routine INPDIS reads Parameter Distributions from a data file, but it is not a suitable routine for the PSP. A write routine to format a Parameter Distribution would be useful only if a standard format could suit all applications.*

Table 8.3/1 shows the fields input by SYVAC3 and the order in which they are read. This order is arbitrary, and could be different for a different application. Having "read" and "write" routines for Parameter Distribution would standardize this order, making it easier to write code to store and retrieve Parameter Distribution objects. Unfortunately, it is not clear what format should be standard. Even a simple double-precision value in Fortran 77 can appear differently, depending on the format code used. For example, the number of digits shown and the use of exponential notation both depend on format codes. There are many more options for an object with the complexity of a Parameter Distribution. For this reason, each application is given the freedom of writing its own input/output routines.

The routine INPDIS reads Parameter Distribution data from an input file in the SYVAC3 standard order. It permits different formats and spacing of individual fields, as long as they appear in the right order. INPDIS does not belong to the PSP for a couple of reasons. First, it uses SYVAC3 file-reading operations, which would make the PSP depend on the SYVAC3 File Reading Package. Second, INPDIS is beyond the scope of the PSP. It actually reads a set of Sampled Parameter descriptions, of which Parameter Distribution objects are only a part. Note that the correlated distributions in Table 8.3/1 refer to a Sampled Parameter object by name rather than to another Parameter Distribution object. To read these fields correctly, a PSP routine would need to know something about Sampled Parameter objects.

INPDIS also writes out the newly read Parameter Distribution data to a SYVAC3 output file, thereby defining an output format. In doing so, it writes both the Probability Interval and Value Interval versions of the Truncation Interval. INPDIS computes Value Intervals for Parameter Distributions where Probability Intervals are input, but it does not store them, since only the Probability Interval appears in the current version of the Parameter Distribution data structure. A standard PSP write routine would not be able to recreate Value Intervals that were input if the bounds lay outside the Lower and Upper Limits of a finite distribution, and so it could not reproduce the output currently provided by INPDIS.

TABLE 8.3/1
FIELDS TO BE READ IN SYVAC3 FOR A PARAMETER DISTRIBUTION

Probability Distribution Subtype	*Parameter Distribution Fields*	*Constraint*
Beta	f1. Subtype descriptor	'BETA'
	f2. Lower limit	f2 ≤ f3
	f3. Upper limit	f2 ≤ f3
	f4. Exponent *a*	0 < f4
	f5. Exponent *b*	0 < f5
	f6. Quantile/value bounds	'Q' or 'V'
	f7. Lower bound	f7 < f8
	f8. Upper bound	f7 < f8
Binomial (optional)	f1. Subtype descriptor	'BINOM'
	f2. Number of trials *n*	0 ≤ f2
	f3. Probability *p*	0 ≤ f3 ≤ 1
	f4. Quantile/value bounds	'Q' or 'V'
	f5. Lower bound	f5 < f6
	f6. Upper bound	f5 < f6
Constant	f1. Subtype descriptor	'CONST'
	f2. Constant value	–
Correlated Lognormal	f1. Subtype descriptor	'CORLN'
	f2. Geometric mean	0 < f2
	f3. Geometric standard deviation	1 < f3
	f4. Correlation coefficient	-1 ≤ f4 ≤ 1
	f5. Quantile/value bounds	'Q' or 'V'
	f6. Lower bound	f6 < f7
	f7. Upper bound	f6 < f7
	f8. Correlated parameter	valid name
Correlated Normal	f1. Subtype descriptor	'CORNR'
	f2. Mean	–
	f3. Standard deviation	0 < f3
	f4. Correlation coefficient	-1 ≤ f4 ≤ 1
	f5. Quantile/value bounds	'Q' or 'V'
	f6. Lower bound	f6 < f7
	f7. Upper bound	f6 < f7
	f8. Correlated parameter	valid name
Lognormal	f1. Subtype descriptor	'LGNRM'
	f2. Geometric mean	0 < f2
	f3. Geometric standard deviation	1 < f3
	f5. Quantile (probability) or value bounds	'Q' or 'V'
	f6. Lower bound	f6 < f7
	f7. Upper bound	f6 < f7
Loguniform	f1. Subtype descriptor	'LGUFM'
	f2. Lower limit	f2 ≤ f3
	f3. Upper limit	f2 ≤ f3
	f4. Quantile/value bounds	'Q' or 'V'
	f5. Lower bound	f5 < f6
	f6. Upper bound	f5 < f6
Normal	f1. Subtype descriptor	'NORML'
	f2. Mean	–
	f3. Standard deviation	0 < f3
	f4. Quantile/value bounds	'Q' or 'V'
	f5. Lower bound	f5 < f6
	f6. Upper bound	f5 < f6
Piecewise Uniform	f1. Subtype descriptor	'PCUFM'
	f2. Number of ranges *m*	0 < *m*
	f3. Quantile/value bounds	'Q' or 'V'
	f4. Lower bound	f4 < f5
	f5. Upper bound	f4 < f5
	m repetitions of:	
	f6. Lower range end	f6 < f7
	f7. Upper range end	f6 < f7
	f8. Mixture weight	0 ≤ f8
Triangular	f1. Subtype descriptor	'TRIAN'
	f2. Lower limit	f2 < f4
	f3. Mode	f2 ≤ f3 ≤ f4
	f4. Upper limit	f2 < f4
	f5. Quantile/value bounds	'Q' or 'V'
	f6. Lower bound	f6 < f7
	f7. Upper bound	f6 < f7
Uniform	f1. Subtype descriptor	'UNIFM'
	f2. Lower limit	f2 ≤ f3
	f3. Upper limit	f2 ≤ f3
	f4. Quantile/value bounds	'Q' or 'V'
	f5. Lower bound	f5 < f6
	f6. Upper bound	f5 < f6

8.4 Argument List for Operations (e.g., CDF) that Do Not Modify Parameter Distributions

The preceding sections have discussed some of the essential Parameter Distribution operations identified in Section 8.1. The remaining operations use Parameter Distribution objects to perform calculations. These operations have a standard section in their argument lists that passes down the Parameter Distribution data structure. The algorithm for CDF appears here as an example of the use of this argument list.

The argument list for an operation that uses a Parameter Distribution object must provide all the data arrays that store Parameter Distribution data. It is not possible to pass down the data fields for just one distribution since some Probability Distribution subtypes (i.e., Correlated Normal and Correlated Lognormal Distributions) refer to more than just one Parameter Distribution. Nor is it possible to leave out arrays that store data for only a few Probability Distribution subtypes (e.g., storage for Piecewise Uniform ranges), since the Parameter Distribution of interest could have any type of Probability Distribution. One simplification can be made, however, when compared to the argument list for the "create" operation in Section 8.2. It is not necessary to pass down the sizes of the arrays because these routines make no additions or changes to the entries. The only pointer required is the index of the Parameter Distribution of interest. Table 8.4/1 shows the standard arguments required for such routines; these arguments are referred to in the algorithms throughout the rest of Section 8.

TABLE 8.4/1
STANDARD ARGUMENTS FOR OPERATIONS ON PARAMETER DISTRIBUTIONS

Argument	*Definition*
INDX	Index i of Parameter Distribution
PSNAME	Names of parameters
DSTTYP	Distiribution types
DSTPAR	Probability Distribution attributes $A(k,l)$
IDXCOR	Indices $K(l)$ of correlated parameters
PUDPAR	Piecewise Uniform detailed attributes $Q(k,l)$
LOBNDD	Lower probability bounds $p_L(i)$
HIBNDD	Upper probability bounds $p_H(i)$

This standard argument list is defined in greater detail in Algorithm 8.4/1. In addition to the required arguments, the argument list contains a list of Parameter Names. These names do not take any part in the main operations of the called routines. They provide unique identifiers for the Parameter Distributions for warning and error messages when some problem arises in the calculations. In SYVAC3 these names are key attributes of Sampled Parameter objects. Each Sampled Parameter possesses a single Parameter Distribution as another attribute, and so there is a one-to-one relationship between Sampled Parameters and Parameter Distributions.

As an example of the use of the argument list in Table 8.4/1, Algorithm 8.4/1 shows how a model could evaluate the CDF of a Parameter Distribution. For Conditional Distribution objects, which have Correlated Normal or Correlated Lognormal Distributions, this algorithm provides the marginal CDF operation, not the conditional CDF (CCDF) operation of a Conditional Distribution. This algorithm uses a call to TRAVAL, which evaluates the CDF of the Probability Distribution component of a Parameter Distribution. The Probability Distribution CDF is all that SYVAC3 requires. The more complete Parameter Distribution CDF described by Algorithm 8.4/1 has not yet been implemented as a separate routine. The CCDF routine for Conditional Distribution objects has also not been implemented, although TRAQUA provides the inverse CCDF routine.

ALGORITHM 8.4/1: CDF of a Parameter Distribution (Ignoring Correlation)

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
INDX	Integer	In	Index i of Parameter Distribution	$0 < i$	1: lower bound
PARVAL	Double Precision	In	Parameter value x	–	Depends on distribution type
PSNAME	Character*(*) (*)	In	Names of parameters	Unique names	See Table 3.5/1
DSTTYP	Character*(*) (*)	In	Distribution types	See Table 3.5/4	See Table 3.5/4
DSTPAR	Double Precision (4,*)	In	Probability Distribution attributes $A(k,l)$	See Table 3.5/4	Derive from Table 3.5/4
IDXCOR	Integer	In	Indices $K(l)$ of correlated parameters	$0 < K(i) < i$ See Table 3.5/4	1: lower bound i-1: upper bound
PUDPAR	Double Precision (3,*)	In	Piecewise Uniform detailed attributes $Q(k,l)$	See Table 3.5/3	$Q(1,l) = Q(2,l)$ $Q(3,l) = 0$ $Q(2,l) = Q(1,l+1)$
LOBNDD	Double Precision (*)	In	Lower probability bounds $p_L(i)$	$0 \le p_L(i) \le p_H(i)$	0: lower bound $p_H(i)$: upper bound
HIBNDD	Double Precision (*)	In	Upper probability bounds $p_H(i)$	$p_L(i) \le p_H(i) \le 1$	$p_L(i)$: lower bound 1: upper bound
QUANTL	Double Precision	Out	CDF p of distribution at x	$0 \le p = F(x) \le 1$	0: lower bound 1: upper bound
TRANOK	Logical	Out	Transformed OK	True or False	

Precondition	All input arguments satisfy their constraints.
Postcondition	All output arguments satisfy their constraints. TRANOK is True if and only if the routine terminated successfully.

Algorithm	*Commentary*
call TRAVAL(INDX, PARVAL, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, QUANTL, TRANOK)	Evaluate the CDF of the Probability Distribution component of the current Parameter Distribution, without considering the Truncation Interval.
if (TRANOK) **then**	
QUANTL ← (QUANTL - $p_L(i)$) / ($p_H(i)$ - $p_L(i)$)	Scale the CDF to the Truncation Interval.
QUANTL ← *max*(0, *min*(1, QUANTL))	Ensure that the CDF lies in [0,1]. Values outside this interval were truncated.
end if	

8.5 Derived Operations Performed by Calling Routines

The operations **HighValue**, **LowValue**, *and* **median** *(identified in Section 8.1) require the calling routine to perform calculations that involve calling TRAQUA in the Parameter Sampling Package (PSP). This section provides algorithms for these calculations. Routines in applications that use the PSP currently encode these algorithms inline. In future implementations, new PSP routines may encapsulate these operations to simplify their use.*

The operations "HighValue" and "LowValue" together find the bounds on the Truncation Interval as a Value Interval rather than a Probability Interval. To translate these bounds they call TRAQUA. That routine inverts the CDF of the Probability Distribution component of a Parameter Distribution, as shown in Algorithm 8.5/1 for "HighValue." The algorithm for "LowValue" is similar, except that the CDF is inverted at the probability zero rather than one.

The operations "HighValue" and "LowValue," when applied to a distribution over a finite interval, always return values between the Lower Limit and the Upper Limit of the distribution. For example, the "HighValue" operation returns the value 5 when applied to an untruncated Uniform Distribution from 3 to 5. When applied to an untruncated infinite distribution, the "HighValue" and "LowValue" operations return values so far out in the tails of the distribution that the CDF at those points is not significantly different from zero and one. These operations do not return positive or negative infinity, as Fortran 77 has no way of representing those values.

ALGORITHM 8.5/1: HighValue for a Parameter Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
The argument list containing INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, and HIBNDD (see Section 8.4) appears first, with the arguments listed below following HIBNDD.					
HIVAL	Double Precision	Out	Upper truncation bound x_H	$F(x_H) \equiv p_H(i)$	$F^{-1}(0)$: lower bound $F^{-1}(1)$: upper bound $\Pr(X = x_H) > 0$
HIGHOK	Logical	Out	High Value found OK	True or False	

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. HIGHOK takes the value True if and only if the routine terminated successfully.

Algorithm	*Commentary*
call TRAQUA(INDX, 1.D0, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, HIVAL, HIGHOK)	Invert the Parameter Distribution CDF at 1. (The inverse at 0 would give the "LowValue.")

Like "HighValue" and "LowValue," the "median" operation finds a particular value in a Parameter Distribution. In the case of "median," the value has the property that the proba-

bility of randomly selecting a higher value equals the probability of selecting a lower value. That is, the "median" operation finds the 0.5 quantile of the Parameter Distribution. Algorithm 8.5/2 shows how to compute this value. The algorithm is very similar to that of "HighValue" and "LowValue."

ALGORITHM 8.5/2: Median for a Parameter Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
The argument list containing INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOGNDD, and HIBNDD (see Section 8.4) appears first, with the arguments listed below following HIBNDD.					
MEDIAN	Double Precision	Out	Median value x_M	$F(x_M) \cong 0.5$	$\Pr(X = x_M) > 0$
MEDOK	Logical	Out	Median found OK	True or False	

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. MEDOK takes the value True if and only if the routine terminated successfully.

Algorithm	*Commentary*
call TRAQUA(INDX, 0.5D0, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, HIBNDD, HIVAL, HIGHOK)	Invert the Parameter Distribution CDF at 0.5. The Truncation Interval applies.

While the algorithms shown here will work for all Probability Distribution types, the results may not always match expectations based on continuous distributions. If X has a continuous distribution and a median of m, $\Pr(X < m) = 0.5$ and $\Pr(X > m) = 0.5$, where Pr(a) is the probability of event A. For a discrete or mixed distribution (i.e., one with finite probabilities of achieving one or more specific values) with median m, the best that can be said is that $\Pr(X < m) \leq 0.5$ and $\Pr(X > m) \geq 0.5$.

For example, consider a variable X associated with a Binomial Distribution object that has only one trial with a probability $p = 0.7$ of success. X can take either the value zero (with a probability of 0.3) or one (with a probability of 0.7). What is the median value? Is it zero, or one, or some value in between? The median cannot be zero because the probability that X exceeds zero is 0.7, which is greater than 0.5. The median cannot be any value between zero and one for the same reason. Therefore, the median must be one. In this case, $\Pr(X < \text{median}) = 0.3$ and $\Pr(X > \text{median}) = 0$. Both these probabilities would be unacceptable for a continuous distribution, but they make sense for a discrete distribution.

The "HighValue" and "LowValue" operations for discrete and mixed distributions also behave in ways that would be unacceptable for continuous distributions. For example, since the "LowValue" of a discrete distribution is a discrete value, it may remain unchanged by significant changes in the corresponding probability in LOBNDD.

8.6 CKDIST: Check a Parameter Distribution

CKDIST checks a Parameter Distribution for adherence to all the constraints in Section 3.5. Its argument list contains the list of arguments in Table 8.4/1. If CKDIST finds problems, it sets a flag and also writes warning or error messages using WRSPWN or WRSPER.

CKDIST uses the generic argument list associated with the arguments in Table 8.4/1. Unlike most other routines that use this argument list, CKDIST may change entries. Specifically, it restricts entries in the arrays LOBNDD and HIBNDD to the interval [0,1].

CKDIST follows Algorithm 8.6/1, which for brevity does not show calls to WRSPWN and WRSPER to write messages. Code derived from the algorithm should generate an error message with WRSPER whenever one of the checks fails. It should generate a warning message with WRSPWN if LOBNDD or HIBNDD is changed.

CKDIST is called by both TRAVAL and TRAQUA whenever these routines are invoked. Because other routines call it frequently, its operation should be made efficient.

ALGORITHM 8.6/1: Subroutine CKDIST: Check Parameter Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
The argument list from Section 8.4 containing INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, and HIBNDD appears here, with one change: LOBNDD and HIBNDD are In/Out rather than In in this routine. The argument listed below appears after HIBNDD.					
DSTOK	Logical	Out	Distribution is OK	True or False	

Precondition	No precondition–input arguments may or may not satisfy their constraints.
Postcondition	$p_L(i)$ has been increased to 0 if its original value was negative. $p_H(i)$ has been reduced to 1 if its original value was greater than 1. Suitable error or warning messages have been written for each input constraint not satisfied. DSTOK takes the value True if and only if: all constraints were satisfied or the only constraints not satisfied were those requiring adjustments of $p_L(i)$ and/or $p_H(i)$. .

Algorithm	*Commentary*
call INDXER(MODNAM, *i*, DSTOK)	Check the index $i > 0$.
if (DSTOK) **then**	
if (DSTTYP(*i*) = 'BETA') **then** *check that:* $A(1,i) \le A(2,i)$ *and* $0 < A(3,i)$ *and* $0 < A(4,i)$	Beta Distribution Order of interval limits; positive shape parameters.
else **if** (DSTTYP(*i*) = 'BINOM') **then** *check that:* $0 \le A(1,i)$ *and* $0 \le A(2,i) \le 1$	Binomial Distribution (optional) Non-negative number of trials, and valid probability.
else if (DSTTYP(*i*) = 'CONST') *then* *no checks needed*	*Constant Distribution* Any attribute value is acceptable.

else if (DSTTYP(i) = 'CORLN') **then** *check that:* $0 < A(1,i)$ *and* $1 < A(2,i)$ *and* $-1 \le A(3,i) \le 1$ **and** $0 < K(i) < i$ **and** ((DSTTYP($K(i)$) = 'NORML' *and* $0 < A(2,K(i))$) **or** (DSTTYP($K(i)$) = 'LGNRM' **and** $0 < A(1,K(i))$ **and** $1 < A(2,K(i))$))	Correlated Lognormal Distribution Geometric mean, geometric std deviation,correlation. Index of correlated Parameter Distribution. Correlated independent Normal Distribution. Correlated independent Lognormal Distribution. *Geometric mean, geometric standard deviation*
else if (DSTTYP(i) = 'CORNR') **then** *check that:* $0 < A(2,i)$ **and** $-1 \le A(3,i) \le 1$ **and** $0 < K(i) < i$ **and** ((DSTTYP($K(i)$) = 'NORML' **and** $0 < A(2,K(i))$) **or** (DSTTYP($K(i)$) = 'LGNRM' **and** $0 < A(1,K(i))$ **and** $1 < A(2,K(i))$))	Correlated Normal Distribution Standard deviation and correlation coefficient. Index of correlated Parameter Distribution. Correlated independent Normal Distribution. Correlated independent Lognormal Distribution. Geometric mean, geometric standard deviation.
else if (DSTTYP(i) = 'LGNRM') **then** *check that:* $0 < A(1,i)$ **and** $1 < A(2,i)$	Lognormal Distribution Geometric mean, geometric standard deviation.
else if (DSTTYP(i) = 'LGUFM') **then** *check that:* $0 < A(1,i) < A(2,i)$	Loguniform Distribution Interval limits are ordered correctly.
else if (DSTTYP(i) = 'NORML' **then** *check that:* $0 < A(2,i)$	Normal Distribution Positive standard deviation.
else if (DSTTYP(i) = 'PCUFM') **then** *check that:* $0 \le A(1,i)$ **and** $0 < A(2,i)$ **and** $0 < A(3,i)$ **and** (PUDPAR($1,j$) $\le$ PUDPAR($2,j$) **and** $0 \le$ PUDPAR($3,j$)) *for* j *from* $A(1,i)+1$ *to* $A(1,i)+A(2,i)$ **and** (PUDPAR($2,j$) $\le$ PUDPAR($1,j+1$)) *for* j *from* $A(1,i)+1$ *to* $A(1,i)+A(2,i)-1$	Piecewise Uniform Distribution Pointer to PUDPAR, number of ranges, total weight. Range ends are ordered correctly and each weight is non-negative. Ranges are ordered correctly.
else if (DSTTYP(i) = 'TRIAN') **then** *check that:* $A(1,i) \le A(2,i) \le A(3,i)$	Triangular Distribution Interval limits ordered right, with mode in middle.
else if (DSTTYP(i) = 'UNIFM') **then** *check that:* $A(1,i) < A(2,i)$	Uniform Distribution Interval limits ordered right.
end if	
if (DSTTYP(i) $\ne$ 'CONST') **then** *check that:* LOBNDD(i) $\le$ HIBNDD(i)	Verify that upper and lower bounds are properly ordered for nonconstant distributions.
end if	
if (*any of the checks above failed*) **then** *write suitable error messages using* WRSPER. DSTOK $\leftarrow$ *False*	The programmer is free to construct suitable messages.
else DSTOK $\leftarrow$ *True*	
end if	

if (DSTTYP(i) $\ne$ 'CONST') **then**
LOBNDD(i) $\leftarrow$ *max*(0, LOBNDD(i))
HIBNDD(i) $\leftarrow$ *min*(1, HIBNDD(i))
if (*either bound required adjustment*) **then**
write a warning message using WRSPWN.
end if
end if
	For nonconstant distributions, the lower and upper bounds should lie in the interval [0,1] since they are probabilities.
	Invalid bounds do not cause a problem, and so they are made to be valid, and only a warning message is written.

end if	

8.7 TRAVAL: CDF of a Probability Distribution

TRAVAL evaluates the CDF of the Probability Distribution component of a Parameter Distribution. Its arguments use the data structure described by Table 8.4/1. TRAVAL calls CKDIST to check incoming arguments to ensure a valid distribution is available. TRAVAL performs simple calculations directly. It calls other routines in the package to perform more complicated CDF calculations. TRAVAL applies scale and location transformations to a standard distribution where appropriate.

TRAVAL uses Algorithm 8.7/1, which reveals some points of interest:

- Discrete/Mixed Distributions–Finite distributions have a finite interval within which the probability density function (PDF) is positive. Values outside this interval have a CDF of either zero or one. TRAVAL should assign CDF values correctly even when the interval is of zero length, and the CDF is discontinuous. Discrete distributions (e.g., Constant Distribution, Binomial Distribution) where the variate can take only discrete values, and mixed distributions (e.g., Piecewise Uniform Distribution) where some of the values the variate can take are discrete, have discontinuous CDFs and must be treated correctly.

- Location and Scale Transformations–Many distributions can be standardized (given standard attribute values) by suitable transformations. CDF calculations on standardized distributions require less computation. Location transformations shift the distribution so that some selected attribute (e.g., the left end of a finite interval, or the mean of a Normal Distribution) takes the value zero. Scale transformations renormalize the width so that another selected attribute (e.g., the right end of a finite interval, or one standard deviation from the mean for a Normal Distribution) takes the value one.

- Subroutine calls–Complicated calculations for normalized distributions take place in independent routines (e.g., NORDIS for the standard normal CDF).

ALGORITHM 8.7/1: Subroutine TRAVAL: Evaluate CDF for a Probability Distribution in a Parameter Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
The argument list from Section 8.4 containing INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, and HIBNDD appear first, with PARVAL inserted between INDX and PSNAME; the following additional arguments appear after HIBNDD.					
PARVAL	Double Precision	In	Parameter value x	–	Depends on distribution type
QUANTL	Double Precision	Out	CDF p of distribution at x	$0 \le p = F(x) \le 1$	0: lower bound 1: upper bound
TRANOK	Logical	Out	Transformed OK	True or False	

Precondition	All input arguments satisfy their constraints.
Postcondition	All output arguments satisfy their constraints. TRANOK takes the value True if and only if QUANTL has been evaluated successfully.

Algorithm	*Commentary*
call CKDIST(*i*, PSNAME, DSTTYP, *A*, *K*, *Q*, p_L, p_H, TRANOK) **if** (TRANOK) **then**	The constraints can be summarized as "Use a valid distribution"; CKDIST checks for validity.
if (DSTTYP(*i*) = 'BETA') **then** call BTADIS(*A*(3,*i*), *A*(4,*i*), (*x* - *A*(1,*i*)) / (*A*(2,*i*) - *A*(1,*i*)), *p*)	Beta Distribution. Evaluate CDF for standard beta distribution.
else if (DSTTYP(*i*) = 'BINOM') **then** call BINDIS(*x*, *A*(1,*i*), *A*(2,*i*), *p*)	Binomial Distribution (optional). Evaluate CDF with BINDIS.
else if (DSTTYP(*i*) = 'CONST') **then** **if** (*x* < *A*(1,*i*)) **then** *p* ← 0 **else** p ← 1 **end if**	Constant Distribution. The CDF jumps from 0 to 1 at *A*(1,*i*).
else if (DSTTYP(*i*) = 'LGNRM' or DSTTYP(*i*) = 'CORLN') **then** **if** (*x* ≤ 0) **then** p ← 0 **else** *z* ← *log*(*x* / *A*(1,*i*)) / *log*(*A*(2,*i*)) call NORDIS(z, *p*) **end if**	Lognormal Distribution or Correlated Lognormal Distribution. CDF = 0 below and at 0. Standard value with mean 0 and std. dev. 1. Evaluate CDF of standard normal.
else if (DSTTYP(*i*) = 'LGUFM') **then** **if** (*x* < *A*(1,*i*)) **then** *p* ← 0 **else if** (*x* ≥ *A*(2,*i*)) **then** *p* ← 1 **else** *p* ← *log*(*x* / *A*(1,*i*)) / *log*(*A*(2,*i*) / *A*(1,*i*)) **end if**	Loguniform Distribution. CDF = 0 below the range. CDF = 1 above the range. Interpolate in linear CDF on logarithmic scale.
else if (DSTTYP(*i*) = 'NORML' *or* DSTTYP(*i*) = 'CORNR') **then** *z* ← (*x* - *A*(1,*i*)) / *A*(2,*i*) call NORDIS(z, *p*)	Normal Distribution or Correlated Normal Distribution. Standard value with mean 0 and std. dev. 1. Evaluate CDF of standard normal.
else if (DSTTYP(*i*) = 'PCUFM') **then** call PUDIS(*x*, *round*(*A*(1,*i*)), *round*(*A*(2,*i*)), *A*(3,*i*), *Q*, *p*)	Piecewise Uniform Distribution. Evaluate CDF. *Round* converts to integer.
else if (DSTTYP(*i*) = 'TRIAN') **then** call TRIDIS((*x* - *A*(1,*i*)) / (*A*(3,*i*) - *A*(1,*i*)), (*A*(2,*i*) - *A*(1,*i*)) / (*A*(3,*i*) - *A*(1,*i*)), *p*)	Triangular Distribution. Transform *x* to standard value in [0,1] and evaluate CDF with TRIDIS.
else if (DSTTYP(*i*) = 'UNIFM') **then** **if** (*x* < *A*(1,*i*)) **then** *p* ← 0 **else if** (*x* ≥ *A*(2,*i*)) **then** p ← 1 **else** *p* ← (*x* - *A*(1,*i*)) / (*A*(2,*i*) - *A*(1,*i*)) **end if** **end if** **end if**	Uniform Distribution. CDF = 0 below the range. CDF = 1 above the range. Interpolate in linear CDF to find probability.

8.8 TRAQUA: Invert the CDF of a Parameter Distribution

TRAQUA transforms a value between 0 and 1 to a value in the domain of a Parameter Distribution by inverting the CDF. Its arguments use the arguments in Table 8.4/1 in the data structure of Algorithm 8.4/1. TRAQUA calls CKDIST to check incoming arguments to ensure a valid distribution is used. It may call other routines to perform complicated inverse calculations.

TRAQUA uses Algorithm 8.8/1 to invert the CDF of a Parameter Distribution. If the Probability Distribution part of the Parameter Distribution has a Distribution Type of either 'CORLN' or 'CORNR,' the Parameter Distribution is a Conditional Distribution. In that case, TRAQUA inverts the conditional CDF (CCDF) of the distribution. TRAQUA and TRAVAL are in many respects inverse operations of each other, but not in their treatment of Conditional Distributions.

When the distribution is discrete (e.g., a Constant Distribution) or mixed (e.g., some Piecewise Uniform Distributions with zero-width intervals), the distribution has a discontinuous CDF. For example, a Constant Distribution has a CDF that jumps from 0 to 1. In such cases TRAQUA maps any input probability that lies between the two discontinuous values onto the location of the discontinuity.

Like TRAVAL, TRAQUA maps distributions that can be standardized onto a standard form. For example, TRAQUA maps a Normal Distribution onto a standard normal with a mean of zero and a standard deviation of one. TRAQUA then inverts the standard distribution, and maps the result back onto the domain of the original distribution.

Finally, it is important to note that the input probability to TRAQUA lies in the interval [0,1], not the interval $[p_L(i), p_H(i)]$. TRAQUA maps every input probability onto the interval $[p_L(i), p_H(i)]$ before inverting.

ALGORITHM 8.8/1: Subroutine TRAQUA: Invert CDF of a Parameter Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
The argument list containing INDX, PSNAME, DSTTYP, DSTPAR, IDXCOR, PUDPAR, LOBNDD, and HIBNDD (see Section 8.4) appears first. The next argument, QUANTL, appears within this argument list between INDX and PSNAME. PARVAL and TRANOK appear after HIBNDD.					
QUANTL	Double Precision	In/Out	Probability p designating a quantile	$0 \le p \le 1$	0: lower bound 1: upper bound
PARVAL	Double Precision (*)	In/Out	Parameter value $X(i)$	–	Depends on distribution type
TRANOK	Logical	Out	Transformed OK	True or False	

Precondition	All input arguments satisfy their constraints.
Postcondition	All output arguments satisfy their contraints. QUANTL may be changed, but only by replacing its value by 0 or 1 if it lies respectively above or below the interval [0,1]. PARVAL(i) = F^{-1}(QUANTL), where F is the CDF of the i^{th} distribution. TRANOK = *True* if and only if the routine terminated successfully.

Algorithm	*Commentary*
call CKDIST(i, PSNAME, DSTTYP, A, K, Q, p_L, p_H, TRANOK)	CKDIST checks distribution validity.
$p \leftarrow max(0, min(1,p))$	Restrict probability to interval [0,1].
$q \leftarrow (1\text{-}p)p_L(i) + p \cdot p_H(i)$	Interpolate between high and low probabilities.
if (TRANOK) **then**	
if (DSTTYP(i) = 'BETA') **then**	Beta Distribution
call INVBTA(q, $A(3,i)$, $A(4,i)$, y)	Find standard beta variate y.
$X(i) \leftarrow (1\text{-}y)A(1,i) + y \cdot A(2,i)$	Map y onto the range of $X(i)$.
else if (DSTTYP(i) = 'BINOM') **then**	Binomical Distribution (optional)
call INVBIN(q, $A(1,i)$, $A(2,i)$, $X(i)$	Invert binomial CDF.
else if (DSTTYP(i) = 'CONST') **then**	Constant Distribution
$X(i) \leftarrow A(1,i)$	Always the same point.
else if (DSTTYP(i) = 'CORLN') **then**	Correlated Lognormal Distribution
if (DSTTYP($K(i)$) = 'NORML') **then**	Correlated to Normal Distribution.
call INVCOR(p, log $A(1,i)$, log $A(2,i)$, $p_L(i)$, $p_H(i)$, $A(3,i)$, $A(1,K(i))$, $A(2,K(i))$, $X(K(i))$, y)	Invert conditional distribution to get log of result.
else if (DSTTYP($K(i)$) = 'LGNRM') **then**	Correlated to Lognormal Distribution.
call INVCOR(p, log $A(1,i)$, log $A(2,i)$, $p_L(i)$, $p_H(i)$, $A(3,i)$, log $A(1,K(i))$, log $A(2,K(i))$, log $X(K(i))$, y)	Invert conditional distribution to get log of result.
end if	
$X(i) \leftarrow e^y$	Transform from log scale to give final value.
else if (DSTTYP(i) = 'CORNR') **then**	Correlated Normal Distribution
if (DSTTYP($K(i)$) = 'NORML') **then**	Correlated to Normal Distribution.
call INVCOR(p, $A(1,i)$, $A(2,i)$, $p_L(i)$, $p_H(i)$, $A(3,i)$, $A(1,K(i))$, $A(2,K(i))$, $X(K(i))$, $X(i)$)	Invert conditional distribution to get result.
else if (DSTTYP($K(i)$) = 'LGNRM') **then**	Correlated to Lognormal Distribution.
call INVCOR(p, $A(1,i)$, $A(2,i)$, $p_L(i)$, $p_H(i)$, $A(3,i)$, log $A(1,K(i))$, log $A(2,K(i))$, log $X(K(i))$, $X(i)$)	Invert conditional distribution to get result.
end if	
else if (DSTTYP(i) = 'LGNRM') **then**	Lognormal Distribution or
call INVNOR(q, y)	Invert standard normal at truncated probability.
$X(i) \leftarrow A(1,i) \cdot A(2,i)^y$	Transform to log scale.
else if (DSTTYP(i) = 'LGUFM') **then**	Loguniform Distribution
$X(i) \leftarrow A(1,i)[A(2,i)/A(1,i)]^q$	Linear interpolation on a log scale.
else if (DSTTYP(i) = 'NORML') **then**	Normal Distribution
call INVNOR(q, y)	Invert standard normal distribution.
$X(i) \leftarrow A(1,i) + y \cdot A(2,i)$	Change location and scale for this distribution.
else if (DSTTYP(i) = 'PCUFM') **then**	Piecewise Uniform Distribution
$q \leftarrow (1\text{-}p)p_L(i) + p \cdot p_H(i)$	Interpolate between high and low probabilities.
call INVPU(q, *round*($A(1,i)$), *round*($A(2,i)$), $A(3,i)$, PUDPAR, $X(i)$)	Invert distribution.
else if (DSTTYP(i) = 'TRIAN') **then**	Triangular Distribution
call INVTRI(q, $[A(2,i)\text{-}A(1,i)]$ / $[A(3,i)\text{-}A(1,i)]$, y)	Invert standard triangular distribution.
$X(i) \leftarrow (1\text{-}y)\ A(1,i) + y \cdot A(3,i)$	Change location and scale for this distribution.
else if (DSTTYP(i) = 'UNIFM') **then**	Uniform Distribution
$X(i) \leftarrow (1\text{-}q)\ A(1,i) + q \cdot A(3,i)$	Interpolate to invert linear CDF.
end if	
end if	

9 ALGORITHMS FOR THE BETA AND BINOMIAL DISTRIBUTIONS

9.1 Overview of the Beta and Binomial Distribution Algorithms

The standard beta distribution is a flexible two-parameter distribution defined over the interval [0,1]. Figure 9.1/1 shows the variety of PDF shapes that can be generated by appropriate choices of the parameters a and b. The object-oriented and mathematical definitions of the Beta Distribution appeared in Sections 5.1 and 5.2. The Parameter Sampling Package (PSP) evaluates the CDF and the inverse CDF for this distribution using three interconnected routines: BTADIS, GAMMAL, and INVBTA (Figure 9.1/2). In addition, the Binomial Distribution CDF routine, BINDIS, calls the BTADIS routine. INVBIN, which inverts the binomial CDF, does not use a Beta Distribution routine.

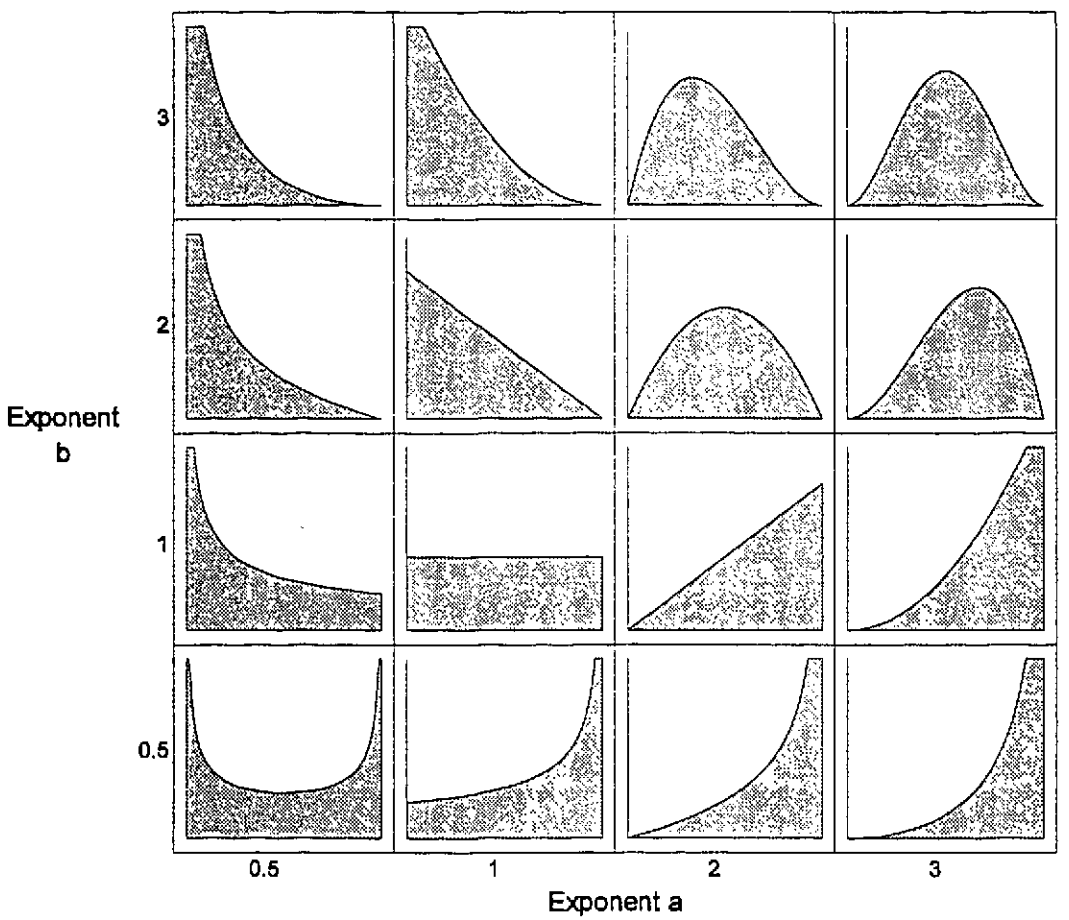

FIGURE 9.1/1: Sample Shapes for the Beta PDF

Figure 9.1/1 (a copy of Figure 5.1/2) shows the variety of shapes that the standard beta PDF can adopt. This flexibility stems from the two shape parameters, *a* and *b*, used by the distribution. It comes at a cost–the CDF, inverse CDF, and even the PDF use functions are difficult to evaluate using standard Fortran 77 function libraries.

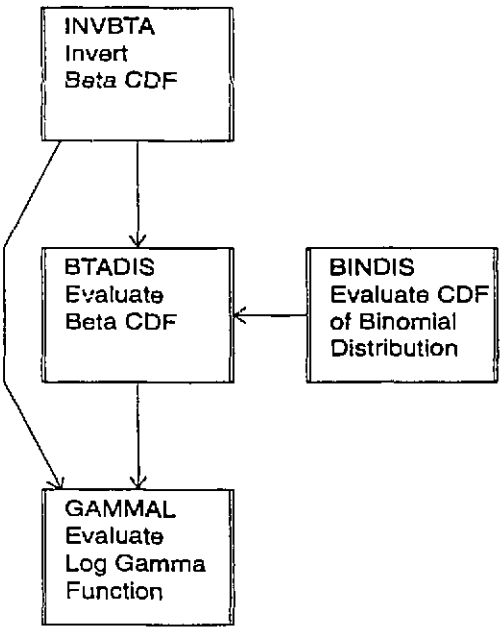

FIGURE 9.1/2: Beta Distribution Routines

There are five routines in the PSP to handle Beta Distribution and Binomial Distribution operations. Figure 9.1/2 (synthesized from Figures 3.1/1 and 3.1/2) shows the relationships among four of these routines. BTADIS and INVBTA evaluate the beta CDF and inverse CDF respectively. BINDIS evaluates the binomial CDF. INVBIN, which evaluates the inverse binomial CDF, is not shown since it does not call any other routines. INVBTA calls BTADIS because it uses Newton's method as described in Section 7.5 to refine estimates of the inverse CDF. Both BTADIS and INVBTA call GAMMAL, which evaluates the log gamma function. This routine is needed to evaluate the complete beta function $B(a,b)$, which appears in the PDF and CDF. Equation (5.2-2) in Section 5.2 defines $B(a,b)$ as a ratio of gamma functions. Since $\Gamma(x)$ gets large very rapidly with x, it is preferable to deal with log $\Gamma(x)$, to avoid floating-point overflow on a computer. There is currently no PSP routine to evaluate $B(a,b)$ directly, which is an oversight that should be corrected in the future.

To evaluate $\Gamma(x)$ for large x, GAMMAL uses Stirling's approximation with a rational correction term provided by Hart et al. (1978). For small x, GAMMAL uses an iteration from ACM Algorithm 291 by Pike and Hill (1966) to transform the variable to a large x.

BTADIS uses ACM Algorithm 179 by Ludwig (1963), with later corrections by Pike and Hill (1966), Bosten and Battiste (1973), and Pike and Soohoo (1975) to evaluate the standard beta CDF, also known as the incomplete beta ratio function.

INVBTA starts with an approximation of 0.5 and uses Newton's method, as described in Section 7.5, to produce successive refinement in the estimated inverse. A bisection technique ensures that the inverse is confined to steadily diminishing intervals.

The Binomial Distribution, described in Sections 5.3 and 5.4, is a discrete distribution. The CDF can be evaluated by finite sums. There is also a closed formula for the CDF (Press et al. 1986) in terms of the incomplete beta ratio function. The binomial CDF routine, BINDIS, uses this approximation, and calls BTADIS. In contrast, the inverse CDF routine, INVBIN, uses finite sums of probabilities since the inverse CDF is discontinuous.

9.2 GAMMAL: The Log Gamma Function

To evaluate $\Gamma(x)$ for large x, GAMMAL uses Stirling's approximation with a rational correction term provided by Hart et al. (1978). For small x, GAMMAL uses an iteration from ACM Algorithm 291 by Pike and Hill (1966) to transform the variable to a large x.

Algorithm 9.2/1 specifies how GAMMAL works. GAMMAL is a subroutine (not a function) with two arguments. The first argument specifies the value x at which the log gamma function is to be evaluated. The second argument returns the calculated value. This routine works only for positive values of x.

GAMMAL has been adapted from CACM Algorithm 291 by Pike and Hill (1966). The basic premise behind this algorithm is that Stirling's formula with a suitable correction term provides an accurate estimate of $\Gamma(x)$ for $x \geq 7$. A recurrence relation can convert smaller values of x to larger ones so that Stirling's formula can be applied for all positive x. The original algorithm provided an accuracy to about 10 decimal places. The implementation of the algorithm in GAMMAL uses a different correction term from the one in Algorithm 291. This approximation, from Hart et al. (1978), provides an accuracy to about 16 decimal places. The transition point between the two parts of the algorithm was raised from 7 to 8 to compensate for the higher accuracy demands.

Stirling's formula takes the form

$$\ln \Gamma(x) = \left(x - \frac{1}{2}\right) \ln x - x + \ln\sqrt{2\pi} + \xi(x) \ , \tag{9.2-1}$$

where $\xi(x)$ is a small correction term. Pike and Hill used a Laurent series (i.e., a power series in $1/x^2$) to estimate $\xi(x)$. Hart et al. recommend a rational approximation instead. GAMMAL uses function LGAM 5443 from Hart et al.:

$$\xi(x) = \frac{1}{x} \frac{\left[0.0691561607375687 \left(\frac{1}{x^2}\right) + 0.498030766924499634\right] \frac{1}{x^2} + 0.28811928393554601533}{\left(\frac{1}{x^2} + 6.09161691641660296\right) \frac{1}{x^2} + 3.4574314072267450698} \ , \tag{9.2-2}$$

The correction function in Equation (9.2-2) applies only for $x \geq 8$. For smaller x values, the following recursion offers a means of transforming the argument to the accepted range:

$$\Gamma(x+1) = x\Gamma(x) \ ,$$

or (9.2-3)

$$\ln\Gamma(x) = \ln\Gamma(x+1) - \ln x \ .$$

Algorithm 9.2/1 shows how these parts fit together.

ALGORITHM 9.2/1: Subroutine GAMMAL—Log Gamma Function

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
X	Double Precision	In	Argument of log gamma	> 0	0: lower bound 8: algorithm switch
LOGGAM	Double Precision	Out	Log gamma function value	= ln Γ(X) > -1	

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
LOGGAM ← 0	Initialize result
while (X < 8)	Loop to get X large enough for Stirling's algorithm
LOGGAM ← LOGGAM - ln X	Correct result using Equation (9.2-3)
X ← X + 1	Increment X
end while	
LOGGAM ← LOGGAM + ln Γ(X)	Use Equation (9.2-1) to evaluate the ln Γ(X) function, with the correction term from Equation (9.2-2)

9.3 BTADIS: The Beta CDF or Incomplete Beta Ratio Function

BTADIS uses ACM Algorithm 179 by Ludwig (1963) with later corrections by Pike and Hill (1966), Bosten and Battiste (1973), and Pike and Soohoo (1975) to evaluate the standard beta CDF, also known as the incomplete beta ratio function.

Algorithm 9.3/1 summarizes ACM Algorithm 179, as corrected by several authors. Bosten and Battiste (1973) explain the ACM Algorithm as follows (equation numbers added).

- Algorithm 179 (modified to include the remark by M.C. Pike and I.D. Hill (1966)) computes the Incomplete Beta Ratio using

$$I_x(p,q) = \frac{INFSUM \cdot x^p \cdot \Gamma(PS+p)}{\Gamma(PS) \cdot \Gamma(p+1)} + \frac{x^p \cdot (1-x)^q \cdot \Gamma(p+q)\ FINSUM}{\Gamma(p) \cdot \Gamma(q+1)} \quad . \tag{9.3-1}$$

- *INFSUM* and *FINSUM* represent two series summations defined as follows:

$$INFSUM = \sum_{i=0}^{\infty} \frac{(1-PS)_i \cdot p}{p+i} \frac{x^i}{i!} \quad \text{, where} \tag{9.3-2}$$

$$\begin{aligned} (1-PS)_i &= 1 && [i = 0] \\ &= (1-PS) \cdot (2-PS) \cdots (i-PS) = \frac{\Gamma(1+i-PS)}{\Gamma(1-PS)} && [i > 0] \end{aligned} \tag{9.3-3}$$

$$FINSUM = \sum_{i=1}^{[q]} \frac{q(q-1)\cdots(q-i+1)}{(p+q-1)(p+q-2)\cdots(p+q-i)} \frac{1}{(1-x)^i} \tag{9.3-4}$$

where [q] is equal to the largest integer less than q. If [q] = 0, then FINSUM = 0. PS is defined as

$$\begin{aligned} &PS = 1, \text{ if q is an integer; otherwise} \\ &= q - [q] \ . \end{aligned} \tag{9.3-5}$$

In this quotation, p is the same as a in this report, and q is the same as b. The names *INFSUM* and *FINSUM* appear in the Fortran code given with Algorithm 179, but they do not have the same definitions as in Equations (9.3-2) and (9.3-4). Instead, *INFSUM* refers to the entire first term of Equation (9.3-1), and *FINSUM* refers to the second term. BTADIS, which is very similar to the published code, retains this convention.

BTADIS transforms its arguments using the following equality to obtain better convergence:

$$I_x(a,b) = 1 - I_{1-x}(b,a) \quad . \tag{9.3-6}$$

ALGORITHM 9.3/1: Subroutine BTADIS—Beta Distribution CDF

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
SHPBT1	Double Precision	In	Shape attribute a	> 0	0: lower bound
SHPBT2	Double Precision	In	Shape attribute b	> 0	0: lower bound
STDVAL	Double Precision	In	Function argument x	–	0: algorithm change 1: algorithm change
BTAPRB	Double Precision	Out	Cumulative probability p	$0 \le p \le 1$	0: lower bound 1: upper bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. BTAPRB = $I_x(a,b)$

Algorithm	*Commentary*
if (STDVAL ≤ 0) **then**	The PDF is nonzero only between 0 and 1.
BTAPRB ← 0	Below 0, the CDF is 0.
else if (STDVAL ≥ 1) **then**	Above 1, the CDF is 1.
BTAPRB ← 1	
else	
if (STDVAL > 0.5) **then**	If the argument is relatively large ...
LARGEX ← *true*	Transform the arguments for a more efficient power
STDVAL ← 1 - STDVAL	series, using Equation (9.3-6).
loca ← *b*; *locb* ← *a*	Exchange local values of a and b.
else	
LARGEX ← *false*	Set flag for later.
loca ← *a*; *locb* ← *b*	
end if	
if (*locb* = integerpart(*locb*)) **then**	Use Equation (9.3-5) to calculate PS.
PS ← 1	
else	
PS ← *locb* - integerpart(*locb*)	
end if	
XB ← *loca* log *x* + log Γ(PS+*loca*) - log Γ(PS) - log Γ(*loca*+1)	Log of the multiplier of INFSUM in Equation (9.3-1). Use calls to GAMMAL.
if (XB > *log of the smallest representable number*) **then**	Sum until terms stop affecting result; use Equation
INFSUM ← *Sum of infinite series* * e^{XB}	(9.3-2) for sum of the series.
else	
INFSUM ← 0	Empty summation.
end if	
if (*locb* > 1) **then**	XB is log of the multiplier of FINSUM in Equation
XB ← *loca* log *x* + *locb* log (1-*x*) + log Γ(*loca*+*locb*) - log Γ(*loca*) - log Γ(*locb*+1)	(9.3-1). Use calls to GAMMAL. Use Equation (9.3-4) for FINSUM. Early and late terms in sum may underflow; scale terms to keep them representable.
FINSUM ← *Sum of finite series* * e^{XB}	
else	
FINSUM ← 0	Empty summation.
end if	
BTAPRB ← INFSUM + FINSUM	Unlike Equation (9.3-1), both INFSUM and FINSUM
if (LARGEX) **then**	already include the coefficients.
BTAPRB ← 1 - BTAPRB	Transform back using Equation (9.3-6).
end if	
end if	

9.4 INVBTA: Inverse of the Beta CDF

INVBTA inverts CDF of the standard beta distribution to yield a value between 0 and 1. INVBTA starts with an approximation of 0.5 and uses Newton's method, as described in Section 7.5, to produce successive refinements in the estimated inverse. A bisection technique ensures that the inverse is confined to steadily diminishing intervals.

Newton's method, as described in Section 7.5, is a general method for finding zeros of smooth, continuous functions. In this case, Newton's method finds a zero of the function $I_x(a,b) - p$, where p is the probability value at which the beta CDF $I_x(a,b)$ (defined in Equation (5.2-4), Section 5.2) is to be inverted. Newton's method converges quickly when started with a good estimate of the zero, except with pathological functions. In a typical case, Newton's method doubles the number of correct significant figures (or the number of correct bits in the binary representation) with each iteration. However, the whole iteration can diverge if the initial estimate of the zero is not sufficiently accurate.

In contrast, successive bisection is a highly reliable, but rather slow, method of finding the zero of a continuous function, once you have an interval containing precisely one zero. The procedure is simple:

(1) Evaluate the function at the endpoints of the interval of interest. One function value should be positive and the other negative.

(2) **While** the current interval is too long
(3) Evaluate the function at the midpoint of the interval.
(4) Find the new subinterval (either the left half or the right) that contains the zero.
(5) **End while.**

(6) Set the final estimate to the midpoint of the final interval.

Each iteration through the *while* loop in this algorithm requires one function evaluation and adds one correct bit to the estimate of the zero.

The combined algorithm uses Newton's method to yield new estimates, and uses the bisection method to prevent Newton's method from diverging if the starting estimate is too crude. The combined algorithm still requires several evaluations of the beta CDF. To improve performance, INVBTA limits the resolution of the result to the minimum required by the specifications, namely one part in 10^8. Algorithm 9.4/1 shows the combined algorithm in detail.

ALGORITHM 9.4/1: Subroutine INVBTA—Inverse Beta Distribution CDF

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANTL	Double Precision	In	Cumulative probability p	$0 \le p \le 1$	0: lower bound 1: upper bound
SHPBT1	Double Precision	In	Shape attribute a	> 0	0: lower bound
SHPBT2	Double Precision	In	Shape attribute b	> 0	0: lower bound
BTAVLU	Double Precision	Out	Beta variate value x	$0 \le x \le 1$	–

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. $I_x(a,b) = p$

Algorithm	*Commentary*
if (QUANTL ≤ 0) **then** BTAVLU ← 0	The only allowed case here is $p = 0$, which yields the left end of the interval.
else if (QUANTL ≥ 1) then BTAVLU ← 1	*The only allowed case here is $p = 1$, which yields* the right end of the interval.

else
  BTAVLU ← 0.5
  LOBND ← 0
  HIBND ← 1
	Use combination of Newton's method and bisection.
	Initial estimate.
	Initial interval endpoints: [LOBND, HIBND]
	We know that CDF(LOBND)-QUANTL < 0
	and CDF(HIBND)-QUANTL > 0

call BTADIS(a, b, BTAVLU, CDFX) while (HIBND-LOBND > 10^{-8} *and* |CDFX-QUANTL| > 10^{-8})	Find CDF at 0.5. Repeat while neither interval length nor fit to QUANTL are good enough.
if (CDFX > QUANTL) then HIBND ← BTAVLU	Pick left-hand half interval.
else LOBND ← BTAVLU	Pick right-hand half interval
end if	
PDFX ← exp(log Γ(a+b) - log Γ(a) - log Γ(b) + (a-1) log x + (b-1) log (1-x))	Use Equation (5.2-1) in Section 5.2 to evaluate the beta pdf.
BTAVLU ← BTAVLU - (CDFX - QUANTL) / PDFX	Apply Newton's method to find new estimate of the zero.
BTAVLU ← *min*(BTAVLU, HIBND) BTAVLU ← *max*(BTAVLU, LOBND)	Restrict new estimate to lie within the current interval.
W ← 0.8 + 0.2 (1 - |CDFX - QUANTL|)^{10}	W is a weight that depends on how well CDFX matched QUANTL last time; it lies in [0.8,1.0].
BTAVLU ← W * BTAVLU + (1 - W) * (LOBND + HIBND)/2	The new point is a weighted average of midpoint and corrected Newton's method estimate.
call BTADIS(a, b, BTAVLU, CDFX)	Find CDF at new point.
end while	
end if	

9.5 BINDIS: The Binomial CDF

The Binomial Distribution, described in Sections 5.3 and 5.4, is a discrete distribution. The CDF can be evaluated by finite sums. There is also a closed formula for the CDF (Press et al. 1986), in terms of the incomplete beta ratio function. The binomial CDF routine, BINDIS, uses this approximation, and calls BTADIS.

Equation (5.4-2) in Section 5.4 provides an expression to compute $g_j(n,p)$, the probability of achieving precisely j successes in a set of n independent Bernoulli trials, each with a probability p of success:

$$g_j(n,p) = \binom{n}{j} p^j (1-p)^{n-j} \qquad 0 \le j \le n,\ 0 \le p \le 1, \tag{9.5-1}$$

where $\binom{n}{j} = \frac{n!}{j!(n-j)!}$ is the number of distinct combinations of n things taken j at a time.
The following recursive relationship, which can easily be derived from the defining equation, provides an efficient means of evaluating a sequence of these binomial probabilities for $p < 1$:

$$\begin{aligned} g_0(n,p) &= (1-p)^n \\ g_j(n,p) &= g_{j-1}(n,p)\frac{(n+1-j)p}{j(1-p)} \end{aligned} \qquad \text{for } j \text{ from 1 to } n. \tag{9.5-2}$$

Cumulative probabilities $F_k(n,p)$ for integer k values are summations of these probabilities from 0 to k, as established by Equation (5.4-3) in Section 5.4:

$$F_k(n,p) = \sum_{j=0}^{k} g_j(n,p) \qquad \text{for } k \text{ from 0 to } n. \tag{9.5-3}$$

An algorithm to evaluate the binomial CDF could easily be constructed on the basis of these equations. It would involve few operations for small k or n, and many more operations for large n and large k. That is the basis of Algorithm 9.5/1, to invert the binomial CDF.

Equation (5.4-4) in Section 5.4 provides a simpler alternative for calculating a single value of the CDF:

$$F_k(n,p) = \sum_{j=0}^{k} g_j(n,p) = I_{1-p}(n-k,k+1) \qquad 0 \le k \le n,\ 0 \le p \le 1. \tag{9.5-4}$$

One function call to BTADIS returns the cumulative probability directly. The algorithm must treat the case where $k = n$ specially since the incomplete beta ratio function evaluated by BTADIS would otherwise receive a shape parameter value of 0, which lies outside the region in which the algorithm applies. Algorithm 9.5/1 shows how BINDIS can handle this approach.

In some cases it would be more efficient to use the summation technique in BINDIS (for example, when $j = 0$). Press et al. (1986) say that for n "larger than a dozen or so," the beta probability technique is better. For smaller n, "either method is acceptable." BINDIS uses only the beta probability technique to keep the code simple and robust.

ALGORITHM 9.5/1: Subroutine BINDIS—Binomial Distribution CDF

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
VARVAL	Double Precision	In	Number of successes k	$0 \le k \le n$	0: lower bound n: upper bound
NTRIAL	Double Precision	In	Number of Bernoulli trials n	$0 \le n$	0: lower bound
SPROB	Double Precision	In	Probability p of success in each trial	$0 \le p \le 1$	0: lower bound 1: upper bound
CPROB	Double Precision	Out	Cumulative probability P	$0 \le P \le 1$	

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. $P = F_k(n,p)$

Algorithm	*Commentary*
if ($k < 0$) **then**	
$P \leftarrow 0$	Probability of negative number of successes is 0.
else if ($k \ge n$) **then**	
$P \leftarrow 1$	Probability of up to n or more successes is 1.
else	
call BTADIS(k+1, n-k, p, P)	Calculate probability using Equations (9.5-4) from Section 9.5 and Equation (9.3-6) from Section 9.3.
$P \leftarrow 1 - P$	
end if	

9.6 INVBIN: Inverse of the Binomial CDF

The Binomial Distribution CDF can be evaluated by finite sums, as Section 9.5 shows. The inverse CDF can evaluate the same sums to determine the inverse. The closed formula for the CDF used in BINDIS is not appropriate for the inverse routine, however, since the distribution is discrete and a discontinuous CDF is difficult to invert using smooth functions. The inverse CDF routine, INVBIN, uses the finite sums approach.

Algorithm 9.6/1 shows how Equation (9.5-2) from Section 9.5 can be used to determine cumulative probabilities efficiently. The iteration that generates these cumulative probabilities stops when the input probability has been reached. The number of the last iteration is the inverse of the CDF.

The inverse calculation can be affected by underflow. For example, the initial probability of zero successes may be too small to represent in a computer's floating-point notation. If the initial probability is incorrectly taken as zero, then all the probabilities will be evaluated as zero because of the recursive formula. This difficulty can be circumvented in at least two ways. First, all the probabilities can be scaled, perhaps by factoring out powers of $(1 - p)$. As the iteration proceeds, values will need to be rescaled. Alternatively, the routine could store logarithms of probabilities to preserve significance. These options do not appear in Algorithm 8.6/1, since nobody has expressed a requirement for handling Binomial Distributions with large numbers of trials, where the underflow problem would arise.

ALGORITHM 9.6/1: Subroutine INVBIN—Inverse Binomial Distribution CDF

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANTL	Double Precision	In	Cumulative probability P	$0 \le P \le 1$	0: lower bound 1: upper bound
NTRIAL	Double Precision	In	Number of Bernoulli trials n	$0 \le n$	0: lower bound
SPROB	Double Precision	In	Probability p of success in each trial	$0 \le p \le 1$	0: lower bound 1: upper bound
BINVLU	Double Precision	Out	Number of successes k	$0 \le k \le n$	

Precondition	Input arguments satisfy their constraints.

Postcondition	Output arguments satisfy their constraints.
	If ($p = 0$) then $k = 0$,
	else if ($p = 1$) then $k = n$,
	else
	if ($P = 0$) then $k = 0$,
	else the probability of k-1 successes $< P \le$ probability of k successes.

Algorithm	*Commentary*
if ($p \le 0$) **then**	No chance for successful trials, so the number of successes is 0.
$k \leftarrow 0$	
else if ($p \ge 1$) **then**	Every trial is a success, so the number of successes is NTRIAL.
$k \leftarrow n$	
else	Use recursion to find the number of successful trials.
LSTPRB $\leftarrow (1 - p)^n$	Initialize probability.
CUMPRB $\leftarrow$ LSTPRB	Initialize cumulative probability.
$k \leftarrow 0$	Start with BINVLU at 0.
while (k < n *and* CUMPRB < P)	Loop until cumulative probability reaches QUANTL.
$k \leftarrow k + 1$	
LSTPRB $\leftarrow$ LSTPRB$[(n + 1 - k)p]/[k(1 - p)]$	Apply recursion in Equation (9.5-2), Section 9.5, to update LSTPRB
CUMPRB $\leftarrow$ CUMPRB + LSTPRB	Update cumulative probability.
end while	
end if	

10 ALGORITHMS FOR THE CONSTANT, TRIANGULAR, AND UNIFORM FAMILY OF DISTRIBUTIONS

10.1 Overview of Algorithms

The Uniform family of distributions includes the Uniform, Loguniform and Piecewise Uniform Distributions. They are usually ad hoc *distributions, applied for practical purpose when there is little theoretical basis for the precise shape of the distribution. The Triangular Distribution, which is not otherwise related to the Uniform Distribution, is also an* ad hoc *distribution. Like the other members of the family, it has a finite domain. Unlike the Uniform and Piecewise Uniform Distributions, it has a unique mode. The Constant Distribution can be considered as a member of this family because it is the same as a Uniform (or Loguniform or Triangular) Distribution with zero width. Algorithms for the Uniform, Loguniform and Constant Distributions are simple. The CDF and inverse CDF can be computed inside TRAVAL and TRAQUA respectively. The Piecewise Uniform and Triangular Distributions have their own CDF and inverse CDF routines (Figure 10.1/1).*

Figure 10.1/1 shows that TRAVAL and TRAQUA contain embedded code to evaluate and invert the Constant, Uniform and Loguniform Distributions. Algorithms for these routines in Chapter 8 specify the CDF calculations. Table 10.1/1 summarizes the formulas used. These come directly from the equations in Chapter 5, and they summarize the relevant parts of Algorithms 8.7/1 and 8.8/1.

TABLE 10.1/1

ALGORITHMS FOR EVALUATING AND INVERTING CDFs

Distribution Type	*Attributes*	$F(x)$	$F^{-1}(p)$
Constant	Constant Value: a	if ($x < a$) then 0 else 1	a
Loguniform	Lower Limit: a Upper Limit: b	if ($x < a$) then 0 else if ($x \geq b$) then 1 else $\ln(x/a)/\ln(b/a)$	if ($p \leq 0$) then a else if ($p \geq 1$) then b else $\exp[(1-p)\ln a + p \cdot \ln b]$
Triangular	Lower Limit: a Mode: b Upper Limit: c	if ($x < a$) then 0 else if ($x \geq c$) then 1 else if ($x < b$) then $(x-a)^2/[(b-a)(c-a)]$ else $1-(c-x)^2/[(c-b)(c-a)]$	if ($p \leq 0$) then a else if ($p \geq 1$) then c else if ($p < (b-a)/(c-a)$) then $a + [p(c-a)(b-a)]^{0.5}$ else $c - [(1-p)(c-a)(c-b)]^{0.5}$
Uniform	Lower Limit: a Upper Limit: b	if ($x < a$) then 0 else if ($x \geq b$) then 1 else $(x-a)/(b-a)$	if ($p \leq 0$) then a else if ($p \geq 1$) then b else $(1-p)a + p \cdot b$

Figure 10.1/1 also shows that the Triangular Distribution requires calls to TRIDIS and INVTRI to perform CDF and inverse CDF calculations. Since the Triangular Distribution PDF is piecewise linear, the CDF is piecewise quadratic. TRIDIS and INVTRI use straightforward formulas that are also shown in Table 10.1/1. TRIDIS and INVTRI assume a standard triangular distribution in which $a = 0$ and $c = 1$, which simplifies the calculations somewhat.

The Piecewise Uniform Distribution has routines PUDIS and INVPU to perform CDF and inverse CDF calculations. Both these routines perform their calculations in two steps. The first step identifies the Uniform Distribution within the Piecewise Uniform Distribution that applies. The second step performs calculations relative to this Uniform Distribution, using the expressions in Table 10.1/1. These routines are described in more detail in Section 10.2.

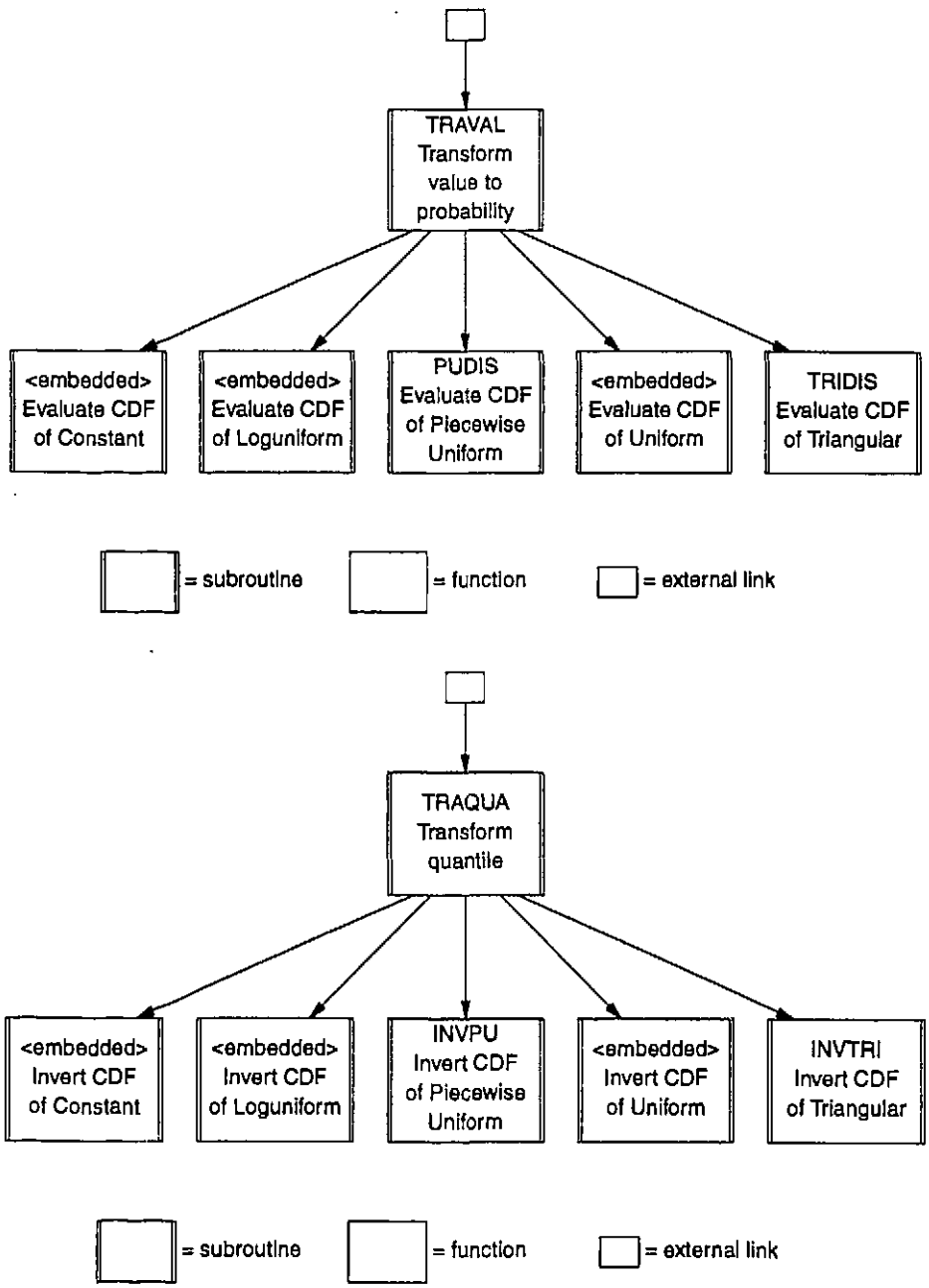

FIGURE 10.1/1: Routines in the Constant, Triangular, and Uniform Family

10.2 TRIDIS and INVTRI: Routines for the Triangular Distribution

Algorithms 10.2/1 and 10.2/2 show the algorithms for TRIDIS and INVTRI, used to calculate the CDF and inverse CDF for a standard Triangular Distribution. Since the Triangular Distribution PDF is piecewise linear, the CDF is piecewise quadratic. The inverse CDF requires a square root since it finds the root of a quadratic equation. TRIDIS and INVTRI use straightforward formulas that appeared in Table 10.1/1.

The routines TRIDIS and INVTRI evaluate and invert the CDF of a standard Triangular Distribution, which ranges from 0 to 1, with a mode M somewhere between. The major routines TRAVAL and TRAQUA call these routines. Other programs wishing to avoid the overhead of the checks performed by these routines can call TRIDIS and INVTRI directly. In such a case, the calling routine must ensure that the mode lies in the appropriate interval.

ALGORITHM 10.2/1: Subroutine TRIDIS—CDF of a Standard Triangular Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
VARVAL	Double Precision	In	Value x in standard triangular distribution	–	0: lower bound 1: upper bound
MODE	Double Precision	In	Mode M of standard distribution	$0 \le M \le 1$	0: lower bound 1: lower bound
TRIPRB	Double Precision	Out	Cumulative probability p	$0 \le p=F(x) \le 1$ where F is the triangular CDF	0: lower bound 1: upper bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($x \le 0$) **then** $p \leftarrow 0$	No chance of values below 0.
else if ($x \ge 1$) **then** $p \leftarrow 1$	All values lie below 1.
else if ($x < M$) **then** $p \leftarrow x^2/M$	Piecewise quadratic to the left of the mode.
else $p \leftarrow 1-(1-x)^2/(1-M)$	Piecewise quadratic to the right of the mode.
end if	

ALGORITHM 10.2/2: Subroutine INVTRI—Inverse CDF of a Standard Triangular Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANTL	Double Precision	In	Cumulative probability p	$0 \le p \le 1$	0: lower bound 1: upper bound
MODE	Double Precision	In	Mode M of standard distribution	$0 \le M \le 1$	0: lower bound 1: lower bound
TRIVLU	Double Precision	Out	Value x in a standard triangular distribution	$0 \le x \le 1$; $F(x) = p$, where F is the triangular CDF	0: lower bound 1: upper bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*

if ($p \le 0$) **then** $x \leftarrow 0$
else if ($p \ge 1$) **then** $x \leftarrow 1$
else if ($p < M$) **then** $x \leftarrow (pM)^{0.5}$
else $x \leftarrow 1-[(1-p)(1-M)]^{0.5}$
end if
	No chance of values below 0.
	All values lie below 1.
	Probability of not exceeding mode M is also M. Invert piecewise quadratic to the right and left of the mode.

10.3 PUDIS and INVPU: Routines for the Piecewise Uniform Distribution

Algorithms 10.3/1 and 10.3/2 show how to calculate the CDF and inverse CDF for the Piecewise Uniform Distribution. The key step is to find the applicable Uniform Distribution, and to transform probabilities appropriately for that distribution. With suitable transformations, calculations then reduce to those for a Uniform Distribution.

ALGORITHM 10.3/1: Subroutine PUDIS—CDF for a Piecewise Uniform Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
STDVAL	Double Precision	In	Value x in distribution	–	$\{Q(i,q+j)\}$ for $i \in \{1, 2\}$, $j \in \{1, 2, \ldots n\}$: transition points
PUPTRS	Integer	In	Pointer q to columns of PUDPAR	$q \geq 0$	0: lower bound
NUMCLS	Integer	In	Number n of columns in PUDPAR used	$n \geq 1$	1: lower bound
TOTWGT	Double Precision	In	Total weight W from all components	$W > 0$ $= \sum_{l=q+1}^{q+n} Q(3,l)$	0: lower bound
PUDPAR	Double Precision	In	Piecewise Uniform detailed attributes $Q(k,l)$	See Table 3.5/3	$Q(1,l) = Q(2,l)$ $Q(2,l) = Q(1,l+1)$ $Q(3,l) = 0$
PUPRB	Double Precision	Out	Cumulative probability p	$0 \leq p = F(x) \leq 1$ where F is the CDF	0: lower bound 1: lower bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($x \leq Q(1,q+1)$) **then** $p \leftarrow 0$	No chance of values to the left of all ranges.
else if ($x \geq Q(2,q+n)$) **then** $p \leftarrow 1$	All values lie to the left of right-hand end.
else *leftwt* $\leftarrow 0$	Between extremes; find where.
bin $\leftarrow q+1$	Initialize loop counter.
while (*bin* $< q+n$ and $Q(2,bin) < x$) **do**	Loop invariants:
leftwt $\leftarrow$ *leftwt* + $Q(3,bin)$	- *bin* lies between $q+1$ and $q+n-1$
bin $\leftarrow$ *bin* + 1	- x lies to right of current bin
end while	- *leftwt* is total weight of bins left of current bin.
if ($x < Q(1,bin)$) **then** $p \leftarrow$ *leftwt* / W	x could lie between bins where probability is constant.
else *rightwt* $\leftarrow$ *leftwt* + $Q(3,bin)$	*rightwt* is cumulative weight including current bin.
if ($x = Q(2,bin)$) **then** $p \leftarrow$ *rightwt* / W	This line is needed in case bin has zero width.
else *ratio* $\leftarrow (x-Q(1,bin)) / (Q(2,bin)-Q(1,bin))$	Relative position inside bin.
$p \leftarrow$ (*ratio*·*rightwt* + (1-*ratio*)·*leftwt*) / W	Linear interpolation.
end if	
end if	
end if	

ALGORITHM 10.3/2: Subroutine INVPU—Inverse CDF for a Piecewise Uniform Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANTL	Double Precision	In	Cumulative probability p	$0 \le p \le 1$	0: lower bound 1: upper bound Probabilities at endpoints of bins
PUPTRS	Integer	In	Pointer q to columns of PUDPAR	$q \ge 0$	0: lower bound
NUMCLS	Integer	In	Number n of columns in PUDPAR used	$n \ge 1$	1: lower bound
TOTWGT	Double Precision	In	Total weight W from all components	$W > 0$ $= \sum_{l=q+1}^{q+n} Q(3,l)$	0: lower bound
PUDPAR	Double Precision	In	Piecewise Uniform detailed attributes $Q(k,l)$	See Table 3.5/3	$Q(1,l) = Q(2,l)$ $Q(2,l) = Q(1,l+1)$ $Q(3,l) = 0$
PUVLU	Double Precision	Out	Value x in Piecewise Uniform Distribution	$F(x) = p$ where F is the CDF	endpoints of bins

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($p \le 0$) **then** $x \leftarrow Q(1,q+1)$	No chance of values to the left of all ranges.
else if ($p \ge 1$) **then** $x \leftarrow Q(2,q+n)$	All values lie to the left of right-hand end.
else $bin \leftarrow q+1$	Between extremes; find where. *bin* is a pointer to the current column of Q. *leftwt* and *rightwt* are cumulative weights to the left and right ends of this bin.
$leftwt \leftarrow 0$	
$rightwt \leftarrow Q(3,bin)$	
$targetweight \leftarrow p \cdot W$	*targetweight* is p converted to a weight.
while ($bin < q+n$ and $rightwt < targetweight$) **do**	Loop invariants at start of loop:
$bin \leftarrow bin + 1$	- *bin* lies between $q+1$ and $q+n-1$
$leftwt \leftarrow rightwt$	- x lies to right of current bin
$rightwt \leftarrow rightwt + Q(3,bin)$	- *leftwt* is total weight of bins left of current bin.
end while	- *rightwt* is total weight of bins including current bin.
if ($Q(1,bin) = Q(2,bin)$) **then** $x \leftarrow Q(1,bin)$	Pick the location of the current zero-width bin.
else $ratio \leftarrow (targetweight - leftwt) / Q(3,bin)$	Relative position inside bin.
$x \leftarrow ratio \cdot Q(2,bin) + (1-ratio) \cdot Q(1,bin)$	Linear interpolation.
end if	
end if	

11 ALGORITHMS FOR THE NORMAL FAMILY OF DISTRIBUTIONS

11.1 Overview of Normal Family Algorithms

The normal family of distributions includes the Normal, Lognormal, Correlated Normal and Correlated Lognormal Distributions. These infinite or semi-infinite distributions have sound theoretical justifications for their use. For example, a Normal Distribution represents the limiting case when a variate is the sum of a large number of random additive influences. Each distribution is based on a Normal Distribution through some transformation. Figure 11.1/1 shows the connection among the routines used in evaluating CDFs and inverse CDFs. The CDF of a Normal Distribution can appear as an error function erf() *or a complementary error function* erfc(). *The Parameter Sampling Package implements these and other related routines that can be used as utility functions in other applications.*

Table 11.1/1 shows how each distribution type in the family relates to the Normal Distribution type. For example, it shows that the logarithm of a variable with a Lognormal Distribution is normally distributed. The attributes of each such Normal Distribution stem from the attributes of the original distribution according to simple formulas shown in the table. The cases where the correlated independent distribution is lognormal rather than normal do not appear in the table. They are very similar to the ones shown (see Section 8.8).

TABLE 11.1/1

RELATIONSHIP OF EACH DISTRIBUTION TYPE WITH THE NORMAL DISTRIBUTION

Distribution Type	*Attributes of Distribution of X*		*Normal Form of Variate*	*Attributes of Related Normal Variate Given Independent Normal Variate Y=y*	
Normal	Mean	μ_X	X	Mean	μ_X
	Standard Deviation	σ_X		Standard Deviation	σ_X
Correlated Normal	Mean	μ_X	X	Mean	$\mu_X + \rho_{XY}(y-\mu_Y)\sigma_X / \sigma_Y$
	Standard Deviation	σ_X		Standard Deviation	$\sigma_X (1-\rho_{XY})^{0.5}$
	Correlation Coefficient	ρ_{XY}			
Lognormal	Geometric Mean	gm_X	$\ln X$	Mean	$\ln gm_X$
	Geometric Standard Deviation	gsd_X		Standard Deviation	$\ln gsd_X$
Correlated Lognormal	Geometric Mean	gm_X	$\ln X$	Mean	$\ln(gm_X) + \rho_{XY}(y-\mu_Y)\cdot \ln(gsd_X) / \sigma_Y$
	Geometric Standard Deviation	gsd_X			
	Correlation Coefficient	ρ_{XY}		Standard Deviation	$\ln(gsd_X) (1-\rho_{XY})^{0.5}$

Figure 11.1/1 shows the structure of the routines that support the normal family of distributions. TRAVAL and TRAQUA appear at the top as main routines for evaluating and inverting CDFs. The routines for evaluating and inverting Normal and Correlated Normal Distribution CDFs appear in the figure as well, although TRAVAL evaluates the Correlated Normal Distribution CDF internally (Section 8.7). TRAVAL and TRAQUA (Section 8.8)

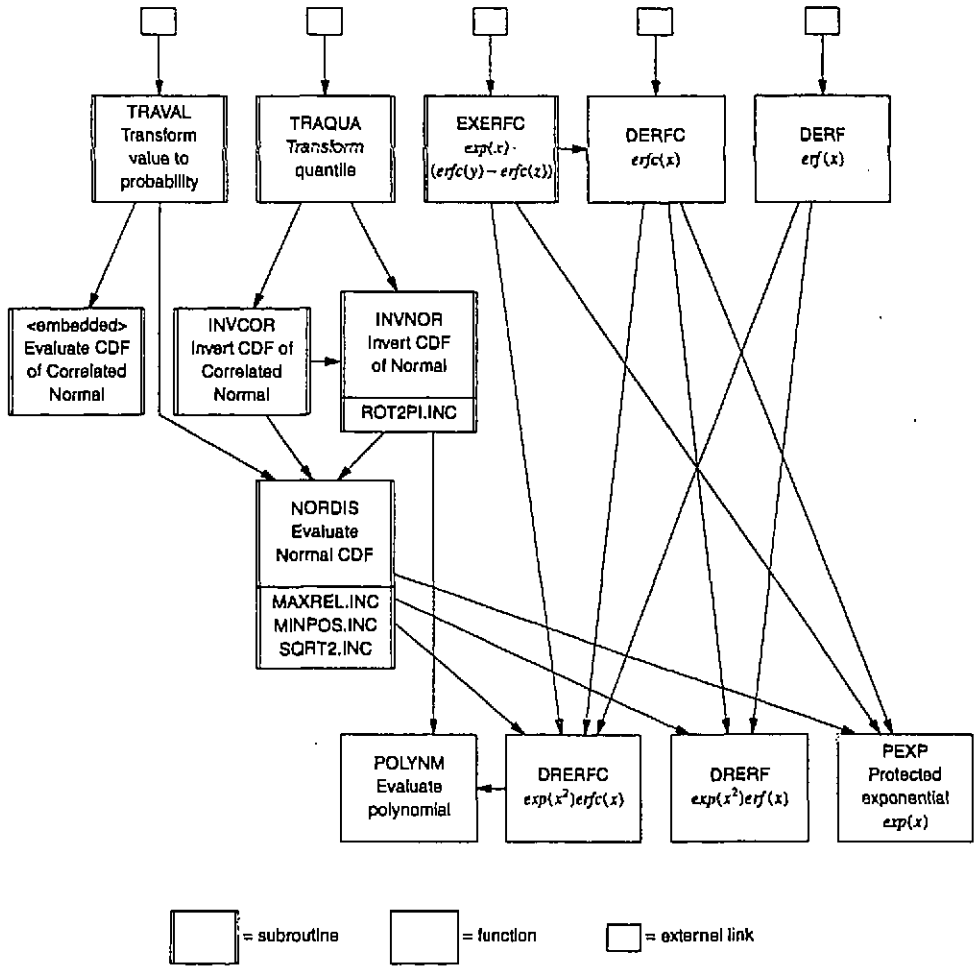

FIGURE 11.1/1: Routines in the Normal Family

perform all of the Lognormal and Correlated Lognormal Distribution CDF calculations internally as well. These do not appear at all in the figure to keep it relatively simple.

NORDIS is a key module in all these calculations, because all of the other distribution types can be expressed in terms of the Normal Distribution, as shown by Table 11.1/1. NORDIS transforms the argument to an appropriate region, and then separates the calculation of the CDF into an exponential part and a correction factor, as recommended by Hart et al. (1978). DRERF and DRERFC evaluate the correction factors. They use a variety of algorithms, depending on the magnitude of the argument.

DERF and DERFC are functions to evaluate the error function erf(x) and complementary error function erfc(x). The algorithms for those are similar to that of NORDIS, and they call the same routines. EXERFC evaluates a three-argument function that turns up in solving nuclide transport problems: e^z(erfc(y) - erfc(z)). Direct computation of this function is difficult because the difference can be extremely small compared to the magnitudes of the terms, and the exponential factor can be very large. EXERFC evaluates the function accurately if it has a significant value, by transforming arguments and calling DRERFC.

11.2 General Strategy for Evaluating Normal Distribution and Error Functions

The routines NORDIS, DERF, DERFC and EXERFC form a closely related nucleus for the operations carried out for the normal family of distributions. Hart et al. (1978) discuss the implementation of the error function erf() *and complementary error function* erfc() *in the real domain. They recommend partitioning the domain of the functions into regions where different approaches are used, as shown in Table 11.2/1. A similar partitioning works for NORDIS and EXERFC, as shown in Table 11.2/2. Lower level routines, DRERF and DRERFC, provide building blocks to handle the calculations in these regions.*

First consider the evaluation of erf(x) and erfc(x). The domain of erf(x) can be restricted to nonnegative values, since erf(x) is an odd function:

$$\text{erf}(-x) \equiv -\text{erf}(x) \; . \tag{11.2-1}$$

The following identity relates the functions erf(x) and erfc(x) to one another:

$$\text{erf}(x) + \text{erfc}(x) \equiv 1 \; . \tag{11.2-2}$$

For maximum accuracy in evaluating erf(x) and erfc(x), the smaller (in absolute magnitude) should be evaluated directly, and the other function can be found by subtracting that value from 1. Which function to evaluate depends on the value of x. Using these two equations, Hart et al. recommended that the domain be divided as shown in Table 11.2/1.

TABLE 11.2/1

EXPRESSIONS FOR ACCURATE EVALUATION Of erf(x) AND erfc(x)

x	erf(x)	erfc(x)
$x < -0.5$	-[1 - erfc(-x)]	2 - erfc(-x)
$-0.5 \le x < 0$	-erf(-x)	1 + erf(-x)
$0 \le x < 0.5$	erf(x)	1 - erf(x)
$0.5 \le x$	1 - erfc(x)	erfc(x)

The standard normal CDF, $\Phi_x(0, 1)$, can also have its domain broken down in this way because it has a linear relationship with both erf(x) and erfc(x), as shown by the following equations adapted from Section 5.11:

$$\Phi_x(0,1) \equiv \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \equiv \frac{1}{2}\left[2 - \text{erfc}\left(\frac{x}{\sqrt{2}}\right)\right] \; . \tag{11.2-3}$$

The routine EXERFC, which calculates the function $e^a(\mathrm{erfc}(x) - \mathrm{erfc}(y))$, introduces an additional requirement. It is not uncommon in transport solutions that the first factor e^a generates a floating-point overflow, while the erfc terms each cause a floating-point underflow. The result of the entire expression may neither underflow nor overflow, but may be a representable floating-point number. To allow cancellation of exponents in this situation, one would like to express erfc(x) as:

$$\mathrm{erfc}(x) \equiv e^{-x^2} r(x) \quad , \tag{11.2-4}$$

thereby defining a new function $r()$. Then,

$$e^a \mathrm{erfc}(x) \equiv e^{a-x^2} r(x) \tag{11.2-5}$$

and large values of a and x can cancel off before the exponentiation takes place. This approach is practical because erfc(x) behaves like $\exp(-x^2)$, to within a factor of x, for large x. Most approximation formulas for erf(x) and erfc(x) have a factor of $\exp(-x^2)$ in them. Accordingly, it is possible to implement a routine for $r(x) \equiv \exp(x^2)\mathrm{erfc}(x)$ efficiently. It is also possible to implement efficiently a routine for the corresponding function $s(x) \equiv \exp(x^2)\mathrm{erf}(x)$.

Table 11.2/2 puts all these conditions together, and summarizes the calculation methods to be followed. It shows that all the required expressions can be written easily in terms of $r(x)$ evaluated for large positive x values, and $s(x)$ evaluated for nonnegative x values close to 0. The routines DRERFC and DRERF implement $r(x)$ and $s(x)$ respectively over the restricted ranges. They use rational approximations or continued fractions as described in later sections.

TABLE 11.2/2

EXPRESSIONS FOR ACCURATE EVALUATION OF FUNCTIONS

x	erf(x)	erfc(x)	$\Phi_x(0,1)$	$e^a\mathrm{erfc}(x)$
$x < -0.5$	$-1 + \exp(-x^2)r(-x)$	$2 - \exp(-x^2)r(-x)$	$\exp(-x^2/2)r(-x/2^{1/2})/2$	$2e^a - \exp(a-x^2)r(x)$
$-0.5 \le x < 0$	$-\exp(-x^2)s(-x)$	$1 + \exp(-x^2)s(-x)$	$[1 - \exp(-x^2/2)s(-x/2^{1/2})]/2$	$e^a + \exp(a-x^2)s(-x)$
$0 \le x < 0.5$	$\exp(-x^2)s(x)$	$1 - \exp(-x^2)s(x)$	$[1 + \exp(-x^2/2)s(x/2^{1/2})]/2$	$e^a - \exp(a-x^2)s(x)$
$0.5 \le x$	$1 - \exp(-x^2)r(x)$	$\exp(-x^2)r(x)$	$1 - \exp(-x^2/2)r(x/2^{1/2})/2$	$\exp(a-x^2)r(x)$

11.3 DERF, DERFC: Routines to Evaluate erf(x) and erfc(x)

Section 11.2 outlines the general approach to be taken in evaluating erf(x) *and* erfc(x). *Algorithms 11.3/1 and 11.3/2 flesh out the procedures. Both routines depend upon DRERF and DRERFC, which implement the function r(x) and s(x) discussed in Section 11.2.*

Algorithm 11.3/1 for DERF departs somewhat from the general approach of Section 11.2. First, the domain of the argument x has more subintervals with different algorithms. The new subintervals prevent underflow in the calculations. Second, to reduce the number of cases to consider, the algorithm evaluates erf($|x|$) first and then corrects for the algebraic sign.

ALGORITHM 11.3/1: Function DERF—Error Function

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	–	0, $\pm 10^{-9}$, ±0.5, ±6.5: algorithm points of change
DERF	Double Precision	Out	Error function at x, z.	$-1 \le z=\text{erf}(x) \le 1$	-1: lower bound 1: upper bound

Precondition	None.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*

if ($x < 0$) **then**
  absx = -*x*
  sign = -1
else if ($x > 0$) **then**
  absx = *x*
  sign = 1
else
  absx = 0
  sign = 0
end if
	Assign *absx*, the absolute value of *x*, and *sign*, a unit value in the direction of *x*.

if (*absx* < 10^{-9}) **then** DERF ← DRERF(*absx*)	Avoid underflow in squaring *absx*. Exponential factor is effectively 1 to more than 16 significant figures.
else if (*absx* < 0.5) **then** DERF ← exp(-*absx*^{2})DRERF(*absx*)	Evaluate DERF in terms of DRERF.
else if (*absx* < 6.5) **then** DERF ← 1 - exp(-*absx*^{2})DRERFC(*absx*)	Evaluate DERF in terms of DRERFC.

else
  DERF ← 1
end if
	DERF is essentially 1 to more than 16 significant figures.

DERF ← DERF · *sign*	Correct for original sign.

To compute erfc(x), Algorithm 11.3/2 uses additional subintervals in computing a functional value, like Algorithm 11.3/1. The added breakpoints are slightly different from those in Algorithm 11.3/1 because underflows do not occur in exactly the same places. Unlike the earlier algorithm, this one does not first evaluate the function at $|x|$, since erfc(x) is neither an odd nor an even function.

ALGORITHM 11.3/2: Function DERFC—Complementary Error Function

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	–	0, $\pm 10^{-18}$, ± 0.5, -6.5, 100: algorithm points of change
DERFC	Double Precision	Out	Complementary error function at x, z.	$0 \le z=\text{erfc}(x) \le 2$	0: lower bound 2: upper bound

Precondition	None.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if (x < -6.5) **then**	
DERFC ← 2	DERFC is effectively 2 to more than 16 significant figures.
else if (x < -0.5) **then**	
DERFC ← 2 - exp($-x^2$)DRERFC($-x$)	Evaluate DERFC in terms of DRERFC.
else if (x < -10^{-18}) **then**	
DERFC ← 1+ exp($-x^2$)DRERF($-x$)	Evaluate DERFC in terms of DRERF.
else if (x < 10^{-18}) **then**	Avoid underflow in squaring x. DERFC is effectively 1 to
DERFC ← 1	more than 16 significant figures.
else if (x < 0.5) **then**	
DERFC ← 1 - exp($-x^2$)DRERF(x)	Evaluate DERFC in terms of DRERF.
else if (x < 100) **then**	Arbitrary large number 100.
expfac ← PEXP($-x^2$)	PEXP() sets result to 0 if underflow occurs in exp().
if (*expfac* > 0) **then**	Underflow in exponential factor?
DERFC ← *expfac* · DRERFC(x)	Evaluate DERFC in terms of DRERFC.
else	
DERFC ← 0	DERFC is 0 to the precision available.
end if	
else	
DERFC ← 0	DERFC is 0 to the precision of any common computer.
end if	

11.4 NORDIS: Routine to Evaluate $\Phi_x(0,1)$, the Standard Normal CDF

$\Phi_x(0,1)$ is the standard normal CDF. Section 11.2 outlines the general approach to be taken in evaluating this function with the subroutine NORDIS. Algorithm 11.4/1 specifies the procedure to follow in more detail. NORDIS depends upon DRERF and DRERFC, which implement the function r(x) and s(x) discussed in Section 11.2. It also requires a "protected exponential" function PEXP to evaluate exponentials that may underflow in a floating-point system.

Like DERFC, NORDIS evaluates a function that is neither an odd nor an even function. Figure 11.4/1 shows the shape of this function, which is symmetric about the point (0,0.5). Representing the value of $\Phi_x(0,1)$ at large negative x values is limited by the exponent range of the floating-point system, since these values can be very small. Representing the value of $\Phi_x(0,1)$ anywhere else is limited by the precision of the floating-point representation, since the values are of significant size compared to the asymptotic value of one.

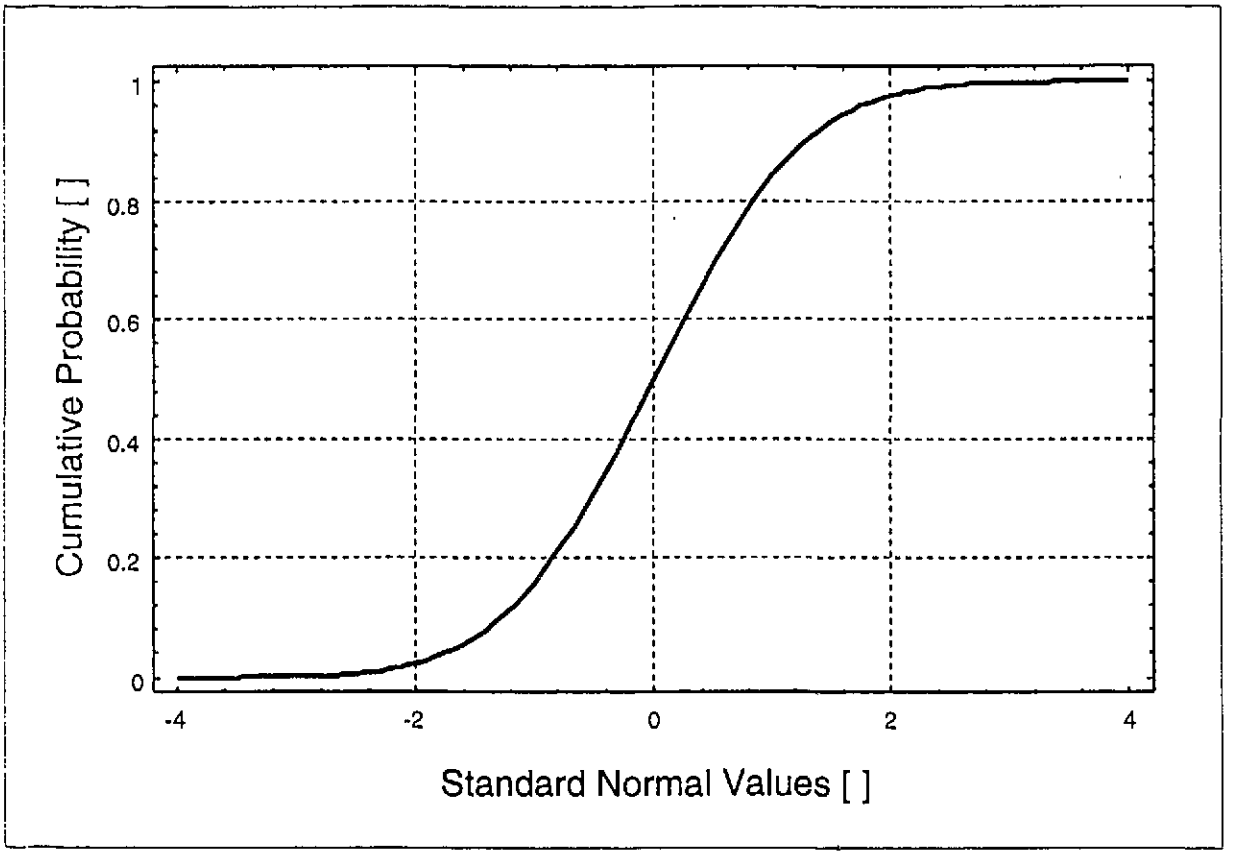

FIGURE 11.4/1: CDF of a Standard Normal Distribution

Algorithm 11.4/1 for NORDIS departs somewhat from the general approach of Section 11.2 to deal with these precision concerns. The domain of the argument x has more subintervals with different algorithms. The new subintervals prevent underflow in the calculations.

ALGORITHM 11.4/1: Subroutine NORDIS—Standard Normal CDF

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	–	0, $\pm 10^{-9}$, ±0.5, 6.5, -100: algorithm points of change
CDF	Double Precision	Out	CDF, p, of a standard normal distribution at x.	$0 \le p = \Phi_x(0,1) \le 1$	-1: lower bound 1: upper bound

Precondition	None.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
xbyrt2 ← *x* / $2^{1/2}$	The actual argument to erf.
if (*xbyrt2* ≤ -100) **then**	Arbitrary large negative number -100. Argument is too small to
CDF ← 0	represent CDF in any current floating-point system.
else if (*xbyrt2* ≤ -0.5) **then**	
expfac ← PEXP(-*xbyrt2*^{2})	PEXP( ) sets result to 0 if underflow occurs in exp( ).
if (*expfac* > 0) **then**	No underflow in exponential factor?
CDF ← *expfac* · DRERFC(-*xbyrt2*) / 2	Evaluate CDF in terms of DRERFC.
else	
CDF ← 0	If there was underflow, the result is 0.
end if	
else if (*xbyrt2* ≤ 10^{-9}) **then**	
CDF ← (1/2) -	Evaluate CDF in terms of DRERF.
exp(-*xbyrt2*^{2}) · DRERF(-*xbyrt2*) / 2	
else if (*xbyrt2* ≤ 0) **then**	Avoid underflow in squaring *xbyrt2*. Suppress the exponential
CDF ← (1/2) - DRERF(-*xbyrt2*) / 2	factor, which is effectively 1 to more than 16 significant figures.
else if (*xbyrt2* ≤ 10^{-9}) **then**	Avoid underflow in squaring *xbyrt2*. Exponential factor is
CDF ← (I/2) + DRERF(*xbyrt2*) / 2	effectively 1 to more than 16 significant figures.
else if (*xbyrt2* ≤ 0.5) **then**	
CDF ← (1/2) +	Evaluate CDF in terms of DRERF.
exp(-*xbyrt2*^{2}) · DRERF(*xbyrt2*) / 2	
else if (*xbyrt2* ≤ 6.5) **then**	
CDF ← I -	Evaluate CDF in terms of DRERFC.
exp(-*xbyrt2*^{2})DRERFC(*xbyrt2*) / 2	
else	
CDF ← 1	CDF is essentially 1 to more than 16 significant figures.
end if	

11.5 EXERFC: Routine to Evaluate e^x(erfc(y) - erfc(z))

Table 11.2/2 summarizes the expressions to be used to yield high precision estimates of e^xerfc(y) *in which exponents cancel to avoid overflow and underflow. Algorithm 11.5/1 takes advantage of these expressions, and also transforms (y,z) pairs to obtain maximum precision in calculating the difference* erfc(y) - erfc(z). *The potential precision gains are great, which is the prime justification for evaluating this entire function in a separate subroutine.*

Figure 11.5/1 shows the shape of the complementary error function erfc(x), which is symmetric about the point (0,1). Large negative values of x correspond to erfc(x) values that are very close to two. If y and z both have large negative values, then computing the difference erfc(y) - erfc(z) can involve loss of precision due to cancellation. For example, in 10-significant-figure floating-point arithmetic:

$$\text{erfc}(-4.5) - \text{erfc}(-4) \cong (2.000000000 - 1.999999985) = 1.5 \cdot 10^{-8} . \qquad (11.5\text{-}1)$$

The final result has only two significant figures. But suppose the following identity is applied:

$$\text{erfc}(x) + \text{erfc}(-x) \equiv 2 . \qquad (11.5\text{-}2)$$

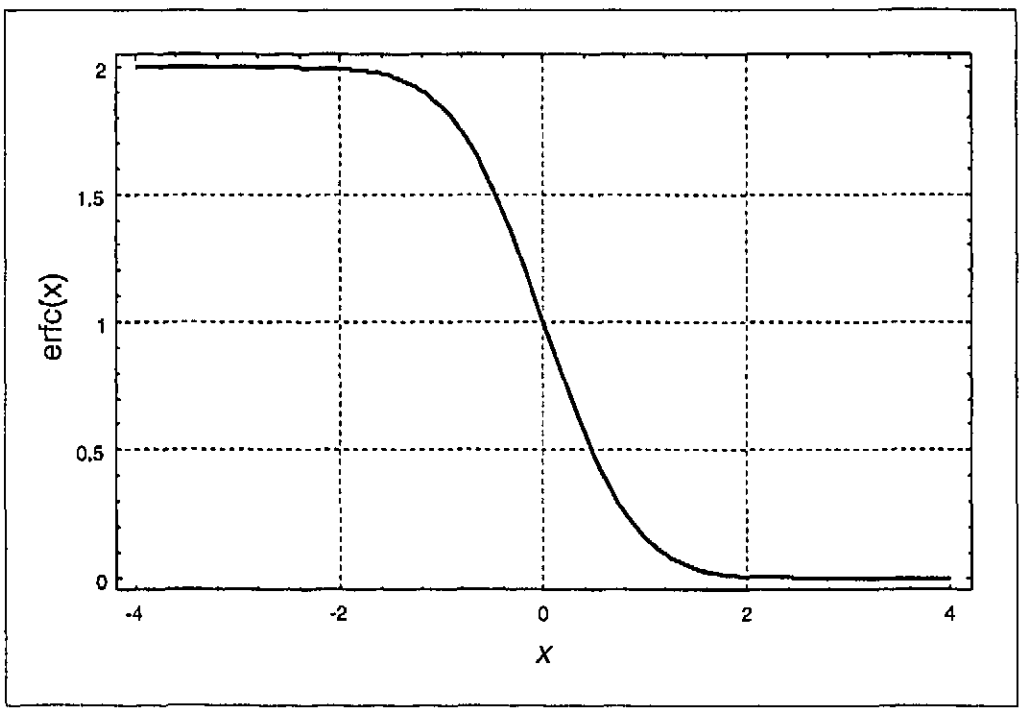

FIGURE 11.5/1: Complementary Error Function erfc(x)

Then:

$$\begin{aligned} \text{erfc}(-4.5) - \text{erfc}(-4) &= (2 - \text{erfc}(4.5)) - (2 - \text{erfc}(4)) = \text{erfc}(4) - \text{erfc}(4.5) \\ &\cong 1.541725790 \cdot 10^{-8} - 1.966160442 \cdot 10^{-10} \cong 1.522064186 \cdot 10^{-8} \ . \end{aligned} \tag{11.5-3}$$

In this version, full precision remains in the result. Algorithm 11.5/1 generalizes this example and arranges computations to retain as much precision as possible in values of the target function.

ALGORITHM 11.5/1: Subroutine EXERFC—Evaluate e^x(erfc(y) - erfc(z))

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	–	–
YND	Double Precision	In	Arbitrary argument y	–	=z: degenerate
ZND	Double Precision	In	Arbitrary argument z	–	=y: degenerate
RESULT	Double Precision	Out	Value w of function	$w = e^x(\text{erfc}(y) - \text{erfc}(z))$	underflow, overflow

Precondition	None.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($y = z$) **then**	Difference vanishes.
$w \leftarrow 0$	
else	
$locy \leftarrow y$	Make local copy of y.
$locz \leftarrow z$	Make local copy of z.
if ($\lvert\min(y, z)\rvert = \max(\lvert y\rvert, \lvert z\rvert)$) **then**	If larger absolute value has a negative sign ...
$locy \leftarrow -z$	Reverse signs of local copies of y and z, and interchange
$locz \leftarrow -y$	them to keep the right sign in the result.
end if	
if ($\min(locy, locz) < 0$) **then**	If values of $locy$ and $locz$ have different signs ...
$diff \leftarrow$ DERFC($locy$) - DERFC($locz$)	Evaluate difference in a straightforward manner.
if ($diff = 0$) **then** RESULT $\leftarrow 0$	Difference may be too small to represent.
else RESULT $\leftarrow$ PEXP(x + log($\lvert diff\rvert$))	Otherwise, avoid overflow in taking e^x in cases where $diff$
if ($diff < 0$) **then**	is so small that the final result can be represented.
RESULT $\leftarrow$ -RESULT	Copy sign from $diff$.
end if	
end if	
else RESULT $\leftarrow$	Otherwise both $locy$ and $locz$ are nonnegative ...
(PEXP(x-$locy^2$) DRERFC($locy$) -	Perform cancellation of exponents for both erfc functions,
PEXP(x-$locz^2$) DRERFC($locz$))	and evaluate expression in terms of DRERFC.
end if	
end if	

11.6 DRERF: Routine to Evaluate $s(x) = \exp(x^2)\mathrm{erf}(x)$

Table 11.2/2 shows how the function s(x) can be used in evaluating several other functions. As shown by the intervals for x in the table, s(x) need only be evaluated over the interval [0,0.5]. DRERF is a function that evaluates s(x) over this interval. It uses a continued fraction approximation given as equation (6.7.9) in the reference by Hart et al. (1978).

Figure 11.6/1 shows the shape of the function $s(x) = \exp(x^2)\mathrm{erf}(x)$ over the interval [0,0.5] in which DRERF evaluates it. Hart et al. (1978) provide a continued fraction form of erf(x) as shown in Equation (11.6-1), which is a copy of their equation (6.7.9):

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} e^{-x^2} \left\{ \frac{x}{1-} \frac{2x^2}{3+} \frac{4x^2}{5-} \frac{6x^2}{7+} \frac{8x^2}{9-} \cdots \right\} \qquad (11.6\text{-}1)$$

Then $s(x)$ is just:

$$s(x) = \frac{2}{\sqrt{\pi}} \left\{ \frac{x}{1-} \frac{2x^2}{3+} \frac{4x^2}{5-} \frac{6x^2}{7+} \frac{8x^2}{9-} \cdots \right\} . \qquad (11.6\text{-}2)$$

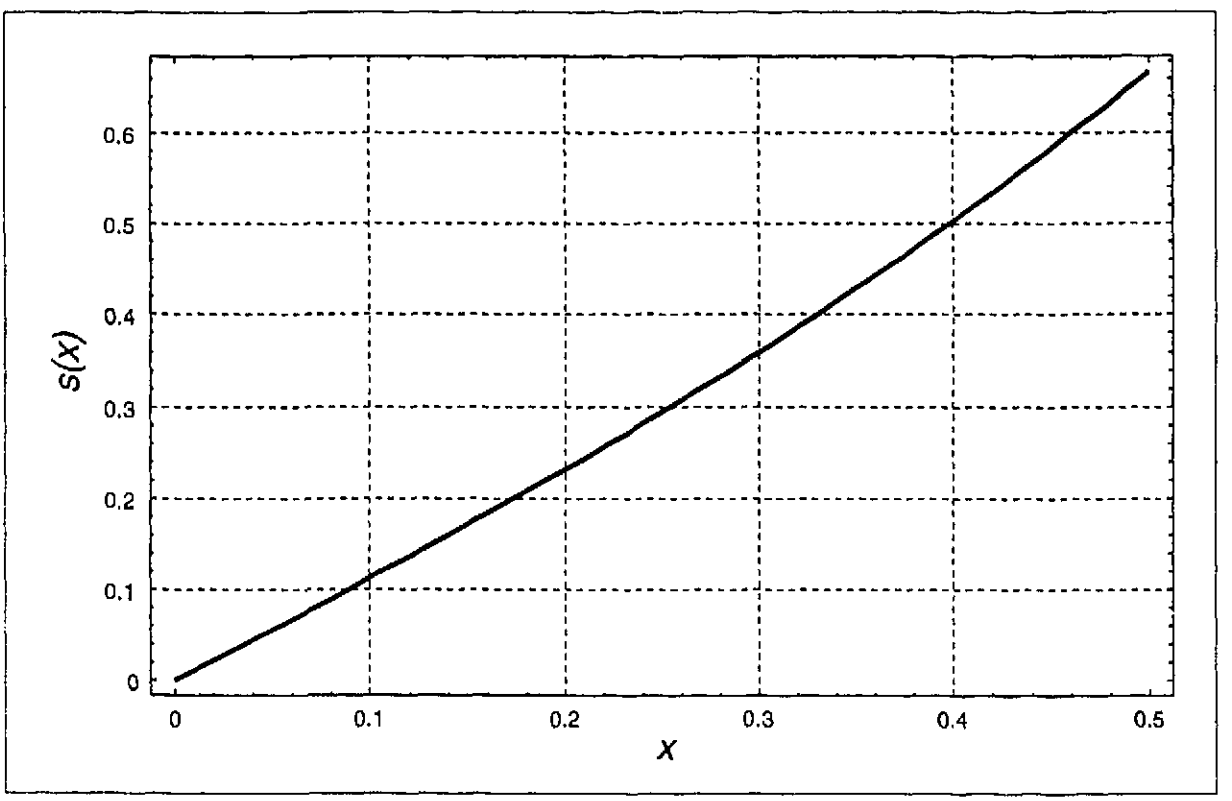

FIGURE 11.6/1: $s(x) = \exp(x^2)\mathrm{erf}(x)$

Algorithm 11.6/1 for DRERF utilizes a recursion technique, also described by Hart et al. (1978), to evaluate the continued fraction. It applies the recursion a fixed number of times, sufficient for 15-decimal-place accuracy across the entire interval. Substantial experimentation with implementations of this function in APL established the required number of iterations to be 10 for arguments in the interval [0,0.5]. Experimentation also showed that the recursion technique was stable, with little accumulated error, across the function domain. In contrast, using a rational function equivalent to the continued fraction yielded unstable results and relatively large errors.

ALGORITHM 11.6/1: Function DRERF—Evaluate $\exp(x^2)\,\mathrm{erf}(x)$

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	$0 \le x \le 0.5$	0: lower bound 0.5: upper bound
DRERF	Double Precision	Out	Value z of function.	$0 \le z = \exp(x^2)\mathrm{erf}(x) \le 1$	0: lower bound ~0.67: upper bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($x < 0$) **then** $locx \leftarrow 0$ **else if** ($x > 0.5$) **then** $locx \leftarrow 0.5$ **else** $locx \leftarrow x$ **end if**	Make a local copy of the argument x and ensure that it falls within the required interval.
$n1 \leftarrow 0$ $n2 \leftarrow 2 \cdot locx \,/\, \pi^{1/2}$ $d1 \leftarrow 1$ $d2 \leftarrow 1$	Initialize the terms in the 2-term recursion for the numerator of the continued fraction. Initialize the terms in the 2-term recursion for the denominator of the continued fraction.

if ($locx > 10^{-9}$) **then**
  $diff \leftarrow f1 \leftarrow 2 \cdot locx^2$
  $f2 \leftarrow 3$
  $sign \leftarrow -1$
  do for j **from** 1 **to** 10
  $n3 \leftarrow sign \cdot f1 \cdot n1 + f2 \cdot n2$
  $d3 \leftarrow sign \cdot f1 \cdot d1 + f2 \cdot d2$
  $n1 \leftarrow n2$
  $n2 \leftarrow n3$
  $d1 \leftarrow d2$
  $d2 \leftarrow d3$
  $f1 \leftarrow f1 + diff$
  $f2 \leftarrow f2 + 2$
  $sign \leftarrow -sign$
  end do
end if
	If more than just a linear approximation is needed ...
	Initialize the two factors used in the recursion.
	Initialize the sign of the parts of the continued fraction.
	Iterate in evaluating the continued fraction.
	Evaluate new numerator iteration.
	Evaluate new denominator iteration.
	Reassign numerator and denominator iterates.
	Increment factors.
	Alternate signs.

$z \leftarrow n2 \,/\, d2$	Final result is ratio of numerator to denominator.

11.7 DRERFC: Routine to Evaluate $r(x) = \exp(x^2)\text{erfc}(x)$

Table 11.2/2 shows how the function r(x) can be used in evaluating several other functions. DRERFC is a function that evaluates r(x) for any x ≥ 0. It uses a rational approximation (Algorithm 5708 in Hart et al. (1978)) or a continued fraction approximation (Equation (6.7.10) in Hart et al. (1978)), depending on the magnitude of x.

Figure 11.7/1 shows the shape of the function $r(x) = \exp(x^2)\text{erfc}(x)$ for nonnegative x. Hart et al. (1978) provide a rational function for $r(x)$ (Algorithm 5708) that maintains about 16 significant figures of precision for x over the interval [0,8]:

$$r(x) = e^{x^2}\text{erfc}(x) \doteq \frac{p_0 + p_1 x + p_2 x^2 + p_3 x^3 + p_4 x^4 + p_5 x^5 + p_6 x^6 + p_7 x^7 + p_8 x^8}{q_0 + q_1 x + q_2 x^2 + q_3 x^3 + q_4 x^4 + q_5 x^5 + q_6 x^6 + q_7 x^7 + q_8 x^8 + q_9 x^9} \quad , \quad (11.7\text{-}1)$$

where the coefficients are given in Table 11.7/1. Algorithm 11.7/1 utilizes this approximation for x in [0,8], and a continued fraction (Equation 6.7.10 from Hart et al. (1978)) to evaluate $s(x)$ for $x > 8$:

$$r(x) = e^{x^2}\text{erfc}(x) = \frac{1}{\sqrt{\pi}} \left(\cfrac{1}{x+} \cfrac{\frac{1}{2}}{x+} \cfrac{\frac{2}{2}}{x+} \cfrac{\frac{3}{2}}{x+} \cdots \right)$$

$$\doteq \frac{1}{\sqrt{\pi}} \frac{p_0 + \dfrac{p_1}{x^2} + \dfrac{p_2}{x^4} + \dfrac{p_3}{x^6} + \dfrac{p_4}{x^8} + \dfrac{p_5}{x^{10}}}{x\left(q_0 + \dfrac{q_1}{x^2} + \dfrac{q_2}{x^4} + \dfrac{q_3}{x^6} + \dfrac{q_4}{x^8} + \dfrac{q_5}{x^{10}} \right)} \quad . \quad (11.7\text{-}2)$$

Experimentation has shown that the rational expression in $(1/x^2)$ in Equation (11.7-2) can be evaluated in a stable manner, yielding 15 significant figures in the result. The advantage of this form over the continued fraction is that every second term vanishes, reducing the work to be performed. The coefficients are shown in Table 11.7/1.

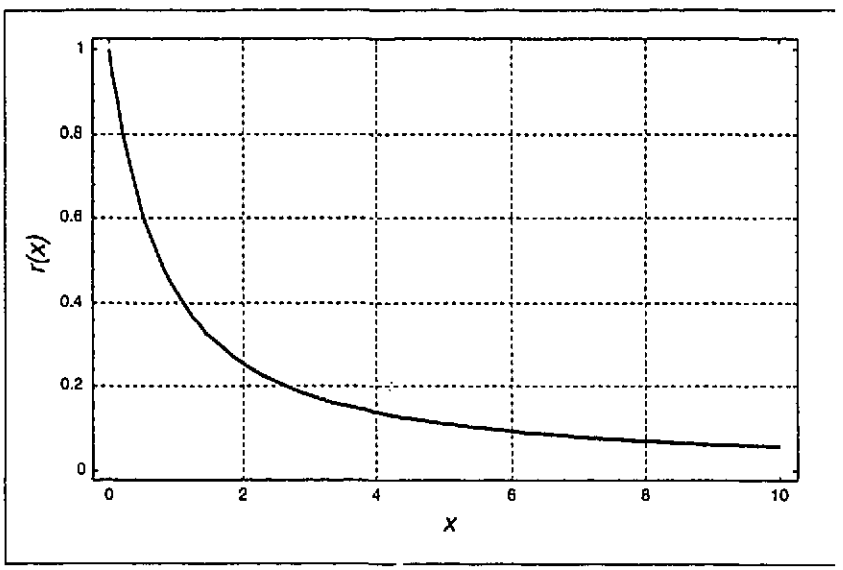

FIGURE 11.7/1: $r(x) = \exp(x^2)\text{erfc}(x)$

ALGORITHM 11.7/1: Function DRERFC—Evaluate $\exp(x^2)$ erf(x)

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	$0 \le x$	0: lower bound
DRERF	Double Precision	Out	Value z of function $r(x)$.	$0 \le z=\exp(x^2)\text{erfc}(x) \le 1$	0: lower bound 1: upper bound

Precondition	Input arguments satisfy their constraints. Local arrays P1 and Q1 contain Equation (11.7-1) coefficients from Table 11.7/1. Local arrays P2 and Q2 contain Equation (11.7-2) coefficients from Table 11.7/1.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
if ($x < 0$) **then** *locx* $\leftarrow 0$ **else** *locx* $\leftarrow x$ **end if**	Make a local copy of the argument x and ensure that it falls within the required interval.
if (*locx* ≤ 8) **then**	If x is small ...
$z \leftarrow$ POLYNM(8, P1, *locx*) / POLYNM(9, Q1, *locx*)	Apply Equation (11.7-1) using polynomial evaluation.
else if (*locx* $\le 10^9$) **then**	If x is large to enormous ...
$y \leftarrow 1$ / *locx*^{2}	Equation (11.7-2) is expressed as a rational function in y.
$z \leftarrow (1/\pi^{1/2})\cdot$POLYNM(5, P2, y) / (*locx*·POLYNM(5, Q2, y))	Apply modified version of Equation (11.7-2).
else	If x is humongous ...
$z \leftarrow (1/\pi^{1/2})$ / *locx*	Use simplified version of Equation (11.7-2).
end if	

TABLE 11.7/1

COEFFICIENTS IN RATIONAL APPROXIMATIONS TO $r(x)$

Eq	j	p_j	q_j
1	0	3723.50798 15548 06722 56717	3723.50798 15548 06543 52472
	1	7113.66324 69540 49873 40998 6	11315.19208 18544 05468 20144 3
	2	6758.21696 41104 85886 33275 86	15802.53599 94020 42527 35884 57
	3	4032.26701 08300 49736 20957 28	13349.34656 12844 57371 72173 17
	4	1631.76026 87537 14696 35150 913	7542.47951 01934 75755 47208 583
	5	456.26145 87060 92630 64180 0311	2968.00490 14823 08716 42765 2719
	6	86.08276 22119 48595 11755 45307	817.62238 63045 44077 02825 02642
	7	10.06485 89749 09542 53550 505591	153.07771 07503 62215 85695 20624
	8	0.56418 95867 61813 61369 25465 862	17.83949 84391 39556 52884 23873 4
	9	Not Applicable	1.
2	0	1	1
	1	27	27.5
	2	234.5	247.5
	3	761.25	866.25
	4	790.3125	1082.8125
	5	120	324.84375

11.8 PEXP: Routine to Evaluate a Protected Exponential

PEXP evaluates an exponential function in such a way that results that would cause floating point underflow return as zero. PEXP is not strictly a part of the Parameter Sampling Packages (PSP), but this specification is given anyway since the PSP uses PEXP in several places. The algorithm is simple: check the argument's value, and return zero if the argument is a large negative number.

The exponential routine exp(x) has a very limited domain in typical floating-point arithmetic. Arguments greater than 1000 cause overflow on any floating-point system, and arguments less than -1000 cause underflow. In most floating-point systems, arguments with much smaller magnitudes can cause underflow or overflow. Usually overflows are unintended. They represent error conditions that should be flagged. Underflows usually indicate very small numbers that can safely be neglected and treated as zero. Algorithm 11.8/1 shows how to deal safely with arguments that would cause underflow exceptions.

ALGORITHM 11.8/1: Function PEXP—Protected Exponential Function

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
XND	Double Precision	In	Arbitrary argument x	–	large magnitudes
PEXP	Double Precision	Out	Value $z = e^x$ of exponential function	$z \geq 0$	underflows, overflows

Common	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
MAXREL:					
MAXREL	Double Precision	In	Maximum double precision value, M	$M > 0$, no overflow	–

Precondition	Input arguments and common variables satisfy their constraints.
Postcondition Output	arguments satisfy their constraints. If e^x would cause floating point underflow, then z=0.

Algorithm	*Commentary*

if ($x \leq -\ln M$) then PEXP $\leftarrow$ 0
else PEXP $\leftarrow$ exp(x)
end if
	Set result to 0 for very small arguments.
	Otherwise use normal exponential; do not check for overflow, which should fail.

11.9 POLYNM: Routine to Evaluate a Polynomial

POLYNM evaluates a polynomial efficiently to simplify routines like DRERFC that evaluate more than one polynomial. It uses Horner's scheme, the standard way of evaluating a polynomial using nested multiplies (Hart et al 1978).

Equation (11.9-1) shows a general form for an n^{th}-degree polynomial in x with coefficients p_i. The only constraint is that the coefficient p_n be nonzero, or else the polynomial would have a lesser degree.

$$p(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1} + p_n x^n \quad , \qquad n \geq 0. \tag{11.9-1}$$

For more efficient evaluation, a polynomial can be expressed in a nested form:

$$p(x) = p_0 + x\{p_1 + x[p_2 + \ldots + x(p_{n-1} + p_n x)]\} \quad . \tag{11.9-2}$$

Algorithm 11.9/1 shows how POLYNM applies this form to evaluate an arbitrary polynomial.

ALGORITHM 11.9/1: Function POLYNM—Evaluate Polynomial

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
DEGREE	Integer	In	Degree n of polynomial	$n \geq 0$	0: lower bound
COEF	Double Precision	In	Constant coefficients p_i, $i = 0(1)n$	$p_n \neq 0$	–
XND	Double Precision	In	Arbitrary argument x	no overflow, no underflow in calculation	
POLYNM	Double Precision	Out	Value $p(x)$ of polynomial.	–	–

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints; $p(x)$ is the value of the polynomial shown in Equation (11.9-1).

Algorithm	*Commentary*
if ($n < 0$) **then** POLYNM ← 0	Invalid degree; set result to 0.
else if ($n = 0$) **then** POLYNM ← p_0	Constant; the result does not depend on x.
else POLYNM ← p_n	Initialize value of polynomial.
do for j **from** n-1 **to** 0 **by** -1	Loop over remaining coefficients
POLYNM ← POLYNM·x + p_j	Add contribution from each coefficient, using nested form.
end do	
end if	

11.10 INVNOR: A Routine to Invert a Standard Normal Distribution

INVNOR is one of two special inversion routines shown in Figure 11.1/1. It inverts the CDF of a standard normal distribution, one with mean 0 and standard deviation 1. TRAQUA (Section 8.8) uses this routine to invert any Normal or Lognormal Distribution CDF. INVNOR uses ACM Algorithm 442 by Hill and Davis (1971) to approximate the inverse CDF to about 16 significant figures.

Hill and Davis (1971) published a family of algorithms as ACM Algorithm 442 to invert a standard normal CDF, given a routine like NORDIS to evaluate the CDF accurately. INVNOR uses the approximation u_4, based on four terms of a Taylor's series for the required correction to an initial approximation, and claimed to give a relative error of 15.95 decimal digits.

Algorithm 442 begins with an initial approximation to the inverse given by

$$x(p) \doteq s - \frac{1637.720 + 494.877s + 7.47395s^2}{659.935 + 908.401s + 117.9407s^2 + s^3} \quad 0 \le p \le 0.5, \tag{11.10-1}$$

where p is the cumulative probability, $s = (-2 \ln p)^{1/2}$, and $x(p)$ is the corresponding quantile. Different members of the family of algorithms use an expression of the same form as Equation (11.10-1), but with different coefficients.

Then the algorithm estimates a relative error in probability, z, corresponding to the initial approximation $x_0 = x(p)$:

$$z = \frac{p - \Phi(x_0)}{\phi(x_0)} , \tag{11.10-2}$$

where Φ is the standard normal CDF, approximated by a call to NORDIS, and ϕ is the probability density function

$$\phi(x) \equiv \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) . \tag{11.10-3}$$

Then Algorithm 442 evaluates the final u_4 approximation:

$$x(p) \doteq \left\{\left[\left\{\left[\left(\tfrac{3}{4}x_0^2 + \tfrac{7}{8}\right)z + x_0\right]x_0 + \tfrac{1}{2}\right\}\tfrac{1}{3}z + \tfrac{1}{2}x_0\right]z + 1\right\}z + x_0 . \tag{11.10-4}$$

Algorithm 11.10/1 specifies this algorithm in more detail.

ALGORITHM 11.10/1: Subroutine INVNOR—Inverse CDF of a Standard Normal

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANLT	Double Precision	In	Cumulative probability p	$0 \le p \le 1$	0: lower bound 1: upper bound
NORVLU	Double Precision	Out	Value x belonging to standard normal distribution	$x = \Phi^{-1}(p)$, where Φ is the standard normal CDF	$-\infty$: lower bound ∞: upper bound

Precondition	Input arguments satisfy their constraints. Local arrays P and Q have been initialized with the coefficients of the numerator and denominator of Equation (11.10-1).
Postcondition	Output arguments satisfy their constraints, except that extreme values are treated in an implementation-specific manner, since floating point systems do not easily deal with infinity.

Algorithm	*Commentary*

if ($p \le 0.5$) **then**
  $redp \leftarrow p$
  $sign \leftarrow -1$
else
  $redp \leftarrow 1-p$
  $sign \leftarrow 1$
end if
	Transform argument to reduced interval [0,0.5], maintaining precision.

if ($redp \le 0$) **then**
  $x \leftarrow 13\ sign$
else
  $s \leftarrow (-2 \ln redp)^{1/2}$
  $x \leftarrow sign \cdot (s$ - POLYNM(2, P, s) /
  POLYNM(3, Q, s))
  call NORDIS(x, $prob$)
  if ($prob \ne p$) **then**
  $z \leftarrow (p - prob) \cdot (2\pi)^{1/2} \cdot \exp(x^2/2)$
  $x \leftarrow [(\{[(0.75x^2+0.875)z + x]x + 0.5\}z/3 + x/2)z + 1]z + x$
  end if
end if
	At the extreme end of the range ...
	So far out that the CDF is close to the precision limit (the value 13 may vary, depending on the floating-point system)
	Argument of rational approximation to inverse CDF.
	Initial estimate of inverse CDF, from Equation (11.10-1).
	
	Find corresponding CDF.
	Check the agreement; if not good enough ...
	Difference divided by PDF.
	
	Final approximation, from Equation (11.10-4).

11.11 INVCOR: A Routine to Invert a Correlated Normal CCDF

TRAQUA (Section 8.8) inverts the CDF of a Parameter Distribution. When the Parameter Distribution is a Conditional Distribution, it inverts the conditional CDF (CCDF). TRAQUA calls INVCOR to invert the CCDF for a Correlated Normal or Correlated Lognormal Distribution. Algorithm 11.11/1 shows the algorithm for this inversion. When the distribution is not truncated, the inversion is straightforward: calculate the attributes in the CCDF and perform a standard Normal Distribution inversion. When the distribution is truncated, care must be taken to ensure that the truncation limits are observed.

The routine INVCOR inverts the CCDF of a Correlated Normal Distribution. The CCDF is the CDF that represents the distribution of values for a parameter X when the value of a correlated parameter Y is known (Section 4.5). The limits on a Correlated Normal Distribution come from a Probability Interval. These probabilistic limits apply to the marginal CDF of the distribution (i.e., to the CDF when the value of Y is unknown). For the CCDF, INVCOR must transform the limits to a Probability Interval that depends on Y. INVCOR should follow Algorithm 11.11/1, which fleshes out the specification in Table 4.6/1.

ALGORITHM 11.11/1: Subroutine INVCOR—Inverse Conditional CDF of a Correlated Normal Distribution

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
QUANTL	Double Precision	In	Cumulative probability, p, in the conditional distribution.	$0 \le p \le 1$	0: lower bound 1: upper bound
MEAN	Double Precision	In	Mean value of distribution, μ_X	–	–
STDDEV	Double Precision	In	Standard deviation of distribution, σ_X	$0 \le \sigma_X$	0: lower bound
LOBND	Double Precision	In	Low probability of the unconditional CDF, p_{LX}	$0 \le p_{LX} \le p_{HX}$	0: lower bound p_{HX}: upper bound
HIBND	Double Precision	In	High probability of the unconditional CDF, p_{HX}	$p_{LX} \le p_{HX} \le 1$	p_{LX}: lower bound 1: upper bound
CORCOF	Double Precision	In	Correlation coefficient, ρ_{XY}	$-1 \le \rho_{XY} \le 1$	-1: lower bound 0: no correlation 1: upper bound
CORMU	Double Precision	In	Mean value of correlated distribution, μ_Y	–	–
CORSIG	Double Precision	In	Standard deviation of correlated distribution, σ_Y	$0 \le \sigma_Y$	0: lower bound
CORVAL	Double Precision	In	Value y of correlated parameter	–	–
CORVLU	Double Precision	Out	Value x in a Correlated Normal Distribution	$p_{LX} \le \text{CDF}(x) \le p_{HX}$ $\text{CCDF}(x) = p$	–

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints.

Algorithm	*Commentary*
$shistdmu \leftarrow \rho_{XY} (y - \mu_Y) / \sigma_Y$ $shistdsig \leftarrow (1 - \rho_{XY}^2)^{1/2}$	Find shifted mean and standard deviation as if this distribution were a standard normal with mean 0 and standard deviation 1.
if ($p_{LX} = 0$ and $p_{HX} = 1$) **then**	If X has an untruncated distribution ...
if ($\rho_{XY} = 1$ or $\rho_{XY} = -1$) **then**	If there is a strict linear relationship between X and Y ...
$x \leftarrow \mu_X + shistdmu \cdot \sigma_X$	Linear interpolation for X.
else call INVNOR(p, z)	Find location of desired quantile in a standard normal.
$x \leftarrow \mu_X +$ $(shistdmu + shistdsig \cdot z) \cdot \sigma_X$	Correct for shifted mean and standard deviation, and then for the actual mean and standard deviation.
end if	
else if ($p_{LX} = p_{HX}$) **then**	If the bounds overlap, the result is always the same.
call INVNOR(p_{LX} , z)	Find the point in a standard distribution corresponding to both
$x \leftarrow \mu_X + z \cdot \sigma_X$	bounds, and map it onto the current distribution.
else call INVNOR(p, z)	Use procedure from Table 4.6/1. Find location in a shifted
call NORDIS($shistdmu + shistdsig \cdot z$, $ship$)	standard normal of the desired point, and evaluate standard CDF.
$truncship \leftarrow (1\text{-}ship) \cdot p_{LX} + ship \cdot p_{HX}$	Apply truncation limits to constrain the probability to the
call INVNOR($truncship$, z)	allowed region, then evaluate the variate value in the current
$x \leftarrow \mu_X + z \cdot \sigma_X$	truncated distribution.
end if	

In fact, INVCOR in SYVAC3 version 3.09 (SV309) does not follow Algorithm 11.11/1, but uses a more complicated algorithm that is subject to numerical problems when the Conditional Distribution is heavily truncated. The two algorithms produce similar results, and in fact the same results for untruncated distributions. The following fragment, if used in place of the last **else** clause in Algorithm 11.11/1, would yield the algorithm currently used.

ALGORITHM 11.11/2: Subroutine INVCOR—Modified Algorithm 11.11/1 Used in SYVAC3 (SV309)

else $shimu \leftarrow \mu_X + shistdmu \cdot \sigma_X$	Shifted mean for the conditional distribution.
$shisig \leftarrow shistdsig \cdot \sigma_X$	Shifted standard deviation for the conditional distribution.
Evaluate $ship_{LX}$ and $ship_{HX}$ using calls to NORDIS and INVNOR.	Evaluate Probability Interval for the conditional distribution that corresponds to the limits imposed by the original Probability Interval (p_{LX} , p_{HX}) on the marginal distribution.
$truncp \leftarrow (1\text{-}p) \cdot ship_{LX} + p \cdot ship_{HX}$	Truncated probability in conditional distribution.
call INVNOR($truncp$, z)	Standard normal variate corresponding to x.
$x \leftarrow shimu + z \cdot shisig$	Final value.
end if	

There are two problems with the current algorithm. First, evaluating $ship_{LX}$ and $ship_{HX}$ is complicated by special cases when p_{LX} or p_{HX} is at an extreme value (zero or one), since the corresponding value bounds are infinite. Second, x produced by this algorithm can in some cases lie outside the original bounds, because of loss of precision in the calculations. In the code it is necessary to check the final x value against the bounds. Future implementations should use Algorithm 11.11/1.

12 ALGORITHMS FOR PSEUDORANDOM NUMBER GENERATORS

12.1 Overview of Algorithms

All versions of SYVAC3 up to SV309 use a linear congruential pseudorandom number generator called "Superduper" that was described by Marsaglia (1972). It is based on the recursion $X_{i+1} = (69069\ X_i + 1) \bmod 2^{32}$. Walker (1985) summarized the properties of this generator and provided a portable implementation in the style of Marse and Roberts (1983). The version in SYVAC3 has been changed slightly from this implementation. Future implementations should adopt combined pseudorandom number generators as described and implemented by L'Ecuyer (1988) and by L'Ecuyer and Côté (1991) to provide pseudorandom numbers with smaller granularity, longer sequences, and independent subsequences from arbitrary single random seeds.

Chapter 6 established the requirements for the Pseudorandom Generator object type that implements a pseudorandom number generator. The major requirement is to generate a sequence of numbers between 0 and 1 that is (1) repeatable, given the same Initial Random Seed(s), (2) uniformly spread over the interval, and (3) random (i.e., unpredictable). Conditions (1) and (3) appear to be mutually contradictory. If the sequence of numbers is repeatable, it is predictable. An observer could simply generate the sequence once, store it, and then be able to predict with 100% certainty each element of the sequence when it is generated again. After discussing a particular method of generating pseudorandom numbers (the "middle-square method"), Knuth (1969) describes the situation this way:

> There is a fairly obvious objection to this technique: how can a sequence generated in such a way be random, since each number is completely determined by its predecessor? The answer is that this sequence *isn't* random, but it *appears* to be. In typical applications the actual relationship between one number and its successor has no physical significance; hence the nonrandom character is not really undesirable.

Two types of pseudorandom number generators have attained preeminence because of their efficiency and good "random" properties: linear congruential and multiplicative congruential generators. A linear congruential generator uses the following recursion to generate successive entries in a long integer sequence:

$$X_{i+1} \equiv aX_i + b \ \ mod\ c \qquad\qquad 0 \le X_i < c. \qquad (12.1\text{-}1)$$

A multiplicative congruential generator looks the same, except that $b = 0$. In both cases, the pseudorandom number U_i is a scaled version of X_i:

$$U_i = X_i / c \ . \qquad (12.1$$

The idea is to choose a large enough that aX_i typically exceeds c. The remainder left over when removing multiples of c bears little connection with X_i. Of course not just any combination of values a, b, and c produces a good pseudorandom number generator. They must be carefully chosen. For example, a multiplicative generator has a maximum period (i.e., sequence length) c - 1 if c is prime and a is a primitive element modulo c (Knuth 1969). Furthermore, the resulting generator must be thoroughly tested to ensure that the sequence it generates displays good statistical properties. Walker (1985) cites some studies that have

been carried out to test generators. Walker selected a linear congruential generator with multiplier $a = 69069$, offset $b = 1$ and modulus $c = 2^{32}$ on the basis of such tests. This generator comes from a statistical package called SuperDuper by Marsaglia (1972).

The SuperDuper generator has been implemented in the SUPRAN routine. It has some limitations for use with SYVAC3:

- Short period–The period of the generator (2^{32}, or about 4×10^9) is too short. If 4000 parameters require random numbers in each simulation, then one miliion simulations would exhaust the generator. While this limit has not been reached yet, it is within reach.

- No support for subsequences–Sampling from independent streams of pseudorandom numbers helps to maintain the independence of parts of a model. Independent streams could be used to sample parameters for different submodels (e.g., a vault model or a biosphere model), different objects (e.g., contaminants U-238 and C-14), and different sources (e.g., contaminants from different wasteforms). A single long sequence such as SUPRAN generates has only those subsequences defined by the user. To identify subsequences, one must pick starting random seeds far enough apart in the sequence so that the subsequences generated from those starting points will not overlap.

- Limited precision–Any 32-bit pseudorandom number generator can produce only 2^{32} distinct results. That means that uniform values between 0 and 1 have only about nine digits of precision, and the granularity (minimum separation between values) is of the order of 10^{-9}. This differs substantially from the precision with which double-precision numbers are represented.

- Slow advancement–It is often desirable to skip thousands of seeds from the sequence generated by a pseudorandom number generator. Skipping values is one of the operations on a Pseudorandom Generator identified in Section 6.2. SUPRAN has no shortcuts; the only way to skip 10 000 values is to produce them one at a time and throw them away.

Pierre L'Ecuyer (1988) has published a description of a set of portable combined pseudorandom number generators that surpass all these limitations. L'Ecuyer and Côté (1991) have published Pascal code to implement such combined generators. Their work should form the basis of a modified Pseudorandom Generator object type to be implemented in SYVAC3.

12.2 SUPRAN: A Routine to Generate Pseudorandom Numbers

In SV309, the Pseudorandom Generator type consists of a single data field, Random Seed, and a single routine, SUPRAN. To support multiple Pseudorandom Generator objects, the calling program need only reserve storage for multiple Random Seed fields, one for each object. The only operation directly supported is to call SUPRAN, passing a Random Seed in the argument list. This call accomplishes two things: first, the Random Seed advances by one position in the sequence of integer values that SUPRAN can generate. Secondly, SUPRAN generates a random value uniformly distributed between 0 and 1 from the Random Seed, and returns it to the calling program. To ensure that multiple Pseudorandom Generator objects are independent, the calling program must provide initial Random Seed values that are widely separated in the sequence of pseudorandom numbers.

Algorithm 12.2/1 shows the operation of the routine SUPRAN, as designed by Walker (1985). It uses an approach similar to that demonstrated by Marse and Roberts (1983). SUPRAN performs arithmetic modulo $c = 2^{32}$ using 32-bit integers in 2's complement integer notation, which can only represent values between -2^{31} and $2^{31}-1$.

The recursion used in SUPRAN is:

$$X_{i+1} \equiv 69069 X_i + 1 \mod 2^{32} \qquad 0 \le X_i < 2^{32}. \tag{12.2-1}$$

SUPRAN cannot store numbers larger than $2^{31}-1$ as integers, because they would be interpreted as negative numbers in 2's complement notation. Instead SUPRAN transforms the values:

$$Y_i = \begin{cases} X_i & \text{if } 0 \le X_i < 2^{31} \\ X_i - 2^{32} & \text{if } 2^{31} \le X_i < 2^{32} \end{cases} \qquad -2^{31} \le Y_i < 2^{31}. \tag{12.2-2}$$

The set of Y_i values provide a different way of representing uniquely all the integers modulo 2^{32}. There is a one-to-one correspondence between the X_i and Y_i values:

$$Y_i \equiv X_i \mod 2^{32} \ . \tag{12.2-3}$$

As a result, the Y_i values also satisfy the recursion in Equation (12.2-1).

A portable implementation of the random number generator must perform the multiplication in Equation (12.2-1) using 32-bit arithmetic without integer overflow. This step can be accomplished by breaking X_i into two parts:

$$X_i = 2^{16} H_i + L_i \ , \tag{12.2-4}$$

where both H_i and L_i lie in the interval $[0, 2^{16}-1]$. Calculations can be performed on the high-order and low-order bits separately. Algorithm 12.2/1 shows this calculation in detail.

ALGORITHM 12.2/1: Subroutine SUPRAN—Portable 32-bit Random Number Generator

Arguments	*Data Type*	*In/Out*	*Definition*	*Constraint*	*Special Cases*
SEED	Integer	In/Out	Random seed Y in pseudorandom sequence	$-2^{31} \le Y \le 2^{31}-1$	-2^{31}: lower bound $2^{31}-1$: upper bound
RNDNUM	Double Precision	Out	Pseudorandom probability p	$0 \le p < 1$	0: lower bound 1: upper bound

Precondition	Input arguments satisfy their constraints.
Postcondition	Output arguments satisfy their constraints. $Y_{out} \equiv 1 + 69069\ Y_{in} \bmod 2^{32}$ $p = Y_{out} / 2^{32}$

Algorithm	*Commentary*

if ($Y < 0$) **then**
  $Y \leftarrow Y + 2^{31}$
  $H \leftarrow \lfloor Y / 2^{16} \rfloor$
  $L \leftarrow Y - 2^{16} H$
  $H \leftarrow H + 2^{15}$
	A negative seed is a transformed version of a seed $\ge 2^{31}$. Transform it back by adding 2^{32} in stages; first add 2^{31} to make it positive. Then take the top 15 bits (H) and the bottom 16 bits (L) separately. Adding 2^{15} to H is equivalent to adding another 2^{31} to the original seed.

else
  $H \leftarrow \lfloor Y / 2^{16} \rfloor$
  $L \leftarrow Y - 2^{16} H$
end if
	A positive seed is not transformed. Separate the top 15 bits and the bottom 16 bits.

$L \leftarrow L \cdot 23023$ $carry \leftarrow \lfloor L / 2^{16} \rfloor$ $L \leftarrow L - 2^{16}\ carry$	$69069 = 3 \cdot 23023$, where $23023 < 2^{16} < 69069$. Multiply in stages to protect against overflow.
$H \leftarrow carry + H \cdot 23023$	Carry the extra bits to the H part of the seed.
$carry \leftarrow \lfloor H / 2^{16} \rfloor$ $H \leftarrow H - 2^{16}\ carry$	Discard the carry from H, since it is congruent to 0 mod 2^{32}.
$L \leftarrow L \cdot 3$ $carry \leftarrow \lfloor L / 2^{16} \rfloor$ $L \leftarrow L - 2^{16}\ carry$	Second stage of multiplication. Carry the extra bits to the H part of the seed.
$H \leftarrow carry + H \cdot 3$ $carry \leftarrow \lfloor H / 2^{16} \rfloor$ $H \leftarrow H - 2^{16}\ carry$	Discard the carry from H.

$carry \leftarrow \lfloor H / 2^{15} \rfloor$
$H \leftarrow H - 2^{15}\ carry$
$Y \leftarrow 2^{16} H + L$
if ($carry = 1$) **then**
  $Y \leftarrow Y - 2^{31}$
end if
	Determine if the new seed is $\ge 2^{31}$. By removing most significant bit, find the part of the new seed that cannot overflow.
	Reconstitute the new seed.
	The new seed would overflow; transform it to a negative number.

$p \leftarrow Y / 2^{32}$ **if** ($Y < 0$) **then** $p \leftarrow p + 1$ **end if**	Find pseudorandom number in the half-open interval [-0.5, 0.5). Transform it to the right interval.

12.3 Table of SUPRAN Seeds

SUPRAN generates a single sequence of 2^{32} random seeds. To create multiple distinct Pseudorandom Generator objects, it is necessary to initialize them with different initial random seeds. For maximum independence, the subsequences of random seeds generated by these Pseudorandom Generator objects should not overlap. That means that starting random seeds should be long distances apart in the underlying sequence. To simplify the selection of initial starting seeds, Table 12.3/1 lists equally-spaced random seeds covering a total of 200 million seeds, about 5% of the total period of SUPRAN's generator.

TABLE 12.3/1

EVERY MILLIONTH SEED IN A SEQUENCE OF 200 MILLION RANDOM SEEDS FROM SUPRAN

$i/10^6$	Y_i	$i/10^6$	Y_i	$i/10^6$	Y_i
0	0				
1	1986273152	31	444780672	61	1223524736
2	-155500800	32	1398009856	62	-231169792
3	-1358602624	33	672665472	63	-1686716288
4	-851280384	34	-959500544	64	1923604480
5	2138217856	35	1568231040	65	-1513357440
6	-208290560	36	437677568	66	1659051776
7	1470880896	37	715558272	67	-672317824
8	-642450432	38	-1121342208	68	854220288
9	-1481565312	39	-6304640	69	-1579516544
10	-274711808	40	537455616	70	1388158208
11	-545105280	41	1281690496	71	1939061888
12	-1520993792	42	-1296815360	72	844946432
13	1864341888	43	-2131342720	73	-1122436224
14	1792719104	44	-450139648	74	1103633152
15	-964110208	45	223578496	75	-295028096
16	-1339426816	46	661563648	76	-251700736
17	1438521216	47	1635567744	77	2005367168
18	-448448768	48	-377624576	78	-1342007040
19	-1933617536	49	-311294080	79	-932136832
20	2049734144	50	-1688656128	80	-288237568
21	-611543680	51	557008512	81	1361442688
22	-555764480	52	-1392482816	82	493688576
23	-1306143616	53	1824556416	83	-2119747968
24	-2090929152	54	-1905023744	84	-1412147712
25	-2138369152	55	1075430528	85	-906726016
26	-676711680	56	-1347230720	86	168269056
27	-1229172096	57	188679040	87	-1710377856
28	1270968832	58	-2135022848	88	-1475947520
29	-994471552	59	1043350144	89	1643312000
30	1336193280	60	1905615360	90	-170781952

continued . . .

TABLE 12.3/1 (concluded)

$i/10^6$	Y_i	$i/10^6$	Y_i	$i/10^6$	Y_i
91	-1851510144	131	819273344	171	-20575616
92	1667846656	132	-929939968	172	-595908096
93	-1725861504	133	-196977280	173	2116241792
94	1624019200	134	-505053952	174	297691392
95	-395661184	135	-1082418048	175	-984840064
96	1576783872	136	-1157317632	176	-959600640
97	-276828288	137	41999232	177	1145161600
98	-889778432	138	-1007682816	178	1806231296
99	509685376	139	760355456	179	1795360384
100	398347776	140	1822898688	180	1884300800
101	-452039296	141	-1343268480	181	-1450162816
102	-1269723904	142	623540480	182	1153656064
103	-1282954112	143	-94857088	183	1877574784
104	280022016	144	1568258048	184	493345280
105	-104010880	145	2089670528	185	772719488
106	-1663300864	146	-2053835008	186	487449344
107	668871296	147	-1500572032	187	1409286784
108	-925677056	148	226244096	188	15016448
109	-1380226688	149	-396601984	189	1371357568
110	76974336	150	1697608960	190	1955094784
111	-77289344	151	-1309305728	191	-1756987264
112	-1071265792	152	-55659520	192	-403202048
113	-2133203072	153	1935332224	193	-1801732224
114	1803618048	154	1140454144	194	-885858560
115	-1373952384	155	-1668541824	195	-1178796416
116	1990739456	156	-1424936448	196	-1908793856
117	-215456384	157	-1651945088	197	1990868352
118	1369146624	158	-1577815808	198	-1592959744
119	-1073634176	159	-430796672	199	996375680
120	1817887744	160	-1734103040	200	1940691968
121	-2069437568	161	-421015680		
122	921043712	162	-14749952		
123	-1323818368	163	256446080		
124	557662720	164	1164324352		
125	-1252695680	165	-814330496		
126	-1688174336	166	-612799232		
127	22978688	167	-1754297216		
128	357548032	168	827894784		
129	87285632	169	-684405888		
130	-16056576	170	-1224480000		

12.4 Combined Multiplicative Congruential Generators for the Future

L'Ecuyer (1988) published a description of pseudorandom number generators based on combining multiplicative congruential generators. L'Ecuyer and Côté (1991) extended the study of such generators, and have published Pascal code to implement such a generator. The combined generator has advantages over SUPRAN in terms of cycle length, generation of subsequences, and rapid skipping of large numbers of seeds. With suitable adaptation it also has a precision advantage. On the other hand, it is less efficient in terms of computer time. Future versions of the Parameter Sampling Package (PSP) should use this type of generator.

L'Ecuyer and Côté employ l maximum period multiplicative congruential generators (MCGs):

$$s_{j,i} \equiv a_j s_{j,i-1} \bmod m_j \qquad j = 1, \ldots, l. \tag{12.4-1}$$

They use MCGs rather than linear congruential generators (LCGs) because it is easier to advance an MCG by an arbitrary number of steps:

$$\begin{aligned} s_{j,i+k} &\equiv a_j^{\,k} s_{j,i} \bmod m_j \\ &\equiv (a_j^{\,k} \bmod m_j) s_{j,i} \bmod m_j \end{aligned} \qquad j = 1, \ldots l,\ 0 \le k\ . \tag{12.4-2}$$

The power in brackets can be stored in a table if k is known in advance; otherwise it can be computed readily with no more than $2\log_2 k$ multiplications using an ancient method described by Knuth (1969, pp. 398-401).

An individual MCG may not have properties that are as good as a well-chosen LCG. For example, the maximal period is m_j - 1 and it is attained only if m is prime and a_j is a primitive element modulo m_j. For a 32-bit generator, the prime moduli are typically chosen to be less than 2^{31}, giving a period that is less than half of SUPRAN's. In the case of a combined random number generator, the use of multiple generators corrects for the deficiencies of the individual generators. L'Ecuyer and Côté described generators for which the period is many orders of magnitude greater than SUPRAN's.

L'Ecuyer and Côté combine their generators in the following way:

$$Z_i \equiv \left(\sum_{j=1}^{l} (-1)^{j-1} s_{j,i} \right) \bmod (m_1 - 1)\ . \tag{12.4-3}$$

This generator has a period that is the least common multiple (LCM) of the individual periods, since there are l independent seeds involved, and each must return to its original value before the generation of seeds can repeat. The total period must then be divisible by the period of each generator, and the LCM is the smallest number with this property. L'Ecuyer and Côté transform Z_i to the interval (0,1) using:

$$U_i = \begin{cases} Z_i/m_1 & \text{if } Z_i > 0 \\ (m_1 - 1)/m_1 & \text{if } Z_i = 0 \end{cases} . \qquad (12.4\text{-}4)$$

Equations (12.4-3) and (12.4-4) are not ideal for use with SYVAC3. From Equation (12.4-3) it is clear that there will be only m_1 - 1 distinct values possible for U_i. That would exacerbate the precision and granularity problems mentioned in Section 12.1. From Equation (12.4-4) it can be seen that there is a slight bias in the generator away from extreme values. As shown in Figure 12.4/1, each real value that the generator can produce lies in one of m_1 - 1 equal-width intervals. Values near zero lie near the right end of the enclosing interval; those near one lie at the left end. Both deficiencies are resolved by the following generator, which first generates a uniform variate between 0 and 1 for each of the l generators (Equation (12.4-5) and then combines thes numbers modulo 1 (i.e., takes the fractional part of the sum):

$$U_{j,i} = s_{j,i}/m_j \qquad j = 1, \ldots l \qquad (12.4\text{-}5)$$

$$W_i = \left(\sum_{j=1}^{l} \frac{(-1)^{j-1}}{2m_j} + \sum_{j=1}^{l} (-1)^{j-1} U_{j,i} \right) \bmod 1 \; . \qquad (12.4\text{-}6)$$

By computing the double-precision uniform variates first, this approach guarantees that the spacing between possible values is much smaller, at the expense of slower execution. The alternating sign structure in Equation (12.4-6) reduces the loss of precision in dropping the integer part of the sum. The first summation in Equation (12.4-6) can be precalculated as a constant. It guarantees that W_i can never be identically 0. It gives the same result as if all possible values for $s_{j,i}$ from 0 to m_j - 1 had been moved to the right by 0.5 to centre them in intervals like those shown in Figure 12.4/1.

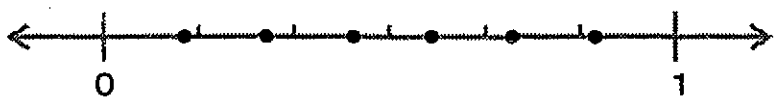

FIGURE 12.4/1: Equation (12.4-4) Puts 6 Points in 6 Intervals if $m_1 = 7$, but They Are Biased in Positioning Away from the Ends

L'Ecuyer and Côté use a full set of initial seeds $s_0 = (s_{1,0}, \ldots s_{l,0})$ to start up their combined generator. It is possible instead to define a reduced set of initial seeds: $s_0 = (s_{1,0}, 1, 1, \ldots 1)$, so that the user specifies only one random seed, $s_{1,0}$, just as he does now. This limits the number of distinct initial random seeds to m_1 - 1, but guarantees that each sequence has a period that is the LCM of m_2 - 1 to m_l - 1.

REFERENCES

Andres, T.H. 1987. Statistical sampling strategies. In Proceedings of the NEA/OECD Workshop on Uncertainty Analysis for Performance Assessments of Radioactive Waste Disposal Systems, Seattle.

Andres, T.H. in preparation. SYVAC3 Manual; AECL Report 10982 (COG-93-422)[1].

Andres, T.H. in preparation. SYVAC3 Time Series Package; AECL Report 10984 (COG-93-424)[1].

Andres, T.H. in preparation. SYVAC3 File Reading Package; AECL Report 10985 (COG-93-425)[1].

Andres, T.H. in preparation. SYVAC3 Specifications[1].

Becker, Richard A., John M. Chambers and Allan R. Wilks. 1988. The New S Language. Wadsworth & Brooks/Cole Advance Books and Software, Pacific Grove, California.

Bosten, Nancy E. and E.L. Battiste. 1973. Remark on Algorithm 179, incomplete beta ratio. In Collected Algorithms from CACM, Volume I, 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York.

Coad, Peter with Edward Yourdon. 1989. Object-Oriented Analysis. Yourdon Press/Prentice-Hall, Englewood Cliffs, New Jersey.

Durand, D. 1971. Stable chaos–An introduction to statistical control. General Learning Press, Morristown, NJ.

Dormuth, K.W. and R.D. Quick. 1989. Accounting for parameter variability in risk assessment for a Canadian nuclear fuel waste disposal vault. International Journal of Energy Systems 1(2), 125-127.

Hart, John F., E.W. Cheney, Charles L. Lawson, Hans J. Maehly, Charles K. Mesztenyi, John R. Rice, Henry G. Thacher, Jr., and Christoph Witzgall. 1978. Computer Approximations (2nd printing with corrections). Robert E. Krieger Publishing Company, Huntington, New York.

Hill, G.W. and A.W. Davis. 1971. Algorithm 442–Normal deviate. In Collected Algorithms from CACM, Volume II, 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York

l'Ecuyer, Pierre. 1988. Efficient and portable combined random number generators. CACM 31(6), 742-774.

l'Ecuyer, Pierre and Serge Côté. 1991. Implementing a random number package with splitting facilities. ACM TOMS 17(1) 98-111.

Knuth, D.E. 1969. The art of computer programming. Addison-Wesley Publishing Company, Reading, MA.

Ludwig, Oliver G. 1963. Algorithm 179 – Incomplete beta ratio. In Collected Algorithms from CACM, Volume I , 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York.

Marsaglia, G. 1972. The structure of linear congruential sequences. In Applications of Number Theory to Numerical Analysis. Academic Press, London, 249-185.

Marse, Ken and Stephen D. Roberts. 1983. Implementing a portable FORTRAN uniform (0,1) generator. Simulation 41(4), 135-139.

Page-Jones, Meilir. 1980. The Practical Guide to Structured Systems Design.

Pike, M.C. and I.D. Hill. 1966. Algorithm 291–Logarithm of Gamma function. In Collected Algorithms from CACM, Volume II, 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York.

Pike, M.C. and I.D. Hill. 1966. Remark on Algorithm 179, incomplete beta ratio. In Collected Algorithms from CACM, Volume I. 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York.

Pike, Malcolm C. and Jennie SooHoo. 1975. Remark on Algorithm 179, incomplete beta ratio. In Collected Algorithms from CACM, Volume I, 1980, Association for Computing Machinery, Inc., 1133 Avenue of the Americas, New York, New York.

Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling. 1986. Numerical Recipes–The Art of Scientific Computing. Cambridge University Press, London.

Rubinstein, R.Y. 1981. Simulation and the Monte Carlo Method. John Wiley and Sons, Inc., Toronto.

Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen. 1991. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey.

Spanier, J. and K.B. Oldham. 1987. An atlas of functions. Hemisphere Publising Corporation, New York.

Stephens, M.E., B.W. Goodwin and T.H. Andres. 1989. Guidelines for defining probability density functions for SYVAC3-CC3 parameters. Atomic Energy of Canada Limited Technical Record, TR-479[1].

von Mises, R. 1919. Grundlagen der warscheinlichkeitsrechnung. Math. Zeitschrift 5, 52-99.

Walker, J.R. 1985. The Pseudo-Random Number generator for SYVAC3. Atomic Energy of Canada Limited Technical Record, TR-373[1].

Wolfram, S. 1991. Mathematica: A system for doing mathematics by computer. Addison-Wesley Publishing Co., Redwood City, California.

---

[1] Unrestricted, unpublished report available from Scientific Document Distribution Office, AECL Research, Chalk River Laboratories, Chalk River, Ontario K0J 1J0.

INDEX

Cat. No. / Nº de cat.: CC2-10983E
ISBN 0-660-15922-8
ISSN 0067-0367

To identify individual documents in the series, we have assigned an AECL- number to each. Please refer to the AECL- number when requesting additional copies of this document from

Scientific Document Distribution Office (SDDO)
AECL Research
Chalk River, Ontario
Canada K0J 1J0

Fax: (613) 584-1745 Tel.: (613) 584-3311
ext. 4623

Price: F

Pour identifier les rapports individuels faisant partie de cette série, nous avons affecté un numéro AECL- à chacun d'eux. Veuillez indiquer le numéro AECL- lorsque vous demandez d'autres exemplaires de ce rapport au

Service de Distribution des documents officiels (SDDO)
EACL Recherche
Chalk River (Ontario)
Canada K0J 1J0

Fax: (613) 584-1745 Tél.: (613) 584-3311
poste 4623

Prix: F